بسم الله الرحمن الرحيم

ملخص البرمجة ١

إعداد

فريق عمل الـ ite

ite.sy.net@gmail.com

الخوارزمية :
هي مجموعة منتهية من الخطوات البسيطة التي يتم تنفيذها بترتيب محدد تصف بدقة و وضوح الطريقة العامة لحل مسألة ما بشكل عام دون التقيد بلغة برمجة محددة .

ربما نجد للمسألة الواحدة عدد من الخوارزميات لكن الخوارزمية الأفضل هي الأقل استهلاكاً لموارد الحاسب أي التي تمتاز بتعقيد زمني منخفض (أي نسبة زمن التنفيذ إلى طول الدخل هي نسبة صغيرة) و بأقل قدر ممكن من استهلاك الذاكرة.

سندخل مباشرة بلغة الباسكال **القباسة** .

قبل البدء بتعليمات اللغة ... سنذكر بعض الملاحظات العامة :

- يمكن أن نضيف لبرامجنا بعض الملاحظات أو التعليقات التي تساعدنا في الفهم و التعديل لإصلاح برامجنا عند الحاجة، لا يرى المترجم (compiler) هذه التعليقات(المترجم compiler هو برنامج يكتشف الأخطاء القواعدية و الدلالية في برنامجنا _ الأخطاء الدلالية كأن نستخدم متحول غير معرف_ و ذلك عند تحويل تعليمات البرنامج إلى لغة أبسط و هي لغةاق‌لة).
تملك لغة الباسكال طريقتين في تعيين التعليقات حيث يمكننا وضع التعليق الذي نريد ضمن القوسين (* و *) أو ضمن القوسين { و } مع ملاحظة أن الـ (turbo pascal) سيعرض التعليق بلون رمادي ضمن الكود (كَلَونٍ افتراضي) .
يتوجب عليك استخدام نفس العلامتين لنفس التعليق، فلا يمكن مثلا أن نكتب { Comment *) سيؤدي ذلك إلى مشكلة .

- لا تميز لغة الباسكال بين الحروف الصغيرة و الكبيرة على عكس لغات برمجة أخرى، أي أن المتحولين Age و age هما أسمين لمتحول واحد في الباسكال .

- تهمل لغة الباسكال الفراغات الزائدة بين الكلمات، بشرط أن يمكن وضع فراغ واحد على الأقل في ذلك المكان، فمثلاً لو كان لدينا متحول باسم MyAge لا يمكننا كتابته بالشكل My Age و لكن لو كنا نريد كتابة :read(x,y) فيمكن كتابتها بالشكل .. ; (y x,)read ، و لا يوجد أي فرق بين الكتابتين .

الجسم الرئيسي للبرنامج :

```
Program ..... اسم البرنامج ... ;
Var
  قسم التصريح عن المتحولات و أنواعها
Begin
  .........
    هنا تكتب تعليمات البرنامج
  .........
End.
```

هـذه الأجـزاء الرئيـسية ضـرورية لأي برنـامج فـي الباسكال .

هناك أقسام أخرى يمكن اسـتخدامها أيـضاً سـنأتي عليها فيما بعد .

التصريح عن المتحولات و أنماطها :

يتم التصريح عن المتحولات في القسم var بذكر اسم المتحول يليه نمطه مع الانتباه إلى قواعد اللغة Syntax (نمط المتحول يعبر عن مجموعة القيم التي يقبلها) .

مثال :

```
X:integer;
X,y:integer;
```

أشهر الأنماط البسيطة المستخدمة في الباسكال :

الأنماط البسيطة هي بعض الأنماط المعرفة مسبقاً ضمن الباسكال و لا حاجة لتعريفها من قبلنا في القسم Type الذي سنتعرف عليه بعد قليل .

- نمط الأعداد الصحيحة integer و تشمل الأعداد الصحيحة التي تتراوح ضمن المجال [-32768,32767] .
- نمط الأعداد الحقيقية real و تشمل الأعداد الحقيقية (بدقة ١٠ أرقام بعد الفاصلة العشرية) و التي تتراوح ضمن المجال $[2.9x10^{-39},1.7x10^{38}]$ و نظرائها من الأعداد السالبة .
- المحارف char .
- سلسلة المحارف string .
- النمط المنطقي Boolean .

ضوابط تسمية المتحولات:

- لا يجوز أن يبدأ اسم المتحول برقم .
- لا يجوز أن يحوي على رموز غير الأحرف والأرقام مثل (@ $ ^ ! ،. إلخ) .
- لا يجوز أن يكون كلمة محجوزة .
- لا يسمح باستخدام الفراغ في اسم المتحول و إذا كان لدينا اسم متحول مركب يمكن الاستعانة بإحدى الطريقتين:
  ١. إما الاستعاضة عن الفراغ بـ (_) مثل : my_age .
  ٢. أو جعل أول حرف من كلمة فيه كبير و باقي الأحرف صغيرة مثل MyAge.
  ٣. أو العديد من الطرق الأخرى

كما يفضل أن تكون أسماء المتحولات ذات دلالة.

التصريح عن الثوابت :

الثابت هو متحول لا يقبل تغيير قيمته .

يتم التصريح عن الثوابت في الباسكال في القسم const الذي يقع في بداية البرنامج بعد اسم البرنامج مباشرة قبل الـ type " إن وجد" وذلك بذكر اسم الثابت يليه إشارة المساواة ثم قيمة هذا الثابت .

مثال :

```
Const
   Pi =3.14 ;
```

و من الأفضل استخدام الثوابت في البرنامج كثوابت رمزية (مثل pi) لا ثوابت حرفية (مثل ٣,١٤) و ذلك لسهولة التعديل عند اللزوم ، فمثلاً لو أردنا(زيادة دقة) هذا الثابت مثلاً لاحتجنا لتغييره في جميع الأماكن التي استخدمناه فيها .

أما لو استخدمناه كثابت رمزي فإننا نكتفي بتغيير قيمته مرة واحدة فقط في قسم التصريح عن الثوابت .
كما أن استخدام الثابت الرمزي أكثر إيحاءاً و وضوحاً عن قراءة الكود .

الكلمات المحجوزة :

كلمات لها معنى قياسي سبق تعريفه في الباسكال .. و هي تظهر بلون أبيض في turbo pascal ، يمكن مراجعة الملحق الوارد مع مقرر البرمجة لمعرفة هذه الكلمات المحجوزة .
مثال : If , while , Begin , end.

العمليات الأساسية على الأنماط المعرفة مسبقاً :

- العمليات الحسابية :
  - الجمع +
  - الطرح -
  - الضرب *
  - القسمة العادية (الحقيقية) /
  - القسمة الصحيحة div
  - باقي القسمة mod
- العمليات المنطقية :
  - أكبر >
  - أصغر <
  - أكبر أو يساوي =>
  - أصغر أو يساوي =<
  - يساوي =
  - لا يساوي <>

و هذه العمليات يكون الناتج فيها عدد جوابها قيمة منطقية false , true. صحيح أو حقيقي .

تعليمات التحكم الأساسية:

- تعليمة القراءة .
- تعليمة الكتابة .
- تعليمة الإسناد .
- التعليمة الشرطية .
- التعليمات التكرارية .

تعليمة القراءة :

لكي يقوم المستخدم بإدخال المعطيات إلى البرنامج (من خلال الدخل النظامي في الباسكال الـ Keyboard) يجب قراءتها باستخدام إحدى تعليمتي القراءة read , readln
مثال :

```
Read(x);
Read(x,y);
Readln(x);
Readln(x,y);
```

ما الفرق بين تعليمتي read , readln ؟

- Readln :

تقرأ قيم المتحول (أو المتحولات) و تنقل المؤشر إلى السطر التالي مهملةً باقي السطر سواء كتبنا عليه شيء أو لم نكتب .

