# C++第二讲

## 一、C++中的动态内存分配和回收

C语言中的动态内存分配和回收用malloc和free函数，而C++中使用new和delete关键字来实现

### 1.1 分配

1> 单个内存分配和回收

格式：数据类型 *指针名 = new 数据类型；

int *p1 = new int;　　//分配了一个整形单位的空间，将首地址给p1

2> 连续内存分配和回收

格式：数据类型 *指针名 = new 数据类型[个数];

int *p2 = new int[5];　　//连续分配5个整形单位的空间，将首地址给p2

### 1.2 回收

1> 单个内存的回收

格式：delete 指针名；

2> 连续内存分配和回收

格式：delete []指针名；

```
#include <iostream>

using namespace std;

int main()
{
    //在堆区申请单个空间，没有初始化
    int *p1 = new int;
    cout<<"*p1 = "<<*p1<<endl;      //?
    *p1 = 520;
    cout<<"*p1 = "<<*p1<<endl;

    //在堆区申请单个空间，并初始化
    int *p2 = new int(1314);
    cout<<"*p1 = "<<*p2<<endl;

```

```
    //释放空间
    delete p1;
    delete  p2;
    p1 = nullptr;
    p2 = nullptr;

    cout<<"*****************************************\n";
    int *p3 = new int[5];            //在堆区连续分配了5个int单位的空间
    for(int i=0; i<5; i++)
    {
        p3[i] = i+10;
    }
    //输出
    for(int i=0; i<5; i++)
    {
        cout<<p3[i]<<"\t";
    }
    cout<<endl;

    //申请连续空间并且初始化
    int *p4 = new int[5]{99,88,77,66,55};
    for(int i=0; i<5; i++)
    {
        cout<<p4[i]<<"\t";
    }

    //释放空间
    delete [] p3;
    delete [] p4;
    p3 = nullptr;
    p4 = nullptr;


    return 0;
}
```

练习：要求使用new和delete完成，申请空间存储5个学生的成绩，输入这5名学生的成绩，并且按照升序排序后输出结果。

```
#include <iostream>

using namespace std;

int main()
```

```
{
    float *p = new float[5];

    for (int i = 0; i < 5; i++)
    {
        cout << "请输入: ";
        cin >> p[i];
    }

    //排序
    for (int i = 1; i < 5; i++)       //趟数
    {
        for (int j = 0; j < 5-i ; j++)      //控制元素
            if (p[j] > p[j+1])
            {
                float temp = p[j];
                p[j] = p[j+1];
                p[j+1] = temp;
            }
    }

    for (int i = 0; i < 5; i++)
    {
        cout << p[i]<<"\t";
    }
    return 0;
}
```

## 1.3 new\delete与malloc\free的区别(面试题)

1> new\delete关键字而malloc\free是函数

2> new申请出来的空间，需要什么类型的地址就返回什么类型，而malloc申请的空间是void*类型

3> new申请空间时，不需要计算所需空间的大小，直接给定，以类型为单位
malloc申请的空间，需要计算所需空间大小，以字节作为单位

4> new申请空间时，会调用构造函数，而malloc不会

5> delete释放空间时，会调用析构函数，而free不会

# 二、C++中的函数

# 2.1 函数重载（overload）

## 2.1.1 概念

能够实现“一名多用”或者“一物多用”

背景：在我们封装函数时，有时会因为函数参数个数不同或者函数参数类型不同，导致同一功能的函数要定义很多个不同形式的函数名，导致代码的可复用性比较低，比较复杂。

此时，C++提供了一个机制：函数重载，能够一个函数名实现多个用途

## 2.1.2 要求

定义函数时：**函数名**必须相同，**形参列表**必须不同（个数不同、类型不同）

注意：跟返回值类型没有关系

## 2.1.3 调用

函数调用时，会根据实参的个数和类型，自动推导要调用的函数进行使用

```
#include <iostream>

using namespace std;

//定义函数返回两个整数的和
int sum(int a, int b)
{
    return a+b;
}

//定义函数实现三个整数的和
int sum(int a, int b, int c)
{
    return a+b+c;
}

//定义函数包括小数
double sum(int a, double b, int c)
{
    return a+b+c;
}

int main()
{
    cout<<sum(2,3)<<endl;         //5
    cout<<sum(2, 3.5,1)<<endl;         //6
```

```
27
28      return 0;
29  }
30
```

