

الباب السادس

مواضيع تكميلية

obeykandl.com

obeikandi.com

1-6 مقدمة

تعتبر المادة التي عرضت في الفصول السابقة أساساً كافياً لتمكين الدارس من كتابة برنامج بلغة باسكال لأي من التطبيقات الإدارية والرياضية ، ولكن لازال في هذه اللغة الكثير من التركيبات التي توفر على المبرمج الوقت والجهد، وتجعل من البرنامج أكثر كفاءة ووضوحاً . وهناك مواضيع لن نتطرق لها بالكامل مثل موضوع الرسم graphics .

في هذا الباب نحاول عرض بعض المواضيع التي لم نتطرق إليها حتى الآن، وهذا يدل على إمكانية كتابة برامج عملية بدونها .

2-6 جملة (في حالة أن) CASE OF

هذه الجملة تشبه إلى حد كبير جملة IF ، من حيث التركيب والغرض،

وصورتها العامة هي :

CASE x OF

i1 : statement 1;

i2 : statement 2;

.....

END ;

وتعني إذا كانت x تساوي i1 أنجز (statement 1) ،

وإذا كانت x تساوي i2 أنجز (statement 2) .

وهكذا ... حتى نصل إلى نهاية جملة CASE بكلمة END ، لاحظ أن x ترمز إلى متغير صحيح أو رمزي أو مسرود و يسمى دليل CASE ، أما الثوابت i_1 ، i_2 ، ... ، فتسمى ثوابت CASE .

مثال (6-2-1)

في الجملة التالية ، يتم حساب z بناء على قيمة المتغير $select$ وهو من النوع CHAR :

CASE select OF

'P': $z = x + y$;

'M': $z = x - y$;

'T': $z = x * y$;

END;

فإذا كانت $select$ تساوي 'P' تتعين العبارة $x + y$ للمتغير z ، وإذا كانت تساوي 'M' تتعين العبارة $x - y$ للمتغير z ، وإذا كانت تساوي 'T' تتعين العبارة $x * y$ للمتغير z . لاحظ أن $select$ يجب أن تكون في هذا المثال من النوع CHAR .

وكمثال آخر على استعمال جملة **case of** ، انظر البرنامج الرئيسي في الشكل (6-2-1) . لاحظ في هذا البرنامج أنه يجوز أن يكون دليل CASE عبارة حسابية أو منطقية .

شكل (6-2-1) : مثال على استعمال جملة CASE .

```
PROGRAM caseExample;
TYPE range= 1..10;
VAR i,j : INTEGER;
    a: ARRAY[range, range]OF
INTEGER;
BEGIN
WRITE('Enter 2 integers-->');
READ(I,j);
a[i,j]:=2;
CASE I+j OF
    5 : a[i,j]:=1;
    6 : a[i,j]:=0;
    7 : a[i,j]:=-1;
END;
WRITELN(a[i,j]);
CASE I>j OF
TRUE : WRITELN('true');
FALSE : WRITELN('false');
    END;
END.
```

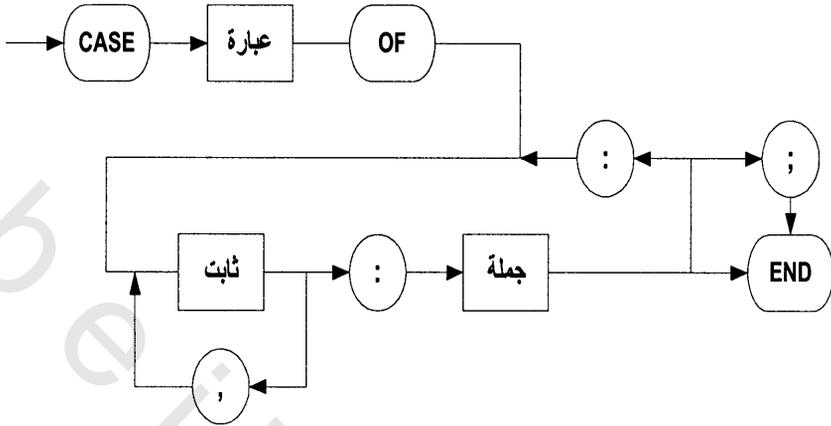
يوضح الشكل (6-2-2) التركيب اللغوي لجملة CASE . ونلاحظ

إمكانية وجود عدة ثوابت تفصل بينها الفاصلة (,) وذلك بعد OF .

مثل :

```
CASE Grade OF
'A', 'B', 'C', 'D': WRITELN ( 'PASS');
'F', 'W': WRITELN ('FAIL')
END;
```

شكل (6-2-2) : التركيب اللغوي لجملة CASE .



