الخوارزميات والبرمجة 2


وقل رب زدني علما
جامعة دمشق
Damascus University

منشورات جامعة دمشق

كلية العلوم

الخوارزميات والبرمجة (2)

لطلاب السنة الثانية إحصاء

تأليف

د. خالد الخنيفس
أستاذ في قسم الرياضيات
كلية العلوم

د. محمد جاسم المحمد
مدرس في قسم الرياضيات
كلية العلوم

د. محمد فراس الحلبي
مدرس في قسم الرياضيات
كلية العلوم

جامعة دمشق

1433 – 1434 هـ

2011 – 2012 م



الخوارزميات والبرمجة 2

السنة الثانية

قسم إحصاء

فهرس المحتويات

فهرس المحتويات..... 5
المقدمة..... 11
الفصل الأول: **الإجراءات والدوال**..... 13
1.مقدمة..... 13
2. الإجرائيات..... 13
1.2 الصفات العامة للإجرائيات PROCEDURS GENERAL FEATURES..... 13
2.2 تعريف الإجرائية واستدعاؤها PROCEDURE DEFINITION AND CALL..... 14
3.2 الإجرائيات بلغة باسكال PASCAL PROCEDURES..... 15
4.2 المتغيرات العامة والمتغيرات الموضعية GLOBAL AND LOCAL VARIABLES..... 16
5.2 الوسطاء PARAMETERS..... 17
6.2 تعدد الوسطاء MULTIPLE PARAMETERS..... 17
7.2 الوسطاء المرجعية والمتغيرات الوسيطية FORMAL PARAMETERS AND ARGUMENTS..... 18
8.2. التصريح عن الوسطاء واستخدامها DECLARING AND USING PARAMETERS..... 20
9.2 الوسطاء المرجعية REFERENCE PARAMETERS..... 21
3. الدوال FUNCTIONS..... 26
1.3 تعريف الدوال في لغة باسكال DEFINING FUNCTIONS IN PASCAL..... 26
2.3 استدعاء الدالة CALLING A FUNCTION..... 27
3.3 تمرير الوسطاء إلى الدوال PASSING PARAMETERS TO FUNCTIONS..... 31
4. مفهوم التعاودية RECURSION..... 31
1.4 ميزات الإجرائية و الدالة التعاودية PROPERTIES OF RECURSIVE FUNCTIONS..... 34
5. الإجراءات والدوال مسبقة التعريف PREDEFINED PROCEDURES AND FUNCTIONS..... 39
الفصل الثاني: **السجلات RECORDS**..... 45
1. نوع البيانات سجل THE RECORD DATA TYPE..... 45
2. الوصول إلى الحقول ضمن السجل ACCESSING FIELDS IN A RECORD..... 46
3. الحقول المركبة ضمن سجل FIELDES OF COMPOSITE TYPE IN A RECORD..... 48
4. متجهات السجلات ARRAYS OF RECORDS..... 53
5. إسناد السجلات RECORD ASSIGNMENT..... 54
6. السجل المتبدل VARIANT RECORD..... 55
7. السجلات الثابتة CONSTANT RECORDS..... 57
8. الكائنات OBJECTS..... 59

9. الوراثة INHERITANCE 65
تمارين 69
الفصل الثالث: المجموعات THE SETS 73
1. نوع البيانات مجموعة THE SET DATA TYPE 73
1.1 المفهوم العام للمجموعة THE SET GENERAL CONCEPT 73
2.1 المجموعة في لغة الباسكال 73
2. تمثيل المجموعات REPRESENTATION OF SETS 75
3. مجموعة خاصة A SPECIAL SET 77
4. العمليات على المجموعات SET OPERATORS 77
1.4 معامل الانتماء IN OPERATOR 77
2.4 معامل الإجتماع UNION OPERATOR 79
3.4 معامل التقاطع INTERSECTION OPERATOR 80
4.4 معامل فرق المجموعات THE SET DIFFERENCE OPERATOR 81
5.4 إضافة و حذف عنصر من مجموعة ADD/REMOVE AN ELEMENT FROM A SET 82
5. مقارنة المجموعات COMPARING SETS 83
1.5 معاملا تساوي وعدم تساوي المجموعات THE SET / INEQUALITY OPERATORS EQUALITY 83
2.5 معامل المجموعة الجزئية THE SUBSET OPERATOR 84
6. المجموعات الثابتة والمجموعات عديمة الأسماء CONSTANT AND ANONYMOUS SETS 86
تمارين 91
الفصل الرابع: الملفات THE FILES 93
1. الملفات FILES 93
2. الملفات المباشرة و العمليات عليها 94
1.2 تعريف الملف برمجياً 94
2.2 ربط المتحول الملفي باسم الملف الخارجي 95
3.2 تهيئة الملف للكتابة 95
4.2 فتح ملف موجود 95
5.2 قراءة عنصر من ملف 96
6.2 كتابة عنصر في ملف 96
7.2 إغلاق الملف 96
4. الملفات النصية TEXT FILES 103
1.4 الإجراء READ مقابل الإجراء READLN READ VS. READLN 104

2.4 العمليات على الملفات النصية OPERATIONS ON TEXT FILES 106
3.4 إعادة تسمية وحذف الملفات RENAMING AND ERASING FILES 108
4.4 الدالتان SEEK EOF و SEEK ELON SEEK EOF & SEEK ELON FUNCTIONS 109
5.4 بعض الأمثلة على الملفات محددة النوع 109
5. الملفات مهملة النوع UNTYPED FILES 128
6. أجهزة الدخل والخرج القياسي THE STANDARD INPUT AND OUTPUT DEVICES 134
1.6 أجهزة الطابعة PRINTER DEVICES 135
2.6 الأجهزة التسلسلية SERIAL DEVICES 136
3.6 الجهاز NUL THE NUL DEVICE 136
تمارين 138
الفصل الخامس: المؤشرات POINTERS 141
1. مفهوم المؤشر POINTER CONCEPT 141
2. التخصيص الساكن للذاكرة STATIC MEMORY ALLOCATION 141
3. التخصيص الديناميكي للذاكرة DYNAMIC MEMORY ALLOCATION 142
4. توضع طبقات البرنامج PROGRAM LAYOUT 143
1.4 مقاطع البرنامج PROGRAM SEGMENT 144
2.4 كومة البرنامج THE PROGRAM HEAP 145
5. المؤشرات POINTERS 146
1.5 تعريف نوع البيانات مؤشر DEFINING POINTER TYPE 147
2.5 التصريح عن متغيرات من النوع مؤشر DECLARING POINTER VARIABLES 148
3.5 تخصيص مساحات تخزين ديناميكية ALLOCATING DYNAMIC STORAGE 148
4.5 الوصول إلى غاية المؤشر ACCESSING TARGET VARIABLE 149
5.5 وضع قيم بمتحول غاية المؤشر ASSIGNING VALUES TO TARGET VARIABLE 150
6.5 التمييز بين المؤشر وغاية المؤشر DISTINCTION BETWEEN POINTER AND TARGET 151
7.5 إسناد المؤشرات POINTER ASSIGNMENT 152
8.5 إسناد العناوين للمؤشرات ASSIGNING ADDRESS TO POINTERS 155
6. استخدام الإجراء GETMEM لتخصيص مساحات تخزين ديناميكية USING GETMEM TO ALLOCATE DYNAMIC STORAGE 157
7. أخطار المؤشرات POINTER PITFALLS 157
1.7 مشكلة إعادة استخدام مؤشر REUSING A POINTER 157
2.7 مشكلة إعادة مساحة من الذاكرة إلى الكومة RETURNING STORAGE TO THE HEAP 159

8. استخدام الإجراء FREEMEM لإعادة مساحات الذاكرة المخصصة USING FREEMEM TO RETURN ALLOCATED STORAGE 163
9. إنهاء تخصيص مجموعة مؤشرات من الكومة DEALLOCATING GROUPS ON THE HEAP 164
10. القيمة لا شيء مسبقة التعريف الخاصة بالمؤشر THE PREDEFINED NIL VALUE FOR POINTER 166
11. بنى التأشير الذاتي SELF- REFRENNTIAL STRUCTURES 168
1.11 ربط بنى التأشير الذاتي LINKING SELF-REFERENTIAL STRUCTURES 169
2.11 بناء القوائم المترابطة BUILDING LINKED LISTS 170
3.11 إضافة العقد إلى بداية القائمة المترابطة ADDING AT THE FRONT OF A LINKED LIST 170
4.11 الإجراء INITLNODE INITLNODE PROCEDURE 172
5.11 الإجراء LLAT FRONT LLAT FRONT PROCEDURE 173
6.11 الإجراء WRITE LIST WRITE LIST PROCEDURE 175
7.11 إضافة العقد في أي مكان ضمن القائمة المترابطة ADDING NODS ELSEWHERE IN A LINKED LIST 176
8.11 ترتيب قائمة مترابطة ORDERING A LINKED LIST 178
تمارين 199
الفصل السادس: الوحدات و الرسم بلغة الباسكال 201
PASCAL UNITS 201
1. مقدمة 201
2. الوحدات الجاهزة بلغة الباسكال PREDEFINED UNITS 202
1.2 وحدة الطابعة PRINTER UNIT 202
2.2 الوحدة CRT 204
3. وحدة الرسم البياني GRAPH UNIT 207
1.3 مفهوم الرسم على الحاسوب COMPUTER GRAPHICS CONCEPT 207
2.3 روتينات إعداد وحدة الرسم SETTING GRAPHICS UNIT ROUTINE 208
3.3 رسم النقطة 214
4.3 رسم خطوط DRAWING LINES 218
5.3 رسم مستطيل DRAWING RECTANGLE 219
6.3 رسم دوائر و أقواس DRAWING CIRCLES AND ARCS 221
7.3 بعض الدوال و الإجرائيات الأخرى 223
4. إنشاء الوحدات UNIT CREATION 306
1.4 تعريف الوحدة UNIT DEFINITION 307

2.4 استخدام الوحدات الأخرى ضمن الوحدة USING OTHER UNITS WITHIN A UNIT ... 310
3.4 ترجمة الوحدة COMPILING A UNIT 311
تمارين 321
المصطلحات العلمية 325
المراجع العلمية 337

Damascus University

وقل رب زدني علما
جامعة دمشق
Damascus University

المقدمة

لقد أصبح الحاسوب أداة لا يمكن الاستغناء عنها لأنّه دخل تقريباً في كل مجالات الحياة. فهو يُستخدم لحل مسائل في الهندسة و الطب و الرياضيات والفلك و غيرها كثير. الحاسوب بطبيعته عبارة عن أداة إلكترونية لا تستطيع القيام بأي عمل من دون تدخل العنصر البشري الذي يزودها بطريقة العمل. لحل مسألة معينة، لابد للعنصر البشري إعطاء الحاسوب مجموعة الخطوات اللازمة لحل تلك المسألة. تُسمى مجموعة الخطوات تقنياً بخوارزمية الحل.

من جهةٍ أخرى حتى نستطيع تزويد الحاسوب بطريقة الحل لمسألة معينة، لابد من لغة تُمكننا من التخاطب مع الحاسوب. هذا بالضبط هو الهدف من لغات البرمجة فهي وسيلة تُمكّن المستخدم من التواصل مع الحاسوب و إعطائه التعليمات اللازمة للقيام بالعمل المطلوب. يقدم هذا الكتاب لغة البرمجة باسكال والتي تُبنى على ما تم طرحه في الكتاب الأول (الخوارزميات و البرمجة (1)). بالتحديد، يُقدّم هذا الكتاب مواضيع مُتقدمة في لغة الباسكال تتخذ من المفاهيم التي نُوقشت في كتابنا الأول أساساً تُبنى عليه موضوعات أكثر تقدماً. من الموضوعات التي يُركّز عليها هذا الكتاب نذكر: المجموعات، والسجلات، والكائنات، والوراثة، والوحدات. تُعَد هذه الموضوعات مهمة للغاية و تضع الطالب في مستوى يؤهله إلى تعلم مفاهيم أكثر تقدماً و توفر له قاعدة علمية كافية لاستيعاب مفاهيم جوهرية في البرمجة مثل البرمجة غرضيه التوجه.

تُعد السجلات مدخلاً جيداً للطالب يُمكّنه من التعرّف إلى بنى معطيات مركبة. إضافة إلى ذلك، فإن مفاهيم الكائنات و الوحدات تُدخل الطالب إلى مفاهيم متقدمة ستُغطى في المقررات اللاحقة و التي تُكرس لمفاهيم أخرى مثل الصفوف و البنى الوراثية و غيرها من المواضيع.

لقد راعينا في هذا الكتاب التوازن بين كثافة طرح المفاهيم النظرية و دعم هذه المفاهيم بالأمثلة المناسبة التي تساعد الطالب على استيعاب هذه المفاهيم. لذلك فإن الكتاب يُعطي الطالب ما يحتاجه لاكتساب المهارات الأساسية في البرمجة و حل المسائل حاسوبياً.

أخيراً، نرجو أن يكون هذا الكتاب مساهمة موفقة و تكون عوناً للطلبة في دراستهم و تُثري المكتبة العربية و تضع بين يدي المهتمين مرجعاً يمكنهم العودة إليه و الاستفادة منه.

و الله الموَفق.

دمشق في 1 شباط 2012

المؤلفون.

الفصل الأول

الإجراءات والدوال
PROCEDURES AND FUNCTIONS

1. مقدمة

يُبنى البرنامج في لغة الباسكال من الدوال والإجرائيات، والتي تتألف بدورها من مجموعة من العبارات. الغاية الأساسية من الإجرائيات و الدوال هو تقسيم البرنامج إلى برامج أبسط وبالتالي تسهيل فهم البرنامج العام وذلك بتقليص عدد الأسطر البرمجية. فيمايلي سنقدم عرضاً موجزاً بأهم خواص الإجرائيات و الدوال.

2. الإجرائيات

1.2 الصفات العامة للإجرائيات Procedurs General Features

تتكون الإجرائية من مجموعة من العبارات البرمجية و التي تقوم بإنجاز عمل برمجي جزئي معين. واسم الإجرائية يشير إلى هذه المجموعة من العبارات. وفي أغلب الأحيان يكون الإجرائية عبارة عن برنامج مصغر، يمكن أن يملك قسم تصريح خاصاً به أو إجراءاته الخاصة به أيضاً.

لقد صُمّمت الإجراءات لإنجاز بعض المهام البرمجية الجزئية، مثل مسح الشاشة، أو ترتيب مجموعة من القيم، ولكن قد نحتاج في بعض الأحيان إلى إجرائيات تُنجز بعض الأعمال العامة مع بعض التغييرات في كل مرة تُنفذ فيها تلك الإجرائيات.

تملك الإجرائيات بشكل عام طريقتين للاتصال مع البرنامج المُستدعي لها:

a) عن طريق المتغيرات العامة **(global variables)**، إذ تُستخدم هذه المتغيرات ضمن أي جزء من أجزاء البرنامج.

b) عن طريق الوسطاء **(parameters)**، والتي يمكن من خلالها تمرير المعلومات من وإلى الإجرائية.

```
Procedure GetDate(Var Month,Date,Year:integer);
```

- التصريح في الإجرائية **The Procedure declaration**

تملك كل إجرائية قسم تصريح خاصاً بها، إذ يمكن أن نصرح فيه عن ثوابت أو متغيرات أو أنواع مختلفة. وطريقة التصريح عن هذه الأنواع تخضع لنفس الطريقة المستخدمة في البرنامج الرئيسي.

- جسم الإجرائية **The Procedure body**

يحتوي جسم الإجرائية على مجموعة من التعليمات البرمجية اللازمة لإنجاز المهمة المطلوبة. يبدأ جسم الإجرائية بالكلمة المحجوزة **Begin** وينتهي بالكلمة المحجوزة **End** متبوعاً بفاصلة منقوطة مشيرة إلى أن الإجرائية قد انتهت. عندما ينتقل التحكم إلى الإجرائية تبدأ بتنفيذ عبارتها الأولى، والتي تأتي مباشرة بعد الكلمة المحجوزة **Begin**، ويستمر هذا التنفيذ إلى أن تصل إلى آخر عبارة في جسم الإجرائية، وهي التي تأتي قبل الكلمة المحجوزة **End**، وبعدها يعاد التحكم ثانية إلى الروتين المستَدعي.

4.2 المتغيرات العامة والمتغيرات الموضعية **Global and Local Variables**

يمكن للإجرائية الاتصال مع الروتين المستَدعي عبر المتغيرات والوسطاء، وسوف نشرح الآن كلاً من المتغيرات العامة **(global variables)** و المتغيرات الموضعية **(local variables)**.

المتغيرات العامة هي المتغيرات التي يمكن استخدامها في البرنامج الرئيسي وفي جميع الروتينات **(routines)** المعرفة ضمن هذا البرنامج الرئيسي. سنكتفي الآن بالحديث عن المتغيرات الخاصة باعتبار أن المتغيرات العامة هي المتغيرات المُصَرَّح عنها ضمن البرنامج الرئيسي.

يمكن أن نُصَرِّح عن متغيرات خاصة بكل إجرائية في قسم التصريح الخاص

بالإجراءات، وهذه المتغيرات تُدعى عندها بالمتغيرات المحلية. هذه المتغيرات تكون مرئية فقط في الإجرائية التي عُرفت فيها.

5.2 الوسطاء Parameters

إن رأس الإجرائية يملك قائمة اختيارية من الوسطاء، وهذه القائمة هي البيئة التي يتم من خلالها إعداد المعلومات التي سوف تُمرر إلى الإجرائية على أنها متغيرات وسيطية. هناك فائدة واضحة للوسطاء كونها تجعل الإجرائية عامة الاستخدام وبالتالي يمكن إعادة استخدام الإجرائية خارج البرنامج الذي عُرفت فيه.

مثال:

```
Program ProcExample;
 Var
     NrTimes:integer;
   Procedure CountVals(How:integer);
     Var Index:integer;
       Begin
          For Index:=1 to How do
             Writeln(Index);
       End;
Begin
      CountVals(30)
      NrTimes:=95;
      CountVals(NrTimes);
      Readln;
End.
```

6.2 تعدد الوسطاء Multiple Parameters

يُمكن التصّريح عن عدة وسطاء ضمن الإجرائية الواحدة، ويمكن أن يكون هؤلاء الوسطاء من نوع واحد أو من أنواع مختلفة.

أمثلة توضيحية.

```
Procedure CountVals(HowHigh:integer);
Procedure BubbleSort(Var Info:String[80];
   ArraySize:integer);
Procedure GetInteger(Message : String; Var
   Value:integer);
Procedure GetString(Message:String; Var Value:
      string);
Procedure GetDate(Var Month,Date,Year:integer);
```

ملاحظة:

إذا كان لدينا عدة وسطاء تملك النوع نفسه فإننا نستطيع دمج تصريح هؤلاء الوسطاء في تصريح واحد مع ملاحظة الفصل بين الوسطاء بفاصلة (,)، أما إذا كان لدينا عدة وسطاء، ولكن لكل منها نوعه الخاص فيجب وضع تعريف خاص لكل وسيط، مع ملاحظة الفصل بين تعريف الوسطاء المختلفة بفاصلة منقوطة.

7.2 الوسطاء المرجعية والمتغيرات الوسيطية Formal Parameters and Arguments

عندما نُعرف وسيطاً شكلياً في رأس الإجرائية فإننا نكون قد عرّفنا منفذاً لعبور المعلومات إلى الإجرائية لاستخدامها فيها، وعند استدعائنا للإجرائية علينا وضع هذه المعلومات على المنافذ بوساطة تحديدها أو تمريرها كمتغيرات وسيطية. ففي حين أن الوسطاء الشكلية تمثل المنافذ فإن المتغيرات الوسيطية تمثل المعلومات الفعلية المستخدمة ضمن الإجرائية، وفي الحقيقة تُعرَف المتغيرات الوسيطية على أنها وسطاء فعلية (actual parameters).

فمن أجل كل وسيط شكلي عرفناه وجب علينا تمرير متغير وسيطي إليه عند استدعاء الإجرائية.

مثال:

```
PROCEDURE GETINTEGER (Message : String;  Var Value : Integer);
```

وسطاء شكلية

أما استدعاء هذه الإجرائية ضمن البرنامج فيتم على الشكل الآتي:

```
GetInteger ( MySte, MyInt ) ;
```

وسطاء فعلية

وعليه يجري استبدال هذه المتغيرات كما يلي:

الوسطاء الشكلية	Message	Value
الوسطاء الفعلية	MyStr	MyInt

بالإضافة إلى ذلك يجب الانتباه إلى وضع المتغيرات الوسيطية بالترتيب الصحيح، فكل وسيط شكلي يملك نوعاً خاصاً به، وتعرّف الوسطاء الشكلية وفق ترتيب معين، لذلك يجب أن يملك كل وسيط فعلي نوعاً خاصاً به أيضاً ومطابقاً لنوع الوسيط الشكلي المناظر له، ويجب أن يأتي بنفس الترتيب الذي أتى به الوسيط الشكلي المناظر.

بعض الإستدعاءات الصحيحة

```
CountVals (30);                    (* يمكن أن نُمرّر قيمة صحيحة*)
CountVals (MyInt);                 (* يمكن للمتغير أن يُمرّر على أنه وسيط*)
CountVals (2*15);                  (* يمكن أن تُمرّر عبارة حسابية*)
GetInteger (MyStr, MyInt);         (* يمكن لمتغير سلسلة محارف أن يمرّر*)
```

GetInteger ('Value?' , MyInt); (* يمكن لسلسلة محارف أن تُمرّر *)

بعض الإستدعاءات الخاطئة

CountVals (MyInt, 30); (* هناك زيادة في عدد المتغيرات الممررة*)

CountVals (2.3); (* يجب أن تكون قيمة هذا المتغير صحيحة*)

GetInteger (MyInt); (* المتغير الأول غير موجود *)

GetInteger (MyInt, 'Value?'); (* هناك خطأ في ترتيب المتغيرات*)

8.2. التصريح عن الوسطاء واستخدامها Declaring and Using Parameters

التصريح هو عبارة عن تعليمة وظيفتها تخصيص مكان تخزين للمتغير عند استدعاء الإجرائية، ومكان التخزين هذا سوف يكون بالحجم الكافي لتمثيل قيمة المتغير، وحسب نوعه المُعرّف في رأس الإجرائية.

عند استدعاء إجرائية مع متغيرات وسيطية، فإن قيم هذه المتغيرات فقط هي التي تهمنا ولا يهمنا مصدر هذه القيم سواءً كانت من داخل برنامج رئيسي أو من إجرائية أخرى. في حال أن قيمة المتغيرات هي التي تُنسخ إلى بيئة الوسيط المحلي للإجرائية وليس عنوان تلك المتغيرات، فإن أي تغيير تُحدثه هذه الإجرائية لن يكون له أثر على متغيرات البرنامج العام ؛ وإلا فإن الإجرائية سوف تعمل بشكل فوضوي غير متوقع وتعطينا نتائج غير صحيحة.

يمكن أن نمرر المعلومات بوساطة قيمها بأشكال مختلفة:

1. على شكل قيمة ثابتة. مثال: 30 أو 'A string constant'.
2. على شكل متغير. مثال Natimes أو GreetingStr.
3. على شكل تعبير. مثال: 3+5 أو 5>1 (من أجل المتحول البولياني).

عندما نصرح عن وسيط فإن حيز تخزين مؤقت يُخصص لهذا الوسيط، ويُربط

اسم الوسيط بحيز التخزين المؤقت هذا. وهذا الإسم يُحدد من خلال بيئة الإجرائية **(procedures environments)**، وبعبارة أدق من خلال قائمة الوسطاء الملحقة باسم الإجرائية. عند الإنتهاء من تنفيذ أي إجرائية فإن وسطاؤها المحلية وأماكن تخزينها تتلاشى ولا يستطيع البرنامج الرئيسي التعرف عليها. في الفقرة التالية سنجد حلاً لهذه المشكلة من خلال ما يسمى بالوسطاء المرجعية.

9.2 الوسطاء المرجعية Reference Parameters

يُستخدَم الوسيط المرجعي (الذي يبدأ التصريح عنه بوساطة الكلمة المحجوزة var في ترويسة الدالة) في إعادة المعلومات من الإجرائية إلى البرنامج الرئيسي. فعند الانتهاء من تنفيذ الإجرائية تكون الوسطاء قد امتلكت قيماً جديدة تختلف عن القيم التي مُررت بها إلى الإجرائية، هذا يعني أن المتغيرات التي تم تمريرها من البرنامج الرئيسي إلى الإجرائية قد امتلكت قيماً جديدة، وعندها تُرجع هذه القيم الجديدة إلى البرنامج الرئيسي. والسبب في ذلك أن كلاً من المتغيرات الوسيطية **(argument)** التي مُررت إلى الإجراء و الوسطاء الشكلية **(formal parameter)** تشير إلى مكان واحد ضمن الذاكرة، وهذا يكون فقط في حالة الوسطاء المرجعية المسبوقة بالكلمة المحجوزة **var** في ترويسة الدالة.

مثال:

```
Program ParamTest;
Var Val1, Val2: integer;
Procedure BadIntSwap(First:integer;
     Second:integer);
Var
    Temp:integer;
 Begin
       Temp:=First;
       First:=Second;
 End;
```

```
Procedure GoodIntSwap(Var First:integer;Var
    Second:integer);
Var
         Temp:integer;
Begin
        Temp:=First;
        First:=Second;
        Second:=Temp;
End;
Procedure GetInteger(Message:String; Var
    Value:integer);
Begin
      Write(Message,'    ');
      Readln(Value);
End;
Begin
   GetInteger('Value 1 ?',Val1);
   GetInteger('Value 2 ?',Val2);
   Writeln('Original Val1=',val1:5,';Val2=',
          Val2:5);
   BadIntSwap(Val1,Val2);
   Writeln('After BadIntSwap');
   Writeln('Val1=',Val1:5,';Val2=',Val2:5);
   GoodIntSwap(Val1,Val2);
   Writeln('After GoodIntSwap');
   Writeln('Val1=',Val1:5,'; Val2=',Val2:5);
   Readln;
End.
```

اعتماداً على التغيير الذي حصل بعد انتهاء الإجراء يمكننا اعتبار أن المتغيرين قد مُررا مباشرةً ولم تُمرَر نسخة عن قيمهما. يُسمى هذا التمرير للوسطاء

بالتمرير المرجعي **(passed by reference)** ، والوسطاء التي تُمرَر بهذه الطريقة يُمكن أن تتغير قيمتها عند استدعاء الإجرائية (أي تتغير قيمة الوسطاء الفعلية). و بالتالي فالوسطاء المُمرَرة مرجعياً، ترتبط بمواضع تخزين في الذاكرة لذلك لا يمكن تمرير قيم مثل: (30)، أو تعبير مثل: (3+5) للوسطاء مرجعية، ولكن يمكن فقط تمرير المتغيرات مرجعياً (أي بشكل مرجعي)، حيث سيعرف البرنامج الرئيسي عنوان هذه المتغيرات في الذاكرة، ومن ثم يقوم بتغيير محتواها حسب تسلسل تعليمات الروتين.

مثال:

```
Program ParamDemo;
Var
     G1,G2 :integer;
Procedure AllValues (X,Y:integer);
Begin
    Writeln('Entering AllValues:X =',X,';Y =',Y);
    X:=X*2;
    Y:=Y*4;
    Writeln('Leaving AllValues:X =',X,';Y=',Y);
End;
Procedure ValueVar(X:Integer;Var Y:integer);
Begin
    Writeln('Entering ValueVar:X=',X,';Y=',Y);
    X := X * 2;
    Y := Y * 4;
    Writeln('Leaving ValueVar:X=',X,';Y=',Y);
End;
Procedure AllVars(Var X, Y:integer);
Begin
     Writeln('Entering AllVars:X=',X, ';Y=',Y);
```

```
    X:=X*2;
    X:=Y*4;
    Writeln('Leaving AllVars: X=', X,';Y =',Y);
 End;
Begin
  G1:=10;
  G2:=20;
  Writeln('Calling AllValues:G1=',G1,';G2=', G2);
  AllValues(G1,G2);
  Writeln('Returning from AllValues:
    G1=',G1,';G2=',G2);
  Writeln;
  Writeln('Calling ValueVar:G1 =',G1,';G2=',G2);
  ValueVar(G1,G2);
  Writeln('Returning from ValueVar :
     G1=',G1,';G2=',G2);
  Writeln;
  Writeln('Calling AllVars:G1=',G1,';G2=',G2);
  AllVars(G1,G2);
  Writeln('Returning from AllVars :
     G1=',G1,';G2=',G2);
  Readln;
End.
```

عند تنفيذ البرنامج نحصل على مايلي:

```
Calling Allvalues:  G1=10; G2=20
Entering ALLValues: X=10;    Y=20
Leaving ALLValues:  X=20;    Y=80
Returning from ALLValues:  G1=10;  G2=20
Calling ValueVar:  G1=10;  G2=20
```

```
Entering ValueVar:     X=10;     Y=20
Leaving ValueVar:      X=20;     Y=80
Returning from ValueVar:  G1=10;  G2=80
```

```
Calling Allvars:  G1=10; G2=80
Entering ALLVars:     X=10;     Y=80
Leaving ALLVars:      X=20;     Y=320
Returning from ALLVars:  G1=20;  G2=320
```

مثال: (حساب محيط و مساحة مثلث).

```
Program name;
Uses crt;
Var
            x,y,z : integer;
            S,m : real;
            P: real ;
            t: Boolean;
begin
     clrscr;
     write('input x,y,z');
     Readln(x,y,z);
     t:=false ;
     p:=(x+y+z)/2 ;
     if((p-x)>0) and ((p-y)>0) and ((p-z)>0) then
        t:=true;
     if (t) then
     Begin
          S:=sqrt(p*(p-x)*(p-y)*(p-z));
          m:=2*p ;
          Write('s=',s,'m=',m)
```

```
End.
```

3. الدوال Functions

تمكن الدوال من حساب قيمة ما (واحدة فقط) وإعادتها إلى جسم البرنامج الرئيسي.

إنَّ الدالة **Sqrt** تحسب الجذر التربيعي لقيمة تُمرر لها. فعند استدعاء هذه الدالة مع متغير وسيطي **(argument)** ستعيد الدالة قيمة واحدة هي الجذر التربيعي للمتغير الممرر. و بالتالي العبارة **Sqrt (81.0)** سيعطي (9.0) كقيمة لهذه الدالة من أجل القيمة الابتدائية (81.0). وفي لغة الباسكال، الدوال هي روتينات تُعيد دائماً قيمة محددة واحدة فقط. والدوال هذه تملك نفس مكونات الإجرائيات من رأس وقسم تصريح وجسم الدالة. والصيغة الكتابية لتعريف الدالة والصيغة الكتابية لاستدعاء الدالة تختلفان عن مقابلاتها بالنسبة للإجرائيات. أما القواعد والقوانين المطبقة على الوسطاء لا تختلف بين الإجرائيات والدوال.

1.3 تعريف الدوال في لغة باسكال Defining Functions in Pascal

مثال: (حساب أكبر و أصغر عدد بين عددين).

```
Function Max(one, two : Real):Real;
  Begin
     If (One>Two) then
        Max:=one
     Else
        Max:=Two;
  End; (*FUNCTION Max:Real*)
Function Min(one,two:Real):Real;
  Begin
      If(one>two)then
         Min:=two
      Else
```

```
        Min:=one;
End;
```

التصريح عن دالة ما Function Declaration

تبدأ الدالة بالكلمة المحجوزة **Function**. يتبع الكلمة **Function** اسم الدالة ثم قائمة وسطاء اختيارية ينتهي رأس الدالة بنقطتين متبوعتين بنوع بيانات يُعطى لاسم الدالة (نوع البيانات التي ستعيدها الدالة). يمكن للدالة أن تعيد أي قيمة من أي نوع بيانات يُحدَد بعد النقطتين في رأس الدالة، ونوع البيانات هذا يمكن أن يكون أي نوع بيانات بسيط بالإضافة إلى نوع البيانات سلسلة المحارف وبعض أنوع البيانات الأخرى. جسم الدالة يقوم بحساب و إعادة قيمة الدالة، لذلك يجب أن تُعطى هذه القيمة إلى الدالة من خلال عبارة إسناد في أي مكان ضمن جسم الدالة، وعبارة الإسناد تجعل الدالة تساوي قيمة ما، أو تساوي قيمة أحد المتغيرات، أو تساوي ناتج تعبير معين ولكن بشرط أن تكون هذه القيمة من نفس نوع البيانات المخصص للدالة. إذ يوضع اسم الدالة على يسار معامل الإسناد دائماً وتوضع القيمة على يمين هذا المعامل.

يبدأ رأس الدالة بالكلمة المحجوزة **Function** وينتهي بنوع بيانات محدد يأتي بعد النقطتين. ويحتوي جسم الدالة على عبارة إسناد لإعادة القيمة التي تحسبها الدالة.

2.3 استدعاء الدالة Calling a Function

يُعتبر استدعاء الدالة في لغة الباسكال عبارة عن قيمة، لذلك يمكن أن يتم استدعاء الدالة في أي مكان يمكن أن توجد فيه قيمة هذه الدالة.

مثال:

اكتب برنامجاً (باستخدام الدوال) يمكن من حساب ضرب عددين صحيحين موجبين.

```
Program name ;
Uses crt ;
```

```
Var
            x,y,z:integer;
Function Mul(n,m:integer):integer;
        Begin
                Mul:=n*m;
        End;
Begin
     clrscr;
     write('input x,y');
     Readln(x,y) ;
     While((x<0) or (y<0)) do
       Begin
           Write('input x,y');
         Readln(x,y);
       End;
     z:=Mul(x,y);
     Write('z=',x,'*',y,'=',z);
 End.
```

مثال:

اكتب برنامجاً يمكن من حساب مضروب عدد صحيح موجب (العاملي).

```
Program name;
Uses crt;
Var
     x,y:integer;
Function Fact(m:integer):longint;
 Var
     I:integer;
     S:Longint;
```

```
Begin
     S:=1;
     For I:=1 to m do
       S=S*I;
     Fact:=S;
 End;
Begin
   clrscr;
   write('input x');
   Readln(x);
   While(x<0) do
     Begin
        write('input x= ');
        Readln(x);
     End;
   Y:=Fact(x);
   Write(x,'!=',Y);
End.
```

مثال:

اكتب برنامجاً بلغة الباسكال يُمَّكن من إيجاد العدد الأكبر والأصغر لعددين حقيقيين.

```
Program FnCalls;
Var
     Smaller,Larger,Val1,Val2:Real;
Function Max(one,two:Real):Real;
   Begin
        If(one>Two) then
                Max:=One
```

```
        Else
                Max:=Two;
    End;
Function Min(one,Two:Real):Real;
    Begin
        If(One>Two) then
                Min:=Two
        Else
                Min:=One;
    End;
Procedure GetReal(Message:String;Var Value:Real);
  Begin
       Write(Message,'     ');
       Readln(Value);
  End;  (* GetReal *)
Begin
      GetReal('Value 1 ? ', Val1);
      GetReal('Value 2 ? ', Val2);
      Smaller:=Min(Val1, Val2);
      Larger:=Max(Val1, Val2);
      Writeln('Val1=',Val1:10:3,';Val2
              =',Val2:10:3);
      Writeln('Smaller=',Smaller:10:3,';
      Larger=',Larger:10:3);
      Writeln('Min=',Min(Val1,Val2):10:
              3,'; Max=',Max(Val1,Val2):10:3);
      Writeln('Sum=',Min(Val1,Val2)+Max
             (Val1,Val2):10:3);
      Readln;
End.
```

يُمكن أن يوضَع استدعاء الدالة في الطرف الأيمن من عبارة الإسناد، أو أن يُمرَر استدعاء الدالة كمتغير وسيطي لإجرائيات أو دوال أخرى. بل يمكن أيضاً طباعته بشكل مباشر بدون إسناده إلى متحول وسيطي.

عندما يتم تنفيذ البرنامج ويصل هذا التنفيذ إلى عبارة استدعاء الدالة فإنه:

1. يتوقف تنفيذ الروتين المُستَدعي بشكل مؤقت ويُنقَل التحكم إلى الدالة. حيث تبدأ الدالة بدورها بتنفيذ تعليماتها.
2. تنفيذ الدالة هذا يتم وفق تسلسل التعليمات، إذ يُبدأ بتنفيذ أول تعليمة في جسم الدالة، ويستمر التنفيذ حتى تُحسَب قيمة مناسبة وتُسنَد لتلك الدالة.
3. بعد أن وتُسنَد هذه القيمة للدالة تُعاد هذه القيمة إلى الروتين المُستدعي والذي يستأنف تنفيذ تعليماته وفق قيمة جديدة حصل عليها من تنفيذ الدالة.

3.3 تمرير الوسطاء إلى الدوال Passing Parameters to Functions

تُمرَر الوسطاء إلى الدوال بقيمتها **(pass by value)**، أو يتم تمريرها مرجعياً **(pass by reference)**. التمرير بالقيمة هو الوضع الافتراضي لتمرير الوسطاء إلى الدوال أما التمرير المرجعي فيتم بوضع الكلمة المحجوزة **VAR** بجانب المتحول في أثناء التصريح عن الدالة. بما أن الدوال تُعيد قيمة يتم إسنادها إلى اسم هذه الدالة لذا سيكون احتياج الدوال إلى تغيير قيمة المتغيرات الوسيطية أقل من حاجة الإجراءات لذلك. و عليه التمرير بالمرجع غالباً ما يُستخدَم في الإجرائيات وقليلاً ما يُستخدَم في الدوال.

4. مفهوم التعاودية Recursion

إن الإجراءات والدوال يُعاد تحفيزها عكسياً **(reactivated backward)**، إذ إن آخر توقف مؤقت يكون أول إعادة تحفيز للدالة، وهكذا. هذه الخاصة القوية

```
Program name;
Uses crt;
Var
    x,y:integer;
Function Fact(M:integer):longint;
    Var
       I:integer;
       S:Longint;
  Begin
      If((M=0) or (M=1)) then
            Fact:=1;
      Fact:= M*Fact(M-1);
 End;
begin
    clrscr;
    write('input x');
    ReadLn(x);
    While(x<0) do
     Begin
       write('input x=');
       ReadLn(x);
     End;
    Y:=Fact(x);
    Write(x,'!=',Y);
End.
```

1.4 ميزات الإجرائية و الدالة التعاودية Properties of Recursive Functions

تتمتع الإجرائية و الدالة التعاودية بالميزات التالية:

1. تحتوي الإجرائية أو الدالة التعاودية على عبارة تنفيذ برمجية.
2. تتضمن الإجرائية أو الدالة التعاودية شرطاً (شرط التوقف) لتحديد فيما إذا كانت الدالة التعاودية تستطيع إنهاء عملية ببساطة أم يحتاج إلى استدعاء نفسها مرة أخرى.
3. بعد اختبار شرط التوقف، يتم تغيير قيمة الوسطاء قبل تمريرها إلى عبارة الاستدعاء الذاتي الواجب تضمينها ضمن هذه الإجرائية أو الدالة التعاودية. في الواقع عملية اختبار شرط التعاودية يجب أن تُوضع قبل الاستدعاء التعاودي، أما إذا أُجريت عملية الاختبار بعد الاستدعاء التعاودي فإن الإجرائية أو الدالة التعاودية لن تتوقف عن الاستدعاءات الذاتية.

ملاحظة:

هناك حالات يتم فيها عدم توقف الاستدعاءات التعاودية بسبب عدم تحقق الشرط التعاودي أبداً.

مثال:

الإجراء التالي لن يتوقف أبداً لأن قيمة val لن تصل إلى القيمة (1) أبداً.

```
Program ss;
Uses crt;
Var
    X:integer;
Procedure Count(Val:integer);
   Begin
      If(Val>1) then
        Begin
             Count(Val+1);
```

```
            Writeln(Val);
            Count(Val);
        End;
    End;
Begin
    Clrscr;
    Write('x=');
    Readln(x);
    While(x<0) do
      Write('x=');
    Readln(x);
    Count(x);
    Readln;
End.
```

مثال:

اكتب برنامجاً يولد سلسلة فيبوناتشي **Fibonacci Numbers** مستخدماً في ذلك استدعائين تعاوديين ضمن الروتين نفسه.

تُعرَّف هذه السلسلة كما يلي:

1. الحدان الأول والثاني يساويان الواحد فرضاً أي أن:

$$Fib(1) = 1 \text{ و } Fib(2) = 1$$

2. كل حد من حدود هذه السلسلة يساوي ناتج جمع الحدين السابقين له في هذه السلسلة، أي أن:

$x \geq 3$، $Fib(x) = Fib(x-1) + Fib(x-2)$

والجدول التالي يحتوي على أول خمس عشر حداً من حدود هذه السلسلة.

Number	Value
Source	

Fib(1) definition	1
Fib(2) definition	1
Fib(3) Fib(1) + Fib(2)	2
Fib(4) Fib(2) + Fib(3)	3
Fib(5) Fib(3) + Fib(4)	5
Fib(6) Fib(4) + Fib(5)	8
Fib(7) Fib(5) + Fib(6)	13
Fib8) Fib(6) + Fib(7)	21
Fib(9) Fib(7) + Fib(8)	34
Fib(10) Fib(8) + Fib(9)	55
Fib(11) Fib(9) + Fib(10)	89
Fib(12) Fib(10) + Fib(11)	144
Fib(13) Fib(11) + Fib(12)	233
Fib(14) Fib(12) + Fib(13)	377

البرنامج:

```
Program FibonacciDemo;
Var
    FibVal:integer;
```

```
Procedure GetInteger(Message:string;Var
          Value:integer);
  Var
     TempStr:String:
 Begin
       Write(Message,'    ');
       Readln(Value);
 End;
Function Fib(Val:Integer):Longint;
 Begin
    If(Val<=2) then
      Fib:=1
    Else
      Fib:=Fib(Val-2)+Fib(Val-1);
 End; (* FIN Fib *)
Begin
  Repeat
      GetInteger('Computing Fibonacci values?',
        FibVal);
      Writeln('Fib(',FibVal,')=',Fib(FibVal));
  Until(FibVal=1);
  Readln;
End.
```

تحسب الدالة Fib حدود سلسلة فيبوناتشي باستخدام الاستدعاءات التعاودية، فإذا كانت القيمة المُمرَرة إلى الوسيط val تساوي (1) أو (2) فإن الدالة سوف تتوقف مباشرة وبدون استدعاء تعاودي، وتُعيد القيمة (1)، وإلا فإن هذا التابع يقسم المشكلة إلى استدعاءين ذاتيين أحدهما يحسب القيمة Fib(n-1)، والثاني يحسب القيمة Fib(n-2)، وهذان الاستدعاءان بدورهما يقسمان المشكلة ويبسطانها إلى

حين الوصول إلى القيمة (1) أو (2) و التي عندها قيمة الدالة تساوي (1).

5. الإجراءات والدوال مسبقة التعريف Predefined Procedures and Functions

تملك لغة الباسكال كماً هائلاً من الإجراءات والدوال مسبقة التعريف الممكن استخدامها ضمن البرامج.

سوف نعرض بعض الإجرائيات والدوال مُسبَقة التعريف في لغة باسكال. معظم الإجرائيات والدوال مسبقة التعريف تأخذ متغيرات من أي نوع.

مثال:

```
Function ABS(IntVal:Shortint):Shortint;
Function ABS(IntVal:Integer):Integer;
Function ABS(IntVal:Longint):Longint;
Function ABS(RVal:Real):Real;
```

تُعيد كل دالة من الدوال السابقة القيمة المطلقة **(absolute value)** لمتغيراتها الوسيطية. يجب أن تكون القيم المُمرة إلى الوسطاء إما صحيحة ذات اشارة (تملك إشارة إما + أو −) أو قيم حقيقية. تكون القيمة المعادة من هذه الدوال من نفس نوع البيانات الذي ينتمي إليه المتغير الوسيطي. القيمة المطلقة لعدد تساويه إذا كان عدداً موجباً و تساوي معاكسه إذا كان عدداً سالباً.

مثال:

```
Program AbsDemo;
Var
   RVa1, Result : Real;
Procedure GetReal(Message:String;
    Var Value:Real);
  Begin
      Write(Message,'    ');
      Readln(Value);
```

```
  End; (* GetReal *)
Begin (* Main Program *)
    Repeat
       GetReal('Value(> 500 to stop )?',RVa1);
       Result:=ABS(RVal);
       Writeln('ABS(',RVa1:8:3,')=',Result:8:3);
    Until(RVa1>500.0);
    Readln;
End.
```

دالة إيجاد الظل العكسي لزاوية ما Arctan()

Function arctan (Value : Real) : Real;

تُعيد هذه الدالة قيمة مثلثية **(trigonometric value)** عبارة عن قيمة الزاوية التي ظلها **(tangent)** ممرر ضمن المتغير الوسيطي الذي يجب أن يكون قيمة حقيقية، وسوف تُعيد هذه الدالة قيمة حقيقية.

بعض الدوال الرياضية مسبقة التعريف

- دالة الجذر التربيعي لعدد x : $\sqrt{x}$ شكلها البرمجي: Sqrt(x)
- دالة القيمة المطلقة لعدد x : $|x|$ شكلها البرمجي: abs(x)
- دالة مربع لعدد x : x^2 شكلها البرمجي: Sqr(x)
- دالة الجيب لزاوية x مقدر بالراديان: $\sin x$ شكلها البرمجي: sin(x)
- دالة الظل لزاوية x مقدر بالراديان: tgx شكلها البرمجي: tan(x)
  - دالة تُعطي باقي قسمة العدد الصحيح a على العدد الصحيح b شكلها البرمجي: c:=a mod b
- دالة ناتج القسم الصحيح لـ a على b شكلها البرمجي: c:=a div b
- دالة الزاوية التي ظلها x ، arctg(x) شكلها البرمجي: Arctan (x)

- دالة التابع الأسي e^x شكلها البرمجي: Exp(x)
- اقتطاع الجزء الصحيح من العدد شكله البرمجي: Trunc (x)
- تقريب العدد إلى أقرب عدد صحيح الشكل البرمجي: Round (x)
- دالة اللوغاريتم الطبيعي لعدد $\ln x, 0 < x$ شكلها البرمجي: Ln(x)

مثال:

برنامج حل معادلة من الدرجة الثانية: $ax^2 + bx + c = 0$.

```
Program name;
Uses crt;
Var
      a,b,c:real;
      x,x1,x2:real;
      D:real;
Begin
   clrscr;
   writeln('input a,b,c');
   Readln(a,b,c);
   D:=b*b-4*a*c;
   If((a=0) and (b=0)) then
      write('c=0 أو a,b,c لا تشكل معادلة')
   Else if(a=0) then
      Begin
           x:=-c/b;
           write('x=', x:10:2);
      End
   Else if(D>=0) then
      Begin
```

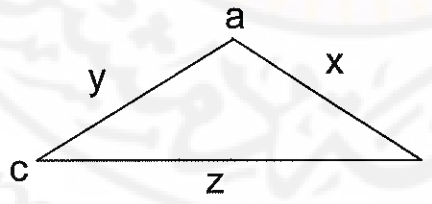
```
            x1 := (-b + Sqrt(D))/(2*a);
            x2 := (-b - Sqrt(D))/(2*a);
            write(x1,x2);
        End
    Else
     Begin
        D:=-D;
         writeln('x1 =',-b/(2*a),'+i',Sqrt(D)/(2*a));
          writeln('x2 =',-b/(2*a),'-i',Sqrt(D)/(2*a));
     End;
 Readln;
End.
```

مثال:

اكتب برنامجاً بلغة الباسكال يقوم بإدخال ثلاث أعداد واختبار فيما إذا كانت تُشكل هذه الأعداد أضلاعاً لمثلث أم لا، وفي حالة الإيجاب يُطلب حساب مساحة المثلث ومحيطه.

نعلم أن في أي مثلث لدينا: $S = \sqrt{p(p-x)(p-y)(p-z)}$

حيث S مساحة المثلث و p نصف محيط المثلث



وهذه العلاقة تتحقق إذا كان ما تحت الجذر موجباً تماماً، أي إذا كان:

$$p-z>0 \quad , \quad p-y>0 \quad , \quad p-x>0$$

وبالتالي يكون البرنامج.

```
Program name ;
Uses crt;
Var
    x,y,z:real;
    S,m:real;
    P:real;
    t:Boolean;
Begin
  clrscr;
  write('input x,y,z');
  Readln(x,y,z);
  t:=false;
  p:=(x+y+z)/2;
  If((p-x)>0)and((p-y)>0)and((p-z)>0)) then
    t:=true;
  if(t) then
     Begin
          S:=Sqrt(p*(p-x)*(p-y)*(p-z));
          m:=2*p;
          Write('s=',s,'m=',m);
     End
  Else
          write('x,y,z لاتصلح لأن تكون أضلاعاً لمثلث');
 Readln;
End.
```

تمارين

1) ضع برنامجاً بلغة الباسكال يقوم بإدخال عددين صحيحين موجبين تماماً a,b و طباعة المضاعف المشترك الأصغر و القاسم المشترك الأكبر لهما و ذلك بإستخدام البرامج الجزئية.

2) ضع برنامجاً بلغة الباسكال يقوم بما يلي :

1. إدخال متجهة A من الأعداد الصحيحة ذات البعد n.
2. ترتيب عناصر المتجهة A ترتيباً تصاعدياً.
3. طباعة جميع الأعداد الأولية من المتجهة A.
4. طباعة العناصر التامة من المتجهة A.
5. طباعة المتوسط الحسابي و الانحراف المعياري و المنوال.
6. طباعة قواسم كل عنصر من عناصر المتجهة A.
7. تحويل كل عنصر من عناصر A إلى النظام الثنائي.

3) ضع برنامجاً بلغة الباسكال يقوم بما يلي:

1. إدخال مصفوفة مربعة A من الأعداد الصحيحة الموجبة من البعد $n \times n$.
2. إيجاد منقول المصفوفة A وبيان فيما إذا كانت A متناظرة.
3. إيجاد أصغر عنصر للمصفوفة A مع تحديد مكان وقوعه.
4. حساب المضاعف المشترك الأصغر لعناصر المصفوفة A.
5. طباعة جميع الأعداد الأولية الموجودة في المصفوفة A.

السجلات Records

1. نوع البيانات سجل The Record Data Type

في معظم التطبيقات يكون لدينا مجموعة من البيانات من أنواع مختلفة و التي تخص كائناً معيناً (ذاتية شخص, معلومات عن كتاب, .. إلخ). من المفيد في مثل هذه الحالة تعريف بنية واحدة تجمع كل هذه البيانات. نسمي مثل هذه البنية المركبة بـ سجل.

مثال توضيحي.

لتمثيل بيانات حول شخص ما والتي تتضمن الاسم (Name) والكنية (LastName) والراتب (Salary). هذه البيانات من أنواع بيانات مختلفة، حيث أن الاسم والكنية تكون من نوع سلسلة محارف String أما الراتب فهو عدد صحيح Integer. بما أن هذه البيانات تخص شخصاً معيناً وهي من أنواع مختلفة فإنه لتمثيل هذه البيانات نستخدم النوع المركب سجل.

كما نعلم أنه لتعريف نوع جديد بلغة الباسكال نستخدم الكلمة المحجوزة Type على النحو التالي:

```
Type
    Person=Record
      Name,LastName:String[50];
      Salary:integer;
    End;
```

لقد عرفنا في المثال السابق نوع بيانات سجل يحتوي على ثلاثة حقول هي الاسم (Name) و الكنية (LastName) وهما من النوع شريط محرفي بطول (50)، والراتب (Salary) من النوع الصحيح. تم التصريح عن هذه الحقول بين الكلمتين

المحجوزتين Record و End حيث يُصرح عن كل حقل بالطريقة المعروفة في لغة الباسكال (نوع بيانات الحقل : اسم الحقل).

بشكل عام التركيب النحوي لسجل يُعطى بالصيغة التالية:

```
Type
     RecordName=Record
        Field1 : type1;
        Field2 : type2;
        .
        .
        Fieldn : typen;
     End;
```

حيث $Field_i$ تُمثل أسماء الحقول المراد تعريفها و $type_i$ تُمثل نوع هذا الحقل. وهذه الأنواع يمكن أن تكون مختلفة و في حالة وجود أكثر من حقل من نفس النوع فإنه يمكن كتابة هذه الحقول على سطر واحد[1].

2. الوصول إلى الحقول ضمن السجل Accessing Fields in a Record

بعد تعريف متحول من نوع سجل، يتم الوصول إلى أي حقل في هذا السجل كمايلي: اسم الحقل.متحول السجل.

مثال 1:

```
Type
  Person=Record
    Name,LastName:String[50];
    Salary:integer;
```

[1] تعريف السجل هو أحد المواضع الثلاثة في لغة باسكال، والتي نضع فيها الكلمة المجوزة End بدون Begin وقد مر معنا موضع آخر هو عبارة CASE.

```
  End;
Var ThePerson:Person;
```

يبين المقطع البرمجي السابق كيفية تعريف نوع سجل و من ثم تعريف متحول من ذلك النوع. في البداية و بعد تعريف نوع السجل Person، تم التصريح عن المتحول ThePerson من هذا النوع.

يمكن إعطاء قيمة ابتدائية للحقل Name من السجل Person كمايلي: ThePerson.Name := "Ahmad". كما يمكن أيضاً إدخال هذه القيمة من خلال لوحة المفاتيح باستخدام التعليمة Read(ThePerson.Name).

مثال 2:

المقطع البرمجي التالي يوضح كيفية التعامل مع حقول السجل.

```
Program MyProgram;
Type
   Person=Record
    Name,LastName:String[50];
    Salary:integer;
 End;
Var ThePerson:Person;
Procedure InitiateRecord(Var P1:Person; Myname,
      MyLast:String[50]; MySal:integer);
  Begin
    P1.Name:=MyName;
    P1.LastName:=MyLast;
    P1.Salary:=MySal;
  End;
Procedure DisplayRecord(q1:Person);
  Begin
    Write('The Name:',q1.Name);
```

```
    Writeln('The Last Name:',q1.LastName);
    Writeln('The Salary:',q1.Salary);
  End;
```

يقوم هذا المقطع البرمجي بتوضيح كيفية التعامل مع السجل و تمريره إلى الإجرائيات (Procedures) حيث إنه يمكن تمرير متحول من نوع سجل إلى الإجرائيات. يتم ذلك إما بالمرجع أو بالقيمة وهذا يشبه عملية تمرير متغيرات من أنواع البيانات الأخرى إلى الدوال.

عندما يُستدعى الإجراء InitiateRecord و بسبب التمرير المرجعي (باستخدام الكلمة المحجوزة var) فإن أي تغيير على الوسيط الشكلي (P1) المُمَرر إلى الإجرائية سيؤثر في الوسيط الفعلي الذي سوف يُستخدم عند استدعاء هذه الإجرائية في البرنامج العام. و بالتالي جميع القيم التي تم تخزينها في حقول الوسيط الشكلي ستُخَزن في حقول الوسيط الفعلي.

عندما يُستدعى الإجراء DisplayRecord فإن محتويات السجل الفعلي تُنسخ، وتُؤخذ هذه النسخة وتُخزن على أنها محتويات للسجل q1 ومن ثم يُظهر الإجراء محتويات هذا السجل على الشكل التالي:

```
The Name      : Ahmad
The Last Name : Ali
The Salary :  2500
```

3. الحقول المركبة ضمن سجل Fieldes of Composite type in a Record

يمكن لحقول السجل أن تكون من أنواع قياسية مختلفة (صحيح, حقيقي, محرفي, .. إلخ), و يمكن تعريف حقول مركبة من نوع متجهات, مصفوفات أو

حتى سجلات. في حالة كون حقل أو أكثر من النوع سجل فإننا ندعو هذه الظاهرة بالسجلات المتداخلة (Nested Records).[2]

مثال:

```
Type
   date=Record
     day: 1..31;
     month: 1..12;
     year: 1985.. 1993;
   End;
  Mark=array[1..5] of 0..100;
  Student=Record
    Name,lastName:String[50];
    StudentID:Longint;
    Marks:mark;
    Birth:date;
  End;
```

يُعَرف المقطع السابق السجل Student لطالب جامعي. يملك السجل Student خمسة حقول: الحقلان Name, LastName من النوع القياسي سلسة محارف وهما يُمثلان الاسم و الكنية، الحقل StudentID من النوع عدد صحيح طويل و يُمثل رقم الطالب، الحقل Marks من النوع المركب متجهة و يُمثل العلامات الفصلية لطالب في خمس مواد. أما الحقل Birth فهو من النوع المركب سجل date وبالتالي وجب تعريف سجل من هذا النوع قبل تعريف السجل Student.

يمكن الوصول إلى الحقل المركب (من نوع متجهة مثلاً) بالطريقة نفسها التي تُستخدم للوصول إلى حقل بسيط. بعد الوصول إلى بيانات الحقل المركب يتم

---

[2] لا يمكن لأي حقل أن يكون من النوع ملف (File), سوف نناقش النوع ملف في الفصل الرابع من هذا الكتاب.

