Pointers المؤشرات

1. Introducing pointer
2- pointer & array
3- pointer & function

جَامعة المَنارة
MANARA UNIVERSITY

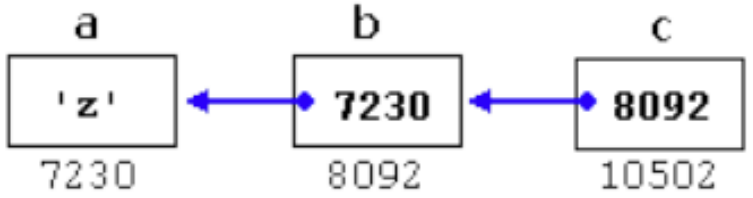
المحاضرة التاسعة
د كندة أبوقاسم

Pointers to Pointers

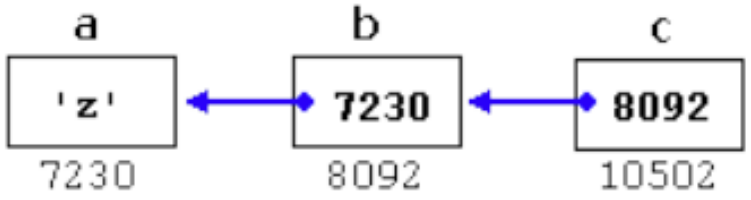
تسمح C ++أيضًا باستخدام المؤشرات التي تشير إلى مؤشرات أخرى ، والتي بدورها تشير إلى البيانات (أو حتى المؤشرات الأخرى). يتطلب بناء الجملة ببساطة علامة النجمة (*) لكل مستوى من مستويات غير المباشرة في إعلان المؤشر

```
char a;
char * b;
char ** c;
a = 'z';
b = &a;
c = &b;
```

بافتراض مواقع الذاكرة المختارة عشوائيًا لكل متغير 7230 و 8092 و 10502 ، يمكن تمثيلها على النحو التالي:



مع تمثيل قيمة كل متغير بداخلهالخلية المقابلة ، وعناوين الذاكرة الخاصة بهم ممثلة بالقيمة تحتها



الجديد هنا هو أن المتغير c، وهو مؤشر إلى مؤشر ، يمكن استخدامه في ثلاثة مستويات مختلفة غير المباشرة ، ولكل منها قيمة مختلفة

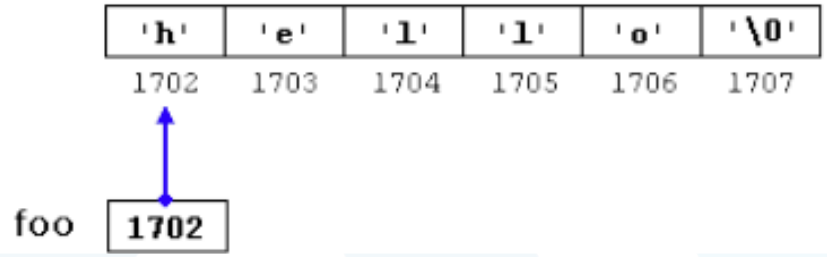
- c is of type char** and a value of 8092
- *c is of type char* and a value of 7230
- **c is of type char and a value of 'z'

Pointers and String Literals

يمكن أيضًا الوصول إلى السلاسل الحرفية مباشرة. إنها مصفوفات من نوع المصفوفة المناسبة لتحتوي على كل محارفها بالإضافة إلى حرف النهاية الصفري ، حيث يكون كل عنصر من نوع الحرف الثابت (حيث لا يمكن تعديلها أبدًا). على سبيل المثال:

```
const char * foo = "hello";
```

اذا يصرح عن مصفوفة بالتمثيل الحرفي لـ " "hello، ثم مؤشر إلى العنصر الأول المسمى foo. إذا تم تخزينه بدءًا من عنوان الذاكرة 1702 ، فيمكن تمثيله كـ



ذلك يشير المؤشر foo إلى سلسلة من الأحرف ، ولأن المؤشرات والمصفوفات يتم التعامل معها بشكل أساسي بنفس الطريقة ، يمكن استخدام foo للوصول إلى الأحرف داخل هذه المصفوفة بالطريقة نفسها ، حيث يكون لكلا الأمرين التاليين القيمة "o"

```
*(foo+4)
foo[4]
```

A pointer to a function

يحتوي مؤشر الدالة على عنوان الدالة في الذاكرة.
رأينا أن اسم المصفوفة هو حقًا العنوان الموجود في ذاكرة العنصر الأول من المصفوفة.
وبالمثل ، فإن اسم ادالة هو حقًا عنوان البداية في ذاكرة الكود الذي يشكل مهمة الدالة .
يمكن تمرير المؤشرات إلى الدوال ، وإرجاعها من الدوال ، وتخزينها في مصفوفات وتعيينها إلى مؤشرات دالة أخرى.

