



لغات البرمجة



منشورات جامعة دمشق
كلية العلوم

لغات البرمجية

الدكتور
رياض طيفور
أستاذ في قسم الجيولوجيا

1429-1430هـ

2009-2010 م

جامعة دمشق



فهرس محتويات الكتاب

11	 مقدمة
13	 الفصل الأول: مقدمة في الخوارزميات
15	 1-1 مقدمة
16	 2-1 قواعد اختيار الخوارزمية
17	 3-1 منهجية حل الخوارزمية
18	 4-1 طرائق كتابة الخوارزمية
20	 5-1 أمثلة على الخوارزميات
24	 6-1 تحويل الخوارزمية إلى برنامج
24	 7-1 لغات البرمجة
25	 1-7-1 لغات المستوى المنخفض
25	 2-7-1 لغات المستوى العالي
29	 الفصل الثاني: تعريف ببنية لغة التروباسكال
31	 1-2 مقدمة
31	 2-2 بنية لغة التروباسكال
36	 3-2 البرنامج في لغة التروباسكال
47	 الفصل الثالث: عناصر لغة التروباسكال
49	 1-3 الرموز الأساسية
49	 2-3 أنواع المعطيات
50	 1-2-3 النوع الصحيح Integer
51	 2-2-3 النوع الحقيقي Real

53	 Char النوع المحرفي 3-2-3
55	 String سلسله - النوع المحرفي 4-2-3
55	 Boolean النوع البوليني 5-2-3
56	 Type النوع التعدادي 6-2-3
57	 3-3 العمليات الحسابية
59	 1-3-3 التعبيرات الحسابية
60	 2-3-3 أولوية تنفيذ العمليات الحسابية
60	 4-3 عمليات المقارنة
62	 5-3 العمليات المنطقية
65	 6-3 التعبيرات المنطقية
67	 الفصل الرابع: تعليمات لغة التريباسكال
69	 1-4 مقدمة
69	 2-4 تعليمات إدخال المعطيات
73	 3-4 تعليمات إخراج المعطيات
76	 1-3-4 طرائق إخراج النتائج
76	 1-1-3-4 طريقة إخراج المعطيات من النوع المحرفي
77	 2-1-3-4 طريقة إخراج المعطيات من النوع الصحيح
79	 3-1-3-4 طريقة إخراج المعطيات من النوع الحقيقي
83	 4-4 تعليمات الحلقات
83	 1-4-4 For تعليمة الحلقة
85	 2-4-4 For تعليمة الحلقة العكسية
87	 3-4-4 While ..Do تعليمة الحلقة

89	 Repeat ..Until	4-4-4	تعليلة الللقة
91		5-4-4	تداخل الللقات
94		6-4-4	الخرج من الللقة
95		5-4	التعلللات الشرطلة
95	 If .. Then	1-5-4	تعليلة
95	 If .. Then ..Else	2-5-4	تعليلة
100	 If .. In	3-5-4	تعليلة
102	 Case ..Of .. End	4-5-4	تعليلة الاختلار
106		6-4	تعلللات اللنتقال (القفز)
107	 Goto	1-6-4	تعلللات اللنتقال للر المشروط
107	 If .. Then	2-6-4	تعلللات اللنتقال المشروط
108		7-4	المعطللات المركبة
109	 Array	1-7-4	المنظومة (المصفوفات)
114		2-7-4	نوع المعطللات التعدادل (المجالل)
116	Type	3-7-4	بعض الإلراءات اللل تتعامل مع النمط
117	 (Record)	4-7-4	نوع المعطللات سجل
125			الفصل الللامس: الإلراءات والتواع
127		1-5	مقدمة
129	 Procedure	2-5	الإلراء
129		1-2-5	تعرف الإلراء واستدعائه
131		2-2-5	المتحوللات العامة والمتحوللات الموضعلة
135		3-2-5	استلخدام الوسطاء

140	 4-2-5 الاستدعاء الذاتي للبرنامج الجزئي
141	 Function التابع 3-5
143	 4-5 الإجراءات والدوال المسبقة التعريف
157	 الفصل السادس: طرائق العرض على الشاشة
159	 1-6 مقدمة
159	 2-6 العرض النصي
159	 1-2-6 المحول الرسومي وذاكرة العرض
161	 2-2-6 أنماط العرض على الشاشة
163	 3-2-6 التحكم بالألوان
164	 4-2-6 استخدام إحداثيات الشاشة
168	 5-2-6 إجراءات أخرى لأنماط العرض على الشاشة
175	 3-6 العرض البياني (الرسومي)
175	 1-3-6 الوحدة Graph
175	 2-3-6 سواقات الرسومات
177	 3-3-6 تعليمات الكتابة على شاشة الرسم
187	 الفصل السابع: الملفات والوحدات
189	 1-7 الملفات
189	 1-1-7 أنواع الملفات
190	 2-1-7 العمليات على الملفات
195	 3-1-7 توابع وعمليات أخرى تُستخدم على الملفات
198	 2-7 الوحدات
199	 1-2-7 بنية الوحدات

200	2-2-7 إنشاء الوحدات
229	الملحق (أ): بعض الإجراءات والتتابع المستخدمة في لغة التربو باسكال
239	الملحق (ب): رسائل الخطأ
265	الملحق (ج): بعض أوامر نافذة التحرير
269	المصطلحات العلمية
277	المراجع



مقدمة:

وضع هذا الكتاب ليكون منسجماً أساساً مع منهاج مقرر لغات البرمجة لطلاب السنة الرابعة في العلوم الجيولوجية - شعبة الجيوفيزياء في كلية العلوم. إضافةً إلى ذلك تم إعداد هذا الكتاب بحيث يصلح أن يكون كتاباً مرجعياً لطلاب كلية العلوم أو الكليات الأخرى، وذلك للطريقة السهلة التي اعتمدها المؤلف في تقديم المعلومات والتي تضمنت العديد من المبادئ البرمجية التي يمكن تخدم جميع المهتمين بلغات البرمجة على مختلف مستوياتهم، وكذلك لما تناوله هذا الكتاب من تطبيقات مختلفة يمكن أن تقع في مجالات إهتمام العديد من الطلاب في اختصاصات مختلفة.

لقد أعد هذا الكتاب بحيث يعالج لغة البرمجة "التربوباسكال"، وذلك لما لهذه اللغة من أهمية بالغة فهذه اللغة تختلف عن باقي لغات البرمجة الأخرى كونها لغة تعليمية وليست لغة برمجية فقط. لقد ناقش هذا الكتاب الإصدار السابع من هذه اللغة وتعرض لهذه اللغة من مراحلها الأولى بدءاً بالمرحلة البسيطة لاستخدام هذه اللغة، بحيث يساعد القارئ المبتدئ على الاستفادة من المعلومات في هذا الكتاب بدون صعوبات ويرسم له الطريق السهلة لكتابة البرنامج، كما أن الكتاب تعرض إلى العديد من التفاصيل والمواضيع المتقدمة، والتي لم تتعرض لها كتب برمجية أخرى، والتي تهم الفئات ذات السوية المتقدمة في لغة التربوباسكال.

لقد جاءت موضوعات هذا الكتاب موزعة على سبعة فصول:

الفصل الأول: يتعرض للخوارزمية من حيث قواعد اختيارها وطرائق كتابتها وتحويلها إلى برنامج.

الفصل الثاني: قدم تعريفاً بيئته وبرنامج لغة التربوباسكال.

الفصل الثالث: درس عناصر لغة التربوباسكال، مثل الرموز وأنواع المعطيات والعمليات بأنواعها (الحسائية، المقارنة، المنطقية).

الفصل الرابع: عرض تعليمات لغة التربوباسكال (تعليمات الإدخال والإخراج، التعليمات الشرطية، تعليمات الانتقال، المعطيات المركبة).

الفصل الخامس: بين كيفية كتابة الإجراءات والتوابع وتعرض أيضاً إلى الإجراءات والدوال مسبقة التعريف.

الفصل السادس: ناقش طرائق العرض على الشاشة، وتحدث عن العرض النصي والعرض الرسومي (البياني).

الفصل السابع: تحدث عن الملفات والوحدات وكيفية إنشائها واستخدامها في البرنامج، وناقش كيفية التعامل مع الملفات.

لقد حاولت من خلال وضعي لهذا الكتاب بذل جهداً كافياً في شرح هذه المادة وتقديمها بشكل مبسط وسهل للقارئ من خلال تقديم العديد من الأمثلة المتنوعة، متوخياً الدقة العلمية في ذلك، ولكن أعتقد بأن هذا الجهد لا يكفي لاستيعاب وفهم هذه المادة العلمية وإنما يجب على القارئ أن يبذل جهداً ملحوظاً من خلال المتابعة والاهتمام في الجوانب العملية لهذه المادة من خلال الأمثلة المعروضة في الكتاب والأمثلة الأخرى التي ستقدم في الجلسات العملية للمقرر.

وأرجو من الله تعالى أن يكون عوناً لنا وأن يوفقنا فيما نصبو إليه لخدمة أمتنا، وهو ولي التوفيق.

رياض طيفور

الفصل الأول

مقدمة في الخوارزميات

1-1 مقدمة

2-1 قواعد اختيار الخوارزمية

3-1 منهجية حل المسألة

4-1 طرائق كتابة الخوارزمية

5-1 أمثلة على الخوارزميات

6-1 تحويل الخوارزمية إلى برنامج

7-1 لغات البرمجة



مقدمة في الخوارزميات

1-1 مقدمة:

الخوارزمية مفهوم قديم يعود إلى مطلع القرن التاسع الميلادي ويرتبط مفهوم الخوارزمية باسم العالم العربي مُحَمَّد بن موسى الخوارزمي الذي عاش في تلك الفترة وعرف عنه شهرته في علوم الرياضيات والفلك، وترجم كتابه فن الحساب إلى اللاتينية بعنوان *Algorithmi de nemero Indrium*.

لقد تطور مفهوم الخوارزمية في السنوات الأخيرة تطوراً كبيراً وذلك نظراً لارتباطه ببرمجة الحواسيب الإلكترونية،

فالخوارزمية عبارة عن مجموعة من التعليمات تصف عملية المعالجة لمسألة ما على شكل خطوات متسلسلة محددة قابلة للتنفيذ من قبل الإنسان أو الآلة خطوة- خطوة وتؤدي إلى نتيجة صحيحة.

ففي مجال الحاسب، الخوارزميات تعبر عن طريقة معالجة يمكن تنفيذها بواسطة الحاسبات عوضاً على الإنسان. والخوارزمية هي مصطلح يمكن أن يطلق على الخطوات المكتوبة باللغة العربية أو الإنجليزية أو باستخدام رسوم توضيحية تمثل الخطوات وتسلسلها أو يمكن أيضاً أن تطلق على البرامج التي تكون مكتوبة بلغة من لغات الحاسوب.

ويمكن أن تتعامل الخوارزمية مع معطيات غير عددية؛ مثل النصوص والرموز والصور، وعندها تسمى بالخوارزمية غير الحسابية.

ويمكن أن تتعامل مع الأعداد والأرقام وتسمى بالخوارزمية الحسابية. إلا أن الخوارزميات لا تقتصر فقط على ذلك وإنما نصادفها في العديد من الحالات في حياتنا العامة، فعلى سبيل المثال: طهي الطعام يتطلب معرفة الخوارزمية التي على أساسها تم

الوصول إلى نتيجة معينة من المذاق والشكل لطبق معين. مكونات الطبق هي بمنزلة المعطيات وطريقة تحضيرها هي الخوارزمية.

أن إيجاد خوارزمية لمسألة معينة أمر يسير للغاية، ولكن إيجاد خوارزمية فعالة وسريعة ليس من السهل في كل الحالات. بفرض أنك تريد السفر من دمشق إلى الرياض، يمكنك فعل ذلك بإحدى هاتين الطريقتين:

دمشق- بيروت - القاهرة - الرياض أو دمشق-الرياض.

بالتأكيد في كلا الحالتين يتم تحقيق الهدف وهو الوصول إلى الرياض، ولكن في الطريقة الثانية يتم توفير الكثير من الوقت والمال، هذا مثال بسيط وهو يماثل الكثير من الأعمال التي نقوم بإنجازها في حياتنا اليومية بدون أن نشعر بأننا نقوم بتنفيذ خوارزمية ما، إلا أن الأمر يصبح أكثر تعقيداً عندما تكون العمل المراد إنجازه أكبر، فلو طورنا مثالنا السابق وأردنا مثلاً زيارة 10 عواصم، ونريد أن نبحت على أقصر طريق، فالمسألة في هذه الحالة تصبح أعقد ولا يمكن هنا تحديد أقصر طريق بمجرد النظر إلى الخارطة، وهنا لا بد من التمعن والبحث عن الحل الأمثل من بين حلول عديدة يمكن أن تؤدي إلى الهدف المطلوب.

نلاحظ إذاً أن المسألة الواحدة يمكن أن تحل بطرائق عديدة، ولكن متى يمكن أن نتحدث عن الخوارزمية المثلى، أو الخوارزمية الأفضل، وما القواعد التي تحكم كفاءة خوارزمية معينة لاداء المهمة؟

1-2 قواعد اختيار الخوارزمية:

هناك عدة قواعد نستطيع بموجبها اختيار الخوارزمية المناسبة لاداء العمل المطلوب

وهي على النحو التالي:

1- صحة النتيجة:

لا بد أن يكون الناتج هو الهدف الذي نصبو إليه, بمعنى أنه لا تعتبر الخوارزمية صالحة لاداء العمل طالما النتيجة غير صحيحة.

2- زمن انجاز الخوارزمية:

يجب أن يكون زمن تنفيذ الخوارزمية محدود, أي لا بد هنا من الوصول الى حل صحيح للمشكلة بعد عدد معين ومحدود من الخطوات.

بعض الخوارزميات تتمكن من الوصول الى الحل في زمن قصير ويمكن للبعض الآخر أن يأخذ زمنا اطول. والخوارزمية التي لا تنتهي الى حل فلا تعتبر خوارزمية.

3- الذاكرة المستخدمة:

تؤدي مساحة التخزين للخوارزمية والمعطيات دوراً هاماً في تقييم الخوارزمية, وهنا يكون لطريقة حفظ المعطيات الدور الأكبر في اختزال مساحة التخزين.

4- السهولة:

سهولة الخوارزمية هي من الخصائص الهامة للخوارزمية المثالية, ولكن يجب هنا تحقيق التوازن بين سهولة الخوارزمية وفعاليتها, والأمر هنا يتعلق بالمسألة المطروحة وبنوع المعطيات.

1-3 منهجية حل المسألة:

من أجل حل مسألة ما باستخدام الحاسوب, لا بد من المرور بعدة خطوات أو مراحل ويكون لكل خطوة من هذه الخطوات مدخلات ومخرجات, ومنهجية الحل تختلف من مسألة لأخرى, إلا أنه بشكل عام يمكن أن نعد المراحل التالية:

1. تعريف المشكلة.

2. تحليل المشكلة.

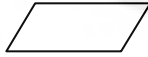
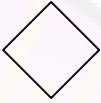
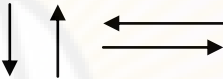
3. تصميم الخوارزمية.

4. كتابة البرنامج.
5. ترجمة البرنامج بواسطة مترجم.
6. تنفيذ البرنامج.

1-4 طرائق كتابة الخوارزمية :

هنالك طرائق عديدة من أجل كتابة الخوارزمية :

1. استخدام طريقة اللغة الطبيعية: وهنا تُستخدم اللغة الطبيعية لصياغة الخوارزمية.
 2. استخدام اللغة الرمزية: وهنا تُستخدم رموز خاصة لصياغة الخوارزمية.
 3. استخدام الطريقة البيانية: وهنا تُستخدم أشكال هندسية محددة لصياغة الخوارزمية.
- لكل طريقة من هذه الطرائق إيجابياتها وسلبياتها، إلا أننا سوف نناقش الطريقة الأكثر استخداماً في صياغة الخوارزمية فقط، وهي الطريقة البيانية؛ ولكن سنلاحظ أنه غالباً ما تُستخدم أكثر من طريقة بالوقت نفسه لصياغة الخوارزمية.
- الطريقة البيانية يمكن أن تكون بشكلين : المخطط الصندوقي - المخطط التدفقي.
- وسنتحدث فقط عن المخطط التدفقي وهو الأكثر استخداماً؛ لأن استخدامه يكون أسهل من المخطط الصندوقي من ناحية ويعطي الخوارزمية وضوحاً أكثر من ناحية أخرى .
- من أجل صياغة الخوارزمية بالطريقة البيانية؛ وباستخدام المخطط التدفقي تُستخدم أشكال هندسية مختلفة لتمثيل هذا المخطط. الجدول (1-1) يوضح بعض هذه الأشكال:

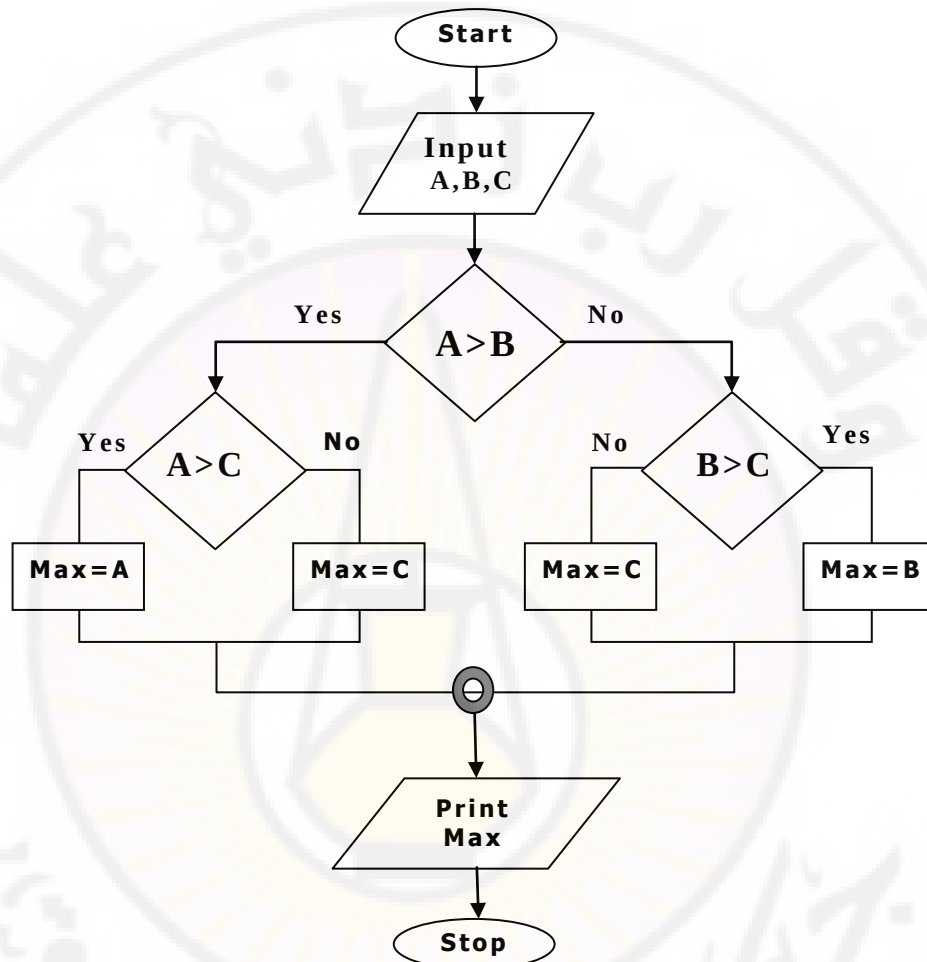
الرقم	الشكل الهندسي	الاستخدام
1		يُستخدم للتعبير عن بداية أو نهاية الخوارزمية
2		يُستخدم للتعبير عن عملية إدخال أو إخراج
3		يُستخدم للتعبير عن عملية حسابية
4		يُستخدم للتعبير عن شرط أو عملية تفرع في البرنامج.
5		يُستخدم للتعبير عن استدعاء برنامج جزئي.
6		تُستخدم للوصل بين رموز العمليات المختلفة.
7		تُستخدم كنقاط اتصال بين المراحل المختلفة.

الجدول (1-1): الأشكال الهندسية المستخدمة في صياغة الخوارزمية بحالة المخطط التدفقي.

5-1 أمثلة على الخوارزميات:

مثال (1):

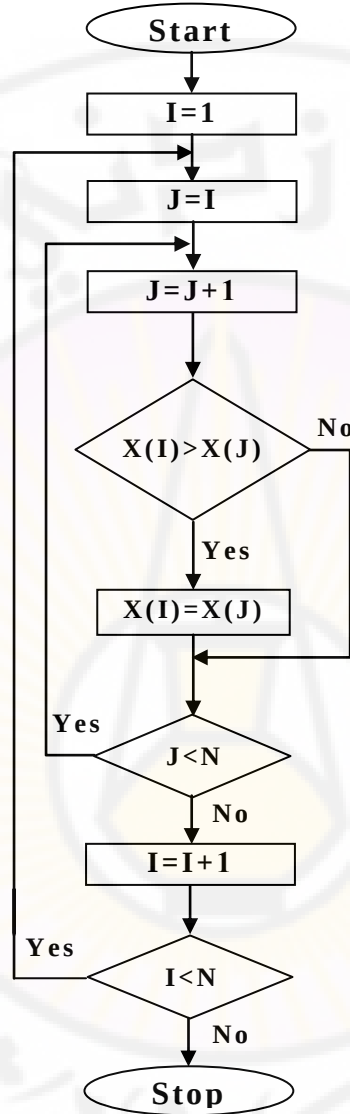
الخوارزمية التالية تستخدم لترتيب ثلاثة أعداد A, B, C :



الشكل (1-1): مخطط تدفقي لترتيب ثلاثة أعداد تصاعدياً

مثال (2):

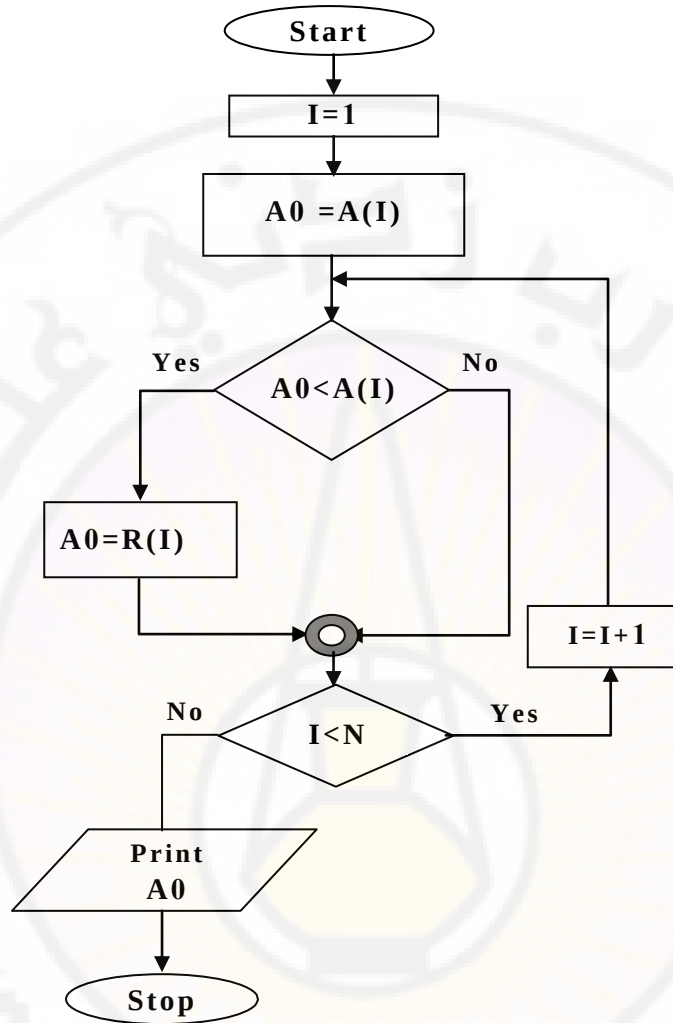
الخوارزمية التالية تستخدم لترتيب سلسلة من الأعداد (N عدد)، وتم تمثيل السلسلة هنا بالمتحول $X(I)$ ، حيث I تتحول من 1 وحتى N :



الشكل (2-1): مخطط تدفقي لترتيب مجموعة من الأعداد تصاعدياً (N : عدد)

مثال (3):

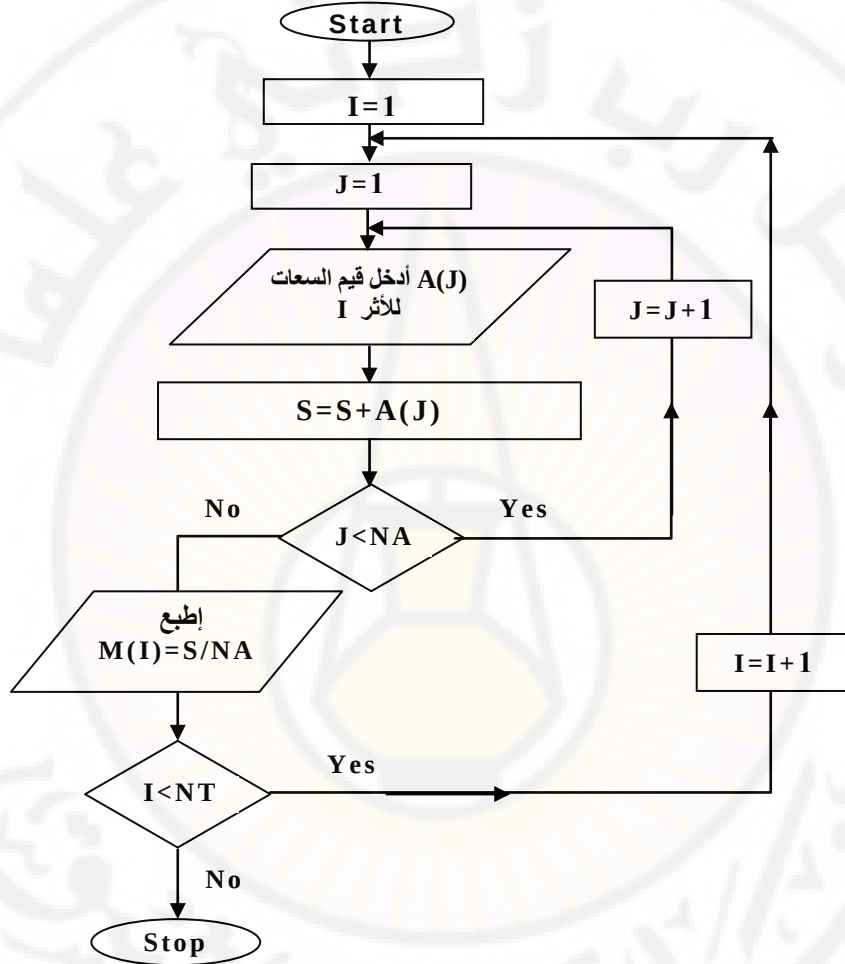
الخوارزمية التالية تحسب القيمة العظمى للسعة في أثر سيزمي مكون من N عينة.



الشكل (3-1): مخطط تدفقي لحساب القيمة العظمى لعينات أثر سيزمي

مثال (4):

يتم في هذه الخوارزمية حساب قيمة المتوسط الحسابي للسعة (A) لكل أثر في مقطع سيزمي مكون من عدد من الآثار السيزمية (NT). علماً بأن عدد العينات في الأثر السيزمي هو (NA).



الشكل (4-1): مخطط تدفقي لحساب القيمة العظمى لعينات أثر سيزمي

1-6 تحويل الخوارزمية إلى برنامج:

لاحظنا أن وضع الخوارزمية هي مرحلة تسبق مرحلة كتابة البرنامج، ولا بد بعد الانتهاء من وضع الخوارزمية من الانتقال إلى المرحلة التالية وهي تحويل الخوارزمية إلى برنامج، وهنا يتعلق الأمر بنوعية الخوارزمية، ففي حالة الخوارزمية السهلة فإن التحويل لا يتطلب من المبرمج سوى كتابة الشفرة المطلوبة، بأي لغة كانت، أما إذا كانت الخوارزمية معقدة، فإن ذلك يتطلب من المبرمج إتخاذ قرارات معينة فيما يتعلق مثلاً بطريقة حفظ المعطيات أو طريقته اختيار نوع المتغيرات، بحيث تناسب و اللغة التي يريد أن يستخدمها. من هنا نلاحظ أن الخوارزمية لا علاقة لها بلغات البرمجة، وإنما لغة البرمجة هي أداة لتطبيق الخوارزمية.

1-7 لغات البرمجة:

ذكرنا سابقاً أن الخوارزمية هي مرحلة تسبق عملية كتابة البرنامج في إحدى لغات البرمجة، فالحاسوب لا يمكن أن ينفذ الخوارزمية بالشكل الذي تمت صياغته سابقاً، وإنما يجب أن تصاغ هذه الخوارزمية المكونة من سلسلة من التعليمات بإحدى لغات البرمجة. فالبرنامج (Program) إذاً هو عبارة عن خوارزمية تمت كتابتها بإحدى لغات البرمجة، فلغات البرمجة - وبالتالي البرنامج - هي الطريقة الوحيدة التي يمكن من خلالها نقل رغباتنا إلى الحاسوب.

إن لغات البرمجة المستخدمة عديدة؛ ويمكن تصنيفها إلى:

- لغات المستوى المنخفض
- لغات المستوى العالي

1-7-1 لغات المستوى المنخفض Low-Level-Language:

وهنا نميز لغتين :

لغة الآلة Machine Language:

هذه اللغة استخدمت في بداية تطوير الحواسيب وهي تستخدم الرمز 1,0 فقط؛ للتعبير عن تعليمات، وأوامر البرنامج هي ما يجعل كتابة البرنامج في هذه اللغة صعبة؛ كما أن إيجاد الأخطاء وإصلاحها صعب أيضاً، هذا إضافة إلى اختلاف هذه اللغة من حاسوب لآخر؛ الشيء الذي يتطلب من المبرمج معرفة مكونات وإمكانات الحاسوب الذي سيقوم بتنفيذ البرنامج، إلا أن الإيجابية الوحيدة في هذه اللغة أن البرنامج يتم تنفيذه بسرعة؛ بحيث لا يحتاج لوسيط (مترجم) من أجل نقله إلى لغة الحاسوب.

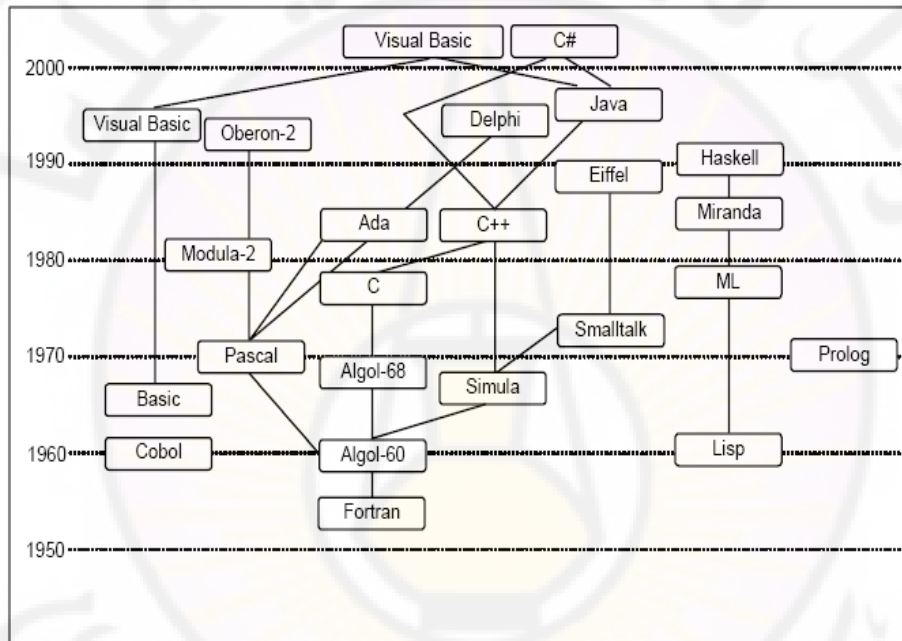
لغة المجمع Assembly Language:

في هذه اللغة يتم استخدام رموز من ثلاثة أو أربعة أحرف للتعبير عن التعليمات وتكتب المعطيات بالنظام الست عشري، وبالتالي تُعد هذه اللغة منخفضة المستوى لاسيما وأنها ترتبط بنوع الحاسوب أيضاً وهي لا تنفذ مباشرة بالحاسوب كما هو الحال في لغة الآلة؛ وإنما تحتاج إلى مترجم يسمى Assembler.

1-7-2 لغات المستوى العالي High-Level-Language:

اهتم المختصون - بسبب صعوبة كتابة البرنامج باللغات منخفضة المستوى وارتباط البرنامج بنوع الحاسوب- بإيجاد لغة جديدة أسهل لكتابة البرنامج ولا ترتبط بنوع الحاسوب، وسميت باللغات عالية المستوى؛ وهذه اللغات تستخدم كلمات قريبة من اللغات الطبيعية (اللغة الإنكليزية) للتعبير عن تعليماتها؛ مما يسهل على المبرمج عملية التعامل معها. أول لغة ظهرت من هذا النوع كانت لغة FORTRAN ثم تعددت

اللغات فظهرت PASCAL ,BASIC COBOL, VISUAL BASIC, DELPHI ,C++, JAVA وغيرها من اللغات الأخرى. الشكل (1-5) يبين مخططاً لمراحل تطور لغات البرمجة العالية المستوى. وبما أن الحاسوب لا يتعامل إلا مع لغة واحدة رموزها (1,0) فإنه يجب -من أجل تنفيذ برامج اللغات العليا في الحاسوب- تحويلها إلى لغة الحاسوب، وبالتالي نحتاج هنا إلى مترجمات للقيام بهذه العملية؛ وهذه المترجمات تكون على نوعين، مترجم فوري Interpreter و مترجم غير فوري Compiler.



الشكل (1-5): مخطط مراحل تطور لغات البرمجة العالية المستوى.

وهنا يجب أن نؤكد أن الخطوة الأساسية لحل مسألة بالحاسوب هي كتابة الخوارزمية بإحدى لغات البرمجة العليا، واختيار لغة البرمجة بحيث تتناسب مع المسألة المطروحة. هذه الخطوة تتطلب من المبرمج معرفة ودراية كافيتين من أجل مقابلة تعليمات الخوارزمية؛ مع تعليمات اللغة التي يريد صياغة الخوارزمية فيها.





الفصل الثاني
تعريف بيئة لغة التربوباسكال

1-2 مقدمة

2-2 بيئة لغة التربوباسكال

3-2 البرنامج في لغة التربوباسكال



تعريف بيئة لغة التربوباسكال

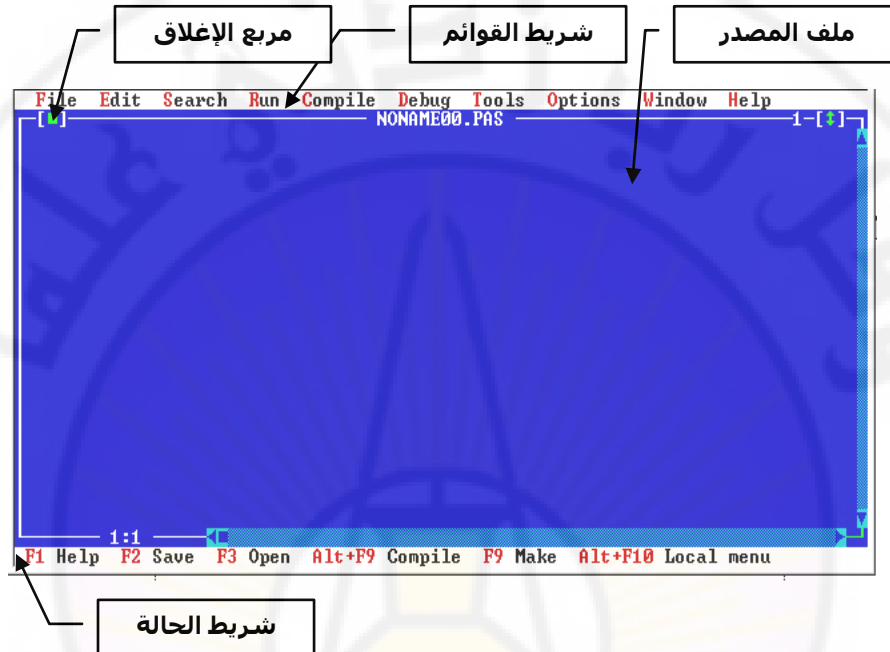
2-1 مقدمة:

لقد طورت لغة باسكال Pascal بين عامي 1968 و 1970 من قبل عالم الكمبيوتر السويسري نيكولوس ويرث (Niklaus Wirth) وسميت باسكال نسبة إلى عالم الرياضيات الفرنسي (Blaise Pascal (1623-1662). لقد طور هذا العالم لغة الباسكال وذلك بعد مشاركته في تطوير لغة Algol ، والتي كانت تتصف بالصعوبة والتعقيد، مما جعل هذا العالم يفكر بتطوير لغة أخرى أقل تعقيداً وسهلة التعلم ومتكاملة، وكانت بالتالي لغة الباسكال التي انضوت تحت فئة اللغات العالية المستوى. لقد دخلت هذه اللغة في دورة تطوير مستمرة، فقد ظهرت للوجود عدة إصدارات وجميعها تضمنت تطويرات جديدة على الإصدارات التي سبقتها، إلى أن قامت شركة بورلاند Borland بتقديم نسختين من لغة الباسكال هما: التربوباسكال ، وبورلاندباسكال، وتتميز لغة التربوباسكال بالعمل في بيئة Dos كما أنها تتمتع ببيئة برمجية قوية، تتضمن محرر نصوص ومترجم ومصحح أخطاء ونوافذ وشاشات مساعدة. هذا إضافة إلى ميزات البرمجة الهدفية التوجه التي تضمنتها لغة التربوباسكال اعتباراً من الإصدار (5.5) ، كما تتمتع الإصدار السابع من التربوباسكال بميزات عديدة فإضافة إلى إمكانية إنشاء برامج باستخدام البرمجة الهدفية التوجه ، يمكن أيضاً هنا استخدام تربوفيجين لإنشاء وإدارة النوافذ والقوائم وصناديق الحوار.

