

# **C++程序设计**

## **实习指导书（2017）**

夏军宝

中国地质大学（北京）信息工程学院

2017 年 9 月

# 目 录

<b>1 C++过程化编程.....</b>	<b>1</b>
1.1 实习目的.....	1
1.2 实习指导.....	1
1.3 实习任务.....	4
1.4 课后练习.....	11
<b>2 类与对象基础.....</b>	<b>17</b>
2.1 实习目的.....	17
2.2 实习任务.....	17
<b>3 类与对象应用.....</b>	<b>27</b>
3.1 实习目的.....	27
3.2 实习任务.....	27
3.3 课后练习.....	41
<b>4 运算符重载.....</b>	<b>48</b>
4.1 实习目的.....	48
4.2 实习指导.....	48
4.3 实习任务.....	49
4.4 课后练习.....	58
<b>5 继承与多态.....</b>	<b>63</b>
5.1 实习目的.....	63
5.2 实习任务.....	63
5.3 课后练习.....	71
<b>6 模板.....</b>	<b>77</b>
6.1 实习目的.....	77
6.2 实习任务.....	77
6.3 课后练习* .....	85

<b>7 STL 及应用 .....</b>	<b>90</b>
7.1 实习目的.....	90
7.2 实习任务.....	90
7.3 课后练习.....	100
<b>8 异常处理与输入输出流.....</b>	<b>106</b>
8.1 实习目的.....	106
8.2 实习指导.....	106
8.3 实习任务.....	107
8.4 课后练习.....	114

# 1 C++过程化编程

## 1.1 实习目的

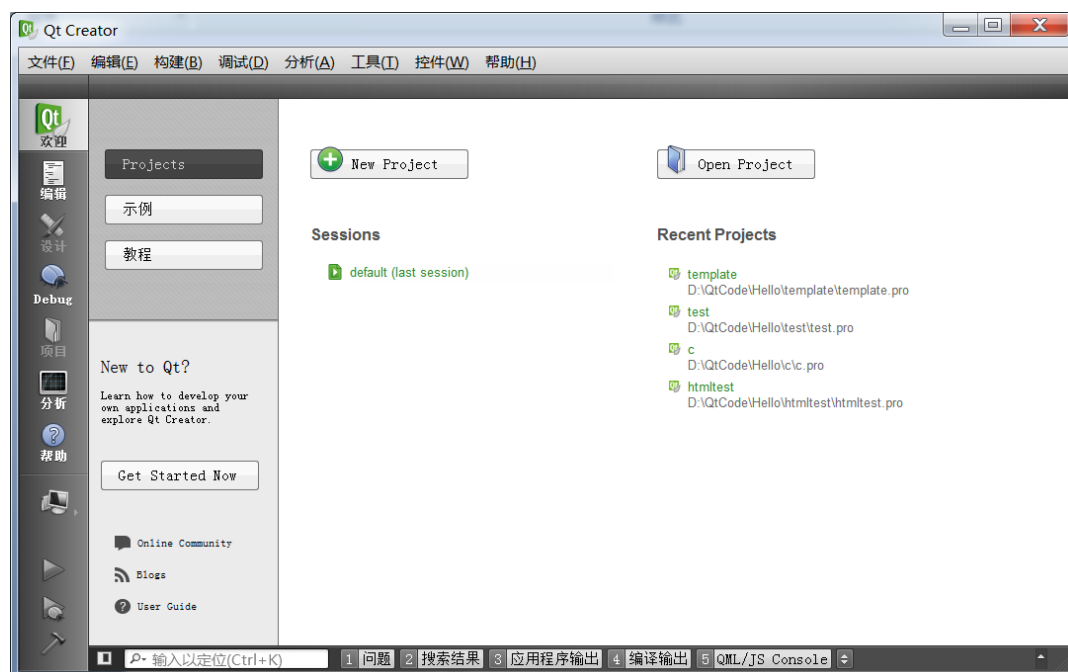
- 1) 掌握 C++ 语言基础（常量、变量、输入输出、命名空间）；
- 2) 掌握指针与动态内存分配的使用；
- 3) 掌握函数传参的概念及扩展应用（引用与指针传参、返回指针、递归、函数默认值、重载的使用）；
- 4) 掌握 C++11 的基础语言扩展机制（auto 类型推导、统一初始化、范围循环等）
- 5) 了解 string 和 vector 的基础应用。

## 1.2 实习指导

### 1) QT5 安装

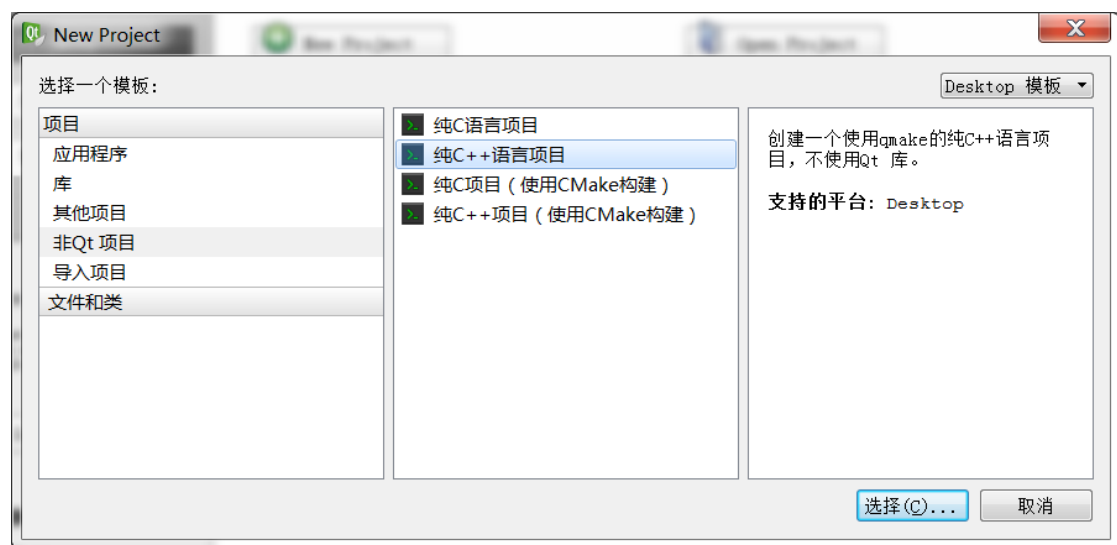
安装 qt-opensource-windows-x86-mingw48\_opengl-5.2.1.exe，勾选全部选项，否则无法编译。

### 2) 新建项目

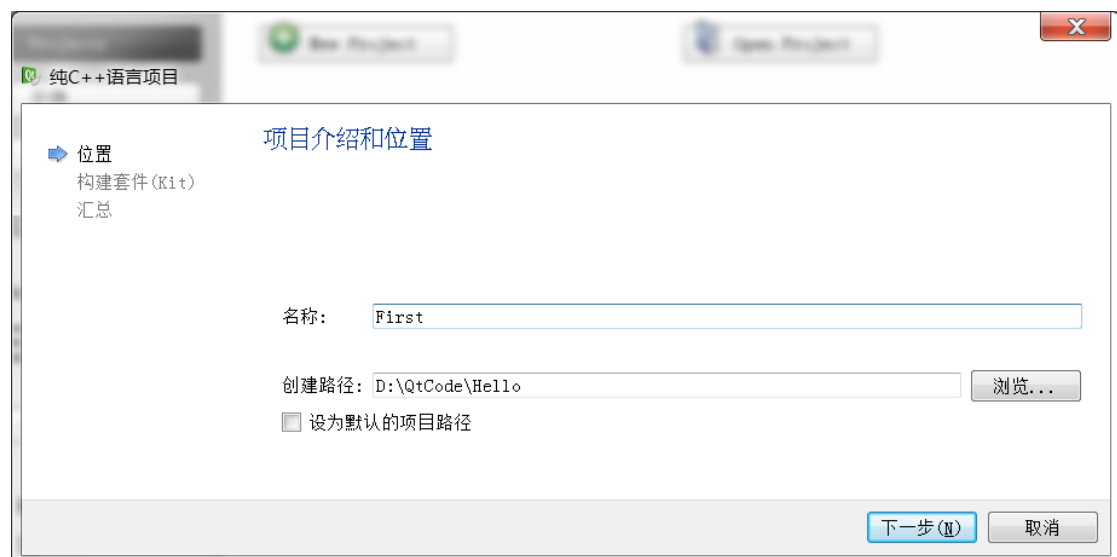


首页中点击“New Project”按钮，弹出向导中，对于控制台 C++ 程序，左侧

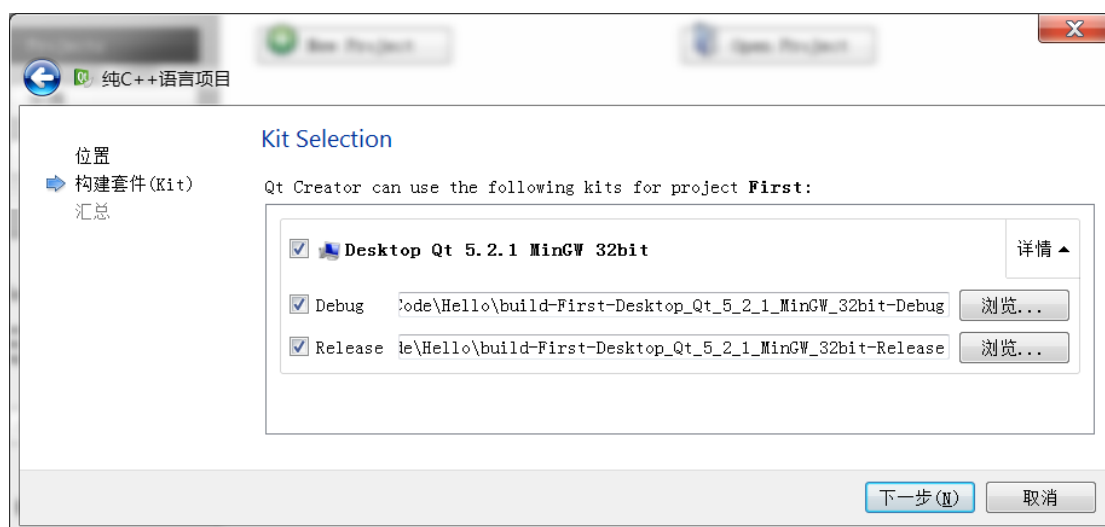
栏中选择“非 Qt 项目”，中间栏选择“纯 C++语言项目”，点击“选择”按钮。



在弹出的对话框中，输入项目名称，指定项目保存路径。设置好之后点击“下一步”按钮。进入构建套件选择页面。



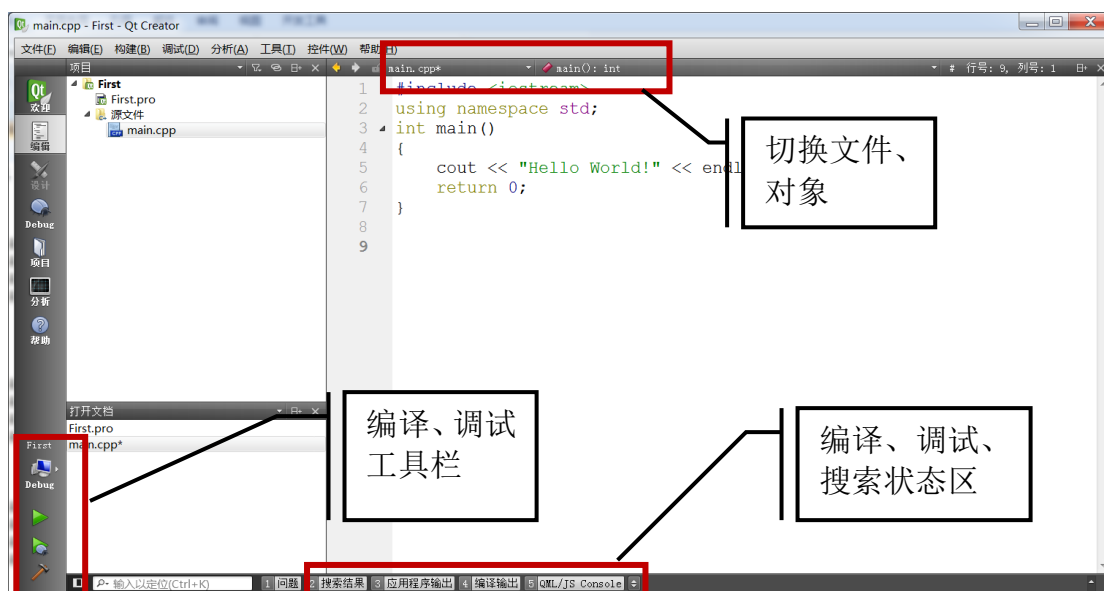
在构建套件选择页面，保持默认选项即可。点击“下一步”进入汇总页面，在汇总页面，点击“完成”按钮完成项目创建。



### 3) 代码编辑和编译

项目中通常会包含 pro 文件、源代码文件及相关的编译过程与结果文件。pro 文件中会保存一些编译选项，较早的版本需要在其中手工添加下面的配置，以支持 C++11 特性，Qt5.7 以后的版本自动支持 C++11。