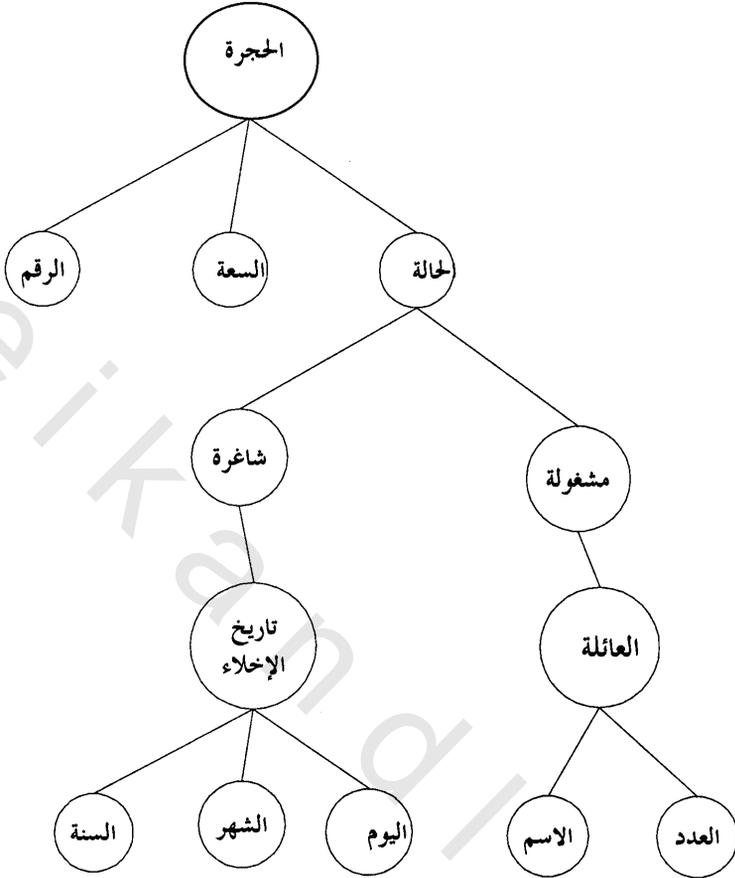
6-3 السجلات ذات المجال المتغير

ليس ضرورياً أن يكون السجل في لغة باسكال ذا مجالات ثابتة ، إذ توفر لنا هذه اللغة إمكانية تحديد المجالات حسب الحالة التي تواجهها . وهذا أمر مفيد جداً في التطبيقات الإدارية . فإذا أخذنا مثلاً السجل الخاص بمعلومات عن الحجرات في فندق ، كما هو مبين بالشكل (6-3-1) ، نجد أن هذه البيانات تعتمد على ما إذا كانت الحجرة شاغرة أم مشغولة ، فإذا كانت الحجرة شاغرة فقد نحتاج إلى تاريخ آخر إخلاء ، وإذا كانت مشغولة فقد نحتاج اسم العائلة أو الأفراد الذين يشغلونها وعددهم .

أي قد يكون سجل الحجرة على النحو التالي :

105 2 F ALI 1

شكل(1-3-6) : سجل حجرة في فندق .



بمعنى أن رقم الحجرة (room. number) هو 105 ، وأن سعتها (room. capacity) و هي 2 وأنها مشغولة 'filled' أو باختصار F ، وأن اسم شاغلها هو 'ALI' بعدد واحد .

في هذه الحالة تكون متغيرات السجل هي:

room. number	رقم الحجرة
room. capacity	سعة الحجرة
room. status	وضع الحجرة
room.family.Name	اسم الشاغل
room.family.Size	عدد العائلة بالحجرة

أما في حالة أن الحجرة شاغرة ، فقد يكون السجل على النحو التالي :

215 3 'v' 25 9 1999

بمعني أن رقم الحجرة هو 215 وسعتها 3 وهي شاغرة (حيث ' v ' اختصار لكلمة vacant) منذ يوم 25 شهر 9 سنة 1999 ؛ أي إن مكونات السجل في هذه الحالة هي :

room. Number	رقم الحجرة
room. capacity	سعة الحجرة
room. status	وضع الحجرة
room. date. day	اليوم (من تاريخ آخر إخلاء)
room. date. month	الشهر
room. date. year	السنة

في لغة باسكال ، يمكن تحديد هذا النوع من السجلات (ويسمى بالسجل ذي المجال المتغير variant field record) على النحو التالي :

```

TYPE room = RECORD
    Number: INTEGER;
    Capacity: INTEGER;
    Status: CHAR;
CASE st : CHAR OF
    'v': ( d: date );
    'f': ( fam: family );
END;

```

حيث تم استخدام جملة CASE متبوعة باسم متغير st من النوع الرمزي وهو يعبر عن وضع الحجره ، فإذا تم إدخال القيمة 'v' فإن المجال الذي يلي status هو d من النوع date ، أما إذا أدخلت القيمة 'f' ، فإن هذا المجال هو fam من نوع family .

لاحظ أن السجل room يحتوي على سجل آخر كأحد مكوناته ، إذ إن date هو سجل يحتوي على اليوم والشهر والسنة على النحو التالي :

```

TYPE date = RECORD
    day: 1 .. 31;
    month: 1 .. 12
    year: INTEGER;
END;

```

كما أن family هو سجل يحتوي على الاسم و العدد ، أي :

```

TYPE family = RECORD
    Name: STRING [20];
    Size: INTEGER;
END;

```

ويبين الشكل (1-5-6) الذي يحتوي على برنامج إدارة فندق ، استخدام السجل room الذي (كما ذكرنا) تعتمد مكوناته على المتغير st ، وهذه المكونات في هذا البرنامج هي من النوع (سجل) أيضاً .

6-4 جملة WITH-DO

عند الإشارة إلى مكونات متغير من نوع السجل rec ، لاحظنا ضرورة استخدام النقطة على النحو التالي :

rec. field1

للدلالة على المتغير في المجال field1 من السجل rec ، كما هو موضح بالجزء التالي :

```
TYPE rec = RECORD
  Field1: INTEGER;
  Field2: REAL;
  Field3: REAL;
END;
VAR r: rec;
BEGIN
```

```
  READLN (r.field1, r.Field2, r.field3);
```

ولتوفير كتابة اسم السجل في كل مرة نشير فيها إلى مكوناته ، بإمكاننا استخدام العبارة :

```
  WITH r DO
```

حيث r ترمز إلى اسم السجل . فبدلاً من الجملة :

```
  READLN (r. field1. r. Field2. r. field3);
```

نكتب :

```
  WITH r DO
```

```
    READLN ( field1, field2, field3);
```

وإذا كانت لدينا مجموعة من الجمل ، تتم فيها الإشارة إلى مكونات

السجل ، وليكن اسمه name ، نستخدم WITH على النحو التالي :

```
  WITH name DO
  BEGIN
```

... مجموعة جمل ...

END;

وكمثال على استخدام هذه الجملة انظر البرنامج بالشكل (6-5-1) .