التعامل معها بالطرق المعروفة. يُمكننا الوصول إلى الحقول في السجلات المتداخلة عن طريق متحول السجل متبوعاً بمتحول السجل الداخلي متبوعاً باسم الحقل.

في مثالنا السابق يمكن الوصول إلى الحقل year في السجل Student وذلك بكتابة العبارة stud.birth.year حيث stud هو متحول من النوع Student.

استخدام عبارة With (With statement)

تُستَخدَم تعليمة With بهدف التخلص من تكرار كتابة اسم متحول السجل عند محاولة الوصول إلى حقل ما ضمن ذلك السجل. هذه التعليمة مفيدة بشكل خاص عندما يكون لدينا مجموعة من السجلات المتداخلة حيث يصبح تكرار متحول السجل بعدد مستويات التداخل.

التركيب النحوي للتعليمة **With** على الشكل التالي:

```
With Rn do
 [Begin]
     Statements;
 [End;]
```

حيث **With** و **Do** كلمات مفتاحيه, Rn هو متحول سجل أما Statements هي تعليمة أو أكثر تشتمل على أسماء حقول موجودة في السجل. في حال كانت Statements أكثر من تعليمة واحدة فيجب وضعها ضمن Begin و End.

مثال 1:

بالعودة إلى مثالنا السابق و الذي يُعرف سجل تاريخ ميلاد طالب (سجل غير متداخل).

```
Type
   date=Record
     day: 1..31;
```

```
      month: 1..12;
      year: 1985..1993;
    End;
Var birth:date;
```

يمكننا الوصول إلى أي حقل من حقول السجل Birth باستخدام التعليمة With على النحو التالي:

```
With birth do
 Begin
   Readln(day);
   month:=7;
   write(year);
 End;
```

لاحظ أنه تم الوصول إلى حقول السجل Birth بشكل مباشر و ذلك بفضل استخدام التعليمة **with**.

مثال 2:

```
Type
   date=Record
     day:1..31;
     month:1..12;
     year:1985..1993;
   End;

  Student=Record
   Name,lastName:String[50];
   StudentID:Longint;
   Birth:date;
  End;
Var stud:student;
```

في هذا المثال لدينا السجل المتداخل Student حيث عرفنا متحول stud من نوع هذا السجل. من أجل الوصول إلى حقول السجل Stud نستخدم التعليمة **With** كما في المثال السابق.

من أجل الوصول إلى حقول السجل Birth و الذي بدوره حقل من السجل Stud فإننا نستخدم إحدى الطريقتين التاليتين:

- **طريقة أولى:**

```
With stud,birth do
  Begin
    Readld(day);
    month:=7;
    write(year);
  End;
```

- **طريقة ثانية:**

```
With stud do
  With birth do
    Begin
      Readln(day);
      month:=7;
      write(year);
    End;
```

يجب الانتباه إلى أن الترتيب مهم في الطريقتين حيث نضع اسم السجل الحاوي قبل أسماء السجلات المحتواة بداخله باستخدام التعليمة **With**. لاحظ أننا لم نضع Begin و end للتعليمة with الخاصة بالسجل stud وذلك لأن تعليمة with الخاصة بالسجل stud لا تحوي إلا تعليمة واحدة و هي تعليمة with الخاصة بالسجل birth و التي بدورها تغلف مجموعة من التعليمات الخاصة بها.

فضاء الأسماء Namespaces

يمكن تعريف متحولين بنفس الاسم وليكن X. الأول كحقل من سجل و الأخر خارج هذا السجل. يقوم المترجم بالتمييز بين هذين المتحولين كمايلي:

1. إذا ذُكر اسم متحول السجل الحاوي على X أو استخدمنا عبارة with للوصول إلى حقول السجل فإن X المقصودة هنا هي حقل السجل.
2. خلاف ذلك فإن X المقصودة هي المتحول المعرف خارج السجل.

مثال:

```
Type
   Student=Record
     Name,lastName:string;
     StudentID:longint;
   End;
Var
   stud:student;Name:String;
```

نلاحظ في المقطع البرمجي أعلاه أن الاسم Name مُعرف كحقل في السجل Student و أيضاً كمتحول خارج السجل معرف ضمن عبارة var.

- عند كتابة العبارة write(Name) فإن Name المقصودة هي المتحول الخارجي المُعرف ضمن عبارة var التابعة للبرنامج العام.
- عند كتابة العبارة write(stud.Name) مثلاً فإن Name المقصودة هنا هي حقل السجل stud. و في هذه الحالة يمكن استخدام عبارة With كما بينّا سابقاً.

4. متجهات السجلات Arrays of Records

عندما يكون لدينا مسألة تُعالج أكثر من سجل واحد من نفس النوع، فإنه من المُفضل وضع هذه السجلات في نوع بيانات يُسّهل عملية التعامل معها. في مثل هذه الحالة تعد المتجهات نوع بيانات مناسب لأنه يُمَكن من احتواء مجموعة من

البيانات ذات النوع الواحد، وبالتالي يمكننا الوصول إلى هذه البيانات عن طريق مرجع واحد (المتحول المتجهي).

يُمكن تعريف متجهة من نوع البيانات سجل. عندئذٍ كل عنصر من عناصر هذه المتجهة هو من النوع سجل.

مثال:

```
Type
   Student=Record
     Name,lastName:string;
     StudentID:Longint;
   End;
 Students=array[1..10] of Student;
Var studs:Students;
```

في المقطع البرمجي أعلاه تم تعريف نوع البيانات Students كمتجهة سجلات من النوع Student. المتحول studs هو عبارة عن متجهة يمكنها تخزين سجلات 10 طلاب. من أجل الوصول إلى حقل من سجل ضمن المتجهة فإنه يجب أولاً الوصول إلى العنصر المناسب عن طريق دليله ثم الوصول إلى ذلك الحقل كما بينا سابقاً. فمثلاً من أجل الوصول إلى اسم الطالب الثالث فإننا نكتب studs[3].Name.

5. إسناد السجلات Record Assignment

يُمكن إسناد محتويات سجل إلى سجل أخر من النوع نفسه. تتم عملية الإسناد من خلال المعامل " =: ". في هذه الحالة يتم إسناد قيمة كل حقل من حقول السجل المُسنَد (الموجود على يمين المعامل) إلى الحقل المقابل في السجل المُسنَد إليه (الموجود على يسار المعامل) بشكل مماثل للمتحولات البسيطة (مثلاً من النوع integer). على الرغم من أن القيم المخزنة في حقول السجلين متطابقة بعد

عملية الإسناد إلا أن كل واحد من السجلين يحجز مكاناً منفصلاً في الذاكرة. و عليه يمكن التعامل مع أي من السجلين بشكل مستقل عن الأخر.

6. السجل المتبدل Variant Record

السجل المتبدل هو عبارة عن سجل يحتوي في حقل أو أكثر على عدة حالات و كل حالة تتطلب تخزين معلومات مختلفة. الهدف هنا هو تخزين المعلومات المحددة لتلك الحالة. فمثلاً عندما نتكلم عن سجل شخص فإن أحد الحقول و الذي يمثل الوضع العائلي له حالتان متزوج أو أعزب. في بعض الأحيان إذا كان الشخص متزوجاً يهمنا المحددات التالية: الطول و العمر. أما إذا كان عازب فيهمنا المحددات التالية: الوزن و الجنسية.

يُعرف السجل المتبدل بلغة الباسكال وفق التركيب النحوي التالي:

```
Type
    T=record
        S1 : t1 ;
        S2 : t2 ;
        …
        Sn-1 : tn-1 ;
        Case sn  : tn of
          W1: (s11: t11; s12: t12; … ; s1m: t1m);
          …
          Wk : (sk1: tk1; sk2:tk2; … ; skm: tkm);
        End;
    End;
```

نلاحظ في الصيغة السابقة أن الحقول من S1 إلى Sn-1 هي حقول ثابتة تُعرف كما عرفنا حقول السجلات سابقاً. أما الحقل Sn من النوع tn فهو حقل متبدل لأن له حالات عدة هي على التوالي من W1 إلى Wk و تختلف المتطلبات

التخزينية بحسب الحالة. على سبيل المثال الحالة W1 لها عدة متحولات هي على التوالي S11 من النوع t11 و S12 من النوع t12، ... ، S1m من النوع t1m.

مثال:

سنقوم بتعريف سجل شخص و له الحقول التالية: الاسم First ، الكنية Last و هي حقول ثابتة و حقل متبدل يمثل الوضع العائلي status من النوع MaritalStatus و الذي بدوره يحتوي على حالتين single و married. في الحالة Single نهتم بالمحددات التالية: سوري أم لا IsSyrian و الوزن Weight. أما في حالة متزوج married نهتم بالمحددات التالية: الطول height و اسم الزوجة wifeName. مجموعة التعليمات التالية تُبين كيف يتم تعريف هذا السجل بلغة الباسكال.

```
Type
  MaritalStatus=(single,married);
  Person=Record
    First,Last:String;
    Case status:MaritalStatus of
      Single:(IsSyrian:boolean;weight:real);
      Married:(height:real;WifeName:String);
    End;
  End;
Var
   per:Person;
```

تبدو التصريحات ضمن المقطع البرمجي السابق سهلة وبسيطة، فالسجل person يحتوي على جزأين أحدهما ثابت والآخر متبدل. إضافة إلى حقول الجزء الثابت، الجزء المتبدل يبدأ بالعبارة CASE status حيث يستطيع السجل person

تخزين إحدى الحالتين المختلفتين single أو married حسب قيمة الحقل status ضمن السجل.

إذا كانت قيمة الحقل per.status هي single فإن السجل per سوف يُخَزن قيمتين فقط هما قيمة بوليانية في المتحول IsSyrian و قيمة حقيقية في المتحول Weight.

أما إذا كانت قيمة الحقل per.statu هي married، فإن السجل per سوف يُخَزن قيمتين مختلفتين هما قيمة حقيقية في المتحول height و قيمة شريطية (سلسلة محارف) في المتحول wifeName .

7. السجلات الثابتة Constant Records

يمكن تعريف سجلات ثابتة في لغة باسكال، أي تعريف سجل وتحديد اسم لهذا السجل ليكون هذا الاسم مرتبطاً تماماً بمجموعة قيم ثابتة مُخزنة في حقول هذا السجل. وهذه القيم لن تتغير طوال فترة تنفيذ البرنامج. يتم ذلك وفق التركيب النحوي التالي "("<قيم>")"="<نوع بيانات سجل>":"<كلمة محجوزة>".

مثال:

يبين البرنامج التالي كيفية تعريف وتهيئة السجلات الثابتة.

```
Program RecConstantDemo;
Type
   DailyWeather=Record
     HiTemp,LoTemp,Wind:Real;
   End;
  DaysData=array[1..2] of Real;
  DaysWeather=Record
    DaysHi,DaysLo,DaysWind:DaysData;
  End;
Const
```

```
 ExtremeWeather:DailyWeather=(HiTemp:115.6;
  LoTemp:-89.9;Wind:183.2);
 ExtremeDays:DaysWeather=(DaysHi:(101.6, 115.6);
  DaysLo:(-89.9,-88.7);DaysWind:( 183.2, 181.8));
Begin
   With ExtremeWeather do
      Begin
        Writeln('HiTemp=',HiTemp:12:2);
        Writeln('LoTemp=',LoTemp:12:2);
        Writeln('Wind=',Wind:12:2);
      End;
  With ExtremeDays do
     Begin
       Writeln('<Data>   <Midnight>   <Noon>');
       Writeln('DaysHi =',DaysHi[1]:12:2,DaysHi
         [2]:12:2);
       Writeln('DaysLo =',  Dayslo[1]:12:2,
         DaysLo[2]:12:2);
       Writeln('Daysaind = ',DaysWind[1] :
         12:2, DaysWind[2] : 12:2);
    End;
End.
```

تُحدَد قيم السجل الثابت ضمن أقواس كما هو الحال بالنسبة للمصفوفات الثابتة، إذ يذكر اسم الحقل متبوعاً بنقطتين، ثم قيمة هذا الحقل. والصيغة الكتابية لتعريف القيم هي:

<اسم الحقل>":"<قيمة>

نضع القيم في السجل الثابت وفق الترتيب الصحيح لحقول السجل كما وردت ضمن تعريف السجل الأصلي، فاصلين بين حقول السجل بفاصلة منقوطة. لذلك

حددنا ضمن السجل الثابت ExtremeWeather ثلاثة حقول هي أسماء حقول السجل الأصلي، وأُتبعت هذه الأسماء بنقطتين، ومن ثم القيمة المراد تخزينها في هذا الحقل. أما السجل الثابت ExtremeDays والذي حقوله عبارة عن مصفوفات كانت عملية تحديد قيم حقول هذا السجل تخضع لاتحاد طريقتي تهيئة المصفوفات الثابتة والسجلات الثابتة، إذ وُضِعت قيم الحقل ضمن قوسين وأُلحِقت باسم الحقل.

هناك نوع آخر من الأنواع الثابتة يمكننا إنشاؤه وهي مصفوفة ثابتة تتألف من سجلات ثابتة.

8. الكائنات Objects

عرفنا في الفقرات سابقة نوع البيانات سجل. كما وجدنا السجل هو بنية منطقية تحوي حقولاً لتخزين بيانات حول شيء ما (شخص، مقرر جامعي،.. إلخ). المشكلة الرئيسية مع السجلات هي أن أي مُستخدم يمكنه الوصول إلى حقول السجل و التعامل معها (قراءتها، تغيير قيمها، طباعتها، .. إلخ). هذا يناقض مبدأ أساسي في البرمجة و الذي يسمى مبدأ التغليف Encapsulation. استجابةً لهذا المبدأ مَكّنت لغة الباسكال من تحقيق مبدأ التغليف عن طريق تعريف نوع اسمه "كائن" أو Object. و ينص مبدأ التغليف على أن كل كائن مسؤول عن بياناته الخاصة حيث يتم التعامل مع تلك البيانات فقط عن طريق هذا الكائن.

الكائن هو نوع يحوي حقولاً تسمى معرفات أو واصفات (Attribute) و يحوي عمليات (دوال أو إجراءات) تقوم بمعالجة هذه البيانات والتواصل مع العالم الخارجي. بهذا الشكل يكون كل كائن مسؤول عن بياناته و يمكن للكائنات الأخرى الاتصال مع هذا الكائن للحصول على خدمة بدلاً من التعامل المباشر مع البيانات الخاصة بالكائن و ذلك عن طريق العمليات المعرفة ضمن هذا الكائن.

في لغة الباسكال يمكن التصريح عن أي كائن بطريقة مشابهة للتصريح عن السجلات و ذلك وفق مايلي:

```
Type
   ObjectType=Object
        AttributeList;
        Method-Signature list;
     private
        PAttribute-list;
        PMethod-Signature list;
   End;
```

حيث AttributeList هي مجموعة من الحقول العامة المعرفة ضمن الكائن. Method-Signature list; هي مجموعة من الدوال و الإجراءات العامة. أي أن الحقول و الدوال و الإجراءات المعرفة هنا متاحة لكل الكائنات الأخرى و يمكن لكل كائن الوصول إليها. أما الحقول و الدوال و الإجراءات المعرفة بعد الكلمة المفتاحية private فهي خاصة بالكائن و لا يمكن لأي كائن آخر الوصول إليها إلا عن طريق الدوال و الإجرائيات العامة المعرفة ضمن هذا الكائن. يظهر مفهوم مبدأ التغليف مباشرةً من التعريف حيث إنه من أجل كل كائن لا يريد للكائنات الأخرى الوصول إلى جزء من بياناته فإننا نضع هذا الجزء من البيانات في القسم الخاص (بعد الكلمة المحجوزة Private).

مثال:

تُعَرف مجموعة التعليمات التالية نوع كائن اسمه Entry.

```
Type
   Entry=Object
     Procedure init(AName,ANumber,AStatus:
       String);
     Function getName:String;
   Private
     Function getCustomerStatus:String;
     Name,Number,status:String;
```

```
End;
```

لاحظ أن الكائن Entry لا يسمح بالوصول إلى الحقول الثلاثة Name, Number, Status إلا من خلال الدوال و الإجراءات التي يوفرها (التي يمتلكها). بكلمات أخرى، يمكن للكائنات الأخرى الوصول إلى الإجراء Init و الدالة getName فقط أما بقية العناصر فهي مغلفة ضمن الكائن Entry و لا يمكن الوصول إليها بشكل مباشر.

حتى يكون النوع كائناً له معنى فيجب كتابة التعليمات البرمجية الخاصة بالدوال و الإجراءات. يتم ذلك في أي مكان في البرنامج بعد تعريف النوع و قبل تعليمة بداية البرنامج العام Begin و المثال التالي يوضح كيفية كتابة التعليمات البرمجية لدوال أو إجراءات كائن معين.

مثال:

سوف نستخدم الكائن Entry المُعَرف في المثال أعلاه و نكتب التعليمات البرمجية للإجراء init. لزيادة التوضيح سوف نعيد كتابة الكائن Entry هنا.

```
Type
   Entry=Object
     Procedure init(AName,ANumber,AStatus:
       String);
     Function getName:String;
   Private
     Function getCustomerStatus:String;
     Name,Number,status:String;
   End;
```

البرنامج:

```
Program ObjIllustartion;
```

```
Type
 Entry=Object
   Procedure init(AName,ANumber,AStatus:String);
   Function getName;
 Private
   Function getCustomerStatus; Name,Number,
   status:String;
 End;
  ...
 Procedure Entry.init(AName,ANumber,
        AStatus:String);
   Begin
      Name:=AName;
      Number:=ANumber;
      Status:=AStatus;
   End;
 …
 Begin {Main program}
 …
 End. {Main program}
```

كما تُلاحظ فإن المقطع البرمجي الجزئي السابق يُصرح عن نوع بيانات كائن اسمه Entry. بعد ذلك قمنا بكتابة التعليمات البرمجية للإجراء init.

هناك نقطتان يجب على القارئ ملاحظتهما. النقطة الأولى هي أن كتابة التعليمات البرمجية تتم بعد التصريح عن النوع كائن و قبل تعليمة بداية البرنامج العام Begin. النقطة الثانية يجب وضع اسم الكائن متبوعاً بنقطة ثم اسم الإجرائية (أو الدالة) و ذلك لكي يتمكن المترجم من معرفة تبعية الإجرائية (أو الدالة). النقاط "..." في البرنامج أعلاه تعني إمكانية وجود تعليمات برمجية أخرى في المكان المُحَدد. بمعنى آخر ليس من الضروري كتابة تعليمات الإجرائية أو

الدالة المُعرفة ضمن الكائن بعد التصريح عن الكائن مباشرةً بل يمكن أن تكون في أي مكان بعد التصريح و لكن قبل بداية البرنامج العام.

استخدام الكائن ضمن البرنامج

يمكن تعريف متحول من نوع كائن بذات الطريقة التي نُعرف فيها المتحولات من أنواع أخرى. مثلاً: نُعرف التعليمة التالية متحول من نوع Entry.

```
Var MyEntry:Entry;
```

بتعريف متحول من نوع كائن ، فإنه يمكن الوصول إلى معرفات (حقول) ، أي دوال و إجراءات هذا الكائن باستخدام المتحول. مثلاً: تستدعي التعليمة التالية الإجراء init.

```
MyEntry.init('Ahmad','011-213-1263','good');
```

لاحظ كيف تم الاستدعاء حيث وضعنا متحول الكائن MyEntry متبوعاً بنقطة ثم اسم الإجراء. بالطبع إذا كان الإجراء له وسطاء فيجب تمرير القيم المناسبة لهذه الوسطاء كما في الاستدعاء أعلاه. لاحظ أيضاً أن التعليمة التالية خاطئة:

```
MyEntry.getCustomerStatus;
```

لأن الدالة getCustomerStatus هي دالة خاصة بالكائن و لا نستطيع استدعاؤها من خارج الكائن (أي لا يمكن استدعاء هذه الدالة إلا من خلال كائن يحويها).

يمكن أيضاً تعريف متجهة كائنات بالطريقة نفسها التي نُعرف فيها متجهة سجلات مثلاً، و يمكن أن يتم ذلك من خلال عبارة تعريف الأنواع Type أو مباشرةً باستخدام تعليمة التصريح عن المتحولات Var.

مثال:

تعريف متجهة كائنات من النوع Entry باستخدام العبارة Type.

```
Type
 {Entry تعريف الكائن }
   ObjArray=Array[1..100] of Entry;
   Var Obj: ObjArray;
```

حيث تُعرف التعليمة أعلاه متجهة من 100 كائن من النوع Entry.

يمكن تعريف متجهة كائنات ضمن تعليمة التصريح Var بالشكل:

```
Var Obj:array[1..100] of Entry;
```

في كلتا الحالتين سيتم تعريف متجهة من 100 كائن و لكن من ناحية برمجية فإننا نفضل التعريف الأول لأنه أوضح و لأنه يُمَكّن من تحقيق مبدأ هام في مجال هندسة البرمجيات هو مبدأ إعادة الإستخدامية (reusability).

يمكّن التعامل مع الكائنات ضمن المتجهة باستخدام الحلقات. مثلاً تقوم التعليمات البرمجية التالية بطباعة الاسم و ذلك باستدعاء الدالة getName على كل كائن في المتجهة.

```
For i:=1 to 100 do
   Write('Name: ',Obj.getName);
```

من الميزات المهمة للكائنات هي أنها تُوفّر حماية للحقول المُعرفة فيها و التالي تسمح للمبرمج بالحد من الوصول غير المسموح به لهذه الحقول. لكن لعل يزة الأكثر أهمية للكائنات أنها تسمح بإعادة الاستخدام. حيث يمكن للمبرمج و ي كتب كائن لحساب مثلاً مساحات بعض الأشكال الهندسية أن يُعيد استخدامه أي برنامج آخر عند الضرورة من دون الحاجة لإعادة كتابة كامل البرنامج.

الدالة المُعرفة ضمن الكائن بعد التصريح عن الكائن مباشرةً بل يمكن أن تكون في أي مكان بعد التصريح و لكن قبل بداية البرنامج العام.

استخدام الكائن ضمن البرنامج

يمكن تعريف متحول من نوع كائن بذات الطريقة التي نُعرف فيها المتحولات من أنواع أخرى. مثلاً: تُعرف التعليمة التالية متحول من نوع Entry.

```
Var MyEntry:Entry;
```

بتعريف متحول من نوع كائن ، فإنه يمكن الوصول إلى معرفات (حقول) ، أي دوال و إجراءات هذا الكائن باستخدام المتحول. مثلاً: تستدعي التعليمة التالية الإجراء init.

```
MyEntry.init('Ahmad','011-213-1263','good');
```

لاحظ كيف تم الاستدعاء حيث وضعنا متحول الكائن MyEntry متبوعاً بنقطة ثم اسم الإجراء. بالطبع إذا كان الإجراء له وسطاء فيجب تمرير القيم المناسبة لهذه الوسطاء كما في الاستدعاء أعلاه. لاحظ أيضاً أن التعليمة التالية خاطئة:

```
MyEntry.getCustomerStatus;
```

لأن الدالة getCustomerStatus هي دالة خاصة بالكائن و لا نستطيع استدعاؤها من خارج الكائن (أي لا يمكن استدعاء هذه الدالة إلا من خلال كائن يحويها).

يمكن أيضاً تعريف متجهة كائنات بالطريقة نفسها التي نُعرف فيها متجهة سجلات مثلاً، و يمكن أن يتم ذلك من خلال عبارة تعريف الأنواع Type أو مباشرةً باستخدام تعليمة التصريح عن المتحولات Var.

مثال:

تعريف متجهة كائنات من النوع Entry باستخدام العبارة Type.

```
Type
 {Entry تعريف الكائن }
   ObjArray=Array[1..100] of Entry;
   Var Obj: ObjArray;
```

حيث تُعرف التعليمة أعلاه متجهة من 100 كائن من النوع Entry.

يمكن تعريف متجهة كائنات ضمن تعليمة التصريح Var بالشكل:

```
Var Obj:array[1..100] of Entry;
```

في كلتا الحالتين سيتم تعريف متجهة من 100 كائن و لكن من ناحية برمجية فإننا نفضل التعريف الأول لأنه أوضح و لأنه يُمَكّن من تحقيق مبدأ هام في مجال هندسة البرمجيات هو مبدأ إعادة الإستخدامية (reusability).

يمكن التعامل مع الكائنات ضمن المتجهة باستخدام الحلقات. مثلاً تقوم التعليمات البرمجية التالية بطباعة الاسم و ذلك باستدعاء الدالة getName على كل كائن في المتجهة.

```
For i:=1 to 100 do
   Write('Name: ',Obj.getName);
```

من الميزات المهمة للكائنات هي أنها تُوَفّر حماية للحقول المُعرفة فيها و بالتالي تسمح للمبرمج بالحد من الوصول غير المسموح به لهذه الحقول. لكن لعل الميزة الأكثر أهمية للكائنات أنها تسمح بإعادة الاستخدام. حيث يمكن للمبرمج و الذي كتب كائن لحساب مثلاً مساحات بعض الأشكال الهندسية أن يُعيد استخدامه في أي برنامج آخر عند الضرورة من دون الحاجة لإعادة كتابة كامل البرنامج.

9. الوراثة Inheritance

كما قلنا سابقاً فإن الكائنات تمكن المبرمجين من إعادة استخدامهم في برامج أخرى غير البرنامج الذي عُرفت فيه. تملك الكائنات ميزة مهمة أخرى و هي إمكانية التوسيع. بكلمات أبسط يمكن للمبرمج توسيع أي كائن معرف سابقاً و ذلك بإضافة دوال/إجراءات أو حقول أو كليهما. بشكل عام، يمكن تعريف الوراثة على أنها تقنية برمجية تُمَكّن المبرمجين من تعريف كائنات جديدة من خلال إعادة استخدام و توسيع كائنات معرفة مسبقاً. عندما نوسع كائن اسمه مثلاً A لنحصل على كائن جديد اسمه B ، فإن B سوف يرث كل الدوال/الإجراءات و الحقول من الكائن A (ما عدا تلك المعرفة في القسم private) و يمكن له إضافة دوال/إجراءات و حقول جديدة خاصة به. نسمي عادة الكائن A بالأب Ancestor و نسمي الكائن الجديد B المُنشأ من A بالابن Descendent. يمكن للأب الواحد أن يملك أي عدد من الأبناء و لكن في لغة الباسكال لكل ابن هناك أب واحد فقط[3].

لتوضيح مفهوم الوراثة أكثر، نورد المثال التالي.

نعلم أن جميع الأشكال الهندسية تشترك في العديد من الخصائص: لها مساحة، محيط، موقع، .. إلخ. لكن هناك أيضاً صفات مُمَيزة لكل شكل، فمثلاً، الدائرة لها نصف قطر بينما المستطيل له طول و عرض. لذلك يمكن كتابة المميزات المشتركة في كائن عام اسمه Figure و من ثم نُعرف الكائنات الأخرى على أنها أبناء لهذا الكائن و بالتالي ترث كافة خصائصه و يُمكن لها إضافة صفاتها الخاصة بها.

---

[3] يمكن للإبن في بعض اللغات الأخرى أن يرث من أكثر من أب واحد وهذا هو مفهوم الوراثة المتعددة (multyple inheritance) في بعض اللغات الكائنية المنحى (Object Oriented Programming, OOP) مثل لغة C++.

المقاطع البرمجية التالية تعرف كائن عام اسمه Figure و كائنان ابن هما دائرة Circle و مستطيل Rectangle.

```
Type
   Figure=Object
      Function getName:String;
      Function ComputeArea:real;
      Function ComputePerim:real;
   Private
       Name:String;
   End;
   Circle=Object(Figure)
       Function getRadius:real;
   Private
        Radius:real;
   End;
   Rectangle=Object(Figure)
      Function getLength:real;
      Function getWidth:real;
   Private
       Length,width:real;
   End;
```

كما نُلاحظ أن المقطع البرمجي السابق، يُعرف الكائن Figure الذي يحوي مجموعة من الدوال:

- getName لإعادة (إسترجاع) اسم الشكل الهندسي.
- ComputeArea لحساب مساحة الشكل الهندسي.
- ComputePerim لحساب محيط الشكل الهندسي.

و يُعرف كذلك حقل خاص Name لتخزين اسم الشكل الهندسي.

قُمنا باستخدام مفهوم الوراثة بتخصيص الكائن Figure ليُصبح دائرة Circle. كما يظهر فإنه لتعريف كائن خاص من كائن عام فإن ذلك يتم من خلال وضع اسم الكائن العام بين قوسين بعد الكلمة المحجوزة Object. بهذا الشكل يُصبح الكائن دائرة Circle يشتمل على جميع الحقول و الدوال المعرفة فيه بالإضافة إلى الحقول و الدوال التي ورثها من الكائن الأب Figure ما عدا الحقل Name لأنه خاص بالأب (private) و لا يمكن توريثه. بنفس الطريقة تم تخصيص الكائن Figure ليصبح مستطيلاً Rectangle.

كما قلنا سابقاً حتى يكون النوع كائناً له معنى، فيجب كتابة التعليمات البرمجية للدوال و الإجراءات. سواء كان الكائن أباً أو ابناً فإن كتابة التعليمات البرمجية له تتم بالطريقة نفسها. كمثال سوف نكتب التعليمات البرمجية للدالة ComputeArea في الكائن Circle.

يمكن كتابة التعليمات البرمجية لهذه الدالة في أي مكان بعد عبارة تعريف الكائن و قبل تعليمة بداية البرنامج العام Begin.

```
Function Circle.ComputeArea:real;
Begin
    ComputeArea:=3.14*radius;
End;
```

التعامل مع الكائنات في حالة الوراثة

لنتأمل التصريحات التالية و التي تُعرف ثلاثة متحولات من الأنواع Figure، Circle، Rectangle:

```
Var
    myFigure:Figure;
    myCircle:Circle;
    myRecatngle:Rectangle;
```

إذا كتبنا التعليمة البرمجية التالي: myFigure.ComputeArea فأي ComputeArea سوف يتم استدعاء؟ هل المُعرَفة داخل الكائن Figure أم الكائن Circle ، أم الكائن Rectangle؟

الجواب سوف تُستدعى الدالة الخاصة بالكائن Figure ذلك لأن المتحول myFigure هو من النوع Figure.

الآن لو وضعنا الإسناد التالي: myFigure:= myCircle ومن ثم كتبنا التعليمة myFigure.ComputeArea. في هذه الحالة سوف تُستدعى الدالة ComputeArea الخاصة بالكائن Circle. ببساطة يمكن القول أيضاً إذا كان لدينا الإسناد التالي: myFigure:=myRecatngle فإن التعليمة myFigure.ComputeArea تستدعى الدالة ComputeArea الخاصة بالكائن Rectangle.

على الرغم من أن الدالة ComputeArea مشتركة في الكائنات الثلاثة إلا أن المترجم يستدعي الدالة الصحيحة اعتماداً على نوع الكائن الذي تم الاستدعاء عليه. تُسمى هذه الخاصة في البرمجة بخاصة الأشكال المتعددة أو خاصة تعدد الأشكال **Polymorphism**.

ملاحظة:

يمكن إسناد متحول من نوع ابن إلى متحول من نوع أب و لكن العكس غير صحيح. أي أن العبارة: myCircle := myFigure خاطئة.

تمارين

1) ضع برنامجاً بلغة الباسكال يقوم بما يلي:

1. تعريف سجل طالب جامعي يحتوي اسمه و رقمه وتاريخ ميلاده و جنسه (ذكر، أنثى) و درجات أربع مواد قدمها في أحد الامتحانات.
2. تعريف عدة سجلات لا تزيد عن 20 سجلاً.
3. كتابة برنامج جزئي يقوم بقراءة سجل طالب .
4. كتابة برنامج جزئي يقوم بطباعة سجل طالب .
5. إدخال سجلات الطلاب و طباعة قائمة بأسماء الطلاب الناجحين و درجاتهم إذا علمت أن الطالب يُعتَبر ناجحاً إذا نجح في المواد الأربع وعلامة النجاح في كل مادة لا تقل عن 60 درجة.
6. طباعة عدد الطلاب الناجحين الذين حصلوا على أعلى مجموع، و إذا كان هناك طالباً ناجحاً واحداً فطباعة سجله.
7. طباعة سجلات الطلاب الناجحين مُرتّبة أبجدياً بحسب أسمائهم.

2) قدّم طلاب السنة الرابعة إحصاء رياضي امتحان الفصل الأول, فإذا علمت أنّ المواد في السنوات الأربع تمّ ترقيمها من 1 إلى 40 : السنة الأولى من 1 حتى 10، السنة الثانية من 11 حتى 20، السنة الثالثة من 21 حتى 30، و السنة الرابعة من 31 حتى 40، و المطلوب كتابة برنامج بلغة الباسكال يقوم بما يلي:

1. إدخال أسماء الطلاب و أرقام المواد الراسبة لكل طالب، إذا علمت أنّ عدد الطلاب في السنة الرابعة إحصاء هو 100 طالب و طالبة.
2. فرز أسماء الطلاب إلى ثلاث مجموعات (متجهات):

- المجموعة الأولى: تحتوي على أسماء الطلاب و الطالبات المتخرجين.
- المجموعة الثانية: تحتوي على أسماء الطلاب الذين يحق لهم

التقدم إلى الدورة الاستثنائية علماً بأن الطالب الذي يحمل أكثر من أربع مواد لا يحق له التقدم إلى الدورة الاستثنائية.

- المجموعة الثالثة: تحتوي على أسماء الطلاب الراسبين.

3. طباعة كل مجموعة من المجموعات السابقة.
4. طباعة أسماء الطلاب الذين يحق لهم التقدم إلى الدورة الاستثنائية مع أرقام المواد التي حملها الطلاب و السنة التي تُدرَّس فيها كل مادة.

3) ضع برنامجاً بلغة الباسكال يقوم بما يلي :

1. تعريف سجل موظف يحتوي على اسمه و كنيته وتاريخ ميلاده و جنسه وحالته العائلية وراتبه وسنة بدء العمل في الشركة.
2. تعريف مجموعة سجلات الموظفين في الشركة علماً أن عدد الموظفين لا يزيد عن عدد معين (100).
3. ضع برنامجاً جزئياً يقوم بإدخال سجل موظف.
4. ضع برنامجاً جزئياً يقوم بطباعة سجل موظف.
5. طباعة سجلات الموظفين الذين يحصلون على أعلى راتب.
6. طباعة أسماء الموظفين الذكور العزب و حساب عددهم.
7. طباعة سجلات الموظفين الأكثر قدماً و حساب عددهم.
8. طباعة سجلات الموظفات الأصغر سناً.

4) ضع برنامجاً بلغة الباسكال يقوم بحساب البعد بين النقطتين a,b في المستوي (الديكارتي و القطبي) مستخدماً سجلاً متبدلاً للتعريف عن إحداثيي نقطتين في المستوي، علماً أنّ دستور البعد بين نقطتين في المستوي الديكارتي يُعطى بالشكل التالي:

$(a,b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$ و دستور البعد بين نقطتين في الإحداثيات القطبية يُعطى بالشكل التالي:

$$d(a,b) = \sqrt{{r_a}^2 + {r_b}^2 - 2r_a r_b \cos(\theta_b - \theta_a)}$$

5) ضع برنامجاً بلغة الباسكال يقوم بتعريف سجل موظف في شركة، حيث إنَّ سجل الموظّف يحتوي على اسمه و كنيته و تاريخ ميلاده و راتبه و جنسه (ذكر، أنثى)، و إذا كان ذكراً فتحديد طوله و بيان فيما إذا كان ملتحياً أم لا، و إذا كانت أنثى فتحديد وزنها و إذا كانت سورية أم لا. ثم يقوم بمايلي:

1. تعريف سجلات الموظفين في الشركة علماً بأن عددهم لا يزيد عن 20 موظفاً.
2. كتابة برنامج جزئي لإدخال سجل موظف و أخر لطباعته.
3. إدخال سجلات الموظفين و طباعة سجل الموظف الذي يحصل على أعلى راتب إن كان وحيداً، و إلا فطباعة عدد الموظفين الذين يحصلون على أعلى راتب و قيمة هذا الراتب.
4. حساب و طباعة عدد الموظفين الذكور الملتحين.

6) اكتب برنامجاً بلغة الباسكال يقوم بتعرف كائن اسمه Student و يقوم هذا الكائن بما يلي:

1. تعريف الطالب من خلال الواصفات التالية: الاسم Name، الكنية LName، العمر Age، تاريخ الميلاد BirthDate (عبارة عن سجل)، متجهة Marks تحتوي على علامات الطالب أو الطالبة في المواد التي نجح بها، السنة الدراسية SYear، الفرع Major الذي يدرس فيه. يجب أن تكون جميع هذه الواصفات خاصة private.
2. تعريف الدوال التالية:
   a. دالة ReadObj و التي تقوم بإدخال قيم لمتحولات الكائن. هذه الدالة يجب أن تكون عامة.
   b. دالة عامة تقوم بإعادة معدل طالب معين.

c. إجراء عام يقوم بطباعة معلومات طالب معين.

d. دالة عامة تُعيد السنة الدراسية التي يدرس فيها الطالب.

e. إجراء يُعيد علامات الطالب في المواد التي نجح فيها.

f. إجراءات و دوال أخرى تراها مناسبة.

7) اكتب برنامجاً بلغة الباسكال يقوم باستخدام الكائن السابق ثم يقوم بمايلي:

1. تعريف متجهة كائنات من النوع Student.
2. طباعة معلومات طالب بمعرفة اسمه.
3. طباعة معدل طالب بمعرفة اسمه و كنيته.
4. طباعة معلومات كل الطلاب.
5. طباعة السنة الدراسية التي يدرس فيها طالب معين بمعرفة اسمه.

Damascus University

المجموعات The Sets

1. نوع البيانات مجموعة The Set Data Type

1.1 المفهوم العام للمجموعة The General Concept of a Set

تُعرَّف المجموعة بشكل عام على أنها تجميع لعدد من العناصر (قد يكون صفر عنصر) و لكن من نوع واحد، حيث إنَّ ترتيب هذه العناصر غير مهم وتكرارها غير مسموح به. عندما يكون عدد عناصر المجموعة يساوي الصفر عندئذٍ ندعوها بالمجموعة الخالية.

يُمكن تمثيل المجموعة رياضياً بوساطة سرد عناصرها جميعاً وبدون تكرار. لنفترض أن لدينا سلسلة المحارف التالية sweeteners، وأننا نريد تمثيل رموز هذه السلسلة ضمن مجموعة ما، فإن هذه المجموعة سوف تحتوي على الرموز التالية: r , n , t , e , w , s، مع ملاحظة أن الحرفين s , e قد وُضِعا ضمن المجموعة مرة واحدة، في حين أنهما ظهرا ضمن السلسلة أكثر من مرة.

2.1 المجموعة في لغة الباسكال

يمكن تمثيل المفهوم الرياضي للمجموعة بلغة الباسكال، حيث إنّه يمكننا تعريف مجموعة، مجموعة جزئية، مجموعة خالية، و يمكن أيضاً تعريف بعض العمليات المعروفة على المجموعات مثل الاجتماع، التقاطع، .. إلخ.

يمكن التصريح عن المجموعة بلغة الباسكال وفق التركيب النحوي التالي:

```
Type
   TypeName=set of dataType;
```

حيث تُستَخدَم الكلمات المحجوزة set of من أجل تعريف نوع بيانات مجموعة يُدعى TypeName و جميع عناصرها من النوع dataType، و بعد ذلك يمكن

تعريف متحول s من النوع المجموعاتي المُصَرَّح عنه TypeName وفق الصيغة التالية Var s : TypeName ;.

هنا يجب التنويه إلى أن النوع dataType هو نوع مُحدود يجب ألاَّ يتجاوز الـ 256 عنصراً و مجال هذه العناصر بين القيمتين 0 و 255 أي أنه يمكن أن يكون فقط Byte أو char هذا يعني أنه لايمكن تمثيل مجموعة من الأعداد الحقيقية أو مجموعة من السلاسل المحرفية أو مجموعة الأعداد الصحيحة بشكل كامل (يُمكن فقط تمثيل جزء منها).

مثال:

يمكن تعريف مجموعة الأحرف الانكليزية الصغيرة كمايلي:

```
Type MySet=set of 'a'..'z';
Var s:MySet;
```

```
var letters:set of 'a'..'z';
```

ويمكن التعريف مباشرةً بالعبارة

وظيفة نوع البيانات مجموعة Set في لغة الباسكال هي تمثيل هذه المجموعات. والبرنامج التالي يُبين لنا كيفية تعريف وتهيئة المجموعات.

```
Type
   Major=(anthropology,art,astronomy,
     biochemistry,Biology,chemistry,complit,
     earthsci,economics,english,literature,
     math,physics,politics,psychology);
     MajorGroup=set of Major;
Var
    AllMajors,Science:MajorGroup;
```

فعلى سبيل المثال التصريح السابق ضمن عبارة var يُنشئ متحولين من النوع MajorGroup. هذين المتحولين يُخزنان مجموعة من 15 عنصراً، كل منها من النوع Major.

لتحديد عناصر المجموعة، علينا ببساطة سرد عناصر هذه المجموعة ضمن قوسين متوسطين [ ] وتفصل بين هذه العناصر الفاصلة العادية. في حال كانت عناصر المجموعة (أو جزء منها) مجالية يمكن سرد هذه العناصر بوضع أصغر عنصر متبوعاً بنقطتين متتاليتين (..) ثم أكبر هذه العناصر.

لنتذكر أن المجموعة تعريفياً غير مرتبة أي أن ترتيب العناصر لا يهم وبالتالي عبارات الإسناد التالية تعطينا نفس المجموعة science ولكن وفق ترتيب مختلف لعناصرها.

```
science:=[Physics,Astronomy..Chemistry,earthsci]
science:=[Astronomy..Chemistry,earthsci,Physics]
```

وعليه عبارات الإسناد السابقة جميعها تضع العناصر التالية astronomy و biochemistry و biology و chemistry و earth science و physics ضمن المجموعة Science.

2. تمثيل المجموعات Representation of Sets

يُمكننا تمثيل المجموعات في لغة الباسكال بمصفوفة من الخانات الثنائية (bits)، تقابل كل خانة موضع قيمة في المجموعة. فإذا كانت قيمة هذه الخانة هي (1) فإن العنصر يكون موجوداً في المجموعة، أما إذا كانت قيمة هذه الخانة هي (0) فإن هذا العنصر المقابل لا يكون في هذه المجموعة.

تمثيل المجموعات على شكل قيم بوليانية يُفسر لنا عدم الاهتمام بترتيب عناصر المجموعة، ويفسر أيضاً لماذا كان تكرار العنصر أمراً لا يؤثر في المجموعة.

وسنزيد الموضوع شرحاً: لنفترض أن لدينا البرنامج التالي الذي يُعرف النوع ThreeDigit على أنه مجموعة من النوع المجالي FiveDigit كما يلي:

```
Program RepresintationSetDemo;
Type
     FiveDigit=1 .. 5;
     ThreeDigit= set of FiveDigit;
Var
     Three:ThreeDigit;
Begin
    Three:=[1, 3, 5];
End.
```

نلاحظ من خلال البرنامج السابق أننا أعطينا المجموعة Three ثلاث قيم هي (1) و (3) و (5). عندئذٍ يمكن تمثيل هذه المجموعة بالمتجهة التالية (الموجودة في الأسفل)، مع العلم بأن الحجرة التي فيها (1) تدل على أن دليل هذه الحجرة ينتمي إلى المجموعة Three، أما الحجرة التي فيها (0) فتدل على أن دليلها لا ينتمي إلى المجموعة Three.

Three

1	1
2	0
3	1
4	0
5	1

3. مجموعة خاصة A Special Set

نُذَّكر بأنه توجد مجموعة خاصة جداً وهي المجموعة التي لا تملك أي عنصر و تُدعى بالمجموعة الفارغة (empty set) أو بالمجموعة الخالية (null set). هذه المجموعة تُشبه الرقم (0) في الرياضيات. وتُحَدَد المجموعة الخالية بوساطة قوسين مربعين متوسطين فارغين أي [ ].

4. العمليات على المجموعات Set Operators

تدعم لغة الباسكال معظم العمليات الرياضية الثنائية المُعرَفة على المجموعات. هذه العمليات هي الاجتماع، التقاطع و الفرق. كما أنها تدعم عملية الانتماء: أي أنه يمكننا اختبار فيما إذا كانت قيمة معينة تنتمي إلى مجموعة ما أم لا. سنناقش هذه العمليات في الفقرات التالية.

1.4 معامل الانتماء In Operator

قد نحتاج في بعض الأحيان إلى اختبار وجود عنصر ما ضمن المجموعة. وفي الحقيقة تُستَخدم عملية الاختبار هذه في إظهار عناصر المجموعة أو أغراض أخرى كما سنرى لاحقاً.

تزودنا لغة باسكال بالمعامل in الذي يقوم باختبار انتماء قيمة ما إلى المجموعة. هذا المعامل الثنائي (binary operator) يأخذ حدين مختلفين، الأول هو عنصر من نوع البيانات الأساسي للمجموعة، والحد الثاني هو المجموعة نفسها. ويعيد هذا المعامل قيمة بوليانية هي True إذا كان العنصر ينتمي إلى المجموعة و False إذا لم ينتمِ العنصر إلى المجموعة.

والبرنامج التالي يوضح لنا المعامل In.

```
Type
     CharSet=set of char;
Var
     MySet:CharSet; x:char;
```

```
Begin
  MySet:=['A' .. 'z'] ;
  x:='D';
  If(x in MySet) then
     Write('yes')
  Else
     Write('No');
End.
```

في المقطع البرمجي السابق، عرفنا النوع مجموعة CharSet عناصرها من النوع المحرفي Char ثم صرَّحنا عن متحولين MySet من النوع CharSet و x من النوع Char ثم أسندنا مجموعة الحروف الكبيرة 'A'..'Z' إلى المجموعة MySet و الحرف 'D' إلى المتحول x.

اختبار الانتماء ضمن العبارة الشرطية سوف يتحقق لأن القيمة المُخَزنة في المتحول x هي 'D' و العنصر 'D' ينتمي إلى المجموعة MySet و بالتالي ستُطبع عبارة 'yes'. أما لو أسندنا للمتحول x القيمة 'd' فإن الشرط سيكون غير محقق و سيتم طباعة العبارة 'No'.

من الجدير بالذكر أنه لا يمكن طباعة المجموعة باستخدام التعليمة write بشكل مباشر. لطباعة المجموعة يجب المرور على جميع عناصرها و طباعتهم واحداً تلو الأخر. يُمكن استخدام معامل الانتماء In لاختبار انتماء عنصر إلى المجموعة من أجل طباعته.

مثال:

```
Type
     ByteSet = set of byte;
Var
     MySet:ByteSet; Index:byte;
Begin
```

```
  MySet:=[1,7, 4, 5, 11, 21, 37, 177];
  Writeln('My Set');
  For Index:=0 to 255 do
    If(Index in MySet) then
       write(Index);
End.
```

يقوم المقطع البرمجي السابق بطباعة عناصر المجموعة Myset. نلاحظ مثلاً أن العناصر 0,2,3,.. لن يتم طباعتها لأنها لا تنتمي للمجموعة MySet، بينما سيتم طباعة جميع الأعداد 1,7, 4, 5, 11, 21, 37, 177 لكونها تنتمي للمجموعة MySet.

2.4 معامل الإجتماع Union Operator

يزودنا معامل اجتماع المجموعات بإمكانية إضافة مجموعة إلى أخرى، حيث إنَّ المجموعة الناتجة عن اجتماع مجموعتين تحتوي على جميع العناصر الموجودة في المجموعتين و بدون تكرار. يُستخدم المعامل + في لغة الباسكال لإجراء عملية الاجتماع. المعامل + هو معامل ثنائي (binary operator) يأخذ متحولين من نوع البيانات الأساسي مجموعة و يقوم بتوليد المجموعة الناتجة عن الاجتماع.

مثال:

```
Type
     CharSet=set of Char;
Var
     WdSet1,WdSet2,WdSet3:CharSet;
     IndexCh:Char;
Begin
  WdSet1:=['b', 'd', 'f', 'h', 'k', '1', 't'];
  WdSet2:=['g', 'j', 'p', 'q', 'y'];
  WdSet3:=WdSet1+WdSet2;
  Writeln('WdSet13');
```

```
  For IndexCh:='a' to 'z' do
    If(IndexCh in WdSet3) then
       write(IndexCh);
End.
```

كما في مفهومها الرياضي، تُمَثِل المجموعة الخالية (null set) في لغة الباسكال العنصر الحيادي بالنسبة لمعامل الاجتماع و بالتالي ناتج اجتماع المجموعة الخالية مع أي مجموعة أخرى هو المجموعة نفسها.

3.4 معامل التقاطع Intersection Operator

يزودنا معامل التقاطع بإمكانية إيجاد العناصر المشتركة بين مجموعتين، حيث إنَّ المجموعة الناتجة عن تقاطع مجموعتين تحتوي على جميع العناصر المشتركة بين المجموعتين و بدون تكرار. يُستخدَم المعامل * في لغة الباسكال لإجراء عملية التقاطع. المعامل * هو معامل ثنائي (binary operator) يأخذ متحولين من نوع البيانات الأساسي مجموعة و يقوم بتوليد المجموعة الناتجة عن تقاطع هاتين المجموعتين.

مثال:

```
Type
     CharSet=set of Char;
Var
     WdSet1,WdSet2,WdSet3:CharSet;
     IndexCh:Char;
Begin
  WdSet1:=['b', 'd', 'f', 'h', 'k', '1', 't'];
  WdSet2:=['g', 'd', 'p', 'h', 'y'];
  WdSet3:=WdSet1*WdSet2;
  Writeln('WdSet13');
  For IndexCh:='a' to 'z' do
     If(IndexCh in WdSet3)  then
```

```
        write(IndexCh);
End.
```

يقوم المقطع البرمجي السابق بطباعة العنصرين 'd', 'h' لأنهما عنصران مشتركان بين المجموعتين WdSetl و WdSet2، بينما لن تتم طباعة أياً من العناصر الباقية كونها ليست مشتركة بين المجموعتين WdSetl و WdSet2، و بالتالي لن ينتموا إلى مجموعة التقاطع.

4.4 معامل فرق المجموعات The set Difference Operator

يزودنا معامل الفرق بإمكانية إيجاد العناصر التي تنتمي إلى المجموعة الأولى و لا تنتمي إلى المجموعة الثانية. يُستخدَم المعامل – في لغة الباسكال لإجراء عملية الفرق. المعامل – هو معامل ثنائي (binary operator) يأخذ متحولين من نوع البيانات الأساسي مجموعة و يقوم بتوليد المجموعة الناتجة عن الفرق.

مثال:

```
Type
     CharSet=set of Char;
Var
     WdSetl,WdSet2,WdSet3:CharSet;
     IndexCh:Char;
Begin
  WdSetl:=['b', 'd', 'f', 'h', 'k', '1', 't'];
  WdSet2:=['g', 'd', 'p', 'h', 'y'];
  WdSet3:=WdSetl - WdSet2;
  Writeln('WdSetl3');
  For IndexCh:='a' to 'z' do
    If(IndexCh in WdSet3) then
      write(IndexCh);
End.
```

يقوم المقطع البرمجي السابق بطباعة العناصر 't', '1', 'k', 'f', 'b' لأنها عناصر موجودة في المجموعة WdSetl و غير موجودة في المجموعة WdSet2، بينما لن تتم طباعة أي من العنصرين 'h' , 'd' كونها مشتركة بين المجموعتين WdSetl و WdSet2 ، و بالتالي لن ينتموا إلى مجموعة الفرق.

5.4 إضافة و حذف عنصر من مجموعة Add/Remove an Element from a Set

بالإضافة للمعاملات السابقة، تزودنا لغة باسكال بمعاملين جديدين هما معامل إضافة عنصر إلى مجموعة Include و معامل حذف عنصر من مجموعة Exclude ويأخذ كل معامل من هذين المعاملين وسيطين. الوسيط الأول عبارة عن مجموعة يمكن أن يكون نوع بياناتها الأساسي أيّ نوع من أنواع البيانات القياسية والوسيط الثاني متحول ينتمي لنفس نوع البيانات الأساسي الخاص بهذه المجموعة. فالمعامل Include يُضيف عنصراً واحداً هو القيمة الممررة إلى الوسيط الثاني. أما المعامل Exclude فيحذف عنصراً من المجموعة، وهذا العنصر هو القيمة الممررة إلى الوسيط الثاني.

مثال:

```
Type
     CharSet=set of Char;
Var
     WdSet:CharSet; IndexCh:Char;
Begin
   WdSet:=['b', 'd', 'f', 'h', 'k', '1', 't'];
   Include(WdSet, 'z');
   Writeln('WdSet after Including');
   For IndexCh:='a' to 'z' do
     If(IndexCh in WdSet)  then
       write(IndexCh);
```

```
  Exclude(WdSet, 'f' );
  Writeln('WdSet after Excluding');
  For IndexCh:='a' to 'z' do
   If(IndexCh in WdSet) then
     write(IndexCh);
End.
```

يقوم المقطع البرمجي السابق بإضافة العنصر 'z' إلى المجموعة Wdset ثم يقوم بطباعة عناصر المجموعة المُعَدَّلة أي يقوم بطباعة العناصر 'b', 'd', 'f', 'h', 't', '1', 'z', 'k'. بعد ذلك يقوم المقطع بحذف العنصر 'f' من المجموعة الجديدة ثم يقوم بطباعة عناصر المجموعة الناتجة و هي 'b', 'd', 'h', 'k', '1', 't', 'z' .

5. مقارنة المجموعات Comparing Sets

تُمَّكن لغة باسكال من استخدام أربعة معاملات لمقارنة المجموعات، وعملية المقارنة بين المجموعات تعتمد على العناصر الموجودة في هذه المجموعات، بالإضافة إلى عدد هذه العناصر. أي يمكن مقارنة مجموعتين لتحديد فيما إذا كانتا تملكان نفس العناصر، حيث يُطلَق على هاتين المجموعتين بالمجموعتين المتساويتين (**equal sets**). معاملات المقارنة تُعيد قيماً بولیانية. وسطاء هذه المعاملات هي مجموعات لذلك نقول: إن معاملات المقارنة تأخذ وسيطين عبارة عن مجموعتين وتُعيد قيمة بولیانية.

1.5 معاملا تساوي وعدم تساوي المجموعات Inequality Operators / The Set Equality

تملك لغة الباسكال معامل التساوي (= Equality Operator) الذي يُمَكن من اختبار تساوي مجموعتين، حيث يُعيد القيمة True في حال تساوي هاتين المجموعتين؛ أي احتواء المجموعتين على نفس العناصر، وإلا فإنه يُعيد القيمة False.

مثال:

ليكن لدينا المجموعتين:

```
Set1:=['a','c'. .'m','p'];
Set2:=['a'..'m','p'..'s'];
```

معامل المساواة في عبارة If التالية سوف يُعيد القيمة FALSE وبالتالي سيتم طباعة العبارة 'not Equal' باعتبار أن المجموعتين Set1 و Set2 غير متساويتين.

```
IF (Set1 = Set2) Then write('Equal') else write ('not Equal');
```

لاحظ أن الحرف 'r' مثلاً هو عنصر من المجموعة Set1 ولكنه ليس عنصراً من المجموعة Set2.

كما يُمكن في لغة باسكال استخدام المعامل "<>" بهدف اختبار عدم تساوي مجموعتين؛ حيث يُعيد هذا المعامل القيمة البوليانية True عند عدم التساوي و القيمة البوليانية False خلاف ذلك.

2.5 معامل المجموعة الجزئية The Subset Operator

تملك لغة باسكال المعامل (<=) الذي يمكن من اختبار فيما إذا كانت مجموعة ما هي مجموعة جزئية (Subest) من مجموعة أخرى، حيث يعيد القيمة True في حال كانت المجموعة الأولى محتواة في المجموعة الثانية (أو تساويها)؛ أي إذا حوت المجموعة الثانية على جميع عناصر المجموعة الأولى على الأقل، وإلا فإنه يعيد القيمة False.

مثال:

ليكن لدينا المجموعتان:

```
Set1:=['a' , 'c' . . 'm' , 'p'];
Set2:=['a'..'m','p'];
```

المعامل <= في عبارة If التالية سوف يُعيد القيمة True وبالتالي سيتم طباعة العبارة Contained باعتبار أن المجموعة Set1 هي مجموعة جزئية من المجموعة Set2.

```
If(Set1 <= Set2) Then
     write('Contained')
Else
     write('not Contained');
```

يُمكن التعبير عن الاحتواء التام بوساطة المعامل "<" الذي يُمَكن من اختبار فيما إذا كانت مجموعة ما هي مجموعة جزئية تماماً من مجموعة أخرى، حيث يُعيد القيمة True في حال كانت المجموعة الأولى محتواة تماماً في المجموعة الثانية، وإلا فإنه يعيد القيمة False.

