

实验十四 模板

一、 问题描述

1. 实验目的：

掌握模板的若干基本概念和特性，并能够应用于程序编写。

掌握验证性实验的基本方法和过程(认知、实验、总结)。

2. 实验内容：

分别编写一段测试代码来回答任务书中的相关问题（每一个问题，用一个 工程文件，同时需要记录相应的调试过程），具体问题请参考“实验任务 说明14.doc”；

调试的过程：（动态调试的相关截图，比如 设置断点、查看当前变量值等）；

编译出来的可执行程序单独放在一个目录下（bin/exe/debug目录下，同时附上输入数据说明和输出结果）

二、 实验过程

一、选择题

1、类模板的模板参数（ D ）。

- A 可作为数据成员的类型 B 可作为成员函数的返回类型
C 可作为成员函数的参数类型 D 以上三者均可

2、有模板类：

```
template <class T,int size=5>
```

```
class apple {    ....    };
```

现要定义类模板apple的成员函数，其正确的格式为(A)

- A **T apple<T, size>::Push(T object)**
B T apple::Push(T object)
C template< class T,int size=5>
T apple<T,size>::Push(T object)
D template< class T,int size=5>
T apple::Push(T object)

二、简答题

1、请描述：函数模板和模板函数、类模板和模板类

答：（1）函数模板是对一批模样相同的函数的说明描述，它不是某一个具体的函数。

（2）模板函数则是将函数模板内的“数据类型参数”具体化后得到的重载函数（就是由模板而来的函数）。

（3）类模板与函数模板类似，将数据类型定义为参数。类模板具体化为模板类后，

可以用于生成具体对象。模板类的成员函数必须是函数模板。

(4) 说明了一个类模板之后，可以创建类模板的实例，称为模板类。

2、请描述：泛型程序设计思想

答：泛型程序设计，简单地说就是使用模板的程序设计法。泛型程序设计的思想是模板机制，以及标准模板库STL。理想的代码通用，能使得代码不受数据类型的影响，并且可以自动使用数据类型的变化。这就是参数多态，C++的模板为泛型程序设计奠定了关键的基础。

3、比较：函数重载和函数模板

答：

(1) ① 函数重载即定义函数名相同而形参列表（形参个数或形参类别）不同的多个函数，这些函数被称为重载函数，重载函数通常执行的操作非常类似，如打印不同的输入对象。调用函数时编译器根据实参的类型确定调用哪个重载函数。② 函数模板就是创建一类实现逻辑（函数体）一样只是用到的参数类型不同的函数的公式，可用来生成针对特定类型的函数版本。调用函数模板时，编译器（通常）用函数实参来推断模板实参。

(2) 区别

① 用处：函数重载用于定义功能相似的同名函数，提高函数的易用性；函数模板则用于为实现逻辑一样只是参数类型不同的一类函数提供统一的模板，提高函数编写的效率。② 形参列表：函数重载要求参数个数或类型不同；函数模板则要求参数个数必须一样。

(3) 联系：函数模板也可以进行重载。

4、C++标准库与STL的关系

答：STL是最新的C++标准函数库中的一个子集，占据了整个库的大约80%。在C++标准函数库中，STL主要包含容器、算法、迭代器。STL是C++标准程序库的核心；STL是泛型程序库，利用先进、高效的算法来管理数据；STL是所有C++编译器和所有操作系统平台都支持的一种库。

二、代码阅读，修正

1、分析一下程序中出现的错误，分析结果。理解模板类的使用

```
#include <iostream.h>
```

```
template <class T>
```

```
T min(T a,T b)
```

```
{ if (a<b)
```

```
    return a;
```

```
    else
```

```

return b;
}
void main() {
    int i1=10,i2=20;
    double d1=3.5,d2=1.2;
    char c1='b',c2='x';
    cout<<min(i1,i2)<<endl;
    cout<<min(d1,d2)<<endl;
    cout<<min(c1,c2)<<endl;
    cout<<min(i1,c2)<<endl;
    cout<<min(i1,d2)<<endl;
}

```

答：由于i1是int型，c2是char型，如果调用mymin将会出现没有与参数列表匹配的函数模板实例的报错。

```

cout << mymin(i1, i2) << endl;
cout << mymin(d1, d2) << endl;
cout << mymin(c1, c2) << endl;
cout << mymin(i1, c2) << endl;
cout << mymin(i1, d2) << endl;