6-5 برنامج إدارة فندق

كتطبيق لبعض المواضيع المتقدمة ، نقدم هنا برنامجاً يقوم بمساعدة إدارة الفندق في متابعة وضع الحجرات والتزلاء . في هذا البرنامج نجد مثلاً للسجل ذي المجال المتغير ، وهو السجل ROOM ، كما سبق وأن أشرنا إليه . كما نجد مثلاً لاستخدام جملة (WITH-DO) عند التعامل مع مكونات المتغير . rm

رغم أن البرنامج المذكور تنقصه الكثير من الأمور العملية التي تهتم إدارة الفندق ، كالناحية المالية ، وما يتعلق بها من حيث مدة الإقامة وغيرها ... ، إلا أنه يقوم بالأعمال الأساسية في إدارة الحجرات التي تعتمد على معرفة أي الحجرات مشغولة وأيها شاغرة ، و أسماء التزلاء بكل حجرة ، وعملية الحجز وإلغائه .

هذه الأعمال نضعها في شاشة العرض على النحو :

1- create room file	تكوين ملف الحجرات
2- list vacant rooms	قائمة الحجرات الشاغرة
3-list filled rooms	قائمة الحجرات المشغولة
4- check in	استقبال نزيل
5- check out	مغادرة نزيل
6- exit	الخروج من البرنامج

ونلاحظ في البرنامج استخدام إجراء لكل مهمة من المهام ، ما عدا الخروج من البرنامج :

1- في الإجراء crRoomFile ، يتم تكوين ملف جديد للحجرات (باستخدام جملة REWRITE) .

2- في الإجراء ListVacant ، يتم عرض الحجرات الشاغرة .

3- في الإجراء ListFilled ، يتم عرض الحجرات المشغولة .

4- في الإجراء checkin ، يتم البحث عن حجرة شاغرة بالسعة المناسبة ، وعند إيجاد هذه الحجرة ، تظهر العبارة :

suggest room number ...

ok ? (y/n)

وهو اقتراح لحجرة معينة ، والمطلوب إدخال إما y في حالة الموافقة أو n في حالة عدم الموافقة .

أما إذا لم يجد حجرة مناسبة (إما لأن الحجرات كلها مشغولة ، أو لعدم وجود السعة المناسبة) فإن الرد يكون :

no suitable vacancy

أي لا توجد حجرة مناسبة و شاغرة .

لاحظ استخدام جملة seek في الإجراء update على النحو :

SEEK (f, I);

وتعني التحول إلى السجل رقم I من الملف f . ويقصد بالرقم هنا رقم ترتيبه في الملف ، وهذا الترتيب يبدأ من الصفر للسجل الأول ، وواحد للسجل الثاني ، وهكذا .

وقد استخدمت جملة SEEK في البرنامج لتعديل السجل بكتابته من جديد في الملف بعد تعديله .

5- في الإجراء checkout ، يتم البحث عن الحجرة المطلوب إخلاؤها وتعديل حالتها من 'f' (أي مشغولة) إلى 'v' (أي شاغرة). أيضاً ، نستخدم هنا جملة SEEK لتعديل الملف ، بعد تحديد موقعه في عملية البحث ، ثم استدعاء الإجراء update .

شكل (1-5-6) : برنامج إدارة عملية الحجز في الفنادق .

```
PROGRAM hotel;
USES CRT;
TYPE
  date=RECORD
    day: 1..31;
    month:1..12;
    year: INTEGER;
  END;
  family=RECORD
    name : STRING[15];
    size : INTEGER;
  END;
  rooms=RECORD
    num: INTEGER;
    status : CHAR;
    capacity : INTEGER;
    CASE st : CHAR OF
      'v' : ( d : date );
      'f' : ( fam : family);
    END;
VAR
  roomsFile : FILE OF rooms;
  rm : rooms;
  fam : family;
  i , rmNum, ch , nr : INTEGER;
```

```
PROCEDURE menu;
```

```
BEGIN
```

```
    WRITELN;
```

```
    FOR i:=1 TO 70 DO WRITE('*');
```

```
    WRITELN;
```

```
    WRITELN('Choose one of the following  
numbers:');
```

```
    WRITELN('1. Create rooms file.');
```

```
    WRITELN('2. List vacant rooms');
```

```
    WRITELN('3. List filled rooms');
```

```
    WRITELN('4. Check In');
```

```
    WRITELN('5. Check out');
```

```
    WRITELN('6. Exit');
```

```
    FOR i:=1 TO 70 DO WRITE('*');
```

```
END;
```

```
PROCEDURE crRoomFile;
```

```
BEGIN
```

```
    REWRITE(roomsFile);
```

```
    WRITE('Enter number of rooms-->');
```

```
    READ(nr);
```

```
    FOR i:=1 TO nr DO
```

```
        BEGIN
```

```
            WITH rm DO
```

```
                BEGIN
```

```
                    WRITE('Enter room number-->');
```

```
                    READLN(num);
```

```
                    WRITE('Enter room capacity-->');
```

```
                    READLN(capacity);
```

```
                    WRITE('Enter room status( f or v)-
```

```
->');
```

```
                    READLN(status);
```

```
                    IF status='v' THEN
```

```
                        BEGIN
```

```
                            WRITELN('Enter    date    of last  
occupation');
```

```
                            WRITE('Enter day-->');
```

```
                            READLN(d.day);
```

```
                            WRITE('Enter month-->');
```

```

        READLN(d.month);
        WRITE('Enter year-->');
READLN(d.year);
        END
        ELSE
        BEGIN
            WRITE('Enter name of occupying
                    family→');
            READLN(fam.name);
            WRITE('Enter size of family-->');
            READLN(fam.size);
        END;
        WRITE(roomsFile, rm);
    END; { with }
    END; { for }
    CLOSE(roomsFile);
END; { procedure crRoomsFile }

