الباب الخامس

الدوال والإجراءات
Functions & Procedures

5–1 مقدمة

تواجه المبرمج في كثير من الأحيان مسائل تتطلب برمجتها إعادة مجموعة من الجمل في البرنامج عدة مرات نظراً لتكرار العمل الذي تؤديه هذه المجموعة في أماكن مختلفة من البرنامج. وهذا بالطبع يؤدي إلى طول غير ضروري وهو مايمكن تجنبه باستخدام ما يسمى بالبرامج الفرعية (Subprograms) . ويوجد نوعان من هذه البرامج هما :

- الإجراءات procedures
- والدوال Functions .

والغرض منها هو تقسيم البرنامج إلى وحدات مستقلة ، كل وحدة تؤدي غرضاً معيناً ، وتستدعى هذه الوحدة عند النقطة المحددة (بمجرد استدعاء اسمها) كلما دعت الحاجة إلى ذلك بهذه الطريقة ، يصبح البرنامج أكثر وضوحاً ، وأكثر قابلية للتعديل وتصحيح الأخطاء .

إن استخدام الإجراءات في البرمجة أمر ضروري لتناسق البرنامج وسهولة قراءته واستخدامه . وبالتالي فإننا نجد أن أغلب برامج باسكال الجيدة تتكون عادة من مجموعة من الإجراءات والدوال المترابطة ، التي يتم استدعاؤها من البرنامج الرئيسي ، أو من واحدة للأخرى .

5–2 إجراء لحساب مجموع مصفوفة

يمكننا ببساطة كتابة برنامج لحساب مجموع عناصر مصفوفة ذات عناصر النوع العددي بدون استخدام الإجراء أو الدالة . ولكن سنبين هنا مزايا استخدام الإجراء ، ونكتب إجراءين بحيث يقوم كل إجراء بوظيفة خاصة ، وذلك لغرض توضيح فكرة استخدام الإجراء في تقسيم البرنامج إلى وحدات مستقلة.

يبين الشكل (**5–2–1**) البرنامج المطلوب ، وهو برنامج يحتوي على إجراءين واحد اسمه ReadArray ، والثاني اسمه FindSum .

يقوم هذا الإجراء الأول بقراءة المصفوفة ، ويقوم الثاني بإيجاد مجموع عناصر المصفوفة .

نلاحظ أن الإجراء يبدأ بكلمة BEGIN وينتهي بكلمة ; END . وأن الإجراء يتم استدعاؤه بمجرد وضع اسمه ، أي أن الجملة :

```
ReadArray;
```

هي طلب بالتحول إلى الإجراء المسمى بهذا الاسم وتنفيذه . وهذا الطلب قد يأتي في أي موضع من البرنامج الرئيسي (main program) الذي يبدأ بعد نهاية الإجراء . كما نلاحظ أن البرنامج الرئيسي قد أصبح مختصراً ، فهو يتكون في هذا المثال من **4** جمل فقط.

لاحظ في حالة كتابة هذا البرنامج بدون استخدام الإجراءين المذكورين لن يكون البرنامج بهذا الوضوح بل يحتاج إلى تركيز أكثر لتتبعه وفهم خطواته .

يجب أن نلاحظ هنا أننا استخدمنا أبسط طريقة في كتابة الإجراء ، وفيها اعتبرنا أن المتغيرات في البرنامج هي متغيرات عامة ، أي يتم استخدامها من قبل البرنامج الرئيسي والفرعي بنفس المدلول والمعنى .

سنكتب في البنود القادمة برامج فرعية أكثر استقلالية وشمولية.

شكل (5-2-1) : برنامج لحساب مجموع مصفوفة .

```
PROGRAM sumOfArray;
VAR n, k : INTEGER;
    x: ARRAY[1..20] OF REAL;
    sum : REAL;
PROCEDURE readArray;
BEGIN
     WRITE('Enter size of array-->');
     READ(n);
     FOR k:=1 TO n DO
     BEGIN
          WRITE('Enter x[',k,']-->');
          READ(x[k])   ;
     END;
END;  { readAray }
PROCEDURE FindSum;
BEGIN
     sum:=0;
     FOR k:= 1 TO n DO
         sum := sum + x[k];
END;
BEGIN  {main}
       WRITELN;
       ReadArray;
       FindSum;
       WRITELN('sum= ',sum:12:3);
END.
```

5-3 تمرير البيانات إلى الإجراء

يعتبر الإجراء (procedure) برنامجاً فرعياً (subprogram) لـــه مدخلاتـــه ومخرجاته. في البرنامج (5-2-1) لم نحدد هذه المدخـــلات والمخرجـــات لأن الإجراء ReadArray لم يكن إلا مجرد مجموعة من الجمل كونت وحدة في البرنامج لقراءة المصفوفة ، و لم يكن له متغيرات مستقلة خاصة به. والافتراض نفسه ينطبق على الإجراء FindSum .

إلا أن الصورة العامة للإجراء تحتوي على إمكانية تحديد المتغيرات الخاصة به، وبذلك يصبح وحدة مستقلة قابلة للاستخدام في أكثر من برنامج رئيسي. والمثال التالي يوضح ذلك .

مثال (1–3–5)

يبين البرنامج في الشكل (1–3–5) و المسمى Histogram كيف نستخدم إجراءً مستقلاً في متغيراته عن البرنامج الرئيسي . فالإجراء :

PROCEDURE stars (J: INTEGER);

شكل (1–3–5) : برنامج لرسم شكل بياني .

```
PROGRAM histogram;
VAR k,n : INTEGER;
    a: ARRAY[1..20]OF INTEGER;
PROCEDURE stars( j : INTEGER);
BEGIN
     WHILE( j>0 ) DO
     BEGIN
          WRITE('*');
          j:=j-1;
     END;
     WRITELN;
END;
BEGIN
     WRITE('Enter number of values-->');
     READ(n);
     FOR k:=1 TO n DO
     BEGIN
          WRITE('Enter a[',k,']-->');
          READ(a[k]);
     END;
     { clear screen } FOR k:=1 TO 25 DO
WRITELN;
     FOR k:= 1 TO n DO stars(a[k]);
END.
```

لا توجد به متغيرات من البرنامج الرئيسي ، ولكن المتغير الوحيد الموجود به هو **j** ويسمى متغير شكلي (formal parameter) ، وقد تم تحديد هذا المتغير ونوعه بين قوسين بعد اسم الاجراء stars مباشرة .