2-2 بيئة لغة التربوباسكال:

تؤمن لغة التربوباسكال بيئة سهلة لكتابة البرنامج، فعند إغلاق البرنامج نحصل على نافذة خاصة يمكن من خلالها تنفيذ عمليات عديدة، الترجمة (Compile) والتحرير

(Edit) والتنفيذ (Run) والتنقيح (Debug). وكل عملية من هذه العمليات تتضمن مجموعة كبيرة من الأوامر. الشكل (1-2) يوضح النافذة الرئيسية في لغة التريبواسكال وهي تتألف من عدة أجزاء رئيسية:



الشكل (1-2): النافذة الرئيسية في بيئة التريبواسكال

1- شريط القوائم: وهو يحتوي على قوائم مختلفة (عشرة قوائم) تتضمن كل قائمة مجموعة من الأوامر، ويمكن اختيار هذه القوائم بالنقر بالماوس أو باستخدام المفتاح Alt مع أول حرف من أسم القائمة،

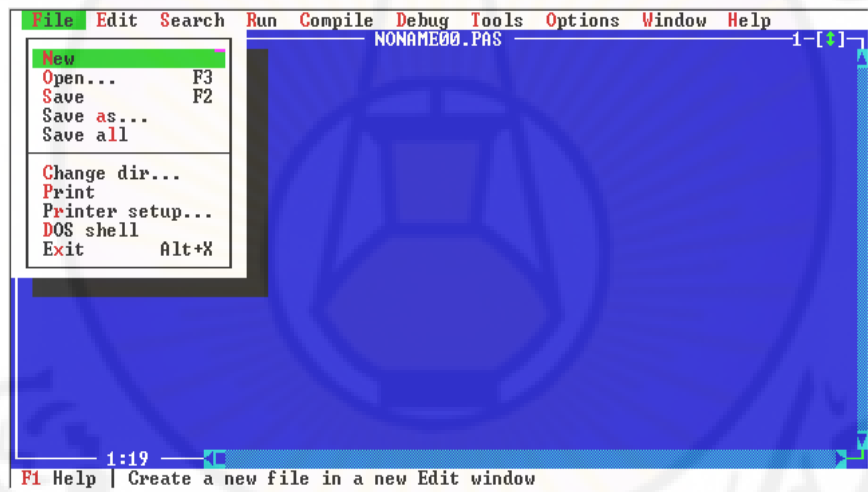
مثلاً من أجل فتح القائمة File نضغط على المفتاح Alt ثم المفتاح F.

2- شريط الحالة: يتم فيه عرض رسائل أثناء معالجة البرنامج، كمثال تعريف بالأوامر المختارة، أو يتم عرض أسماء مفاتيح يمكن أن تكون مفيدة أثناء المعالجة.

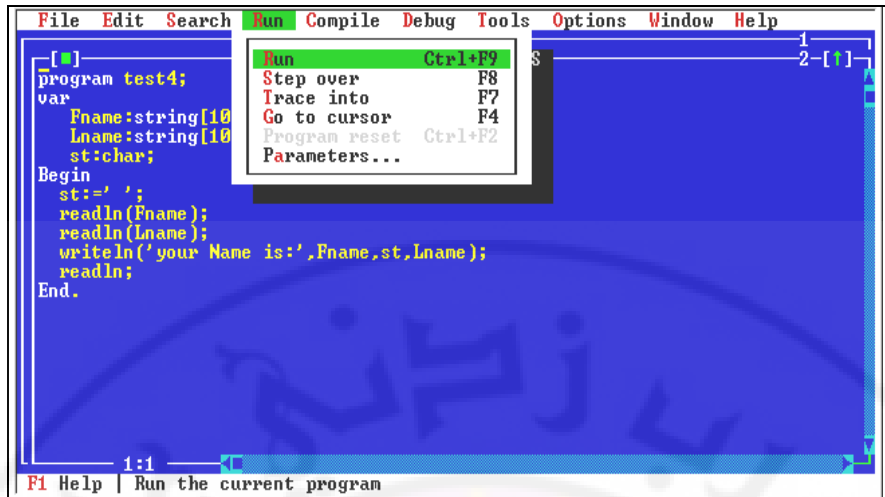
3- مربع الإغلاق: يُستخدم لإغلاق النوافذ بواسطة النقر بالماوس.

4- ملف المصدر: وهو يمثل الجزء الذي يتم فيه كتابة نص البرنامج.

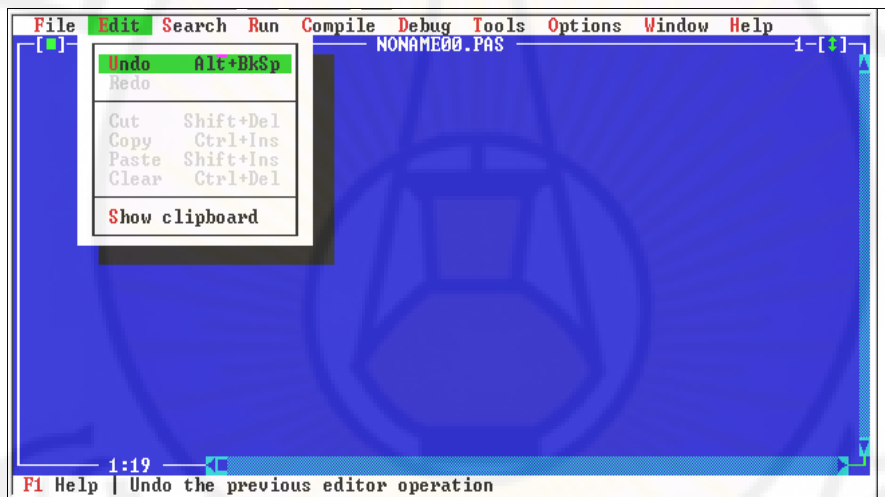
الأشكال التالية تبين النافذة الرئيسية في التريوباسكال مع عرض أوامر بعض القوائم الرئيسية مثل File، Edit، Run :



الشكل (2-2): النافذة الرئيسية في بيئة التريوباسكال (شريط القوائم - القائمة File)



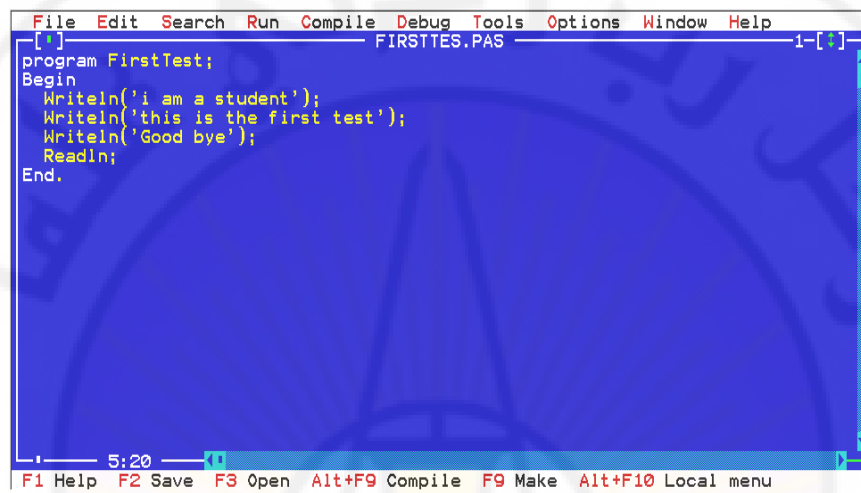
الشكل (2-3): النافذة الرئيسية في بيئة التزويباسكال (شريط القوائم - القائمة Edit)



الشكل (2-4): النافذة الرئيسية في بيئة التزويباسكال (شريط القوائم - القائمة Run)

نريد في البداية بأن نخطو خطوة صغيرة إلى الأمام ونكتب برنامجاً صغيراً بهدف التعرف فقط إلى كيفية كتابة البرنامج في لغة التروبايسكال.

نقوم أولاً بتشغيل برنامج التروبايسكال من المكان المحدد له على وحدات التخزين الثانوي ، فنحصل على النافذة الرئيسة في بيئة التروبايسكال ، فنقوم بكتابة نص البرنامج باستخدام محرر النصوص الخاص بالتروبايسكال فتبدو النافذة كالاتي:



```
File Edit Search Run Compile Debug Tools Options Window Help
FIRSTES.PAS 1-[ ]
program FirstTest;
begin
  writeln('i am a student');
  writeln('this is the first test');
  writeln('Good bye');
  readln;
end.
```

5:20

F1 Help F2 Save F3 Open Alt+F9 Compile F9 Make Alt+F10 Local menu

الشكل (2-5): النافذة الرئيسة (محرر النصوص)

نقوم بتخزين البرنامج باستخدام الأمر **Save as** من قائمة **File**، وهنا يتم تخزين البرنامج حسب المسار المحدد من قبل المستخدم ويتم هنا تحديد لاحقة افتراضية لبرنامج التروبايسكال وهي **.pas**، وبعد ذلك نبدأ بتنفيذ البرنامج من اختيار الأمر **Run** من قائمة **Run**.

نتائج تنفيذ هذا البرنامج يكون على الشكل التالي:

I am a student
This is the first test
Good bye

وهنا لا بد من الإشارة إلى أن كل التفاصيل المتعلقة بالقوائم والأوامر المختلفة سيتم التعرف إليه بالتفصيل في جلسات العملي.

2-3 البرنامج في لغة التريبواسكال:

يمكن تقسيم البرنامج في لغة الباسكال إلى عدة أجزاء رئيسية :

1. **العنوان :** وهو يبدأ بكتابة كلمة `program` هذه الكلمة تُتبع باسم البرنامج وكذلك في بعض أنواع التريبواسكال تتبع بقوسين يتم فيها كتابة معاملات البرنامج. وهذه المعاملات تكون عبارة عن مصطلحات (متحولات مساعدة) من أجل الملفات المستخدمة في البرنامج.

مثال (1) :

`program geophysic (input , output);`

هذا يعني أنه يتم في البرنامج عملية إدخال وإخراج للمعطيات للمعطيات حيث يدل المعامل `input` على عمليات الإدخال والمعامل `output` على عمليات الإخراج.

في الإصدارات الحديثة للتريوباسكال يكفي فقط كتابة اسم البرنامج بعد كلمة program ولا داعي لذكر المعاملات السابقة، إضافةً إلى هذه المعلومات أيضاً يمكن أن يكتب في قسم العنوان ملاحظات (تعليق)، هذه الملاحظات يتم ملاحظتها من البرنامج ولكن لا يتم تنفيذها على أنها تعليمة وهذا التعليق مفيد للمبرمج في تسهيل قراءة البرنامج وفهمه.

أيضاً يتم في قسم العنوان استدعاء الوحدات الجاهزة في لغة لريبوباسكال مثل وحدة CRT ، DOS ، GRAPH ، وتستخدم هنا عبارة Uses لاستدعاء هذه الوحدات وذلك على الشكل التالي:

Uses Dos, Graph

إضافةً إلى ذلك يمكن للمبرمج أيضاً بناء مجموعة من الروتينات التي يمكن أن تشكل وحدة Unit يمكن استدعائها بنفس الطريقة السابقة.

ويجب الانتباه هنا إلى أن كل عبارة في برنامج لغة الباسكال تنتهي بالفاصلة المنقوطة (؛) ما عدا بعض الحالات سنتعرض لها فيما بعد .

ويجب أن ننتبه عند اختيار اسم البرنامج إلى بعض الأمور الأساسية ، حيث أنه هناك عدد كبير من التعليمات (الكلمات) في لغة التريبوباسكال والتي يختلف عددها من مصنف لآخر ومن صيغة لأخرى لا يجوز استخدامها كأسم للبرنامج ، ومن هذه الكلمات:

absolute
and

function
goto

record
repeat

array	if	set
begin	in	shl
case	inline	shr
const	label	string
div	mod	then
do	nil	to
down	not	type
else	overlay	until
end	of	var
external	or	while
file	packed	with
for	procedure	xor
forward	program	uses
unit	interface	object
implementation	inherited	library
constructor	destructor	exports
asm		

إضافةً إلى ذلك فإن اختيار اسم البرنامج يخضع لقواعد عديدة أيضاً وهي :

1- يجب أن يتكون اسم البرنامج من رموز (حروف وأرقام) ويجب أن يبدأ الاسم أولاً بحرف.

2- لا يجوز أن يرد ضمن الاسم أي رمز آخر بما فيها رمز الفراغ.

3- هناك الكثير من المترجمات لا تقبل أكثر من ثمانية رموز كاسم للبرنامج لذلك يفضل عدم الإطالة في اسم البرنامج بأكثر من ثمانية رموز ، علماً بأن الإصدارات الحديثة بعد turbo5 تقبل أن يتضمن اسم البرنامج حتى 127 محرف.

أمثلة صحيحة لأسم البرنامج:

praxi5
seismic1

2. جسم البرنامج :

وهو يتألف من قسمين :

- 1- **قسم التعريف:** وهو يأتي بعد قسم العنوان وفي هذا القسم يتم وصف أو كتابة المعطيات المستخدمة في البرنامج ونقصد بهذه المعطيات ما يلي:
- العلامة (Label): تستخدم من أجل توجيه سير البرنامج ويتم كتابتها عند الحاجة .

كل تعليمة في لغة الباسكال يمكن أن تعطى علامة (label) بحيث يمكن الانتقال إلى هذه التعليمة متى نريد في البرنامج باستخدام تعليمة (goto). هذا العلامة (label) يجب أن تعرف مسبقاً في قسم التعريف في البرنامج وشكل التعريف يكون على الشكل التالي : يبدأ التعريف باستخدام الكلمة المحجوزة (label) هذه الكلمة تلحق بمجموعة من أسماء العلامات (label) مفصول بعضها عن بعض بفواصل، ثم ينتهي السطر باستخدام الفاصلة المنقوطة.

مثال (2):

label 6 , error, 11 , again;

- **الثوابت (Const):**

الثوابت هي تلك العناصر في البرنامج التي لا تتغير قيمها أثناء سير البرنامج (معالجة البرنامج)، هذه الثوابت تحدد قيمتها في بداية البرنامج وتبقى ثابتة كما هي وعندما يراد تغيير هذه القيمة فإنه يجب العودة إلى بداية البرنامج وإجراء التغيير في البرنامج، لذلك يُفضل في حالة استخدام ثوابت تتكرر عدة مرات في البرنامج استخدام رمز

واحد لهذا الثابت حيث أنه عند إجراء عملية التغير تحتاج فقط إلى تغيير قيمة هذا الرمز مرة واحدة فقط، وليس بعدد مرات التكرار في البرنامج .

- يحتوي قسم تعريف الثوابت على تعريف كل الرموز للثوابت ، حيث أن التعريف يبدأ بالكلمة المحجوزة (const) والتي يليها قائمة من الثوابت والتي تكون مفصولة عن بعضها بفواصل منقوطة ، بالنسبة لأسم الثابت يجوز أن يصل طوله حتى 127 رمزاً.

مثال (3):

Const

```
limit = 255;  
max = 1024;  
password = 'sesam';  
F = 1.34
```

- قسم تعريف الثوابت يمكن أن يحتوي أيضاً على تعريف الأنماط المجالية للمتحولات Type مثال ذلك:

Type n=min..max

حيث تمثل min و max حدود المجال المعرف وستتعرف إلى طريقة التعريف عند الحديث عن أنماط المعطيات المستخدمة في التربوباسكال.
في التربوباسكال يوجد ثوابت معرفة مسبقاً وموجودة دائماً في النظام والتي يمكن استخدامها بدون حاجة لتعريفها مسبقاً .

Name	Type	Value
pi	real	3.1415926536E+00
false	Boolean	
true	Boolean	
maxint	Integer	32767

نلاحظ من القيم الثابتة هنا أنه ليس من الضروري أن تكون قيم عددية وإنما يمكن أن تكون قيمة محرفية (سلسلة من الرموز) .

مثال (4):

```
const
n=20;
r=1.5;
f='turbo';
```

حيث:

n : من النمط الصحيح.

r : من النمط الحقيقي.

f : من النمط المحرفي ، ويجب أن توضع بين فاصلتين علويتين.

- المتحولات البسيطة والمتحولات ذات الأبعاد: في هذا القسم يتم تعريف المتحولات والمتحولات هي عبارة عن عناصر تتغير قيمتها أثناء تنفيذ البرنامج وهي عبارة عن أسماء تُستخدم لحجز مساحات من الذاكرة تستخدم لتخزين المعلومات. فعندما نعرف متحول في البرنامج فإن المترجم يقوم بتخصيص مساحة له من الذاكرة .

- كل متحول يظهر في البرنامج يجب تعريفه مسبقاً قبل ظهوره وذلك حتى يكون معروف مسبقاً من قبل المترجم قبل استخدامه.
- إن تعريف المتحولات في البرنامج يبدأ باستخدام الكلمة المحجوزة (var) والتي يليها رمز (اسم) أو عدة أسماء تفصل فيما بينها فواصل ، هذه القائمة من الأسماء تليها نقطتين (:). ثم يأتي بعد ذلك نمط (نوع) المتحول.

مثال (5):

```
var
  result,sum :real;
  I,x,y :integer;
  P:Boolean
  c1,c2:char;
```

بالنسبة لأسماء المتحولات المستخدمة يمكن أن يتكون اسم المتحول من حرف واحد أو عدة محارف يمكن أن تصل إلى 127 حرف في ونشير هنا إلى أنه يمكن أن تعرف ثوابت أو متحولات في قسم التعريف في البرنامج وذلك من دون أن تستخدم في قسم التعليمات ولا يؤثر ذلك في سير عمل البرنامج.

- التتابع والبرامج الفرعية : functions and procedure

التتابع والإجراءات هي عبارة عن مجموعة من العبارات أو التعليمات المجمعة والتي تُستخدم من أجل إنجاز عمل جزئي، وغالباً ما يمثل الإجراء برنامج مصغر، يمكن استدعائه في مواقع مختلفة من البرنامج الرئيسي أو من برنامج جزئي آخر، ولكن حتى يتم التعرف إلى التابع أو الإجراء فلا بد أولاً من تعريفه في قسم التعريف، وهذا ما سوف نتعرف إليه عند الحديث عن التتابع والإجراءات.

2- قسم التعليمات : في هذا القسم يتم تحديد الأعمال التي يراد إنجازها باستخدام (

أو بمساعدة) المعطيات المذكورة سابقاً في قسم التعريف .

في هذا القسم من البرنامج يجوز فقط استخدام المعطيات المذكورة في الجزء السابق

وكذلك التعليمات المعرفة مسبقاً في نظام _ الباسكال والتي تكون معروفة من قبل

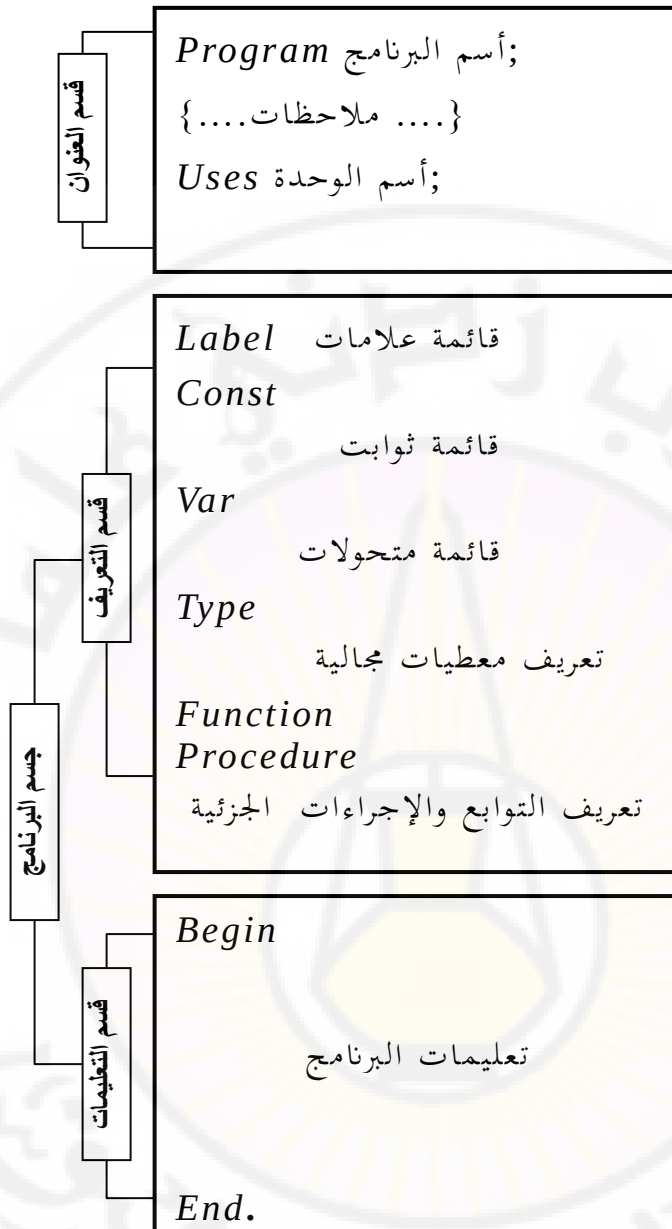
المترجم .

هذا القسم من البرنامج يبدأ بالتعليمة (الكلمة) begin في سطر مستقل وينتهي

بالتعليمة (end.) أيضاً في سطر مستقل وحيث أن النقطة هنا ضرورية .

ومن الأفضل أن تبدأ كلمة begin و end في العمود نفسه.

الشكل (2-6) يبين مخطط يوضح أقسام البرنامج في لغة التريبوباسكال.



الشكل (2-6): مخطط البرنامج في لغة التريباسكال

مثال (6):

حتى الآن لم يتم الحديث عن تعليمات لغة التريباسكال إلا أننا بهدف إيضاح أقسام البرنامج في لغة التريباسكال نريد عرض برنامج بسيط يقوم بحساب قيمة تعبير حسابي من الشكل:

$$Y=a+b-c$$

حيث a, b, y متحولات من النوع الصحيح و c ثابت.

```
Program test1;  
{ This is the first example}  
const  
     $c=5;$   
var  
     $a, b, y: real;$   
BEGIN  
     $a:=6; b:=3;$   
     $y:=a+b-c;$   
     $writeln(y);$   
END.
```



الفصل الثالث

عناصر لغة التروباسكال

3-1 الرموز الأساسية

3-2 أنواع المعطيات

3-3 العمليات الحسابية

3-4 عمليات المقارنة

3-5 العمليات المنطقية



عناصر لغة التروباسكال

3-1 الرموز الأساسية:

هناك مجموعة من الرموز الأساسية التي تستخدم في لغة التروباسكال ، هذه الرموز يمكن أن تكون رموز بسيطة أو مزدوجة أو كلمات ومنها رموز مخزنة مسبقاً في الحاسب هذه الرموز جميعاً يمكن الحصول عليها بدءاً من لوحة المفاتيح.

تصنف هذه الرموز كالآتي:

(1) الأحرف الكبيرة A-Z والأحرف الصغيرة a-z ولا يوجد فرق في لغة الباسكال بين استخدام الأحرف الكبيرة أو الصغيرة.

(2) الأرقام 0-9

(3) رموز خاصة : = , ^ , / , * , - , [] , { } , () , . , : , ' , ; , # , + , < , > ,

(4) رموز مزدوجة : = , < , > , < > , := , .. , (.) , (* *)

إن الأقواس { } لها معنى الأقواس نفسها (* *)

والأقواس [] لها معنى الأقواس نفسها (.)

أما الكلمات المستخدمة في لغة التروباسكال فهي كثيرة وعديدة وتختلف من مترجم لآخر كما أشرنا سابقاً.

3-2 أنواع المعطيات:

تُستخدم في لغة التروباسكال عدة أنواع من المعطيات وهي:

Integer , Real , Boolean , Char , String

3-2-1 النوع الصحيح Integer : وهي عبارة عن أعداد صحيحة في التربوباسكال تقع في المجال $+32767 - 32768$ وهي تحتاج في الذاكرة إلى 2byt ومن خلال هذا التمثيل يمكن تخزين قيم ضمن المجال $(1 - 2^{15}) - (-2^{15})$ ، حيث أن البت السادس عشر يُستخدم هنا لتمثيل الإشارة، ويُستخدم في لغة التربوباسكال ثابت معرف مسبقاً يُدعى MaxInt وقيمته هي أكبر قيمة في المجال المحدد (32767) وهنا يجب بشكل خاص ملاحظة أن تجاوز المجال المحدد للأعداد الصحيحة سوف يسبب خطأً في النتائج وهذا التجاوز لن يسبب إيقاف تنفيذ البرنامج كما هو الحال عند تجاوز المجال المحدد للأعداد الحقيقية.

مثال (1):

عند إضافة العدد 1 إلى العدد 32767 فإن الناتج سيكون العدد -32768 وليس العدد 32768. وكذلك الأمر فإنه عند جمع العدد 2 إلى العدد 32767 فإن الناتج سيكون -32767 وليس 32769. هذه الحالة تُسمى حالة فيض (overflow) وكما ذكرنا لا يتم الإعلان عنها أثناء سير البرنامج. إذاً يجب الانتباه هنا إلى العمليات التي تجري ضمن البرنامج والنتائج البينية لها يجب أن تقع ضمن المجال المسموح وإلا فسوف نصل على نتائج خاطئة. أيضاً زدتنا لغة التربو باسكال بأنماط أخرى لنوع المعطيات صحيح وهي تختلف عن النوع الصحيح Integer بمجال التمثيل، وهذه الأنواع ممثلة في الجدول الآتي:

Type	Size	Minimum Value	Maximum Value
ShortInt	1 Byte	-128	127
Byte	1 Byte	0	255
Integer	2 Bytes	-32768	32767
Word	2 Bytes	0	65535
LongInt	4 Bytes	-2,147,483,648	2,147,483,647

الجدول (1-3): أنماط المعطيات من النوع الصحيح.

مثال (2):

يتم تعريف هذه المتحولات في البرنامج على الشكل التالي:

Var

m,n: integer;

s,d: byte;

2-2-3 النوع الحقيقي Real: وهي عبارة عن الأعداد الكسرية التي تحتوي على

الفاصل العشري، وهذا النمط من المعطيات يقع في المجال :

$2.9 * 10^{-39}$

through

$1.7 * 10^{38}$

$- 2.9 * 10^{-39}$

through

$- 1.7 * 10^{38}$

ويحتاج هذا النموذج في الذاكرة إلى 6 بايت .وتقدم لغة التريباسكال أنماط مختلفة لنوع

المعطيات الحقيقي وذلك كما في الجدول الآتي:

Type	Size	Minimum Value	Maximum Value
Single	4 Byte	$-1.5 \times 10^{-45} \dots -3.4 \times 10^{38}$	$+1.5 \times 10^{-45} \dots +3.4 \times 10^{38}$
Real	6 Byte	$-2.9 \times 10^{-39} \dots -1.7 \times 10^{38}$	$+2.9 \times 10^{-39} \dots +1.7 \times 10^{38}$
Double	8 Bytes	$-5.0 \times 10^{-324} \dots -1.7 \times 10^{308}$	$+5.0 \times 10^{-324} \dots +1.7 \times 10^{308}$
Comp	8 Bytes	$-2^{63} + 1$	$2^{63} - 1$
Extended	10 Bytes	$-3.4 \times 10^{-4932} \dots -1.1 \times 10^{4932}$	$+3.4 \times 10^{-4932} \dots +1.1 \times 10^{4932}$

الجدول (2-3): أنماط المعطيات من النوع الحقيقي.

من أجل التمثيل العلمي لهذا النوع من المعطيات تُستخدم طريقة الفاصلة العائمة، حيث الشكل العام لهذا التمثيل هو كالتالي:

$$Z = ME \pm S$$

حيث:

Z: يمثل العدد المراد تمثيله.

M: العدد ممثل بطريقة الفاصلة العائمة.

E: الأساس ويساوي 10.

S: الأس.

فمثلاً يمكن تمثيل النمط Real بهذه الطريقة على الشكل الآتي:

$$-2.9E-39 \dots -1.7E+38$$

وهنا نشير إلى أنه عندما يحصل تجاوز (overflow) للمجال المحدد لهذه الأنماط المختلفة فإن البرنامج يتوقف ويعلم عن وقوع خطأ.

مثال (3):

1234.56 1.0 1E6 2E-7
-1.2345678901E+12

مثال (4):

يتم تعريف هذه المتحولات في البرنامج على الشكل التالي:

```
Var  
x,y: real;  
s,d: single;
```

3-2-3 النوع الحرفي Char:

إن قيم هذا النوع من المعطيات هي عبارة عن رموز جدول الـ ASCII (الجدول 3-3) ورموز الـ ASCII العددية هي عبارة عن 256 محرف تقع في المجال (0-255) وتضم محارف لوحة المفاتيح بالإضافة إلى بعض محارف التحكم. ويحتاج هذا النوع من أجل التخزين في الذاكرة إلى (1 Byte).

وهنا نشير إلى أنه عند استخدام هذا النوع من المتحولات فإن القيم المسندة (الثابت الحرفي) يجب أن توضع بين فاصلتين علويتين وذلك كالآتي:

```
ch1='a'
```

مثال (5):

يتم تعريف هذه المتحولات في البرنامج على الشكل الآتي:

```
Var  
ch1,ch2: char;
```

الرمز العشري	المحرف	الرمز العشري	المحرف	الرمز العشري	المحرف	الرمز العشري	المحرف
0	NUL	33	!	65	A	97	a
1	SOH	34	"	66	B	98	b
2	STX	35	#	67	C	99	c
3	ETX	36	\$	68	D	100	d
4	EOT	37	%	69	E	101	e
5	ENQ	38	&	70	F	102	f
6	ACK	39	'	71	G	103	g
7	BEL	40	(	72	H	104	h
8	BS	41	)	73	I	105	i
9	HT	42	*	74	J	106	j
10	LF	43	+	75	K	107	k
11	VT	44	,	76	L	108	l
12	FF	45	-	77	M	109	m
13	CR	46	.	78	N	110	n
14	SO	47	/	79	O	111	o
15	SI	48	0	80	P	112	p
16	DLE	49	1	81	Q	113	q
17	DC1	50	2	82	R	114	r
18	DC2	51	3	83	S	115	s
19	DC3	52	4	84	T	116	t
20	DC4	53	5	85	U	117	u
21	NAK	54	6	86	V	118	v
22	SYN	55	7	87	W	119	w
23	ETB	56	8	88	X	120	x
24	CAN	57	9	89	Y	121	y
25	EM	58	:	90	Z	122	z
26	SUB	59	;	91	[	123	{
27	ESC	60	<	92	\	124	!
28	FS	61	=	93	]	125	}
29	GS	62	>	94	^	126	~
30	RS	63	?	95	_	127	DEL
31	US	64	@	96	`		
32	SP						
محارف التحكم							

الجدول (3-3): ترميز الحروف حسب نظام ASCII

3-2-4 النوع المحرفي- سلسلة String: يستخدم هذا النوع من أجل تعريف

المعطيات المحرفية وتتم عملية التعريف هنا باستخدام الكلمة string وذلك كما يلي :

string [10]

حيث يدل الرقم بين قوسين على طول السلسلة المحرفية أو عدد الرموز في السلسلة وإن هذا الرقم هو من النوع الصحيح ويقع في المجال (1-255)، يتم فيه تخزين الطول الحالي للمتحول

إن متحول محرفي بطول (n) يحتاج في الذاكرة إلى (n+1)byte حيث أن البايت الإضافي يتم فيه تخزين الطول الحالي للمتحول.

عندما نريد إسناد سلسلة محرفية إلى متحول فإنه يجب وضع هذه السلسلة بين فاصلتين علويتين كالآتي:

Name='Ahmed'

مثال (6):

يتم تعريف هذه المتحولات في البرنامج على الشكل الآتي:

Var

name: string[10];

3-2-5 النوع البولائي Boolean :

يتضمن هذا النوع من المعطيات المتحولات المنطقية والتي يكون لها قيمتين فقط إما أن تأخذ القيمة صح (True) أو القيمة خطأ (False) وهذا النوع من المعطيات يحتاج

لتخزينه في الذاكرة إلى بايت واحد. وتعرف على قيم هذا النوع من المعطيات عمليات المقارنة والعمليات المنطقية.

مثال (7):

يتم تعريف هذه المتحولات في البرنامج على الشكل التالي:

```
Var  
m,n: boolean;
```

3-2-6 النوع التعدادي (المجالي) Type:

في كثير من الحالات نحتاج إلى استخدام متحولات تقع قيمها في مجال محدد، وبالتالي لا بد هنا من تعريف هذا النوع من المتحولات في قسم التعريف في البرنامج كما ذكرنا سابقاً.

من أجل ذلك تؤمن لغة ترينوباسكال إمكانية تعريف متحولات ضمن مجال محدد وذلك من خلال استخدام التعبير Type وذلك على الشكل الآتي:

```
Type N=min..max
```

وتمثل القيمتان min و max هنا، حدود هذا المجال.

وهنا نشير إلى أن هذا النوع من المعطيات لا يمكن استخدامه مباشرة في البرنامج وإنما يجب أولاً التصريح عن هذه المتحولات.

مثال (8):

يتم استخدام هذه المتحولات في البرنامج على الشكل التالي:

```
Type year = 1900..2000;
```

```
Type day = 1..31;
Var
  y: year;
  d: day;
```

بـهذه الطريقة يمكن استخدام هذه المتحولات في البرنامج ويمكن هنا مثلاً تنفيذ عملية إسناد على الشكل التالي: $y:=1995$ بينما لا يمكن تنفيذ عملية إسناد من الشكل $y:=2006$ لأن هذه القيمة تقع خارج المجال المحدد.

3-3 العمليات الحسابية:

تُستخدم في لغة التروباسكال العمليات الحسابية التالية:

العملية	الرمز
الضرب	*
القسمة	/
الجمع	+
الطرح	-
ناتج القسمة	DIV
باقي القسمة	MOD

الجدول (3-4): العمليات الحسابية في لغة التروباسكال..

الدالة Div:

تُطبق هذه الدالة على الأعداد الصحيحة فقط ويكون ناتجها صحيحاً أيضاً، وناتج هذه الدالة يمثل ناتج القسمة الصحيح لطرفي الدالة بدون باقي.

الشكل العام لهذه الدالة:

$$i \text{ div } j$$

الناتج يكون عدد صحيح أصغر من i/j
عندما يكون i أو j موجبان أو سالبان فإن النتيجة ستكون موجبة.
أما إذا كان أحدهما سالباً فإن النتيجة ستكون سالبة.

مثال (9):

$$\begin{aligned} a &:= 123 \text{ div } 4 = 30 \\ b &:= 55 \text{ div } 3 = 18 \end{aligned}$$

الدالة **Mod**:

تُطبق هذه الدالة أيضاً على الأعداد الصحيحة فقط ويكون ناتجها صحيحاً أيضاً، وناتج هذه الدالة يمثل باقي القسمة الصحيح لطرفي الدالة. ويمكن حساب الناتج كما يلي:

$$i \text{ mod } j = i - (i \text{ div } j) * j$$

مثال (10):

$$\begin{aligned} a &:= 12 \text{ mod } 5 = 2 \\ b &:= 55 \text{ mod } 3 = 1 \end{aligned}$$

الرفع إلى قوة:

نلاحظ أنه في لغة التربوباسكال لا يوجد عملية خاصة بالرفع إلى قوة كما هي الحال في اللغات الأخرى، Basic و Fortran إلا أنه يمكن تنفيذ عملية الرفع إلى قوة باستخدام توابع رياضية أخرى مستخدمة في لغة التربوباسكال، كاستخدام التابع Sqr وهو تابع الرفع إلى قوة.

مثال(11):

$Sqr(x)$ (الرفع إلى القوة 2)
 $Sqr(x) * x$ (الرفع إلى القوة 3)
 $Sqr(Sqr(x))$ (الرفع إلى القوة 4)

يمكن أيضاً تنفيذ عملية الرفع إلى قوة باستخدام العلاقة:

$$xy = \text{EXP}(\text{LN}(x) * y)$$

مثال (12):

في البرنامج التالي يتم رفع أي عدد إلى قوة:

PROGRAM PowerOperator;

VAR

a, b :Real;

BEGIN

*Write('Enter the base and the exponent separated by a
space: ');*

Readln (a,b);

*Writeln ('The value of ',a:0:2,' raised to the power ',b:0:2,' is
, Exp(LN(a)*b):0:2)*

END.

3-3-1 التعبيرات الحسابية:

هي عبارة عن تراكيب من الثوابت والمتحولات والعمليات الحسابية، يتم حسابها في البرنامج وإنتاج قيم جديدة، وتكون قيمة التعبير في هذه الحالة قيمة حسابية.

مثال (13):

التعبير الحسابي التالي:

$$Y = \frac{x}{a+2} + 3b - 1$$

يمكن كتابته بلغة التريباسكال على الشكل الآتي:

$$Y = x / (a + 2) + 3 * b - 1$$

وهنا نجد أنه في مثل هذه الحالة لا بد من وضع الأقواس بالنسبة للتركيب في البسط، كم أنه يجب توضيح العمليات الحسابية.

3-2 أولوية تنفيذ العمليات الحسابية:

إن أولوية تنفيذ العمليات في تعبير حسابي تكون على الشكل التالي:

- 1- العمليات بين الأقواس وفي حالة وجود أقواس داخلية، فتنفذ أولاً الأقواس الداخلية ثم الخارجية.
- 2- عمليات الضرب ، القسمة ، Div ، Mod ، وفي حال ورود عدة عمليات في التعبير الحسابي فإن الأولوية تكون للعملية التي تقع إلى اليسار.
- 3- الجمع والطرح وفي حال ورود عدة عمليات في التعبير الحسابي فإن الأولوية تكون للعملية التي تقع إلى اليسار.