```

```

PROCEDURE ListVacant;
BEGIN
    WRITELN(' List of vacant rooms ');
    WRITELN;
    WRITELN('R.N.':5,'capacity':10, ' date of
last occupation');
    RESET(roomsFile);
    WHILE NOT EOF(roomsFile) DO4
    BEGIN
        READ(roomsFile, rm);
        IF rm.status='v' THEN
            WITH rm DO
                WRITELN(num:5,capacity:10,
                    d.day:2,'/',d.month:2,'/',d.year:5);
            END;
        CLOSE(roomsFile);
    END;
END; {ListVacant}

```

```

PROCEDURE ListFilled;
BEGIN
    WRITELN;
    WRITELN('List of filled rooms':30);
    WRITELN;
    WRITELN('rnNum':5,'Name':15,'persons':10);
    RESET(roomsFile);
    WHILE NOT EOF(roomsFile) DO
    BEGIN
        READ(roomsFile, rm);
        IF rm.status='f' THEN
        BEGIN
            WITH rm DO
                WRITELN(num:5,fam.name:15,fam.size:10);
        END;
    END;
    CLOSE(roomsFile);
END;

```

```

PROCEDURE update(recNum : INTEGER);
BEGIN
    RESET(roomsFile);
    seek(roomsFile, recNum);
    WRITE(roomsFile, rm);
    CLOSE(roomsFile);
END;

```

```

PROCEDURE CheckIn;
VAR
    i, famsize : INTEGER;
    found : BOOLEAN;
    answer : CHAR;
BEGIN
    WRITE('Enter number of persons-->');
    READLN(famsize);
        { find the first vacant room with
suitable capacity}
    RESET(roomsFile);
    found:=FALSE;
    i:=0;
    WHILE NOT found AND NOT EOF(roomsFile) DO
    BEGIN
        READ(roomsFile, rm);
        IF (rm.capacity>=famsize)AND
(rm.status='v') THEN
            BEGIN
                WRITELN('suggest room number ',rm.num);
                found:= TRUE;
            END;
            i:=i+1;
        END;
    CLOSE(roomsFile);
    IF NOT found THEN
        WRITELN(' No suitable room is available')
    ELSE
    BEGIN
        WRITE('OK? y or n-->');
        READLN(answer);
        IF(answer='y') THEN
            BEGIN
                rm.status:='f';
                WRITE('Enter name-->');
                READLN(rm.fam.name);
                rm.fam.size:=famsize;
                update(i-1);
                WRITELN('check-in completed');
            END;
        END;
    END;

```

```

    END;
END; { CheckIn }

PROCEDURE CheckOut;
VAR
    today:1..31;
    mm: 1..12;
    yr : INTEGER;
    found : BOOLEAN;
BEGIN
    WRITE('Enter room number-->');
    READ(rmNum);
    WRITE('Enter date: d m y -->');
    READ(today, mm, yr);
    RESET(roomsFile);
    found:=FALSE;
    i:=0;
    WHILE (NOTfound)AND (NOT EOF(roomsFile)) DO
    BEGIN
        READ(roomsFile, rm);
        IF (rm.num=rmNum) THEN
            found:=TRUE;
            i:=i+1;
        END;
    CLOSE(roomsFile);
    IF found THEN
    BEGIN
        WITH rm DO
        BEGIN
            status := 'v';
            d.day:=today;
            d.month:=mm;
            d.year:=yr;
        END;
        update(i-1);
        WRITELN('Check-out is completed.');
```

```

    END
    ELSE
        WRITELN('Room was not found');
```

```
{ main program }
```

```
BEGIN
```

```
  CLRSCR;  
  WRITELN ('HOTEL MANAGEMENT PROGRAM':40);  
  WRITELN('WRITTEN BY Dr. Omar Zarty');  
  WRITELN;  
  ASSIGN(roomsFile,'rooms.dta');  
  REPEAT  
    menu;  
    WRITELN;  
    WRITE('Your choice -->');  
    READ(ch);  
    CASE ch OF  
      1: crRoomFile;  
      2: ListVacant;  
      3: ListFilled;  
      4: CheckIn;  
      5: CheckOut ;  
    END;  
  UNTIL  
    ch>5;
```

```
END.
```

6-6 تحديد الثوابت (جملة CONST)

قد نحتاج أحياناً إلى أن تكون جملة تحديد النوع قابلة للتغيير ، كأن نطلب أن يكون الاسم متغيراً حرفياً ، ولكن دون تحديد للعدد الأقصى لهذه الحروف . في مثل هذه الحالة ، من المفيد أن نحدد هذا العدد في جملة تحديد الثوابت في بداية البرنامج . فمثلاً الجملة :

```
CONST mx = 20;
```

تعني أن mx مقدار ثابت بقيمة 20 ، وهذا النوع من الجمل يأتي قبل جملة TYPE ، وبالتالي يمكن استخدام mx في هذه الجملة على سبيل المثال على النحو التالي :

```
TYPE name = STRING [mx];
```

أو :

```
TYPE mat = ARRAY [1 .. mx] OF CHAR;
```

ويمكننا أن نحدد أكثر من ثابت في جملة CONST ، مثل :

```
CONST W = 5.2;  
pi = 3.14;  
c = '?';  
t = true;
```

ويجب الإشارة هنا إلى أن موضع تحديد الثابت في البرنامج يأتي

بعد تحديد العنوان Label Declaration للجمل وقبل تحديد الأنواع TYPE .