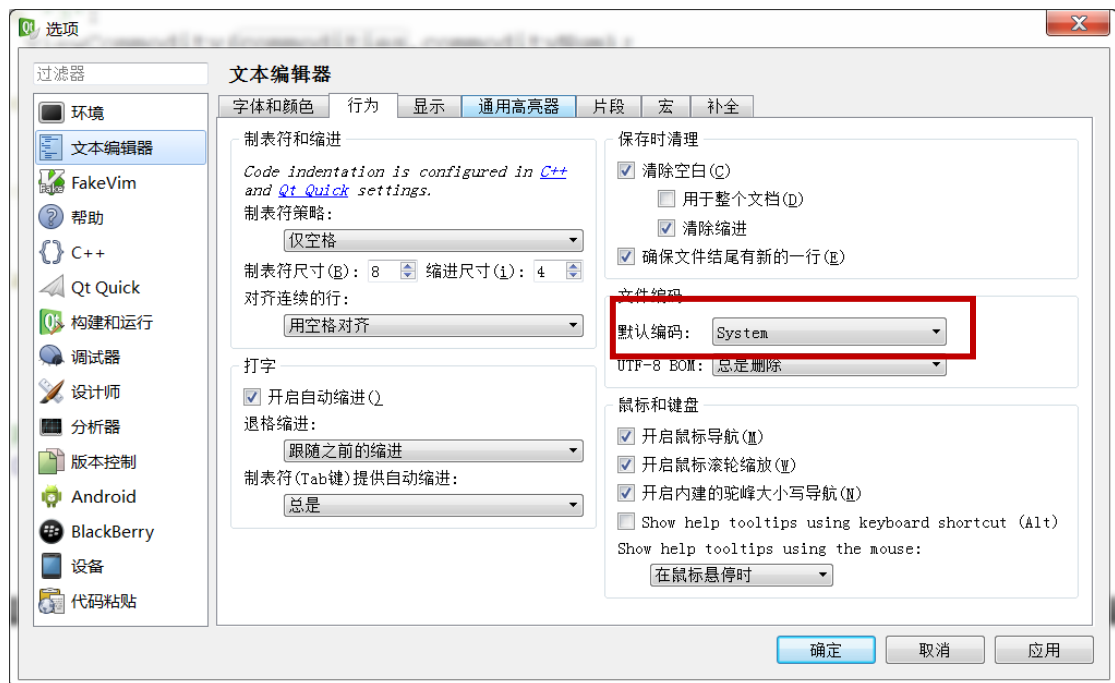
CONFIG +=C++11



### 4) 控制台中文支持

如果希望在控制台程序中输出中文，需要在系统中进行配置，将编码设置为 system 编码，否则会出现乱码。

执行“工具”→“选项”菜单项，在弹出的对话框中，选择“文本编辑器”选项，在“行为”tab 页，设置默认编码为“system”。



## 1.3 实习任务

### 1.3.1 实习任务一

- 1) 调试运行下面的程序，记录程序的运行结果。

```
namespace xxx{
    double fun(double a,double b){
        return a+b;
    }
}

namespace yyy{
    double fun(double a,double b){
        return (a+b)/2;
    }
}

#include <iostream>
using namespace std;
using namespace yyy;
int main(){
    cout<<fun(1,4)<<endl;
    using xxx::fun;
```

```

    cout<<fun(1,4)<<endl;
    cout<<yyy::fun(1,4)<<endl;
    return 0;
}

```

运行结果: \_\_\_\_\_

---

- 2) 调试运行下面的程序，记录程序运行结果。

```

#include <iostream>
#include <string>
using namespace std;
string add(const string& s1,const string s2){
    return s1+s2;
}
double add(double a,double b){
    return a+b;
}
int main(){
    cout<<add("hello","world")<<endl;
    cout<<add(3,5)<<endl;
    cout<<add('3','5')<<endl;
    return 0;
}

```

运行结果: \_\_\_\_\_

---

- 3) 调试运行下面的程序，记录程序运行结果。

```

#include <iostream>
using namespace std;
int gcd(int m, int n){
    if(n==0)
        return m;
    return gcd(n, m% n);
}
int main(){
    cout<<"1:"<<gcd(20,8)<<endl;
    cout<<"2:"<<gcd(36,64)<<endl;
    return 0;
}

```



}

运行结果: \_\_\_\_\_  
\_\_\_\_\_

### 1.3.2 实习任务二

- 1) 编写函数 reverse，将数组中的元素逆序，函数原型如下。

```
void reverse( int array[], int size );
```

其中 array 为指向数组的指针，size 表示数组的大小。

---

---

---

---

---

---

---

---

---

---

- 2) 编写函数 getMax，计算数组的最大值，函数原型如下。

```
int getMax(int *array, int size);
```

其中 array 为指向数组的指针，size 表示数组的大小。

---

---

---

---

---

---

---

---

---

---

---

---

---

### 1.3.3 实习任务三\*

- 1) 编写函数 `getDiagonal`，获得  $n \times n$  方阵的对角线元素，通过动态内存分配技术，返回指针指向动态创建的对角线元素数组。函数原型如下：

```
int * getDiagonal(int ** matrix, int n);
```

其中 `matrix` 是二级指针，指向动态创建的方阵，`n` 为方阵的大小；返回类型为 `int *`，指向创建的的对角线元素。

```
#include <iostream>
```

```
using namespace std;
```

```
int * getDiagonal(int ** matrix, int n){
```

---

---

---

---

---

---

---

---

---

---

```
}
```

```
int main(){
```

```
    int **data, n=5;
```

```
    data =new int*[n]; //动态创建指针数组
```

```
    for(int i=0;i<n;++i)
```

```
        data[i]=new int[n]; //每个指针数组指向动态创建的 1 维数组
```

```
    for(int i=0;i<n;++i){ //初始化数组元素
```

```
        for(int j=0;j<n;++j)
```

2) 重新实现 `getDiagonal`，使用 `vector` 存储  $n \times n$  方阵数据，使用 `vector` 保存获得的对角线元素，通过返回 `vector` 返回计算结果。函数原型如下：

其中 `matrix` 是 `vector` 数组，其中的元素是嵌套的 `vector` 数组，嵌套的 `vector` 包含 `int` 数据，从而构成二维方阵；返回类型为 `vector`，保存方阵的对角线元素。

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

```

}
int main(){
    vector<vector<int>> matrix{
        {0,1,2,3,4},
        {1,2,3,4,5},
        {2,3,4,5,6},
        {3,4,5,6,7},
        {4,5,6,7,8}
    };
    vector<int> diagonal=getDiagonal(matrix);
    for(const auto e: diagonal){
        cout<<e<<endl;
    }
    return 0;
}

```

#### 1.3.4 实习任务四\*

下面的程序实现小学生 4 则运算（加、减、乘）的模拟测试，每次测试时，随机生成 2 位整数进行运算，根据用户的输入判断计算结果是否正确。

以下是实现题目要求的参考程序代码，请认真阅读并上机调试程序，验证程序功能是否正确。程序中通过函数指针，传递指定的运算函数作为参数，在 answerQuestion 函数中，通过比较调用函数指针的运算结果和传入的答案判断结果是否正确。用户输入不同选项时，将函数指针指向相应的计算函数（add、sub、multiply），从而实现灵活性。

```

#include <iostream>
#include <ctime>
using namespace std;
int add(int a,int b){
    return a+b;
}
int sub(int a,int b){
    return a-b;
}
int multiply(int a,int b){

```

```

        return a*b;
    }
    char menu(){
        char choice;
        cout<<"1) add two number\n";
        cout<<"2) sub two number\n";
        cout<<"3) multiply two number\n";
        cout<<"0) quit\n";
        cout<<"Enter your choice:\n";
        cin>>choice;
        return choice;
    }
    bool answerQuestion(int num1,int num2,int (*f)(int ,int ), int answer ){
        return f(num1,num2)==answer;
    }
    int main(){
        while(true) {
            char choice=menu();
            if(choice=='0')
                break;
            int num1, num2;
            num1=rand()%90+10;    //得到 10~99 的整数
            num2=rand()%90+10;
            decltype(add) *pf;
            char op;
            switch(choice){
                case '1':pf=add; op='+'; break;
                case '2':pf=sub; op='-'; break;
                case '3':pf=multiply; op='*'; break;
                default:continue;
            }
            int answer;
            cout<<num1<<op<<num2<<"=?";
            cin>>answer;
            if(answerQuestion(num1,num2,pf,answer)==true)
                cout<<"Correct!"<<endl;
            else

```

```

        cout<<"Wrong!"<<endl;
    }
    return 0;
}

```

试着按下面的需求修改程序，上机调试并验证程序是否正确：①用户输入答案错误后，可以再次输入答案，但最多只能输入 3 次；②最终退出 `while` 循环时，报告答题数量和正确率；③添加除法功能，生成随机数时，需要确保 2 个数能够整除。

## 1.4 课后练习

1. 写出下面程序的运行结果，假定输入"Hello\_123"。

```

#include <iostream>
using namespace std;
int main(){
    char word[50];
    cout<<"Enter a word:";
    cin>>word;
    for(int i=0; word[i]!='\0'; ++i) {
        if(word[i]>='a' && word[i]<='z' )
            word[i]-= 32;
    }
    cout<<"Upper case: "<<word<<endl;
    return 0;
}

```

2. 写出下面程序的运行结果，假定输入"Hello123\_World"。

```

#include <iostream>
using namespace std;
int main(){
    char word[50];
    cout<<"Enter a string:";
    cin>>word;
    int pos=0;
    for(int i=0; word[i]!='\0'; ++i) {
        if(word[i]<'0' || word[i]>'9'){

```

```

        word[pos]=word[i];
        ++pos;
    }
}
word[pos]='\0';
cout<<"result: "<<word<<endl;
return 0;
}

```

3. 写出下面程序的运行结果。

```

#include <iostream>
using namespace std;
int sum( int a, int b=1, int c=3 ){
    return a+b+c;
}
int main(){
    int sum(int a, int b=3, int c=4);
    cout<<sum (2)<<endl;
    cout<<sum (2,5)<<endl;
    cout<<sum (2,3,6)<<endl;
    return 0;
}

```

4. 写出下面程序的运行结果。

```

#include <iostream>
using namespace std;
char & elem(char *s, int n){
    return s[n];
}
int main(){
    char str[]="HelloWorld";
    elem(str,1)= 'A';
    cout<<str<<endl;
    return 0;
}

```

5. 写出下面程序的运行结果。

```

#include <iostream>
using namespace std;

```

```

int x=10;
int main(){
    int x=15;
    cout<<x<<endl;
    cout<<::x<<endl;
    return 0;
}

```

6. 写出下面程序的运行结果。

```

#include <iostream>
using namespace std;
void xhg(int *a,int *b){
    int *tmp;
    tmp=b; b=a; a=tmp;
    cout<<*a<<' '<<*b<<endl;
}
int main(){
    int x(5),y(4);
    xhg(&x,&y);
    cout<<x<<' '<<y<<endl;
    return 0;
}

```

7. 写出下面程序的运行结果。

```

#include <iostream>
using namespace std;
void xhg(int &a,int &b){
    int tmp;
    tmp=b; b=a; a=tmp;
    cout<<a<<' '<<b<<endl;
}
int main(){
    int x(5),y(4);
    xhg(x,y);
    cout<<x<<' '<<y<<endl;
    return 0;
}

```

8. 写出下面程序的运行结果。



```

#include <iostream>
using namespace std;
int ff(int *a,int size){
    if(size==1)
        return a[0];
    return a[size-1]+ff(a,size-1);
}
int main(){
    int a[5]={ 1,2,3,4,5};
    cout<<"result: "<<ff(a,5)<<endl;
    return 0;
}

```

9. 写出下面程序的运行结果。

```

#include <iostream>
using namespace std;
void f(const string& s,int n){
    cout<<s[n-1];
    if(n>1)
        f(s,n-1);
}
int main(){
    f("animal",6);
    cout<<endl;
    f("hello",3);
    return 0;
}

```

10. 写出下面程序的运行结果。

```

#include <iostream>
using namespace std;
int func(int data[],int size){
    int a=data[0];
    int b=data[0];
    for(int i=1;i<size;++i)    {
        if(data[i]>a) a=data[i];
        if(data[i]<b) b=data[i];
    }
    return a-b;
}

```

```

}
int main(){
    int a[]{9,3,2,-1,8,0,4};
    cout<<func(a,7)<<endl;
    cout<<func(a+2,4)<<endl;
    return 0;
}

```

11. 写出下面程序的执行结果。

```

#include <iostream>
using namespace std;
int fun(int interval=1){
    int sum=0, i=0;
    for(i=0; i<100; i+=interval)
        sum+=i;
    return sum;
}
int main(){
    cout<<"Result1: "<<fun(2)<<endl;
    cout<<"Result2: "<<fun()<<endl;
    return 0;
}

```

12. 写出下面程序的执行结果。

```

#include <iostream>
using namespace std;
double func( double pData[ ], int size);
int main(){
    double array[]{2.2, 3.8, 6, 5.4};
    cout<<"Result: "<<func(array, 4)<<endl;
    cout<<"Result: "<<func(array, 3)<<endl;
    return 0;
}
double func( double pData[ ], int size){
    double result=0;
    int i;
    for(i=0; i<size; ++i) {
        result+=pData[i];
    }
}

```

```

        result /= size;
        return result;
    }

```

13. 写出下面程序的执行结果。

```

#include <iostream>
#include <vector>
using namespace std;
int main(){
    vector<int> vec{ 2,4,5,6,10,15,3,21,36,72,9,13};
    for(int i=0;i<vec.size ();++i)
        cout<<vec[i]<<" ";
    cout<<endl;
    for(auto it=vec.begin ();it!=vec.end ();++it)
        cout<<*it<<" ";
    cout<<endl;
    for(auto e : vec)
        cout<<e<<" ";
    cout<<endl;
    return 0;
}