عندما نقول إنّ المجموعة Set1 هي مجموعة جزئية (Subest) من المجموعة Set2 فإن هذا يكافئ القول بأن المجموعة Set2 هي مجموعة حاوية (superset) للمجموعة Set1. في لغة باسكال معامل المجموعة الحاوية يُمَكن من اختبار هذه العلاقة بين المجموعات. العبارة التالية مكافئة تماماً لسابقتها التي تستخدم معامل المجموعة الجزئية.

```
If(Set2>=Set1) Then
   write('Set2 Contains Set1')
Else
   write('Set1 is not Contained in Set2');
```

تجدر الإشارة إلى أنه يمكن استخدام المعامل ">" لاختبار فيما إذا كانت مجموعة ما حاوية تماماً لمجموعة أخرى.

يبين الجدول التالي المعاملات المطبقة على نوع البيانات مجموعة

المعامل (Operator)	الوسيط الأيسر (Left	الوسيط الأيمن (Right	الناتج (Result)

	Operand)	Operand)	
BOOLEAN	مجموعة	قيمة	الانتماء (IN)
مجموعة	مجموعة	مجموعة	الاجتماع (+)
مجموعة	مجموعة	مجموعة	التقاطع (*)
مجموعة	مجموعة	مجموعة	الفرق (-)
BOOLEAN	مجموعة	مجموعة	التساوي (=)
BOOLEAN	مجموعة	مجموعة	عدم التساوي (<>)
BOOLEAN	مجموعة	مجموعة	المجموعة الجزئية (<=, <)
BOOLEAN	مجموعة	مجموعة	مجموعة حاوية (>=, >)

6. المجموعات الثابتة والمجموعات عديمة الأسماء Constant and Anonymous Sets

المجموعة الثابتة هي المجموعة التي عناصرها ثابتة وبالتالي لا يمكن التعديل عليها بعد مرحلة التصريح عنها. يمكن تعريف المجموعة الثابتة في لغة باسكال وفق ما يلي:

```
Type
    CharSet=set of Char;
Const
SmallLetters:CharSet=['a','c','m'..'o','x','z'];
DigitSet=set of 0..9;
```

تقوم مجموعة التعليمات السابقة بالتصريح عن المجموعتين الثابتتين SmallLetters و DigitSet. المجموعة SmallLetters تشمل بعض الأحرف الانكليزية الصغيرة و المجموعة DigitSet تشمل الأرقام من 0 إلى 9. الجدير بالذكر أن هاتين المجموعتين لايُمكن تغييرهما في أي مكان في البرنامج.

أما المجموعة عديمة الاسم فهي المجموعة التي تُستخدم ضمن تعليمات البرنامج و لكن بدون إعطائها اسماً مُحَدداً. هذا يعني أننا نكتفي فقط بوضع العناصر ضمن القوسين المربعين. مثلاً: التعليمات التالية تختبر إذا كان محرفاً ما CharValue هو حرف أبجدي أم لا.

```
If(CharValue in ['a'..'Z']) then
    Writeln('yes, alphbatical');
Else
    Writeln('no, not alphabetical');
```

أمثلة محلولة.

مثال 1:

ضع برنامجاً بلغة الباسكال يقوم بتوليد عشرة أعداد عشوائية مختلفة صحيحة من المجموعة التي تحتوي على الأعداد من 1 إلى 100.

الحل:

```
Program RandomNumber;
Uses crt;
Type
   SetInt = set of 1 .. 100;
Var
     A: setInt;
     i, j: 1 .. 100;
  Begin
     clrscr;
     Randomize;
     A:= [ ];
     i:= 1;
     while (i <= 10) do
       Begin
```

```
            j := trunk(100 * Random) +1;
            if(not (j in A)) then
              Begin
                 A:= A + [j];
                 i:= i + 1;
              End;
        End;
      For i:=1 o 100 do
            if (i in A) then
                 write(i, ' ');
      Readln;
End.
```

مثال 2:

ضع برنامجاً بلغة الباسكال يقوم بإدخال شريط من المحارف لا يتجاوز 256 محرفاً و فرز عناصره إلى خمس مجموعات.

- المجموعة الأولى تحتوي على الأحرف الإنكليزية الصغيرة.
- المجموعة الثانية تحتوي على الأحرف الإنكليزية الكبيرة.
- المجموعة الثالثة تحتوي على الإشارا ت الحسابية [+, -, *, /].
- المجموعة الرابعة تحتوي على الأقواس بأشكالها المختلفة.
- المجموعة الخامسة تحتوي على بقية المحارف.

ثم طباعة العناصر المختلفة لكل مجموعة من المجموعات السابقة.

الحل:

```
Program EngChars;
Uses crt;
Const
     set1=['a'..'z'];
```

```
     Set2= ['A' .. 'Z'];
     Set3= ['+', '-', '*', '/'];
     Set4 = ['(', ')', '[', ']', '{', '}'];
Var
     A,B,C,D,E:Set of char;
     Txt:String;
     L:Integer;
Begin
     Clrscr;
     Writeln('input a text ');
     Readln(txt);
     L:=length(txt);
     A:=[]; B:=[]; C:= []; D:=[]; E:= [];
     For i:=1 to L do
          If(txt[i] in Set1) then
               A:= A + [txt[i]]
          Else if(txt[i] in Set2) then
               B:=B + [txt[i]];
          Else if(txt[i] in Set3) then
               C:=C+ [txt[i]];
          Else if(txt[i] in Set4) then
               D:=D + [txt[i]];
          Else
               E:=E +[txt[i]];
               Writeln('Set A');
          For ch:= 'a' to 'z' do
          If(ch in A) then
               Write(ch, ' ');
          Writeln('Set B');
          For ch:= 'A' to 'Z' do
```

```
        If(ch in B) then
           Write(ch, ' ');
           Writeln('Set C');
        For i := 0  to 256 do
        If(char(i) in C) then
           Write(char(i), ' ');
        Writeln('Set D');
        For i := 0  to 256 do
        If(char(i) in D) then
           Write(char(i), ' ');
        Writeln('Set E');
        For i := 0  to 256 do
           If(char(i) in E) then
             Write(char(i), ' ');
 Readln;
End.
```

Damascus University

تمارين

1) بين أي من المجموعات التالية معرفة بشكل صحيح

a) [1, 2, 3, 1..5]
b) ['1', '3', '1'..'5']
c) [1, 3, '1'..'5']
d) ['1', '3', 'A'..'C']
e) [1, 10, 500..50]

2) أي من المجموعات التالية يمكن تعريفها بلغة الباسكال، و إذا كانت المجموعة قابلة للتعريف؟ قم بتعريفها.

1. مجموعة الأعداد الصحيحة.
2. مجموعة الأعداد الطبيعية.
3. مجموعة الحروف الصغيرة.
4. مجموعة الحروف الكبيرة.
5. الأعداد الحقيقية في المجال [0, 1].
6. الأعداد الصحيحة من 1900 إلى 2000.

3) لتكن لدينا المجموعتان التاليتان:

small := [‘a’, ‘c’, ‘d’, ‘I'..'p’], CAPS :=[‘A’..’K’, ‘M’]

و المطلوب:

1. اكتب جميع عناصر المجموعات X, Y, Z الناتجة عن تنفيذ التعليمات الثلاث التالية:

a) X :=small + CAPS;

b) Y:= small * CAPS;

c) Z:=Small – CAPS;

2. اكتب مجموعة تعليمات (لاتكتب برنامجاً كاملاً) بلغة الباسكال لطباعة عناصر المجموعة X.

4) بين ما هي نتائج العمليات التالية:

1. [1, 3] <= [1, 2, 3, 4]
2. [1,3] <=[1,3]
3. [1,2,3,4]<=[1,3]
4. [1,3]<=[ ]
5. [ ] <=[1,3]
6. [1,6] >=[1]
7. [1,2,3,4,5]>=[1..4]
8. [1,2,3] >[1,2,3]
9. [1,2,3]<[1,2,3]
10. [1,2,3] – [1,2]
11. ([1,2,3,4,5]-[5, 7,8])*[1,3]
12. ([1,2]+[3,6]-[3])*[7, 12]

5) عرّف مجموعة تحتوي على علامات الترقيم ثم استخدم هذه المجموعة لمعرفة فيما إذا كان محرف مدخل ما هو علامة ترقيم أم لا.

6) اكتب برنامجاً بلغة باسكال يقوم بقراءة شريط من المحارف (String) ثم يقوم بطباعة ما يلي (إن وجدت): الأحرف الكبيرة، الأحرف الصغيرة، علامات الترقيم.

7) ضع برنامجاً بلغة الباسكال يقوم بإدخال متجهتين X,Y من الأعداد الصحيحة المختصرة (**Byte**)، ثم يقوم بمايلي:

1. إيجاد العناصر المختلفة في المتجهة X و كذلك بالنسبة لـ Y و طباعة كل منهما.
2. طباعة العناصر المختلفة في **X,Y**، ثم العناصر المشتركة بين **X** و **Y**.
3. بيان فيما إذا كانت العناصر المختلفة الموجودة في **X** تطابق العناصر المختلفة الموجودة في **Y**.
4. إسناد العناصر المختلفة (ثم المشتركة) بين **X** و **Y** إلى متجهة **Z** ثم طباعتها.

الفصل الرابع

الملفات The Files

1. الملفات Files

بشكل عام يُمكن تعريف الملف على أنه مجموعة من المعلومات المخزنة تحت اسم معين على وسط تخزين. فالمقالة المكتوبة في محرر النصوص word تُعتَبر ملفاً إذا ما تم تخزينها، و جدول أسماء الطلاب و علاماتهم إذا تم تخزينه تُعتَبر ملفاً أيضاً، ولذلك للملفات مفهوم واسع في البرمجيات. والسؤال كيف يتم إنشاء ملف وتخزينه على وسط تخزين (القرص الصلب مثلاً)؟ في الواقع الجواب على هذا السؤال يختلف من برنامج إلى أخر. فمثلاً في البرامج التطبيقية مثل word يتم تخزين النص بعد كتابته مباشرةً فيصبح هذا النص ملفاً. أما في لغات البرمجة فيتم ذلك من خلال مجموعة من التعليمات البرمجية و التي هي موضوع بحثنا في هذا الفصل.

لقد استخدمنا في برامجنا حتى الآن وحدة الدخل القياسية، وهي لوحة المفاتيح (Keyboard)، ووحدة الخرج القياسية وهي الشاشة (screen). البرامج الحاسوبية تحصل على المعلومات من قراءة مايكتبه المستخدم عند محث (prompt) نظام التشغيل، ومن ثم يُظهر البرنامج خرجه على الشاشة و لكن إذا أردنا تنفيذ هذا البرنامج مرة أخرى فإننا نحتاج إلى تكرار إدخال نفس البيانات مرة أخرى من لوحة المفاتيح و هكذا كل مرة.

في الواقع الملفات تسمح لنا ليس فقط بإدخال البيانات في الملف و إنما تخزينها أيضاً الأمر الذي يُمكننا من إعادة استخدامها مستقبلاً متى دعت الحاجة إلى ذلك.

تملك لغة الباسكال نوع بيانات قياسياً يُسمى ملف (file) وتوفر مجموعة من التعليمات التي تُمكننا من انجاز مجموعة من العمليات على الملفات من إنشاء،

فتح، تعديل،.. إلخ. يتم النظر إلى الملف بلغة الباسكال على أنه مجموعة من العناصر المتتالية من نوع قياسي واحد و غير محدودة العدد سلفاً (ولذلك تُعد الملفات من بُنى المعطيات الديناميكية). بشكل عام يمكن تقسيم الملفات في لغة الباسكال إلى نوعيين رئيسيين هما ملفات ذات نوع و ملفات بلا نوع. تُقسم الملفات ذات نوع بدورها إلى نوعين:

1. الملفات المباشرة: وتُسمى أيضاً الملفات الثنائية و تمتاز بإمكانية الوصول إلى أي عنصر عن طريق ترتيبه في الملف. و منها الملفات العددية وملفات السجلات.
2. الملفات النصية: و تُسمى أيضاً الملفات التتابعية حيث لا يمكن الوصول إلى أي عنصر فيها إلا بعد المرور بجميع العناصر التي تسبقه في الملف.

2. الملفات المباشرة و العمليات عليها

1.2 تعريف الملف برمجياً

كما مر معنا سابقاً فإنه يُمكن تعريف نوع البيانات ملف (كغيره من المتحولات المركبة) بطريقتين: إما مباشرةً عن طريق استخدام الكلمة المحجوزة var أو باستخدام الكلمة المحجوزة type وذلك لتعريف نوع ملفي ومن ثم التصريح عن متحول من هذا النوع باستخدام الكلمة var وفق مايلي:

```
Type
    T=file of T0 ;
Var f: T;
```

في المقطع البرمجي السابق تم التصريح عن النوع الملفي T والذي جميع عناصره من النوع القياسي المحدد T_0 وذلك باستخدام الكلمة المحجوزة file و من ثم تم تعريف المتحول الملفي f باستخدام الكلمة المحجوزة var. يمكن الوصول إلى الغاية نفسها و ذلك بكتابة العبارة var f: file of T_0.

2.2 ربط المتحول الملفي باسم الملف الخارجي

قبل التعامل مع الملف يجب ربط المتحول الملفي باسم الملف الخارجي المُخزَن على وسط التخزين و يتم ذلك وفق التعليمة التالية:

```
assign(f, 'مسار الملف');
```

يجب وضع هذه التعليمة في بداية البرنامج لتُمَكن من ربط المتحول الملفي f بالملف الموجود على وسط التخزين وذلك بوضع مسار هذا الأخير باستخدام التعليمة assign.

إذا كان الوسيط الثاني للتعليمة assign هو سلسلة فارغة فإن هذا الإجراء سوف يربط متحول الملف مع وحدة الإدخال والإخراج الأساسية (consol)، لذلك سيتم ربط الملف FConsol مع الشاشة عبر الاستدعاء التالي:

```
assign (FConsol, '');
```

3.2 تهيئة الملف للكتابة

قبل التعامل مع الملف و بعد ربط المتحول الملفي بالبيئة الخارجية (أي بالملف الموجود على وسط التخزين) فإنه يجب تهيئته للكتابة عليه و يتم ذلك باستخدام التعليمة rewrite وفق مايلي: rewrite(f).

4.2 فتح ملف موجود

يمكننا استدعاء الإجراء reset عوضاً عن rewrite إذا كان الملف المطلوب موجوداً. فالإجراء reset يفتح الملف المحدد ضمن متحوله الوسيطي الوحيد، مفترضاً أن هذا الملف موجود وسوف يتم تهيئته و يُفتح للقراءة و توضع المؤشرة في بدايته و يتم كل ذلك بكتابة العبارة reset(f).

يُقرأ الملف عنصراً عنصراً، فبعد أن يُقرأ عنصر ما، يفترض البرنامج أنه يستطيع قراءة العنصر التالي، وهكذا ما دام هناك عناصر متبقية لم تُقرأ من الملف. ويتم التعامل مع الملفات وفق الصيغة الفيزيائية المخزنة بها على القرص والتي تمكن البرنامج من جلب عناصر الملف. يرتبط اسم الملف بمحتوياته، أي

أن اسم الملف يشير إلى نوع هذا الملف. سوف نبين كيف نقرأ و نكتب المعلومات في الملفات.

5.2 قراءة عنصر من ملف

تُستخدم التعليمة read من أجل قراءة عنصر من ملف، وذلك عبر تمرير متحول الملف كمتحول وسيطي أول للتعليمة read، ومن ثم تمرير العنصر المراد قراءته بعد ذلك ضمن المتغير الوسيطي الثاني. فمثلاً التعليمة read(f,x) تُمكن من قراءة العنصر x من الملف f.

6.2 كتابة عنصر في ملف

تُستخدم التعليمة write من أجل كتابة عنصر في ملف، وذلك عبر تمرير متحول الملف كمتحول وسيطي أول للتعليمة write، ومن ثم تمرير العنصر المراد كتابته (في الملف) بعد ذلك ضمن المتغير الوسيطي الثاني. فمثلاً التعليمة write(f,x) تمكن من كتابة العنصر x في الملف f.

7.2 إغلاق الملف

في الحقيقة القيم الموجودة في ملف يمكن أن تضيع بسبب عدم تخزين محتويات الملف. فمن أجل تخزين محتويات الملف قبل إنهاء البرنامج، علينا استخدام الإجراء close. التعليمة close(f) تُمَكن من إغلاق الملف f وتخزين محتوياته.

بعد التصريح عن متحول من النوع ملف علينا القيام بعدة مراحل وهي:

1. كتابة عبارة Assign لتربط متحول الملف f باسم الملف FileName، فمتحول الملف يُمرَر على أنه متحول وسيطي أول، واسم الملف يُمرَر كمتحول وسيطي ثانٍ.
2. إنشاء وفتح ملف جديد بوساطة الإجراء rewrite، والذي يأخذ متحولاً وسيطياً واحداً هو متحول الملف. فاستدعاؤنا للإجراء rewrite هذا سوف

يفتح الملف الذي تم ربط اسمه بمتحول الملف f الموجود ضمن البرنامج. إن الإجراء rewrite يُنشئ دائماً ملفاً جديداً، فإذا كان هناك ملف قديم يحمل نفس اسم الملف الجديد، فإن المعلومات الموجودة في الملف القديم سوف تضيع، ويُستبدل الملف القديم بالملف الجديد الفارغ.

3. بعد إنشاء الملف، تُمَكن التعليمة reset من فتح ذلك الملف. فالتعليمة reset تُمَكن إذاً من فتح ملف موجود مسبقاً.
4. قراءة (كتابة) المعلومات من (في) الملف باستخدام إجراءات مسبقة التعريف read (write) بحيث يكون المتغير الوسيطي الأول متحول الملف المطلوب القراءة منه (الكتابة عليه)، أما المتغير الوسيطي الأخر فيُحَدد المعلومات التي ستُقرأ من (ستُكتَب على) الملف.
5. بعد أن كتبنا ما نريد من معلومات على الملف، علينا الآن إغلاق الملف بوساطة استدعاء الإجراء close، حيث يتم تفريغ الذاكرة الوسيطة المرتبطة بالملف، بوساطة كتابة محتوياتها على الملف الموجود على القرص، أي تخزين هذه المعلومات، ومن ثم إغلاق هذا الملف.

الملاحظات الهامة

- لا يمكن تعريف ملف بدلالة ملف.
- لا يجوز وضع تعليمتي rewrite و reset بشكل متتالي و ذلك من أجل الملف نفسه.
- لا يجوز إسناد متحول ملفي إلى متحول ملفي أخر.
- يُستَخدَم المُحدد المنطقي eof (End Of File) لاختبار وصول المؤشر إلى نهاية الملف. فإذا كان eof(f)=true فإن ذلك يعني أن المؤشر

موجود في نهاية الملف f. أما إذا كان eof(f)=false فإن المؤشر يمكن أن يكون في أي مكان في الملف و لكن ليس في النهاية.

- يمكن حساب عدد عناصر الملف العددي f باستخدام الدالة filesize وذلك كمايلي L :=filesize(f).
- تُمكن الدالة seek(f,i) من الانتقال المباشر إلى العنصر رقم i في الملف f.

3. ملفات السجلات

يُمكن للملفات ذات النوع أن تحوي مجموعة من السجلات. ندعوها عندئذٍ بملفات السجلات. فيمايلي نتناول ملفات السجلات من خلال مجموعة من الأمثلة التوضيحية.

مثال 1:

إنشاء ملف سجلات

ضع برنامجاً بلغة الباسكال يقوم بإنشاء ملفاً لسجلات طلاب الثالث الثانوي العلمي المتقدمين للإمتحان النهائي حيث يُمَثل سجل كل طالب كمايلي:

اسم الطالب، رقم الطالب، الدرجات التي حصل عليها في كل مادة. علماً أن المواد مرقمة من 1 إلى 8 بالترتيب: علوم، قومية، ديانة، فيزياء، كيمياء، رياضيات، انكليزي، عربي.

الحل:

يُعطى البرنامج بالشكل التالي:

```
Program prog4;
  Uses crt;
  Type
      Grade = array [1..8] of 0 .. 60;
```

```
      Student = Record
         Name: String[15];
         Number: integer;
         M:Grade;
       End;
 Var
    Stud: Student;
    f: file of Student;
    Fname: String;
Procedure readRecord (Var stud: Student);
    Var i: Integer;
  Begin
     With stud do
       Begin
          Write('enter name of the student:');
          Readln(Name);
          Write('enter number of the student: ');
          Readln(Number);
          Write('enter the grades of student :');
          For i:= 1 to 8 do
             Readln(M[i]);
       End;
Begin
   Clrscr;
   Write('enter the name of the file: ');
   Readln(Fname);
   Assign(f, Fname);
   Rewrite(f);
   readRecord(stud);
    while((stud.Name<>'zzz')and(stud.Number>0)) do
```

```
        begin
            write(f, stud);
            readRecord(stud);
        End;
    close (f);
End.
```

مثال 2:

اكتب برنامجاً بلغة الباسكال يقوم بقراءة الملف المُنشأ في المثال (1) أعلاه و طباعة سجلات الطلاب الناجحين و مجموع كل منهم علماً بأن الطالب يكون راسباً إذا حقق أحد الشروط التالية:

1. إذا رسب بمادة اللغة العربية.
2. إذا رسب بثلاثة مواد أو أكثر.
3. إذا حصل على مجموع أقل من 96.
4. إذا رسب بمادتين و لم يحصل على ربع مجموعهما.

الحل:

فيما يلي البرنامج الذي يقوم بالعمليات المطلوبة.

```
Program Prog5;
     Uses crt;
     Type
          Grade=array[1..8] of 0 .. 60;
          Student = Record
            Name : String[15];
            Number : integer;
            M: Grade;
          End;
    Var
```

```
    Stud: Student;
    f:file of Student;
    Fname: String;
    t:Boolean;
    s, j, l, w, k: Intger;
procedure writeRecord(Stud: Student);
   var sum, I :Integer;
  Begin
    with Stud do
      Begin
         writeln('Name: ', name);
         writeln('Number: ', Number);
         writeln('Grades of student: ');
         For i:= 1 to 8 do
            Write(M[i], ' ');
         Writeln;
         Sum:= 0;
         For i:= 1 to 8 do
         Sum:= sum + M[i];
         Writeln('Sum = ',sum);
         Writeln('------------');
      End;
  End; {Procedure}
Begin {Main program}
    Write('enter the name of the file: ');
    Readln(Fname);
    Assign(f, Fname);
    Reset(f);
    While(not(eof(f))) do
       Begin
```

```
Read(f, stud);
S:=0; {مجموع طالب}
For j:= 1 to 8 do
  t:= true;
l:=0; {عدد المواد الراسب فيها الطالب}
w:=0; {مجموع درجات المواد الراسبة}
k:=0; {مجموع الدرجات العظمى المقابلة للمواد الراسبة }
With stud do
  Begin
     If(M[1] <12) then {علوم}
       Begin
         l:=l +1;
         w:=w + M[1];
         k:=k + 30;
       End;
     If(M[2]<8) then {القومية}
       Begin
         l:=l +1;
         w:=w + M[2];
         k:=k + 20;
       End;
     If(M[3]<8) then {الديانة}
       Begin
         l:=l +1;
         w:=w + M[3];
         k:=k + 20;
       End;
       .....
```

```
            If(M[8]<40) then {اللغة العربية}
              Begin
               l:=l +3;
              End;
            If(l> 2) then
              t:= false;
            If(s < 96) then
              t:= false;
            If((l=2) and (w<k/4)) then
              t := false;
            If(t) then
              writeRecord(Stud);
        End;
    Readln;
End. {Main}
```

4. الملفات النصية Text Files

يُمكن للملفات النصية تخزين نصوص سلاسل محارف (Strings)، أو قيم عددية. لذلك يُطلق على الملف النصي (text file) بملف ASCII (ASCII file)، حيث يحتوي هذا الملف على معلومات مُمثلة بشفرة ASCII، ويمكننا قراءة الملفات النصية بوساطة أي محرر نصوص. تُستَخدَم الإجراءات write و writeln و read و readln في عمليات الإدخال والإخراج التي تُطبق على الملفات النصية.

يوجد في لغة باسكال نوع بيانات مسبق التعريف هو نوع البيانات ملف نصي (text type) يُستَخدَم هذا النوع لتمثيل الملفات النصية. التعليمة التالية تُبين كيفية التصريح عن متحول ينتمي إلى نوع البيانات ملف نصي:

```
Var
    Fileval : Text;
```

1.4 الإجراء Read مقابل الإجراء Readln Read Vs. Readln

يُستخدم كلا الإجرائين read و readln لقراءة أسطر من الملف. إذا كان كل سطر يحوي قيمة واحدة (سلسلة من المحارف تنتهي بفراغ) فلا يوجد فرق بين read و readln. لكن هناك فرق جوهري بين الإجرائين إذا كان السطر يحوي أكثر من قيمة واحدة. يقوم الإجراء readln بقراءة أول قيمة من السطر وينتقل إلى السطر التالي. هذا يعني أن بقية القيم في السطر لن تُقرأ. أما الإجراء read فيقرأ كل قيم السطر وبعدها ينتقل لقراءة السطر التالي.

مثال:

```
Program ReadDemo2;
Const
     MaxTrials=125;
     MaxPerLine=5;
     FileName='zwzw.wzw';
Type
     Stats=array[0..4] of real;
Var
     SampleFile:Text;
     Index,WhichCell:integer;
     Vals:Stats;
     Result:real;
Function FOpened(Var TheFile:Text; FName:String)
      :boolean;
  Begin
     Assign(TheFile,FName);
     (*$I-*)
     Reset(TheFile);
     (*$I+*)
     If(IOResult=0) then
```

```
        FOpened:=true
      Else
        FOpened:=false;
  End; (*FIN FOpened*)
Procedure InitStats(Var Vals:Stats);
    Var Index:integer;
  Begin
     For Index:=0 to 4 do
        Vals[Index]:=0.0;
  End;
Procedure DisplayMeans(Vals:Stats; ValsPerCell:
     integer);
    Var Index:integer;
  Begin
     For Index:=1 to 4 do
     Writeln(Index,' :' ,Vals[Index]/ValsPerCell
       :10:5);
     Writeln(5, ': ', Vals[0]/ValsPerCeli:10:5);
  End;
Procedure AwaitUser;
     Var Ch:char;
  Begin
     Write('Press Enter to continue.');
     Readln;
  End;
Begin
     Randomize;
     InitStats(Vais);
     If(FOpened(Sample File, FileName)=true) then
      Begin
```

```
        For Index:=1 to MaxTrials do
             Begin
                Readln(SampieFile, Result);
                WhichCall:=Index mod MaxPsrLine;
                Vals[Wh_chCell]:=Vals[WhichCell]+
                  Result;
                Write(Vais[WhichCall]:10:5);
                If(Index mod MaxPerLine=0) then
                  Writeln;
             End;
        DisplayMeans(Vals,MaxTrials div
            MaxPerLine);
        Close(SampleFile);
      End;
     AwaitUser ;
End.
```

إن الإجراء readln ينتقل إلى سطر جديد بعد قراءة كل قيمة، لذلك سوف تُقرأ قيمة واحدة من كل خمس قيم موجودة في السطر، وبالتالي بعد خمسة وعشرين استدعاء للإجراء readln سيصل البرنامج إلى نهاية الملف، والذي يحتوي أصلاً على خمسة وعشرين سطراً. وعندها سيُعيد الإجراء readln القيمة (0) عند استدعائه ولن تتغير قيم حجرات المصفوفة.

هناك فرق كبير في استخدام الإجراءين Read و Readln للحصول على معلومات من الملفات. حيث يُستَخدَم الإجراء Read إذا أردنا قراءة عدة قيم ضمن سطر واحد، لأن استخدام الإجراء Readln في هذه الحالة سوف يُسبب فقدان معظم هذه القيم لأنه يقرأ أول قيمة وينتقل بعدها إلى سطر جديد.

2.4 العمليات على الملفات النصية Operations On Text Files

بالنسبة للملفات النصية فإننا نورد الملاحظات التالية:

1. يجب استخدام الإجراء Assign لربط متحول من النوع Text، مع الملف الموجود على القرص، ويتم ذلك بوساطة تمرير متغير الملف النصي كمتحول وسيطي أول لهذا الإجراء، وتمرير اسم الملف الموجود على القرص كمتحول وسيطي ثانٍ.
2. يُستخدم الإجراء Rewrite لإنشاء ملف جديد وفتحه للكتابة فقط وفي حال وجود ملف يحمل الاسم نفسه، فإن الملف القديم سوف يُستبَدل بالملف الجديد، وتضيع المعلومات الموجودة فيه.
3. يمكن استخدام الإجراء Reset لفتح ملف موجود للقراءة. يجب أن يكون الملف موجوداً وإلا سوف يفشل تنفيذ البرنامج، ويُعطينا رسالة خطأ في التنفيذ.
4. في الملفات النصية (بشكل خاص) نستطيع استخدام التعليمة Append لفتح ملف موجود وكتابة بعض المعلومات في نهايته. أي أن التعليمة Append تُمَكن من فتح ملف نصي موجود مسبقاً و تضع المؤشرة في نهاية ذلك الملف.
5. يُستخدَم الإجراء Close لإغلاق الملف الذي قد فتحناه بأحد الإجراءات السابقة، حيث يُفرّغ هذا الإجراء الذاكرة الوسيطة (Buffer) في الملف الموجود على القرص، ومن ثم يغلق الملف، أي يخزن الملف ويحفظه من الضياع.
6. يُستخدَم الإجراءان Read و Readln لقراءة المعلومات من الملف النصي، وهذه المعلومات يمكن أن تكون من أي نوع بيانات بسيط، عدا نوع البيانات التعدادي ونوع البيانات بولياني ويمكنهما أيضاً قراءة سلاسل محارف من الملف.

7. يُستخدَم الإجراءان Write و Writeln لكتابة المعلومات في الملفات النصية، وهذه المعلومات يمكن أن تكون من أي نوع بيانات بسيط، عدا نوع البيانات التعدادي. ويمكنهما أيضاً كتابة سلاسل محارف على الملف.

8. يمكن استخدام الدالة IO Result لاختبار نتيجة الدخل أو الخرج. وتُستدعى هذه الدالة فقط عندما تكون حالة اختبار الدخل أو الخرج هي الاختبار اليدوي أي $\{\$I-\}$، واعتماداً على القيمة التي تُعيدها هذا الدالة يمكن للبرنامج اتخاذ خطوات مناسبة لضمان سلامة تنفيذ المهمة الموكلة للبرنامج. فالقيمة (0) إذا أعادتها هذه الدالة تعني أن عملية الإدخال أو الإخراج قد تمت بنجاح.

9. كما في الملفات التتابعية، يُستخدَم المُحدِد Eof لتحديد فيما إذا وصول المؤشر إلى نهاية الملف المُحدَد أم لا.

10. يُستخدَم المُحدد Eoln (End of Line) من أجل تحديد وصول المؤشر إلى نهاية السطر الحالي أم لا وذلك فقط في الملفات النصية.

3.4 إعادة تسمية وحذف الملفات Renaming and Erasing Files

تتطلب عملية إدارة الملفات النصية التمكن من إعادة تسمية ملف ما أو حذف هذا الملف إذا تطلّب الأمر. من أجل ذلك زودتنا لغة الباسكال بالإجرائين Rename و Erase. فالأول Rename مهمته إعادة تسمية ملف ما، وذلك بعد ربط اسم هذا الملف بمتحول لهذا الملف عبر الإجراء Assign، ومن ثم يُستدعي الإجراء Rename.

مثال:

```
Assign ( FileVariable, 'FILE.OLD');
Rename ( FileVariable, 'FILE.NEW');
```

أما الإجراء Erase فيحذف الملف من القرص، وذلك بعد ربط هذا الملف بمتحول لهذا الملف وذلك وفق الطريقة التالية:

```
Assign ( FileVariable, 'FILE.OLD');
Erase ( FileVariable);
```

4.4 الدالتان Seek Eof و Seek Elon & Seek Functions Seek Eof Elon

تزودنا لغة الباسكال بالدالتين Seek Eof و Seek Elon اللتين تقومان بما يقوم به نظيراهما الدالتان Eof و Eoln. حيث تُعيد الدالة Seek Eof القيمة True إذا وصل مؤشر الملف إلى نهاية الملف، أما الدالة Seek Elon فتُعيد القيمة True إذا وصل مؤشر الملف إلى نهاية سطر ما، لكن الفرق بين هاتين الدالتين و الدالتين السابقين هو أن الدالتين Seek Eof و Seek Eoln تتجاهلان اختبار رموز التحكم في شفرة ASCII، والتي تتراوح شيفراتها بين (0) و (32)، وبالتالي فإن الدالة Seek Eof تُعيد القيمة True حتى ولو بقي بعض الرموز في الملف، ما دامت هذه الرموز هي رموز تحكم أو رمز الفراغ.

5.4 بعض الأمثلة على الملفات محددة النوع

فيما يلي سنقوم بعرض بعض الأمثلة الداعمة لفهم الملفات محددة النوع.

مثال 1:

يبين البرنامج التالي كيفية التعامل مع الملفات محددة النوع.

```
Program TypedDemo;
    Const FileName='qqqq.int';
          MaxTriais=50;
    Type
          IFile=File of integer;
    Var
          SampleFile:IFile;
Function iFMade(Var TheFile:IFile;FName:string):
```

```
      boolean;
   Begin
     assign(TheFile,FName);
     (*$I-*)
     Rewrite(TheFile);
     (*$I+*)
     If(IOResult=0) then
       iFMade:=true
     Else
       iFMade:=false;
   End;
Function iFOpened(Var TheFile:IFile;FName:string)
     :boolean;
   Begin
     assign(TheFile,FName);
     (*$I-*)
     Reset(TheFile);
     (*$I+*)
     If(IOResult=0) then
        iFOpened:=true
     Else
     iFOpened:=false;
   End;
Procedure DispInts(Value,Precision,Count,
     NrPerLine:integer);
   Begin
     Write(Value:Precision);
     If(Count mod NrPerLine=0) then
       Writeln;
   End;
```

```
Procedure GenerateInts(Var
        TheFile:IFile;Nr:integer);
    Var Count, Result:integer;
  Begin
     For Count:=1 to Nr do
       Begin
          Result:=Random(MAXINT);
          Write(TheFile, Result);
          DispInts(Result, 10, Count, 5);
       End;
  End;
Procedure FReadInts(Var TheFile:IFile);
   VAR Count,Result:integer;
  Begin
     Count:=0;
     While (not EOF(TheFile)) do
          Begin
             Read(TheFile, Result);
             INC(Count);
             DispInts(Result,10,Count,5);
          End;
  End;
Procedure DrawLine(Length:integer;Element:char);
     Var Count:integer;
  Begin
     For Count:=1 to Length do
        Write(Element);
     Writeln;
  End; (*DrawLine*)
Procedure AwaitUser;
```

```
    VAR Ch : Char;
  Begin
    Write('Press ENTER to continue');
    Readln;
  End;
Begin (*Main Program*)
    Randomize;
    If(iFMade(SampleFile, FileName) then
      Begin
        GenerateInts(SampleFile,.MaxTrials);
        Close(SampleFile);
        DrawLine(50,'-');
        If(iFOpened(SampleFile, FileName)) then
          Begin
            FReadInts(SampleFile);
            Close(SampleFile);
          End
        Else
          Writeln('Could not open', FileName);
      End
    Else
        Writeln('Could not create',FileName);
    AwaitUser;
End.
```

يولد المقطع البرمجي السابق (50) قيمة صحيحة شبه عشوائية، ويخزنها في الملف ويُظهرها على الشاشة، ومن ثم يقرأ هذه القيم من الملف ويُظهرها مرة أخرى.

بعد الانتهاء من تنفيذ البرنامج السابق افتح الملف QQQQ.INT بوساطة الضغط على المفتاح F3 لاستدعاء خيار فتح الملفات (open files box)، ومن ثم

اكتب اسم الملف qqqq.int. اضغط مفتاح الإدخال، عندئذ سوف يقوم محرر نصوص لغة الباسكال بتحميل هذا الملف وإظهاره، لكن ستفاجئ برؤية رموز لا معنى لها على الشاشة، وذلك بسبب أن الملف ليس ملفاً نصياً، لأن البرنامج قد خزَّن هذا الملف وفق الصيغة الثنائية (binary format) وليس بصيغة ASCII (ASCII format)، لذلك لا نستطيع عرض هذه الملفات أو تنقيحها بوساطة محررات النصوص.

الاجراء مسبق التعريف Seek يمكننا من الإنتقال من أي موقع ضمن الملف إلى الموقع المطلوب ضمن الملفات المحددة النوع، وهذا الاجراء يأخذ متحولين وسيطين هما:

- الملف المحدد النوع المراد البحث ضمنه.
- متحول من النوع longint يحدد الموقع المراد الانتقال إلية ضمن الملف.

مثال 2:

البرنامج التالي يكتب كمية كبيرة من الأعداد الحقيقية شبه العشوائية في ملف، ومن ثم يُخزن هذا الملف، وبعد ذلك يُعيد البرنامج فتحه لقراءة عناصر مختارة منه. يتم تحديد هذه العناصر المختارة بوساطة تحديد موقعها.

```
Program SeekDemo;

Const
     MaxItems=2000;
     FileName='qqqq.see';
Type
     RFile=File of real;
Var
     RF:RFile;
     Count:integer;
```

```
      WhichElem:longint;
      CurrVal:real;
Procedure AwaitUser;
      Var ch:char;
   Begin
       Write('Press ENTER to continue');
       Readln;
Function rFMade(Var TheFile:RFile;FName:string):
          boolean;
     Begin
          assign(TheFile,FName);
          (*$I-*)
          Rewrite(TheFile);
          (*$I+*)
          If(IOResult=0) then
             rFMade:=true;
          Else
             rFMade:=false;
     End; (*FN rFMade*)
Function rFOpened(Var TheFile:RFile;FName:
         string):boolean;
     Begin
          assign(TheFile, FName);
          (*$I-*)
          Reset(TheFile);
          (*$I+*)
          If(IOResult=0) then
             rFOpened:=true
          Else
             rFOpened:=false;
```

```
    End;
Begin
    Randomize;
    If(rFMade(RF,FileName)) then
      Begin
        For Count:=1 to MaxItems do
          Begin
            CurrVal:=Random;
            Write(RF,CurrVal);
          End;
        Close(RF);
        Writeln('Done') ;
        If(rFOpened(RF,FileName)) then
          Begin
            For Count:=1 to 10 do
              Begin
                WhichElem:=Random(100)+1900;
                Seek(RF,WhichElem);
                Read(RF,CurrVal);
                Writeln(WhichElem:5,':',CurrV
                  al:10:5);
              End;
            Close(RF);
          End
        Else
          Writeln('Could not open',FileName);
      End
   Else
      Writeln('Could not create',FileName);
  AwaitUser;
```

```
End.
```

إن استدعاء الإجراء seek يضع المؤشر عند العنصر المطلوب، هذا يعني أن القيمة التالية التي ستُقرأ من الملف ستكون العنصر المطلوب نفسه.

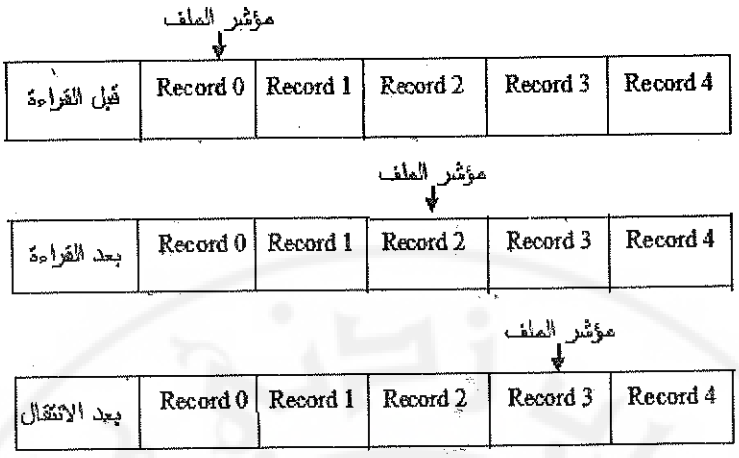
من أجل إيضاح مفهوم مؤشر الملف (file pointer)، ولتمثيل عملية انتقال هذا المؤشر نورد التعريف التالي:

```
Type
     ExampleRecord=Record
       Fieldl,Field2:integer;
     End;
     ExampleFila=File of ExampleRecord;
Var
     Efile:ExampleFile;
     Value:ExampleRecord;
```

من خلال التعريف السابق تم تعريف نوع البيانات ExampleFile على أنها نوع بيانات مؤلف من ملف محدد النوع، ونزع بياناته الأساسي هو السجل ExampleRecord، فإذا كان لدينا العبارات التالية:

```
Seek(Efile,2);
Read(Efile,Value);
Seek(Efile,3);
```

عندئذٍ يمكن تمثيل الملف السابق بالشكل التالي، الذي يبين الملف أثناء كل عبارة من العبارات السابقة:



إن طريقة الوصول العشوائية أسرع في الوصول إلى المعلومات المطلوبة منها في الطريقة التسلسلية، وخاصة في الملفات الكبيرة.

مثال 3:

ملف يحتوي على مصفوفات A File Containing Arrays

يبين البرنامج التالي كيفية تعريف ملف عناصره مصفوفات تتألف من عشرين عنصراً ينتمي إلى نوع البيانات عدد حقيقي.

```
Program ArrayTypedDemo;
Const
     MaxArrays=3;
     ArrSize=20;
     FileName='qqqq.rea';
Type
     RStats=array[1..ArrSize] of Real;
     RFile=File of RStats;
Var
     RF:RFile;
     RData:RStats;
     ArrCount,Count:integer;
```

```
     NrElements:longint;
Procedure AwaitUser;
     Var Ch:char;
   Begin
     Write('Press ENTER to continue.');
     Readln;
   End;
Function rFMade(Var TheFile:RFile;FName:string):
        boolean;
   Begin
      assign(TheFile,FName);
      (*$I-*)
      Rewrite(TheFile);
      (*$I+*)
      If(IOResult=0) then
         rFMade:=true
      Else
         rFMade:=false;
  End; (*FN rFMad*)
Function rFOpened(Var TheFile:RFile;FName:string)
       :boolean;
     Begin
          assign(TheFile,FName);
          (*$I-*)
          Reset(TheFile);
          (*$I+*)
          If(IOResult=0) then
               rFOpened:=true
          Else
               rFOpened:=false;
```

```
    End; (*FN rFOpened*)
Procedure DispReals(Value:Real;Width,Precision,
        Count,NrPerLine:integer);
    Begin
        Write(Value:Width:Precision);
        If(Count mod NrPerLine=0) then
           Writeln;
    End;
Procedure DrawLine(Length:integer;Element:char);
    Var Count:integer;
  Begin
      For Count:=1 to Length do
         Write(Element);
      Writeln;
  End;
Begin (*Main program*)
  Randomize;
  Writeln('RStats take',sizeof(RStats),'bytes
    each');
  If(rFMade(RF,FileName)) then
    Begin
        Writeln('Writing arrays…');
        For Count:=1 to MaxArrays do
           Begin
              For ArrCount:=1 to ArrSize do
                 RData[ArrCount]:=Random;
                 Write(RF,RData);
           End;
        Close(RF);
        Writeln('Done writing.');
```

```
        If(rFOpened(RF,FileName)) then
          Begin
            While(not EOF(RF)) do
               Begin
                  DrawLine(50,'-');
                  Read(RF,RData);
                  INC(ArrCount);
                  For ArrCount:=1 to ArrSize do
                     DispReals(RData[ArrCount],
                        10,5,ArrCount,5);
               End;
            DrawLine ( 50'-' ,);
          End
    Else
     Begin
       Writeln('Could not open',FileName);
       NrElements:=FileSize(RF);
       Writeln(FileName,'contains',NrElements,  '
         elements(arrays)' );
       Close(RF);
       AwaitUser;
     End
   Else
       Writeln('Could not create',FileName);
End.
```

بعد أن عرفنا نوع البيانات RStats على أنه مصفوفة من الأعداد الحقيقية، عرفنا نوع البيانات rFile، وهو عبارة عن ملف يتألف من مصفوفات.

يشبه الإجراءان rFMade و rFOpened الإجرائين المعرفين في برامج سابقة، والفرق الوحيد فقط هو الوسيط الأول في هذين الإجراءين. يحتوي البرنامج السابق

على بعض الإجراءات مثل الإجراء AwaitUser ومهمته انتظار أن يضغط المستخدم أي مفتاح ليستأنف البرنامج عمله. لدينا أيضاً الإجراء DrawLine الذي يكتب الرمز المُمرَر إليه ضمن وسيطه الثاني، عدداً محدداً من المرات يساوي القيمة الممررة إليه في الوسيط الأول. ولدينا الإجراء DispReals الذي يُظهر القيم الحقيقية على الشاشة.

إنَّ استدعاء الإجراءين Read و Write لا يختلف عن استدعائهما من أجل ملف له نوع آخر يختلف عن نوع بيانات هذا الملف، لكن تختلف كمية ونوع البيانات التي يتعامل معها هذان الإجراءان من نوع بيانات إلى آخر، وذلك حسب نوع البيانات الأساسي الخاص بالملف.

استدعاء الدالة مسبقة التعريف FileSize، حيث تأخذ هذه الدالة متحولاً وسيطياً واحداً عبارة عن متحول الملف محدد النوع، وتُعيد هذه الدالة عدد العناصر المحتواة في هذا الملف وليس عدد بايتات هذا الملف.

مثال 4:

ملف يحتوي على سجل ومجموعة A File Containing Record Set

يُمكننا إنشاء ملف محدد النوع أكثر تعقيداً، حيث يستخدم البرنامج التالي نوع المعطيات سجل كنوع بيانات أساسي له، وأحد حقول هذا السجل أيضاً تنتمي إلى نوع البيانات مجموعة CharSet وهي عبارة عن مجموعة من الرموز.

```
Program ChRecTypedDemo;
Const
     MaxTrials=10;
     FileName='qqqq.crc';
Type
     CharSet=set of char;
     ChRec=Record
```

```
        SetInfo:CharSet;
        MaxItems:integer;
      End;
      ChRecFile=File of ChRec;
Var
      ChRF:ChRecFile;
      ChRecData:ChRec;
      SetCount,NrTimes,ArrCount,Count:integer;
      ChIndex:char;
Procedure AwaitUser;
      Var Ch:char;
   Begin
        Write('Press Enter to continue.');
        Readln;
   End;
Procedure DrawLine(Length:integer;Element:char);
     Var Count:integer;
   Begin
        For Count:=1 to Length do
          Write(Element) ;
        Writeln;
   End;
Function chrFMade(Var TheFile:ChRecFile;
        FXama:string):boolean;
   Begin
        assign(TheFile,FName);
        (*$I-*);
        Rewrite(TheFile);
        (*$I+*)
        If(IQResult=0) then
```

```
          chrFMade:=true
        Else
         chrFMade:=false;
End;
Function chrFCoened(Var TheFile:ChRecFile;FName:
       string):boolean;
   Begin
        assign(TheFile,FName);
        (*$I-*)
        Reset(TheFile);
        (*$I+*)
        If(IOResult=0) then
          chrFOpened:=true
        Else
          chrFOpened:=false;
   End;
Begin
     Randomize;
     Writeln('CharSets take',sizeof(CharSet),'
        bytes');
     Writeln('ChRecs take',sizeof(ChRec),'
       bytes');
     If(chrFMade(ChRF,FileName)) then
       Begin
          For Count:=1 to MaxTrials do
             Begin
               NrTimes:=Random(27);
               ChRecData.SetInfo:=[];
               ChRecData.MaxItems:=NrTimes;
               For SetCount:=1 to NrTimes do
```

```
            With ChRecData do
              SetInfo:=SetInfo+[CHR(Random
                 (26)+97)];
              Write(ChRF,ChRecData);
      End;
 Close(ChRF);
 DrawLine(50,'-');
 If(chrFOpened(ChRF,FileName)) then
  Begin
    While(not EOF(ChRF)) do
      Begin
        Read(ChRF,ChRecData);
        Write('Max=',ChRecData.MaxItems,':');
        For ChIndex :='a' to 'z' do
           Begin
              If(ChIndex in
                ChRecData.SetInfo) then
                  Write(ChIndex,' ');
           End;
        Writeln;
      End;
    DrawLine(50,'-');
    Writeln(FileName,'Contains',FileSize,
       (ChRF),elements,(ChRec));
    AwaitUser;
  End (*If chrFopenend then*)
 Else
    Writeln('Could not open',FileName);
 End (*If chrFMade *)
Else
```

```
        Writeln('Could not Creet',FileName);
End.
```

مثال 5:

ملف يحتوي على نوع بيانات تعدادي File Containing Enumerated Type

يُمكننا أيضاً تعريف ملف محتوياته تنتمي لنوع بيانات تعدادي، ولكن قبل كل شيء علينا التذكر أننا لا نستطيع كتابة القيم التعدادية مباشرة على الشاشة، أو على ملف نصي لكن يمكننا كتابة هذه القيم في الملفات محددة النوع، وذلك لأن القيم الترتيبية يمكن تخزينها بسهولة، أما إذا أردت إظهار هذه القيم على الشاشة فهناك عدة خطوات يجب إتباعها لإظهار القيم التعدادية. يبين البرنامج التالي ملف يحتوي بيانات تعدادية.

```
Program EnTypeDemo;
Const
     FileName='qqqq.clr';
Type
     Color=(red,green,blue,yellow,white,black);
     CFile=File of Color;
Var
     CF:CFile;
     CurrColor:Color;
Procedure AwaitUser;
     Var Ch:char;
   Begin
      Write('Press ENTER to continue.');
      Readln;
   End;
Procedure DrawLine(Length:integer;Element:char);
     Var Count:integer;
```

```
  Begin
     For Count:=1 to Length do
        Write(Element);
     Writeln;
  End;
Function cFMade(Var TheFile:CFile;FName:string):
       boolean;
  Begin
     assign(TheFile,FName);
     (*$I-*)
     Rewrite(TheFile);
     (*$I+*)
     If(IQResult=0) then
        cFMade:=true
     Else
        cFMade:=false;
  End;
Function cFOpened(Var TheFile:CFile;FName:string)
      :boolean;
   Begin
        assign(TheFile,FName);
        (*$I-*)
        Reset(TheFile);
        (*$I+*)
        If(IOResult=0) then
          cFOpened:=true
        Else
          cFOpened:=false;
   End;
Begin (*Main Program*)
```

```
If(cFMade(CF,FileName)) then
  Begin
     CurrColor=red;
     Write(CF,CurrColor);
     CurrColor:=black;
     Write(CF,CurrColor);
     CurrColor:=green;
     Write(CF,CurrColor);
     CurrColor:=white;
     Write(CF,CurrColor);
     CurrColor:=blue;
     Write(CF,CurrColor);
     CurrColor:=yellow;
     Write(CF,CurrColor);
     Close(CF);
     If(cFOpened(CF,FileName)) then
        Begin
          While(not EOF(CF)) do
             Begin
               Read(CF,CurrColor);
               Case CurrColor of
                  red:Writeln('red');
                  green:Writeln('green');
                  blue:Writeln('blue');
                  yellow:Writeln('yellow');
                  white:Writeln('white');
                  black:Writeln('black');
             End;
        End;
     Close(CF);
```

```
If(cFMade(CF,FileName)) then
  Begin
     CurrColor=red;
     Write(CF,CurrColor);
     CurrColor:=black;
     Write(CF,CurrColor);
     CurrColor:=green;
     Write(CF,CurrColor);
     CurrColor:=white;
     Write(CF,CurrColor);
     CurrColor:=blue;
     Write(CF,CurrColor);
     CurrColor:=yellow;
     Write(CF,CurrColor);
     Close(CF);
     If(cFOpened(CF,FileName)) then
        Begin
          While(not EOF(CF)) do
             Begin
               Read(CF,CurrColor);
               Case CurrColor of
                  red:Writeln('red');
                  green:Writeln('green');
                  blue:Writeln('blue');
                  yellow:Writeln('yellow');
                  white:Writeln('white');
                  black:Writeln('black');
               End;
             End;
          Close(CF);
```

```
Const
     MaxTrials=10;
     FileName='qqqq.unt';
Var
     UT:File;
     Count:integer;
     Val:Real;
Procedure AwaitUser;
     Var Ch:char;
   Begin
       Write('Press ENTER to continue');
       Readln;

   End;
Procedure DrawLine(Length:integer;Element:char);
     Var Count:integer;
   Begin
        For Count:=1 to Length do
           Write(Element);
        Writeln;
   End;
Function utFMade(Var TheFile:File;FName:string;
        ChunkSize:word):boolean;
    Begin
        assign(TheFile,FName);
        (*$I-*)
        Rewrite(TheFile,ChunkSize);
        (*$I+*)
        If(IOResult=0) then
           utFMade:=true
```

```
        Else
           utFMade:=false;
        End; (*FN utFMads*)
Function utFOpened(Var TheFile:File;FName:string;
        ChunkSize:word):boolean;
   Begin
        assign(TheFile,FName);
        (*$I-*)
        Reset(TheFile,ChunkSize);
        (*$I+*)
        If(IOResult=0) then
          utFOpened:=true
        Else
          utFOpened:=false;
   End;
Procedure
DispReals(Value:Real;Width,Precision,Count,
       NrPerLine:integer);
  Begin
       Write(Value:Width:Precision);
       If(Count mod NrPerLine=0) then
          Writeln;
  End;
Begin (*Main program*)
   Randomize;
   If(utFMade(UT,FileName,6)) then
     Begin
          For Count:=1 to MaxTrials do
             Begin
               Val:=Random;
```

```
            BlockWrite(UT,Val,1);
            DispReals(Val,10,5,Count,5);
          End;

      Close(UT);
      DrawLine (50,'-');
      If(utFOpened(UT,FileName,6)) then
         Begin
           While(not EOF(UT)) do
             Begin
                BlockRead(UT,Val,1);
                INC(Count);
                DispReals(Val,10,5,Count,5);
              End;
            DrawLine(50,'-');
            AwaitUser;
            Close(UT);
         End
       Else
          Writeln('Could not open', FileName);
       End
  Else
      Writeln('Could not create',FileName);
End.
```

تأخذ الدالتين utFMade و utFopened وسيطاً آخر إضافة إلى وسطائهما المستخدمة ضمن الملفات محددة النوع، وهذا الوسيط الثالث يحدد عدد البايتات التي سوف تعتبر سجلاً واحداً: عند نقل المعلومات من وإلى الملف.

إن القيمة النظامية لحجم الدفعات التي تُنقَل بين البرنامج و الملف هي (128) بايتاً، لكن يمكننا اختيار حجم معين ، وهذا الحجم يعتمد على نوع المعلومات التي

ستُخَزن ضمن الملف. فإذا كنا نُخَزن قيماً صحيحة فيجب علينا نقل بايتين في كل مرة، أما إذا كان الملف مهمل النوع يُخزن مزيجاً من المعلومات، فإن علينا استخدام خوارزمية آمنة في النقل، وهي نقل المعلومات بايتاً بايتاً، ومن ثم يوكل للبرنامج مهمة تجميع هذه البايتات وربطها لبناء القيم التي نريد. إذاً الوسيط ChunkSize يُستخدَم ضمن استدعاء الإجراءين Reset و Rewrite ضمن الدالتين utFopened و utFMade على الترتيب، وقد أضيف هذان الإجراءان لكي يتعاملا مع الملفات مهملة النوع، ولا يُستخدمان عند التعامل مع الملفات محددة النوع أو الملفات النصية.

لقد ذكرت آنفاً أننا نستخدم الإجراء BlockWrite من أجل كتابة المعلومات في الملفات مهملة النوع، وهذا الإجراء يأخذ ثلاثة متغيرات وسيطية، ويمكن أن يأخذ متحولاً وسيطياً رابعاً إذا أردنا، وهذه المتغيرات الوسيطة هي:

1. المتغير الوسيطي الأول هو متحول الملف مهمل النوع.
2. المتغير الوسيطي الثاني يمكن أن يكون أي متحول، أو موقع في الذاكرة. حيث يفهم الإجراء هذا المتغير على أنه سلسلة مواقع من الذاكرة الوسيطة، طولها يُحدده المتغير الوسيطي الثالث، وهذا المتغير الوسيطي الثاني يجب أن يُمرر مرجعياً.
3. المتغير الوسيطي الثالث يُحدد عدد البايتات التي سوف تُقرأ في كل مرة، وهذا المتغير يجب أن يكون من النوع Word، وحجم هذه الدفعات كما ذكرت يعتمد على نوع البيانات المستخدمة، وهذا المتغير الوسيطي يجب أن يُمرر بوساطة قيمته.
4. يمكننا إضافة متحول وسيطي رابع ينتمي أيضاً لنوع البيانات Word، حيث يزودنا هذا المتغير الوسيطي بمعلومات حول عدد الدفعات التي كُتبت بعد الانتهاء من تنفيذ هذا الإجراء. يُعيد هذا الإجراء قيمة ضمن

هذا المتغير الوسيطي تمثل عدد البايتات التي كتبها، وهذا العدد يمكن أن يكون مساوياً لقيمة المتغير الوسيطي الثالث أو أقل، وأيضاً يجب أن يُمرَر مرجعياً.