وكمثال على الترتيب ، قد يكون لدينا هذا الجزء من البرنامج :

```
PROGRAM order;  
LABEL 10, 15, 20;  
CONST dim = 12; ch = '*';  
TYPE Names = ARRAY [1..dim] OF CHAR  
VAR names: Names; i: 1..dim
```

6-7 المؤشرات Pointers

تستخدم المؤشرات في تمثيل تراكيب البيانات مثل القائمة المتصلة linked lists ، وبصورة عامة للتخزين في الذاكرة بطريقة ديناميكية حيث يتم تخصيص موقع في الذاكرة للمتغير أثناء تنفيذ البرنامج ، وبالتالي نتجنب طريقة حجز الذاكرة من البداية كما هو الحال في المصفوفات .
بالإضافة إلى ذلك ، توفر لنا المؤشرات طريقة غير مباشرة لمعالجة قيمة متغير أو سجل كما في البرنامج التالي :

```
PROGRAM pointers1;  
TYPE intPointer = ^INTEGER;  
VAR ip : intPointer;  
BEGIN  
    NEW(ip);  
    Ip^ := 12 ;  
    WRITELN(' The value is ' , ip^ );  
END.
```

في هذا البرنامج نلاحظ ما يلي :

1- الجملة :

```
TYPE intPointer = ^INTEGER;
```

- . تحدد نوعا جديدا هو intPointer وهو عبارة عن مؤشر للنوع INTEGER .
- . أي إن أي متغير من هذا النوع هو مؤشر لمتغير من النوع الصحيح integer .
- لاحظ هنا وجود الرمز ^ الذي يقع على يسار النوع INTEGER ، وهذا الرمز هو الذي يحدد أن النوع المحدد هو عبارة عن مؤشر .

2- الجملة :

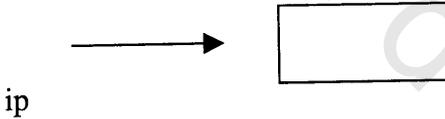
VAR ip : intPointer;

تحدد أن المتغير ip هو من النوع intPointer ، وبالتالي فإن ip ليس من النوع الذي يخزن قيمة عددية بل هو مؤشر ؛ أي إنه يحتوي على عنوان لموقع في الذاكرة .

3- الجملة :

NEW(ip);

هي استدعاء إجراء جاهز في لغة باسكال يقوم بإنشاء موقع (متغير) جديد أثناء تنفيذ البرنامج . ويعتبر ip في هذه الحالة مؤشراً لهذا الموقع . وعادة ما تمثل ذلك بالرسم التالي :



4- الجملة :

Ip^ := 12 ;

تعني طلباً بتخزين القيمة العددية الصحيحة 12 في الموقع المؤشر له بالمؤشر

. ip

6-8 القوائم المتصلة LINKED LISTS

لاحظنا عند كتابة برنامج لترتيب البيانات ضرورة استخدام المصفوفات كوسيلة لتخزين البيانات في الذاكرة وإعادة ترتيبها ؛ حتى نحصل على الترتيب التنازلي أو التصاعدي المطلوب .

إلا أن هناك طريقة أخرى قد تكون أكثر كفاءة ، وتعتمد هذه الطريقة على مفهوم المؤشرات Pointers . فبدلاً من تحديد موضع اسم في القائمة باستخدام دليل المصفوفة ، يمكننا ترتيب القائمة بوضع مؤشر (مع كل اسم) يشير إلى الاسم الذي يليه في القائمة . فإذا كان - على سبيل المثال - كل طالب في مجموعة من الطلبة يعرف الطالب الذي يليه في القائمة ، وكان الطالب الموجود في رأس القائمة (Listhead) معروفاً ، يمكننا ترتيب هذه المجموعة بحيث يشير الطالب الأول إلى الطالب الثاني ، ويشير الطالب الثاني إلى الطالب الثالث... وهكذا .

إن ميزة هذا النوع من الترتيب يتمثل في سهولة إضافة طالب أو إلغاءه من المجموعة دون التأثير على الترتيب كله ، بل يكفي عند الإضافة أن تحدد للطالب الجديد من يأتي بعده أو قبله في الترتيب . وتسمى القائمة التي تحدد بطريقة المؤشرات بالقائمة المتصلة linked list . لاحظ أن التعامل مع القوائم الموصولة - مثل المقارنة والتعيين - يتم على مؤسراتها وليس على سجلاتها مما يجعل المعالجة أسرع وأكثر كفاءة .

ونظراً لإمكانية الزيادة أو النقص في حجم الجزء الذي تحتله القائمة المتصلة أثناء تنفيذ البرنامج ، فتعتبر تركيبة القائمة المتصلة من النوع الحركي dynamic structure بينما تعتبر المصفوفة من النوع الساكن static structure نظراً لأن حجمها يجب أن يحدد مسبقاً ويبقى ثابتاً عند التنفيذ .

وكما ذكرنا سابقاً فإن لغة باسكال تحتوي على هذا النوع من التراكيب باستخدام الرمز \wedge . فمثلاً إذا كان name نضيداً يتكون من 20 حرفاً ، فإن الجملة :

```
TYPE name = STRING [20];  
NamePointer =  $\wedge$ name;
```

تجعل النوع namePointer مؤشراً للنوع name . وإذا كان السجل rec على النحو التالي :

```
rec = RECORD  
    item: name;  
    pt: namePointer;  
END
```

فذلك يعني أن القائمة تتكون من سجلات ، كل سجل يتكون من مجالين، الأول من النوع name والثاني من النوع المؤشر له ، ونستخدم هذا المؤشر عادة ليؤشر على السجل الموالي في القائمة .
وقبل أن نستخدم متغيراً من النوع namePointre يجب أن نحجز له حيزاً في الذاكرة (وفي هذا المثال يبلغ الحيز 20 بايت) ، وهذا ما يقوم به الإجراء الجاهز NEW ، أي أن الجملة :

```
NEW ( pt );
```

هي عملية استدعاء الإجراء NEW لحجز موقع في الذاكرة لمتغير من النوع namePointer المؤشر له بالمؤشر pt .
ويمكننا تخزين اسم الموقع المؤشر له بالمؤشر pt باستخدام الرمز \wedge على يمين pt على النحو التالي :

```
pt $\wedge$  := 'ahmad' ;
```

لاحظ الفرق بين pt المؤشر و pt \wedge المتغير المؤشر له .