```

2、阅读并修改程序，分析运行过程和结果。重点分析：类模板和模板类之间的转换

```
#include <iostream.h>
```

```
template <class T>
```

```
class Myclass {
```

```
    private:
```

```
        T n;
```

```
    public:
```

```
        Myclass(T a);
```

```
        T getn ();
```

```
        void setn(T b);
```

```
};
```

```
Myclass::Myclass(T a) {
```

```

    n=a;
}
T Myclass::getn() {
    return n;
}
void Myclass::setn(T b) {
    n=b;
}

void main() {
    Myclass <T> obj1(10);
    cout<<"n="<<obj1.getn()<<endl;
    obj1.setn(20);
    cout<<"n="<<obj1.getn()<<endl;
    Myclass <T> obj2('a');
    cout<<"n="<<obj2.getn()<<endl;
    obj2.setn('b');
    cout<<"n="<<obj2.getn()<<endl;
}

```

答：在说明了一个类模板之后，可以创建类模板的实例，称为模板类。定义模板类的对象的格式应该为：类模板名 <实际类型> 对象名(实参表)；而成员函数的定义应该为：**template<模板形参表>** 返回值类型 类模板名<类型名表>::成员函数名(参数表) { 成员函数体 }。修改代码：

```

#include <iostream>
using namespace std; //类模板和模板类之间的转换
template <class T>
class Myclass {
private:
    T n;
public:
    Myclass(T a);
    T getn();
    void setn(T b);
};

template<class T>
Myclass<T>::Myclass(T a) {
    n = a;
}

template<class T>
T Myclass<T>::getn() { return n;}
template<class T>
void Myclass<T>::setn(T b) { n = b;}

void main() {
    Myclass <int> obj1(10);
    cout << "n = " << obj1.getn() << endl;
    obj1.setn(20);
    cout << "n = " << obj1.getn() << endl;
    Myclass <char> obj2('a');
    cout << "n = " << obj2.getn() << endl;
    obj2.setn('b');
    cout << "n = " << obj2.getn() << endl;
}

```

Microsoft Visual Studio 调试控制台

```

n = 10
n = 20
n = a
n = b

```

3、填写代码，分析代码执行过程和运行结果

```

#include <iostream>
using namespace std;
class Sample {
    T n;
public:
    Sample() { }
    Sample(T i) {n=i;}
    Sample <T> & operator+ (const Sample<T> &);
    void disp() {
        cout<<"n="<<n<<endl;
    }
}

```

```

Sample<T> &Sample<T>::operator+(const Sample<T> & s)    {
static Sample<T> temp;

    temp.n=n+s.n;

    return temp;
}

int main( ) {
    Sample<int> s1(10),s2(20),s3;
    s3=s1+s2;
    s3.disp( );
    return 0;
}

```

答：（1）添加运算符重载：Sample<T> & operator+ (const Sample<T> &);的原因。

Sample类的对象s1和s2不能直接通过原本的+运算符实现相加，故对运算符进行重载。将s1对象和s2对象的成员变量n进行相加。返回相加结果，即另一个Samlpe对象的引用。最后输出的结果是n=30。

（2）static Sample<T> temp; 语句功能：这里的static的功能是表明该变量的值不会因为函数终止而丢失。

附：

static关键字的作用：c/c++共有（1）修饰全局变量时，表明一个全局变量只对定义在同一文件中的函数可见。（2）修饰局部变量时，表明该变量的值不会因为函数终止而丢失。（3）修饰函数时，表明该函数只在同一文件中调用。

c++独有：（1）：修饰类的数据成员，表明对该类所有对象这个数据成员都只有一个实例。即该实例归所有对象共有。（2）用static修饰不访问非静态数据成员类成员函数。这意味着一个静态成员函数只能访问它的参数、类的静态数据成员和全局变量。

```

#include <iostream>
using namespace std;

template<class T>
class Sample {
    T n;
public:
    Sample() { }
    Sample(T i) { n = i; }
    Sample<T>& operator+ (const Sample<T>&);
    void disp() {
        cout << "n = " << n << endl;
    }
};

template<class T>
Sample<T>& Sample<T>::operator+(const Sample<T>& s) {
    static Sample<T> temp;
    temp.n = n + s.n;
    return temp;
}

int main() {
    Sample<int> s1(10), s2(20), s3;
    s3 = s1 + s2;
    s3.disp();
    return 0;
}