3-4 عمليات المقارنة:

نحتاج في البرنامج في كثير من الحالات إلى مقارنة قيمتين، وعملية المقارنة يمكن أن تُطبق على القيم العددية أو غير العددية وناتج المقارنة يكون دائماً إما صح (True) أو خطأ (False).

تُستخدم في لغة التروبو باسكال عدة عمليات للمقارنة. الجدول (3-5) يبين مقارنة بين عمليات المقارنة المستخدمة في الرياضيات وتلك المستخدمة في لغة التروبو باسكال. إذا تمت عملية المقارنة بين التعابير الرقمية فإنه يستخدم من أجل تقييم نتيجة المقارنة قيمة التعبير الرقمي. إذا تمت مقارنة تعابير محرفية بعضها ببعض؛ فإنه يستخدم لذلك الرموز المقابلة لمكونات هذه التعابير، وذلك حسب الترميز بنظام أسكي (ASCII).

المعنى	الرمز في لغة التروبو باسكال	الرمز في الرياضيات
يساوي	=	=
أصغر	<	<
أكبر	>	>
أصغر ويساوي	<=	≤
أكبر أو يساوي	>=	≥
لا يساوي	< >	≠

الجدول (3-5): عمليات المقارنة في لغة التروبو باسكال.

مثال (14):

النتيجة	العملية
True	15>10
False	6<4
False	-3>-1
True	'hello'='hello'
True	'E'>'A'
True	'Riad'>'jamal'
True	'geophysic'>'geology'

3-5 العمليات المنطقية:

تُستخدم في لغة التروبايسكال مجموعة من العمليات المنطقية و هي:

AND , OR , XOR , NOT

هذه العمليات تُستخدم كما سنلاحظ لاحقاً لتشكيل التعابير المنطقية المركبة؛ والتي بدورها تلعب دور الشرط في البرنامج.

1. العملية AND:

إذا كانت كل المداخل محققة فإننا نحصل على مخرج محقق (TRUE)، وما عدا ذلك نحصل على مخرج غير محقق (FALSE).

مثال (15):

العملية	النتيجة
$(5 < 8) \text{ AND } (3 < 4)$	TRUE
$(4 < 5) \text{ AND } (6 < 5)$	FALSE
$(2 < 3) \text{ AND } (3 < 2)$	FALSE
$(2 < -2) \text{ AND } (2 < 4)$	FALSE
$(x > 4) \text{ AND } (x \leq 7)$	الناتج يكون محققاً من أجل القيم $(x=5,6,7)$ وكذلك من أجل القيم العشرية الواقعة بين 4 و 7.

2. العملية OR:

يكون المخرج محققاً إذا كانت إحدى المداخل على الأقل محققة.

مثال (16):

العملية	النتيجة
$(4 < 6)$ OR $(5 < 8)$	TRUE
$(3 < 5)$ OR $(6 < 2)$	TRUE
$(1 < -1)$ OR $(2 < 4)$	TRUE
$(7 < 3)$ OR $(4 < 2)$	FALSE

3. العملية XOR:

يكون المخرج محققاً إذا كانت إحدى المداخل محققة فقط.

مثال (17):

العملية	النتيجة
$(4 < 6)$ OR $(5 < 8)$	FALSE
$(3 < 5)$ OR $(6 < 2)$	TRUE
$(1 < -1)$ OR $(2 < 4)$	TRUE
$(7 < 3)$ OR $(4 < 2)$	FALSE

4. العملية NOT:

يكون المخرج محققاً إذا كان المدخل غير محقق.

مثال (18):

النتيجة	العملية
TRUE	$9 > 6$
FALSE	$9 < 6$
TRUE	NOT ($9 < 6$)
FALSE	NOT ($6 < 9$)
FALSE	NOT NOT FALSE
FALSE	NOT ($4 > 7$) AND ($2 > 4$)
TRUE	NOT(($4 > 7$) AND ($2 > 4$))

الجدول (3-6) يبين ناتج كل من هذه العمليات:

A	B	AND	OR	XOR	NOT
T	T	T	T	F	F
T	F	F	T	T	F
F	T	F	T	T	T
F	F	F	F	F	T

الجدول (3-6): العمليات المنطقية في لغة التريو باسكال.

كما لاحظنا بحالة التعابير الحسابية، يمكن هنا التحدث عن تعابير منطقية أيضاً ويمكن التمييز بين تعابير بسيطة وأخرى مركبة، وناتج التعبير المنطقي يكون دائماً إما True أو False.

3-6 التعابير المنطقية:

لاحظنا سابقاً بأن التعابير الحسابية هي ناتج تركيب من الثوابت والمتحولات والعمليات الحسابية، ويكون ناتجها دائماً قيمة حسابية. أيضاً نجد هنا أن التعابير المنطقية هي ناتج تركيب من عمليات المقارنة، أو تركيب من عمليات المقارنة والعمليات المنطقية وفي كلا الحالتين يكون ناتج التعبير المنطقي قيمة منطقية.

مثال (19):

العملية	النتيجة
$5 > 3$	TRUE
$6 < 4$	FALSE
$(3 > 2) \text{ AND } (5 > 7)$	FALSE
$(3 > 2) \text{ OR } (5 > 7) \text{ AND } (9 > 6)$	TRUE
'C' > 'A'	TRUE



الفصل الرابع

تعليمات لغة التروباسكال

1-4 مقدمة

2-4 تعليمات إدخال المعطيات

3-4 تعليمات إخراج المعطيات

4-4 تعليمات الحلقات

5-4 التعليمات الشرطية

6-4 تعليمات الانتقال (القفز)

7-4 المعطيات المركبة



تعليمات لغة التروباسكال

4-1 مقدمة:

تعرفنا سابقاً إلى تركيب البرنامج وعلى أنواع المعطيات التي تُستخدم في لغة التروباسكال، وحتى يتم تحقيق الهدف من البرنامج لابد من توفر معطيات (بيانات) لمعالجتها في البرنامج، من أجل توفير هذه المعطيات لا بد من البحث عن طرائق إدخال لهذه المعطيات إلى البرنامج.

من أجل ذلك توفر لغة التروباسكال عدة طرائق لإدخال المعطيات:

- 1- مباشرة في جسم البرنامج (قسم التعليمات) باستخدام عبارة الإسناد.
- 2- عن طريق لوحة المفاتيح والشاشة.
- 3- عن طريق ملفات خاصة وهذا يتطلب استخدام تعليمات خاصة سنتحدث عنها لاحقاً.

4-2 تعليمات إدخال المعطيات:

1- عبارة الإسناد (=):

إن الذاكرة في الحاسوب تحتوي على مواقع عديدة فارغة لتخزين المعطيات، ومن أجل التخزين في هذه المواقع لا بد من إعطائها أسماء، وكنا قد عرفنا سابقاً بأن المتحولات هي عبارة عن أسماء تُستخدم لحجز مواقع في الذاكرة، إذاً يمكن الآن تسمية هذه المواقع بأسماء متحولات مختلفة ويمكن أيضاً تخزين المعطيات تحت هذه الأسماء من خلال إسناد قيم معطيات إلى هذه المتحولات من خلال عبارة الإسناد.

والشكل العام لعبارة الإسناد هو كالتالي:

قيمة =: متحول

وهنا يتم دائماً إسناد القيمة الموجودة إلى اليمين إلى المتحول الموجود إلى اليسار، والقيمة الموجودة لليمين يمكن أن تكون ثابت أو متحول أو تعبير حسابي.

مثال (1):

$x:=5$

يتم إسناد العدد الصحيح 5 إلى المتحول x .

$y:=x+1$

يتم إسناد قيم التركيب $x+1$ إلى المتحول y .

$n:=n+1$

يتم إسناد قيم التركيب $n+1$ إلى المتحول n .

وهنا لا بد من الإشارة إلى أنه عند استخدام عبارات الإسناد لا بد من أن كل من القيمة إلى اليمين والمتحول إلى اليسار من النوع نفسه.

ومن أجل إيضاح العلاقة بين الأنواع المختلفة للمعطيات نعرض الجدول (4-1)، وهنا يمثل سطر الجدول المتحول، بينما يمثل العمود القيمة المسندة.

نوع المتحول	Integer	Byte	Real	Char	String	Boolean
Integer	+	+	-	-	-	-
Byte	±	+	-	-	-	-
Real	+	+	+	-	-	-
Char	-	-	-	+	-	-
String	-	-	-	+	+	-
Boolean	-	-	-	-	-	+

الجدول (4-1): العلاقة بين أنواع المعطيات المختلفة.

وُثُشير الإشارة (+) إلى أن الإسناد صحيحاً، بينما تُشير الإشارة (-) إلى أن الإسناد غير صحيحاً، في حين تُشير الإشارة ($\pm$) إلى أن الإسناد صحيح على مجالاً جزئياً.

مثال (2):

في هذا المثال نريد حساب قيمة تعبيرين مختلفين x و y ومن ثم مبادلة قيم المتحولين، بحيث تُسند قيمة x لـ y وقيمة y لـ x :
التعبير الأول: $x=a+b$ ، التعبير الثاني: $y=a-b$

```
Program test1;  
var  
   $a,b,x,y,z$ : Integer;  
BEGIN  
   $a:=10$ ;  $b:=4$ ;  
   $x:=a+b$ ;  
   $y:=a-b$ ;  
  writeln( $x,y$ );  
   $z:=x$ ;  
   $x:=y$ ;  
   $y:=z$ ;  
  writeln( $x,y$ );  
END.
```

2- التعليمة **READ** ، **READLN**:

الشكل العام لهذه التعليمة هو كالتالي:

```
READ( $var1, var2, \dots, varn$ )  
READLN( $var1, var2, \dots, varn$ )
```

حيث $var1, var2, \dots, varn$ متحولات.

عند استخدام هذه التعليمات يتم إدخال المعطيات من خلال لوحة المفاتيح ويتم عند الانتهاء من عملية الإدخال الضغط على المفتاح Enter. لا يوجد فارق مهم بين تعليمة Read و تعليمة Readln عندما تجري عملية الإدخال من خلال لوحة المفاتيح، إلا أن الفرق يكون أكبر ومهم عند استخدام هاتين التعليمتين بإدخال المعطيات عن طريق الملفات وهذا ما سنتحدث عنه لاحقاً .

مثال (3):

في هذا المثال يتم كتابة البرنامج السابق Test2، ولكن يتم هنا تنفيذ عملية الإدخال باستخدام التعليمة Readln بدلاً عن عملية الإسناد.

```
Program test2;  
var  
    a,b,x,y,z: Integer;  
BEGIN  
    readln(a);  
    readln(b);  
    x:=a+b;  
    y:=a-b;  
    writeln(x,y);  
    z:=x;  
    x:=y;  
    y:=z;  
    writeln(x,y);  
END.
```

عند تنفيذ هذا البرنامج نجد أن التنفيذ يتوقف عند السطر Readln(a) وهنا يجب على المستخدم إدخال قيمة من لوحة المفاتيح، هذه القيمة سوف تُسند إلى المتحول a. وهكذا بالنسبة لتعليمة Readln التالية.

وهنا يجب الانتباه إلى أن القيمة المدخلة من لوحة المفاتيح يجب أن تكون متوافقة مع نوع المتحول المدون خلف تعليمة Readln. الشيء نفسه يسري أيضاً في حالة استخدام التعليمة Read.

3-4 تعليمات إخراج المعطيات:

لاحظنا أنه أصبح باستطاعتنا إدخال معطيات إلى الحاسوب ومعالجة هذه المعطيات من خلال البرنامج إلا أنه لا نستطيع الحكم على هذه الخطوات (الإدخال، البرنامج) إلا عند ملاحظة النتائج ومعرفة فيما إذا كانت صحيحة أو خاطئة. من أجل عرض (إخراج) النتائج في الحاسوب عند استخدام لغة الباسكال توجد عدة طرائق:

- 1- إخراج النتائج عن طريق شاشة الحاسب.
 - 2- إخراج النتائج عن طريق الطابعة.
 - 3- إخراج النتائج عن طريق ملف وهذا يتطلب تعليمات خاصة من أجل تعريف الملف وفتحه وإغلاقه سنتعرض لها لاحقاً.
- وهنا لا بد في البداية أن نحدد للحاسوب من خلال البرنامج ما هي النتائج التي نريد إخراجها وما هو الشكل أو الترتيب الذي نريد به إخراج هذه النتائج. من أجل إخراج النتائج في هذه الحالات الثلاث تُستخدم تعليمة WRITE, WRITELN, ولكن استخدامها يختلف من حالة إلى أخرى إلا أننا سنناقش حالياً فقط إخراج النتائج عن طريق شاشة الحاسوب.
1. التعليمة WRITE:
- الشكل العام لهذه التعليمة:

WRITE (var1, var2,, varn);

حيث أن المتحولات var1, var2,, varn هي عبارة عن متحولات.

2. التعليمة WRITELN:

الشكل العام لهذه التعليمة :

WRITELN (var1, var2,, varn);

ولها مفعول التعليمة نفسها Write إلا أن الاختلاف يتمثل بأنه عند استخدام التعليمة Write يتم إخراج قيم المتحولات أو الثوابت لأي عملية إخراج لاحقة لـ Write في السطر نفسه الذي أخرجت فيه النتائج السابقة، أما في حالة التعليمة Writeln فإن الأمر يختلف حيث أن المشيرة ستقفز إلى السطر التالي للسطر بعد انتهاء عملية الإخراج أي أن أي عملية إخراج لاحقة ستجري في السطر التالي.

مثال (4):

Write ('Turbo');Write('Pascal');

النتيجة في هذه الحالة ستكون TurboPascal ، أما إذا كتبنا:

Writeln("Turbo");Write('Pascal');

فإن النتيجة ستكون:

Turbo
Pascal

مثال (5):

البرنامج التالي يبين الفرق بين استخدام التعليمة Write والتعليمة Writeln:

Program test3;

var

x,y,z:integer;

BEGIN

x=5 ;y=6; z=7;

write(x); write(y);


```

write(x);writeln(y);
write(z);
write('Turbo ');write('Pascal ');
writeln('Turbo');writeln('Pascal');
END.

```

بعد تنفيذ هذا البرنامج ستكون النتيجة:

```

5656
7
Turbo Pascal Turbo
Pascal

```

مثال (6):

البرنامج التالي يبين نتائج بعض العمليات الحسابية:

```

Program Test4;
Var
a, b :Integer;
BEGIN
  a := 25;
  b := 2;
  Writeln ('a=',a);
  Writeln ('b=',b);
  Writeln ('a+b=',a+b);
  Writeln ('a-b=',a-b);
  Writeln ('a*b=',a*b);
  Writeln ('a/b=',a/b);
  Writeln ('a div b=',a DIV b);
  Writeln ('a mod b=',a MOD b);
END.

```

بعد تنفيذ البرنامج ستكون النتيجة كالتالي:

```
a=25
b=2
a+b=27
a-b=23
a*b=50
a/b= 1.2500000000E+01
a div b=12
a mod b=1
```

4-3-1 طرائق إخراج النتائج:

إن طريقة عرض النتائج تعتبر مهمة في البرنامج، ولغة التريباسكال تؤمن لنا عدة طرائق لإخراج النتائج وذلك حسب نوع المعطيات المراد إخراجها.

4-3-1-1 طريقة إخراج المعطيات من النوع المحرفي:

لاحظنا سابقاً أن المعطيات المحرفية يمكن أن تكون من النوع Char، وهنا تكون قيمة المتحول أو الثابت مساوية لمحرف واحد فقط، ويمكن أن تكون من النوع المحرفي String، وفي هذه الحالة يكون المتحول أو الثابت مكون من عدة محارف. وفي كلتا الحالتين يتم عند إخراج هذه المتحولات أو الثوابت باستخدام تعليمات الإخراج، عرض هذه المحارف كاملة.

مثال (7):

البرنامج التالي يبين طريقة إخراج المعطيات المحرفية المعرفة باستخدام String:

```
program Test5;
var
  Fname: string[10];
  Lname: string[10];
  st: char;
```

```

BEGIN
  St:= ' ';
  readln(Fname);
  readln(Lname);
  writeln('your Name is:',Fname,st,Lname);
  readln;
END.

```

مثال (8):

البرنامج التالي يبين البرنامج في المثال (7) بعد إجراء بعض التعديلات عليه:

```

program Test6;
label 5;
const st=' ';
var
  Fname: string[10];
  Lname: string[10];
BEGIN
  5: Write ('your FirstName:');readln(Fname);
    Write ('your LastName');readln(Lname);
    Writeln ('your Name is:',Fname,st,Lname);
    readln;
    goto 5;
END.

```

2-1-3-4 طريقة إخراج المعطيات من النوع الصحيح:

من أجل إخراج المعطيات الصحيحة يمكن استخدام طريقتين:

الطريقة الأولى: يتم استخدام الثابت أو المتحول الصحيح في تعليمة الإخراج بدون استخدام أي طريقة لتوصيف عملية الإخراج، مثال ذلك أن نكتب :

Writeln(x);

حيث X هنا متحول صحيح، وفي هذه الحالة يتم حجز عدد خانات يساوي عدد خانات العدد المراد إخراجها، وتُملأ هذه الخانات بدون ترك أي فراغ. الطريقة الثانية: استخدام واصفات (Format) من أجل التحكم بعملية الإخراج، والشكل العام لهذه الواصفات هو كالآتي:

Writeln(Write)(Z:M)

في هذه الحالة يتم حجز M خانة للعدد ويتم ملئ الخانات من اليمين إلى اليسار وفي حال كانت الخانات المحجوزة أكبر من خانات العدد المراد إخراجها فإنه سيتم ترك فراغات إلى اليسار.

مثال(9):

البرنامج التالي يبين طريقة إخراج المعطيات من النوع الصحيح باستخدام الواصفات .Format

Program Test7;

```
var
  a,b,c:integer;
BEGIN
  a:=3231;
  b:=-5432;
  c:= 7765;
  writeln('1234567890');
  writeln(a);
  writeln(b);
  writeln(c);
  writeln(a:5);
  writeln(b:6);
  writeln(c:7);
  readln;
END.
```

بعد تنفيذ هذا البرنامج سوف نحصل على النتائج التالية:

1234567890
3231
-5432
7765
3231
-5432
7765

4-3-1-3 طرائق إخراج المعطيات من النوع الحقيقي:

من أجل إخراج الأعداد الحقيقية توجد طريقتين:

الطريقة الأولى: كما هي الحال في إخراج الأعداد الصحيحة، فإنه يتم استخدام الثابت الحقيقي أو المتحول من دون استخدام أي طريقة للتحكم بالإخراج. في هذه الحالة تجري عملية الإخراج بطريقة الفاصلة العائمة. وهنا يتم حجز 17 خانة لإخراج القيمة الحقيقية، 4 خانات منها لإخراج الأساس E وخانتين للأش وإشارة الأس، و 13 خانة لإخراج العدد M مع الإشارة والفاصلة العشرية.

مثال (10):

البرنامج التالي يبين طريقة إخراج المعطيات من النوع الحقيقي باستخدام الواصفات

:Format

Program Test8;

var

a,b,c,d: real;

BEGIN

a:=123.45;

b:=-543.546;

```

c:= 1.654;
d:=-0.000078;
writeln('12345678901234567890');
writeln(a);
writeln(b);
writeln(c);
writeln(d);
readln;
END.

```

بعد تنفيذ هذا البرنامج تكون النتائج على الشكل التالي:

```

12345678901234567890
1.2345000000E+02
-5.4354600000E+02
1.6540000000E+00
-7.8000000000E-05

```

الطريقة الثانية: يمكن هنا التحكم بطريقة الإخراج ويمكن هنا إخراج المعطيات بطريقتين: طريقة الفاصلة الثابتة وطريقة الفاصلة العائمة.

في كلا الحالتين يتم استخدام واصفات للتحكم بطريقة الإخراج.

فعند الإخراج بطريقة الفاصلة الثابتة يكون شكل الواصفات كالتالي:

Writeln(Z:M:N);

وهنا يمثل Z العدد المراد إخراجهِ و M عدد الخانات الكلي المحجوز للعدد بما فيه الفاصلة العشرية والإشارة وعدد الخانات بعد الفاصلة N ، ويكون عدد الخانات قبل الفاصلة بما فيها خانة الإشارة مساوياً إلى $(M-N-1)$.

مثال (11):

البرنامج التالي يبين طريقة إخراج المعطيات من النوع الصحيح باستخدام الواصفات
:Format

Program Test9;

var

a,b,c,d:real;

BEGIN

a:=123.456;

b:=-543.546;

c:= 1.654;

d:=-0.0000799;

writeln('12345678901234567890');

writeln(a:8:2);

writeln(b:9:3);

writeln(c:7:2);

writeln(d:10:2);

readln;

END.

بعد تنفيذ هذا البرنامج سوف نحصل على النتائج التالية:

12345678901234567890

123.46

-543.546

1.65

-0.00

يمكن أيضاً بالطريقة الثانية إخراج المعطيات بطريقة الفاصلة العائمة وباستخدام الواصفات
وفي هذه الحالة تكون الواصفات على الشكل التالي:

Writeln(Z:M);

مثال (12):

في هذا المثال نريد استخدام البرنامج السابق Test9 مع إجراء بعض التعديلات:

Program Test10;

var

a,b,c,d:real;

BEGIN

a:=123.456;

b:=-543.546;

c:= 1.654;

d:=-0.0000799;

writeln('12345678901234567890');

writeln(a:8);

writeln(b:9);

writeln(c:7);

writeln(d:10);

readln;

END.

بعد تنفيذ هذا البرنامج سوف نحصل على النتائج التالية:

12345678901234567890
1.2E+02
-5.44E+02
1.7E+00
-7.990E-05

يجب الانتباه إلى أنه عند إخراج الأعداد الحقيقية فإن الخانة الأولى تكون بشكل دائم محجوزة للإشارة وجدت الإشارة أم لم توجد وهذا يتم أيضاً في حالة الوصف بطريقة الفاصلة العائمة .

إذا كان عدد الخانات المخصص للجزء M أقل من عدد خانات العدد المراد إخراج
فإنه تجري عملية التصحيح بحيث يتم إخراج العدد بالشكل الصحيح.
عدد الخانات قبل الفاصلة مع الإشارة = $M-N-1$

4-4 تعليمات الحلقات:

1-4-4 تعليمة الحلقة FOR:

في حالات كثيرة نحتاج إلى تكرار تعليمة أو مجموعة تعليمات ما عدة مرات في البرنامج،
بحيث يتم في كل مرة إجراء تغيير قيمة متحول ما أو أكثر في هذه التعليمة للحصول ناتج
جديد.

أيضاً قد نحتاج في حالات كثيرة إلى استخدام متحولات ذات أبعاد، أو ذات دلائل
حيث نستخدم اسم المتحول نفسه ولكن بدلائل مختلفة.
في كل هذه الحالات يفيدنا كثيراً في البرمجة استخدام تعليمات التكرار أو الحلقات والتي
ستبدأ بها بالتحدث عن حلقة FOR.
إن الشكل العام لهذه الحلقة هو :

FOR I:=B TO E DO

حيث:

I: يمثل متحول الحلقة .

B: تمثل القيمة البدائية لمتحول الحلقة .

E: القيمة النهائية لمتحول الحلقة .

إن متحول الحلقة هو متحول بسيط من النمط الصحيح أو النمط المحرفي Char، إذا
كان متحول الحلقة من النمط الصحيح فإن الخطوة ستكون بمقدار (1) ، أما إذا كان
متحول الحلقة من النمط Char فإن الخطوة ستكون الحرف التالي في الترتيب الأبجدي.
في هذه الحالة يجب أن تكون القيمة البدائية B أصغر من القيمة النهائية E وبما أن

الخطوة هي (1) فإن القيمة البدائية تزداد في كل دورة بمقدار (1) حتى تصل إلى القيمة النهائية وفي كل دورة يتم اختبار قيمة متحول الحلقة فعندما يصبح متحول الحلقة $I=E$ فإنه يتم تنفيذ الحلقة للمرة الأخيرة وينتقل التنفيذ بعدها إلى التعليمة الواقعة بعد الحلقة مباشرة .

إذا أردنا تكرار تعليمة واحدة فقط باستخدام الحلقة فإنه يمكن كتابة هذه التعليمة مباشرة بعد سطر تعليمة For .

أما إذا كان لدينا أكثر من تعليمة نريد تكرارها فإنه يجب وضع هذه التعليمات بين تعليمتي Begin و End بحيث تأتي Begin بعد تعليمة For ثم تأتي مجموعة التعليمات المراد تكرارها ثم تعليمة End التابعة لـ Begin.

مثال(13):

البرنامج التالي يبين طريقة استخدام الحلقة For:

```
Program Test11;  
var  
    i: integer;  
    c: char;  
BEGIN  
    for i:=1 to 10 do  
        write (i, ' ');  
    for c:='A' to 'J' do  
        write (c, ' ');  
    readln;  
END.
```

بعد تنفيذ هذا البرنامج سوف نحصل على النتائج التالية:

1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 ,
A , B , C , D , E , F , G , H , I , J ,

2-4-4 تعليمة الحلقة العكسية FOR:

لاحظنا أنه في حالة حلقة For يتم تكرار جزء من البرنامج عدة مرات وبحيث يتم تغيير متحول الحلقة بخطوة متزايدة، يوجد شكل آخر للحلقة يتم فيه تغيير متحول الحلقة بشكل عكسياً.
الشكل العام لهذه الحلقة:

FOR I:= B DOWNT0 E DO

نلاحظ أن هذا الشكل يستخدم عندما تكون القيمة البدائية B أكبر من القيمة النهائية E.
وهنا تصبح الخطوة للحلقة (-1)، وليس (1) ما عدا ذلك تنطبق كل القواعد في حالة الحلقة الأولى على الحالة الثانية .

مثال (14):

البرنامج التالي يبين طريقة استخدام حلقة For العكسية:

```
Program Test12;  
var  
i: integer;  
c:char;  
BEGIN
```

```

for i:=10 downto 1 do
  write(i,' ');writeln;
for c:='J' downto 'A' do
  write (c,' ');
Readln;
END.

```

بعد تنفيذ هذا البرنامج سوف نحصل على النتائج التالية:

10, 9,8 , 7 , 6 , 5 , 4, 3 ,2 ,1,
J , I ,H , G , F , E , D, C ,B ,A,

مثال (15):

في هذا البرنامج يتم حساب المتوسط الحسابي لثلاث مقررات لعشرة طلاب ويتم إخراج النتيجة في كل مرة على الشاشة:

```

Program Test13;
uses crt;
var
  i: integer;
  stname:string [10];
  Math,Chem,Eng,m:real;
BEGIN
  Clrscr;
  for i:=1 to 10 do
    begin
      write('Enter the studentname Number:
        ',i,':');readln(stname);
      write ('Enter The Marks for the student :',stname, ':');
      readln (Math,Chem,eng);
    
```

```

m:=(math+Chem+Eng)/3;
write('the average for student :',sname,'=',m);
readln;
end;
END.

```

مثال(16):

في هذا المثال يتم حساب المجموع لعدة أعداد:

```

Program Test14;
var
  i:integer;
  n,sum:real;
BEGIN
for i:=1 to 5 do
  begin
    readln (k);
    sum:=sum+k;
  end;
  writeln('sum=',sum);
  readln;
END.

```

3-4-4 تعليمة الحلقة :WHILE... DO

لاحظنا في حلقة For ..Do أن هذه الحلقة تجعل الحاسوب يكرر عملاً ما عدة مرات ويرتبط ذلك بالقيمة البدائية والنهائية للحلقة. كذلك الأمر في حلقة While....Do، فهذه الحلقة تجعل الحاسوب يكرر عملاً ما عدة مرات، إلا أن عدد مرات التكرار يتعلق هنا بقيمة التعبير المنطقي المرتبط بهذه الحلقة حيث أن عملية تكرار العمل تستمر طالما يكون ناتج هذا التعبير المنطقي محقق .

إن الشكل العام لهذه الحلقة هو :

DO تعبير منطقي (شرط) WHILE

وهنا يجب أن نشير إلى أنه قبل عملية البدء بالتكرار يتم اختبار الشرط في بداية الحلقة فإذا لم يكن محققاً فإن عملية التنفيذ للحلقة لا تتم حيث يتم القفز فوق التعليمات الموجودة داخل الحلقة إلى الحلقة التالية مباشرة للحلقة.

نلاحظ أنه كما هو الحال في حلقة For.....Do إذا كان الجزء المراد تكراره (جسم الحلقة) بسيط أي مؤلف من تعليمة واحدة فقط, فإنه لا داعي لوضعه بين عبارتي Begin و End.

مثال (17):

البرنامج التالي يبين طريقة استخدام حلقة While:

```
Program Test15 ;  
var  
  x:real;  
BEGIN  
  x:=1;  
  while x>0 do  
    readln(x);  
END.
```

مثال (18):

البرنامج التالي يبين استخدام آخر لحلقة While:

```
Program Test16;  
var  
  a:real;  
BEGIN  
  a:=1;
```

```

while a<=10 do
  begin
    a:=a+1;
    writeln(a);
  end;
END.

```

4-4-4 :REPEAT....UNTIL تعليمة الحلقة

هذه الحلقة تشبه الحلقتين السابقتين For و While.

والشكل العام لها:

```

Repeat
.
.
.
Until <تعبير منطقي>

```

تقوم هذه التعليمة بتكرار جسم الحلقة (مجموعة التعليمات بين عبارتي Repeat و Until) مرّاتٍ عدّة حتى يصبح ناتج التعبير المنطقي محقق. وهنا نلاحظ على عكس تعليمة While أن اختبار الشرط يتم في نهاية الحلقة. هذا يعني أن تعليمات جسم الحلقة يجب أن تنفذ مرة واحدة على الأقل حتى إذا كان ناتج التعبير المنطقي غير محقق.

مثال (19):

البرنامج التالي يبين طريقة استخدام حلقة Repeat:

```

Program Test17 ;
var
  x,y,z: real;
BEGIN
  x:=1;
  repeat

```

```
    read(x);  
    until x>0  
END.
```

مثال (20):

البرنامج التالي يبين استخدام آخر حلقة Repeat:

```
Program Test18 ;  
var  
    ch:char;  
BEGIN  
    repeat  
        readln(ch);  
    until ch='y';  
END.
```

مثال (21):

البرنامج التالي يبين استخدام آخر حلقة Repeat:

```
Program Test19 ;  
var  
    a,b:real;  
BEGIN  
    b:=0;  
    repeat  
        a:=1;  
        b:=b+a;  
    until b>10;  
END.
```

4-4-5 تداخل الحلقات:

نحتاج في كثير من الحالات إلى تنفيذ عملية تداخل للحلقات، وهنا يتم كتابة حلقة ضمن حلقة أخرى، ويجب الانتباه عند تداخل الحلقات أن لا يكون متحول الحلقة متماثلاً، كما أنه يجب أن لا يحصل تقاطع بين الحلقات المتداخلة.

مثال (22):

في البرنامج التالي يتم تنفيذ عملية تداخل لحلقتين، الحلقة الأولى متحولها *i* والحلقة الثانية متحولها *j*:

```
Program Test20;  
uses crt;  
var  
  i,j,x: integer;  
BEGIN  
  clrscr;  
  for i:=1 to 3 do  
    for j:=1 to 4 do  
      begin  
        x:=i+j;  
        writeln('i=',i,' ', 'j=',j,' ', 'x=',x);  
      end;  
  Readln;  
END..
```

بعد تنفيذ هذا البرنامج نحصل على النتائج الآتية:

i=1	j=1	x=2
i=1	j=2	x=3
i=1	j=3	x=4
i=1	j=4	x=5
i=2	j=1	x=3
i=2	j=2	x=4
i=2	j=3	x=5
i=2	j=4	x=6
i=3	j=1	x=4
i=3	j=2	x=5
i=3	j=3	x=6
i=3	j=4	x=7

من النتائج نلاحظ أنه من أجل كل قيمة لمتحول الحلقة الخارجية (الحلقة التي متحولها i)، يتم تنفيذ الحلقة الداخلية (الحلقة التي متحولها j) كاملةً .

مثال (23):

البرنامج التالي يبين حالة الحلقات غير المتداخلة:

```
Program Test21;
var
  i,j,k: integer;
BEGIN
  for i:=1 to 3 do
    writeln('x=',i+1);
  for j:=1 to 2 do
    writeln('i=', i, ' ', 'j=', j);
  readln;
END.
```

بعد تنفيذ البرنامج نحصل على النتائج الآتية:

```
x=2  
x=3  
x=4  
i=3 j=1  
i=3 j=2
```

نلاحظ أن الحلقتين في البرنامج ليستا متداخلتين. من أجل جعلها متداخلة يجب وضع عبارة Begin بعد التعليمة في السطر رقم 4 وعبارة end; بعد التعليمة في السطر رقم 7 وبالتالي يصبح البرنامج على الشكل الآتي:

```
Program Test22;  
var  
  i,j,k: integer;  
BEGIN  
  for i:=1 to 3 do  
    begin  
      writeln('x=',i+1);  
      for j:=1 to 2 do  
        writeln('i=', i, ', 'j=', j);  
      end;  
    readln;  
  END.
```

في هذه الحالة تكون نتائج البرنامج على الشكل الآتي:

```
x=2  
i=1 j=1  
i=1 j=2  
x=3  
i=2 j=1  
i=2 j=2  
x=4  
i=3 j=1  
i=3 j=2
```

4-4-6 الخروج من الحلقة:

من أجل الخروج من الحلقة تُستخدم التعليمة Exit. فعند ورود هذه التعليمة في جسم الحلقة سيتم إنهاء البرنامج.

مثال (24):

البرنامج التالي يبين كيفية الخروج من الحلقة:

```
Program Test23;  
uses Crt;  
procedure WasteTime;  
begin  
  repeat  
    if KeyPressed then Exit;  
    Write('Xx');  
  until False;  
end;  
BEGIN  
  WasteTime;  
END.
```

4-5 التعليمات الشرطية:

في العديد من المسائل تصادفنا حالات يرتبط حلها بشرط أو بعدة شروط ، وبحسب نتيجة هذا الشرط يمكن التفرع في البرنامج واختيار حالة ما من حالتين أو من عدة حالات ، مثل هذه الحالات يمكن تحقيقها في البرنامج من خلال استخدام التعليمات الشرطية:

1-5-4 تعليمة IF.....Then :

الشكل العام لهذه التعليمة هو كالتالي:

<عبارة> Then <تغيير منطقي> IF

والتعبير المنطقي يمثل هنا عملية شرط يمكن أن يكون ناتجه محقق أو غير محقق، وهذا التعبير يمكن أن يكون بسيطاً أو مركباً.

أما العبارة هنا فيمكن أن تكون إحدى تعليمات تروباסקال.

عندما تصل عملية التنفيذ إلى تعليمة If يتم اختبار التعبير المنطقي، فإذا كان الناتج محقق، فإن التنفيذ سينتقل إلى التعليمة الواقعة بعد Then أما إذا لم يكن ناتج التعبير المنطقي محقق، فإن التنفيذ سوف ينتقل إلى التعليمة الواقعة بعد تعليمة If مباشرة.

2-5-4 تعليمة If.....Then.....Else :

الشكل العام لهذه التعليمة هو كالتالي:

<عبارة> Else <عبارة> Then <تغيير منطقي> IF

في هذه الحالة إذا كان ناتج التعبير المنطقي محقق فإنه سيتم تنفيذ التعليمة الواقعة بعد Then، أما إذا كان ناتج التعبير المنطقي غير محقق فإنه سيتم تنفيذ التعليمة الواقعة بعد Else.

مثال (25) :

البرنامج التالي يبين كيفية استخدام التعليمات الشرطية:

Program Test24;**var****x: integer;****BEGIN****readln(x);****if $x > 0$ then writeln('x is positive');****if $x < 0$ then writeln('x is negative ') else writeln('x=0');****readln;****END.**

عند تنفيذ هذا البرنامج وإسناد قيمة موجبة لـ X ، فإن النتيجة ستكون كالتالي:

x is positive

x=0

أما إذا أسندت قيمة سالبة للمتحول X فإن النتيجة ستكون:

x is negative

وعند إسناد قيمة مساوية للصفر تكون النتيجة:

x=0

مثال (26):

في هذا المثال يتم مقارنة ثلاثة أعداد حسب قيمتها وإخراج نتائج المقارنة بما يتناسب وقيم هذه الأعداد. يمكن كتابة هذا البرنامج بطريقة مختصرة وأسهل، إلا أنني اعتمدت هذه الطريقة بهدف التعرف على الاستخدامات المتعددة للتعليمات المستخدمة في هذا البرنامج.

Program Test25;**uses crt;****label 1;****var****x,y,z: Real;**

```

a,b:Boolean;
BEGIN
  Clrscr;
  writeln;
  writeln('Compare Three Number: ');
  writeln;
  writeln('Please Enter The Data: ');
  write('1.Number='); readln(x);
  write('2.Number='); readln(y);
  write('3.Number='); readln(z);
  a:=x=y;  b:=y=z;
  if a and b then begin
    writeln('All three Numbers are equal');
    goto 1;
  end;
  if a then begin
    writeln('the first two Numbers are equal');
    if x<z then writeln (' the third Number is greater
                        then the first two Numbers. ')
    else writeln ('The third Number is smaller then
                  the first two Numbers. ');
    goto 1;
  end;
  If b then begin
    writeln('the last two Numbers are equal. ');
    if x<z then writeln ('The first Number is smaller then
                        last two Number. ')
    else writeln ('The first Number is greater then the
                  last two Numbers ');
    goto 1;
  End;
  if (x>y) and (y>z) then

```

```

begin
    writeln('The Order of the Number according
           the value are 3-2-1. ');
    goto 1;
end;
if (x<y) and (y<z) then
    begin
        writeln('The Order of the Number are 1-2-3. ');
        goto 1;
    end
else
    writeln('The entered Data are different. ');
1: readln;
END.

