خوارزم

بيئة لكتابة الخوارزميات المجردة وترجمتها إلى PASCAL و C

أ. زقور جمال الدين

ESI : Ecole Supérieure d'Informatique

(ALGER)

مـلـخـص

1. مقدمة عامة
مقدمة
نوافذ
الخدمات
لغة Z
الوثائق

2. مراحل من أجل تحقيق برنامج في إطار خوارزم
التعرف على لغة خوارزمية
تعديل الخوارزمية
فحص البنية
التنفيذ
محاكاة
تتبع
تبديل إلى لغة برمجة
برمجة PASCAL أو C
مثال خوارزمة
ترجمة إلى PASCAL
ترجمة إلى C

3. لغة زاي (القاعدة)
نظرة عامة
هيكل الخوارزمية
تعليمات الشرطية
التعليمات البسيطة
عدديات
مؤشرات
تعبيرات
التعاليق
تعليمات متطورة
دوال قياسية
دوال لتوليد عشوائي
دوال على السلاسل
تعريف إجراء
تعريف دالة
مثال خوارزمية

4. لغة Z (تراكيب المعطيات)
البنى
الجداول
القوائم
القوائم المزدوجة
الصفوف
الأكوام
الأشجار البحث الثنائي
الأشجار البحث المتعددة

الملفات

5. الآلات المجردة

مفهوم الآلة المجردة
جداول
بنى
قوائم
قوائم ثنائية الاتجاه
أكوام
صفوف
أشجار البحث الثنائي
أشجار البحث المتعددة
ملفات

6. الانتقال من Z إلى PASCAL

تعريف
تعبيرات
تخصيص
تكرار المشروط
تكرار المعدود
الاشتراط
قراءة
كتابة
إجراء
دالة
دوال قياسية
خوارزمية

7. تطبيق الآلات المجردة في باسكال (PASCAL)

جداول
بنى
قوائم
قوائم مزدوجة
أكوام
صفوف
أشجار البحث الثنائية
أشجارالبحث المتعددة
ملفات

8. الانتقال من Z إلى C

تعريف
تعبيرات
تخصيص
تكرار المشروط
تكرار المعدود
الاشتراط
قراءة
كتابة
إجراء

دالة
دوال قياسية
خوارزمية

9. تطبيق الآلات المجردة في سي (C)

جداول
بنى
قوائم
قوائم مزدوجة
أكوام
صفوف
أشجار البحث الثنائية
أشجار البحث المتعددة
ملفات

10. فهرس الكلمات الرئيسية

عرض تقديمي

مقدمة
نوافذ
الخدمات
لغة Z
الوثائق

مقدمة

خوارزم هي بيئة للتعلم وتعميق هياكل البيانات والملفات الرئيسية.
يوفر خوارزم إمكانية كتابة الخوارزميات بلغة خوارزمية (لغة Z) كما يوفر تنسيقها و تنفيذها و محاكاتها وتحويلها تلقائيا إلى لغات البرمجة PASCAL و C.
يهدف خوارزم إلى مساعدة تصميم الخوارزميات.

نوافذ

يقدم خوارزم العديد من النوافذ التي تظهر :

- البيانات (قراءات)- نتائج المدى (كتابات)
- نتائج المحاكاة (تتبع كامل)

في أي وقت في خوارزم ، يمكنك استدعاء المساعدة أو تنشيط عمليات مختلفة باستخدام الأزرار.

خدمات

تقدم خوارزم الخدمات التالية:

-محرر لكتابة الخوارزميات الخاصة بك توفير جميع الوثائق على لغة Z.

-منظم التنسيق لترتيب الخوارزميات الخاصة بك.
ميزاته الرئيسية هي :

.كل تعليمة مكتوبة على سطر مختلف,
.يتم تمييز هياكل التحكم,
.تعليمات من نفس المستوى تبدأ في نفس العمود .خطوة التنسيق متغيرة.

- جهاز لتشغيل الخوارزميات الخاصة بك ، وإعطاء نتيجة لجميع الكتابات المنبعثة (نافذة "نتائج").

- جهاز محاكاة لإعطاء التقدم الكامل (نافذة "محاكاة") من الخوارزميات الخاصة بك من خلال إظهار تطور جميع المتغيرات التلاعب بها. هذا يساعدك على تصحيح أو بناء الخوارزميات الخاصة بك.

-جهاز تحويل تلقائي للخوارزميات إلى برامج PASCAL أو C.

لغة Z

في بيئة خوارزم ، يتم التعبير عن الخوارزميات بلغة خوارزمية (لغة Z).
خصوصية لغة Z تكمن في حقيقة أنه يمكن كتابة الخوارزميات على آلات مجردة محاكاة هياكل البيانات الرئيسية.

تم تصميم لغة Zفي المقام الأول للأغراض التالية:

-تجربة هياكل البيانات الرئيسية ، بغض النظر عن تمثيلها في الذاكرة ، من خلال تطوير الخوارزميات على

الجداول
الهياكل،
القوائم المرتبطة،
القوائم المرتبطة المزدوجة،
الصفوف،
الأكوام،
الأشجار البحث الثنائي،
الأشجار البحث المتعدد.

-إنشاء وإدارة هياكل البيانات المعقدة بما في ذلك
قائمة مرتبطة لصفوف،
قائمة مرتبطة لأكوام ،
شجرة القوائم المرتبطة،
قائمة مرتبطة من أكوام من جداول،
.الخ...

-كتابة الخوارزميات العودية.

بفضل آلات مجردة على الملفات ، لغة Zتسمح أيضا باستخدام الملفات وبناء كل من هياكل الملفات البسيطة والمعقدة.

الوثائق

يقدم "خوارزم "جميع الوثائق على لغة .Z
يوفر "خوارزم "الترجمة من Z إلى PASCAL أو C
يعطي "خوارزم "بعض التطبيقات الممكنة في PASCAL و C من الآلات المجردة المختلفة التي تم النظر فيها في لغة .Z
يتم جمع جميع الوثائق في نص مفرط.

مراحل من أجل تحقيق برنامج في إطار خوارزم

التعرف على لغة خوارزمية
تعديل الخوارزمية
فحص البنية
التنفيذ
محاكاة
تتبع
تبديل إلى لغة برمجة
برمجة PASCAL أو C
مثال خوارزمة
ترجمة إلى PASCAL
ترجمة إلى C

التعرف على لغة خوارزمية

تعلم اللغة الخوارزمية المستخدمة.
استخدم المساعدة المقدمة.

تعديل الخوارزمية

اكتب خوارزمية جديدة أو صحح خوارزمية موجودة.

فحص البنية

قم بتشغيل وحدة الترتيب.
كرر ما دامت هناك أخطاء
.صحح الأخطاء
.أعد تشغيل وحدة الترتيب

في هذه المرحلة ، تكون الخوارزمية مكتوبة بشكل جيد وتم وضع مسافة بادئة لها من أجلك) .يمكنك تغيير أوضاع العرض للخوارزمية الخاصة بك)انظر قائمة "الخيارات("

التنفيذ

ابدأ تنفيذ الخوارزمية الخاصة بك.
ثم تظهر النوافذ
- البيانات التي تمت قراءتها بواسطة الخوارزمية (زر البيانات)
- الإدخالات الصادرة عن الخوارزمية الخاصة بك (زر النتائج)

إما أن تعطي الخوارزمية النتائج المتوقعة أم لا .في الحالة الأخيرة ، قم بتشغيل المحاكاة لمحاولة تحديد الأخطاء المنطقية.

محاكاة

ابدأ محاكاة الخوارزمية الخاصة بك.
وهذا يعني تشغيل الخوارزمية مع ضمان إمكانية التتبع.
تظهر النوافذ
- قراءة البيانات بواسطة الخوارزمية (زر البيانات)
- المخرجات المنبعثة من الخوارزمية الخاصة بك (زر النتائج)
- جميع التغييرات التي تم إجراؤها على المتغيرات المستخدمة (زر محاكاة)

لذلك لديك التتبع الكامل للخوارزمية التي يمكنك طباعتها وتحليلها بحثًا عن الأخطاء.

إذا كنت تريد رؤية الخطوات المختلفة للخوارزمية عن قرب ، فاطلب مخططًا.

تتبع

اطلب المحاكاة مرة أخرى مع التتبع
يمكنك بعد ذلك متابعة تطور الخوارزمية الخاصة بك خطوة بخطوة ، والخروج من الحلقة الحالية أو حتى الوحدة النمطية الحالية.

لتجنب وجود أثر كامل يمكن أن يكون طويلاً ، من الممكن تحديد طول الحلقات المستخدمة في الخوارزمية الخاصة بك. يمكنك تغيير أوضاع المحاكاة (انظر قائمة "الخيارات").

تبديل إلى لغة برمجة

بمجرد أن تصبح الخوارزمية "قيد التشغيل" ، يمكن ترجمتها تلقائيًا إلى PASCAL أو C. ما عليك سوى النقر على الزر "To Pascal" أو الزر "To C". ثم يتم عرض نافذتين منظمتين في "تجانب". يحتوي أحدهما على الخوارزمية الخاصة بك والآخر على نتيجة الترجمة.
يمكنك الرجوع إلى المساعدة المتعلقة بالانتقال إلى PASCAL أو C.

في هذه المساعدة سوف تجد
- ما يعادل Z إلى PASCAL و Z إلى C.
- جميع تطبيقات آلات Z.

تنتهي مهمة خوارزم هنا.

برمجة PASCAL أو C.

استخدم مترجم PASCAL أو C لإنهاء برنامجك بشكل نهائي. على وجه الخصوص ، يجب عليك إضافة جميع الوحدات النمطية لإدخال البيانات وإرجاع النتائج. لا تقدم لغة Z تسهيلات لمثل هذه المهام.

مثال خوارزمة

```
{ إيجاد قائمة في أخرى؟ }
ليكن
ق1 ، ق2 : قوائم ؛
بحث ، إيجاد : دالة ( منطيقي ) ؛
بداية
إنشاء_قائمة ( ق1 ، [ 2 ، 5 ، 9 ، 8 ، 3 ، 6 ] ) ؛
إنشاء_قائمة ( ق2 ، [ 12 ، 5 ، 19 ، 8 ، 3 ، 6 ، 2 ، 9 ] ) ؛
اكتب ( إيجاد ( ق1 ، ق2 ) )
نهاية
{ بحث عن كلمة في قائمة }
دالة بحث ( ق ، كلما ) : منطيقي
ليكن
ق : قائمة ؛
كلما : صحيح ؛
بداية
إذا ق = عدم
بحث := خطأ
وإلا
إذا قيمة_خلية ( ق ) = كلما
بحث := صواب
وإلا
بحث := بحث ( تالي ( ق ) ، كلما )
نهاية_إذا
نهاية_إذا
نهاية
{ هل القائمة ق1 موجودة في القائمة ق2؟ }
```

دالة إيجاد (ق1 ، ق2) : منطيقي
ليكن
ق1 ، ق2 : قوائم ؛
بداية
إذا ق1 = عدم
اهجاد := صواب
وإلا
إذا لا بحث (ق2 ، قيمة_خلية (ق1))
إيجاد := خطأ
وإلا
إيجاد := إيجاد (تالي (ق1) ، ق2)
نهاية_إذا
نهاية_إذا
نهاية