كما نلاحظ أن هذا المتغير لم تحدد له قيمة داخل الإجراء و لكن الاستدعاء .

```
stars (A [k]);
```

يمرر قيمة A[k] المعلومة إلى المتغير j . وهذا ما يعرف بالاستدعاء بالقيمة (call-by-value) من البرنامج الرئيسي إلى البرنامج الفرعي (الإجراء) ، ويسمى المتغير A[k] في هذه الحالة متغيراً فعلياً (actual parameter) .

ونلاحظ أن العمل الذي يقوم به الإجراء stars هو طباعة * في سطر واحد عدد **j** من المرات . أما البرنامج الرئيسي فيقوم بإدخال عدد من القيم مقداره **n** وبتمرير هذه القيم إلى الإجراء نحصل على رسم بياني بالأعمدة . فمثلاً إذا كانت .

```
n = 3
a [1] = 4, a [2] = 6, a [3] = 2
```

فإننا نحصل على الرسم التالي

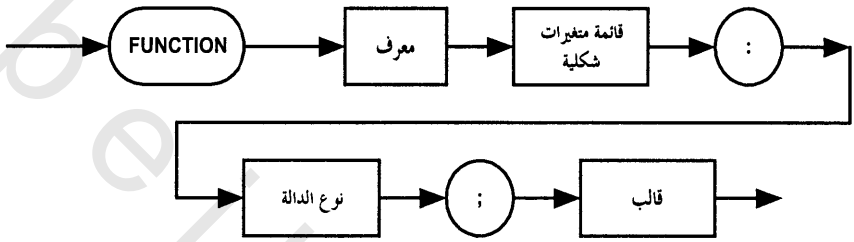
```
* * * *
* * * * * *
* *
```

5-4 الدالة Function

رأينا أن الإجراء يقوم بوظيفة معينة مع إمكانية تمرير بعض المدخلات له ففي المثال (1-3-5) قام الإجراء stars بطباعة عدد **j** من الرموز في سطر واحد حيث **j** متغير خاص بالإجراء ، وتعطي له قيمة من البرنامج الرئيسي. ولكن ماذا لو أردنا الحصول على نتيجة من نوع مقابل المدخلات ؟ هذا ما يقوم به الدالة (Function) في لغة باسكال .

تعتبر الدالة (مثل الإجراء) وحدة مستقلة في بنية البرنامج ، و تتكون من مقدمةheading (اسم الدالة ومتغيراتها الشكلية) ثم نوع الدالة ثم القالب block (تحديد المتغيرات والجزء التنفيذي) .و يبين الشكل (1–4–5) رسماً للبنية اللغوية للدالة .

شكل (1–4–5) : بنية الدالة .



لاحظ أن قائمة المتغيرات الشكلية يجب أن تكون محصورة بين قوسين ( ) ، وأن نوع كل متغير يجب تحديده ، كما في المثال التالي :

مثال (1–4–5) :

اكتب برنامجا لحساب الدالة:

$$p(x) = 2x^2 + 5x - 3$$

عند x=3.5 و x=4.5 .

يبين البرنامج في الشكل (**2–4–5**) مثالاً لاستخدام الدالة التي تقوم بحساب **p(x)** حيث:

$$p(x) = 2x^2 + 5x - 3$$

وحيث x متغير كسري . في البرنامج الرئيسي تم استدعاء هذه الدالة مرتين :

المرة الأولى في جملة :

```
a: = p(3.5);
```

و المرة الثانية في جملة :

b: = p(4.5);

أي إن p(x) تعتبر قيمة تعتمد على المتغير الشكلي x ونتحصل عليها من الدالة بإدخال قيمة x الفعلية. و بما أن نوع p(x) تم تحديده من النوع الكسري في الجملة :

FUNCTION p(x: REAL): REAL;

فإن p(3.5) و p(4.5) هما قيمتان كسريتان ، ونتيجة لتنفيذ هذا البرنامج فإننا نحصل على المخرجات :

39.00 62.32

شكل (2–4–5) : حساب دالة ذات متغير واحد .

```
PROGRAM func;
VAR a,b: REAL;
FUNCTION p(x : REAL): REAL;
BEGIN
     p:= 2*x*x+5*x-3;
END;
BEGIN
     a:= p(3.5);
     b:= p(4.6);
     WRITELN(a:10:2, b:10:2);
END.
```

لاحظ أن الدالة **p** في هذا البرنامج تتكون من جملة واحدة محصورة بين كلمة BEGIN و END . وقد تتكون الدالة من أكثر من جملة واحدة ، ولكن يجب دائماً حصر هذه الجمل بين هاتين الكلمتين .

وليس من الضروري أن يكون للدالة متغير مستقل واحد ، بل يمكن أن يكون لها أكثر من متغير كما في المثال التالي :

مثال (2-4-5)

لإيضاح عملية استخدام دالة ذات 3 متغيرات نفترض أن لدينا 3 قيم ، والمطلوب كتابة دالة تعطي مجموع أعلى قيمتين.

الشكل (3-4-5) يوضح هذه الدالة مع البرنامج الرئيسي لها. نلاحظ أن الفكرة المستخدمة في هذه الدالة تعتمد على وجود 3 احتمالات للأرقام المعطاة ، وهي أن هناك قيمة من القيم الثلاثة أصغر من أو تساوي القيمتين الأخريين. فإذا كانت هذه القيم هي مثلا (52, 43, 75) فإن الناتج المطلوب هو :

$$52 + 75 = 127$$

يقوم البرنامج الرئيسي بقراءة 3 أعداد صحيحة للمتغيرات الصحيحة الثلاثة (**a, b, c**) ويستدعي الدالة best بحساب مجموع أعلى عددين ، وترجع هذه النتيجة للبرنامج الرئيسي لطباعتها .

شكل (3-4-5) : دالة لحساب أكبر مجموع من بين 3 قيم

```
PROGRAM bestTwo;
VAR a, b, c , d : INTEGER;
FUNCTION best( t1, t2,t3 : INTEGER ):
                                        INTEGER;
BEGIN
     best:= t2 +t3;
     IF (t2 <=t1) AND (t2<=t3) THEN
                    Best:=   t1 + t3;
     IF (t3 <=t1) AND (t3<=t2) THEN
                       Best := t1 + t2;
END;

BEGIN   { main program }
     WRITE('Enter 3 values-->');
     READ(a,b,c);
     d:= best(a,b,c);
     WRITELN('Sum of best 2 values=',d);
END.
```

مثال (4-4-5)

اكتب برنامجا لحساب 2.0^3 و 1.5^4 (أي 2 أس 3 و 3.5 أس 4) وذلك بأن يستدعي الدالة p التي تقوم بحساب x أس n حيث x متغير كسري و n متغير صحيح .