- Read :

بعد قراءة قيم المتحولات تبقي المؤشر على نفس السطر .

مثال :

إذا كان a,b أعداد صحيحة و كتبنا ما يلي في جسم البرنامج :

```
Readln(a) ;
Readln (b) ;
```

و أثناء التنفيذ كتبنا على الشاشة :

```
1 5 8 10
7 6
```

(لاحظ ضرورة إدخال فراغ space بين كل عددين)

فإن البرنامج يقرأ 1 و يعطي القيمة لـ a و يهمل باقي السطر ثم يقرأ 7 و يعطي القيمة لـ b و يهمل باقي السطر .

تعليمة الكتابة :

لكي يستطيع المستخدم رؤية النتائج يجب أن تظهر هذه النتائج على الخرج النظامي في باسكال و هو الشاشة بواسطة تعليمتي writeln , write .

مثال :

```
Write(x);
Write(x,y);
Writeln(x);
Writeln(x,y);
```

ما الفرق بين تعليمتي الكتابة writeln , write ؟

- Writeln :

تكتب المتحولات على الشاشة و تنقل المؤشر إلى السطر التالي .

- Write :

تكتب المتحولات على الشاشة و تبقي المؤشر على نفس السطر .

مثال :

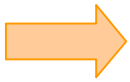
إذا كانت a,b أعداد صحيحة و كان a=1 , b=2 و كتبنا التعليمات التالية في جسم البرنامج :

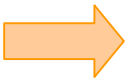
```
Write(a) ;
Write(b);
```

الخرج 12

```
Writeln(a) ;
Writeln(b);
```

الخرج 1
2

Writeln(' Ite is a good website')  Ite is a good website

Writeln(a,' Ite is a good website')  1 Ite is a good website

ملاحظة:
أحياناً نكتب تعليمة writeln; بدون أن نمرر لها أي متحولات لتكتبها ، تكون مهمتها ترك سطر فارغ فقط .
و أحياناً نكتب تعليمة readln; بدون أن نمرر لها أي متحولات لتقرأها، تكون مهمتها إيقاف تنفيذ البرنامج عند مرحلة معينة إلى أن نكبس زر enter .
و أهم استخدام لها في نهاية البرنامج لأن البرنامج عندما ينتهي من التنفيذ يعود مباشرة إلى شاشة الكود الزرقاء دون أن يترك لنا زمن لقراءة النتيجة، و هنا ستحل readln هذه المشكلة .

مثال :

```
Program Inserting_YourName;
Var S : string;
BEGIN
   Write(' What Is Your Name ?   ');
   Readln(S);
   Writeln(' Hallo  ',S,'   You Are Welcome .');
   Readln;
END.
```

تعليمة الإسناد :
تستخدم هذه التعليمة لنسخ قيمة مباشرة أو قيمة صيغة على يمين الإشارة إلى متحول ما على يسار الإشارة (:=)
حيث أنه عند إجراء الإسناد بين متحولين يتم نسخ قيمة المتحول المسند إلى المتحول المسند إليه.

مثال :

```
C:=1;
C:= C+!;
```

ملاحظة :
يجب التمييز بين معنى الإسناد و معنى المساواة رغم تشابه الرمز ، فالمساواة (=) هي إشارة منطقية ترد قيمة true,false مثل :

```
If (a+b=c) then
      .....
```

أما الإسناد (:=) فهو عملية تعني نسخ القيمة التي على يمين إشارة الإسناد و إعطائها لمتحول على يسار هذه الإشارة .

مثال :

A:=1
B:=2
C:=3
} جميعها عمليات إسناد

الآن لو كتبنا :

```
If (a+b=c) then
```

لاحظ أننا هنا لا نقوم بإعطاء قيمة لـ c و إنما نتحقق من صحة تحقق المطابقة السابقة فالإشارة هنا مساواة .

مثال آخر :

```
B:Boolean;
B := (a = b) ;
```

حيث أننا نقوم بإسناد قيمة العبارة المنطقية (a = b) إلى المتحول المنطقي B (فيأخذ قيمة true أو false)

ملاحظة هامة :

تجدر الإشارة هنا إلى أنه يجب أن تتم عملية الإسناد بين المتحولات المتماثلة في نمطها أو بين متحولات من أنماط يمكن تحويلها بين بعضها بسهولة و بدون فقدان بيانات (كأن نسند متحول من نمط integer إلى متحول من نمط real)

.

مثال :

```
Var x,y:real;
Z:integer;
```

X:=y صحيحة
X:=z صحيحة ،،،. على اعتبار أن نمط الـ Integer محتوى في الـ real
Z:=y خطـــأ

التعليمة الشرطية :

يجري فيها تنفيذ تعليمة أو كتلة التعليمات التي تليها عند تحقق شرط ما أو تجاوزها إذا لم يتحقق الشرط و لها شكلين :

الأول :

إذا (تحقق شرط) نفذ
كتلة تعليمات

```
If ( condition ) then
  Statements block
```

الثاني :

إذا (تحقق شرط) نفذ
كتلة تعليمات ١
و إلا
كتلة تعليمات ٢

```
If ( condition ) then
    Statements block 1
Else
    Statements block 2
```

مع الانتباه إلى أن else لا يتم وضع فاصلة منقوطة في آخر تعليمة تسبقها و علماً أن الشرط هو صيغة تقيم بصح أو خطأ (true,false) ، نعم أو لا ، كالصيغ التي تتضمن التراجح (أكبر ، أصغر) أو المساواة (أو عدم المساواة) بالإضافة إلى أدوات التركيب المنطقية لجمع الشروط (and,or,not) .
من الممكن استخدام تعليمة if من أجل عدد أكثر من الفروع إذا استخدمنا أكثر من تعليمة متداخلة مثل :

```
If (condition 1) then
    ..........................
Else if (condition 2 ) then
    ..........................
Else if (condition 3 ) then
    ..........................
Else
    ..........................
```

عند وجود أكثر من تعليمة if متتابعة فإن كل else تتبع للـ if التي قبلها .

```
If (condition1) do
    ........
 If (condtion2) then
    ........
  Else
    ........
```

خطأ

صح

تعليمة case :

إذا كان C متحول من نمط معدود و كان لدينا مجموعة تعليمات كما يلي ...

```
If (c=const1 ) then
  Statements block 1
If (c=const2 ) then
  Statements block 2
If (c=const3 ) then
  Statements block 3
.........................
```

إذا (c = ثابت ١) نفذ
كتلة تعليمات ١
إذا (c = ثابت ٢) نفذ
كتلة تعليمات ٢
إذا (c = ثابت ٣) نفذ
كتلة تعليمات ٣
.....

عندها يمكن أن نستبدل تعليمات if كلها بتعليمة case واحدة كما يلي :

```
Case c of
const1 :  Statements
block 1
const2 :  Statements
block 2
const3 :  Statements
block 3
.....................
End;
```

في حالة c تأخذ
ثابت ١ : كتلة تعليمات ١
ثابت ٢ : كتلة تعليمات ٢
ثابت ٣ : كتلة تعليمات ٣
.....
.....

تحتاج تعليمة case إلى end; في نهايتها .
و من الممكن استخدام else معها أيضاً في حال لم يأخذ المتحول أياً من القيم المذكورة سابقاً تنفذ كتلة التعليمات التي بعد else .
و أكثر من ذلك، إذا كان لدينا كتلة تعليمات ضمن الـ case نريد تنفيذها عندما يأخذ المتحول عدة قيم (const1 , const2 ,const3) نكتب :

```
Const1,const2,const3 : statements block;
```

أما إذا كان لدينا كتلة تعليمات ضمن الـ case نريد تنفيذها عندما يأخذ المتحول قيم متتالية متتابعة محصورة بين const1 و const2 نكتب :

```
Const1 .. const2 : statements block ;
```

تستخدم عبارة CASE فقط مع الأنماط المعدودة .