استدعاء التابع بالمؤشر

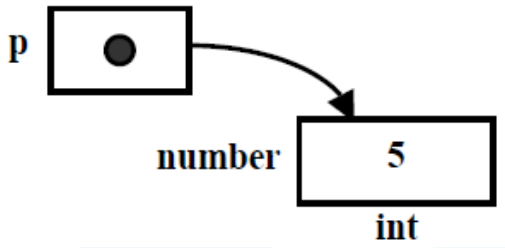
Call function by pointer

يمكن استدعاء التابع بالعنوان ولهذا الاستدعاء نوعان استدعاء بالمرجع واستدعاء بالمؤشر.

عند استدعاء التابع بالمؤشر يجب على التابع الذي يتلقى عنواناً لوسيط فعلي أن يتضمن في تعريفه مؤشراً كوسيط شكلي

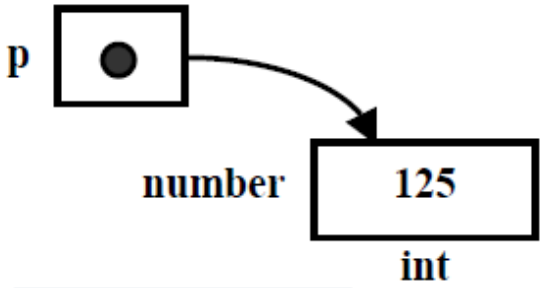
في هذا المثال يتضمن إعلان التابع cube المؤشر int *p الذي يتلقى عنوان المتغير الفعلي number (&number) الذي خصصت له القيمة 5 ، ويمكن تصور ذلك

العلاقة بين المؤشر p والمتغير num قبل تنفيذ التابع cube



داخل التابع cube تم تغيير القيمة التي يشير إليها المؤشر إلى 125 أي 5*5*5 لذلك أصبح p يشير إلى القيمة 1

العلاقة بين المؤشر p والمتغير num بعد تنفيذ التابع cube



استدعاء التابع بالمؤشر:

```
#include<iostream.h>
void cube(int *p)
{
 *p=*p* *p* *p;
}
void main()
{
 int number=5;
cube(&number);
cout<<"5 power 3 = "<<number<<endl;
}
```

Output

```
5 power 3 = 125
```

المؤشرات والتوابع Pointers to Functions

يسمح C ++للمؤشرات بالدوال. الاستخدام المعتاد لهذا هو تمرير دالة كوسيطة لدالة أخرى.
يتم التصريح عن المؤشرات إلى الدوال بنفس بناء الجملة كإعلان دالة عادي ، باستثناء اسم الدالة محاطًا بين قوسين () ويتم وضع علامة النجمة (*) قبل الاسم:

اعادة مؤشر من التابع
Return pointer from function

من الممكن أن تكون القيمة العائدة من التابع مؤشراً عندئذ لابد أن يعرف رأس التابع
بالشكل:

```
Type*function_name(parameter_lists)
```

مثلاً:

```
int*loc(int *a1 ,int *a2,int n1,int n2)
```

كتابة رأس التابع بهذا الشكل يعلن أن التابع يعيد مؤشر إلى عدد صحيح.

يسمح C ++ للمؤشرات بالدوال
. الاستخدام النموذجي لهذا هو تمرير دالة كوسيطة إلى دالة أخرى . يتم التصريح عن المؤشرات إلى الدوال بنفس بناء الجملة كإعلان دالة عادي ، باستثناء اسم الدالة محاطًا بين قوسين () ويتم وضع علامة النجمة (*) قبل الاسم:

في المثال السابق ، يعد الطرح مؤشرًا لوظيفة بها معلمتان من النوع int. يتم تهيئته مباشرة للإشارة إلى دالة الطرح
int (* minus)(int,int) = subtraction;

```
// pointer to functions
#include <iostream>
using namespace std;
int addition (int a, int b)
{ return (a+b); }
int subtraction (int a, int b)
{ return (a-b); }
int operation (int x, int y, int (*functocall)(int,int))
{
 int g;
 g = (*functocall)(x,y);
 return (g);
}
int main ()
{
 int m,n;
 int (*minus)(int,int) = subtraction;
 m = operation (7, 5, addition);
 m = operation (7, 5, subtraction);
 n = operation (20, m, minus);
 cout <<"n="<<n<<"    "<<"m="<<m;
 return 0;
}
```

Output
n=18   m=2

```
#include <stdio.h>
int main(){
char a[] = "character array";
char *b = "constant array";
a=b; /*compiler error: incompatible types
 in assignment*/
a[0]='C';
b[0]='A'; /*runtime error: SegmentationFault*/
return 0;
}
```

جَامعة المَنارة
MANARA UNIVERSITY

المحاضرة العاشرة
د كندة أبوقاسم

1. مدخل الى البرمجة الهيكلية
2. struct
3. مدخل الى البرمجة غرضية التوجه

Object oriented programming oop

السجلات Structures

هي مجموعات من المتغيرات المرتبطة تحت اسم واحد.