练习：定义函数实现，求两个整数的最大值、两个double数据的最大值、一个整数一个double类型数据的最大值，主调函数中进行调用

## 2.2 函数的默认参数

1>　C语言中函数形参的值必须由实参进行传递，不能设置缺省值

2>　C++中提供了函数默认参数机制，即：定义函数时，给函数形参一个缺省值，函数调用时，如果实参给定了，就使用实参的值，如果实参没给定，就使用默认的缺省值

3>　设置缺省值原则：靠右原则,有多个参数时，只有靠右边的参数都是缺省值时，当前参数才能设置缺省值，原因是，实参传递时是靠左原则

4>　当主调函数写到被调函数前面时，被调函数需要进行函数声明，此时，如果被调函数有默认参数，那么要将默认参数写在声明部分，函数的定义部分就不要加默认参数了。

```
1   #include <iostream>
2
3   using namespace std;
4
5   int sum(int a=100, int b=100, int c=0) ;          //函数声明
6
7
8   int main()
9   {
10
11      cout<<sum()<<endl;          //300
12      cout<<sum(1)<<endl;          //201
13      cout<<sum(1,2)<<endl;        //103
14      cout<<sum(1,2,3)<<endl;      //6
15
16      return 0;
17  }
18
19
20  //定义函数实现三个整数的和
21  int sum(int a, int b, int c)  //sumiii
22  {
23      return a+b+c;
24  }
```

```
25
```

## 2.3 函数重载和默认参数同时出现

```
1  //以下代码时错误的
2  #include <iostream>
3
4  using namespace std;
5
6  int sum(int a=100, int b=100, int c=100) ;           //函数声明
7  int sum(int x, int y)           //此时就不能定义参数类型相同个数小于含
   默认参数的函数了
8  {
9
10 }
11
12 int main()
13 {
14     cout<<sum()<<endl;         //300
15     cout<<sum(1)<<endl;         //201
16     cout<<sum(1,2)<<endl;       //103    //报错，因为不知道去调用哪一
   个
17     cout<<sum(1,2,3)<<endl;     //6
18
19     return 0;
20 }
21
22 //定义函数实现三个整数的和
23 int sum(int a, int b, int c)  //sumiii
24 {
25     return a+b+c;
26 }
27
```

## 2.4 哑元（了解）

所谓哑元，就是在定义函数时，某个或者某几个参数，只有参数名，在函数体内没有使用，仅仅只是起到占位左右，没有实际意义

使用场景一：一般用于代码升级时：原本实现某一功能的函数参数有多个，但是，经过代码升级后，不需要用那么多参数了，但是，此时该函数已经被调用无数多次了，此时更新新函数，不现实，那么我们就可以使用哑元来解决。

```
1  #include <iostream>
2
```

```cpp
using namespace std;

int sum(int a, int , int c)          //第二个参数仅仅为了占位
{
    return a+c;
}

int main()
{
    cout << sum(1,2,3) << endl;
    cout << sum(1,2,3) << endl;
    cout << sum(1,2,3) << endl;
    cout << sum(1,2,3) << endl;
    cout << sum(1,2,3) << endl;
    cout << sum(1,2,3) << endl;
    cout << sum(1,2,3) << endl;
    cout << sum(1,2,3) << endl;
    cout << sum(1,2,3) << endl;
    cout << sum(1,2,3) << endl;
    cout << sum(1,2,3) << endl;
    cout << sum(1,2,3) << endl;

    return 0;
}
```

使用场景二：自增自减运算符重载时使用（后期会讲）

## 2.5 内联函数（inline）

该函数会建议编译器在程序编译的时候将函数进行展开

要求：

1> 函数体比较小

2> 调用比较频繁的函数

优点：使得代码比较高效简洁

缺点：造成代码膨胀

定义格式：在定义函数之前加关键字inline

inline 返回值类型 函数名（参数列表）

```cpp
#include <iostream>

using namespace std;

inline int myMax(int a, int b)          //定义内联函数
{
```

```
    return a>b? a:b;
}


int main()
{
    cout<<myMax(3,5)<<endl;

    return 0;
}
```