إذا امتلأ القرص بالمعلومات قبل أن يُنهي الإجراء BlockWrite عملية كتابة كل البايتات المحددة له، فإن بعض هذه المعلومات سوف تُكتَب والأخرى سوف تُهمَل. لذلك من أجل معرفة عدد البايتات التي كُتبت، يمكننا اختبار الوسيط الرابع، والذي يحتوي على عدد البايتات التي كُتبت، وبعد أن ينهي الإجراء BlockWrite عمله، فإن مؤشر الملف يُعدَل تلقائياً ليصبح في نهاية الملف، حيث ستُضاف المعلومات الجديدة.

يملك الإجراء BlockRead نفس الوسطاء المُستخدمة مع الإجراء BlockWrite، حيث يقرأ الإجراء BlockRead عدداً محدداً من البايتات من ملف مهمل النوع، ويُخزن هذه البايتات في موقع من الذاكرة محدد من خلال وسيطه الثاني، والذي يمكن أن يكون متحولاً أيضاً. وأخيراً حتى تعمل هذه الإجراءات يجب أن يُفتَح الملف مهمل النوع أولاً.

توجد إجراءات ودوال أخرى تأخذ الملفات كمتغيرات وسيطية لها. يحتوي الجدول التالي على الإجراءات والدوال الرئيسية:

الإجراء	ملف نصي	ملف محدد النوع	ملف مهمل النوع	ملاحظات
Assign	Ok	Ok	Ok	الملف موجود
Reset	Ok	Ok	Ok	
Rewrite	Ok	Ok	Ok	
Close	Ok	Ok	Ok	
Append	Ok			

الملف مفتوح		Ok Ok	Ok Ok Ok Ok	Write Writeln Read Readln
الملف مفتوح	Ok Ok			BlockWrite BlockRead
الملف مفتوح يحتاج لـ (*$I-*)	Ok Ok Ok Ok	Ok Ok Ok Ok	 Ok Ok Ok	Seek FileSize Eof Eoln IOResult

6. أجهزة الدخل والخرج القياسي The Standard Input and Output Devices

انطلاقاً من مفهوم أن عمليات الدخل والخرج في لغة الباسكال تتم عبر الأجهزة فإننا نستطيع القول بأن استدعاء الإجراء Readln يُخبر باسكال بأن تقبل الإدخالات من جهاز الدخل القياسي، كذلك الأمر بالنسبة للإجراء Writeln الذي يُخبرها بأن تُرسل المخرجات إلى جهاز الخرج القياسي.

يُدعى الجهاز CON بجهاز الدخل والخرج القياسي (console) وهو يشير إلى لوحة المفاتيح كجهاز دخل والشاشة كجهاز خرج، وعلى ذلك يمكن استخدام الجهاز CON كأي ملف من ملفات القرص، وبذلك يمكن استعماله مع إجراءات مثل Reset و Rewrite.

مثال:

```
Program DeviceTest;
Var
     TextFile:text;
```

```
    Str:string;
Begin
  assign(TextFile,'CON');
  Rewrite(TextFile);
  Writeln(TextFile,'Output to CON') ;
  Writeln;
  Writeln('Enter string using Readln(Str)');
  Write('Type a string. Press ENTER when done:');
  Readln(Str);
  Writeln('>',Str);
  assign(TextFile,'CON');
  Reset(TextFile);
  Writeln('Enter string from using Readln(
        TextFile,Str)');
  Write('Type a string.Press ENTER when done:');
  Readln(TextFile,Str);
  Writeln('>',Str);
  Writeln;
  Write('Press Enter');
  Readln;
End.
```

1.6 أجهزة الطابعة Printer Devices

يدعم نظام التشغيل عدة أجهزة طابعة هي: PRN و LPT1 و LPT2 و LPT3 علماً بأن الجهاز PRN هو نفسه LPT1. يستخدم معظم الناس طابعة واحدة موصولة مع منفذ الطابعة الأول. نشير عادة إلى الطابعة بالجهاز PRN الذي يستخدم الطابعة كجهاز خرج فقط لذلك إذا حاولنا فتح الجهاز PRN للقراءة عبر الإجراء Reset فإن باسكال سوف تولد نهاية للملف مباشرةً (أي ستعيد الدالةEof القيمة True).

إذاً لإرسال المعلومات إلى الطابعة يُفتَح ملف الجهاز PRN للكتابة وتجري عليه عملية الكتابة. توجد طريقة أخرى في الكتابة على الطابعة وذلك من خلال استخدام وحدة مُسبقة التعريف في لغة الباسكال هي Printer[4]. تحتوي هذه الوحدة على متحول ملف نصي مُسبق التعريف يُدعى LST وظيفته توجيه المخرجات مباشرة إلى الطابعة.

مثال:

```
Program TestPtr;
Uses PRINTER;
Begin
     Writeln(LST,'ABC');
End.
```

2.6 الأجهزة التسلسلية Serial Devices

تملك معظم الأجهزة منافذ تسلسلية (serial ports) تُستخدَم لربط الطابعات أحياناً والموديمات والشبكات المحلية ولأغراض اتصالات أخرى.

تدعم لغة الباسكال جهازين تسلسلين هما COM1 و COM2 بالإضافة إلى الجهاز الوسيطي AUX والذي يشير إلى COM1، وبالتالي فهما متكافئان.

مع أن الأجهزة جعلت إرسال واستقبال المعلومات عبر المنافذ التسلسلية أمراً سهلاً إلا أن استخدامها محدود لأغراض خاصة جداً، فبرامج الاتصالات مثلاً تتجاهل عادة أجهزة نظام التشغيل ومن ثم تذهب مباشرة إلى المنفذ التسلسلي.

3.6 الجهاز NUL The NUL Device

تملك لغة الباسكال جهازاً آخر هو الجهاز NUL. لهذا الجهاز استخدام خاص مميز جداً إذ أنه يتجاهل أي شيء يُرسل إليه. لا بد أنك مستغرب من

---

[4] سنقوم بعرض مفهوم الوحدة في لغة الباسكال بشيء من التفصيل في الفصل السادس من هذا الكتاب.

أهمية جهاز كهذا ولكن أهميته تكمن عند تصميم برنامج اتصالات يقوم بإرسالات. فعندما نريد تجريب مراحل البرنامج عدا مرحلة الإرسال فإن الجهاز عندها سيحل محل الجهاز الذي تريد الإرسال إليه، وبعد الانتهاء من اختبار مراحل البرنامج التي لا تتعلق بالإرسال يمكن إعادة الجهاز الأصلي وتجريب مرحلة الإرسال فقط.

Damascus University

تمارين

1) لنفترض أن X متحول من النوع الحقيقي، N متحول من النوع الصحيح، C متحول محرفي، و S متحول شريطي. ولنفرض أن لدينا المعلومات التالية المخزنة في ملف اسمه InData.

```
123 3.145 XYX<eoln>
35 Z<eoln>
```

بين محتويات كل متحول من المتحولات X, N, C, S بعد تنفيذ تعليمة Read و Readln التالية:

1. Readln(InData, N, X)
2. Read(InData, N, X, C)
3. Read (InData, N, X, C, C)
4. Readln(InData, N); Read(InData, C
5. Readln(InData, C, C, C, X); Read(InData, C, N)

2) اكتب برنامج بلغة باسكال يقوم بفتح ملف أعداد صحيحة ثم يقوم بقراءة كل عدد ويحسب جذره التربيعي ومربعه. النتائج يجب أن تُخزَن في ملف آخر.

3) لنفترض أننا نريد تعريف ملف بمعلومات عن طلاب السنة الثانية. كل طالب يجب أن يُعرَّف بسجل يحتوي على المعلومات التالية:

اسم الطالب، الكنية، السنة الدراسية، تاريخ ميلاد الطالب، متجهة تحتوي على علامات المواد التي نجح فيها الطالب. ضع برنامجاً بلغة الباسكال يقوم بما يلي:

1. تعريف الملف السابق.
2. كتابة إجراء يقوم بإدخال معلومات الطلاب وتخزينها في الملف السابق.
3. كتابة إجراء يقوم بالبحث عن طالب معين من خلال اسمه.

4. كتابة دالة تقوم بقراءة طالب معين من الملف وحساب معدله في المواد التي نجح فيها.
5. كتابة إجراء يقوم بحساب معدل الطلاب في الملف ويطبع النتيجة على الشاشة.
6. كتابة إجراء يقوم بتحديث سجل طالب بدلالة دليل هذا السجل في الملف. (اختر أي معلومة تريد تحديثها).
7. كتابة إجراء يقوم بطباعة أسماء الطلاب الذين يتجاوز عمرهم مقداراً معيناً.

4) لنفترض أن لدينا ملفاً بأسماء الزبائن في شركة اتصالات ولنفرض أيضاً أن معلومات كل زبون معرفة بسجل مكون من الاسم، الكنية، ورقم الهاتف. هذا الملف غير مرتب ونريد ترتيبه. اكتب برنامج بلغة الباسكال يقوم بفتح ملف الزبائن ويقوم بترتيبه.

5) ادرس جيداً البرنامج التالي و الذي يقوم بتعديل الملف IntFile و بحيث يستبدل كل عدد زوجي بـ 1-. حدد جميع الأخطاء سواءً أكانت لغوية (compile errors) أو منطقية (run time) و أضف أي تعليمات ناقصة (إن وجدَت) بحيث يُصبح البرنامج قابلاً للتنفيذ.

```
Program A;
  Type
     f = file of Integer;
  Var
     IntFile: f; num,u: integer;
  Begin
      Rewrite(IntFile);
      Assign(IntFile);
      u:= 0;
```

```
While (eof(IntFile)) do
 Begin
    Read(num, IntFile);
    If(num mod 2 = 0) then
       Begin
           num := -1;
           …….
           Write(IntFile, num);
       End; {if}
   u:= u +1;
 End; {while}
…..
End. {main program}
```

6) ضع برنامجاً بلغة الباسكال يقوم بإدخال مجموعة من الأعداد الصحيحة الموجبة المنتهية بالصفر (أو أي عدد سالب) و إنشاء ملف يحتوي على العناصر الزوجية منها.

7) ضع برنامجاً بلغة الباسكال يقوم بإنشاء ملف نصوص ينتهي بالحروف zzz في السطر الأخير. ثم ضع برنامجاً يقوم بقراءة الملف السابق و حساب عدد أسطره و طباعتها على الشاشة كما هي في الملف ثم طباعتها معكوسة (أي السطر الأخير في البداية و السطر الأول و النهاية وهكذا...). أخيراً ضع برنامجاً يقوم بقراءة الملف السابق و طباعة أسطره معكوسة على ملف جديد و حساب عدد كلمات الملف الأصلي.

8) اكتب برنامجاً بلغة الباسكال يقوم بقراءة ملف أعداد صحيحة موجود على القرص ثم يُنشئ ملفاً آخر يُخزَن فيه مربعات الأعداد االصحيحة المقروءة.

المؤشرات Pointers

1. مفهوم المؤشر Pointer Concept

بشكل عام يمكن تقسيم أنواع المعطيات إلى نوعين: بنى المُعطيات الساكنة (static) و بنى المُعطيات المتحركة (dynamic). بنى المُعطيات الساكنة يمكن أن تكون بسيطة مثل الأنواع القياسية البسيطة أو مركبة مثل المصفوفات و السجلات. أما بنى المُعطيات المتحركة هي بنى معطيات ديناميكية مثل الملفات.

في هذا الفصل سوف ندرس نوعاً أخر يُدعى بالمؤشرات. تُستخدَم المؤشرات من أجل الوصول إلى حجرات الذاكرة المخصصة لتخزين المتحولات. سنقوم وبمساعدة المؤشرات باستخدام طريقة جديدة في التعامل مع الذاكرة وهي تخصيص حجرات الذاكرة (حجز المتغيرات) أثناء تنفيذ البرنامج، وليس أثناء ترجمة (compile) البرنامج. هذه الطريقة تُمكننا من القيام بمهام صعبة، وقد تكون مستحيلة بدون نوع البيانات مؤشرات. إن تخصيص حجرة ذاكرة في أثناء تنفيذ البرنامج يسمح لنا بالتعامل مع كتل لا نعرف هل ستملك قيماً أم لا.

2. التخصيص الساكن للذاكرة Static Memory Allocation

عندما نُصرح عن متحول ضمن برنامج مكتوب بلغة الباسكال، فإن المترجم يعرف تماماً كمية الذاكرة التي يحتاجها هذا المتغير. حيث يُخصص (allocate) المترجم حجرات الذاكرة الخاصة بالمتغيرات العامة (global variables)، والثوابت (constants) ضمن مقطع البيانات (data segment). ومقطع البيانات هذا عبارة عن مساحة من الذاكرة مُحدودة الطول، يُحدَد ويُقرَر حجمها بناءً على عدد ونوع المتغيرات العامة والثوابت المُصرَح عنها ضمن البرنامج. أما بالنسبة للمتغيرات المحلية (local variables)، والوسطاء (parameters) المُمرَرة إلى الدوال، فإن المترجم يُخصص لها حجرات في الذاكرة عندما يتطلب تنفيذ البرنامج ذلك. لكن

المترجم يحجز بشكل أولي كمية من الذاكرة تُدعى بمقطع المكدس (stack segment) من أجل استخدامها لهذه المتغيرات المحلية. طول مقطع المكدس ثابت، ويُحدَد في أثناء ترجمة البرنامج. يمكن الاستعانة بتوجيه المترجم {$-S} لتحديد طول المكدس (stack) في أثناء ترجمة البرنامج، وإلا فإن الطول الافتراضي لهذا المكدس سوف يُستخدَم تلقائياً.

يُعتبر مقطع البيانات ومقطع المكدس أمثلة عن التخصيص الساكن للذاكرة (static memory allocation) وقد استخدمنا صفة ساكن (static) لأن كمية الذاكرة المُخصصة يتم تحديدها في أثناء ترجمة البرنامج وقبل تنفيذه، ولأن هذه الكمية ثابتة لا تتغير أبداً.

مساوئ التخصيص الساكن للذاكرة Drawbacks of Static Memory Allocation

قد لا نستطيع في بعض الأحيان تخصيص حجرات من الذاكرة بشكل ثابت. فمثلاً: لكتابة برامج لتصنيف بيانات عددية، فإن هذا البرنامج سوف يحصل على البيانات من المستخدم، لكن عدد هذه البيانات الواجب تخزينها وتصنيفها ليس ثابتاً. فمن أجل القيام بهذه المهمة توجد طريقتان:

1. التصريح عن مصفوفة ذي حجرات كثيرة، بحيث تكفي لتخزين القيم المتوقع إدخالها من المستخدم. وهذه الذاكرة تستَخدم التخصيص الساكن للذاكرة.
2. استخدام التخصيص الديناميكي للذاكرة وذلك باستخدام المؤشرات والتي سوف يتم شرحها مطولاً في هذا الفصل.

3. التخصيص الديناميكي للذاكرة Dynamic Memory Allocation

إن عملية الحجز الديناميكي للذاكرة لا تتم في أثناء ترجمة البرنامج وقبل تنفيذه، ولكنها تحدث وقت التنفيذ. عندما يحتاج البرنامج في أثناء عمله إلى

حجرات من الذاكرة، فإنه يُخصص بعضها ويستخدمها ضمن البرنامج. وحجرات الذاكرة هذه تقتطع مساحة من الذاكرة تدعى بالكومة (heap). الكومة هي مساحة من آخر الذاكرة يُحدَد موقعها تماماً بعد تحميل مقطع شيفرة التعليمات (code segment)، ومقطع البيانات (data segment)، ومقطع المكدس (stack segment) الخاصة بالبرنامج.

4. توضع طبقات البرنامج Program Layout

يظهر لنا الشكل أدناه رسماً تخطيطياً تقريبياً لتوضع طبقات البرنامج في أثناء التنفيذ، حيث يحتوي على المكونات الأساسية لحيز الذاكرة الذي يشغلها البرنامج في أثناء التنفيذ. وهذا التوضع لمحتويات الذاكرة قد يختلف من برنامج إلى آخر بسبب بعض الصفات الخاصة مثل: التراكب (loverlays)، وحجم وعدد المتغيرات والثوابت المستخدمة ضمن البرنامج.

تُعد الذاكرة العنصر الأهم في أثناء تنفيذ البرنامج، لذلك يُرتب نظام التشغيل ويُخزن المعلومات فيها ووفق الترتيب المذكور في الجدول التالي.

إن جزءاً من نظام التشغيل هنا سوف يُخزَن في الذاكرة، وسوف تُخزَن

أعلى الذاكرة
الكومة (مساحة غير مستخدمة من الذاكرة) تُستخدَم للتخصيص الديناميكي
مقطع خاص بالمكدس (stack segment) ويمثل متغيرات محلية ووسطاء
مقطع البيانات (data segment) ويمثل متغيرات عامة وثوابت
مقطع شيفرة التعليمات (code segment) ويمثل تعليمات البرنامج
مقطع الأغراض الخاصة و تستخدمه بعض البرامج
أسفل الذاكرة DOS

البرامج الأخرى المحملة في الذاكرة في أثناء تنفيذ البرنامج أيضاً في القسم نفسه، والذي يشغل الجزء السفلي من ذاكرة نظام التشغيل. مثلاً إذا كنا نعمل ضمن هذا البرنامج على برنامج محرر نصوص فإن هذا البرنامج سوف يُخزَن أيضاً في القسم نفسه من الذاكرة وهذا القسم يُدعى بقسم البرامج المتنوعة (miscellaneous).

1.4 مقاطع البرنامج Program Segment

بعد أن وُضِعَت البرامج في القسم السفلي من الذاكرة، يوضع مقطع شيفرة التعليمات الخاص بالبرنامج، ومن ثم يُتبَع هذا المقطع بمقطع البيانات، ومقطع المكدس. وهذه المقاطع ثابتة الطول ويتعلق طولها بالبرنامج نفسه؛ حيث تُحدِد كمية تعليمات البرنامج حجم مقطع شيفرة التعليمات، ويُحدِد حجم المتغيرات العامة والثوابت و حجم مقطع البيانات، وهكذا يتعلق حجم المكدس المُستخدم في البرنامج بالوسطاء الموجودة في البرنامج، والبيئات المحلية للاجرائيات التي يتوقف عملها مؤقتاً من أجل نقل التحكم إلى الاجرائيات الأخرى.

مقطع المكدس هذا له خصوصية في عمله، إذ تُخزَن فيه المعلومات آنياً حتى تُطلَب من جديد، لذلك يتغير الحجم المُستخدَم من هذا المقطع حسب عدد الاجراءات المُستدعاة فحجمه ينقص ويزداد تبعاً لذلك، حيث يزداد (grows) طول المكدس عكسياً أي من العنوان الأعلى إلى العنوان الأدنى لحجرات المكدس.

عندما ينتهي تنفيذ اجراء ما، يُفرَّغ جزء من المكدس الذي استُخدِم لتخزين البيئة المحلية لهذا الاجراء وتُعاد هذه المعلومات لاستخدامها من قبل البرنامج. وفي حال احتاج البرنامج حجماً للمكدس أكبر من الحجم المتاح له فإن البرنامج سوف يتوقف تنفيذه مع خطأ هو حدوث طفحان في المكدس (stack overflow). إن حدوث مثل هذا الخطأ يعتمد على حالة إختبار المكدس هل هي مُحَفِزة أم لا، والتي يتحكم بها توجيه المترجم {-S}. إذا كانت حالة إختبار المكدس فعَّالة، فإن البرنامج يختبر وجود مساحة في المكدس تكفي لتخزين مستلزمات الاجراء قبل

البدء بتنفيذه، وإلا فإن البرنامج سوف ينتهي مع خطأ. أما إذا كانت حالة إختبار المكدس هي عدم الإختبار، فإن البرنامج لن يختبر المكدس قبل تحميل البيئة الموضعية للإجراء. في هذه الحالة لن يفشل تنفيذ البرنامج وحده بل سيفشل معه النظام الداخلي للحاسوب وسوف يستدعي هذا إعادة إقلاع (reboot) للحاسوب من جديد. بشكل نظامي فإن حالة إختبار المكدس تكون مُحَفِزة ، فإذا أردنا تغيير حالة الإختبار تلك يمكننا إضافة أحد التوجيهين التاليين:

- اختبار المكدس (*$I+)
- إنهاء إختبار المكدس (*$I-*)

2.4 كومة البرنامج The Program Heap

الكومة بالتعريف هي المساحة من الذاكرة المُتبقية دون استخدام، والتي تقع بعد مقاطع البرنامج بعد تحميلها. إذا كان البرنامج المُحمّل صغيراً، ويملك القليل من البيانات العامة، ولا توجد برامج ضخمة مُحمّلة في الذاكرة، فإن الكومة ستكون كبيرة جداً، والعكس صحيح.

مثال:

```
Program HeapTest;
Begin
     Write('MEMAVAIL=',MEMAVAIL:7,'bytes;');
     Writeln('MAXAVAIL=',MAXAVAIL:7,'bytes');
     Readln;
End.
```

تُعيد الدالة Memavail قيمة صحيحة طويلة LongInt تُمثل عدد البايتات من الذاكرة التي تستطيع الكومة استخدامها، وهذه البايتات ليست متعاقبة بالضرورة. أما الدالة Maxavail فتُعيد قيمة صحيحة طويلة، لكنها تمثل حجماً أكبر حيز متعاقب تستطيع الكومة استخدامه وهذا الحجم قد يتغير حسب كمية الذاكرة

المتبقية والخاصة بالكومة، والتي تتغير إذا كان البرنامج يستخدم التخصيص الديناميكي للذاكرة.

تُدار عناصر الكومة في لغة الباسكال بوساطة مكتبة تدعى بمدير الكومة (heap manger) ومدير الكومة هذا يحتفظ بقائمة تتضمن حالة مساحات التخزين المخصصة للكومة.

5. المؤشرات Pointers

المؤشرات عبارة عن نوع بيانات يُشير إلى (Point to) مكان محدد ضمن الذاكرة، وعليه فإن المتغير من النوع مؤشر يحتوي على العنوان الذي خُزنت فيه قيمة ما. هذا العنوان يتعلق بمواقع التخزين في الذاكرة، والمسموح استخدامها ضمن البرنامج والتي تستطيع المؤشرات الوصول إليها. حجم مواضع التخزين هذه يتعلق بنوع البيانات التي سيتم تخزينها في هذه المواضع. إن هذا التخصيص للذاكرة يُدعى تخصيصاً ديناميكياً، لأن التخصيص يحدث عندما يحتاج البرنامج إلى ذلك وأثناء التنفيذ.

مثال:

يبين البرنامج التالي كيفية التصريح عن المؤشرات و الحصول على المعلومات التي تُشير إليها هذه المؤشرات.

```
Program PtrDemo;
Type
     IntPtr= ^integer;
     RealPtr= ^real;
Var
     IP:IntPtr;
     RP: RealPtr;
     ISize, RSize: integer;
Begin
```

```
  Write('MEMAVAIL=',MEMAVAIL:7,'bytes;');
  Writeln('MAXAVAIL=',MAXAVAIL:7,'bytes');
  New(RP);
  RSize:=sizeof(RP);
  Writeln('After allocating',RSize:7,'bytes':);
  Write('MEMAVAIL=',MEMAVAIL:7,'bytes;');
  Writeln('MAXAVAIL=',MAXAVAIL:1,'bytes');
  New(IP);
  ISize:=sizeof(IP);
  Writeln('After allocating',ISize:7,'bytes':);
  Write('MEMAVAIL=',MEMAVAIL:7,'bytes;');
  Writeln('MAXAVAIL=',MAXAVAIL:7,'bytes');
  Readln;
End.
```

يُطلق على المساحة من الذاكرة، والتي تحتوي على قيمة المتحول المؤشر عليه (أو تجاوزاً قيمة المؤشر) بغاية المؤشر (target)، أما محتويات المؤشر (المؤشر نفسه) فتمثل عنوان غاية المؤشر. الفقرات التالية سوف تقوم بشرح المعلومات السابقة بشيء من التفصيل.

1.5 تعريف نوع البيانات مؤشر Defining Pointer type

التركيب النحوي لتعريف المؤشرات على الشكل التالي:

<نوع البيانات الأساسي الخاص بغاية المؤشر>"^"="<مميز>

يُشير الرمز (^) الذي يسبق نوع البيانات الأساسي إلى أننا نُعرف مؤشراً. إن نوع البيانات الأساسي لهذا المؤشر يمكن أن يكون بسيطاً أو مركباً. في الحقيقة يمكن أن يكون بدوره مؤشراً أيضاً. في كل الأحوال نوع البيانات هذا يجب تحديده في أثناء تعريف نوع البيانات مؤشر حتى يعرف البرنامج مُسبقاً حجم الذاكرة الواجب تخصيصها لتخزين غاية المؤشر.

2.5 التصريح عن متغيرات من النوع مؤشر Declaring Pointer Variables

بعد أن نُحدد نوع البيانات الأساسي للمؤشر، يُمكننا التصريح عن متغيرات تنتمي لنوع البيانات مؤشر هذا. التركيب النحوي لهذا التصريح، و الذي يُطابق مثيله بالنسبة لأنواع البيانات الأخرى، يُعطى بالصيغة التالية:

<نوع البيانات المؤشر>":"<اسم متحول>

يتم حجز حيز لتخزين متغيرات المؤشرات و يُخصَص بشكل ساكن (static). أما حيز التخزين لغايات هذه المؤشرات فسوف تُخصَص بشكل ديناميكي (dynamic).

3.5 تخصيص مساحات تخزين ديناميكية Allocating Dynamic Storage

يُخصَص استدعاء الإجراء مسبق التعريف PtrDemo1 ضمن البرنامج السابق مساحة تخزين في أثناء تنفيذ البرنامج. وذلك من خلال العبارة التالية:

New (RP);

يجب أن يكون المتغير الوسيطي المُمرَر إلى NEW متحول من النوع مؤشر. ونوع البيانات الأساسي لغاية المؤشر سوف يُحدِد حجم ومساحة الذاكرة الواجب تخصيصها. كل مساحة مخصصة كما ذكرت يجب أن توضع في الكومة وفق الخطوات التالية:

1. يطلب البرنامج من مدير الكومة (heap managr) مساحة متعاقبة من الذاكرة، فعلاً بطول ستة بايتات في مثالنا لتخزين عدد حقيقي.
2. يبحث مدير الكومة عن هذه المساحة، وبعد أن يجدها يُعطيها للبرنامج، ويحذف هذه المساحة من قائمة مواضع الذاكرة المُخصَصة للكومة، والتي يُنشئها مدير الكومة.

3. يُخزَن عنوان هذه المساحة من الذاكرة في المتغير RP، أي أن RP سوف يملك قيمة عنوان عند تنفيذ الإجراء NEW.

4. يُقسم مدير الكومة، الكومة إلى مساحات أطوالها من مضاعفات العدد 8.

إذا لم يستطع مدير الكومة تخصيص مساحة الذاكرة المطلوبة، فإن البرنامج سوف يتوقف مع خطأ في التنفيذ. من أجل تجنب مثل هذه المشكلة، نختبر كمية الذاكرة المُتبقية في الكومة قبل تخصيص أي كمية من الذاكرة.

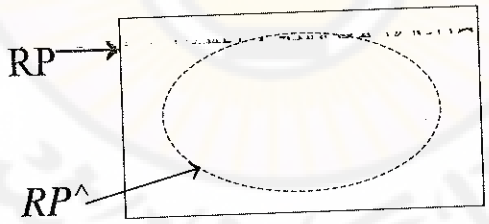
حتى نستطيع تخصيص مساحة من الذاكرة لمتحول ديناميكي جديد يجب أن تكون المساحة المتبقية في الكومة أكبر أو تساوي المساحة اللازم تخصيصها لهذا المتحول الجديد.

4.5 الوصول إلى غاية المؤشر Accessing Target Variable

فيمايلي نعرض مثالاً يبين كيفية الإشارة إلى متحول غاية المؤشر:

```
RSize := Sizeof (RP^);
```

يُمثل المتغير $RP^$ متحول غاية المؤشر (target variable) والذي عنوانه مُخزَن في المؤشر RP. بعبارة أدق يمكننا القول بأن المتغير $RP^$ يعني المتغير الذي يشير إليه المؤشر RP كما في الشكل التالي:



كما لاحظنا فإن متحول غاية المؤشر لا يملك اسماً مستقلاً ولكن يُشار إليه بوساطة اسم مؤشره الخاص. تُدعى عملية الإشارة إلى هذا المتغير بالوصول غير المباشرة (indirect reference) على عكس الوصول المباشرة (direct reference)

للمتغيرات، والتي تتم بتحديد اسم المتغير مباشرة. إن عبارة الإلحاق السابقة تقوم بما يلي:

1. تحديد قيمة كمية الذاكرة المُخصَصة للمتحول RP^.

2. إلحاق هذه القيمة بالمتغير RSize.

5.5 وضع قيم بمتحول غاية المؤشر Assigning Values to Target Variable

يبين البرنامج التالي كيفية إلحاق قيم بمتحول غاية المؤشر:

```
Program PtrDemo2;
Type
     IntPtr= ^integer;
     RealPtr= ^real;
Var
     IP:IntPtr;
     RP:RealPtr;
Begin
  Randomize;
  Writeln('MEMAVAIL=',MEMAVAIL : 10,'bytes');
  New(RP);
  New(IP) ;
  Repeat
   RP^ :=random;
   IP^ :=random(500) ;
   Writeln('RP^ =', RP^ : 10:5,'; IP^=', IP^: 5);
Until((RP^<0.5) and (IP^<250));
Writeln('MEMAVAIL=', MEMAVAIL : 10, ' bytes');
Readln;
End.
```

كما يبين المقطع السابق، ألحقت قيمة حقيقية بالمتغير $RP^{\wedge}$، وقيمة صحيحة بالمتغير $IP^{\wedge}$. مع أن هاتين القيمتين تنتميان إلى نوعي بيانات مختلفين، إلا أن كمية الذاكرة المُخصَصة لهما والمُقتطَعة من الكومة متساوية، لأن مدير الكومة يقسم هذه الذاكرة إلى أجزاء من مضاعفات العدد 8.

6.5 التمييز بين المؤشر وغاية المؤشر Distinction Between Pointer and Target

يُمكننا إسناد عنوان لمتحول المؤشر، أي لا يمكن إسناد قيمة للمتغير RP ضمن البرنامج السابق، وبالتالي العبارة التالية غير صحيحة:

```
RP :=23.5;
```

ذلك لأن المتغير RP لا ينتمي إلى نوع البيانات عدد حقيقي، وإنما هو مؤشر يشير إلى عدد حقيقي. كذلك الأمر بالنسبة لمتحول غاية المؤشر، والذي يحتوي دائماً على قيمة تنتمي لنوع بيانات ما، فلا يُمكننا تخصيص حجر ذاكرة له باستخدام الإجراء New، أي العبارة التالية أيضاً غير صحيحة:

```
New (RP^);
```

ذلك لأن الإجراء New يأخذ متحولاً وسيطياً عبارة عن مؤشر، في حين أن $RP^{\wedge}$ متحول من النوع عدد حقيقي. إذاً عندما نُعرِف نوع بيانات مؤشر ونصرح عن متغير ينتمي إليه، فإن هذا المتغير يُدعى بمتحول المؤشر، ويحتوي هذا المؤشر دائماً على عنوان موضع في الذاكرة، يُحدِد حجمه نوع البيانات الأساسي للمؤشر. يسمى هذا الموضع غاية المؤشر (pointer target) (أي القيمة الذي يشير إليه المؤشر). فغاية المؤشر عبارة عن تتابع حجرات من الذاكرة توضع فيها القيمة المُراد تخزينها.

بشكل مختصر لتخزين غاية المؤشر يتم إلحاقها بمتحول المؤشر مباشرة، أما قيمة المؤشر تُلحق بمتحول غاية المؤشر، أي بمتحول المؤشر متبوعاً بالرمز (^).

7.5 إسناد المؤشرات Pointer Assignment

يُمكننا إسناد متحول مؤشر إلى متحول مؤشر آخر إذا كان كلا المتحولين ينتمي إلى نفس نوع البيانات أي يجب أن يشير المؤشران إلى متحولين من نفس النوع.

مثال:

يبين البرنامج التالي كيفية إلحاق محتويات متحول مؤشر بمتحول مؤشر آخر.

```
Program PtrDemo3;
Type
     IntPtr= ^integer;
     RealPtr= ^real;
Var
     IP,IP2:IntPtr;
     RP,RP2:RealPtr;
Begin
  Writeln('MEMAVAIL =',MEMAVAIL : 10, ' bytes');
  New(RP);
  RP^ :=23.5;
  RP2:=RP;
  Writeln('RP^=',RP^:10:5,'; RP2^=',RP2^:10 : 5);
  New(IP);
  IP^ :=23;
  Writeln('RP^=',RP^: 10:5, '; IP^ = ', IP^ : 5);
  Writeln('MEMAVAIL=',MEMAVAIL : 10, ' bytes');
  Readln;
End.
```

إن متغيرات المؤشرات تُخزَن ضمن مقطع البيانات أو مقطع المكدس، ويُحدِد ذلك كونها متغيرات عامة (global variables) أو متغيرات محلية (local variables). في حين أن متغيرات غايات المؤشرات تُخزَن في الكومة (heap).

عندما ألحقنا قيمة متحول المؤشر RP بمتحول المؤشر RP2، أصبح لدينا مؤشران يشيران إلى موضع واحد في الذاكرة. حيث تُعرف هذه العملية بالتسمية المضاعفة أو البديلة (aliasing)، وهي تسبب بعض المشاكل أحياناً، إن لم نكن حريصين بالقدر الكافي. فإذا سُمي موضع ما في الذاكرة بعدة أسماء، فإن البرنامج سوف يستطيع تغيير القيمة المخزنة في هذا الموضع بعدة طرق، وهذا التنوع سيُشكّل صعوبة في تتبع خطوات البرنامج، مما يؤدي إلى صعوبة في اكتشاف مثل هذا النوع من الأخطاء.

مثال:

أدخل قيماً صغيرة بين (0 و1):

```
Program AliasDemo;
Type
     RealPtr= ^real;
Var
     RP,RP2: RealPtr;
Procedure GetReal(Message:string;Var Value:real);
  Begin
     Write(Message,' ') ;
     Readln(Value);
  End;
Procedure Transform(Var ThePtr:RealPtr);
      Const MultFactor=1000.0;
   Begin
     If((ThePtr^>= 0)and(ThePtr^ <= 1.0)) then
```

```
        ThePtr^ := MultFactor * ThePtr^;
        ThePtr^ := 1 / ThePtr^;
    End; (*Transform*)
Begin
    New(RP);
    Repeat
        GerReal('Value?(< 0 to stop)', RP^);
        RP2:=RP;
        Writeln('RP^ = ', RP^ : 10 : 5);
        Transform(RP2) ;
        Writeln('After Transform: ');
        Writeln('Rp^ = ', RP^ : 10 : 5);
    Until(RP^<0);
    Readln;
End.
```

يقوم البرنامج السابق بتحويل القيم الكبيرة إلى قيم صغيرة، وبالعكس. عندما يُدخل المستخدم قيمة ما، فإن البرنامج سوف يُخزنها في المتغير *RP^* الذي يُستخدَم فقط عند استدعاء الإجراء GetReal وضمن استدعائين للإجراء Writeln.

بعد أن ألحق البرنامج قيمة بالمتغير RP من خلال استدعاء الإجراء New، ألحق قيمة المتغير RP بالمتغير RP2، أي جعل المؤشر الثاني يُشير إلى نفس موضع الذاكرة الذي يُشير إليه المؤشر الأول، أو بمعنى آخر جَعَلَ كلا المتغيرين RP و RP2 يحتوي على العنوان نفسه. أما استدعاء الإجراء Transform فهو يتعامل مع المتغير RP2، حيث يعمل على تغيير قيمة المتغير The Ptr ^ ، والذي يساوي تماماً المتغير RP2 ^ الذي تم تمريره كمتحول وسيطي إلى الإجراء. لكن بما أن RP2 و RP يُشيران إلى نفس الموضع من الذاكرة، فإن تغيير غاية المؤشر RP2 هي فعلياً تغيير لقيمة غاية المؤشر RP، وذلك لأن *RP^* و *RP2^* هما اسمان لمتحول واحد.

الوسيط الوحيد في الإجراء Transform سُبق بالكلمة المحجوزة Var، لكن الإجراء فعلياً لا يُغير قيمة المتغير RP2، وإنما يُغير قيمة المتغير RP2^، لذلك فإن حذف الكلمة المحجوزة Var من أمام الوسيط ThePtr لا يؤثر في تنفيذ البرنامج.

ومن أجل فهم أكبر للمؤشرات لنفترض أن عناوين حُجر الذاكرة هي أعداد من النوع الصحيح، عندئذ سوف تُخزن في متحول المؤشر قيمة تتألف من بايتين: الأول منها يحتوي على عنوان المقطع (segment) المُخزَن فيه غاية المؤشر، والثاني يُمثل إزاحة (offset) غاية المؤشر من بداية المقطع. فإذا كان لدينا العنوان (5000) مُخزَن ضمن المتغير RP2 (أي في RP أيضاً)، واستدعينا الإجراء Transform، وتم تمرير المتغير RP2 بقيمته، فإن القيمة المحتواة في RP2 سوف تُنسخ وتُمرَر إلى الإجراء، وسوف يُفَسِر الإجراء هذه القيمة على أنها عنوان في الذاكرة، وسيفهم المتغير ThePtr^ على أنه القيمة المخزنة في العنوان (5000). فإذا تغيرت هذه القيمة فإن هذا التغيير سوف يُحفَظ ويبقى حتى بعد انتهاء تنفيذ الإجراء، أما القيمة المُخزَنة في المتغير RP2 فلا يمكن الوصول إليها على الرغم من ارتباط هذه القيمة بغاية المؤشر. لذلك إذا قمنا بتمرير مؤشر بقيمته عندئذ يُمكننا تغيير محتويات غاية هذا المؤشر لكن لا نستطيع تغيير قيمة المؤشر نفسه.

8.5 إسناد العناوين للمؤشرات Assigning Address to Pointers

يتم تهيئة المؤشرات بعد استدعاء الإجراء New، أو بوساطة إسناد متحول مؤشر إلى متحول مؤشر آخر من نفس النوع. لكن لغة الباسكال مكَّنت من إسناد عنوان متحول معين إلى متحول مؤشر.

1. معامل العنوان (@) The Address Operator

تزودنا لغة الباسكال بالمعامل @ والدالة Addr من أجل التعامل مع العناوين. حيث يقوم معامل العنوان (address operator) بتوليد موقع متحوله

الوسيطي، وهذا المتغير الوسيطي يُمكن أن يكون متحولاً أو وسيطاً أو اجراءً. إذ يأخذ المعامل متحولاً وسيطياً واحداً يملك موقعاً في الذاكرة، ويعيد عنوان هذا المتغير الوسيطي. وهذا المعامل هو معامل أحادي (unary operator) ويملك الأولية الأعلى.

2. الدالة Addr The Addr Function

تأخذ الدالة **Addr** متحول، أو اسم إجراء أو دالة، أو بشكل عام اسم موضع في الذاكرة، وتُعيد موقع هذا المتغير في الذاكرة. بما أن هذه الدالة تُعيد عنواناً في الذاكرة لذلك يمكن أن يُقال أنها تُعيد مؤشراً.

يبين البرنامج التالي كيفية استخدام المعامل @ والدالة Addr للحصول على عناوين في الذاكرة.

```
Program AddressOpDemo;
Type
     RealPtr= ^real;
Var
     RP1,RP2:RealPtr;
     RVal1,RVal2:Real;
Begin
 RVal1:=100.0;
 RVal2:=200.0;
 RP1:=@RVal1;
 RP2:=ADDR ( RVal1) ;
 If(RP1=RP2) then
   Writeln('Pointers contain the same address')
Else
Writeln('Pointers contain different addresses');
Writeln('RP1^=',RP1^:10:5 ;'RP2^=',RP2^:10:5);
Readln;
```

End.

تجدر الاشارة إلى أن المعامل @ والدالة Addr يقومان بالعمل نفسه، لذلك لا فرق في استخدام أحدهما مكان الآخر.

6. استخدام الإجراء Getmem لتخصيص مساحات تخزين ديناميكية Using Getmem to Allocate Dynamic Storage

بالإضافة إلى الإجراء New، يوجد في لغة الباسكال إجراء آخر لتخصيص مساحات تخزين ديناميكية، هذا الإجراء هو Getmem، وهو يأخذ متحولين وسيطين هما:

1. متحول مؤشر.
2. متحول من النوع Word يُمثل عدد البايتات الواجب تخصيصها من الإجراء ديناميكياً.

تُخصِص العبارة التالية ستة بايتات من الذاكرة وتُلحِق هذه البايتات الستة بالمتغير PtrVar.

```
Getmem ( pTRvAR, sizeof (real));
```

يُخصِص مدير الكومة (heap manager) ثمانية بايتات كما ذكرت لأنها أقل وحدة تخصيص يسمح مدير الكومة لبرنامج باستخدامها.

7. أخطار المؤشرات Pointer Pitfalls

تُعتبر المؤشرات أدوات برمجية ممتازة. نبين في الفقرات التالية بعض المشاكل التي تواجهنا في أثناء استخدام المؤشرات.

1.7 مشكلة إعادة استخدام مؤشر Reusing a Pointer

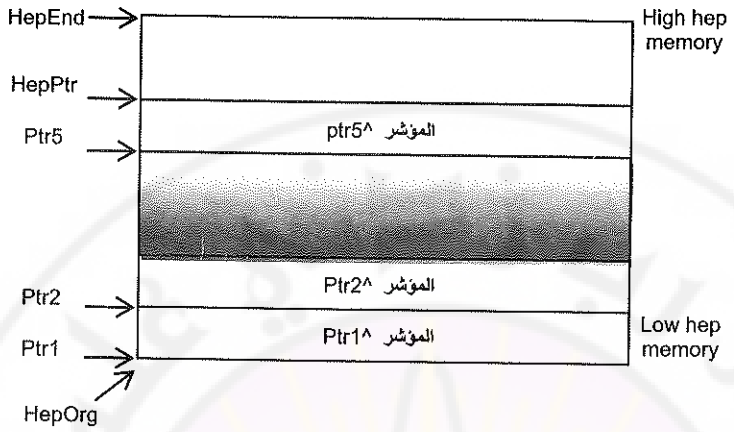
تأخذ المؤشرات غاياتها من كومة البرنامج من خلال التخصيص الديناميكي لذاكرة الكومة (heap).

يبين البرنامج التالي ماذا يحدث إذا استدعينا الإجراء New عدة مرات مع المؤشر نفسه في كل مرة:

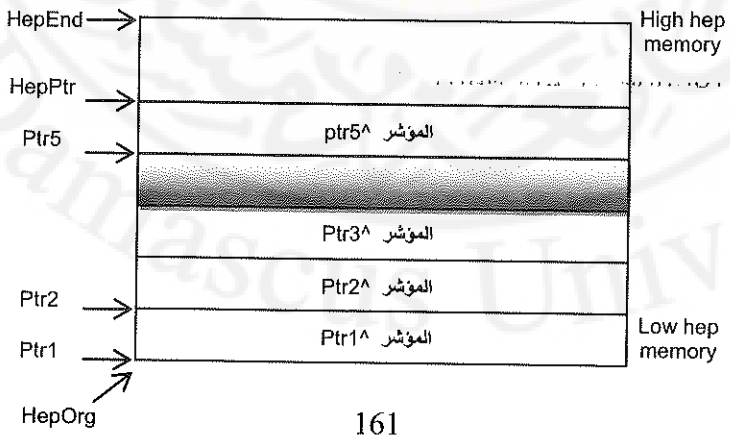
```
Program ReUseDemo;
Const
     MaxTimes=20;
Type
     RealPtr= ^real;
Var
     RP:RealPtr;
     Count,FirstMem,LastMem:integer;
Procedure GetVal(Var ThePtr:RealPtr;Count
     :integer);
   Begin
     New(ThePtr);
     ThePtr^ :=random;
     Writeln(Count:2,':ThePtr^=',ThePtr^:10 : 5);
   End;
Begin
 Randomize;
 FirstMem :=MEMAVAIL;
 Writeln('MEMAVAIL = ', MEMAVAIL : 10, ' bytes');
 For Count := 1 TO MaxTimes do
   GetVal(RP,Count);
 Writeln('RP^      = ', RP^ : 10 : 5);
 LastMem := MEMAVAIL;
 Writeln('MEMAVAIL = ', MEMAVAIL : 10, ' bytes');
 Writeln('The heap was decrease',FirstMen
  LastMen,'byted');
 Readln;
End.
```

يقوم البرنامج السابق باستدعاء الإجراء GetVal عدة مرات، وفي كل مرة يُخصص الإجراء مساحة لتخزين عدد حقيقي، ويجعل المؤشر ThePtr يؤشر

من أجل استدعاء آخر للإجراء Dispose مع المؤشر Ptr4، عندئذ ستصبح الكومة كما يلي:



نلاحظ من خلال الشكلين السابقين أن عملية إنهاء التخصيص تلك قد تركت فجوة فارغة (hole) ضمن المساحة المُستخدمة من الكومة، وعند تخصيص مؤشر جديد يجري اختبار هذه المساحات من حيث كونها تكفي لتخزين غاية المؤشر الجديد. إذا كانت تكفي عندئذ يتم تخصيص هذا المؤشر في هذه المساحة، وإلا فإنها تُترك فارغة ويُخصَص هذا المؤشر في مساحة فارغة أخرى. فإذا أعدنا مثلاً تخصيص المؤشر Ptr3 مرة أخرى فسوف يتوضع في ذاكرة الكومة على الشكل التالي (إن لم تُشغل هذه المساحة بغاية مؤشر آخر):



يبين البرنامج التالي الإجراء Dispose الذي يمكننا من انهاء تخصيص المؤشرات.

```
Program ReturnDemo;
Const
     MaxTimes=20;
Type
     RealPtr= ^real;
Var
     RP:RealPtr;Count,FirstMem,LastMem:integer;
Procedure GetVal(Var ThePtr:Realptr;
      Count:integer) ;
   Begin
       Dispose(ThePtr);
       New(ThePtr);
       ThePtr^ :=random;
       Writeln(Count:2,';ThePtr =',ThePtr^:10:5);
   End;
Begin
     Randomize;
     FirstMem:=MEMAVIL;
     Writeln('MEilAVAIL=',MEMAVAlL:10,'bytes');
     New(RP^);
     For Count := 1 to MaxTimes do
       GetVal(RP,Count);
     Writeln('RP^ =', RP^ : 10 : 5);
     LastMem:=MEMAVIL;
     Writeln('MEMAVAlL=',MEMAVAlL:10,'bytes');
     Writeln('The heap was decrease', FirstMen-
          LastMen, 'bytes');
     Readln;
```

```
End.
```

يُسبب الإجراء Dispose خطأ في تنفيذ البرنامج إذ مُرر إليه متحول وسيطي من النوع مؤشر لم تُخصَص له أي مساحة من الذاكرة بعد. لذلك استدعاء الإجراء New يُجنب مثل هذه الأخطاء.

8. استخدام الإجراء Freemem لإعادة مساحات الذاكرة المخصصة
Using Freemem to Return Allocated Storage

يُستخدم الإجراء Freemem لإعادة حجرات من الذاكرة خُصِصَت بوساطة الإجراء Getmem، حيث يأخذ هذا الإجراء متحولين وسيطين هما:

1. متحول مؤشر.
2. متحول من النوع Word يُمثل عدد البايتات التي ستُعاد إلى الكومة.

يبين البرنامج التالي كيفية استخدام الإجراء Freemem.

```
Program FreeMeniDemo;
Type
     RealArray=array[1..500] of real;
     RAPtr=^RealArray; RealPtr=^real;
     LongIntPtr=^longint;
Var
     RA1:RAPtr; RP1:RealPtr; LP1:LongIntPtr;
Begin
   Write('MEMAVAIL=',MEMAVAIL:10,'bytes');
   Writeln('MEMAVAIL=',MEMAVAIL:10,'Bytes');
   GetMem(RA1,sizeof( RealArray)) ;
   GetMem(RP1,sizeof( REAL)) ;
   GetMem(LP1,sizeof( LONGINT));
   Writeln('After allocating RA1^ RP1^and LP1^');
   Write('MEMAVAIL = ', MEMAVAIL 10,'bytes');
   Writeln('————————-');
```

```
    Writeln('————————');
    Freemem(RA1,sizeof(RealArray)) ;
    Freemem(RP1,sizeof(REAL));
    Freemem (LP1,sizeof(LONGINT)) ;
    Writeln('After freeing RA1^, RP1^, and LP1^');
    Write('MEMAVAIL=', MEMAVAIL : 10, ' bytes');
    Writeln('; MAXAVAIL=',MAXAVAIL: 10, 'bytes');
    Readln;
End.
```

إن الإجراء Freemem يُنهي تخصيص حجرات من الذاكرة خُصِصَت بوساطة الإجراء Getmem. عدد هذه الحجرات يساوي تماماً عدد الحجرات المُخصَصة بوساطة الإجراء Getmem. هذا الإجراء يترك فجوات فارغة (holes) بالطريقة نفسها التي يتركها الإجراء Dispose وتُعالج هذه الفراغات بنفس الطريقة التي عولِجَت بها سابقاً.

9. إنهاء تخصيص مجموعة مؤشرات من الكومة Deallocating Groups on the Heap

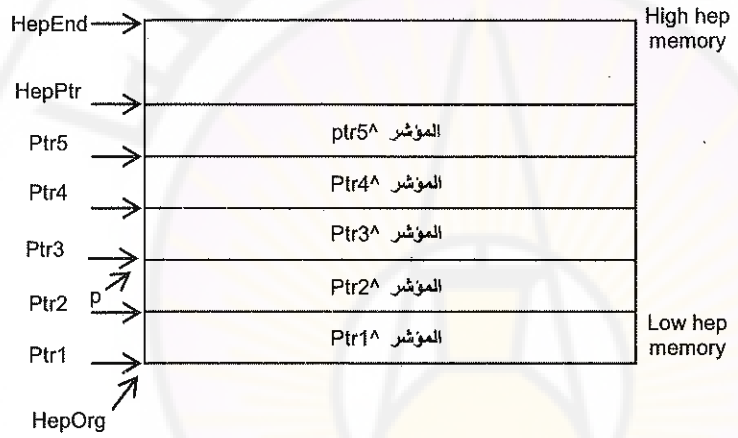
لقد استخدمنا حتى الآن الإجرائين Dispose و Freemem من أجل إنهاء تخصيص حجر من الذاكرة قد خُصِصَت بوساطة الإجراءين New و Getmem. الإجراءان Dispose و Freemem يتركان فجوات ضمن ذاكرة الكومة عند إنهاء تخصيص مؤشر ما. في الواقع يمكننا استخدام الإجرائين Mark و Release لإنهاء تخصيص مجموعة من المؤشرات دفعة واحدة، وذلك بإلحاق القيمة الحالية لمؤشر قمة الكومة (Heap Pointer) بالمتغير الوسيطي المُمرَر للإجراء Mark ومن ثم استدعاء الإجراء Release (بعد عدد من الخطوات) مع نفس المتغير الوسيطي السابق، عندئذ سيقوم هذا الإجراء بإنهاء تخصيص كل المؤشرات التي خُصِصَت بعد استدعاء الإجراء Mark.

مثال:

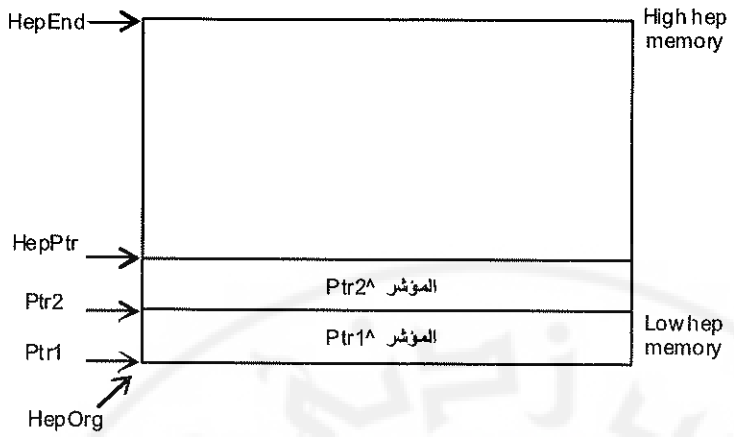
ليكن لدينا تسلسل استدعاءات الإجراءات التالية:

```
New(Ptr1);
New(Ptr2);
New(Ptr3);
New(Ptr4);
New(Ptr5);
```

عندئذ ستُخصص غايات المؤشر كما في الشكل التالي:



نلاحظ أن شكل الكومة هنا يُشبه تماماً شكلها السابق بدون استدعاء الإجراء Mark، ولكن الفرق هنا أن المؤشر P قد أصبح يُشير إلى قمة الكومة بعد أول استدعاءين. إذا استدعينا الآن الإجراء Release مع المتغير الوسيطي P فسيجري إنهاء تخصيص جميع المؤشرات التي تم تخصيصها بعد استدعاء الإجراء Mark، أي أن المؤشرات Ptr3 و Ptr4 و Ptr5 تُصبح كما هو موضح في الشكل التالي:



نلاحظ من هذا التمثيل لذاكرة الكومة أن الإجراء Release لا يترك فجوات فارغة ضمن الكومة.

أخيراً استدعاء الإجراء Release مع المؤشر HeapOrg سوف يُنهي تخصيص جميع المؤشرات الموجودة في الكومة، لأن هذا المؤشر مُسبق التعريف وهو يُشير دائماً إلى أسفل الكومة، أي يُشير إلى أول بايت في الكومة. لذلك فإن استدعاء الإجراء Release مع هذا المؤشر سوف يحذف كل المؤشرات من الذاكرة.

10. القيمة لا شيء مسبقة التعريف الخاصة بالمؤشر The predefined NIL Value for Pointer

هناك طريقة أخرى تُجنبنا الخطأ الذي سيحدث إذا استدعينا الإجراء Dispose مع متحول وسيطي مؤشر غير معروف، إذ يُمكننا استخدام عنوان خاص هو اللاشيء NIL. حيث إن NIL هي قيمة مُسبقة التعريف تُستخدَم مع المؤشرات، وتُمثل موضعاً خاصاً في الذاكرة يُلحَق بالمؤشر لإعطائه قيمة داخلية. المؤشر الذي يملك القيمة NIL لا يملك غاية (target)، ولكن تمت تهيئة هذا المؤشر وأخذ قيمة ما.

البرنامج التالي لم يستدعِ الإجراء New ضمن البرنامج الرئيسي لكنه يُسنِد للمؤشر RP القيمة NIL.

```
Program NILDemo;
```

```
Const
     MaxTimes=20;
Type
     RealPtr= ^real;
Var
     RP:RealPtr; Count:integer;
Procedure GetVal(Var ThePtr:RealPtr;
       Count:integer);
  Begin
     If(ThePtr<>NIL) then
       Dispose(ThePtr) ;
     New(ThePtr) ;
     ThePtr^:= random;
     Writeln(Count:2,':ThePtr=',ThePtr^:10:5);
  End;
Begin
     Randomize;
     Writeln('MEMAVAIL=',MEMAVAIL:10,'bytes');
     RP:=NIL;
     For Count := 1 to MaxTimes do
       GetVal(RP,Count);
     Writeln('RP^ = ',RP^:10:5);
     Writeln('MEMAVAIL =',MEMAVAIL : 10,'bytes');
     Readln;
End.
```

يختبر الإجراء GetVal في البرنامج السابق متحول المؤشر ThePtr، فإذا أُعطي هذا المتغير القيمة NIL، فإن الإجراء لن يستدعي الإجراء Dispose.

11. بنى التأشير الذاتي Self- Refrenntial Structures

تستَخدم البرامج عادةً عدداً من العناصر التي تنتمي لنفس نوع البيانات، وعدد العناصر تلك يتم تحديده عادةً في أثناء ترجمة (compiling) البرنامج، لكن قد نحتاج عدد من العناصر عددها لا نستطيع تحديده في أثناء ترجمة البرنامج وإنما عند تنفيذه. مثلاً عند تحليل أحرف الكلمات المقروءة من ملف ما، إذا كنا لا نعلم مُسبقاً عدد الكلمات التي سوف تُقرأ من الملف. نستَخدم التخصيص الديناميكي للذاكرة للقيام بالعمل المطلوب.