ونرجع الآن للقائمة المتصلة ، وبالخصوص إلى آخر سجل في هذه القائمة،
أي السجل الذي لا يؤشر إلى سجل بعده ؛ حيث يمكننا تعرف هذا السجل إذا
كان مؤشره هو NIL وهي كلمة محجوزة تفيد بأن المؤشر يؤشر إلى لا شيء .

مثال (6-8-1)

ماذا يطبع البرنامج التالي ؟

```
PROGRAM pointers;
TYPE pt=^rec;
    rec=RECORD
        f1 : INTEGER;
        f2 : pt;
    END;
VAR p, head : pt;
    i : INTEGER;
BEGIN
    head := NIL;
    FOR i:=1 TO 5 DO
    BEGIN
        NEW(p) ;
        p^.f1 := 3*i;
        p^.f2 := head;
        head:=p;
    END;
    p := head;
    FOR i:= 1 TO 5 DO
    BEGIN
        WRITELN(p^.f1) ;
        p:=p^.f2;
    END;
END.
```

إذا تتبعنا هذا البرنامج نجد أولاً أن النوع pt يستخدم للإشارة إلى السجلات من النوع rec المتكونه بدورها من مجالين : الأول من النوع INEGER والثاني من النوع pt . ونستخدم في البرنامج مؤشرين هما p و head من النوع pt. في البداية نعطي head (وهو يعني رأس القائمة) القيمة NIL ، وهذا يعني أن القائمة في البداية خالية empty . ثم يبدأ تعيين 5 عناصر للقائمة باستخدام دورة FOR . لاحظ هنا أن المجال الأول للسجل المؤشر له بالمؤشر p هو $i * 3$ حيث ($i = 1, 2, 3, 4, 5$) ، أي القيم 3 ثم 6 ثم 9 ثم 12 ثم 15 تتعين في المجال الأول للسجلات الخمسة التي يجري تكوينها . أما المجال الثاني لهذه السجلات فيحتوي على مؤشر يشير إلى السجل الذي قبله في الترتيب . ويحتوي السجل الأول على NIL في المجال الثاني لعدم وجود سجل قبله. لاحظ أيضاً أن head يشير إلى آخر سجل تم تكوينه ، لذلك تتم طباعة القيم (15 ثم 12 ثم 9 ثم 6 ثم 3) أي بعكس الترتيب الذي تكونت به .

مثال (2-8-6) :

المطلوب كتابة برنامج يقوم بالإجراءات التالية :

أ. قراءة مجموعة من الأسماء و تخزينها في قائمة موصولة .

ب. طباعة هذه القائمة بنفس الترتيب المدخلة به .

ت. حساب عدد الأسماء بالقائمة و طباعته .

يبين الشكل (1-8-6) البرنامج المطلوب ، و يتكون هذا البرنامج من 3

إجراءات ، هي :

1- إجراء readList لقراءة الأسماء person المدخلة من لوحة المفاتيح 'con' ،

ولإنهاء الإدخال نضغط على المفاتيح CTRL Z .

نلاحظ أن المؤشر p هو من النوع pointer الذي يشير إلى سجل له مجللان هما الاسم name ومؤشر next الذي يشير إلى السجل الموالي في القائمة . وبالتللي فإن p^{\wedge} يعبر عن سجل (وليس مؤشراً) . كما نلاحظ أن المؤشر s قد استخدم لتخزين قيمة p قبل زيادتها بواسطة الإجراء new (p) .

2- الإجراء writeList لطباعة القائمة ابتداء من رأس القائمة head إلى أن يصبح المؤشر next يشير إلى NIL (أي لا شيء) .

3- الإجراء (count Name) لحساب عدد (count) الأسماء الموجودة بالقائمة، ابتداء من السجل الذي يحمل head كمؤشر (رأس قائمة) ، حتى نصل إلى السجل الذي يحمل القيمة NIL للمؤشر الذي يشير إلى السجل الموالي ؛ أي إن نهاية القائمة تتحدد عندما يفيدنا السجل الحالي (أي المدخل) بأنه لا سجل بعده .

شكل (1-8-6) : برنامج لمعالجة قائمة موصولة باستخدام المؤشرات.

```
PROGRAM CountNamesInList;
CONST n=10;
TYPE names=STRING[n];
      pointer = ^rec;
      rec= RECORD
          name : names;
          next : pointer;
      END;
VAR
  data : TEXT;
  t, head : pointer;
  c : INTEGER;
```

```

PROCEDURE readList(VAR f:text ; VAR
head:pointer);
VAR person : names;
    p,s : pointer;
BEGIN
    p:=head;
    WHILE NOT EOF(f) DO
    BEGIN
        readln(f, person);
        p^.name := person;
        s:=p;
        NEW(p);
        s^.next:=p;
    END;
    p^.next:=NIL;
END; { ReadList }

PROCEDURE writeList( head : pointer);
VAR p:pointer;
BEGIN
    WRITELN;
    p:=head;
    WHILE p^.next<> NIL DO
    BEGIN
        WRITELN(p^.name);
        p:=p^.next;
    END;
END; {writeList}

PROCEDURE countList( head:pointer; VAR
count : INTEGER);
VAR p:pointer;
BEGIN
    p:=head;
    count:=0;
    WHILE p^.next<> NIL DO
    BEGIN
        count := count + 1;
        p:=p^.next;
    END;
END;

```

```
END;  
END;{countList}
```

```
BEGIN {main}  
  ASSIGN (data, '\tp\prgs\data.lst');  
  RESET(data);  
  readList(data, head);  
  writeList(head);  
  countList(head, c);  
  WRITELN;  
  WRITELN('Number of names in the  
List s ', c);  
END.
```

لتوضيح أهمية استخدام المؤشرات في معالجة البيانات ، نريد هنا أن نكتب إجراء لإضافة اسم لقائمة مرتبة أبجدياً (تصاعدياً) دون أن يرتبك الترتيب .