ترجمة إلى PASCAL

```
{**-----------------------------------------------**/
/** T r a n s l a t i o n Z to PASCAL (Standard) **/
/** By Pr. D.E ZEGOUR                             **/
/** E S I - Algier                                **/
/** Copywrite 2014                                **/
/**-----------------------------------------------**}

PROGRAM My_program;

{ Implementation : LIST Of INTEGERS}

{ Linked lists }
TYPE
Typeelem_LI = INTEGER;
Pointer_LI = ^Maillon_LI; { type du champ 'Adresse' }
  Maillon_LI = RECORD
  Val : Typeelem_LI;
  Next : Pointer_LI
END;

PROCEDURE Allocate_cell_LI ( VAR P : Pointer_LI ) ;
BEGIN NEW(P) END;

PROCEDURE Free_LI ( P : Pointer_LI ) ;
BEGIN DISPOSE(P) END;

PROCEDURE Ass_val_LI(P : Pointer_LI; Val : Typeelem_LI );
BEGIN P^.Val := Val END;

FUNCTION Cell_value_LI (P : Pointer_LI) : Typeelem_LI;
BEGIN Cell_value_LI := P^.Val END;
```

```
FUNCTION Next_LI( P : Pointer_LI) : Pointer_LI;
BEGIN Next_LI := P^.Next END;

PROCEDURE Ass_adr_LI( P, Q : Pointer_LI ) ;
BEGIN P^.Next := Q END;


TYPE
  Typeelem_V6I = INTEGER ;
  Typetab_V6I = ARRAY[1..6] of Typeelem_V6I ;

TYPE
  Typeelem_V8I = INTEGER ;
  Typetab_V8I = ARRAY[1..8] of Typeelem_V8I ;

{ Declaration part of variables }
VAR
  L1 : Pointer_LI;
  L2 : Pointer_LI;
 T_L1 : Typetab_V6I ;
 T_L2 : Typetab_V8I ;

{ High level operations }

{ Creating a linked list }
PROCEDURE Create_list_LI ( VAR L : Pointer_LI ; Tab : Typetab_V6I ; N: INTEGER) ;
VAR
  I : INTEGER;
  P, Q : Pointer_LI ;
BEGIN
  L:=NIL;
  FOR I := 1 TO N DO
    BEGIN
      Allocate_cell_LI( Q ) ;
      Ass_val_LI (Q, Tab[I]);
      Ass_adr_LI (Q, NIL);
      IF L = NIL
     THEN L := Q
      ELSE Ass_adr_LI (P, Q);
      P := Q
    END;
END;


{ Procedure and/or function prototypes }

FUNCTION Search (VAR L : Pointer_LI ; VAR Val : INTEGER) : BOOLEAN; FORWARD;
FUNCTION All (VAR L1 : Pointer_LI ; VAR L2 : Pointer_LI) : BOOLEAN; FORWARD;

{ Search for a value in a linked list }
```

```
FUNCTION Search (VAR L : Pointer_LI ; VAR Val : INTEGER) : BOOLEAN;
{ Declaration part of variables }
VAR
  Search2 : BOOLEAN ;
  _Px1 : Pointer_LI;

BEGIN
  IF L = NIL THEN BEGIN
  Search2 := FALSE END
  ELSE
     BEGIN
        IF Cell_value_LI(L ) = Val THEN BEGIN
        Search2 := TRUE END
        ELSE
           BEGIN
              _Px1 := Next_LI(L ) ;
              Search2 := Search (_Px1, Val )
           END
     END
  ;Search := Search2 ;
END;
{ Is L1 included in L2? }

FUNCTION All (VAR L1 : Pointer_LI ; VAR L2 : Pointer_LI) : BOOLEAN;
{ Declaration part of variables }
VAR
  All2 : BOOLEAN ;
  _Px1 : INTEGER ;
  _Px2 : Pointer_LI;

BEGIN
  IF L1 = NIL THEN BEGIN
  All2 := TRUE END
  ELSE
     BEGIN
        _Px1 := Cell_value_LI(L1 ) ;
        IF NOT Search ( L2 ,_Px1) THEN BEGIN
        All2 := FALSE END
        ELSE
           BEGIN
              _Px2 := Next_LI(L1 ) ;
              All2 := All (_Px2, L2 )
           END
       END
    ;All := All2 ;
END;

{ Body of main program }
BEGIN
  T_L1 [ 1 ] := 2 ;
```

```
  T_L1 [ 2 ] := 5 ;
  T_L1 [ 3 ] := 9 ;
  T_L1 [ 4 ] := 8 ;
  T_L1 [ 5 ] := 3 ;
  T_L1 [ 6 ] := 6 ;
  Create_list_LI ( L1 , T_L1 , 6 ) ;
  T_L2 [ 1 ] := 12 ;
  T_L2 [ 2 ] := 5 ;
  T_L2 [ 3 ] := 19 ;
  T_L2 [ 4 ] := 8 ;
  T_L2 [ 5 ] := 3 ;
  T_L2 [ 6 ] := 6 ;
  T_L2 [ 7 ] := 2 ;
  T_L2 [ 8 ] := 9 ;
  Create_list_LI ( L2 , T_L2 , 8 ) ;
  WRITELN ( All(L1,L2) )
  ;READLN;
END.
```

ترجمة إلى C

```
/**-----------------------------------------**/
/** T r a n s l a t i o n Z to C (Standard) **/
/** By Pr. D.E ZEGOUR                       **/
/** E S I - Algier                          **/
/** Copywrite 2014                          **/
/**-----------------------------------------**/

#include <stdio.h>
#include <stdlib.h>

typedef int bool ;

#define True 1
#define False 0

/** Implementation **\: LIST Of INTEGERS**/

/** Linked lists **/

typedef int Typeelem_Li ;
typedef struct Maillon_Li * Pointer_Li ;

struct Maillon_Li
{
Typeelem_Li Val ;
Pointer_Li Next ;
} ;
```

```
Pointer_Li Allocate_cell_Li (Pointer_Li *P)
{
*P = (struct Maillon_Li *) malloc( sizeof( struct Maillon_Li)) ;
(*P)->Next = NULL;
}

void Ass_val_Li(Pointer_Li P, Typeelem_Li Val)
{
P->Val = Val ;
}

void Ass_adr_Li( Pointer_Li P, Pointer_Li Q)
{
P->Next = Q ;
}

Pointer_Li Next_Li( Pointer_Li P)
{ return( P->Next ) ; }

Typeelem_Li Cell_value_Li( Pointer_Li P)
{ return( P->Val) ; }

void Free_Li ( Pointer_Li P)
{ free (P);}

/** For temporary variables **/
typedef int Typeelem_V6i;
typedef Typeelem_V6i Typetab_V6i[6];

/** For temporary variables **/
typedef int Typeelem_V8i;
typedef Typeelem_V8i Typetab_V8i[8];

/** Variables of main program **/
Pointer_Li U1=NULL;
Pointer_Li U2=NULL;
Typetab_V6i T_U1;
Typetab_V8i T_U2;

/** High level operations **/

/** creating a linked list **/
void Create_list_Li ( Pointer_Li *L, Typetab_V6i Tab, int N)
{
int I ;
Pointer_Li P, Q ;

*L =NULL;
for( I=1;I<=N;++I)
{
```

```
Allocate_cell_Li( &Q ) ;
Ass_val_Li (Q, Tab[I-1]);
Ass_adr_Li (Q, NULL);
if (*L == NULL)
*L = Q ;
else Ass_adr_Li (P, Q);
P = Q ;
}
}

/** Function prototypes **/
bool Search (Pointer_Li *U , int *Val) ;
bool All (Pointer_Li *U1 , Pointer_Li *U2) ;

/* ??? ?? ???? ?? ????? */
bool Search (Pointer_Li *U , int *Val)
{
/** Local variables **/
bool Search2 ;
Pointer_Li _Px1=NULL;

/** Body of function **/
if( *U == NULL) {
Search2 = False; }
else
{
if( Cell_value_Li ( *U ) == *Val) {
Search2 = True; }
else
{
_Px1 = Next_Li ( *U ) ;
Search2 = Search ( &_Px1, & *Val );
}
}
return Search2 ;
}
/* ?? ??????? ?1 ?????? ?? ??????? ?2? */
bool All (Pointer_Li *U1 , Pointer_Li *U2)
{
/** Local variables **/
bool All2 ;
int _Px1;
Pointer_Li _Px2=NULL;

/** Body of function **/
if( *U1 == NULL) {
All2 = True; }
else
{
_Px1 = Cell_value_Li ( *U1 ) ;
if( ! Search ( & *U2 , &_Px1)) {
```

```
All2 = False; }
else
{
_Px2 = Next_Li ( *U1 ) ;
All2 = All ( &_Px2, & *U2 );
}
}
return All2 ;
}

int main(int argc, char *argv[])
{
T_U1 [ 0 ] = 2 ;
T_U1 [ 1 ] = 5 ;
T_U1 [ 2 ] = 9 ;
T_U1 [ 3 ] = 8 ;
T_U1 [ 4 ] = 3 ;
T_U1 [ 5 ] = 6 ;
Create_list_Li (&U1 , T_U1 , 6 ) ;
T_U2 [ 0 ] = 12 ;
T_U2 [ 1 ] = 5 ;
T_U2 [ 2 ] = 19 ;
T_U2 [ 3 ] = 8 ;
T_U2 [ 4 ] = 3 ;
T_U2 [ 5 ] = 6 ;
T_U2 [ 6 ] = 2 ;
T_U2 [ 7 ] = 9 ;
Create_list_Li (&U2 , T_U2 , 8 ) ;
printf ( " %d", Azeah(&U1,&U2) );
system("PAUSE");
return 0;
}
```

نظرة عامة
هيكل الخوارزمية
تعليمات الشرطية
التعليمات البسيطة
عدديات
مؤشرات
تعبيرات
التعاليق
تعليمات متطورة
دوال قياسية
دوال لتوليد عشوائي
دوال على السلاسل
تعريف إجراء
تعريف دالة
مثال خوارزمية

نظرة عامة

< الخوارزمية هي مجموعة من وحدات متوازية ، أولها رئيسي والآخرون إما إجراءات أو دوال من أي نوع.

> لغة زاي تقبل وحدات عودية

> يتم الاتصال بين الوحدات عبر الوسطاء والمتغيرات الشاملة.

> الغة تسمح
- أي نوع من المعلمات : العددية، البنى، القوائم ، الملفات، الجداول، الأكوام، الصفوف، الأشجار و حتى نوع معقد.
- التخصيص الديناميكي للجداول و البنى
- تخصيص المتغيرات الشاملة من أي نوع

> يسمح بأربعة أنواع بسيطة: صحيح-منطقي-محرف-سلسلة .

> بعض الدوال المعتادة محددة مسبقا: باقي قسمة، أكبر، أصغر،...

> اللغة هي مجموعة الخوارزميات المجردة-مكتوبة على أساس النماذج (الآلات المجردة).

> يتم تعريف آلات مجردة على البنى، الجداول من أي حجم، الأكوام، الصفوف، أشجار البحث الثنائية، أشجار البحث المتعددة، القوائم أحادية الاتجاه، القوائم ثنائية الاتجاه.

> يتم تعريف أيضا آلة مجردة على الملفات التي تسمح باستخدامها وبناء كل من هياكل الملفات البسيطة والهياكل الأكثر تعقيدا.

> اللغة تسمح أنواع مركبة مثل كومة من صفوف من قائمة من ....آخر ذكر هو من النوع البسيط أو بنية بسيطة

> اللغة مجهزة بعمليات عالية المستوى تسمح ببناء قوائم-أشجار-صفوف-إلخ. من مجموعة من القيم (التعبيرات) أو البنى.

> اللغة توفر اثنين من دوال مفيدة جدا السماح لتوليد عشوائيا السلاسل (سلسلة_عشوائية) والأعداد الصحيحة (عدد_عشوائي).

> اللغة تسمح القراءة والكتابة من العددية-من جداول من أي بعد من العددية و من الهياكل بسيطة أو معقدة.

هيكل الخوارزمية

ليكن
{ كائنات محلية و شاملة }
{ إعلان الوحدات }
بداية
{ تعليمات }
نهاية
وحدة 1
....
وحدة ن
يمكن أن تكون كل وحدة إما إجراء أو دالة.

تعليمات الشرطية

الاشتراط
إذا ع [:]
{ تعليمات }
[وإلا
{ تعليمات }]
نهاية_إذا

التكرار المشروط
مادام ع [:]
{ تعليمات }
نهاية_مادام

التكرار المعدود
لكل := ع1، ع2 [،ع3]
{ تعليمات }
نهاية_لكل
ع3، إذا كان موجودا، يدل على الخطوة.

التعليمات البسيطة

م يدل على متغير، ع يدل على عبارة و معف يدل على إسم وحدة
[]يشير إلى جزء اختياري
{ }مجموعة

التخصيص : م := ع
القراءة : اقرأ(م1، م2،...)
يمكن أن تكون المتغيرات عددية، بنى أو جداول من أي بعد .
يمكن أن تكون التعبيرات عددية، بنى أو جداول من أي بعد
الكتابة : اكتب(ع1،ع2،...)

عدديات

أربعة أنواع عديدية مسموح بها :صحيح، منطيقي، محرف، سلسلة.

تعريف العدديات :
ليكن ‹ق› : نوع
‹ق› : قائمة المعرفات مفصولة بفواصل.
نوع هو نوع عديدي .

أمثلة:

ليكن
ب :منطيقي
س1، س2 : سلاسل
م :محرف
ص1، ص2، ص3 : صحاح

مؤشرات
يتم تعريف متغيرات ذات النوع "المؤشر "لاستخدام التراكيب البيانات .
ليكن ‹ق› : مؤشر نحو نوع-م من نوع-م من ... من ننوع-ب
‹ق› : قائمة المعرفات مفصولة بفواصل.

نوع-م في { جدول، كومة، صف، قائمة، قائمة_مزدوجة، شجرة_بحث_ثنائية، شجرة_بحث_متعددة }
نوع-ب هو عددي أو بنية بسيطة.