قد تحتوي **السجلات** على متغيرات من العديد من أنواع البيانات المختلفة - على عكس المصفوفات التي تحتوي فقط على عناصر من نفس نوع البيانات. تُستخدم **struct** بشكل شائع لتعريف السجلات التي سيتم تخزينها في ملفات

- **typedefs** لإنشاء أسماء مستعارة لأنواع البيانات المحددة مسبقًا.
- الاجتماع **unions**- على غرار الهياكل ، ولكن مع أعضاء يتشاركون نفس مساحة التخزين.
- معاملات البت **bitwise operators**- لمعالجة بتات المعاملات المتكاملة.
- حقول البت **bit fields—unsigned int or int**- أعضاء intأو intغير الموقعة في الهياكل أو النقابات التي أنت من أجلها حدد عدد وحدات البت التي يتم تخزين الأعضاء بها ، مما يساعدك على حزم التشكيل بإحكام.
- التعداد - **enumerations** مجموعات من ثوابت الأعداد الصحيحة ممثلة بمعرفات.

تسمي تعريف الهيكل وتستخدم مع الهيكل لتعريف متغيرات نوع الهيكل - على سبيل المثال ، بطاقة الهيكل. المتغيرات المعلنة داخل أقواس تعريف الهيكل هي أعضاء الهيكل. يجب أن يكون للأعضاء من نفس نوع البنية أسماء فريدة ، ولكن قد يحتوي نوعان مختلفان من الهياكل على أعضاء بالاسم نفسه دون تعارض يجب أن ينتهي كل تعريف بنية بفاصلة منقوطة

السجلات(التركيبات) struct

تعريف السجل:

هو مجموعة من البيانات المختلفة في النوع مع بعضها البعض بحيث يمكن التعامل معها كوحدة واحدة .

مثلاً :

لكتابة برنامج لتسجيل بيانات موظفين في الشركة نحتاج إلى تخزين :

اسم الموظف وهو من نوع مصفوفة حرفية

```
Char name [40];
```

عنوان وهو من نوع مصفوفة حرفية

```
Char address [40];
```

عمره متحول من نوع عدد صحيح

```
Int age;
```

راتبه متحول من نوع عدد حقيقي

```
Float salary ;
```

وكما نلاحظ فإن جميع هذه البيانات يجب التعامل معها كوحدة واحدة لأنها لموظف واحد ولذلك فإننا بحاجة إلى سجل خاص لهذا الموظف

كيفية الإعلام عن السجل :

وهي اختصار struct للتصريح عن سجل نستخدم الكلمة المحجوزة ومعناها تركيب لكلمة structure

```
#include <iostream.h>
Struct employee  اسم السجل
{
---------;
---------;
---------;
---------;
}; لا ننسى الفاصلة المنقوطة هنا
Void main ( )
{
Struct employee emp;
```

هنا نصرح عن متحول خاص بالسجل

```
#include <iostream.h>
struct  employee
{
char name[40];          مصفوفة لكتابة اسم الموظف
char address[40];   مصفوفة لكتابة عنوانه
int age;                متحول لكتابة عمره
float salary;           متحول لكتابة راتبه
};
```

ونصرح طبعاً عن متحول خاص بهذا السجل كالتالي:

```
Void main ( )
{
struct  employee emp;
```

كيفية إدخال وإخراج المعلومات داخل هذا السجل

عندما نريد مثلاً إدخال عمر الموظف فإننا نكتب

```
Cin >> emp . age ;
```

وعندما نريد إدخال اسم الموظف نكتب

```
Cin >> emp . name;
```

لا نكتب الأقواس عندما نريد إدخال الاسم كاملاً كما تعلمنا في المصفوفات

ما عندما نريد إخراج أي قيمة فإننا نستبدل كما هو معروف cout بـ cin

ويمكننا أن نعلم عن التصريح بطريقة أخرى

```
Void main ( )
{
Struct employee
{
-----;
-----;
-----;
-----;
};
```

ونكمل البرنامج

مثال عملي:

اكتب برنامج يقوم بإدخال معلومات عن

١ -اسم الموظف

٢ –عنوان

٣ -عمره

٤ -راتب

طبعا لا نضع هنا فاصلة منقوطة

```
#include<iostream.h>
Struct employee
{
char name[40];
char address[40];
int age;
float salary;
};
void main()
{
struct employee emp;
cout<<"enter name"<<endl;
cin>>emp.name;
cout<<"enter address"<<endl;
cin>>emp.address;
cout<<"enter age"<<endl;
cin>>emp.age;
cout<<"enter salary"<<endl;
cin>>emp.salary;
}
```

كيفية التصريح عن مصفوفة سجلات

نفس الطريقة السابقة ولكن نجعل المتحول عبارة عن مصفوفة ونضيف سطر برمجي وهو حلقة for .

ولكن إن كان لدينا أكثر من موظف مثلاً ثلاثة عشر موظف فإننا بحاجة إلى مصفوفة سجلات

```
#include<iostream.h>
struct employee
{
char name[40];
char address[40];
int age;
float salary;
};
void main()
{ struct employee emp[13];
for(int i=0;i<13;i++)
      {  cout<<"enter name"<<endl;
            cin>>emp[i].name;
            cout<<"enter address"<<endl;
            cin>>emp[i].address;
            cout<<"enter age"<<endl;
            cin>>emp[i].age;
            cout<<"enter salary"<<endl;
            cin>>emp[i].salary;
       }
for(i=0;i<13;i++)
cout<<emp[i].name<<emp[i].address<<emp[i].age
<<emp[i].salary<<endl;
}
```

```
#include <iostream>
using namespace std;
struct Person
{
   char name[50];
   int age;
   float salary;
};

int main()
{
   Person p1;
   cout << "Enter Full name: ";
   cin.get(p1.name, 50);
   cout << "Enter age: ";
   cin >> p1.age;
   cout << "Enter salary: ";
   cin >> p1.salary;
   cout << "\nDisplaying Information." << endl;
   cout << "Name: " << p1.name << endl;
   cout <<"Age: " << p1.age << endl;
   cout << "Salary: " << p1.salary;
   return 0;
}
```

enumeration

enumeration لتعداد

enum، هو الذي تم تقديمه بواسطة الكلمة الأساسية
مجموعة من ثوابت التعداد الصحيح التي تمثلها المعرفات.
تبدأ القيم في التعداد بـ 0 ، ما لم يتم تحديد خلاف ذلك ،
وتزداد بمقدار 1. على سبيل المثال ، التعداد

```
enum months { JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP,
OCT, NOV, DEC };
```

To number the months 1 to 12, use:

```
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL,
AUG, SEP, OCT, NOV, DEC };
```

```
#include <iostream>

// enumeration constants represent months of the year enum
months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP,
OCT, NOV, DEC };
 int main(void)
 {
// initialize array of pointers
const char *monthName[] = { "", "January", "February",
"March", "April", "May", "June", "July", "August", "September",
"October", "November", "December" };
// loop through months

}
}
```

```
enum months { Jan, Feb, Mar, Apr, May, Jun,
Jul, Aug, Sep, Oct, Nov, Dec }

enum switch { off, on };
```

```
#include <iostream>
using namespace std;
//specify enum type
enum days_of_week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
int main()
{
days_of_week day1, day2;      //define variables
//of type days_of_week
day1 = Mon;      //give values to
day2 = Thu;      //variables
int diff = day2 - day1;       //can do integer arithmetic
cout << “Days between = “ << diff << endl;
if(day1 < day2)       //can do comparisons
cout << “day1 comes before day2\n”;
return 0;
}
```

```
#include <iostream>
#include <string>
struct Person{
 string name;
 int age;
 char gender;
};
int main(){
        Person p;
        p.name = "Christopher";
        p.age = 34;
        p.gender = 'M';
        cout << "Name: " << p.name << endl;
        cout << "Age: " << p.age << endl;
        cout << "Gender: " << p.gender << endl;
        return 0;
       }
```

مدخل الى البرمجة غرضية التوجه
Object oriented programming
oop

معنى البرمجة الكائنية OOP

أسلوب جديد في البرمجة من حيث وحدة بناء البرنامج ومن حيث الخصائص الجديدة التي يسمح بها هذا الأسلوب حيث يعتبر وحدة بناء البرنامج هو (الصنف) Class الذي يتكون من البيانات ومعها الدوال (العمليات) التي تعمل على هذه البيانات

ولها عدة مسميات منها :-
1. البرمجة غرضيه التوجه
2. البرمجة الموجهة نحو الأشياء
3. البرمجة الكائنية الموجهة

ويختلف البرنامج المكتوب بأسلوب OOPعن البرنامج المكتوب بالأسلوب الهيكلي فيما يلي :

1 - وحدة بناء البرنامج Procedural programming: كتابة البرنامج بالأسلوب الهيكلي عبارة عن داله رئيسيه ومجموعة دوال فرعيه , ويتم استدعاء الدوال الفرعية من داخل الدالة الرئيسية حسب تسلسل البرنامج .

