C++作业三

姓名：李畅　　　学号：2012329620003

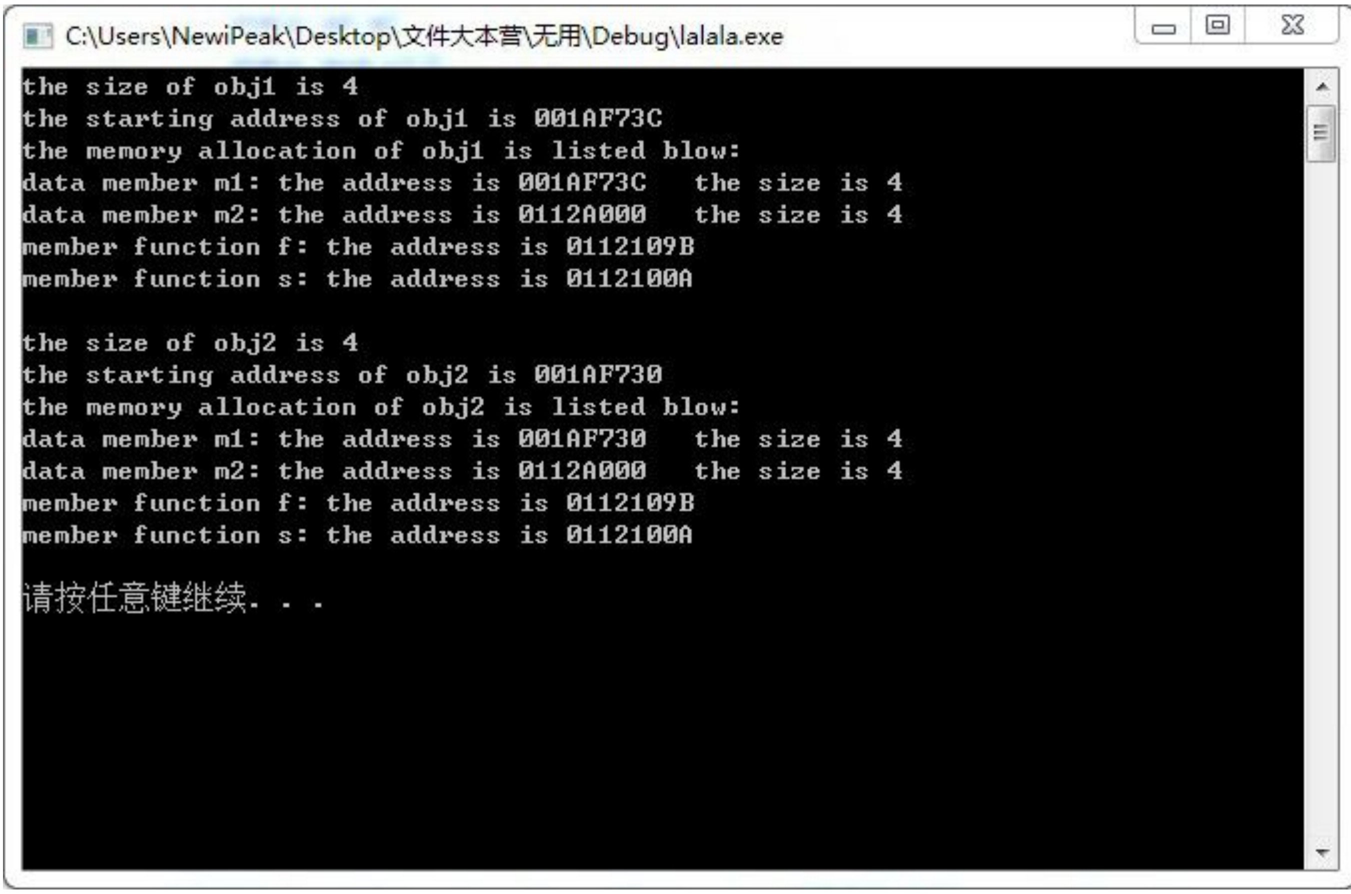
班级：12 级计算机科学与技术 1 班

一、完善上一次关于验证类成员空间分配的程序，增加验证静态成员（包括静态数据成员和静态成员函数）空间分配情况的内容。

```
#include<iostream>
#include<stdlib.h>
using namespace std;
class C{
public:
      int m1;
      void f(){}
       static int m2;
       static void s(){};
};
int C::m2=1;
void main(){
 C obj1,obj2;
 cout<<"the size of obj1 is "<<sizeof(obj1)<<endl; //1
 cout<<"the starting address of obj1 is "<<&obj1<<endl; //2
 cout<<"the memory allocation of obj1 is listed blow:"<<endl; //3
 cout<<"data member m1: the address is "<<&obj1.m1<<"the size is "<<sizeof(obj1.m1)<<endl;
//4
 cout<<"data member m2: the address is "<<&obj1.m2<<"the size is "<<sizeof(obj1.m2)<<endl;
//5
 cout<<"member function f: the address is ";
 printf("%p\n",&C::f);  //6
 cout<<"member function s: the address is ";
 printf("%p\n\n",&C::s);  //7

 cout<<"the size of obj2 is "<<sizeof(obj2)<<endl; //8
 cout<<"the starting address of obj2 is "<<&obj2<<endl; //9
 cout<<"the memory allocation of obj2 is listed blow:"<<endl;  //10
 cout<<"data member m1: the address is "<<&obj2.m1<<"the size is "<<sizeof(obj2.m1)<<endl;
//11
 cout<<"data member m2: the address is "<<&obj2.m2<<" the size is "<<sizeof(obj2.m2)<<endl;
//12
```

```
 cout<<"member function f: the address is ";
 printf("%p\n",&C::f);  //13
 cout<<"member function s: the address is ";
 printf("%p\n\n",&C::s); //14
 system("pause");
}
```



C:\Users\NewiPeak\Desktop\文件大本营\无用\Debug\lalala.exe

```
the size of obj1 is 4