ملاحظة:
يمكننا الاستغناء عن هذا النوع .في الواقع ، التصريحات
ليكن ق :قائمة
ليكن ق :مؤشر نحو قائمة
متكافئة

أمثلة:
ليكن
م1: مؤشر نحو قائمة أكوام
ن1، ن2 : مؤشرات نحو أشجار-البحث-الثنائي

التعبيرات
كما هو الحال في لغات البرمجة.
أمثلة:
(أ +ب) / ج ، لا موجود، (م # 5) و لا موجود، ز(م) <> 5
تعبيرات حسابية : + , - , / , *
تعبيرات منطقية :و، أو، لا
تعبيرات على سلاسل الحروف : +
تعبيرات علائقية : <،=>، <، =<، =، <> (أو #)
ثوابت منطقية :صواب، خطأ
ثابت مؤشر :عدم

التعاليق
فيكتب التعليق ما بين حاضنتين { و } أو ما بين /* و */

تعليمات متطورة
تملأ الآلات المجردة باستخدام التعليمات المتطورة المتوفرة للإنشاء أو للملء.
إنشاء_شجرة_بحث_ثنائية، إنشاء_قائمة، إنشاء_قائمة_مزدوجة، إنشاء_شجرة_بحث_متعددة، إنشاء_كومة، إنشاء_صف، ملء_جدول ، ملء_بنية

بناء الجمل :
إنشاء_قائمة (ق ،[ع1،ع2،...])
إنشاء_قائمة_مزدوجة (قم ،[ع1،ع2،...])
إنشاء_شجرة_بحث_ثنائية (شث ،[ع1،ع2،...])
إنشاء_شجرة_بحث_متعددة (شم ،[ع1،ع2،...])

إنشاء_صف (ص ،[ع1،ع2،...])
إنشاء_كومة (ك ،[ع1،ع2،...])
ملء_بنية (ب ،[ع1،ع2،...])
ملء_جدول (ج ،[ع1،ع2،...])
ع1،ع2، ...تعبيرات من النوع البسيط أو بنى.

مثال:
إنشاء_قائمة (س، [12، 23، 67، ذ، ذ+ر])
إنشاء القائمة ق بالقيم الموجودة بين قوسين مربعين بالترتيب المذكور.
إذا كان ق1 و ق2 قائمتين من الأعداد الصحيحة و قق قائمة من الأقوام، فيمكننا الكتبة:
إنشاء_قائمة (ق1 ، [2 ، 4 ، 67 ، 778]) ؛
إنشاء_قائمة (ق2 ، [12 ، 14 ، 167 ، 1778]) ؛
إنشاء_قائمة (قق ، [ق1 ، ق2]) ؛

دوال قياسية
باقي_قسمة(أ، ب) : ترجع ما تبقى من قسة أ على ب.
أقصى(أ، ب) : يعطي الحد الأقصى بين أ و ب.
أدنى(أ، ب) : يعطي الحد الأدنى بين أ و ب.
أس (أ، ب) : أ أس ب

دوال لتوليد عشوائي
سلسلة_عشوائية (ن) : ينتج سلسلة متكونة من ن محارف.
عدد_عشوائي (ن) : ينتج عددا صحيحا بين 0 و ن.

دالات على السلاسل
محرف_سلسلة (جملة , ر) :
طول(جملة)

تعريف إجراء
إجراء إسم (و1، و2...)
{ كائنات محلية و شاملة و وسطاء }
بداية
{ تعليمات }
نهاية

يتم استدعاء إجراء من خلال التعليمة أنفذ متبوعة بإسم الإجراء و الوسطاء
الوسطاء غير محمية من الإجراء
يجب الإعلان عن إجراء في الوحدة الرئيسية.

تعريف دالة
دالة إسم (و1، و2...) : نوع
{ كائنات محلية و شاملة و وسطاء }
بداية
{ تعليمات }
نهاية

النوع يمكن أن يكون أي .
يتم استخدام دالة مباشرة في تعبير.
الوسطاء غير محمية من الدالة.

يجب الإعلان عن دالة في الوحدة الرئيسية من خلال تحديد نوعها.

يجب أن توجد تخصيص في جسم الدالة من النوع الاسم: = العبارة.

مثال خوارزمية

```
ليكن
    س1، س2 : قوائم؛
    ع، ف : دالة(منطيقي)؛
بداية
    إنشاء_قائمة(س1، [2، 5، 9، 8، 3، 6 ])؛
    إنشاء_قائمة(س2، [12، 5، 19، 8، 3، 6، 2،9])؛
    اكتب( ف(س1، س2) )
نهاية

دالة ع ( س، ك ) : منطيقي
ليكن
    س : قائمة؛ ك : صحيح؛
بداية
    إذا س = عدم :
        ع := خطأ
    وإلا
        إذا قيمة_خلية(س) = ك :
            ع := صواب
        وإلا
            ع := ع(تالي(س)، ك )
        نهاية_إذا
    نهاية_إذا
نهاية

دالة ف ( س1، س2 ) : منطيقي
ليكن
    س1، س2 : قوائم؛
بداية
    إذا س1 = عدم :
        ف := صواب
    وإلا
        إذا لا ع(س2، قيمة_خلية(س1) )
            ف := خطأ
        وإلا
            ف := ف(تالي(س1)، س2)
        نهاية_إذا
    نهاية_إذا
نهاية
```

لغة Z (تراكيب المعطيات)

البنى
الجداول
القوائم
القوائم المزدوجة
الصفوف
الأكوام
الأشجار البحث الثنائي
الأشجار البحث المتعددة
الملفات

البنى

البنية عبارة عن مجموعة من العناصر من أنواع مختلفة .يمكن أن يكون عنصر البنية عدديا أو جدول أحادية البعد من العددية.
يمكن أن تكون البنى ثابتة أو ديناميكية

يمكن أن تكون البنية بسيطة ، أي تتكون فقط من العديديات.
يمكن أن تكون البنية معقدًا ، أي تتكون من عديديات و /أو جداول أحادية البعد من العديديات .

تعريف البنى:
ليكن ‹قم› : [بنية] (نوع1، نوع2،...،نوع-ن،...) [ديناميكي]
‹قم› : قائمة المعرفات مفصولة بفواصل.
نوع-ن إما أن يكون نوعًا عديديا أو جدول ذات بعد واحد من العديديات.

أمثلة:
ب1 : (صحيح، سلسلة) ؛
ب2 : بنية (سلسلة، صحيح، منطيقي) ؛
ب3 : (صحيح، جدول(5) من سلسلة) ديناميكي؛

الجداول

الجدول عبارة عن مجموعة من العناصر المتجانسة.
يمكن أن يكون الجدول بسيط ، أي يتكون فقط من العديديات.
يمكن أن يكون الجدول معقدًا ، أي يتكون من عديديات و /أو جداول ذات بعد واحد من العديديات.
يمكن أن يكون الجدول ثابتًا أو ديناميكيًا.

تعريف الجداول:
ليكن ‹قم› : جدول (بعد1، بعد2،...) من نوع-م من نوع-م ...من نوع-ب [ديناميكي]
‹قم› : قائمة المعرفات مفصولة بفواصل.
نوع-م في{ جدول، كومة، صف، قائمة، قائمة_مزدوجة، شجرة_بحث_ثنائية، شجرة_بحث_متعددة }
نوع-ب هو عددي أو بنية بسيطة.

أمثلة:
ج1 : جدول (5) ديناميكي؛
ج2، ج3 : جداول (3، 8) من سلاسل ؛

القوائم

القائمة المرتبطة هي مجموعة من الخلايا المخصصة ديناميكيا (أي أثناء تنفيذ البرنامج) المرتبطة ببعضها البعض. تحتوي الخلية عادة على حقلين: قيمة وعنوان.
المستخدم هو الذي ينشئ القائمة ، الذي يعدلها عن طريق إضافة أو حذف الخلايا ، الذي يتصفحها بهدف القيام بعملية معينة.
يمكن أن يكون حقل "القيمة "من أي نوع.

تعريف القوائم :
ليكن ‹قم› : قائمة [من نوع-م من نوع-م ...] [من نوع-ب]
‹قم› : قائمة المعرفات مفصولة بفواصل.
نوع-م في { جدول، كومة، صف، قائمة، قائمة_مزدوجة، شجرة_بحث_ثنائية، شجرة_بحث_متعددة }نوع-ب هو عددي أو بنية بسيطة.

أمثلة:
ق1 : قائمة من (سلسلة، صحيح)؛
ق2 : قائمة من كومة من سلسلة ؛
ق3 : قائمة من سلاسل؛

القوائم المزدوجة
القائمة ثنائية الاتجاه هي قائمة مرتبطة متكونة من مجموعة من الخلايا يمكن اجتيازها في كلا الاتجاهين
يحتوي العنصر على ثلاثة حقول :القيمة والعنوان الأيسر والعنوان الأيمن.
يمكن أن يكون حقل "القيمة "من أي نوع.

تعريف القوائم المزدوجة:
ليكن ‹قم› : قائمة_مزدوجة [من نوع-م من نوع-م ...] [من نوع-ب]

‹قم› : قائمة المعرفات مفصولة بفواصل.
نوع-م في { جدول، كومة، صف، قائمة، قائمة_مزدوجة، شجرة_بحث_ثنائية، شجرة_بحث_متعددة}
نوع-ب هو عددي أو بنية بسيطة .

أمثلة:
قم1 : قائمة_مزدوجة من (سلسلة، صحيح)؛
قم2 : قائمة_مزدوجة من كومة من سلسلة ؛
قم3 : قائمة_مزدوجة من سلاسل؛

الصفوف
الصف يطبع طريقة فيفو)ما يدخل أولا يخرج أولا .(إنها مجموعة من العناصر بحيث يتم إدراج أي عنصر جديد في النهاية ويتم حذف أي عنصر في البداية.
يمكن أن يكون العنصر من أي نوع.

تعريف الصفوف :
ليكن ‹قم› : صف [من نوع-م من نوع-م ...] [من نوع-ب]

‹قم› : قائمة المعرفات مفصولة بفواصل.
نوع-م في { جدول، كومة، صف، قائمة، قائمة_مزدوجة، شجرة_بحث_ثنائية، شجرة_بحث_متعددة }
نوع-ب هو عددي أو بنية بسيطة.

أمثلة:
ص1 : صف من (سلسلة، صحيح)؛
ص2 : صف من كومة من سلسلة ؛
ص3 : صف من سلاسل؛

الأكوام

كومة يطيع مبدأ 'ليفو':(ما يدخل اخيرا يخرج أول). إنها مجموعة من العناصر بحيث يتم إدراج أي عنصر جديد في البداية ويتم إزالة أي عنصر في البداية.
يمكن أن يكون العنصر من أي نوع.

تعريف الأكوام:
ليكن ‹قم› : كومة [من نوع-م من نوع-م ...] [من نوع-ب]

‹قم› : قائمة المعرفات مفصولة بفواصل.
نوع-م في { جدول، كومة، صف، قائمة، قائمة_مزدوجة، شجرة_بحث_ثنائية، شجرة_بحث_متعددة }نوع-ب هو عددي أو بنية بسيطة.

أمثلة:
ك1 : كومة من (سلسلة، صحيح)؛
ك2 : كومة من كومة من سلسلة ؛
ك3 : كومة من سلاسل؛

الأشجار البحث الثنائي

تمثل شجرة البحث الثنائية مجموعة من البيانات القابلة للمقارنة .جميع البيانات الموجودة في الشجرة الفرعية اليسرى لأي عقدة بالمعلومات س أقل من س ،
جميع البيانات الموجودة في الشجرة الفرعية اليمنى لأي عقدة بالمعلومات س أكبر من س.
بشكل حدسي ، تتكون العقدة من 3حقول :المعلومات وحقلي عنوان يوفران الأسلاك اليمنى واليسرى
شجرة البحث الثنائي هي تركيب المعطيات ديناميكية بشكل عام.

تعريف الأشجار البحث الثنائي:
ليكن ‹قم› : شجرة_بحث_ثنائية [من نوع-م من نوع-م ...] [من نوع-ب]

‹قم› : قائمة المعرفات مفصولة بفواصل.
نوع-م في { جدول، كومة، صف، قائمة، قائمة_مزدوجة، شجرة_بحث_ثنائية، شجرة_بحث_متعددة }
نوع-ب هو عددي أو بنية بسيطة.

أمثلة:
ش1 : شجرة_بحث_ثنائية من (سلسلة، صحيح)؛
ش2 : شجرة_بحث_ثنائية من كومة من سلسلة ؛
ش3 : شجرة_بحث_ثنائية من سلاسل؛

الأشجار البحث المتعددة

تسمح لك شجرة البحث المتعددة بتمثيل مجموعة من البيانات القابلة للمقارنة .هذا هو تعميم شجرة البحث الثنائية .بدلا من الامتلاك على قيمة واحدة وحقلي عنوان ،
لدينا ص عناوين و (ص-1) قيم ، وفي هذه الحالة يقال إن شجرة البحث المتعددة ذات ترتيب ص.
بشكل حدسي ، تحتوي العقدة على النموذج التالي: (عنوان 1, قيمة 1, عنوان 2, قيمة 2,..... قيمة ص-1 ، عنوان ص).
جميع بيانات الشجرة الفرعية التي لها جذر عنوان 1 أقل من قيمة 1 ،
وجميع البيانات الموجودة في الشجرة الفرعية التي لها جذر عنوان 2 أكبر من قيمة 1 وأقل من قيمة 2 ، إلخ.
يمكن أن يكون العنصر(عقدة) من أي نوع.