يبين الشكل (4-4-5) البرنامج المطلوب حيث يحتوي البرنامج على الدالة p التي تملك بارامترين (متغيرين) هما x و n . في داخل هذه الدالة نستعمل متغيرين محليين local هما k كدليل لدورة for والمتغير power الذي فيه نخزن قيمة x مضروبة في نفسها **n** من المرات.

شكل (4-4-5) : دالة حساب الأس .

```
PROGRAM power;

FUNCTION p(x:REAL; n:INTEGER):REAL;
VAR k:INTEGER;
    Power:REAL;
BEGIN
     power:=1;
     FOR k:=1 TO n DO
           Power:=power*x;
     p:=power;
END;
BEGIN { main }
      WRITELN;
      WRITELN(p(2.0, 3):10:3);
      WRITELN(p(1.5,4):10:3);
END.
```

لاحظ أن قيمة power يجب أن تبدأ بواحد ثم نضرب هذه القيمة في x ، ونكرر ذلك n من المرات .

5-5 الدوال الجاهزة

توفر لنا لغة باسكال الكثير من الدوال الجاهزة للاستعمال بمجرد استدعاء اسمها في البرنامج . ويبين الشكل (1-5-5) بعض الدوال الخاصة بصيغة تربو باسكال (Turbol Pascal). ولمعرفة المزيد عن الدوال المتوفرة بإمكان المبرمج الرجوع إلى دليل استعمال هذه اللغة ، أو استخدام help بالمفتاح F1 للحصول على قائمة الدوال ، كما في الشكل (3-5-5).

إلا أننا نتناول هنا بعض الأمثلة لاستخدام الدوال الجاهزة التي تصادف كثيراً في التطبيقات العلمية .

ويوضح البرنامج في الشكل (1-5-5) طريقة استخدام إحدى عشر دالة من دوال باسكال ، وهي :

شكل (1-5-5) : برنامج لحساب الدوال الجاهزة .

```
PROGRAM ReadyFuntions;
BEGIN
    WRITELN('abs(-5)=',ABS(-5));
    WRITELN('sin(pi/2)',SIN(PI/2));
    WRITELN('exp(1.0)=',EXP(1.0));
    WRITELN('ln(1.0)-',LN(1.)));
    WRITELN('frac(3.65)=',FRAC(3.65));
    WRITELN('int(4.58)=',INT(4.58);
    WRITELN('trunc(2.33)=',TRUNC(2.33));
    WRITELN('sqr(4.0)=',SQR(4.0));
    WRITELN('sqrt(4.0=',SQRT(4.0));
    WRITELN('arctan(1.0)=',ARCTAN(1.0));
    WRITELN('odd(7)=',ODD(7));
    WRITELN('round(4.54)=',ROUND(4.54));
    WRITELN('length('huda')=',length('huda');
     WRITELN('succ(45)=',succ(45));
END.
```

في هذا البرنامج استعملنا الدوال التالية:

1- دالة القيمة المطلقة ABS وهي اختصار لكلمة ABSOLUTE وتعطي القيمة الموجبة للعدد ، فمثلاً :

ABS (-5) = 5

2- دالة جيب الزاوية SIN ، ويلاحظ في هذه الدالة المثلثية أن الزاوية يجب أن تكون بالتقدير الدائري .

ومن نتائج البرنامج ، نرى مثلاً أن :

SIN (pi/2) = 1

علما بأن لغة باسكال تتعرف على الثابت pi أنه المعروف في الرياضيات بالمقدار الثابت 3.14159

3- دالة الأس الطبيعي EXP وهي الدالة الأسية المعروفة في الرياضيات بالرمز e^x . من نتائج البرنامج نرى أن :

EXP (1.0) = 2.718218285E + 00

4- دالة الجزء الكسري FRAC (اختصار لكلمةFRACTION) ، وهي تعطي قيمة الكسر العشري لأي عدد كسري مثل :

FRAC (6.45) = 0.45

5- دالة الجزء الصحيح INT (اختصار لكلمة INTEGER) ، وهي تعطي قيمة الجزء الصحيح من أي عدد كسري مثل :

INT (3.5) = 3.0

6- دالة الحذف TRUNC (اختصار لكلمة TRUNCATION) ، وهي تحذف الجزء الكسري وتعطي عدداً صحيحاً ، مثل :

TRUNC (1.9) = 1

7- دالة التربيع SQR ، وتقوم بتربيع العدد ، مثل :

SQR (3.0) = 9.0

8- دالة الجذر التربيعي SQRT ، وتقوم بإيجاد الجذر التربيعي لأي عدد كسري ، فمثلاً :

SQRT (4.0) = 2.0

9- دالة معكوس الظل ARCTAN ، وتعطي الزاوية بالتقدير الدائري ، فمثلاً(1.0) ARCTAN هي الزاوية التي ظلها يساوي 1 وهي تساوي pi/4 . أي:

ARCTAN (1.0) = 1.3258176637

10- دالة اختبار العدد الفردي ODD وتعطي قيمة منطقية TRUE في حالة العدد الفردي و FALSE في حالة العدد الزوجي ، أي إن :

ODD (5) = TRUE
ODD (6) = FALSE

11- دالة تقريب العدد الكسري إلى أقرب عدد صحيح ROUND ، أي إن:

ROUND (3.7) = 4

ROUND (2.4) = 2

لاحظ أن هذه الدوال الجاهزة من أنواع مختلفة ، منها الكسري وما هـــو رمزي أو أي نوع معرف مسبقاً .

5-6 تمرير البيانات من و إلى الإجراء

في البند (4-5) استعرضنا إمكانية تمرير بعض البيانات إلى الإجراء حتى يقوم بالوظيفة المطلوب منه . تسمى مثل هذه العملية بالاستدعاء بالقيمة call-by-value و فيها يكون للإجراء بارامترات تسمى بارامترات قيمة value parameters، و هي طريقة تعطي للإجراء استقلالية عن البرنامج الرئيسي ، وتجعله قابلاً للاستخدام في أغراض مختلفة . لذلك فإننا لا نستخدم متغيرات البرنامج الرئيسي (وتسمى بمتغيرات عامة global variables) في داخل الإجراء إذا أردنا أن يكون الإجراء صالحاً في برنامج رئيسي (أو إجراء) آخر .

إلا أن الاستدعاء بواسطة القيمة ، يمكننا من تمرير البيانات من البرنامج المستدعي إلى الإجراء فقط وليس العكس . وهو أمر عالجناه باستخدام الدالة، إذ إن الدالة لها بارمترات إدخال ، ونحصل مقابلها على إخراج واحد ، أي أن البيانات تدخل الدالة ، ونحصل مقابل ذلك على نتيجة معينة هي قيمة الدالة نفسها. ولكن هذه النتيجة لا تزيد عن قيمة واحدة إما عددية أو منطقية أو غيرها.