```

## 2 类与对象基础

### 2.1 实习目的

- 1) 掌握类的定义；
- 2) 掌握对象的定义和方法调用；
- 3) 熟悉构造函数和析构函数的定义和执行过程；
- 4) 掌握复制构造函数和初始化列表的使用。

### 2.2 实习任务

#### 2.2.1 实习任务一

- 1) 运行调试下面的程序，记录程序运行的结果。

```
#include <iostream>
using namespace std;
class TestClass{
public:
    TestClass(int a){
        aa=a;
        cout<<aa<<" Constructed!\n";
    }
    ~TestClass(){
        cout<<aa<<" Destructed!\n";
    }
private:
    int aa;
};
TestClass AA(3);    //全局对象
int main(){
    cout<<"In MainFuction."<<endl;
    TestClass BB(5);
    return 0;
}
```

运行结果\_\_\_\_\_

---

---

---

【思考】在 main 函数执行之前，系统会创建所有的全局对象和数据，为这些数据执行初始化操作。

2) 调试运行下面的程序，写出程序的输出结果。

```
#include <iostream>
using namespace std;
class TestClass{
public:
    TestClass() { cout<<"Constructed!\n"; }
    ~TestClass() { cout<<"Destructed!\n"; }
};
int main(){
    TestClass t1;
    TestClass *p;
    p=new TestClass;
    delete p;
    return 0;
}
```

运行结果: \_\_\_\_\_

---

---

---

3) 调试运行下面的程序，写出程序的输出结果。

```
#include <iostream>
using namespace std;
class TestClass{
public:
    TestClass() {
        cout<<"Constructed!\n";
        value = 10;
    }
}
```

```

~TestClass() { cout<<"Destructed!\n"; }
void setValue( int new Value) { value = new Value; }
int getValue()const { return value; }
private:
    int value;
};
int main(){
    TestClass t1;
    cout<<t1.getValue()<<endl;
    TestClass &rt1 = t1;
    rt1.setValue(20);
    cout<<t1.getValue()<<endl;
    TestClass *pt=&t1;
    pt->setValue(30);
    cout<<t1.getValue()<<endl;
    return 0;
}

```

运行结果: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

- 4) 调试运行下面的程序，写出程序的输出结果。

```

#include <iostream>
#include <vector>
using namespace std;
class Test{
public:
    Test():a(1){cout<<a<<endl;}
    Test(int a){
        cout<<this->a<<endl;
        this->a=a;
        cout<<this->a<<endl;
    }
private:
    int a=3;

```

```
};
int main(){
    Test t1;
    Test t2(10);
    return 0;
}
```

运行结果: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

### 2.2.2 实习任务二

编写 Complex 类，封装分数的基本功能，Complex 的如下：

```
#include <iostream>
using namespace std;
class Complex{
public:
    Complex(); //缺省构造函数，实部和虚数部分为 0
    Complex(double n); //只有 1 个参数，虚数为 0
    Complex(double n, double d); //设置实部和虚部
    void setValue(double n, double d); //设置实部和虚部
    double getReal(); //获取实部值
    double getImage(); //获取虚部值
    double getDistance(); //获取实数和原点距离
    void output(); //按实数形式输出分数
private:
    double real; //实部
    double image; //虚部
};
```

给出所有成员函数的实现：

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

[illegible]



---

---

---

---

---

---

---

按照下面的主程序测试所写的复数类，检查结果是否正确。

```
int main(){
    Complex c1, c2(2), c3(3,4);
    c1.output();
    c2.output();
    c3.output();
    c1.setValue(6, 4);
    c1.output();
    cout<<c1.getDistance( )<<endl;
    return 0;
}
```

### 2.2.3 实习任务三

编写 `Time` 类，封装对时间的操作，`Time` 类处理 1 天内的时间，精确到分钟。

`Time` 类的定义如下，请实现 `Time` 类的成员函数定义，并编写主程序进行测试。

```
class Time{
public:
    Time(); //指定小时和分钟数为 0 构造 Time 对象
    Time(int h,int m); //通过指定 h 和 m 构造 Time 对象
    void setTime(int h, int m); //设置新的时间
    void output(); //以 hh:mm 规范格式输出时间
    int getHour(); //获得小时
    int getMinute(); //获得分钟
    int getTotalMinutes(); //获得从 0 点 0 分起的总分钟数
private:
    int hour;
```

给出所有成员函数的实现:

[illegible]

---

---

---

---

---

---

---

测试主程序如下。

```
int main(){
    Time t1(12, 75);
    t1.output();
    t1.setTime(8, 65);
    t1.output();
    cout<<"t1 Hour: "<<t1.getHour()<<endl;
    cout<<"t1 Minute: "<<t1.getMinute()<<endl;
    cout<<"t1 TotalMinutes: "<<t1.getTotalMinutes()<<endl;
    return 0;
}
```

【试一试】Time 类的设计方案并不是唯一的，请尝试使用“一天内的总分钟数”作为数据成员重新实现 Time 类，保持 Time 类的构造函数和访问接口不变，测试的主程序也无需修改。

## 2.2.4 实习任务四\*

编写随机数类。随机数类的定义如下，请实现所要求的成员函数。

```
#include <iostream>
using namespace std;
#include <ctime>
class RandomNum{
public:
    RandomNum(unsigned int seed); //用传入的 seed 设置随机数种子
                                //如果 seed 为 0 设置为当前时间
    void setSeed(unsigned int seed); //重置不同的随机数种子
    int random(); //产生 0~RAND_MAX 的整数
```

给出所有成员函数的实现:

[illegible]

---

---

---

---

---

测试 RandomNum 类的主程序如下。

```
int main(){
    RandomNum randomGenerator(100);
    int i;
    for(i=0;i<100;++i)
        cout<<randomGenerator.random(100)<<' ';
    cout<<"\n";
    for(i=0;i<100;++i)
        cout<<randomGenerator.frandom()<<' ';
    cout<<"\n";
    return 0;
}
```

## 3 类与对象应用

### 3.1 实习目的

- 1) 掌握静态数据成员及静态成员函数的使用;
- 2) 掌握友元函数的使用;
- 3) 掌握常成员函数的应用;
- 4) 掌握对象成员的使用;
- 5) 掌握 C++11 中移动复制构造函数、委托构造函数等语言扩展机制。

### 3.2 实习任务

#### 3.2.1 实习任务一

- 1) 调试运行下面的程序，记录程序运行结果

```
#include <iostream>
#include <string>
using namespace std;
class Mouse{
public:
    Mouse( string newName );
    ~Mouse();
    string getName() { return name; }
    static int mouseNum;
private:
    string name;
};
int Mouse::mouseNum = 0;
Mouse::Mouse( string newName ) : name(newName){
    cout<<name<<" is born!\n";
    mouseNum++;
}
Mouse::~~Mouse(){
    cout<<name<<" is gone...\n";
```

```

        mouseNum--;
    }
class Cat{
public:
    Cat( const string& newName): name(newName){
        cout<<name<<" is coming!\n";
    }
    void catchMouse( Mouse *pMouse);
private:
    string name;
};
void Cat::catchMouse( Mouse *pMouse){
    cout<<"I catch you! I never want to see you again. "
        <<pMouse->getName()<<"!"<<endl;
    delete pMouse;
}
int main(){
    Cat cat("Black Cat Detective");
    Mouse *pMouse1 = new Mouse("Micky");
    cout<<Mouse::mouseNum<<" mouse left.\n";
    Mouse *pMouse2 = new Mouse("Xiaohua");
    cout<<Mouse::mouseNum<<" mouse left.\n";
    cat.catchMouse(pMouse2);
    cout<<Mouse::mouseNum<<" mouse left.\n";
    cat.catchMouse(pMouse1);
    cout<<Mouse::mouseNum<<" mouse left.\n";
    return 0;
}

```

运行结果: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

---

---

2) 运行调试下面的程序，写出程序运行结果

```
#include <iostream>
using namespace std;
class TestClass{
public:
    TestClass( int new Value=0) {
        value = new Value;
        cout<<"Value: "<<value<<" , Constructed!\n";
    }
    TestClass( const TestClass & rhs){
        value = rhs.value;
        cout<<"Value: "<<value<<" , Copy Constructed!\n";
    }
    ~TestClass() { cout<<"Value: "<<value<<" , Destructed!\n"; }
    void setValue( int new Value) { value = new Value; }
    int getValue()const { return value; }
private:
    int value;
};

TestClass fooFun( TestClass t){
    t.setValue(20);
    return t;
}

int main(){
    TestClass t1(10),t2(t1),t3;
    t3=fooFun(t1);
    return 0;
}
```

运行结果: \_\_\_\_\_

---

---

---

---

---



---

---

---

---

- 3) 将程序 1) 中的 fooFun 传递对象作为参数和返回对象修改为传递引用以及返回引用，重新运行程序，记录运行结果。修改后的 fooFun 如下：

```
TestClass & fooFun(TestClass &t){  
    t.setValue(20);  
    return t;  
}
```

运行结果： \_\_\_\_\_

---

---

### 3.2.2 实习任务二

实习 2 中的分数类还很不完善，请按照下面定义和要求完善分数类。

- 将 Complex 类中的 getDoubleValue、getNum、getDen、output 定义为常成员函数。将构造函数的实现写成初始化列表的形式。
- 添加常成员函数 add，实现复数相加计算，计算结果返回一个新的复数对象，为 Complex 类型。
- 添加成员函数 multiply，实现当前复数的向量放大计算，实部和虚部同时放大，返回类型为 void。

```
#include <iostream>  
using namespace std;  
class Complex{  
public:  
    Complex();  
    Complex(int n);  
    Complex(int n,int d);  
    void setValue(int n, int d);  
    double getReal() const ;  
    double getImage() const;
```

给出新增或修改成员函数的定义:

[illegible]

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

### 3.2.3 实习任务三

### 1) 完善实习 2 中的 Time 类

在 `Time` 类中增加 1 个公有成员函数 `getTimeSpan`, 计算 2 个时间之间的时间差, 返回结果也是一个 `Time` 对象。

此外为类增加 1 个构造函数，通过传递 1 个参数（时间的总分钟数）来构造时间对象。将 `output`、`getHour`、`getMinute`、`getTotalMinutes` 定义为常成员函数。

```
class Time{
public:
    Time();
    Time(int h,int m); //通过指定 h 和 m 构造 Time 对象
    Time( int minutes); //通过指定分钟数构造 Time 对象
    void setTime(int h, int m); //设置新的时间
    void output() const; //以 hh:mm 格式输出时间
    int getHour() const; //获得小时
    int getMinute() const; //获得分钟
};
```

```

    int getTotalMinutes() const; //获得从 0 点 0 分起的总分钟数
    Time getTimeSpan( const Time &t )const; //计算时间差
private:
    int hour;
    int minute;
    void normalizeTime(); //规范化小时为 0~23, 分钟为 0~59
};

Time::Time(int minutes){
    _____
    _____
}

Time Time::getTimeSpan( const Time & t)const{
    _____
    _____
    _____
}

int main(){
    Time t1(9,20),t2(11,35),t3;
    t3=t1.getTimeSpan(t2);
    t3.output();
    return 0;
}

```

【提示】计算时间差，可先计算出 2 个时间的总分钟数，得到分钟数差值，再根据分钟数构造 1 个 **Time** 对象并返回。

## 2) 停车收费问题

编写 **ParkingCard** 类，解决停车计费问题，类的定义及部分成员函数实现如下所示，请补充未实现的成员函数，编写主程序进行测试。

**ParkingCard** 类包含 2 个 **Time** 类对象作为数据成员，用来保存开始停车和离开时间。

```

#include "time.h"
class ParkingCard{
public:
    ParkingCard(double newRate) { rate = newRate; }

```

```

void setRate(double newRate) { rate = newRate; } //设置每小时费率
double getRate() const{ return rate; }
void setParkingTime( const Time &time); //设置开始停车时间
void setLeavingTime( const Time &time ); //设置离开时间
double getTotalExpenses() const; //计算停车费用
//停车时间分钟数不足半小时按半小时计算，不足 1 小时按 1 小时计算
void output() const; //输出停车起始时间、费率及总费用
private:
    double rate; //停车费率，按元/每小时计算
    Time parkingTime; //开始停车时间
    Time leavingTime; //离开时间
};
void ParkingCard::setParkingTime(const Time &time){
    parkingTime = time;
}

```

【提示】计算停车费用时，调用 `parkingTime.getTimeSpan(leavingTime)`; 可以计算 2 个时间对象的时间差，返回 1 个时间对象；通过调用时间对象的 `getHour` 和 `getMinute` 可以获得对应的小时数和分钟数。在显示停车详细信息时，可以调用 `parkingTime` 和 `leavingTime` 的 `output` 方法，而不用重复编写输出时间的处理代码。

按下面的主程序测试 `ParkingCard` 类，检查结果是否正确。

```

int main(){
    ParkingCard card(5);
    card.setParkingTime(Time(9, 20)); //构造临时时间对象并作为参数
    card.setLeavingTime(Time(11, 35));
    cout<<"Expenses: "<<card.getTotalExpenses()<<endl;
    cout<<"Detailed info:\n";
    card.output();
    return 0;
}

```

给出所有成员函数的实现：

---



---



---

[illegible]