تعريف الأشجار البحث المتعددة:
ليكن ‹قم› : شجرة_بحث_متعددة[من نوع-م من نوع-م ...] [من نوع-ب]

‹قم› : قائمة المعرفات مفصولة بفواصل.
نوع-م في { جدول، كومة، صف، قائمة، قائمة_مزدوجة، شجرة_بحث_ثنائية، شجرة_بحث_متعددة}نوع-ب هو عددي أو بنية بسيطة.

أمثلة:
ش1 : شجرة_بحث_متعددة(4) من (سلسلة، صحيح)؛

ش2 : شجرة_بحث_متعددة(2) من كومة من سلسلة ؛
ش3 : شجرة_بحث_متعددة(3) من سلاسل؛

الملفات

الملف عبارة عن مجموعة من البنى ، يتم تخزينها عادة على القرص. يمكن أن تكون البنى سجلات (مستوى المستخدم) أو كتل (مستوى المصمم).
يحتوي الملف على بنية معينة (رأس الملف) ضرورية لتصميم هياكل الملفات.

تعريف الملفات:
ليكن م : ملف من نوع-م [صدر_ملف (نوع-ب، نوع-ب،...)] مخزن ‹قم›

‹قم› : قائمة المعرفات مفصولة بفواصل.

تعريف الملف له 3 أجزاء :
** الجزء الأول (ملف) يحدد طبيعة عناصر الملف.
عنصر من ملف يمكن أن يكون:
-عديديا وهذا يعني من نوع بسيط (صحيح-منطقي-محرف-سلسلة)،
- جدول أحادية البعد من العددية
- بنية يمكن أن يحتوي على عددي أو جداول أحادية البعد من عددي.
** الجزء الثاني (صدر_ملف) يحدد خصائص الملف عن طريق تحديد نوع كل الخصائص. يستخدم هذا الجزء بشكل أساسي لإنشاء تركيب ملفات المستخدم ويستخدم لتخزين جميع المعلومات المفيدة لاستغلال الملف.
** يحدد الجزء الثالث (مخزن) المتغيرات المستخدمة في عمليات القراءة والكتابة.

أمثلة:
م1 : ملف من سلاسل مخزن ك1، ك2؛
م2 : ملف من جدول(5) من صحيح مخزن ك ؛
م3 : ملف من (صحيح، جدول(3) من محرف) صدر_ملف(صحيح،صحيح) مخزن ك3 ؛
م4 : ملف من محرف صدر_ملف (صحيح، سلسلة، منطيقي) مخزن ك4 ؛

الآلات المجردة

مفهوم الآلة المجردة
جداول
بنى
قوائم
قوائم ثنائية الاتجاه
أكوام
صفوف
أشجار البحث الثنائي
أشجار البحث المتعددة
ملفات

مفهوم الآلة المجردة

في كل تركيب المعطيات ، يتم تعريف الآلة المجردة بمجموعة العمليات الخاصة بها.

ليكن ‹قم› : ‹آلة مجردة›
‹قم› : مجموعة من المعرفين

أمثلة:
ق1، ق2 : قوائم
ص : صف
ج : جدول(10,60)
قكج : قائمة من أكوام من جداول(5)

آلة مجردة على الجداول

عنصر(ج [ب، ج، ...]) :
الحصول على العنصر ج[ب، ع، ...] للجدول ج.
ضع_عنصر(ج[ب، ج، ...], ق) :
تخصيص القيمة ق إلى العنصر ج[ب، ع، ...] للجدول ج.
احجز_جدول(ج) :
حجز جدول.
سرح_جدول(ج) :
تسريح جدول.

آلة مجردة على البنى

حقل(ب, ر) :
الحصول على الحقل رقم ر.
ضع_حقل(ب, ر, ق) :
تخصيص القيمة ق إلى الحقل رقم ر.
احجز_بنية(ب) :
حجز بنية
سرح_بنية(ب) :
تسريح بنية

آلة مجردة على القوائم

احجز_خلية(ق) :

حجز خلية مع المعطية ق و إرجاع عنوان العقدة.
سرح(ق) :
تسريح الخلية بعنوان ق.
تالي(ق) :
الحصول على الحقل "عنوان يمين" للعقدة المشار إليها بواسطة ق.
قيمة_خلية(ق) :
الحصول على الحقل "قيمة" للعقدة المشار إليها بواسطة ق.
ضع_عنوان(ق, ل) :
تخصيص العنوان ل إلى الحقل "عنوان" للعقدة المشار إليها بواسطة ق.
ضع_قيمة_خلية(ق, قيمة) :
تخصيص القيمة قيمة إلى الحقل "قيمة "للعقدة المشار إليها بواسطة ق.

آلة مجردة على القوائم ثنائية الإتجاه

احجز_خلية(ق) :
حجز خلية مع المعطية ق و إرجاع عنوان العقدة.
سرح(ق) :
تسريح الخلية بعنوان ق.
تالي(ق) :
الحصول على الحقل "عنوان يمين" للعقدة المشار إليها بواسطة ق.
سابق(ق) :
الحصول على الحقل "عنوان يسار" للعقدة المشار إليها بواسطة ق.
قيمة_خلية(ق) :
الحصول على الحقل "قيمة" للعقدة المشار إليها بواسطة ق.
ضع_قيمة_خلية(ق, قيمة) :
تخصيص القيمة قيمة إلى الحقل "قيمة" للعقدة المشار إليها بواسطة ع.
ضع_عنوان_يمين(ق, ل) :
تخصيص العنوان ل إلى الحقل "عنوان يمين" للعقدة المشار إليها بواسطة ق.
ضع_عنوان_يسار(ق, ل) :
تخصيص العنوان ل إلى الحقل "عنوان يسار "للعقدة المشار إليها بواسطة ق.

آلة مجردة على الصفوف

احجز_صف(ص) :(
حجز صف فارغ.
صف_فارغ(ص) :
تحقق مما كان الصف فارغ.
ضف_صف(ص, ق) :
إظافة القيمة ق الى ذيل الصف ص.
نزع_صف(ص, ق) :
انسحاب القيمة الموجودة فى رأس الصف ص و جعلها في المتغير ق .

آلة مجردة على الأكوام

احجز_كومة(ك) :
حجز كومة فارغة.
كومة_فارغة(ك) :
تحقق مما كانت الكومة فارغة.
ضف_كومة(ك, ق) :
إظافة القيمة ق الى قمة الكومة ك.
نزع_كومة(ك, ق) :
انسحاب القيمة الموجودة فى قمة الكومة ك و جعلها في المتغير ق.

آلة مجردة على الأشجار البحث الثنائي

احجز_عقدة(ق) :

حجز عقدة مع المعطية ق و إرجاع عنوان العقدة. تخصيص "لا شيء" إلى الحقول الأخرى للعقدة.

سرح_عقدة(ع) :

تسريح العقدة بعنوان ع

إبن_يسار(ع) :

الحصول على الحقل "إبن_يسار" للعقدة المشار إليها بواسطة ع .

إبن_يمين(ع) :

الحصول على الحقل "إبن_يمين" للعقدة المشار إليها بواسطة ع .

أب(ع) :

الحصول على الحقل "أب" للعقدة المشار إليها بواسطة ع.

قيمة_عقدة(ع) :

الحصول على الحقل "قيمة" للعقدة المشار إليها بواسطة ع.

ضع_إبن_يسار(ع, ل) :

تخصيص العنوان ل إلى الحقل "إبن_يسار" للعقدة المشار إليها بواسطة ع.

ضع_إبن_يمين(ع, ل) :

تخصيص العنوان ل إلى الحقل "إبن_يمين" للعقدة المشار إليها بواسطة ع.

ضع_أب(ع, ل) :

تخصيص العنوان ل إلى الحقل "أب" للعقدة المشار إليها بواسطة ع.

ضع_قيمة_عقدة(ع, ق) :

تخصيص القيمة ق إلى الحقل "قيمة "للعقدة المشار إليها بواسطة ع.

آلة مجردة على الأشجار البحث المتعددة

احجز_عقدة(ق) :

حجز عقدة مع المعطية ق و إرجاع عنوان العقدة. تخصيص "لا شيء" إلى الحقول الأخرى للعقدة.

سرح_عقدة(ع) :

تسريح العقدة بعنوان ع

إبن(ع, ر) :

الحصول على العنوان (من رتبة ر) من عقدة المشار إليها بواسطة ع

أب(ع) :

الحصول على الحقل أب للعقدة المشار إليها بواسطة ع

قيمة_عقدة_شم(ع, ر) :

الحصول على المعطية (من رتبة ر) من عقدة المشار إليها بواسطة ع

ضع_إبن(ع, ر, ل) :

تخصيص العنوان ل كالإبن (من رتبة ر) من عقدة المشار إليها بواسطة ع

ضع_أب(ع, ل) :

تخصيص العنوان ل إلى الحقل أب للعقدة المشار إليها بواسطة ع.

ضع_قيمة_عقدة_شم(ع, ر, ق) :

تخصيص القيمة ق كالمعطية (من رتبة ر) من عقدة المشار إليها بواسطة س

درجة(ع) :

عدد البيانات المخزنة في العقدة المشار إليها بواسطة ع.

ضع_درجة(ع, ر) :

تخصيص القيمة ر إلى الحقل درجة للعقدة المشار إليها بواسطة ع

آلة مجردة على الملفات

افتح_ملف (م, مف, طارقة) :

فتح الملف المنطقي و ربطه بالملف الفعلي تحديد ما إذا كان الملف هو جديد ('ن') أو القديم ('أ').

أغلق_ملف(م) :

إغلاق الملف.

اقرأ_تسلسلا_ملف (م, ق) :

القراءة في المتغير ق البنية الموجودة في الموضع الحالي.

اكتب_تسلسلا_ملف (م, ق) :

كتابة البنية ق في الوضع الحالي.

اقرأ_مباشرة_ملف (م, ق, ر) :

قراءة البنية ر من الملف.

اكتب_مباشرة_ملف (م, ق, ر) :

كتابة البنية ق في المركز ر

اظف_ملف (م, ق) :

إظافة البنية ق في نهاية الملف

نهاية_ملف (م) :

دالة مساوية لصواب إذا واجهت نهاية الملف ، خطأ خلاف ذلك.

احجز_كتلة_ملف(م) :

موقف كتلة (أو بنية) التي يمكننا الكتابة فيها.

صدر_ملف (م, ر) :

استعادة خاصية ر من الملف.

ضع_صدر_ملف(م, ر, ق) :

تعيين ق كخاصية ر من الملف.

الانتقال من Z إلى PASCAL

تعريف
تعبيرات
تخصيص
تكرار المشروط
تكرار المعدود
الاشتراط
قراءة
كتابة
إجراء
دالة
دوال قياسية
خوارزمية

تعريف المتغيرات

يتم إعلان المتغيرات في PASCAL بواسطة

<Li> : Type ;

حيث <Li>يدل على قائمة المعرفات .

ليكن يترجم إلى **VAR.**

الكائنات بسيطة

مكافئات الكائنات Z إلى PASCAL:

Z	PASCAL
صحيح	**INTEGER**
منطيقي	**BOOLEAN**
محرف	**CHAR**
سلسلة	**STRING**

التعبيرات

يتم تضمين قواعد تعبيرات Z في قواعد اللغة PASCAL.

التخصيص

نفس بناء الجملة

التكرار المشروط

-------Z---------

مادام شرط :

تعليمات

نهاية_مادام

يترجم إلى

-------PASCAL------

```
WHILE ( Cond ) DO
  BEGIN
    Statements
  END
```

التكرار المعدود

----------Z----------

لكل :=ع1، ع2 [،ع3]

تعليمات

نهاية_لكل

إذا ع 3غير موجود أو يساوي 1

يترجم إلى

--------PASCAL---------

```
FOR V:= Exp1 TO Exp2 DO
   BEGIN
      Instructions
   END
```

إذا ع3 # 1 :

----------PASCAL------

```
V := Exp1;
WHILE ( V <= Exp2 ) DO
   BEGIN
     Statements ;
    V := V + Exp3
   END;
```

الاشتراط

----------Z----------

إذا شرط :
تعليمات
[وإلا
تعليمات]
نهاية_إذا

يترجم إلى

-------PASCAL---------

```
IF Cond
THEN
   BEGIN
     Statements
   END
[ELSE
   BEGIN
    Statements
   END]
```

القراءة

--------Z--------

اقرأ(م1، م2،...)

يترجم إلى

------PASCAL------

```
READLN(V1, V2, ...)
```

كتابة

----------Z----------
اكتب(ع1،ع2،...)