```

نلاحظ من خلال هذا البرنامج أنه يمكن استخدام أكثر من تعليمة بعد **Then** أو **Else** ولكن يجب في مثل هذه الحالات حصر هذه التعليمات بين **Begin** و **End**. هناك أيضاً شكل موسع لتعليمة **Then ... If**، حيث يمكن هنا استخدام أكثر من عبارة لـ **If** وذلك على الشكل الآتي:

If تعبير منطقي **Then**

تعليمة (مجموعة تعليمات)

Else if تعبير منطقي **Then**

تعليمة (مجموعة تعليمات)

Else if تعبير منطقي **Then**

تعلیمة (مجموعة تعلیمات)

Else

تعلیمة (مجموعة تعلیمات)

وكما هي الحال في الشكل البسيط السابق، يجب هنا أيضاً في حالة استخدام أكثر من تعلیمة (مجموعة تعلیمات) وضعها بين عبارتي **Begin** و **End**.

مثال (27):

البرنامج التالي يبين الاستخدام الموسع لتعلیمة **If....Then**:

Program Test26;

label 5;

var

i,x:integer;

y:real;

c:char;

BEGIN

5: write('Enter a Number x=');readln(x);

if (x>=0) and (x<=5) then

begin

y:=x*x;

writeln('x exist in the rang: [0...5]');

end

else if (x>=6) and (x<=10) then

begin

y:=sqrt(x);

writeln('y=',y);

writeln('x exist in the rang:[6...10]');

end

```

else
begin
    y:=x;
    writeln('x exist in the rang: [...-1] OR [11...] ');
end;
Readln;
write('Do you continue (y/n) ');
readln(c);
if (c='y') or (c='Y') then goto 5;
END.

```

وهنا لا بد من الإشارة إلى أنه يمكن باستخدام تعليمات الاختيار تحقيق عمليات انتقال (قفز) في البرنامج.

3-5-4 تعليمة IF...IN

الشكل العام لهذه التعليمة هو:

تعليمة **Then** [قائمة من الثوابت الرقمية أو الحرفية] **In** تعبير **If**

والتعبير هنا يُقصد به التعبير الحسابي ويمكن هنا أيضاً استخدام متحول، بدلاً من التعبير. أما القائمة من الثوابت الرقمية أو الحرفية، فيمكن أن تكون هنا عبارة عن قائمة تُحدد بدايتها ونهايتها وذلك كالاتي:

[B..E]

حيث تمثل B القيمة البدائية، و E القيمة النهائية، ويمكن أن تتضمن القائمة مجموعة من الثوابت الرقمية أو الحرفية تفصل بينها فواصل كالاتي:

[1,3,6,7,8]

مثال (28):

في البرنامج التالي يتم إدخال درجة الطالب وإخراج ترتيبه باستخدام تعليمة IF :

Program Test27;

var

m:integer;

BEGIN

write('The Mark of Student:');

readln(m);

if (m>=90) and (m<=100) then writeln('excellent');

if (m>=80) and (m<90) then writeln('Very good');

if (m>=70) and (m<80) then writeln('Good');

if (m>=60) and (m<70) then writeln('Accepted');

if (m<60) then writeln('bad');

readln;

END.

مثال (29):

يمكن كتابة البرنامج السابق باستخدام التعليمة In والحصول على النتيجة نفسها وذلك

كالآتي:

Program Test28;

var

m:integer;

BEGIN

write('The Mark of Student:');

readln(m);

if m In [90..100] then writeln('excellent');

if m In [80..90] then writeln('Very good');

if m In [70..80] then writeln('Very good');

if m In [60..70] then writeln('Good');

if m<60 then writeln('bad');

readln;

END.

4-5-4 تعليمة الاختيار :CASEOF.....END

لاحظنا أنه باستخدام التعليمة IF...THEN ... ELSE أمكن الاختيار بين حالتين (أي حالة واحدة من حالتين) . إلا أنه في حالات كثيرة تصادفنا نحتاج فيها للاختيار من عدة حالات (أكثر من حالتين) أي اختيار حالة واحدة فقط منها من عدة حالات. من أجل ذلك نستخدم في التربو باسكال التعليمة CASE...OF.....END. إن الشكل العام لهذه التعليمة هو كالآتي :

CASE OF تعبير

```
m1:Statement 1
m2:Statement 2
.....
.....
mn:Statement n
```

END

- إن معنى هذه التعليمة هو اختر حالة واحدة من الحالات التالية، وفي هذه الحالة يتم تنفيذ إحدى التعليمات الموجودة بين سطر CASE.....OF و سطر END. التعليمة التي يتم تنفيذها هي تلك التعليمة التي يتوافق رقمها مع قيمة التعبير بين CASE....OF حيث أن قيمة التعبير يجب أن تحدد مسبقاً، وعلى أساس ذلك فإنه :
- يجب أن تكون أرقام التعليمات من نوع التعبير المستخدم نفسه.
 - نوع التعبير المستخدم يجب أن يكون إما صحيح أو محرفي char أو منطقي Boolean ولا يجوز أن يكون Real
 - إن رقم التعليمات يمكن أن يكون ثابتاً أو مجالياً جزئياً أو رمزاً .

مثال (30):

المثال التالي يبين مقطع من برنامج لاستخدام التعليمات Case...Of:

CASE تعبير **OF**

'a'..'z':kb:=kb+1;

'A'..'Z':gb:=gb+1;

'0'..'9':z:=z+1;

.....:s:=s+1;

END

إذا كانت قيمة التعبير غير موجودة بين أرقام التعليمات فإنه لا يتم تنفيذ أي تعليمات من التعليمات بين Case ...of وبين end وإنما سيتم الانتقال إلى تنفيذ التعليمات الواقعة بعد end مباشرة.

مثال (31):

البرنامج التالي يبين كيفية استخدام التعليمات Case... Of:

Program Test29;

var

ch:char;

BEGIN

write('Please enter the character:');;

readln(ch);

Case ch of

'A'..'Z', 'a'..'z' : Writeln('Letter');

'0'..'9' :Writeln('Digit');

',' , '*' , '-' , '+' :Writeln('Operator');

End;

Readln;

END.

من المثالين السابقين نلاحظ أن التعليمات المستخدمة بسيطة (تعليمية واحدة) أما إذا كانت التعليمات ليست بسيطة أي إن التعليمية المعطاة رقم أو رمز، تتألف من أكثر من تعليمية فإنه يجب بهذه الحالة استخدام **Begin** و **End** لحصر هذه التعليمات تحت رقم أو رمز واحد (نقصد بـ **End** و **Begin** من أجل كل تعليمية على حدة (Statement)، وليست من أجل كل التعليمات مجتمعة) وكل التعليمات مجتمعة لا تحتاج هنا إلى **End**, **Begin**.

مثال (32):

البرنامج التالي يبين حساب الحجم والمساحة والمحيط لكرة:

```
program Test30;
uses crt;
label 4,5;
var
    v,o,m,r,h: real;
    w:char;
    select: byte;
const
    pi=3.1415926;
Begin
4: clrscr;
   writeln('please select between:');
   writeln;writeln(' 1. Calculate the volume:');
   writeln(' 2. Calculate the manteloberflaeche:');
   writeln(' 3. Calculate the totalsurface:');
   readln(select);
   writeln;write('High of the cylinder im meter :');readln(h);
   write (' radius of cylinder in meter :');readln(r);
```

Case select of

1: begin

```
writeln;writeln('Calculate of volume');  
V:= h*sqr(r)*pi;  
writeln('The value of volume is :',v:10:2, ' m3');
```

end;

2: begin

```
writeln;writeln('Calculate of Surface mantl');  
m:= h*2*pi*r;  
writeln('The value of Surface mantl is :', m:10:2, ' m2');
```

end;

3: begin

```
writeln;writeln('Calculate of total surface');  
o:= 2*pi*r*(r+h);  
writeln('The value of total surface is :', o:10:2, ' m2');
```

end

End;

```
writeln; writeln;  
write('Do you want to continue calculating (y/n)? ');  
readln(w);  
if (w='y') or (w='Y') then goto 4;
```

END.

ذكرنا سابقاً أنه إذا كانت قيمة التعبير المختارة غير موجودة بين أرقام أو رموز التعليمات بين Case ...of وبين end فإنه لا يتم تنفيذ أي تعليمة من هذه التعليمات، وإنما يتم الانتقال إلى التعليمة الموجودة بعد end مباشرة. لذلك لتجنب مثل هذه الحالة زودت تعليمة Case ...Of بعبارة Else.

مثال (33):

البرنامج التالي يبين استخدام التعليمة Case...Of مع Else :

Program Test31;

var

ch:char;

BEGIN

write('ch='); readln(ch);

case ch of

'A'..'Z': writeln('capital Letter');

'a'..'z': writeln('small Letter');

0'..'9': writeln('Digit');

else

writeln('Special Character');

end;

Readln;

END..

6-4 تعليمات الانتقال (القفز) :

يتم تنفيذ تعليمات البرنامج في لغة التروبواسكال على التسلسل وذلك حسب مكان ورودها في البرنامج، إلا أنه في كثير من الحالات وحسب المسألة المطروحة يحتاج المبرمج إلى تغيير اتجاه تسلسل تنفيذ تعليمات البرنامج، ويمكن أن يتم ذلك إما بالانتقال (القفز) إلى الخلف وبالتالي تكرار مجموعة من التعليمات مرات عديدة، أو بالانتقال (القفز) إلى الأمام وبالتالي تجاوز مجموعة من التعليمات. في كلا الحالتين التي يتم فيهما تغيير اتجاه مسار البرنامج نحتاج إلى تعليمات خاصة للقيام بذلك، هذه التعليمات يمكن أن تكون مرتبطة بشرط أو غير مرتبطة بشرط، والشرط في لغة التروبواسكال هو عبارة عن تعبير منطقي. وعلى أساس ذلك يمكن في لغة التروبواسكال التمييز بين تعليمات انتقال غير مرتبطة بشرط وتعليمات انتقال مرتبطة بشرط.

4-6-1 تعليمة الانتقال غير المشروط GOTO:

الشكل العام لهذه التعليمة هو كالتالي:

GOTO n

حيث n هنا عبارة عن علامة لأحد أسطر البرنامج (تعليمة)، المراد الانتقال إليه ويتم تعريفها مسبقاً في قسم التعريف في البرنامج باستخدام التعليمة Label. وكما نلاحظ هنا أن عملية الانتقال لا ترتبط بأي شرط، وأن البرنامج عندما يصل إلى هذه التعليمة سينقل عملية التنفيذ مباشرة إلى السطر (التعليمة) ذات الرقم n.

مثال (34):

Goto 5
Goto man

4-6-2 تعليمة الانتقال المشروط IF.....Then:

وهي نفس تعليمة الانتقال السابقة التي شكلها العام:

<عبارة> Then <تغيير منطقي> IF

ولكن يتم هنا استخدام تعليمة Goto بعد Then . يمكن أيضاً استخدام الشكل الآخر لتعليمة If لتنفيذ عمليات انتقال وهنا يجب أن تكون إحدى العبارات (<عبارة>) تعليمة انتقال goto أيضاً إذا تطلبت المسألة المطروحة ذلك:

<عبارة> Else <عبارة> Then <تغيير منطقي> IF

مثال (35):

البرنامج التالي يبين استخدام تعليمة الانتقال المشروط :If... Then

Program Test32;

label 2;

var

i,x: integer;

BEGIN

x:=0;

2: x:=x+i;

i:=i+1;

if i<=5 then goto 2;

writeln('x=',x);

readln;

END.

في هذا المثال يتم تنفيذ عملية انتقال إلى الخلف عدة مرات حيث يتم هنا تشكيل حلقة يتم فيها حساب المجموع للأعداد من 1 وحتى 5 .

4-7 المعطيات المركبة :

إن أنواع المعطيات التي درسناها حتى الآن هي عبارة عن معطيات بسيطة، هذا يعني أن المتحولات في هذه المعطيات تخزن حد واحد فقط من المعطيات، باستثناء نوع المعطيات String الذي يخزن سلسلة من الرموز.

إلا أن لغة التروبايسكال تدعم نوع آخر من المعطيات والتي تسمى بالمعطيات المركبة والتي يمكن من خلالها تخزين حدود متعددة ضمن متحول واحد. من هذه المعطيات Array, Record, Set

4-7-1 المنظمة ARRAY (المصفوفات) :

يتألف هذا النوع من المعطيات من مجموعة من العناصر وهذه العناصر تعود إلى النوع من المعطيات نفسه.

لاحظنا عند الحديث عن المتحولات المستخدمة في التريباسكال وعن شروط اختيار المتحولات أن عملية اختيار المتحولات بأعداد كبيرة ليست سهلة حيث أن الرموز المتوفرة قد لا تكفي لتشكيل عدد كبير من المتحولات قد نحتاجه في هذا البرنامج، إضافة إلى أنه نحتاج في كثير من الحالات إلى إجراء الحسابات على متحولات ذات دلائل (دليل أو أكثر) .

هذه المشاكل حلت في لغة الباسكال باستخدام تعليمة المصفوفات (ARRAY) حيث يمكن باستخدام هذه التعليمة حجز عدد كبير من الأماكن في الذاكرة تحت اسم متحول واحد. الشكل العام لهذه التعليمة هو :

V: ARRAY (A..M) OF TYPE

V: المتحول الذي نريد الحجز له باستخدام ARRAY

الثابت A : يمثل الدليل الأدنى .

الثابت M: يمثل الدليل الأعلى.

TYPE: تمثل نوع المعطيات (CHAR , INTEGER , REAL)

وهنا يجب أن يكون $A < M$.

هذا الشكل يمثل حالة بعد واحد حيث تم هنا حجز عدة أماكن من A إلى N للمتحول V في الذاكرة.

مثال (36):

X:Array (1..5) of Integer;

في هذا المثال يتم حجز خمسة أماكن للمتحول X في الذاكرة وهي:
X[1], x[2], x[3], x[4], x[5]

في حالة استخدام بعددين فإن هذه التعليمة تصبح بالشكل :
V : ARRAY (A..M,B..N)OF TYPE;

وهنا يجب أن يكون $A < M, B < N$.

مثال (37):

Fn2 : Array (1..2,1..3) Of Integer;

في هذه الحالة يتم حجز الأماكن التالية تحت اسم المتحول (FN2)
FN2[1,1],FN2[1,2],FN2[1,3]
FN2[2,1],FN2[2,2],FN2[2,3]

وهنا يجب الإشارة إلى القواعد التالية:

- يجب التقيد بوضع النقطتين بين الحد الأدنى والأعلى من دون أية فواصل.
- يجب وضع الفاصلة بين الأبعاد (الدلائل) في حالة عندما نتعامل مع أكثر من بعد.
- لا يجوز أن يكون الدليل متحولاً أو تعبيراً (يجوز أن يكون ثابتاً معروفاً مسبقاً تحت اسم Const) .
- الدليل الأدنى يجب أن يكون دائماً أقل من الدليل الأعلى.

ملاحظة: عند استخدام Array بالنسبة للمتحول Char يجوز استخدام حقل بدليل واحد فقط.

مثال (38):

في هذا البرنامج يتم حساب الوسطي للدرجات الإمتحانية لطالب:

Program Test33;

```
var
i: integer;
s, Average: Real;
Mark:array[1..10] of Real;
BEGIN
s:=0;
for i:=1 to 10 do
  begin
    writeln(' ',i, ' Student ');
    write ('Mark=');readln(Mark[I]);
    s:=s+Mark[I];
  end;
  average:=s/i;
  writeln(' Average= ',average:4:2);
END.
```

مثال(39):

في البرنامج التالي يتم تنفيذ عملية ثني (Convolution) بين إشارتين:

Program Test34;

```
var
x,y,z:array[1..5] of integer;
i,j:integer;
BEGIN
  x[1]:=2;x[2]:=1;x[3]:=1;
  y[1]:=1;y[2]:=2;y[3]:=3;
  for i:=1 to 5 do
    begin
```

```

z[i]:=0;
for j:=1 to i do
  begin
    z[i]:=z[i]+x[i-j+1]*y[j];
  end;
writeln(z[i]);
end;
writeln('-----');
readln;
END.

```

مثال (40):

البرنامج الآتي يبين شكل آخر لتنفيذ عملية ثني (Convolution) بين إشارتين:

Program Test35;

uses crt;

var

x,y,f: array [1..10] of real;

i,j,n1,n2:integer;

BEGIN

textColor(4);

write('how many Samples in the first Signal:'); readln(n1);

write('how many Samples in the second Signal:'); readln(n2);

TextColor(3);

write('please enter the value of the first Signal');

for i:=1 to n1 do

read(x[i]);

write ('please enter the value of the Second Signal');

for j:=1 to n2 do

read (f[j]);

for i:=1 to 5 do

begin

y[i]:=0;

```

        for j:=1 to i do
            begin
                y[i]:=y[i]+x[j]*f[i-j+1];
            end;
        end;
    TextColor(2);
    writeln('-----');
    for i:=1 to 5 do
        writeln(y[I]);
    writeln('-----');
readln;
END.

```

مثال (41):

البرنامج التالي يُستخدم لضرب مصفوفتين:

```

Program Test36;
Uses crt;
Var
    i,j:integer;
    x,a,b:array [1..3,1..3] of integer;
BEGIN
    Write ('Enter The Elements of the first Matrix (a):');
    For i:= 1 to 3 do
        begin
            For j:=1 to 3 do
                Read (a[i,j]);
            end;
        Write ('Enter The Elements of the second Matrix (b):');
        For i:= 1 to 3 do
            begin
                For j:=1 to 3 do
                    Read(b[i,j]);

```

```

end;
for i:=1 to 3 do
begin
  for j:= 1 to 3 do
    x[i,j]:=x[i,j] + ( (a[i,j]) * (b[i,j]) ) ;
  end;
for i:= 1 to 3 do
begin
  for j:= 1 to 3 do
    writeln(x[i,j]);
  end;
end;
readln;
END.

```

4-7-2 نوع المعطيات التعدادي (المجالي) : Enumerated Types

عرضنا سابقاً عند الحديث عن أنماط المعطيات في التروباسكال عن نوع المعطيات التعدادي (المجالي)، وسنعرض الآن كيفية استخدام هذا النوع من المعطيات في البرنامج.

يتم في البرنامج التصريح عن هذا النوع من المعطيات باستخدام العبارة Type ، وهنا يتم تعريف مجموعة من القيم في البرنامج تحت أسم متغير ما، ويمكن هنا التعامل مع هذا المتغير في البرنامج وكأنه نوع معطيات جديد يسري عليه ما يسري على أنواع المعطيات الأخرى ما عدا بعض التحديدات، التي سنذكرها لاحقاً.

مثال (42):

يمكن تعريف مجموعة أو نمط من المعطيات باسم Name بحيث تتضمن هذه المجموعة عدة أسماء يتم تحديدها من قبل المستخدم (كمثال Ahmed, Lana, samir, sara,

Riad). وهذه المجموعة يمكن التعامل معها في البرنامج كأي نمط من أنماط المعطيات المعروفة:

```
Type
Name= (Ahmed, Lana, Samir, Sara, Riad);
Var
N: name;
```

مثال (43):

يوضح كيفية استخدام النمط Type:

```
Type
Day =(Monday,Tuesday,Wednesday,Thursday, Friday,
Saturday, Sunday);
WorkingDay = Monday..Friday;
Var
Today, Yesterday, Tomorrow, Holiday :Day;
DayOff :WorkingDay;
```

مثال (44):

يتم في البرنامج التالي التعرف إلى أيام العطل في الأسبوع:

```
Program Test37;
type
day=(sat,sun,Mon,tue,wed,thu,fri);
var
weekend:day;
BEGIN
if (weekend=sat) or (weekend=fri) then
writeln ('This day is free day')
else
```

```
writeln ('this day is Workday');  
readln;  
END.
```

3-7-4 بعض الإجراءات التي تتعامل مع النمط **Type**:

هناك بعض الإجراءات التي تتعامل مع النمط التعدادي للمعطيات، من هذه الإجراءات **Ord, Char, Succ, Pred**، ولهذه الإجراءات استخدامات أخرى سنتعرف عليها لاحقاً.

مثال (45):

البرنامج التالي يوضح استخدامين للإجراء **Ord**، فهو يُستخدم لتحديد موقع الحرف في جدول الأسكي، وكذلك يُستخدم لتحديد موقع الجملة المعرفة في العبارة **Type**:

```
Program Test38;  
uses Crt;  
Type  
Colors = (RED,BLUE,GREEN) ;  
BEGIN  
Writeln('BLUE has an ordinal value of,'  
Ord(BLUE)) ;  
Writeln('The ASCII code for "c" is,'  
Ord('c'), ' decimal');readln;  
END.
```

مثال (46):

البرنامج التالي يبين استخدام الإجرائين **Succ** و **Pred**، فالإجراء **Pred** يحدد القيمة السابقة، بينما يعطي الإجراء **Succ** القيمة التالية:

```

Program Test39;
uses crt;
type
  Colors = (RED,BLUE,GREEN) ;
BEGIN
  Writeln('The predecessor of 5 is',Pred(5)) ;
  Writeln('The successor of 10 is', Succ(10)) ;
    if Succ(RED) = BLUE then
      Writeln('In the type Colors, RED Is',
        'the predecessor of BLUE');
  readln;
END.

```

4-7-4 نوع المعطيات سجل (Record):

ذكرنا عند الحديث عن المصفوفات بأنها عبارة عن نوع من المعطيات المركبة التي يمكن أن تخزن عناصر متعددة ولكن يجب أن تكون هذه المعطيات من نفس النوع. إلا أنه في كثير من الحالات نحتاج إلى استخدام متحول يمكن أن يخزن عدد من العناصر ولكن هذه العناصر تعود إلى أنواع مختلفة من المعطيات، من أجل ذلك يمكن استخدام السجلات Record.

إذا السجلات Record تشبه المصفوفات Array بأن كل منها يتضمن مجموعة من المتحولات، ما عد أن المصفوفات يجب أن تكون متحولاتها من نفس النوع، اما السجلات فإنه يمكن أن تكون متحولاتها من أنواع مختلفة. فالسجلات هو نوع من أنواع المعطيات المعرفة في البرنامج وهي تنطوي تحت نوع المعطيات التعدادية Type ، وغالباً ما يتم تعريف الـ Type قبل الإعلان عنها.

المتحولات المختلفة الأنواع المتضمنة ضمن السجل تسمى حقول Fields ويجب أن تكون موجودة ضمن مجموعة واحدة محصورة بين عبارة Record و عبارة End; .

مثال (47):

لنلاحظ السجل التالي الذي يُستخدم من أجل تخزين معلومات تتعلق بالموظفين في شركة:

Employee Record		
Field #	Information	Possible Data Type
1.	Name	STRING
2.	Address	STRING
3.	Phone number	STRING/INTEGER
4.	Hourly rate	REAL
5.	Marital status	CHAR/Enumeration

نلاحظ أن السجل يحتوي على عدة حقول بأنواع مختلفة.
هذا السجل يمكن كتابته بلغة التروباسكال على الشكل التالي:

Type
EmployeeRecord = Record
 Name :STRING[25];
 Address :STRING[40];
 Phone :STRING[12];
 MaritalStatus :CHAR;
End;

مثال (48):

المثال التالي يبين طريقة استخدام السجل في البرنامج:

Type
Point = Record
 X, Y: Real;
End;

```

Vector = array[0..1] of Point;
Month = (Jan, Feb, Mar, Apr, May, Jun, Jly, Aug, Sep, Oct,
Nov, Dec);
Date = Record
      D: 1..31;
      M: Month;
      Y: 1900..1999;
End;

```

■ استخدام السجل في البرنامج:

من أجل استخدام السجل في البرنامج لا بد من توفر متحول يحمل اسم هذا السجل،
ففي مثالنا السابق حالة الموظفين في شركة، يمكن
استخدام السجل في قائمة المتحولات في البرنامج كآتي:

```

Var
EmployeeRec :EmployeeRecord;

```

من أجل استخدام هذا المتحول EmployeeRec يوجد طريقتين:

- الطريقة الأولى:

ربط المتحول بالحقول المعرفة في السجل، وبالتالي يمكن بعدها استخدام هذا المتحول في
جسم البرنامج بتنفيذ عمليات إسناد أو عمليات إدخال و إخراج، كمثال على ذلك:

```

EmployeeRec.Name := 'Ahmed';
WRITELN('Employee Name: ', EmployeeRec.Name);

```

المتحولات المركبة يُسمى بالمتحولات الحقلية Fielded Variables

مثال (49):

البرنامج التالي يبين كيفية استخدام السجل:

Program Test40;

type

EmployeeRecord = **Record**

Name :**STRING**[25];

Address :**STRING**[40];

Phone :**STRING**[12];

MaritalStatus :**CHAR**;

End;

Var

EmployeeRec :**EmployeeRecord**;

BEGIN

{ Assign values to the fielded variables }

EmployeeRec.Name := 'Riad Taifour';

EmployeeRec.Address := 'Syria, Damascus Univ. ,
Department of Geology';

EmployeeRec.Phone := '00963-11-2129856';

EmployeeRec.MaritalStatus := 'S';

{ Display record information }

Writeln('Employee Name: ', *EmployeeRec.Name*);

Writeln ('Address: ', *EmployeeRec.Address*);

Writeln ('Telephone : ', *EmployeeRec.Phone*);

Writeln ('Marital Status: ', *EmployeeRec.MaritalStatus*);

END.

بعد تنفيذ هذا البرنامج فإن النتيجة ستكون على الشكل الآتي:

Employee Name: Riad Taifour

Address: Syria, Damascus Univ. , Department of Geology

Telephone: 00963-11-2129856

Marital Status: S

يتم في هذه الطريقة استخدام التعليمة With ، وهنا لا داعي لاستخدام المتحولات الحقلية، وإنما يمكن استخدام أسماء الحقول مباشرة.

مثال (50):

مقطع من برنامج يبين استخدام التعليمة With:

```
With EmployeeRec do  
Begin  
    Name := 'Riad Taifour';  
    Address := 'Syria-Damascus Univ.-Department of  
                Geology';  
    Phone := '00963-11-2129856';  
    MaritalStatus := 'M';  
End;
```

مثال (51):

البرنامج التالي يبين كيفية استخدام التعليمة With:

```
Program Test41;  
Type  
EmployeeRecord = Record  
    Name :STRING[25];  
    Address :STRING[40];  
    Phone :STRING[12];  
    MaritalStatus :CHAR;  
End;  
  
Var  
EmployeeRec :EmployeeRecord;  
{ ----- Procedure Drawline ----- }  
Procedure DrawLine(LineLength,TabLength: Integer);  
Const  
    Dash = '-';
```

```

Var
Counter :Integer;
begin
  For Counter := 1 To TabLength do
    Write ( ' ');
    For Counter := 1 To LineLength do
      Write(Dash);
    Writeln
  end;
  { ----- Procedure GetData ----- }
  Procedure GetData(Var Employee :EmployeeRecord);
  { Assign values to fields }
  begin
    With Employee do
      begin
        Name := 'Riad Taifour';
        Address := 'Syria-Damascus Univ.- Department of
                    Geology';
        Phone := '00963-11-2119856';
        MaritalStatus := 'M'
      end
    end;
  { ----- Procedure DisplayInfo ----- }
  Procedure DisplayInfo(Employee :EmployeeRecord);
  { Display record information }
  Const
    Header ='Record of ';
  Var
    Len, Tab, Counter :Integer;
    HeaderText, Status :String;
  begin
    With Employee do

```



```

begin
  HeaderText := Concat(Header, Name);
  Len := Length(HeaderText);
  Tab := (80- Len) div 2;
  DrawLine(Len, Tab);
  For Counter := 1 To Tab do
    Write ( ' ');
  Writeln(HeaderText);
  DrawLine(Len, Tab);
  Writeln('Address: ', Address);
  Writeln ('Telephone : ', Phone);
  IF MaritalStatus = 'M' Then
    Status := 'Married'
  Else
    Status := 'Single';
  Writeln ('Marital Status: ', Status)
end
end;
{ -----Main Program ----- }
begin
  GetData(EmployeeRec);
  DisplayInfo(EmployeeRec);
END.

```



الفصل الخامس الإجراءات والتتابع

1-5 مقدمة

2-5 الإجراء (Procedure)

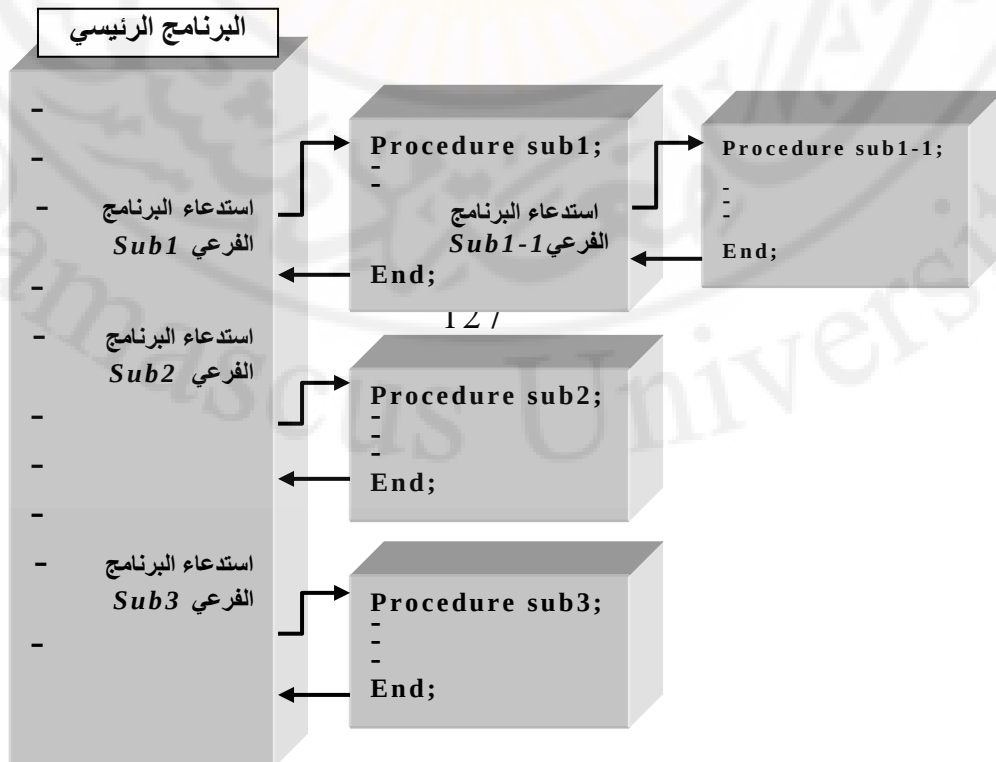
3-5 التابع (Function)

4-5 الإجراءات والدوال مسبقة التعريف

الإجراءات والتوابع

1-5 مقدمة:

تعرفنا في الفصول السابقة إلى العناصر المختلفة المكونة لبرنامج التروباسكال مثل الثوابت والمتحولات والإجراءات المعرفة مسبقاً في لغة التروباسكال، وسندرس في هذا الفصل نوعين آخر من العناصر المكونة لبرنامج التروباسكال وهي الإجراءات والتتابع. في كثير من الحالات يتطلب الحل في البرنامج إعادة تكرار جزء ما أو أية عملية ما مرات عديدة في البرنامج الشيء الذي يتطلب تكرار كتابة هذه العملية أو تعليمات هذا الجزء عدة مرات يساوي إلى عدد مرات التكرار . في مثل هذه الحالات يمكن فصل هذا الجزء المراد استخدامه عدة مرات في البرنامج في برنامج منفصل نسميه بالإجراء أو التابع أو يُسمى أيضاً بالبرنامج الجزئي أو الفرعي حيث يتم كتابته لمرة واحدة فقط واستدعائه مرات عديدة وذلك عندما نحتاجه في البرنامج الرئيسي أو أي برنامج فرعي آخر.



الشكل (5-1): العلاقة بين البرنامج الرئيسي والبرنامج الجزئي

لقد صممت الإجراءات من أجل إنجاز بعض المهام الجزئية، مثل مسح الشاشة ، أو ترتيب بعض القيم،، ولكن في كثير من الحالات نحتاجها أيضاً في تنفيذ أعمالٍ بحجم أكبر، يمكن إجراء عملية تغيير عليها في كل مرة يتم استدعاؤها، لذلك يجب على الإجراءات أن تكون قادرة على التعامل مع هذه التغيرات حتى تتمكن من إنجاز أعمالها بالشكل المناسب.

يمكن التمييز بين نوعين من البرامج الجزئية:

1- الإجراءات PROCEDURE.

2- التابع FUNCTION:

وفي كلا النوعين يجب تعريفها قبل استخدامها وذلك في قسم التعريف الذي كنا قد درسناه سابقاً حيث يتم تعريف البرنامج الجزئي في نهاية قسم التعريف كما سنلاحظ فيما بعد .

5-2 الإجراء PROCEDURE:

وهو عبارة عن برنامج مكون من مجموعة من التعليمات لإنجاز عمل جزئي معين قد نحتاج إلى تكراره في البرنامج الرئيسي أكثر من مرة، مثل مسح الشاشة مثلاً، أو إصدار صوت محدد، أو ترتيب مجموعة من الأعداد،.....، وفي كل مرة نحتاج فيها إلى تنفيذ هذا الإجراء قد نحتاج إلى إجراء تغييرات في المتغيرات المستخدمة في هذا الإجراء بحيث تمكن من الوصول إلى النتائج المرغوبة. حتى تتمكن الإجراءات من التعامل مع المتغيرات المستخدمة يجب أن تكون الإجراءات قادرة على الاتصال مع البرنامج الرئيسي أو بشكل عام البرنامج المستدعي (البرنامج الذي تم استدعاء الإجراء من خلاله).

5-2-1 تعريف الإجراء واستدعائه:

قبل استخدام الإجراء (استدعائه) يجب أولاً تعريفه في البرنامج، ولكن قبل الحديث عن تعريف الإجراء في البرنامج يجب أولاً التعرف إلى أجزاء الإجراء. الإجراء كما ذكرنا سابقاً هو عبارة عن برنامج صغير وهو يشبه في بناءه برنامج التروباسكال الرئيسي فهو يتكون من:

العنوان: يتم فيه كتابة اسم الإجراء وعلى خلاف البرنامج الرئيسي يُسبق اسم الإجراء هنا بكلمة procedure.

قسم التعريف: ويتم فيه تعريف الثوابت والمتحولات ويمكن تعريف الإجراءات والتتابع.

جسم البرنامج: وهو يتضمن مجموعة من التعليمات والعبارات وهو يبدأ بعبارة Begin وينتهي بعبارة end.

يتم تعريف الإجراء في بداية البرنامج الرئيسي في قسم التعريف، أما استدعائه فيتم من خلال استخدام اسم الإجراء في المكان المراد استخدامه به، ويمكن أن يتم استخدام اسم الإجراء مع متحولات أو من دون متحولات حسب طريقة تعريف الإجراء.

مثال (1):

البرنامج التالي يبين مثلاً بسيطاً لاستخدام البرنامج الجزئي، حيث تم تعريف البرنامج الجزئي **pause** في قسم التعريف في البرنامج، وبعد ذلك تم استدعاء البرنامج الجزئي عدة مرات في البرنامج الرئيسي.

```
Program Test42;  
uses crt;  
Procedure pause;  
begin  
    Writeln ('press any key to continue ');  
    Readln;  
end;  
{--- Main Program ---}  
BEGIN  
    Writeln ('my name is Riad Taifour ');  
    Pause;  
  
    Writeln ('I like Pascal Language ');  
    Pause;  
    Writeln ('good luck !! ');  
    Pause;  
    Writeln ('Programing Language is very important for  
            Geophysics ');  
    Pause;  
END.
```

5-2-2 المتحولات العامة والمتحولات الموضعية:

ذكرنا سابقاً أن الاتصال بين الإجراء وبين البرنامج الذي يتم استدعاء الإجراء منه يتم باستخدام المتحولات. والمتحولات المستخدمة في هذه الحالة يمكن أن تكون متحولات عامة (Global Variable) ومتحولات موضعية (Local Variable).

المتحولات العامة يتم التصريح عنها في البرنامج الرئيسي، ويمكن استخدامها في أي مكان في البرنامج، أما المتحولات الموضعية فهي تُعرف ضمن الإجراء وتسري فقط ضمن الإجراء المعرفة فيه ولا يجوز استخدامها خارج الإجراء وفي حال استخدامها فإنه سنحصل على رسالة خطأ تدل على أنه لا يمكن استخدام هذه المتحولات ضمن هذا الجزء من البرنامج وذلك لأن البيئة التي تتبع لها هذه المتحولات (الإجراء) غير فعالة.

مثال (2):

البرنامج التالي يبين كيفية استخدام المتحولات المحلية:

```
Program Test43;  
Procedure local;  
var  
  x,y:integer;  
begin  
  Readln(x);  
  y:=x*2;  
  writeln('y=',y);;  
end;  
{--- Main Program ---}  
BEGIN  
  local;  
  readln;  
END.
```

من البرنامج نلاحظ أن المتحولين x, y معرفين فقط في البرنامج الجزئي Local وهما يسريان فقط في ضمن البرنامج الجزئي. يمكن ملاحظة ذلك من خلال إجراء تعديل على البرنامج السابق كالآتي :

مثال (3):

```
Program Test44;  
var  
    g: Integer;  
Procedure local;  
var  
    x,y:integer;  
begin  
    Readln(x);  
    y:=x*2;  
    writeln('y=',y);;  
end;  
{--- Main Program ---}  
BEGIN  
    local;  
    g:=x/2;  
    writeln('g=',g);  
    readln;  
END.
```

التعديل الأول الذي أجري على هذا البرنامج تمثل بإضافة المتحول g إلى البرنامج الرئيسي وهو هنا يُعد متحولاً عاماً وبالتالي يسري لكل من البرنامج الرئيسي والبرنامج الجزئي. التعديل الثاني تمثل في جسم البرنامج الرئيسي وذلك بإضافة العبارة $g:=x/2$ ، لاحظ هنا أن المتحول x هو متحول محلي ويسري فقط بالنسبة للبرنامج الجزئي.