---

---

---

---

---

---

---

---

### 3.2.4 实习任务四

#### 1) Point 类

实现 1 个简单的二维点类，点类的定义如下，实现该类的成员函数。构造函数请用初始化列表实现对数据成员的初始化。

```
class Point{
public:
    Point( double newX=0, double newY=0);
    Point(const Point& p);
    ~Point();
    void setValue(double newX, double newY);
    double getX( ) const;
    double getY() const;
private:
    double x, y;
};
```

---

---

---

---

---

---

---

---





---

---

---

---

---

---

测试的主程序如下：

```
int main(){
    Point p1(3,4);
    Point p2(5,2);
    double distance = p1.getDistance(p2); //getDistance 是 p1 的成员函数
    cout<<"Distance: "<<distance<<endl;
    return 0;
}
```

### 3) 计算 2 点之间距离：普通函数版本

定义普通函数 `getDistance`，计算 2 点之间距离，函数原型如下：

```
double getDistance(const Point& p1, const Point& p2);
```

---

---

---

---

---

---

---

---

---

---

测试的主程序如下：

```
int main(){
    Point p1(3, 4);
    Point p2(5, 2);
    double distance = getDistance(p1,p2);
```

```

        cout<<"Distance: "<<distance<<endl;
        return 0;
    }

```

#### 4) 计算 2 点之间距离：友元函数版本

将 3)中的普通函数声明为 **Point** 的友元，重新实现 **getDistance** 函数，在函数中不调用 **getX** 和 **getY** 即可访问 **Point** 对象的数据成员。

```

class Point{
public:
    Point( double newX=0, double newY=0);
    Point(const Point& p);
    ~Point();
    void setValue(double newX, double newY);
    double getX( )const;
    double getY()const;

```

```

private:
    double x, y;
};

```

测试的主程序和 3)中的相同。

#### 5) 实现三角形类 **Triangle**

基于 **Point** 类，实现三角形类 **Triangle**，**Triangle** 类包含 3 个 **Point** 类对象，描述其 3 个顶点坐标，**Triangle** 类的定义如下。

```

class Triangle{

```

[illegible]

测试主程序如下

```
int main(){
    Point p1(0,0), p2(0,3), p3(4,0);
    Triangle t(p1,p2,p3);
    cout<<"Area: "<<t.getArea()<<endl;
    cout<<"Perimeter: "<<t.getPerimeter()<<endl;
    return 0;
}
```

### 3.3 课后练习

1. 写出程序输出结果

```
#include <iostream>
using namespace std;
class Example{
private:
    int i;
public:
    Example(int n) {
        i=n;
        cout<<"Constructing. "<<endl;
    }
    ~Example() { cout<<"Destructing. "<<endl; }
    int get_i() { return i; }
};
int sqrt_it(Example o) {
    return o.get_i()*o.get_i();
}
int main(){
    Example x(10);
    cout<<x.get_i()<<endl;
    cout<<sqrt_it(x)<<endl;
    return 0;
}
```

2. 写出程序输出结果

```
#include <iostream>
```

```

using namespace std;
class Test{
public:
    Test() { cout<< "Default constructor." <<endl; }
    Test(const Test& t)
    { cout<< "Copy constructor!" <<endl; }
};
void fun(Test p) {}
int main(){
    Test a;
    fun(a);
    return 0;
}

```

3. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Dog{
public:
    static int number;
    Dog() {
        number++;
        cout<<"New Dog"<<endl;
    }
    ~Dog() {
        number--;
        cout<<"A Dog Die"<<endl;
    }
};
int Dog::number=0;
int main(){
    Dog dog;
    Dog *pDog=new Dog();
    delete pDog;
    cout<<Dog::number<<endl;
    return 0;
}

```

4. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Test{
public:
    Test(int xx=1):x(xx){ }
    void output()const{ cout<<"x:"<<x<<endl;}
private:
    int x;
};
int main(){
    Test t;
    t.output();
    t=4;
    t.output();
    return 0;
}

```

5. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Test{
public:
    Test(){ cout<<"Default Constructor\n";}
    Test(int xx):x(xx){ cout<<"Int Constructor\n";}
    Test(const Test& t):x(t.x){ cout<<"Copy Constructor\n";}
private:
    int x;
};
Test t;
int main(){
    cout<<"-----\n";
    Test tt(t);
    return 0;
}

```

6. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Point {

```

```

        int x,y;
public:
    Point(int x1=0, int y1=0) :x(x1), y(y1) {
        cout<<"Point:"<<x<<' '<<y<<'\n';
    }
    ~Point() {
        cout<<"Point destructor!\n";
    }
};

class Circle {
    Point center;//圆心位置
    int radius;//半径
public:
    Circle(int cx,int cy, int r):center(cx,cy),radius(r) {
        cout<<"Circle radius:"<<radius<<'\n';
    }
    ~Circle() {cout<<"Circle destructor!\n";}
};

int main(){
    Circle c(3,4,5);
    return 0;
}

```

#### 7. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Test{
public:
    Test() { cout<< "Hello: " << ++i<<endl; }
    static int i;
};

int Test::i=0;
int main(){
    Test t[2];
    Test *p;
    p=new Test[2];
    return 0;
}

```

8. 写出程序输出结果

```
#include <iostream>
#include <string>
using namespace std;
class Person{
private:
    string name;
    int age;
public:
    Person(string name,int age);
    ~Person(){
        cout<<"Bye! My name is "<<name<<",I'm "<<age
        <<" years old."<<endl;
    }
    void growup() { age++; }
};
Person::Person(string name, int age){
    this->name=name;
    this->age=age;
    cout<<"Hello,"<<name<<" is comming!"<<endl;
}
int main(){
    Person p("zhang",1);
    for(int i=0;i<90;++i)
        p.growup();
    return 0;
}
```

9. 下面的程序无法通过编译，分析原因并提出修改办法。

```
#include <iostream>
#include <vector>
class Test{
private:
    int a;
public:
    Test(const Test& t)=default;
};
int main(){
```



```

    Test t;
    return 0;
}

```

10. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Test{
private:
    int a,b;
public:
    Test():Test(1){ cout<<"Constructor without parameter!\n";}
    Test(int x):Test(x,10){ cout<<"Constructor with 1 parameter!\n";}
    Test(int x,int y):a(x),b(y){ cout<<"Constructor with 2 paramter!\n";}
    void output()const{ cout<<a<<" "<<b<<endl;}
};
int main(){
    Test t;
    t.output();
    Test t1(3,5);
    t1.output();
    return 0;
}

```

11. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Test{
private:
    int a;
public:
    Test(){ cout<<"Default constructor!\n";}
    Test(int x):a(x){ cout<<"Constructor!\n";}
    Test(const Test& t){
        a=t.a;
        cout<<"Copy constructor!\n";
    }
    Test(Test&& t){
        a=t.a;
    }
}

```

```

        cout<<"move copy constructor!\n";
    }
};

Test fun(){
    return Test(3);
}

int main(){
    Test t;
    Test t2=move(fun());
    return 0;
}

```

12. 写出程序输出结果

```

#include <iostream>
#include <vector>
using namespace std;
class Element{
private:
    int a;
public:
    Element(int e=0):a(e){ }
    Element(const Element& e):Element(e.a){ cout<<"Copy constructor!\n";}
    Element(Element&& e):Element(e.a){ cout<<"Move copy constructor!\n";}
};

int main(){
    vector<Element> vec;
    vec.reserve(10);
    vec.push_back(Element(3));
    Element e(5);
    vec.push_back(e);
    return 0;
}

```

## 4 运算符重载

### 4.1 实习目的

- 1) 掌握运算符函数的定义;
- 2) 掌握+、-、++、+=等常用运算符的重载;
- 3) 了解<<运算符、前置和后置++运算符的使用;
- 4) 掌握 C++11 中移动赋值运算符等语言扩展机制。

### 4.2 实习指导

#### 1) 重载输入输出运算符简介

C++中的 `cin` 和 `cout` 用于键盘输入和控制台输出，它们是定义在 `std` 中的全局对象，`cin` 所属的类型是 `istream`，`cout` 所属类型是 `ostream`。

`istream` 重载了 `operator>>`运算符，`ostream` 重载了 `operator<<`运算符，以支持对基本数据类型的输入和输出，但不支持自定义类对象的输入输出。

若希望通过>>和<<运算符输入输出自定义类对象，必须要在自定义类中重载 `operator<<`和 `operator>>`运算符，这两个运算符需要通过友元函数实现。

假定要在 `MyClass` 类中重载 `operator<<`和 `operator>>`，声明的原型如下：

```
MyClass
{
    ... //类中的其它定义省略

    friend ostream& operator<<(ostream& out, const MyClass& o);
    friend istream& operator>>(istream& in, MyClass& o);
};
```

实现了全局的 `operator<<`和 `operator>>`函数后，在主程序中，就可以通过如下代码输出自定义对象。

```
MyClass obj1, obj2;

cout<<obj1<<obj2;

cout<<obj1 相当于调用 operator<<(cout, obj1), 将 cout 传入全局的 operator<<
```

函数中，实现对 obj1 的输出；operator<<返回 ostream&是为了支持级联输出。

## 4.3 实习任务

### 4.3.1 实习任务一

- 1) 在实习 3 的 Complex 类基础上重载 operator+、operator\*、operator+=、operator\*=、前置++和后置++运算符，Complex 的定义如下：

```
class Complex{
public:
    Complex();
    Complex(int n);
    Complex(int n,int d);
    Complex(const Complex& c);
    ~Complex();
    void setValue(int n, int d);
    double getReal() const ;
    double getImage() const;
    double getDistance() const;
    void output() const;
    Complex operator+(const Complex& f)const;
    Complex operator*(const Complex& f)const;
    Complex & operator+=(const Complex & f);
    Complex & operator*=(const Complex & f);
    Complex & Complex ++();//前置++, 实部加 1
    Complex operator++(int);//后置++, 实部加 1
    Complex add(const Complex& c)const;
    void multiply(double coe);
private:
    double real;
    double image;
};
```

给出新增的函数定义。

[illegible]

---

---

---

---

---

---

【提示】注意 `operator+` 和 `operator+=` 的返回值类型，`operator+` 计算 2 个复数对象的和，返回一个新的复数对象；`operator+=` 将右侧的分数加到左侧复数之上，并返回修改后的复数对象引用。

2) 为 1) 中的 `Complex` 类添加 `operator-`、`operator-=` 和 `operator<<` 运算符，其中 `operator-` 和 `operator<<` 使用友元函数的方法实现，`operator-=` 使用成员函数的方法实现。修改后的 `Complex` 类定义如下。

```
class Complex{
public:
    Complex();
    Complex(int n);
    Complex(int n,int d);
    Complex(const Complex& c);
    ~Complex();
    void setValue(int n, int d);
    double getReal() const ;
    double getImage() const;
    double getDistance() const;
void output() const;
    friend ostream& operator<<(ostream& out, const Fraction& f);
    Complex operator+(const Complex& f)const;
    Complex operator*(const Complex& f)const;
    Complex & operator+=(const Complex & f);
    Complex & operator*=(const Complex & f);
    Complex & Complex ++();//前置++, 实部加 1
    Complex operator++(int);//后置++, 实部加 1
    Complex& operator-=(const Complex & f);
    friend Complex operator-(const Complex & f1, const Complex & f2);
private:
```

[illegible]

### 1) 完善 Time 类

删除 Time 类中的 output 方法，通过重载 operator<<运算符进行替代。删除

Time 类的 getTimeSpan 方法，通过重载 operator-运算符进行替代。

```
class Time{
public:
    Time();
    Time(int h,int m); //通过指定 h 和 m 构造 Time 对象
    Time( int minutes ); //通过指定分钟数构造 Time 对象
    void setTime(int h, int m); //设置新的时间
void output() const; //以 hh:mm 格式输出时间
    friend ostream& operator<<(ostream& out, const Time& t);
    int getHour() const; //获得小时
    int getMinute() const; //获得分钟
    int getTotalMinutes() const; //获得从 0 点 0 分起的总分钟数
Time getTimeSpan( Time &newTime ); //计算时间差
    Time operator-(const Time& newTime); //计算时间差
private:
    int hour;
    int minute;
    void normalizeTime(); //规范化小时为 0~23, 分钟为 0~59
};
```

[illegible]



---

---

---

---

---

## 2) 完善 ParkingCard 类

在实习 3 中的 ParkingCard 类基础上，修改 getTotalExpense 方法，不再调用 Time 类的 getTimeSpan 方法，而是通过 operator- 计算时间差。删除 ParkingCard 的 output 方法，通过重载 operator<< 运算符进行替代。

```
class ParkingCard{
public:
    ParkingCard(double newRate) { rate = newRate; }
    void setRate(double newRate) { rate = newRate; } //设置每小时费率
    double getRate() const { return rate; }
    void setParkingTime( const Time &time); //设置开始停车时间
    void setLeavingTime( const Time &time ); //设置离开时间
    double getTotalExpenses() const; //计算停车费用
    //停车时间分钟数不足半小时按半小时计算，不足 1 小时按 1 小时计算
    void output(); //输出停车起始时间、总时间、费率及总费用
    friend ostream& operator<<(ostream& out,
        const ParkingCard& t);
private:
    double rate; //停车费率，按元/每小时计算
    Time parkingTime; //开始停车时间
    Time leavingTime; //离开时间
};
```

测试的主程序如下：

