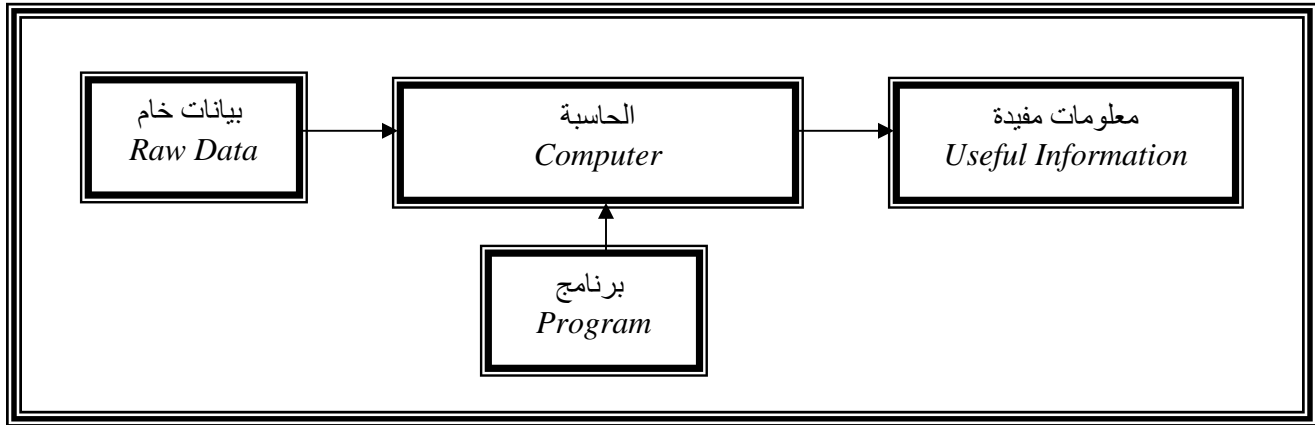


الحاسبة Computer :

جهاز متعدد الأغراض *Multipurpose Machine* وهي بصورة عامة أساسية مجموعة من المكونات الإلكترونية التي تتقبل البيانات الخام (الأولية) وباستعمال برنامج تنتج معلومات مفيدة.



الشكل يوضح أنّ الحاسبة هي جهاز معالجة البيانات *Data Processing Machine*

مزايا الحاسبة Computer Properties :

1. السرعة العالية جداً: *Very High Speed*
تقاس سرعة الحاسبة بمقياس المليون تعليمة خلال الثانية *MIPS* وهي مختصر لـ *Million Instruction Per Second*

2. الدقة العالية: *Accuracy*
القابلية على إجراء العمليات الحسابية والمنطقية وعمليات المقارنة وعمليات أخرى وبدون خطأ، وإن وجدت أخطاء تكون أما من البيانات الأولية أو من البرنامج.

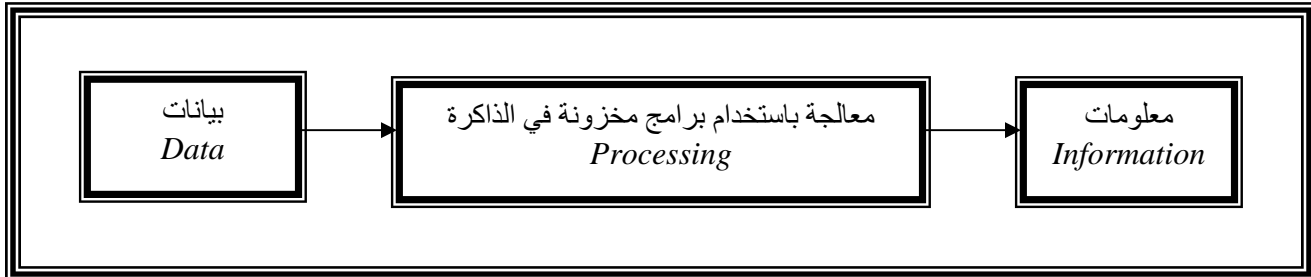
3. الخزن العالي: *Vast Storage*
قابلية خزن واسترجاع وتداول كميات كبيرة جداً من البيانات. يقاس الخزن بوحد *BYTE* حيث يستطيع كل *byte* خزن رمز واحد فقط.

مراحل مقارنة بسيطة بين الإنسان والحاسبة:

من حيث	الحاسب	الإنسان
سرعة المعالجة	فائقة السرعة	بطيء نسبياً
تحمل الإستمرار بالعمل	عالي	ضعيف
الذاكرة والإسترجاع	قوية وسريعة	ضعيفة وبطيئة نسبياً
الدقة في إجراء العمليات	لا يخطئ	قابل للخطأ
إتباع التعليمات	مثالي	غير مثالي
التصرف في ظروف جديدة	غير موجود	جيد
التعلم بالخبرة	غير موجود	جيد

مراحل معالجة البيانات: Data Processing Stages

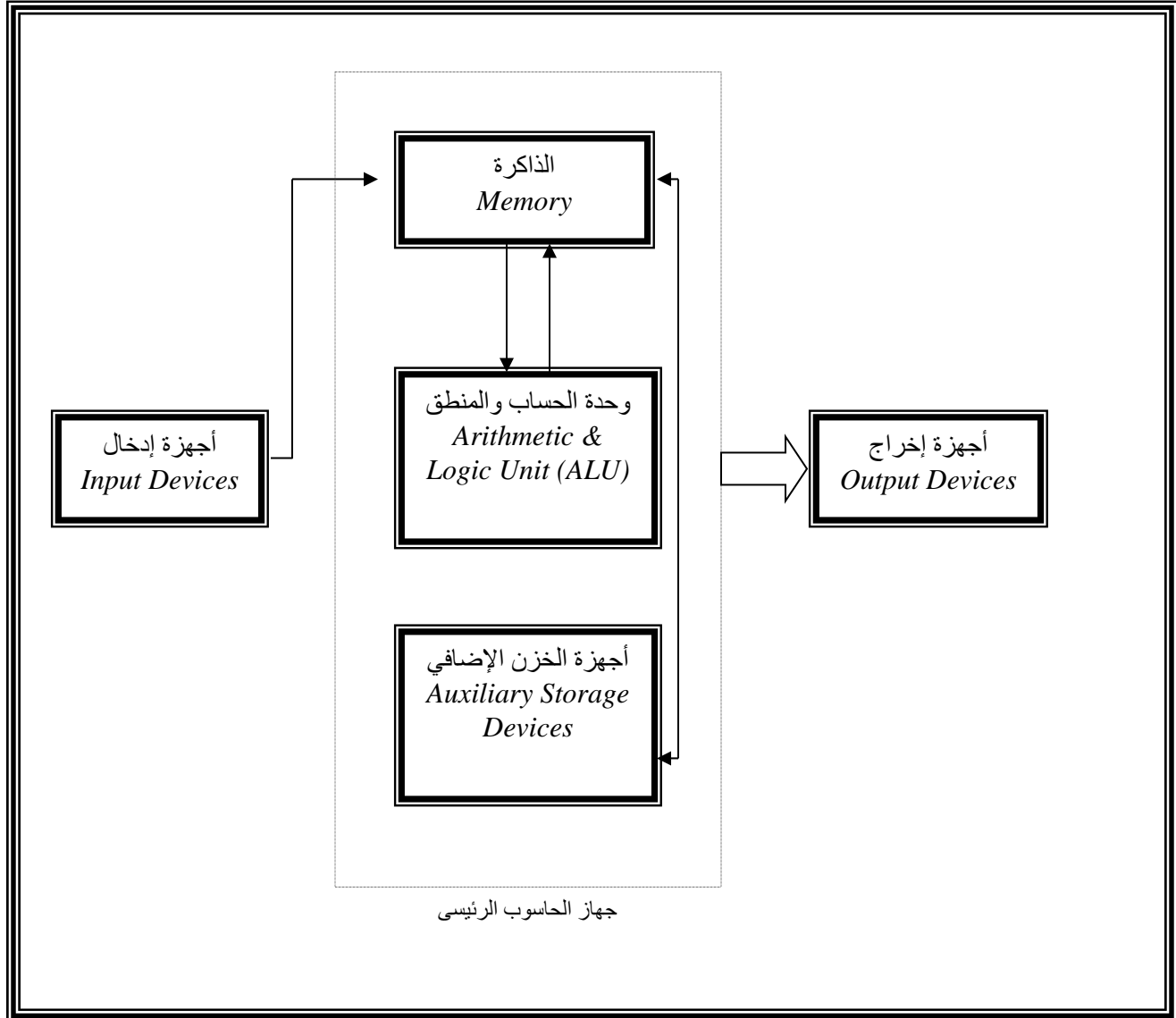
READ إقرأ
PROCESS عالج
WRITE أكتب



الشكل يوضح مراحل معالجة البيانات Data Processing Stages

عملية معالجة البيانات Data Processing Operation

تتم عملية معالجة البيانات كما موضح بالشكل التالي:



الشكل يوضح عملية معالجة البيانات Data Processing Operation

توضيح فقرات الشكل:

1. **أجهزة الإدخال: Input Devices**
 باستخدام هذه الأجهزة يتم إدخال البيانات الأولية إلى الحاسوب (أوضح مثال لوحة المفاتيح *keyboard*)

2. **الذاكرة memory**
 الذاكرة تخزن بشكل مؤقت طيلة فترة تشغيل الحاسبة لكن حالما ينقطع التيار الكهربائي عن الحاسوب فإن الذاكرة تصبح خالية من المعلومات، ويقاس حجمها بالـ *Byte*

$Kilo\ Byte\ (KB) = 1024\ Byte$
 $Mega\ Byte\ (MB) = (1024 * 1024)\ Byte$
 $Gega\ Byte = (1024 * 1024 * 1024)\ Byte$

3. **وحدة الحساب والمنطق Arithmetic & Logic Unit (ALU)**
 يتم داخل هذه الوحدة إجراء العمليات الحسابية والمنطقية وعمليات المقارنة حيث يتم أخذ نسخة من البيانات من الذاكرة وتجرى عليها العمليات الحسابية والمنطقية ويعاد ناتج العمليات إلى الذاكرة أيضاً.

4. **أجهزة التخزين الإضافي Auxiliary Storage Devices**
 هناك أنواع عديدة من هذه الأجهزة أهمها:

- **القرص المرن: floppy disk**
 توجد أقراص مرنة قياسها 3.5" إنج وهي المستعملة حالياً وهناك أقراص مرنة قديمة جداً قياسها 5.25 إنج. ويكون هذا القرص سهل الحمل *portable* أي يمكن حمله واستخدامه لنقل البيانات والمعلومات من حاسب لآخر بسهولة.
- **القرص الصلب: hard disk**

ويكون موجوداً داخل جهاز الحاسوب (غير ظاهر خارجياً) وهو مؤلف من مجموعة من الأقراص مرتبطة بعتلة حديدية ولكل قرص وجهان وتوجد مجموعة

من رؤوس القراءة والكتابة بين الأقراص مربوطة بالعتلة الحديدية ويكون داخل جهاز مفرغ من الهواء ومغطى بمواد مغناطيسية لغرض الحفظ. ووحدة قياس حجمه هي **Byte** حيث يحتوي القرص الصلب على عدد كبير من وحدات الخزن **Byte** وباستخدام التشفير الثنائي (1,0) فقط لإدخال البيانات داخل الحاسوب حيث يقسم الـ **Byte** إلى أربع مناطق ثنائية حيث : $1 \text{ Byte} = 8 \text{ Bit}$

5. أجهزة الإخراج **Output Devices**

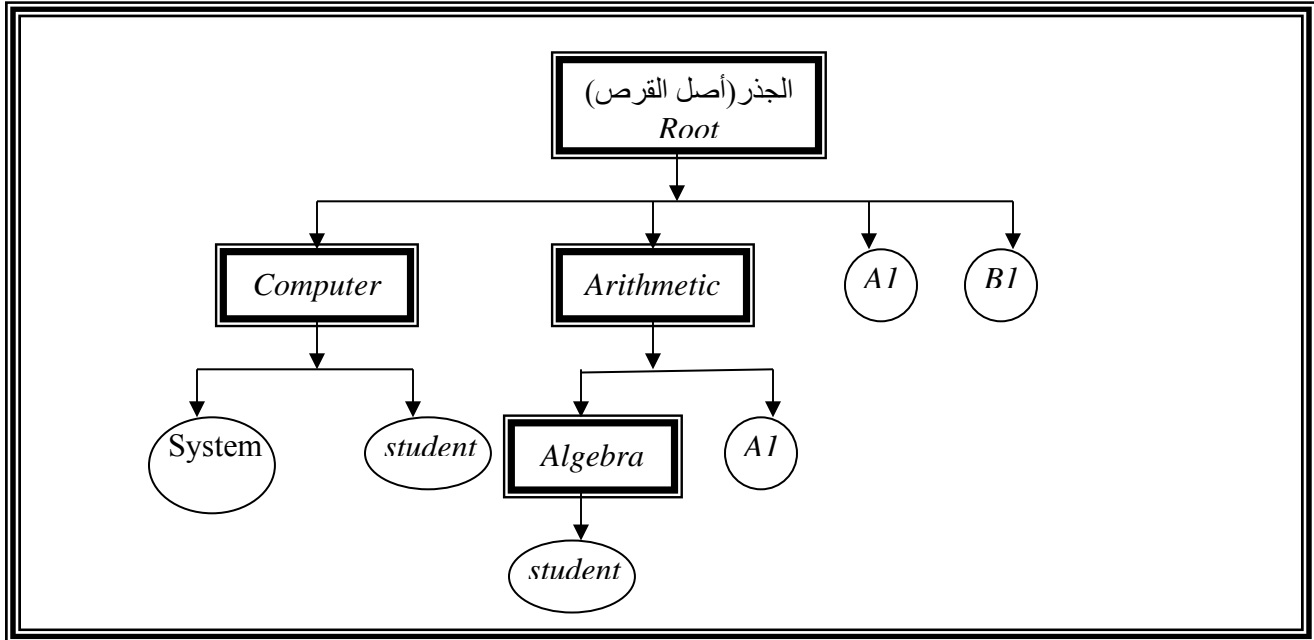
تستخدم لغرض عرض النتائج التي توصل إليها الحاسوب في معالجة البيانات (النتائج النهائية) وأوضح مثال لهذه الأجهزة هي الشاشة **Monitor** والطابعة **Printer**.

