الفصل 6 : الأنواع الخاصّة

1. مقدّمة

في البرمجة، يجب أن يكون لكلّ جزء من البيانات، التي يتمّ التعامل معها، نوع خاصّ بها. ممّا يسمح للمجمِّع التحقّق من صحّة القيم والعمليّات التي سيتمّ تطبيقها. وهو ما يساعد المبرمج في اكتشاف الأخطاء وتجنُّبها. هناك عدّة أنواع معرّفة مسبّقا في لغة البرمجة. مثل الأعداد الصحيحة والرموز. كما يمكن للمبرمج تعريف أنواع خاصّة به، مشتقّة من الأنواع الأساسية. مثل الجداول، التي تطرّقنا لها في الفصل السابق، التعداد، السجلاّت، وأنواع أخرى.

2. التعداد ENUMERATIONS

2.1.التعريف

النوع enum (التعداد): هو عبارة عن نوع بيانات، يتمّ تعريفه من قبل المبرمِج. وهو قائمة مرتّبة من القيم الثابتة، معرَّفة عن طريق سرد أسماء لها. والتي لها معنى بالنسبة للمبرمج فقط، ممّا يسهّل عليه تذكّرها، وفهم البرنامج. يُستعمل نوع enum لتعريف نوع جديد، لا يحتوي إلاّ على هذه القيم.

2.2.التصريح

```
enum type_name {const_name1, ...} var_list ;
```

كلمة enum هي كلمة محجوزة تُستخدم لتحديد مجموعة من الثوابت من النوع int.

- type_name هو الاسم الذي يعطيه المبرمج لهذه المجموعة. ويجب أن يكون معرِّفا صحيحا (اختياريّ).
- const_name أسماء ثوابت تشير إلى عناصر المجموعة. لها معنى بالنسبة للمبرمج فقط. يمكن تعيين قيمها.
- var_list هي قائمة متغيّرات من هذا النوع (اختياريّة).

بالنسبة للغة C، هذه الأسماء عبارة عن ثوابت من نوع int، قيمها هي 0، 1، ... . كما يمكن إسناد قيم أخرى لها كما يلي:

```
enum type_name {const_name= val1, ...}var_list ;
```

- val هو عدد صحيح. إذا كان val غير موجود ، فان قيمته هي قيمة الثابت الذي قبله في القائمة +1. بالنسبة للثابت الصحيح الأول في القائمة، تكون قيمته الافتراضية هي 0.

على الرغم من أنّ كُلًّا من أسماء المتغيّرات واسم التعداد اختياريا، إلّا أنّه يجب أن يظهر أحدُ الاثنين.

مثال

```
enum Days {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};
Days d;
```

2.3.الاستعمال

تُوفّر الثوابت أسماء من السهل تذكّرها أثناء البرمجة أفضل من الأعداد، كما تحدّد مجموعة القيم المقبولة. ممّا يجنب المبرمج الوقوع في الخطأ أثناء البرمجة، إذ لا يمكن إسناد أيّ قيمة خارج المجموعة لمتغيّر من نوع enum (C++).

يمكن استعمال عمليّات المقارنة مثل >، <، =، ≠. كما يمكن استخدام هذا النوع مع التعليمة selon (switch).

2.4.مثال

```
enum Sexe {Male, Femelle };                                          :الـجنس
enum Mois {Janv, Fevr, Mars, Avr, Mai, Juin, Juil, Aout, Sept, Oct, Nov, Dec};   :الأشهر
enum Days {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};      :الأيّـام
Days d;
d=Tuesday ;
```

```
switch(d)
{
   case Friday: printf("Weekend\n") ;   break;
   case Saturday: printf("de 08:00 à 12:00\n") ;   break;
   default : printf("de 08:00 à 16:00\n") ;
}
```

3. السجلّات أو البُنى

3.1.التعريف

البُنية: (structure) وتدعى أيضا سِجلّ (بالفرنسيّة enregistrement، وبالإنجليزيّة record): هي نوع مركّب، يمثّل مجموعة من العناصر المسمّاة، والتي يمكن الوصول إليها عن طريق أسمائها. ويدعى كلّ عنصر منها حقلا. ويمكن لهذه الحقول أن تكون من أي نوع كان. تُستخدم البُنية لتجميع المتغيّرات في سجل واحد.