```
int main(){
    ParkingCard card(5);
    card.setParkingTime(Time(9, 20) );
    card.setLeavingTime(Time(11, 35) );
    cout<<"Expenses: "<<card.getTotalExpenses()<<endl;
    cout<<"Detailed info:\n";
    cout<<card<<endl;
    return 0;
```

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

定义并实现 **MyString** 类，模拟简单的字符串处理功能。**MyString** 通过 `char *` 管理动态分配的字符串。实现字符串输出、连接、下标运算符、赋值运算符等功能，支持移动语义的拷贝构造函数和赋值运算符，**MyString** 类的定义如下。

```
#include <iostream>
#include <vector>
#include <cstring>
using namespace std;
```

[illegible]

[illegible]

```

int main(){
    MyString s("Hello");
    cout<<s<<endl;
    MyString s2=s+"World";
    cout<<s2<<endl;
    cout<<"Index 5:"<<s2[5]<<endl;
    MyString s3(move(s2));
    cout<<s3<<endl;
    return 0;
}

```

## 4.4 课后练习

### 1. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Test{
private:
    int x;
public:
    Test(int xx=0):x(xx){ }
    Test& operator++(){ x++;return *this;}
    Test operator++(int){ Test temp(*this);  x++;  return temp;}
    int getValue()const{return x;}
};
int main(){
    Test t;
    cout<<t.getValue()<<endl;
    cout<<(t++).getValue()<<endl;
    cout<<(++t).getValue()<<endl;
    return 0;
}

```

### 2. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Matrix{

```

```

    double * elem;
    int row,col;
public:
    Matrix(int r,int c){
        row=r;col=c;
        elem=new double[row *col];
    }
    double &operator() (int x,int y)    {
        return elem[ col *(x-1)+y-1];
    }
    double &operator()(int x,int y) const    {
        return elem[ col*(x-1)+y-1];
    }
    ~Matrix(){ delete [] elem; }
};

int main(){
    Matrix m(5,8);
    int i;
    for(i=1;i<6;i++)
        m(i,1)=i+5;
    for(i=1; i<6; i++)
        cout<<m(i, 1)<<" , " ;
    cout<<endl;
    return 0;
}

```

### 3. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Array{
private:
    int *pdata;
    int size;
public:
    Array():size(0){ pdata=nullptr;}
    Array(int s):size(s){
        pdata=new int[size];
    }
}

```

```

~Array(){ delete[] pdata;}
Array(const Array& a):size(a.size){
    pdata=new int[size];
    for(int i=0;i<size;++i) pdata[i]=a[i];
}
Array(Array&& a):size(a.size){
    pdata=a.pdata;
    a.size=0;
    a.pdata=nullptr;
}
Array& operator=(const Array& a){
    if(this==&a) return *this;
    size=a.size;
    pdata=new int[size];
    for(int i=0;i<size;++i) pdata[i]=a[i];
    return *this;
}
Array& operator=(Array&& a){
    if(this==&a) return *this;
    size=a.size;
    pdata=a.pdata;
    a.size=0;
    a.pdata=nullptr;
    return *this;
}
const int& operator[](int index)const{ return pdata[index];}
int& operator[](int index){ return pdata[index];}
int length()const{ return size;}
};

int main(){
    Array a1(3);
    for(int i=0;i<3;++i) a1[i]=i;
    cout<<"length of a1:"<<a1.length()<<endl;
    Array a2(a1);
    cout<<"length of a1:"<<a1.length()<<endl;
    cout<<"length of a2:"<<a2.length()<<endl;
    Array a3(move(a1));

```

```

        cout<<"length of a1:"<<a1.length()<<endl;
        cout<<"length of a3:"<<a3.length()<<endl;
        return 0;
    }

```

4. 写出程序输出结果

```

#include <iostream>
#include <vector>
using namespace std;
class Fraction{
private:
    int num,den;
public:
    explicit Fraction(int n=0,int d=1):num(n),den(d){ }
    explicit operator double()const { return 1.0*num/den; }
    Fraction operator+(const Fraction& f)const{
        return Fraction(num*f.den+den*f.num,den*f.den);
    }
    void output()const{ cout<<num<<"/"<<den<<endl;}
};

int main(){
    Fraction f1(1,3);
    f1.output();
    Fraction f2=f1+static_cast<Fraction>(1);
    f2.output();
    cout<<1+static_cast<double>(f1)<<endl;
    return 0;
}

```

5. 写出程序输出结果

```

#include <iostream>
#include <vector>
using namespace std;
class Array1D{
private:
    vector<int> vec;
public:
    void push_back(int data){ vec.push_back(data);}
    int& operator[](int index){ return vec[index];}
}

```



```

        const& operator[](int index)const{return vec[index];}
};
class Array2D{
private:
    vector<Array1D> vec;
public:
    void push_back(const Array1D& a){ vec.push_back(a);}
    Array1D& operator[](int index){ return vec[index];}
    const Array1D& operator[](int index)const{return vec[index];}
};
int main(){
    Array2D aa;
    for(int i=0;i<3;++i){
        Array1D a;
        for(int j=0;j<4;++j)
            a.push_back(j+i);
        aa.push_back(a);
    }
    for(int i=0;i<3;++i){
        for(int j=0;j<4;++j) cout<<aa[i][j]<<" ";
        cout<<endl;
    }
    return 0;
}

```

## 5 继承与多态

### 5.1 实习目的

- 1) 掌握派生类的定义和使用；
- 2) 掌握派生类构造与析构函数的定义；
- 3) 掌握重写基类的成员函数；
- 4) 掌握通过基类指针或引用实现多态的方法。

### 5.2 实习任务

#### 5.2.1 实习任务一

- 1) 导入实习 3 的 Point 类，其定义如下。

```
#include <iostream>
#include <cmath>
using namespace std;
class Point{
public:
    Point( double newX=0, double newY=0);
    Point(const Point& p);
    ~Point();
    void setValue(double newX, double newY);
    double getX( ) const;
    double getY() const;
    double getDistance( const Point& p2) const;
private:
    double x, y;
};
```

现要定义处理 3 维点的类，而又不能直接修改 Point 类，以 Point 类作为基类派生得到 Point3D，Point3D 的定义和部分功能已经实现，请补充未完成的部分成员函数。要求构造函数通过初始化列表实现。

```
class Point3D : public Point{
public:
```

[illegible]

测试的主程序如下:

64

```

    Point p1(3, 4), p2(5,3);
    Point3D p1_3D(3,4,6);
    Point3D p2_3D(2,6,9);
    double dis=p1.getDistance(p2); //计算二维点 p1 和 p2 的距离
    cout<<"Distance between p1 and p2: "<<dis<<endl;
    dis=p1_3D.getDistance(p2_3D); //计算 3 维点 p1_3D 和 p2_3D 的距离
    cout<<"Distance between p1_3D and p2_3D: "<<dis<<endl;
    return 0;
}

```

2) 修改主程序如下，记录程序的运行结果。

```

int main(){
    Point p1(3, 4), p2(5,3);
    Point3D p1_3D(3,4,6);
    Point3D p2_3D(2,6,9);
    double dis=p1.getDistance(p2); //计算二维点 p1 和 p2 的距离
    cout<<"Distance between p1 and p2: "<<dis<<endl;
    dis=p1.getDistance(p2_3D); //计算 p1 和 p2_3D 的距离
    cout<<"Distance between p1 and p2_3D: "<<dis<<endl;
    return 0;
}

```

运行结果: \_\_\_\_\_

---

3) 修改主程序如下。

```

int main(){
    Point p1(3, 4), p2(5,3);
    Point3D p1_3D(3,4,6);
    Point3D p2_3D(2,6,9);
    double dis=p1.getDistance(p2); //计算二维点 p1 和 p2 的距离
    cout<<"Distance between p1 and p2: "<<dis<<endl;
    dis=p1.getDistance(p2_3D); //计算 p1 和 p2_3D 的距离
    cout<<"Distance between p1 and p2_3D: "<<dis<<endl;
    dis=p1_3D.getDistance(p2); //计算点 p1_3D 和 p2 的距离
    cout<<"Distance between p1_3D and p2: "<<dis<<endl;
    return 0;
}

```

该程序无法通过编译，请分析可能的原因：

---

---

【提示】如果某个函数调用需要传递基类对象，可以将派生类对象传递给他，此时将发生切片效果，只将基类需要的数据复制过去。相反，如果需要 1 个派生类对象，传递基类对象将无法通过编译。

- 4) 修改 Point3D 类，为其增加如下的构造函数，通过 1 个 Point 对象构造 Point3D 对象，实现办法是将 z 设置为 0。

```
class Point3D : public Point{
public:
    Point3D(double newX=0, double newY=0, double newZ=0);
    Point3D(const Point& p);
    double getZ()const;
    double getDistance( const Point3D& p)const;
private:
    double z;
};
Point3D::Point3D(const Point& p): Point(p), z(0) { }
```

重新编译程序，记录运行结果。

运行结果： \_\_\_\_\_

---

【提示】只含有 1 个参数的构造函数称为类型转换构造函数，可以将传入参数类型转换为当前类类型。如果需要将基类对象复制给派生类对象，可以通过类型转换构造函数间接实现。

## 5.2.2 实习任务二

- 1) 调试运行下面的程序，写出程序的运行结果。

```
#include <iostream>
using namespace std;
const double PI=3.14;
class Shape{
public:
```

```

        double getArea()const { return 0; }
        double getPerimeter()const { return 0; }
};

class Circle: public Shape{
public:
    Circle(double r) : radius(r) {}
    double getArea() const;
    double getPerimeter() const;
private:
    double radius;
};

double Circle::getArea()const{
    return PI*radius*radius;
}

double Circle::getPerimeter()const{
    return 2*PI*radius;
}

class Rectangle: public Shape{
public:
    Rectangle( double w, double h): width(w), height(h) {}
    double getArea()const;
    double getPerimeter()const;
private:
    double width;
    double height;
};

double Rectangle::getArea()const{
    return width*height;
}

double Rectangle::getPerimeter()const{
    return 2*(width+height);
}

void outputInfo(const Shape& sh);

int main(){
    Shape shape0;
    Rectangle shape1(3,4);
    Circle shape2(1.0);

```

```

        outputInfo(shape0);
        outputInfo(shape1);
        outputInfo(shape2);
        return 0;
    }
    void outputInfo(const Shape& sh){
        cout<<"Area: "<<sh.getArea()<<endl;
        cout<<"Perimeter: "<<sh.getPerimeter()<<endl;
    }

```

运行结果: \_\_\_\_\_

---



---



---



---



---



---

【想一想】输出结果和你的预期一致吗？请分析输出结果的原因。

---



---



---

- 2) 修改 1)中的 Shape 类，将 `getArea` 和 `getPerimeter` 定义为虚函数，以实现多态效果。重新运行程序，记录程序的运行效果。

```

class Shape{
public:

```

---



---

```

};

```

运行结果: \_\_\_\_\_

---



---



---

- 
- 
- 3) 修改 2) 中的 Shape 类，将 `getArea` 和 `getPerimeter` 定义为纯虚函数，以实现多态效果。修改主函数，重新运行程序，记录程序的运行效果。

```
class Shape{  
public:  
  
  
  
  
};  
int main(){  
    Shape shape0;  
    Rectangle shape1(3,4);  
    Circle shape2(1.0);  
    outputInfo(shape0);  
    outputInfo(shape1);  
    outputInfo(shape2);  
    return 0;  
}
```

运行结果: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

- 4) 从 Shape 派生得到 Triangle 类（三角形类），在其中定义 3 个边长数据成员，实现构造函数和虚函数 `getArea` 和 `getPerimeter`。类的定义如下：

```
class Triangle : public Shape{  
public:  
    Triangle(double newA, double newB, double newC);  
    double getArea( )const;  
    double getPerimeter( )const;  
private:  
    double a, b, c;  
};
```

【提示】计算三角形面积，可以参考海伦公式。调用 `sqrt` 函数时需要包含头



This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

```
void outputInfo(const Shape& sh);

int main(){

    Rectangle shape1(3,4);
    Circle shape2(1.0);
    Triangle shape3(3,4,5);
    outputInfo(shape1);
    outputInfo(shape2);
    outputInfo(shape3);
    return 0;
}

void outputInfo(const Shape& sh){
    cout<<"Area: "<<sh.getArea()<<endl;
    cout<<"Perimeter: "<<sh.getPerimeter()<<endl;
}
```

运行结果: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

## 5.3 课后练习

### 1. 写出程序输出结果

```
#include <iostream>
using namespace std;
class Base {
public:
    void display() {cout<<"Base display"<<endl; }
};
class Derived : public Base {
public:
    void display() { cout<<"Derived display"<<endl; }
};
void display(Base & rr){
    rr.display();
}
int main(){
    Base b;
    Derived d;
    display(b);
    display(d);
    return 0;
}
```

### 2. 写出程序输出结果

```
#include <iostream>
using namespace std;
class Person{
public:
```

```

    Person() {
        cout<<"Person 构造函数! "<<endl;
    }
    ~Person() {
        cout<<"Person 被析构! "<<endl;
    }
};
class Student : public Person
{
public:
    Student() { cout<<"Student 构造函数! "<<endl;}
    ~Student() { cout<<"Student 被析构! "<<endl;}
};
class Teacher : public Person{
public:
    Teacher() {
        cout<<"Teacher 构造函数! "<<endl;
    }
    ~Teacher()
    { cout<<"Teacher 被析构! "<<endl;}
};
int main(){
    Student s;
    Teacher t;
    return 0;
}