التعليمات التكرارية :

لماذا نستخدم التعليمات التكرارية (أو الحلقات التكرارية) ؟
تخيل أنك تريد طباعة عبارة ما على الشاشة مائة مرة، فهذا يكافئ أنه يتوجب عليك تكرار كتابة تعليمة writeln مائة مرة !!!!!!
و ماذا لو كنت تريد تكرار تعليمة ما عدداً غير ثابتاً من المرات أيضاً ...؟؟
لتجنب الكثير من المواقف المماثلة لهذه الحالة و منعا لإهدار الوقت وجدت الحلقات التكرارية فبدلاً من المائة سطر السابق يتم المطلوب بسطرين .

- حلقة while :

تستعمل لتكرار مجموعة من التعليمات ما دام الشرط محققاً .

```
While ( condition ) do
Statements block
```

طالما (تحقق شرط) كرر كتلة تعليمات

ملاحظات :

- علينا ملاحظة أن مجموعة التعليمات هذه يجب أن تتضمن بالضرورة عمليات تساهم في تغيير متحولات الشرط و تقربنا من حالة عدم تحقق شرط الحلقة و إلا سندخل في حلقة لا نهائية لأن الشرط سيبقى محقق دوماً.
- قد لا يكون معلوم لدينا بشكل ظاهر عدد مرات التكرار.
- يجب الانتباه إلى وضع قيم ابتدائية للعداد بشكل صحيح إن وجد .
- (لا تنفذ حلقة while أبداً عند عدم تحقق شرط الحلقة).

مثال :

```
Program test ;
Const  Max = 100;
Var Count:integer;
Begin
 Count:=1;
 While ( count<=Max) Do
   Begin
```

إسناد قيمة ابتدائية للعداد لا تفترض أن الحاسب سيقوم بإعطاء قيمة ابتدائية لأي متحول **تذكر ذلك دوما**

الشرط الذي يتم اختباره

```
        Writeln('   Hello !!');
        Count:=Count+1;
    End;
End.
```

هنا يتم التغير لمحاولة كسر شرط الدخول إلى الحلقة ...و بدونه ستصبح الحلقة مغلقة وتتكرر إلى مالانهاية

v حلقة for :

عندما يكون عدد مرات التكرار معلوم لدينا (أي أن العداد من نمط محدود تتزايد قيمته (أو تتناقص)خطوة خطوة بين قيمة بدائية و قيمة نهائية) نستطيع استخدام تعليمة for اختصاراً .

و لهذه الحلقة شكلان :

♣ الشكل الأول :

إذا أردنا أن نبدأ بالعداد بالقيمة الدنيا و ننتهي بالعليا نستخدم to :

من أجل العداد = القيمة الدنيا حتى القيمة العليا كرر كتلة التعليمات

```
For counter:=exp1 to exp2  do
   Statements block
```

مثال :

```
Program test ;
Const  Max = 100;
Var
  I:integer;
Begin
  For i:=1 to Max do
    Writeln('   Hello !!');
End.
```

♣ الشكل الثاني :

إذا أردنا الابتداء بالقيمة العليا و الانتهاء بالدنيا نستخدم downto بدل to و نبدل بين القيمة العليا و الدنيا في التعليمة .

من أجل العداد = القيمة العليا حتى القيمة الدنيا كرر كتلة التعليمات

```
For counter:=exp2 downto exp1  do
   Statements block
```

مثال :

```
Program test ;
Const  Max = 100;
Var
  I:integer;
Begin
  For i:=Max down to 1 do
    Writeln('   Hello !!');

End.
```

ملاحظة:

إذا بدلنا بين القيمة العليا و الدنيا لا يتم تنفيذ تعليمات هذه الحلقة .
و إذا تساوت القيمتين العليا و الدنيا يتم تنفيذ الحلقة مرة واحدة فقط .

- حلقة Repeat .. until :

تستخدم لتكرار مجموعة من التعليمات طالما لم يتحقق شرط .

```
Repeat
  Statements block
Until ( condition )
```

كرر
مجموعة تعليمات
حتى (شرط)

مثال:

```
Program test ;
Const  Max = 100;
Var Count:integer;
Begin
   Count:=1;
   Repeat
     Writeln('   Hello !!');
     Count:=Count+1;
   Until ( count >= Max);
End.
```

تستخدم هذه التعليمة لنفس استخدام تعليمة while مع وجود فارقين هما :

١. في تعليمة repeat .. until تنفذ التعليمات لأول مرة بغض النظر عن الشرط (سواء كان الشرط محققاً أو لا) ، أما تعليمة while فلا تنفذ التعليمات و لو لمرة واحدة إن لم يتحقق الشرط.

٢. نستخدم في تعليمة while شرط للاستمرار أي طالما تحقق الشرط نفذ كتلة التعليمات ، أما في الـ repeat .. until فنستخدم شرط للتوقف أي كرر حتى يتحقق الشرط و عندما يتحقق توقف عن التكرار .

ملاحظة ١ :

في جميع مجموعات التعليمات السابقة (if , case , while , for) عدا تلك المستخدمة في الـ repeat .. until إذا كانت تلك المجموعة مؤلفة من أكثر من تعليمة فإننا بحاجة لحصرها بكلمتي begin .. end; (وهذا ما يدعى بـ scope) أما إن كانت مؤلفة من تعليمة واحدة فقط فلا داعي لذلك .
مع ملاحظة أننا في تعليمة case نعتبر التعليمات الموجودة بعد كل ثابت كتلة تعليمات منفصلة عن الموجودة بعد بقية الثوابت .
أما كتلة التعليمات الخاصة بـ repeat .. until فهي محصورة بكلمات التعليمة نفسها repeat في البداية و until (condition) في النهاية و لا داعي لاستخدام begin .. end ;

ملاحظة ٢ :
عندما نبدأ بحلقة جديدة يستحسن أن نترك مسافة قبل بداية السطر في كل تعليماتها لسهولة القراءة و الفهم كما يلي مثلاً :

```
For i:=1 to n do
    Begin
      Readln(....);
     If ( .....) then
        Write(.....);
    End;
```

العمليات المنطقية المركبة :

هناك عمليات منطقية تقوم بربط أكثر من عبارة منطقية معاً و هي not , or , and .

- **v** Not :
  - Not T = false
  - Not F = true
- **v** Or :
  - T or T = true
  - T or F = true
  - F or T = true
  - F or F = false
- **v** And :
  - T and T = true
  - T and F =false
  - F and T =false
  - F and F =false

♣ قانونا دومرغان :

- Not (T and f) = (not T) or (not F) = F or T = T
- Not (T or F) = (not T) and (not F) = F and T = F

النمط المنطقي Boolean :

هو النمط الذي يأخذ أحد قيمتين true , false .
نستطيع إسناد أي قيمة منطقية إلى المتحول المنطقي مثل :

```
B:=(N>0);
B:=(N>0) and (M<0);
B:=b1 or (N >0) ;  //where b1:Boolean
```

النمط المحرفي char :

هو النمط الذي يعرف مجموعة المحارف التي يمكن إدخالها من لوحة المفاتيح و يقابل كل محرف منها رقم في جدول الآسكي (ASCII) .
(ASCII = American Standard Code for Information Interchange)

- هناك تابعان مهمان يتعاملان مع هذا النمط هما :
  - Ord: دخله متحول من نمط char و خرجه ترتيب ذلك المحرف في جدول الأسكي .

