程序设计实习

C++ 面向对象程序设计

张勤健
zqj@pku.edu.cn

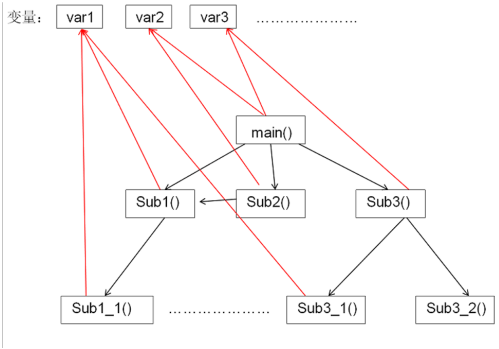
北京大学信息科学技术学院

2025 年 2 月 19 日

大纲

结构化程序设计

C 语言使用结构化程序设计：
程序 = 数据结构 + 算法
程序由全局变量以及众多相互调用的函数组成。
算法以函数的形式实现，用于对数据结构进行操作。



结构化程序设计的不足

结构化程序设计中，函数和其所操作的数据结构，没有直观的联系。
随着程序规模的增加，程序逐渐难以理解，很难一下子看出来：

- 某个数据结构到底有哪些函数可以对它进行操作?
- 某个函数到底是用来操作哪些数据结构的?
- 任何两个函数之间存在怎样的调用关系?

结构化程序设计的不足

结构化程序设计没有“封装”和“隐藏”的概念。要访问某个数据结构中的某个变量，就可以直接访问，那么当该变量的定义有改动的时候，就要把所有访问该变量的语句找出来修改，十分不利于程序的维护、扩充。

结构化程序设计的不足

重用：在编写某个程序时，发现其需要的某项功能，在现有的某个程序里已经有了相同或类似的实现，那么自然希望能够将那部分代码抽取出来，在新程序中使用。

在结构化程序设计中，随着程序规模的增大，由于程序大量函数、变量之间的关系错综复杂，要抽取这部分代码，会变得十分困难。

结构化程序设计的不足

综上，规模庞大时:

- 会变得难以理解
- 难以扩充（增加新功能）
- 难以查错
- 难以重用

面向对象的程序设计

面向对象的程序 = 类 + 类 + ...+ 类
设计程序的过程，就是设计类的过程。

面向对象的程序设计

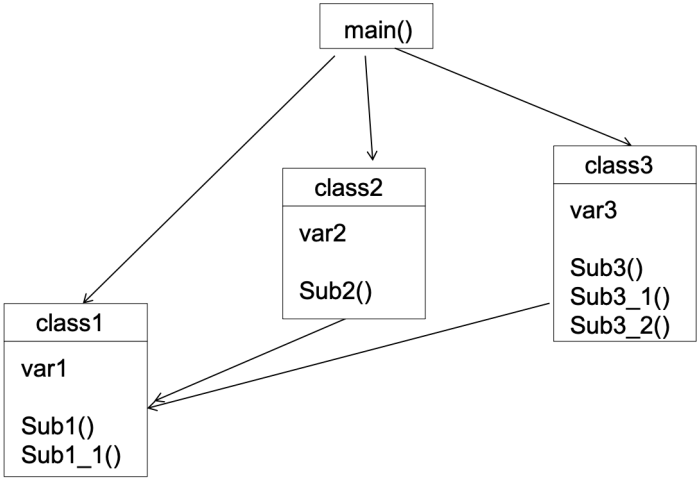
面向对象的程序设计方法：

- 将某类客观事物共同特点（属性）归纳出来，形成一个数据结构（可以用多个变量描述事物的属性）；
- 将这类事物所能进行的行为也归纳出来，形成一个个函数，这些函数可以用来操作数据结构。

然后，通过某种语法形式，将数据结构和操作该数据结构的函数“捆绑”在一起，形成一个“类”，从而使得数据结构和操作该数据结构的算法呈现出显而易见的紧密关系，这就是“封装”。

面向对象的程序设计具有“抽象”，“封装”“继承”“多态”四个基本特点。

面向对象的程序模式


main()
class1
var1
Sub1()
Sub1_1()
class2
var2
Sub2()
class3
var3
Sub3()
Sub3_1()
Sub3_2()

从客观事物抽象出类

写一个程序，输入矩形的长和宽，输出面积和周长。

从客观事物抽象出类

写一个程序，输入矩形的长和宽，输出面积和周长。

比如对于“矩形”这种东西，要用一个类来表示，该如何做“抽象”呢?

从客观事物抽象出类

写一个程序，输入矩形的长和宽，输出面积和周长。

比如对于“矩形”这种东西，要用一个类来表示，该如何做“抽象”呢?

矩形的属性就是长和宽。因此需要两个变量，分别代表长和宽。

从客观事物抽象出类

写一个程序，输入矩形的长和宽，输出面积和周长。

比如对于“矩形”这种东西，要用一个类来表示，该如何做“抽象”呢?

矩形的属性就是长和宽。因此需要两个变量，分别代表长和宽。

一个矩形，可以有哪些行为呢（或可以对矩形进行哪些操作)?