```

### 3. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Animal{
public:
    virtual void Report() { cout<<"Report from Animal! "<<endl; }
};
class Tiger : public Animal{
public:
    virtual void Report() { cout<<"Report from Tiger! "<<endl; }
};

```

```

class Monkey : public Animal{
public:
    virtual void Report() {
        cout<<"Report from Monkey! "<<endl;
    }
};

void show(Animal *p){
    p->Report();
}

int main(){
    Tiger tiger;
    Monkey monkey;
    Animal animal=tiger;
    show(&tiger);
    show(&monkey);
    show(&animal);
    return 0;
}

```

4. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Base{
private:
    int base;
public:
    Base(int b) {
        base=b;
        cout<<"base="<<b<<endl;
    }
    ~Base() { }
};

class Derived : public Base{
private:
    Base bb;
    int derived;
public:
    Derived(int d,int b,int c) : bb(c) , Base(b) {

```

```

        derived=d;
        cout<<"derived="<<derived<<endl;
    }
    ~Derived() { }
};

int main(){
    Derived d(3,4,5);
    return 0;
}

```

5. 写出程序输出结果

```

#include <iostream>
using namespace std;
class Base{
public:
    Base (int i,int j){ x0=i; y0=j;}
    void Move(int x,int y){ x0+=x; y0+=y;}
    void Show(){ cout<<"Base("<<x0<<","<<y0<<")"<<endl;}
private:
    int x0,y0;
};

class Derived: private Base{
public:
    Derived(int i,int j,int m,int n):Base(i,j){ x=m; y=n;}
    void Show (){cout<<"Next("<<x<<","<<y<<")"<<endl;}
    void Move1(){Move(2,3);}
    void Show1(){Base::Show();}
private:
    int x,y;
};

int main( ){
    Base b(1,2);
    b.Show();
    Derived d(3,4,10,15);
    d.Move1();
    d.Show();
    d.Show1();
    return 0;
}

```

```
}
```

6. 写出程序输出结果

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
class Sales{
private:
    string product;
    double price;
    double quantity;
public:
    Sales(string prod,double p,double q):product(prod),price(p),quantity(q){ }
    virtual double net_price()const{return price*quantity;}
};
class DiscountSales:public Sales{
private:
    double rate;
public:
    DiscountSales(string prod,double p,double q,double r)
        :Sales(prod,p,q),rate(r){ }
    virtual double net_price()const{return Sales::net_price()*rate;}
};
class FullDiscountSales:public Sales{
private:
    double fullMoney,discountMoney;
public:
    FullDiscountSales(string prod,double p,double q,double f,double d)
        :Sales(prod,p,q),fullMoney(f),discountMoney(d){ }
    virtual double net_price()const{
        double money=Sales::net_price();
        int count=0;
        while((money-=fullMoney)>=0) { count++;}
        return Sales::net_price()-count*discountMoney;
    }
};
int main(){
```

```

vector< Sales* > vec;
vec.push_back(new DiscountSales("C++",100,2,0.8));
vec.push_back(new FullDiscountSales("Java",80,5,200,30));
cout<<vec[0]->net_price()<<endl;
cout<<vec[1]->net_price()<<endl;
double totalPrice=0;
for(auto p : vec) totalPrice+=p->net_price();
cout<<"totalPrice:"<<totalPrice<<endl;
for(auto p: vec) delete p;
return 0;
}

```

## 6 模板

### 6.1 实习目的

- 1) 熟悉类模板的定义;
- 2) 熟悉函数模板的定义与参数匹配;
- 3) 了解栈的基础知识。

### 6.2 实习任务

#### 6.2.1 实习任务一

- 1) 调试运行下面的程序，记录运行结果。

```
#include <iostream>
using namespace std;
template<typename T>
T average( T *pd, int n) {
    T sum=0;
    int i=0;
    while(i<n)
        sum+=pd[i++];
    return sum/n;
}
int main(){
    double a[]={2.5,4.5,6.5,8.5};
    cout<<"Average of a: "<<average(a, 4)<<endl;
    int b[]={3,5,7,8};
    cout<<"Average of b: "<<average(b, 4)<<endl;
    return 0;
}
```

运行结果: \_\_\_\_\_

---

- 2) 调试运行下面的程序，记录运行结果。

```
#include <iostream>
```



```

using namespace std;
template<typename T1,typename T2,typename T3>
T1 sum(T2 a, T3 b){
    return a+b;
}
int main(){
    auto r1=sum<double,int,double>(2,5.5);
    cout<<"result1: "<<r1<<endl;
    auto r2=sum<double,int,int>(2.3,5.6);
    cout<<"result2: "<<r2<<endl;
    auto r3=sum<int>(2.4,5.5);
    cout<<"result3: "<<r3<<endl;
    return 0;
}

```

运行结果: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

## 6.2.2 实习任务二

- 1) 存放整型数据的栈类定义及实现如下:

```

class Stack{
public:
    Stack(int size=100); //按指定的大小构建堆栈
    ~Stack();
    void push(int data); //向栈顶添加元素
    int pop(); //弹出并返回栈顶元素
    bool isEmpty(); //判断堆栈是否为空
    bool isFull(); //判断堆栈是否满
private:
    int *pData; //指向堆中为堆栈分配的数据
    int stackSize; //堆栈当前容量
    int top; //栈顶的下标位置
};
Stack::Stack(int size){
    pData=new int[size];
}

```

```

        stackSize=size;
        top=-1;
    }
    Stack::~Stack(){
        delete [] pData;
    }
    void Stack::push(int data){
        if(isFull())    {
            cout<<"Stack is full!\n";
            exit(0);
        }
        top++;
        pData[top]=data;
    }
    int Stack::pop(){
        if(isEmpty()) {
            cout<<"Stack is empty!\n";
            exit(0);
        }
        int temp=pData[top];
        top--;
        return temp;
    }
    bool Stack::isEmpty(){
        return top==-1;
    }
    bool Stack::isFull(){
        return top==stackSize-1;
    }

```

按照如下的主程序进行测试，记录运行结果。

```

int main(){
    int array[10]={ 1, 3, 2, 4};
    Stack s(4);    //创建堆栈 s，设置初始大小为 4
    int i;
    for(i=0;i<3;i++)
        s.push(array[i]);
    while(!s.isEmpty())

```

```

        cout<<s.pop()<<' ';
    for( i=0;i<10;i++)
        s.push(array[i]);
    cout<<endl;
    return 0;
}

```

运行结果: \_\_\_\_\_

\_\_\_\_\_

- 2) 将 1)中实现的堆栈数据结构修改为基于模板类型，类的定义如下，给出所有成员函数的实现。

```

template <typename T>
class Stack{
public:
    Stack(int size=100);
    ~Stack();
    void push(T data);
    T pop();
    bool isEmpty();
    bool isFull();
private:
    T *pData;
    int stackSize;
    int top;
};

```

---



---



---



---



---



---



---



---



---



---

[illegible]

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and extend across the width of the page. There are no margins, text, or other markings on the paper.

【提示】编写模板类时，将定义和成员函数的实现放在同一个文件（头文件中），在主程序中包含头文件，才能顺利通过编译。

测试的主程序如下:

```
int main(){
    int array[10]={ 1, 3, 2, 4};
    Stack<int> s(4);    //创建堆栈 s， 设置初始大小为 4
    int i;
    for( i=0;i<3;i++)
        s.push(array[i]);
    while(!s.isEmpty())
        cout<<s.pop()<<' ';
    for( i=0;i<10;i++)
        s.push(array[i]);
    cout<<endl;
    return 0;
}
```

### 6.2.3 实习任务三

- 1) 将课件中的 Array1D 类改造为基于模板实现，以实现对不同数据类型的统一处理。数组类的定义及部分成员函数实现如下，给出其它成员函数的实现。

```
template <typename T>
class Array1D{
public:
    Array1D (int newSize); //指定数组大小
    Array1D (T *p, int newSize); //指定数组大小并传递初始值
    Array1D ( const Array1D <T>& a); //拷贝构造函数
    ~Array1D ();
    int getSize()const; //获取数组的大小
    T max()const; //获取数组的最大值
    void reverse(); //逆转数组元素
    const T& operator[](int index)const;
    T& operator[](int index);
    Array& operator=(const Array<T>& a); //赋值运算符
    friend ostream& operator<<(ostream& out, const Array1D <T>& t);
private:
    T *pData; //指向堆中数组的指针
    int size; //数组元素个数
};
```

测试主程序如下：

```
int main(){
    Array1D <int> array(5);
    for(i=0;i<array.getSize();++i)
        array [i]=i+1;
    cout<<"Max: "<< array.max()<<endl;
    for(i=0;i<array.getSize();++i)
        cout<<array[i]<< " ";
    cout<<endl;
    cout<<array;
    return 0;
}
```

---

---

[illegible]

[illegible]

### 6.3 课后练习\*

1. 写出下面程序的运行结果

```
#include <iostream>
#include <string>
using namespace std;
template <typename T>
void f(const T& a,const T& b){
    cout<<a+b<<endl;
```



```

}
int main(){
    string s="Time",t="Marches";
    f(4,5);
    f(s,t);
    return 0;
}

```

2. 写出下面程序的运行结果

```

#include <iostream>
#include <vector>
using namespace std;
template<typename T>
class Array1D{
private:
    vector<T> vec;
public:
    Array1D(int size){ vec.resize (size);}
    T& operator[](int index){ return vec[index];}
    const T& operator[](int index)const{
        return vec[index];
    }
    int size()const{ return vec.size ();}
};
int main(){
    Array1D<int> a(10);
    cout<<"size:"<<a.size ()<<endl;
    for(int i=0;i<a.size ();++i)
        a[i]=i*i;
    for(int i=0;i<a.size();++i)
        cout<<a[i]<<" ";
    cout<<endl;
    return 0;
}

```

3. 写出下面程序的运行结果

```

#include <iostream>
#include <string>
using namespace std;

```

```

template<typename T1,typename T2>
auto sum(const T1& a,const T2& b)->decltype(a+b){
    return a+b;
}
int main(){
    char str[]{"helloworld"};
    cout<<"first:"<<sum(string("hello"),"world")<<endl;
    cout<<"second:"<<sum(10,'a')<<endl;
    cout<<"third:"<<sum(string("hello"),'w')<<endl;
    cout<<"fourth:"<<sum(str,3)<<endl;
    return 0;
}

```

4. 写出下面程序的运行结果

```

#include <iostream>
#include <string>
using namespace std;
template<typename T>
T add(T a, T b){return a+b;}
template<typename T>
T sub(T a, T b){return a-b;}
template<typename T1, typename T2>
T1 cal(T1 a, T1 b, T2 f){
    return f(a,b);
}
int main(){
    int a=3,b=5;
    cout<<"First:"<<cal(a,b,add<decltype(a+b)>)<<endl;
    cout<<"Second:"<<cal(a,b,sub<decltype(a-b)>)<<endl;
    string s1{"hello"};
    string s2{"world"};
    cout<<"Third:"<<cal(s1,s2,add<string>)<<endl;
    return 0;
}

```

5. 写出下面程序的运行结果

```

#include <iostream>
#include <string>
#include <vector>

```

```

using namespace std;
template<typename T,typename V>
T search(T begin,T end,V value){
    while(begin!=end){
        if(*begin==value) return begin;
        begin++;
    }
    return end;
}
int main(){
    int a[3]={ 5,9,10};
    auto it=search(begin(a),end(a),9);
    if(it==end(a)) cout<<"No\n";
    else
        cout<<*it<<" Yes\n";
    vector<string> v{ "hello","world","welcome" };
    if(search(v.begin (),v.end (), "test")==v.end ())
        cout<<"No\n";
    else
        cout<<"Yes\n";
    return 0;
}

```

6. 写出下面程序的运行结果

```

#include <iostream>
#include <vector>
using namespace std;
template<typename T>
bool greaterFun(T a,T b){return a>=b;}
template<typename T>
bool lessFun(T a,T b){return a<b;}
template<typename T,typename T2,typename C>
T search(T begin,T end,T2 value,C con){
    while(begin!=end){
        if(con(*begin,value)) return begin;
        begin++;
    }
    return end;
}

```

```

}
int main(){
    int a[]={90,87,56,100,82};
    auto it=search(begin(a),end(a),100,greaterFun<int>);
    if(it!=end(a))
        cout<<"First:"<<*it<<endl;
    it=search(begin(a),end(a),60,lessFun<int>);
    if(it!=end(a))
        cout<<"Second:"<<*it<<endl;
    return 0;
}

```

## 7 STL 及应用

### 7.1 实习目的

- 1) 了解标准模板库中容器、迭代器、算法的基础概念;
- 2) 掌握基础容器和常用算法的应用;
- 3) 熟悉函数对象、Lambda 等工具的使用。

### 7.2 实习任务

#### 7.2.1 实习任务一

- 1) 调试运行下面的程序，记录运行结果。

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main(){
    vector<int> vec{ 1,3,6,20,3,5};
    vec.push_back (17);
    for(auto e: vec)cout<<" "<<e;
    cout<<endl;
    transform(vec.begin (),vec.end (),vec.begin (),
               [](int e){ return e%2==0?e:e*2;});
    for_each(vec.begin (),vec.end (),
              [](int e){ cout<<" "<<e;});
    cout<<endl;
    return 0;
}
```