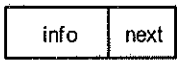
عنصر البيانات الأساسي المُستخدَم في مثل هذه البرامج هو نوع معين من السجلات. السجل عادةً لا يمكن أن يحتوي حقلاً ينتمي لنوع البيانات سجل نفسه، لكن إذا استخدمنا هذا السجل لتمثيل غاية المؤشر، فيمكن عندئذ أن ينتمي أحد حقول هذا السجل إلى نوع البيانات مؤشر، والذي نحن بصدد تعريفه.

مثال:

```
Type
     LNodePtr=^LNode;
     LNode=Record
       Info:real;
       Next: ^LNodePtr;
     End;
Var Root:LNodePtr;
```

تُصرّح التعريفات السابقة عن نوع بيانات سجل اسمه LNode، يملك حقلين أحدهما يُدعى Info، وينتمي إلى نوع البيانات عدد حقيقي، والآخر يُدعى Next، وهو عبارة عن مؤشر يُشير إلى عنصر من نفس نوع البيانات، أي يُشير إلى سجل آخر من النوع LNode.

تُعتبر البنية LNode بنية خاصة لأنها تحتوي على معلومات عددية ومؤشر. والشكل التالي يوضح طريقة تمثيل حقول هذه البنية.



1.11 ربط بنى التأشير الذاتي Linking Self-Referential Structures

بما أن سجل من النوع LNode يُمكن أن يُشير إلى سجل LNode آخر، عندئذ يُمكننا بناء سلسلة طويلة من هذه البنى، وذلك بربط هذه البنى بعضها مع بعض بوساطة الحقل Next. لذلك يُمكننا تعريف ما يُسمى بالقائمة المترابطة (linked list)، و هي سلسلة من العناصر رُبط كل عنصر فيها بالعنصر الذي يليه وهكذا..[5]. ندعو كل عنصر من عناصر السلسلة بالعقدة (node) وأُطلق هذا الاسم تجاوزاً، حيث يُشير إلى اتحاد عدد من أنواع بيانات مختلفة. في مثالنا السابق كان السجل LNode يشكل عقدة من القائمة المترابطة.

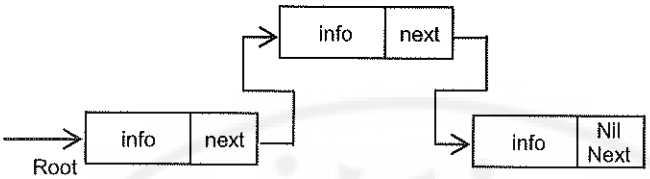
يُشار إلى العقدة الأولى في القائمة المترابطة بمؤشر يُشير إلى LNode، وهذا المؤشر هو مؤشر مُستقل، يجب أن يكون المؤشر حقلاً في سجل ما، مثلاً حقل في السجل LNode. حيث يُصرَّح عن متحول الجذر (root variable) في مقدمة القائمة المترابطة، أما الحقل Next الموجود ضمن أول عقدة LNode فسوف يُشير بدوره إلى عقدة تالية وهكذا...

يُمكن أن تحتوي القائمة على عدد من العقد تكفي لاستخدامات البرنامج، أو بقدر ما تبقّى في الكومة (heap) من ذاكرة. أما الحقل Next في العقدة الأخيرة من القائمة المترابطة فيجب أن يُشير إلى اللاشيء (أي NIL)، حتى يُخبرنا بانتهاء عُقد القائمة المترابطة.

---

[5] هناك نوع آخر من هذه السلاسل يُدعى بالأشجار (trees).

الشكل التالي يبين قائمة مترابطة تحتوي ثلاثة عناصر.



تجدر الإشارة هنا إلى أن مؤشر الجذر Root ليس عقدة، مثل LNode، وإنما هو مؤشر عادي يُشير إلى العقدة LNode.

2.11 بناء القوائم المترابطة Building Linked Lists

القائمة المترابطة عبارة عن بنية بيانات ديناميكية، أي أن حجمها يتحدد فقط عند تنفيذ البرنامج. وهذه البنى تختلف عن المصفوفات التي يتم تحديد حجمها ونوعها مسبقاً. يُمكننا بناء القائمة المترابطة عقدة عقدة. فمن أجل إضافة عقدة إلى قائمة مترابطة موجودة نقوم بما يلي:

1. ننشئ عقدة و ذلك بوساطة استدعاء الإجراء New مع متحول وسيطي مناسب.
2. نُخَزن المعلومات المطلوب تخزينها في العقدة.
3. نربط العقدة الجديدة بالقائمة المترابطة القديمة عبر بعض التعديلات على المؤشرات لجعلها تُشير إلى العقدة الجديدة Next، وجعل الحقل في العقدة الجديدة يُشير إلى عقدة معينة في القائمة المترابطة.

3.11 إضافة العقد إلى بداية القائمة المترابطة Adding at the Front of a Linked List

يُمكننا بناء قائمة مترابطة وذلك بإضافة كل عقدة جديدة إلى بداية هذه القائمة. يبين البرنامج التالي طريقة القيام بهذه العملية.

```
Program LListDemo;
Type
```

```
      LNodePtr= ^LNode;
      LNode=Record
        Info:real;
        Next:LNodePtr;
      End;
Var
      Root,Temp:LNodePtr;
Procedure GetReal(Message:string;Var Value:real);
  Begin
     Write(Message,' ');
     Readln(Value);
  End;
Procedure WriteList(Var TheNode:LNodePtr);
  Begin
     Writeln(TheNode^.Info : 10 : 5);
     If(TheNode^.Next <> NIL) then
       WriteList(TheNode^.Next);
  End;
Procedure InitLNode(Var TheNode:LNodePtr);
  Begin
     New(TheNode) ;
     TheNode^.Info = 0.0;
     TheNode^.Next = NIL;
  End;
Procedure LLAtFront(ToAdd:LNodePtr; Var
    WheretoAdd:LNodePtr);
  Begin
     ToAdd^.Next:=WhereToAdd;
     WhereToAdd:=ToAdd;
  End;
```

```
Begin (* Main *)
      Root := NIL;
      Temp := NIL;
      Repeat
         InitLNode(Temp);
         GetReal('Value(< 0 to stop)'.Temp^.Info);
         LLAtFront(Temp,Root) ;
      Until(Temp^.Info < 0.0);
      WriteList(Root);
      Readln;
End.
```

أدخل القيم التالية على الترتيب (6) و (3) و (23) و (4) و (5) و (3) و (-1) ولاحظ الخرج.

يُعرِف البرنامج السابق المتغيرين Root و Temp على أنهما مؤشران يُشيران إلى السجل LNode. حيث يعمل المؤشر Root كدليل داعم للقائمة المترابطة التي سوف تُبنى، لذلك يشير المؤشر Root دائماً إلى أول عقدة في هذه القائمة، حتى ولو تغيرت هذه العقدة في أثناء تنفيذ البرنامج، ونحتاج إلى المؤشر Root حتى نصل إلى أي عنصر في القائمة المترابطة.

يُشير المؤشر Temp إلى العقدة الجديدة في أثناء تهيئتها، وقبل إضافتها إلى القائمة المترابطة، إذ يُشير Temp إلى موضع التخزين المُخصَص للعقدة الجديدة، ومن ثم تُضاف هذه العقدة إلى القائمة المترابطة. وعندئذ يُشير Temp إلى عقدة جديدة ويتخلى عن تأشيره للعقدة السابقة، وهكذا.

4.11 الإجراء InitLNode Procedure InitLNode

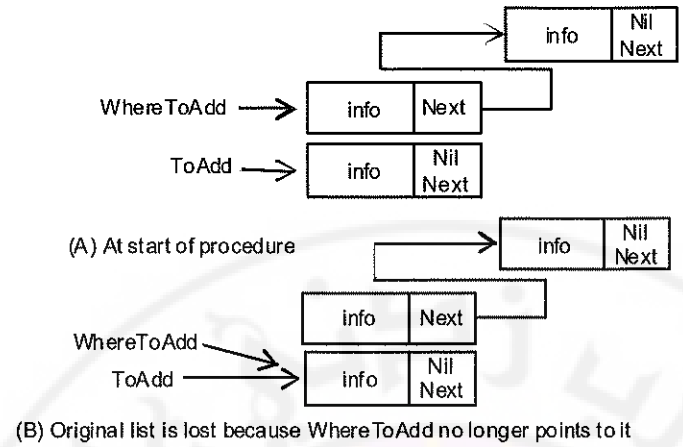
يُخصِص الإجراء InitLNode مساحات الذاكرة المطلوبة من أجل عقدة جديدة، ويُعيد هذا الإجراء، مع مساحات الذاكرة تلك، مؤشر يُشير إلى هذه المساحات. هذا الإجراء يستخدم الإجراء New من أجل تخصيص مساحات

الذاكرة، لذلك ينطبق عليه ما ينطبق على الإجراء New. إذا لم تبق مساحات من الذاكرة تكفي لعقدة جديدة، فإن البرنامج لن يستطيع تخصيص مساحات من الذاكرة، وسوف يحدث خطأ في تنفيذ البرنامج، ويُمكن معالجة وتجنب مثل هذه المشكلة. يجب التأكد من أن حقل Next في كل عقدة جديدة قد أُعطي القيمة NIL (أي أصبح يُشير إلى اللاشيء) لأن معظم الإجراءات التي تتعامل مع القوائم المترابطة تبحث عن القيمة NIL لتحدد وصولها إلى نهاية القائمة أم لا، وإذا لم تُهيأ هذه المؤشرات، فإن آخر عقدة لن تحتوي القيمة NIL.

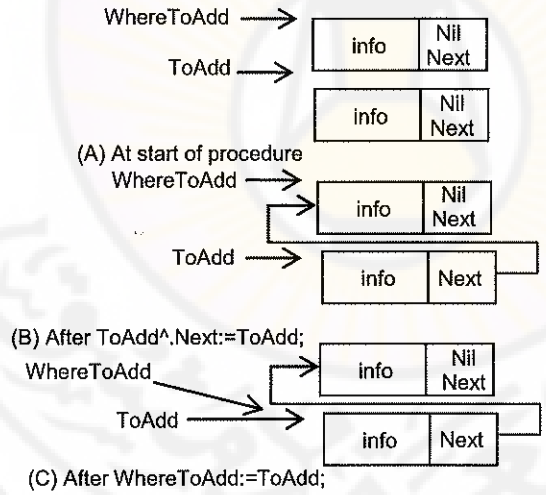
5.11 الإجراء LLAt Front Procedure LLAt Front

بعد أن يتم تخصيص مساحة من الذاكرة لعقدة جديدة، فإن الحقل Info في هذه العقدة سوف يُعطى القيمة (0) ومن ثم يتم استدعاء الإجراء LLAtFront الذي يضع العقدة الجديدة في بداية اللائحة المترابطة. لاحظ أن عبارة واحدة قد جعلت المؤشرين WhereToAdd و To Add.Next ^ (أو Root و Temp.Next ^ على الترتيب ضمن البرنامج الرئيسي) يُشيران إلى نفس العقدة. إن الخوارزمية تقوم على جعل الحقل Next من المتغير ^ ToAdd يُشير إلى العقدة الأولى ومن ثم جعل المؤشر WhereToAdd والذي يكافئ Root يشير إلى العقدة الجديدة، وترتيب هذه العمليات مهم.

الطريقة الوحيدة للوصول إلى عقد القائمة تتم عن طريق مؤشر الجذر، أي المؤشر الذي يُشير إلى أول عقدة، لأن هذا المؤشر هو المتغير الوحيد الذي خُصِص تخصيصاً ساكناً (statically) ضمن القائمة المترابطة، أي أن هذا المتغير هو الوحيد الذي يمكن الوصول إليه مباشرة. يبين الشكل التالي عملية إضافة العقدة الثانية.



إذا جعلنا المؤشر WhereToAdd يُشير إلى العقدة الجديدة، فإننا لا نستطيع الوصول إلى لائحة مفقودة، لأن عبارة الإلحاق غيرت الموقع الذي يؤشر إليه المؤشر مما يجعلنا نفقد عنوان العنصر الأول وبالتالي يتعذر الوصول إليه. يبين الشكل التالي كيف تحدث عملية الفقدان تلك.



يُظهر الجزء A في الشكل السابق وضع المؤشرات قبل استدعاء الإجراء LLAtFront، ويبين عقدة جديدة تنتظر إضافتها إلى القائمة المترابطة، وفي

هذه القائمة كما نلاحظ يتم الوصول إلى عناصرها من خلال المؤشر WhereToAdd (أو ما يُدعى بمؤشر الجذر Root ضمن البرنامج الرئيسي).

أما في الجزء B فإن دليل القائمة المترابطة أشار إلى العقدة الجديدة مباشرة، وعندئذ يحصل فقدان القائمة الأصلية، و لا نستطيع الوصول إلى العنصر الأول من خلال دليل القائمة. حتى نتجنب هذه المشكلة، والتي قد تُسبب ضياعاً كبيراً للمعلومات، يجب أولاً جعل حقل Next في العقدة الجديدة يُشير إلى بداية اللائحة، عندئذٍ فقط يمكن فك الربط بين دليل القائمة والقائمة نفسها بأمان.

6.11 الإجراء Write list Write list Procedure

يُظهر هذا الإجراء محتويات العقد المُستقلة في القائمة المترابطة بنفس ترتيبها الموجود في تلك القائمة. لقد استُخدِم في بناء هذا الإجراء تقنية التعاودية (recursive)، حيث يُظهر هذا الإجراء محتويات الحقل Info للعقدة الحالية، ومن ثم يستدعي نفسه من أجل العقدة التالية في القائمة، حتى يصل هذا الاجراء إلى العقدة الأخيرة، أي التي يؤشر فيها الحقل Next إلى اللاشيء NIL. يُستدعى الإجراء مع جذر القائمة المترابطة، أي العقدة الأولى، فيُظهِر هذا الإجراء محتويات الحقل Info فيها، ومن ثم يختبر الإجراء وصوله إلى نهاية القائمة، فإذا كانت نتيجة التعبير المنطقي التالي هي True.

TheNode^.Next <> NIL

فإن هذا يعني أن الإجراء لم يصل إلى نهاية القائمة، عندئذٍ يُعيد الإجراء استدعاء نفسه مع مؤشر العقدة التالية، أي مع مؤشر الحقل Next للعقد الحالية، وذلك باستخدام العبارة التالية.

```
WriteList (TheNode^.Next);
```

أما إذا كان ناتج التعبير السابق هو False، فهذا يعني وصول الإجراء إلى نهاية القائمة المترابطة وعندئذ يُنهي هذا الإجراء نفسه.

بما أن تعليمه إظهار محتويات الحقل Info، أي استدعاء الإجراء Writeln، قد سبقت الاستدعاء التعاودي، فإن محتويات هذا الحقل سوف تُكتَب ابتداءً من العقدة الأولى وحتى العقدة الأخيرة. لكن إذا غيَّرنا موضع العبارة Writeln في الإجراء السابق، ووضعناها بعد الاستدعاء التعاودي فإن الإجراء **WriteList** سوف يكتب محتويات العقد ابتداءً من العقدة الأخيرة وحتى العقدة الأولى.

7.11 إضافة العقد في أي مكان ضمن القائمة المترابطة Adding Nods Elsewhere in a Linked List

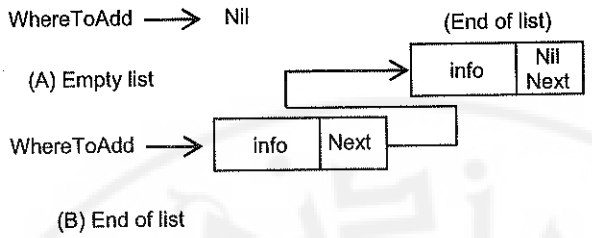
يمكننا أيضاً بناء قائمة مترابطة بإضافة عناصر جديدة في نهاية هذه القائمة كما هو موضح في المثال التالي.

مثال.

```
Procedure LLAtSack(ToAdd:LNodePtr;Var,
     WhereToAdd:LNodePtr);
  Begin
     If(WhereToAdd=NIL) then
       WhereToAdd:=ToAdd
     Else
       LLAtBack(ToAdd, WhereToAdd^.Next);
  End;
```

أضف هذا الإجراء إلى قسم التصريح للبرنامج السابق LListDemo، ومن ثم استبدل استدعاء الإجراء LLAtFront في البرنامج الرئيسي بالاستدعاء LLAtBack. يُضيف الإجراء LLAtSack عقد جديدة إلى نهاية القائمة الحالية، وخوارزمية عملية الإضافة تلك تعتمد على عملية العبور (traverse)، أي الانتقال ضمن القائمة حتى نصل إلى نهايتها، حيث سنجد المؤشر يشير إلى القيمة NIL.

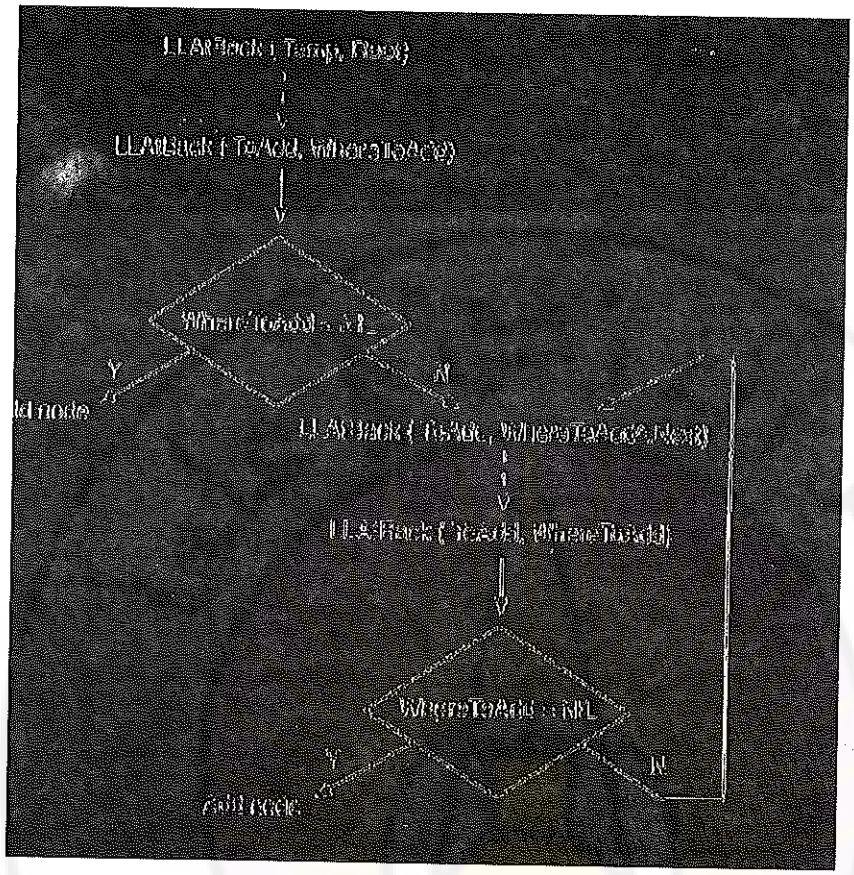
يبين الشكل التالي موضعين يمكن أن نجد فيهما القيمة NIL ضمن القائمة المترابطة.



عندما يصل الإجراء السابق إلى مؤشر يُشير إلى القيمة NIL، فإن العقدة الجديدة سوف تُضاف إلى القائمة، وهذه القائمة المُعدلة سوف تُعاد إلى الاجراء المُستَدعي. إذا كانت هناك عقدة في الموقع الحالي للمؤشر WhereToAdd، فإن الإجراء LLAtBack يستدعي نفسه مع المؤشر الموجود في الحقل لهذه العقدة (أي WhereToAdd.Next^) كمتحول وسيطي جديد، حتى يصل إلى نهاية القائمة.

يبين الشكل التالي المخطط المنطقي للعمليات التي تتم ضمن LLAt Back.

Damascus University



يُشير السطر الأول في الشكل إلى الوسطاء نفسها المستخدمة في استدعاء الإجراء من قبل الاجراء المُستدَعى. أما في السطر الثاني فإن الوسطاء المُستخدَمة هي الوسطاء التي استخدمها الإجراء نفسه.

8.11 ترتيب قائمة مترابطة Ordering a Linked List

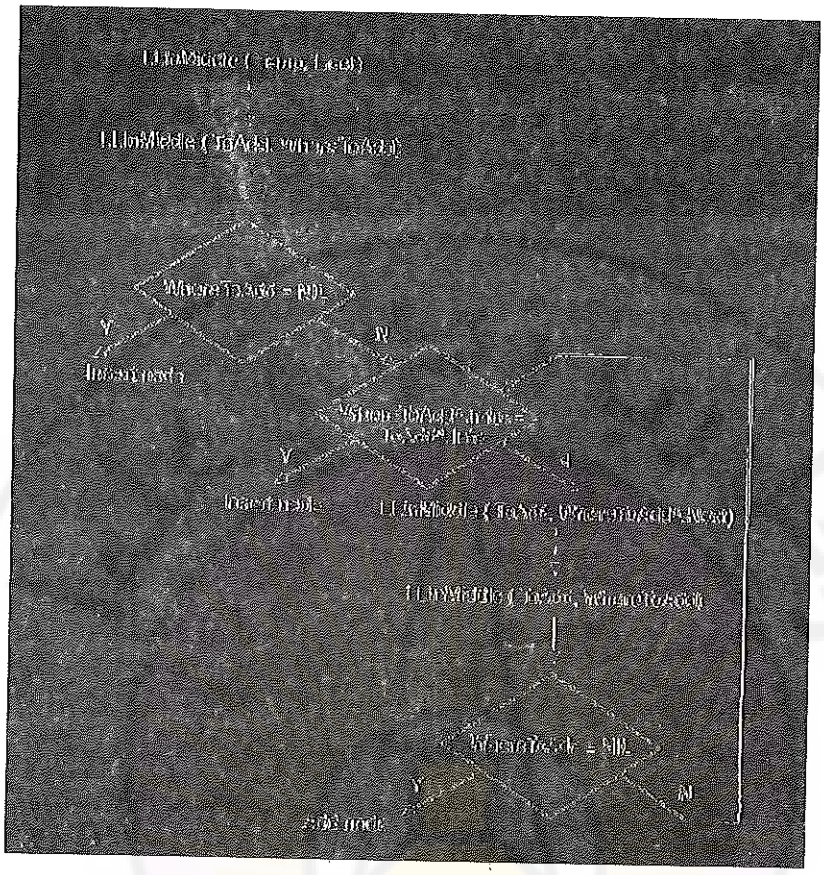
يُمكن ترتيب عناصر القائمة المترابطة وفق طريقة ما. يقوم البرنامج التالي بترتيب عناصر عقد القائمة المترابطة وفق قيمها العددية الموجودة في الحقل Info. ويبني الإجراء LLInMiddle قائمة مرتبة واضعاً العقدة التي قيمة الحقل Info فيها أقل من باقي العقد في بداية هذه القائمة. أضف للبرنامج LListDemo، واستبدل استدعاء الإجراء LLAtBack باستدعاء الإجراء LLInMiddle.

```
Procedure LLInMiddle(ToAdd:LNodePtr; Var
```

```
    WhereToAdd:LNodePtr);
Begin
    If(WhereToAdd=NIL) then
      WhereToAdd:=ToAdd
    Else if(WhereToAdd^.Info>=ToAdd^.Info) then
      begin
         ToAdd^.Next:=WhereToAdd;
         WhereToAdd:=ToAdd;
      End
    Else
        LLInMiddle(ToAdd,WhereToAdd^.Next);
End;
```

هذا الإجراء يتحرك ضمن القائمة مقارناً قيم الحقل للعقد Info باحثاً عن المكان المناسب للعقدة الجديدة. فإذا كانت القائمة فارغة (empty)، أو وصل الإجراء إلى نهاية القائمة المترابطة، فإن العقدة الجديدة سوف تُضاف إلى مقدمة القائمة المترابطة، كما هو الحال في الإجراء LLAtBack. أما إذا كانت قيمة الحقل Info لعقد القائمة أكبر من القيمة الجديدة، فإن العقدة الجديدة سوف تُضاف إلى مقدمة القائمة المترابطة. في الحقيقة نحن نُضيف دائماً العقدة الجديدة إلى مقدمة قائمة فرعية (sublist)، تحتوي على العقد التي قيم الحقل Info لها أكبر من قيمة هذا الحقل في العقدة الجديدة.

بعد أن تُضاف العقدة الجديدة، فإن القائمة سوف تُعدَّل، وسوف تُمرَر وفق تسلسل الاستدعاءات، حتى تصل إلى البرنامج الرئيسي، الذي سوف يستقبل هذه القائمة المُعدَّلة. يبين الشكل التالي المخطط المنطقي للعمليات التي تتم ضمن LLInMiddle.



مثال:

البرنامج التالي يبين كيفية قراءة محتويات ملف ما ومن ثم إعادة كتابة هذه المحتويات وفق ترتيبها الأبجدي و ذلك باستخدام القوائم المترابطة.

```
Program SortLines;
Type
     RegStr=string[80];
     LNodePtr= ^LNode;
     LNode=Rocord
      Info:RegStr;
      Next:LNodePtr;
     End;
```

```
Var
      Root,Temp:LNodePtr;
      Source,Log:Text;
      SName,LName:string;
      NrLines:integer;
Function FOpened(Var TheFile:Text; FName:string)
         :boolean;
    Begin
         assign(TheFile,FName);
         (*$I-*)
         Reset(TheFile);
         (*$I+*)
         If(IOResult=0) then
           FOpened:=true
         Else
           FOpened:=false;
         End;
Function FMade(VAR TheFile: Text; FName:string):
        boolean;
    Begin
       Assign(TheFile,FName);
       (*$I-*)
       Rewrite(TheFile);
       (*$I+*)
       If(IOResult=0) then
         FMade:=true
       Else
         FMade:=false;
       End;
Procedure GetString(Message:string; Var Value:
```

```
        string);
    Begin
       WP.ITE(Message,' ');
       Readln(Value);
    End;
 Procedure ShowProgress(Val, Small, Large:
        longint);
     const MarkerCh='.';
  Begin
     If(Val mod Small=(Small-1)) then
       Write(MarkerCh);
     If(Val mod Large=(Large-1)) then
       Writeln;
  End;
 Procedure InitLNode(Var TheNode:LNodePtr);
   Begin
      New(TheNode);
      TheNode^.Info:= '';
      TheNode^.Next:=NIL;
   End;
 Procedure LLInMiddle(ToAdd:LNodePtr; Var
     WhereToAdd:LNodePtr);
 Begin
      If(WhereToAdd=NIL) then
      WhereToAdd:=ToAdd
 Else
      If(WhereToAdd^.Info>=ToAdd^. Info) then
        Begin
            ToAdd^.Next:=WhereToAdd;
            WhereToAdd:=ToAdd;
```

```
        End
      Else
           LLInMiddle(ToAdd,WhereToAdd^.Next);
      End;
Procedure IterFWriteList(Var TheFile:Text;
      TheNode:LNodePtr);
  Begin
       While(TheNode <> NIL) do
         Begin
            Writeln(TheFile,TheNode^.Info);
            TheNode:=TheNode^.Next;
         End;
   END;
Begin (* Main* )
    Root:=NIL;
    Temp:=NIL;
    MrLines:=0;
    Writeln('MEMAVAIL=',MEMAVAIL:10,' bytes');
    GetString('Source file?',SNane);
    GetString('Log file?', LNaroe);
    If (FOpened(Source, SName) and FMade(Log,
        LName)) then
     Begin
      While (not EOF( Source)) and
        (MAXAVAIL>sizeof(LNode)) do
        Begin
         InitLNode(Temp);
         Readln(Source,Temp^.Info);
         INC(NrLines);
         ShowProgress(NrLines,5,250);
```

```
          LLInMiddle(Temp,Root);
        End;
      If(not EOF(Source)) then
       Begin
        Writeln('Terminated early;not enough
          memory');
        IterFWriteList(Log,Root);
        Close(Log);
        Close(Source);
        Writeln;
        Writeln(NrLines:5,'lines processed');
       Writeln('MEMAVAIL=', MEMAYAIL:10,'bytes');
       End
     Else
      Writeln('File error');
   End;
 Readln;
End.
```

يقرأ البرنامج السابق أسطراً من الملف النصي Source، ومن ثم يُضيف هذه الأسطر إلى قائمة مترابطة تُرتَب عقدها على أساس السلاسل المحرفية، (أي محتويات الحقل Info) ومن ثم يكتب البرنامج محتويات القائمة المترابطة في الملف النصي log.

تعريف العقدة LNode في هذا البرنامج يختلف عن تعريفها في البرامج السابقة، حيث ينتمي الحقل Info في هذا البرنامج إلى نوع البيانات سلسلة محارف String وليس نوع البيانات عدد حقيقي كما كان. وقد تم تحديد هذه السلسلة المحرفية على ألا يتجاوز (80) رمزاً على الأكثر، وهذا التحديد سوف يُقلل من استهلاك ذاكرة الكومة، وبالتالي يُمَكّن من تخصيص عقد أكثر في ذاكرة الكومة.

ملاحظات:

1. إن اجرائيات معالج اللائحة المترابطة **تُستَدعى** في حلقة While في البرنامج الرئيسي، حيث يُخَصِص البرنامج مساحة من الذاكرة لكل سطر جديد باستخدام الإجراء InitLNode.
2. يُضاف مساحة الذاكرة تلك إلى القائمة المترابطة بوساطة الإجراء LLInMiddle. وهذا الاجراء مُطابق تماماً للإجراء الموجود في البرنامج السابق.
3. سوف تُكتَب الأسطر المُرتبة في الملف باستخدام الإجراء IterFWriteList، وهو إجراء غير تعاودي، ولكن يمكن جعله تعاودياً بسهولة.
4. يكتب الإجراء IterFWriteList الأسطر المرتبة المخزنة في القائمة المترابطة في الملف Log، وذلك من خلال عبور كامل القائمة، حيث يأخذ كل عقدة The Node من القائمة على حدة، ويكتب الحقل Info من الملف، ومن ثم يختبر وصوله إلى نهاية اللائحة من خلال بحثه عن القيم NIL التي تعني نهاية القائمة.

لقد تم تمرير المتغير الوسيطي الثاني للإجراء IterFWriteList بوساطة قيمته، وهذا المتغير الوسيطي عبارة عن اللائحة المترابطة الأساسية، أي التي يؤشّر عليها Root.

تمرير هذه القائمة بقيمتها يعني الحصول على نسخة من العناوين المُخزَنة في المؤشر Root وتمريرها إلى الإجراء. السبب في تمرير هذه القائمة بقيمتها أن هذا التمرير مأمون عند تغيير قيمة العقدة The Node في أثناء تنفيذ الإجراء السابق. إذا مُرِرَت العقدة مرجعياً سوف تُحفَظ التغيرات التي تطرأ على القائمة، أي أن تغيير مؤشرات القائمة ضمن الإجراء السابق سوف

يُحفَظ، وبالتالي بعد الانتهاء من تنفيذ التكرارية الموجودة في الإجراء السابق سوف نفقد القائمة كلها، ولن نستطيع الوصول إلى أي عقدة فيها.

استخدام عقد أكثر تعقيداً Using more Complex Nodes

إن العقد التي يُمكن إضافتها إلى عقد القائمة المترابطة تعتمد وبشكل أساسي على أمرين اثنين: أولهما احتياجات البرنامج، وثانيهما مقدرة المبرمج على تخيل هذه العقد. إن تعديل القائمة المترابطة يعتمد على تعديل تعريف العقدة LNode.

لقد استخدمنا تعريفاً للعقدة في البرامج السابقة يُستخدَم لبناء القوائم المترابطة الأحادية (singly linked lists) هذا يعني أن كل عقدة في هذه القائمة تشير إلى عقدة أخرى. في الحقيقة كل عقدة تُشير إلى عقدة تالية عدا العقدة الأخيرة التي تُشير إلى اللاشيء NIL. يُمكن إنشاء قوائم مترابطة مضاعفة (doubly linked lists)؛ هذا يعني أن كل عقدة تُشير إلى عقدة أخرى أو عقدتين كما. فيما يلي تعريف لعقدة اللائحة المترابطة المضاعفة.

```
Type
    DLLPtr=^DList;
    DList=Record
       Info:Real;
       Next, Preceding:DLLPtr;
    End;
```

تختلف القائمة المترابطة المضاعفة عن سابقتها الأحادية في أن عبورها (traverse) يمكن أن يتم بالاتجاهين، لأن وجود مؤشرين في سجل واحد يُعطينا مرونة كبيرة في طريقة التنقل. مثلاً يمكن استخدام المؤشرين لبناء قائمتين مختلفتين في نفس العقد، بالاعتماد على معلومات مختلفة ضمن هاتين القائمتين وفي هذه الحالة سوف يُستخدَم كل مؤشر لغرض مختلف.

نبني العقدة المطلوبة من الحقول التالية:

1. الحقل الأول عبارة عن سلسلة محارف من أجل تخزين الكلمة المأخوذة من الملف.
2. الحقل الثاني عبارة عن عدد صحيح لتمثيل تردد هذه الكلمة.
3. الحقل الثالث والحقل الرابع عبارة عن مؤشرين، أحدهما يُستخدَم لبناء قائمة مُرتَبة من الكلمات، والآخر يُستخدَم لبناء قائمة مُرتَبة من ترددات الكلمات.

يُمكن استخدام التصريحات التالية.

```
Type
    ShortStr=string[30];
    DLNodePtr=^DLNode;
    DLNode=Record
      Wd:ShortStr;
      Freq:integer;
      StrNext,FreqNext:DLNodePtr;
    End;
```

إن العقد تُمثَّل بسجلات، و قد مُثِّلت العقدة بسجل يحتوي أربعة حقول، يُستخدَم حقلان منهم وهما Wd و StrNext لبناء قائمة الكلمات، أما الحقلان الآخران Freq و FreqNext فيُستخدَمان لبناء قائمة الترددات.

مثال:

يقوم البرنامج التالي FileStats بالعمليات التالية:

1. يقرأ الملف سطراً سطراً.
2. يُضيف الكلمات المُستقلة في كل سطر إلى لائحة الكلمات، حيث تُخصَص عقدة DLNode لكل كلمة، وفي حال كون الكلمة موجودة في القائمة فإن

البرنامج سوف يزيد الحقل Freq الخاص بالكلمة، أما إذا لم تكن الكلمة موجودة في القائمة، فإن البرنامج سوف يضعها في مكانها المناسب في القائمة.

3. بعد مسح الملفات بشكل كامل فإن البرنامج يبني قائمة مترابطة أخرى تعتمد تردد الكلمات أساساً لها، ويقوم بالبحث ضمن ملف، ويُعيد عدد مرات تكرار الكلمات في ذلك الملف، حيث ستُقرأ الكلمات وتُوضع مُرتَبة ضمن القائمة المُترابطة، وستُربط كل كلمة بترددها (frequency) أي عدد مرات تكرار هذه الكلمة في الملف.

```
Program FileStats;
Const
     EmptyString=' ';
Type
     CharSet=set of char;
     ShortStr=string[30];
     DLNodePtr=^DLNode;
     DLNode=record
        Wd:ShortStr;
        Freq:integer;
        StrNext,FreqNext:DLNodePtr;
     End;
Var
   FreqRoot,Root,Temp:DLNodePtr;
   CurrStr,CurrWd:string;
   Source,Log:Text;
   SName,LName:string;
   NrWds,NrUnique:longint;
   Choice:integer;
Procedure RemoveWd(Var Wd, Sentence:string);
```

```
    Const Blank=' ';
    Var Where:integer;
  Begin
    While(POS(Blank,Sentence)=1) do
       Delete(Sentence, 1, 1);
    Where:=POS(Blank, Sentence);
    If (Where > 0) then
      Begin
         Wd:=COPY(Sentence, 1, Where-1);
         Delete(Sentence, 1, Where);

      End
     Else
      Begin
        Wd:=Sentence;
        Sentence:=EmptyString;
      End;
  End;
Function FOpened(Var TheFile:Text;
          FName:string):boolean;
   Begin
      assign(TheFile, FName);
      (*$I-*)
      Reset(TheFile);
      (*$I+*)
      If(IOResult = 0) then
        FOpened:=true
      Else
        FOoened:=false;
   End;
```

```
Function FMade(Var TheFile:Text; FName:string):
     boolean;
  Begin
     assign(TheFile, FName);
     (*$I-*)
     Rewrite(TheFile);
     (*$I+*)
     If(IOResult=0) then
        FMade:=true
     Else
        FMade:=false;
  End;
Procedure GetString(Message:string; Var
        Value:string);
  Begin
     Write(Message, ' ');
     Readln(Value);
  End; (* GetString* )
Procedure ShowProgress(Val,Small,Large:longint);
   Const MarkerCh='.';
  Begin
    If(Val mod Small=(Small-1)) then
      Write(MarkerCh);
    If(Val mod Large=(Large-1)) then
       Writeln;
  End;
Procedure MakeStrLower(Var TheStr:string);
   Const Offset=32;
   Var Index, HowLong:integer;
  Begin
```

```
    HowLong:=Length(TheStr);

    For Index:=1 to HowLong do
      begin
        If TheStr[Index] in ['A'..'Z'] then
          INC(TheStr[Index], Offset);
      End;
  End;
Procedure CleanStr(Var TheStr:string;
    TheSet:CharSet);
  Const Blank=' ';
  Var Index,HowLong:integer;
 Begin
    HowLong:=Length(TheStr);
    For Index:=1 to HowLong do
      Begin
       If(not(TheStr[Index] in TheSet)) then
          TheStr[Index]:=Blank;

      End;
  End;
Procedure GetBoundedInt(Message:string;
    MinVal,MaxVal:integer; Var Value:integer);
Procedure AdjustVals(Var Small,Large:integer);
    Var Temp:integer;
  Begin
    If(Small > Large) then
      Begin
        Temp:=Small;
        Small:=Large;
```

```
            Large:=Temp;
        End;
    End;
Function IsBetween(Val, Small, Large:integer):
     boolean;
   Begin
     IsBetween:=((Small<=Val) and (Large>=Val));
   End;
Begin
     adjustVals(MinVal, MaxVal);
     Repeat
          Write(Message,' ');
          Readln(Value);
     Until (IsBetween(Value, MinVal, MaxVal));
End;
Procedure InitDLNode(Var TheNode:DLNodePtr);
     Begin
          New(TheNode);
          TheNode^.Wd:=EmptyString;
          TheNode^.Freq:=0;
          TheNode^.FreqNext:=NIL;
          TheNode^.StrNext:=NIL;
     End;
Procedure IterFWriteDList(Var TheFile:Text;
     TheNode:DLNodePtr);
  Var Count:integer;
Begin
     Count:=0 ;
     While(TheNode <> NIL) do
          Begin
```

```
                Write(TheFile,TheNode^.Wd,' : ',
                  TheNode^.Freq);
                INC(Count);
                TheNode:=TheNode^.StrNext;
            End;
        Writeln(Count:4,'entries in word list');
    End;
Procedure iIterFWriteDList(Var TheFile:Text;
          TheNode:DLNodePtr);
        Var Count:integer;
    Begin
        Count:=0;
        While(TheNode <> NIL) do
          Begin
            Writeln(TheFile,TheNode^. Freq:4,':
              ',TheNode^.Wd);
            INC(Count);
            TheNode:=TheNode^.FreqNext;
          End;
       Writeln(Count:4,'entries in frequency list');
    End;
Procedure DLLInMiddle(ToAdd:DLNodePtr; Var
          WhereToAdd:DLNodePtr);
      Begin
        If(WhereToAdd=NIL) then
            WhereToAdd:=ToAdd
        Else if(WhereToAdd^.Wd > ToAdd^.Wd) then
          Begin
            ToAdd^.StrNext := WhereToAdd;
            WhereToAdd:=ToAdd;
```

```
        End
      Else if(WhereToAdd^.Wd=ToAdd^.Wd) then
        Begin
          INC(WhereToAdd^.Freq);
        End
      Else
          DLLInMiddle(ToAdd, WhereToAdd^.StrNext);
   End;
Procedure iDLLInMiddle(ToAdd:DLNodePtr;
     Var WhereToAdd:DLNodePtr);
  Begin
     If(WhereToAdd=NIL) then
       WhereToAdd:=ToAdd
     Else if(WhereToAdd^.Freq<=ToAdd^.Freq) then
       Begin
        ToAdd^.FreqNext:=WhereToAdd;
        WhereToAdd:=ToAdd;
       End
     Else
       iDLLInMiddle(ToAdd, WhereToAdd^.FreqNext);
  End;
Begin (* Main program *)
     Writeln('MEMAVAIL=',MEMAVAIL:10,'bytes');
     Writeln('MAXAVAIL=',MAXAVAIL:10,'bytes');
     Root:=NIL;
     FreqRoot:=NIL;
     Temp:=NIL;
     NrWds:=0;
     NrUnique:=0;
     GetString('Source file?',SName);
```

```
GetString('Log file?',LName);
If(FOpened(Source,SName) and
    FMade(Log,LName)) then
  Begin
    While(not EOF(Source)) and (MAXAVAIL>
        sizeof(DLNode)) do
      Begin
        Readln(Source,CurrStr);
        MakeStrLower(CurrStr);
        CleanStr(CurrStr,['a'..'z','A'..'Z'
          ,'-',''''']);
        Repeat
          RemoveWd(CurrWd,CurrStr);
          If(CurrWd <> EmptyString) then
            Begin
              INC(NrWds);
              ShowProgress(NrWds,5,250);
              InitDLNode(Temp);
              Temp^.Wd:=CurrWd;
              Temp^.Freq:=1;
              DLLInMiddle(Temp,Root);
            End;
        Until(CurrStr=EmptyString);
      End;
  If(not EOF(Source)) then
    Writeln('Terminated early;not enough
      memory.');
  Writeln;
  Writeln(NrWds:10,'words');
  Temp:=Root;
```

```
While(Temp 0 NIL) do
   Begin
     iDLLInMiddle(Temp,FreqRoot);
     Temp:=Temp A.StrNext;
   End;
 GetBoundedInt('Save by 1) word; 2)
     frequency; 3) both',1, 3, Choice);
 Case (Choice) of
  1:IterFWriteDList(Log,Root);
  2:iIterFWriteDList(Log,FreqRoot);
  3: Begin
        IterFWriteDList(Log,Root);
        Writeln(Log, '----------------');
        iIterFWriteDList(Log,FreqRoot);
     End;(*3*)
 End;
   Writeln('MEMAVAIL=',MEMAVAIL:10,'bytes');
   Writeln('MAXAVAIL=',MAXAVAIL:10,'bytes');
   Close(Source);
   Close(Log);
  End (Fopened and FMade)
 Else
  Writeln('File error.');
Readln;
End.
```

يبني البرنامج السابق قائمة الكلمات أولاً بتخصيص مساحات من الذاكرة لكل عقدة. من أجل كل كلمة يقرأها البرنامج من الملف Source، يستدعي البرنامج الإجراء lnitDLNode. بعد بعض العمليات المطلوبة من إلحاق وتعديل قيم، يُضيف البرنامج هذه الكلمة، أي العقدة التي تحتوي الكلمة، إلى قائمة الكلمات،

وذلك باستخدام الإجراء DLLInMiddle، وهذا الإجراء مُشابه تقريباً لما سبق، إلا أننا أضفنا هنا جزءاً else في عبارة if، حيث يُعالج هذا الجزء حالة كون الكلمة الجديدة مطابقة لكلمة موجودة في القائمة، حيث يتم إختبار هذه الحالة باستخدام التعبير: WhereToAdd^.Wd=ToAdd^Wd;

إذا كانت قيمة هذا التعبير true، فإن الكلمة لن تُضاف، ولكن سوف تزداد قيمة ترددها بمقدار (1). بما أن هذه العقدة لن تُضاف إلى القائمة فإن مساحة التخزين المخصصة لها سوف تُفقَد عندما يستدعي البرنامج الإجراء InitDDLNode مرة أخرى.

يُمكن تعديل البرنامج السابق حتى يُعيد مساحات التخزين غير المُستخدَمة إلى الكومة وهذا التعديل سوف يُمَّكن من معالجة عدد أكبر من الكلمات المختلفة.

تُبنى لائحة الترددات والتي دليلها هو FreqRoot على النحو الآتي:

1. يتفحص البرنامج لائحة الكلمات حتى التأكد من أن كل الكلمات قد أُضيفَت إلى لائحة الترددات ويُستخدَم في ذلك المؤشر الوسيطي Temp.
2. يضيف البرنامج إلى قائمة الترددات كل عقدة من قائمة الكلمات معتمداً على حقل التردد لهذه العقدة، وذلك باستخدام الإجراء iDDLnMiddle. المؤشر Freq Root لا علاقة له بالمؤشر Root، لأن قائمة الكلمات يجب أن تكون مستقلة عن قائمة الترددات، حتى لو كانت القائمتان تُستخدَمان نفس العقد.

تجدر الإشارة إلى أننا لم نستدع الإجراء InitDLNode في الجزء الثاني من البرنامج السابق FileStats، لأن كل العقد المطلوبة قد تم تخصيصها حيث يُعالِج الجزء الثاني من البرنامج مؤشرات العقد فقط، ولا يبني عقداً

جديدة. وأخيراً يستخدم البرنامج الإجراءين IterFWriteList و IterFWriteList في كتابة قائمة الكلمات وقائمة الترددات على الترتيب.

تمارين

1) ضع برنامجاً بلغة الباسكال يقوم بإدخال متتالية من الأعداد الصحيحة المنتهية بالصفر، و بناء قائمة مترابطة تحتوي على الأعداد الموجبة منها، ثمّ طباعة عناصر هذه القائمة.

2) ضع برنامجاً بلغة الباسكال يقوم بإنشاء ملف من الأعداد الصحيحة المنتهية بالصفر. ثم ضع برنامجاً آخر يقوم بقراءة الملف السابق و إنشاء قائمة مترابطة تحتوي على الأعداد الموجبة فقط، ثمّ قم بحذف الأعداد الزوجية من القائمة و طباعة هذه القائمة الأخيرة من جديد.

3) ضع برنامجاً بلغة الباسكال يقوم بمايلي:

a) تعريف كائن موظف في شركة معينة. يجب أن يحتوي الكائن على مجموعة من الحقول: الاسم، الكنية، الرقم الذاتي، الراتب، الحالة العائلية، تاريخ الميلاد. يجب أن يوفر الكائن الدوال التالية: دالة تعرف قيم (مدخلة من قبل المستخدم) لحقول الكائن، دالة تعيد كل حقل من حقول الكائن، دالة تمكن من تحديث كل حقل من حقول الكائن.

b) تعريف قائمة مترابطة تحتوي على هذه الكائنات.

c) كتابة برنامج جزئي يقوم ببناء القائمة.

d) كتابة برنامج جزئي يقوم بالبحث ضمن القائمة عن موظف من خلال اسمه.

e) كتابة برنامج جزئي يقوم بحذف عنصر من القائمة.

4) ضع برنامجاً بلغة الباسكال يقوم بتعريف مؤشر من النوع الصحيح ثم يقوم بجعله يؤشر على متحول من نوع صحيح. إطبع القيمة التي يؤشر عليها المؤشر و قيمة العنوان التي يحتويها ذلك الموشر. أسند قيمة للمؤشر ثم إطبع قيمة المتحول الصحيح (ماذا تلاحظ؟).


وقل رب زدني علما
جامعة دمشق
Damascus University

الفصل السادس

الوحدات و الرسم بلغة الباسكال Pascal Units

1. مقدمة

جميع البرامج التي كتبناها في الفصول السابقة هي برامج بسيطة و مُصمَمة لحل مسألة معينة. لكن في أغلب الأحيان تحتاج التطبيقات العملية إلى برامج ضخمة لحل المسائل. بسبب صعوبة البرمجة و فهم البرامج المكتوبة و خصوصاً عندما تكون هذه البرامج كبيرة، فإننا نقوم بتقسيم البرامج إلى مكونات متكاملة ندعوها Modules و ذلك لتسهيل فهمها و كشف الأخطاء إن وجدت. يؤدي كل مكون وظيفة محددة و قائمة بحد ذاتها، و بالتالي يتم فهم البرنامج من خلال فهم المكونات المُشَكِلة له و ربط هذه المكونات بعضها مع بعض.

المكونات في لغة الباسكال القياسية هي عبارة عن دوال و إجراءات. في الواقع ، فإن الدوال و الإجراءات تُسَّهل كثيراً فهم البرامج و ذلك لأنه يُمكن فهم كل دالة و إجراء بشكل أسهل من فهم البرنامج بأكمله. المُشكلة مع ذلك هي أن الدوال و الإجراءات يجب أن تكون في نفس الملف مع البرنامج العام الذي يستخدمها. هذا الشكل من تقسيم البرنامج إلى مكونات متكاملة على الرغم من أهميته إلا أنه يُبقي على بعض الصعوبات في فهم البرامج.

لزيادة فعالية تقسيم البرامج الكبيرة إلى مكونات تم إضافة مكون آخر أكثر قوة و فعالية يُسمى الوحدة أو Unit. بكلمات بسيطة الوحدة هي برنامج يحتوي على إجراءات و دوال و حقول و يمكن لأي برنامج آخر استخدام هذه الوحدة و استدعاء الإجراءات و الدوال و حتى الحقول الموجودة فيها دون الحاجة لوضع هذه الإجراءات و الدوال في البرنامج العام. من الواضح أن هذه الفلسفة بكتابة البرامج تجعل حجم البرنامج أصغر مما يزيد إمكانية فهمه بشكل أسهل و بنفس الوقت يسَهل عملية كشف الأخطاء إن وجدت.

سوف نبدأ هذا الفصل بمجموعة من الوحدات القياسية المُعرَّفة بلغة الباسكال و نبين كيفية استخدامها. بعد ذلك سوف نسهب في شرح كيفية كتابة وحدات جديدة.

2. الوحدات الجاهزة بلغة الباسكال Predefined Units

تشتمل لغة الباسكال على العديد من الوحدات الجاهزة و التي تزود المبرمجين بالعديد من الإجراءات و الدوال الجاهزة. سنستعرض بعض هذه الوحدات و نبين كيفية استخدامها. لكن قبل البدء في مناقشة هذه الوحدات لابد من التعريف بكيفية ربط وحدة ما بالبرنامج و بالتالي استخدام الإجراءات و الدوال المتوفرة فيها.

يُمكن الاستفادة من خدمات أي وحدة من خلال التعليمة uses و التي يُمكن وضعها في بداية البرنامج الذي يستخدام الوحدة. الصيغة العامة لتعليمة ربط الوحدات بالبرامج هي:

Uses unitlist;

حيث unitlist هي عبارة عن وحدة واحدة أو عدة وحدات مفصولة بعضها عن بعض بفاصلة عادية ",". مثلاً تقوم التعليمة **uses wincrt;** بربط الوحدة wincrt في البرنامج و بالتالي يُمكن للبرنامج أن يستفيد من خدمات هذه الوحدة. كذلك تقوم التعليمة **uses wincrt, printer;** بربط كل من الوحدتين wincrt و printer بالبرنامج الذي وضِعت فيه هذه التعليمة.

1.2 وحدة الطابعة Printer Unit

تُساعد هذه الوحدة في طباعة ملفات نصية على الطابعة. حيث تُعَّرف هذه الوحدة متحول ملف اسمه Lst، وتقوم بربط هذا المتحول مع نظام الطابعة، ثم تفتح ملف الخرج المُرتبط مع الطابعة. البرنامج التالي يوضح كيفية استخدام هذه الوحدة.

يقوم البرنامج بطلب اسم الملف المُراد طباعته ومن ثم يفتح الملف و يرسله إلى الطابعة.

```
Program printFile;
Uses printer; {ربط وحدة الطابعة في البرنامج}
Var
    InFile:Text; {متحول ملف}
    InFileName,Line: String; {هو اسم الملف المراد طباعته
        InFileName}
 Begin
    Write('Type the name of the file to be
       printed >');
    Readln(InFileName);
    Assign(InFile,InFileName);
    Reset(InFile);
    While (not EOF(InFile)) do {حلقة طباعة الملف}
      Begin
        Readln(InFile,Line);
        Writeln(Lst,Line);{الطابعة هنا يتم إرسال السطر إلى}
      End; {while}
    Close(InFile);
    Writeln('File', InFileName ,'is printed');
 End.
```

لاحظ كيف تم ربط وحدة الطابعة ببرنامج الباسكال بوساطة التعليمة uses. كذلك و بسبب أن متحول الملف Lst هو أول وسيط في تعليمة الطباعة Writeln فإن السطر Line سيوجَّه إلى الطابعة بدلاً من وحدة الخرج القياسية الشاشة.

2.2 الوحدة Crt

توفر هذه الوحدة العديد من الإجراءات التي تُمَكننا من التحكم بلوحة المفاتيح و شاشة العرض. تتألف شاشة التنفيذ بلغة الباسكال من 80 عمود و 25 سطر. المحور الأفقي هو المحور X و المحور الشاقولي هو المحور Y، و بالتالي فإن أي نقطة في الشاشة تُعرَّف بالثنائية (X, Y) حيث تأخذ X القيم من 1 إلى 80 و تأخذ Y القيم من 1 إلى 25. نقطة البداية (1, 1) هي الزاوية العليا اليسارية و نقطة النهاية هي الزاوية السفلى اليمينية و إحداثياتها (80, 25).

هناك العديد من الإجراءات المفيدة التي توفرها هذه الوحدة. فيمايلي سنقوم بشرح بعض هذه الإجراءات.

الإجراء Window (X1, Y1, X2, Y2)

يرسم هذا الإجراء نافذة ضمن شاشة التنفيذ بين النقطتين (X1, Y1) و (X2, Y2) و يضع المشيرة في بداية النافذة. بطبيعة الحال فإن مجال قيم X1, X2 هي من 1 إلى 80 و مجال قيم Y1, Y2 هي من 1 إلى 25. فمثلاً، التعليمة Window (1, 1, 80, 25) ترسم نافذة بحجم شاشة التنفيذ بينما التعليمة Window (20, 10, 60, 15) ترسم نافذة في وسط شاشة التنفيذ بـ 41 عمود و 6 أسطر.

الإجراء Clrscr

يقوم بمسح محتويات شاشة التنفيذ و يضع المشيرة في بداية الشاشة. يُمكن استدعاء هذا الإجراء في أي مكان من البرنامج.

الإجراء GoToXY (X, Y)

يقوم هذا الإجراء بنقل المشيرة إلى الموقع المُحدَد بـ X, Y في النافذة الحالية. مثلاً: التعليمة GoToXY(12, 6) تقوم بنقل المشيرة إلى العمود 12 و السطر 6 في النافذة الحالية.

الإجراء TextBackground (Color)

يقوم هذا الإجراء بتلوين خلفية الشاشة أو النافذة باللون المُحدَد بـ Color. يُمكن أن نضع مكان الوسيط Color إما ثابت (مثلاً Red) أو القيمة العددية المقابلة لهذا الثابت (مثلاً 4). عندما نكتب التعليمة TextBackground (Blue) فإن خلفية شاشة التنفيذ أو النافذة ستُلوَن باللون الأزرق. يبين الجدول التالي الثوابت اللونية المُتاحة بلغة الباسكال.

الثابت	القيمة	الثابت	القيمة
Black	0	DrakGray	8
Blue	1	LightBlue	9
Green	2	LightGreen	10
Cyan	3	LightCyan	11
Red	4	LightRed	12
Magenta	5	LighMagenta	13
Brown	6	Yellow	14
LightGray	7	White	15

الألوان و رموزها بلغة الباسكال.

الإجراء TextColor (Color)

يقوم هذا الإجراء بتغيير لون المحارف إلى اللون المُحدَد بـ Color. بشكل مشابه لإجراء تغيير لون خلفية الشاشة، يُمكن أن نضع مكان الوسيط Color إما ثابت أو القيمة العددية المقابلة لهذا الثابت. مثلاً: عندما نكتب التعليمة TextColor (Blue) فإن لون المحارف يصبح أزرق.

الإجراءات Delay (DelayTime) و KeyPressed

يُفيد الإجراء Delay(DelayTime) بتعريف دالة انتظار زمني قبل استئناف تنفيذ الأوامر. يُحدَد زمن الانتظار بالقيمة الممررة إلى الوسيط DelayTime و تُقاس بملي ثانية (1 ملي ثانية يساوي 1/1000 من الثانية).

يُعيد الإجراء KeyPressed القيمة المنطقية True عندما يقوم المُستخدِم بالضغط على أي مفتاح من لوحة المفاتيح.