يبين البرنامج بالشكل (2-7-6) كيف نقوم بهذه المهمة ، وذلك باستخدام ثلاث إجراءات ، هي :

- 1- readList وفيه تتم قراءة البيانات المرتبة من الملف f . وهذا الإجراء يكافئ الإجراء بالإسم نفسه في المثال (2-8-6) .
- 2- الإجراء writeList وهو يقوم بطباعة قائمة الأسماء كلها كما في المثال السابق .
- 3- الإجراء المطلوب في هذا المثال AddToList يقوم بإضافة الاسم nme إلى القائمة المرتبة مع وضعه في مكانه من الترتيب .

شكل (6-7-2) : إضافة اسم لقائمة مرتبة باستخدام المؤشرات .

```
PROGRAM LinkedList;
USES CRT;
CONST n=20;
TYPE
    names=STRING[n];
    pointer=^person;
    person=RECORD
        name: names;
        next : pointer;
    END;
VAR
    newName : names;
    head : pointer;
    SortedFile, UnsortedFile : TEXT;
    file1 , file2 : names;

PROCEDURE WriteList(VAR f : TEXT);
VAR p : pointer;
BEGIN
    p:=head;
    WHILE p<> NIL DO
    BEGIN
        WRITELN(f, p^.name);
        WRITELN(p^.name);
        p:=p^.next;
    END;
    CLOSE(f);
END; { writeList }
```

```

PROCEDURE AddToList(item: names);
VAR
    r, lastr, newr : pointer;
    Located : BOOLEAN;
BEGIN
    r:=head;
    located := FALSE;
    WHILE NOT located AND (r<>NIL) DO
        IF NewName <= r^.name THEN
            Located := true
        ELSE
            BEGIN
                lastr := r;
                r:=r^.next;
            END;
        NEW(newr);
        newr^.name:=item;
        newr^.next:=r;
        IF r=head THEN
            head:=newr
        ELSE
            lastr^.next:=newr;
    END; {AddToList }

```

```

BEGIN {main}
  CLRSCR;
  WRITE('Enter name of unsorted file-->');
  READLN(file1);
  WRITE('Enter name of file where to save
sorted list-->');
  READLN(file2);
  ASSIGN(unsortedFile, file1);
  ASSIGN(sortedFile, file2);
  RESET(unsortedFile);
  REWRITE(sortedFile);
  head:=NIL;
  WRITELN('Unsorted List:'); WRITELN;
  WHILE NOT EOF(unsortedFile) DO
  BEGIN
    READLN(unsortedFile, NewName);
    WRITELN(NewName);
    AddToList(NewName);
  END;

  WRITELN;
  WRITELN('Sorted List:'); WRITELN;
  WriteList(sortedFile);
  WRITELN(' This list is saved in file :
',file2);
  WRITELN(' You can view the sorted file by
using TYPE comand');

END.

```

ولتوضيح هذا البرنامج ، نفترض أن لدينا القائمة المرتبة التالية :

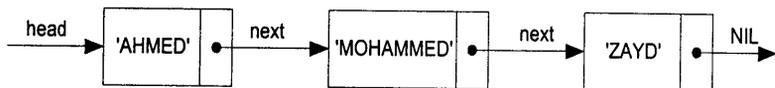
'AHMED'

'MOHAMMED'

'ZAYD'

بعد قراءة هذه القائمة بالإجراء readList ، تصبح القائمة موصولة على

النحو التالي :

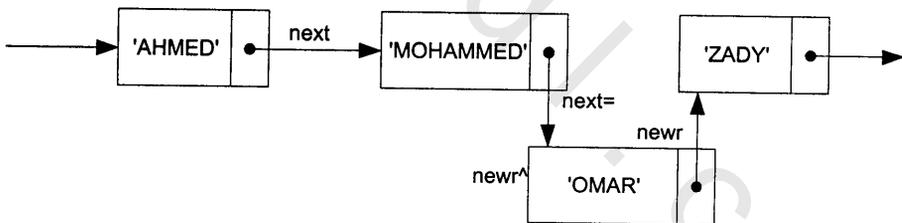


والآن نفترض أننا نريد إضافة الاسم 'OMAR' للقائمة ؛ حيث إن :

'MOHAMMED' < 'OMAR' < 'ZADY'

فإن الإجراء AddToList يقوم بوضع 'OMAR' في موضع جديد هو newr^

على أن يشير مجاله الثاني إلى الموضع 'ZAYD' الذي كان يشير إلى 'MOHAMMED' ، وتصبح القائمة موصولة على النحو التالي :