عند تنفيذ البرنامج نلاحظ أنه ستظهر رسالة الخطأ التالية: "unknown identifier", مما يدل على أن المتحول x غير معروف في ساحة البرنامج الرئيسي.

مثال (4):

البرنامج التالي يبين طريقة تعريف الإجراء وكيفية استدعائه.

```
Program Test45;  
var  
    a,b,x:real;  
Procedure multi;  
begin  
    x:=x*2;  
end;  
{--- Main Program ---}  
BEGIN  
    write('Enter 1. Number:'); readln(a);  
    x:=a;  
    multi;  
    a:=x;  
    writeln('Result=',a);  
    write('Enter 2. Number:'); readln(b);  
    x:=b;  
    multi;  
    writeln('Result=',x);  
    readln;  
END.
```

نلاحظ من البرنامج كيف يتم تعريف الإجراء وكيف تجري طريقة استدعائه. من خلال هذا البرنامج نلاحظ أن المتحولات a, b, x هي متحولات عامة تسري لكامل البرنامج أي أنها تُستخدم في البرنامج الرئيسي والجزئي (الإجراء). بعد تنفيذ البرنامج السابق وبعد إسناد القيمة 2 للمتحول a ، والقيمة 3 للمتحول b . فإن النتيجة تكون على الشكل التالي:

```
Enter 1. Number:2
Result= 4.0000000000E+00
Enter 2. Number:3
Result= 6.0000000000E+00
```

لا بد هنا من الإشارة إلى أنه في حالة تعريف المتحول في البرنامج الرئيسي وفي البرنامج الجزئي، فإن المتحول العام ستكون له الغلبة أو الأولوية في البرنامج الرئيسي.

مثال (5):

لنجرّ تعديلاً على البرنامج السابق بحيث نضيف في البرنامج الجزئي تعريفاً للمتحول x فيصبح البرنامج على الشكل التالي:

```
Program Test46;
var
  a,b,x:real;
procedure multi;
var
  x: real;
begin
  x:=x*2;
end;
{--- Main Program ---}
```

BEGIN

```
write('Enter 1. Number:'); readln(a);
```

```
x:=a;
```

```
multi;
```

```
a:=x;
```

```
writeln('Result=',a);
```

```
write('Enter 2. Number:'); readln(b);
```

```
x:=b;
```

```
multi;
```

```
writeln('Result=',x);
```

```
readln;
```

END.

إن ناتج تنفيذ هذا البرنامج بعد إسناد القيم نفسها لكل من a و b ، أي a=2 و b=3 يكون على الشكل التالي:

```
Enter 1. Number:2
Result= 2.0000000000E+00
Enter 2. Number:3
Result= 3.0000000000E+00
```

3-2-5 استخدام الوسطاء:

لاحظنا في البرنامج السابقة أن تمرير المتحولات من البرنامج المستدعي إلى الإجراء جرت من خلال إسناد قيمة إلى المتحول المستخدم في الإجراء وذلك قبل استدعائه. إلا أنه في كثير من الحالات نحتاج إلى تعريف المتحولات ضمن الإجراء وفي هذه الحالة يمكن أن تجري عملية تمرير المتحولات من خلال استخدام ما يسمى بالوسطاء. هذه الوسطاء يمكن أن تمرر المتحولات بحالتين:

- تمرير باتجاه واحد، أي تمرير من البرنامج المستدعي إلى الإجراء.
- تمرير باتجاهين، أي تمرير من البرنامج المستدعي إلى الإجراء وبالعكس.

مثال (6):

في البرنامج التالي يتم تمرير المتحولات باتجاه واحد، أي من البرنامج المستدعي إلى الإجراء:

```
Program Test47;
var
  a,b:real;
procedure multi( x:real);
begin
  x:=x*2;
  writeln('result-Sub=',x);
end;
{--- Main Program ---}
BEGIN
  write('Enter 1. Number:');readln(a);
  multi(a);writeln('Result1-Main=',a);
  write('Enter 2. Number:');readln(b);
  multi(b);writeln('Result2-Main=',b);
  readln;
END.
```

في هذا البرنامج تم استخدام الإجراء multi واستخدم هنا المتحول x كما هو مبين في البرنامج. بهذه الطريقة يمكن أن يتم تمرير المتحول باتجاه واحد فقط، أي من البرنامج

المستدعي إلى الإجراء. بعد تنفيذ هذا البرنامج وإسناد القيمة 2 إلى المتحول a والقيمة 3 إلى المتحول b فإن النتيجة ستكون على الشكل الآتي:

```
Enter 1. Number:2
result-Sub= 4.0000000000E+00
Result1-Main= 2.0000000000E+00
Enter 2. Number:3
result-Sub= 6.0000000000E+00
Result2-Main= 3.0000000000E+00
```

مثال (7):

في البرنامج التالي يتم تمرير المتحولات باتجاهين، أي من البرنامج المستدعي إلى الإجراء وبالعكس:

```
Program Test48;
var
  a,b:real;
procedure multi(var x:real);
begin
  x:=x*2;
  writeln('result-Sub=',x);
end;
{--- Main Program ---}
BEGIN
  write('Enter 1. Number:');readln(a);
  multi(a);writeln('Result1-Main=',a);
  write('Enter 2. Number:');readln(b);
  multi(b);writeln('Result2-Main=',b);
  readln;
END.
```

من متابعة البرنامج نلاحظ أن الفرق بين البرنامج السابق وهذا البرنامج هو فقط عبارة Var التي استخدمت في تعريف الإجراء multi. في هذه الحالة يمكن تمرير الوسيط باتجاهين. تنفيذ البرنامج باستخدام نفس المعاملات السابقة أي إسناد القيمة 2 إلى المتحول a والقيمة 3 إلى المتحول b فإن النتيجة ستكون على الشكل التالي:

```
Enter 1. Number:2
result-Sub= 4.0000000000E+00
Result1-Main= 4.0000000000E+00
Enter 2. Number:3
result-Sub= 6.0000000000E+00
Result2-Main= 6.0000000000E+00
```

ملاحظة:

عند استخدام العبارة VAR مع اسم البرنامج الجزئي (الإجراء proc7)، فإن المتحولات المستخدمة تكون متحولات إدخال وإخراج أي أن الإدخال والإخراج يتم في البرنامج الرئيسي، بينما إذا لم تستخدم العبارة VAR فإن الإدخال يتم في البرنامج الرئيسي أما الإخراج فيتم في البرنامج الفرعي (الإجراء proc6).

مثال (8):

يقوم البرنامج التالي بترتيب مجموعة من القيم العددية تصاعدياً، ويستخدم من أجل عدة برامج جزئية:

```
Program Test49 ;  
Const  
ArraySize = 6;  
Type
```



```

Range = 1..ArraySize;
NumbersArray = Array[Range] OF Integer;
Var
Numbers :NumbersArray;
{ ----- Read procedure ----- }
Procedure ReadNumbers(L: Integer; Var R
:NumbersArray);
Var
I :Integer;
begin
For I := 1 To L do
begin
Write ('Enter element #', I,': ');
Readln(R[I])
end
end;
{ ----- Sort procedure ----- }
Procedure SortNumbers(M: Integer; VAR S
:NumbersArray);
Var
I, J, Pot :Integer;
begin
For I := 1 To M-1 do
For J := I+1 To M do
If S[I] > S[J] Then
begin { Swap contents }
Pot := S[J];
S[J] := S[I];
S[I] := Pot
end
end;
{ ----- Print procedure ----- }

```

```

Procedure PrintNumbers(N: Integer; T :NumbersArray);
Var
I :integer;
begin
    Writeln;
    Write('The sorted array is: ');
    For I := 1 To N do
        Write (T[I], ' ');
        Writeln;
    end;
{ ----- Main Program ----- }
BEGIN
    ReadNumbers(ArraySize, Numbers);
    SortNumbers(ArraySize, Numbers);
    PrintNumbers(ArraySize, Numbers);
    Writeln ('Press ENTER to continue..');
    Readln
END.

```

4-2-5 الاستدعاء الذاتي للبرنامج الجزئي:

لاحظنا أن البرنامج الجزئي يتم استدعائه من قبل البرنامج الرئيسي أو من قبل برنامج جزئي آخر، إلا أننا سنلاحظ بأنه يمكن أيضاً للبرنامج الجزئي أن يستدعي نفسه، وفي هذه الحالة يجب أن يُربط الاستدعاء بشروط وإلا سندخل في حلقة لا نهائية.

مثال (9):

البرنامج الآتي يبين كيفية الاستدعاء الذاتي للبرنامج الجزئي:

```

Program Test50;
var

```

```

a:integer;
procedure self;
begin
  readln(a);
  if a<10 then self;
end;
{----- Main Program -----}
BEGIN
  writeln('Please Enter a Number:');
  self;
END.

```

3-5 التابع Function:

لاحظنا أن الإجراء يستخدم من أجل القيام بعمل ما، وهذه الإجراءات يمكن أن تعيد قيمة إلى البرنامج المستدعي أو لا تعيد، أما التوابع فإنها تستخدم من أجل حساب قيمة ما، وهذه التوابع يجب أن تعيد قيمة إلى البرنامج المستدعي.

إن بناء التابع يشبه بناء الإجراء ما عدا أن قسم العنوان يبدأ باستخدام العبارة Function متبوعة باسم التابع ويمكن أن يلحق اسم التابع بمتحولات وسيطية.

إن مفهوم التابع هو نفسه مفهوم التابع في الرياضيات فعندما نريد حساب قيمة التابع: $y = x^2$ من أجل قيمة ما لـ x ، حيث x هنا متحول يتم تمريره، فإن هذا التابع سيعيد قيمة ما حسب قيمة المتحول الممرر. وفي لغة التروباسكال التابع يعيد دائماً قيمة محددة.

مثال (10):

البرنامج التالي يبين طريقة استخدام التابع وكيفية استدعائه:

```

Program Test51;
var

```

```

x,a,b,c,d:real;
function multi:real;
begin
    multi:=x*2;
end;
{--- Main Program ---}
BEGIN
    write('Enter 1. Number:');readln(x);
    c:=multi ; writeln('Result=',c);
    write('Enter 2. Number:');readln(x);
    d:=multi; writeln('Result=',d);readln;
END.

```

في البرنامج نلاحظ كيف يتم تعريف التابع وكذلك طريقة استدعائه. تم في هذا البرنامج استخدام التعبير الحسابي الذي استخدم في الأمثلة السابقة عند مناقشة الإجراءات، وبما أن هذه الحالة بسيطة لذلك لا نجد خلافاً بالنتائج.

يمكن هنا أيضاً استخدام متحولات وسيطة كما هي الحال بالإجراء، إلا أنه في حالة التابع قد لا نحتاج إلى مثل هذه الحالات وذلك لأن التابع يعيد قيمة مباشرة.

مثال (11):

البرنامج التالي يبين حالة استخدام التابع مع متحولات وسيطة:

```

Program Test52;
var
    a,b,c,d:real;
function multi(x:real):real;
begin
    multi:=x*2;
end;
{--- Main Program ---}
BEGIN

```

```
write('Enter 1. Number:');readln(a);  
c:=multi(a);writeln('Result=',c);  
write('Enter 2. Number:');readln(a);  
d:=multi(a);writeln('Result=',d);  
readln;  
END.
```

4-5 الإجراءات والدوال المسبقة التعريف:

تحدثنا سابقاً عن الإجراءات والتوابع ولاحظنا بأنه يتم إعدادها من قبل المترجم وهي لا تختلف من حيث البنية عن البرنامج الرئيسي، إلا أن لغة التريبوباسكال تقدم لنا نمطاً آخر من الإجراءات والتوابع البسيطة المسبقة التعريف في بيئة التريبوباسكال والتي يمكن استخدامها مباشرة في البرنامج من خلال كتابة أسمها ومتحولاتها الوسيطة، من هذه الإجراءات والتوابع نذكر الآتي:

1- Round (x) : تقرب العدد الحقيقي X إلى أقرب عدد صحيح.

مثال (12):

Round (1.3)=1
Round (1.5)=2
Round (1.7)=2
Round (-1.5)=-1

2- Trunc (x) : تقطع الجزء العشري من العدد الحقيقي X :

مثال (13):

Trunc (1.2)=1
Trunc (1.5)=1

3- **Chr(x)** : تستخدم على الأعداد الصحيحة من (0-255) وتعطي ما يقابل هذه الأعداد من جدول ASCII :

مثال (14):

$$\text{Chr}(65)='A'$$

$$\text{Chr}(97)='a'$$

4- **Ord(x)** : وهي المعاكسة للدالة **Chr(x)** وهي تعطي الرقم المقابل للمحرف في جدول ASCII .

مثال (15):

$$\text{Ord}('A')=65$$

$$\text{Ord}('a')=97$$

5- **Abs(x)** : تعطي القيمة المطلقة للعدد X (صحيح أو حقيقي)

6- **Sin(x)** : تُعطي جيب الزاوية X المدخلة بالراديان .

7- **Cos(x)** : تُعطي جيب الزاوية X المدخلة بالراديان .

8- **Arctan(x)** : تُعطي الزاوية التي ظلها X مقدرة بالراديان .

9- **Sqr(x)** : تُعطي مربع القيمة X .

10- **Sqrt(x)** : تُعطي الجذر التربيعي للعدد X .

11- **Exp(x)** : تُعطي القيمة e^x .

12- **Int(x)** : تُعطي الجزء الصحيح من العدد الحقيقي X ويمثل الناتج بشكل حقيقي .

13- **Frac(x)** : تُعطي الجزء الكسري من العدد الحقيقي X .

14- **Ln(x)** : تُعطي اللوغارتم الطبيعي لـ X .

15- **Succ (x)** : تُعطي القيمة اللاحقة (التالية في الترتيب) لقيمة X .

مثال (16):

$$\text{Succ}(5)= 6$$

$$\text{Succ}('C')='D'$$

Pred(x)-16 : تُعطي القيمة السابقة لقيمة x :

مثال (17):

$$\text{Pred}(5)=4$$

$$\text{Pred}('D')='C'$$

Upcase(x)-17 : تُعطي الحرف اللاتيني الكبير المقابل للحرف اللاتيني الصغير x ،

(x : من النوع char).

Odd(x)-18 : تُعطي ناتج محقق (True) إذا كان x فردي و ناتج غير محقق

(False) إذا كانت x زوجي، حيث x صحيح.

مثال (18) :

$$\text{Odd}(x)=\text{True} \quad x : \text{ فردي}$$

$$= \text{False} \quad x : \text{ زوجي}$$

Inc(x,n)-19 : يجعل قيمة x مساوية إلى القيمة التالية بمقدار n :

مثال (19):

$$\text{Inc}(5,3)=8$$

$$\text{Inc}('C')='D'$$

Dec(x,n)-20 : يجعل قيمة x مساوية إلى قيمة تسبقها بالترتيب بمقدار n .

مثال (20):

$$\text{Dec}('C')='B'$$

$$\text{Dec}(5,3)=2$$

إذا لم تذكر n تكون مساوية 1 .

21-Str(v,s1): يحول عدد إلى سلسلة رمزية (string).

حيث v تمثل تعبير ويمكن أن يكون حقيقياً أو صحيحاً.

S1: تعبير من النمط سلسلة محرفية (String).

مثال (21):

البرنامج التالي يبين كيفية استخدام الإجراء Str:

```
Program Test53;  
var  
  s1 : string[30];  
  Value: Integer;  
BEGIN  
  Readln(Value);  
  Str(Value,s1);  
  Writeln('s1=',s1);  
  readln;  
END.
```

22-Val(S1,V,R): يحول السلسلة المحرفية S1 إلى عدد ويخزنه تحت الاسم V.

حيث **S1:** تعبير من النمط سلسلة محرفية (String).

V: تعبير من النمط الحقيقي أو الصحيح.

R: متحول من النمط الصحيح وهذا المتحول يحدد الموقع في السلسلة المحرفية،

الذي يمكن أن يظهر عنده الخطأ، عندما لا يحصل خطأ فإنه يكون $R=0$.

مثال (22):

البرنامج التالي يبين كيفية استخدام الإجراء Val:

```
Program Test54;  
var  
  s1: string[30];
```


Value: Integer;
Result: Integer;

BEGIN
 Readln(s1);
 val(s1,Value,Result);
 if Result = 0 then
 writeln('value=',value)
 else
 begin
 Writeln('Error in String at the Position ');
 writeln(Result);
 end;
 readln;
END.

Length(S)-23: يعطي طول سلسلة محرفية ، حيث **S** تمثل السلسلة المحرفية،
النتيجة في هذه الحالة تكون من النمط الصحيح.

مثال (23):

البرنامج التالي يبين كيفية استخدام التابع **Length**:

Program Test55;
var
 S1: string [30];
 I: Integer;
BEGIN
 Readln(S1);
 Writeln('Length='Length(S1));
 For I:=1 to Length (s1) do
 Write(S1[I]);
END.

24-Insert(S,S1,P): يقوم بحشر سلسلة محرفية S ضمن سلسلة أخرى S1 وذلك بدءاً بالموقع P من السلسلة S1.

إذا كانت قيمة P أكبر من طول السلسلة S1 فإنه يتم وصل السلسلة S بالسلسلة S1 ، أما إذا كان طول السلسلة S1 بعد حشر النص S أكبر من 255 محرف فإنه سيتم اقتطاع المحارف بعد المحرف 255.

مثال (24): البرنامج التالي يبين كيفية استخدام التابع Insert:

```
Program Test56;  
var  
    S1: string [30];  
BEGIN  
    Readln(S1);  
    Insert(' Example ',S1,1);  
    Write('S1=',S1);  
END.
```

25-Concat (S1,S2,.. ,Sn): يقوم هذا الإجراء بوصل عدة سلاسل محرفية مع بعضها البعض. إذا كان طول السلسلة بعد الوصل أكبر من 255 محرف فإنه سيتم اقتطاع المحارف الزائدة بعد المحرف 255.

مثال (25):

البرنامج التالي يبين كيفية استخدام التابع Concat:

```
Program Test57;  
var  
    s1: String[30];
```

```

s2: String[80];
s3: String[120];
BEGIN
  Readln(s1);
  Readln(s2);
  s3:=Concat (s1,s2);
  writeln('s3= ',s3);
  readln;
END.

```

26- Copy (S1,P,L): يقوم هذا التابع باقتطاع سلسلة محرفية جزئية من سلسلة أخرى.

حيث S1 تمثل السلسلة المراد الاقتطاع منها.

P الموقع الذي اعتباراً منه يتم الاقتطاع.

L طول السلسلة الجزئية المقتطعة.

إذا كانت قيمة P أكبر من طول السلسلة S1 فإن

النتيجة تكون عبارة عن سلسلة فارغة .

إذا كانت L أكبر من المحارف المتبقية في السلسلة S1

بدءاً من الموقع P ، فإن النتيجة ستكون فقط من

المحارف المتبقية.

مثال (26):

البرنامج التالي يبين كيفية استخدام التابع Copy:

```

Program Test58;
var
  s1: String[30];

```

```

s2: String[80];
BEGIN
  Readln(s1);
  s2:=Copy(s1,1,4);
  writeln('s2= ',s2);
  readln;
END.

```

مثال (27):

البرنامج التالي يبين استخدام التابعين Copy و Concat :

```

Program Test59;
var
  Name :String[30];
  First, Middle, Last :String[10];
BEGIN
  Write('Please enter your first name: ');
  Readln(First);
  First := Concat(First, ' ');
  Write('Please enter your middle name: ');
  Readln(Middle);
  Middle := Copy(Middle, 1, 1);
  Middle := Concat(Middle, '. ');
  Write('Please enter your last name: ');
  Readln (Last);
  Name := Concat(First, Middle, Last);
  Writeln;
  Writeln('Your complete name is: ',Name)
END.

```

27- Pos(S,S1): يقوم بالبحث عن السلسلة المحرفية S ضمن السلسلة المحرفية S1. الناتج في هذه الحالة سيكون من النمط Byte وهو يكون قيمة الترتيب في السلسلة الذي تظهر عنده الحرف الأول من السلسلة المبحوث عنها. في حالة عدم العثور على السلسلة المبحوث عنها فإن الناتج سيكون صفراً.

مثال (28):

البرنامج التالي يبين استخدام التابع Pos:

```
Program Test60;  
var  
  s1: String[30];  
  P: Byte;  
BEGIN  
  Readln(s1);  
  Insert('Example ',S1,1);  
  P:= Pos('ple',S1);  
  Writeln('Pos=',P);  
readln;  
END.
```

28- Delete (S1,P,L): يقوم بمسح سلسلة محرفية جزئية من سلسلة محرفية أخرى.

حيث S1 تمثل السلسلة المراد مسح جزء منها.

P الموقع الذي اعتباراً منه تبدأ عملية المسح.

L طول السلسلة الجزئية المراد مسحها.

إذا كانت قيمة P أكبر من طول السلسلة S1 فإنه لا
يتم مسح أي محرف.
إذا كانت L أكبر من المحارف المتبقية في السلسلة S1
اعتباراً من الموقع P ، فإن سيتم فقط مسح المحارف
المتبقية.

مثال (29):

البرنامج التالي يبين استخدام التابع Delete:

```
Program Test61;  
var  
  s1: String[30];  
BEGIN  
  Readln(s1);  
  Delete(s1,1,1);  
  Writeln('S1=',S1);  
  Readln;  
END.
```

مثال (30):

البرنامج التالي يبين استخدام الدالتين Int و Frac :

```
Program Test62;  
var X: Real;  
BEGIN  
  X := Frac(135.321); writeln(x);  
  X := Frac(-135.321); writeln(x);  
  x:=Int(135.321);writeln(X);  
  X:=INT(-135.321);Writeln(X);  
  Readln;  
END.
```

مثال (31):

البرنامج التالي يبين استخدام الدالة Exp :

```
Program Test63;  
uses Crt;  
BEGIN  
  Writeln('e = ',Exp(1.0));readln;  
END.
```

مثال (32):

البرنامج التالي يبين استخدام الدالة Inc :

```
Program Test64;  
var  
  x: Integer;  
BEGIN  
  Inc(x);writeln(x);  
  Inc(x, 5);writeln(x);readln;  
END.
```

مثال (33):

البرنامج التالي يبين استخدام الدالتين Succ, Pred :

```
Program Test65 ;  
var  
  N,X: Integer;  
BEGIN  
  N:=pred(5);writeln(N);  
  N:=succ(5);writeln(N);  
END.
```

مثال (34):

البرنامج التالي يبين استخدام الدالتين Trunc, Round :

Program Test66;

uses Crt;

BEGIN

textcolor(14);

Writeln(1.4, ' rounds to ', Round(1.4));

Writeln(1.5, ' rounds to ', Round(1.5));

Writeln(-1.4, ' rounds to ', Round(-1.4));

Writeln(-1.5, ' rounds to ', Round(-1.5));

Writeln('=====');

textcolor(3);

Writeln(1.4, ' becomes ', Trunc(1.4));

Writeln(1.5, ' becomes ', Trunc(1.5));

Writeln(-1.4, ' becomes ', Trunc(-1.4));

Writeln(-1.5, ' becomes ', Trunc(-1.5));

readln;

END.

مثال (35):

البرنامج التالي يبين استخدام الدالتين Str :

Program Test67;

var

I:integer;

S: string[10];

BEGIN

write('Enter Int. Value:');readln(I);

Str(I, S);

writeln('Note: I is Integer:(I='i,') and S is String:(S='s, ')');

readln;

END.

مثال (36):

البرنامج التالي يبين استخدام الدالتين Chr, Ord :

Program Test68;

uses Crt;

var

I: Integer;

BEGIN

Clrscr;

textcolor(12);

for I := 32 to 126 do

Write(Chr(I), ' ');

Writeln;

Writeln('=====');

textcolor(2);

for I := 32 to 126 do

Write(ord(Chr(I), ' ')) ;

Readln;

END.

مثال (37):

البرنامج التالي يبين استخدام الدالتين Val, Str :

Program Test69 ;

uses crt;

var

i,c:integer;

s: string[10];

BEGIN

textcolor(12);

writeln('Invert from Numeric To String: ');

write('Enter Numeric Value: ');readln(i);

str(i, s);

writeln('Note: i is Integer:(i= ',i,') and s is

String:(s= ',s,')');

```
textcolor(14);  
writeln('Invert from String To Numeric');  
val(s,i,c);  
writeln('i=';i);readln;  
END.
```



الفصل السادس
طرائق العرض على الشاشة

1-6 مقدمة

2-6 العرض النصي

3-6 العرض البياني (الرسومي)



طرائق العرض على الشاشة

6-1 مقدمة:

الحاسوب الشخصي يمكن أن يقوم بعرض المعطيات بحالتين، عرض النصوص (text Display) وعرض الرسوم (Graphics Display) وسنتحدث الآن عن طريقة عرض النصوص وسوف نناقش لاحقاً كيفية عرض الرسوم البيانية. في حالة عرض النصوص يمكن للحاسوب الشخصي أن يتعامل مع كل الرموز الـ 256 في جدول الأسكي. من أجل عرض الرموز يستخدم الكمبيوتر محول فيديو (Video adapter) يقوم بوصل الكمبيوتر بالشاشة. معظم الحواسيب تمتلك إما محول أحادي اللون أو محول رسومي ملون (CGA) مع ذلك فإن المحولات الرسومية (EGA: enhanced Graphics adapter و (VGA: Video Graphics adapter) هي الشائعة الاستخدام.

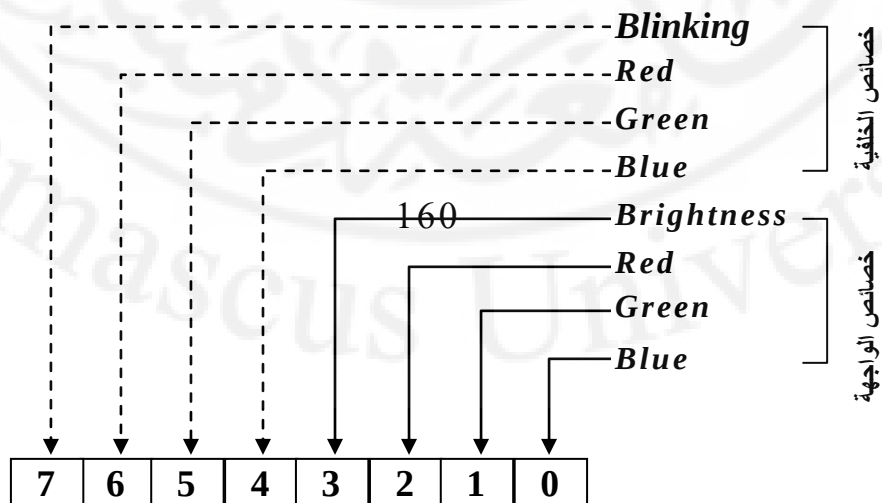
6-2 العرض النصي:

6-2-1 المحول الرسومي وذاكرة العرض:

الشاشة في الحاسوب يمكن أن تعرض حتى 80 رمزاً بالاتجاه الأفقي و 25 بالاتجاه العمودي. أو بشكل عام 2000 رمز في كامل الشاشة. كل رمز يكون له لون خاص به ولون خاص بالخلفية. اللون الخاص بالرمز يكون هو لون الرمز نفسه. لون الخلفية يمثل لون المنطقة المحيطة بالرمز. الحاسوب يقوم بتخزين الرموز والألوان الخاصة فيها في ذاكرة تسمى بذاكرة العرض (Display Memory)، هذه الذاكرة تحدد للحاسوب الرموز والألوان المراد عرضها. مع أن هذه الذاكرة موجودة على بطاقة المحول الرسومي إلا أنها تُعتبر جزء من الذاكرة Ram.

البايت الأول في ذاكرة العرض تتضمن ترميز المحرف الأول الذي سيتم عرضه في الزاوية اليسارية العليا من الشاشة. فإذا وجد مثلاً الرمز 41h (وهو الترميز الست عشري في جدول الأسكي للمحرف A)، فإنه سيتم عرض المحرف A في الزاوية اليسرى العليا من الشاشة. البايت الثاني في ذاكرة العرض يُسمى بايت الخصائص للمحرف الأول، وهو يتضمن لون المحرف إضافة إلى معلومات أخرى تتعلق بالعرض. هذا النموذج محرف - خصائص، محرف - خصائص - يتم تكراره بالنسبة لـ 2000 محرف التي ستظهر على الشاشة. لذلك فإن الذاكرة البيانية بحالة شاشة بسيطة تتضمن 4000 بايت.

إن محتوى بايت الخصائص يكون على الشكل التالي: البتات الثلاثة الأولى (bits) تتحكم بألوان المحرف بينما البت الرابع يتحكم بالسطوع، يمكن استخدام هذه الألوان الأربعة للحصول على الـ 16 لون المختلفة. البتات رقم 4، 5، 6، تحدد لون الخلفية للمحرف، بينما يحدد البت السابع يحدد الإضاءة (Blinking) وذلك لأن الخلفية ليس لها عنصر يحدد السطوع، لذلك فإنه يكون لها فقط ثمانية ألوان (الشكل 6-1).



الشكل (6-1): محتوى بايت الخصائص

6-2-2 أنماط العرض على الشاشة:

يمكن في الحاسوب الشخصي استخدام 16 نمط مختلف للعرض على الشاشة. هذه الأنماط تتحكم باللون وحجم الخط على الشاشة. الجدول التالي يبين هذه الأنماط المختلفة:

Mode	Size	Colors	Adapter
0 (BW40)	40*25	Shades of Gray	CGA,EGA
1 (BW80)	40*25	Colors	CGA,EGA
2 (C40)	80*25	Shades of Gray	CGA,EGA
3 (C80)	80*25	Colors	CGA,EGA
7 (mono)	80*25	Black and white	Monochrom

وهنا لابد من الإشارة إلى أنه عند استخدام العرض النصي على الشاشة يجب في البداية استدعاء الوحدة الخاصة بالعرض النصي وهي الوحدة CRT ويتم استدعائها في بداية البرنامج بالعبارة:

Uses Crt

مثال (1):

البرنامج التالي يبين كيفية العرض على الشاشة:

```
Program Test70;  
uses crt;  
var  
i:byte;  
Col:Byte;  
s:string[80] ;  
BEGIN  
Textmode(c80);  
S:='One color Row';  
i:=1;  
col:=1;  
while i<=length(s) do  
  begin  
    Textcolor (i);  
    Write(s[i]);  
    if s[i]<> ' ' then Inc(col);  
    Inc(i);  
  end;
```


Readln;
END.

3-2-6 التحكم بالألوان:

يمكن في لغة التريباسكال التحكم بلون الخلفية والواجهة للنص المكتوب ولك باستخدام التعليمات:

TextColor
TextBackground

المحول الرسومي يؤمن 16 لوناً كما هو مبين بالجدول التالي:

Dark Colors		Light Color	
Number	Colors	Number	Colors
0	Black	8	DarkGray
1	Blue	9	LightBlue
2	Green	10	LightGreen
3	Cyan	11	LightCyan
4	Red	12	LightRed
5	Magenta	13	LightMagenta
6	Brown	14	Yellow
7	LightGray	15	White

مثال (2):

البرنامج التالي يبين كيفية التحكم بألوان النص على الشاشة:

```
Program Test71;  
uses crt;  
BEGIN
```

```

Clrscr;
Textcolor(red);
writeln('ColorTest');
Textcolor(Cyan);
writeln('ColorTest');
Textcolor(Cyan+blink);
writeln('ColorTest');
readln;
END.

```

مثال (3):

البرنامج التالي يبين كيفية التحكم بألوان النص والخلفية على الشاشة:

```

Program Test72;
uses crt;
BEGIN
  TextBackGround (green);
  Clrscr;
  TextBackGround(Red);
  TextColor(Yellow);
  Writeln('This is a Test');
  Readln;
END.

```

6-2-4 استخدام إحداثيات الشاشة:

إن شاشة الحاسوب تشبه الخارطة فهي تملك مواقع مختلفة يمكن تعريفها من خلال الإحداثيات. وهنا يُعبر عن الإحداثيات بالسطر X والعمود Y. لذلك فإن إحداثيات الزاوية اليسارية العلوية من الشاشة تكون (1,1) بينما إحداثيات الزاوية اليمينية السفلية تكون (80,25). يمكن التحكم بمؤشر الكتابة على الشاشة وذلك باستخدام الإجراءات:

1- الإجراءات GOTOXY:

من خلال هذا الإجراء يتم تحريك مؤشر الكتابة على النافذة الفعالة وذلك من خلال تحديد إحداثياتها.

الشكل العام لهذه التعليمة:

GOTOXY(X,Y)

مثال (4):

البرنامج الآتي يبين كيفية تحريك مؤشر الكتابة على الشاشة:

```
Program Test73;  
uses crt;  
var  
x:byte;  
s:string[80];  
BEGIN  
Clrscr;  
S:='This is a Test';  
for x:=1 to Length(s) do  
begin  
    gotoXY(x,3);  
    write (S[x]);  
    delay(10000);  
end;  
Readln;  
END.
```

3- الإجراء WhereX , WhereY:

يُستخدم هذان الإجراءان من أجل تحديد الموقع الحالي لمؤشر الكتابة على الشاشة، فالإجراء WhereX تحدد الإحداثي x والإجراء WhereY تحدد الإحداثي y .

مثال (5):

البرنامج الآتي يبين كيفية تحديد موقع مؤشر الكتابة على الشاشة:

```
Program Test74;  
uses crt;  
var  
  i:byte;  
  s:string[80];  
Procedure Outchar(ch:char);  
var  x: Byte;  
      y: Byte;  
  begin  
    x:=whereX;  
    y:=whereY;  
    GotoXY(75,25);  
    write(x:2,',',y:2);  
    textColor(7);  
    GotoXY(x,y);  
    write(ch);  
    Delay(50000);  
  end;  
BEGIN  
  GotoXY(30,15);  
  s:='a Test of WhereX';  
  for i:=1 to Length(s) do  
    OutChar(s[i]);  
END.
```

مثال (6):

البرنامج الآتي يبين كيفية التحكم بمؤشر الكتابة على الشاشة:

```
Program Test75;
```

```

uses crt;
var
  i, code,x,y:integer;
  s: string[20];
Procedure DrawBox(x1,y1,x2,y2,fg,bg:Integer);
var
  i: byte;
begin
  textColor (fg);
  textBackGround(bg);
  for i:= (x1+1) to (x2-1) do
    begin
      gotoXY(i,y1);
      write(#205);
      gotoXY(i,y2);
      write(#205);
    end;
  for i:= (y1+1) To (y2-1) do
    begin
      gotoXY(x1,I);
      write(#186);
      gotoXY(x2,I);
      write(#186);
    end;
      gotoXY(x1,y1);
      write(#201);
      gotoXY(x2,y1);
      write(#187);
      gotoXY(x1,y2);
      write(#200);
      gotoXY(x2,y2);
      write(#188);

```

```

end;
Procedure GetNumber;
begin
    drawBox(5,14,35,16,white,Black);
    gotoXY(7,15);
    write('Enter a Number (1-10): ');
    x:=whereX;
    y:=whereY;
    repeat
        gotoXY(x,y);
        s:= ' ';
        write(s);
        gotoXY(x,y);
        readln(s);
        val(s,i,code);
    until ((i>0) and (i<11) and (code=0));
end;
BEGIN
    clrscr;
    textMode(c40);
    GetNumber;
    clrscr;
    textMode(c80);
    GetNumber;
    gotoXY(1,20);
    writeln;
    write('Press Enter.....');
    readln;
END.

```

5-2-6 إجراءات أخرى لأنماط العرض على الشاشة:

تحدثنا حتى الآن عن بعض التعليمات (إجراءات، توابع، دوال) التي تُستخدم للتعامل مع أنماط عرض النصوص على الشاشة، وقد لاحظنا أن جميع هذه التعليمات لا يمكن تنفيذها إذا لم تكن الوحدة CRT مفعلة في البرنامج.
نريد الآن عرض بعض الإجراءات الأخرى التي تعمل مع هذه الوحدة:

1- الإجراءات Window :

يُستخدم هذا الإجراء من أجل تعريف نافذة في حالة العرض بالنمط النصي، الشكل العام لهذا الإجراء:

Window (left,Top,Right,Bottom)

حيث:

Left, Top : تعرف زاوية النافذة اليسارية العلوية.

Right,Bottom : تعرف زاوية النافذة اليمينية السفلية.

عند إقلاع البرنامج تكون النافذة الفعالة إما (1,1,80,25) أو (1,1,40,25) وذلك حسب النمط النصي المستخدم (TextMode)

مثال(7):

البرنامج الآتي يبين كيفية إنشاء نافذة على الشاشة والتحكم بألوان هذه الشاشة:

```
Program Test76;  
uses crt;  
Procedure FillWindow(Color:Byte);  
begin  
  TextBackGround(Color);  
  Clrscr;  
end;
```

BEGIN

```
Fillwindow(White);  
Window(5,5,35,25);  
Fillwindow(Red);  
Window(30,8,60,15);  
Fillwindow(Blue);  
Window(55,13,78,20);  
Fillwindow(LightGray);  
Window(40,7,75,14);  
Fillwindow(Black);  
Readln;
```

END.

2- الإجراء ClrScr :

يُستخدم هذا الإجراء من أجل مسح النافذة الفعالة، ويتم نقل مؤشر الكتابة إلى الزاوية اليسارية العلوية، أي إلى الإحداثيات (1,1).

3- الإجراء Delay:

يُستخدم هذه الإجراء من أجل تأخير تنفيذ البرنامج، الشكل العام لهذه التعليمة:

Delay(Time)

وتمثل المعامل Time هنا مدة التأخير بالملي ثانية.

4- الإجراء DelLine :

يُستخدم هذا الإجراء من أجل مسح السطر الحالي للنتائج، أي السطر الذي يوجد فيه مؤشر الكتابة.

مثال (8):

البرنامج الآتي يبين استخدام الإجراء DelLine من أجل مسح سطر من الشاشة:

Program Test77;

uses Crt;

BEGIN

Clrscr;

textcolor(5);

writeln('Please Wait');

Window(10, 10, 60, 20);

writeln('This Line will be not delete');

write('This Line will be delete');

DelLine;

Readln;

END.