نظام التشغيل *Operating System*

هو برنامج يحمل إلى ذاكرة الحاسوب فور تشغيل الحاسوب ويكون الوسيط بين المكونات المادية للحاسوب (*Hardware Interface*) ومستخدم الحاسوب (*User*) ويساعد في تنظيم عمل الحاسوب. مثال عن أنظمة التشغيل هو نظام *MS_DOS* وهو مختصر *Microsoft Disk Operating System* ونظام النوافذ *Windows*.

المفاهيم الأساسية التي يستخدمها نظام التشغيل:

1. الملف : *File* هو الوحدة النهائية التي يتم تخزين البيانات ضمنها.
2. الدليل الفرعي *Directory* : هو وحدة تنظيمية تستطيع تخزين الملفات أو أدلة فرعية أخرى بداخلها.



الشكل يوضح مثال لشجرة من البيانات

تعريف البرمجة Programming Definition

هي مجموعة من الأساليب والإجراءات التي تعالج مشكلة معينة علاجاً منطقياً بهدف الوصول إلى الحل الأمثل وفق أسس قياسية متفق عليها دولياً، وتؤدي البرمجة إلى رفع إنتاجية الحاسوب، وزيادة المرونة باستخدامها، وزيادة اعتمادية حل المشاكل على الحاسوب.

تصاغ تلك الإجراءات على شكل مجموعة من الإيعازات *Instructions* متسلسلة تسلسلاً منطقياً، لتكوّن ما يسمى بالبرنامج *Program* لذا فالبرنامج هو الأسلوب الذي يستخدمه الإنسان لتوجيه الحاسوب من أجل تأدية عمل معين ((البرنامج: هو مجموعة من التعليمات التي تنفذ بتسلسل محدد مسبقاً لتوجيه الحاسوب بصورة تفصيلية حول العمليات المطلوبة للحصول على نتائج معينة))، كما ويعبر عن البرنامج بإحدى اللغات التي يتعامل معها الحاسوب، وإن الشخص الذي يقوم بتصميم وترميز البرنامج يطلق عليه المبرمج *Programmer*.

خطوات برمجة المشكلة Problem Programming Steps

1. تعريف المشكلة : *Problem Definition*

وتشمل هذه الخطوة ما يلي:

- تحديد الهدف *Goal* : أي تحديد ما يراد تحقيقه من حل المشكلة.
- تحديد المدخلات *Input*: وهي البيانات الواجب توفرها من أجل حل المشكلة.
- تحديد المعالجة *Processing*: وتتضمن الإجراءات والأساليب الكفيلة بحل المشكلة في ضوء المدخلات المتوفرة لتوظيفها وصولاً للهدف.
- تحديد المخرجات *Output* : وهي النتائج التي يراد الوصول إليها، والمخرجات هي التي تحدد المدخلات المطلوبة وطبيعة الإجراءات التي من شأنها معالجة المشكلة.

2. تصميم البرنامج *Program Design* :

يمكن عرض صيغة المشكلة بصيغ متعددة، منها ما يكون بصيغة خطوات ذات تسلسل منطقي (الخوارزميات) ومنها ما يكون على شكل مخططات (المخطط الإنسيابي).

3. ترميز البرنامج *Program Coding* :

يتم تحويل صيغة الحل (الخوارزمية والمخطط الإنسيابي) إلى صيغة يمكن للحاسوب أن يتعامل معها. أي إحدى لغات البرمجة (مثل لغة *pascal, fortran, basic,...*) أما تحديد اللغة المراد استخدامها فيتوقف على طبيعة المشكلة المراد حلها، إضافة إلى توفر مترجم اللغة وإمكانياته في الحاسوب المستخدم.

وتنقسم لغات البرمجة إلى :

1. لغات ذات المستوى الواطئ: *Low Level Language*

وتشمل لغة الماكينة *Machine Language* التي يعبر عنها ب (0،1) واللغة التجميعية *Assembly Language* والتي تعتمد على الرموز المعبرة في كتابة البرامج.

ومما يؤخذ على هذه اللغات الصعوبة في كتابة وتحديث برامجها واكتشاف أخطاء البرامج، إضافة إلى إعتمايتها على الماكينة، فعندما ينقل برنامج مكتوباً بهذه اللغات من حاسوب إلى آخر فإن ذلك يتطلب إجراء تغييرات جوهرية عديدة، إلا إن هذه اللغات تمتاز بسرعتها في تنفيذ الإيعازات.


2. لغات ذات المستوى العالي: *High Level Languages*


تمتاز عن لغات المستوى الواطئ بكونها قريبة من لغة المستفيد، واعتماديتها على الماكينة أقل وسهولة تحديث برامجها واكتشاف أخطاءها إذ أن عملية نقل أي برنامج من حاسوب إلى آخر لا يستوجب إلا إجراء بعض التغييرات الطفيفة، إضافة إلى أن البرمجة بهذه اللغات هي أسهل بكثير من لغات المستوى الواطئ (من أمثلتها *pascal, fortran, basic*)


4. تنفيذ البرنامج *Program Execution*:

وتتلخص بعملية إدخال البرنامج إلى الحاسوب عن طريق إحدى وسائل الإدخال ومن ثم الإيعاز للحاسوب بتنفيذ البرنامج فإذا كانت اللغة المستخدمة هي اللغة التجميعية أو إحدى لغات المستوى العالي يتم إستدعاء المترجم *Translator* لتحويلها إلى لغة الماكينة.

ويتخذ المترجم أحد الأنواع التالية :

 المؤلف *Compiler* وهو برنامج يقوم بتحويل البرامج المعبر عنها بإحدى اللغات ذات المستوى العالي (البرنامج المصدر) إلى اللغة التجميعية أو لغة الماكينة (البرنامج الهدف) حيث يقوم المؤلف بترجمة البرنامج بكامله إلى لغة الماكينة وطبع تقرير مفصل بالأخطاء في حالة وجودها.

 المفسر *Interpreter* : ومهمته هي نفس مهمة المؤلف، إلا أنه يقوم على أساس ترجمة الإيعاز وتنفيذه ومن ثم الانتقال إلى إيعاز آخر، على خلاف ما يقوم به المؤلف الذي يترجم البرنامج بشكل كامل قبل تنفيذه، وعليه فإنه هنا الأخطاء تكتشف عند ترجمة كل إيعاز مباشرة.

 المجمع *Assembler* : وهو برنامج يقوم بتحويل البرامج المعبر عنها باللغة التجميعية إلى لغة الماكينة.

5. تصحيح البرنامج *Program Correction*:

عند تنفيذ البرنامج ستظهر مجموعة من الأخطاء التي يرتكبها المبرمج في ترميز البرنامج، والنتيجة عن عدم الالتزام بالشروط والقواعد الواجب التقيد بها أثناء ترميز البرنامج. ويطلق على هذه الأخطاء تسمية الأخطاء اللغوية *Syntax Errors* إن مهمة تحديد تلك الأخطاء هي مسؤولية المترجم والذي يؤشر الخطأ ويقدم وصفاً موجزاً عن الخطأ، ولأجل تنفيذ البرنامج، على المبرمج أن يزيل تلك الأخطاء من البرنامج ومن ثم إعادة تنفيذه حتى يتم تنقيته من الأخطاء.

6. اختبار البرنامج: *Program Testing*

قد تحدث في بعض الأحيان أخطاء منطقية في البرنامج *Logical Errors* تتعلق بمنطق الحل ولا يكشفها المترجم حيث قد يكون البرنامج لغوياً صحيحاً ومنطقياً خاطئاً مما يؤدي إلى ظهور نتائج خاطئة ويتم في هذه الخطوة تنقية البرنامج من الأخطاء المنطقية بالكامل ويتحقق ذلك بإدخال مجاميع إفتراضية مختلفة من البيانات (معروفة النتائج) كمدخلات للبرنامج ومن ثم التحقق من صحة النتائج.

7. توثيق البرنامج *Program Documentation*

بعد إنجاز الخطوات السابقة على المبرمج توثيق البرنامج لغرض تسهيل مهمة تشغيله وتعديله في المستقبل. ونقصد بالتوثيق هو ضرورة وجود قائمة بالبرنامج ووصفاً دقيقاً للمعلومات الداخلة والخارجة، ومخططاً أو خوارزمية توضح منطق البرنامج وقائمة بالأخطاء المتوقعة حدوثها ومرشد للمستخدم *User Guide* يده على كيفية تشغيل البرنامج. إن أسلوب ترميز البرنامج له أثر كبير في التوثيق، حيث يعتمد عند ترميزه على جعله يتخذ شكلاً مهيكلًا، فيظهر مكوناً من عدة أجزاء، كل جزء يختص بأداء وظيفة معينة وهذا يؤدي إلى سهولة تتبعه ويجعله أكثر وضوحاً.

8. تحديث البرنامج *Program Updating*

وتشتمل هذه الخطوة على الإضافات المستقبلية التي سببها ظهور حالات جديدة لم تؤخذ بعين الاعتبار أثناء تصميم البرنامج، إضافة إلى معالجة الأخطاء التي لم تعالج في مرحلة الاختبار. هذا ونصعب عملية تحديث البرنامج إذا لم يكن موثقاً بشكل جيد.

الملفات Files :

يستخدم الملف في لغة Pascal لخرن المدخلات أو المخرجات أو كليهما

الفائدة من استخدام الملفات

1. أن الإدخال المتكرر لنفس مجموعة المدخلات ضمن تطبيقات مختلفة قد يحتمل الخطأ في الإدخال لذا من الأفضل خزن بيانات الإدخال داخل ملف نصي text file.
2. مخرجات البرامج تظهر على شاشة التنفيذ وتختفي حال غلق الشاشة أما عند خزنها على الملف فإنها ستبقى محفوظة لفترة أطول.
3. عند خزن المدخلات والمخرجات على الملفات يمكن مشاهدة نواتج تنفيذ البرنامج دون الحاجة لإعادة تنفيذه (وهذا مفيد جدا في حالة البرامج الضخمة التي تأخذ وقت طويل في التنفيذ).

تعريف الملف النصي text files:

- هو ملف تعاقبي sequential file يخزن تعاقب من الخيوط الرمزية strings متغيرة الطول.
- كل سطر ينتهي بإشارة end_of_line، الدالة () EOLN تعيد true إذا كان المؤشر في نهاية السطر وإلا فإنها تعيد false.
- الملف بأكمله ينتهي بإشارة end_of_file الدالة () eof تعيد true إذا كان المؤشر عند نهاية الملف وإلا فإنها تعيد false.

عند التعامل مع الملفات يجب مراعاة ما يلي (الفرق بين المصفوفة والملف):

1. موقع السطر المعطى في الملف لا يمكن حسابه ولذلك الملف النصي يجب أن يعالج دائما تعاقبيا ابتداءً من السطر الأول.
2. القراءة والكتابة لا يمكن أن تتم سوياً في نفس الوقت بخلاف المصفوفة.
3. حجم الملف يكون متغيراً وليس ثابتاً كالمصفوفة.

التعامل مع الملفات داخل برامج pascal :**1. اسم الملف :**

في لغة pascal اسم الملف يتكون من 1- 8 رموز متبوعة بإضافات إختيارية تدل على نوع الملف على أن لا تزيد عن 3 رموز مثل (phone.dir)
اسم الملف أيضاً يمكن أن يسبق باسم الجذر الذي سيوضع عليه مثلاً
(A:\phone.dir)

2. متغيرات الملف وإجراء Assing :

يستخدم متغير ملف من النوع القياسي text حيث ينسب اسم الملف الفعلي لهذا المتغير في بداية المعالجة عن طريق الإجراء Assign.

3. الإعلان عن متغير الملف:

Var

Filevar:text;

يستخدم النوع القياسي text لتعريف متغير نوعه ملف حيث filevar يمثل أي identifier مقبول ضمن البرنامج مثلاً:

Var

Infile, outfile :text;

قبل أن تتم أي معالجة على الملف يجب أن ينسب المتغير إلى ملف قرصي(مخزون على القرص) هذه العملية تتم باستدعاء الإجراء ASSIGN() كالتالي

ASSIGN (FileVar , 'FileName') ;

حيث :

fileVar متغير ملف معلن عن نوعه text ، و fileName اسم ملف فعلي قانوني مثلاً

Assign (f , 'Phone.Dir') ;

ينسب للمتغير f الملف Phone.Dir كما يمكن أن يكون الاستدعاء كالتالي
Assign (FileVar , str) ;

حيث str هو تعبير نصي يحتوي على اسم الملف القانوني

كمثال

```

var
    f:text ; S:string ;
begin
    s:='Phone.dir' ;           {OR}      {readln(s);}
    Assing( f, s);

```

ملاحظة

عبارة القراءة تفيد في تغيير اسم الملف لكل تنفيذ

4. تكوين ملف جديد:-

عملية الكتابة على الملف تعني بالفعل تكوين ملف جديد يتم عن طريق الخطوات التالية:

```

Assign (filevar, 'filename'); -----a.
Rewrite(filevar); -----b.
-----
-----statements;----- ;-----c.
-----
Close(filevar); -----d.