مثال:

يقوم الإجراءان التاليان بإنشاء نافذة إدخال للبيانات.

```
Procedure Restore;
  Begin
     Window(1, 1, 80, 25); {كامل شاشة التنفيذ}
     TextBackground(Black); {إعطاء الخلفية لون أسود}
     TextColor(White);     {لون المحارف أبيض}
     ClrScr;
  End;
Procedure PopUpWindow(Prompt:string;WinSize,
       WinColor:integer);
    Begin
     Window(20, 10, 60, 15); {إنشاء نافذة داخل شاشة التنفيذ}
     TextBackground(WinColor);
     ClrScr;
     GoToXY(5, 3); {إذهب إلى السطر الثالث في النافذة الجديدة}
     Writeln(Prompt);
     Window(24, 14, 24+WinSize, 14);
     TextBackground(LightGray);
     TextColor(Black);
     Clrscr;
```

```
    Write('>');
  End;
```

يقوم الإجراء Restore بإعادة شاشة التنفيذ إلى حجمها الطبيعي و يجعل لون الخلفية أسود و لون المحارف أبيض. الإجراء PopUpWindow يرسم نافذتين داخل شاشة التنفيذ ضمن الأبعاد المحددة. تتوسط النافذة الأولى شاشة التنفيذ و لون خلفيتها يُحدَد في أثناء التنفيذ بقيمة اللون الممررة إلى الوسيط WinColor. النافذة الثانية تُرسَم ضمن النافذة الأولى وتُستخدَم لإدخال البيانات من رسالة تُعرَض على المستخدم بتمريرها إلى الوسيط Prompt.

خلفية النافذة الثانية ملونة بالرمادي الفاتح (LightGray) و لون المحارف أسود (Black). يتم تحديد عدد أعمدة النافذة الثانية من خلال الوسيط WinSize. مثلاً، لو طلبنا الإجراء PopUpWindow كما يلي:

```
PopUpWindow ('Check amount', 10, Blue);
```

تظهر لدينا الشاشة التالية:



3. وحدة الرسم البياني Graph unit

1.3 مفهوم الرسم على الحاسوب Computer Graphics Concept

الرسم على الحاسوب هو طريقة لإدخال وإخراج الأشكال والصور المرئية إلى الحاسوب، بما فيها إنشاء، معالجة وعرض الصور باستخدام الحاسوب. فالرسم

على الحاسوب هو حقل من حقول علم الحاسوب وهو سريع التوسع وله مصطلحات وطرق ومكونات مادية خاصة به، حيث تظهر باستمرار طرق حديثة لحل مشاكل جديدة ويتم تطوير التقانات بغية التوصل إلى أفضل أنظمة للرسوم سهلة الاستخدام حتى من قبل غير المتخصصين في مجال العمل على الحاسوب وخاصة الفنانين. البحث في هذا المجال مستمر لتحسين خوارزميات الرسم لجعل عملية عرض الرسوم أسرع.

2.3 روتينات إعداد وحدة الرسم Setting Graphics unit Routine

هناك بعض الروتينات (إجراءات أو دوال) التي تعتبر بمثابة أدوات حيث إنها لا تقوم بعمل أية رسومات ولكنها تعد وحدة الرسم وتهيئها لكي نتمكن من تنفيذ برامج الرسوم عليها، حيث أن بعضها مهم جداً ولابد من تضمينه في كل برنامج رسومي والبعض الأخر يتم استدعائه للقيام بمهام خاصة في بعض برامج الرسوم أو في بعض الحالات الخاصة.

إن كل هذه الروتينات جاهزة تقريباً وتخص نظام الرسوم المستخدم. فيما يلي نعرض بعض هذه الروتينات.

- الإجراء cleardevice

وصيغته العامة كما يلي Procedure Cleardevice;

يستخدم هذا الإجراء من أجل مسح شاشة الرسم البياني وذلك يملئها بلون خلفية الشاشة قيد الاستخدام، ويعيد المؤشر إلى الموقع (0,0)، الزاوية العليا اليسرى للشاشة، في الإحداثيات المطلقة.

مثال:

يبين البرنامج التالي كيفية استدعاء الإجراء Cleardevice بلغة باسكال حيث يرسم أولاً خطوط مستقيمة بصورة عشوائية بواسطة الإجراء lineto ويستمر إلى أن يضغط المبرمج أي مفتاح فيتم تنظيف الشاشة.

```
Program clrdevce;
uses crt,graph;
Var
   gd,gm : integer;
Begin
     detectgraph(gd,gm); {سيتم دراسة هذا الإجراء لاحقاً}
     initgraph(gd,gm,'c:\tp\gri');
     if(graphResult <> grOK) then
        Halt(1);
     Randomize;
     Repeat
       LineTo(Randome(200),Randome (200))
     Unit keyPressed;
     Cleardevice;
     Readlin;
     CloseGrahph;
End.
```

- الإجراء DetectGraph

وصيغته العامة كما يلي:

```
Procedure DetectGraph(Var GraphDriver,GraphMode:
        integer);
```

وهو يستكشف سواق الرسم (GD) قيد الاستخدام ونمط الرسم (GM) المناسب الذي يمكن أن تستخدم من قبل مهايء Adapter العرض المستخدم. القيم

المحصلة تعود في البارمترات GraphDriver و Graphmode. يمكن استخدام الإجراءات DetectGraph لفحص قيم (GD) و (GM) المستكشفة بصورة اتوماتيكية، فيما إذا كانت مناسبة أم لا، وإذا كانت مناسبة فيمكن تغيرها قبل أن تمرر إلى الإجراء InitGraph التالي.

- الإجراء Initgraph

وصيغته العامة كما يلي:

```
Procedure Initgraph(Var GraphDriver,GraphMode:
    integer; DriverPath:String);
```

يقوم هذا الإجراء بالإعداد الأولي لنظام الرسوم وللمكونات المادية لنمط الرسوم اتوماتيكاً. البارمتر Driverpath هو المسار الذي يستخدمه الإجراء للبحث عن الملفات المناسبة والتي يكون امتداداها BGI.

ولكي تسمح لنواة الرسم أن تختار أفضل نمط رسم يجب إسناد القيمة الثابتة في وحدة الرسم Detect للمتغير GraphDriver كما في المثال التالي:

```
GraphDrive:=Detect;
InitGraph(GraphDriver,Graphmode,'C:\TP\BGI');
```

يقوم هذا الإجراء-بالإعداد الأولي Initialize لنظام الرسوم وللمكونات المادية لنمط الرسوم أوتوماتيكياً. البارامتر DriverPath هو المسار الذي يستخدمه الإجراء للبحث عن الملفات المناسبة والتي يكون امتدادها BGI، ولكي تسمح لنواة الرسم أن تختار أحسن نمط رسوم متوفر يجب إسناد القيمة الثابتة في وحدة الرسم Detect للمتغير GraphDriver كما في المثال التالي:

```
GraphDriver:= Detect;
Initgraph(graphDriver,Graphmode,'C:\TP\BGI');
```

تستخدم العبارتين السابقتين بصورة عامة في بداية برامج الرسوم. إذا كانت قيمة GraphDriver صفراً في InitGraph، يتم استدعاء الإجراء

DetectGraph أوتوماتيكياً للحصول على أفضل سواقة رسم ونمط رسم بأعلى قوة تحليل. القيم المحصلة أوتوماتيكياً تعاد إلى المتغيرين GraphDriver و Graphmode. ولإلغاء هذه القيم المحصلة أوتوماتيكياً يتم ذلك بإسناد القيم المرغوب بها للسواقة ونمط الرسوم للمتغيرين GraphDriver و Graphmode. أما القيمة "C:\TP\BGI" فهي تمثل المسار الذي سيبحث فيه الإجراء عن الملفات ذات الامتداد BGI.

```
Program initgrf;
uses Graph;
var
   grDriver:Integer;
   grMode:Integer;
   ErrCode:Integer;
Begin
  grDriver:=Detect;
  InitGraph(grDriver, grMode,'');
  ErrCode:=GraphResult;
  If(ErrCode=grOk) then
    Begin  {Do graphics}
      Line(0,0,GetMaxX,GetMaxY);
      Readln;
      CloseGraph;
    End
  Else
    Writeln('Graphics error:',
  GraphErrorMsg(ErrCode));
  Readln;
End.
```

- الدالة GetmaxX

وصيغتها العامة كما يلي Function GetMaxX :integer;

تعيد هذه الدالة القيمة العظمى للإحداثي الأفقي (السيني) لطور الرسوم قيد الاستخدام حيث x من النوع الصحيح.

مثال:

```
x := getmaxX;
```

- الدالة GetmaxY

وصيغتها العامة كما يلي Function GetMaxY :integer;

تعيد هذه الدالة القيمة العظمى للإحداثي العمودي (الصادي) لطور الرسوم قيد الاستخدام حيث y من النوع الصحيح.

مثال:

```
y := getmaxY;
```

يمكن استخدام الدالتين السابقتين لتحديد أي موقع على شاشة الرسم بدلاً من إعطاء قيم ثابتة للموقع، وهذا يساعد على تنفيذ البرامج على أي جهاز دون معرفة مسبقة لنظام إحداثياته. فنقطة وسط الشاشة لأي نظام إحداثيات بلغة باسكال هي:

```
(getMaxX div 2, getMaxY div 2)
```

البرنامج التالي بلغة الباسكال يستدعي كل من الإجرائيتين getmaxx و getmaxy.

```
program getmxx;
 {Sample code for the GetMaxX function.}
uses Graph;
var
   Gd, Gm: Integer;
Begin
  Gd:=Detect;
```

```
  InitGraph(Gd, Gm,'c:\TP\BGI');
  If(GraphResult <> grOk) then
     Halt(1);
  Rectangle(0, 0, GetMaxX, GetMaxY);  {Draw a
                          full-screen box}

  Readln;
  CloseGraph;
End.
```

- الدالة Getx

وصيغتها العامة كما يلي Function GetX :integer;

تعيد هذه الدالة قيمة الإحداثي الأفقي (السيني) لطور الرسوم قيد الاستخدام حيث x من النوع الصحيح.

مثال:

```
x := getX;
```

- الدالة Gety

وصيغتها العامة كما يلي Function GetY :integer;

تعيد هذه الدالة قيمة الإحداثي العمودي (الصادي) لطور الرسوم قيد الاستخدام حيث y من النوع الصحيح.

مثال:

```
y := getY;
```

البرنامج التالي بلغة الباسكال يستدعي كل من الإجرائيتين getx و gety.

```
program getx;
 {Sample code for the GetX function.}
uses Graph;
var
  Gd,Gm: Integer;
```

```
  X,Y: Integer;
Begin
  Gd:=Detect;
  InitGraph(Gd, Gm,'c:\TP\BGI');
  If(GraphResult <> grOk) then
     Halt(1);
  OutText('Starting here. ');
  X:=GetX;
  Y:=GetY;
  OutTextXY(20, 10, 'Now over here...');
  OutTextXY(X, Y, 'Now back over here.');
  Readln;
  CloseGraph;
End.
```

3.3 رسم النقطة

العنصر الأساسي لرسم أية صورة على شاشة الرسم هو النقطة الملونة Pixel وهي أصغر وحدة عرض حيث تتكون كل شاشات العرض البياني من عدد كبير من الـ Pixel مرتبة بشكل خطوط أفقية وعمودية.

في بيئة الرسم CGA تكون النقطة كبيرة نسبياً، لهذا فإن هناك 320 نقطة أفقياً و 200 نقطة عمودياً (أي 200x320).

أما بيئة الرسم VGA ذو قوة التحليل العالية جداً ونقاط صغيرة جداً (480x640). بالطبع كلما كانت النقاط صغيرة ازداد عددها للصورة الواحدة وبالتالي فإن ذلك يعني الحصول على صور أفضل.

البرنامج التالي يرسم 600 نقطة مبعثرة عشوائياً على الشاشة.

```
program putpix;
uses crt,Graph;
var
```

```
   i,graphmode,graphDriver:integer;
   Color:word;
begin
   detectgraph(graphmode,graphdriver);
   initgraph(graphmode, graphdriver,'c:\tp\bgi');
   if(graphresult <> grOK) then
      halt(1);
   color:=getMaxcolor;
   Randomize;
   for i:=1 to 600 do
     putpixel(Random(getmaxx),
            random(getmaxy),color);
   readln;
   closegraph;
end.
```

البرنامج التالي بلغة الباسكال يستدعي كل من الإجرائيتين getmasx و getmaxy.

```
program getmxx;
{Sample code for the GetMaxX function.}
uses Graph;
var Gd,Gm: integer;
begin
  Gd:=Detect;
  InitGraph(Gd,Gm,'c:\TP\BGI');
  if (GraphResult <> grOk) then
    Halt(1);
   {Draw a full-screen box}
  Rectangle(0,0,GetMaxX,GetMaxY);
  Readln;
  CloseGraph;
```

```
end.
```

البرنامج التالي بلغة باسكال يستدعي كل من الإجرائيتين getx و gety.

```
program getx;
{Sample code for the GetX function.}
uses Graph;
var
  Gd,Gm: integer;
  X,Y: integer;
Begin
  Gd:=Detect;
  InitGraph(Gd,Gm,'c:\TP\BGI');
  If(GraphResult <> grOk) then
    Halt(1);
  OutText('Starting here. ');
  X:=GetX;
  Y:=GetY;
  OutTextXY(20,10,'Now over here...');
  OutTextXY(X,Y,'Now back over here. ');
  Readln;
  CloseGraph;
End.
```

عند تنفيذ البرنامج السابق ستظهر العبارة النصية في الموقع (319,239) على شاشة الرسم.

توفر وحدة الرسم البياني مجموعة من الدوال و الإجرائيات المُسبقة التعريف و التي تُمكِّن من رسم مجموعة من الأشكال الهندسية.

يُمكن تصور شاشة التنفيذ على أنها مجموعة من النقاط الملونة ندعوها بيكسل Pixels موزعة على شكل شبكة Grid. بشكل خاص يمكن النظر إلى شاشة التنفيذ على أنها شبكة بمحور أفقي X و محور شاقولي Y. مثلاً أبعاد بعض

الشاشات 480 X 640 تعني أن هناك 640 نقطة أفقية (X) و 480 نقطة شاقولية (Y).

يُمكن دائماً معرفة أبعاد الشاشة الأفقية و العمودية من خلال الدالتين: GetMaxX و التي تُعيد أكبر عدد من النقاط الأفقية للشاشة و GetMaxY و التي تُعيد أكبر عدد من النقاط العمودية للشاشة.

بشكل مختلف عن الوحدات التي درسناها حتى الآن فإنه يجب تهيئة وحدة الرسم البياني قبل إمكانية استخدامها في أي مكان في البرنامج و قبل استخدام إجراءات هذه الوحدة.

يتم تهيئة وحدة الرسم البياني بوساطة التعليمة التالية:

```
InitGraph(Driver,Mode,Directory);
```

الشيء الوحيد الذي يجب أن نُحدِدَه في هذه التعليمة هو الوسيط Directory و الذي يُحدِد الموقع المُخزَّنة عليه وحدة الرسم البياني في أثناء تنصيب لغة الباسكال. عادةً يكون المجلد Directory الذي يحوي وحدة الرسم هو: "C:\BP\BGI" أو أحياناً "C:\TP\BGI".

بعد تهيئة وحدة الرسم البياني ، يُمكن و بسهولة استخدام كافة الدوال و الإجراءات المتوفرة فيها. سنَعرض في الفقرات التالية مجموعة من الدوال و الإجرائيات الخاصة بوحدة الرسم البياني.

الإجراءان SetColor و SetBKColor

يقوم الإجراء SetColor(Color) بتغيير لون المحارف بحسب قيمة اللون الممرر إلى الوسيط Color. بشكل مماثل، يقوم الإجراء SetBKColor(Color) بتغيير لون خلفية الشاشة إلى قيمة اللون المُمرَر إلى الوسيط Color. رموز الألوان معرفة في الجدول (1) أعلاه. فمثلاً، عند كتابة

الأمر SetColor (Green) فإن لون المحارف في شاشة التنفيذ تصبح أخضر.

أما الأمر SetBKColor (Red) يجعل خلفية شاشة التنفيذ حمراء.

4.3 رسم خطوط Drawing Lines

تُمكننا التعليمة Line من رسم خط مستقيم بين نقطتين مُحددتين على الشاشة. التركيب النحوي لهذا الأمر هو Line (X1, Y1, X2, Y2). هذا الأمر يقوم برسم مستقيم من النقطة (X1, Y1) إلى النقطة (X2, Y2).

مثال:

يقوم البرنامج التالي برسم مستقيمين قطريين في شاشة التنفيذ.

```
Program IntersectLine;
Uses Graph; {استدعاء مكتبة الرسم البياني}
Const
      {المسار الذي يحوي وحدة الرسم البياني}
      Directory='C:\BP\BGI';
Var
      Driver,Mode:integer;
Begin
      {تهيئة وحدة الرسم البياني}
      InitGraph (Driver, Mode, Directory);
      {رسم مستقيم واصل بين النقطتين المحددتين}
      Line (0,0,GetMaxX,GetMaxY);
      SetColor(LightGray); {تغيير لون الخط إلى رمادي فاتح}
      Line(GetMaxX,0,0,GetMaxY);
End.
```

يبدأ البرنامج العام بتجهيز وحدة الرسم البياني بوساطة التعليمة InitGraph. الوسيط المهم هو Directory و الذي يُحدِد موضع وحدة الرسم

البياني في القرص. لاحظ أننا افترضنا وجود الوحدة على المسار C:\BP\BGI ، و لكن إذا كانت الوحدة موجودة على مسار مختلف على الجهاز فيجب تغيير المسار إلى المسار المناسب.

بعد تهيئة وحدة الرسم، ثم كتابة أمر رسم مستقيم أول من الزاوية العليا اليسارية (0,0) إلى الزاوية السفلية اليمينية (GetMaxX, GetMaxY).

غيرنا لون الخط بوساطة SetColor و من ثم رسمنا مستقيماً آخر من الزاوية اليمينية العليا (GetMaxX, 0) إلى الزاوية اليسارية السفلية (0, GetMaxY).

5.3 رسم مستطيل Drawing Rectangle

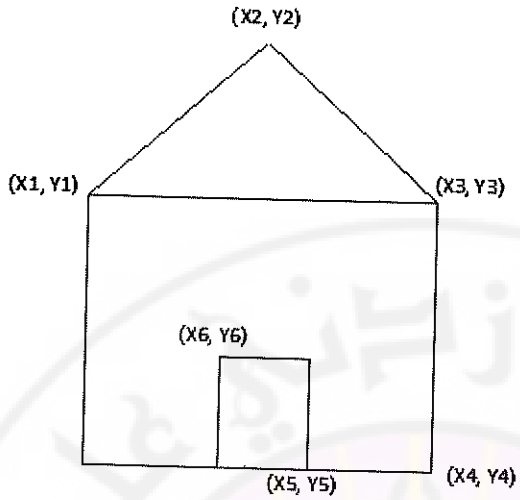
تُمكننا لغة الباسكال من رسم أي مستطيل و ذلك من خلال التعليمة Rectangle. التركيب النحوي لهذه التعليمة هي:

```
Rectangle(X1,Y1,X2,Y2);
```

تقوم هذه التعليمة برسم مستطيل بين النقطتين (X1, Y1) و (X2, Y2). في الواقع يتم رسم المستطيل من خلال رسم خط قطري بين النقطتين المذكورتين و من ثم إغلاق الأضلاع الأربعة.

مثال:

سوف نقوم بهذا المثال برسم الشكل المبين أدناه. وُضِعت النقاط على بعض رؤوس الشكل و ذلك لسهولة الربط مع البرنامج و تبسيط الفهم.




```
Program House;
Uses Graph;
Const
      {اسم المسار الذي يحوي وحدة الرسم}
      Directory='C:\BP\BGI';
Var
      Driver, Mode,
      X1, Y1, X2, Y2, X3, Y3,
      X4, Y4, X5, Y5, X6, Y6 :integer;
Begin {House}
      {تهيئة وحدة الرسم البياني}
      InitGraph(Driver,Mode,Directory);
      {تعريف زوايا الشكل}
      X1:=100;
      Y1:=200; {هذه النقاط اختيارية ويمكن للمستخدم تغيرها}
      X2:=300; Y2:=100;
      X3:=500; Y3:=200;
      X4:=500; Y4:=400;
```

```
    X5:=325; Y5:=400;
    X6:=275; Y6:=325;
    {رسم سطح الشكل}
    Line(X1,Y1,X2,Y2);
    Line(X2,Y2,X3,Y3);
    {رسم المستطيل الكبير}
    Rectangle(X1,Y1,X4,Y4);
    {رسم المستطيل الداخلي}
    Rectangle(X5,Y5,X6,Y6)
End. {House}
```

نلاحظ أن فكرة البرنامج بسيطة حيث أنه يستخدم فقط إجراءان لرسم مستقيم و مستطيل. ربما الصعوبة الوحيدة هي في تحديد موقع النقاط و هذا يحتاج إلى فكر هندسي بسيط و ليس برمجياً. في الواقع ، ننصح القارئ بمحاولة التدريب على تغيير النقاط و رسم الشكل نفسه و لكن بمواقع مختلفة و أحجام مختلفة و ذلك بغية التمرين.

6.3 رسم دوائر و أقواس Drawing Circles and Arcs

توفر لغة الباسكال الإجراء Circle لرسم دائرة. التركيب النحوية لهذا الإجراء هو:

```
Circle(X,Y,Radius);
```

حيث (X, Y) تُمَثل إحداثيات مركز الدائرة و Radius هو نصف القطر.

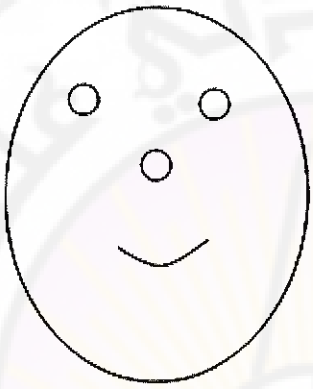
يُمَّكِن الإجراء Arc من رسم قوس من دائرة. لرسم قوس دائرة، لابد من تحديد مركز القوس، نصف قطره، زاوية البداية و زاوية النهاية. يجب أن تُحدَد زاوية البداية و زاوية النهاية **بالدرجات**. التركيب النحوية لهذا الأمر هو:

```
Arc (X, Y, SAngle, EAngle, Radius);
```

حيث (X, Y) هي مركز القوس، SAngle زاوية البداية، EAngle زاوية النهاية، و Radius هو نصف القطر. مثلاً: التعليمة Arc (X, Y, 0, 180, Radius) ترسم قوساً يُمثل نصف دائرة مركزها النقطة (X, Y) و نصف قطرها Radius.

مثال:

سوف نكتب برنامج لرسم الشكل التالي:



لاحظ أن هذا الشكل بسيط فهو عبارة عن مجموعة من الدوائر و قوس.

ربما الصعوبة الوحيدة هي فقط كيفية تحديد النقاط و الزوايا.

```
Program HappyFace;
Uses Graph;
Const
     {اسم المسار الذي يحوي وحدة الرسم}
     Directory='C:\BP\BGI';
Var
     Driver, Mode,
     MidX, MidY, LeftEye, RightEye, EyeY,
     NoseX, NoseY, HeadRadius, EyeNoseRedius,
          SmileRadius, StepX, StepY : Integer;
Begin {HappyFace}
     {تهيئة وحدة الرسم البياني}
```

```
    InitGraph(Driver,Mode,Directory);
    {رسم الرأس}
    MidX:=GetMaxX div 2;
    MidY:=GetMaxY div 2;
    HeadRadius:=GetMaxY div 4;
    Circle(MidX,MidY,HeadRadius);
    {رسم العيون}
    StepX:=HeadRadius div 4;
    StepY:=StepX;
    LeftEyeX:=MidX-StepX;
    RightEyeX:=MidX+StepX;
    EyeNoseRadius:=HeadRadius div 100;
    Circle(LeftEyeX,EyeY,EyeNoseRadius);
    Circle(RightEyeX,EyeY,EyeNoseRadius);
    {رسم الأنف}
    NoseX:=MidX;
    NoseY:=MidY+StepY;
    Circle(NoseX,NoseY,EyeNoseRadius);
    {رسم الفم}
    SmileRadius:=round(0.75* HeadRadius);
    Arc(MidX,MidY,210,330,SmileRadius);
End. {HappyFace}
```

7.3 بعض الدوال و الإجرائيات الأخرى

1. الدالة GraphErrorMsg

وصيغته العامة كما يلي:

```
Function GraphErrorMsg (Code : Intger) : String;
```

تعيد هذه الدالة رسالة خطأ تتناسب مع حالة الخطأ المبينة في البارمتر Code على شكل قيمة صحيحة. القيم التي يمكن تمريرها إلى GraphErrormsg محددة بالقيم من 0 إلى 15 الموافقة للعبارات المبينة في الجدول التالي:

const	
grOK	0
grNoInitGraph	1
grNotDetect	2
grFileNotFound	3
grInvaliadDriver	4
GrNoLoadMem	5
grnoScanMem	6
grNoFloodMem	7
grFontNotFound	8
grNoFunctionMem	9
grInvalidMode	10
grEror	11
grIOError	12
grInvalidFont	13
grInvalidFontNum	14
GrInvalidDeviceNum	15

يبين البرنامج التالي كيفية استخدام هذه الدالة.

```
program grferr;
uses Graph;
```

```
var
  GraphDriver,GraphMode:Integer;
  ErrorCode:Integer;
Begin
  GraphDriver:=Detect;
  InitGraph(GraphDriver,GraphMode,'');
  ErrorCode:=GraphResult;
  If(ErrorCode <> grOk) then
    Begin
       Writeln('Graphics error: ',
       GraphErrorMsg(ErrorCode));
       Readln;
       Halt(1);
    End;
  Line(0, 0, GetMaxX, GetMaxY);
  Readln;
  CloseGraph;
End.
```

2. الدالة GraphResult

وصيغتها العامة كما يلي:

```
Function GraphResult : integer;
```

هذه الدالة للتدقيق فيما لو حصل أي خطأ رسومي في وقت تنفيذ البرنامج، القيمة العددية التي ستعيدها هذه الدالة محددة بالقيم من 0 إلى 15.

ويمكن ترجمة هذه القيمة العددية إلى رسالة خطأ نصية تصف نوع الخطأ وذلك باستخدام الدالة GraphErrorMsg. يتم إسناد القيمة صفر إلى الدالة GraphResalt حالما يتم استدعائها، ولهذا يجب تخزين أي قيمة نحصل عليها من هذه الدالة قبل أن يتم استدعائها مرة أخرى إذا دعت الحاجة لذلك فيما بعد.

جزء البرنامج التالي يستكشف الأخطاء التي تحصل أثناء تنفيذ البرنامج ويطبع النتيجة على الشاشة بشكل قيمة عددية ورسالة نصية توضح نوع الخطأ. يمكن تضمين هذا البرنامج كجزء من أي برنامج رسوم على شكل روتين لفحص الأخطاء الرسومية، مثل فقدان أي من الملفات ذات الامتداد BGI، أو لبيان طقم الحروف المطبعية الغير متوفرة وغيرها، أي معلومات متعلقة بسواقة الرسوم ستعرض أيضاً.

```
program grfref;
uses Graph;
var
  GraphDriver,GraphMode:Integer;
  ErrorCode:Integer;
Begin
  GraphDriver := Detect;
  InitGraph(GraphDriver, GraphMode, 'c:\TP\BGI');
  ErrorCode := GraphResult;  {cherck for errors}
  If (ErrorCode <> grOk) then
  Begin
    Writeln('Graphics error: ');
    Writeln(GraphErrorMsg(ErrorCode));
    Writeln('Program aborted');
    Readln;
    Halt(1);
  End;
  clearDevice;  {Do some Graphics ...}
  Rectangle(200,200,getMaxX,getmaxY);
  Readln;
  CloseGraph;
End.
```

3. الإجراء SetGraphMode

وصيغته العامة كما يلي:

```
Procedure SetGraphMode (Mode : integer);
```

يعمل هذا الإجراء على إعداد نمط الرسوم قيد الاستخدام إلى النمط المخصص في البارامتر Mode، ثم ينظف شاشة العرض. يمكن الاستفادة من هذا الإجراء لإلغاء قيمة نمط الرسوم المعدة من قبل الإجراء InitGraph.

البرنامج التالي بلغة باسكال يوضح كيفية استدعاء هذا الإجراء.

```
program setgrfmod;
uses Graph;
var
  GD,GM:Integer;
  Code:Integer;
Begin
  GD:=Detect;
  InitGraph(GD,GM,'c:\TP\BGI');
  Code:=GraphResult;  {cherck for errors}
  If(Code <> grOk) then
     Begin
        Halt(1);
     End;
  outtext('<Enter> to enter graphics mode');
  Readln;
  setgraphMode(getGraphmode);
  outTextXY(0,0,'Back in Grapgics mode');
  outTextXY(0,TextHeight('H'),'<Enter> to quit');
  Readln;
  CloseGraph;
End.
```

4. الدالة getGraphmode

وصيغتها العامة كما يلي:

```
Function getGraphmode : word;
```

تعيد هذه الدالة القيمة العددية لنمط الرسوم قيد الاستخدام في الحاسوب والتي تم إعدادها باستدعاء سابق لأحد الإجرائيتين SetGraphmode أو InitGraph. القيمة العددية لنمط الرسوم يجب أن تترجم بالاتحاد مع سواق الرسوم وهي تتراوح بين 0 و 5.

يبين البرنامج التالي كيفية استخدام الإجراء SetGraphMode والدالة GetGraphmode.

```
program grfmd;
uses Graph;
var
  GD,GM:Integer;
  Code:Integer;
Begin
  GD:=Detect;
  InitGraph(GD, GM,'c:\TP\BGI');
  Code:= GraphResult;  {cherck for errors}
  If (Code <> grOk) then
     Begin
       Halt(1);
     End;
  outtext('<Enter> to leave graphic mode.');
  Readln;
  restoreCrtMode;
  writeln('now in text mode');
  write('<enter> to enter graphic mode');
```

```
  Readln;
  setgraphMode(getGraphmode);
  outTextXY(0,0,'Back in Grapgics mode');
  outTextXY(0,TextHeight('H'),'<Enter> to quit');
  Readln;
  CloseGraph;
End.
```

5. الدالة getDrivername

والصيغة العامة كالتالي:

```
Function getDriverName: string
```

تعيد هذه الدالة نصاً يصف اسم سواق الرسوم graphDriver. يمكن استدعاء هذه الدالى بعد استدعاء الإجراء InitGraph.

```
program GTDRVNM;
uses Graph;
var
  Driver,mode:Integer;
  x,y:Integer;
Begin
  Driver:=Detect; {put in graphic mode}
  InitGraph(Driver,mode, 'c:\TP\BGI');
  If(graphresult <> grOk) then
     Halt(1);
  x:=getmaxx div 2;
  y:=getmaxy div 2;
  outtextxy(x,y,'Using driver' + getDrivername);
  Readln;
  closegraph;
End.
```

خرج البرنامج هي العبارة التالية في وسط الشاشة:

using driver EGAVGA

يعيد هذا البرنامج اسم سواقة الرسم المستخدمة في وسط الشاشة كما يلي:

EGAVGA

6. الدالة Getmodename

وصيغتها العامة كما يلي:

function getmodename(modenumber:word) : string;

تعيد هذه الدالة نص يصف نمط الرسوم البينية قيد الاستخدام والمعطى رقمه في البارامتر modeNumber، أي إنها تقبل قيمة صحيحة وتعيد قيمة على شكل سلسة والتي هي في لغة C++ مؤشر (pointer) إلى أول عنصر في السلسلة.

البرنامج التالي بلغة الباسكال يستدعي هذه الدالة:

```
program GTMDNM;
uses Graph;
var
  Driver,mode:Integer;
  I:Integer;
Begin
  Driver:=Detect; {put in graphic mode}
  InitGraph(Driver,mode, 'c:\TP\BGI');
  If(graphresult < 0) then
    Halt(1);
  For I:=0 to getmaxmode do {Desplay all mode
                             names}
     outtextxy(10,10*Succ(1),getmodename(I));
  Readln;
```

```
  closegraph;
End.
```

خرج البرنامج على الشكل التالي:

640x200 EGA
640x350 EGA
640x480 VGA

7. الإجراء getmodeRange

وصيغته العامة كما يلي:

```
procedure getModeRange(graphDriver :integer;var
lowmode, Himode:integer);
```

يعيد هذا الإجراء أعلى قيمة Himode وأقل قيمة Lowmode (المدى) لأطوار التحليل لسواقة الرسوم البيانية المخصص في المتغير GraphDriver. فمثلاً السواقة EGA64 ستعطي القيمة 0 للبارامتر Lowmode والقيمة 1 للبارامتر Himode عند استخدام هذا الإجراء.

البرنامج التالي بلغة الباسكال يعيد أعلى وأقل قيمة لطور التحليل لسواقة الرسوم المستخدمة.

```
program gmdrng;
uses Graph;
var
   lowest,Highest:Integer;
Begin
     getModerange(EGA64,lowest,highest);
     write('lowest =',lowest);
     write('hight = ',highest);
     Readln;
End.
```

كان خرج البرنامج كما يلي:

lowest =0 Highest = 1

8. الإجراء GraphDefaults

وصيغته العامة كما يلي:

```
procedure GraphDefaults;
```

يحدد إعدادات الرسم البياني إلى الحالة التي تلي استدعاء الإجراء InitGraph مباشرة.

9. الإجراء RestoreCRTMode

والصيغته العامة في لغة الباسكال كما يلي:

```
procedure RestoreCRTMode;
```

يعيد هذا الإجراء خزن نمط العرض الفيديوي المستخدم سابقاً قبل الإعدادات الابتدائية لنمط الرسوم. هذا الإجراء يفيد لعودة إلى نمط النص بعد أن يتم استخدام نمط الرسوم على فرض أن نمط النص قد تم استخدامه مسبقاً وقبل نمط الرسوم.

البرنامج التالي بلغة الباسكال يستدعي هذا الإجراء ليتحول من نمط الرسم إلى نمط الكتابة وبالعكس.

```
program restocrtmd;
uses Graph;
var
  GD,GM:Integer;
  Mode:Integer;
Begin
  GD:=Detect;
  InitGraph(GD, GM,'c:\TP\BGI');
  If(graphresult <> grOk) then
    Begin
```

```
      Halt(1);
    End;
  Rectangle(200,200,300,300);
  outtextxy(100, 100 ,
     '<enter> to leave  graphics mode:');
  Readln;
  writeln('Now in text mode');
  write('<Enter> to enter graphic mode:');
  Readln;
  setGraphmode(getgraphmode);
  outtextxy(20,50,'Back in graphic mode');
  bar(120,120,220,220);
  outtextxy(20,textheight('H')+20,
       '<Enter> to quit');
  Readln;
  CloseGraph;
End.
```

خرج البرنامج له الشكل التالي:

<Enter> to leave graphics:



وعند الضغط على المفتاح Enter تظهر الرسالة التالية:

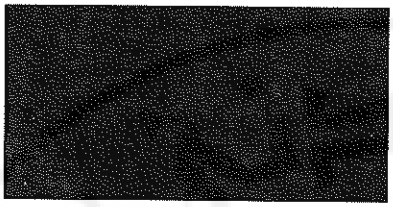
Now in text mode

<Enter> to enter graphic mode:

وعند الضغط على المفتاح Enter مرة أخرى تظهر شاشة الرسم كما يلي:

<Enter> to quit:

Back in graphic mode



10. الإجراء setVisualPage

وصيغته العامة كما يلي:

```
procedure setVisualPage(Page : word);
```

يختار هذا الإجراء صفحة الرسم البياني المحدد رقمها في البارامتر Page والتي هي من النوع الصحيح لعرضها.

11. الإجراء setActivePage

وصيغته العامة كما يلي:

```
procedure setActivePage(Page : word);
```

يحدد هذا الإجراء صفحة الرسم الفعالة (لمهايء العرض الذي يسند عدة صفحات) والتي سيظهر عليها الخرج بالصفحة المخصصة في البارامتر Page. أي خرج رسومي ستوجه إلى هذه الصفحة.

البرنامج التالي بلغة الباسكال يستدعي كل من الإجرائيتين SetActivePage و setvisualPage السابقتين:

```
program setactvisuapg;
uses Graph;
var
```

```
  GD,GM:Integer;
Begin
  GD:=Detect;
  InitGraph(GD, GM,'c:\TP\BGI');
  If(graphresult <> grOk) then
    Begin
       Halt(1);
    End;
  If((GD= hercmono) or (GD=EGA) or
      (GD= EGA64) or (Gd=VGA)) then
    Begin
       setvisualPage(0);
       setActivePage(1);
       Rectangle(100,100,200,200);
       setvisualPage(1);
    End
  Else
    outTextxy(40,40,'No Paging supported');
    Readln;
  CloseGraph;
End.
```

12. الإجراء setGraphBuffSize

وصيغته العامة كما يلي:

```
procedure setGraphBuffSize (BuffSize : word);
```

يعمل هذا الإجراء على تحديد حجم خزان الشكل.

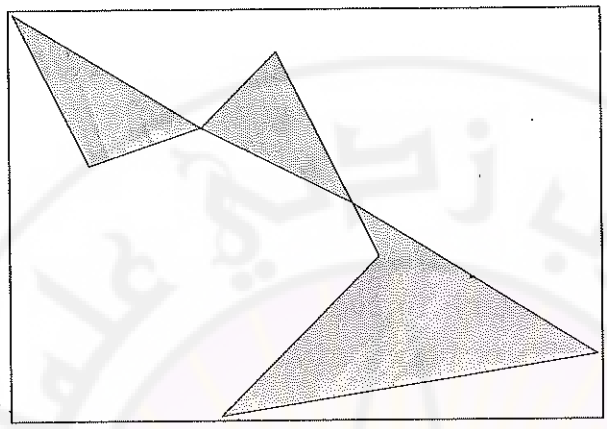
يستدعي البرنامج التالي بلغة الباسكال الإجراء setgraphbuffsize.

```
program setgbfsz;
uses Graph;
```

```
const
  BigPoly:array[1..7] of PointType =
   ((x : 50 ; y : 100),
   (x : 100; y : 200),
   (x : 150; y : 180),
   (x : 200; y : 140),
   (x : 250; y : 250),
   (x : 180; y : 350),
   (x : 350; y : 290));

var
  GD,GM:Integer;
Begin
     {Allocate 1/4 k of buffer space so we can
       fill a 7-vector polygon}
   setgraphbufsize(1024 div 4);
   GD:=Detect;
   InitGraph(GD, GM,'c:\TP\BGI');
   If(graphresult <> grOk) then
     Begin
       Halt(1);
     End;
   fillPoly(sizeof(BigPoly) div
       sizeof(PointType),BigPoly);
   Readln;
   CloseGraph;
End.
```

لاحظ أننا حددنا خزان الشكل بالحجم $1/4k$ لتخزين نقاط الشكل السبعة، حيث تم رسم الشكل بواسطة الإجراء FillPoly والذي يرسم الشكل متعدد الأضلاع التالي ويملأه.



13. الدالة RegisterBGIDriver

وصيغتها العامة كما يلي:

```
procedure RegisterBGIDriver (Driver : pointer) : integer;
```

تقوم هذه الدالة بتسجيل السواقة BGI مع نواة الرسم، المؤشر Driver يخبر نواة الرسم عن مكان وجود السواقة. عندما يحصل خطأ، فإن هذه الدالة ستعيد قيمة سالبة. يمكن استخدام الدالة GraphErrorMsg والدالة GraphResult لعرض الخطأ في هذه الحالة. الدالة RegisterBGIDriver مفيدة في حالة التحميل المسبق لسواقات رسوم مختلفة في الذاكرة وبذلك نستغرق أقل من وقت التبديل من سواقة إلى أخرى. حالما يتم تسجيل السواقة فسيتم استخدامها من قبل الإجراء Initgraph وذلك بتمرير رقم السواقة له.

```
program rbgid;
uses Graph;
var
```

```
  Driver,mode:Integer;
  DriverF: file;
  DriverP:pointer;
Begin
    {dirctvidio:= false;}
  Driver:= Detect; {put in graphics mode}
  {open driver file , read,register}
  Assign(DriverF,'c:\CGA.BGI');
  Reset(DriverF,1);
  getmem(DriverP,Filesize(DriverF));
  blockRead(driverF,DriverP^,Filesize(DriverF));
  If(registerBGIdriver(DriverP) < 0) then
    Begin
      writeln('Error registering driver.',
         Grapherrormsg(graphresult));
      halt(1);
    End;
  {init graphics}
  Driver:=CGA;
  mode:=CGAHI;
  initgraph(Driver,mode,'c:\tc\bgi');
  If(graphresult < 0) then
     Halt(1);
  outtext('Driver loaded by user program');
  Readln;
  CloseGraph;
End.
```

البرنامج التالي مكمل للبرنامج السابق ويعمل على فتح ملف السواقة DriverF وقراءته وتسجيله.

```
program loaddriv;
```

```
uses Graph;
var
  Driver,mode:Integer;
  DriverF:file;
  DriverP:pointer;
Begin
  {open driver file , read,register}
  Assign(DriverF,'c:\CGA.BGI');
  Reset(DriverF,1);
  getmem(DriverP,Filesize(DriverF));
  blockRead(driverF,DriverP^,Filesize(DriverF));
  If(registerBGIdriver(DriverP) < 0) then
    Begin
       writeln('Error registering driver.',
          Grapherrormsg(graphresult));
       Halt(1);
    End;
  {init graphics}
  Driver:=CGA;
  mode:=CGAHI;
  initgraph(Driver,mode,'c:\tc\bgi');
  If(graphresult < 0) then
     halt(1);
  outtext('Driver loaded by user program');
  Readln;
  CloseGraph;
End.
```

14. الإجراء SetAspectRatio

وصيغته العامة كما يلي:

procedure SetAspectRatio (Xasp,Yasp:word);

يغير هذا الإجراء نسبة الهيئة التي تستخدم لعرض الرسومات البيانية والتي تساوي Xasp/Yasp.

15. الإجراء getaspectration

وصيغته العامة كما يلي:

procedure GetAspectRatio (var Xasp,Yasp:word);

يعيد هذا الإجراء قيم في البارمترات Xasp و Yasp والتي يمكن من خلالها حساب قوة التحليل المؤثرة لشاشة العرض ويسمى أحياناً عامل التصحيح أو نسبة الهيئة من قسمة Xasp على Yasp.

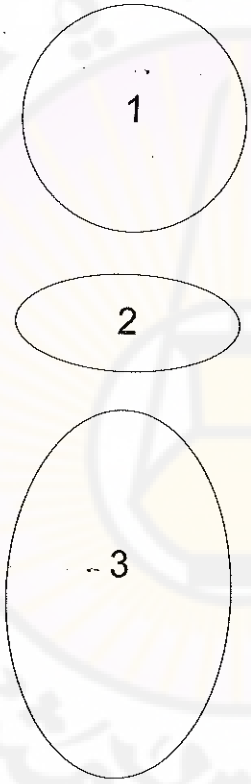
البرنامج التالي بلغة الباسكال يستدعي كل من الإجرائيتين SetAspectRation، GetAspectRatio.

```
program gtaspctratio;
uses crt ,Graph;
const
  R=50;
var
  Driver,mode: Integer;
  Xasp,Yasp:Word;
Begin
  {dirctvidio:=false;}
  Driver:=Detect; {put in graphics mode}
  initgraph(Driver,mode,'c:\tc\bgi');
  If(graphresult < 0) then
     Halt(1);
  getaspectRatio(Xasp,Yasp); {get default aspect
                              ratio}
  If(Xasp=Yasp) then
```

```
{adjast for VGA and 8514 they have 1:1 aspect}
  Yasp:= 5 * Xasp;
While((Xasp < Yasp) and not keypressed) do
{keep modifing aspect ratio until 1:1 or key is
  pressed}
  Begin
     setAspectratio (Xasp,Yasp);
     Circle(getMaxx div 2,getmaxy div 2, R);
     inc(Xasp, 20);
   End;
 setTextjustify(Centertext,CenterText);
 outTextxy(getmaxx div 2,
     getmaxy div 2,'Done!');
 Readln;
 CloseGraph;
End.
```

Damascus University

أما خرج البرنامج فهو كما في الشكل التالي حيث تظهر أولاً الدائرة المنتظمة رقم 1 على الشاشة حسب نسبة الهيئة الاعتيادية (الاعتباطية) للجهاز المستخدم، وعند ضغط المفتاح Enter يظهر الشكل الاهليجي رقم 2 حيث تم تغير نسبة الهيئة بقسمة Xasp على 2 أي بنصف قوة التحليل على المحور السيني، وعند الضغط على المفتاح Enter يظهر الشكل الاهليجي رقم 3 حيث تم تغير نسبة الهيئة بتقسيم Yasp على 2 أي بنصف قوة التحليل على المحور الصادي.



16. الدالة InstallUserDriver

وصيغتها العامة كما يلي:

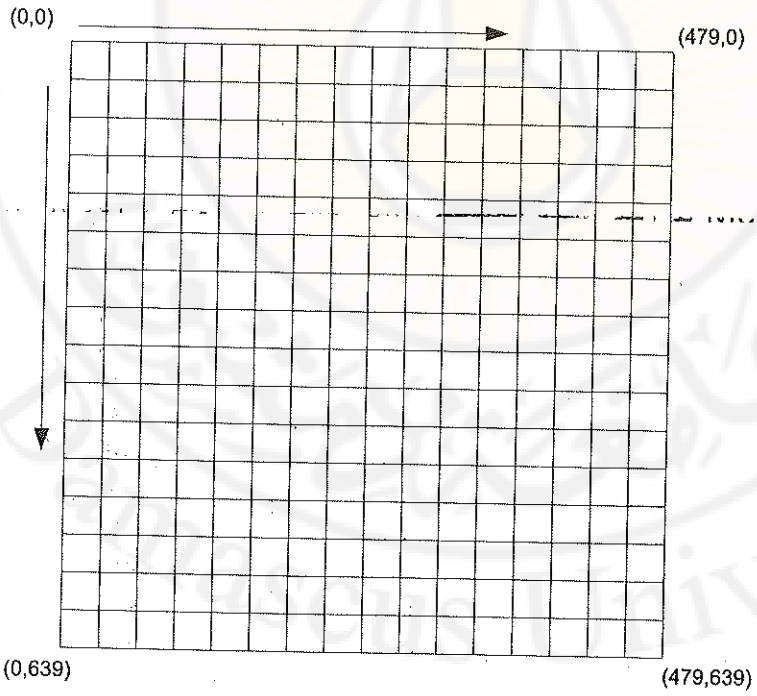
```
Function InstallUserDriver(Name:string; AutoDetectPtr:
 pointer): integer;
```

تسمح هذه الدالة للمستخدم بتركيب جهاز سواقة جديد. الدالة تعيد رقم السواقة المسند إلى هذه السواقة الجديدة، البارميتر name هو اسم الملف للسواقة الجديدة. البارمتر AutoDetectPtr هو مؤشر الدالة detetct الأتوماتيكي للجهاز الجديد، إذا كان هناك واحداً. إذا حصل أي خطأ عند تركيب السواقة الجديدة، فإن هذه الدالة ستعيد القيمة 11-.

```
program instudrv;
  {Sample code for the InstallUserDriver
    function.}
 uses Graph;
var
  Driver, Mode,
  TestDriver,
  ErrCode:Integer;
{$F+}
 Function TestDetect:Integer;
   {Autodetect function: assume hardware is
    Always present; return value = recommended
    default mode}
  Begin
    TestDetect:= 1; { Default mode = 1 }
  End;
{$F-}
Begin
   {Install the driver }
   TestDriver:= InstallUserDriver('TEST',
      @TestDetect);
   If(GraphResult <> grOk) then
     Begin
       Writeln('Error installing TestDriver');
```

```
      Halt(1);
    End;
  Driver:=Detect;   { Put in graphics mode }
  InitGraph(Driver, Mode, '');
  ErrCode:=GraphResult;
  If(ErrCode <> grOk) then
    Begin
      Writeln('Error during Init: ', ErrCode);
      Halt(1);
    End;
  OutText('Installable drivers supported...');
  Readln;
  CloseGraph;
End.
```

press any key to halt عند التنفيذ تظهر الرسالة



لاحظ أن المحاور على الشاشة مختلفة عما عليه على الورق، فلا يوجد قيم سالبة لإحداثيات x,y على الشاشة مطلقاً، بينما يمكن رسم قيم سالبة على الورق (تصور الشاشة كلها عبارة عن الربع الأول فقط وبصورة مقلوبة) وكذلك يجب استخدام قيم صحيحة فقط على الشاشة، بينما يجوز استخدام قيم حقيقية أو صحيحة على الورق ومع هذا فيمكننا تمثيل القيم السالبة على شاشة من خلال برامج خاصة.

17. الإجراء moveRel

وصيغته العامة بلغة الباسكال كما يلي:

```
Procedure MoveRel (dx,dy:integer);
```

يتم بواسطة هذا الإجراء التحرك من Dx من النقاط أفقياً و Dy من النقاط عمودياً نسبة للموقع الحالي وليس لنقطة الأصل (0,0) كما في المثال التالي:

```
MoveRel (200,200);
Lineto(300,300);
```

هذا الجزء من البرنامج يرسم خط مستقيم من الموقع (200,200) إلى الموقع (300,300)، حيث يتم التحرك 200 نقطة أفقياً و 200 نقطة عمودياً عن نقطة الأصل (0,0) باستدعاء الإجراء MoveRel(200,200) ثم رسم الخط المستقيم من الموقع (200,200) عند استدعاء الإجراء LineTo. بينما لو وضعنا عبارة MoveTo(50,50) قبل هاتين العبارتين فسيتم رسم الخط المستقيم من الموقع (250,250) إلى الموقع (300,300).

البرنامج التالي بلغة الباسكال يستدعي الإجراء MoveRel، حيث يتحرك أولاً إلى النقطة (100,100) بواسطة الإجراء MoveTo ثم يتحرك 10 نقاط على المحور x و 10 نقاط على المحور y بالنسبة للنقطة الأولى ويرسم نقطة هناك ثم يرسم مستقيم بين النقطتين (120,220) و (200,220).

```
program moverel;
 {Sample code for the MoveRel procedure.}
uses Graph;
var
   Gd,Gm:Integer;
Begin
  Gd:=Detect;
  InitGraph(Gd, Gm,'c:\Tp\BGI');
  If(GraphResult <> grOk) then
     Halt(1);
  MoveTo(100, 100);
  MoveRel(10, 10); {Move to the point (110, 210)}
  PutPixel(GetX, GetY, GetMaxColor);
  line(getx + 10 ,gety+10 , getx+80, gety+10);
   {draw line from 120,220 to 200,220}
  Readln;
  CloseGraph;
End.
```

18. الإجراء LineRel

وصيغته العامة بلغة الباسكال كما يلي:

```
Procedure LineRel (dx,dy:integer);
```

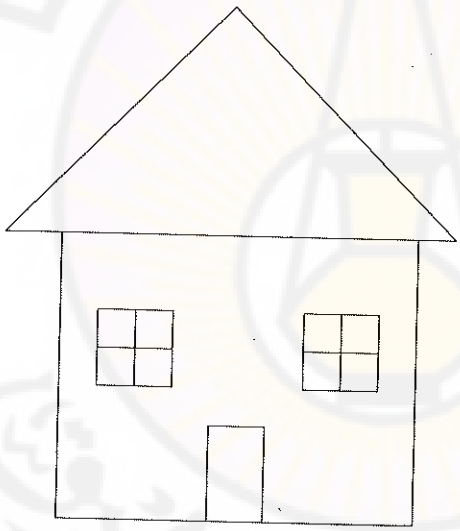
يقوم هذا الإجراء برسم خط مستقيم من الموقع الحالي إلى الموقع الذي يبعد عنه Dx من النقاط أفقياً و Dy من النقاط عمودياً، فمثلاً إذا كنا في الموقع (50,50) على شاشة الرسم فالاستدعاء Linerel(150,150) سيؤدي إلى خط مستقيم من النقطة (50,50) إلى النقطة (200,200) كما في المثال التالي:

```
MoveTo (50,50);
LineRel(200,200);
```

والآن يمكن التدرب على كل الإجراءات السابقة بإعادة رسم الأشكال التي تم رسمها باستخدام الإجراء Line فقط. البرنامج التالي يستخدم بعض الإجراءات الأنفة الذكر بلغة الباسكال ويرسم شكل بيت.

```
program House1;
uses Crt, Graph;
var
   Gd,Gm:Integer;
   dummy:char;
 Begin
  Gd:=Detect;
  InitGraph(Gd, Gm,'c:\Tp\BGI');
   {the house roof}
  line(320,50,200,200);
  line(200,200,430,200);
  line(430,200,320,50);
   {the hose}
  Line(400,200,400,350);
  moveto (430,200);
  line(400,350,230,350);
  line(230,350,230,200);
  {the door}
  line(300,350,300,300);
  line(300,300,330,300);
  line(330,300,330,350);
   {the left window}
  line(350,250,380,250);
  line(380,250,380,280);
  line(380,280,350,280);
  line(350,280,350,250);
```

```
    line(365,250,365,280);
    line(350,265,380,265);
     {the Right window}
    line(250,250,280,250);
    line(280,250,280,280);
    line(280,280,250,280);
    line(250,280,250,250);
    line(265,250,265,280);
    line(250,265,280,265);
    dummy:=readkey;
    closegraph;
End.
```



19. الإجراء bar3D

وصيغته العامة كما يلي:

```
procedure Bar3D(x1,y1,x2,y2 :integer ;Depth:word
;Topflag : boolean);
```

يرسم هذا الإجراء متوازي مستطيلات ويطلي واجهته باللون والتشكيلة قيد الاستخدام. حيث:

(x1,y1): تمثلان النقطة التي تمثل الزاوية اليسرى العليا للواجهة.

(x2,y2): تمثلان النقطة التي تمثل الزاوية اليمنى السفلى للواجهة.

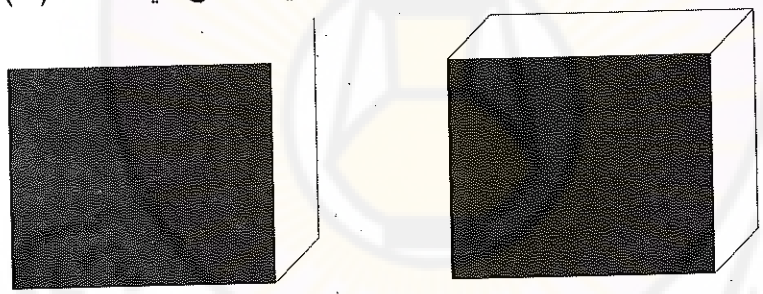
Depth: يمثل عرض القاعدة.

Topflag: يتم رسم القاعدة العليا (الغطاء) لمتوازي المستطيلات إذا كانت قيمة البارامتر البولياني بلغة باسكال تساوي True، ولا يتم رسمها إذا كانت قيمه False

مثال:

```
Topfalg := true;
Bar3D (50,30,150,100,40,Topflag);
```

هذا الجزء من البرنامج سيرسم الشكل التالي (أ) أما إذا كانت قيمة Top تساوي False فسكون الشكل بلا قاعدة عليا كما هي موضح في الشكل (ب).



(ب) (أ)

20. الإجراء DrawPoly

وصيغته في لغة الباسكال كما يلي:

```
Procedure DrawPoly (numPoints:word; Var PolyPoints);
```

يرسم هذا الإجراء مضلع باستخدام شكل ولون الخط الحالي، عدد أضلاعه يعطى في البارامتر NumPoints، أما البارامتر PolyPoints فيمثل مصفوفة

نقاط رؤوس المضلع مضافاً إليه واحد، فمثلاً للشكل السداسي سيكون حجمها 7 وللشكل الثماني 9 وهكذا ...

مثال:

```
DrawPoly (7,hex);
```

على أن نعرف البارامتر hex كما يلي:

```
Type
   hextype = array [1..7] of Pointtype;
Var
   hex : hextype;
```

حيث pointsType معرف ضمن النظام على أنه سجل record ذو حقلين هما x,y يمثلان إحداثيات كل نقطة من نقاط الرؤوس.

5.3 رسم قطاع دائرة Drawing Pie slice

الإجراء PieSlice1 في البرنامج التالي يستخدم طريقة التمثيل التزايدي لرسم الدائرة، حيث تم رسم قوس دائرة محصور بين زاوية البداية start وزاوية النهاية ending واللتان أعطيتا القيمتان 0 و 90 في البرنامج ثم تم إيصال نهايتي القوس بمركز الدائرة لرسم قطاع دائري، حيث:

Radius : نصف قطر الدائرة.

x0,y0 : تمثل إحداثيات نقطة مركز الدائرة.

St : زاوية البداية بالدرجات.