يترجم إلى

--------PASCAL---------

```
WRITELN(Exp1, Exp2, ...)
```

إجراء

----------Z----------
إجراء **إسم (و1، و2...)**
ليكن
تعريف الكائنات المحلية و الوسطاء
بداية
تعليمات
نهاية

يترجم إلى

----------PASCAL----------

```
PROCEDURE Name ( VAR P1: typ; VAR P2:typ, ...);
  VAR
      Declaration of local objects and parameters
  BEGIN
      Statements
  END
```

دالة

----------Z----------
دالة إسم (و1، و2...) : نوع
ليكن
تعريف الكائنات المحلية و الوسطاء
بداية
تعليمات
نهاية

يترجم إلى

----------PASCAL----------

```
FUNCTION Name ( VAR P1: typ; VAR P2:typ, ...) : Type;
  VAR
      Declaration of local objects and parameters
  BEGIN
      Statements
  END
```

دوال قياسية

باقي_قسمة(أ، ب)

```
FUNCTION Mod (a, b : INTEGER) : INTEGER;
  BEGIN
    Mod := a Mod b
  END;
```

أدنى(أ، ب)

```
FUNCTION Min (a, b: INTEGER) : INTEGER;
  BEGIN
    Min := a; IF b < a THEN Min := b;
  END;
```

أقصى(أ، ب)

```
FUNCTION Max (a, b: INTEGER) : INTEGER;
  BEGIN
    Max := a; IF b > a THEN Max := b;
  END;
```

أس(أ، ب)

```
FUNCTION Exp (a, b: INTEGER) : INTEGER;
  VAR I : INTEGER;
  BEGIN
    Exp := 1;
    FOR I:= 1 TO b DO Exp := Exp * a
  END;
```

عدد_عشوائي (ن)

```
FUNCTION Randnumer (N: INTEGER) : INTEGER;
  BEGIN
    Randnumer := Random( N );
  END;
```

سلسلة_عشوائية (ن)

```
FUNCTION Randstring(N: INTEGER) : STRING;
  VAR
    K : BYTE;
    Chaine : STRING;
  BEGIN
    Chaine := '';
    FOR K:=1 TO N DO
      CASE Random(2) OF
      0 : Chaine := Chaine + CHR(97+Random(26) ) ;
```

```
    1 : Chaine := Chaine + CHR(65+Random(26) )
    END;
  Randstring := Chaine;
END;
```

طول(جملة)

```
FUNCTION Stringlength(C : STRING): INTEGER;
 BEGIN
  Stringlength := length (C)
 END;
```

خوارزمية

----------Z----------

ليكن

تعريف الكائنات المحلية و الشاملة

الإعلان عن الوحدات

بداية

تعليمات

نهاية

وحدة 1

....

وحدة ن

يترجم إلى

----------PASCAL----------

```
PROGRAM  Pascal;
  VAR

  {Local and static objects }
  { Definition of modules }
  Module 1
  Module 2
  ...
  Module n

  BEGIN
      Statements
  END.
```

الآلات المجردة في باسكال (PASCAL)

جداول
بنى
قوائم
قوائم مزدوجة
أكوام
صفوف
أشجار البحث الثنائية
أشجارالبحث المتعددة
ملفات

تطبيق الجداول في PASCAL
ليكن ج : جدول (5 ، 10) ؛

LET Tab : ARRAY (5 , 10) ;

```
{ -Implementation- : ARRAY OF INTEGERS}

{ Arrays }
TYPE
Typeelem_V5_10I = INTEGER;
Typetab_V5_10I = ARRAY[1..5,1..10] OF Typeelem_V5_10I;
Typevect_V5_10I = ^ Typetab_V5_10I;

FUNCTION Element_V5_10I ( V:Typevect_V5_10I; I1 , I2 : INTEGER ) : Typeelem_V5_10I ;
BEGIN
Element_V5_10I := V^[I1 ,I2];
END;

PROCEDURE Ass_element_V5_10I ( V :Typevect_V5_10I; I1 , I2 :INTEGER; Val : Typeelem_V5_10I );
BEGIN
V^[I1 ,I2] := Val;
END;


{ Declaration part of variables }
VAR
Tab : Typevect_V5_10I;
{ Body of main program }
BEGIN
NEW(Tab);
END.
```

تطبيق البنى في PASCAL
ليكن س : بنية (سلسلة ، صحيح) ؛
LET S : STRUCTURE (STRING , INTEGER) ;

```
TYPE Typestring = STRING[255];

{ Structures }
TYPE
```

```
Type1_TSI = Typestring;
Type2_TSI = INTEGER;
Typestr_TSI = ^ Type_TSI ;
Type_TSI = record
Field1 : Type1_TSI ;
Field2 : Type2_TSI ;
END;

FUNCTION Struct1_TSI ( S: Typestr_TSI) : Type1_TSI;
BEGIN
STRUCT1_TSI := S^.Field1;
END;

FUNCTION Struct2_TSI ( S: Typestr_TSI) : Type2_TSI;
BEGIN
STRUCT2_TSI := S^.Field2;
END;

PROCEDURE Ass_struct1_TSI ( S: Typestr_TSI; Val :Type1_TSI );
BEGIN
S^.Field1 := Val;
END;

PROCEDURE Ass_struct2_TSI ( S: Typestr_TSI; Val :Type2_TSI );
BEGIN
S^.Field2 := Val;
END;

{ Declaration part of variables }
VAR
S : Typestr_TSI;
{ Body of main program }
BEGIN
NEW(S);
END.
```

تطبيق القوائم في PASCAL

ليكن ق :قائمة ؛

LET L : LIST ;

```
{ -Implementation- : LIST Of INTEGERS}

{ Linked lists }
TYPE
Typeelem_LI = INTEGER;
Pointer_LI = ^Maillon_LI; { type du champ 'Adresse' }
Maillon_LI = RECORD
Val : Typeelem_LI;
```

```
Next : Pointer_LI
END;

PROCEDURE Allocate_cell_LI ( VAR P : Pointer_LI ) ;
BEGIN NEW(P) END;

PROCEDURE Free_LI ( P : Pointer_LI ) ;
BEGIN DISPOSE(P) END;

PROCEDURE Ass_val_LI(P : Pointer_LI; Val : Typeelem_LI );
BEGIN P^.Val := Val END;

FUNCTION Cell_value_LI (P : Pointer_LI) : Typeelem_LI;
BEGIN Cell_value_LI := P^.Val END;

FUNCTION Next_LI( P : Pointer_LI) : Pointer_LI;
BEGIN Next_LI := P^.Next END;

PROCEDURE Ass_adr_LI( P, Q : Pointer_LI ) ;
BEGIN P^.Next := Q END;

{ Declaration part of variables }
VAR
L : Pointeur_LI;
{ Body of main program }
BEGIN
END.
```

تطبيق القوائم مزدوجة في PASCAL
ليكن ق :قائمة_مزدوجة ؛

LET L : BILIST ;

```
{ -Implementation- : BIDIRECTIONAL LIST OF INTEGERS}

{ Bidirectional linked lists }
TYPE
Typeelem_RI = INTEGER;
Pointer_RI = ^Maillon_RI; { type du champ 'Adresse' }
Maillon_RI = RECORD
Val : Typeelem_RI;
Next : Pointer_RI;
Prev : Pointer_RI
END;

PROCEDURE Allocate_cell_RI ( VAR P : Pointer_RI ) ;
BEGIN NEW(P) END;

PROCEDURE Free_RI ( P : Pointer_RI ) ;
```

```
BEGIN DISPOSE(P) END;

PROCEDURE Ass_val_RI (P : Pointer_RI; Val : Typeelem_RI );
BEGIN P^.Val := Val END;

FUNCTION Cell_value_RI (P : Pointer_RI) : Typeelem_RI;
BEGIN Cell_value_RI := P^.Val END;

FUNCTION Next_RI( P : Pointer_RI) : Pointer_RI;
BEGIN Next_RI := P^.Next END;

FUNCTION Previous_RI( P : Pointer_RI) : Pointer_RI;
BEGIN Previous_RI := P^.Prev END;

PROCEDURE Ass_r_adr_RI( P, Q : Pointer_RI ) ;
BEGIN P^.Next := Q END;

PROCEDURE Ass_l_adr_RI( P, Q : Pointer_RI ) ;
BEGIN P^.Prev := Q END;
{ Declaration part of variables }
VAR
Lb : Pointeur_RI;
{ Body of main program }
BEGIN
END.
```

تطبيق الأشجار البحث الثنائية في PASCAL

ليكن ا :شجرة_بحث_ثنائية ؛

LET A : BST ;

```
{ -Implementation- : BINARY SEARCH TREE OF INTEGERS}

{ Binary search trees }
TYPE
Typeelem_AI = INTEGER;
Pointer_AI = ^Noeud_AI;
Noeud_AI = RECORD
Element : Typeelem_AI;
Lc, Rc, Parent : Pointer_AI ;
END;

FUNCTION Node_value_AI(P : Pointer_AI) : Typeelem_AI;
BEGIN Node_value_AI := P^.Element END;

FUNCTION Lc_AI( P : Pointer_AI) : Pointer_AI;
BEGIN Lc_AI := P^.Lc END;

FUNCTION Rc_AI( P : Pointer_AI) : Pointer_AI;
```

```
BEGIN Rc_AI := P^.Rc END;

FUNCTION Parent_AI( P : Pointer_AI) : Pointer_AI;
BEGIN Parent_AI := P^.Parent END;

PROCEDURE Ass_node_val_AI ( P : Pointer_AI; Val : Typeelem_AI);
BEGIN P^.Element := Val END;

PROCEDURE Ass_lc_AI( P : Pointer_AI; Q : Pointer_AI);
BEGIN P^.Lc := Q END;

PROCEDURE Ass_rc_AI( P : Pointer_AI; Q : Pointer_AI);
BEGIN P^.Rc := Q END;

PROCEDURE Ass_parent_AI( P : Pointer_AI; Q : Pointer_AI);
BEGIN P^.Parent := Q END;

PROCEDURE Allocate_node_AI( VAR P : Pointer_AI) ;
BEGIN
NEW ( P ) ;
P^.Lc := Nil;
P^.Rc := Nil;
P^.Parent := Nil;
END;

PROCEDURE Free_node_AI( P : Pointer_AI);
BEGIN
DISPOSE ( P )
END;
{ Declaration part of variables }
VAR
A : Pointeur_AI;
{ Body of main program }
BEGIN
END.
```

تطبيق الأشجار البحث المتعددة في PASCAL

ليكن ش :شجرة_بحث_متعددة (4)؛

LET M : MST (4) ;

```
 { -Implementation- : M-ARY SEARCH TREE OF INTEGERS}

{ M-ary search trees }
TYPE
Typeelem_M4I = INTEGER;
Pointer_M4I = ^Noeud_M4I;
Noeud_M4I = RECORD
```

```
Infor : ARRAY[1..4] of Typeelem_M4I;
Child : ARRAY[1..4] of Pointer_M4I;
Degree : Byte ;
Parent : Pointer_M4I
END;

FUNCTION Node_value_mst_M4I(P : Pointer_M4I; I: INTEGER) : Typeelem_M4I;
BEGIN Node_value_mst_M4I := P^.Infor[I] END;

FUNCTION Child_M4I( P : Pointer_M4I; I : INTEGER) : Pointer_M4I;
BEGIN Child_M4I := P^.Child[I] END;

FUNCTION Parent_M4I( P : Pointer_M4I) : Pointer_M4I;
BEGIN Parent_M4I := P^.Parent END;

PROCEDURE Ass_node_val_mst_M4I ( P : Pointer_M4I; I:INTEGER; Val : Typeelem_M4I);
BEGIN P^.Infor[I] := Val END;


PROCEDURE Ass_child_M4I( P : Pointer_M4I; I:INTEGER; Q : Pointer_M4I);
BEGIN P^.Child[I] := Q END;

PROCEDURE Aff_parent_M4I( P : Pointer_M4I; Q : Pointer_M4I);
BEGIN P^.Parent := Q END;

PROCEDURE Allocate_node_M4I( VAR P : Pointer_M4I ) ;
VAR
I : BYTE;
BEGIN
NEW ( P ) ;
For I:=1 TO 4 Do P^.Child[I] := NIL;
P^.degree := 0
END;

FUNCTION Degree_M4I ( P : Pointer_M4I ) : BYTE;
BEGIN
Degree_M4I := P^.Degree
END;

PROCEDURE Aff_Degree_M4I ( VAR P : Pointer_M4I; N : BYTE);
BEGIN
P^.Degree := N
END;

PROCEDURE Free_node_M4I( P : Pointer_M4I);
BEGIN
DISPOSE ( P )
END;

{ Declaration part of variables }
VAR
```

```
M : Pointeur_M4I;
{ Body of main program }
BEGIN
END.
```

تطبيق الأكوام في PASCAL

ليكن ك :كومة ؛

LET P : STACK ;

```
{ -Implementation- : STACK OF INTEGERS}

{ Stacks }
TYPE
Typeelem_PI = INTEGER; { Any type }
Pointer_PI = ^Maillon_PI ;
Maillon_PI = RECORD
Valeur : Typeelem_PI;
Next : Pointer_PI
END;

PROCEDURE Createstack_PI( VAR P : Pointer_PI );
BEGIN
P := NIL;
END;

FUNCTION Empty_stack_PI ( P : Pointer_PI ) : BOOLEAN;
BEGIN
Empty_stack_PI := ( P = NIL )
END;

PROCEDURE Push_PI ( VAR P : Pointer_PI; Val : Typeelem_PI );
VAR
Q : Pointer_PI;
BEGIN
NEW(Q);
Q^.Valeur := Val;
Q^.Next := P;
P := Q;
END;

PROCEDURE Pop_PI ( VAR P : Pointer_PI; VAR V :Typeelem_PI );
VAR Save : Pointer_PI;
BEGIN
IF NOT Empty_stack_PI (P)
THEN
BEGIN
V := P^.Valeur;
Save := P;
```

```
P := P^.Next;
DISPOSE(Save);
END
ELSE WRITELN('Pile Vide');
END;

{ Declaration part of variables }
VAR
P : Pointeur_PI;
{ Body of main program }
BEGIN
END.
```