2- أما البرنامج المكتوب بأسلوب OOP Object-Oriented programming فقد أصبحت وحدة البرنامج فيه هي الصنف Class التي تتكون من البيانات والدوال التي تعمل على هذه البيانات ويتم استدعاء الدوال كعناصر للأصناف

المفاهيم الأساسية في البرمجة الكائنية OOP:

الكائنobject . هو عبارة عن وحدة تحوي مجموعة من البيانات تسمى خصائص أو صفات و معرفه عليها مجموعة عمليات(دوال)

مثال:- طالب , قلم , حاسب .

الصنف Class عبارة عن نوع يحوي مجموعة من الكائنات التي تشترك في الخصائص والعمليات. مثال :- صنف الحاسبات , صنف الطالب .

والصنف يمثل المواصفات العامة للكائنات التي تنتمي لهذا الصنف , بينما الكائنات تمثل شيء قائم بذاته أو شيء له ذاتية تنتمي لذلك الصنف .

الفرق بين الصنف والكائن:

كل ما في الوجود هو كائن فأنا وأنت وهذه الورقة والقلم كلها كائنات Objects ولكل منها خصائص محددة ويستطيع القيام بعمليات محددة.

أما الصنف فهو مجموعة من الكائنات المتشابهة فالرجال صنف وزيد كائن منه والنساء صنف وأمل كائن منه.

مثال الصنف البرمجي: "بطاقة دوام" الذي يحوي الطر ق المطلوبة لحساب الأجر و عدد ساعات الدوام

فوائد البرمجة الشيئية(OOPs)

- **قابلية إعادة الاستخدام Reusability**: في برامج OOP ، يمكن إعادة استخدام الوظائف والوحدات النمطية التي كتبها مستخدم من قبل مستخدمين آخرين دون أي تعديل.
- **الميراث Inheritance:** من خلال هذا يمكننا التخلص من التعليمات البرمجية الزائدة وتوسيع استخدام الفئات الموجودة.
- **إخفاء البيانات Data Hiding**: يمكن للمبرمج إخفاء البيانات والوظائف في الفصل عن الفئات الأخرى. يساعد المبرمج على بناء البرامج الآمنة
- **تقليل تعقيد المشكلة Reduced complexity of a problem**: يمكن النظر إلى المشكلة المحددة على أنها مجموعة من الكائنات المختلفة. كل كائن مسؤول عن مهمة محددة. يتم حل المشكلة عن طريق ربط الكائنات. تقلل هذه التقنية من تعقيد تصميم البرنامج.
- **سهولة الصيانة والترقية** Easy to Maintain and Upgrade يجعل OOP من السهل صيانة وتعديل الكود الحالي حيث يمكن إنشاء كائنات جديدة مع اختلافات صغيرة عن تلك الموجودة. يمكن إدارة تعقيد البرامج بسهولة.
- **تمرير الرسائل Message Passing**إن تقنية اتصال الرسائل بين الكائنات تجعل الواجهة مع الأنظمة الخارجية أسهل.
- **قابلية التعديل Modifiability** من السهل إجراء تغييرات طفيفة في تمثيل البيانات أو الإجراءات في برنامج OOP. التغييرات داخل الصف لا تؤثر على أي جزء آخر من البرنامج ، الواجهة العامة التي يمتلكها العالم الخارجي للصف هي من خلال استخدام الأساليب.

بعض المفاهيم الأساسية في البرمجة الكائنية : OOP

الكائن object هو عبارة عن وحدة تحوي مجموعة من البيانات تسمى خصائص أو صفات attributes و معرفه عليها مجموعة عمليات Functions(دوال) مثال:- طالب , قلم , حاسب .
الصنف Class هو عبارة عن نوع يحوي مجموعة من الكائنات التي تشترك في الخصائص والعمليات. مثال - صنف الحاسبات , صنف الطالب .
والصنف يمثل المواصفات العامة للكائنات التي تنتمي لهذا الصنف , بينما الكائنات تمثل شيء قائم بذاته أو شيء له ذاتية تنتمي لذلك الصنف .

الفرق بين الصنف والكائن:
كل ما في الوجود هو كائن فأنا وأنت وهذه الورقة والقلم كلها كائنات Objects ولكل منها خصائص محددة ويستطيع القيام بعمليات محددة.