3.2.التصريح

struct struct_name { type field_names ; … }structure_variables ;	structure_variables : **struct** field_names :type … **finStruct**

حيث أنّ كلمة struct هي كلمة محجوزة في C.

- struct_name: اسم للبنية (اختياريّ) ويجب أن يكون معرِّفا صحيحا.
- type field_names: التصريح بالبيانات (الحقول) المكوّنة للبنية. يمكنك تعريف العناصر في السجل عن طريق تسمية النوع (type)، متبوعا بواحد أو أكثر من أسماء المتغيّرات (field-name) (مفصولة بفاصلة ","). والمتغيرات المختلفة النوع بواسطة فاصلة منقوطة ";".
- structure_variables: أسماء المتغيّرات من نوع البُنية، وهي أيضا اختياريّة.

على الرغم من أن كُلّا من أسماء المتغيّرات واسم للبنية اختياريّة، إلّا أنه يجب أن يظهر أحدُ الاثنين.

مثال

struct Etudiant { char nom[20] ; float bac; }e1, e2;	e1, e2 : **struct** nom : chaine de caractères bac : réel **finStruct**

هنا تمّ التصريح بمتغيّرين e1 وe2 من نوع struct Etudiant. حيث انّ كلّ متغيّر يحتوي على حقلين هما: nom من نوع سلسلة وbac من نوع عدد حقيقيّ.

يمكن تأخير التصريح بالمتغيّرات هكذا

struct Etudiant {
 char nom[20] ;
 float bac;
};
 	
struct Etudiant e1, e2 ;
	type Etudiant = **struct**
	 nom : chaine de caractères
	 bac : réel
	finStruct
	var
	e1, e2 : Etudiant

حيث انّ ";" ضروريّة بعد }، كما أنّه يجب إعادة كلمة struct قبل اسم البنية في لغة C. وهي غير ضروريّة في C++.

يُفضّل استعمال typedef، في C، للتصريح بنوع البنية، قبل التصريح بالمتغيّرات. فيصبح المثال السابق كالتالي:

typedef struct { char nom[20] ; float bac ; } Etudiant ;	**type** Etudiant = **struct** nom : chaine de caractères bac : réel **finStruct** **var**

`Etudiant e1, e2 ;`	`e1, e2 : Etudiant`

حيث انّ اسم البنية غير موجود، وEtudiant هو الاسم الجديد للبنية.

يمكن للحقول أن تكون هي أيضا من نوع بنية، شريطة أن تكون معرّفة قبلها.

نظرًا لأن نطاق الحقل يقتصر على السجلّ الذي ينتمي إليه ، فلا داعي للقلق بشأن تعارض التسمية بين اسماء الحقول والمتغيّرات الأخرى.

3.3.التمثيل

أثناء التعريف بالنوع بنية، لا يتمّ حجز الذاكرة إلى حين التصريح بالمتغيّر من نوع بنية. ويتمّ تمثيل البنية في الذاكرة على شكل متغيّرات متجاورة. مثلا، أثناء التعريف بالنوع Etudiant، لا يتمّ تخصيص أي ذاكرة للحقلين nom وbac، ولكن يتمّ تخصيص الذاكرة لهما عند إنشاء للسجلّيْن e1 وe2. يمثّل الشكل التالي للسجلّيْن e1 وe2.

e1		e2	
nom	bac	nom	bac
Ahmed	13.41	Souad	12.50

حيث أنّ حجم البنية هو مجموع أحجام الحقول المكوّنة لها.

3.4.التهيئة initialization

في C يمكن تحديد قيم ابتدائيّة لجميع عناصر البنية داخل حاضنتين { و } أثناء التصريح بها. حيث يتمّ فصل القيم باستخدام الفاصلة "،"، ويجب أن تكون هاته القيم من نفس نوع الحقول، وترتيبها، وعددها.

