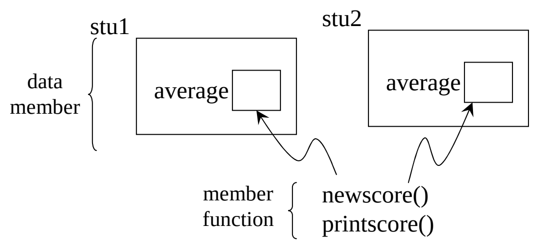
C++ 的類別 (Class)

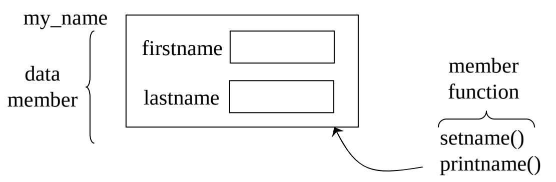
- ☐ 類別(Class) 是一種資料型態,可用來宣告物件
- ☐ 類別內含有資料成員(Data member) 和成員函式(Member function)
- ☐ 類別中不論是 Data Member 或 Member function 都可在 public 區或 private區宣告
- ☐ 因OOP特性之一是隱藏資料,一般會將 data member 以 private方式宣告保護起來,並將 member function開放給外界操作
- ☐ C++ 提供 Private, Protected 和 Public 來設定成員的保護等級
  - ❍ Private(私有)
    只有類別中的 member function 才可直接使用(存取)資料成員
  - ❍ Protected(保護):
    僅 member function 及繼承此類別之 member function 可直接使用資料成員
  - ❍ Public(共用):
    任何函式或敘述均可直接使用資料成員,存取方式為 物件.成員
- ☐ Class 預設的保護等級為 Private
- ☐ 類別函數的存取必須透過屬於該類別型態的物件和點運算子.
  類別函數的取用方式: o.f 或 p->f
  其中 o表物件名稱, f 表類別函數, p表指標
- ☐ 定義在Class 內的 function 為 inline function, 在 class 外每次呼叫該 function 時,其 function code 會在呼叫處展開一次.

```
Class  類別型態變數名稱
{  private:
      私有資料成員變數之宣告;
      私有成員函式宣告及定義;
   public:
      公用成員函式;
}  類別變數(即物件);
```



```
#include <iostream.h>    //Page:7-4
class  score
{ private:
       float average;  //私有資料成員
   public:
       void newscore(float avg)
             { average=avg; }
       void printscore()
             { cout << "Average of score:";
               cout << average << endl;
              }
};
```

```
void main()
{ score  stu1,stu2; //產生兩個物件
 //透過 member function 來存取 data member
   stu1.newscore(88.5); // 物件. Member function
   stu2.newscore(92.5);
   stu1.printscore();
   stu2.printscore();
   average=88.5;          } 錯誤的寫法
   cout << average;
}
```



```
#include <iostream.h>
class Name
{ private:
    char firstname[10];
    char lastname[10];
  public:
    void setname()
    { cout << "What's your first name:";
      cin >> firstname;
      cout << "What's your last name:";
      cin >> lastname; }
    void printname()
    { cout << "\n The name is:"
                  << firstname << ' '
                  << lastname << '\n'; }
 };
void main()
{ Name my_name;
  my_name.setname();
  my_name.printname();
}
```

Member function 若設計在 class 內,
呼叫此 member function 時 compiler
以 inline 方式處理。

```
void main()
{ Name my_name;
  //  my_name.setname();
       cout << "What's your first name:";
       cin >> firstname;
       cout << "What's your last name:";
       cin >> lastname;
  // my_name.printname();
       cout << "\n The name is:"
            << firstname << ' '
            << lastname << '\n';
}
```

Data member (資料成員)的使用

☐ data member 若在 private 區
- ⌧ 僅可供 member function 直接取用

☐ data member 若在 protected 區
- ⌧ 可供 member function 直接取用
- ⌧ 可供繼承此類別之 類別其 member function直接取用

☐ data member 若在 public 區
- ⌧可供 member function 直接取用
- ⌧可供繼承此類別之 類別其 member function直接取用
- ⌧可供一般的敘述或 function以 物件.資料成員 方式取用

Member function(成員函式) 的使用

☐ member function 若在 private 區
- ⌧ 僅可供 member function 直接呼叫

☐ member function 若在 protected 區
- ⌧ 可供 member function 直接呼叫
- ⌧ 可供繼承此類別之 類別其 member function直接呼叫