运行结果: \_\_\_\_\_

\_\_\_\_\_

- 2) 调试运行下面的程序，记录运行结果。

下面的程序由用户输入形如 3\*5 的表达式，系统完成计算。测试如下的输入：

3\*5

50-34

5+11

```
#include <iostream>
#include <ctime>
#include <functional>
using namespace std;
int main(){
    function<int(int,int)> ope;
    while(true){
        int num1,num2;
        char op;
        cout<<"Input the expression(5*3):";
        cin>>num1>>op>>num2;
        if(num1==0)break;
        switch(op){
            case '+':ope=plus<int>();break;
            case '-':ope=minus<int>();break;
            case '*':ope=multiplies<int>();break;
        }
        cout<<num1<<op<<num2<<"="<<ope(num1,num2)<<endl;
    }
    return 0;
}
```

运行结果: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

【提示】C++11 中通过 `function` 模板定义仿函数对象及 `Lambda` 表达式（匿名函数对象）的数据类型，`function<int(int,int)>` 表示的函数对象返回值类型为 `int`，2 个参数类型也为 `int`，可以使用 `auto` 简化这种类型表达。

## 7.2.2 实习任务二

- 1) 下面的程序将容器中的 3 的倍数且不大于 20 的元素复制到另外一个容器之中，判定规则使用普通函数实现，给出 `predicate` 的实现。通过 `copy_if` 算法可以根据指定规则实现选择复制。

```

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
bool predicate(int data){
    _____
    _____
}

int main(){
    vector<int> vec{2,4,5,6,10,15,3,21,36,72,9,13};
    vector<int> result;
    result.resize (vec.size());
    auto end=copy_if(vec.begin (),vec.end (),result.begin(),predicate);
    result.erase(end,result.end ());
    for_each(result.begin (),result.end (),(int e){cout<<e<<endl;});
    return 0;
}

```

【提示】考虑到从 vec 中符合条件复制到 result 中的元素数量的不确定性，初始时设置 result 和 vec 相同大小，复制完成后，copy\_if 返回的 end 表示实际有效元素的下一个位置，通过 erase 方法可以删除 end 到容器尾（result.end()）之间的无效元素。

- 2) 将 1)中的 predicate 方法改用函数对象实现，定义 Predicate 函数对象，在其中实现函数调用运算符，作为判定规则。给出 Predicate 类的实现。

```

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class Predicate{
public:
    _____
    _____
    _____
    _____
};

```

```

int main(){
    vector<int> vec{2,4,5,6,10,15,3,21,36,72,9,13};
    vector<int> result;
    result.resize (vec.size());
    auto end=copy_if(vec.begin (),vec.end (),result.begin(),Predicate());
    result.erase(end,result.end ());
    for (auto e : result ) cout<<e<<endl;
    return 0;
}

```

- 3) 将 2)的基础之上，不使用 Predicate 类，直接在 copy\_if 算法中使用 Lambda 表达式实现相同的判断规则。

```

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main(){
    vector<int> vec{2,4,5,6,10,15,3,21,36,72,9,13};
    vector<int> result;
    result.resize (vec.size());
    auto end=copy_if(vec.begin (),vec.end (),result.begin(),
    _____
    );
    result.erase(end,result.end ());
    for (auto e : result ) cout<<e<<endl;
    return 0;
}

```

- 4) 将 2)中实现的 Predicate 基础之上，增加约束条件，最多复制前 3 个符合条件的元素。给出 Predicate 类的实现。

```

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class Predicate{
public:

```



```

};
int main(){
    vector<int> vec{2,4,5,6,10,15,3,21,36,72,9,13};
    vector<int> result;
    result.resize (vec.size());
    auto end=copy_if(vec.begin (),vec.end (),result.begin(),Predicate(0));
    result.erase(end,result.end ());
    for(auto e: result) cout<<e<<endl;
    return 0;
}

```

【提示】在 Predicate 类中定义成员变量 count，记录找到的符合条件的数据数量，通过构造函数初始化为 0，如果已经达到预定数量，则在 operator()函数中直接返回 false，从而限制数量上限。根据需要，还可以将上限值作为参数传入，增加灵活性。

函数对象相对普通函数，更具有灵活性，可以通过数据成员保存处理状态，如果要在普通函数中实现这种灵活性，需要依赖于全局变量，破坏了封装性，建议对于复杂逻辑处理的情况下，尽量使用函数对象。

5) 将 4)的基础之上，不使用函数对象，使用 Lambda 表达式直接在 copy\_if 算法指定相同的规则。

```

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main(){
    vector<int> vec{2,4,5,6,10,15,3,21,36,72,9,13};

```

```

vector<int> result;
result.resize (vec.size());
int count=0;
auto end=copy_if(vec.begin (),vec.end (),result.begin(),

);

result.erase(end,result.end ());
for(auto e: result) cout<<e<<endl;
return 0;
}

```

【提示】在 main 方法中定义局部变量 count，初始为 0，用于计数。在判断规则的 Lambda 表达式中，需要捕获该变量，并进行计数判断。

Lambda 表达式相对于函数对象，表达更加简洁，避免了定义大量的小型类。但不易在 Lambda 表达式中定义特别复杂的处理逻辑。

### 7.2.3 实习任务三\*

#### 1) 定义 Student 类

Student 类封装了简单的学生信息，其定义如下：

```

#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
class Student{
private:
    string name;
    string specialty;//专业
    long id;// 学号
    double creditPoint; //学分积点
public:
    Student(string n,string s,long i,double c)
        :name(n),specialty(s),id(i),creditPoint(c){ }
    void setCreditPoint(double c){ creditPoint=c;}
    double getCreditPoint()const{return creditPoint;}
}

```

```

    string getName()const{return name;}
    long getId()const{return id;}
    string getSpecialty()const{return specialty;}
    friend ostream& operator<<(ostream& out,const Student& s);
};

ostream& operator<<(ostream& out,const Student& s){
    cout<<"Name:"<<s.name<<endl;
    cout<<"  Id:"<<s.id<<endl;
    cout<<"  Specialty:"<<s.specialty<<endl;
    cout<<"  CreditPoint:"<<s.creditPoint<<endl;
}

int main(){
    vector<Student> vec{
        {"zhang","Computer",11001,4.2},
        {"wang","Computer",11002,3.8},
        {"Li","English",12001,4.1},
        {"Tang","English",12002,3.9},
        {"Qian","Computer",11003,4.0},
        {"Song","Geology",10001,4.1}
    };
    for(auto e: vec)
        cout<<e;
    return 0;
}

```

## 2) 定义 StudentManage 类

在 StudentManage 类中定义 vector<Student>数据成员,保存所有的学生信息,同时定义添加、删除、查询和输出所有学生信息的接口。

添加学生时,输入学生信息,将学生保存到容器中,需要检查学号是否已经存在;删除学生时,输入学号,移除指定学号的学生;查询接口应提供根据姓名和学号的查询功能,按姓名查询时,显示所有姓名匹配的学生信息;输出学生信息时,按照指定的排序规则输出所有学生信息,可根据学号、姓名、专业或学分积点排序。StudengManage 类的接口定义如下,写出未实现的成员函数。

```

class StudentManage{
public:
    enum SortType{BY_NAME,BY_ID,

```

```

        BY_SPECIALTY,BY_CRDITPOINT});
StudentManage()=default;
void addStudent();
void removeStudent();
void setSortType(SortType st);
void findStudent();
void size()const{ return students.size();}
friend ostream& operator<<(ostream& out,const StudentManage& sm);
private:
    vector<Student> students;
    SortType sortType=BY_ID;
};
void StudentManage::setSortType (SortType st){
    function<bool(const Student&,const Student&)> f;
    switch(st){
    case BY_ID: f=[](const Student& s1,const Student& s2)
        {return s1.getName()<s2.getName();};break;
    case BY_NAME:f=[](const Student& s1,const Student& s2)
        {return s1.getId()<s2.getId();};break;
    case BY_SPECIALTY:f=[](const Student& s1,const Student& s2)
        {return s1.getSpecialty()<s2.getSpecialty();};break;
    case BY_CRDITPOINT:f=[](const Student& s1,const Student& s2)
        {return s1.getCreditPoint()<s2.getCreditPoint();};break;
    }
    sort(students.begin(),students.end(),f);
}

```

【补充说明】在 StudentManage 类中定义枚举类型 SortType，用于标识排序类型，在 StudentManage 外部访问该枚举类型时通过 StudentManage::SortType 访问。students 为保存学生信息的 vector 容器，sortType 是 SortType 类型的变量，默认为 BY\_ID（根据 ID 排序）。每次调用 addStudent 方法后，应调用 setSortType 重新排序，removeStudent 后元素仍然有序，无需重新排序。

---



---



---



---

[illegible]



```

        cout<<sm;
        sm.findStudent ();
        sm.removeStudent ();
        cout<<sm;
        return 0;
    }

```

【扩展应用】可以在主程序中添加菜单选项，允许用户多次操作各项功能，实现更加友好的交互功能。

## 7.3 课后练习

1. 写出下面程序的运行结果

```

#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
int main(){
    vector<int>data;
    data.push_back(1);
    data.push_back(3);
    data.push_back(2);
    sort(data.begin(),data.end());
    for(unsigned int i=0; i!=data.size();i++)
        cout<<data.at(i)<< " ";
    cout<<endl;
    return 0;
}

```

2. 写出下面程序的运行结果

```

#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
int main(){
    vector<int>data;
    data.push_back(1);
    data.push_back(3);

```

```

    data.push_back(2);
    sort(data.begin(),data.end());
    vector<int>::iterator it=data.begin();
    for(; it!=data.end(); it++)
        cout<<*it<<" ";
    cout<<endl;
    return 0;
}

```

3. 写出下面程序的运行结果

```

#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
int main(){
    vector<int>data;
    data.push_back(1);
    data.push_back(3);
    data.push_back(2);
    sort(data.begin(),data.end());
    for( auto e :data)
        cout<<e<<" ";
    cout<<endl;
    return 0;
}

```

4. 写出下面程序的运行结果

假定用户输入 hello hello me me me

```

#include <iostream>
#include <map>
#include <string>
using namespace std;
int main(){
    map<string,int> counts;
    string word;
    while(cin>>word)
        ++counts[word];
    for(auto e : counts)
        cout<<e.first<<"\t"<<e.second<<"\n";
}

```



```

        return 0;
    }

```

5. 写出下面程序的运行结果

```

#include <iostream>
#include <list>
#include <algorithm>
#include <numeric>
using namespace std;
double arithMean(const list<int>& nums){
    double sum=accumulate(nums.begin(),nums.end(),0);
    return sum/nums.size();
}
int main(){
    list<int> data;
    data.push_back(2);
    data.push_back(3);
    data.push_back(1);
    data.push_back(4);
    cout<<"mean:"<<arithMean(data)<<endl;
    return 0;
}

```

6. 写出下面程序的运行结果

```

#include <iostream>
#include <list>
#include <algorithm>
#include <numeric>
#include <cmath>
using namespace std;
int product(int num1, int num2){
    return num1*num2;
}
double geoMean(const list<int>& nums){
    double mult=accumulate(nums.begin(),nums.end(),1,product);
    //pow 函数计算 mult 的 1.0/nums.size()次幂
    return pow(mult, 1.0/nums.size());
}
int main(){

```

```

    list<int> data;
    data.push_back(2);
    data.push_back(4);
    data.push_back(1);
    cout<<"mean:"<<geoMean(data)<<endl;
    return 0;
}

```

7. 写出下面程序的运行结果

```

#include <algorithm>
#include <map>
#include <iostream>
using namespace std;
int main( ){
    map<int, int> myMap;
    myMap.insert(make_pair(5, 50));
    myMap.insert(make_pair(6, 60));
    myMap.insert(make_pair(4, 40));
    myMap.insert(make_pair(7, 70));
    for_each(myMap.begin(), myMap.end(),
        [](const pair<int,int>& e)
            { cout<<e.first<<"-"<<e.second<<endl;});
    return (0);
}

```

8. 写出下面程序的运行结果

```

#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
using namespace std;
using namespace std::placeholders;
int main( ){
    vector<int> myVector;
    myVector.push_back(50);
    myVector.push_back(76);
    myVector.push_back(68);
    for_each(myVector.begin(), myVector.end(),
        [](int e){ cout<<e<<" ";});
}

```

```

        cout << endl;
        transform(myVector.begin(), myVector.end(), myVector.begin(),
            bind(plus<int>(), _1,10));
        for(auto e : myVector) cout<<e<<" ";
        cout << endl;
        return (0);
    }

```

9. 写出下面程序的运行结果

```

#include <iostream>
#include <string>
#include <functional>
using namespace std;
template<typename T1, typename T2>
T1 cal(T1 a, T1 b, T2 f){
    return f(a,b);
}
int main(){
    int a=3,b=5;
    cout<<"First:"<<cal(a,b,plus<int>())<<endl;
    cout<<"Second:"<<cal(a,b,minus<int>())<<endl;
    string s1{"hello"};
    string s2{"world"};
    cout<<"Third:"<<cal(s1,s2,plus<string>())<<endl;
    return 0;
}

```

10. 写出下面程序的运行结果

```

#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
using namespace std;
using namespace std::placeholders;
int main(){
    vector<int> vec{ 1,3,5,2,6,9};
    int value=4;
    int cnt=count_if(vec.begin(),vec.end(),
        [=](int i) {return i<value; } );
}

```

```
cout<<"count:"<<cnt<<endl;
cnt=count_if(vec.begin(),vec.end(),
             bind(greater_equal<int>(),_1,value));
cout<<"count:"<<cnt<<endl;
return 0;
}
```