```

حيث:

a : تنسيب اسم ملف لمتغير.

b : rewrite هو إجراء لتكوين ملف جديد فارغ مهيأ للكتابة بإسم 'filename' للكتابة فقط (كمخرجات مثلا) ولا يمكن أن نقرأ منه إلا بعد غلقه وفتحه للقراءة مرة ثانية.

وإذا كان الاسم المعطى filename هو لملف مخزون على القرص عندئذ سيلغى الملف الموجود أوتوماتيكيا ويكون الملف الجديد بنفس الاسم .

c : التعليمات : يمكن أن تكون أي تعليمة من تعليمات لغة Pascal ، عادة في هذه التعليمات ناتج العمليات الحسابية سيخزن ضمن الملف على القرص المكون لهذا الغرض وهذا يتم من خلال الإجراءين:

```

Write(filevar, outputlist);   or
Writeln(filevar, outputlist);

```

وبذلك سترسل النتائج إلى الملف المخزون المنسب للمتغير.

d : close هو إجراء لغلق الملف القرصي filename المنسب للمتغير filevar للإستخدام في وقت لاحق. هذا الإجراء واجب استخدامه وإلا فإن بعض النتائج ستكون مفقودة.

5. فتح ملف لأغراض القراءة :

يتم عن طريق الخطوات التالية:

```
Assign (filevar, 'filename'); -----a.
Reset(filevar); -----b.
-----statements;-----c.
Close(filevar); -----d.
```

حيث:

- a : تنسيب اسم ملف لمتغير.
- b : reset هو إجراء لفتح ملف موجود فعلاً بإسم 'filename' لأغراض القراءة فقط (كمدخلات مثلاً) ولا يمكن الكتابة عليه إلا بعد غلقه وفتحه للكتابة مرة ثانية.
- c : التعليمات : يمكن أن تكون أي تعليمة من تعليمات لغة Pascal شرط أن تتضمن عمليات القراءة من الملف وهذا يتم من خلال الإجراءين:
- ```
read(filevar, inputlist); or
readln(filevar, inputlist);
```

d : close هو إجراء لغلق الملف القرصي filename المنسب للمتغير filevar .

مثال: أكتب برنامج إدخال المصفوفة  $a(3,5)$  وكتابتها على الملف 'infile.pas' ثم قراءتها من نفس الملف وتدوير أعمدتها باتجاه اليمين ثم طباعة المصفوفة الناتجة على الملف 'outfile.pas'. استخدم البرامج الفرعية المناسبة.

*Program Rotation;*

*Uses*

*Wincrt;*

*Const*

*M = 3 ; N = 5;*

*Type*

*Arr = array [1.. m, 1.. n] of integer;*

*Var*

*A,b :arr; f1,f2 :text;*

*{\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$}*

*{\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$}*

*Procedure readmat(var x:arr);*

*Var*

*I,j :integer;*

*Begin*

*Writeln (' enter the matrix');*

*For I := 1 to m do*

*For j:= 1 to n do*

*Readln(x[I,j]);*

*End;*

*{\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$}*

*Procedure WriteOnFile ( x:arr; var f:text);*

*Var*

*I, j : integer;*

*Begin*

*Rewrite ( f);*

*For i:= 1 to m do begin*

*For j:= 1 to n do*

*Write (f, x [I,j]);*

*Writeln(f);*

*End;*

*Close(f);*

*End;*

```

{$$$}

```

```

Procedure ReadFromFile (var x:arr; var f:text);

```

```

 Var
 I, j : integer;
 Begin
 Reset (f);
 For i:= 1 to m do begin
 For j:= 1 to n do
 read (f, x [I,j]);
 readln(f);
 End;
 Close(f);

```

```

End;

```

```

{$$$}

```

```

Procedure rotate (var x:arr);

```

```

 Var
 I, j, t : integer;
 Begin
 For i:= 1 to m do begin
 T :=x [I,n];
 For j:= n DownTo 2 do
 x[I,j] := x[I, j-1];
 x[I,1]:=T;
 end;
 end;

```

```

{$$$}

```

```

{$$$}

```

```

Begin {main}

```

```

 Assign(f1, 'infile.pas');
 Assign(f2, 'outfile.pas');
 Readmat(a);
 WriteOnFile(a, f1);
 ReadFromFile(b, f1);
 Rotate(b);
 WriteOnFile(b, f2);

```

```

End. {main}

```



## الخوارزميات Algorithms

### تعريف definition

الخوارزمية عبارة عن خطوات متسلسلة تصف بصورة دقيقة وبدون أي غموض مجموعة من الخطوات الرياضية والمنطقية اللازمة لحل مشكلة ما، ويمكن تصميم أكثر من خوارزمية لحل نفس المشكلة حيث أن الأسلوب أو الطريقة تختلف من واحدة إلى أخرى وكذلك الحال بالنسبة للكفاءة التي تختلف هي الأخرى بحسب الطريقة.

### مقومات الخوارزمية

عند تصميم أي خوارزمية يجب أن تتوفر فيها المقومات التالية:

1. البداية **start** يجب أن يكون لكل خوارزمية بداية (عبارة عن إسم الخوارزمية)
2. التعريف **definition** يجب أن تكون كل خطوة من خطوات الخوارزمية واضحة، وخالية من الغموض ومعرفة بدقة.
3. المدخلات **inputs** تحتوي كل خوارزمية على مدخل واحد أو أكثر أو قد لا تحتاج الخوارزمية إلى مدخلات، وتمثل المدخلات البيانات الواجب توفرها لضمان حل المشكلة.
4. المخرجات **outputs** يجب أن تحتوي كل خوارزمية على مخرجات (واحد على الأقل) وتمثل الخوارزمية نتيجة حل المشكلة التي صممت الخوارزمية لأجلها.
5. النهاية **end** يجب أن تنتهي الخوارزمية عند نقطة معينة أي أنها يجب أن تصل إلى حالة النهاية أو التوقف بعد أن يتم تنفيذ عدد معين من الخطوات.
6. الكفاءة **effectiveness** يمكن تصميم عدة خوارزميات لحل نفس المشكلة وعلينا أن نختار منها الخوارزمية الكفوءة، إذ أن الكفاءة تتحدد بوقت التنفيذ **execution time** والمجال التخزيني **storage space** الذين تتطلبهما الخوارزمية عند تنفيذها على الحاسوب (بعد تحويلها إلى برنامج) فكلما كان وقت التنفيذ والمجال التخزيني أقل كلما زادت كفاءتها، كما تزداد كفاءة الخوارزمية كلما كانت واضحة وبسيطة وخالية من التعقيد.

## هيكل الخوارزمية *Algorithm Structure*

تتخذ الخوارزميات الصيغة الموحدة الآتية

|                    |                  |                                                                                   |
|--------------------|------------------|-----------------------------------------------------------------------------------|
| (1)<br>(Algorithm) | (2)<br>(Name)    | (3)<br>[ وصف الغرض من الخوارزمية مع وصف المدخلات والمخرجات والمتغيرات المستخدمة ] |
| (4)<br>Step (0):   | (5)<br>[       ] | (6)<br>Statement(s)                                                               |
| Step (1):          | [       ]        | Statement(s)                                                                      |
| .                  | .                | .                                                                                 |
| .                  | .                | .                                                                                 |
| .                  | .                | .                                                                                 |
| Step (n):          | [       ]        | Statement(s)                                                                      |

الشكل يوضح هيكل الخوارزمية *Algorithm Structure*

وفيما يلي توضيح لفقرات الشكل:

- (1) يجب أن تبدأ الخوارزمية بكلمة *Algorithm*.
- (2) لكل خوارزمية إسم يلي كلمة *Algorithm*، ويفضل أن يعكس طبيعة عمل الخوارزمية.
- (3) يوضع داخل الأقواس وصف مختصر لعمل الخوارزمية ومدخلاتها ومخرجاتها والمتغيرات المستخدمة.
- (4) تستخدم لترقيم وتعريف وتمييز كل خطوة عن الخطوات الأخرى.
- (5) الأقواس يوضع بداخلها شرح مختصر لطبيعة عمل الخطوة.
- (6) تتكون الخطوة من عبارة واحدة أو عدة عبارات تتخذ كل منها إحدى الصيغ الآتية:

### التنسيب (الإحلال) *Assignment*

يستخدم الرمز " $\leftarrow$ " للتعبير عن عملية إحلال قيمة معينة محل أخرى، ففي العبارة التالية يتم إحلال قيمة الطرف الأيمن  $B$  محل قيمة الطرف الأيسر  $A$ :

$$A \leftarrow B$$

حيث:

$A, B$  يمثلان إسمان لموقعين في وحدة الذاكرة الرئيسية ويطلق على كل منهما إسم المتغير *Variable*. العبارة أعلاه تنقل نسخة من محتويات الموقع  $B$  إلى الموقع  $A$  بينما تبقى محتويات الموقع  $B$  كما هي.

أمثلة:

- إذا كانت  $A=10$  ،  $B=20$  فإن العبارة  $A \leftarrow B$  تعني أن نسخة من موقع  $B$  ستحل محل قيمة  $A$  فتصبح قيمة  $A$  هي 20 بدلاً من 10.
- العبارة  $C \leftarrow 10$  تعني تخزين القيمة 10 في المتغير  $C$ .
- العبارة  $10 \leftarrow C$  تعتبر خاطئة لعدم توفر موقع لتخزين القيمة.
- إذا كان  $I = 10$  فإن العبارة  $I \leftarrow I+1$  تعني أن قيمة  $I$  ستكون 11.
- إذا كان  $A = 20$  فإن العبارة  $A \leftarrow A+A$  تعني أن قيمة  $A$  ستكون 40 (إضافة قيمة الموقع إلى نفسه وخزن الناتج في نفس الموقع  $A$ ).
- إذا كانت  $C=10$  ،  $B=20$  فإن العبارة  $A \leftarrow B+C$  تعني جمع قيمتي الموقعين  $C, B$  وخزن الناتج في الموقع  $A$  فتصبح قيمة  $A$  تساوي 30.

### التبادل *Exchange*

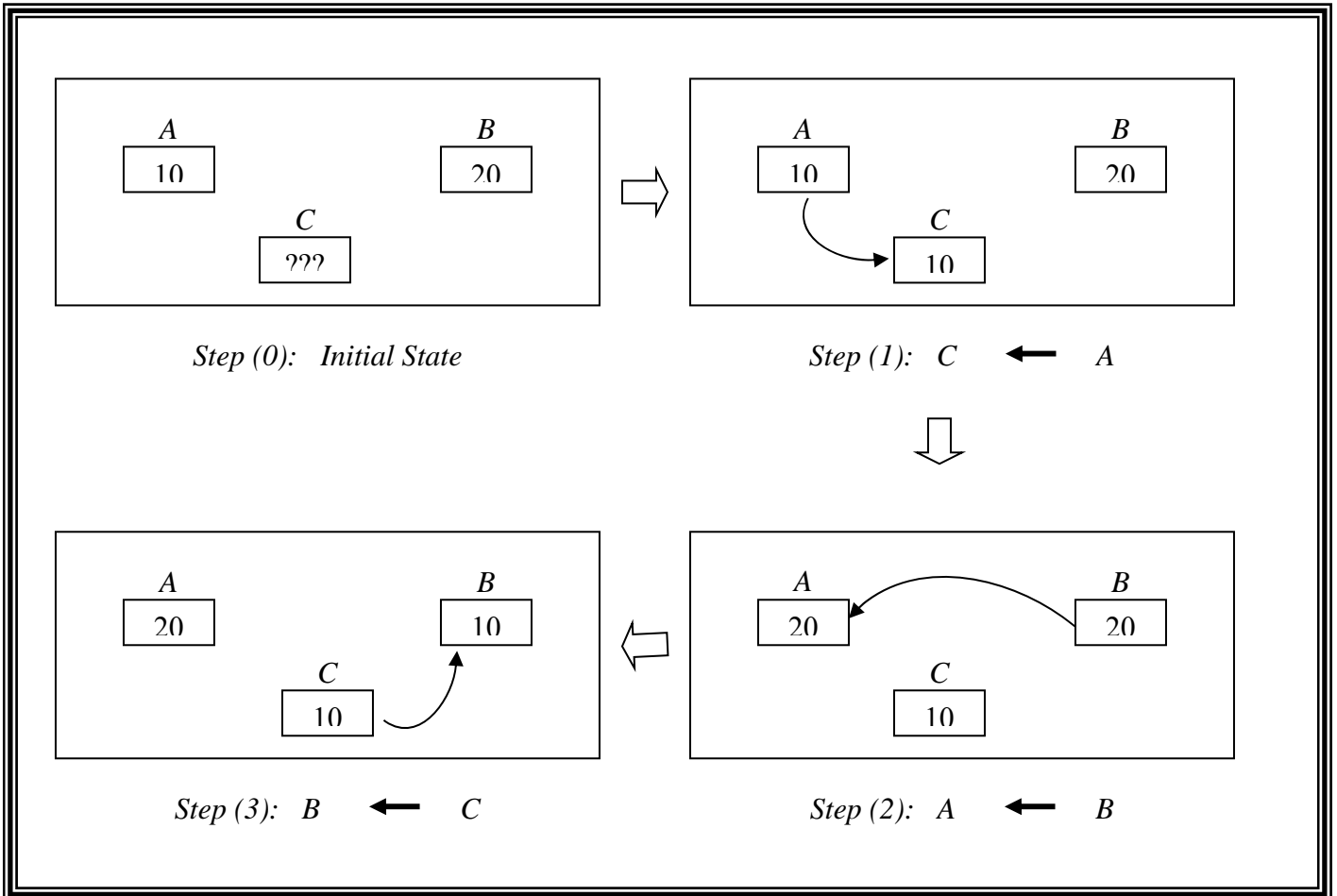
عملية التبادل بين متغيرين هي عملية إحلال قيمتي المتغيرين الواحدة محل الأخرى ويرمز لها رياضياً بالرمز " $\longleftrightarrow$ " ، فعملية تبادل الموقعين  $A, B$  يرمز لها رياضياً بالرمز:

$$A \longleftrightarrow B$$

وتتضمن هذه العملية إنجاز عمليات إحلال بإستحداث موقع تخزيني مؤقت (وسيط) هو الموقع  $C$  وكالتالي:

$$\begin{array}{l} C \leftarrow A \\ A \leftarrow B \\ B \leftarrow C \end{array}$$

حيث  $C$  موقع لتخزين القيمة مؤقتاً. مثال : إذا كانت  $A=10$  ،  $B=20$  فإن  $A \leftrightarrow B$  تتم كما موضح بالشكل التالي:



الشكل يوضح عملية الإستبدال *Exchange Operation*

### عبارة الإدخال : Input Statement

تستخدم كلمة *Input* أو كلمة *Read* متبوعة بأسماء المتغيرات المراد قراءتها للدلالة على إدخال البيانات عن طريق إحدى وسائل الإدخال.