5- الإجراء HighVideo:

عند استخدام هذا الإجراء تظهر الحارف على الشاشة بشكل ساطع، وللعودة إلى الحالة الطبيعية، أي الحالة التي كانت قبل إقلاع البرنامج نستخدم الإجراء NormVideo.

6- الإجراء LowVideo:

عند استخدام هذا الإجراء تظهر الحارف على الشاشة أقل سطوعاً (نصف ساطع) ، وللعودة إلى الحالة الطبيعية، أي الحالة التي كانت قبل إقلاع البرنامج نستخدم الإجراء NormVideo.

7- الإجراء Sound :

عند استخدام هذا الإجراء يتم تشغيل مكبر الصوت وإصدار صوت حسب التردد المستخدم في هذا الإجراء.

مثال (9):

البرنامج الآتي يبين كيفية إصدار صوت وإخفاؤه من الحاسوب:

```

Program Test78;
uses Crt;
var
  i: integer;
BEGIN
  clrscr;
  for i:=1 to 1000 do
  begin
    sound(i);
    delay(100);
  end;
  nosound;
  readln;
END.

```

من أجل إلغاء مفعول هذا الإجراء يُستخدم الإجراء Nosound.

8- الإجراء ClrEol:

يقوم هذا الإجراء بمسح كل المحارف اعتباراً من موقع مؤشر الكتابة (على شاشة النتائج) وحتى نهاية السطر من دون تحريك مؤشر الكتابة، وهنا يتم استبدال المحارف المراد مسحها بمحارف فارغة ذات لون مماثل للون الخلفية وفي حال عدم تعريف لون الخلفية فإن المحارف الفارغة تأخذ اللون الأسود.

9- الإجراء InsLine:

يُستخدم هذا الإجراء من أجل حشر سطر فارغ عند الموقع الحالي لمؤشر الكتابة. كل الأسطر الواقعة للأسفل سوف تتزاح إلى الأسفل بمقدار سطر.

10- الإجراء ReadKey:

تحدثنا سابقاً عن الإجراءات Read و Readln ، ولاحظنا أنه يمكن باستخدام هذه الإجراءات قراءة المعطيات المدخلة من قبل المستخدم، أو من خلال ملف. الإجراء ReadKey المعرف في الوحدة CRT يستقبل أيضاً إدخالات من قبل المستخدم وذلك من خلال لوحة المفاتيح ولكنه في هذه الحالة لا يستجيب لهذه المفاتيح، أي أنه لا يُظهر هذا الرمز على الشاشة، وإذا أردنا إظهار هذا الرمز لا بد من استخدام تعليمات أخرى في البرنامج.

يتميز هذا الإجراء أيضاً بأنه يمكن أن يقرأ المفاتيح الخاصة، مثل مفاتيح الوظائف، وتتم هذه العملية هنا بأن كل مفتاح يكون له شفرة خاصة به تُعرف بشفرة المسح (Scan Code)، ففي حالة الرموز العادية تكون شفرة المسح هي شفرة الأسكي (Ascii)، أما في حالة المفاتيح الخاصة فإن الأمر يتعلق بشفرة مسح موسعة (extended scan code) لهذه المفاتيح، هذه الشفرة مكونة من جزئين:

الجزء الأول يمثل بالرمز #0 وهي تخبر البرنامج بأن البحث يتم ضمن الشفرة الموسعة. الجزء الثاني يمثل شفرة المفتاح، وهاتان الشفرتان لا تحتاج إلى ضغط المفتاح مرتين وإنما تحتاج إلى استدعاء الإجراء ReadKey مرتين.

كمثال على ذلك إذا ضغطنا على السهم اليساري (Left arrow)، فإن التابع ReadKey سوف يُعيد شفرة مسح موسعة جزئها الأول هو الشفرة #0 ، وجزئها الثاني هو الشفرة #75 ، وهذا الجزء الثاني نحصل عليه بعد استدعاء الإجراء ReadKey مرة ثانية).

مثال (10):

البرنامج التالي يتعرف على المحرف الذي تم ضغطه من لوحة المفاتيح ويظهر الشفرة الخاصة بهذا المفتاح:

Program Test79;*uses Crt;**var**c: Char;***BEGIN***clrscr;**writeln('Please press a key');**c := Readkey;**writeln(' You pressed ', C, ', whose ASCII value is ', Ord(C), '');**readln;***END.**

نلاحظ أن هذا البرنامج لا يتعرف على المفاتيح الخاصة (المفاتيح الوظيفية) الأخرى في لوحة المفاتيح.

مثال (11):

في هذا البرنامج يتم قراءة مفتاح من لوحة المفاتيح وإظهار شفرة هذا المفتاح، بغض النظر، فيما إذا كان هذا المفتاح، مفتاحاً خاصاً أو أي مفتاح آخر:

Program Test80;*uses Crt;**var**FuncKey: Boolean;**Ch: Char;***Procedure ReadChar;****begin***ch:=Readkey;**if ch=#0 then***begin***FuncKey:=True;**ch:=Readkey;***end**

```

Else
    FuncKey:=False;
end;
BEGIN
Readchar;
If FuncKey then
    Writeln('special Key was Pressed, whose Code is: ',
ord(ch))
Else
    Write('You Pressed Char ',ch,', whose Code is: ', ord
(ch));
readln;
END.

```

3-6 العرض البياني (الرسومي):

تحدثنا سابقاً عن العرض النصي في لغة التريبواسكال وتعرفنا إلى تقسيمات الشاشة في حالة العرض النصي، ولاحظنا أن الرموز يتم توزيعها وفق شبكة وهذا يختلف حسب تقسيمات الشاشة، أي عدد الأعمدة وعدد الأسطر فيها. عند كل موضع من الشبكة يجب أن يكتب به رمز من الرموز، ولا يجوز كتابة رمزين في موضع واحد، وكذلك لا يجوز كتابة رمز في موضعين.

هذا في حالة العرض النصي أما في حالة العرض الرسومي فإن الأمر يكون مختلفاً، فهنا يمكن التحكم بكل نقاط الشاشة (Pixels)، وعدد هذه النقاط يختلف حسب نوع ملائم الشاشة (Graphic Adapter)

قبل التعرف إلى بعض التعليمات الخاصة بالرسم البياني في لغة التريبواسكال، لا بد في البداية من التعرض إلى بعض المفاهيم والأدوات الأساسية الضرورية لتنفيذ عمليات الرسم البياني.

1-3-6 الوحدة Graph:

أنشئت هذه الوحدة من قبل شركة بورلاند Borland كإحدى أدوات ملائمة الرسومات ، حيث يمكن من خلالها تزويد لغة التريباسكال بمجموعة من روتينات الرسم .

6-3-2 سواقات الرسومات Graphics Drivers:

السواقة (Driver) هي عبارة عن برنامج يعمل وسطياً بين البرنامج وبين عتاد الحاسوب. وبما أن عملية الرسم في الحاسوب تحتاج كثيراً إلى عتاد الحاسوب، لذلك لا بد من وجود وسيلة للربط بين عتاد الحاسوب وبين البرنامج الذي يتضمن تعليمات أو روتينات الرسم.

إن بيئة التريباسكال تؤمن لنا مجموعة من الملائمات الرسومية (Adapters):

CGA, MCGA
EGA
VGA
Hercules
AT&T 400 series
IBM 3270 PC
IBM 8514

وهناك أيضاً مجموعة من الملائمات الأخرى المتوافقة مع هذه الملائمات. إن سواقات هذه الملائمات محتواة ضمن بيئة التريباسكال وذلك في الملفات التي لاحقتها (BGI) .

من أجل الرسم في التريباسكال يجب أولاً استدعاء وحدة الرسم Graph ، ومن ثم إجراء عملية ملائمة بين عتاد الحاسوب وبين البرامج المسؤولة عن الرسم ضمن هذه الوحدة. من أجل ذلك تُستخدم التعليمة :

InitGraph (Driver, mode, Path)

حيث:

Driver: عبارة عن متحول يمثل السواعة التي سوف تستدعي نظام الرسم، وهذا المتحول يجب أن يُستدعى مسبقاً، وهو يأخذ قيمة صحيحة.

Mode: وهو مسؤول عن كيفية استخدام نمط الرسم الذي يتم اختياره، وهو عبارة عن متحول صحيح أيضاً.

بشكل عام Initgraph يقوم بفحص عتاد الحاسوب للتعرف إليه، ومن ثم يتم تحديد قيم مناسبة لكل من الـ Driver و الـ Mode وذلك من أجل اختيار الطريقة المناسبة لاستخدام الرسم على هذا العتاد.

ويُسند هنا للمتحول Driver القيمة detect وذلك كالآتي:

Driver:=detect

وبذلك يتم استدعاء البرنامج DetectGraph الذي يقوم بالتعرف إلى عتاد الحاسوب ويؤمن الـ driver والـ mode المناسبين.

3-3-6 تعليمات الكتابة على شاشة الرسم:

1- تعليمة تحريك مؤشر الكتابة:

MoveTo (x,y)

تُستخدم هذه التعليمة من أجل تحريك المشيرة على الشاشة على الإحداثيات (x,y).

2- تعليمة طباعة نص على الشاشة:

OutText(' '):

تُستخدم من أجل طباعة الثابت المحرفي (النص) الموجود بين القوسين على الشاشة، وغالباً ما تُستخدم هذه التعليمة مع التعليمة MoveTo.

ويمكن هنا أيضاً استخدام تعليمة أخرى يمكن أن تحل مكان التعليمتين السابقتين
MoveTo و Outtext ، وهذه التعليمة هي:

OutTextXY (x,y,' ')

حيث يتم هنا إخراج النص بدءاً من الإحداثيات x,y.

3- تعليمة تحديد خصائص خط النص:

SetTextStyle(font,Dir,Size)

تُستخدم هذه التعليمة من أجل تحديد نوع خط الرسم، وحجمه، واتجاهه، حيث :
Font: تحدد نوع الخط، وهي تأخذ القيم الآتية:

- 0: Defaultfont
- 1: Triplexfont
- 2: Smallfont
- 3: Sansseriffont
- 4: Gothicfont

Dir: تحدد اتجاه الخط، وهي تأخذ القيم الآتية:

- 0: HorizDir (كتابة بالاتجاه الأفقي)
- 1: VertDir (كتابة بالاتجاه العمودي)

Size: تحدد حجم الخط ويمكن أن تأخذ إحدى القيم من 1 - 10 والحجم العادي هو
1.

4- تعليمة تغيير اللون:

SetColor(n)

تُستخدم لتحديد لون الطباعة على الشاشة (نص أو رسم) ، حيث تمثل n رقم اللون.

5- تعليمة تحديد خصائص خط الرسم:

SetLineStyle(Style,Pattern,Thickness)

حيث:

Style متحول وسيطي يحدد نوع الخط ويأخذ إحدى القيم الآتية:

القيمة	نوع الخط
0	خط مستقيم عادي (SolidLn)
1	خط مستقيم منقط (DottedLn)
2	خط مستقيم مكون من خطوط أفقية مختلفة الطول (CenterLn)
3	خط مستقيم مكون من خطوط أفقية متساوية الطول (DashedLn)
4	يتم تحديد شكل الخط باستخدام المتحول الوسيطي Pattern

Thickness: متحول وسيطي يحدد حجم الخط، ويأخذ إحدى قيمتين:

القيمة	حجم الخط
1	حجم عادي (Norm Width)
	حجم مضاعف (Thick Width)

عندما نستخدم القيمة 3 فإنه سيتم رسم خط مضاعف وفي هذه الحالة يتم رسم ثلاثة خطوط قريب بعضها من بعض فيبدو الخط مضاعفاً.

عند استخدام أي قيم أخرى غير هذه القيم (1 و 3)، فإنه سيتم العمل بالخط العادي (1).

Pattern: يُستخدم هذا المتحول فقط عندما يأخذ المتحول Style القيمة (4)، وهو يستخدم نمط من البتات لرسم المستقيم .

6- تعليمة رسم خط مستقيم:

Line(x1,y1,x2,y2)

تُستخدم لرسم خط مستقيم بين نقطتين إحداثياتهما (x1,y1) و (x2,y2) ، وهي تُستخدم أيضاً لرسم المنحنيات.

مثال(12):

في هذا المثال يتم رسم عدة خطوط مستقيمة بألوان مختلفة:

```
Program Test81;  
uses Crt, Graph;  
var  
    Gd, Gm,j: Integer;  
BEGIN  
    Gd := Detect;  
    InitGraph(Gd, Gm, "");  
    for j:=1 to 20 do  
        begin  
            setcolor (j);  
            line (20,20+j*5,400,20+j*5);  
        end;  
        Readln;  
    CloseGraph;  
END.
```

مثال (13):

في هذا المثال يتم رسم عدة خطوط عشوائياً:

```
Program Test82;  
uses Crt, Graph;  
var Gd, Gm,j: Integer;  
BEGIN  
  Gd := Detect;  
  InitGraph(Gd, Gm, "");  
  if GraphResult <> grOk then  
    Halt(1);  
  Randomize;  
  repeat  
    setcolor(j);  
    Line(Random(300), Random(300), Random(300),  
Random(300));  
    j:=j+1;  
  until KeyPressed;  
  Readln;  
  CloseGraph;  
END.
```

يمكن أيضاً استخدام تعليمة أخرى لرسم خط مستقيم وهي:

LineTo (x,y)

وترسم هذه التعليمة خط مستقيم من الموقع الحالي للمشيرة إلى النقطة التي إحداثياتها (x,y). وغالباً ما تُستخدم هذه التعليمة مع التعليمة MoveTo (x,y) وفي هذه الحالة يُعطي استخدام هاتين التعليمتين معاً، مفعول استخدام التعليمة Line نفسه.

مثال (14):

التعليمة :
تعيين النتيجة نفسها كالتعليمة:

```
MoveTo(100,100)  
LineTo(200,200)
```

```
Line(100,100,200,200)
```

7- تعليمة رسم نقطة:

PutPixel(x,y,c)

تُستخدم لرسم نقطة، حيث (x,y) تمثل إحداثيات هذه النقطة و C لونها.

مثال (15):

يتم في هذا المثال رسم خط مائل مكون من مجموعة من النقاط بألوان مختلفة.

```
Program Test83;  
uses Crt, Graph;  
var Gd, Gm,j: Integer;  
BEGIN  
  Gd := Detect;  
  InitGraph(Gd, Gm, "");  
  for j:=1 to 500 do  
    begin  
      setcolor (j);  
      Putpixel(50+j,50+j,j);  
    end;  
  Readln;  
  CloseGraph;  
END.
```

8- تعلیمة رسم مستطیل:

Rectangle (Left,Top,Right,Bottom)

تُستخدم لرسم مستطیل زاویته العليا اليسرى محدد بالنقطة التي إحداثياتها (Left,Top) ، والزاوية اليمنى السفلى محددة بالنقطة التي إحداثياتها (Right, Bottom) .

مثال (16):

يتم في هذا المثال رسم مستطیلين بإحداثيات مختلفة، وبنمط خط رسم مختلف:

```
Program Test84;  
uses Crt, Graph;  
var Gd, Gm,j: Integer;  
BEGIN  
  Gd := Detect;  
  InitGraph(Gd, Gm, "");  
  rectangle (50,50,500,200);  
  setlinestyle(0,1,3);  
  outtextxy(30,20,'Riad');  
  setcolor(4);  
  outtextxy(90,80,'Taifour');  
  rectangle(100,100,600,300);  
  Readln;  
  CloseGraph;  
END.
```

9- تعلیمة رسم دائرة:

Circle (x,y,Rad)

يتم رسم دائرة مركزها النقطة (x,y) وقطرها rad .

مثال (17):

في هذا البرنامج يتم رسم عدة دوائر متمركزة:

```
Program Test85;  
uses Graph;  
var  
    Gd, Gm: Integer;  
    Radius: Integer;  
    c:char;  
BEGIN  
    Gd := Detect;  
    InitGraph(Gd, Gm, ' ');  
    if GraphResult <> grOk then  
        Halt(1);  
    for Radius := 1 to 20 do  
        begin  
            setcolor (radius);  
            Circle(300, 200, Radius * 10);  
        end;  
    Readln;  
    CloseGraph;  
END.
```

10- تعلیمة رسم قطع ناقص:

Ellipse(X,Y,WB,WE,Xrad,Yrad)

يتم رسم قطع ناقص إحداثيات النقطة المتوسطة له (X,Y) ، وحيث تمثل:

WB: زاوية البداية

WE: زاوية النهاية

Xrad: المحور الأفقي

Yrad: المحور العمودي

مثال (18):

في هذا البرنامج يتم رسم قطع ناقص:

```
Program Test86;  
uses Crt, Graph;  
var  
    Gd, Gm,j: Integer;  
BEGIN  
    Gd := Detect;  
    InitGraph(Gd, Gm, "");  
    setcolor(4);  
    Ellipse(100,100,20,120,200,100);  
    setcolor(14);  
    outtextxy(60,80,'Test Ellipse');  
    Readln;  
    CloseGraph;  
END.
```



الفصل السابع
الملفات والوحدات

Files 1-7 الملفات

Units 2-7 الوحدات



الملفات والوحدات

7-1 الملفات:

إن جهاز الإدخال القياسي المستخدم في الحاسوب هو لوحة المفاتيح، وجهاز الإخراج هي الشاشة. إن استخدام هذه الأجهزة لعمليات الإدخال والإخراج تجري غالباً عندما تكون المعطيات المدخلة والمخرجة ليست كثيرة. إلا أنه عندما تكون هذه المعطيات كثيرة وتملك صفة الإستمرارية، أي أنه لا تتغير كثيراً فإنه لا بد في مثل هذه الحالات من استخدام الملفات.

الملف هو عبارة عن موقع على القرص يبحث فيه البرنامج عما يحتاجه وفق أسلوب منطقي، وبغض النظر عن طريقة توضع المعلومات فيه. والملف يمكن أن يتضمن معلومات عديدة، سلاسل رمزية، قيم عددية،....، وتتم قراءة عناصر الملف عن طريق السواقات (Driver) وبمساعدة بيئة التروباسكال.

7-1-1 أنواع الملفات:

ملفات نصية: وهي الملفات التي تكون محتوياتها عبارة عن نصوص.

ملفات غير نصية: وهي نوعين:

1- ملفات محددة نوع المعطيات (typed Files)

2- ملفات غير محددة نوع المعطيات (untyped Files)

سنقتصر في دراستنا هنا على الملفات النصية فقط.

■ الملفات النصية:

كما ذكرنا أعلاه أن محتويات هذه الملفات تكون عبارة عن نصوص، وهذه النصوص يمكن أن تكون سلاسل رمزية أو قيم عددية، لذلك يُسمى الملف النصي بملف أسكي لأن جميع بياناته هي من محارف جدول أسكي وتكون جميع هذه البيانات مشفرة وفق جدول الأسكي. بالإضافة إلى ذلك يتميز هذا النوع من الملفات بأنه يمكن قراءة محتواه بأي نوع من أنواع محررات النصوص. وهنا نشير إلى أنه يوجد في لغة التريبوباسكال نمط معطيات مسبق التعريف هو نوع المعطيات ملف نصي (Text).

7-1-2 العمليات على الملفات:

تنفذ على الملفات عدة عمليات، إنشاء ملف، قراءة ملف وإيضاً إضافة إلى ملف...

1- إنشاء ملف:

من أجل إنشاء ملف نصي يتم اتباع الخطوات التالية:

- أ- تعريف متحول الملف، وغالباً يتم تعريفه من النوع النصي.
- ب- يتم الربط بين اسم الملف المراد إنشاؤه، أي الاسم المعروف من قبل نظام التشغيل، وبين متحول الملف، المصرح عنه في البرنامج، ويتم ذلك باستخدام التعليمة Assign.
- ت- تستخدم تعليمة أو إجراء إنشاء الملف متبوعة بمتحول الملف وذلك على الشكل التالي:

Rewrite(متحول الملف)

ث- يتم كتابة المعطيات المراد إدخالها في الملف باستخدام التعليمة Write كالتالي:

Write (القيمة/متحول الملف)

ج- يُغلق الملف باستخدام التعليمة Close كالتالي:

Close (متحول الملف)

باستدعاء الإجراء Close يتم تفريغ الذاكرة الوسيطة (Buffer) على القرص، والذاكرة الوسيطة هي عبارة عن ذاكرة صغيرة الحجم تعادل 128 بايت خاصة بالملفات النصية في لغة التريبواسكال، وتكتب فيها المعلومات قبل تفريغها على القرص، وفي حال امتلاء هذه الذاكرة قبل إغلاق الملف، فإنه سيتم تفريغها في الملف على القرص.

يمكن هنا أيضاً القيام بعملية تفريغ للذاكرة الوسيطة حتى قبل امتلاءها أو قبل إغلاق الملف، وذلك عن طريق استخدام الإجراء Flush ويجب أن يُستخدم متحول الملف مع هذا الإجراء وذلك كالتالي:

Flush(filevar)

مثال (1):

البرنامج التالي يبين كيفية إنشاء ملف:

Program Test87;

var

i,j: integer;

filename: string[10];

filevar:text;

x:array[1..10] of real;

BEGIN

writeln('Enter value of x[i]');

```

for i:=1 to 5 do
  begin
    write('x[' ,i, ']=');
    readln(x[i]);
  end;
write('Enter filename:');
readln(filename);
assign(filevar,filename);
rewrite(filevar);
for j:=1 to 5 do
  write(filevar,x[j]);
close (filevar);
readln;
END.

```

2- قراءة ملف:

- من أجل قراءة ملف مخزن باسم محدد، نقوم بالخطوات التالية.
- أ- تكرر الخطوات أ و ب في خطوات إنشاء ملف.
 - ب- يتم فتح الملف باستخدام الإجراء Reset على الشكل التالي:
(متحول الملف) Reset
 - ت- يتم قراءة المعطيات المخزنة بالملف باستخدام التعليمة read على الشكل التالي:

(القيمة،متحول الملف) Read

- ث- يُغلق الملف باستخدام التعليمة Close كالتالي:

(متحول الملف) Close

مثال (2):

البرنامج التالي يبين كيفية قراءة ملف:

Program Test88;

var

i,j: integer;

filename: string[10];

filevar:text;

x:array[1..10] of real;

BEGIN

write('Enter filename:');

readln(filename);

assign(filevar,filename);

reset(filevar);

for j:=1 to 5 do

read(filevar,x[j]);

close (filevar);

for j:=1 to 5 do

write(x[j]);

readln;

END.

عند قراءة ملف، غالباً ما تكون نهاية ملف غير معروفة وفي مثل هذه الحالات يتم استخدام تعليمة Eof لتحديد نهاية الملف.

مثال (3):

البرنامج السابق (Test88) بعد استخدام Eof لتحديد نهاية الملف:

Program Test89;

uses crt;

var

i,j: integer;

filename:string[10];

filevar:text;

x:array[1..5] of integer;

BEGIN

```

clrscr;
write('enter the filename:');
readln(filename);
assign(filevar,filename);
reset (filevar);
i:=i+1;
repeat
read (filevar,x[i]);
i:=i+1;
until eof(filevar);
close(filevar);
for i:=1 to 5 do
writeln(x[i]);readln;
END.

```

3- إضافة بيانات إلى ملف:

من أجل إضافة بيانات إلى ملف موجود، يتم استخدام الخطوات المستخدمة في إنشاء ملف نفسها، ما عدا أنه يتم استبدال التعليمة Rewrite بالتعليمة Append . عند استخدام الإجراء Append سيتم فتح الملف المراد إضافة بيانات إليه، ويتوقف مؤشر الملف في نهاية الملف لتلقي البيانات المراد إضافتها. يجب الحذر عند استدعاء الإجراء Append لملف مفتوح، فإنه سيتم إغلاق الملف المفتوح ثم فتحه من جديد، وهذه العملية قد تؤدي إلى ضياع في المعلومات، لذلك يُفضل عدم استخدام الإجراء Append على ملف مفتوح.

4- إعادة تسمية ملف:

تمكن لغة التريبواسكال من إعادة تسمية الملفات من خلال البرنامج مباشرة، ويُستخدم من أجل ذلك الإجراء Rename. ويجب قبل ذلك ربط اسم الملف بمتحول الملف وذلك باستخدام الإجراء Assign وذلك كالتالي:

Assign (filevar, filename(oldfile));
Rename(filevar, filename(newfile));

5- مسح ملف:

يمكن أيضاً في برنامج لغة التريبواسكال القيام بعملية مسح ملف من القرص، على أن يتم ربط اسم الملف بمتحول الملف وذلك باستخدام الإجراء Assign وذلك كالتالي:

Assign (filevar, filename);
Erase(filevar);

7-1-3 توابع وعمليات أخرى تُستخدم على الملفات:

1- فحص نتائج العمليات على الملفات:

يمكن أن تحدث عند تنفيذ البرنامج بعض الأخطاء تؤدي إلى عدم اتمام العمليات على الملفات، فقد يحصل خطأ يؤدي إلى عدم قراءة الملف من القرص، أو عدم تخزين الملف على القرص. ويمكن أن يكون لذلك أسباب عديدة يمكن أن تتعلق بأسماء ملفات أو بالمساحة المخصصة للتخزين، أو لأسباب أخرى.

إن لغة التريبواسكال تجري عملية فحص لجميع استدعاءات إجراءات الدخول والخروج المستخدمة في البرنامج، ففي حالة الإجراء Rewrite مثلاً، فإنه يتم تتبع تنفيذ هذا الإجراء، فإذا كان هذا الإجراء غير قادر على إنشاء الملف فإنه سيتم توقف البرنامج وإظهار رسالة الخطأ.

إن عملية الفحص للإجراءات يمكن التحكم بها في لغة التريباسكال وبالتالي التحكم بتوجيه البرنامج قبل عملية التوقف الفجائي عند اكتشاف الخطأ.

من أجل ذلك يمكن استخدام توجيه المترجم `{I-}` والذي يعمل على عدم إنهاء البرنامج في حال عدم نجاح عملية الدخل أو الخرج فعملية التفحص التلقائية الافتراضية يتم إنهاء العمل بها هنا عند استخدام توجيه المترجم `{I-}`. ويتم هنا تتبع وتفحص عملية الدخل أو الخرج في البرنامج من خلال استخدام التابع مسبق التعريف `IOResult` والذي يعيد قيمة من النمط `Word` تمثل نتيجة آخر عملية إدخال أو إخراج، ويتمثل دور هذا التابع هنا بأنه سيعيد القيمة (0) إذا تمت عملية الإدخال أو الإخراج بنجاح، وإلا فإنه سيعيد شيفرة الخطأ.

من أجل العودة إلى وضعية الفحص الافتراضي في البرنامج يجب أن نلغي توجيه المترجم السابق باستخدام توجيه المترجم `{I+}`.

مثال (4):

لنأخذ البرنامج السابق لقراءة ملف `Read_File` ولنقم بإدخال اسم الملف المراد قراءته، ولنراقب النتيجة في حال كون اسم الملف المراد قراءته غير موجود على القرص.

لنجر تعديلًا على هذا البرنامج باستخدام توجيه المترجم السابق `{I-}` ولنراقب النتيجة عند الحالة السابقة نفسها وذلك في حال أن اسم الملف المراد قراءته غير موجود على القرص.

```
Program Test90;
var
  i,j: integer;
  filename: string[10];
  filevar:text;
```

```

x:array[1..10] of real;
BEGIN
  write('Enter filename:');
  readln(filename);
  assign(filevar,filename);
  {$I-}
  reset(filevar);
  {$I+}
  If ioresult <> 0 then
    begin
      writeln ('Error while reading file', Filename);
      halt;
    end;
    for j:=1 to 5 do
      read(filevar,x[j]);
    close (filevar);
    for j:=1 to 5 do
      write(x[j]);
    readln;
  END.

```

2- التابع Eoln:

تحدثنا سابقاً عن التابع Eof ولا حظنا أنه يحدد نهاية الملف وهو يُعطي القيمة True عندما يصل مؤشر الملف إلى نهاية الملف. التابع Eoln يشبه عمله عمل التابع Eof فهو يحدد نهاية السطر في الملف، وهو يعيد القيمة True عندما يصل مؤشر الملف إلى نهاية السطر.

3- التابعان SeekEoln ، SeekEof:

إن التابعين SeekEof و SeekEoln لهما أثرٌ مشابه للتابعين Eof و Eoln مع بعض الفوارق البسيطة، فالتابع Eof يحدد نهاية الملف ويُعيد القيمة True عندما يصل مؤشر الملف إلى نهاية الملف، أما الملف SeekEoln فإنه يحدد نهاية السطر في الملف وهو يُعيد القيمة True عندما يصل مؤشر الملف إلى نهاية السطر. الفرق بين هذين التابعين والتابعين السابقين، هو أن التابعين SeekEof و SeekEoln لا يقومان بعملية فحص لرموز التحكم في حال وجودها في نهاية الملف أو نهاية السطر، فكلما التابعين يُعيدان القيمة True حتى ولو بقيت هناك رموز تحكم أو رموز فراغ في نهاية السطر أو نهاية الملف. ولا بد من الإشارة هنا إلى أن المقصود برموز التحكم ، رموز شفرة أسكي ASCII التي تقع بين الرقم 0 والرقم 32 في جدول الأسكي.

7-2 الوحدات Units:

تملك لغة لترتوباسكال عدداً كبيراً من الإجراءات والتوابع مسبقة التعريف والتي يمكن استخدامها مباشرة في البرنامج، من هذه التوابع والإجراءات ما ذكرناه سابقاً عند الحديث عن التوابع الرياضية وتوابع السلاسل الرمزية، وكذلك هناك بعض الإجراءات مثل Read, Write . هذه التوابع والإجراءات المختلفة تكون مخزنة في مكتبة لغة الترتوباسكال TURBO.TPL.

إضافة إلى هذه التوابع والإجراءات المختلفة تملك لغة الترتوباسكال أيضاً مجموعة من الوحدات الجاهزة وهذه الوحدات موجودة أيضاً في المكتبة TURBO.TPL. من أهم هذه الوحدات نذكر :

الوحدة CRT: تحتوي هذه الوحدة على روتينات وتعريفات تتعلق بعمليات الدخل والخرج وبالتعليمات والإجراءات الأخرى التي تتعامل مع الشاشة مثل تحديد أبعاد

الشاشة، وتحديد لون الكتابة ولون الخلفية، والتعامل مع جزء من الشاشة (نافذة) وغيرها من الإجراءات والتعريفات الأخرى.

الوحدة DOS: تحتوي هذه الوحدة على روتينات وتعريفات للتعامل مع نظام التشغيل Dos، مثل التعامل مع الملفات، والحصول على معلومات عن نظام التشغيل مثل الحصول على التاريخ والوقت، أو بعرض محتويات الأدلة والحصول على خدمات نظام التشغيل المختلفة.

الوحدة GRAPH: تحتوي هذه الوحدة على روتينات وتعريفات للتعامل مع شاشة

العرض البياني وستتعرف لاحقاً عند الحديث عن الرسم البياني إلى بعض هذه الروتينات والتعريفات.

الوحدة PRINTER : تحتوي هذه الوحدة على روتينات وتعريفات معدة للتعامل مع الطابعة، فهي تُستخدم من أجل إخراج النتائج على الطابعة وذلك من خلال استخدام الإجراءات Write و WriteLn.

بالإضافة إلى هذه التتابع والإجراءات والوحدات الجاهزة، يمكن من خلال لغة التريباسكال إنشاء وحدات مختلفة وتضمينها في البرنامج، ويمكن أن يوضع في الوحدة جزء من برنامج أو برنامج كامل، هذا الشيء يمكن أن يجنبنا مشكلة الذاكرة المحدودة في لغة التريباسكال كما أن وقت الترجمة للبرنامج تصبح أقصر، هذا إضافة إلى أنه يمكن استخدام الوحدات من برامج أخرى.

7-2-1 بنية الوحدات:

الوحدة Unit هي عبارة عن ناتج تجميع من ثوابت، متحولات، إجراءات، توابع، وهي تكون شبيهة تماماً ببرنامج التريباسكال، فالوحدة تحتوي مثلاً على جزء خاص

بالتعليمات ويمكن في الوحدة وضع جزء من برنامج ما، ويمكن تنفيذ عملية ترجمة Compile خاصة بها.

تتكون الوحدة من قسمين رئيسيين:

قسم الاتصال الخارجي ((Interface)): في هذا الجزء يتم الإعلان عن النماذج، الثوابت، المتحولات، التوابع والإجراءات، عملية التعريف أو الإعلان عن هذه المعطيات تتم هنا حسب القواعد المعتمدة في البرامج العادية. أما التوابع والإجراءات فيتم في هذا الجزء تعريفها فقط ، أما تنفيذها فسيتم في الجزء التالي.

عند استخدام وحدة أخرى ضمن هذه الوحدة فإنه يجب الإعلان عنها في بداية هذا الجزء وذلك بعد عبارة USES.

قسم التنفيذ ((Implementation)):

المعطيات التي يُعلن عنها في هذا الجزء لا تكون معروفة بالنسبة للبرنامج الأساسي. التوابع والإجراءات التي تُعرف في قسم الاتصال الخارجي يتم تنفيذها في هذا القسم. بالإضافة إلى ذلك يحتوي هذا القسم على ما يسمى بكود التهيئة Initialization Code وهي تقوم بتهيئة المتحولات التي تحتاجها أعمال الوحدة. وهذا الجزء ليس من الضروري وجوده، وفي حالة وجوده فإنه يأتي في نهاية الوحدة ويُسبق بعبارة Begin ، إذاً في هذا الجزء يتم إعداد ترتيبات محددة تكون ضرورية لعمل الوحدة.

7-2-2 إنشاء الوحدات:

عند إنشاء الوحدة يجب اتباع التكوين الآتي:

< أسم الوحدة> Unit

INTERFACE

<قائمة الوحدات المستخدمة ضمن هذه الوحدة> USES

--

--

IMPLEMENTATION

تعريف الإجراءات والتوابع

{Initialisation Part}

BEGIN

--

--

END.

ويجب الإشارة هنا إلى أن اسم الوحدة المكونة تُخزن على القرص بنفس طريقة تخزين البرنامج وتكون لاحقاً أيضاً نفس لاحقة البرنامج، ويجب أن تُخزن بنفس اسم الوحدة المستخدم في الوحدة و عند استدعاء هذه الوحدة من البرنامج أن تُستدعى بنفس الاسم الذي خُزنت به.

في حال تخزين الوحدة باسم ملف مختلف عن الاسم المستخدم ضمن الوحدة، كأن يكون اسم الوحدة مثلاً (FirstUnit) واسم الملف الذي خُزنت فيه هذه الوحدة هو (Unit1) ، فإن عملية الاستدعاء في هذه الحالة تجري على الشكل الآتي:

Uses {\$U Unit1} FirstUnit;

مثال (5):

نريد حساب القيمة العظمى والقيمة الصغرى من أجل قيمتين فقط. من أجل ذلك نحتاج إلى تابعين جزئيين Function، أحدهما لحساب القيمة الصغرى Min والآخر لحساب القيمة العظمى Max، وسيتم من أجل ذلك إنشاء وحدة Unit تتضمن هذين التابعين، وسنسمي هذه الوحدة MyF_Unit:

Unit MyF_Unit;

Interface

Function Min(A,B:Integer):Integer;

Function Max(A,B:Integer):Integer;

Implementation

Function Min;

begin

if A<=B Then

Min:=A

Else

Min:=B;

end;

Function Max;

begin

if A>=B Then

Max:=A

Else

Max:=B;

end;

{No Initialisation part}

End.

من أجل استخدام هذه الوحدة في البرنامج نقوم بكتابة البرنامج الآتي:

Program Test91;

uses Crt,MyF_Unit;

var

A: Integer;

B: Integer;

BEGIN

Clrscr;

GoToXY(1,5);

Write('A=');

Readln(A);


```

Write('B= ');
Readln(B);
Writeln('Max=',Max(A,B):6);
Writeln('Min=',Min(A,B):6);
readln;
END.

```

أمثلة متنوعة:

مثال (1):

البرنامج التالي يرسم موجة جيبية:

```

Program Test92;
uses graph;
var
  i,j,driver,mode,y,x0,y0,f:integer;
  x:real;
  filename:string[10];
  filev:text;
BEGIN
  driver:=detect;
  initgraph(driver,mode,'d:\pascal7');
  x0:=10;y0:=100;
  for i:=1 to 1000 do
    begin
      x:=sin(2*3.1415*1000*i)*50;
      y:=round(x);
      line(x0,y0,i*4+10,y+100);
      x0:=i*4+10;y0:=y+100;
    end;
  readln;

```

```
closegraph;  
END.
```

مثال (2):

في البرنامج التالي يتم حساب تابع الارتباط المتصالب لإشارتين:

```
Program Test93;  
uses Crt;  
var  
  x,y,z:array[1..10] of integer;  
  i,j,m,Nx,Ny,n1,n2:integer;  
BEGIN  
  y[1]:=1;y[2]:=2;y[3]:=3; }  
  writeln('Enter The Number of Samples of the first Signal=');  
  Readln(n1);  
  writeln('Enter The value of the first Signal X:');  
  for i:=1 to n1 do  
    begin  
      writeln('X['i,']=');  
      Readln (X[i]);  
    end;  
  writeln('Enter The Number of Samples of the first Signal=');  
  Readln(n2);  
  writeln('Enter The value of the second Signal Y:');  
  for i:=1 to n2 do  
    begin  
      writeln('Y['i,']=');  
      Readln (Y[i]);  
    end;  
  writeln('Please correlate the Signal X :');  
  textColor (4);  
  writeln( ' x[1]:=2;x[2]:=1;x[3]:=1 ');  
  writeln('with the signal Y :') ;
```

```

textColor (3);
writeln(' y[1]:=1;y[2]:=2;y[3]:=3');
for i:=1 to n1+n2-1 do
  begin
    z[i]:=0;
    for j:=1 to i do
      begin
        m:=n2+1;
        z[i]:=z[i]+x[i-j+1]*y[m-j];
      end;
      writeln('z[' ,i ,']=' , z[i]);
    end;
  writeln('-----');
  readln;
END.