- Chr : دخله رقم integer محصور بين (0 à 127) و خرجه المحرف المقابل لهذا الرقم في جدول الأسكي .

- النمط المحرفي نمط مرتب إذاً يمكن إجراء عمليات المقارنة عليه <،>،،،
- كما أنه نمط متقطع لذلك يمكن أن نستخدم معه التوابع :
  - Succ : (لإعطاء العنصر التالي في جدول الأسكي) مثل succ('b')='c'
  - Pred : (لإعطاء العنصر السابق في جدول الأسكي) مثل pred('b')='a'

- إذا أردنا إسناد قيمة محرف ما إلى متغير من نمط char علينا إحاطته بإشارتي ' ' كما يلي :

Ch:='A'; عبارة صحيحة
Ch:=A; عبارة خاطئة

حيث أننا نعامل A على أنها المحرف A و ليست متغير من نمط char .
علماً أن هذه العبارة صحيحة إذا كان A متحولاً من النمط char عندها يأخذ المتحول ch قيمة المتحول A .

تعريف أنماط جديدة:

أحياناً نحتاج للتعامل مع متحولات من أنماط أخرى غير تلك المعرفة في الباسكال، إما بهدف استخدام بنية معطيات معقدة (مثل المصفوفة التي سنتحدث عنها بعد قليل أو التسجيلة record التي ستمر معنا في مقرر البرمجة ٢) ، أو بهدف تعريف أنماط بسيطة متقطعة غير معرفة مسبقاً ضمن الباسكال سنتحدث عنها بعد قليل.
يتم ذلك في القسم Type السابق لقسم الـ var ، نستخدم هنا إشارة المساواة و كمثال سابق لأوانه :

```
Type
A = array [1..10] of integer;
```

ميز :
بقسم الـ var نستخدم :

```
Var
Toto : A;
```

ما الفرق بين تعريف نمط و تعريف متحول ..؟؟؟
لتمييز الفرق سنحاول إعطاء مثال يحاكي الواقع :
لو أردنا تعريف شخص يدعى (رامي) في برنامجنا ، فإننا نقوم بتعريف نمط أسمه (إنسان) في قسم الـ Type ، ثم نعرف (رامي) من نمط (إنسان) في القسم var عندها يكون (رامي) هو واحد من مجموعة البشر الذين عرفنا نمطهم في القسم type.
إننا لا نستطيع التعامل مع الإنسان لأن الإنسان نمط تخيلي مجرد فلا يوجد شخص يدعى إنسان إنما يوجد شخص من نمط إنسان يدعى رامي مثلاً .
و هكذا نحن لا نستطيع التعامل مع النمط في البرنامج و إنما نتعامل مع متحول معرف من هذا النمط .

الأنماط المعدودة Enumerated :

هي الأنماط التي تنفصل فيها كل قيمة عن القيمة التي تليها مثال integer, char , Boolean إضافة إلى الأنماط التي نقوم نحن بتعريفها .
أي إذا أخذنا أي قيمة منها نستطيع معرفة القيمة التي تسبقها و التي تليها ، فمثلاً لو كان لدينا العدد الصحيح ٥ أعلم أن السابق ٤ و التالي ٦ .
أما لو كان لدينا ٥ كعدد حقيقي فإنني لا أستطيع معرفة العدد السابق و لا التالي ، هل السابق هو ٤,٩ أم ٤,٩٩ أم ٤,٩٩٩٩٩٩،٤,؟؟!!! إلى ما لا نهاية .

يمكننا تعريف أنماط المعدودة غير معرفة مسبقاً في القسم Type الموجود _ عادةً _ قبل قسم التصريح عن المتحولات var و ذلك عندما نستخدم متحولات تتصف بصفة مشتركة و تأخذ قيم محددة و معلومة لتسهيل العمل ، كأن نعرف نمط أشهر أو أيام أسبوع .
مثال :

```
Type
 Days=(sat , sun , mon , tues , wed , thurs, Fri);
```

علماً أننا يمكن أن نصل إلى ترتيب هذا المتحول عن طريق تابع يسمى ord و يبدأ الترتيب بالصفر ، ففي مثالنا هذا :

```
Ord(sat)=0;
Ord(sun)=1;
.
.
```

كما يمكن أن نستخدم معها تابعي :

- Succ : لإعطاء العنصر التالي مثل succ(mon)=tues
- pred : لإعطاء العنصر السابق مثل pred(mon)=sun

لكن المزعج هنا أنه لا يمكن قراءة و لا كتابة أي متحول من هذا النمط مباشرة .

التوابع المسبقة التعريف في الباسكال :

في البداية ... ماذا نعني بتابع مسبق التعريف (مبيت) ؟؟
مفهوم التابع البرمجي هو نفسه مفهوم التابع الرياضي حيث نعطيه قيمة فيرد لنا قيمة أخرى بعد إجراء بعض العمليات عليه، أما التوابع المسبقة التعريف فهي توابع نستخدمها في برنامجنا دون أن نعرفها بأنفسنا ، و ذلك لأنها جاءت معرفة مع لغة البرمجة .

أشهرها :

- التوابع المعرفة على الأعداد الحقيقية :

دخلها real و خرجها real .. بفرض r: real

تابع التربيع	Sqr (r)
تابع الجذر التربيعي	Sqrt (r)
تابع القيم المطلقة	abs (r)

تابع الجيب	sin (r)
تابع التجيب	cos (r)
تابع الظل العكسي	arctan (r)
التابع الأسي e^r	exp (r)
التابع اللوغاريتمي الطبيعي $\log_e(r)$	ln (r)

ملاحظة :

يمكن حساب a^r عن طريق التابع exp( ) كما يلي :

$a^r = e^{(\ln(a)*r)} = \exp(\ln(a)*r)$

بشرط a>0

- التوابع المعرفة على الأعداد الصحيحة :

♣ تابع القطع (trunc) :

Trunc : real **à** integer

دخله عدد حقيقي و خرجه القسم الصحيح من هذا العدد (من غير تقريب) .

مثال :

Trunc(-35.5)=-35
Trunc (10.2)=10

♣ تابع التقريب round :

round : real **à** integer

تابع يقوم بتقريب العدد الحقيقي لأقرب عدد صحيح .

مثال :

Round(-6.5)=-7
Round(6.5)=7
Round(6.1)=6

♣ تابع الـ Frac :

Frac:real **à** real

يقوم باقتطاع القسم العشري من العدد؛ دخله و خرجه من النمط الحقيقي .

مثال :

Frac(3.33)=0.33

♣ توابع الترتيب السلمي succ,pred :

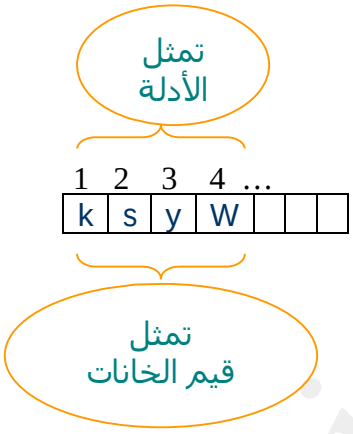
- Succ : يعطي العنصر التالي succ(i)=i+1
- pred: يعطي العنصرالسابق pred(i)=i-1

المصفوفات
Array
Vector and Matrix

يمكن تخيل المصفوفة في البرمجة كأنها جدول من الخانات و لكل من تلك الخانات عنوان(دليل) يمكن الوصول إلى الخانة مباشرة بواسطته . يستفاد منها في تجميع المعطيات المتماثلة في النمط .
جميع الأنماط التي ذكرت سابقاً هي أنماط بسيطة ، أما المصفوفة تعتبر من بنى المعطيات المعقدة .
للمصفوفات عدة أنواع تبعاً لعدد الأبعاد لكن المستخدمة بكثرة هي الأحادية و الثنائية .