*Read Variable List*

*Input Variable List*

إن العبارة *Input A,B* تشير إلى حجز موقعين في الذاكرة إسميهما  $(A,B)$  ، وعند إدخال البيانات يجب إدخال قيمتين فقط كل منهما تخصص إلى موقع، فإذا افترضنا إدخال  $(10,20)$  تصبح محتويات الموقعين  $A=10, B=20$ .

### عبارة الإخراج : Output Statement

تستخدم كلمة *Print* أو كلمة *Write* متبوعة بأسماء المتغيرات المراد طبعتها، للدلالة على إخراج المعلومات أو طبع النتائج.

*Print Variable List*

*Write Variable List*

إن العبارة *Print C,D* تشير إلى طبع قيمتي موقعين في الذاكرة إسميهما  $(C,D)$  على إحدى وسائل الإخراج ليتسنى للمبرمج أن يعرف نتيجة التنفيذ وهنا سوف تطبع قيمتان الأولى للمتغير  $C$  والثانية للمتغير  $D$ .

### عبارة الإنهاء : End Statement

تستخدم كلمة *End* أو كلمة *Stop* للدلالة على انتهاء أو توقف الخوارزمية، فإذا وصل التنفيذ عند هذه العبارة فإن الخوارزمية تتوقف ولا تنفذ أي عبارة تأتي بعد هذه العبارة.

### عبارة الانتقال غير الشرطي *Unconditional Jump Statement* :

تتخذ خطوات الخوارزمية حسب التسلسل المحدد ونحتاج في بعض المشاكل إلى تغيير مسار التنفيذ بالانتقال إلى خطوة سابقة أو لاحقة (لكن ليس التالية) أو تكرار تنفيذ مجموعة من الخطوات وتستخدم عبارة (GoTo) متبوعة برقم الخطوة المراد انتقال التنفيذ إليها لهذا الغرض.

GoTo (No.)

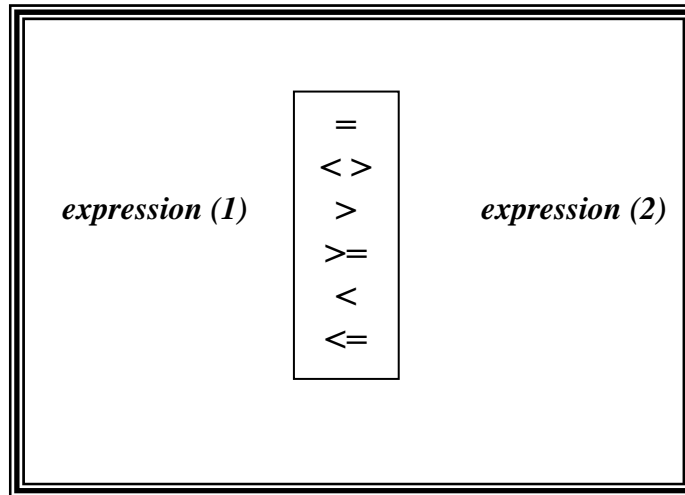
مثلاً العبارة التالية تعني انتقال سير التنفيذ للخطوة (3) *Step(8): GoTo step(3)*

### عبارة الانتقال الشرطي *Conditional Jump Statement* :

هذه العبارة تعني انتقال سير التنفيذ تبعاً لتحقيق شرط معين وتأخذ الصيغة التالية

*If C : Statement(s)*

حيث أن *C* تمثل شرطاً إن تحقق فسيؤدي إلى تنفيذ العبارة (أو العبارات) التالية له وتهمل تلك العبارة ما لم يتحقق ذلك الشرط وينتقل التنفيذ إلى العبارة التالية حسب تسلسل الخوارزمية وتكون صيغة الشرط بإحدى الصيغ التالية:



الشكل يوضح صيغ الشرط *Condition Forms*

حيث:

|                |     |                  |    |
|----------------|-----|------------------|----|
| $C = 1$        | مثل | يساوي            | =  |
| $A <> B$       | مثل | لا يساوي         | <> |
| $K-1 > 1$      | مثل | أكبر من          | >  |
| $K+B \geq C+1$ | مثل | أكبر من أو يساوي | >= |
| $M/5 < 2$      | مثل | أصغر             | <  |
| $I \leq 0$     | مثل | أصغر من أو يساوي | <= |

ويرمز لحالة تحقق الشرط ( صحيح *True* ) ولعدم تحققه ( خطأ *False* ) فإذا أطلقنا على هذا الشرط اسم الشرط البسيط، فإنه يمكن ربط شرطين بسيطين ليكونان شرطاً مركباً باستخدام العلاقات المنطقية (*AND, OR, Not*) ولتوضيح عمل هذه العلاقات نفترض توفر شرط مركب يتكون من شرطين بسيطين فقط، مثلاً:

$$C1 : A > B \quad , \quad C2 : X \leq Y$$

الجدول التالي يوضح دلائل الشروط المركبة حيث *F* تعني *False* و *T* تعني *True* ويطلق على هذا الجدول اسم جدول الحقائق *Truth Table* :

جدول الحقائق *Truth Table*

| $C1$<br>$A > B$ | $C2$<br>$X \leq Y$ | $C1 \text{ AND } C2$ | $C1 \text{ OR } C2$ | $NOT \ C1$ |
|-----------------|--------------------|----------------------|---------------------|------------|
| <i>T</i>        | <i>T</i>           | <i>T</i>             | <i>T</i>            | <i>F</i>   |
| <i>T</i>        | <i>F</i>           | <i>F</i>             | <i>T</i>            | <i>F</i>   |
| <i>F</i>        | <i>T</i>           | <i>F</i>             | <i>T</i>            | <i>T</i>   |
| <i>F</i>        | <i>F</i>           | <i>F</i>             | <i>F</i>            | <i>T</i>   |

من الجدول السابق نستنتج:

- **AND**: تجعل قيمة الشرط المركب (صحيح *True*) في حالة تحقق الشرطين البسيطين، وتجعل قيمته (خطأ *False*) في حالة عدم تحقق أحد الشرطين أو كليهما.
- **OR**: تجعل قيمة الشرط المركب (صحيح *True*) في حالة تحقق أحد الشرطين البسيطين أو كليهما، وتجعل قيمته (خطأ *False*) في حالة عدم تحقق الشرطين معاً.
- **NOT**: تجعل قيمة الشرط المركب (صحيح *True*) في حالة عدم تحقق الشرط البسيط، وتجعل قيمته (خطأ *False*) في حالة تحقق الشرط.



## المخططات الانسيابية Flowcharts

### تعريف definition

المخطط الانسيابي هو تمثيل بياني أو صوري لخطوات حل المشكلة، فهو بمثابة صورة أخرى للخوارزمية، لذا فإن مقومات المخطط الانسيابي هي نفسها مقومات الخوارزمية الموضحة سابقاً.

### رموز المخطط الانسيابي :

يتكون المخطط الانسيابي من مجموعة رموز بيانية متفق عليها دولياً وترتبط فيما بينها بأسمهم تشير إلى اتجاه سير التنفيذ.

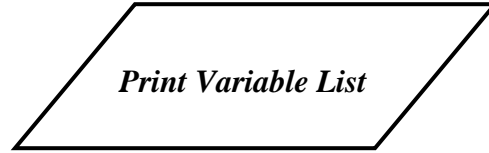
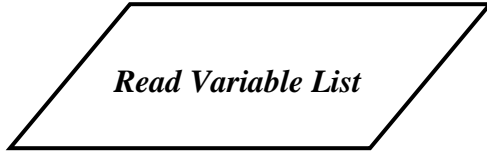
#### 1. رمز البداية والنهاية *End / Start Symbol*

ويستخدم للدلالة على بداية أو نهاية سير البرنامج. ويأخذ هذا الرمز الشكل البيضوي وكالتالي:



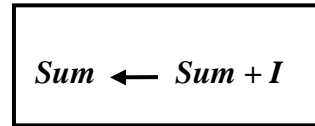
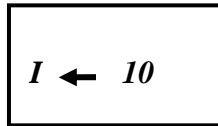
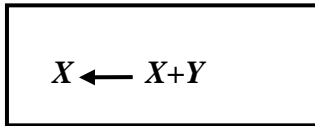
#### 2. رمز الإدخال والإخراج *Input / Output Symbol*

ويستخدم للدلالة على إدخال المعلومات أو إخراجها. في حالة الإدخال تكتب بداخله عبارة الإدخال وفي حالة الإخراج تكتب بداخله عبارة الإخراج. ويأخذ هذا الرمز شكل متوازي الأضلاع.



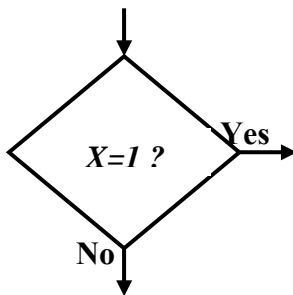
### 3. رمز المعالجة *Processing Symbol*

ويستخدم للدلالة على معالجة المعلومات وحركتها، وتشمل العمليات الحسابية والمنطقية ونقل المعلومات من موقع إلى آخر ويمثل عملية الإحلال ويأخذ هذا الرمز شكل المستطيل.



### 4. رمز القرار *Decision Symbol*

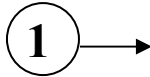
ويستخدم لتحديد اتجاه سير التنفيذ وفقاً لشرط يذكر داخل الرمز، فإذا تحقق الشرط عندئذ يوجه سير التنفيذ بواسطة سهم خارج من رمز القرار ويؤشر عليه بكلمة (*Yes*) أما إذا لم يتحقق الشرط فيوجه سير التنفيذ بواسطة سهم خارج من رمز القرار ويؤشر عليه بكلمة (*No*). ويأخذ هذا الرمز الشكل المعيني. [ ويقوم بنفس عمل عبارة (*If*) في الخوارزمية ].



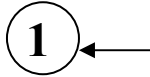
**5. رمز التوصيل Connector Symbol**

ويسمى أيضاً الرابط، وتستخدم الدائرة للتعبير عن هذا الرمز ويوضع داخل الدائرة رقم أو حرف يشير إلى موقع الانتقال ويرتبط بالدائرة شكلين من الأسهم هما:

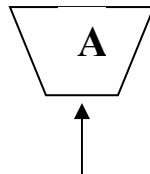
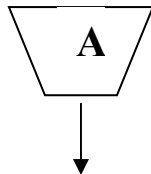
- **السهم الخارج** والذي يشير إلى الموقع الذي تم الانتقال منه



- **السهم الداخل** والذي يشير إلى الموقع المراد الانتقال إليه

**ملاحظة:**

- يجب أن يقابل كل سهم خارج بسهم داخل، وذلك لأن الأول يحدد موقع بداية الانتقال والثاني يحدد موقع نهاية الانتقال.
- قد يستخدم الرمز التالي بدلا من الدائرة المرتبط بها سهم في حالة المخططات الانسيابية التي تتطلب أكثر من صفحة واحدة.



## البرمجة بلغة باسكال Programming With Pascal

### مقدمة Introduction

ظهرت لغة *Pascal* عام 1971 وهي حالياً واحدة من أوسع لغات البرمجة التي تستخدم لتعليم البرمجة، والإقبال على استخدام هذه اللغة يعود لكون قواعدها اللغوية نسبياً سهلة التعلم، كما أن إمكانياتها في كتابة البرامج المهيكلية *structured programs* والتي تمتاز بكونها مقروءة (سهلة القراءة وسهلة الفهم والتحديث) .