أما الصنف فهو مجموعة من الكائنات المتشابهة فالرجال صنف وزيد كائن منه والنساء صنف وأمل كائن منه.
وكمثال آخر الصنف البرمجي: "بطاقة دوام" الذي يحوي الطر ق المطلوبة لحساب الأجر و عدد ساعات الدوام

Classes Syntax in C++

```
Class class_name {
public:
public members
declarations
};
```

الصنف هو نوع معطيات جديد معرف من قبل المستخدم. **class**

تعرف المتغيرات ضمنا لصنف خاص Private لايمكن الوصول اليها من خارج الصنف

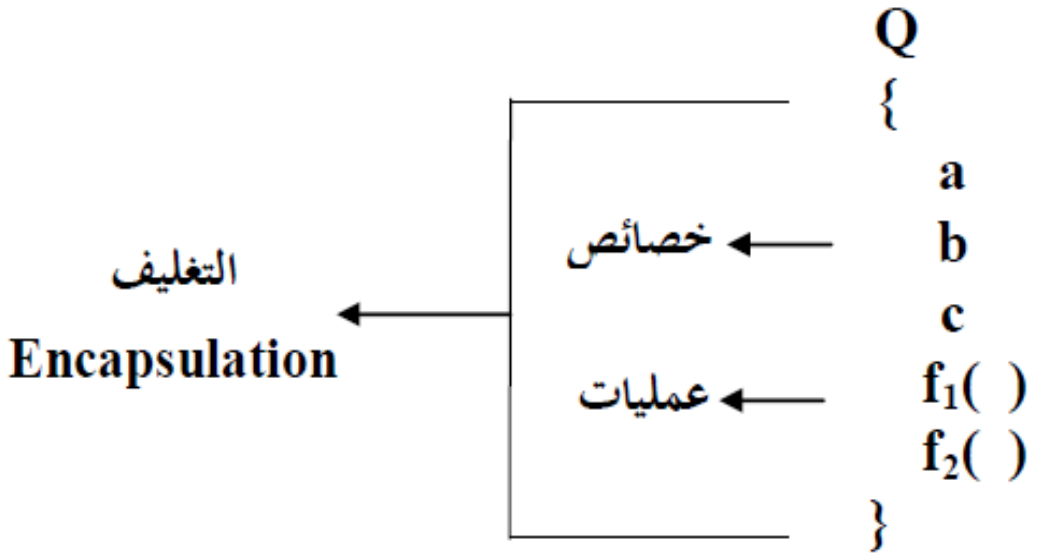
كلمة مفتاحية تحدد إمكانية الوصول الى أعضاء الصنف public

الأعضاء الموجودة القسم العام يمكن الوصول اليها من specifier access محدد وصول خارج الصنف

يمكن أن تكون خصائص الصنف: توابع ومتحولات

1- تعرف المتغيرات private الشكل الافتراضي في class هو private

2- تعرف التوابع public يمكن الوصول اليها من خارج الصنف



1- التجريد Abstraction

وهو عملية تحديد كل الخصائص والعمليات التي تنتمي لصنف أو كائن معني وهي نوعان :

- تجريد البيانات Data Abstraction
- تجريد العملياتMethods Abstraction

2 . الكبسلة (التغليف) Encapsulation

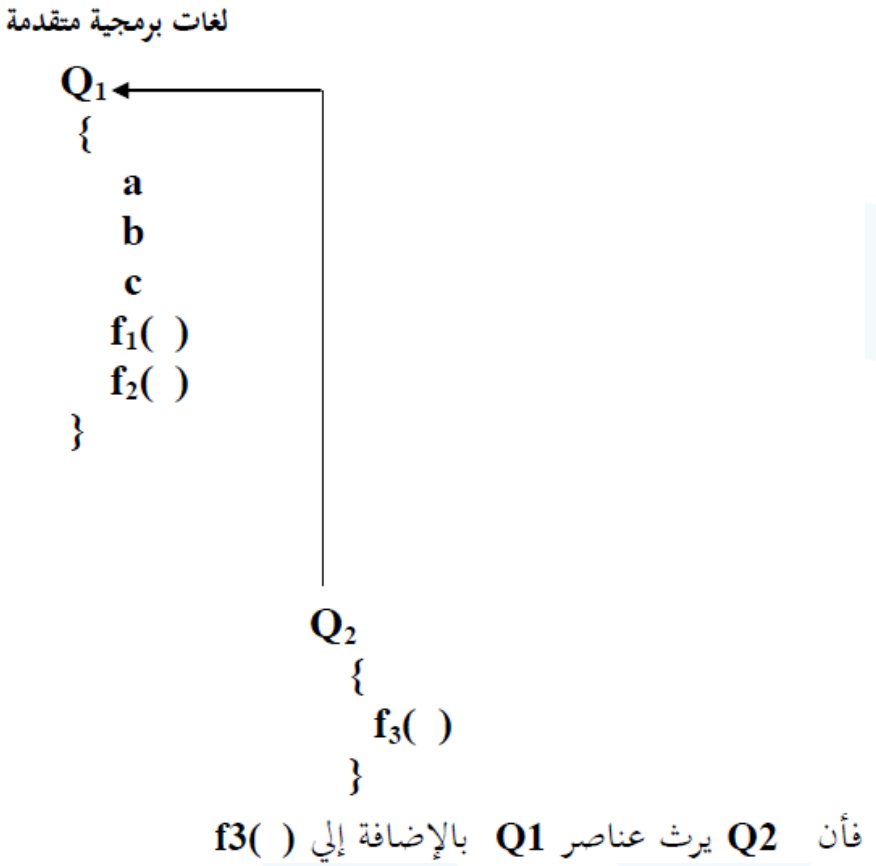
هي عملية تجميع كل الخصائص والعمليات في وحدة واحدة تسمى الصنف (داخل غلاف واحد) ولا يمكن الوصول إلى هذه الخصائص والعمليات إلا عن طريق الكائن