内联函数与有参宏的区别

1> 有参宏是宏定义，是指令，而内联函数是函数定义

2> 有参宏是在预处理时展开，而内联函数是在编译时展开

3> 有参宏不占用运行空间，而内联函数占用运行空间

```
#include <iostream>

using namespace std;

#define MAX(x,y) x>y?x:y

inline int myMax(int a, int b)        //定义内联函数
{
    return a>b? a:b;
}

int main()
{
    int a = 1;
    int b = 1;
    int res = MAX(++a,b);  //++a>b ? ++a:b;
    cout << a <<"  "<<b<<"  "<< res <<endl;    //? 3  1  3

//    int a = 1;
//    int b = 1;
//    int res = myMax(++a,b);
//    cout << a <<"  "<<b<<"  "<< res <<endl;    //?  2  1  2

    return 0;
}

```

# 三、C++中的结构体

1> C++中的结构体内是允许封装函数
2> C++中的结构体定义变量时，可以不用加struct

```
#include <iostream>

using namespace std;

struct Stu
{
    int num;         //学号    4
    string name;       //姓名   32
    double score;       //分数   8

    //C++中的结构体可以封装函数
    void show();              //在结构体内声明
};

void Stu::show()          //结构体外定义
{
    cout<<"num = "<<num<<"    name = "<<name<<"    score = "
<<score<<endl;

}


int main()
{
    struct Stu s1 = {1001, "zhangpp", 99.99};
    s1.show();

    cout<<sizeof (string)<<endl;         //32
    cout<<sizeof (s1)<<endl;              //48

    Stu s2 = {1002, "zhangsan", 88.88};
    s2.show();

    return 0;
}
```

# 四、类

面向对象的三大特征：封装、继承、多态，如果说有第四特征，外加一个抽象

## 4.1 c++中的类

C++中的类是由结构体演化而来的。只不过在成员中加了访问权限控制而已。

## 4.2 类的定义格式

```
class 类名
{
    public:
        公用的成员属性或方法;
    private:
        私有的成员属性或方法;
    protected:
        受保护的成员属性和方法;
};
public权限: 类内、子类中、类外都可以访问该权限下的成员
private权限:该权限下的成员只能被类内访问
protected权限:该权限下的成员能被类内、子类中访问，类外不允许访问
```

```
#include <iostream>

using namespace std;

//定义一个学生类
class Stu
{
private:
    int age;
protected:
    int pwd;          //银行卡密码
public:
    string name;
    int score;

    void show()
    {
        cout<<"name = "<<name<<endl;
        cout<<"pwd = "<<pwd<<endl;
        cout<<"age = "<<age<<endl;
        cout<<"score = "<<score<<endl;
    }
};
```

```
24
25  int main()
26  {
27      Stu s1;                 //用类定义变量，我们也称作实例化对象的过程
28      cout<<"name = "<<s1.name<<endl;      //不报错，因为是公用成员，类外可以被访问
29      //cout<<"pwd = "<<s1.pwd<<endl;       //报错，受保护的成员，不能在类外被访问
30      //cout<<"age = "<<s1.age<<endl;        //报错，私有成员，不能在类外被访问
31      cout<<"score = "<<s1.score<<endl; //不报错，因为是公用成员，类外可以被访问
32      s1.show();                          //不报错，因为是公用成员，类外可以被访问
33      return 0;
34  }
```

实例

```
1   #include <iostream>
2
3   using namespace std;
4
5   class Circle
6   {
7   private:
8       double radius;        //半径
9
10  public:
11      void set_r(double r)
12      {
13          if(r<0)
14          {
15              cout<<"半径设置失败\n";
16          }else
17          {
18              radius = r;
19          }
20      }
21      double get_r()
22      {
23          return  radius;
24      }
25
26      //获取周长
```

```
    double get_zhouchang()
    {
        return 2*3.14*radius;
    }

    //获取面积
    double get_s()
    {
        return 3.14*radius*radius;
    }


};



int main()
{
    Circle c1;
   // c1.radius = -1;
    c1.set_r(1);
    cout<<"c1的半径："<<c1.get_r()<<endl;
    cout<<"c1的周长："<<c1.get_zhouchang()<<endl;
    cout<<"c1的面积："<<c1.get_s()<<endl;

    Circle c2;
    c2.set_r(2);
    cout<<"c2的半径："<<c2.get_r()<<endl;
    cout<<"c2的周长："<<c2.get_zhouchang()<<endl;
    cout<<"c2的面积："<<c2.get_s()<<endl;
    return 0;
}