إن لغة باسكال هي لغة مهيكلية *structured language* وباستخدامها يمكن كتابة أو بناء برامج عبارة عن تتابع (سلسلة) من أجزاء منفصلة والتي تتناسق مع بعضها لتكوّن حلاً برمجياً أمثلاً للمشكلة المطروحة، وهذا يعود إلى مبدأ (فرّق تسد) حيث يقوم هذا المبدأ على أساس تقسيم المشكلة الكبرى إلى مشاكل أصغر فرعية، وبدلاً من البحث عن حل واسع لمشكلة كبيرة فإنه يتم البحث عن حلول صغيرة لمشاكل صغيرة (مهام جزئية) ثم بناء الحل الكبير الواسع اعتماداً على تنسيق الحلول الصغيرة ليكون حل أمثل للمشكلة الكبيرة الأصلية. إن لغة باسكال هي إحدى لغات البرمجة ذات المستوى العالي لذا فهي تحتاج إلى مترجم ويكون من نوع المؤلف *compiler* لغرض ترجمة البرنامج قبل تنفيذه.

### تخطيط برنامج باسكال The Layout of Pascal Program

يقسم برنامج باسكال إلى عدة مقاطع *segments*، يجب أن يتم مسبقاً تعريف لأنواع البيانات المستخدمة في البرنامج ومن ثم كتابة إيعازات البرنامج الذي سيعمل على هذه البيانات. الصيغة العامة لبرنامج باسكال موضحة بالشكل التالي (سيدرج فيما بعد شرح تفصيلي لكل مقطع).

```

Program ; { Program heading }
Uses ; { Uses clause }
Label ; { Labels }
Const ; { Constants }
Type ; { Types }
Var ; { Variables }
Procedure; { Procedures }
Function; { Factions }
Begin { Main Program }
 Statement; { Statements }

End.

```

الشكل يوضح تخطيط برنامج باسكال *Layout of Pascal Program*

وفيما يلي بعض الملاحظات التوضيحية عن الشكل:

- رأس البرنامج *program heading* : يتبع كلمة *program* إسم للبرنامج يفضل أن يدل على عمل البرنامج.
- في مقطع *Uses* يتم الإعلان عن الوحدات *Units* المستخدمة في البرنامج.
- المقاطع *function, procedure, var, type, const, label* يتم إدراجها في أي ترتيب كان مالم تعتمد أحدهم على الآخر، وإذا اعتمد أحد المقاطع على آخر يجب تقديم المعتمد عليه أولاً لذا يفضل إدراجها بالترتيب الموضح في الشكل أعلاه.
- كل برنامج مكتوب بلغة باسكال يجب أن يحتوي على الأقل جزء واحد *one block* هو جزء البرنامج الرئيسي. (والمقصود بالـ *Block* هو مجموعة من الإيعازات والتي تبدأ بكلمة *Begin* وتنتهي بكلمة *End*) ويحتوي جزء البرنامج الرئيسي على كلمة *Begin* قبل الإيعازات الرئيسية وكلمة *End.* بعد نهاية الإيعازات, مع ضرورة إتباع كلمة *End* بالنقطة *period* (.) للدلالة على نهاية البرنامج.
- يحتوي البرنامج على مجموعة من الإيعازات *statements* والتي يتم تنفيذها بالترتيب المحدد عند تنفيذ البرنامج.

- كل إيعاز *statement* يجب أن ينتهي بفارزة منقوطة (:) وبما أن لغة باسكال تتمتع بصياغة حرة (معنى الصياغة الحرة أن لغة باسكال تسمح بالجمل والإيعازات الطويلة دون تقييد معين فمثلاً يمكن إضافة فراغات أو ترك أسطر فارغة دون أي تقييد وذلك لأجل جعل البرنامج مقروء وسهل الفهم، حيث أن الجمل والإيعازات الطويلة يمكن أن تستمر ليتم تكملتها على السطر التالي) ولذلك فإن الفارزة المنقوطة من الضروري وجودها في نهاية الإيعاز لكي نعلم المؤلف أن الإيعاز قد انتهى.

## مكونات برامج لغة باسكال *Pascal Programs' Components*

### الرموز المستخدمة في لغة باسكال *Used Symbols in Pascal*

أولاً: الأحرف الإنكليزية *letters* ، وعددها 26 حرف هي:

الحروف الكبيرة *A, B,....., Z*  
الحروف الصغيرة *a, b,....., z*

ثانياً: الأرقام *digits* ، وعددها 10 *0, 1, ....., 9*

ثالثاً: الرموز الخاصة *special characters* ، وهي الموضحة في الجدول التالي:

### جدول يوضح الرموز الخاصة *Special Symbols*

| الرمز | الإسم      | الغرض من استخدامه |
|-------|------------|-------------------|
| +     | (PLUS)     | علامة الجمع       |
| -     | (MINUS)    | علامة الطرح       |
| *     | (ASTERISK) | علامة الضرب       |
| /     | (SLASH)    | علامة القسمة      |

|                                                                                                                   |                          |         |
|-------------------------------------------------------------------------------------------------------------------|--------------------------|---------|
| يساوي                                                                                                             | (EQUAL)                  | =       |
| أصغر من                                                                                                           | (LESS THAN)              | <       |
| أكبر من                                                                                                           | (GREATER THAN)           | >       |
| لا يساوي                                                                                                          | (LESS THAN/GREATER THAN) | <>      |
| أصغر من أو يساوي                                                                                                  | (LESS THAN/EQUAL)        | <=      |
| أكبر من أو يساوي                                                                                                  | (GREATER THAN/EQUAL)     | >=      |
| تكون في نهاية البرنامج بعد كلمة End الأخيرة، وكذلك تستخدم في البيانات من سجل record                               | (PERIOD)                 | .       |
| لفصل بين المتغيرات في جزء الاعلان وفي تعليمتي القراءة والكتابة، وكذلك تستخدم للفصل بين فهرسي المصفوفة ذات البعدين | (COMMA)                  | ,       |
| لفصل بين إعلان الدالة ونوعها، وكذلك للفصل بين إعلان أسماء المتغيرات وأنواعها في جزء الاعلان                       | (COLON)                  | :       |
| تفصل بين إيعاز وآخر (نهاية كل إيعاز)                                                                              | (SEMI-COLON)             | ;       |
| لتحديد الرسائل التوضيحية عند الطباعة                                                                              | (SINGLE QUTE)            | '...'   |
| يستخدم عند وجود أنواع بيانات خاصة (مثل القوائم الموصولة)                                                          | (POINTER)                | ^       |
| يستخدم في عبارة التنسيب                                                                                           | (COLON/EQUAL)            | :=      |
| لتحديد فهارس المجموعات والمصفوفات                                                                                 | (LEFT/RIGHT BRACKET)     | [...]   |
| تستخدم في التعابير الحسابية والمنطقية لتمييز أسبقيات التنفيذ ، وكذلك تستخدم للإحاطة بمعاملات الدوال والإجراءات    | (LEFT/RIGHT PARENTHESIS) | (...)   |
| لتحديد التعليقات بداخلها                                                                                          | (MEDIUM PARENTHESIS)     | {...}   |
| لتحديد التعليقات بداخلها                                                                                          | (STARS/ PARENTHESIS)     | (*...*) |
| لتحديد المدى range                                                                                                | (PERIOD/PERIOD)          | ..      |

### المحددات (المعرفات) *Identifiers*

المقصود بالمحددات (المعرفات) إسم البرنامج، أسماء الدوال والإجراءات الفرعية، وكذلك أسماء الثوابت والمتغيرات وتعريفها المستخدمة في برامج باسكال. هذه المحددات يجب أن تبدأ دائماً بحرف تليه أي مجموعة من الحروف والأرقام ويمكن أن يتكون المحدد من حرف واحد فقط ويشترط في إسم المحدد أن لا يحتوي أي فراغات أو رموز أخرى (لا يوجد فرق بين استخدام الحروف الكبيرة أو الحروف الصغيرة، ويمكن استخدام النوعين من الحروف معاً)، ويمكن أن يكون طول المحدد (المعرف) أي عدد من الحروف والأرقام لكن في لغة باسكال القياسية يؤخذ بنظر الاعتبار فقط أول 63 رمز فما دون، ويمكن تمثيل المحددات بالقاعدة التالية:

$$\text{Identifier} = \text{Letter} \{ \text{Letter} / \text{Digit} \}$$

### الكلمات المحجوزة أو الكلمات الدالة *Reserved Words / Keywords*

المقصود بها بعض الكلمات (مكوّنة من مجموعة حروف) تكون ذات دلالة خاصة ندل على إيعازات اللغة ، هذه الكلمات محجوزة من قبل مترجم اللغة (المؤلف) ولا يمكن للمبرمج إستخدامها كمحددات (معرفات)، ويضم الجدول التالي أكثر الكلمات المحجوزة استخداماً مع وصف بسيط للغرض من استخدامها:

#### جدول يوضح الكلمات المحجوزة *Reserved Words*

| الكلمة المحجوزة | الغرض من استخدامها                                                |
|-----------------|-------------------------------------------------------------------|
| AND             | رابط منطقي (يربط بين شرطين بسيطين لتكوين شرط مركّب)               |
| ARRAY           | لتعريف مصفوفة ذات بعدين                                           |
| BEGIN           | للبدء بمقطع من عدد من التعليمات block                             |
| CASE            | للبدء بتعليمة case                                                |
| CONST           | للإعلان عن الثوابت                                                |
| DIV             | للقسمة الصحيحة                                                    |
| DO              | تتبع تعليمة WHILE وتعليمة FOR لأجل التكرار                        |
| DOWNT0          | في تعليمة FOR loop باعتبار أن عدّاد تعليمة FOR يتناقص عند كل دورة |



|                                                                                                                    |           |
|--------------------------------------------------------------------------------------------------------------------|-----------|
| إذا كان التعبير المنطقي في تعليمة IF أعطى إجابة خطأ false عندئذ التعليمات التي تتبع ELSE هي التي ستنفذ             | ELSE      |
| لإنهاء مقطع من عدد من التعليمات block كما تستخدم في تعليمة case وكذلك في جزء الإعلان عن السجلات record declaration | END       |
| للإعلان عن متغير من نوع ملف file                                                                                   | FILE      |
| لتكرار تنفيذ تعليمة أو أكثر عدد من المرات من خلال loop يتحكم بها متغير ضمن مدى معين                                | FOR       |
| للإعلان عن دالة مساعدة للبرنامج                                                                                    | FUNCTION  |
| لنقل سير التنفيذ إلى العنوان المحدد                                                                                | GOTO      |
| لاختبار تعبير منطقي وتنفيذ تعليمات معينة إذا كان التعبير يعطي الإجابة صواب true                                    | IF        |
| لاختبار وجود قيمة معينة ضمن مجموعة محددة مسبقاً                                                                    | IN        |
| للإعلان عن العناوين المستخدمة لنقل سير التنفيذ إليها من خلال تعليمة GOTO                                           | LABEL     |
| لإيجاد باقي القسمة الصحيحة                                                                                         | MOD       |
| لإعطاء قيمة Null لمؤشر pointer                                                                                     | NIL       |
| لنفي قيمة تعبير منطقي                                                                                              | NOT       |
| يستخدم في تعليمة CASE بعد المتغير الخاص بالتعليمة                                                                  | OF        |
| رابط منطقي (يربط بين شرطين بسيطين لتكوين شرط مركب)                                                                 | OR        |
| يستخدم مع التعاريف ARRAY, FILE, RECORD, SET لأجل تجميع مجموعة بيانات لغرض التخزين                                  | PACKED    |
| للإعلان عن إجراء مساعد للبرنامج                                                                                    | PROCEDURE |
| لتكوين رأس البرنامج                                                                                                | PROGRAM   |
| للإعلان عن متغير من نوع سجل record                                                                                 | RECORD    |
| لبداية تكرار تنفيذ تعليمة أو أكثر عدد من المرات من خلال REPEAT/UNTIL loop                                          | REPEAT    |
| للإعلان عن مجموعة                                                                                                  | SET       |
| تتبع التعبير المنطقي بعد تعليمة IF                                                                                 | THEN      |
| في تعليمة FOR loop باعتبار أن عداد تعليمة FOR يتزايد عند كل دورة                                                   | TO        |
| للإعلان عن أنواع المتغيرات الخاصة (غير القياسية)                                                                   | TYPE      |
| لإنهاء تكرار تنفيذ تعليمة أو أكثر عدد من المرات من خلال REPEAT/UNTIL loop                                          | UNTIL     |
| للإعلان عن متغيرات البرنامج                                                                                        | VAR       |
| لبداية تكرار تنفيذ تعليمة أو أكثر عدد من المرات إلى أن تصبح قيمة الشرط false                                       | WHILE     |
| لتحديد متغير من نوع سجل record لكي يستخدم مع مجموعة تعليمات                                                        | WITH      |

### التعليقات Comments

المقصود بها أي تتابع مجموعة من الرموز (حروف أو أرقام أو رموز أخرى) المحصورة بين الرمز { } أو الرمز ( \* ) شرط عدم إحتواء هذه المجموعة على أي من الرموز المذكورة { } ، { } ، ( \* ) ، \* دلالة على أن هذه المجموعة توضيحية وليست ضمن إيعازات البرنامج لذا يهملها المترجم ولا يأخذها بنظر الإعتبار، ويمكن ظهور التعليقات في أي مكان في البرنامج ويمكن من خلالها توضيح وذكر ما نريد توضيحه وذكره دون أي قواعد أو قيود.