هناك طريقة أخرى تمكننا من تمرير البيانات من الإجراء إلى البرنامج المستدعي له ، وتسمى بالاستدعاء بالمرجع Call-by-reference . عندما يمرر متغير بالمرجع إلى الإجراء، يمرر عنوان (موقع) المتغير (بدلاً من قيمته) ، وبالتالي فإن أي تغيير يحدث على محتوى هذا الموقع داخل الدالة يقابله التغيير نفسه في البرنامج المستدعي .

ولتمييز هذه الطريقة عن الاستدعاء بالقيمة نضع كلمة VAR قبل اسم البارامتر المتغير (أو مجموعة من المتغيرات) ؛ حتى يمكننا ترجيع قيمته إلى البرنامج المستدعي . و المثال التالي يوضح هذه الطريقة .

مثال (1–6–5)

اكتب برنامجا فرعيا يقوم بحساب الضريبة (15 % من الدخل) و صافي المرتب بعد خصم الضريبة من الدخل . واستعمله في برنامج رئيسي .

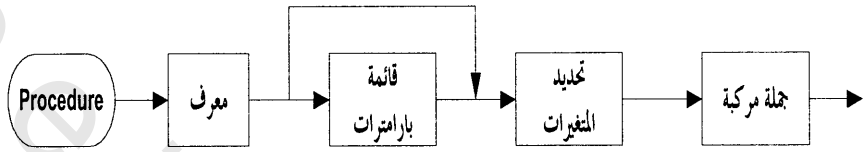
يبين الشكل (1-6-5) البرنامج المطلوب حيث تمت تسمية البرنامج الفرعي باسم ComputeTax أي احسب الضريبة. يستخدم هذا الإجراء بارامتر قيمة هو الدخل income و بارامترين متغيرين ، هما : incomeTax ضريبة الدخل، وnetIncome صافي المرتب .

شكل (1–6–5) : برنامج لتوضيح الاستدعاء بالمرجع .

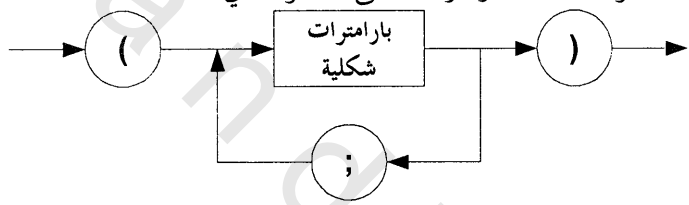
```
PROGRAM pay;
VAR sal, tax, net : REAL;
PROCEDURE ComputeTax (income:REAL;
                VAR   incomeTax,   netIncome
:REAL);
BEGIN
     IncomeTax := 0.15*income;
     NetIncome :=income - incomeTax;
END;

BEGIN   { main }
     WRITE('Enter salary-->);
     READ(sal);
     ComputeTax(sal,tax,net);
     WRITELN(sal:10:3,tax:10:3, net:10:3);
END.
```

أي إن الإجراء ComputeTax يستقبل قيمة income ، ويقـــوم بـــترجيع قيمتين هما الضريبة incomeTax ، والصافي netIncome. وهما متغيران يقابلـــهما في البرنامج الرئيسي التغيران tax و net على التوالي، و بالتالي يتكافئان في القيمة .

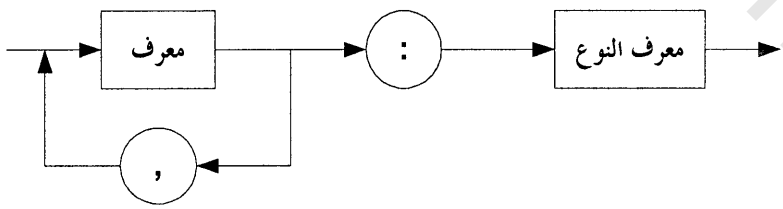
ويمكننا تلخيص تركيبة الإجراء على النحو التالي :



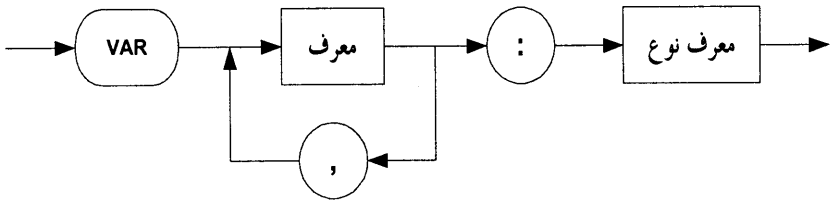
حيث تكون قائمة البارمترات على النحو التالي :



والبارمترات الشكلية إما أن تكون بارامترات قيمة على الشكل التالي :



أو أن تكون بارامترات متغيرة variable parameters على الشكل التالي :



7–5 الإجراءات الجاهزة

كما توجد دوال جاهزة للاستعمال في لغة باسكال ، توجد أيضاً إجراءات جاهزة ، الغرض منها توفير الجهد (البرمجي) على مستخدم هذه اللغة.

نذكر فيما يلي بعض هذه الإجراءات ، والتي يحتاج إليها كل مبرمج بصورة عامة . لمعرفة المزيد عن الإجراءات الجاهزة ، على الدارس مراجعة دليل اللغة ، أو الاستعانة بالمفتاح F1 (help) للحصول على قوائم كما في الملحق ، علماً بأن الإجراءات المذكورة بهذه القوائم تخص صيغة تربو باسكال 6 وقد تختلف إلى حد ما عن الصيغ الأخرى .

1– الإدخال و الإخراج

لقد استخدمنا خلال الفصول السابقة الجمل الخاصة بعمليات الإدخال والإخراج ، ونشير هنا إلى أن هذه الجمل ما هي إلا استدعاء لإجراءات جاهزة، وتحمل الأسماء :

READ
READLN
WRITE
WRITELN

حيث نستخدم الإجراءين READ و READLN للقراءة و WRITE و WRITELN للكتابة علما بأن الحرفين LN في نهاية كلمة READLN وكلمة WRITELN هما اختصار لكلمة LINE ، و الإجراء WRITELN يقوم باستدعاء الإجراء WRITE مع وضع علامة لنهاية السطر ، أما الإجراء READLN فيقوم باستخدام الإجراء READ مع التحول إلى السطر الموالي من الملف بعد قراءة المتغيرات .