مثال

```
Etudiant e1= { "Ahmed", 13.41} ;
```

3.5.الاستعمال

يُستخدم الرمز النقطة "." للوصول إلى العناصر الموجودة في البنية. وذلك باستعمال اسم المتغيّر من نوع بنية، متبوعا بالنقطة، ثمّ اسم أحد حقوله.

مثال

`strcpy(e1.nom, "Ahmed") ;`
`scanf("%f",&e1.moy) ;`
`gets(e2.nom) ;`
`e2.moy=e1.moy+1 ;`
	`e1.nom←"Ahmed"`
	`lire(e1.moy)`
	`lire(e1.nom)`
	`e2.moy←e1.moy+1`

يمكن إسناد متغيّر من نوع بنية إلى متغيّر آخر من نفس النوع، ليتمّ نسخ جميع الحقول إلى المتغيّر الثاني. ولكن لا يمكن المقارنة بينهما. كما يمكن استخدام جدول من السجلّات.

```
e2=e1 ;
Etudiant T[100] ;
```

3.6.مثال

اكتب البرنامج الذي يعرّف بنية تحتوي على معلومات الطلبة، (رقم الطالب، اسم الطالب، تاريخ الازدياد، معدّل الباك).

علما أنّ تاريخ الازدياد عبارة عن بنية تحتوي على (اليوم، الشهر، السنة). ثمّ يملأ جدولا من N طالب، ثمّ يطلب من المستعمل تاريخا، ليظهر جميع الطلبة المولودين بهذا التاريخ.

`#include <stdio.h>`
`#include <string.h>`
`typedef struct{`
`  int Day, Month , Year;`
	`Algorithme etudiants`
	`type`
	`Date =struct`
	`  Day, Month , Year : entier`

```
} Date ;
typedef struct{
  int num;
  char name [20] ;
  Date birthday ;
  float bac ;
} Student ;
int main(){
Student st[100] ;
int i, N ;
Date d ;
printf("entrer le nbr des étudiants\n") ;
scanf("%d",&N) ;
//remplir le tableau
for(i=0 ;i<N :i++){
  printf("étudiants %d\n",i) ;
  printf("Num : ") ;
  scanf("%d",&st[i].num) ;
  printf("Nom : ") ;
  gets(st[i].name) ;
  printf("Date de naissance (j/m/a) : ") ;
  scanf("%d%d%d", &st[i].birthday.Day,
    &st[i].birthday.Month,
    &st[i].birthday.Year ) ;
  printf("Bac : ") ;
  scanf("%d",&st[i].bac) ;
}
printf("entrer une date (j/m/a) : ") ;
scanf("%d%d%d", &d.Day, &d.Month,
  &d.Year ) ;
//affichage
printf("Num \tNom \tBac\n") ;
for(i=0 ;i<N :i++)
  if( (st[i].birthday.Day==d.Day) &&
    (st[i].birthday.Month==d.Month) &&
    (st[i].birthday.Year==d.Year) )
     printf("%d \t%s \t%.2f\n", st[i].num,
       st[i].name, st[i].bac) ;
return 0 ;
}
```

```
finStruct
Student =struct
  num : entier
  name : chaine de caractères
  birthday :Date
  bac :réel
finStruct

var
  st[100] : tableau  de Student
  i, N : entier
  d : Date
Début
écrire("entrer le nbr des étudiants")
lire(N) ;
Pour i←0 à N-1 faire
  écrire("étudiants ", i)
  écrire("Num : ")
  lire(st[i].num)
  écrire("Nom : ")
  lire(st[i].name)
  écrire("Date de naissance (j/m/a) : ")
  lire(st[i].birthday.Day,
    st[i].birthday.Month,
    st[i].birthday.Year )
  écrire("Bac : ")
  lire(st[i].bac)
finPour
écrire("entrer une date (j/m/a) : ")
lire(d.Day, d.Month, d.Year )
écrire("Num   Nom   Bac")
Pour i←0 à N-1 faire
  si (st[i].birthday.Day==d.Day) &&
    (st[i].birthday.Month==d.Month) &&
    (st[i].birthday.Year==d.Year) alors
      écrire(st[i].num, "   ", st[i].name,
        "   ", st[i].bac)
  finSi
finPour
fin
```

4. طرق أخرى للتصريح بالبيانات

union.4.1

4.1.1. التعريف

الإتّحاد ، مثل البنية ، هو مجموعة من العناصر من أنواع مختلفة. لكن لا يمكنه ان يحتوي، في كل لحظة من البرنامج، إلّا على قيمة واحدة لعنصر واحد من عناصره.