تطبيق الصفوف في PASCAL

ليكن ص :صف ؛

LET F : QUEUE ;

```
{ -Implementation- : QUEUE OF INTEGERS}

{ Queues }
TYPE
Typeelem_FI = INTEGER;
Ptliste_FI = ^Maillon_FI;
Maillon_FI = RECORD
Val : Typeelem_FI;
Next : Ptliste_FI
END;

Pointer_FI = ^ Filedattente_FI;
Filedattente_FI = RECORD
Tete, Queue : Ptliste_FI
END;

PROCEDURE Createqueue_FI (VAR Fil : Pointer_FI );
BEGIN
New (Fil);
Fil^.Tete := NIL ;
Fil^.Queue := Nil
END;

FUNCTION Empty_queue_FI (Fil : Pointer_FI) : BOOLEAN;
BEGIN Empty_queue_FI := Fil^.Tete = NIL END;

PROCEDURE Enqueue_FI (VAR Fil : Pointer_FI; Val : Typeelem_FI );
VAR
P : Ptliste_FI;
BEGIN
NEW(P);
```

```
P^.Val := Val;
P^.Next := NIL;
IF NOT Empty_queue_FI(Fil)
THEN Fil^.Queue^.Next := P
ELSE Fil^.Tete := P;
Fil^.Queue := P;
END;

PROCEDURE Dequeue_FI (VAR Fil : Pointer_FI ; VAR Val : Typeelem_FI );
BEGIN
IF NOT Empty_queue_FI(Fil)
THEN
BEGIN
Val := Fil^.Tete^.Val;
Fil^.Tete := Fil^.Tete^.Next;
END
ELSE WRITELN(' File Vide ');
END;


{ Declaration part of variables }
VAR
F : Pointeur_FI;
{ Body of main program }
BEGIN
END.
```

تطبيق الملفات في PASCAL

ليكن م : ملف من (سلاسل ، صحيح) صدر_ملف (صحيح ، صحيح) مخزن ب1 ؛

LET F : FILE OF (STRINGS , INTEGER) HEADER (INTEGER , INTEGER) BUFFER B1 ;

```
{ -Implementation- : FILE }

{ Managing open files }

TYPE
_Ptr_Noeud = ^_Noeud;
_Noeud = RECORD
Var_fich : Thandle;
Nom_fich : string;
Save_pos : Longint;
Next : _Ptr_Noeud
END;

VAR
_Stack_ouverts : _Ptr_Noeud = NIL;

FUNCTION _Ouvert (Fp : String) : _Ptr_Noeud;
```

```
VAR
P : _Ptr_Noeud;
Found : boolean;
BEGIN
P := _Stack_Open; Found := False ;
WHILE (P <> NIL) AND NOT Found DO
IF P^.Nom_Fich = Fp
THEN Found := True
ELSE P := P^.Next;
_Ouvert := P;
END;

PROCEDURE _Push_ouvert ( Fp : string; VAR Fl: Thandle);
VAR
P : _Ptr_Noeud ;
BEGIN
New(P);
P^.Nom_fich := Fp;
P^.Var_fich := Fl;
P^.Next := _Stack_Open;
_Stack_Open := P
END ;

FUNCTION _Pop_ouvert ( Fl : Thandle) : String;
VAR
P, Prev : _Ptr_Noeud ;
BEGIN
P:= _Stack_Open;
Prev := Nil;
WHILE P^.Var_fich <> Fl DO
BEGIN Prev := P ; P := P^.Next END;
_Pop_ouvert := P^.Nom_fich ;
IF Prev <> NIL
THEN Prev^.Next := P^.Next
ELSE _Stack_Open := P^.Next;
Dispose (P);
END;


{ Files }
TYPE
{ Types of block fields }
Typefield1_SIEII = Typestring;
Typefield2_SIEII = INTEGER;

{ Type of file block structure }
Typestruct_SIEII = ^ Typestruct_SIEII_ ;
Typestruct_SIEII_ = RECORD
Field1 : Typefield1_SIEII ;
Field2 : Typefield2_SIEII ;
END;
```

```
{ Type of File data block }
Typestruct_SIEII_Buf = RECORD
Field1 : Typefield1_SIEII ;
Field2 : Typefield2_SIEII ;
END;


{ Types of header fields }
Typeentete1_SIEII = INTEGER;
Typeentete2_SIEII = INTEGER;

{ Type of file feature block }
Typestruct_SIEII_entete = RECORD
Entete1 : Typeentete1_SIEII ;
Entete2 : Typeentete2_SIEII ;
END;


{ Declaration of header block }

VAR
Buf_caract_SIEII : Typestruct_SIEII_entete ;

{ Operations on files }

PROCEDURE Open_SIEII (VAR Fl : Thandle ; Fp, Mode : STRING );
VAR
P : _Ptr_Noeud;
BEGIN
P := _Open (Fp);
IF P <> NIL
THEN
BEGIN
{ Save the current position of the file and close it }
P^.Save_pos :=FILESEEK(P^.Var_fich, 0, 1);
FILESEEK(P^.Var_fich,0,0);
FILEWRITE(P^.Var_fich, Buf_caract_SIEII, sizeof(Buf_caract_SIEII) );
FILECLOSE (P^.Var_fich);
END;

{ Open or re open the file }
IF Mode = 'A'
THEN
BEGIN
Fl:=FILEOPEN(Fp,fmOpenReadWrite);
FILEREAD(Fl, Buf_caract_SIEII, sizeof(Buf_caract_SIEII) )
END
ELSE
BEGIN
Fl:=FILECREATE(Fp);
```

```
FILEWRITE(Fl, Buf_caract_SIEII, sizeof(Buf_caract_SIEII) )
END ;
_Push_Open(Fp, Fl);
END;

PROCEDURE Close_SIEII ( VAR Fl : Thandle);
VAR
P : _Ptr_Noeud;
Fp : String;
BEGIN
Fp := _Pop_Open(Fl);

FILESEEK(Fl,0, 0);
FILEWRITE(Fl, Buf_caract_SIEII, sizeof(Buf_caract_SIEII) );
FILECLOSE(Fl);

{ Is there a file open with the same name? }
{ If yes, open it again at the saved position }
P := _Open (Fp);
IF P <> NIL
THEN
BEGIN
Fl:=FILEOPEN(P^.Nom_fich,fmOpenReadWrite);
FILEREAD(Fl, Buf_caract_SIEII, sizeof(Buf_caract_SIEII) );
FILESEEK(Fl, P^.Save_pos, 0)
END;
END;

FUNCTION Entete1_SIEII( VAR Fl : Thandle): Typeentete1_SIEII;
BEGIN
Entete1_SIEII := Buf_caract_SIEII.Entete1;
END;

FUNCTION Entete2_SIEII( VAR Fl : Thandle): Typeentete2_SIEII;
BEGIN
Entete2_SIEII := Buf_caract_SIEII.Entete2;
END;

PROCEDURE Aff_entete1_SIEII ( VAR Fl: Thandle; VAL : Typeentete1_SIEII);
BEGIN
Buf_caract_SIEII.Entete1 := VAL
END;

PROCEDURE Aff_entete2_SIEII ( VAR Fl: Thandle; VAL : Typeentete2_SIEII);
BEGIN
Buf_caract_SIEII.Entete2 := VAL
END;

PROCEDURE Writeseq_SIEII ( VAR Fl: Thandle; Buf : Typestruct_SIEII );
VAR
Buffer : Typestruct_SIEII_Buf ;
```

```
I : Integer;
BEGIN
Buffer.Field1:= Buf^.Field1;
Buffer.Field2:= Buf^.Field2;
FILEWRITE(Fl, Buffer, Sizeof(Buffer))
END;

PROCEDURE Writedir_SIEII ( VAR Fl: Thandle; Buf : Typestruct_SIEII; N: INTEGER );
VAR
Buffer : Typestruct_SIEII_Buf ;
I : Integer;
BEGIN
Buffer.Field1:= Buf^.Field1;
Buffer.Field2:= Buf^.Field2;
FILESEEK(Fl, Sizeof(Buf_caract_SIEII) + (N-1)*Sizeof(Buffer ), 0);
FILEWRITE(Fl, Buffer, Sizeof(Buffer))
END;

PROCEDURE Readseq_SIEII ( VAR Fl: Thandle; VAR Buf : Typestruct_SIEII );
VAR
Buffer : Typestruct_SIEII_Buf ;
I : Integer ;
BEGIN
FILEREAD(Fl, Buffer, Sizeof(Buffer));
Buf^.Field1:= Buffer.Field1;
Buf^.Field2:= Buffer.Field2;
END;

PROCEDURE Readdir_SIEII ( VAR Fl: Thandle; VAR Buf : Typestruct_SIEII; N: INTEGER );
VAR
Buffer : Typestruct_SIEII_Buf ;
I : Integer ;
BEGIN
FILESEEK(Fl, Sizeof(Buf_caract_SIEII) + (N-1)*Sizeof(Buffer ), 0);
FILEREAD(Fl, Buffer, Sizeof(Buffer));
Buf^.Field1:= Buffer.Field1;
Buf^.Field2:= Buffer.Field2;
END;

PROCEDURE Add_SIEII ( VAR Fl: Thandle; Buf : Typestruct_SIEII);
VAR
Buffer : Typestruct_SIEII_Buf ;
I : Integer;
BEGIN
Buffer.Field1:= Buf^.Field1;
Buffer.Field2:= Buf^.Field2;
FILESEEK(Fl, 0, 2);
FILEWRITE(Fl, Buffer, Sizeof(Buffer))
END;

FUNCTION Endfile_SIEII ( VAR Fl : Thandle): BOOLEAN;
```

```
VAR
K, K2 : Longint;
BEGIN
K := FILESEEK(Fl, 0, 1); { Current position }
K2 :=FILESEEK(Fl, 0, 2); { Last position }
IF K = K2
THEN
Endfile_SIEII := true
ELSE
BEGIN
FILESEEK(Fl, K, 0);
Endfile_SIEII := False
END;
END;

FUNCTION Alloc_block_SIEII ( VAR Fl : Thandle) : INTEGER;
VAR
K : Longint;
BEGIN
K := FILESEEK(Fl, 0, 2); { End of file }
K := K - Sizeof( Typestruct_SIEII_entete); { Ignore the header }
K := K DIV Sizeof (Typestruct_SIEII_Buf);
K := K + 1;
Alloc_block_SIEII := K;
END;

{ Declaration part of variables }
VAR
F :  Thandle;
B1 : Typestruct_SIEII ;
{ Body of main program }
BEGIN
NEW(B1);
END.
```

الانتقال من Z إلى C

تعريف
تعبيرات
تخصيص
تكرار المشروط
تكرار المعدود
الاشتراط
قراءة
كتابة
إجراء
دالة
دوال قياسية
خوارزمية

تعريف المتغيرات

يتم إعلان المتغيرات في C بواسطة

```
Type <Li>;
```

حيث <Li>يدل على قائمة المعرفات .

الكائنات بسيطة

مكافئات الكائنات Z إلى C :

صحيح يترجم إلى .**INT**
محرف يترجم إلى .**CHAR**

إلى النوع "سلسلة "نربط النوع Stالذي تم إنشاؤه والذي نحدده بواسطة
typedef char St[256]

في C النوع المنطقي غير موجود.
لمواصلة العمل دائما مع هذا النوع ، فقط أضف في بداية البرنامج
typdef int Booleen
وتعيين القيم صواب و خطأ على النحو التالي :
#define true 1
#define false 0

بنى

لتحديد بنية في C، يجب عليك اختيار التطبيق.
التطبيق هو اختيار تمثيل الذاكرة (عادة ثابتة أو ديناميكية) وترجمة عمليات آلة مجردة في هذا التمثيل. يكفي استبدال البنية س بواسطة المؤشر وإضافة على مستوى رأس البرنامج C التطبيق المطلوب حيث سيتم تحديد نوع المؤشر.

التعبيرات

يتم تضمين قواعد تعبيرات Z في قواعد اللغة C.

التخصيص

نفس بناء الجملة مع ' = 'بدلا من '.=:'

التكرار المشروط

----------Z----------
مادام شرط :
تعليمات
نهاية_مادام

يترجم إلى

----------C----------

```
while ( Cond )
  {
   Statements
  }
```

التكرار المعدود

----------Z----------

لكل :=ع1، ع2 [،ع3]

تعليمات

نهاية_لكل

يترجم إلى

----------C----------

```
for (V=Exp1; V<=Exp2; V=V+Exp3)
   {
     Statements
   }
```

الاشتراط

----------Z----------

إذا شرط :

تعليمات

[وإلا

تعليمات]

نهاية_إذا

يترجم إلى

----------C----------

```
if ( Cond )
 {
   Statements
 }
[else
 { Statements } ]
```

القراءة

--------Z--------

اقرأ(م1، م2،...)