## 8 异常处理与输入输出流

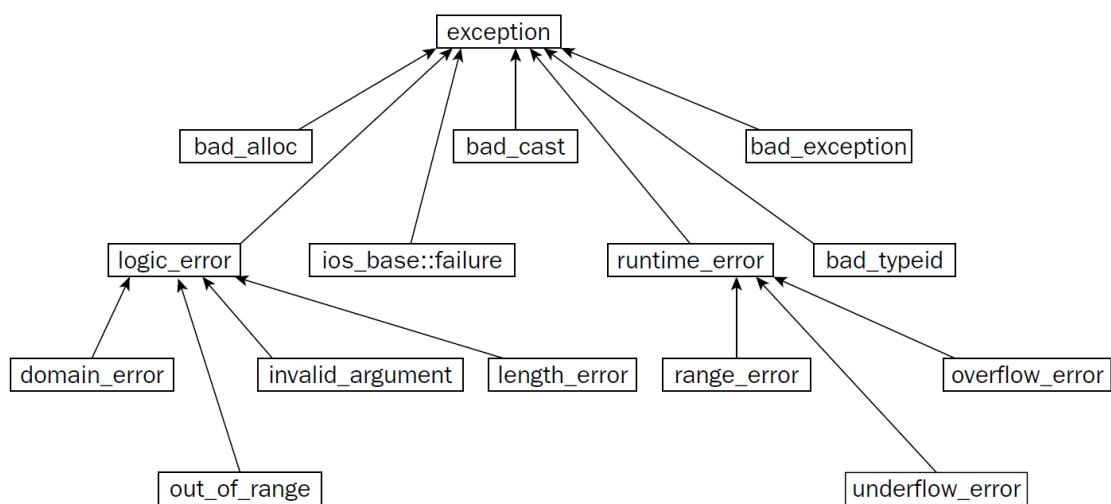
### 8.1 实习目的

- 1) 掌握异常处理的机制和方法；
- 2) 了解基于多态的异常类体系；
- 3) 熟悉常见输入输出流的应用；
- 4) 掌握文件的读写操作。

### 8.2 实习指导

#### 1) C++中的异常类体系简介

C++标准库中定义了常用的异常类体系，如下图所示。



`exception` 类是所有异常类的基类，所有异常类都通过传入 1 个字符串参数来构造（字符串描述异常信息）。所有异常类中都包含 `what` 虚函数，`what` 虚函数返回 `const char *`，指向所包含的异常信息。

若需要使用 `exception` 类及其派生类，需要包含头文件：

```
#include <stdexcept>
```

如果程序中只抛出 `exception` 及其子类对象，采用多态技术，在捕获异常中只需要 1 个 `catch` 分支即可捕获所有的异常对象并能够多态地处理。处理的框架

代码如下：

```
catch(exception & e) { //一定要捕获异常基类引用，才能多态处理
    cout<<e.what()<<endl;
}
```

## 8.3 实习任务

### 8.3.1 实习任务一

1) 调试运行下面的程序，输入不同数据，记录运行结果。

分别测试 b 为 0、 $a < b$ 、无异常输入等几种情况。

```
#include <cmath>
#include <iostream>
using namespace std;
double sqrtIt(double a, double b){
    if( (a-b) < 0 )
        throw "Sqrt negative number!\n";
    return sqrt( a-b );
}
double divide(double a, double b){
    if(b==0)
        throw 0;
    return a/b;
}
int main( ){
    double a, b;
    cout<<"Enter a and b: ";
    cin>>a>>b;
    try {
        cout<<"The Sqrt of a-b: "<<sqrtIt(a, b)<<endl;
        cout<<"a/b: "<<divide(a,b)<<endl;
    }
    catch(const char * pError) {
        cout<<pError<<endl;
    }
}
```

```

        catch ( int error)    {
            cout<<"divided by zero!"<<endl;
        }
    return 0;
}

```

运行结果: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

2) 调试运行下面的程序，记录运行结果。

```

#include <iostream>
#include <fstream>
using namespace std;
int main(){
    ofstream outf("D:\\tmp.txt",ios::trunc);
    outf<<"World Wide Web";
    outf.close();
    ifstream inf("D:\\tmp.txt");
    char str[40];
    inf>>str;
    inf.close();
    cout<<str <<endl;
    return 0;
}

```

运行结果: \_\_\_\_\_

\_\_\_\_\_

## 8.3.2 实习任务二

1) 在实习 6 的堆栈类基础之上，分析代码中可能存在异常的地方。

- 当堆栈已经为空时，执行 pop 操作；
- 当堆栈已满时，执行 push 操作。

根据可能出现的各种异常，自定义系统所需要的异常类体系，并在堆栈类的相应操作中抛出异常。异常基类 `ExceptionBase` 的定义如下。

```
class ExceptionBase { //异常基类
public:
    virtual void showReason() {
        cout<<"Base Exception!\n";
    }
};
```

`PushOnFull` 类和 `PopOnEmpty` 类从 `ExceptionBase` 类派生，给出它们的定义和实现代码。

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

在堆栈类的 `push` 和 `pop` 方法中判断并抛出异常。

```
template <typename T>
void Stack<T>::push(T data){ //栈满时 push 数据抛出 PushOnFull 异常
```



---

---

---

```
    top++;
    pData[top]=data;
}
template <typename T>
T Stack::pop(){ //栈空时 pop 数据抛出 PopOnEmpty 异常
```

---

---

---

```
    T temp=pData[top];
    top--;
    return temp;
}
```

在主程序中，捕获堆栈类中抛出的异常，在 `catch` 中捕获的是基类的引用，从而实现多态异常处理，此时调用 `showReason` 为虚函数。

```
int main(){
    int array[10]={ 1, 3, 2, 4};
    Stack<int> s(4);    //创建堆栈 s，设置初始大小为 4
    cout<<endl;
    try {
        for(int i=0;i<3;i++)
            s.push(array[i]);
        while(!s.isEmpty())
            cout<<s.pop()<<' ';
        for(int i=0;i<10;i++)
            s.push(array[i]);
    }
    catch(ExceptionBase & e) {
        e.showReason();
    }
    return 0;
}
```

## 2) 增加对堆栈类构造函数中非法参数的异常捕获

在 Stack 类的构造函数中，判断初始堆栈大小参数，如果该参数 $\leq 0$ ，则抛出异常。首先定义 ExceptionBase 类的派生类 IllegalParameter，实现其中的 showReason 方法。在 Stack 的构造函数中判断初始化化参数非法，则抛出相应异常。

```
_____  
_____  
_____  
_____  
_____  
_____  
_____  
  
template<typename T>  
Stack<T>::Stack(int size){
```

```
_____  
_____  
_____  
  
    pData=new T[size];  
    stackSize=size;  
    top=-1;  
}
```

测试主程序如下：

```
int main(){  
    int array[10]={ 1, 3, 2, 4};  
    int size;  
    cout<<"Input Stack Size:";  
    cin>>size;  
    try{  
        Stack<int> s(size);  
        for(int i=0;i<3;i++)  
            s.push(array[i]);  
        while(!s.isEmpty())  
            cout<<s.pop()<<' '  
    }  
}
```

```

        catch(ExceptionBase & e){
            e.showReason();
        }
        return 0;
    }

```

运行程序，尝试输入不同的参数，查看显示的不同异常信息。体会多态异常的特点和优势。

### 8.3.3 实习任务三

在实习 7 的 StudentManage 类基础之上，添加 saveData 和 loadData 接口，实现将学生数据保存磁盘文件中以及从磁盘文件中加载学生数据。

```

class StudentManage{
public:
    enum SortType{ BY_NAME,BY_ID,
                  BY_SPECIALTY,BY_CRDITPOINT};
    StudentManage()=default;
    void addStudent();
    void removeStudent();
    void setSortType(SortType st);
    void findStudent();
    void loadData( string filename); //从文件中加载数据
    void saveData( string filename); //将数据保存到磁盘文件中
    friend ostream& operator<<(ostream& out,const StudentManage& sm);
private:
    vector<Student> students;
    SortType sortType=BY_ID;
};

```

---



---



---



---



---



---

[illegible]

测试主程序如下：

```
int main(){
    StudentManage sm;
    sm.loadData("d:\\students.txt" );    //从文件中加载数据
    if(sm.size()==0){
        sm.addStudent();
        sm.addStudent();
        sm.addStudent();
        sm.addStudent();
    }
    cout<<"sort type(0-name,1-id,2-specialty,3-creditpoint):";
    int choice;
    cin>>choice;
    sm.setSortType (static_cast<StudentManage::SortType>(choice));
    cout<<sm;
    sm.findStudent ();
    sm.removeStudent ();
    cout<<sm;
    sm.saveData("d:\\students.txt" );    //将数据保存到文件中
}
```

```

        return 0;
    }

```

【提示】读取文件时首先需要判断文件是否存在，如果文件不存在则不读取数据。保存数据时，应首先保存学生数量，然后依次保存每个学生的基本信息（各属性）；读取数据时，首先读取学生数量，再根据数量循环读取每个学生的信息，创建学生对象并添加到 sm 中。

## 8.4 课后练习

1. 写出下面程序的运行结果

```

#include <iostream>
using namespace std;
class CException{
public:
    virtual void display() {
        cout<<"数组访问越界!\n";
    }
};
class CUpperBoundException : public CException{
public:
    void display()
    { cout<<"数组访问超过最大许可下标"<<endl; }
};
class CLowerBoundException : public CException {
public:
    void display()
    { cout<<"数组访问小于最小许可下标"<<endl; }
};
#define N 3
int getElement(int *p,int index)
{
    if(index>=N) throw CUpperBoundException();
    if(index<0) throw CLowerBoundException();
    return p[index];
}
int main(){

```

```

int a[N]={ 1,2,3};
try {
    cout<<getElement(a,2)<<endl;
    cout<<getElement(a,-1)<<endl;
}
catch (CException & ex)
{ ex.display(); }
try { cout<<getElement(a,4)<<endl; }
catch(CException & ex) { ex.display(); }
return 0;
}

```

2. 写出下面程序的运行结果

假定用户输入 "Every student love C++", 写出程序执行后文件 "D:\1.txt" 中存储的内容。

```

#include <iostream>
#include <fstream>
using namespace std;
int main(){
    char buf[100];
    cin>>buf;
    ofstream of;
    of.open("D:\\1.txt");
    of<<buf;
    of.close();
    return 0;
}

```

3. 写出下面程序的运行结果

```

#include <iostream>
using namespace std;
int f(int n){
    if(n<=0)
        throw n;
    int s=1;
    for(int i=1;i<=n;i++)
        s*=i;
    return s;
}

```

```

int main(){
    try {
        cout<<f(4)<<endl;
        cout<<f(-2)<<endl;
    }
    catch(int n) {
        cout<<"n="<<n<<"不能计算 n! "<<endl;
        cout<<"程序执行结束"<<endl;
    }
    return 0;
}

```

4. 写出下面程序的运行结果

```

#include <iostream>
#include <iomanip>
using namespace std;
int main(){
    cout<<hex<<20<<endl;
    cout<<oct<<10<<endl;
    cout<<setfill('x')<<setw(10);
    cout<<100<<"aa"<<endl;
    return 0;
}

```

5. 写出下面程序的运行结束后，文件 tmp.txt 中存储的内容。

```

#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
using namespace std;
class Fraction{
private:
    int num,den;
public:
    Fraction(int n,int d):num(n),den(d){ }
    void output()const{ cout<<num<<"/"<<den<<endl;}
    int getNum()const{return num;}
    int getDen()const{return den;}
};

```

```

int main(){
    vector<Fraction> vec{{3,5},{2,7},{1,8}};
    ofstream f("d:\\tmp.txt");
    if(!f) return -1;
    f<<vec.size ()<<endl;
    for_each(vec.begin (),vec.end (), [&f](const Fraction& e)
        {f<<e.getNum()<<" "<<e.getDen()<<endl;});
    return 0;
}

```

6. 写出下面程序的运行输出结果。

假定 d:\\tmp.txt 的文件内容如下：

```

3
2 9
3 7
1 4
3 8
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;
class Fraction{
private:
    int num,den;
public:
    Fraction(int n,int d):num(n),den(d){ }
    void output()const{cout<<num<<"/"<<den<<endl;}
    int getNum()const{return num;}
    int getDen()const{return den;}
};
int main(){
    vector<Fraction> vec;
    ifstream f("d:\\tmp.txt");
    if(!f) return -1;
    int size;
    f>>size;
    for(int i=0;i<size;++i){
        int num,den;

```



```

        f>>num>>den;
        vec.push_back (Fraction(num,den));
    }
    for(auto e:vec) e.output();
    cout<<endl;
    return 0;
}

```

7. 写出下面程序的运行输出结果。

```

#include <iostream>
#include <sstream>
using namespace std;
string writeString(int a,int b,char op){
    ostringstream ostr;
    ostr<<a<<op<<b;
    return (ostr.str());
}
int calString(const string& s){
    int a,b,result;
    char op;
    istringstream istr(s);
    istr>>a>>op>>b;
    switch(op){
        case '+':result=a+b;break;
        case '-':result=a-b;break;
        case '*':result=a*b;break;
        default:result=0;
    }
    return result;
}
int main(){
    cout<<calString(writeString(3,4,'+'));
    cout<<calString(writeString(3,5,'-'));
    cout<<calString(writeString(3,4,'*'));
    cout<<calString(writeString(4,2,'/'));
    return 0;
}

```