从客观事物抽象出类

写一个程序，输入矩形的长和宽，输出面积和周长。

比如对于“矩形”这种东西，要用一个类来表示，该如何做“抽象”呢?

矩形的属性就是长和宽。因此需要两个变量，分别代表长和宽。

一个矩形，可以有哪些行为呢（或可以对矩形进行哪些操作)?
矩形可以有设置长和宽，算面积，和算周长这三种行为（当然也可以有其他行为)。
这三种行为，可以各用一个函数来实现，他们都需要用到长和宽这两个变量。

从客观事物抽象出类

将长、宽变量和设置长，宽，求面积，以及求周长的三个函数“封装”在一起，就能形成一个“矩形类”。

从客观事物抽象出类

将长、宽变量和设置长，宽，求面积，以及求周长的三个函数“封装”在一起，就能形成一个“矩形类”。

长、宽变量成为该“矩形类”的“成员变量”，三个函数成为该类的“成员函数”。成员变量和成员函数统称为类的成员。

从客观事物抽象出类

将长、宽变量和设置长，宽，求面积，以及求周长的三个函数“封装”在一起，就能形成一个“矩形类”。

长、宽变量成为该“矩形类”的“成员变量”，三个函数成为该类的“成员函数”。成员变量和成员函数统称为类的成员。

实际上，“类”看上去就像“带函数的结构”。

从客观事物抽象出类

```
4  class CRectangle {
5  public:
6    int w, h;
7    int Area() {
8      return w * h;
9    }
10   int Perimeter() {
11     return 2 * (w + h);
12   }
13   void Init(int w_, int h_) {
14     w = w_;
15     h = h_;
16   }
17 }; //必须有分号
18 int main() {
19   int w, h;
20   CRectangle r; // r 是一个对象
21   cin >> w >> h;
22   r.Init(w, h);
23   cout << r.Area() << endl << r.Perimeter();
24   return 0;
25 }
```

对象

通过类，可以定义变量。类定义出来的变量，也称为类的实例，就是我们所说的“对象”。C++ 中，类的名字就是用户自定义的类型的名字。可以像使用基本类型那样来使用它。CRectangle 就是一种用户自定义的类型。

对象的内存分配

和结构变量一样，对象所占用的内存空间的大小，等于所有成员变量的大小之和。对于上面的 CRectangle 类，sizeof(CRectangle) = 8
每个对象各有自己的存储空间。一个对象的某个成员变量被改变了，不会影响到另一个对象。

对象间的运算

和结构变量一样，对象之间可以用“=”进行赋值，但是不能用“==”，“!=”，“>”，“<”“>=”“<=”进行比较，除非这些运算符经过了“重载”。

使用类的成员变量和成员函数

用法 1：对象名. 成员名

```
1  CRectangle r1, r2;
2  r1.w = 5;
3  r2.Init(5, 4);
```

Init 函数作用在 r2 上，即 Init 函数执行期间访问的 w 和 h 是属于 r2 这个对象的, 执行 r2.Init 不会影响到 r1。

使用类的成员变量和成员函数

用法 2. 指针-> 成员名

```
1  CRectangle r1, r2;
2  CRectangle *p1 = &r1;
3  CRectangle *p2 = &r2;
4  p1->w = 5;
5  p2->Init(5, 4); // Init 作用在 p2 指向的对象上
```

Init 函数作用在 r2 上，即 Init 函数执行期间访问的 w 和 h 是属于 r2 这个对象的, 执行 r2.Init 不会影响到 r1。

使用类的成员变量和成员函数

用法 3：引用名. 成员名

```
1  CRectangle r2;
2  CRectangle &rr = r2;
3  rr.w = 5;
4  rr.Init(5, 4); // rr 的值变了，r2 的值也变
5
6  void PrintRectangle(CRectangle &r) {
7    cout << r.Area() << "," << r.Perimeter();
8  }
9  CRectangle r3;
10 r3.Init(5, 4);
11 PrintRectangle(r3);
```

引用的概念

下面的写法定义了一个引用，并将其初始化为引用某个变量。

```
类型名 & 引用名 = 某变量名;
```

引用的概念

下面的写法定义了一个引用，并将其初始化为引用某个变量。

```
类型名 & 引用名 = 某变量名;
```

```
int n = 4;
int &r = n;  // r 引用了 n, r 的类型是 int &
```

引用的概念

下面的写法定义了一个引用，并将其初始化为引用某个变量。

```
类型名 & 引用名 = 某变量名;
```

```
int n = 4;
int &r = n;  // r 引用了 n, r 的类型是 int &
```

某个变量的引用，等价于这个变量，相当于该变量的一个别名。

引用的概念

```
1  int n = 4;
2  int &r = n;
3  r = 4;
4  cout << r;
5  cout << n;
6  n = 5;
7  cout << r;
```

引用的概念

- 定义引用时一定要将其初始化成引用某个变量。
- 初始化后，它就一直引用该变量，不会再引用别的变量了。
- 引用只能引用变量，不能引用常量和表达式。

引用的概念

```
1  double a = 4, b = 5;
2  double &r1 = a;
3  double &r2 = r1;
4  r2 = 10;
5  cout << a << endl;
6  r1 = b;
7  cout << a << endl;
```