3- إخفاء البياناتData Hiding.

وهي ميزة ناتجة عن كبسلة البيانات وتعني إخفاء بعض البيانات وإضافة مستوى حماية معني عليها حتى نمنع الوصول الخطأ إليها

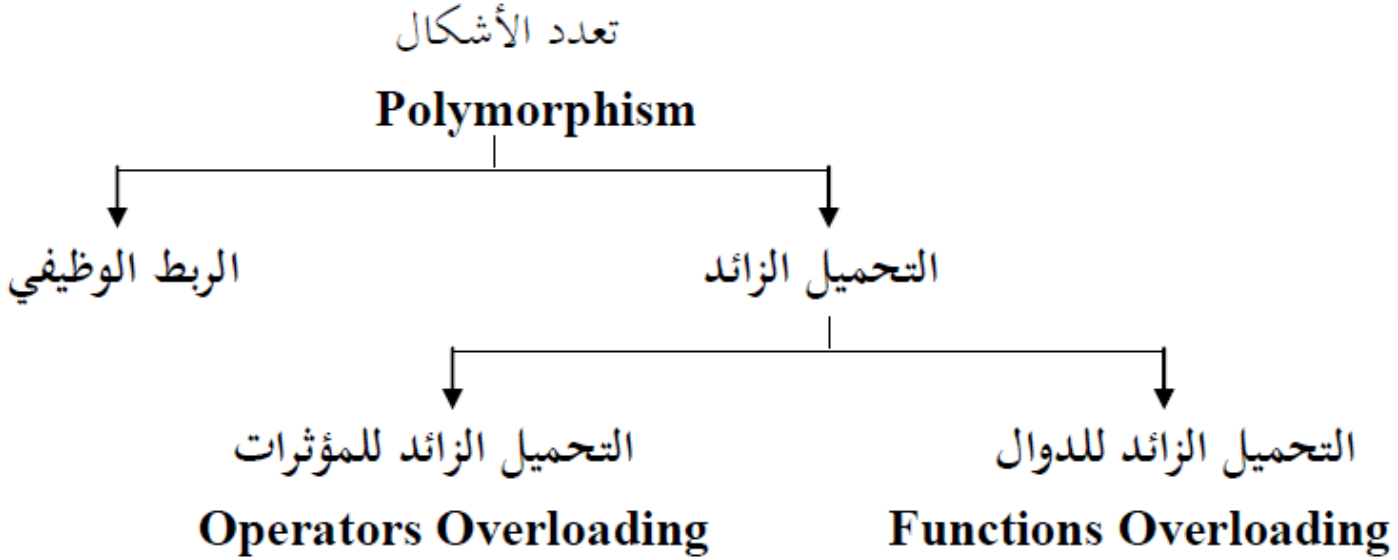
4- الوراثة Inheritance.