- المصفوفات الأحادية البعد (الشعاع vector) :

يمكن تخيلها بشكل خانات مرتبة قرب بعضها البعض ذات أدلة متتالية كما في الشكل التالي مثلاً :



يتم تعريف المصفوفة الأحادية كما يلي :

```
Array name : array [ const1..const2 ] of   Type of Elements ;
```

على أن تكون هذه الثوابت من نمط متقطع و أن يكون الثابت الثاني أكبر من الأول حيث ستتراوح قيمة الدليل من الثابت الأول إلى الثاني و يكون حجم المصفوفة (طولها) الأعظمي (عدد الخانات الأعظمي المسموح باستخدامها) يساوي طول المجال بين الثابتين ، و تكون عناصر المصفوفة من النمط الذي نضعه بعد of .

مثال :

```
Vec1: array [1..10] of integer;
Vec2: array [1..10] of char ;
Vec3: array['a'..'f']of real;
```

تتم قراءة المصفوفة الأحادية و كتابتها و التعامل معها باستخدام إحدى التعليمات التكرارية ... للوصول إلى كل الخانات المقصودة .
مثال :

```
For i:=1 to n do
  Read(A[i]);
```

- المصفوفة الثنائية البعد (Matrix) :

المصفوفة الثنائية البعد هي عبارة عن مصفوفة مصفوفات أحادية أي لو اعتبرنا كل خانة في المصفوفة الأحادية هي عبارة عن مصفوفة أحادية ينتج لدينا مصفوفة ثنائية .
كما يمكن اعتبار المصفوفة الأحادية عبارة عن حالة خاصة من مصفوفة ثنائية ذات سطر واحد .

يتم تعريفها كما يلي :

```
Array name :array [Lines Domain, columns Domain ]of Type of Elements ;
```

مثال :

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

تتم قراءة المصفوفة الثنائية و كتابتها و التعامل معها باستخدام اثنتين من الحلقات التكرارية المتداخلة ... للوصول إلى كل الخانات المقصودة .

مثال:

```
For i:=1 to n do
  For j:=1 to m do
    Read(A[i,j]);
```

- السلسلة المحرفية string :

السلسلة المحرفية هي مصفوفة أحادية مضمونها محارف و دليلها Integer يبدأ من الواحد .

لا تتجاوز السلسلة المحرفية الـ ٢٥٥ حرفاً ، إذا لم نحدد الطول الأعظمي للسلسة عند تعريفها فإنه يكون ٢٥٥ .

عند تعريف مصفوفة يحجز لها خانات بطولها + خانتين إضافيتين هما الخانة رقم ٠ و تحوي طول السلسلة و الخانة التي تزيد عن طولها ب ١ و تضم رمز لاغي مهمته الدلالة على طول السلسلة

يتم تعريفها إما بتحديد طولها :

```
S:string [20];
```

أو بدون تحديد الطول عندها يكون ٢٥٥ :

```
S:string ;
```

يمكن معاملة الـ string كما المصفوفة كما أنها تمتاز بأشياء إضافية فمثلاً لكتابتها و قراءتها يمكن الاكتفاء بكتابة أسمها دون الحاجة لتعليمة for كما يلي :

```
Writeln(s);
Readln(s);
```

من التوابع المسبقة التعريف التي تتعامل مع السلاسل المحرفية string :

Length : الذي يأخذ كوسيط سلسلة محرفية و يرد طولها (عدد محارفها) .

مثال :

```
S:= 'Hello world'
Writeln (Length(s));
```

الخرج
11

Concat : يقوم بضم مجموعة من السلاسل الممررة إليه كوسطاء في سلسلة واحدة

مثال :

```
S1:='Hello ';
S2:='world' ;
S3:=concat(s1,s2);
Writeln (s3);
```

الخرج :
Hello world

البرامج الجزئية
التوابع و الإجرائيات

من الأفضل تقسيم البرنامج لعدة كتل برمجية لكل كتلة وظيفة معينة تقوم بها، فمبدأ التجزئة بهدف السيطرة (فرق تسد) مفيد جداً في البرمجة للأسباب التالية :

١. سهولة فحص كل كتلة على حدا، لسهولة معرفة موقع الخطأ في البرنامج بدقة من قبل المبرمج .
٢. سهولة التعديل عند الحاجة .
٣. لمنع التكرار؛ فلو اضطررنا إلى تكرار كتلة من التعليمات تقوم بمهمة معينة عدة مرات فمن الأفضل وضعها في برنامج جزئي و نكتفي بذكر اسم هذا البرنامج الجزئي عند الحاجة إليه .
٤. سهولة فهم الكود عند قراءته مبسطاً مقسماً .

تسمى هذه الكتل البرمجية بالبرامج الجزئية Sub_Programs و تتوضع هذه البرامج الجزئية بعد قسم الـ var في الجسم الرئيس للبرنامج .

و هي على نوعين :

- الإجرائية procedure .
- التابع function .

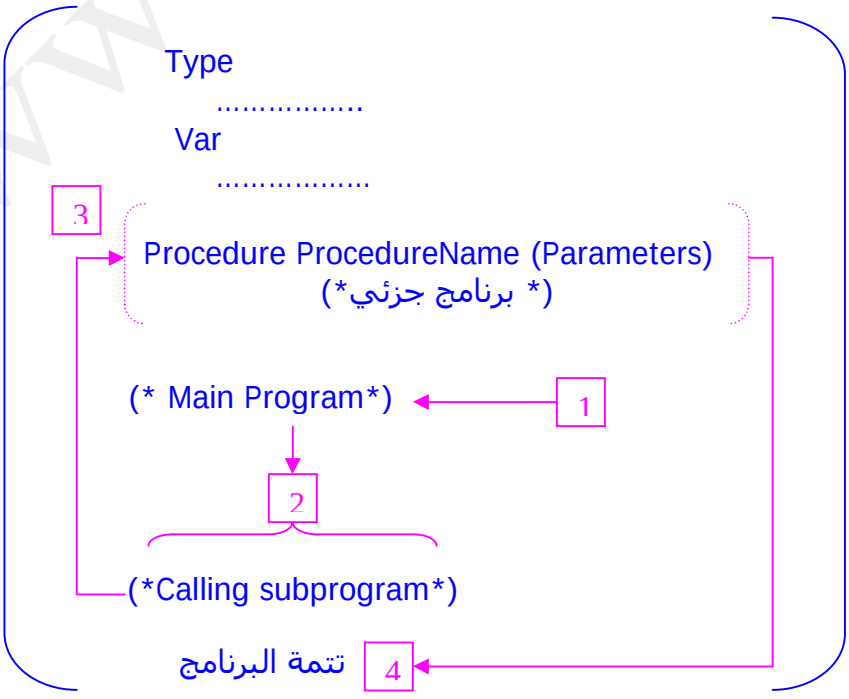
الإجرائية (Procedure) :

- تعريف الإجراء و استدعاءه :

من المهم أولاً التمييز بين الإجراء و استدعائه ، فمن أجل استخدام الإجراء يجب تعريفه أولاً ثم استدعاءه من خلال البرنامج .

و استدعاء الإجراء هو عملية يتم من خلالها نقل تنفيذ البرنامج من العبارة المعالجة قبل الاستدعاء إلى تعليمات الإجراء المستدعى، فعندما نستدعي الإجراء تنفذ مجموعة التعليمات المحتواة في هذا الإجراء ، و بعد الانتهاء من تنفيذ آخر تعليمة من تعليمات الإجراء ينقل التحكم إلى النقطة التي تأتي مباشرة بعد أمر استدعاء الإجراء في البرنامج الرئيسي .