the starting address of obj1 is 001AF73C
the memory allocation of obj1 is listed blow:
data member m1: the address is 001AF73C   the size is 4
data member m2: the address is 0112A000   the size is 4
member function f: the address is 0112109B
member function s: the address is 0112100A

the size of obj2 is 4
the starting address of obj2 is 001AF730
the memory allocation of obj2 is listed blow:
data member m1: the address is 001AF730   the size is 4
data member m2: the address is 0112A000   the size is 4
member function f: the address is 0112109B
member function s: the address is 0112100A

请按任意键继续. . .
```

1、从语句1和语句8的执行结果可以看出，同一个类的不同对象所占内存空间大小是一样的，为其数据成员所占空间的总和，而成员函数在对象中是不占用任何空间的。
2、从语句2和语句4以及语句8和语句10的执行结果可以看出，对象的地址即是类中首个数据成员的地址。
3、从语句4和语句5及语句11和语句12的执行结果可以看出，数据成员在内存中的存储顺序是按其在类中声明顺序分配的。
4、从语句6、7和语句13、14以及执行结果可以看出，同一个类的不同对象的成员函数所占空间是独立于对象之外的，而且函数地址是相同的，也就是这些不同对象的成员函数代码空间是共享的，是属于这个类的，而不是某一个对象的。

二、用C++实现单件模式，即设计一个类，该类仅允许被实例化

一次

```
#ifndef __SINGLETON__
#define __SINGLETON__
class Singleton {
    public:
        static Singleton *getInstance();
    private:
        Singleton();
        Singleton(const Singleton &s);
        void operator=(const Singleton &rhs);
        static Singleton *instance;
};

#endif
```

三、分析下面程序中的错误

```
class X{
private:
int a=0;
int &b;
const int c;
void setA(int i){a=i;}
X (int i){a=i;}
public:
int X(){a=b=c=0;}
X(int i,int j,int k){a=i;b=j;c=k;}
static void setB(int k){b=k;}
set C(int k) const{c=c+k;}
};
void main(){
X x1;
X x2(3);
X x3(1,2,3);
x1.setA(3);
}
```