4.1.2. التصريح

union union_name { type field_names ; … }union_variables ;	union_variables : **union** field_names :type … **finUnion**

حيث أنّ كلمة union هي كلمة محجوزة في C.

- union_name: اسم الإتّحاد (اختياريّ) ويجب أن يكون معرّفا صحيحا.
- type field_names: التصريح بالبيانات (الحقول) المكوّنة للإتّحاد. يمكنك تعريف العناصر في الإتّحاد عن طريق تسمية النوع (type)، متبوعا بواحد أو أكثر من أسماء المتغيّرات (field-name) (مفصولة بفاصلة ","). والمتغيّرات المختلفة النوع بواسطة فاصلة منقوطة ";".

– union_variables: أسماء المتغيّرات من نوع الإتّحاد، وهي أيضا اختياريّة.

على الرغم من أن كُلّا من أسماء المتغيرات واسم الإتّحاد اختياريّة، إلّا أنّه يجب أن يظهر أحدُ الاثنين.

مثال

union Result{
  char grade;
  float moy;
}r1, r2;
	r1, r2 : **union**
	  grade: caractère
	  moy : réel
	finUnion

هنا تمّ التصريح بمتغيّرين r1 وr2 من نوع Result **union** حيث أنّ كلّ متغيّر يحتوي على حقلين هما grade من نوع رمز، وmoy من نوع عدد حقيقيّ.

يفضل استعمال typedef في C للتصريح بنوع الإتّحاد قبل التصريح بالمتغيّرات. فيصبح المثال السابق كالتالي:

typedef union {
  char grade;
  float moy;
} Result;
 	
Etudiant e1, e2 ;
	type Result = **union**
	  grade: caractère
	  moy: réel
	finStruct
	var
	r1, r2 : Result

4.1.3. التمثيل

أثناء التعريف بالنوع **union**، لا يتمّ حجز الذاكرة إلى حين التصريح بالمتغيّر من نوع **union**. ويتمّ تمثيل **union** في الذاكرة على شكل متغيّر واحد، يأخذ حجم أكبر عناصر الإتّحاد. مثلا، أثناء التصريح بالمتغيّر r1 من النوع Result، فلو فرضنا أنّ حجم char grade هو 1 بايت، وحجم float moy هو 4 بايت، فإنّ حجم المتغيّر r1 هو 4 بايت، وليس 5 بايت. فإذا تمّ استعمال الحقل grade سيتم استعمال البايت الأول وتجاهل الثلاثة الباقين، وإذا تمّ استعمال الحقل moy، سيتم استعمالها في 4.

حيث إذا غيّرنا أحدَ الحقول يتغيّر الآخر، لأنّه في الحقيقة لا يوجد إلّا مكان واحد، يتمّ ترجمته حسب نوع الحقل المستعمل.

```
r1.grade='A';
```

r1

grade/moy
A

```
r1.moy=12;
```

r1

grade/moy
12

4.2.أنواع أخرى

- Reference (C++ فقط) المراجع تسمح بالتعامل مع المتغيّر باسم آخر غير الاسم الذي تمّ التصريح به.

الصيغة:
```
type &référence = identificateur ;
```
مثال:
```
int &ref = i ;
```
هنا يصبح i وref اسمان لنفس المتغيّر.

- المؤشّر Pointer: للتصريح بمتغيّر يمكنه أن يحتوي على عناوين في الذاكرة (السداسي الثاني).
- القوائم المترابطة، الطوابير والمكدسات (السداسي الثاني) الأشجار (السنة الثانية).
- الصنف class (C++ فقط) : هي مجموعة من البيانات (مثل البنية)، ومجموعة من الدوال على هاته البيانات (السنة الثانية).