ويوضح البرنامج المبين بالشكل (1-7-5) كيفية استخدام هذه الإجراءات ، وذلك بقراءة 3 قيم عددية صحيحة هي (a1, a2, a3,) يتم إدخالها من لوحة المفاتيح (Keyboard) ؛ نظراً لعدم تحديد الملف المقروء منه. وتخزن هذه القيم في ملف نصي اسمه (t.dta) بواسطة الاستدعاء .

WRITELN (t, a1, a2, a3);

شكل (1-7-5) : برنامج لتوضيح إجراءات الإدخال و الإخراج .

```
PROGRAM inOut;
VAR a1,a2,a3: REAL;
    z1, z2, z3 : REAL;
    tests : text;
BEGIN
     ASSIGN(tests, 'tests.dta');
     REWRITE(tests);
     READLN(a1,a2,a3);
     WRITELN(tests, a1, a2, a3);
     CLOSE(tests);
     RESET(tests);
     READLN(tests, z1, z2, z3);
     WRITELN(z1, z2, z3);
     CLOSE(tests);
END.
```

ثم تقرأ هذه البيانات في المتغيرات b1, b2, b3 من هذا الملف ، وتطبـــــع على الشاشة . لاحظ أن عدم تحديد الملف في الاستدعاء :

WRITELN (b1, b2, b3)

يعني الطباعة على الشاشة .

2-اجراءات لمعالجة الملفات

هناك عدد من الإجراءات الجاهزة و الخاصة بمعالجـــة الملفـــات في تربـــو باسكال ، نذكر منها ما يلي :

Append	لإضافة سجل في ملف
Assign	لتعيين اسم للملف
Close	لقفل الملف
Erase	لإلغاء الملف
Reset	لفتح الملف للقراءة منه
Rewrite	لفتح الملف للكتابة فيه
Rename	لإعادة تسمية الملف
Seek	للبحث عن سجل معين

وقد سبق وأن استخدمنا Rewrite , Reset , Close , Assign في بند الملفات.

أما الإجراء Append فيستخدم لإضافة سجل في ملف من النوع النصـي (text) على الشكل :

APPEND (t);

حيث t ترمز للملف المطلوب الإضافة إليه .

3 – إجراءات لعمليات الرسم Graphics

من مزايا تربو باسكال توفر إجراءات جاهزة به تساعد في عملية الرسـم Graphics مستغلة في ذلك توفر الشاشات ذات الألوان بالأجهزة الشـخصية.

ونذكر من هذه الإجراءات :

ClrScr	لمسح ما هو موجود على الشاشة
Line	لرسم خط مستقيم بين نقطتين
Rectangle	لرسم مستطيل
SetColor	لتحديد لون الرسم
Circle	لرسم دائرة
Bar	لرسم أعمدة بيانية
SetBKColor	لتحديد لون خلفية الشاشة عند الرسم
TextColor	لتحديد لون الكتابة
TextBackground	لتحديد لون خلفية الشاشة عند الكتابة

Window	لتحديد نافذة على الشاشة للكتابة
Arc	لرسم جزء من دائرة

8–5 استدعاء البرنامج الفرعي لنفسه

إذا أردنا كتابة برنامج لحساب مضروب العدد n (ويرمز له عادةً بالرمز n!) من الصيغة :

$$n! = n(n - 1)(n - 2) \ldots (2)(1)$$

فبإمكاننا تحويل هذه الصيغة أولاً إلى العلاقات التالية :

$$n! = n(n - 1)!$$

و إذا استخدمنا التعريف

$$n! = \text{factorial } (n)$$

حيث كلمة factorial تعني المضروب ، فإن :

$$\text{factorial } (n) = n \text{ factorial } (n - 1)$$

لكتابة برنامج لحساب مضروب عدد صحيح ، لدينا اختياران : الأول نسميه بالأسلوب التكراري iterative ، والثاني بالأسلوب التتابعي recursive . والفرق بينهما أن في الاختيار الثاني يتم استدعاء الدالة لنفسها ، بينما في الاختبار الأول يتم الاستدعاء من خارج البرنامج الفرعي للدالة .

يبين الشكل (1–8–5) برنامجًا فرعيًّا للدالة factorial التي تقوم بحساب مضروب أي عدد صحيح n بالطريقة التكرارية المعتادة ، وذلك باستخدام جملة WHILE ، أي أن الحلقة تتم داخل الدالة نفسها .

شكل (1–8–5) : دالة مضروب العدد بالطريقة التكرارية .

```
PROGRAM iterative;
FUNCTION   factorial(n  :  INTEGER)  :
INTEGER;
VAR i, f : INTEGER;
BEGIN
    f := 1;
    i := 1;
    WHILE i<n  DO
    BEGIN
         i := i + 1;
         f := f * i;
    END;
    factorial := f;
END;  { function }

BEGIN {main }
      WRITELN(factorial(5));
END.
```

أما في الشكل (2-8-5) فنجد البرنامج الفرعي للدالة factorial التي تحسب أيضاً مضروب العدد n ، ولكن بطريقة مختلفة إذ لا يوجد أي حلقة داخل الدالة بل مجرد جملة تعريف تتابعي للدالة نفسها.

فإذا كان البرنامج الرئيسي لهذه الدالة على النحو التالي :

```
PROGRAM RECURSIV;
BEGIN
      WRITELN (factorial (5));
END.
```

فإن القيمة 5 تتعين للمتغير n في الدالة factorial ، و أول جملة تنفيذ في هذه الدالة هي :

factorial = 5 * factorial (4)

شكل (2–8–5): دالة مضروب العدد بالطريقة التتابعية (recursive) .

```
PROGRAM recursive;
VAR n: INTEGER;
FUNCTION    factorial(    n:    INTEGER)    :
INTEGER;
BEGIN
     IF n=0 THEN factorial:=1
     ELSE
     factorial := n*factorial(n-1);
END;

BEGIN
     WRITELN(factorial(5));
END.
```

وهي تحتوي على استدعاء للدالة مرة أخرى ، ولكن بقيمة 4 للمتغير n، وهكذا إلى أن نصل إلى قيمة 0 للمتغير n ، وعندها نتوقف عن استدعاء الدالة مع تعيين قيمة 1 للدالة. وترجع قيمة المضروب (1 * 2 * 3 * 4 * 5) أي (120) للبرنامج الرئيسي. نلاحظ أن الأسلوب التتابعي هو أفضل من حيث الوضوح وسهولة الاستخدام ، إلا أن بعض المبرمجين يفضلون الطريقة المعتادة ، ويعتبرونها أكثر وضوحاً . ولكن هناك بعض المسائل في البرمجة تحتاج إلى الأسلوب التتابعي أكثر من غيرها .