引用的概念

```
1  double a = 4, b = 5;
2  double &r1 = a;
3  double &r2 = r1;
4  r2 = 10;
5  cout << a << endl;
6  r1 = b;
7  cout << a << endl;
```

输出

```
10
5
```

引用应用的简单示例

C 语言中，如何编写交换两个整型变量值的函数?

```
1  void swap(int *a, int *b){
2    int tmp;
3    tmp = *a;
4    *a = *b;
5    *b = tmp;
6  }
7  int n1, n2;
8  swap(&n1, &n2); // n1,n2 的值被交换
```

引用应用的简单示例

C 语言中，如何编写交换两个整型变量值的函数?

```c
void swap(int *a, int *b){
  int tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;
}
int n1, n2;
swap(&n1, &n2); // n1,n2 的值被交换
```

有了 C++ 的引用:

```c
void swap(int &a, int &b) {
  int tmp;
  tmp = a;
  a = b;
  b = tmp;
}
int n1, n2;
swap(n1, n2); // n1,n2 的值被交换
```

引用作为函数的返回值

```
1  int n = 4;
2  int &SetValue() {
3    return n;
4  }
5  int main() {
6    SetValue() = 40;
7    cout << n;
8    return 0;
9  }
```

引用作为函数的返回值

```
1  int n = 4;
2  int &SetValue() {
3    return n;
4  }
5  int main() {
6    SetValue() = 40;
7    cout << n;
8    return 0;
9  }
```

输出：40

常引用

定义引用时，前面加const关键字，即为“常引用”

```
1  int n;
2  const int &r = n;
```

常引用

定义引用时，前面加const关键字，即为“常引用”

```
1  int n;
2  const int &r = n;
```

r 的类型是 const int &

常引用

定义引用时，前面加const关键字，即为“常引用”

```
1  int n;
2  const int &r = n;
```

r 的类型是 const int &
不能通过常引用去修改其引用的内容:

```
1  int n = 100;
2  const int &r = n;
3  r = 200;   //编译错
4  n = 300;  // 没问题
```

常引用和非常引用的转换

`const T &` 和`T &` 是不同的类型!!!

`T &`类型的引用或T类型的变量可以用来初始化`const T &`类型的引用。

`const T`类型的常变量和`const T &` 类型的引用则不能用来初始化`T &`类型的引用，除非进行强制类型转换。

单选题

下面程序片段哪个没错?

A. `int n = 4; int &r = n * 5;`

B. `int n = 6; const int &r = n; r = 7;`

C. `int n = 8; const int &r1 = n; int &r2 = r1;`

D. `int n = 8; int &r1 = n; const int r2 = r1;`

单选题

下面程序片段哪个没错?

A. `int n = 4; int &r = n * 5;`

B. `int n = 6; const int &r = n; r = 7;`

C. `int n = 8; const int &r1 = n; int &r2 = r1;`

D. `int n = 8; int &r1 = n; const int r2 = r1;`

答案：D

单选题

下面程序片段输出结果是什么?

```
1  int a = 1, b = 2;
2  int &r = a;
3  r = b;
4  r = 7;
5  cout << a << endl;
```

A. 1
B. 2
C. 7
D. 都不是

单选题

下面程序片段输出结果是什么?

```
1  int a = 1, b = 2;
2  int &r = a;
3  r = b;
4  r = 7;
5  cout << a << endl;
```

A 1

B 2

C 7

D 都不是

答案：C

定义常量

```
1 const int MAX_VAL = 23;
2 const string SCHOOL_NAME = "Peking University";
```

定义常量指针

不可通过常量指针修改其指向的内容

```
1  int n, m;
2  const int *p = &n;
3  *p = 5; //编译出错
4  n = 4;  // ok
5  p = &m; // ok, 常量指针的指向可以变化
```

定义常量指针

不能把常量指针赋值给非常量指针，反过来可以

```
1  const int *p1;
2  int *p2;
3  p1 = p2;         // ok
4  p2 = p1;         // error
5  p2 = (int *)p1; // ok, 强制类型转换
```

定义常量指针

函数参数为常量指针时，可避免函数内部不小心改变参数指针所指地方的内容

```
1  void myPrintf(const char *p) {
2    strcpy(p, "this"); //编译出错
3    printf("%s", p);   // ok
4  }
```

定义常引用

不能通过常引用修改其引用的变量

```
1  int n;
2  const int &r = n;
3  r = 5; // error
4  n = 4; // ok
```

单选题

下面说法哪种是对的?

- A 常引用所引用的变量，其值不能被修改
- B 不能通过常量指针，去修改其指向的变量
- C 常量指针一旦指向某个变量，就不能再指向其他变量
- D 以上都不对

单选题

下面说法哪种是对的?

A. 常引用所引用的变量，其值不能被修改
B. 不能通过常量指针，去修改其指向的变量
C. 常量指针一旦指向某个变量，就不能再指向其他变量
D. 以上都不对

答案：B