### أنواع البيانات Data Types

تحتاج البرامج في لغة باسكال إلى أن يكون لكل عنصر بياني نوع بياني محدد. هذا النوع البياني يحجز للعنصر البياني موقع في ذاكرة الحاسوب ويعرّف له مدى معين من القيم التي يمكن تخزينها في ذلك الموقع وكذلك يحدد مجموعة من العمليات التي يمكن إجراؤها على ذلك العنصر من النوع البياني المحدد، وبشكل عام فهناك قسمين من أنواع البيانات، هي:

#### أولاً: أنواع البيانات القياسية Standard Data Types

في لغة باسكال القياسية هناك أربعة أنواع بيانية هي:

##### 1. النوع البياني الصحيح Integer Data Type :

عند الإعلان عن متغير معين أنه من النوع الصحيح Integer ، هذا يعني أن ذلك المتغير يستطيع أن يخزن أي عدد صحيح ضمن المدى +32767 ..... -32768 . هذا المتغير سيخزن قيم صحيحة فقط مهما كانت طويلة لكن ليس بإمكانه تخزين القيم التي تحتوي على أجزاء عشرية.

##### 2. النوع البياني الحقيقي Real Data Type :

عند الإعلان عن متغير معين أنه من النوع الحقيقي Real ، هذا يعني أن ذلك المتغير يستطيع أن يخزن القيم الحقيقية المعبر عنها بالفاصلة العشرية floating point .

**ملاحظة:** تستخدم طريقة التدوين الأسّي *E - Notation* لتمثيل القيم الحقيقية وخاصة القيم الكبيرة أو الصغيرة جداً وذلك لغرض تقليص حيز التخزين الذي تشغله هذه القيم من ذاكرة الحاسوب، وفي هذه الطريقة يستخدم الرمز  $[E+BB]$  للإشارة إلى أن العدد الذي يسبقه مضروباً بالعدد 10 مرفوعاً للقوة *BB* وهي عدد صحيح، والجدول التالي يوضح بعض الأمثلة للقيم الحقيقية .

جدول يوضح بعض الأمثلة للقيم الحقيقية

| الرقم        | صورته المختصرة           | صورته بالتدوين الأسّي |
|--------------|--------------------------|-----------------------|
| -0.000001982 | $-0.1982 \times 10^{-5}$ | -0.1982E-5            |
| 43800000000  | $4.38 \times 10^{10}$    | 4.38E+10              |
| 0.000000324  | $3.24 \times 10^{-7}$    | 3.24E-7               |
| -673000000   | $-6.73 \times 10^8$      | -6.73E+8              |

### 3. النوع البياني الرمزي *Character Data Type*

عند الإعلان عن متغير معين أنه من النوع الرمزي *Character*، هذا يعني أن ذلك المتغير يستطيع أن يخزن رمز طوله *1Byte* ولتنسيب بيانات رمزية لمتغير فإن الرمز يجب أن يحصر بين الفاصلة المفردة مثلاً 'A'.

### 4. النوع البياني البولياني *Boolean Data Type*

عند الإعلان عن متغير معين أنه من النوع البولياني *Boolean*، هذا يعني أن ذلك المتغير يستطيع أن يخزن القيم المنطقية *True* أو *False* نستخدم مثل هذا المتغير في البرامج التي نحتاج فيها خزن إحدى هاتين القيمتين أو كليهما.

**ملاحظة:** هذه الأنواع الأربعة هي الأنواع الأساسية والمعرفة في باسكال القياسية، وهناك أنواع أخرى استحدثت في الإصدارات المتقدمة من لغة باسكال أهمها:

جدول يوضح الأنواع القياسية المستحدثة

| النوع           | المدى                                              | الحجم<br>(بالبايث) | الملاحظات                |
|-----------------|----------------------------------------------------|--------------------|--------------------------|
| <i>Byte</i>     | 0 .. 255                                           | 1                  | أعداد صحيحة<br>موجبة فقط |
| <i>Word</i>     | 0 .. 65535                                         | 2                  | أعداد صحيحة<br>موجبة فقط |
| <i>Shortint</i> | -128 .. 127                                        | 1                  | أعداد صحيحة              |
| <i>Longint</i>  | -2146473648 ... 2146473647                         | 4                  | أعداد صحيحة              |
| <i>Single</i>   | $1.5 \times 10^{-45} \dots 3.4 \times 10^{38}$     | 4                  | أعداد حقيقية             |
| <i>Double</i>   | $5.0 \times 10^{-324} \dots 1.7 \times 10^{308}$   | 8                  | أعداد حقيقية             |
| <i>Extended</i> | $3.4 \times 10^{-4932} \dots 1.1 \times 10^{4932}$ | 10                 | أعداد حقيقية             |
| <i>String</i>   | 255 حرف                                            | 255                | غير رقمية                |

### ثانياً: أنواع البيانات المعرفة من قبل المستخدم (غير القياسية) *User-Defined Data Types*

تتيح لغة باسكال للمبرمج تعريف أنواع خاصة ومحددة أكثر من البيانات، ويتم تعريف هذا النوع من البيانات في مقطع *Type* حيث يعرف المبرمج أنواع بيانات جديدة (غير قياسية) خاصة ببرنامجه فقط مثل : المجموعات، المصفوفات، السجلات وغيرها، وسيأتي لاحقاً شرح تفصيلي لبعضها. مثال: تعريف مجموعة أيام الأسبوع كالتالي:

Type

Week\_Days = (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday);

لذا فعند الإعلان عن متغير من نوع Week\_Days فإن هذا المتغير سيأخذ واحدة فقط من القيم السبعة المذكورة في المجموعة، ويكون الإعلان كالتالي:

Var

Day : Week\_Days;

### الإعلان عن المتغيرات Variables Declaration

تفرض لغة باسكال على المبرمج أن يعلن في بداية البرنامج عن جميع المتغيرات المستخدمة في البرنامج، حيث يتم الإعلان عن المتغيرات المستخدمة في البرنامج بأنواعها (الأنواع القياسية المذكورة سابقاً والأنواع المعرفة من قبل المبرمج التي يتم تعريفها في جزء Type) هذا الإعلان يتم من خلال مقطعي Const و Var حيث هناك نوعين من المتغيرات:

#### أولاً : الثوابت Constants

المقصود بها المتغيرات التي تمتلك قيمة ثابتة لا تتغير خلال البرنامج ولا تعتمد على الإدخال فقيمتها معروفة مسبقاً (قبل بداية البرنامج) ولا تتغير أثناء البرنامج، يتم الإعلان عن هذه المتغيرات في مقطع const من البرنامج حيث يتم الإعلان عن الثوابت مرة واحدة في حين يمكن الإشارة إليها أكثر من مرة خلال البرنامج بالأسماء المعلن عنها. الصيغة العامة للإعلان عن الثوابت هي:

Const

Constant\_name = expression;

مثلاً : الإعلان عن النسبة الثابتة بإعتبارها لا تتغير

Const

PI = 3.141592;

## الفائدة من الإعلان عن الثوابت :

1. قد تكون قيمة الثابت قيمة طويلة ومعقدة وتحتاج إلى الدقة عند ذكرها فبدلاً من تكرارها داخل البرنامج نعلن عنها في مقطع الثوابت ونستخدم الاسم البسيط المعروفة من خلاله بدلاً من استخدام القيم نفسها وتكرارها، مثلاً:

*Const*

*UNV = 'AlMustanseriya University';*

2. قد يتم من خلال الثوابت تحديد قيمة عظمى أو قيمة معينة لعملية معينة يتم الإشارة إليها أكثر من مرة خلال البرنامج، فعندما نريد تغيير هذه القيمة المعينة يكفي أن نغيرها في مقطع الثوابت مرة واحدة بدلاً من تغييرها عدة مرات في البرنامج، مثلاً:

*Const*

*Max = 100;*

## ثانياً : المتغيرات *Variables*:

المقصود بها المتغيرات التي يتم تغيير قيمها أثناء البرنامج، حيث تأخذ هذه المتغيرات أكثر من قيمة أثناء تنفيذ البرنامج شرط أن تكون هذه القيم المأخوذة ضمن النوع البياني والمدى المحدد لهذا النوع، والذي يتم الإعلان عنه (إسم المتغير، النوع البياني التابع له المتغير) في مقطع التغيرات *Var* من البرنامج، الصيغة العامة للإعلان عن المتغيرات هي:

*Var*

*Variable\_List : Type;*

## مثال:

*Var*

*x: Integer;* للإعلان عن المتغير *X* من نوع صحيح *integer* العبارة

*y, z, w: real;* للإعلان عن المتغيرات *y, z, w* من نوع حقيقي *real* العبارة

*c: char;* للإعلان عن المتغير *c* من نوع رمزي *character* العبارة

*yes, no: Boolean;* للإعلان عن المتغيرات *yes, no* من النوع البوليفاني *Boolean* العبارة

## الدوال المبنية داخلياً *Built-In Functions*:

توفر لغة باسكال للمبرمج مجموعة من الدوال التي تسهل عمل المبرمج وتنبو عنه في أداء مهمة معينة، هذه الدوال تتضمن العمليات المهمة وكثيرة الاستخدام إختصاراً للتعقيدات، فيما يلي نذكر بعضها:

| الدالة       | الوظيفة                                                                            | مثال                                                                                   |
|--------------|------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| <i>abs</i>   | إيجاد القيمة المطلقة للمعامل                                                       | <i>abs</i> (-5) = 5                                                                    |
| <i>sqr</i>   | إيجاد مربع المعامل                                                                 | <i>sqr</i> (5) = 25                                                                    |
| <i>sqrt</i>  | إيجاد الجذر التربيعي للمعامل                                                       | <i>sqr</i> (25) = 5.0000000000E+00                                                     |
| <i>sin</i>   | إيجاد جيب الزاوية (تستخدم مع الزوايا)                                              | <i>sin</i> (30*PI/180) = 5.0000000000E-01                                              |
| <i>cos</i>   | إيجاد جيب تمام الزاوية (تستخدم مع الزوايا)                                         | <i>cos</i> (60*PI/180) = 5.0000000000E-01                                              |
| <i>trunc</i> | تحويل الأعداد الحقيقية إلى أعداد صحيحة وذلك بحذف الجزء العشري                      | <i>trunc</i> (5.2431)=5, <i>trunc</i> (-5.6)= -5                                       |
| <i>round</i> | تحويل الأعداد الحقيقية إلى أعداد صحيحة وذلك بتقريب العدد الحقيقي إلى أقرب عدد صحيح | <i>round</i> (5.2431)=5, <i>round</i> (-5.6)= -6                                       |
| <i>int</i>   | حذف المراتب العشرية من العدد الحقيقي مع بقاء العدد من النوع الحقيقي                | <i>int</i> (2.91)= 2.0000000000E+00                                                    |
| <i>odd</i>   | إعطاء قيمة <i>true</i> إذا كان المعامل فردي وإلا فإنها تعطي القيمة <i>false</i>    | <i>odd</i> (5) = <i>true</i> , <i>odd</i> (4)= <i>false</i>                            |
| <i>pred</i>  | إعطاء القيمة السابقة للمعامل (لا تتعامل مع القيم من نوع <i>real</i> )              | <i>pred</i> (5)=4, <i>pred</i> ('b')='a',<br><i>pred</i> ( <i>true</i> )= <i>false</i> |
| <i>succ</i>  | إعطاء القيمة التالية للمعامل (لا تتعامل مع القيم من نوع <i>real</i> )              | <i>succ</i> (4)=5, <i>succ</i> ('A')='B',<br><i>succ</i> ( <i>false</i> )= <i>true</i> |
| <i>ord</i>   | إعطاء شفرة <i>ASCII</i> للمعامل (يكون معاملها من نوع <i>char</i> )                 | <i>ord</i> ('A')=65, <i>ord</i> ('a')=97                                               |

|                                                                                   |                                                                |            |
|-----------------------------------------------------------------------------------|----------------------------------------------------------------|------------|
| $chr(65) = 'A', \quad chr(97) = 'a'$                                              | إعطاء الرمز المناظر للمعامل (يكون معاملها عبارة عن شفرة ASCII) | <i>chr</i> |
| $e^x \rightarrow exp(x)$                                                          | التعبير عن الأساس الطبيعي $e$ بمعنى <i>exponential</i>         | <i>exp</i> |
| بفرض أن $x=10$ فإن:<br>$inc(x) \rightarrow x=11, \quad inc(x,5) \rightarrow x=15$ | زيادة قيمة المعامل بواحد أو بمقدار محدد                        | <i>inc</i> |
| بفرض أن $x=15$ فإن:<br>$dec(x) \rightarrow x=14, \quad dec(x,5) \rightarrow x=10$ | نقصان قيمة المعامل بواحد أو بمقدار محدد                        | <i>dec</i> |

ملاحظة تحتاج الدالة إلى أقواس ( ) لغرض الإحاطة بمعاملها والذي سيطبق عليه عمل الدالة وكما هو موضح في الأمثلة في الجدول أعلاه.



## عبارات لغة باسكال Pascal Language Statements

### 1. تعليمة التنسيب Assignment Statement

يستخدم الرمز (=:) للدلالة على تنسيب قيمة ثابتة لمتغير أو تعبير حسابي أو منطقي لمتغير. كما في الأمثلة:

I := 1;

I := I + 5;

Z := x <= y;

ملاحظات:

1. لا يمكن تنسيب متغير لقيمة ثابتة.
2. يجب أن يكون طرفاً تعليمة التنسيب متكافئين.