```

مثال (3):

في البرنامج التالي يتم ترتيب مجموعة من الأعداد تصاعدياً:

```

Program Test94;
uses Crt;
var
  n,i,k,b: Integer;
  a: Array[1..20] of integer;
BEGIN
  Clrscr;
  Writeln;writeln;
  Write (Number of Numbers to Sort=');
  Readln(n);
  Writeln;writeln;
  for i:=1 to n do
    Begin
      write('Please Enter The Number Nr.',i );

```

```

        readln(a[i]);
    end;
    for i:=1 to n-1 do
        begin
            for k:=1 to n-1 do
                begin
                    If a[i]>a[i+k] then
                        begin
                            b:=a[i+k];
                            a[i+k]:=a[i];
                            a[i]:=b;
                        end;
                end;
            end;
        end;
        Writeln;writeln;
        writeln(' The Order of TheNumber:');
        writeln;
        for i:=1 to n do
            writeln('a(',i,')=',a[i];
    END.

```

مثال (4):

يتم في هذا البرنامج قراءة التاريخ من الحاسوب:

```

Program Test95;
uses Dos;
Type weekdays = (monday, tuesday, wednesday, thursday,
    friday, saturday, sonday);
    datew = object
        day : 1..31;
        month : 1..12;
        year : 0..2000;
        day_of_the_week : weekdays;

```

```

        procedure init;
    end;
    Procedure datew.init;
    var
        d, m, y, w : word;
    begin
        getdate(y, m, d, w);
        day := d;
        month := m;
        year := y;
        day_of_the_week := weekdays(w);
    end;
    var datum : datew;
    BEGIN
        datum.init;
        with datum do
            writeln(day, '.', month, '.', year);
        END.

```

مثال (5)

يتم في البرنامج التالي حساب $N!$:

```

Program Test96;
    Function fak(n: integer) : integer;
    begin
        IF n = 1 THEN
            fak:= 1
        ELSE
            fak := n * fak(n-1);
        end;
    var
        m, n : integer;

```

```

BEGIN
  for n:=1 to 5 do
    begin
      m := fak(n);
      writeln('n = ', n:2, ' n! = ', m:10);
    end;
  readln;
END.

```

مثال (6):

يتم في هذا البرنامج رفع عدد ما إلى قوة:

```

Program Test97;
  var number, exp: integer;
      result: LongInt;
  Procedure Pot(Basis, Exp : Integer; Var Result :
LongInt);
  var i : Integer;
  begin
    result := 1;
    for i := 1 To exp do
      result:= result* basis;
    end;
BEGIN
  Repeat
    Write ('Please Enter The Number and the Exponent : ');
    Readln (Number, Exp);
    Pot(Number, Exp, Result);
    If Not ((Number= 0) And (Exp = 0)) Then
      Writeln (Number , ' high ', Exp, ' = ', Result);
  Until (Number = 0) And (Exp = 0);
END.

```

مثال (7):

البرنامج التالي يقوم بعملية جمع مصفوفتين:

Program Test98;

Type

matrix = array [1..4, 1..5] of Integer;

var

matrix1, matrix2, sum : matrix;

Procedure InputMatrix (start : Integer; var operand :
matrix);

var

index1, index2 : Integer;

begin

for index2 := 1 to 5 do

begin

For index1 := 1 to 4 do

operand[index1, index2] := start * index1;

start := start + 1;

end;

end;

Procedure OutputMatrix (operand : matrix);

var

index1, index2 : Integer;

begin

for index2 := 1 to 5 do

begin

for index1 := 1 to 4 do

write (operand[index1, index2] : 5);

writeln;

end;

writeln;

```

        writeln;
    end;
Procedure AddMatrix (operand1, operand2 : matrix;
        var result : matrix);
var
    index1, index2 : Integer;
begin
    for index2 := 1 to 5 do
        for index1 := 1 to 4 do
            result[index1, index2] := operand1[index1, index2]
                + operand2[index1, index2];
        end;
    BEGIN
        InputMatrix (5, matrix1);
        OutputMatrix (matrix1);
        InputMatrix (11, matrix2);
        OutputMatrix (matrix2);
        AddMatrix (matrix1, matrix2, sum);
        OutputMatrix (sum);
        readln;
    END.

```

مثال (8):

البرنامج التالي يمثل قاعدة بيانات لموظفين، تتضمن جميع المعلومات المتعلقة بالموظف:

```

Program Test99;
Const
    FileName = 'payroll.txt';
    TempFile = 'temp.txt';
    Header = '----- Main Menu -----';
    Header1 = '----- Employee DataBase -----';
    Header2 = '----- Employee Record -----';

```


Separator = '-----';

Type

EmployeeRecord = RECORD

ID :STRING[5];

Using Variant Records 223

Name, Position :STRING[20];

SSN :STRING[11];

CASE Category :CHAR OF

'1' : (MonthlySalary :REAL);

'2' : (HourlyRate :REAL);

'3' : (Commission,

BasicSalary :REAL;

Area :STRING[20])

End;

SSNstring = STRING[11];

Var

NewFile, PayrollFile :TEXT;

EmployeeRec :EmployeeRecord;

Title :ARRAY [1..9] OF STRING[20];

OneLine :STRING[80];

{ ----- Procedure SearchRec ----- }

Procedure SearchRec(var PayrollFile :Text;

Employee :EmployeeRecord;

SSNstring :SSNstring;

Var

Found :Integer);

Begin

Found := 0;

assign(PayrollFile, FileName);

reset(PayrollFile);

while not eof (PayrollFile) do

begin

```

With Employee do
begin
    readln(PayrollFile, ID);
    readln(PayrollFile, Name);
    readln(PayrollFile, Position);
    readln(PayrollFile, SSN);
    readln(PayrollFile, Category);
    case Category of
        '1' : readln(PayrollFile, MonthlySalary);
        '2' : readln(PayrollFile, HourlyRate);
        '3' : begin
            readln(PayrollFile, Commission);
            readln(PayrollFile, BasicSalary);
            readln(PayrollFile, Area)
        end
    end; { End of CASE structure }
    if SSNumber = SSN then
        found := 1;
    end { End of WITH block }
end;
close(PayrollFile);
End;
{ ----- Procedure ReadRec ----- }
Procedure ReadRec(Var PayrollFile :TEXT;
Employee :EmployeeRecord);
Var
SSNumber :String[11];
Found :Integer;
begin
    Writeln;
    Write('Please enter the SSN of the employee: ');
    Readln(SSNumber);

```

```

SearchRec(PayrollFile, EmployeeRec, SSNumber, Found);
If Found =1 Then
    begin
        Assign(PayrollFile, FileName);
        Reset(PayrollFile);
        While Not Eof (PayrollFile) Do
            begin
                With Employee Do
                    begin
                        Readln(PayrollFile, ID);
                        Readln(PayrollFile, Name);
                        Readln(PayrollFile, Position);
                        Readln(PayrollFile, SSN);
                        Readln(PayrollFile, Category);
                        Case Category Of
                            '1':Readln(PayrollFile, MonthlySalary);
                            '2': Readln(PayrollFile, HourlyRate);
                            '3': begin
                                Readln(PayrollFile, Commission);
                                Readln(PayrollFile, BasicSalary);
                                Readln(PayrollFile, Area)
                            end
                        end; { End of CASE structure }
                    end
                If SSNumber = SSN Then
                    begin
                        Writeln(Header2);
                        Writeln(Title[1],ID);
                        Writeln(Title[2],Name);
                        Writeln(Title[3],Position);
                        Writeln(Title[4], SSN);
                        Case Category Of
                            '1' : Writeln(Title[5], MonthlySalary:0:2);

```

```

'2' : Writeln(Title[6], HourlyRate:0:2);
'3' : begin
        Writeln(Title[7], Commission:0:2);
        Writeln(Title[8], BasicSalary:0:2);
        Writeln(Title[9], Area)
    end
end; { End of CASE structure }
end
end { End of WITH block }
end;
Close(PayrollFile)
end
Else { If not found }
begin
    Writeln('SSN not found in file. ');
    Writeln('Please try again. ');
    Writeln
end
end;
{ ----- Procedure DelRec ----- }
Procedure DelRec(VAR NewFile, PayrollFile :Text;
Employee :EmployeeRecord);
Var
SSNumber :String[11];
Found :Integer;
Begin
    Write('Please enter the SSN of the employee to be deleted:
        ');
    Readln(SSNumber);
    SearchRec(PayrollFile, EmployeeRec, SSNumber, Found);
    If Found =1 Then
Begin

```

```

Assign (NewFile, TempFile);
Rewrite(NewFile);
Assign(PayrollFile, FileName);
Reset(PayrollFile);
While Not Eof (PayrollFile) Do
  Begin
    With Employee Do
      Begin
        Readln(PayrollFile, ID);
        Readln(PayrollFile, Name);
        Readln(PayrollFile, Position);
        Readln(PayrollFile, SSN);
        Readln(PayrollFile, Category);
        Case Category Of
          '1' : Readln(PayrollFile, MonthlySalary);
          '2' : Readln(PayrollFile, HourlyRate);
          '3' : begin
              Readln(PayrollFile, Commission);
              Readln(PayrollFile, BasicSalary);
              Readln(PayrollFile, Area)
            End
          End; { End of CASE structure }
        If SSNNumber <> SSN Then
          Begin
            Writeln(NewFile, ID);
            Writeln(NewFile, Name);
            Writeln(NewFile, Position);
            Writeln(NewFile, SSN);
            Writeln(NewFile, Category);
            Case Category Of
              '1' : Writeln(NewFile, MonthlySalary:0:2);
              '2' : Writeln(NewFile, HourlyRate:0:2);

```

```

    '3' : Begin
        Writeln(NewFile, Commission:0:2);
        Writeln(NewFile, BasicSalary:0:2);
        Writeln(NewFile, Area)
        End
    End; { End of CASE structure }
End;
End { End of WITH block }
End; {End of DO }
Close(NewFile);
Close(PayrollFile);
{ Copy NewFile back to Payroll File }
Assign(PayrollFile, FileName);
Rewrite(PayrollFile);
Assign(NewFile, TempFile);
Reset(NewFile);
While Not Eof (NewFile) Do
    Begin
        Readln(NewFile, OneLine);
        Writeln(PayrollFile, OneLine)
    End;
    Close(NewFile);
    Erase(NewFile); { Erase the temporary file }
    Close(PayrollFile);
    { User Messages }
    Writeln('The employee ', SSNumber,
        ' is removed from file.')
    End { End of the "IF Found.." block }
    Else { IF not found }
        Begin
            Writeln('The SSN ', SSNumber, ' is not found. ');
            Writeln('Check the number and try again. ');

```

```

        Writeln
    End
End;
{ ----- Procedure AddRec ----- }
Procedure AddRec(VAR NewFile, PayrollFile :Text;
Employee: EmployeeRecord);
Begin
    Assign(PayrollFile, FileName);
    Reset(PayrollFile);
    Assign(NewFile, TempFile);
    Rewrite(NewFile);
    While Not Eof(PayrollFile) Do
        Begin
            { Copy each record from PayrollFile to the NewFile }
            Readln(PayrollFile, OneLine);
            Writeln(NewFile, OneLine)
        End;
    { Accept a new record from the keyboard }
    With Employee Do
        Begin
            Write('Please enter Employee ID: ');
            Readln(ID);
            Write('Name: '); Readln(Name);
            Write('Position: '); Readln(Position);
            Write('SSN (xxx-xx-xxxx): '); Readln(SSN);
            Write('Payroll category: '); Readln(Category);
            Case Category Of
                '1' : Begin
                    Write('Monthly Salary: ');
                    Readln(MonthlySalary)
                End;
                '2' : Begin

```

```

        Write('Rate: ');
        Readln(HourlyRate)
    End;
'3' : Begin
        Write('Commission: ');
        Readln(Commission);
        Write('Basic salary: ');
        Readln(BasicSalary);
        Write('Area: ');
        Readln(Area)
    End
End;
{ Store the information in NewFile }
Writeln(NewFile, ID);
Writeln(NewFile, Name);
Writeln(NewFile, Position);
Writeln(NewFile, SSN);
Writeln(NewFile, Category);
Case Category Of
    '1' : Writeln(NewFile, MonthlySalary:0:2);
    '2' : Writeln(NewFile, HourlyRate:0:2);
    '3' : Begin
        Writeln(NewFile, Commission:0:2);
        Writeln(NewFile, BasicSalary:0:2);
        Writeln(NewFile, Area)
    End
End
End;
Close(NewFile);
Close(PayrollFile);
{ Copy NewFile back to Payroll File }
Assign(PayrollFile, FileName);

```



```

Rewrite(PayrollFile);
Assign(NewFile, TempFile);
Reset(NewFile);
While Not Eof (NewFile) Do
    Begin
        Readln(NewFile, OneLine);
        Writeln(PayrollFile, OneLine)
    End;
Close(NewFile);
Erase(NewFile); { Erase the temporary file }
Close(PayrollFile)
End;
{ ----- Procedure UpdateRec ----- }
Procedure UpdateRec(VAR NewFile, PayrollFile :Text;
Employee :EmployeeRecord);
Var
SSNumber :String[11];
Found :Integer;
Begin
    Write('Please enter the SSN of the employee to be updated:
        ');
    Readln(SSNumber);
    SearchRec(PayrollFile, EmployeeRec, SSNumber, Found);
    If Found =1 Then
        Begin
            Assign(PayrollFile, FileName);
            Reset(PayrollFile);
            Assign(NewFile, TempFile);
            Rewrite(NewFile);
            While Not Eof(PayrollFile) Do
                Begin
                    With Employee Do

```

```

Begin
  Readln(PayrollFile, ID);
  Readln(PayrollFile, Name);
  Readln(PayrollFile, Position);
  Readln(PayrollFile, SSN);
  Readln(PayrollFile, Category);
  Case Category Of
    '1' : Readln(PayrollFile, MonthlySalary);
    '2' : Readln(PayrollFile, HourlyRate);
    '3' : Begin
      Readln(PayrollFile, Commission);
      Readln(PayrollFile, BasicSalary);
      Readln(PayrollFile, Area)
    End
  End; { End of CASE structure }
If SSNumber <> SSN Then
  Begin
    Writeln(NewFile, ID);
    Writeln(NewFile, Name);
    Writeln(NewFile, Position);
    Writeln(NewFile, SSN);
    Writeln(NewFile, Category);
    Case Category Of
      '1' : Writeln(NewFile, MonthlySalary:0:2);
      '2' : Writeln(NewFile, HourlyRate:0:2);
      '3' : Begin
        Writeln(NewFile, Commission:0:2);
        Writeln(NewFile, BasicSalary:0:2);
        Writeln(NewFile, Area)
      End
    End { End of CASE structure }
  End { End of IF block }

```

Else

Begin

Writeln('Please enter the updated information:');

Write('ID: '); Readln(ID);

Writeln(NewFile, ID);

Write('Name: '); Readln(Name);

Writeln(NewFile, Name);

Write('Position: '); READLN(Position);

Writeln(NewFile, Position);

Writeln (NewFile, SSN);

Write('Category: '); READLN(Category);

Writeln(NewFile, Category);

Case Category Of

'1' : Begin

Write('Salary: ');

Readln(MonthlySalary);

Writeln(NewFile, MonthlySalary:0:2)

End;

'2' : Begin

Write('Hourly Rate: ');

Readln(HourlyRate);

Writeln(NewFile, HourlyRate:0:2)

End;

'3' : Begin

Write('Commission: ');

Readln(Commission);

Writeln(NewFile, Commission:0:2);

Write('Basic Salary: ');

Readln(BasicSalary);

Writeln(NewFile, BasicSalary:0:2);

Write('Area: ');

Readln(Area);

```

        Writeln(NewFile, Area)
    End
End { End of CASE structure }
End { End of ELSE block }
End { End of WITH block }
End; { End of DO }
CLOSE(NewFile);
CLOSE(PayrollFile);
{ Copy NewFile back to Payroll File }
Assign(PayrollFile, FileName);
Rewrite(PayrollFile);
Assign(NewFile, TempFile);
Reset(NewFile);
While Not Eof (NewFile) Do
    Begin
        Readln(NewFile, OneLine);
        Writeln(PayrollFile, OneLine)
    End;
Close(NewFile);
Erase(NewFile); { Erase the temporary file }
Close(PayrollFile);
{ User Messages }
Writeln('The employee ', SSNumber, ' is updated.')
End { End of IF block }
Else
    Begin
        Writeln('The SSN ', SSNumber, ' is not found. ');
        Writeln('Check the number and try again. ');
        Writeln
    End
End;
{ ----- Procedure Menu ----- }

```

Procedure Menu;

Var

Option :Integer;

Begin

Writeln(Header);

Writeln;

Writeln('1. Display an employee record.');

Writeln('2. Add a new employee.');

Writeln('3. Delete an employee.');

Writeln('4. Update an employee record.');

Writeln('5. Exit.');

Writeln(Separator);

Write('Make a choice and press a number: ');

Readln(Option);

Case Option Of

1 : ReadRec(PayrollFile, EmployeeRec);

2 : AddRec(NewFile, PayrollFile, EmployeeRec);

3 : DelRec(NewFile, PayrollFile, EmployeeRec);

4 : UpdateRec(NewFile, PayrollFile, EmployeeRec);

5 : Exit

End;

Menu

End;

{ ----- Main Program ----- }

BEGIN

{ Assign titles }

Title[1] := 'ID: ';

Title[2] := 'Name: ';

Title[3] := 'Position: ';

Title[4] := 'SSN: ';

Title[5] := 'Salary: ';

Title[6] := 'Rate: ';

```
Title[7] := 'Commission: ';
Title[8] := 'Basic Salary: ';
Title[9] := 'Area: ';
Menu
END.
```

عند تنفيذ البرنامج سيتم إظهار خمسة خيارات على الشاشة كالتالي:

----- Main Menu -----

1. Display an employee record.
 2. Add a new employee
 3. Delete an employee
 4. Update an employee record
 5. Exit.
-

في الخيار الأول يتم عرض البيانات لموظف معين عن طريق إدخال رقم خاص بالموظف، وفي الخيار الثاني يتم إضافة بيانات لموظف جديد ، في الخيار الثالث يتم مسح البيانات الخاصة بموظف، وفي الخيار الرابع يتم تحديث بيانات موظف. في جميع الحالات السابقة يتم التعامل مع ملف موجود على القرص ، ويتضمن هذا الملف ثلاث سجلات لثلاث فئات مختلفة من الموظفين. الخيار الخامس هو للخروج من البرنامج.

مثال (9):

البرنامج التالي يرسم ويملى جزء من دائرة:

```
Program Test100;
uses
  Crt, Graph;
var
  Driver, Mode: Integer;
```

BEGIN

```
Driver:= Detect;  
InitGraph(Driver, Mode, "");  
PieSlice(70,70,0,45,55);  
readln;  
setcolor (red);  
PieSlice(70,70,55,145,55);  
readln;  
setcolor (green);  
PieSlice(70,70,155,245,55);  
readln;  
setcolor (blue);  
PieSlice(70,70,155,245,55);  
readln;  
CloseGraph;  
END.
```

مثال (10):

في البرنامج التالي يتم تحديد مرتبة (Grad) الطالب وذلك استناداً لدرجته (معدله) (Score) وذلك وفق التصنيف الآتي:

- المرتبة A للدرجة من المعدل من 90% - 100% .
- المرتبة B للدرجة من المعدل من 80% - 89% .
- المرتبة C للدرجة من المعدل من 70% - 79% .
- المرتبة D للدرجة من المعدل من 60% - 69% .
- المرتبة F للدرجة من المعدل أقل من 60% .

Program Test101;

```

uses crt;
var
  Score :integer;;
BEGIN
  textcolor(4);
  write('Please enter the score: ');
  readln(Score);
  writeln;
  textcolor(3);
  { Beginning of the IF construct }
  { ----- }
  if Score > 59 then
  if Score > 69 then
  if Score > 79 then
  if Score > 89 then
    writeln ('Excellent. Your grade is "A"')
  else
    writeln('Very good. Your grade is "B"')
  else
    writeln ('Good. Your grade is "C"')
  else
    writeln ('Passed. Your grade is "D"')
  else
    writeln ('Better luck next time. Your grade is "F"');
  { End of the IF construct }
  { ----- }
  writeln ('Press ENTER to continue..');
  readln;
END.

```

مثال (11):

يبين البرنامج السابق نفسه ولكن تمت كتابته بطريقة أخرى:

Program Test1102;

var

Score :integer;

A, B, C, D, F :boolean;

BEGIN

write ('Please enter the score: ');

readln(Score);

A := (Score >= 90) and (Score <= 100);

B := (Score >= 80) and (Score < 90);

C := (Score >= 70) and (Score < 80);

D := (Score >= 60) and (Score < 70);

F := (Score < 60) and (Score >= 0);

writeln;

{ Beginning of the IF construct }

{ ----- }

If A then

writeln('Excellent. Your grade is "A"')

else if B then

writeln('Very good. Your grade is "B"')

else if C then

writeln ('Good. Your grade is "C"')

else if D then

writeln ('Passed. Your grade is "D"')

else if F then

writeln ('Better luck next time. Your grade is "F"')

else

writeln ('This number is out of range.');

{ End of the IF construct }

{ ----- }

writeln ('Press ENTER to continue..');

readln

END.

مثال (12):

في هذا البرنامج يتم حساب عدد أيام الشهر من أجل كل سنة يتم إدخالها في البرنامج،
يمكن اختبار البرنامج من أجل شهر شباط:

Program Test103;

var

Days, Month, Year :integer;

BEGIN

write('Please enter the number of the month: ');

readln(Month);

Case Month of

1,3,5,7,8,10,12 : Days := 31;

4,6,9,11 : Days := 30;

2 : begin

write('Enter the year:');

readln(Year);

if year mod 4 = 0 then

days :=29

else

days :=28

end;

end;

writeln('There are ',Days,' days in this month.');

writeln('Press ENTER to continue..');

readln
END.

الملحق (أ)

بعض الإجراءات والتوابع المستخدمة في لغة التريباسكال:

سنعرض في هذا الملحق بعض التوابع والإجراءات الهامة المتوفرة في لغة التريباسكال، هذه الإجراءات والتوابع منها ما هو متوفر في مكتبة اللغة التنفيذية ومنها ما هو متوفر في وحدات لغة التريباسكال. وسنقوم بتصنيف هذه التوابع والإجراءات حسب مكان وجودها:

1- بعض التوابع والإجراءات المعرفة في المكتبة التنفيذية:

الوظيفة	الإجراء أو التابع
يعطي القيمة المطلقة لقيمة متحوّله الوسيط.	Abs
يفتح هذا الإجراء ملفاً نصياً من أجل إضافة البيانات إلى نهاية الملف.	Append
يعطي ظل الزاوية مقدراً بالراديان.	ArcTan
يستخدم للتعامل مع الملفات وهو يقوم بربط أسم الملف الخارجي (الأسم المعرف من قبل نظام التشغيل) بمتحول	Assign

	الملف.
Break	يُستخدم لإنهاء تنفيذ الحلقة.
Chr	تستخدم على الأعداد الصحيحة من (0-255) وتعطي ما يقابل هذه الأعداد من جدول ASCII.
ChDir	يغير الدليل الحالي إلى دليل آخر محدد في المتحول الوسيطي.
Close	يُستخدم لإغلاق ملف مفتوح بإحدى الإجراءات Reset، Append، Rewrite.
Concat	يُستخدم لوصل عدة سلاسل محرفية مع بعضها البعض.
Copy	يقوم هذا الإجراء باقتطاع سلسلة محرفية جزئية من سلسلة أخرى.
Cos	يعطي جيب الزاوية بالراديان.
Dec	يُنقص قيمة المتحول الوسيطي بمقدار قيمة المتحول الوسيطي الثاني، وفي حال عدم وجود المتحول الوسيطي الثاني فإن قيمة المتحول الوسيطي الأول تنقص بمقدار (1).
Delete	يقوم بمسح سلسلة محرفية جزئية من سلسلة محرفية أخرى.
Eof	يُستخدم في الملفات وهو يعيد القيمة True عندما يصل مؤشر الملف إلى نهاية الملف، ما عدا ذلك فإنه يعيد القيمة False.
Eoln	يعيد القيمة True عندما يصل مؤشر إلى نهاية السطر ، ما عدا ذلك فإنه يعيد القيمة False.

Erase	يمسح ملف موجود على القرص.
Exit	ينهي تنفيذ المقطع الحالي.
Exp	يعطينا رفع الأساس e إلى القوة المحددة.
FilePos	يعطي الموقع الحالي لمؤشر الملف ، والملف يجب أن يكون غير نصي ويجب أن يكون مفتوحاً.
FileSize	يعطي حجم الملف المحدد ويجب أن لا يكون الملف نصياً.
Frac	يعطي الجزء الكسري من القيمة الحقيقية.
Halt	يوقف التنفيذ في البرنامج وينقل التحكم إلى نظام التشغيل.
Inc	يزيد قيمة المتحول الوسيطي الأول بمقدار قيمة المتحول الوسيطي الثاني، وفي حال عدم وجود المتحول الوسيطي الثاني، يُزيد قيمة المتحول الوسيطي الأول بمقدار (1).
Insert	يقوم بحشر سلسلة محرفية ضمن سلسلة أخرى وذلك بدءاً من موقع محدد.
Int	يعطي القيمة الصحيحة من القيمة الحقيقية للمتحول الوسيطي.
IOResult	يعطي معلومات عن آخر عملية إدخال وإخراج.
Length	يعطي طول سلسلة محرفية.
Ln	يعطي اللوغارتم الطبيعي لقيمة المتحول الوسيطي.
MkDir	يقوم بإنشاء دليل على القرص.
Odd	يعطي القيمة True إذا كانت قيمة المتحول الوسيطي فردية،

	ما عدا ذلك يعطي القيمة False .
Ord	وهي يعطي الرقم المقابل للمحرف في جدول ASCII، وكذلك يُستخدم لتحديد موقع الجملة المعرفة في العبارة Type .
Pi	يعطي القيمة π .
Pos	يعطي موقع سلسلة محرفية ضمن سلسلة محرفية أخرى.
Pred	يعطي القيمة السابقة لقيمة المتحول الوسيط.
Read	يقرأ قيمة من لوحة المفاتيح أو من ملف ويُسند القيمة إلى متحول مع بقاء مؤشر القراءة في نفس السطر.
Readln	يقرأ قيمة من لوحة المفاتيح أو من ملف ويُسند القيمة إلى متحول مع انتقال مؤشر القراءة إلى السطر التالي.
Rename	يُستخدم لإعادة تسمية ملف على القرص.
Reset	يفتح ملف موجود على القرص من أجل القراءة.
Rewrite	ينشئ ويفتح ملفاً جديداً للكتابة.
Rmdir	يحذف دليل فارغ موجود على القرص.
Round	يقرب قيمة المتحول الوسيط الحقيقي إلى أقرب عدد صحيح
Sin	يعطي جيب الزاوية بالراديان.
Sqr	يعطي مربع المتحول الوسيط.
Sqrt	يعطي الجذر التربيعي للمتحول الوسيط.
Str	يحول قيمة عددية إلى قيمة محرفية.

Succ	يعطي القيمة اللاحقة (التالية في الترتيب) لقيمة المتحول الوسيطي.
Trunc	يقتطع الجزء العشري من المتحول الوسيطي الحقيقي.
UpCase	يحول محارف المتحول الوسيطي الحرفي إلى حروف كبيرة.
Val	يحول سلسلة محرفية إلى عددية.
Write	يكتب قيمة المتحول في ملف نصي أو يخرجها على جهاز إخراج (شاشة، طابعة،...) مع بقاء المؤشر في نفس السطر.
Writeln	يكتب قيمة المتحول في ملف نصي أو يخرجها على جهاز إخراج (شاشة، طابعة،...) مع انتقال المؤشر إلى السطر التالي.

2- بعض الإجراءات المتضمنة في الوحدة Crt:

الإجراء أو التابع	الوظيفة
ClrEol	يحذف الرموز الموجودة على الشاشة بدءاً من موقع المؤشر وحتى نهاية السطر.
ClrScr	يمسح الشاشة أو النافذة المفعلة وينقل المشريرة إلى الزاوية العليا اليسارية من الشاشة.
Delay	يجعل البرنامج ينتظر فترة محددة من الزمن تتعلق بالمتحول الوسيطي، والفترة مقدرة بالميلي ثانية.

DelLine	يحذف السطر الحالي.
GotoXY	ينقل المؤشر إلى موقع محدد على الشاشة يتعلق بقيم المتحولات الوسيطة.
HighVideo	يختار نمط الشدة العالية (الكثافة) للعرض على الشاشة.
InsLine	يحشر سطر في موقع وجود المؤشر ويُحرك الأسطر بدءاً من موقع المؤشر إلى الأسفل.
KeyPressed	يعطي القيمة True في حالة الضغط على أي مفتاح، ما عدا ذلك يعطي القيمة False.
LowVideo	يختار نمط الشدة المنخفضة (الكثافة) للعرض على الشاشة.
NormVideo	يختار نمط الشدة العادي (الكثافة) للعرض على الشاشة.
NoSound	يوقف الصوت الذي يصدر عن مولد الأصوات الداخلي في الحاسوب.
ReadKey	يقرأ رمز من لوحة المفاتيح.
Sound	يصدر صوتاً من مولد الأصوات الداخلي في الحاسوب وذلك وفق تردد معين.

TextBackGround	يُحدد لون الخلفية للنافذة في حالة النمط النصي.
TextColor	يُحدد لون النص في حالة النمط النصي.
TextMode	يُحدد نمط النص المراد استخدامه.
WhereX	يعطي قيمة الإحداثي X لموقع المؤشيرة الحالي ضمن النافذة المفعلة.
WhereY	يعطي قيمة الإحداثي Y لموقع المؤشيرة الحالي ضمن النافذة المفعلة.
Window	يعرف نافذة نصية على الشاشة تُحدد إحداثياتها من خلال المتحولات الوسيطة.

3- بعض الإجراءات المتضمنة في الوحدة Graph:

الوظيفة	الإجراء أو التابع
يرسم قوساً من دائرة، كما يرسم قوس بين زاويتين.	Arc
يرسم مستطيلاً مملوءاً تحدد إحداثياته ولون الملىء من خلال المتحولات الوسيطة.	Bar
يرسم مستطيلاً ثلاثي الأبعاد.	Bar3D
يرسم دائرة يحدد مركزها ونصف قطرها من خلال المتحولات الوسيطة.	Circle

ClearDevice	يمسح الشاشة الرسومية وينقل المؤشرة إلى الزاوية العليا اليسارية من النافذة.
CloseGraph	ينهي الشاشة الرسومية ويعيد العمل إلى الشاشة النصية.
DetectGraph	يفحص عتاد الحاسوب لتحديد سواقة وخط الرسومات المراد استخدامه.
DrawPoly	يرسم مضلع من خلال تحديد عدة نقاط فيه.
Ellipse	يُستخدم لرسم قطع ناقص، تُحدد إحداثياته وأبعاده من خلال المتحولات الوسيطة.
FillEllipse	يُستخدم لرسم وملئ قطع ناقص.
FillPoly	يُستخدم لرسم وملئ مضلع.
GetX	يعطي قيمة الموقع الحالي للمؤشر على محور الإحداثيات x في حالة النمط الرسومي.
GetY	يعطي قيمة الموقع الحالي للمؤشر على محور الإحداثيات y في حالة النمط الرسومي.
GraphDefault	يعيد تحميل البيئة الرسومية ، ويتوضع المؤشر في الزاوية العلوية اليسارية.
InitGraph	يفعل البيئة الرسومية ويجهز عتاد الحاسوب للتعامل مع النمط الرسومي.
Line	يرسم خطاً مستقيماً بين نقطتين يتم تحديد إحداثياتها من خلال المتحولات الوسيطة.

LineRel	يرسم خطاً مستقيماً بدءاً من الموقع الحالي للمشيّة وحتى نقطة محددة وفق بعد نسبي .
LineTo	يرسم خطاً مستقيماً بدءاً من الموقع الحالي للمشيّة وحتى نقطة محددة.
MoveRel	ينقل المشيّة من الموقع الحالي إلى موقع جديد يبعد بمسافة نسبية.
MoveTo	ينقل المشيّة من الموقع الحالي إلى موقع محدد.
OutText	يخرج (يطبع) سلسلة محرفية على الشاشة بدءاً من الموقع الحالي للمشيّة.
OutTextXY	يخرج (يطبع) سلسلة محرفية على الشاشة بدءاً من موقع محدد.
PieSlice	يرسم ويملئ جزء من الدائرة.
PutPixel	يرسم نقطة تحدد إحداثياتها من خلال المتحولات الوسيطة.
RectAngle	يرسم مستطيلاً تحدد إحداثياته وأبعاده من خلال المتحولات الوسيطة.
SetBkColor	يحدد لون الخلفية في البيئة الرسومية.
SetColor	يحدد لون الرسم الحالي.
SetFillStyle	يحدد نموذج وشكل المملئ للأشكال الرسومية.
SetGraphMode	يجهز النمط الرسومي ويمسح الشاشة.

SetLineStyle	يحدد شكل وعرض خط الرسم.
SetTextStyle	يحدد نوع وشكل نص الكتابة.
TextWidth	يحدد عرض السلسلة المحرفية مقدراً بالنقطة (Pixel).

4- بعض الإجراءات المتضمنة في الوحدة Dos:

الوظيفة	الإجراء أو التابع
يعطي مساحة التخزين الفارغة على القرص	DiskFree
يعطي مساحة التخزين الكلية على القرص	DiskSize
يعطي رقم الإصدار لنظام الـ Dos	DosVersion
ينفذ برنامجاً من خلال نظام التشغيل	Exec
يبحث عن ملف محدد	Fsearch
يعطي التاريخ الحالي في الحاسوب	GetDate
يعطي الوقت الحالي في الحاسوب	GetTime
يعين التاريخ الحالي للحاسوب	SetDate
يعين تاريخ ووقت إنشاء الملف المحدد	SetFTime

SetTime	يعين الوقت الحالي للحاسوب
---------	---------------------------

الملحق (ب)

رسائل الخطأ



معنى الخطأ	نوع الخطأ	رقم الخطأ
انتهاء الذاكرة: يحدث هذا الخطأ عندما تصبح سعة الذاكرة غير كافية لإتمام عملية ترجمة البرامج، وهناك بعض الحلول لتجاوز هذه المشكلة، منها: - إذا كان خيار مكان الترجمة هو الذاكرة، أي ان الخيار Destination في قائمة Compile هو الذاكرة Memory، فإنه يجب تغييره إلى Disk وإعادة المحاولة. - إذا كان خيار ربط الذواكر الوسيطة (Link Buffer) هو الذاكرة، أي الخيار Memory، ضمن قائمة Linker من القائمة Options فإنه يجب تغييره إلى الخيار Disk، وإعادة المحاولة. - إذا كنت تستخدم برنامج Turbo.exe، أستخدم البرنامج Tpc.exe عوضاً عنه.	Out of memory	1
يظهر هذا الخطأ عندما يتم إعادة تعريف أحد الكلمات المحجوزة في لغة التريباسكال.	Identifier Expected	2

3	Unknown Identifier	يظهر هذا الخطأ عندما يتم التصريح عن متحول في البرنامج لم يتم تعريفه مسبقاً.
4	Duplicate Identifier	يظهر هذا الخطأ عندما يتم تكرار تعريف متحول.
5	Syntax Error	يحدث هذا الخطأ عندما يجد المترجم مجموعة من الرموز لا معنى لها ضمن البرنامج (غير مفهومة). مثال: "I" (Integer)
6	Error in Real Constant	خطأ في كتابة ثابت من النوع الحقيقي .
7	Error in Integer Constant	خطأ في كتابة ثابت من النوع الصحيح .
8	String Constant Exceeds Line	طول ثابت السلسلة المحرفية تجاوز طول السطر ، غالباً نسيت وضع الرمز " " في نهاية السلسلة.
10	Unexpected End of File	يحدث هذا الخطأ في بعض الحالات منها: - لم يتم إنهاء البرنامج بـ End - عدد الـ Begin في البرنامج لا يتوافق مع عدد الـ End .
11	Line too long	يحدث هذا الخطأ عندما يتجاوز طول السطر الـ 126 حرفاً.
12	Type identifier expected	المتحول لم يرتبط بنوع المعطيات بالشكل الصحيح .

13	Too Many Open File	يحدث هذا الخطأ عندما عدد الملفات المفتوحة كبير جداً ، ويمكن التحكم بعدد الملفات المفتوحة في وقت واحد من خلال التعليمة File=xx في الملف Config.sys
14	Invalid File Name	في هذه الحالة يكون اسم الملف المستخدم غير صحيح، أو أن الملف المطلوب غير موجود ضمن المسار المحدد .
15	File Not Found	لم يتم العثور على الملف في الدليل الحالي، أو في الأدلة المسموح البحث فيها عن هذا الملف .
16	Disk Full	القرص ممتلئ ، احذف بعض الملفات من هذا القرص، أو استخدم قرصاً جديداً .
17	Invalid Compiler Directive	لم يتم التعرف على توجيه المترجم في البرنامج، أو أن أحد وسطاء هذا التوجيه غير صحيح.
18	Too Many File	يوجد عدد كبير من الملفات المستخدمة أثناء ترجمة البرامج أو الوحدة ، لذلك حاول عدم استخدام جميع هذه الملفات في الوقت نفسه.
19	Undefined type in pointer definition	نوع معطيات تم استخدامه ضمن تعريف نوع المعطيات مؤشر ولكن لم يُصرح عنه مسبقاً.