错误：

1、第三行中只有静态常量整型数据成员才可以在类中初始化

2、第七行中必须在构造函数基/成员初始值设定项列表中初始化

3、第十一行中静态成员函数不能访问非静态成员

4、第十二行中常量成员函数不可以修改数据成员的值

四、读程序，写出程序运行结果

（1）、

```
#include<iostream.h>
#include<string.h>
class X{
int a;char *b; float c;
public:
    X(int x1,char *x2,float x3):a(x1),c(x3){
    b=new char[sizeof(x2)+1];
    strcpy(b,x2);
    }
    X():a(0),b("X::X()"),c(10){}
    X(int x1,char *x2="X::X(...)",int x3=10):a(x1),b(x2),c(x3){}
    X(const X&other){
        a=other.a;
        b="X::X(const X &other)";
        c=other.c;}
    void print(){
    cout<<"a="<<a<<"\t"<<"b="<<b<<"\t"<<"c="<<c<<endl;
    }
};
void main(){
    X *A=new X(4,"X::X(int,char,float)",32);
    X B,C(10),D(B);
    A->print();
    B.print(); C.print(); D.print();
}
```

结果：

```
a=4     b=X::X(int,char,float)        c=32
a=0     b=X::X()    c=10
a=10    b=X::X(…)       c=10
a=0     b=X::X(const X &other)    c=10
```

（2）、

```
#include<iostream>
using std::cout;
using std::endl;
class Implementation{
public:
    Implementation(int v){value=v;}
    void setValue(int v){value=v;}
    int getValue() const{return value;}
private:
    int value;};
    class Interface{
    public:
        Interface(int);
        void setValue(int);
        int getValue() const;
    private:
        Implementation *ptr;
    };
    Interface::Interface(int v):ptr(new Implementation(v)){}
    void Interface::setValue(int v){ptr->setValue(v);}
    int Interface::getValue() const{return ptr->getValue();}
    void main(){
    Interface i(5);
    cout<<i.getValue()<<endl;
    i.setValue(10);
    cout<<i.getValue()<<endl;
    system("PAUSE");
    }
```

结果：5
　　　10

（3）、

```
#include<iostream>
using namespace std;
class A{
int x;
public:
    A():x(0){cout<<"constructor A() called..."<<endl;}
    A(int i):x(i){cout<<"X"<<x<<"\tconstructor..."<<endl;}
    ~A(){cout<<"X"<<x<<"\tdestructor..."<<endl;}
};
class B{
    int y;
    A X1;
    A X2[3];
public:
    B(int j):X1(j),y(j){cout<<"B"<<j<<"\tdestructor..."<<endl;}
    ~B(){cout<<"B"<<y<<"\tdestructor..."<<endl;}
};
void main(){
    A X1(1),X2(2);
    B B1(3);
system("PAUSE");
}
```

结果:

```
X1  constructor…
X2  constructor…
X3  constructor…
constructor A()called…
constructor A()called…
constructor A()called…
B3  destructor…
```

五、设计一个整型链表类list,能够实现链表节点的插入（insert)、删除（delete）,以及链表数据的输出操作（print)。

```
class Node
{
    int a[30];
    Node * Next;
public:
    Node();
    void SetText(int*);
    void SetNext(Node*);
    Node* GetNext();
    int* GetText();
};
class LIST
{
private:
    Node *Handle;
public:
    LIST();
    ~LIST();
    void Insert(Node*);
    void Delete(Node*);
    void print();

};

#include "LIST.h"
#include <iostream>
using namespace std;
Node::Node()
{
    Next=NULL;
}
void Node::SetNext(Node*pN)
{
    Next=pN;
}
void Node::SetText(int* text)
{
    strcpy_s(a,text);
}
int* Node::GetText()
{
    return a;
}
Node* Node::GetNext()
{
    return Next;
}
LIST::LIST()
{
    Handle=NULL;
}
LIST::~LIST()
{
    Node* pN,*pCurrent;
    for(pCurrent=Handle;pCurrent!=NULL;pCurrent=pN)
    {
        pN=pCurrent->GetNext();
        delete pCurrent;
    }
}
void LIST::Insert(Node* toAdd)
{
    Node* pN=new Node();
    pN->SetNext(NULL);
    pN->SetText(toAdd);
    if(Handle==NULL)
        Handle=pN;
    else
    {
        Node* pCurrent=Handle;
        while(pCurrent->GetNext()!=NULL)
            pCurrent=pCurrent->GetNext();
        pCurrent->SetNext(pN);
    }
}


void LIST::print()
{
    Node* pN=Handle;
    while(pN!=NULL)
    {
        cout<<pN->GetText()<<endl;
        pN=pN->GetNext();
    }
}
```