يترجم إلى

----------C----------

```
scanf("...", &V1, &V2, ...)
```

كتابة

--------Z----------

اكتب(ع1،ع2،...)

يترجم إلى

----------C----------

```
printf(E1, E2, ...)
```

إجراء

----------Z----------

إجراء **إسم (و1، و2...)**

ليكن

تعريف الكائنات المحلية و الوسطاء

بداية

تعليمات

نهاية

يترجم إلى

----------C----------

```
void Name ( typ1 P1 , typ2 P2, ...)
   {
      Definition of local objects
      Statements
   }
```

دالة

----------Z----------

دالة **إسم (و1، و2...) : نوع**

ليكن

تعريف الكائنات المحلية و الوسطاء

بداية

تعليمات

نهاية

يترجم إلى

----------C----------

```
type Name ( typ1 P1, typ2 P2, ...)
   {
      Definition of local objects
      Statements
   }
```

دوال قياسية

باقي_قسمة(أ، ب)

```
int Mod( int a, int b)
   { return ( a % b ); }
```

أدنى(أ، ب)

```
int Min (int a, int b)
   {
     if (a < b) return(a);
```

```
    else return(b);
  }
```

أقصى(أ، ب)

```
int Max (int a, int b)
  {
    if (a > b) return(a);
    else return(b);
  }
```

أس(أ، ب)

```
int Exp (int a, int b)
  {
    int i; int Ex ;
    Ex = 1;
    for (i= 1; i<=b; i++)
      Ex = Ex * a ;
    return (Ex);
  }
```

عدد_عشوائي (ن)

```
int Aleanombre( int N )
  { return ( rand() % N ); }
```

سلسلة_عشوائية (ن)

```
char *Aleachaine ( int N )
    {
      int k;
      char * Chaine = malloc(N);
      char Chr1[26] = "abcdefghijklmnopqrstuvwxyz";
      char Chr2[26] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

      for (k=0;k<N; k++)
         switch ( rand() % 2 ){
         case 0 : *(Chaine+k) = Chr1[rand() % 26] ; break ;
         case 1 : *(Chaine+k) = Chr2[rand() % 26] ; break ;
         }
       Chaine[k] = '\0' ;

        return (Chaine);
  }
```

طول(جملة)

```
int Longchaine ( string255 Ch )
  {
    return strlen(Ch);
  }
```

محرف_سلسلة (جملة , ر)

```
char *Caract ( string255 Ch , int I )
    {
      char *s = malloc(2);
      s[0] = Ch[I-1];
      s[1] = '\0';
     return s;
    }
```

خوارزمية

----------Z----------

ليكن

تعريف الكائنات المحلية و الشاملة

الإعلان عن الوحدات

بداية

تعليمات

نهاية

وحدة 1

....

وحدة ن

يترجم إلى

----------C----------

```
Local and static objects

Definition of functions
Function 1
Function 2
...
Function  n

main()
  {
      Statements
  }
```

تطبيق الآلات المجردة في سي (C)

جداول
بنى
قوائم
قوائم مزدوجة
أكوام
صفوف
أشجار البحث الثنائية
أشجار البحث المتعددة
ملفات

تطبيق الجداول في C

ليكن ج : جدول (5 ، 10) ؛

LET Tab : ARRAY (5 , 10) ;

```
/** -Implementation- **\: ARRAY OF INTEGERS**/

/** Arrays **/

typedef int Typeelem_V5_10i ;
typedef Typeelem_V5_10i * Typevect_V5_10i ;

Typeelem_V5_10i Element_V5_10i ( Typevect_V5_10i V , int I1 , int I2 )
{
return *(V + (I1-1)*10 + (I2-1) ) ;
}

void Ass_element_V5_10i ( Typevect_V5_10i V , int I1 , int I2, Typeelem_V5_10i Val )
{
*(V + (I1-1)*10 + (I2-1) ) = Val ;
}

/** Variables of main program **/
Typevect_V5_10i Tab;
/** Main program **/
int main(int argc, char *argv[])
{
Tab = malloc(5*10 * sizeof(int));
}
```

تطبيق البنى في C

ليكن س : بنية (سلسلة ، صحيح) ؛

LET S : STRUCTURE (STRING , INTEGER) ;

```
typedef char * string256 ;

/** Structures **/
typedef struct Tsi Type_Tsi ;
typedef Type_Tsi * Typestr_Tsi ;
typedef string255 Type1_Tsi ;
typedef int Type2_Tsi ;
struct Tsi
{
Type1_Tsi Field1 ;
Type2_Tsi Field2 ;
```

```
};

Type1_Tsi Struct1_Tsi ( Typestr_Tsi S)
{
return S->Field1 ;
}

Type2_Tsi Struct2_Tsi ( Typestr_Tsi S)
{
return S->Field2 ;
}

void Ass_struct1_Tsi ( Typestr_Tsi S, Type1_Tsi Val )
{
strcpy( S->Field1 , Val );
}

void Ass_struct2_Tsi ( Typestr_Tsi S, Type2_Tsi Val )
{
S->Field2 = Val ;
}

/** Variables of main program **/
Typevect_V5_10i Tab;
Typestr_Tsi S;
/** Main program **/
int main(int argc, char *argv[])
{
S = malloc(sizeof(Typestr_Tsi));
S->Field1 = malloc(sizeof(string256));
}
```

تطبيق القوائم في C
ليكن ق :قائمة ؛

LET L : LIST ;

```
/** -Implementation- **\: LIST Of INTEGERS**/

/** Linked lists **/

typedef int Typeelem_Li ;
typedef struct Maillon_Li * Pointer_Li ;

struct Maillon_Li
{
Typeelem_Li Val ;
Pointer_Li Next ;
} ;
```

```
Pointer_Li Allocate_cell_Li (Pointer_Li *P)
{
*P = (struct Maillon_Li *) malloc( sizeof( struct Maillon_Li)) ;
(*P)->Next = NULL;
}

void Ass_val_Li(Pointer_Li P, Typeelem_Li Val)
{
P->Val = Val ;
}

void Ass_adr_Li( Pointer_Li P, Pointer_Li Q)
{
P->Next = Q ;
}

Pointer_Li Next_Li( Pointer_Li P)
{ return( P->Next ) ; }

Typeelem_Li Cell_value_Li( Pointer_Li P)
{ return( P->Val) ; }

void Free_Li ( Pointer_Li P)
{ free (P);}


/** Variables of main program **/
Pointeur_Li L=NULL;
/** Main program **/
int main(int argc, char *argv[])
{
}
```

تطبيق القوائم مزدوجة في C
ليكن ق :قائمة_مزدوجة ؛

LET L : BILIST ;

```
 /** -Implementation- **\: BIDIRECTIONAL LIST OF INTEGERS**/

/** Bidirectional linked lists **/

typedef int Typeelem_Ri ;
typedef struct Maillon_Ri * Pointer_Ri ;

struct Maillon_Ri
{
```

```
Typeelem_Ri Val ;
Pointer_Ri Next ;
Pointer_Ri Prev ;
} ;

Pointer_Ri Allocate_cell_Ri (Pointer_Ri *P)
{
*P = (struct Maillon_Ri *) malloc( sizeof( struct Maillon_Ri)) ;
(*P)->Next = NULL;
(*P)->Prev = NULL;
}

void Ass_val_Ri(Pointer_Ri P, Typeelem_Ri Val)
{
P->Val = Val ;
}

void Ass_r_adr_Ri( Pointer_Ri P, Pointer_Ri Q)
{ P->Next = Q; }

void Ass_l_adr_Ri( Pointer_Ri P, Pointer_Ri Q)
{ P->Prev = Q; }

Pointer_Ri Next_Ri( Pointer_Ri P)
{ return( P->Next ); }

Pointer_Ri Previous_Ri( Pointer_Ri P)
{ return( P->Prev ); }

Typeelem_Ri Cell_value_Ri( Pointer_Ri P)
{ return( P->Val) ; }

void Free_Ri ( Pointer_Ri P)
{ free (P) ; }


/** Variables of main program **/
Pointeur_Ri L=NULL;

/** Main program **/
int main(int argc, char *argv[])
{
}
```

تطبيق الأشجار البحث الثنائية في C
ليكن ا :شجرة_بحث_ثنائية ؛

LET A : BST ;

```
/** -Implementation- **\: BINARY SERACH TREE OF INTEGERS**/

/** Binary search trees **/

typedef int Typeelem_Ai ;
typedef struct Noeud_Ai * Pointer_Ai ;

struct Noeud_Ai
{
Typeelem_Ai Val ;
Pointer_Ai Lc ;
Pointer_Ai Rc ;
Pointer_Ai Parent ;
} ;

Typeelem_Ai Node_value_Ai( Pointer_Ai P )
{ return P->Val; }

Pointer_Ai Lc_Ai( Pointer_Ai P)
{ return P->Lc ; }

Pointer_Ai Rc_Ai( Pointer_Ai P)
{ return P->Rc ; }

Pointer_Ai Parent_Ai( Pointer_Ai P)
{ return P->Parent ; }

void Ass_node_val_Ai ( Pointer_Ai P, Typeelem_Ai Val)
{
P->Val = Val ;
}

void Ass_lc_Ai( Pointer_Ai P, Pointer_Ai Q)
{ P->Lc = Q; }

void Ass_rc_Ai( Pointer_Ai P, Pointer_Ai Q)
{ P->Rc = Q ; }

void Ass_parent_Ai( Pointer_Ai P, Pointer_Ai Q)
{ P->Parent = Q ; }

void Allocate_node_Ai( Pointer_Ai *P)
{
*P = (struct Noeud_Ai *) malloc( sizeof( struct Noeud_Ai)) ;
(*P)->Lc = NULL;
(*P)->Rc = NULL;
(*P)->Parent = NULL;
}
```

```
void Free_node_Ai( Pointer_Ai P)
{ free( P ) ; }

/** Variables of main program **/
Pointeur_Ai A=NULL;
/** Main program **/
int main(int argc, char *argv[])
{
}
```

تطبيق الأشجار البحث المتعددة في C
ليكن ش :شجرة_بحث_متعددة (4)؛

LET M : MST (4) ;

```
 /** -Implementation- **\: M-ARY SEARCH TREE OF INTEGERS**/

/** M-ary search trees **/

typedef int Typeelem_M4i ;
typedef struct Noeud_M4i * Pointer_M4i ;

struct Noeud_M4i
{
int Degree ;
Pointer_M4i Child[4] ;
Typeelem_M4i Infor[4] ;
Pointer_M4i Parent ;
} ;

Typeelem_M4i Node_value_mst_M4i(Pointer_M4i P, int I)
{ return P->Infor[I-1] ; }

Pointer_M4i Child_M4i( Pointer_M4i P, int I)
{ return P->Child[I-1] ; }

Pointer_M4i Parent_M4i( Pointer_M4i P)
{ return P->Parent ; }

void Ass_node_val_mst_M4i ( Pointer_M4i P, int I, Typeelem_M4i Val)
{
P->Infor[I-1] = Val ;
}

void Ass_child_M4i( Pointer_M4i P, int I, Pointer_M4i Q)
{ P->Child[I-1] = Q ; }

void Aff_parent_M4i( Pointer_M4i P, Pointer_M4i Q)
{ P->Parent = Q ; }
```

```
void Allocate_node_M4i( Pointer_M4i *P )
{
int I ;

*P = (struct Noeud_M4i *) malloc( sizeof( struct Noeud_M4i)) ;
for (I=0; I< 4; ++I) (*P)->Child[I] = NULL;
(*P)->Degree = 0 ;
}

int Degree_M4i ( Pointer_M4i P )
{ return P->Degree ; }

void Ass_degree_M4i ( Pointer_M4i P, int N)
{ P->Degree = N ; }

void Free_node_M4i(Pointer_M4i P)
{ free ( P );}


/** Variables of main program **/
Pointeur_M4i M=NULL;
/** Main program **/
int main(int argc, char *argv[])
{
}
```

تطبيق الأكوام في C
ليكن ك :كومة ؛

LET P : STACK ;

```
/** -Implementation- **\: STACK OF INTEGERS**/
/** Stacks**/

typedef int Typeelem_Pi ;
typedef struct Maillon_Pi * Pointer_Pi ;
typedef Pointer_Pi Typepile_Pi ;

struct Maillon_Pi
{
Typeelem_Pi Val ;
Pointer_Pi Next ;
} ;

void Createstack_Pi( Pointer_Pi *P )
{ *P = NULL ; }

bool Empty_stack_Pi ( Pointer_Pi P )
```

```
{ return (P == NULL ); }

void Push_Pi ( Pointer_Pi *P, Typeelem_Pi Val )
{
Pointer_Pi Q;

Q = (struct Maillon_Pi *) malloc( sizeof( struct Maillon_Pi)) ;
Q->Val = Val ;
Q->Next = *P;
*P = Q;
}

void Pop_Pi ( Pointer_Pi *P, Typeelem_Pi *Val )
{
Pointer_Pi Save;

if (! Empty_stack_Pi (*P) )
{
*Val = (*P)->Val ;
Save = *P;
*P = (*P)->Next;
free(Save);
}
else printf ("%s \n", "Stack is empty");
}

/** Variables of main program **/
Pointeur_Pi P=NULL;
/** Main program **/
int main(int argc, char *argv[])
{
}
```