مثال (1–8–5) :

المطلوب كتابة برنامج لقراءة عدد من الكلمات تكون سطراً من 80 خانة أو أقل ، بحيث يقوم البرنامج بطباعة هذا السطر معكوساً .

الغرض من هذا البرنامج هو التطبيق في مجال اللغة العربية و استخدامها في الحاسوب ؛ حيث تظهر الكلمات على الشاشة من اليمين (أي ابتداء من الخانة 80 في الكتابة اللاتينية) إلى اليسار . فمثلاً إذا كان الإدخال مثلاً :

abc defg hijk

يطبع البرنامج ما يلي :

kjih gfed cba

ويوضح الشكل (3-8-5) البرنامج المطلوب في هذا المثال .

عند تنفيذ هذا البرنامج تظهر العبارة

type a line

أي اطبع سطراً ، و المقصود بالسطر هنا هو سلسلة الحروف (نضيد) تنتهي بالضغط على المفتاح enter . و من المهم هنا أن نتتبع هذا البرنامج حتى تتضح لنا بعض الخواص.

نلاحظ أولاً في البداية تحديد المتغير letter بأنه مصفوفة متكونة من 80 حرفاً. وهذه المصفوفة هي التي تتم قراءتها من العنصر الأول إلى الأخير ، وتتم طباعتها عكسياً ، أي من الأخير إلى الأول . لاحظ أن السطر على الشاشة يحتوي عادة على 80 خانة ، ومن ذلك جاء تحديد بعد المصفوفة .

ويقوم بعملية الانعكاس الإجراء inv بطريقة تتابعية (recursively) ابتداء من القيمة 80 للمتغير count ، وتنازليًّا حتى الوصول إلى قيمة الصفر .

أما وظيفة البرنامج الرئيسي هنا فهي قراءة النضيد letter ، ونستعمل لهذا الغرض دالة نهاية السطر EOLN (وهي اختصار للكلمات End Of Line أي نهاية السطر) . وهي إحدى الدوال الجاهزة ، وتكون إما True أو False.

شكل (3–8–5) : برنامج لقلب المخلات بالطريقة التتابعية .

```
PROGRAM invert;
VAR i, j, L :integer;
    letter : ARRAY[1..80]OF CHAR;
PROCEDURE inv( count : INTEGER);
BEGIN
     IF count=0 THEN WRTELN
     ELSE
     BEGIN
          WRITE(letter[count]);
          inv(count-1);
     END;
END;

BEGIN
     WRITELN('Type a line');
     I:=0;
     REPEAT
           i:=i+1;
           READ(letter[i]);
     UNTIL
          EOLN OR (I=80);
     FOR j:= i+1 TO 80 DO
         letter[j]:=' ';
     L:=80;
     inv(L);
END.
```

لاحظ في البرنامج ملء بقية الخانات بالفراغ عندما ينتهي السـطر قبل الخانة 80.

من الأمثلة التي سقناها لتوضيح عملية استدعاء الدالة لنفسها نلاحظ ما يلي :

1- البرنامج الفرعي التتابعي (سواء كان دالة أو اجراء) يجب أن ينفذ مـرة واحدة على الأقل دون استدعاء نفسه .

في البرنامج (2-8-5) نلاحظ أن الحالة n = 0 هي التي لا تستدعي فيها الدالة نفسها وتتسبب في انتهاء الاستدعاء الذاتي . وفي البرنامج (3-8-5) نرى أن الحالة count = 0 تقوم بنفس العمل .

2- يجب أن يستدعي البرنامج الفرعي التتابعي نفسه بطريقة تقربه في كل مرة من الحالة المذكورة في الملاحظة الأولى ، أي الحالة التي تجعله يتوقف عن عملية الاستدعاء . وإذا لم يتوفر هذا الشرط ، فإن التنفيذ لن يتوقف.

9-5 تداخل البرامج الفرعية

استعرضنا كيف نستخدم الإجراءات و الدوال لتجزئة البرنامج إلى وحدات تقوم كل واحدة بوظيفة معينة ، وذلك لغرض تسهيل كتابة أو قراءة البرنامج. وقد احتوت الأمثلة حتى الآن على برامج تحتوي على إجراء واحد أو دالة واحدة ، ولكن من الجائز أن تتعدد الإجراءات والدوال في برنامج واحد ، بل يعتبر هذا هو الوضع الشائع .

يبين الشكل (1-9-5) مثالاً لما يمكن أن يكون عليه الهيكل العام لبرنامج . و يحتوي هذا البرنامج على 3 إجراءات ودالة ، على النحو التالي :

1- الإجراء initial إجراء مستقل ، أي غير متداخل مع إجراء آخر .

2- الإجراء rooms ويحتوي على الإجراء persons ، وهذا الإجراء يحتوي بدوره على الدالة last .

وبشكل عام فإن الإجراء والدوال في لغة باسكال يمكن أن تكون مستقلة أو يحتوي بعضها على الآخر (Nested) .

شكل (1–9–5) : مثال لتداخل البرامج الفرعية .

```
PROGRAM Hotel;
TYPE ...
VAR ....
PROCEDURE iniate;
BEGIN
.....
END;
PROCEDURE rooms;
BEGIN
          PROCEDURE persons;
          VAR ...;
          BEGIN
              FUNCTION last;
              BEGIN  {last}
              ...
              END;  {last}
          ...
          END; {persons}
          ...
END; {rooms}
BEGIN  {main}
...
END.
```

5–10 مثال تطبيقي (ملفات الموظفين)

نقوم هنا بكتابة برنامج لمعالجة ملفات الموظفين بمؤسسة ما ، علماً بـــأن هذا البرنامج ليس واقعياً ، بل هو مثال يقصد توضيح أهمية تقسيم البرنـــامج إلى

مجموعة إجراءات ، يقوم كل إجراء بوظيفة معينة ، مما يجعل كتابة البرنامج وقراءته عملاً مبسطاً .

ولذلك نفترض أن ملف الموظفين (employfile) ، يحتوي على مجموعة من السجلات (employ) ، كل سجل يتكون من ثلاثة مجالات فقط (في الواقع يكون أكثر تعقيداً من هذا الافتراض) :

1- رقم الموظف = ID (عدد صحيح)
2- اسمه = NM (لا يزيد عن 20 حرف)
3- المرتب = BS (كسري)

و المطلوب من البرنامج إمكانية أداء ما يلي :

1- تكوبن الملف الأصلي .
2- عرض الملف كله على الشاشة .
3- عرض سجل موظف معين من رقمه .
4- إضافة سجل موظف للملف في نهايته .
5- إلغاء سجل موظف من الملف

ويبين البرنامج بالشكل (1-10-5) كيف نقوم بهذا العمل باستخدام مجموعة من الإجراءات ، هي :

1- الإجراء menu لعرض الخدمات التي يمكن أن يقوم بها البرنامج .
2- الإجراء createmp الذي يقوم بتكوين الملف employeeFile بعدد ne من السجلات .