ET : زاوية النهاية بالدرجات.

```
procedure Pieslice1(x0,y0,st,et,a:integer);
   var
     theta,x1,y1,temp,x1temp,y1temp,x2temp,y2temp
     :integer;
```

```
    x,y:real;
Begin
  x1temp:=round(x0+a*cos(st*pi/180));
  y1temp:=round(y0+a*sin(st*pi/180));
  line(x0,y0,x1temp,y1temp);
  x2temp:=round(x0+a*cos(et*pi/180));
  y2temp:=round(y0+a*sin(et*pi/180));
  line(x0,y0,x2temp,y2temp);
  For theta:= st+1 to et-1 do
    Begin
      x:=x0+a*cos(theta*pi/180);
      y:=y0+a*sin(theta*pi/180);
    End;
End;
```

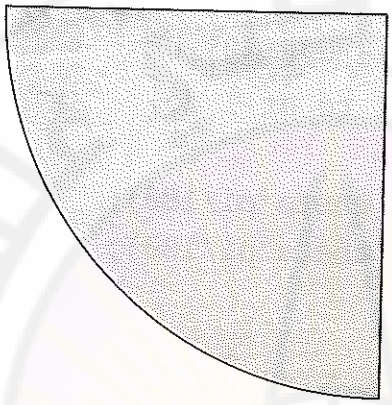
ويمكن التمرن على رسم القطوع الدائرية الناقصة بأحجام مختلفة وألوان مختلفة باستخدام طرق رسم الدوائر.

البرنامج التالي بلغة الباسكال يستخدم الإجراء القياسي poeslice ليرسم قطاع دائرة مركزها هو (180,180) ونصف قطره 150، زاويتي البداية والنهاية هما 180 ، 270 على التوالي

```
program pieslic;
uses Graph;
const
  Radius=150;
var
   Gd,Gm:Integer;
Begin
  Gd:=Detect;
  InitGraph(Gd, Gm,'c:\tp\bgi');
  If(GraphResult <> grOk) then
```

```
    Halt(1);
  PieSlice(180, 180, 180, 270, Radius);
  Readln;
  CloseGraph;
End.
```

الخرج سيكون كما في الشكل التالي:



6.3 رسم القطوع الناقصة وقطاعاتها Drawing ellipse and Pie Slice

لرسم شكل القطاع الناقص تستخدم العلاقات التالية:

$$x = x_0 + a\cos\theta$$

$$y = y_0 + b\sin\theta$$

حيث:

x_0, y_0 هي إحداثيات نقطة مركز القطع الناقص.

a : نصف القطر الأفقي (السيني).

b : نصف القطر العمودي (الصادي).

إن حساب عدة نقاط للشكل القطع الناقص بهذه الطريقة يستغرق وقتاً طويلاً نسبياً بسبب حسابات $\sin\theta$ و $\cos\theta$ حيث θ تتغير من الصفر إلى 2π فإذا

كان التغير بالزاوية صغير ولكل زاوية يتم حساب نقطة على محيط الشكل القطع الناقص، وعليه يمكن استخدام أسلوب التمثيل التزايدي الذي استخدمناه سابقاً لرسم الدائرة. ولإيجاد أي نقطة على المحيط بدلالة النقطة السابقة لها يتم من خلال العلاقات التالية:

$$x_2 = x_1 + a\cos(\theta + d\theta)$$

$$y_2 = y_0 + b\sin(\theta + d\theta)$$

مثال:

البرنامج التالي يستخدم العلاقتين السابقتين وكذلك علاقات التناظر لتقليل وقت حساب النقاط إلى الربع:

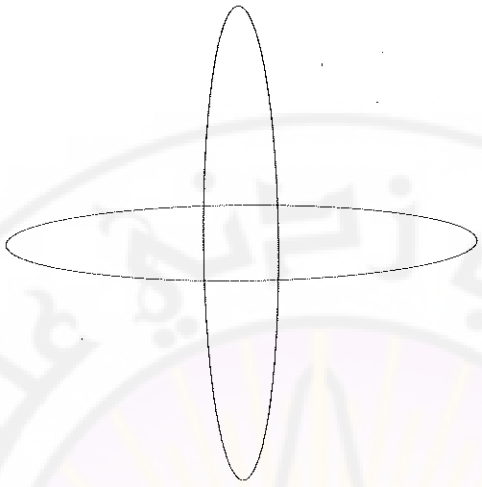
```
Program elipse1;
uses crt ,  Graph;
var
   Gd,Gm,ax:Integer;
   maxx,maxy,a,b,x0,y0,theta:Integer;
   x1,y1,x2,y2,x3,y3,x4,y4,x5,x6,x7:Integer;
   x8,y5,y6,y7,y8 :integer;
Procedure symmetry;
  Begin
      x2:=2*x0-x1;
      y2:=y1;
      putpixel(x2,y2,14);
      x3:=x2;
      y3:=2*y0-y1;
      putpixel(x3,y3,14);
      x4:=x1;
      y4:=y3;
      putpixel(x4,y4,14);
```

```
  End;
Begin
  Gd:=Detect;
  InitGraph(Gd, Gm,'c:\Tp\BGI');
  maxx:=getmaxX;
  maxy:=getmaxY;
  a:=150;
  b:=60;
  x0:=maxx div 2;
  y0:=maxy div 2;
  For theta:=0 to 90 do
    Begin
      x1:=x0 + a * cos((theta * pi) / 180);
      y1:=y0 + b * sin((theta * pi) / 180);
      putpixel(x1,y1,14);
      symetry;
    End;
  Readln;
  closegraph;
End.
```

وبإضافة العبارات التالية:

```
x5 := y1 –y0 +x0; y5 := x1 –x0 +y0;
putpixel(x5,y5,15);
x6 := y2 –y0 +x0; y6 := x2 –x0 +y0;
putpixel(x6,y6,15);
x7 := y3 –y0 +x0; y7 := x3 –x0 +y0;
putpixel(x7,y7,15);
x8 := y4 –y0 +x0; y8 := x4 –x0 +y0;
putpixel(x8,y8,15);
```

قبل نهاية الإجراء Symmetry فسيتم رسم شكلان قطعان ناقصان متماثلان ومتقاطعان بصورة عمودية ولهما نفس المركز، كما في الشكل التالي:



وبتطبيق قواعد المثلثات تصبح العلاقات كما يلي:

$$x_2 = x_1 \cos(d\theta) - (a/b) y_1 \sin(d\theta)$$

$$y_2 = y_1 \cos(d\theta) - (b/a) x_1 \sin(d\theta)$$

حيث تستخدم $d\theta$ والتي يتم حسابها مرة واحدة فقط بدلاً من θ.

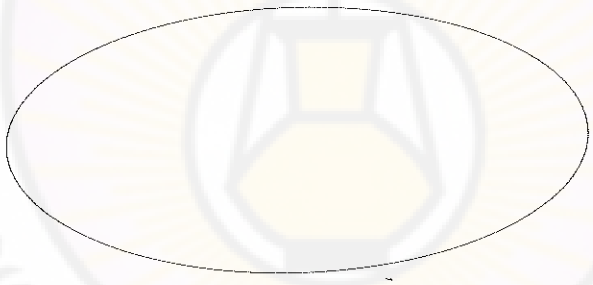
مثال:

البرنامج التالي يستخدم هذه الطريقة لرسم شكل قطع ناقص كما يوضح الشكل التالي للبرنامج.

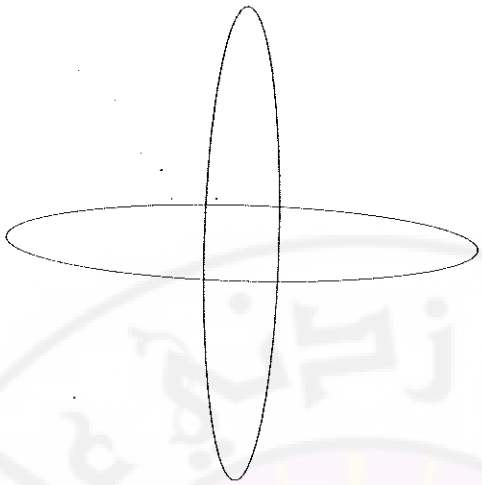
```
Program elipse2;
uses crt, Graph;
const
   pi=3.14;
var
   Gd,Gm,ax:Integer;
   maxx,maxy,a,b,x0,y0,theta: Integer;
   x1,y1,x2,y2,x3,y3,x4,y4,x5,x6,x7: Integer;
```

```
    x8,y5,y6,y7,y8: integer;
    x,y,xx,yy,dtheta,cosdtheta,sindtheta: real;
Procedure symmetry;
  Begin
      x2:=2*x0-x1;
      y2:=y1;
      putpixel(x2,y2,14);
      x3:=x2;
      y3:=2*y0-y1;
      putpixel(x3,y3,14);
      x4:=x1;
      y4:=y3;
      putpixel(x4,y4,14);
  End;
Begin
  Gd:=Detect;
  InitGraph(Gd, Gm, 'c:\Tp\BGI');
  maxx:= getmaxX;
  maxy:= getmaxY;
  a:=150;
  b:=60;
  x0:=maxx div 2;
  y0:=maxy div 2;
  x:=x0 + a*cos(theta * pi / 180);
  y:=y0 + a*sin(theta * pi / 180);
  x1:=round(x);
  y1:=round(y);
  putpixel(x1,y1,14);
  symmetry;
  dtheta:=1;
```

```
cosdtheta:=cos(dtheta * pi /180);
sindtheta:=sin(dtheta * pi /180);
For theta:=0 to 90 do
  Begin
     xx:=x*cosdtheta-a/b*(y-y0)*sindtheta;
     x1:=round(xx);
     yy:=y*cosdtheta+b/a*(x-x0)*sindtheta;
     y1:=round (yy);
     putpixel(x1,y1,14);
     x:= xx; y := yy;
     symmetry;
   End;
 Readln;
 closegraph;
End.
```



بإضافة العبارات التالية قبل نهاية الإجراء Symmetry فسيتم رسم قطعين ناقصين متماثلين ومتقاطعين بصورة عمودية ولهما نفس المركز، كما في الشكل التالي:



```
x5 := y1 –y0 +x0; y5 := x1 –x0 +y0;
putpixel(x5,y5,15);
x6 := y2 –y0 +x0; y6 := x2 –x0 +y0;
putpixel(x6,y6,15);
x7 := y3 –y0 +x0; y7 := x3 –x0 +y0;
putpixel(x7,y7,15);
x8 := y4 –y0 +x0; y8 := x4 –x0 +y0;
putpixel(x8,y8,15);
```

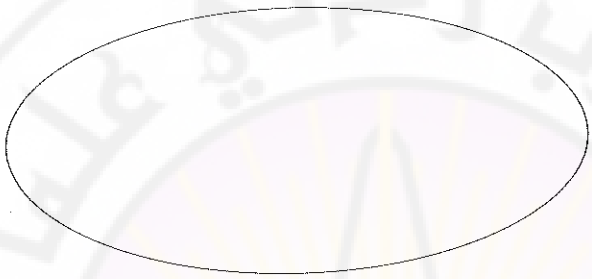
مثال:

البرنامج التالي بلغة الباسكال لرسم قطع ناقص باستخدام الإجراء القياسي ellipse.

```
program Ellips3;
  {Sample code for the Ellipse procedure.}
uses Graph;
var Gd, Gm: Integer;
Begin
  Gd:=Detect;
  InitGraph(Gd, Gm, 'c:\tp\bgi');
```

```
  If(GraphResult <> grOk) then
     Halt(1);
  Ellipse(200, 200, 0, 360, 150, 70);
  Readln;
  CloseGraph;
End.
```

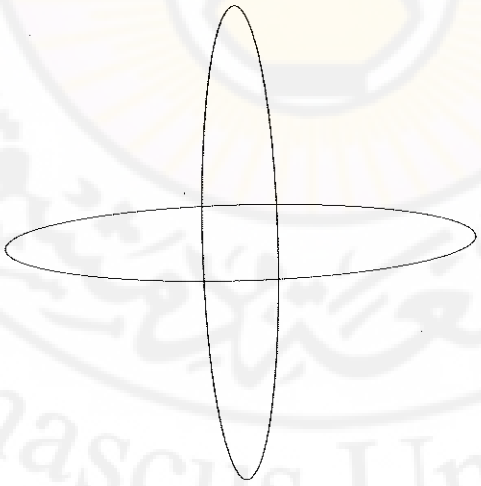
خرج البرنامج هو تماماً كما في الشكل التالي:



ويمكن إضافة العبارة

Ellipse(200,200,0,360,70,150);

بعد عبارة الاستدعاء الأولى لنحصل على شكلين لقطعين ناقصين متعامدين كما في الشكل التالي:



والإجراء القياسي لرسم قطاع شكل قطع ناقص هو Sector وصيغته العامة كما يلي:

```
procedure Sector (x,y : integer; stangle,
                  enangel,Xradius,yradius:word);
```

يرسم هذا الإجراء قطاع شكل قطع ناقص مطلي باللون والتشكيلة قيد الاستخدام حيث:

x,y : نمثل إحداثيات نقطة المركز للشكل القطع الناقص.

stangel : زاوية البداية بالدرجات.

endangel : زاوية النهاية بالدرجات.

Xradius : نصف القطر الأفقي للشكل القطع الناقص.

Yradius : نصف قطر العمودي للشكل القطع الناقص.

مثال:

```
sector (350,250,90,0,60,100);
```

استدعاء كهذا سيرسم جزء من شكل قطع ناقص مركزه عند النقطة (250,350) ويبدأ من الزاوية 90 درجة وينتهي بالزاوية صفر وبنصف قطر أفقي مقداره 60 ونصف قطر عمودي مقداره 100.

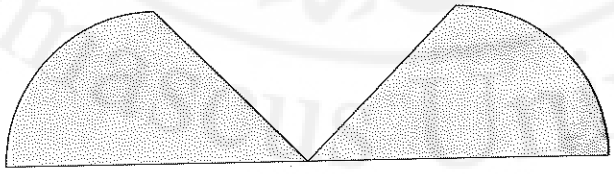
ملاحظة:

في حالة كون Xradius مساوية لـ Yradius فإنه سيرسم قطاع دائري وليس قطاع شكل قطع ناقص.

مثال:

البرنامج التالي يستدعي الإجراء Sector ليرسم قطاع دائرة حيث تم يستخدام نصفي قطرين متشابهين في البارمترات الثابت R والذي قيمته 150 ثم يرسم صورة للقطاع تماماً كما تظهر الصورة لأي شكل في المرآة (الشكل التالي للبرنامج).

```
program SectorP;
  {Sample code for the Sector procedure.}
 uses Graph;
 const R = 150;
 var
    Driver, Mode: Integer;
    Xasp, Yasp: Word;
Begin
  Driver:=Detect;{Put in graphics mode}
  InitGraph(Driver,Mode,'c:\tp\bgi');
  If(GraphResult < 0) then
     Halt(1);
  Sector(GetMaxX div 2,GetMaxY div 2, 0,45,R,R);
  GetAspectRatio(Xasp, Yasp);
    {Draw circular sector }
  Sector(GetMaxX div 2, GetMaxY div 2,{Center
   point 180, 135,{ Mirror angle above } R,R*
   Longint(Xasp) div Yasp);{ Circular }
  Readln;
  CloseGraph;
 End.
```



1.4 روتينات التشكيل والتلوين

1. الإجراء SetFillstyle

وصيغته العامة هي:

```
procedure setFillstyle (Pattern : word ; color : word);
```

يعد هذا الإجراء التشكيلة واللون لملئ الأشكال المغلقة بالتشكيلات التي تستخدم من قبل الإجراءات FillPoly و Bar و Bar3D و Pieslice. يتم إعطاء اسم التشكيلة أو رقمها واسم اللون أو رقمه في البارامترات Pattern و color على التوالي. إذا تم الاستدعاء بقيم مقبولة سيعيد الإجراء القيمة 11- وتبقى القيمة الجارية للتشكيلة واللون بدون تغيير.

القيمة الاعتباطية للتشكيلة هي Solidfill واللون الاعتباطي هو Maxcolor أي أعلى قيمة للألوان.

2. الإجراء Getfillsettings

وصيغته العامة هي:

```
procedure getFillsettings (var fillinfo : fillsettingsType);
```

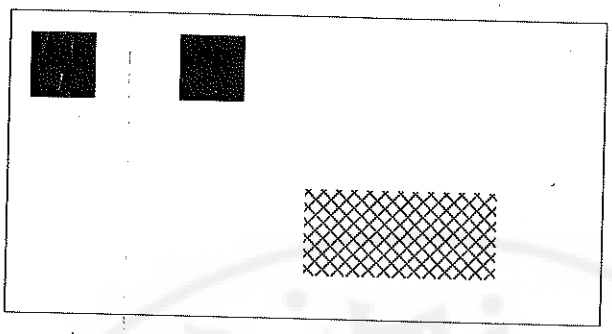
يعيد هذا الإجراء تشكيلة لملئ الجارية واللون كما تم إعدادهما بواسطة الإجراء أو Setfillpattern. إذا كانت قيمة التشكيلة تساوي userfill فاستخدم الإجراء getfillPattern للحصول على التشكيلة المعرفة من قبل المستخدم والتي تم اختيارها. النوع FillsettingsType معرف في النظام ولا يحتاج إلى إعادة تعريفه حيث هو سجل يتكون من حقلين أحدهما يمثل التشكيلة والآخر يمثل اللون كما هو موضح فيما يلي:

```
fillsettingsType = record
   Pattern : word;
   color : word;
```

```
   end;
```

البرنامج التالي بلغة الباسكال يستدعي كل من الإجراءين getfillsettings و setfillsettings. الخرج موضح في الشكل التالي للبرنامج.

```
program Getfsets
 {Sample code for the GetFillSettings procedure.}
uses Graph;
var
  Gd, Gm: Integer;
  FillInfo: FillSettingsType;
Begin
 Gd:= Detect;
 InitGraph(Gd, Gm, '');
 If(GraphResult <> grOk) then
    Halt(1);
 GetFillSettings(FillInfo); {Save fill style
                                and color}
 Bar(0, 0, 50, 50);
 SetFillStyle(XHatchFill,GetMaxColor);{New style}
 Bar(50, 0, 100, 50);
 with FillInfo do
     SetFillStyle(Pattern, Color);{Restore old
                                   fill style}
 Bar(100, 0, 150, 50);
 Readln;
 CloseGraph;
End.
```



3. الإجراء Floodfill

وصيغته العامة هي

```
procedure FloodFill(x,y : integer ; Border : word);
```

يعمل هذا الإجراء على ملئ مساحة الشكل المغلق باللون والتشكيلة الجاريين حيث تمثل x,y إحداثيات أي نقطة تقع ضمن الشكل المغلق وتسمى البذرة أما Border فيمثل لون الخط المحيط بالشكل. يتم تحديد اللون والتشكيلة المطلوبة بواسطة الإجراء setFillStyle أو الإجراء setfillPattern. إذا كانت البذرة تقع خارج حدود الشكل المغلق أو إن الشكل لم يكن مغلق تماماً فإن اللون والتشكيلة المحددان ستنتشر خارج الشكل وإذا لم يكن هناك أي خط مغلق يحيط بالشكل فإن الطلاء سينتشر على كل الشاشة.

مثال:

البرنامج التالي يرسم 13 دائرة ويملأها بالتشكيلات المتاحة في النظام وذلك حسب رقم التشكيلة المحدد في المتغير i الذي يبدأ بالقيمة صفر وينتهي بالقيمة 12 بزيادة واحد في كل دورة من خلال حلقة الدوران repeat-until وعند الضغط على مفتاح Enter يختفي الرسم وتظهر على الشاشة 16 دائرة كل منها ملونة بأحد الألوان المتوفرة من اللون الأول رقم صفر إلى الون رقم خمسة عشر، حيث يبدأ الرسم من جهة اليسار. ويمكن الاستغناء عن أرقام التشكيلات والألوان بأسمائها.

```
program FillcolorDemo;
uses crt, Graph;
var
  I,x,y,x2 :Integer;
  Gd,Gm:Integer;
  dummy:char;
Begin
  Gd:=Detect;
  InitGraph(Gd, Gm,'c:\tp\Bgi');
  If(GraphResult <> grOk) then
     Halt(1);
  I:=0;
  x:=100;
  y:=100;
  x2:=10;
  Repeat
     setfillstyle(I,15);
     circle(x,y,40);
     floodfill(x,y,15);
     x:=x+80;
     I:=I+1;
     If(I >= 7) then
       Begin
           x2:=x2+90;
           x:=x2;
           y:=230;
        End;
   Until(I= 13);
   dummy := readkey;
   cleardevice;
```

```
    I:=0;
    x:=100;
    y:=100;
    x2:=10;
    Repeat
       setfillstyle(solidfill,I);
       circle (x,y,30);
       floodfill(x,y,15);
       x:=x+72;
       I:=I+1;
       If(I >= 8) then
         Begin
             x2:=x2+73;
             x:=x2;
             y:=230;
         End;
   Until(I=16);
   dummy:=readkey;
   CloseGraph;
End.
```

نفذ البرنامج على جهازك لترى شكل الخرج.

4. الإجراء setFillPattern

صيغته العامة هي:

```
procedure setfillpattern (pattern : fillpatternType; color : word);
```

يختار هذا الإجراء اللون والتشكيلة المعدة من قبل المستخدم، حيث Fillpattrentype معرفة في النظام كما يلي:

```
type
```

```
fillpatternType = array [1.. 8] of byte;
```

انظر الإجراء التالي.

5. الإجراء getfillpattern

صيغته العامة هي:

```
Procedure getfillpattern (var fillpattern : fillpaterntype );
```

يعيد هذا الإجراء اللون والتشكيلة الجارية المستخدمة في ملئ الأشكال المغلقة والتي تم إعدادها بالإجراء setfillpattern. إذا لم يتم استدعائها setfillpattern فسيعيد getfillpattern مصفوفة مملوءة بالقيمة $FF. حيث FillPattern معرفة مسبقاً في النظام بالشكل التالي:

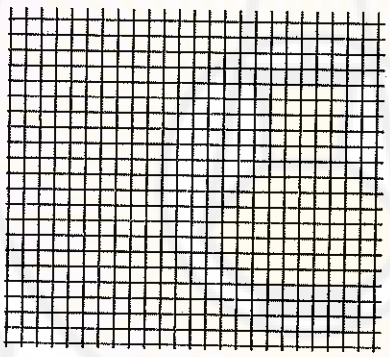
```
type
   fillpatternType = array [1.. 8] of byte;
```

مثال:

البرنامج التالي بلغة الباسكال يستدعي كل من الإجراءين getfillpattern و setfillpattern.

```
procedure getfpatt;
  {Example for GetFillPattern and SetFillPattern }
  uses Graph;
const
   Gray50:FillPatternType=($AA, $55, $AA,
       $55, $AA, $55, $AA, $55);
var
  Gd, Gm : Integer;
  OldPattern : FillPatternType;
Begin
  Gd:=Detect; InitGraph(Gd, Gm, '');
  If(GraphResult <> grOk) then
```

```
    Halt(1);
  GetFillPattern(OldPattern);
  SetFillPattern(Gray50, White);
   {Draw bar 50% gray }
  Bar(100, 100, 300, 300);
  ReadLn;
  SetFillPattern(OldPattern,White);
   {Draw bar with old pattern }
   {(normally filled with $FF }
   {on startup)               }
  Bar(0, 0, 100, 100);
  ReadLn;
  CloseGraph;
End.
```



6. الإجراء fillpoly

يرسم هذا الإجراء شكل متعدد أضلاع باستخدام شكل ولون الخط الجاريين ثم يلونه بالتشكيلة واللون المحدد بواسطة setfillstyle أو setfillpattern. أما إذا لم يتم استدعاء أي من هذين الإجراءين قبله فسيتم ملء متعدد الأضلاع باللون والتشكيلة الاعتباطية وهما اللون الأبيض والتشكيلة solidfill، الصيغة العامة بلغة الباسكال كما يلي:

```
procedure fillpoly(Numpoints : word; var polypoints);
```

حيث:

numpoints : تمثل عدد نقاط رؤوس المضلع.

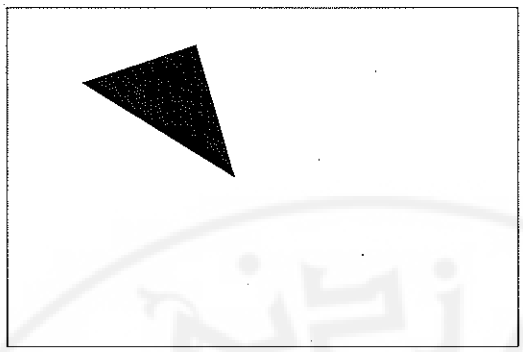
PolyPoints : تمثل مصفوفة إحداثيات رؤوس المضلع من النوع Pointtype تماماً كما هو موضحة في الأجزاء السابقة، حجم هذه المصفوفة بنفس عدد نقاط رؤوس المضلع لأن هذا الإجراء يغلق المضلع اوتماتيكياً وليس كما هو الحال في الإجراء (Drawpoly). إذا حصل أي خطأ أثناء ملء متعدد الأضلاع سيعيد هذا الإجراء القيمة grNoScanMem.

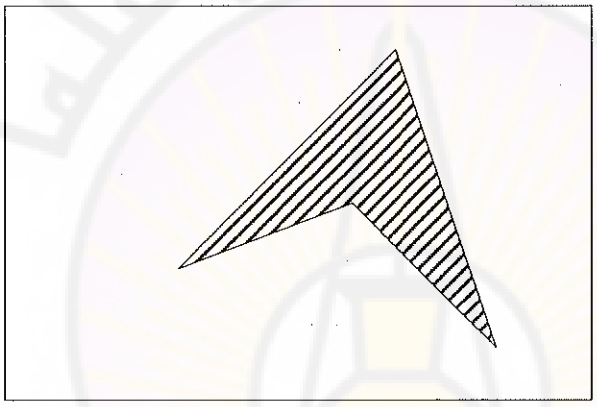
مثال:

البرنامج التالي بلغة الباسكال يرسم مثلث (كما في الأشكال التالية للبرنامج) ويلونه باللون والتشكيلة الجاريين وهما اللون الأبيض والتشكيلة solidfill.

```
program fillpol;
  {Sample code for the FillPoly procedure.}
uses Graph;
const
  Triangle:array [1..3] of PointType=((X:50;Y:
    100),(X: 100; Y: 100), (X: 150; Y: 150));
var
   Gd,Gm:Integer;
Begin
  Gd:=Detect;
  InitGraph(Gd, Gm, 'c:\tp\bgi');
  If(GraphResult <> grOk) then
    Halt(1);
  FillPoly(SizeOf(Triangle) div
      SizeOf(PointType), Triangle);
  Readln;
  CloseGraph;
```

End.





7. الإجراء Fillellipse

وصيغته العامة هي:

procedure fillellipse (x,y:integer; xradius, yradius : word);

يعمل هذا الإجراء على رسم شكل قطع ناقص وملئه باللون والتشكيلة الجارية، حيث:

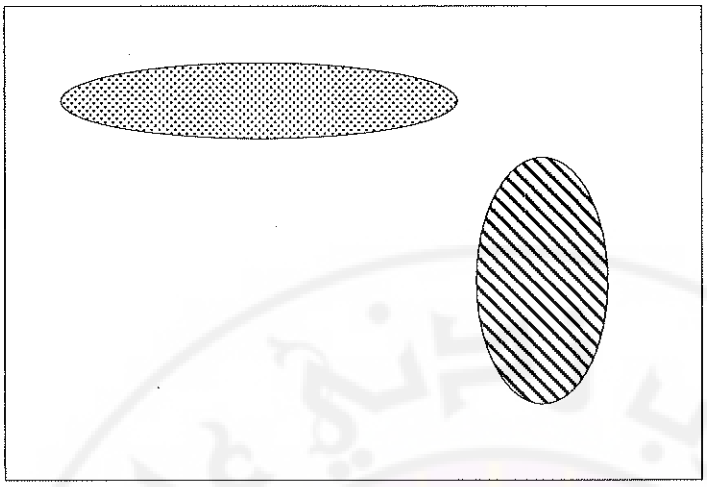
x,y : هي إحداثيات نقطة المركز.

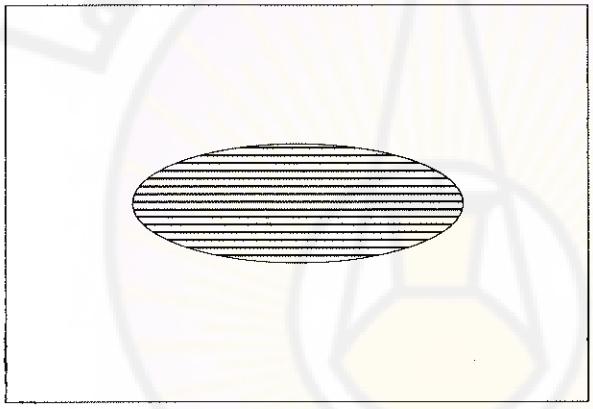
xradius, yradius : هما القطران الأفقي والعمودي للقطع الناقص.

مثال:

البرنامج التالي بلغة الباسكال يرسم قطعين ناقصين مختلفين ويملأهما بشكيلتين مختلفتين كما هو موضح في الأشكال التالية للبرنامج.

```
program fillellips;
  {Sample code for the FillEllipse procedure.}
uses Graph;
const
    R=130;
var
  Driver, Mode: Integer;
  Xasp, Yasp: Word;
Begin
 Driver:=Detect; {Put in graphics mode}
 InitGraph(Driver, Mode,'c:\tp\Bgi');
 If(GraphResult < 0) then
     Halt(1);
 setfillstyle(5,3);
    {Draw ellipse}
 FillEllipse(GetMaxX div 2,GetMaxY div 2,50, 80);
  setfillstyle(11,3);
  GetAspectRatio(Xasp, Yasp);
  {Circular ellipse}
  FillEllipse(R, R, R, R * Longint(Xasp)
              div Yasp-100);
  Readln;
  CloseGraph;
End.
```





8- الإجراء getPalette

وصيغته العامة هي:

```
procedure getPalette (var Palette : PaletteType);
```

يعيد هذا الإجراء القيمة الحالية لحجم الألوان والقيم الرقمية لألوانها في حقلي المتغير palette حيث هو من نوع السجل وهو معرف مسبقاً في النظام كما يلي:

```
Type
   paletteType = record
         size : Byte;
```

```
    colors : array [0 .. maxcolors] of shortint;
end;
```

حيث Size عادة يعيد حجم لوحة الألوان، Colors يعيد القيم الرقمية المناظرة للألوان المستخدمة في لوحة الألوان. ويستخدم هذا النوع من قبل الإجراءات setPalette و getPalette و getdefualtPalette و setAllPalette.

باستدعاء مفرد لهذه الإجراءات ستعرف ما هي الألوان المتاحة لك للاستخدام في الرسوم ويتم استدعائها كما يلي:

```
GetPalette(Palette );
```

9. الإجراء setcolor

وصيغته العامة بلغة الباسكال هي:

```
procedure setcolor (color : word);
```

يقوم هذا الإجراء بتحديد لون الرسم البياني الحالي بالقيمة المعطاة في البارامتر color على شرط أن يكون هذا اللون هو أحد الألوان الموجودة في لوحة الألوان، حيث يمكن استدعاء هذا الإجراء قبل أي أجراء من إجراءات الرسم التي ذكرناها سابقاً لإعداد لون الرسم إلى اللون المطلوب.

مثال:

```
setcolor (Red);
Line (100,100,400,400);
```

حيث سيتم رسم خط مستقيم بين النقطتين (100,100) و (400,400) باللون الأحمر ويمكن الاستعاضة عن اسم اللون برقمه كما يلي:

```
setcolor (4);
```

مقطع البرنامج التالي يستخدم الإجراء Line لرسم 16 خط مستقيم بستة عشرة لون وملتقية في نقطة وسط الشاشة كما موضح في الشكل التالي للبرنامج.

```
maxx := getmaxx div 2;      dx:=0;
```

```
maxy := getmaxy div 2;        dy := 0;
For i:= 0 to 15 do
  Begin
    setcolor(i);
    Line (maxx,maxy,maxx +120+dx,maxy+dy);
    dx := dx -2;
    dy := dy -15;
  End;
```

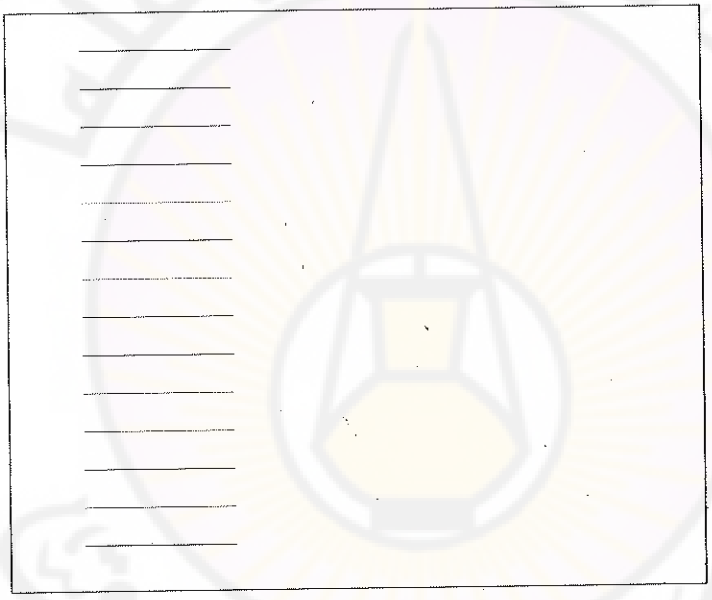
البرنامج التالي بلغة الباسكال يستدعي كل من الإجرائين getpalette و setpalette، حيث الخرج موضح في الشكل التالي للبرنامج.

```
program getplt;
 {Sample code for the GetPalette procedure.}
uses Graph;

var
  Gd,Gm:Integer;
  Color:Word;
  Palette:PaletteType;
Begin
  Gd:=Detect;
  InitGraph(Gd, Gm,'c:\tp\bgi');
  If(GraphResult <> grOk) then
    Halt(1);
  GetPalette(Palette);
  If(Palette.Size <> 1) then
    For Color:=0 to Pred(Palette.Size) do
      Begin
        SetColor(Color);
        Line(100, Color * 20, 200, Color * 20);
```

```
    End
  Else
    Line(50, 50, 150, 50);
  Readln;
  CloseGraph;
End.
```

لاحظ أنه باستدعاء الإجراء setcolor قبل رسم خط مستقيم وباستخدام قيم مختلفة للبارامتر color نحصل على الخطوط بألوان مختلفة كما في الشكل التالي:



10. الدالة getMaxColor

صيغتها العامة بلغة الباسكال هي كما يلي:

```
function getmaxcolor : word;
```

تعيد هذه الدالة أعلى كود للألوان المتوفرة في لوحة الألوان المستخدمة.

```
color := getmaxcolor;
```

وغالباً ما تكون أعلى قيمة تمثل اللون الأبيض.

```
program getmaxcol;
  {Sample code for the GetMaxColor, SetColor
    functions.}
uses Crt, Graph;
var
   GraphDriver, GraphMode : Integer;
Begin
  GraphDriver:=Detect;
  InitGraph(GraphDriver, GraphMode,'c:\tp\bgi');
  If(GraphResult <> grOk) then
     Halt(1);
  Randomize;
  Repeat
     SetColor(Random(GetMaxColor)+1);
     LineTo(Random(GetMaxX),Random(GetMaxY));
  Until KeyPressed;
End.
```

11. الدالة getbkcolor

صيغتها العامة بلغة الباسكال هي كما يلي:

```
Function getBkcolor : word;
```

تعيد هذه الدالة لون خلفية الشاشة الجاري، حيث تتراوح ألوان الخلفية في المدى م 0 إلى 15 اعتماداً على سواقة الرسم قيد الاستخدام ونمط الرسم.

مثال:

```
color := getBkcolor;
```

مثال:

البرنامج التالي بلغة الباسكال يستدعي الدالة getBKcolor والدالة setbkcolor حيث يسند القيمة التي تلي القيمة الجارية للون الخلفية إلى لون الخلفية التالي ويرسم خطوط مستقيمة بصورة عشوائية وعندما يصل لون الخلفية إلى أعلى قيمة له يسند القيمة 0 إلى اللون ويستمر بالدوران إلى أن يتم الضغط على المفتاح Enter.

```
program getbkcol;
  {Sample code for the GetBkColor function.}

uses Crt, Graph;
var
   Gd,Gm:Integer;
   Color:Word;
   Pal:PaletteType;
Begin
  Gd:=Detect;
  InitGraph(Gd, Gm,'c:\tp\bgi');
  If(GraphResult <> grOk) then
     Halt(1);
  Randomize;
  GetPalette(Pal);
  If(Pal.Size <> 1) then
    Begin
      Repeat  {Cycle through colors}
        Color:= Succ(GetBkColor);
        If(Color > Pal.Size-1) then
           Color:= 0;
        SetBkColor(Color);
```

```
        LineTo(Random(GetMaxX), Random(GetMaxY));
      Until KeyPressed;
    End
  Else
     Line(0, 0, GetMaxX, GetMaxY);
  Readln;
  CloseGraph;
End.
```

12. الإجراء setPlaette

صيغته العامة بلغة الباسكال هي كما يلي:

procedure setpalette (ColorNum : word; color :shortint);

يعمل هذا الإجراء على تغير اللون colornum في لوحة الألوان إلى color.

مثال:

التعليمة setpalette(0,lightcyan); تجعل أول لون في لوحة الألوان هو lightcyan. يمكن أن يتغير colorNum من 0 إلى 15 إعتماداً على سواقة الرسم ونمط الرسم قيد الاستخدام. إذا تم تمرير قيم غير مقبولة إلى setpalette فستعيد الدالة GraphResult.

القيمة grerror وتبقي الألوان في لوحة الألوان بدون تغيير. التغيرات التي تتم على اللوحة يمكن مشاهدتها على الشاشة مباشرة.

مثال:

البرنامج التالي يرسم خطوط مستقيمة على الشاشة بعدد الألوان المتوفرة في لوحة المفاتيح ثم يستدعي الإجراء setpalette لتغيير ألوان اللوحة بصورة

عشوائية. في كل مرة يتم بها تغيير لون من اللوحة فإن ظهور هذا اللون يتم تغييره إلى قيمة اللون الجديدة. حاول تنفيذ البرنامج على جهازك وراقب تغير الألوان.

```
program setplt;
  {Sample code for the SetPalette procedure.}
uses Crt, Graph;
var
  GraphDriver, GraphMode: Integer;
  Color: Word;
  Palette:PaletteType;
Begin
  GraphDriver:=Detect;
  InitGraph(GraphDriver, GraphMode,'c:\tp\bgi');
  If(GraphResult <> grOk) then
     Halt(1);
  GetPalette(Palette);
  If(Palette.Size <> 1) then
    Begin
       For Color:=0 to Pred(Palette.Size) do
         Begin
           SetColor(Color);
           Line(100, Color * 30, 200, Color*30);
         End;
       Randomize;
       Repeat
          Setpalette(Random(Palette.Size),Random(
          Palette.Size));
       Until KeyPressed;
    End
  Else
     Line(0, 0, 100, 0);
```

```
  Readln;
  CloseGraph;
End.
```

13. الإجراء getdefaultpalette

وصيغته العامة بلغة الباسكال كما يلي:

Procedure getDefualtpallet (var palette : paletteType);

تعيد هذا الإجراء بيانات تخص الحجم واللون للوحة الألوان المتاحة، تخزن النتائج في المتغير Palette الذي هو من النوع PaletteType المعرف مسبقاً في النظام.

14. الإجراء setLineStyle

صيغته العامة بلغة الباسكال هي كما يلي:

procedure setLinestyle (lineStyle : word; Pattern : word; thickness : word);

يستخدم هذا الإجراء لتحديد شكل الخط المطلوب لرسم الأشكال بواسطة الإجراءات line، drawpoly، rectangle، lineto وذلك بإعطاء قيم للبارامترات:

Linestyle: ويمثل أسلوب رسم الخط ويمكن أن يأخذ أي قيمة صحيحة بين 0 و 4 كما موضح في الجدول أدناه:

Name	Value	Description
SOLID_LINE	0	Solid line
DOTTED_LINE	1	Dotted line
CENTER_LINE	2	Centered line
DASHED_LINE	3	Dashed line
USERBIT_LINE	4	User-defined line style

Pattern: يمثل شكل الخط ويمكن أن يأخذ أي قيمة صحيحة. يظهر تأثير هذا البارمتر فقط عندما تسند القيمة الثابتة UserBitLn إلى البارامتر LineSty;e عدا ذلك فليس له أي تأثير.

Thickness : يمثل سمك الخط المستقيم ويمكن أن يأخذ أحد القيمتين 1، 3 حيث:

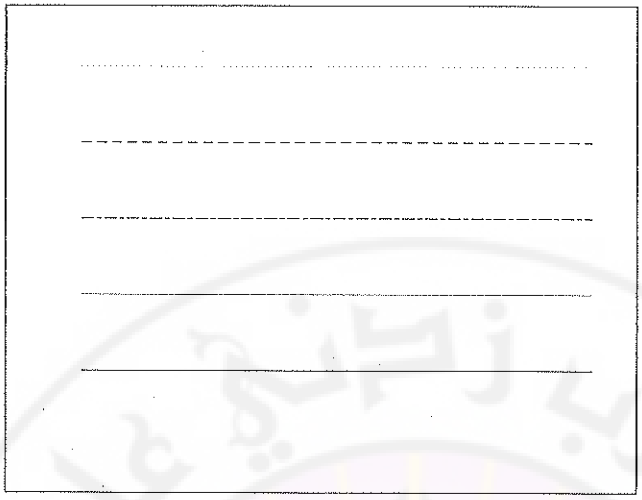
Name	Value	Description
NORM_WIDTH	1	1pixel wide
THICK_WIDTH	3	3pixels wide

1 يمثل السمك العادي، و 3 يمثل الخط السميك.

يستخدم هذا الإجزاء مع الإجزاءات line, lineto, rectangle, drawpoly و fillpoly. الإجراءات Pieslice, Arc , Circle تستخدم دائماً الخط الصلد SolidLn ولا تتأثر بشكل الخط الناتج عن هذا الإجراء.

الشكل أدناه يوضح خرج المقطع البرمجي التالي، حيث البارامتر LineStyle يأخذ القيم الصحيحة 0-4.

```
maxx := getmaxx;
maxy := getmaxy;
For i:= 0 to 4 do
  Begin
    setlinestyle(i,2,1);
     line(0,maxy div 2-i*20,maxx div 2 , maxy div 2 – i*20);
  End;
```



المقطع البرمجي التالي يسند قيمة ثابتة للبارامتر LineStyle ويعطي قيم متغيرة للبارامتر pattern من خلال حلقة الدوران Repeat-until.

يمكنك تجربة قيم أخرى للبارامتر pattern وملاحظة أشكال الخطوط التي ستحصل عليها في الخرج.

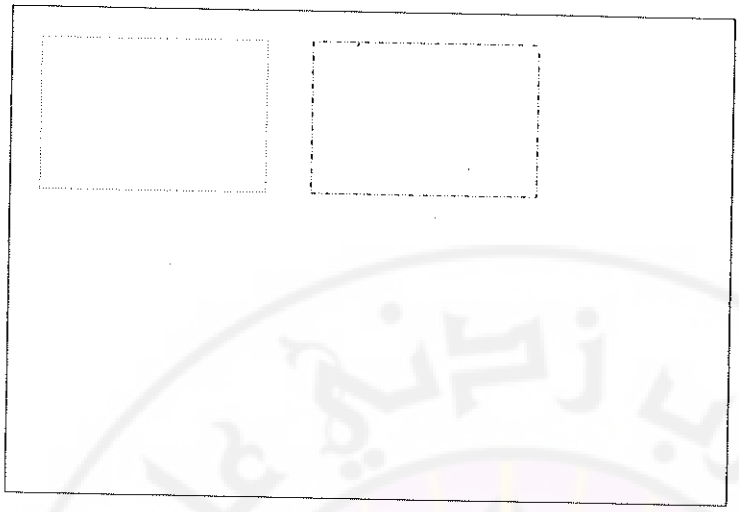
```
maxx := getmaxx;
maxy := getmaxy;
Repeat
   setlinestyle(userbitln,i,1);
   line(0,maxy div 2- 100 +i*10,maxx div 2 , maxy div 2 –
        100 + i*10);
Untile(i=29);
```

استبدل عبارة Line في كل من المقطعين السابقين بعبارة Circle ولاحظ عدم تأثر شكل الخط الذي سترسم به الدائرة بالإجراء setLineStyle مطلقاً.

مثال:

البرنامج التالي يرسم مستطيل باستخدام الإجراء Rectangle حيث يتم استبدال شكل الخط بالإجراء setLineStyle، لاحظ الخرج موضح في الشكل التالي للبرنامج.

```
program setinst1;
 {Sample code for the SetLineStyle procedure.}
uses Graph;
var
  Gd,Gm:Integer;
  X1,Y1,X2,Y2:Integer;
Begin
  Gd:=Detect;
  InitGraph(Gd,Gm,'c:\tp\bgi');
  If(GraphResult <> grOk) then
    Halt(1);
  X1:=10;
  Y1:=10;
  X2:=200;
  Y2:=150;
  SetLineStyle(DottedLn, 0, NormWidth);
  Rectangle(X1, Y1, X2, Y2);
  SetLineStyle(UserBitLn, $C3, ThickWidth);
  Rectangle(X1+230, Y1, X2+230, Y2);
   {Rectangle(Pred(X1), Pred(Y1), Succ(X2),
     Succ(Y2));}
  Readln;
  CloseGraph;
 End.
```



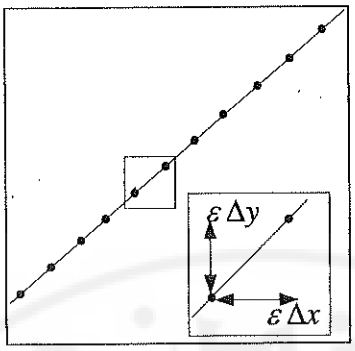
1.5 بعض التطبيقات الرياضية و الإحصائية باستخدام الرسم

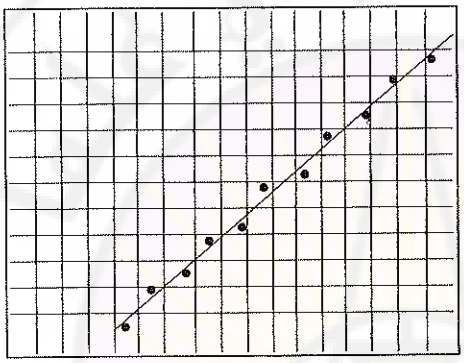
1.5.1 طريقة محلل الفروقات الرقمية البسيط The simple Digital Differential analyzer SDDA

في هذه الطريقة تعطى نقطة البدء للخط المستقيم (x_1, y_1) ونقطة النهاية (x_2, y_2) ثم تحسب قيم الفروق كمايلي:

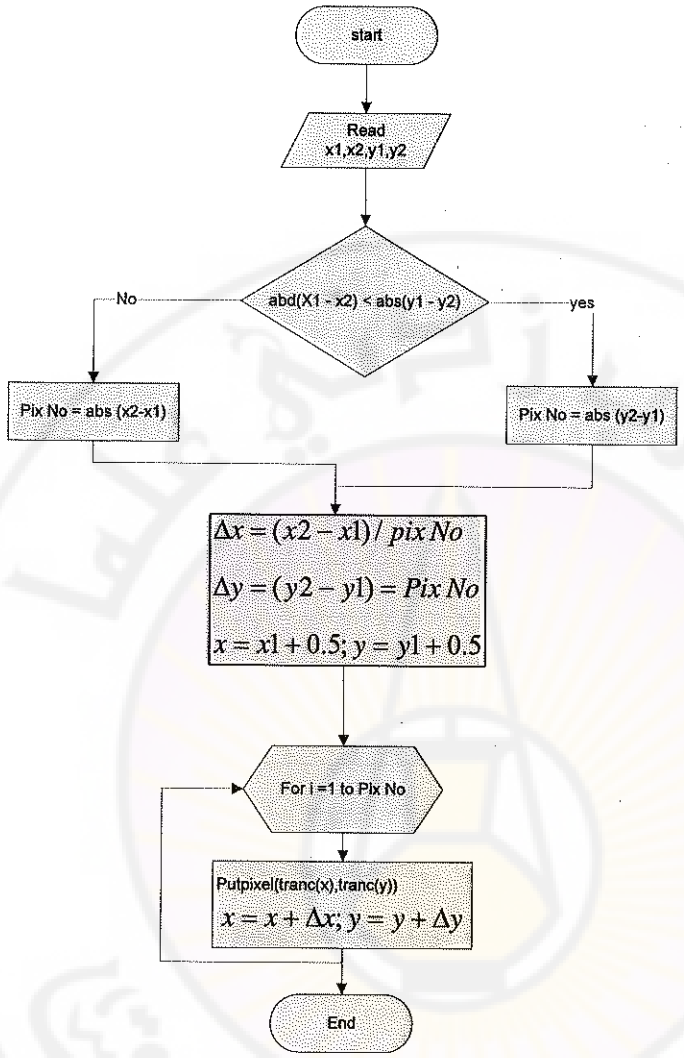
$$\Delta y = abs(y_2 - y_2) \quad \text{و} \quad \Delta x = abs(x_2 - x_1)$$

وتستخدم القيمة الأكبر لأي من Δx و Δy كتخمين لطول الخط المستقيم (عدد النقاط *Length* في البرنامج) ويتم توليد نقاط الخط المستقيم بتكرار زيادة قيمة x بمقدار $\Delta x/length$ وقيمة y بمقدار $\Delta y/Length$ إلى أنه يتم الوصول إلى آخر نقطة وهي *Length* (تمثل طول الخط المستقيم) بحيث أن لا تزيد قيم $\varepsilon\Delta x$ أو $\varepsilon\Delta y$ عن واحد، حيث ε هي أي مقدار صغير كما في الشكل التالي الأول. أي يتم توليد خطوة واحدة باتجاه الحركة الأكبر أي بزيادة مقدارها ∓ 1 لأحد الإحداثيتين كما في الشكل التالي الثاني.





تكون قيم x, y المولدة حقيقية وليست قيم صحيحة، وبما أنه لا يمكن رسم القيم الحقيقية على الحاسوب فيتم التقريب rounding إلى أقرب قيمة صحيحة بعد كل زيادة ورسم النقطة، ولكن تم استخدام البتر truncating هنا بدلاً من التقريب بعد إعطاء قيم ابتدائية لكل من x, y بمقدار 0.5 لتحقيق تقريب حقيقي وبذلك نمنع رسم نفس النقطة مرتين. ومن عيوب هذه الطريقة هي استخدام القيم الحقيقية التي تتطلب وقت معالجة كبير نسبياً بالإضافة إلى استخدام الدالة Tranc في لغة الباسكال، المخطط الانسيابي لهذه الطريقة موضح في المخطط التدفقي التالي:



البرنامج التالي يستخدم الخوارزمية المبينة في المخطط السابق و يمكن من رسم خط مستقيم بين كل زوج من النقاط الأربع التالية (كما هو موضح في الشكل التالي للبرنامج).

(maxX div 2, 0,maxX div 2,maxY)

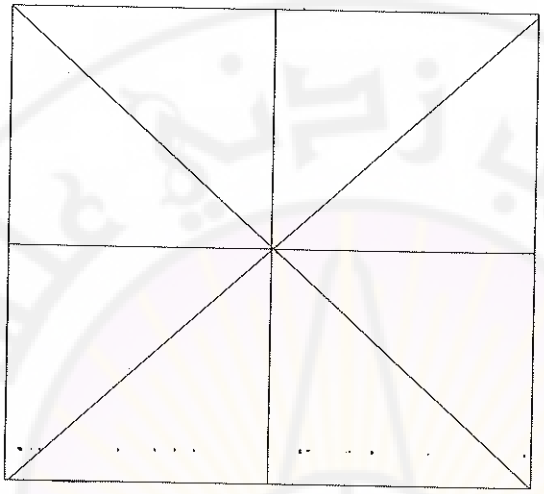
(0,MaxY div 2, maxX,maxY div 2)

(0,0,maxX,maxy)

(maxX,0,0,MaxY)

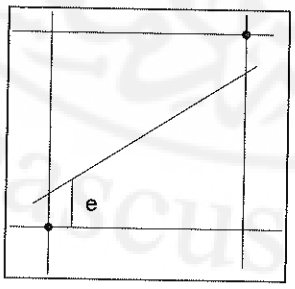
```
program SDDALine;
uses crt ,Graph;
var
  gm,gD,maxX,MaxY:integer;
procedure dda(x1,y1,x2,y2 :integer);
 var
   Length,i:integer;
   x,y,dx,dy:real;
 Begin
   Length:=abs(x2-x1);
   If(abs (y2 -y1) > length) then
      length:=abs(y2-y1);
   dx:=(x2-x1)/length;
   dy:=(y2-y1)/length;
   x:=x1 +0.5;
   y:=y+ 0.5;
   For i:=1 to length do
     Begin
        putpixel(trunc(x),trunc(y),white);
        x:=x+dx;
        y:=y+dy;
     End;
 End;
Begin {main program}
  Gd:=Detect;
  InitGraph(Gd, Gm, 'c:\tp\BGI');
  maxX:=getmaxX;
  maxY:=getmaxy;
  dda(maxX div 2,0,maxX div 2,maxY);
  dda(0,0,maxX,MaxY);
```

```
    dda(0,maxY div 2,maxX,maxY div 2);
    dda(maxX,0,0,maxY);
    Readln;
    CloseGraph;
End.
```



1.5.2 خوارزمية محلل الفروق الصحيحة (خوارزمية برزنهام) Digital Differential Integer Analyzer (DDIA)

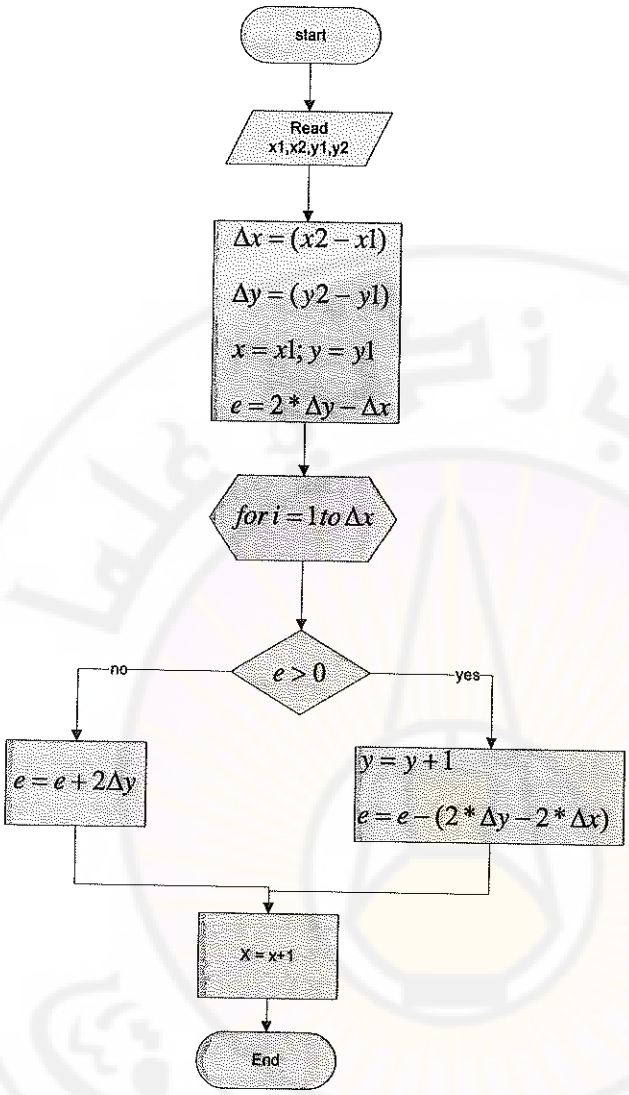
هذه الطريقة تشبه سابقتها SDDA حيث إنها مصممة لتغير قيمة أحد الإحداثيين بمقدار 1∓ في كل تكرار اعتماداً على قيمة (Error) e، حيث e تمثل المسافة العمودية بين النقطة المرسومة والإحداثي ذو التغير الأكبر (المسافة بين المسار الحقيقي للخط والنقطة الحقيقية المتولدة)، كما يبين الشكل التالي:



ففي كل تكرار يضاف الميل $\Delta x/\Delta y$ إلى e بعد أن يتم فحص قيمة e، فإذا كانت موجبة يعني أن المسار الحقيقي للخط يقع فوق النقطة الجارية لهذا يتم زيادة الإحداثي y وتقل قيمة e بواحد، أما إذا كانت e سالبة فتبقى قيمة y كما هي.

عيوب هذه الخوارزمية تكمن في الوقت المطلوب لعملية القسمة لحساب قيمة e، وكذلك عند زيادة قيمة e. ويمكن الاستغناء عن هذه القسمة بضرب e بثابت، وفحص إشارة e فقط. المخطط التدفقي في الصفحة التالية يوضح هذه الطريقة حيث لا يتضمن أي عملية قسمة ويتطلب الضرب بـ 2 فقط.