و المخطط التالي يوضح كيف يتم ذلك :



إن البنية العامة للإجراء في لغة الباسكال تشبه إلى حد كبير بنية البرنامج الرئيسي في لغة باسكال . و يتألف الإجراء على وجه التحديد من الأجزاء التالية :

رأس الإجراء A Procedure Head
قسم التصريح A Declaration Section
جسم الإجراء A Procedure Body

♣ أولاً : رأس الإجراء :

يتألف رأس الإجراء من الكلمة المحجوزة Procedure متبوعة باسم الإجراء و يمكن أن يتبع الاسم بلائحة من الوسطاء(Parameters) ، و يجب أن ينتهي رأس الإجراء بفاصلة منقوطة لنفصله عن باقي أجزاء الإجراء .
و بذلك يكون الشكل العام لترويسة الإجراء:

```
Procedure it's name  (parameter1 :it's type ; parameter2 : it's type ;.....) ;
```

♣ ثانياً : قسم التصريح :

يملك كل إجراء قسم تصريح خاص به ، يمكن أن نصرح فيه عن ثوابت أو متحولات أو أنماط مختلفة ، أو أن تعرف فيه إجراءات و توابع أخرى أيضاً .
و طريقة التصريح عن هذه الأنواع تخضع لنفس الطريقة المستخدمة في البرنامج الرئيسي مع بعض الفوارق البسيطة في إمكانية الوصول إلى التعريفات و التصريحات حيث لا يمكن استخدام هذه المتحولات إلا ضمن التابع نفسه لأنها متحولات محلية (local).

♣ جسم الإجراء :

يحتوي جسم الإجراء على مجموعة من العبارات الضرورية ليقوم الإجراء بالمهمة الموكلة إليه . يبدأ جسم الإجراء بـ Begin و ينتهي بـ End متبوعة بفاصلة منقوطة .

♦ المتحولات العامة و المتحولات الموضعية global variable and local variable :

المتحولات العامة هي المتحولات المصرح عنها ضمن قسم التصريح الخاص بالبرنامج الرئيسي و هذه المتحولات يمكن استخدامها في البرنامج الرئيسي و في جميع التوابع و الإجرائيات المعرفة ضمن هذا البرنامج ، يتلف المتحول العام و تتحرر الذاكرة التي حجزها عند انتهاء البرنامج .

أما المتحولات الموضعية فهي المتحولات المصرح عنها في قسم التصريح لكل من الإجرائيات و التوابع . يتلف و تتحرر الذاكرة التي حجزها عند انتهاء تنفيذ الإجرائية التي عرف فيها) .

ملاحظة :

لو عرفنا متحول عام وآخر محلي بنفس الاسم فالإجرائية سوف تتعامل مع المتحول المحلي و لا يمكنها أن تجري أي تغيير على المتحول العام .

مثال :

```
Program test ;
Var
  X:integer;
Procedure change ;
Var
  X:integer;
```

متحول عام
Global

متحول محلي
Local

```
Begin
  X:=1 ; (* x local *)
End;
Begin
 X:=0; (* x global *)
Change;
Writeln('x=',x); (* x global *)
End;
```

إن خرج هذا البرنامج هو 0 لأن التغييرات ضمن الإجرائية طرأت على المتحول المحلي x و ليس على المتحول العام x .

♦ الوسطاء (Parameters) :

يفضل أن نحدد للإجراء المتحولات اللازمة لعمله عند استدعائه ، يتم ذلك في بداية الإجراء حيث يملك رأس الإجراء لائحة اختيارية من الوسطاء مع نوع كل منها وتضم وسيط واحد أو أكثر كما يمكن الاستغناء عنها كلها .

و السؤال الذي يطرح نفسه هنا ، لماذا لا نستخدم المتحولات العامة في التابع بدلاً من الوسطاء ..؟؟؟

لأننا قد نستخدم نفس الإجراء أكثر من مرة على أكثر من متحول ، و عندما نستعمل المتحولات العامة بدلاً من الوسطاء سوف نضطر إلى كتابة إجراء جديد مماثل للسابق و لكن بأسماء متحولات مختلفة ... و لن تتحقق الفائدة المرجوة من استخدام الإجراءات .

v استدعاء الإجرائية :

عند استدعاء الإجرائية علينا ذكر أسم الإجراء يليه لائحة من المتحولات الوسيطية بنفس عدد و ترتيب وسطائه الشكلية المعرفة ضمن الترويسة أو الـ Header الخاص به .

مثال على الإجرائية :

إجرائية لكتابة مصفوفة :


```
Procedure printVec (vec:vector,n:integer);
Var
 I:integer;
Begin
  For i:=1 to n do
     Print(vec[i]);
End;
```

أما الاستدعاء فسيكون ضمن الجسم الرئيسي للبرنامج على الشكل :

```
PrintVec(mat,m);
```

حيث mat من نمط vector المعرف سابقاً في قسم الـ Type الخاص بالبرنامج الرئيسي و m من نمط Integer .

هناك نوعان لتمرير الوسطاء إلى الإجراء :

- التمرير بواسطة القيمة Passed by value
- التمرير بواسطة المرجع Passed by reference

♣ التمرير بواسطة القيمة :

و هو التمرير العادي الذي ألفناه في تعاملنا السابق مع التوابع و الإجرائيات ، لكن هنا علينا التمييز بين الوسطاء الشكلية و المتحولات الوسيطية Formal Parameters and Arguments :

- *فالوسيط الشكلي (parameter)* هو الذي نعرفه في رأس الإجراء .
- *و المتحول الوسيطي(argument formal)* هو الذي نمرره للإجراء عند الاستدعاء .

إنهما متشابهان و لكنهما منفصلين أي أن لكل منهما موقع منفصل في الذاكرة (في الحالة العامة) .

فعندما نمرر للإجراء متحول وسيطي يتم نسخ قيمته و إعطائها لمكان تخزين مؤقت ضمن الذاكرة مخصص للوسيط الشكلي و يتم تنفيذ جميع العمليات ضمن الإجراء على الوسيط الشكلي ، و إن تغير الوسيط الشكلي ضمن هذا الإجراء فإن المتحول الوسيطي سيبقى محافظاً على قيمته السابقة قبل الاستدعاء .

كما أن عمر الوسيط الشكلي يقتصر على فترة تنفيذ الإجراء ، فبعد الإنتهاء من تنفيذه يتم تحرير موقع الذاكرة الذي كان محجوزاً للوسيط الشكلي و يمكن استخدامه من قبل متحولات أخرى .

ما الذي يسمح لنا بتمريره للإجراء بواسطة القيمة ؟؟؟

يسمح بتمرير :

- القيم الثابتة ... مثال: 30 أو pi حيث أننا عرفنا الpi كثابت في بداية الرنامج و زودناه بالقيمة 3.14 .
- المتحولات ... مثال : x, Age .
- التعابير ، مثال : a+b, a>b حيث أن التعبير a>b سوف تؤخذ نتيجته المنطقية و تمرر للإجراء مكان الوسيط الشكلي المنطقي .

♦ الوسطاء ذات الاتجاهين Two-Way Parameter :

في بعض الأحيان نكون بحاجة للاحتفاظ بقيمة المتحولات بعد الانتهاء من تنفيذ الإجرائية، لكن الطريقة السابقة في تمرير المتحولات و التي تسمى التمرير بواسطة القيمة Passed by value لا تؤمن لي هذا .

لذلك وجدت طريقة أخرى للتمرير تسمى التمرير المرجعي أو Passed by reference و فيها لا يتم حجز مكان جديد للذاكرة من أجل الوسيط الشكلي، إنما سيصبح الوسيط الشكلي و المتحول الوسيطي اسمان لنفس الموقع في الذاكرة و ما يجري على الوسيط الشكلي من تغيير سيطرأ على المتحول الوسيطي أيضاً .