20	Variable identifier Expected	توقع لوجود مميز متحول : هذا المميز لم يشر إلى المتحول كما ينبغي .
21	Error in type	خطأ في نوع المعطيات.
22	Structure Too Large	بنية المعطيات كبيرة جداً، حيث أن الحجم الاعظمي التي يمكن أن تستوعبها بنية المعطيات هو (65520) بايت .
23	Set base Type out of Range	نوع المعطيات الأساسي في المجموعة تجاوز المجال المحدد، و يجب أن يقع نوع المعطيات الأساسي للمجموعة ضمن المجال (0) من وحتى (255) وإذا كان نوعه تعدادياً يجب أن لا تتجاوز حدوده القيمة (256).
24	File components may not be files or objects	عناصر الملف لا يجوز أن يكون ملفات أخرى أو أهداف، فمثلاً لا يجوز تعريف الملف كآلي: file of file أو file of objects.
25	Invalid string length	طول السلسلة الرمزية غير صحيح : يجب أن يقع الطول المصرح عنه للسلسلة الحرفية ضمن المجال من (1) وحتى (255) .
26	Type mismatch	عدم توافق في الأنواع : يمكن أن يعود ذلك لأحد الحالات الآتية: 1- عدم توافقية بين نوع المحتويات والتعابير في عبارة الإلحاق .

		2- عدم توافقية بين نوع الوسطاء الشكلية والوسطاء الحقيقية المستخدمة في إستدعاء البرامج الفرعية (procedure, Function). 3- عدم توافقية بين نوع التعبير وبين نوع دليل النسق. 4- عدم توافقية بين أنواع الحدود في التعبير الواحد.
27	Invalid subrange base type	خطأ في نوع المعطيات الأساسي الخاص بالمجال الجزئي .
28	Lower bound greater than upper bound	عندما يتم التصريح عن أنواع المعطيات، مجال جزئي، يحدث خطأ عند تحديد الحد السفلي أكبر من الحد العلوي لهذا المجال .
29	Ordinal type expected	توقع لوجود نوع معطيات ترتيبي : هنا لا يجوز استخدام أنواع المعطيات الآتية: الحقيقية، سلسلة محرفية، المركبة ، مؤشر.
30	Integer constant expected	توقع لوجود ثابت صحيح : يسمح هنا فقط بوجود ثابت صحيح .
31	Constant expected	توقع لوجود ثابت : يسمح هنا فقط بوجود ثابت .

32	Integer or real constant expected	توقع لوجود ثابت صحيح أو حقيقي : يسمح هنا فقط بوجود ثابت عددي.
33	Pointer type identifier expected	توقع لوجود نوع المعطيات مؤشر، يحدث الخطأ بسبب عدم الربط الصحيح.
34	Invalid function result type	نوع المعطيات الخاص بنتائج التابع غير صحيح : هنا يجب أن تكون نتائج التابع من أحد أنواع المعطيات البسيطة الآتية: سلسلة حرفية أو مؤشر .
35		توقع لوجود علامة (label): لم يتم استخدام العلامة بالشكل الصحيح.
36	Begin expected	توقع لوجود التعليمة Begin: يحدث الخطأ في حالة عدم استخدام التعليمة Begin ، أو أنه حصل خطأ في ترتيب البرنامج أو الوحدة .
37	End expected	توقع لوجود التعليمة End: يحدث الخطأ في حالة عدم استخدام التعليمة End ، أو أنه حصل خطأ في ترتيب البرنامج أو الوحدة .
38	Integer expression expected	توقع لوجود تعبير من النوع الصحيح.
39	Ordinal expression expected	توقع لوجود تعبير من النوع الترتيبي.

40	Boolean expression expected	توقع لوجود تعبير من النوع البولياني.
41	Operand types do not match operator	عدم تطابق بين أنواع المعطيات والمعاملات المطبقة عليها.
42	Error in expression	خطأ في التعبير : يجب التدقيق في التعبير ربما نسيت وضع معامل بين حدود التعبير .
43	Illegal assignment	إلحاق فاشل : ويعود ذلك إلى أن: 1- متحولات الملفات لا يجوز إلحاق قيم فيها. 2- يجوز إلحاق قيم بالتابع Function فقط ضمن جزء التعليمات الخاص به، وليس خارج هذا الجزء .
44	Field identifier expected	توقع لوجود مميز لحقل : لم يتم الربط بين المميز ونوع الحقل ، ضمن متحول السجل .
45	Object file too large	الملف الهدف كبير جدا : لا تستطيع تربو باسكال ربط ملفات هدف (Obj). ذات طول أكبر من (64) كيلو بايت
46	Undefined external	غياب التعريف الخارجي : الإجراءات والتوابع الخارجية external لا تطابق التعريف public ضمن الملف الهدف، يجب التأكد هنا من توجيه المترجم.
47	Invalid Object File	سجل الملف الهدف غير صحيح : الملف

	Record	الهدف يحتوي على سجل هدف غير صحيح ، يجب التأكد من أن الملف المراد استخدامه هو ملف هدف Obj.
48	Code Segment Too Large	مقطع شيفرة التعليمات كبير جداً : لا يجوز أن يتجاوز حجم مقطع الشيفرة 65520 بايت ، وهنا يجب تقسيم البرنامج إلى إجراءات أو توابع أو وحدات.
49	Data Segment Too Large	مقطع المعطيات كبير جداً : لا يجوز أن يتجاوز حجم مقطع الشيفرة 65520 بايت ، للتخلص من المشكلة يمكن استخدام المؤشرات.
50	Do Expected	يُتوقع وجود عبارة Do، ولكن يبدو أنه لم تظهر في الموقع الصحيح.
51	Invalid public definition	التعريف Public غير صحيح
52	Invalid extrn definition	التعريف extrn غير صحيح
53	Too many extrn definition	تعريفات extrn كثيرة جداً : لا يمكن التعامل مع ملفات هدف (obj) تملك أكثر من (256) تعريفاً من النوع الخارجي extern
54	Of expected	يُتوقع وجود عبارة Of، ولكن يبدو أنه لم تظهر في الموقع الصحيح.

55	Interface expected	يُتوقع وجود تعليمة Interface، ولكن يبدو أنه لم تظهر في الموقع الصحيح.
56	Invalid Relocatable Reference	عناوين التنقل المرجعية غير صحيحة : يحدث هذا الخطأ عندما يتم التصريح عن معطيات وعناوين تنقل في مواقع غير صحيحة.
57	Then Expected	يُتوقع وجود عبارة Then، ولكن يبدو أنه لم تظهر في الموقع الصحيح.
58	To Or DownTo Expected	يُتوقع وجود العبارات To أو DownTo، ولكن يبدو أنه لم تظهر في الموقع الصحيح لها.
59	Undefined Forward	لا يوجد تعريف مسبق : يحدث هذا الخطأ عندما لا يتم التصريح عن التتابع والإجراءات أو الوحدات مسبقاً في الأماكن المخصصة لتعريفها.
60	Too Many Procedure	الإجراءات كثيرة جداً : يجوز فقط استخدام 512 إجراءً في كل برنامج. في حال حصول هذا الخطأ في البرنامج فإنه يجب تقسيم البرنامج إلى وحدات، وفي حال حصول الخطأ في الوحدة فإنه يجب تقسيم الوحدة إلى وحدات أصغر.
61	Invalid Typecast	قلب النمط غير صحيح : يحدث هذا الخطأ عندما يتم قلب تعبير إلى متحول، أو قلب نمط متحول إلى نمط آخر، يختلف حجمه عن

		النمط الأول.
62	Division by zero	قسمة على صفر
63	Invalid file type	نوع معطيات الملف غير صحيح : يحدث هذا الخطأ عندما لا تتلائم الإجراءات المطبقة على ملف مع نوع المعطيات في الملف .
64	Cannot read or write variables of this type	لا يستطيع قراءة أو كتابة متحولات من هذا النمط: يحدث هذا الخطأ عندما يتم قراءة أو كتابة معطيات غير مسموح استخدامها مع الإجراءات Read و Write
65	Pointer variable expected	توقع لمتحول مؤشر
66	String variable expected	توقع لمتحول سلسلة
67	String expression expected	توقع لتعبير سلسلة محرفية
68	Circular unit reference	يحدث هذا الخطأ في حالة الوحدات وعندما يت استخدام الوحدات بشكل متقابل أي أن Unit1 تستخدم Unit2 و Unit2 تستخدم Unit1، فعند الترجمة يحدث هذا الخطأ.
69	Unit name mismatch	عدم تطابق في اسم الوحدة : اسم الوحدة الموجود في الملف (TPU) لا يطابق الاسم المخصص ضمن عبارة Uses.

70	Unit version mismatch	عدم تطابق في نسخة الوحدة : يحدث هذا الخطأ عندما يتم تعديل في إحدى الوحدات المستخدمة، لذلك يجب إعادة ترجمة الوحدة.
71	Internal stack overflow	فائض في المكسد الداخلي : يجب هنا التخلص من العبارات المتداخلة في البرنامج.
72	Unit file format error	خطأ في تشكيل ملف الوحدة
73	Implementation expected	توقع للعبارة Implementation: العبارة Implementation لم تظهر في الموقع الصحيح.
74	Constant and case Types do not match	عدم تطابق بين نوع الثابت المستخدم في عبارة Case وبين تعبير اختيار حالات Case.
75	Record or object variable expected	يُتوقع وجود سجل أو متحول هدف.
76	Constant out of range	قيمة الثابت تقع خارج المجال المسموح
77	File variable expected	يُتوقع وجود متحول من النمط ملف.
78	Pointer expression expected	يُتوقع وجود تعبير مؤشر
79	Integer or real expression expected	يُتوقع وجود تعبير من النمط الصحيح أو الحقيقي.
80	Label not within current block	العلامة ليست ضمن الكتلة الحالية

81	Label already defined	العلامة تم تعريفها مسبقاً
82	Undefined label in preceding statement Bar	العلامة غير معرفة ضمن جزء العبارة السابقة، أي تم التصريح عن العلامة إلا أنها لم تعرف بعد.
83	Invalid @ argument	متحول وسيطي خاطئ للتابع @، المتحول الوسيطي لهذا التابع يجب أن يكون متحول أو اسم تابع، أو اسم إجراء
84	Unit expected	يُتوقع وجود عبارة Unit، ولكن يبدو أنه لم تظهر في الموقع الصحيح.
85	";" expected	يُتوقع وجود ل ";"، ولكن يبدو أنه لم تظهر في الموقع الصحيح.
86	":" expected	يُتوقع وجود ل ":"، ولكن يبدو أنه لم تظهر في الموقع الصحيح.
87	"," expected	يُتوقع وجود ل ","، ولكن يبدو أنه لم تظهر في الموقع الصحيح.
88	"(" expected	يُتوقع وجود ل "("، ولكن يبدو أنه لم تظهر في الموقع الصحيح.
89	")" expected	يُتوقع وجود ل ")"، ولكن يبدو أنه لم تظهر في الموقع الصحيح.

90	"=" expected	يُتوقع وجود ل "=", ولكن يبدو أنه لم تظهر في الموقع الصحيح.
91	":=" expected	يُتوقع وجود ل ":", ولكن يبدو أنه لم تظهر في الموقع الصحيح.
92	"[" Or "(." expected	يُتوقع وجود ل "[" أو "(.", ولكن يبدو أنه لم تظهر في الموقع الصحيح.
93	"]" Or ")." expected	يُتوقع وجود ل "]" أو ").", ولكن يبدو أنه لم تظهر في الموقع الصحيح.
94	"." expected	يُتوقع وجود ل ".", ولكن يبدو أنه لم تظهر في الموقع الصحيح.
95	".." expected	يُتوقع وجود ل "..", ولكن يبدو أنه لم تظهر في الموقع الصحيح.
96	Too many variables	عدد كبير جداً من المتحولات : يجب أن لا يزيد الحجم الكلي للمتحولات العامة (المصرح عنها في البرنامج)، أو الحجم الكلي للمتحولات الموضعية (المصرح عنها ضمن الإجراء أو التابع) عن 64 كيلوبايت.
97	Invalid for cotrol variable	متحول التحكم بحلقة for غير صحيح

98	Integer variable expected	توقع لوجود متحول من النمط صحيح
99	File or procedure types are not allowed here	لا يسمح هنا باستخدام أنواع الملفات أو الإجراءات
100	String length mismatch	عدم تطابق بين طول السلسلة المحرفية وعدد الحروف في السلسلة
101	Invalid ordering of fields	ترتيب الحقول غير صحيح : تم ترتيب الحقول بشكل مخالف لترتيبها في التصريح
102	String constant expected	يُتوقع وجود سلسلة محرفية ثابتة لم تظهر في موقعها الصحيح .
103	Integer or real variable expected	توقع لوجود متحول صحيح أو حقيقي
104	Ordinal variable expected	توقع لوجود متحول ترتيبى
105	Inline error	خطأ Inline : لا يسمح باستخدام المعامل < مع العناوين المرجعية للمتحويلات
106	Character expression expected	توقع لوجود تعبير محرفي.
107	Too many relocation items	العناصر المنتقلة كثيرة جداً: يحدث هذا الخطأ عندما يكون البرنامج المستخدم كبير جداً، وفي هذه الحالة يجب تقسيم البرنامج إلى أجزاء بشكل برامج فرعية
108	Overflow in	فائض في العملية الحسابية : ناتج العملية

	arithmetic operation	الحسابية السابقة لا يقع ضمن مجال الأعداد الصحيحة من النمط الطويل الطويلة longint
109	No enclosing for while or repeat statement	لم يغلف بعبارة for أو repeat أو while : يظهر هذا الخطأ عندما يتم استخدام الإجراءات break و continue خارج مجال إحدى الحلقات for أو while أو repeat.
110	Debug information table overflow	فائض في جدول معلومات التنقيح: يحدث هذا الخطأ عندما يحتوي البرنامج المستخدم على أكثر من (65536) حرفاً، أو يتألف من أكثر من (65536) سطراً .
112	Case constant out of range	ثابت عبارة case خارج المجال المسموح
113	Error in statement	خطأ في العبارة : لا يمكن أن تبتدأ العبارة بهذا الرمز .
114	Cannot call an interrupt procedure	لا يمكن استدعاء إجراء المقاطعة
116	Must be in 8087 mode to compile this.	يجب أن نكون في النمط 8087 من أجل ترجمة هذه : يتطلب هنا توجيه المترجم من الحالة {\$N-} إلى الحالة {\$N+} لأن العمليات على الأنواع الحقيقية المسموحة في المعالج الرياضي 8087 (النوع الحقيقي العادي و المضاعف والمعقد...) لا يمكنها العمل في حالة المترجم هي {\$N-} .

117	Target address not found	لا يمكن العثور على العنوان الهدف
118	Include files are not allowed here	الملفات المضمنة ليس مسموح بها هنا
119	No inherited methods are accessible here	لا يمكن الوصول هنا إلى أي نهج موروث: يحدث هذا الخطأ عندما يتم استخدام إحدى الكلمات المحجوزة خارج النطاق المعرفة فيه.
121	Invalid qualifier	وصف غير صحيح : يحدث هذا الخطأ في حالات مختلفة مثل وضع دليل لمتحول غير معرف على أنه نسق، أو نحدد حقول لمتحول ليس سجلاً، أو أن يُعطى عنوان لمتحول ليس مؤشراً.
122	Invalid variable reference	العنوان المرجعي للمتحول غير صحيح
123	Too many symbols	رموز كثيرة جداً: الحد الأعظمي للرموز التي يمكن أن تُعرف في البرنامج أو الوحدة لا يجوز أن تتجاوز الـ 64 كيلوبايت، ومن أجل تجاوز هذه المشكلة يجب إعادة تغيير توجيه المترجم <code>.\$d</code> .
124	Statement part too large	جزء العبارة كبير جداً : لا يجوز في التروبايسكال أن يتجاوز حجم العبارة من الرموز 24 كيلوبايت ، في حال وقوع هذا الخطأ، فإنه يجب تقسيم العبارة إلى أجزاء أصغر أو استخدام الإجراءات.

126	Files must be var parameters	الملفات يجب أن تكون وسطاء من النوع var
127	Too many conditional symbols	الرموز الشرطية كثيرة جدا
128	Misplaced conditional directive	وضع غير صحيح للتوجيهات الشرطية : { \$else } صائد المترجم أحد التوجيهات , أو { \$endif } , ولم يتم العثور على أحد التوجيهات { \$ifdef } , أو { \$ifndef } , أو { \$ifopt } .
129	Endif directive missing	التوجيه endif غير موجود
130	Error in initial conditional defines	خطأ في التعريفات الشرطية الأولية
131	Header does not match previous definition	الرأس لا يطابق التعريف السابق: رأس الإجراء أو التابع المحدد ضمن قسم الإتصال الخارجي interface أو ضمن التصريح الأمامي forward لا يطابق هذا الرأس
133	Cannot evaluate this expression	غير ممكن حساب هذا التعبير
134	Expression incorrectly terminated	انتهاء غير صحيح للتعبير
135	Invalid format specifier	تعريف صيغة غير صحيح

136	Invalid indirect reference	إرجاع غير مباشر و غير صحيح
137	Structured variables are not allowed here	متحولات بنى المعطيات لا يُسمح باستخدامها هنا
138	Cannot evaluate without system unit	لا يمكن إجراء الحسابات بدون وجود الوحدة system
139	Cannot access this symbol	لا يمكن الوصول إلى هذا الرمز : يحدث هذا الخطأ عندما يتعذر الوصول إلى بعض الرموز الخاصة بعد ترجمة البرنامج
140	Invalid floating point operation	عملية غير صحيح لأعداد من نمط الفاصلة العائمة (الحقيقية) .
141	Cannot compile overlays to memory	لا يمكن ترجمة البرامج المتراكبة في الذاكرة : في هذه الحالة يجب أن تجري الترجمة على القرص .
142	Pointer or procedural variable expected	توقع لوجود متحول إجرائي أو متحول مؤشر
143	Invalid procedure or function reference	استخدام غير صحيح لإجراء أو تابع
144	Cannot overlay this unit	لا يمكن إجراء تراكب على هذه الوحدة: يحدث هذا الخطأ عند محاولة إجراء تراكب على وحدة لم تترجم وفق الموجه { \$O+ }
145	Too many nested scopes	مجاللات التداخل كثيرة جدا : يحدث هذا الخطأ عندما يتم تجاوز المجال المسموح

		للتدخلات، حيث لا يجوز أن تتجاوز التدخلات للروتينات عن 512 تدخلاً في البرنامج، وعلى أن لا تتجاوز التدخلات الـ 128 تدخلاً لكل روتين. ومصدر التدخلات يمكن أن يكون من: - الوحدات ضمن عبارة <code>uses</code>. - السجلات المتداخلة ضمن قسم التصريح. - عبارات <code>with</code> المتداخلة.
146	File access denied	تعذر الوصول للملف: يحدث هذا الخطأ إحدى حالتين: - استخدام ملف القراءة كملف خرج. - استخدام اسم لدليل كأسم لملف خرج.
147	Object type expected	توقع لوجود نمط هديفي
148	Local object types are not allowed	الأنماط الهدفية المحلية غير مسموح استخدامها: لا يجوز تعريف الأنماط الهدفية ضمن الإجراءات أو التوابع، وإنما يجب أن تعرف ضمن المجال الخارجي للبرنامج أو الوحدة
149	Virtual expected	توقع لوجود <code>virtual</code>
150	Method identifier expected	توقع لوجود مميز منهج (طريقة) <code>Method</code>
151	Virtual constructors are not allowed	لا يسمح باستخدام البواني الوهمية

152	Constructor identifier expected	توقع لوجود مميز لمنهج باي
153	Destructor identifier expected	توقع لوجود مميز لمنهج هادم
154	Fail only allowed within constructors	الإجراء المعياري fail يسمح به فقط ضمن المناهج البانية.
155	Invalid combination of opcode and operands	دمج غير صحيح بين شفرات التعليمات وبين حدودها (تعليمات المجمع Assembler لا تقبل هذا النوع من الدمج بين الحدود
156	Memory reference expected	توقع لاستخدام مرجعي للذاكرة : حدود أوامر المجمع ليست حدود تشير للذاكرة : يحدث هذا الخطأ غالباً في حالة عدم وضع الأقواس المربعة حول الحد ليشير للذاكرة . فعلى سبيل المثال : قد تكون كتبت $ax\ mov\ ,\ bx + si$ بدلاً عن $mov\ ax\ ,\ [bx + si]$
157	Cannot odd or subtract relocatable symbols	لا يمكن جمع رموز قابلة للتنقل أو طرحها: الرموز القابلة للتنقل هي : المتحولات أو الإجراءات أو التتابع أو العلامات Labels
158	Invalid register combination	دمج المسجلات غير صحيح
159	287 instruction are 286 /not enabled	لا يمكن استخدام تعليمات 286/287 : من أجل استخدامها يجب توجيه المترجم

		{ \$G+ }
160	Invalid symbol reference	استخدام غير صحيح للرمز : هذا الرمز لا يستخدم ضمن هذا الحد لل Assembler ويحدث هذا الخطأ في حالات هي : - لقد حاولت أن تصل إلى إجراء قياسي , أو تابع قياسي , أو إحدى الأنساق الخاصة mem أو memw أو meml أو port أو portw ضمن حد لتعليمات المجمع. - لقد حاولت أن تصل إلى سلسلة محرفية , أو عدد حقيقي , أو مجموعة من الثوابت ضمن حد لتعليمات المجمع - لقد حاولت أن تصل إلى إجراء inline أو تابع inline ضمن حد لتعليمات المجمع . - لقد حاولت أن تصل إلى الرمز الخاص @ result خارج تابع ما. - لقد حاولت أن تولد تعليمة قفز قصير jmp ، ولكن إلى شيء آخر غير العلامات .
161	Code generation error	خطأ في توليد شفرات البرنامج : العبارة السابقة تحتوي إحدى التعليمات loopne أو loope أو loop أو jcxz , التي لا

		تستطيع الوصول إلى العلامات المحددة لها.
--	--	---



162	Asm expected	توقع لوجود المميز Asm
163	Duplicate dynamic method index	دليل منهج ديناميكي مكرر
164	Duplicate resource identifier	مميز مورد معطيات مكرر
165	Duplicate or invalid export index	دليل غير صحيح أو مكرر : الرقم الترتيبي المحدد ضمن عبارة الدليل الذي سيصدر إلى روتين آخر ، أم أنه: - ليس ضمن المجال من (1) حتى (32767). - قد استخدم من قبل روتين مصدر آخر .
166	Procedure or function identifier expected	توقع لوجود مميز إجراء أو تابع : العبارات المصدرة يجوز أن إجراءات أو توابع فقط.
167	Cannot export this symbol	لا يمكن تصدير هذا الرمز : لا يمكن تصدير الإجراء أو التابع مالم يُصرح عنه من خلال توجيه تصدير الإجراءات .
168	Duplicate export name	اسم تصدير مكرر
169	Executable file header too large	رأس الملف التنفيذي كبير جدا : حجم رأس الملف التنفيذي (exe) الذي نقوم بتوليده يزيد عن (64) كيلو بايت , وهو الحد الأعظمي الذي يسمح به برنامج الربط

		.linker إذا كنت : - تستورد عددا كبيرا من الإجراءات و التوابع بأسمائها. - تصدر عددا كبير من الإجراءات والتوابع بأسمائها . - تملك عددا كبيرا من الموارد عندئذ حاول أن تقوم باستيراد أو تصدير الروتينات باستخدام اعداد ترتيبية .
170	Too many Segments	عدد كبير جداً من المقاطع : عدد المقاطع في الملف التنفيذي يتجاوز (254) مقطعاً ، وهو الحد الأعظمي لعدد المقاطع للملفات التنفيذية ، في النمط المحمي للنظام DOS أو windows . الحجم المحدد لمقطع شفرة العمليات (code segment) في الملفات التنفيذية صغير جداً ، ومن أجل تحديد حجم أكبر لهذه المقطع ، يمكنك استخدام توجيه المترجم {\$S} .

أخطاء تحدث خلال التنفيذ

أخطاء تحدث أثناء التنفيذ : هناك مجموعة من الأخطاء تحدث أثناء تنفيذ البرنامج ، وهذه الأخطاء تظهر الرسالة

RUN-TIME ERROR nnn at xxxx : yyyy

ثم ينتهي التنفيذ،

حيث nnn هو رقم هذا الخطأ أما xxxx:yyyy فهو يشكل عنوان المقطع والإزاحة الخاص بالخطأ . أخطاء التنفيذ تقسم إلى أربعة أقسام هي :

- 1- أخطاء dos :هي من (1) وحتى (99)
- 2- أخطاء دخل/خرج (I/O): وهي من (100) حتى (149)
- 3- أخطاء حرجة : وهي من (200) وحتى (255) .
- 4- أخطاء قاتلة : وهي من (200) حتى (255) . وهذه الأرقام ليست مستخدمة كلها ، ونعرض فيما يلي المستخدم منها:

أخطاء DOS

رقم الخطأ	نوع الخطأ	معنى الخطأ
2	file not found	ملف غير موجود : يحدث هذا الخطأ عند استخدام إحدى الإجراءات reset أو append أو rename أو erase ، وذلك في حال غياب الملف المطلوب عند الاستدعاء.
3	path not found	المسار غير موجود: يحدث هذا الخطأ في إحدى

		الحالات الآتية: - عند استخدام إحدى الإجراءات <code>reset</code>، أو <code>rewrite</code>، أو <code>append</code>، أو <code>rename</code>، أو <code>erase</code>، إذا كان أسم المسار الملحق بأسم الملف غير صحيح، أو لم يعثر على الملف في ذلك الدليل الفرعي - عند استخدام إحدى الإجراءات <code>chdir</code> أو <code>mkdir</code>، أو <code>rmdir</code>، إذا كان المسار المحدد غير صحيح أو أن المسار غير موجود.
4	Too many open files	الملفات المفتوحة كبيرة جداً : يحدث هذا الخطأ عند استخدام إحدى الإجراءات <code>reset</code> أو <code>rewrite</code> أو <code>append</code>، كان عدد الملفات المفتوحة أكبر من (15) ملفاً، إذا كان عدد الملفات المفتوحة أقل من (15) وظهر هذا الخطأ، فإنه يجب العودة إلى الملف <code>config.sys</code> والتأكد من وجود الأمر <code>files = xx</code> والذي يحدد عدد الملفات التي يستطيع نظام <code>dos</code> فتحه.
5	Files access denied	يتعذر الوصول للملف : يحدث هذا الخطأ في حالة الاستخدام الغير صحيح لإحدى الإجراءات: <code>reset, rename, erase, rmdir, read append, rewrite, mkdir, block write , block, readwrite</code>
6	Invalid file	تمرير الملف غير صحيح :

	handle	يحدث هذا الخطأ عند استخدام رقم غير صحيح لتمرير الملف إلى Dos.
12	Invalid file access code	شفرة الوصول للملف غير صحيحة: يحدث هذا الخطأ عند استخدام قبل الإجراء reset أو الإجراء append وذلك في حالة استخدام قيمة غير صحيحة للمتحول filemode.
13	Invalid driver number	رقم السواعة غير صحيح : يحدث هذا الخطأ عند استخدام الإجراء getdir ، أو الإجراء chdir ، وذلك عندما يكون رقم السواعة الممرر إليهما غير صحيح .
16	Cannot remove current directory	لا يمكن مسح الدليل الحالي: يحدث هذا الخطأ عند استخدام الإجراء rmdir ، وذلك عندما يكون المسار الممرر هو الدليل الحالي .
17	Cannot rename across drives	لا يمكن تغيير الأسماء من خلال السواعات: يحدث هذا الخطأ هذا الخطأ عند استخدام الإجراء rename ، وذلك عندما يكون الأسمان المراد تبديلهما غير موجودان ضمن السواعة الحالية .
18	No more files	لا يوجد مزيد من الملفات : يولد هذا الخطأ من قبل الإجراء doserror إذا كان استدعاء الإجراء findfirst أو الإجراء findnext لا يعيد أي ملفات.

الملحق (ج)

بعض أوامر نافذة التحرير

الوظيفة	الأمر
ينقل المشيرة إلى السطر السابق لموقعها	CTRL-E أو السهم العلوي
ينقل المشيرة إلى السطر التالي	CTRL-X أو السهم السفلي
ينقل المشيرة رمزاً واحداً لليسار	CTRL-S أو السهم ليساري
ينقل المشيرة رمزاً واحداً لليمين	CTRL -D أو السهم اليميني
ينقل المشيرة إلى بداية الكلمة الحالية أو السابقة	CTRL -A أو السهم اليساري - CTRL
ينقل المشير إلى بداية الكلمة التالية	CTRL -F أو السهم اليميني - CTRL
ينقل المشيرة إلى بداية السطر الحالي	CTRL -QS أو HOME
ينقل المشيرة إلى نهاية السطر الحالي	CTRL -QD أو END
ينقل المشيرة إلى أعلى الشاشة	CTRL -R
ينقل المشيرة إلى أسفل الشاشة	CTRL -C
ينقل المشيرة إلى بداية الملف	CTRL -PGUP
ينقل المشيرة إلى نهاية الملف	CTRL -PGDN
ينقل المشيرة إلى السطر العلوي من الشاشة	CTRL -HOME
ينقل المشيرة إلى السطر السفلي من الشاشة ، مبقياً المشيرة في نفس العمود.	CTRL -END
ينتقل بين نمطي الحشر أو الكتابة فوق الحروف	CTRL-V أو INS
يضيف سطرًا فارغاً في موضوع المشيرة	CTRL-N

	الحالي ، بعد إزاحة الأسطر إلى الأسفل.
DEL أو CTRL-G	يحذف رمزاً في موقع المšíرة الحالي.
BACKSPACE أو CTRL-H	يحذف رمزاً على يسار موضوع المšíرة الحالي.
CTRL-Y	يحذف السطر الحالي.
CTRL-QY	يحذف من موقع المšíرة الحالي إلى نهاية السطر الحالي.
CTRL-T	يحذف من موقع المšíرة الحالي إلى نهاية الكلمة الحالية.
ALT- CTRL –EU BACKSPACE	يلغي آخر حركة أو عملية حذف.
CTRL-KB	يعلم بداية كتلة من النصوص .
CTRL-KK	يعلم نهاية كتلة من النصوص.
CTRL-KT	يعلم كلمة واحدة ، تكون المšíرة واقعة عليها ، ما إذا كانت المšíرة واقعة بين كلمتين ، فسيعلم هذا الأمر الكلمة الواقعة يسار المšíرة.
CTRL-KY	يحذف الكتلة المعلمة.
CTRL-KV	ينقل الكتلة المعلمة من موقعها الحالي إلى موقع المšíرة.
CTRL-KC	ينسخ الكتلة المعلمة إلى موقع ثانٍ ، وبهذا تظهر الكتلة نفسها مرتين ضمن الملف.
CTRL-KW	يكتب الكتلة المعلمة في ملف جديد ، يتم تحديد أسمه مفترضاً أن امتداده (Pas) وإن

	لم نحدد غير ذلك.
CTRL-KR	يقرأ محتويات ملف يتم تحديد اسمه, ويضيف محتوياته في موقع المشيرة الحالي.
CTRL-KP	يطبع الكتلة المعلمة , وفي حال عدم تعلم أي كتلة يطبع الملف كاملاً.
CTRL-Q [أو CTRL-Q]	يبحث عن محددات مطابقة , هذا يعني إذا كانت المشيرة متوضعة فوق احد الرموز التالية : { [* < , عن إذ نستطيع العثور على الرموز المقابلة اليمنى , أما إذا كانت المشيرة متوضعة على احد الرموز : { [>) فسوف يبحث الأمر عن الرموز اليسرى المقابلة لها .
CTRL-Kn	يعلم عن موضع المشيرة الحالي ويعطيه الرقم n , وهذا الرقم يجب أن يكون بين (0) و (9) .
CTRL-Qn	ينقل المشيرة إلى الموقع ذي الرقم n.
ALT-SF أو CTRL-QF	يبحث عن سلسلة رمزية يحدها له.
ALT-SR أو CTRL-QA	يبحث عم سلسلة رمزية يحدها له ويستبدلها بأخرى يحدها له أيضاً .
CTRL-L	يكرر آخر أمر بحث أو أمر بحث واستبدال.
CTRL-P	يسمح لنا بكتابة رموز تحكم في الملف , في الحقيقة يتألف الملف من رموز قابلة للطباعة , مثل الحروف , والأرقام , والرموز المتنوعة

	، لكن أحياناً نحتاج إلى إضافة رموز تحكم في الملف ، وذلك بضغط على P-CTRL ،ومن ثم كتابة.
--	---

المصطلحات العلمية	
انكليزي	عربي
adapter	محول

address	عنوان
algorithm	خوارزمية
append	إضافة
array	متجهة، أبعاد، نسق
ASCII (American Standard Code for Information Interchange)	الترميز الأمريكي المعياري للتبادل البيني للمعلومات
assembly Language	لغة الجمع
assign	ربط
begin	بداية
bit	رقم ثنائي
blinking	إضاءة
block	كتلة
boolean	بوليان (منطقي)
brightness	سطوع
buffer	منطقة تخزين صغيرة مؤقتة
byte	ثمانية
call	استدعاء
case	حالة
change	تغيير
character	محرف
circle	دائرة
close	إغلاق
code	ترميز

compiler	مترجم غير مباشر
compiler directives	موجهات المترجم
computer	حاسوب
condition	شرط
constant	ثابت
convolution	ثني
copy	نسخ
data	بيانات أو معطيات
data structure	بنية معطيات
debug	تنقيح
declaration	تصريح
definition part	جزء التعريف
delay	تأخير
delete	مسح
digital	رقمي
display	عرض
display memory	ذاكرة العرض
dos (disk operating System)	نظام تشغيل
driver	سواقة
editor	محرر نصوص
element	عنصر
end	نهاية
end of file (eof)	نهاية الملف

enumerated types	نمط المعطيات المجالي
exit	خروج
exponential	أسي
expression	تعبير
external	خارجي
false	غير محقق (خطأ)
field	حقول
file	ملف
file variable	متحول ملف
floating point	الفاصلة العائمة
flow chart	مخطط تدفقي
formal parameter	معاملات شكلية
format	واصفات
function	تابع
global variable	متحولات عامة
graph	رسم بياني
graphic	بياني
graphic adapter	محول بياني
hard ware	كيان مادي
help	مساعدة
high- level language	لغة عالية المستوى
Implementation	تنفيذي
index	دليل

information	معلومات
Initialization	تهيئة
input	إدخال
integer	صحيح
Interface	
interpreter	مترجم فوري
key board	لوحة مفاتيح
label	علامة
level	مستوى
library	مكتبة
line	خط
local	موضعي
local Variable	متحول محلي
logical variable	متحول منطقي
low- level language	لغة منخفضة المستوى
machine language	لغة الآلة
main program	برنامج رئيسي
matrix	مصفوفة
maximum	قيمة عظمى
memory	ذاكرة
minimum	قيمة دنيا
name	أسم
new	جديد

open	فتح
operating system	نظام تشغيل
operator	عملية
option	خيار
output	إخراج
overflow	فائض
parameter	معامل - وسيط
pass	اجتياز - مرور
path	مسار
pixel	نقطة مضيئة (من نقاط الشاشة)
pointer	مؤشر
procedure	إجراء
procedure call	استدعاء إجراء
program	برنامج
programming	برمجة
Ram (Random Access Memory)	إختصار لذاكرة الوصول لعشوائي
real	حقيقي
record	سجل
register	مسجل
rename	إعادة تسمية
repeat	تكرار
reserved word	الكلمة المحجوزة

result	نتيجة
rewrite	إعادة الكتابة
run	تنفيذ
save	تخزين
set	مجموعة
stack	تكديس
start	البداية - إنطلاق
statement	تعليلة
stop	توقف
string	شريط محرفي
subprogram	برنامج جزئي
subroutine	برنامج جزئي
symbol	رمز
syntax error	خطأ بالصياغة (فني)
table	جدول
text	نص
tools	أدوات
true	محقق
type	نمط
typed Files	ملفات محددة نوع المعطيات
unit	وحدة
untyped Files	ملفات غير محددة نوع المعطيات
value	قيمة

window	نافذة
write	كتابة





المراجع العربية

- 1- الدكتور وجدي عصام عبد الرحيم : البرمجة بلغة باسكال
- 2- الدكتور رياض طيفور: البرمجة والجيوإحصاء- منشورات جامعة دمشق 2005.

منشورات جامعة دمشق، 2006

3- ورنر فييل : تريوباسكال 7 (الدليل الكامل)

شعاع للنشر والعلوم ، 1995

4- الدكتور علي جمال الدين - الخوارزميات (1)- منشورات جامعة دمشق 1991.

5- الدكتور علي جمال الدين، الدكتور مُجَّد صبح - الخوارزميات (2)- منشورات جامعة دمشق 1991.

المراجع الأجنبية

- 1- Thomas Kirste: Einführung in die Programmierung, die Sprache „Pascal“

- 2- Ulrike Böttcher, Christian Münster Programmierung–
Grundlagen.
HERDT-Verlag für Bildungsmedien GmbH,
Bodenheim,2006.
- 3- Abolrous, Sam A.: Learn Pascal
Wordware Publishing, Inc., USA, 2000.
- 4- Karacorlu Tuba :The programming language PASCAL
Universität : Technische Universität Wien.
- 5- P. Böhme: Programmiersprache Pascal
Martin-Luther-Universität Halle-Wittenberg, 1996.
- 6- Stephen k.O'Brien: Turbo Pascal 6 (TheComplete
Reference)
Osborne McGraw-Hill-USA,1991.
- 7- Ed Mitchell: The Turbo Pascal Reference
A Freeware Book, Que, Corporation, 1992.
- 8- Marco Cantù: Essential Pascal
2nd Edition,(Version 2.01), March 2003
www.marcoantu.com.
- 9- U. Böttcher,C. Münster Programmierung Grundlagen
HERDT-Verlag für Bildungsmedien GmbH, Bodenheim,
2006.
- 10- k. Georgiadis: Turbo Pascal 4
Verlag GmbH- Germany,1989.

11- E. Hering, J. Rasch: Turbo Pascal 5 fuer Naturwissen-
schaftler und Ingenieure.
Braunschweg- Germany, 1989.

12- G. Paulin: Turbo Pascal
VEB, verlag, Berlin 1988.

المدقق اللغوي:

الأستاذ الدكتور علي أبو زيد



حقوق الطبع والترجمة والنشر محفوظة لمديرية الكتب والمطبوعات