☐ data member 若在 public 區
- ⌧可供 member function 直接取用與繼承此類別 之member function直接取用
- ⌧可供一般的敘述或 function以 物件.成員函數 方式取用

```
Class Base
{ private:
    int a;
    void func1(){…}
  protected:
    int b;
    void func2(){…}
   public:
    int c;
    void func3(){…}
};
void main()
{…
}
```

私有區data及function
僅供自己類別內存取

```
void func1(){a=2;}
void func2(){a=3; func1();}
void func3(){a=3; func1();}
```

```
void main()
{a=2;
 func1();}
```

保護區data及function僅供
自己類別內及繼承之兒子存取

```
void func2(){b=2;}
void func3(){a=3; func1();}
```

```
void main()
{b=2;}
```

公用區data及function供
所有敘述及函數存取

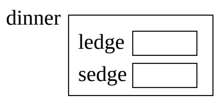
```
void func3(){c=3; func1();}
void main()
{ base bs1;
   bs1.c=3;
   bs1.func3();}
```

```
void main()
{base bs1;
  c=2;
  func3();}
```

//Private, Protected, Public 的差別

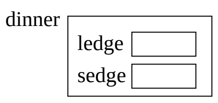
```
#include <iostream.h>
class test1
{ private:
    int a;
    void func1()
      { cout << "private function in test1\n"; }
  protected:
    int b;
    void func2()
      { func1();
        cout << "protected function in test1\n";}
  public:
    int c;
    void func3()
      { func2();
        a=1;b=2;c=3;
        cout << "a=" << a << "b=" << b
              << "c=" << c << endl;}
    void func4()
      { func3(); }
};
class test2:private test1
{ private:
    int d;
    void func5()
      { cout << "private function in test2\n";}
  public:
    int f;
    void func7()
      { // func1(); error!
         func2();
         // a=4; error!
         b=5;  c=6; }
};
 void main()
 { test1 t1;
   test2 t2;
   // t1.func1(); error!
   // t1.func2(); error!
   // t1.a=5; error!
   // t1.b=6; error!
   t1.c=7;
   t1.func3();
   t2.func7();
 }
```

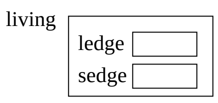
```
#include <iostream.h>  // Page 7-6
class score
{ private:
    long number;
    float average;
  public:
    void newscore()   //輸入學號及平均
      { cout << "enter number:";
                   cin >> number;
                   cout << "enter average:";
                   cin >> average;
                  }
     void print_score()   //印出學號及平均
     { cout << "student number is:" << number;
                  cout << "student average:" << average;
      }
};

main()
{ score stu1,stu2;
  stu1.newscore();
  stu2.newscore();
  stu1.print_score();
  stu2.print_score();
}
```



```
// 類別的定義與使用

#include <iostream.h>
class room
{  private:
      float ledge;   // 長
      float sedge;   // 寬
   public:
      void setlength(float le, float se)  //設定長寬
      { ledge=le;  sedge=se; }

      void showsquare()  //計算面積並印出
      { cout << ledge*sedge << endl;}
 }  dinner;
void main()
{ dinner.setlength(5.0,3.5);
  dinner.showsquare();
}
```

```
// 類別的定義與使用 Page:7-10
/* data member 之值可以在class 變數宣告時
   給值,亦可透過 member function 供使用
   者輸入 */

#include <iostream.h>
class room
{ private:
    float ledge;
    float sedge;
  public:
    void setlength(float le, float se)
    { ledge=le; sedge=se }
```





```
    void getlength()
    { cout << "Input large edge:";
      cin >> ledge;
      cout << "Input small edge:";
      cin >> sedge; }

    void showsquare()
    { cout << ledge*sedge << endl; }
 }  dinner;

void main()
{ room living;
  dinner.setlength(3.2,3.1);
  living.getlength();
  cout << "Square of dinner room is:";
  dinner.showsquare();
  cout << "Square of living room is:";
  living.showsquare();
}
```

□ 雙冒號:: 是範圍解析算符(Scope resolution operator),可用來表明該函式是屬於那一個類別的成員。當我們在類別以外的地方定義函式內容時，一定要用 :: 來指明所屬的類別。而呼叫此成員函式的方式為非 inline 方式。

```
class ID
{  int  id_no;
   public:
      void  set(int  i)
      { id_no=i; }

      int  get()
      {
        return(id_no);
      }
};
```