يسمى الوسيط الشكلي السابق ذو اتجاهين أو متحول دخل و خرج .

و يتم تعريف وسيط ذو اتجاهين بوضع كلمة var قبل الوسيط الشكلي في رأس الإجراء.

ما الذي يسمح لنا بتمريره للإجراء بواسطة المرجع .؟؟

لا يمكن تمرير قيمة ثابتة أو تعبير ما للإجراء مكان وسيط ذو اتجاهين، و يقتصر التمرير هنا على المتحولات فقط .

مثال :
لو أردنا كتابة إجرائية (swap) للإبدال بين قيمتي المتحولين لاحظ الفرق بين الإجرائيتين التاليتين لندرك الفرق بين التمرير بالقيمة و التمرير بالمرجع :

التمرير بواسطة القيمة passed by value	التمرير بواسطة المرجع passed by reference

Program test ;
Var
a,b:integer;
procedure swap (x,y:integer);
var
T:integer ;
Begin
T:=x;
X:=y;
Y:=T;
Writeln ('x=',x,' , y=',y);
End;
 	
Begin
a:=10;
b:=5;
Swap(a,b);
Writeln ('a=',a,' , b=',b);
Readln;
End.
خرج هذا البرنامج
X:=5,y=10
A=10,b=5
 	
أي أن التغييرات التي طرأت على الوسطاء الشكلية لم تتم على المتحولات الوسيطية فبقيت الأخيرة محافظة على قيمتها بعد استدعاء الإجرائية .
	Program test ;
	Var
	a,b:integer;
	procedure swap (var x,y:integer);
	var
	T:integer ;
	Begin
	T:=x;
	X:=y;
	Y:=T;
	Writeln ('x=',x,' , y=',y);
	End;
	
	Begin
	a:=10;
	b:=5;
	Swap(a,b);
	Writeln ('a=',a,' , b=',b);
	Readln;
	End.
	خرج هذا البرنامج
	X:=5,y=10
	A=5,b=10
	
	أي أن التغييرات التي طرأت على الوسطاء الشكلية قد تمت على المتحولات الوسيطية أيضاً .

التابع Function :

إن الاختلاف بين التوابع و الإجرائيات بسيط جداً من ناحية القواعد الإملائية، لكن الأهم هو أن نميز بينهما من ناحية المفهوم ، لكي نعرف متى نستخدم التابع و متى نستخدم الإجرائية .

لقد صممت الإجرائية للقيام بعمل ما فقط ، مثل قراءة مصفوفة أو طباعتها أو التبديل بين متحولين .
أما التابع فقد صمم لحساب قيمة ما أيضاً ، مثل حساب قيمة الجذر التربيعي لرقم، أو إيجاد مربع رقم ما ، إلخ ...

أي أننا نتوقع من التابع أن يعيد قيمة ما عند انتهاء تنفيذه حتى لو كان ذو مهمة معينة مثل قراءة مصفوفة إلا أنني قد أستخدمه في هذه الحالة ليعيد عدد العناصر التي تمت قراءتها بنجاح مثلاً .

هل أدركنا الآن الفرق بين التابع و الإجرائية ؟!

- تعريف التوابع في الباسكال :

يبدأ رأس التابع بالكلمة المحجوزة Function يليه اسم التابع ثم لائحة من الوسطاء الاختيارية محصورة بين قوسين، و ننهي الترويسة بالنمط الذي سيرده التابع و **يجب أن يكون نمطاً بسيطاً** .

و بذلك يكون الشكل العام لترويسة التابع :

```
function it's name (parameter1:it's type ; parameter2 : it's type ;.....):function type ;
```

و ما تبقى من باقي أجزاء التابع فإنها مماثلة لمقابلاتها في الإجرائية، إلا أن جسم التابع يجب أن يحوي على تعليمة إسناد قيمة ما لاسم التابع، و لكن بشرط أن تكون هذه القيمة من نفس نوع المعطيات الذي يردّه التابع .

- استدعاء التابع calling function :

يمكن أن نعامل التابع عند الاستدعاء كما نعامل الثوابت تقريباً ، فمن الممكن إسناد قيمة التابع لمتحول ما و من الممكن استخدام قيمته في تعبير ما مباشرة كما من الممكن تمرير قيمته لنستخدمها في إجراء أو تابع آخر، و لكن من غير الممكن إسناد قيمة ما للتابع مثلاً و من غير الممكن قراءة التابع .

أمثلة على الاستدعاءات الصحيحة :

```
Y:=sq r(x);
Write(sq r(x));
X:=1+sqr(y);
```

أمثلة على الاستدعاءات الخاطئة :

```
Sqr(x):=y;
Read(sqr(x));
Sqr(x):=1+Y;
```

بالطبع إذا كنا نفكر بطريقة منطقية فلن نقع في أخطاء استدعاء التابع ؛ لأن جميع الكتابات الخاطئة السابقة لاستدعاء الإجراء تعطي التابع قيمة ما و **التابع يرد قيمة و لا يأخذ قيمة** .

و ما تبقى من حديثنا عن الإجرائيات ينطبق على التوابع أيضاً من تمرير وسطاء و غيره .

مثال على التابع :
تابع يقوم بإرجاع القيمة الكبرى من قيمتين نمررهما له :

```
Function Max (a,b:integer):integer;
Begin
  If(a>b) then
```

```
    Max:=a
  Else
    Max:=b;
End;
```

ملاحظة :

١. نطلق اسم روتين على البرنامج الجزئي سواء كان تابع أو إجرائية .
٢. بما أن الروتين هو عبارة عن برنامج جزئي و ينطبق عليه صفات البرنامج الرئيسي فيمكن استدعاء روتينات أخرى بداخله شريطة أن تكون معرفة قبله أو مصرح عنها قبله كما يمكن للروتين نفسه أن يستدعي نفسه بداخله (العودية) ، و أكثر من ذلك ... يمكن أن نعرف روتينات أخرى ضمنه كما نعرفها في البرنامج الرئيسي عادة (الروتينات المتداخلة) ، و لكنها تخضع لعدة شروط صلاحية و نفاذ [للمزيد من المعلومات عن الروتينات المتداخلة قم بعودة إلى الفصل السادس من المرجع الثاني المذكور في النهاية علماً أنها غير مذكورة في منهاج البرمجة ١] .

العَودية
Recuserve

ما معنى العودية ؟؟؟؟

أي أن الروتين يعود ليستدعي نفسه بداخله و بالتالي عندما يتم تنفيذ الروتين مرة أخرى بعد الاستدعاء الداخلي الأول يجب أن يستدعي نفسه مرة أخرى أيضاً و هكذا، لذلك يجب أن يكون هناك شرط ليوقف استدعاء الروتين لنفسه و إلا سندخل في حلقة لا نهائية من التعليمات .

متى يمكن أن نفكر بالحل العودي ؟؟

نفكر به عندما يمكننا معرفة قيمة الروتين من أجل قيمة ما n إذا عرفنا قيمة هذا الروتين عند قيمة ما لها علاقة بـ n، كأن تنتج عن n بطرح ١ أو بقسمتها على ٢ مثلاً .

ما هي خواص الروتين العودي ؟؟

١. أن يتضمن الروتين شرطاً لإيقاف الاستدعاء العودي .
٢. أن يتم تغيير قيمة أحد الوسطاء قبل تمريرها إلى تعليمة استدعاء الروتين من جديد عودياً، على أن يكون هو الوسيط الذي يتوقف عليه شرط التوقف .

مخالفة أحد هذين الشرطين ستدخل البرنامج في حلقة لا نهائية من التعليمات .