كما نلاحظ أن الإجراء AddToList يأخذ في الاعتبار أن الاسم المضاف

قد يكون ترتيبه قبل رأس القائمة 'AHMAD' ، وفي هذه الحالة نعين المؤشر head

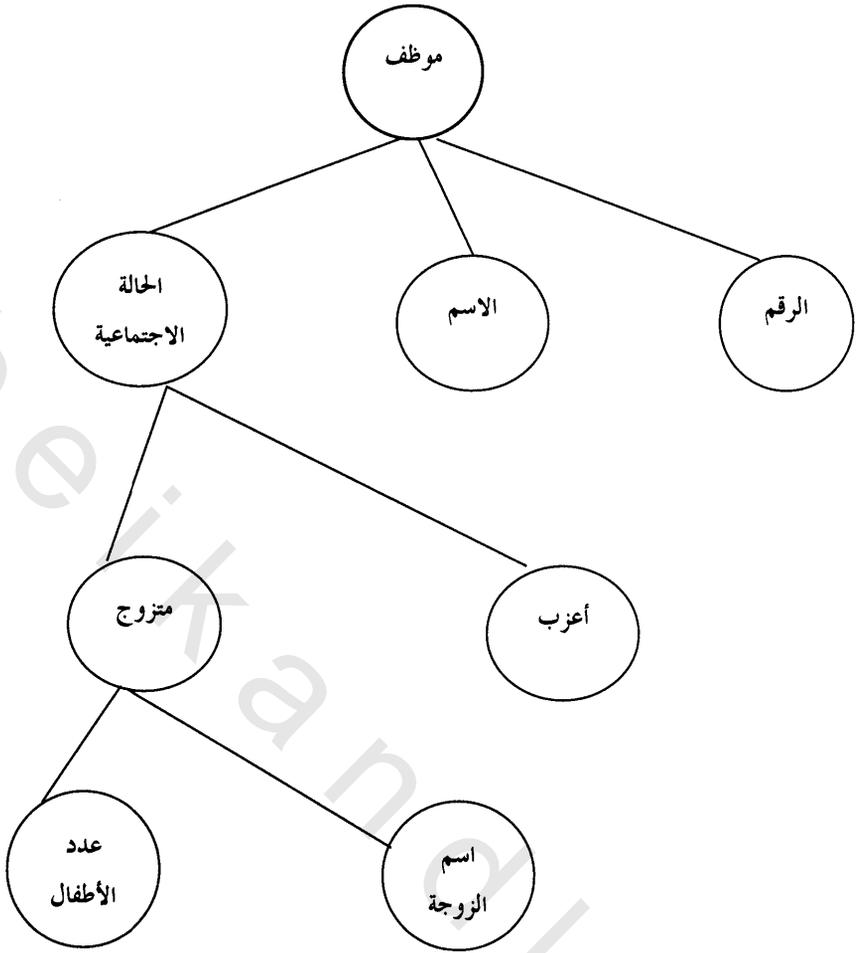
للمؤشر newr .

9-6 تمارين

1- ماذا يطبع البرنامج التالي :

```
PROGRAM EXAMPLE;  
TYPE DAYS = ( SAT, SUN, MON, TUE, WED, THU,  
FRI )  
VAR D: DAYS;  
BEGIN  
  FOR D: = SAT TO FRI DO  
    CASE D OF  
      SAT, FRI:  
        WRITELN  
          ('No WORK');  
      SUN..THU:  
        WRITELN ('WORK');  
    END;  
  END.  
END.
```

2- استخدم جملة TYPE لتحديد نوع السجل عن الموظف في مؤسسة على النحو التالي :



3- اكتب برنامجاً لتكوين ملف من السجلات المبينة في التمرين (2) . استخدم جملة WITH-DO في البرنامج .

4- البرنامج المبين في الشكل (1-5-6) يقوم ببعض المهام الخاصة بإدارة الفندق ، ولكن ليس كل المهام . حاول أن تدخل بعض التحسينات في البرنامج حتى يكون أكثر واقعية . وعلى سبيل المثال :

أ. ضف إلى سجل الحجرة بيانات عن التكلفة .

ب. يقوم البرنامج بعرض (أول) حجرة شاغرة بالسعة المطلوبة . ولكن قد لا تكون هذه الحجرة ملائمة لسبب من الأسباب . عدل في البرنامج بحيث يتم التحول إلى الحجرة الشاغرة التي تليها في الترتيب في حالة عدم قبول الحجرة المعروضة.

تم بتعديل في طريقة عرض المعلومات على الشاشة لتصبح أكثر وضوحاً .
جـ . قد يحتاج الفندق إلى التعديل في ملف الحجرات من حين إلى آخر. أضف إلى قائمة الإجراءات إجراء يقوم بمثل هذا التعديل .
5- ما معني الكلمات التالية ؟

pointer
dynamic structure
static structure
Linked list
listhead
nil

6- ما الغرض من الإجراءات الجاهزة التالية وكيف تستخدم ؟

NEW
DISPOSE
SEEK

7- لديك ملف يحتوي على أرقام الطلبة وأسمائهم ودرجاتهم النهائية . المطلوب استخدام المؤشرات لقراءة هذا الملف تم طباعته بعكس الترتيب الموجود به .

8- اكتب برنامجاً لقراءة سطر من البيانات (من اليسار إلى اليمين) ثم طباعته معكوساً (أي من اليمين إلى اليسار) مستخدماً المؤشرات .

9- استخدم المؤشرات في كتابة برنامج لحساب

$$\sum_{i=1}^{10} (x_i - \bar{x})^2$$

حيث إن :

$$\bar{x} = \frac{1}{10} \sum_{i=1}^{10} x_i$$

لا تستخدم المصفوفات في البرنامج . والبيانات x_i (I من 1 إلى 10)
تقرأ مرة واحدة .

10- تتبع البرنامج التالي وبين ناتج الطباعة .

```
program pt3;
type pt = ^rec;
      rec = record
          f1: char;
          f2: pt;
        end;
var p, last: pt;
    k: char;
begin
  last := nil;
  for k := 'a' to 'h' do
  begin
    new (p);
    p^. f1 := k;
    p^. f2 := last;
    last := p;
    writeln (p^. f1)
  end;
  p := last;
  while p <> nil do
  begin
    writeln (p^. f1);
    p := p^. f2
  end
END
END.
```

11- اكتب برنامجاً يستخدم المؤشرات ويقوم بالتالي :

أ. قراءة قائمة مرتبة من أسماء الطلبة ودرجاتهم (الترتيب حسب الدرجة) .

ب. إضافة طالب لهذه القائمة ووضعه في الترتيب الصحيح .

ت. إلغاء طالب من القائمة دون الإخلال بالترتيب.

* * *