### 2. تعليمة الطباعة Printing Statement

تستخدم التعليمة (write) أو التعليمة (writeln) للطباعة وكالتالي:

Write (variable list);

Writeln (variable list);

مثال:

Write (a,b);

Writeln(c);

ملاحظات:

1. تستخدم التعليمة write للطباعة مع بقاء مؤشر الطباعة على نفس السطر، بينما التعليمة writeln تستخدم للطباعة مع تغيير موقع مؤشر الطباعة إلى السطر التالي.
2. يجب أن تحصر معاملات التعليمة (الأشياء المراد طباعتها) بين الأقواس الدائرية.
3. لطباعة الثوابت الرمزية (مثلاً الرسائل التوضيحية) تحصر بين علامة الاقتباس المنفردة، حيث تظهر هذه السلاسل كما هي على شاشة التنفيذ. كما في المثال:

Writeln ( ' Enter the Number ' );

### 3. تعليمة القراءة Reading Statement

تستخدم التعليمة (read) أو التعليمة (readln) للقراءة وكالتالي:

Read (variable list);

Readln (variable list);

مثال:

Read (a,b);

Readln (c);

ملاحظات:

1. تستخدم التعليمة read للقراءة مع بقاء المؤشر على نفس السطر، بينما التعليمة readln تستخدم للقراءة مع تغيير موقع المؤشر إلى السطر التالي.
2. يجب أن تحصر معاملات التعليمة (الأشياء المراد قراءتها) بين الأقواس الدائرية.
3. عند التنفيذ بعد كتابة القيم المراد إدخالها يجب الضغط على مفتاح enter.

### 4. تعليمة إذا المنطقية Logical IF Statement

ولها عدة صيغ:

a. If condition then statement;

ملاحظة:

إذا أعطى الشرط true فإن التعليمة ستنفذ وإلا سوف تهمل.

مثال:

If a>b then max:=a;

b. If condition then stat.1

Else stat.2;

ملاحظة:

إذا أعطى الشرط true فإن التعليمة stat.1 ستنفذ وإلا سوف تنفذ التعليمة stat.2.

مثال:

```
If a>b then max:=a
 Else max:=b;
```

```
c. If cond.1 then stat.1
 Else If cond.2 then stat.2
 Else stat.3;
```

ملاحظة:

إذا أعطى الشرط الأول true فإن التعليمة stat.1 ستنفذ وإلا سيتم السؤال عن الشرط الثاني و إذا أعطى الشرط الثاني true فإن التعليمة stat.2 ستنفذ وإلا سوف تنفذ التعليمة stat.3.

مثال:

```
If a>b then max:=a
 Else if b >c then max:=b
 Else max:=c;
```

ملاحظة :

في أي صيغة من صيغ تعليمة اذا المنطقية (if statement) إذا كانت التعليمة statement تمثل أكثر من تعليمة عندئذ ستكون ما يسمى بـ (block of statements) وستحتاج أن تكون مسبقة بـ (begin) ومتبوعة بـ (end;)

**مثال(1):** أكتب برنامج لإدخال معدل طالب وطباعة كلمة successful إذا كان ناجحا أو كلمة failure إذا كان راسبا.

### الحل:

```

Program Testing;
Uses
 Wincrt;
Var
 Av:real;
Begin
 Writeln('Enter the Average:'); readln(Av);
 If Av >= 50 then writeln ('successful')
 Else writeln ('failure');
End.

```

مثال(2): حول التعابير الرياضية التالية إلى تعابير بلغة pascal

1.  $Z = \sqrt{2x - 4 \cos 2y}$
2.  $W = -u + 5k^2$
3.  $P = 5j^3 + 2 \sin^2 v$

مثال(3): أكتب اعلان المتغيرات فيما يلي:

1.  $I := x < 5;$
2.  $Y := \text{round}(z);$
3.  $K := x + 4.5;$

أسبقيات العوامل الرياضية:

1. الرفع للقوى
2. الضرب والقسمة حسب ورودها من اليسار الى اليمين
3. الجمع والطرح حسب ورودها من اليسار الى اليمين

للتغلب على هذه الأسبقيات تستخدم الأقواس الدائرية

مثال: اذا كانت  $a=6, b=4, c=2$

$Z:=a+b/c;$

فإن  $z=8$

بينما

$z:= (a+b)/c$

فإن  $z=5$

5. تعليمة الانتقال *GOTO statement*

GOTO label

الصيغة العامة للإيعاز

تستخدم لنقل سير التنفيذ إلى العنوان Label هذا يستوجب تعريف هذا العنوان في جزء الاعلان عن العناوين (Label) الذي يلي الجزء (Uses) .

ملاحظة :

باستخدام تعليمتي IF و GOTO يمكن تكوين حلقة تكرارية LOOP

6. تعليمة الإنتقاء *Case statement*

## الصيغة الأولى :

Case expression of  
*selector1* : *statement1* ;  
*selector2* : *statement2* ;  
*selector3* : *statement3* ;  
*selector n* : *statement n* ;  
 End; {case}

## الصيغة الثانية :

Case expression of  
*selector1* : *statement1* ;  
*selector2* : *statement2* ;  
*selector3* : *statement3* ;  
*selector n* : *statement n* ;  
 Else  
     Another statement;  
 End; {case}

مثال :

Case *daynumber* of  
 1 : writeln('Sunday') ;  
 2 : writeln('Monday') ;;  
 3 : writeln('Tuesday') ;  
 4 : writeln('Wednesday') ;  
 5 : writeln('Thursday') ;  
 6 : writeln('Friday') ;  
 7 : writeln('Saturday') ;  
 End; {case}

ملاحظات

1. يمكن أن يكون الاختيار (selector) مدى من القيم :

Case *ch* of

```
'A'..'Z' , 'a'..'z' : writeln ('letter') ;
 '0'..'9' : writeln ('digit') ;
 '+', '-', '*', '/' : writeln ('operator');
```

Else

```
 Writeln ('special character');
End; {case}
```

2. إذا كانت التعليمة *statement* تمثل أكثر من تعليمة عندئذ ستكوّن ما يسمّى بـ ( *block of statements* ) وستحتاج أن تكون مسبقة بـ ( *begin* ) ومتبوعة بـ ( *end;* )

## هياكل التكرار Loop Structures

تستخدم لتكرار تعليمة واحدة أو أكثر وهي على ثلاثة أنواع:

### الهيكل الأول

#### for structure هيكل التكرار

الصيغة الأولى:

**For variable := initial to final do**  
**Statement;**

الصيغة الثانية:

**For variable := final downto initial do**  
**Statement;**

Where:

- **Variable** : متغير يعمل كعداد يكون في الاغلب عدد صحيح (integer) و أحيانا رمز (char) لكن لا يمكن أن يكون عدد صحيح (real)
- **Initial**: القيمة الابتدائية للعداد
- **Final**: القيمة النهائية للعداد بعد زيادته بمقدار واحد واحد

### ملاحظات:

1. لا يجوز التلاعب بقيمة العداد داخل الحلقة التكرارية
2. إذا كانت initial أكبر من final عندئذ سيهمل المترجم (compiler) تنفيذ الحلقة التكرارية محدثاً خطأ منطقياً.
3. إذا كانت التعليمة statement تمثل أكثر من تعليمة عندئذ ستكون ما يسمى بـ (block of statements) وستحتاج أن تكون مسبقة بـ (begin) ومتبوعة بـ (end;)

### الهيكل الثاني

#### While structure هيكل التكرار

**While condition do**  
**Statement;**

### ملاحظات

1. تتكرر التعليمات طالما يكون الشرط True وعندما يكون الشرط false يخرج خارج الحلقة التكرارية.
2. قد لا تنفذ هذه الحلقة التكرارية ولا مرة.



3. إذا كانت التعليمة statement تمثل أكثر من تعليمة عندئذ ستكوّن ما يسمّى  
 بـ ( block of statements ) وستحتاج أن تكون مسبقة بـ (begin) ومتبوعة  
 بـ (end;)

### الهيكل الثالث

#### هيكل التكرار Repeat .....Until structure

Repeat  
     Statement;  
 Until condition;

### ملاحظات

1. تتكرر التعليمات طالما يكون الشرط False وعندما يكون الشرط True يخرج خارج الحلقة التكرارية.
2. تنفذ التعليمات داخل الحلقة التكرارية مرة واحدة على الأقل.
3. إذا كانت التعليمة statement تمثل أكثر من تعليمة عندئذ ستكوّن ما يسمّى  
 بـ ( block of statements ) وستحتاج أن تكون مسبقة بـ (begin) ومتبوعة  
 بـ (end;)

## المصفوفات Matrices

المصفوفة هي هيكل بياني data structure يستخدم لخرن مجموعة بيانات من نفس النوع البياني. وهي على نوعين:

### 1. المصفوفات ذات البعد الواحد One Dimensional Array وتسمى أيضا المتجهات vectors حيث تعرف بالشكل التالي

Var

ArrName : array [ range] of element\_Type ;

Example:

Var

A: array [1..10] of integer;

B : array [1..50] of real;

Symbols : array [1..100] of char;

ملاحظة:

1. لقراءة المتجه نستخدم هيكل تكرار. مثلا المتجه X عدد عناصره N

For i:= 1 to N do

Readln ( X[i] ) ;

2. لطباعة المتجه نستخدم هيكل تكرار. مثلا المتجه X عدد عناصره N

For i:= 1 to N do

Writeln ( X[i] ) ;

-

### 2. المصفوفات ذات البعدين Two Dimensional Array تعرف بالشكل التالي

Var

ArrName : array [ row\_range , col\_range] of element\_Type ;

Example:

```
Var
 X: array [1..10 , 1..5] of integer;
```

ملاحظة:

1. لقراءة المصفوفة نستخدم هيكل تكرار. مثلا المصفوفة X(10,5)

```
For i:= 1 to 10 do
 For j:= 1 to 5 do
 Readln (X[I, j]) ;
```

2. لطباعة المصفوفة نستخدم هيكل تكرار. مثلا المصفوفة X(10,2)

```
For i:= 1 to 10 do begin
 For j := 1 to 2 do
 Write (X[I , j]) ;
 Writeln
End;
```

-

## البرامج الفرعية SubPrograms

هي برامج مساعدة للبرنامج الرئيسي مهمتها القيام بجزء من مهام البرنامج الرئيسي وذلك لغرض تسهيل مهمة البرنامج الرئيسي وكذلك تسهيل عملية اكتشاف الأخطاء وتحديد مواقعها والعمل ضمن المبدأ المذكور (فرق تسد). وهي على نوعين:

1. الإجراءات Procedures
2. الدوال Functions

### ملاحظات :

1. كل البرامج الفرعية تكتب قبل `begin {main}`
2. تسلسل البرامج الفرعية عند كتابتها غير مهم لكن المهم مراعاة التسلسل عند الاستدعاء.
3. لا تنفذ البرامج الفرعية إلا عند استدعائها.

### أولاً: الإجراءات Procedures

تعتبر برامج صغيرة قائمة بذاتها , مجرد ذكر اسمها مع معاملاتها (إن وجدت) يعتبر تعليمة من تعليمات لغة Pascal بخلاف الدوال التي لا تعتبر أسماءها مع معاملاتها تعليمة من تعليمات اللغة. لذلك يستدعى الإجراء بمجرد ذكر اسمه مع معاملاته (إن وجدت) متبوعاً ب (؛) شأنه بذلك شأن أي تعليمة من تعليمات اللغة.

## أنواع المعاملات : Types of Parameters

**المعامل parameter:** وهو المتغير الذي تجرى عليه وظيفة الإجراء. حيث يمكن أن تحتوي الإجراءات على معاملات أو قد لا تحتوي على معاملات. الإجراءات المحتوية على معاملات قد تكون معاملاتها أحد النوعين التاليين أو كليهما:

### 1. معاملات داخلية pass-in parameters

تدخل إلى الإجراء بقيمة معينة ولا تعيد أي قيمة للبرنامج الرئيسي. فهي فقط وسيلة لنقل المعامل من البرنامج الرئيسي إلى الإجراء لأداء المهمة المطلوبة منه.

### 2. معاملات خارجية pass-out parameters

وهي المتغيرات التي تعيد قيمة معينة إلى البرنامج الرئيسي قد تكون هذه القيمة محسوبة ضمن الإجراء أو قد تمرر للإجراء لإعادة حسابها مرة أخرى وإرجاعها للبرنامج الرئيسي. في هذه الحالة يجب أن يسبق اسم هذه المعاملات كلمة VAR في تعريفها ضمن الإجراء.

وفي كلا النوعين يجب أن يحدد النوع البياني للمعاملات مباشرة عند بداية الإجراء ( مع إعلان اسم الإجراء).

## أنواع المتغيرات Types of Variables

### 1. متغيرات محلية Local Variables:

وهي المتغيرات التي يتم الإعلان عنها في الجزء var داخل الإجراء. قيم هذه المتغيرات تكون متاحة لذلك الإجراء بصورة خاصة، فهي غير متاحة وغير معرفة للبرنامج الرئيسي أو البرامج الفرعية الأخرى. هذه المتغيرات يعلن عنها الإجراء لأداء المهمة المطلوبة منه.

### 2. متغيرات عالمية Global Variables:

وهي المتغيرات التي يتم الإعلان عنها في الجزء var التابع للبرنامج الرئيسي. قيم هذه المتغيرات تكون متاحة للبرنامج الرئيسي وجميع البرامج الفرعية الأخرى.