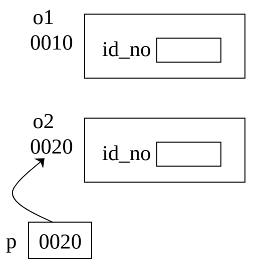
```
class ID
{  int  id_no;
   public:
      void  set(int  i);
      int  get();
};

void ID::set(int  i)
{  id_no=i; }

int ID::get()
{  return  id_no; }
```

```
class ID
{  int  id_no;
   public:
      void  set(int  i);
      int  get();
};

inline void ID::set(int  i)
{  id_no=i; }

int ID::get()
{  return  id_no; }
```

□ 類別函數的存取必須透過屬於該類別型態的物件和點運算子.
類別函數的取用方式: o.f 或 p->f
其中 o表物件名稱, f 表類別函數, p表指標

```
#include <iostream.h>
class ID
{
 private:
    int id_no;
 public:
    void set(int i)
    { id_no=i; }

    int get()
    { return(id_no); }
};
```

```
void main()
{ ID  o1,o2,*p=&o2;
  o1.set(1);
  p->set(2);
  cout << "o1=" << o1.get() << endl;
  cout << "o2=" << o2.get() << endl;
  cout << "*p=" << p->get() << endl;
}
```



類別的建構函數與解建構函數

- 建構函數的名稱必須與類別名稱相同
- 建構函數不能有傳回值
- 建構函數可以接受參數以作為資料成員設定初值之用
- 在宣告類別變數(物件)時，系統會自動執行建構函數

```
// 不用建構函數來設定初值的方法
#include <iostream.h>
class room
{  private:
      float ledge;   // 長
      float sedge;   // 寬
   public:
      void setlength(float le, float se)  //設定長寬
      { ledge=le;  sedge=se; }
      void showsquare()  //計算面積並印出
      { cout << ledge*sedge << endl;}
 } ;
void main()
{   room  dinner;
     dinner.setlength(5.0,3.5);
     dinner.showsquare();
}
```

```
// 用建構函數來設定初值的方法
#include <iostream.h>
class room
{ private:
     float ledge,sedge;
   public:
     room()
     { ledge=6.0;
       sedge=4.8; }
     void showsquare()
     { cout << ledge * sedge < endl; }
};
void main()
{ room dinner;
  cout << "square of dinner room is:";
  dinner.showsquare();
}
```

- 建構函數可以接受參數以作為資料成員設定初值之用
- 在指定物件初始值時,可以用 “=初值” 或 “(初值)”來表明,但若同時要設定多個資料成員的初值,只有用小括號方式才行.

```
// 用建構函數之參數來設定初值的方法
// 一個初值的設定
#include <iostream.h>
class room
{ private:
     float edge;
  public:
     room(float a)
     { edge=a; }
     void showsquare()
     { cout << edge * edge << endl; }
};
void main()
{ room dinner=6.0,living(5.0);
  cout << "square of dinner room is:";
  dinner.showsquare();
  cout << "square of living room is:";
  living.showsquare();  }
```

```
// 用建構函數之參數來設定初值的方法
// 兩個以上的初值
#include <iostream.h>  //Page: 7-29
class room
{ private:
     float ledge,sedge;
  public:
     room(float le,float se)
     { ledge=le;
       sedge=se; }
     void showsquare()
     { cout << ledge * sedge << endl; }
};
void main()
{ room dinner(6.5,4.8);
  cout << "square of dinner room is:";
  dinner.showsquare();
}
```

```
// 用建構函數範例  Page:7-18,7-25
#include <iostream.h>
#include <conio.h>
class counter
{ private:
    unsigned int count;
  public:
    counter()  // constructor
    { count=0; }
  void countchar();
    int getcount()
    { return count; }
};
void counter::countchar()
{  char ch;
    cout << "\nPlease enter a string: \n";
    while ((ch=getche())!='\r')
        { count++; }
}
void main()
{ counter c1;
  c1.countchar();
  cout << "\n Consists " << c1.getcount();
  cout << "characters" << endl;
}
```

- 一個類別可以有一個以上的 constructor 我們稱為 overloaded constructor,　只要 constructor 之引數個數或資料型態不一樣, 則 compiler 便可視為不同之 constructor

```
#include <iostream.h>
class String
{ char *str;
  public:
     String();
     String(char *);
     void print()
     { cout << str << endl;}
};
String::String()
{ str="abcde"; }
String::String(char *ptr)
{ str=ptr; }
void main()
{String a; // call String()
 String b("xyz"); // call String(char *)
 a.print();
 b.print();
}
```