تطبيق الصفوف في C
ليكن ص :صف ؛

LET F : QUEUE ;

```
/** -Implementation- **\: QUEUE OF INTEGERS**/
/** Queues **/

typedef int Typeelem_Fi ;
typedef struct Filedattente_Fi * Pointer_Fi ;
typedef struct Maillon_Fi * Ptliste_Fi ;

struct Maillon_Fi
{
Typeelem_Fi Val ;
Ptliste_Fi Next ;
```

```
};

struct Filedattente_Fi
{
Ptliste_Fi Head, Queue ;
};

void Createqueue_Fi ( Pointer_Fi *Fil )
{
*Fil = (struct Filedattente_Fi *) malloc( sizeof( struct Filedattente_Fi)) ;
(*Fil)->Head = NULL ;
(*Fil)->Queue = NULL ;
}

bool Empty_queue_Fi (Pointer_Fi Fil )
{ return Fil->Head == NULL ;}

void Enqueue_Fi ( Pointer_Fi Fil , Typeelem_Fi Val )
{
Ptliste_Fi Q;

Q = (struct Maillon_Fi *) malloc( sizeof( struct Maillon_Fi)) ;
Q->Val = Val ;
Q->Next = NULL;
if ( ! Empty_queue_Fi(Fil) )
Fil->Queue->Next = Q ;
else Fil->Head = Q;
Fil->Queue = Q;
}

void Dequeue_Fi (Pointer_Fi Fil, Typeelem_Fi *Val )
{
if (! Empty_queue_Fi(Fil) )
{
*Val = Fil->Head->Val ;
Fil->Head = Fil->Head->Next;
}
else printf ("%s \n", "Queue is empty");
}

/** Variables of main program **/
Pointeur_Fi F=NULL;
/** Main program **/
int main(int argc, char *argv[])
{
}
```

تطبيق الملفات في C

ليكن م : ملف من (سلاسل ، صحيح) صدر_ملف (صحيح ، صحيح) مخزن ب1 ؛

LET F : FILE OF (STRINGS , INTEGER) HEADER (INTEGER , INTEGER) BUFFER B1 ;

```
/** -Implementation- **\: FILE **/

/* Managing open files */

struct _Node
{
FILE * Var_file ;
char * Name_file ;
int Save_pos;
struct _Node *Next ;
} ;

typedef struct _Node * _Ptr_Node;

_Ptr_Node _Stack_Opens = NULL;

/* Test if a file is open */
_Ptr_Node _Open ( char * Fp)
{
_Ptr_Node P;
bool Trouv ;
P = _Stack_Opens; Trouv = False ;
while ((P != NULL) && ! Trouv )
if ( strcmp(P->Name_file, Fp) == 0)
Trouv = True;
else P = P->Next;
return P;
}

/* Add an open file */
void _Push_Open ( char *Fp, FILE *Fl)
{
_Ptr_Node P ;
P = (_Ptr_Node) malloc( sizeof( struct _Node)) ;
P->Name_file = Fp;
P->Var_file = Fl;
P->Next = _Stack_Opens;
_Stack_Opens = P;
}

/* Delete an open file and return its name */
char * _Pop_Open ( FILE *Fl)
{
char * Fp = malloc (100);
_Ptr_Node P, Prec ;
P= _Stack_Opens;
Prec = NULL;
while (P->Var_file != Fl )
{ Prec = P ; P = P->Next ;}
```

```
strcpy(Fp, P->Name_file);
if (Prec != NULL)
Prec->Next = P->Next;
else _Stack_Opens = P->Next;
free (P);
return Fp ;
}


/** Files **/

typedef char _Tx[255];
/** Types of block fields **/
typedef string255 Typefield1_siEii;
typedef _Tx Typefield1_siEii_Buf ;
typedef int Typefield2_siEii;

/** Types of header fields **/
typedef int Typeentete1_siEii ;
typedef int Typeentete2_siEii ;

/** Type of file data block **/
typedef struct
{
Typefield1_siEii_Buf Field1 ;
Typefield2_siEii Field2 ;
} Typestruct1_siEii_Buf;

/** Type of File data block structure **/
typedef struct
{
Typefield1_siEii Field1 ;
Typefield2_siEii Field2 ;
} Typestruct1_siEii_;
typedef Typestruct1_siEii_ * Typestruct1_siEii ;


/** Type of block of file features **/
typedef struct
{
Typeentete1_siEii Entete1 ;
Typeentete2_siEii Entete2 ;
} Typestruct2_siEii ;

/** Declaration of header block **/

Typestruct2_siEii Bloc_caract_siEii;

/** Operations on files **/

void Open_siEii ( FILE **siEii , char *Fp , char * Mode )
```

```
{
_Ptr_Node P = _Open(Fp);
if ( P != NULL )
/* The file is already open */
{
P->Save_pos = ftell (P->Var_file);
fseek( P->Var_file, 0, 0);
fwrite(&Bloc_caract_siEii, sizeof(Typestruct2_siEii), 1, P->Var_file);
fclose(P->Var_file);
}
/* The file is not open */
if ( strcmp(Mode,"A") == 0)
{
*siEii = fopen(Fp, "r+b");
fread(&Bloc_caract_siEii, sizeof(Typestruct2_siEii), 1, *siEii);
}
else
{
*siEii = fopen(Fp, "w+b");
fwrite(&Bloc_caract_siEii, sizeof(Typestruct2_siEii), 1, *siEii) ;
}
_Push_Open( Fp, *siEii);
}

void Close_siEii ( FILE * siEii )
{
char * Fp = malloc(100);
_Ptr_Node P ;
strcpy(Fp, _Pop_Open(siEii));
fseek( siEii, 0, 0);
fwrite(&Bloc_caract_siEii, sizeof(Typestruct2_siEii), 1, siEii);
fclose(siEii) ;
/* Is there a file open with the same name? */
/* If yes, open it again at the saved position */
P = _Open (Fp);
if ( P != NULL)
{
siEii = fopen(P->Name_file, "r+b");
fread(&Bloc_caract_siEii, sizeof(Typestruct2_siEii), 1, siEii);
fseek(siEii, P->Save_pos, 0);
}
}

Typeentete1_siEii Entete1_siEii( FILE * siEii)
{
return Bloc_caract_siEii.Entete1;
}

Typeentete2_siEii Entete2_siEii( FILE * siEii)
{
return Bloc_caract_siEii.Entete2;
```

```
}

void Aff_entete1_siEii ( FILE * siEii, Typeentete1_siEii VAL)
{
Bloc_caract_siEii.Entete1 = VAL ;
}

void Aff_entete2_siEii ( FILE * siEii, Typeentete2_siEii VAL)
{
Bloc_caract_siEii.Entete2 = VAL ;
}

void Writeseq_siEii ( FILE * siEii, Typestruct1_siEii Buf )
{
Typestruct1_siEii_Buf Buffer ;
int I, J;
for(J=0; J<= strlen(Buf->Field1); ++J)
Buffer.Field1[J] = Buf->Field1[J];
Buffer.Field2 = Buf->Field2;
fwrite(&Buffer, sizeof( Typestruct1_siEii_Buf), 1, siEii) ;
}

void Writedir_siEii ( FILE * siEii, Typestruct1_siEii Buf, int N )
{
Typestruct1_siEii_Buf Buffer ;
int I, J;
for(J=0; J<= strlen(Buf->Field1); ++J)
Buffer.Field1[J] = Buf->Field1[J];
Buffer.Field2 = Buf->Field2;
fseek(siEii, (long) ((N-1)* sizeof( Typestruct1_siEii_Buf) +
sizeof( Typestruct2_siEii)), 0 );
fwrite(&Buffer, sizeof( Typestruct1_siEii_Buf), 1, siEii) ;
}

void Readseq_siEii ( FILE * siEii, Typestruct1_siEii Buf )
{
Typestruct1_siEii_Buf Buffer ;
int I, J;
if (fread(&Buffer, sizeof( Typestruct1_siEii_Buf), 1, siEii)!=0){
for(J=0; J<= strlen(Buffer.Field1); ++J)
Buf->Field1[J] = Buffer.Field1[J];
Buf->Field2= Buffer.Field2;
}
}

void Readdir_siEii ( FILE * siEii, Typestruct1_siEii Buf, int N)
{
Typestruct1_siEii_Buf Buffer ;
int I, J;
fseek(siEii, (long) ((N-1)* sizeof( Typestruct1_siEii_Buf) +
sizeof( Typestruct2_siEii)), 0 );
```

```
fread(&Buffer, sizeof( Typestruct1_siEii_Buf), 1, siEii);
for(J=0; J<= strlen(Buffer.Field1); ++J)
Buf->Field1[J] = Buffer.Field1[J];
Buf->Field2= Buffer.Field2;
}

void Add_siEii ( FILE * siEii, Typestruct1_siEii Buf )
{
Typestruct1_siEii_Buf Buffer ;
int I, J;
for(J=0; J<= strlen(Buf->Field1); ++J)
Buffer.Field1[J] = Buf->Field1[J];
Buffer.Field2 = Buf->Field2;
fseek(siEii, 0, 2); /* End of file */
fwrite(&Buffer, sizeof( Typestruct1_siEii_Buf), 1, siEii) ;
}

bool Endfile_siEii (FILE * siEii)
{
long K = ftell(siEii);
fseek(siEii, 0, 2); /* End of file */
long K2 = ftell(siEii); /* Position from beginning */
if (K==K2)
{ fseek(siEii, K, 0); return 1;}
else
{ fseek(siEii, K, 0); return 0;}
}

int Alloc_block_siEii (FILE * siEii)
{
long K;
fseek(siEii, 0, 2); /* End of file */
K = ftell(siEii); /* Position from beginning */
K = K - sizeof( Typestruct2_siEii); /* Ignore the header */
K = K / sizeof (Typestruct1_siEii_Buf);
K ++;
return(K);
}

/** Variables of main program **/
FILE *F;
Typestruct1_siEii B1 ;
/** Main program **/
int main(int argc, char *argv[])
{
B1 = malloc(sizeof(Typestruct1_siEii));
B1->Field1 = malloc(255 * sizeof(string255));
}
```

فهرس الكلمات الرئيسية

ا

أكوام
أب
إبن
إجراء
إجراءات
احجز_كتلة_ملف
احجز_بنية
احجز_جدول
احجز_خلية
انفذ
اظف_ملف
اقرأ
اقرأ_مباشرة_ملف
اقرأ_تسلسلا_ملف
إبن_يمين
أغلق_ملف
إبن_يسار

أو
افتح_ملف
أس
احجز_صف
احجز_عقدة
احجز_كومة
أقصى
أدنى
اكتب
اكتب_مباشرة_ملف
اكتب_تسلسلا_ملف
إذا
إلى
إنشاء_شجرة_بحث_ثنائية
إنشاء_شجرة_بحث_متعددة
إنشاء_صف
إنشاء_قائمة
إنشاء_قائمة_مزدوجة
إنشاء_كومة

ب

بداية
باقي_قسمة
بنية
بنى

ت

تالي

ج

جدول
جداول

ح

حقل

خ

خطأ

د

درجة
ديناميكي
ديناميكية
دالة
دوال

س

سابق

سرح
سرح_عقدة
سرح_بنية
سرح_جدول
سلسلة_عشوائية
سلسلة
سلاسل

ش

شجرة_بحث_ثنائية
شجرة_بحث_متعددة

ص

صدر_ملف
صحيح
صحاح
صواب
صف
صفوف
صف_فارغ

ض

ضع_عنوان
ضع_عنوان_يمين
ضع_عنوان_يسار
ضع_درجة
ضع_عنصر
ضع_صدر_ملف
ضع_إبن_يمين
ضع_إبن_يسار
ضع_إبن
ضع_قيمة_عقدة
ضع_قيمة_عقدة_شم
ضع_أب
ضع_حقل
ضع_قيمة_خلية
ضف_كومة
ضف_صف

ط

طول_سلسلة

ع

عدد_عشوائي
عدم
عنصر

ق

قائمة
قيمة_خلية
قيمة_عقدة
قيمة_عقدة_شم
قوائم
قائمة_مزدوجة

ك

كومة
كومة_فارغة

ل

لا
لكل
ليكن

م

منطيقي
مناطقة
مخزن
محرف
محرف
محارف
محرف_سلسلة
من
ملء_بنية
ملء_جدول
مادام
ملف
ملفات
مؤشر
مؤشرات

ن

نزع_صف
نزع_كومة
نهاية
نهاية_ملف
نهاية_لكل
نهاية_إذا
نهاية_مادام

و

و
وإلا