3- الإجراء ListOne الذي يقوم بعرض سجل موظف واحد .

4- الإجراء ListAll الذي يقوم بعرض الملف كله .

5- الإجراء fcopy الذي يقوم بنسخ الملف oldf في الملف newf . لاحظ أن هذين البارامترين شكليان (formal parameters) ، وقد قمنا بكتابة هذا الإجراء لحاجتنا إليه في إجراءات أخري .

6- الإجراء AddRecord لإضافة سجل في الملف عند نهايته .

7- الإجراء DelRecord لإلغاء سجل من الملف .

ملاحظات حول البرنامج :

1- الجملة (uses crt ;) تستخدم عندما نستعمل أحد الإجراءات الجاهزة في الوحدة crt مثل الإجراء clrscr لمسح الشاشة .

2- الإجراء fcopy له بارامتران متغيران من النوع filetype . ومن الضروري في تربو باسكال أن تكون البارمترات التي تصف ملفات من هذا النوع متغيرة ، أي يجب وضع VAR قبل سردها .

3- الإجراء addRecord لإضافة سجل للملف ، يتبع الخطوات التالية :

- نسخ الملف الحالي في ملف جديد .
- كتابة السجل الجديد في الملف الجديد قبل قفله .
- نسخ الملف الجديد في الملف الحالي .
- إلغاء الملف الجديد .

4- الإجراء delRecord لإلغاء سجل موظف من الملف يقوم بنسخ سجلات الملف الأصلي (employeeFile) باستثناء السجل الذي يحمل رقم الموظف

المطلوب إلغاؤه لنحصل على ملف جديد (newfile) . وقبل نهاية الإجـــراء ، نجعل الملف الجديد يحل محل الملف القديم لإتمام عملية التعديـل update ؛ أي إن الملف newfile هو ملف مؤقت .

لاحظ هنا أنه بالإمكان إلغاء سجل بطرق أخـــرى ، كـأن نســتخدم المصفوفات بدلاً من استخدام ملف آخر كوسيط ، و لكن الغرض من الطريقـــة المتبعة في البرنامج التدرب على معالجة الملفات . (انظر تمرين 18) .

5- إذا أدخلنا رقماً للإختيار choice غير الأرقام من 1 إلى 5 فإن البرنامج ينتهي بدون عمل أي شيء .

6- إذا أدخلنا الرقم 1 في الاختيار فذلك يعني تكوين ملف جديد ، لذلـــك يجب أخذ الحذر من خطأ ضياع الملف الأصلي إذا كان موجـــوداً ، لأن الإجراء rewrite سوف يلغي هذا الملف .

شكل (1-10-5) : برنامج معالجة ملف الموظفين في مؤسسة .

```
PROGRAM EmployeesFile;
USES CRT;
TYPE employee = RECORD
     Id : INTEGER;
     Nm : STRING[20];
     Ba : REAL;
     END;
     FileType = FILE OF employee;
VAR employeeFile, newFile : fileType;
    Emp : employee;
    ne, choice, n , i : INTEGER;
```

```
PROCEDURE menu;
BEGIN
     CLRSCR;
     GOTOXY(30,3);
     WRITE('EMPLOYEES DATA PROCESSING');
     GOTOXY(30,5);
     WRITE('1--Create a new employees file.');
     GOTOXY(30,6);
     WRITE('2--List all records.');
     GOTOXY(30,7);
     WRITE('3--List a particular record.') ;
     GOTOXY(30,8);
     WRITE('4--Add a new record.');
     GOTOXY(30,9);
     WRITE('5--Delete a record.');
     GOTOXY(30,10);
     WRITE('6--Exit.');
END;

PROCEDURE creatEmp;
BEGIN
     CLRSCR;
     GOTOXY(20,2);
     WRITE('Enter number of Employees-->');
     READ(ne);
     REWRITE(employeeFile);
     FOR i:= 1 TO ne DO
     BEGIN
          CLRSCR;
          GOTOXY(20,5);
          WRITE('Enter employee id--> ');
          READLN(emp.id);
          GOTOXY(20,6);
          WRITE('Enter employee name-->');
          READLN(emp.nm);
          GOTOXY(20,7);
          WRITE('Enter employee basic salary--
 >');
          READLN(emp.ba);
          WRITE(employeeFile, emp);
```

```
     END;
END;  {creatEmp}
PROCEDURE ListALL;
VAR row : INTEGER;
BEGIN
     CLRSCR;
     Gotoxy(30,1);
     WRITE('List Of All Employees');
     Gotoxy(20,3);
     WRITE('number':10,'name   ':15,'    basic
salary':20);
     RESET(employeeFile);
     Row :=4;
    WHILE NOT EOF(employeeFile) DO
     BEGIN
          READ(employeeFile, emp);
          row := row +1;
          GOTOXY(20,row);
          WRITE(emp.id:10,          emp.nm:20,
emp.ba:15:3);
     END;
     CLOSE(employeeFile);
END; {ListAll}

PROCEDURE ListOne;
VAR empNum , flag: INTEGER;
BEGIN
     CLRSCR;
     GOTOXY(30,10);
     WRITE('Enter id of Employee-->');
     READ(empNum);
     RESET(employeeFile);
     Flag:=0;
     WHILE NOT EOF(employeeFile) DO
     BEGIN
          READ(employeeFile, emp);
          IF(emp.id=empNum) THEN
          BEGIN
            GOTOXY(30,11);
            WRITELN('Employee Name:',emp.nm);
```

```
                GOTOXY(30,12);
                WRITELN('Employee basic salary'
                                ,emp.ba:10:3);
                flag:=1;
              END;
      END;
       IF flag=0 THEN WRITELN('No such number');
END;        {ListOne}

PROCEDURE fcopy( VAR oldFile, newFile :
                                        FileType);
BEGIN
      REWRITE(newFile);
      RESET(oldFile);
      WHILE NOT EOF(oldFile) DO
      BEGIN
           READ(oldFile, emp);
           WRITE(newFile, emp);
      END;
      CLOSE(oldFile);
      CLOSE(newFile);
END;

PROCEDURE AddRecord;
BEGIN
      RESET(employeeFile);
      WHILE NOT EOF(employeeFile) DO
            READ(employeeFile, emp);
      CLRSCR;
      GOTOXY(30,10);
      WRITE('Enter id-->'); READLN(emp.id);
      GOTOXY(30,11);
      WRITE('Enter name-->'); READLN(emp.nm);
      GOTOXY(30,13);
      WRITE('Entersalary-->'); READLN(emp.ba);
      WRITE(employeeFile, emp);
      CLOSE(employeeFile);
END;
PROCEDURE DelRecord;
VAR empNum : INTEGER;
```

```
BEGIN
      CLRSCR;
      WRITE('Enter   id   of   employee   to   be
deleted-->');
      READ(empNum);
      RESET(employeeFile);
      REWRITE(newFile);
      WHILE NOT EOF(employeeFile) DO
      BEGIN
            READ(employeeFile, emp);
            IF      emp.id<>     empNum      THEN
WRITE(newfile,emp);
      END;
      CLOSE(employeeFile);
      CLOSE(newFile);
      Fcopy(newfile, employeeFile);
      ERASE(newfile);
END;


BEGIN {main}
      ASSIGN(employeeFile,'emp.dta');
      ASSIGN(newFile, 'new.dta');
      Menu;
      GOTOXY(30,15);
      WRITE('Enter Your choice-->');
      READ(choice);
      IF choice=1 THEN createEmp;
      IF choice=2 THEN ListAll;
      IF choice=3 THEN ListOne;
      IF choice=4 THEN AddRecord;
                        IF choice=5 THEN DelRecord;
END.
```