- 另一種 Constructor 初始值的設定方法:將初值設定在 constructor 之引數中, 呼叫時有設初值的引數可省略不寫。

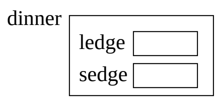
```
#include <iostream.h>
class Time
{ private:
    int hour,minute,second;
  public:
    Time(int hr=0, int min=0,int sec=0)
    { hour=hr; minute=min; second=sec; }
    void print()
    { cout << hour << ":"
           << minute << ":"
           << second << endl;}
};
void main()
{Time t1,t2(2),t3(21,34),t4(12,25,42);
 t1.print();
 t2.print();
 t3.print();
 t4.print();
}
```

```
#include <iostream.h>
class Time
{ private:
    int hour,minute,second;
  public:
    Time(int hr, int min,int sec)
    { hour=hr; minute=min; second=sec; }
    void set(int hr, int min, int sec)
   {hour=hr; minute=min; second=sec; }
    void print()
    { cout << hour << ":"
           << minute << ":"
           << second << endl;}
};
void main()
{Time t1,t2(2),t3(21,34)  // error!
  Time  t4(12,25,42);
  t4.print();
  t4.set(13,24,55);
  t4.print();
}
```

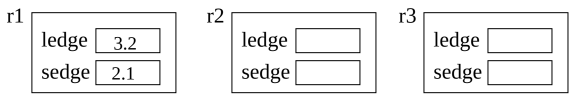
- 建構函數會在物件宣告後自動執行,可以有參數但不可有傳回值
- 解建構函數(Destructor)會在物件消失時自動執行
- 解建構函數不可有參數亦不可有傳回值
- 解建構函數的名稱和類別名稱相同,但其前須加上‘~’符號
- 解建構函數之執行會將建構函數所配置的物件記憶體空間釋回

```
#include <iostream.h> //7-22
class room
{ private:
     float  ledge,sedge;
   public:
     room()
     { ledge=6.0;
        sedge=4.8; }
    float showsquare()
       { return ledge * sedge; }
    ~room()
       { cout << "Object deallocated"; }
  };
```

```
void main()
{ room dinner;
  cout << "square of dinner room is:";
  cout << dinner.showsquare() << endl;
}
```



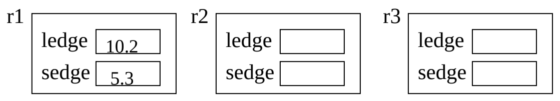
- 以物件作為函數之參數

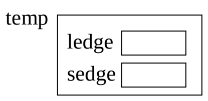


```
#include <iostream.h> //7-32
#include <iomanip.h>
class room
{ private:
    float ledge,sedge;
  public:
    room() {}
    room(float le,float se)
    { ledge=le; sedge=se; }
    void getlength()  //輸入物件長與寬
    { cout << "Input large edge:";
      cin >> ledge;
      cout << "Input small edge:";
      cin >> sedge;
    }
    void showsquare()  //計算面積並顯示
    { cout << setprecision(3) << ledge*sedge << endl;}
    void addsquare(room r1,room r2);
};
```

```
void room::addsquare(room r1,room r2)
// 將r1及r2兩物件之長寬分別加總後
// 存入本物件之長與寬並印出本物件周長
{ ledge=r1.ledge+r2.ledge;
  sedge=r1.sedge+r2.sedge;
  cout << endl << "Total of room length: ";
  cout << setprecision(3) << (ledge+sedge)*2 << endl;
};

void main()
{ room r2,r3;
  room r1(3.2,2.1);
  r2.getlength();
  cout << "\nSquare of r1 room is: ";
  r1.showsquare();
  cout << "\nSquare of r2 room is: ";
  r2.showsquare();
  cout << "\nSquare of r3 room is: ";
  r3.addsquare(r1,r2);
}
```