يرث صنف معني الخصائص والعمليات المعرفة في صنف آخر مما يساعد على إعادة استخدام الأصناف التي تم أنشاؤها من قبل



5. تعدد الأشكال Polymorphism

تمكن من إنشاء كائنات لها القدرة على القيام بأكثر من وظيفة أو إنشاء دوال لها القدرة على القيام بأكثر من وظيفة



مستويات الحماية (محددات الوصول داخل الأصناف)
هي عملية تحديد مدى التعامل مع الأعضاء (ولتكن البيانات) هل هذه البيانات خاصة بهذا الصنف أم لأي داله في أي صنف أن تتعامل مع هذه البيانات؟ وهي ثلاثة مستويات :

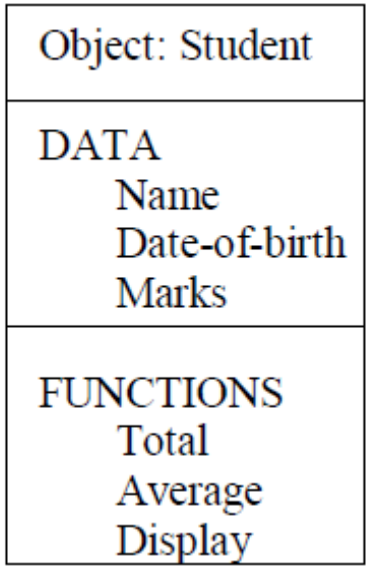
فائدة تحديد درجة الحماية :
يتم عادة تحديد درجة حماية من نوع Publicللأعضاء الدوال ودرجة حماية من نوع private للأعضاء المتغيرات

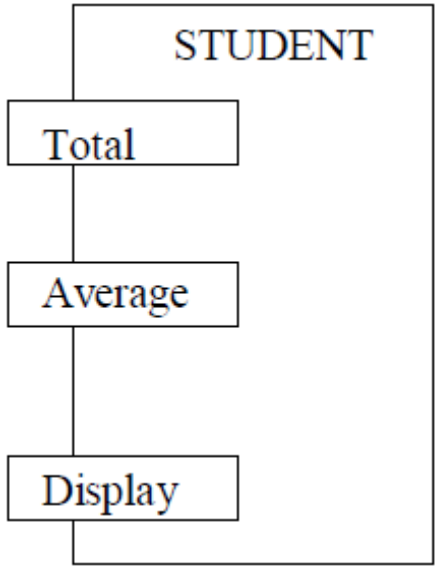
1. **مستوى الحماية الخاص Private**
يستخدم مستوى الحماية الخاص لتعريف الأعضاء الذي لا يمكن الوصول إليها من خارج الصنف .
2. **مستوى الحماية المحمي Protected**
يشبه مستوى الحماية الخاص غير أنه يمكن توريثه إلي صنف آخر .
3. **مستوى الحماية العام Public**
يستخدم لتعريف الأعضاء التي يمكن الوصول إليها من خارج الصنف ويمكن تور يثها إلي صنف آخر

ومن فوائد إخفاء البيانات تقليل الخطأ في استخدام البيانات حيث تثل الدوال واجهة الاستخدام الوحيدة مع البيانات

ميزات البرمجة OOP

1. التركيز على العمل وليس على الإجراء.
2. تنقسم البرامج إلى ما يعرف بالكائنات.
3. تم تصميم هياكل البيانات بحيث تميز الأشياء.
4. ترتبط الدوال التي تعمل على بيانات كائن معًا في بنية البيانات.
5. البيانات مخفية ولا يمكن الوصول إليها بواسطة دوال خارجية.
6. تتواصل الكائنات مع بعضها البعض من خلال الدوال.
7. يمكن إضافة بيانات ودوال جديدة بسهولة.
8. يتبع النهج التصاعدي في تصميم البرنامج





قد يكون جزء البرنامج من الكائن عبارة عن مجموعة من البرامج (استعادة البيانات ، وتغيير العمر ، وتغيير الراتب). بشكل عام ، يمكن استخدام أي نوع محدد من قبل المستخدم مثل الموظف. في كائن أميت ، يُطلق على الاسم والعمر والراتب سمات الكائن.

الصنف عبارة عن مجموعة من الكائنات التي تشترك في خصائص وعلاقات مشتركة.

في لغة C ++ ، الصنف عبارة عن نوع بيانات جديد يحتوي على متغيرات الأعضاء ووظائف الأعضاء التي تعمل على المتغيرات.

يتم تحديد الصنف مع صنف الكلمة الأساسية. يسمح بإخفاء البيانات ، إذا لزم الأمر من الاستخدام الخارجي.

تتكون مواصفات الصنف بشكل عام من جزأين- :

أ) إعلان الصنف

ب) تعريف وظيفة الصنف

يصف إعلان الصنف نوع ونطاق أعضائه. يصف تعريف دالة الصنف كيفية تنفيذ دوال الصنف

Syntax:-

```
class class-name
        {
        private:
                variable declarations;
                function declaration ;
         public:
         variable declarations;
         function declaration;
          };
```

يمكن الوصول إلى الأعضاء الذين تم إعلانهم على أنهم private فقط من خلال الصنف.

يمكن الوصول إلى الأعضاء public من خارج الصف أيضًا.