11–5 تمارين

1- اكتب برنامجاً فرعيًّا للإجراء الذي يقوم بطباعة الأسطر التالية :

1- LIST OF STUDENTS
2- LIST OF COURSES
3- EXIT

2- اكتب برنامجاً فرعيًّا لقراءة تاريخ اليوم (أي اليوم والشهر والسنة) .
واستخدم هذا الإجراء في برنامج رئيسي لطباعة التاريخ .

3- اكتب برنامجاً فرعيًّا لقراءة مصفوفة أحادية البعد (يتم تحديد نوعها في البرنامج الرئيسي) تتكون من 50 عنصراً من النوع العددي الصحيح .

4- اكتب برنامج فرعيًّا لترتيب مصفوفة عددية ترتيباً تنازليًّا ، و برنامجاً فرعيًّا آخر لطباعة المصفوفة بعد الترتيب ، استخدم البرنامج الفرعي في التمرين (3) مع هذين الإجراءين في كتابة برنامج كامل لترتيب مصفوفة تنازلياً .

5- اكتب الإجراء :

PROCEDURE printx (n: INTEGER)

الذي يقوم بطباعة الحرف **X** أفقيًّا عدد **n** من المرات .

6- اكتب الإجراء

PROCEDURE blanks (n: INTEGER)

الذي يقوم بترك n فراغ من بداية السطر ، أي إنه يجعل المؤشر (cursor) عند العمود (n + 1) على الشاشة .

7- اكتب برنامجاً فرعيًّا للدالة :

$P(x) = 5x^2 + 2x - 1/x$

مع برنامج رئيسي لحساب هذه الدالة لجميع قيم x من 1 إلى 5 بخطـــوة 0.5 .

8- اكتب برنامجاً فرعيًّا و برنامجاً رئيسيًّا لحساب الدالة

$density = 0.051 * pressure * (temp)^2$

وذلك لجميع قيم pressure من 100 إلى 500 بخطوة 5 ، و جميع قيم temp مـــن 200 إلى 400 بخطوة 10 .

9- اكتب برنامجاً فرعيًّا للدالة المنطقية (البولية)

test (x, y)

التي تكون قيمتها TRUE إذا كانت x أكبر من أو تساوي y ، والقيمـــة FALSE إذا كانت x أصغر من y ، حيث x و y متغيران كسريان .

10- اكتب قيم الدوال و الثوابت التالية عند النقاط المحددة :

ABS (- 10)	SIN (0.0)
FRAC (4.7)	ROUND (6.8)
SQRT (9.0)	PI
SQR (9.0)	COS (0.0)
ODD (14)	EXP (0.0)

11- لحساب الدالة اللوغارتمية $LOG_e(x)$ للأساس **e** بإمكاننـــا اســـتخدام المتسلسلة :

$$\log_e = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots$$

اكتب برنامجاً فرعيًّا لهذه الدالة ، و استخدمه في **حساب** log(2) . **استخدم n** من الحدود في المتسلسلة بحيث :

$$10^{-8} \geq \frac{x^n}{n}$$

12 – نظراً لتوفر الدالة الأسية **EXP** ، فبإمكاننا كتابة برنامج فرعي للدالة :

$$f(x, y) = x^y$$

باستغلال العلاقة

$$x^y = e^{y \log_e x}$$

والدالة $\log_e(x)$ المعرفة في التمرين (11) . اكتب برنامجاً لحساب وطباعة $(0.52)^{0.38}$ بهذه الطريقة .

13– اكتب برنامج فرعيًّا للإجراء الذي يقوم بحل المعادلتين

$$a_1x + b_1y = c_1$$

$$a_2x + b_2y = c_2$$

حيث المدخلات هي $c_1, c_2, b_2, b_1, a_2, a_1$.

14– اكتب برنامجاً فرعياً لإجراء الذي يقوم بقراءة نضيد من الرموز (string) ، تنتهي بالضغط على المفتاح (enter) و يقوم بحساب عدد الفراغات في هذه السلسلة و عدد الفصول (,) .

15- استخدم الأسلوب التتابعي في حساب الدالة

$$f(n) = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \ldots \pm \frac{1}{n}$$

16- استخدم الأسلوب التتابعي في كتابة برنامج لطباعة جميع الكلمات الـــتي يمكن تكوينها من 4 حروف معطيات ، علماً بأن الكلمة ليست بالضرورة ذات معنى لغوي ، و يجب ألا تتكرر أكثر من مرة واحدة .

17- ماذا يتم إخراجه بواسطة البرنامج التالي ؟ و ماذا يحدث إذا ألغينا كلمة VAR في الجملة :

PROCEDURE sq (VAR t: REAL);

هل يبقي الإخراج كما هو ؟

```
PROGRAM VARPAR;
VAR x: REAL;

PROCEDURE sq (VAR t:
REAL );
BEGIN
T: = t * t
END ;

BEGIN
X: = 2.0;
Sq (x)
WRITELN (x: 3: 1)
END.
```

18- قم بتعديل البرنامج بالشكل (1-10-5) وذلك :

أ . بتعديل الإجراء addRecord لإضافة سجل للملف الأصلي .

ب. بتعديل الإجراء delRecord لإلغاء سجل من الملف الأصلي .

حيث تستخدم المصفوفات بدلاً من استعمال ملف آخر وسيط .

* * *