☐ 以物件作為函數之傳回值





```
#include <iostream.h> //7-37
#include <iomanip.h>
class room
{ private:
    float ledge;
    float sedge;
  public:
    room() {}
    room(float le,float se)  // 供宣告物件時給長寬值
    { ledge=le; sedge=se; }
    void getlength()  // 供輸入長寬值
    { cout << "Input large edge:";
      cin >> ledge;
      cout << "Input small edge:";
      cin >> sedge; }
    void showlength() // 顯示物件周長
    { cout << "Total of room length:"
           << setprecision(3)
           << (ledge+sedge)*2 << endl; }
room tlength(room r2)
// 將物件 r2 之長寬加上本物件之長寬並
// 存入 temp 物件之長寬後傳回 temp 物件
    { room temp;
      temp.ledge=ledge+r2.ledge;
      temp.sedge=sedge+r2.sedge;
      return temp; }
};

void main()
{ room r2;
  room r1(10.2,5.3);
  cout << "Length of r2 room:\n";
  r2.getlength();
  room r3=r1.tlength(r2);
  r3.showlength();
}
```

C++之動態記憶體配置

☐ C 之動態記憶體配置函數為 指標變數=malloc(容量) 及 free(指標變數), malloc 通常搭配 sizeof(型態變數) 以配置程式師所指定的記憶體容量。

例如: int *ptr;

$$\text{ptr=(int) malloc(}\underbrace{\text{10*}\underbrace{\text{sizeof(int)}}_{\text{2 bytes}}}_{\text{20 bytes}}\text{);}$$

以 sizeof 將 int 之bytes 數算出,以 malloc 將使電腦配置 20 bytes 記憶體並將起始位址存入指標變數 ptr 中。

☐ C++之動態記憶體配置指令為為 New 與 Delete

可省略

- 🖂 格式一: 指標變數= new 基本型態變數 $\boxed{\text{[個數]}}$
- 🖂 格式二: 指標變數= new 自定型態變數 $\boxed{\text{(初始化之值)}}$
- 🖂 將new 所配置之記憶體釋回: delete 指標變數

例: int *ptr;	例: float *ptr;
ptr=new int [100];	ptr=new float (3.14);
…	...
delete ptr;	delete ptr

- 通常利用 constructor 來配置記憶體,並利用 destructor 來釋回

```
#include <iostream.h>
#include <string.h>
class Strings
{ private:
    char *str;
  public:
    Strings()
      { strcpy(str,""); }
    Strings(char *st)
      { str=new char[strlen(st)+1];
        strcpy(str,st); }
    ~Strings()
      { delete str; }
    void set(char *ptr)
      { str=new char[strlen(ptr)+1];
        strcpy(str,ptr); }
    void printstr()
      { cout << str << endl; }
};
```

```
void main()
{ char *title="London bridge is falling down !";
  Strings ps1;  // call String()
  ps1.set("London bridge");         / /use  m.f.  for initial
  ps1.printstr();
  Strings ps2(title);     // call Strings(char *st) for initial
  ps2.printstr();
  Strings ps3("falling down"); //call String(char *st)
  ps3.printstr();                        for initial
  Strings ps4="is falling down"; //call String(char *st)
  ps4.printstr();                          for initial
}
```

成員的初始化串列

- 當我們用類別來定義物件時,系統會先為類別內的資料成員配置好記憶體空間,然後再呼叫適當的建構函數來設定初值。然而,有時我們會希望系統在配置空間時能同時作初始化的工作,這時就可以用“成員初始化串列”。
- 成員初始化串列必須出現在 constructor 的定義(而非宣告)之中:
  constructor 名稱(參數…): 資料成員名稱(初值運算式)...

```
#include <iostream.h>
#include <conio.h>
class Test
{ private:
    int i;
    int &ri;
    const int ci;
  public:
    Test(int a, int &b, int c);
    void Put()
    { cout << "i=" << i << endl;
      cout << "ri=" << ri << endl;
      cout << "ci=" << ci << endl; }
};
Test::Test(int a, int &b, int c) :  ri(b),ci(c)
{ i=a; }
```

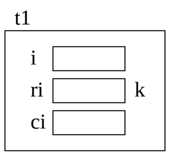
配置順序

初始化串列

```
void main()
{ clrscr();
  int k=4;
  Test t1(2,k,6);
  t1.Put();
}
```

```
Output:
i=2
ri=4
ci=6
```

✶ 建立 i 之空間
✸ 建立 ri, 並設定 ri 為 b之 reference(綽號)
✹ 建立 ci, ci ← c
❈ 執行 constructor, i ← a



每個資料成員在串列中最多只能出現一次,初值的運算可以是常數、變數或複雜運算式,其排列次序不重要,系統為資料配置時依他們在類別定義(宣告) 中出現的順序來執行