```

练习：定义一个矩形类，私有成员：2条边，公有成员：设置边长、获取每条边，获取周长，获取面积

```
#include <iostream>

using namespace std;

class Rec
{
private:
```

```
    int length;
    int width;

public:
    //定音初始化函数
    void init(int l, int w)
    {
        if(l>=0 && w>=0)
        {
            length = l;
            width = w;
        }else
        {
            cout<<"初始化失败\n";
            return;
        }
    }

    //获取长度
    int get_l()
    {
        return length;
    }
    //获取宽度
    int get_w()
    {
        return width;
    }

    //获取周长
    int get_c()
    {
        return (length+width)*2;
    }

    //获取面积
    int get_s()
    {
        return length*width;
    }

};

int main()
{
```

```
    Rec r1;
    r1.init(2,3);
    cout<<"r1::length = "<<r1.get_l()<<endl;
    cout<<"r1::width = "<<r1.get_w()<<endl;
    cout<<"r1::周长 = "<<r1.get_c()<<endl;
    cout<<"r1::面积 = "<<r1.get_s()<<endl;

    return 0;
}

```

## 4.3 封装

1> 面向对象三大特征：封装、继承、多态

2> 封装：将实现某一事物的所有的行为（成员方法）和属性（成员变量）都封装到一个类中，并且提供公共的接口，用户可以通过接口来操纵该类实例化出来的对象

3> 访问权限

| 关键字 | 类内 | 子类 | 类外 |
| --- | --- | --- | --- |
| public | ! | ! | ! |
| protected | ! | ! | X |
| private | ! | X | X |

4> 类内的成员可以访问类内的所有权限下的属性，包括私有成员

5> 访问权限是针对于整个类的，而不是针对于某个类对象

6> 一个访问权限可以控制其下面的所有成员，直到下一个访问权限出现为止

7> 同一个访问权限在一个类中可以出现多次，可以放在不同位置，但是，为了美观，我们将同一权限下的所有成员都放在一起

8> 如果类内成员没有加访问权限，默认是私有权限

练习：在上一题的基础上，加一个全局函数，实现判断两个矩形是否相等

（长==长&&宽==宽）||（长==宽&&宽==长）

```
#include <iostream>

using namespace std;

class Rec
{
private:
    int length;
```

```
    int width;

public:
    //定音初始化函数
    void init(int l, int w);

    //获取长度
    int get_l();
    //获取宽度
    int get_w();
    //获取周长
    int get_c();

    //获取面积
    int get_s();

    //判断两个类对象是否相等
    bool judge( Rec &a,  Rec &b);

    //判断两个类对象是否相等
    bool judge(Rec &b);

};

void Rec::init(int l, int w)
{
    if(l>=0 && w>=0)
    {
        length = l;
        width = w;
    }else
    {
        cout<<"初始化失败\n";
        return;
    }
}

//获取长度
int Rec::get_l()
{
    return length;
}
//获取宽度
int Rec::get_w()
{
```

```
    return width;
}

//获取周长
int Rec::get_c()
{
    return (length+width)*2;
}

//获取面积
int Rec::get_s()
{
    return length*width;
}

//判断两个类对象是否相等
bool Rec::judge( Rec &a,  Rec &b)
{
    if((a.get_l()==b.get_l()&&a.get_w()==b.get_w()) ||
(a.get_l()==b.get_w()&&a.get_w()==b.get_l()))
    {
        return true;
    }else
    {
        return false;
    }
}

//判断两个类对象是否相等
bool Rec::judge(Rec &b)
{
    if((length==b.get_l()&&width==b.get_w()) ||
(length==b.get_w()&&width==b.get_l()))
    {
        return true;
    }else
    {
        return false;
    }
}

//定义全局函数判断两个矩形是否相等
bool judge( Rec &a,  Rec &b)
{
```

```
    if((a.get_l()==b.get_l()&&a.get_w()==b.get_w()) ||
(a.get_l()==b.get_w()&&a.get_w()==b.get_l()))
    {
        return true;
    }else
    {
        return false;
    }
}

int main()
{
    Rec r1;
    r1.init(2,3);
    cout<<"r1::length = "<<r1.get_l()<<endl;
    cout<<"r1::width = "<<r1.get_w()<<endl;
    cout<<"r1::周长 = "<<r1.get_c()<<endl;
    cout<<"r1::面积 = "<<r1.get_s()<<endl;

    Rec r2;
    r2.init(2,5);

    if(r1.judge(r2))
    {
        cout<<"yes"<<endl;
    }else
    {
        cout<<"no"<<endl;
    }

    return 0;
}

