一、动态内存分配

1.第一种用法，分配一个变量：

P = new T;

T 是任意**类型名**，P 是类型为 **T*的指针。动态分配出一片大小为 **sizeof(T)**字节的内存空间，并且将该内存空间的**起始地址**赋值给 P。比如：

```
int *pn;
pn = new int;
*pn = 5;
```

2.第二种用法，分配一个数组：

P= new T[N];

T：任意**类型名**；

P：类型为 **T*的指针；

N：要分配的数组元素的**个数**，可以是整型**表达式**。

动态分配出一片大小为 N*sizeof(T)字节的内存空间，并且将该内存空间的起始地址赋给 P。

例如：

```
int *pn;
int i = 5;
pn = new int[i*20];
pn[0]=20;
pn[100] =30;//编译没有问题，运行时导致数组越界，最大下标为 99 才对
```

new 运算符返回值的类型都是 **T***。

3.用 delete 运算符释放动态分配的内存

用 new 动态分配的内存空间，一定要用 delete 运算符进行释放。

格式如下：

delete 指针;//该指针必须指向 new 出来的空间

例如：

```
int *p = new int;
*p = 5;
delete p;
delete p;//异常，一片空间不能被 delete 多次
```

用 delete 释放动态分配的数组，要加“**[]**”。，格式如下：

delete[] 指针;//该指针**必须**指向 new 出来的数组。

如果不加[]，那么 delete 的空间就不会被释放完全。

二、内联函数和重载函数，函数参数缺省值

1.内联函数

函数调用是由时间开销的。如果函数本身只有几条语句，之行很快，而且执行次数较多，那么调用函数产生额外开销比较大，。为了减少函数调用的开销，引入了**内联函数**机制。**编译器处理对内联函数的调用语句时，是将整个函数的代码插入到调用语句处，而不会产生调用函数的语句。**

内联函数的书写格式：

```
inline 函数返回值类型 函数名称(参数)
{
```

函数体;
}
2.函数重载

同 java 类似，不再累述，**根据调用语句中的实参的个数和类型判断应该调用哪个函数**。但是若实参的个数和类型一致，只是返回值类型不同，就不是重载，会报错。

3.函数参数缺省值

定义函数的时候可以让**最右边的连续若干个**参数有缺省值。

三、类和对象的基本概念与用法

结构化程序设计中，函数和其操作的数据结构没有直观联系。

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。

面向对象的程序设计具有：**抽象**、**封装**、**继承**和**多态**四个基本特点。

例程：

```
class CRectangle
{
    public:
    int w,h;
    int Area(){
        return w*h;
    }
    int Perimeter(){
        return 2*(w+h);
    }
    void Init(intw_,inth_){
        w = w_;
        h = h_;
    }
};//必须要有分号
```

在主程序中使用：

```
int main()
{
    int w,h;
    CRectangle r;//新建一个类
    cin>>w>>h;
    r.Init(w,j);;//初始化赋值
    cout<<r.Area()<<endl<<r.Perimeter();
    return 0;
}
```

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**（**不包括成员函数**）的大小之和。

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另一个对象的成员变量。

和结构变量一样，对象之间可以用“=”进行赋值，但不可以运用别的运算符号，除非重载。

第二种使用类的成员变量和成员函数的方法：

指针->成员名。

第一种类似于 Java，不说了。

用法 3：引用名.成员名

```
CRectangle r2;
CRectangle & rr =r2;
rr.w=5;
rr.Init(5,4);//rr 的值边了，r2 的值也变
```