كما يمكن تبسيط الخوارزمية باستخدام حسابات الأعداد الصحيحة فقط. تصلح هذه الطريقة للتطبيق على المعالجات البسيطة.



1.5.3 بعض التحويلات الخطية

التحويلات الخطية هي أي تركيب من التحويلات التالية:

1. الإزاحة Translation.
2. التقييس scaling.
3. الدوران Rotation.

4. الانعكاس reflection.

5. القص shears.

سنقوم بدراسة بعض هذه التحويلات مثل الإزاحة و الدوران نظراً لأهميتها، يمكن دراسة التحويلات الأخرى بطريقة مماثلة.

1. إزاحة الأشكال الرسومية Translation of Graphics Objects

لإزاحة أي شكل رسومي، يجب أن تزاح كل نقطة (x, y) من النقاط التي تمثل الشكل إلى الموقع الجديد $(\bar{x}, \bar{y})$

$$(x, y) \rightarrow (\bar{x}, \bar{y})$$

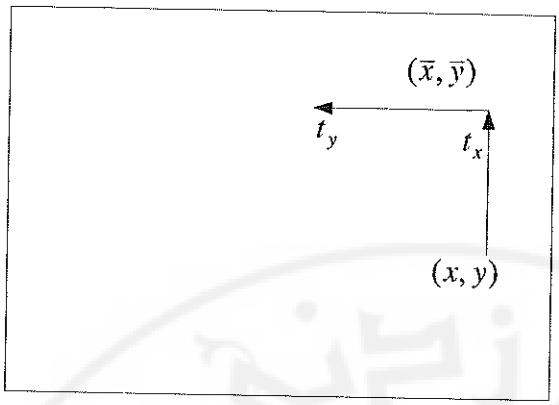
حيث:

$$\bar{x} = x + t_s$$

$$\bar{y} = y + t_y$$

إذا كانت قيمة t_x موجبة فيتم إزاحة النقطة إلى اليمين أما إذا كانت سالبة فيتم إزاحة النقطة إلى اليسار. عندما تكون $t_x = 0$ فيعني أن النقاط تتحرك موازية المحور الصادي.

إذا كانت t_y موجبة فيتم إزاحة النقطة إلى الأسفل أما إذا كانت سالبة فيتم إزاحة النقطة نحو الأعلى، وعندما تكون $t_y = 0$ فهذا يعني أن النقطة تتحرك بموازاة المحور السيني كما في الشكل التالي:



لإزاحة النقطة $p(x, y)$ إلى نقطة الأصل $(0,0)$ فإن ذلك يتم بطرح إحداثيات تلك النقطة من نفسها لتكون $p(x-x, y-y)$، الإجراء Translatepoint التالي بلغة الباسكال يوضح ذلك:

```
Procedure Translatepoint(x,y:integer; var
      px,py:integer);
   Begin
      px:=px-x;
      py:=py-y;
   End;
```

في أغلب الأحيان يكون المطلوب هو إزاحة الشكل أو عدة أشكال رسومية وليس نقطة واحدة. لعمل ذلك يجب إزاحة الشكل أولاً إلى نقطة الأصل نسبة إلى أحد النقط التي يتم تعريف الشكل أو الأشكال الرسومية بواسطتها ويتم ذلك بطرح إحداثيات تلك النقطة من إحداثيات كل نقطة من نقاط الشكل أي تطبيق الإجراء Translatepoint السابق على كل نقطة ليتم إزاحة كل النقاط بنفس المسافة ثم يمكن رسم الشكل باستخدام النقاط الجديدة التي تم حسابها من إجراء الإزاحة، فلو فرضنا بأننا نريد إزاحة الشكل المربع الذي رؤوسه النقاط $(x1, y1)$، $(x2, y2)$،

$(x3, y3)$، $(x4, y4)$ إلى موقع جديد فيجب أولاً إزاحته إلى نقطة الأصل نسبة إلى إحدى نقطه ولتكن النقطة $(x1, y1)$، أي نقاط الشكل الجديدة ستكون:

$$(x1-x1, y1-y1), (x2-x1, y2-y1), (x3-x1, y3-y1), (x4-x1, y4-x4)$$

ومن ثم إزاحته من نقطة الأصل إلى الموقع الجديد بإضافة قيم إحداثيات الموقع الجديد إلى قيم إحداثيات كل نقطة من نقطه.

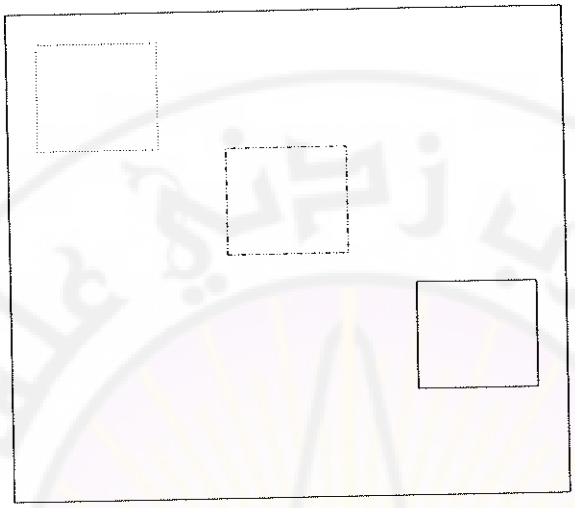
البرنامج التالي يوضح هذه العمليات حيث تتم الإزاحة من خلال الإجراء Translate الذي يتم استدعائه عدة مرات.

```
program translatsqure;
uses crt,Graph;
var
  Gd,Gm:Integer;
  x,x1,x2,x3,x4:integer;
  y,y1,y2,y3,y4:integer;
procedure Drawsqure(px1,py1,px2,py2,px3,
      py3,px4,py4: integer);
  Begin
     line(px1,py1,px2,py2);
     line(px2,py2,px3,py3);
     line(px3,py3,px4,py4);
     line(px4,py4,px1,py1);
  End;
procedure Translat(flag,x11,y11:integer;var
     px1,py1,px2,py2,px3,py3,px4,py4:integer);
  Begin
     px2:=px2-(x11 * flag);
     py2:=py2-(y11 * flag);
     px3:=px3-(x11 * flag);
     py3:=py3-(y11 * flag);
```

```
      px4:=px4-(x11 * flag);
      py4:=py4-(y11 * flag);
      px1:=px1-(x11 * flag);
      py1:=py1-(y11 * flag);
   End;
Begin {main program}
      Gd:=Detect;
      InitGraph(Gd, Gm,'c:\tp\bgi');
      x1:=300;    y1:=200;
      x2:=390;    y2:=200;
      x3:=390;    y3:=290;
      x4:=300;    y4:=290;
        {Draw the squre in its orignal position}
      Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
         {translate the squre to the origion
          Point (0,0)}
      Translat(1,x1,y1,x1,y1,x2,y2,x3,y3,x4,y4);
       {redraw the squre inthe origion
        point(0,0)}
      setLinestyle(1,1,1);
      Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
      x:=150; y:=120;
        {translate the squre to the origion
         Point (150,120)}
      Translat(-1,x,y,x1,y1,x2,y2,x3,y3,x4,y4);
       {redraw the squre in the new postion}
      setLinestyle(1,1,1);
      Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
      Readln;
      CloseGraph;
```

End.

لاحظ خرج البرنامج السابق في الشكل التالي حيث تم إزاحة المربع بدون أي تغير في الحجم أو الشكل.



كذلك يمكن إجراء عملية الإزاحة باستخدام مصفوفة الإزاحة، لإزاحة المربع السابق إلى أي موقع يتم أولاً إزاحته إلى نقطة الأصل نسبة لإحدى نقطه ولنفرضها النقطة (x, y) وذلك بطرح المصفوفة التالية من مصفوفة الشكل الأصلية:

$$\begin{bmatrix} x1 & y1 \\ x1 & y1 \\ x1 & y1 \\ x1 & y1 \end{bmatrix}$$

لاحظ أن أسطر هذه المصفوفة متشابهة لكي يتم إزاحة كل نقطة من نقاط الشكل بنفس المسافة، عدا ذلك فإن أي اختلاف في أسطر هذه المصفوفة سيؤدي إلى تشويه في الشكل الناتج.

وبعد إتمام عملية الإزاحة هذه تجمع المصفوفة الناتجة مع مصفوفة الإزاحة (والتي تتكون أسطرها من إحداثيات الموقع المراد إزاحة الشكل إليه) والمصفوفة الناتجة تمثل مصفوفة نقاط الشكل الجديدة بعد إزاحتها إلى الموقع المطلوب، أي إن تسلسل العمليات يكون كالتالي:

(مصفوفة الشكل الجديدة) = (مصفوفة الإزاحة) + (مصفوفة الشكل بعد إزاحتها إلى نقطة الأصل).

حيث يتم جمع كل عنصر في المصفوفة إلى العنصر المناظر له في المصفوفة الأخرى. البرنامج التالي يستخدم هذه الطريقة لإزاحة المربع في المثال السابق إلى نفس النقاط السابقة.

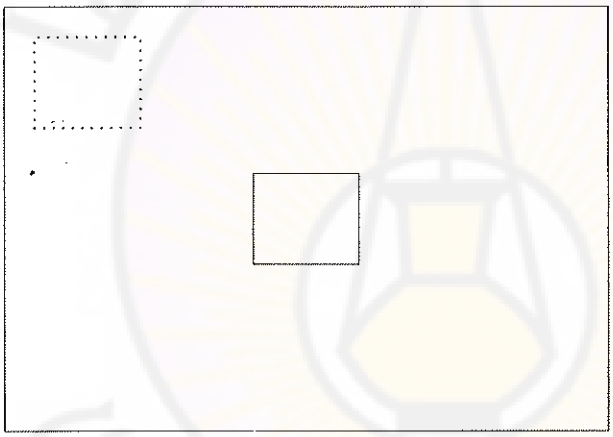
```
program translatsqure2;
uses crt , Graph;
type
    polytype=array[1 .. 6] of pointtype;
var
  Gd,Gm:Integer;
  poly,poly1:polytype;
  x,y,n1:integer;
procedure readpoly(var pobject : polyType);
   {to read all the points which defin the object
   of type polytype, it could be read dirctly
   from the keybord or from a data file}
  Begin
     pobject[1].x:=300; pobject[1].y:=200;
     pobject[2].x:=390; pobject[2].y:=200;
     pobject[3].x:=390; pobject[3].y:=290;
     pobject[4].x:=300; pobject[4].y:=290;
     pobject[5].x:=300; pobject[5].y:=200;
```

```
    drawpoly(5,pobject);
  End;
procedure Transpoly(flag:integer; var pobject:
        polytype; px,py,n:integer);
var
   i:integer;
  Begin
    For i:=1 to n do
      Begin
        pobject[i].x:=pobject[i].x-(px*flag);
        pobject[i].y:=pobject[i].y-(py*flag);
      End;
    drawpoly(n,pobject);
  End;
Begin {main program}
     Gd:=Detect;
     InitGraph(Gd, Gm,'c:\tp\bgi');
     readpoly(poly);
     n1:=5;
     x:=300; y := 200;
       {translate the squre to the origion
        Point (0,0)}
     setLinestyle(1,1,1);
     Transpoly (1,poly,x,y,n1);
       {translate the squre to its original
        postion}
     setLinestyle(0,1,1);
     Transpoly(-1,poly,x,y,n1);
     Readln;
     CloseGraph;
```

End.

لاحظ أننا عرفنا الشكل من النوع polytype وذلك لكي يكون البرنامج عام لأي شكل متعدد الأضلاع، وقد ساعد استخدام حلقة الدوران for هنا في إجراء الإزاحة Transpoly على جعل هذا الإجراء قصير وسهل الفهم. كما يجب ملاحظة أننا أزحنا المربع إلى نقطة الأصل نسبة إلى أول نقطة فيه وهي $x = 300, y = 200$ ثم أعدناه إلى موقعه الأصلي بإضافة هذه النقطة لكل من نقاطه، ويمكننا إعادته إلى أي موقع آخر ترغب به وذلك بإضافة قيم x, y لذلك الموقع لكل نقطة من نقاطه.

خرج البرنامج السابق سيكون على الشكل التالي:



2. الدوران Rotation

من المفيد أحيانا تدوير شكل رسومي حول نقطة معينة تسمى نقطة الارتكاز أو محور معين يسمى المركز بزاوية معينة θ ولأجل أن يتم التدوير حول هذه النقطة يجب أن يرجع الشكل إلى نقطة الأصل أي أن يزاح الشكل إلى نقطة الأصل بالنسبة لنقطة الارتكاز. يتم تدوير الشكل بالنسبة إلى نقطة الأصل وذلك بتدوير كل نقطة من النقاط التي تمثله.

فلو فرضنا إن إحدى نقاطه هي $p(x, y)$ يراد تدويرها بزاوية θ فإن:

$$(x, y) \rightarrow (\overline{x}, \overline{y})$$

بعد إزاحتها إلى نقطة الأصل حيث:

$$\overline{x} = x - x_p$$

$$\overline{y} = y - y_p$$

ثم يتم تدوير $p(\overline{x}, \overline{y})$ لتعطينا نقطة جديدة $p(\overline{\overline{x}}, \overline{\overline{y}})$ حيث:

$$\overline{\overline{x}} = (x - x_p)\cos\theta - (y - y_\theta)\sin\theta$$

$$\overline{\overline{y}} = (y - y_p)\cos\theta + (x - x_p)\sin\theta$$

حيث θ يجب أن تكون بالزوايا النصف قطرية وتكون مضاعفاتها $2\pi/2$.

ثم يتم إعادة إزاحته إلى موقعه الأصلي بعد التدوير لتعطينا النقطة $p(\overline{\overline{\overline{x}}}, \overline{\overline{\overline{y}}})$ حيث:

$$\overline{\overline{\overline{x}}} = (x - x_p)\cos\theta - (y - y_\theta)\sin\theta + x_p$$

$$\overline{\overline{\overline{y}}} = (y - y_p)\cos\theta + (x - x_p)\sin\theta + y_p$$

حيث يكون التدوير باتجاه عقرب الساعة ولتدويره عكس عقرب الساعة يجب فقط تغير الزاوية إلى $-\theta$ بدلاً من θ.

الإجراء rotatpoint في البرنامج التالي يوضح عملية دوران نقطة، ثم يتم استدعائه بعدد النقاط التي تعرف الشكل المطلوب تدويره.

سنستخدِم هذا الإجراء لتدوير المربع المستخدم في البرامج السابقة بزاوية مقدارها 30 درجة لاحظ الخرج الذي يبين موقع المربع الأصلي ثم موقعه بعد الإزاحة إلى نقطة الأصل ثم تدويره وإرجاعه إلى موقعه الأصلي (الشكل التالي للبرنامج).

```
program rtatesqure;
uses crt , Graph;
var
  Gd, Gm: Integer;
```

```
  x,x1,x2,x3,x4 : integer;
  y,y1,y2,y3,y4 : integer;
  alpha:real;
procedure Drawsqure(px1,py1,px2,py2,px3,py3,
          px4,py4:integer);
   Begin
     line(px1,py1,px2,py2);
     line(px2,py2,px3,py3);
     line(px3,py3,px4,py4);
     line(px4,py4,px1,py1);
   End;
procedure Translat(flag,x11,y11:integer; var
      px1,py1,px2,py2,px3,py3,px4,py4:integer);
  Begin
     px2:=px2-(x11 * flag);
     py2:=py2-(y11 * flag);
     px3:=px3-(x11 * flag);
     py3:=py3-(y11 * flag);
     px4:=px4-(x11 * flag);
     py4:=py4-(y11 * flag);
     px1:=px1-(x11 * flag);
     py1:=py1-(y11 * flag);
  End;
procedure rotatepoint(alpha:real; var x,y:
        integer);
  var
     xtemp,ytemp,sine,cosine:real;
   Begin
     xtemp:=x; ytemp:=y;
     cosine:=cos(alpha * pi / 180);
```

```
    sine:=sin(alpha * pi / 180);
    x:=round(cosine * xtemp - sine * ytemp);
    y:=round(sine * xtemp + cosine * ytemp);
  End;
Begin {main program}
    Gd:=Detect;
    InitGraph(Gd, Gm, 'c:\tp\bgi');
    x1:=300; y1:=200;
    x2:=390; y2:=200;
    x3:=390; y3:=290;
    x4:=300; y4:=290;
    x:=300;  y:=200;
    alpha:=30;
      {Draw the squre in first position}
    Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
     {translate the squre to the point (0,0)
      whith respect to the point (x1,y1) by
      subresents this point from each of the
      points which represents the squre}
    Translat (1,x1,y1,x1,y1,x2,y2,x3,y3,x4,y4);
    {redraw the squre inthe origion point(0,0)}
    Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);

      {rotat the squre in the position (0,0)}
    rotatepoint(alpha,x1,y1);
    rotatepoint(alpha,x2,y2);
    rotatepoint(alpha,x3,y3);
    rotatepoint(alpha,x4,y4);
    {redraw the squre inthe origion point(0,0)}
    Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
```

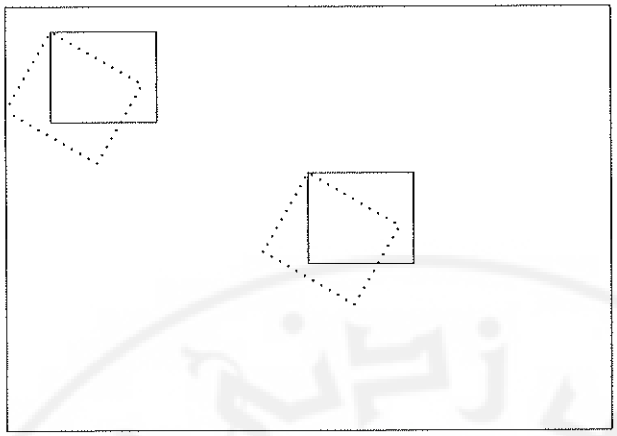
```
        {translate the squre to its original
         postion by adding (x1,x2) to each point
         of the squre after scaling}
       Translat (-1,x,y,x1,y1,x2,y2,x3,y3,x4,y4);
        {redraw the squre in the new postion}
       Drawsqure(x1,y1,x2,y2,x3,y3,x4,y4);
       Readln;
       CloseGraph;
End.
```

كما يمكن استخدام مصفوفة الدوران لتدوير شكل رسومي بزاوية مقدارها θ حيث يجب إزاحة الشكل المراد تدويره حول نقطة معينة (نقطة الارتكاز) إلى نقطة الأصل كما مر معنا سابقاً، ثم تدويره وذلك بضرب مصفوفة الشكل بمصفوفة الدوران التالية:

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

حيث تقاس θ بالزوايا النصف قطرية. المصفوفة السابقة تؤدي إلى تدوير الجسم باتجاه عقرب الساعة وإذا كان تدويره بعكس عقرب الساعة فيجب ضرب مصفوفة الجسم بمصفوفة الدوران التالية:

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$



البرنامج التالي يعمل على تدوير أي شكل رسومي بواسطة الإجراء RotatPoly، حيث تم تعريف الشكل بمصفوفة من النوع polytype وقد تم تحديد حجمها هنا بستة فقط ويمكن للمستخدم تغيير ذلك حسب حاجته.

البارامتر الأول flag يستخدم لتحديد اتجاه التدوير ويمكن أن يأخذ أحد القيمتين 1 لتدوير الشكل باتجاه عقرب الساعة أو القيمة 1- لتدويره بعكس عقرب الساعة.

البارامتر theta تمثل مقدار الزاوية المراد تدوير الشكل بها وتعطى بالدرجات ويتم تحويلها إلى الزاوية نصف القطرية في الإجراء. n تمثل عدد نقاط رؤوس الشكل.

```
program rotatesqure2;
uses crt , Graph;

type
    polytype=array[1..6] of pointtype;
var
  Gd,Gm:Integer;
  poly,poly1:polytype;
  x,y,n1:integer;
```

```
procedure readpoly(var pobject:polyType);
  {to read all the points which defin the object
   of type polytype, it could be read dirctly
   from the keybord or from a data file}
 Begin
     pobject[1].x:=300; pobject[1].y:=200;
     pobject[2].x:=390; pobject[2].y:=200;
     pobject[3].x:=390; pobject[3].y:=290;
     pobject[4].x:=300; pobject[4].y:=290;
     pobject[5].x:=300; pobject[5].y:=200;
     drawpoly(5,pobject);
 End;
procedure Transpoly(flag:integer; var pobject:
      polytype; px,py,n:integer);
  var
     i:integer;
 Begin
     For i:=1 to n do
       Begin
          pobject[i].x:=pobject[i].x-(px*flag);
          pobject[i].y:=pobject[i].y-(py*flag);
       End;
     drawpoly(n,pobject);
 End;
procedure rotatepoly(flag:integer; var rpobject:
 polytype;pobject:polytype;theta:real;n:integer);
   var
     i:integer;
 Begin
     theta:=theta*pi / 180;
```

```
    For i:= 1 to n do
      Begin
        rpobject[i].x:=round(pobject[i].x*
         cos(theta)+pobject[i].y*(
         1*flag*sin(theta)));
        rpobject[i].y:=round(pobject[i].x*flag*
         sin(theta)+pobject[i].y*(cos(theta)));
      End;
    drawpoly(n,rpobject);
End;
Begin {main program}
      Gd:=Detect;
      InitGraph(Gd, Gm,'c:\tp\bgi');
      readpoly(poly);
      n1:=5;
      x:=300; y := 200;
       {translate the squre to the origion point
        (0,0)}
      Transpoly(1,poly,x,y,n1);
       {scale the squre by multiply each of it's
        points with the scaling factor matrix
        which is equal 3 here}
      rotatepoly(-1,poly1,poly,30,5);
       {translate the squre to its original
        postion}
      Transpoly(-1,poly1,x,y,n1);
      Readln;
      CloseGraph;
End.
```

4. إنشاء الوحدات Unit Creation

تعرفنا في الفقرات السابقة على وحدة الرسم بشيء من التفصيل. وقد وجدنا أن الوحدات مهمة للغاية كونها تُمكننا من استخدام الكثير من الإجراءات بدون الحاجة لكتابة الكود البرمجي لهذه الإجراءات في برنامجنا. يكفي أن نضع عبارة تضمين الوحدة في بداية البرنامج لنتمكن من استدعاء كل الإجراءات و الدوال الموجودة في هذه الوحدة. لهذا الأمر عدة فوائد، يُمكن أن نذكر منها مايلي:

1. الفائدة الأولى أنها تُمكّننا من كتابة برامج صغيرة الحجم (عدد أسطر التعليمات ليس كبيراً) مما يجعل عملية اكتشاف الأخطاء و التأكد من صلاحية البرنامج للمسألة التي كُتِب لحلها سهلاً نسبياً.
2. الفائدة الثانية للوحدات أن تسمح بإعادة استخدام الاجراءات و الدوال في كل برنامج يحتاج لها. مثلاً: لنفترض أن لدينا وحدة تحوي مجموعة من الدوال و الإجرائيات لحساب مختلف مقاييس النزعة المركزية و التشتت و اختبار الفرضيات. عندئذٍ في أي برنامج يحتاج مثلاً لحساب المتوسط الحسابي و الإنحراف المعياري يكفي أن نضع عبارة تضمين لهذه الوحدة في البرنامج و نستدعي هذه الإجرائيات و الدوال بعد تمرير قيمة مناسبة لوسطائها.
3. الفائدة الثالثة أن الوحدة تُمكن من تحقيق خاصة التغليف كما رأينا في الكائنات. يتم ذلك من خلال فصل واجهة الاتصال بالوحدة (Interface) عن كيف تقوم الوحدة بعملها (التحقيقImplementation). هذا يعني أن البرامج التي تستَخدم هذه الوحدة لاتتأثر بأي تعديل على كيفية عمل الوحدة طالما أن أسماء الدوال و الإجراءات ووسطائها لم يتغير. استجابةً لهذه الأهمية التي تملكها الوحدات، فقد ارتأينا تعريف القارئ بكيفية إنشاء الوحدات بصورة عامة و التعامل معها فيما تبقى من هذا الفصل.

1.4 تعريف الوحدة Unit Definition

الوحدة في لغة الباسكال هي برنامج قائم بحد ذاته يُشبه كثيراً برنامج الباسكال العادي. الفرق بينها و بين أي برنامج أخر مكتوب بلغة الباسكال هو أنها لا يمكن تنفيذ الوحدة بل يمكن فقط استدعاء دوالها و إجراءاتها و استخدام الحقول المُعرَّفة فيها. كما تختلف الوحدة أيضاً بطريقة ترجمتها (Compiling) حيث أنها تُترجَم على القرص بدلاً من أن تُترجَم في الذاكرة كما هو الحال في البرنامج العادي. هناك أيضاً اختلافات طفيفة ببعض التراكيب النحوية كما سنرى.

التركيب النحوي للوحدة يمكن كتابته على الشكل التالي:

```
Unit UnitName;
Interface
{تعريف مجموعة من الدوال و الإجراءات و الحقول. فقط ترويسة الدالة
أو الإجراء و لا تُكتَب أي تعليمات أو كلمات مفتاحيه مثل Begin أو
End}

Implementation
{كتابة التعليمات البرمجية لكل دالة أو إجراء كُتِب في القسم
Interface. يُمكن تعريف دوال و إجراءات جديدة هنا و لكن يجب
الانتباه إلى أن هذه الدوال و الإجرائيات ستكون خاصة بالوحدة و لا
يمكن لأي برنامج خارج الوحدة استخدامها (أي تصبح خاصة بالوحدة و
يتم الوصول إليها من داخل الوحدة فقط. كذلك يُمكن تهيئة المتحولات
المُعرَّفة و ذلك بإعطائها قيم بدائية.}

End.
```

كما نلاحظ من التركيب النحوية فإن الوحدة تبدأ بالكلمة المفتاحية Unit و هي تُشير لمترجم الباسكال أن هذا البرنامج هو وحدة و ليس برنامج باسكال عادي. يجب وضع اسم للوحدة بعد الكلمة المفتاحية Unit. يخضع اسم الوحدة

إلى قواعد تسمية المتحولات بلغة الباسكال و يُفَضَّل أن يكون هذا الاسم مُعبّر عن عمل الوحدة.

تحوي الوحدة أيضاً على الكلمة المفتاحية Interface و فيها تُعرف الدوال و الإجرائيات دون كتابة التعليمات البرمجية لهذه الإجراءات و الدوال (أي فقط ترويسة الدالة أو الإجرائية). يُمكن أن يُعَرف في هذه الجزء أيضاً أنواع، متحولات، أو ثوابت. ننوه هنا إلى أن جميع الدوال و الإجرائيات غيرها المعرفة في هذا الجزء هي عامة و يمكن للبرامج التي تستَخدم هذه الوحدة أن تصل إليها و تتعامل معها.

بعد الكلمة المفتاحية Implementation تتم كتابة التعليمات البرمجية لكل دالة أو إجراء تم تعريفه في الجزء Interface. يمكن أيضاً كتابة دوال و إجرائيات جديدة لم تُذكر ترويستها في الجزء Interface و لكن يجب الانتباه إلى هذه الدوال و الإجرائيات الجديدة يمكن استخدامها فقط داخل الوحدة و لا يُمكن لأي برنامج أخر الوصول إليها لأنها خاصة بالوحدة. يُمكن أيضاً تعريف أنواع و متحولات و ثوابت في هذا الجزء و لكنها تبقى خاصة و لا يمكن لأي برنامج آخر الوصول إليها. بالإضافة إلى كتابة التعليمات البرمجية للدوال و الإجرائيات، يُمكن أيضاً إعطاء قيم ابتدائية للمتحولات إذا أردنا ذلك. يجب الانتباه إلى أنه إذا أردنا إعطاء قيم ابتدائية للمتغيرات فيجب أن يتم ذلك بعد الكلمة المحجوزة Begin (و هنا لا توجد End بعد Begin).

بعد هذا يجب إنهاء الوحدة بالكلمة المفتاحية End كما نفعل في برنامج الباسكال. بشكل أيضاً مغاير لبرنامج الباسكال، لا تبدأ الوحدة بالكلمة المفتاحية Begin. لتوضيح فكرة الوحدة سنطرح المثال التالي.

مثال:

أنشئ وحدة تقوم بتوصيف بعض الإجراءات الإحصائية (مقاييس النزعة المركزية).

```
Unit StatOps;
Interface
     Type
          Sample=array[1..100] of real;
     Procedure ComputeMean(A: Sample, Size:
           integer;var Mean:real);
     Function ComputeStdev(B:Sample,Size:
           integer):real;
Implementation
     Procedure ComputeMean(A:Sample,Size:
              integer; var Mean: real);
        Var i: integer;
       Begin
          Mean:=0.0;
          For i:=1 to Size do
             Mean:=Mean+A[i];
          Mean:=Mean/Size;
       End;
     Function ComputeStdev(B:Sample,Size:
            integer):real;
        Var i: integer;
          Stdev, Mean: real;
       Begin
          Stdev:=0.0;
          ComputeMean(B,Size,Mean);
          For i:=1 to Size do
           Stdev:=Stdev+(B[i]-Mean)*(B[i]- Mean);
           ComputeStdev:= Sqrt(Stdev/(Size -1));
       End;
     Var unitName:String;
```

```
Begin {تهيئة المتحولات تبدأ من هنا}
    unitName:='StatOps';

End. {Unit}
```

توَفر هذه الوحدة نوع بيانات عبارة عن متجهة مؤلفة من 100 عنصر من النوع الحقيقي و توفر أيضاً الإجراء ComputeMean و الدالة ComputeStdev. هذان البرنامجان الجزئيان عامان لأنه تم تعريفهما في الجزء Interface. لاحظ هنا أنه في هذا القسم تم تعريف ترويسة كل منهم فقط. في القسم Implementation تم كتابة التعليمات البرمجية للإجراء و الدالة. كذلك تم تعريف المتحول unitName من النوع String ضمن الجزء implementation. بسبب تعريف المتحول في هذا الجزء فإن هذا المتحول يُعَد متحولاً خاصاً و لا يمكن لأي برنامج خارج الوحدة الوصول إليه أو التعامل معه بشكل مباشر. تم أيضاً وضع قيمة بدائية للمتحول "StatOps". لاحظ أن إعطاء قيمة للمتحول تمت بعد الكلمة المفتاحية Begin.

2.4 استخدام الوحدات الأخرى ضمن الوحدة Using Other Units within a Unit

يُمكن للوحدة استخدام وحدات أخرى للاستفادة من خدماتهم. يُمكن تضمين أي وحدة ضمن الوحدة التي تم إنشاؤها في القسم Interface أو القسم Implementation، و لكن يجب تضمين الوحدة تماماً بعد الكلمة المفتاحية Interface أو الكلمة المفتاحية Implemetation. مثلاً: إذا أردنا تضمين الوحدة Crt ضمن وحدة مُعيَّنة و في الجزء Interface، فإننا نكتب بعد الكلمة المفتاحية Interface مباشرةً تعليمة التضمين uses Crt;. في هذه الحالة تُصبح كل من إجرائيات و دوال الوحدة Crt قابلة للاستخدام في أي مكان من الوحدة التي تم إنشاؤها. لاحظ أن كتابة التعليمة نفسها مباشرة بعد الكلمة المفتاحية

Implementation يحصر استخدام إجرائيات و دوال Crt فقط في الجزء Implementation.

3.4 ترجمة الوحدة Compiling a Unit

يجب ترجمة الوحدة التي تم إنشاؤها قبل إمكانية استخدامها. ترجمة الوحدة تتم على القرص المُخزَنة عليه و ليس في الذاكرة. لترجمة الوحدة على القرص يجب أولاً فتح ملف الوحدة إذا لم يكن مفتوحاً مسبقاً. بعد ذلك يجب تغيير مكان تخزين الملف الناتج عن الترجمة من الذاكرة إلى القرص. يُمكن القيام بذلك من خلال فتح القائمة Compile و من ثم الضغط على المفتاح D (من لوحة المفاتيح). بعدها نقوم بترجمة الوحدة كما نُترجم أي برنامج أخر مكتوب بلغة الباسكال. بعد الانتهاء من ترجمة الوحدة و تصحيح الأخطاء إن وجدت، يمكن إعادة وضع المترجم إلى الوضع الافتراضي (الذاكرة) من خلال فتح القائمة Compile و الضغط على المفتاح D.

يمكن استخدام الوحدة بعد ترجمتها كما تستخدم أي وحدة أخرى. فقط ضع علامة تضمين المكتبة مباشرةً في بداية برنامج الباسكال.

أمثلة :

مثال 1:

لنحاول كتابة وحدة تقوم بتوصيف مجموعة من الإجرائيات و الدوال التي تُمكّن من قراءة بعض القيم التي لا يُمكِن قراءتها بشكل مباشر من خلال لغة الباسكال (مثل النوع البولياني). بشكل خاص سنكتب وحدة تقوم بقراءة قيمة بوليانية من خلال الإجراء ReadBool المُعَرَف ضمن الوحدة و بقراءة عدد صحيح ضمن مجال مُحدَّد بقيمتين من خلال الإجراء EnterInt.

```
Unit EnterData;
Interface
     Procedure ReadBool(Var BoolVal:boolean);
```

```
{يقوم الإجراء السابق بقراءة قيمة بولياية}
Procedure EnterInt(MinN,MaxN:integer;var N:
          integer);
{يقوم الإجراء السابق بقراءة عدد صحيح ضمن مجال مدخل بين
                                  القيمة MinN و القيمة MaxN}
Implementation
     Procedure ReadBool(Var BoolVal:boolean);
          Var NextChar:Char;
        Begin
          Repeat
               Write('type T or F>');
               Readln(NextChar);
               NextChar:=UpCase(NextChar);
               BoolVal:=(NextChar='T');
          Until((NextChar='T')or(NextChar='F'));
        End;
     Procedure EnterInt(MinN,MaxN:integer;var N:
            integer);
          Var IntRange:boolean;
        Begin
          InRange:=false;
          If(MinN>MaxN) then
            Begin
               Writeln('Error-Empty range');
               InRange:=true
            End;
          While(not(InRange)) do
            Begin
               Write('Enter an Integer from');
```

```
            Write(MinN,'to',MaxN,'>');
            Readln(N);
            InRange:=((N>=MinN)and(N<=MaxN))
        End; {while}
End. {Unit EnterData}
```

بعد كتابة الوحدة يجب ترجمَتها كما نوهنا سابقاً. لإعطاء القارئ معلومات أكثر عن كيفية استخدام الوحدة في البرامج، سنقوم بكتابة برنامج بلغة الباسكال يستَخدِم الوحدة السابقة.

```
Program TestEnterData;
Uses EnterData; {تضمين الوحدة في البرنامج}
Var
    NextBool:boolean;
    Num:integer;
Begin {main program}
    ReadBool(NextBool); { استدعاء الإجراء من الوحدة}
    Writeln('Boolean Value read was',NextBool);
    EnterInt(5,-5,Num);
    EnterInt(-5,5);
    Writeln('Integer read was',Num);
End. {Main program}
```

يُمكن ترجمة هذا البرنامج و تنفيذه. عند تنفيذ هذا البرنامج نحصل على المخرجات التالية:

```
Type T or F> b
Type T or F> t
Boolean Value read was True
```

```
Error-Empty Range
Enter an integer from -5 to 5>0
Integer read was 0
```

مثال 2 :

اكتب وحدة تقوم بتوفير بعض الدوال لحساب مساحات بعض الأشكال الهندسية. للسهولة سوف نكتب دوال لحساب مساحات أربعة أشكال هندسية فقط دائرة Circle، مربع Square ، مستطيل Rectangle ، و مثلث Triangle.

```
unit Areas;
(*This unit includes a collection of functions to
calculate the*)
(*areas of four different geometric shapes.*)
interface
  function CircleArea(Radius:real):real;
  function SquareArea(Length_Of_Side:real):real;
  function RectangleArea(Length,Width:real):real;
  fnction TriangleArea(Base,Height:real):real;
implementation
  var My_Pi:real;
  procedure Mult_Two_Numbers(Number1,Number2:
          real;var Result:real);
     Begin
         Result:=Number1*Number2;
     End;
  Function CircleArea(Radius:real):real;
      var Rad_Squared:real;
    Begin
     Mult_Two_Numbers(Radius,Radius,Rad_Squared);
     CircleArea:=Rad_Squared* My_Pi;
```

```
    End;
  Function SquareArea(Length_Of_Side:real):real;
     Begin
      SquareArea:=Length_Of_Side* Length_Of_Side;
     End;
  Function RectangleArea(Length,Width:real):real;
    Begin
       RectangleArea:=Length*Width;
    End;
  Function TriangleArea(Base,Height:real):real;
    Begin
       TriangleArea:=Base*Height/2.0;
    End;
Begin
   (*This is the initialization code*)
   My_Pi:=3.14159267;
End.
```

مثال 3:

يبين هذا المثال كيفية استخدام الوحدة السابقة ضمن البرنامج.

```
program Calculate_Area_Of_Shapes;
uses Areas,Crt;
var
     In_Char:char;
     Length,Width,Height,Base,Radius:real;
Begin (* main program*)
  Repeat
      Writeln;
      Writeln('Please input the first letter of
          the selection');
```

```
Writeln('Select shape; Square Rectangle
    Triangle Circle Quit');
Write('Requested shape is');
Repeat until Keypressed;
In_Char:= ReadKey;
case UpCase(In_Char) of
 'S':
    begin
      Write('Square Enter length of side');
      Readln(Length);
      Writeln('Area=',SquareArea(Length));
    end;
 'R':
    begin
       Write('Rectangle Enter width');
       Readln(Width);
       Write('Enter height');
       Read(Height);
       Writeln('Area=',RectangleArea
          (Width,Height));
    end;

 'T':
    begin
       Write('Triangle Enter base');
       Readln(Base);
       Write('Enter height');
       Read(Height);
       Writeln('Area=',TriangleArea
         (Base,Height));
    end;
```

```
        'C':
           begin
              Write('Circle Enter radius');
              Readln(Radius);
              Writeln('Area=',CircleArea(Radius));
           end;

        'Q': Writeln('Quit');
        else Writeln('undefined entry');
      end;
   Until((In_Char='Q')or(In_Char='q'));
End. (*of main program*)
```

مثال 4:

توَضِح الوحدة التالية كيف نستَخدِم الكائنات ضمن الوحدة. سوف نُعَرِّف كائن يحتوي على اسم الشخص و رقم الهاتف. كذلك يحوي على الدوال و الإجراءات التي تقوم بمختلف العمليات على الحقول الموجودة ضمن الكائن.

```
Unit Entry;
Interface
    Type
       Entry=object
          Procedure init (AName,ANumber:String);
          Procedure ReadEntry(var InFile:Text;
             FileName:string);
          Function getName: String;
          Function getNumber: String;
          Function EqualTo (E: Entry): Boolean;
          Procedure DisplayEntry;
        Private
          Name,Number:string;
```

```
        End;
Implementation
   Procedure Entry.Init(AName,ANumber:String);
      Begin
        Name:=AName;
        Number:=ANumber;
      End;
   Procedure Entry.ReadEntry(var InFile:Text;
           FileName:string);
      Begin
          If(FileName='') then
            Begin
               Write('Name >');
               Readln(Name);
               Write('Number >');
               Readln(Number);
            End
          Else
            Begin
               Readln(InFile,Name);
               Readln(InFile,Number);
            End;
      End;
   Function Entry.getName:string;
      Begin
          getName:=Name;
      End;
   Function Entry.getNumber:string;
      Begin
          getNumber:=Number;
```

```
    End;
  Function Entry.EqualTo(E:Entry):boolean;
   Begin
    EqualTo:=((Name=E.Name)and(Number=E.Number);
   End;
  Procedure DisplayEntry;
    Begin
        Writeln(Name);
        Writeln(Number)
    End;
End. {Unit Entry}
```

تُعَّرف الوحدة السابقة نوع كائن يُسمى Entry حيث يوفر هذا الكائن مجموعة من الدوال و الإجراءات و هذه الدوال و الإجراءات عامة؛ يُمكن لأي برنامج استدعاؤهم. كذلك تم تعريف حقول خاصة Name, Number لا يُمكن لأي برنامج يستَخدِم الوحدة الوصول إليه بشكل مباشر. الطريقة الوحيدة للوصول إليهم هي عن طريق الدوال و الإجراءات. تم كتابة التعليمات البرمجية لكل دالة و إجراء في القسم المعنون بـ Implementation. لاحظ أن اسم كل دالة أو إجراء مسبوق باسم الكائن و ذلك بسبب تعريفهم ضمن الكائن.

إن عدم وضع اسم الكائن قبل اسم الدالة و الإجراء يُعطي خطأ أثناء الترجمة Compile Error.

لتوضيح كيف نستخدم الوحدة السابقة، فإننا سوف نكتب برنامجاً بسيطاً يقوم بذلك.

```
Program EntryClient;
Uses Entry;
Var
```

```
    AnEntry,Entry2:Entry;
    AName:string;
Begin {Main Program}
      AnEntry.Init('', '');
      Writeln('Enter a directory entry:');
      AnEntry.ReadEntry(Input,'');
      Write('Enter a name to match');
      Readln(AName);
      Entry2.Init(AName,'');
      If(AnEntry.EqualTo(Entry2) then
          Writeln('Number=',AnEntry.getNumber);
      Else
          Writeln('Name is not listed');
      AnEntry.DisplayEntry;
End.{EntryClient}
```

Damascus University

، برنامجاً بلغة الباسكال يقوم بتعريف الوحدة **MyMatrix** التي تُمَكّن مما

:

1. إدخال مصفوفة مربعة **A** من الأعداد الصحيحة الموجبة من البعد $n \times n$.
2. إيجاد منقول المصفوفة **A** و طباعتها.
3. إيجاد أصغر عنصر للمصفوفة **A** مع تحديد مكان وقوعه.
4. حساب المضاعف المشترك الأصغر لعناصر المصفوفة **A**.
5. بيان فيما إذا كانت المصفوفة **A** متناظرة.
6. طباعة جميع الأعداد الأولية الموجودة في المصفوفة A.

، وحدة اسمها Stat توَفر ما يلي:

1. إجراء لحساب المتوسط الحسابي والانحراف المعياري لعينة مدخلة.
2. إجراء لحساب الوسط، والربيعي الأول والثالث.
3. دالة لحساب المنوال.
4. دالة لاختبار فرضية معينة.
5. إجراء لحساب مجال ثقة.

، برنامجاً بلغة الباسكال يقوم بإدخال عينة ويستخدم الوحدة السابقة للقيام سابات التالية:

1. حساب معاملات النزعة المركزية للعينة.
2. حساب مجال ثقة حول متوسط العينة المدخلة.
3. اختبار فرضيات مختلفة حول متوسط العينة.

، جيداً الوحدة التالية و التي تُعَرِف كائناً اسمه Emp. **حدد جميع** طاء سواءً أكانت لغوية (compile errors) أم منطقية (run

تمارين

1) ضع برنامجاً بلغة الباسكال يقوم بما يلي:

1. كتابة وحدة (Unit) إسمها MObject ؛ Student يحتوي على الحقول و الإجراءا

الحقول: اسم الطالب، الكنية، السنة الد متجهة تحتوي على علامات المو

الإجراءات: إجراء لحساب و إعادة معدل و الكنية و السنة الدراسية لطالب

2. كتابة برنامج بلغة الباسكال اسمه Unit السابقة و يقوم بـ:

I. تعريف متجهة عدد عناصرها 10 من

II. طباعة أسماء الطلاب الذين تزيد معدل

III. طباعة اسم و كنية الطالب الأقدم من .

2) اكتب وحدة بلغة الباسكال اسمها adPrimitives

1. توّفر إجراء لقراءة قيمة بوليانية (lse
2. توّفر دالة لقراءة قيمة حقيقية ضمن ا
3. توّفر إجراء لقراءة قيمة من نوع بدائي
4. دالة للتحقق من أن شريط مُدخَل لا (I, A, e, u, o).
5. دالة للتحقق من أن عدد مُدخَل يقع ،

اكتب برنامج بلغة الباسكال يستخدم جميع الإجراءات السابقة.

تمارين

1) ضع برنامجاً بلغة الباسكال يقوم بما يلي:

1. كتابة وحدة (Unit) إسمها MObject توفر كائناً (Object) اسمه Student يحتوي على الحقول و الإجراءات التالية:

الحقول: اسم الطالب، الكنية، السنة الدراسية، سنة الدخول للجامعة، متجهة تحتوي على علامات المواد التي نجح بها الطالب.

الإجراءات: إجراء لحساب و إعادة معدل طالب، إجراء لإعادة الاسم و الكنية و السنة الدراسية لطالب معين.

2. كتابة برنامج بلغة الباسكال اسمه UseUnit يقوم باستخدام الوحدة السابقة و يقوم بـ:

I. تعريف متجهة عدد عناصرها 10 من النوع Student.

II. طباعة أسماء الطلاب الذين تزيد معدلاتهم عن 70.

III. طباعة اسم و كنية الطالب الأقدم من حيث سنة الدخول للجامعة.

2) اكتب وحدة بلغة الباسكال اسمها ReadPrimitives تقوم بالعمليات التالية:

1. توّفر إجراء لقراءة قيمة بولبانية (True, False).
2. توّفر دالة لقراءة قيمة حقيقية ضمن المجال [5, 5-].
3. توّفر إجراء لقراءة قيمة من نوع بدائي Enumerated.
4. دالة للتحقق من أن شريط مُدخَل لا يحوي أية حروف صوتية (I, A, e, u, o).
5. دالة للتحقق من أن عدد مُدخَل يقع ضمن المجال [5,5-]. ثم اكتب برنامج بلغة الباسكال يستخدم الوحدة السابقة ويطلب جميع الإجراءات السابقة.

3) ضع برنامجاً بلغة الباسكال يقوم بتعريف الوحدة **MyMatrix** التي تُمَكّن مما يلي:

1. إدخال مصفوفة مربعة **A** من الأعداد الصحيحة الموجبة من البعد $n \times n$.
2. إيجاد منقول المصفوفة **A** و طباعتها.
3. إيجاد أصغر عنصر للمصفوفة **A** مع تحديد مكان وقوعه.
4. حساب المضاعف المشترك الأصغر لعناصر المصفوفة **A**.
5. بيان فيما إذا كانت المصفوفة **A** متناظرة.
6. طباعة جميع الأعداد الأولية الموجودة في المصفوفة *A*.

4) اكتب وحدة اسمها Stat توَفر ما يلي:

1. إجراء لحساب المتوسط الحسابي والانحراف المعياري لعينة مدخلة.
2. إجراء لحساب الوسط، والربيعي الأول والثالث.
3. دالة لحساب المنوال.
4. دالة لاختبار فرضية معينة.
5. إجراء لحساب مجال ثقة.

5) اكتب برنامجاً بلغة الباسكال يقوم بإدخال عينة ويستخدم الوحدة السابقة للقيام بالحسابات التالية:

1. حساب معاملات النزعة المركزية للعينة.
2. حساب مجال ثقة حول متوسط العينة المدخلة.
3. اختبار فرضيات مختلفة حول متوسط العينة.

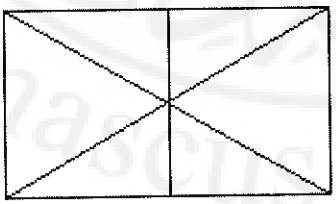
6) ادرس جيداً الوحدة التالية و التي تُعَرِف كائناً اسمه Emp. **حدد جميع** الأخطاء سواءً أكانت لغوية (compile errors) أم منطقية (run

time) و **أضف** أي تعليمات ناقصة (إن وجدت) بحيث تُصبح الوحدة قابلة للتنفيذ.

```
Unit A;
…
     Emp = Object;
       Name, LName: String;
       Function getName:String;
       Procedure init(AName: string);
…
…
Function getName:String;
Begin
      getName:= Name;
End;
Procedure init (AName: String)
  Begin
     Name:=AName;
  End;
…
```

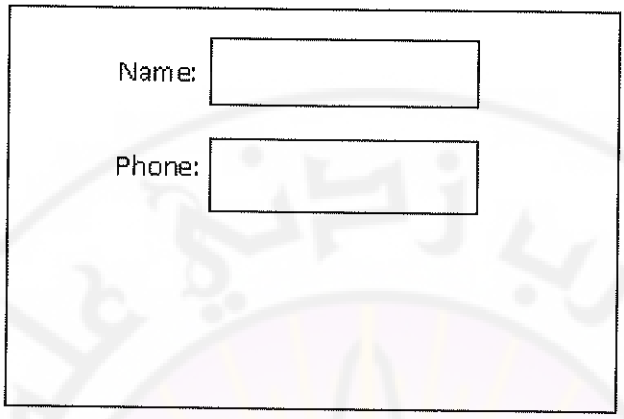
7) ضع برنامجاً بلغة الباسكال يقوم بإدخال علامات مجموعة من الطلاب في مادة البرمجة و الخوارزميات و تمثيل علامات الطلاب بطريقة الأعمدة، ثم رسم المضلع الناتج من رؤوس هذه الأعمدة .

8) اكتب برنامج بلغة الباسكال يقوم برسم الشكل التالي:



ثم يطبع رسالة نصية تحت هذا الشكل.

9) اكتب برنامج بلغة الباسكال يقوم برسم شاشة إدخال بيانات مماثلة للشكل التالي:



Damascus University

المصطلحات العلمية

Address	عنوان
Algorithm	خوارزمية
Amount	كمية
Append	إلحق، تذييل
Area of memory	مساحة الذاكرة
Argument	المعلمات
Arithmetic	حسابي
Array	مصفوفة
Assignment	إسناد، تعيين
Assign	ربط، تعيين
Average	متوسط
Bar	شريط
Basic	أساسي
Block	كتلة
Boolean	منطقي
Bounded	محدد
Brace	{ }
Brackets	[ ]

Browser	استعراض
Buffer	ذاكرة مؤقتة
Build	بناء
Call	استدعاء (دالة)
Comment	تعليق
Comparison	مقارنة
Compilation	ترجمة
Compiler	مترجم
Complexity	تعقيد
Component	مكوّن
Condition	شرط
Configuration	تكوين عام
Constant	ثابت
Container	حاوي
Coordinates	إحداثيات
Cursor	مؤشر
Data	بيانات
Debugging	التصحيح
Declaration	تصريح

Decrement	تناقص
Definition	تعريف
Denominator	مقام
Development Menu Bar	تطوير
Dialog box	مربع حوار
Digit	رقمي
Edit	تحرير
Efficiency	كفائة
Element	عنصر
Enumerated	تعداد
Environment	بيئة
Executable file	ملف تنفيذي
Execute	تنفيذ
Exponent	أس
Expression	تعبير
Extraction	استخراج
External	خارجي
Feasible solution	الحل الملائم
Field	حقل

File	ملف
Frequensy count	عدد التكرار
Function	دالة
Fundamentals	أساسيات
Graph	بيان
Generate	توليد
Global	عام (شامل)
Head	رأس
Heap	ذاكرة كومة
Help	مساعدة
Increment	تزايد
Index, subscript	فهرس
Internal	داخلي
Input	دخل
Insert	إدراج
Install	تنصيب
Integer	عدد صحيح
Interaction	تفاعل
Interpreter	مفسِّر

Item	حد
Iterator	مُكرِّر
Label	وسم
Least significant digit	الرقم الأقل أهمية
Level	مستوى
Linear	خطي
Link	ارتباط
Linked list	قائمة مرتبطة
Linker	رابط
List	قائمة
List box	مربع سرد
Local	محلي (موضعي)
Logical errors	الأخطاء المنطقية
Logical operator	معامل منطقي
Long	طويل
Loop	حلقة
Lower limit	الحد الأدنى
Mathematical	رياضي
Mathematics	رياضيات

Member	عضو
Menu	قائمة
Mode	نمط
Module	وحدة
Multimedia	وسائط متعددة
Natural logarithm	اللوغاريتم الطبيعي
Negation	عكس العلامة
Negative	سالب
Nested	متداخل
Node	عقدة
Not equal	لا يساوي
Null	خامل (صفري)
Object	كائن
Operating system	نظام التشغيل
Operator	معامل
Optimal solution	الحل الأمثل
Optimal storage	التخزين الأمثل
Options	خيارات
Order of function	ترتيب الدوال

Output	خرج
Overflow	فائض
Parameters	وسطاء
Path length	طول المسار
Permutation	تباديل
Pointer	مؤشر
Pointer array	مؤشر مصفوفة
Positive	موجب
Postfix	التالي
Power	طاقة، أس
Precedence	أولوية
Precision	دقة
Prefix	السابق
Print	طباعة
Private	خاص
Probability	إحتمالات
Procedure	إجرائية
Processor	معالج
Program	برنامج

Prompt	موجّه
Properties	خصائص
Protected	مَحمي
Public	عام
Push	دفع
Quadratic	تربيعي
Queue	طابور
Quick sort	الفرز السريع
Quotation	علامة اقتباس
Radix sort	الفرز الأساسي
Random access machine	آلة التنفيذ العشوائي
Rate of growth	معدل النمو
Read	أقرأ
Real	عدد حقيقي
Record	سجل
Recursive programming	البرمجة التعاودية
Redirection	تغيير الوجهة
Reference	مرجع
Register	مسجّل

Relational	علائقي
Remainder	الباقي
Reset	يهيئ، يجهز
Resource	مصّدر (مورد)
Relative order	الترتيب النسبي
Root	الجذر
Root directory	دليل جذري
Run	تشغيل
Run time errors	أخطاء التنفيذ
Scope	مدى (متغيّر)
Scorll	تدرج
Seek	يبحث، يفتش عن
Selection sort	الفرز المحدد
Settings	إعدادات
Set	مجموعة
Setup	تنصيب
Single	أحادي
Source	مصدر
Space	مسافة

Split point	نقطة الفصل
Square root	جذر تربيعي
Stability	الأستقرار
Standard	قياسي
State	حالة
Statement	عبارة
Stream	دفق
String	سلسلة محارف
Structure	بنية
Successor	الاحقة
Syntax errors	أخطاء لغوية
Tab	حرف جدولة
Tail	ذيل
Tapes	أشرطة
Template	قالب
Time of execution	زمن التنفيذ
Time sharing	وقت المشاركة
Turing machine	آلة تورينغ
Uniform distribution	توزيع المنتظم

Update	تحديث
Upper limit	النهاية العظمة
Variable	متغير
Vector	متّجه
Verification	التحقق
Version	إصدار
View	عرض
Window	نافذة


وقل رب زدني علما
جامعة دمشق
Damascus University

المراجع العلمية

1. آلان تيران، ترجمة: محمد الحجار، " لغة باسكال"، المؤسسة الجامعية للدراسات والنشر، 1998 .
2. د. حكمت عبد القادر "البرمجة بلغة تربو باسكال"، مدرية الكتب والمطبوعات- جامعة البعث، 2006-2007.
3. د. خالد خنيفس، د. محمد جاسم، وأخرون، "الخوارزميات والبرمجة (1)"، منشورات جامعة دمشق، 2008-2009.
4. د. فرحان اسماعيل، لغات البرمجة، منشورات جامعة دمشق، 2005-2006.
5. ورنر فيبل، م. ياسر نهاد العقاد، "تربو باسكال – الدليل الكامل7"، دار شعاع الطبعة الأولى 1997. (مترجم)
6. Baron, Naomi S. (1986) Computer Languages a Guide for the *Perplexed*. Anchor Press/Doubleday, New York.
7. Dershem, Herbet L. (1990) *Programming Languages Structures and Models*..Wadsworth,California.
8. Ledgard, Herny F.(1984) The American Pascal Standard. Springer-Verlag, New York.
9. Reges, Stuart. (1987) Building Pascal Programs: An Introduction to Computer Science. Little Brown, Boston.
10. Rich, Robert P. Internal Sorting Methods Illustrated with PL/I Programs. Englewood Cliffs: Prentice-Hall, 1972.
11. Sebesta, Robert W. (1996)Concepts of Programming Languages. Addison-Wesley Publishing Company, Menlo Park California.
12. Sebesta, Robert W. Concepts of Programming Languages. 3rd ed. Menlo Park: Addison-Wesley, 1996.

13. Seidman, Arthur H. (1984) The Handbook of Computers and Computing.Van Nostrand Reinhold company Inc, NewYork.
14. Welsh, Jim. (1979) Introduction to Pascal. Prentice/Hall International, New Jersy.

Damascus University

التدقيق اللغوي

د. غسان السيد

التدقيق العلمي

أ.د. غسان فلوح: كلية الهندسة الميكانيكية والكهربائية، جامعة دمشق.

د. غصون جيرودي: قسم الرياضيات، كلية العلوم، جامعة دمشق.

د. ريمة القمحة: قسم الرياضيات، كلية العلوم، جامعة دمشق.

حقوق الطبع والترجمة والنشر محفوظة لمديرية الكتب والمطبوعات في جامعة دمشق


وقل رب زدني علما
جامعة دمشق
Damascus University