## Merging vectors and matrices

### A. Merging vectors

#### Question(1)

Write program in PASCAL language to merge two vectors A(4) and B(5) and find the new vector C

أكتب برنامج بلغة PASCAL لدمج المتجهين A(4) & B(5) وتكوين المتجه الجديد C

#### Solution

Program merging;

Uses wincrt;

Type List =array [1..9] of integer;

Var A,B,C :list;

{\*\*\*\*\*}

Procedure readvect( var x:list; k:integer);

Var I :integer;

Begin writeln( 'enter the vector' );

For i:= 1 to k do

Readln( x[i]);

End;

{\*\*\*\*\*}

Procedure writevect( x:list; k:integer);

Var I :integer;

Begin

Writeln('the vector is:');

For i:= 1 to k do

write( x[i]:5);

writeln;

End;

{\*\*\*\*\*}

Procedure merge(x,y:list; var z:list);

Var I:integer;

Begin

```

For i:= 1 to 9 do
 If i<= 4 then
 Z[i] := x[i]
 Else
 Z[i] := y[i-4];
End;

{*****}

Begin {main}
 Readvect(A,4); Readvect(B,5); Merge(A,B,C);
 Writevect(A,4); Writevect(B,5); Writevect(C,9);
End.

{*****}

```

**Note that:**

The procedure merge can be written in another way as follows:

لاحظ أن : يمكن كتابة procedure merge بطرق أخرى كما يلي (باقي البرنامج لا يتغير)

```

{*****}

Procedure merge(x,y:list; var z:list);
Var
 I:integer;
Begin
 For i:= 1 to 4 do
 Z[i] := x[i];
 For i:= 1 to 5 do
 Z[i+4] := y[i];
 End;

 {*****}

Procedure merge(x,y:list; var z:list);
Var I:integer;
Begin
 For i:= 1 to 4 do

```

```

 Z[i] := x[i];
 For i:= 5 to 9 do
 Z[i] := y[i-4];
End;
{*****}

```

**Question(2)**

Write program in PASCAL language to split the long vector C into two vectors A(4) and B(5)

أكتب برنامج بلغة PASCAL لتقسيم المتجه الطول C الى المتجهين A(4) & B(5)

**Solution**

Program splitting;

Uses wincrt;

Type List =array [1..9] of integer;

Var A,B,C :list;

```

{*****}

```

Procedure readvect( var x:list; k:integer);

Var I :integer;

Begin writeln( 'enter the vector' );

For i:= 1 to k do Readln( x[i]);

End;

```

{*****}

```

Procedure writevect( x:list; k:integer);

Var I :integer;

Begin

Writeln('the vector is:');

For i:= 1 to k do write( x[i]:5);

writeln;

End;

```

{*****}

```

Procedure split(var x,y:list; z:list);

Var I:integer;



---

Begin

For i:= 1 to 9 do

    If i<= 4 then

        x[i] := Z[i]

    Else

        y[i-4] := Z[i];

End;

{\*\*\*\*\*}

Begin {main}

    Readvect(C,9);           split(A,B,C);

    Writevect(A,4);       Writevect(B,5);       Writevect(C,9);

End.

{\*\*\*\*\*}

**Note that:**

The *procedure split* can be written in another way as follows:

لاحظ أن : يمكن كتابة *procedure split* بطرق أخرى كما يلي (باقي البرنامج لا يتغير)

{\*\*\*\*\*}

Procedure split(var x,y:list; z:list);

Var    I:integer;

Begin

    For i:= 1 to 4 do

        x[i] := Z[i];

    For i:= 1 to 5 do

        y[i] := Z[i+4];

End;

{\*\*\*\*\*}

Procedure split(var x,y:list; z:list);

Var    I:integer;

Begin

    For i:= 1 to 4 do

```

 x[i] := Z[i];
 For i:= 5 to 9 do
 y[i-4] := Z[i];
End;
{*****}

```

## B. Merging Matrices

### Question(3)

Write program in PASCAL language to merge two matrices A(4,6) and B(5,6) and find the new matrix C

أكتب برنامج بلغة PASCAL لدمج المصفوفتين A(4,6) & B(5,6) وتكوين المصفوفة الجديدة C

### Solution

Program merging;

Uses wincrt;

Type

arr=array [1..9,1..6] of integer;

Var

A,B,C :arr;

```

{*****}

```

Procedure readmat( var x:arr; m,n:integer);

Var I ,j:integer;

Begin writeln( 'enter the matrix' );

For i:= 1 to m do

For j:= 1 to n do

Readln( x[I,j]);

End;

```

{*****}

```

Procedure writemat( x:arr; m,n:integer);

Var

I ,j:integer;

Begin

---

```

 Writeln('the matrix is:');
 For i:= 1 to m do begin
 For j:= 1 to n do
 write(x[I,j]:5);

 writeln;
 end;
 writeln;

 End;

{*****}

Procedure merge(x,y:arr; var z:arr);
Var
 I , j :integer;
Begin
 For i:= 1 to 9 do
 For j:= 1 to 6 do
 If i<= 4 then
 Z[I,j] := x[I,j]
 Else
 Z[I, j] := y[i-4, j];

 End;

 {*****}

Begin {main}

 Readmat(A,4,6);
 Readmat(B,5,6);
 Merge(A,B,C);
 Writemat(A,4,6);
 Writemat(B,5,6);
 Writemat(C,9,6);

End.

{*****}

```

**Note that:**

The procedure merge can be written in another way as follows:

لاحظ أن : يمكن كتابة procedure merge بطرق أخرى كما يلي (باقي البرنامج لا يتغير)

```
{*****}
```

```
Procedure merge(x,y:arr; var z:arr);
```

```
Var
```

```
 I,j:integer;
```

```
Begin
```

```
 For i:= 1 to 4 do
```

```
 For j:= 1 to 6 do
```

```
 Z[I,j] := x[I,j];
```

```
 For i:= 1 to 5 do
```

```
 For j:= 1 to 6 do
```

```
 Z[i+4,j] := y[I,j];
```

```
End;
```

```
{*****}
```

```
Procedure merge(x,y: arr; var z:arr);
```

```
Var
```

```
 I,j:integer;
```

```
Begin
```

```
 For i:= 1 to 4 do
```

```
 For j := 1 to 6 do
```

```
 Z[i,j] := x[i,j];
```

```
 For i:= 5 to 9 do
```

```
 For j := 1 to 6 do
```

```
 Z[i,j] := y[i-4,j];
```

```
End;
```

```
{*****}
```

**Question(2)**

Write program in PASCAL language to split the big matrix C into two matrices A(4,6) and B(5,6)

أكتب برنامج بلغة PASCAL لتجزئة المصفوفة الكبيرة C الى المصفوفتين A(4,6) & B(5,6)

### Solution

Program splitting;

Uses wincrt;

Type

arr=array [1..9, 1..6] of integer;

Var

A,B,C : arr;

{\*\*\*\*\*}

Procedure readmat( var x: arr; m,n :integer);

Var I ,j :integer;

Begin writeln( 'enter the matrix' );

For i:= 1 to m do

For j:= 1 to n do

Readln( x[i ,j]);

End;

{\*\*\*\*\*}

Procedure writemat( x: arr; m, n :integer);

Var

I ,j:integer;

Begin

Writeln('the matrix is:');

For i:= 1 to m do begin

For j:= 1 to n do

write( x[i ,j]:5);

writeln;

end;

writeln

End;

{\*\*\*\*\*}

---

```

Procedure split(var x,y: arr; z: arr);
Var
 I ,j:integer;
Begin
 For i:= 1 to 9 do
 For j:= 1 to 6 do
 If i<= 4 then
 x[i ,j] := Z[i ,j]
 Else
 y[i-4 ,j] := Z[i ,j];
 End;
 End;
 {*****}
Begin {main}
 Readmat(C,9,6); split(A,B,C);
 Writemat(A,4,6); Writemat(B,5,6); Writemat(C,9,6);
End.
 {*****}

```

**Note that:**

The *procedure split* can be written in another way as follows:

لاحظ أن : يمكن كتابة *procedure split* بطرق أخرى كما يلي (باقي البرنامج لا يتغير)

```

 {*****}
Procedure split(var x,y: arr; z: arr);
Var
 I ,j:integer;
Begin
 For i:= 1 to 4 do
 For j :=1 to 6 do
 x[i ,j] := Z[i ,j];
 End;
 For i:= 1 to 5 do
 For j :=1 to 6 do

```

---

```

 y[i ,j] := Z[i+4 ,j];

End;

{*****}

Procedure split(var x,y:arr; z: arr);

Var

 I ,j:integer;

Begin

 For i:= 1 to 4 do

 For j :=1 to 6 do

 x[i ,j] := Z[i ,j];

 For i:= 5 to 9 do

 For j :=1 to 6 do

 y[i-4 ,j] := Z[i ,j];

 End;

 {*****}

```

### **Homework**

1. write PASCAL program to merge the two matrices L(3,7) & K(3,2) to construct the new matrix S
2. write PASCAL program to merge the four matrices A(2,2), B(2,2), C(2,2) & D(2,2), to construct the new matrix E(4,4).

### **Note That :**

يمكن دمج المصفوفات بدمج أعمدتها مع بقاء عدد الصفوف ثابت في هذه الحالة يثبت فهرس الصفوف (الفهرس I) ويكون التلاعب والتغيير لفهرس الأعمدة فقط (الفهرس J) .

السجلات records

السجل هو هيكل بياني data structure يتضمن تعريف متغيرات (لخزن قيم) ذات أنواع بيانية مختلفة وهذا ما يميز هذا الهيكل البياني عن الهيكل البياني السابق (المصفوفة).

لتعريف النوع record:

Type

```

Rec_Name = record
 Field1 : data_type;
 Field2 : data_type;
 Field3 : data_type;

 Fieldn : data_type;
End;{record}

```

للإعلان عن النوع record

Var

```

Id: Rec_Name;

```

مثال:

Type

```

Student = record
 Id :integer;
 Name: string;
 Age : integer;
 Av : real;
End;

```

Var

```

S: student;

```

للتعامل مع السجل داخل البرنامج

Begin

```

s.id := 10; {or} { readln(s.id)}
s.name:='ali';
s.age := 20;
readln(s.av);
writeln(s.name, s.age, s.av);

```

end.

مثال: أكتب تعريف سجل معلومات عن موظفين يتضمن اسم الموظف وراتبه وعنوانه. ثم بين عملية الإدخال.



Type

```
Emp = record
 Name, address : string;
 Salary : real;
End ;
```

Var

```
Employee : emp;
```

Begin

```
Writeln('enter name'); Readln (employee.name);
Writeln('enter salary'); Readln (employee.salary);
Writeln('enter address'); Readln (employee.address);
```

End.

نلاحظ أن في كل تعليمة تخصص حقل من حقول السجل يجب إعادة اسم متغير السجل (employee) يمكن اختصار ذكر اسم متغير السجل مع الحقل باستخدام تعليمة **do with** كالتالي:

With employee do

Begin

```
 Readln(name);
 Readln(address);
 Salary := 200;
```

End;

---

---

### ملاحظات هامة

---

---

أولاً:

قد يكون أحد حقول السجل عبارة عن مصفوفة .  
مثلاً: قد يكون الموظف يتقاضى ثلاث رواتب من جهات مختلفة ، سيكون التعريف عندئذ كالتالي:

Type

```
Emp = record
 Name, address : string;
 Salary : array [1..3] of real;
End ;
```

Var

```
Employee : emp;
```

```

Procedure reademp (var em : emp);
 Var
 k: integer;
 Begin
 Writeln(' enter the employee info');
 With em do begin
 Readln(name); readln(address);
 For k:= 1 to 3 do
 Readln(salary[k]);
 End;
 End;

```

---

ثانياً:

يمكن تعريف قائمة عناصرها عبارة عن سجلات.  
 يمكن تعريف قائمة بمعلومات عدة موظفين (مثلاً 10) وكالتالي

```

Type
 Emp = record
 Name, address : string;
 Salary : array [1..3] of real;
 End ;
 List = array [1.. 10] of emp;
Var
 Employees : list;

```

```

Procedure writeemps (var emps : list);
 Var
 I, k: integer;
 Begin
 Writeln(' the employees info are :');
 For I := 1 to 10 do
 With emps [i] do begin
 writeln('name: ', name); writeln('address: ', address);
 For k:= 1 to 3 do
 writeln(salary[k]);
 End;
 End;
 End;

```

---

ملاحظة: يمكن اعادة الاجراء السابق لتكون الطباعة مرتبة بشكل قائمة من المعلومات:

```

Procedure writeemps (var emps : list);
 Var
 I, k: integer;
 Begin
 Writeln(' the employees info are :');
 Writeln('name':10, 'add.':7, 'sal.1' :5, 'sal.2' :5, 'sal.3' :5);
 Writeln('-----');
 For I := 1 to 10 do begin
 With emps [i] do begin
 write(name:10); write(address :7);
 For k:= 1 to 3 do
 write(salary[k] : 5);
 End;
 Writeln;
 End;
 End;
 End;

```

مثال:

أكتب برنامج لإدخال معلومات موظفين وطباعتها بشكل قائمة معلومات

```

program information;
uses wincrt;
const max = 10;
Type
 Emp = record
 Name, address : string;
 Salary : array [1..3] of real;
 End ;
 List = array [1.. 10] of emp;
Var
 Employees : list;
 {#####}

```

```

Procedure reademps (var emps : list);
 Var
 I, k: integer;
 Begin
 Writeln(' enter the employees info :');
 For I := 1 to max do
 With emps [i] do begin
 readln(name); readln(address);
 For k:= 1 to 3 do
 readln(salary[k]);
 End;
 End;
 End;
 {#####}

Procedure writeemps (emps : list);
 Var
 I, k: integer;
 Begin
 Writeln(' the employees info are :');
 Writeln('name':15,'address.':15,'salalry.1':15,'salary.2':15,
 'salary.3':15);
 Writeln('-----');
 For I := 1 to max do begin
 With emps [i] do begin
 write(name:15); write(address :15);
 For k:= 1 to 3 do
 write(salary[k] : 15);
 End;
 Writeln;
 End;
 End;
 {#####}

 Begin {main}
 reademps(employees);
 writeemps(employees);
 end.

```