```

## 4.4 C++的分文件编译

```
头文件
#ifndef REC_H
#define REC_H
#include <iostream>

using namespace std;

class Rec
```

```
{
private:
    int length;
    int width;

public:
    //定音初始化函数
    void init(int l, int w);

    //获取长度
    int get_l();
    //获取宽度
    int get_w();
    //获取周长
    int get_c();

    //获取面积
    int get_s();

    //判断两个类对象是否相等
    bool judge( Rec &a,  Rec &b);

    //判断两个类对象是否相等
    bool judge(Rec &b);

};

bool judge( Rec &a,  Rec &b);        //声明全局函数


#endif // REC_H
```

---

源文件：

```
#include"rec.h"

void Rec::init(int l, int w)
{
    if(l>=0 && w>=0)
    {
        length = l;
        width = w;
    }else
    {
```

```
53          cout<<"初始化失败\n";
54          return;
55      }
56  }
57
58  //获取长度
59  int Rec::get_l()
60  {
61      return length;
62  }
63  //获取宽度
64  int Rec::get_w()
65  {
66      return width;
67  }
68
69  //获取周长
70  int Rec::get_c()
71  {
72      return (length+width)*2;
73  }
74
75  //获取面积
76  int Rec::get_s()
77  {
78      return length*width;
79  }
80
81  //判断两个类对象是否相等
82  bool Rec::judge( Rec &a,  Rec &b)
83  {
84      if((a.get_l()==b.get_l()&&a.get_w()==b.get_w()) ||
    (a.get_l()==b.get_w()&&a.get_w()==b.get_l()))
85      {
86          return true;
87      }else
88      {
89          return false;
90      }
91  }
92
93  //判断两个类对象是否相等
94  bool Rec::judge(Rec &b)
95  {
```

```
    if((length==b.get_l()&&width==b.get_w()) ||
(length==b.get_w()&&width==b.get_l()))
    {
        return true;
    }else
    {
        return false;
    }
}

//定义全局函数判断两个矩形是否相等
bool judge( Rec &a,  Rec &b)
{
    if((a.get_l()==b.get_l()&&a.get_w()==b.get_w()) ||
(a.get_l()==b.get_w()&&a.get_w()==b.get_l()))
    {
        return true;
    }else
    {
        return false;
    }
}
```

---

测试文件

```
#include"rec.h"


int main()
{
    Rec r1;
    r1.init(2,3);
    cout<<"r1::length = "<<r1.get_l()<<endl;
    cout<<"r1::width = "<<r1.get_w()<<endl;
    cout<<"r1::周长 = "<<r1.get_c()<<endl;
    cout<<"r1::面积 = "<<r1.get_s()<<endl;

    Rec r2;
    r2.init(2,5);

    if(r1.judge(r2))
    {
        cout<<"yes"<<endl;
    }else
```

```
    {
        cout<<"no"<<endl;
    }

    return 0;
}

```

# 作业：

作业一：封装一个循环顺序队列

```
class Queue
{
private:
    datatype data[MAX];
    int front;      //对头
    int tail;     //队尾
public:
    void init(int front, int tail);
    bool empty();
    bool full();
    int size();
    void push(datatype e);
    void pop();
    void show();
};
```

作业二：封装一个班级

成员属性：存放学生成绩的数组

成员方法：输入学生成绩

输出学生成绩

排序

求最值

求总分

作业三：整理思维导图