ما هي الآلية التي يتم وفقها الاستدعاء العودي ؟؟

مفهوم العوديّة يعتمد على بنية المكدّس و الفكرة الرئيسيّة في هذه البنية هي : first in last out أي أنّ المتحوّل الذي يدخل في البداية سوف يكون هو ناتج الخرج النهائي .

و في الروتينات العودية الروتين الذي يستدعى أولاً سيكون آخر روتين ننتهي من تنفيذه ، و الروتين الذي يستدعى آخراً هو أول روتين ننتهي من تنفيذه .

لإيضاح فكرة العودية نأخذ مثال العاملي، و بفرض أننا أردنا حساب عاملي الـ ٥ علماً أن عاملي الـ ١ هو ١ .

نعلم أن إيجاد العاملي مسألة يمكن تعريفها عودياً ببساطة بالشكل :

$$Fact(n) = \begin{cases} Fact(1) & \text{if } n=1 \\ Fact(n-1)*n & \text{if } n>1 \end{cases}$$

fact(5)=fact(4)*5
but : fact(4)=fact(3)*4
but : fact(3)=fact(2)*3
but : fact(2)=fact(1)*2
but : fact(1)=1

then : fact(2)=1*2=2
then : fact(3)=2*3=6
then : fact(4)=6*4=24
then : fact(5)=24*5=120

و يكتب تابع العاملي كما يلي :

```
Function fact (x : integer):integer;
Begin
  If (x=1) then
     Fact:=1
  Else
     Fact:=x * fact(x-1);
End;
```

هذا شرط التوقف عن الاستدعاء

لاحظ كيف تم تغيير قيمة الوسيط قبل تمريره للتابع من جديد

هناك نوع آخر من العودية بأن يقوم روتين باستدعاء روتين آخر و يقوم الروتين الثاني باستدعاء الروتين الأول أيضاً، لكن للقيام بذلك يجب أن يكون الروتين المعرف أولاً على معرفة أنه هناك تابع معرف بهذا الاسم ، لذلك نقوم بوضع الترويسة أو الـ Header الخاص بالروتين الثاني قبل تعريف الروتين الأول متبوعاً بكلمة forward ثم فاصلة منقوطة
مثل :

```
Function func2 (...  parameters and there types ..)  : نمطه  ; forward ;
```

أو ،

```
Procedure proced2 (...parameters and there types ...) ; forward ;
```

[لمزيد من المعلومات حول هذا النوع من الروتينات العودية راجع الفصل السادس من المرجع الثاني المذكور في النهاية، علماً أنها غير مذكورة في منهاج البرمجة١].

ملاحظة :
لكل خوارزمية عودية خوارزمية تكرارية مكافئة تماماً .
و يمكن التحويل من الخوارزميات العودية إلى التكرارية و بالعكس .
علماً أنه قد تكون الحلول التكرارية أسرع بكثير من الحلول العودية في بعض المسائل، و قد تكون الحلول العودية أسرع من الحلول التكرارية في مسائل أخرى .

ملحق :

سلسلة فيبوناتشي (Fibonacci) العودية و مشكلة توالد الأرانب J :

من المعروف أن الأرانب تتوالد بشكل سريع جداً، لأن فترة الحمل عندها تقارب الشهر، و لأن الأرانب تصبح قادرة على الإنجاب بعد حوالي الشهر من ولادتها، كما أنها لا تمضي إلا بعض الأيام في العناية بصغارها .

و لم يتمكن المختصون من إيجاد طريقة يحصون فيها عدد الأرانب التي يحصلون عليها بعد عدة أشهر انطلاقاً من زوج صغير من الأرانب ، فجاء العالم فيبوناتشي ليوجد طريقة رياضية لإيجاد عدد الأرانب في كل شهر و هنا رأت سلسلة فيبوناتشي النور .

المعطيات:
١. انطلق فيبوناتشي من زوج صغير من الأرانب حديث الولادة .
٢. كل شهر يلد زوج الأرانب زوجاً جديداً .
٣. زوج الأرانب المولود يصبح قابلاً للإنجاب بعد شهر من ولادته .

في الشهر الأول أحضرنا زوج صغير من الأرانب مولود حديثاً ، (لدينا زوج واحد) .
في الشهر الثاني أصبح هذا الزوج قادراً على الإنجاب، (لدينا زوج واحد) .
في الشهر الثالث أنجب الزوج السابق زوجاً جديداً من الأرانب، (أصبح لدينا زوجان).
في الشهر الرابع عاد الزوج الأصلي لينجب زوجاً جديداً، كما أن الزوج الصغير الذي أنجبه في الشهر الماضي أصبح كبيراً، (أصبح لدينا ثلاثة أزواج) .
في الشهر الخامس أنجب الزوج الأصلي زوجاً و الزوج الذي أنجباه في الشهر الثالث أيضاً أنجب زوجاً، كما أن الزوج الذي أنجبه الزوج الأصلي في الشهر الرابع أصبح كبيراً، وهكذا أصبه لدينا خمس أزواج من الأرانب) .
و هكذا .....
لو أردنا تمثيل التولد السابق كما يلي :

زوج صغير
زوج كبير
زوج كبير +زوج صغير
زوج صغير + زوج كبير + زوج كبير
زوج كبير + زوج صغير + زوج كبير + زوج كبير + زوج صغير
زوج صغير+زوج كبير +زوج كبير + زوج صغير +زوج كبير+ زوج كبير+زوج صغير+ زوج كبير

و هكذا كل شهر الزوج الصغير يصبح كبير و الكبير يصبح صغير و كبير ، نلاحظ أنه في كل شهر تستطيع الأسرة أن تنجب أزواجاً بعدد الأزواج في الشهر الذي قبل السابق، لأن عدد الأزواج الكبيرة القادرة على الإنجاب في هذا الشهر يساوي عدد الأزواج التي كانت

كبيرة في ذلك الشهر (قبل السابق) + عدد الأزواج التي كانت صغيرة في ذلك الشهر (قبل السابق) و استغرقت شهر لتكبر و شهر لتنجب .

إذاً عدد الأزواج في الشهر الحالي =عدد الأزواج التي أنجبت حديثاً + عدد الأزواج الأصلية.
أي = عدد الأزواج في الشهر قبل السابق + عدد الأزواج في الشهر السابق .

هنا جاء فيبوناتشي و أنشأ سلسلة سماها باسمه أرقام حدود هذه السلسلة تمثل الشهر المقصود و قيمة الحد تمثل عدد الأرانب المولودة حتى ذلك الشهر، أي قيمة كل حد من متتالية فيبوناتشي = قيمة الحد السابق + قيمة الحد قبل السابق .

```
Fib(n)=fib(n-1)+fib(n-2)
```

علماً أن :

```
Fib(1)=1
Fib(2)=1
```

فيما بعد تم تطبيق هذه المتوالية مع بعض النباتات أيضاً .

لكن سلسلة فيبوناتشي هذه تحتاج لبعض التعديلات إذا طبقت على أرض الواقع ، لأن مدة حمل الأرانب ٤٠ يوم و ليس شهراً **J** كما أن الأرانب بحاجة لحوالي أسبوعين بعد ولادتها تقضيها في العناية بأبنائها، بالإضافة إلى أن الأرانب لن تنجب صغاراً بعد خمس سنين تقريباً من ولادتها !!!

المراجع المستخدمة :

- كتاب البرمجة ١ ، من منشورات جامعة دمشق .
- Turbo Pascal 7 إصدار دار شعاع للنشر و العلوم .

لأسئلتكم و اقتراحاتكم
ite.sy.net@gmail.com

مع تحيات
فريق عمل الـ ite

تم بعونه تعالى