```

 Microsoft Visual Studio 调试控制台

```
n = 30
```

三、代码题

- 1、设计一个函数模板，找出一个任意类型数组中的最大者和最小者。

```

#include <iostream>
using namespace std;
template<typename T1> //模板声明, T1为类型参数
T1 max(T1* p1, T1 n) //定义模板函数max, 求最大值
{
    int j = 0;
    for (int i = 1; i < n; i++)
        if (p1[i] > p1[j]) j = i;
    return p1[j];
}

template<typename T2>
T2 min(T2* p2, T2 m) //定义模板函数min, 求最小值
{
    int j = 1;
    for (int i = 0; i < m; i++)
        if (p2[i] < p2[j]) j = i;
    return p2[j];
}

int main()
{
    int a[] = { 5, 6, 4, 58, 2 };
    double b[] = { 2.3, 1.1, 2.2, 3.3, 88.01 };
    cout << "数组a为: " << endl;
    for (int i = 0; i < 5; i++) cout << a[i] << " "; cout << endl;
    cout << "数组a中最大值为: " << max(a, 5) << endl;
    cout << "数组a中最小值为: " << min(a, 5) << endl;
    cout << "数组b为: " << endl;
    for (int j = 0; j < 5; j++) cout << b[j] << " "; cout << endl;
    cout << "数组b中最大值为: " << max(b, 5.0) << endl;
    cout << "数组b中最小值为: " << min(b, 5.0) << endl;
    return 0;
}

```

Microsoft Visual Studio 调试控制台

```

数组a为:
5 6 4 58 2
数组a中最大值为: 58
数组a中最小值为: 2
数组b为:
2.3 1.1 2.2 3.3 88.01
数组b中最大值为: 88.01
数组b中最小值为: 1.1

```

2、设计一个类模板，可用于T类型的数组，且定义成员函数：

(1) 求所有元素的和

(2) 查找指定元素是否存在，如果存在，则返回其所在数组元素的下标值，否则返回-1


```

#include<iostream>
using namespace std;

template <class T>
class Array
{
    T* set;
    int n;
public:
    Array(T* data, int i) { set = data; n = i; }
    void sort(); // 排序
    int seek(T key); // 查找指定的元素
    T sum(); // 求和
    void disp(); // 显示所有的元素
};

template<class T>
void Array<T>::sort()
{
    int i, j;
    T temp;
    for (i = 1; i < n; i++)
        for (j = n - 1; j >= i; j--)
            if (set[j - 1] > set[j]) {
                temp = set[j - 1]; set[j - 1] = set[j]; set[j] = temp;
            }
}

template <class T>
int Array<T>::seek(T key)
{
    int i;
    for (i = 0; i < n; i++)
        if (set[i] == key) return i;
    return -1;
}

template<class T>
T Array<T>::sum()
{
    T s = 0; int i;
    for (i = 0; i < n; i++) s += set[i];
    return s;
}

template<class T>
void Array<T>::disp()
{
    int i;
    for (i = 0; i < n; i++) cout << set[i] << " ";
    cout << endl;
}

```

```

void main()
{
    int a[] = { 6, 3, 8, 1, 9, 4, 7, 5, 2 };
    double b[] = { 2.3, 6.1, 1.5, 8.4, 6.7, 3.8 };
    Array<int>arr1(a, 9);
    Array<double>arr2(b, 6);
    cout << " arr1:" << endl;
    cout << " 原序列:"; arr1.disp();
    cout << " 8在arr1中的位置:" << arr1.seek(8) << endl;
    arr1.sort();
    cout << " 排序后:"; arr1.disp();
    cout << " arr2:" << endl;
    cout << " 原序列:"; arr2.disp();
    cout << " 8.4在arr2中的位置:" << arr2.seek(8.4) << endl;
    arr2.sort();
    cout << " 排序后:"; arr2.disp();
}

```

Microsoft Visual Studio 调试控制台

```

arr1:
原序列:6 3 8 1 9 4 7 5 2
8在arr1中的位置:2
排序后:1 2 3 4 5 6 7 8 9
arr2:
原序列:2.3 6.1 1.5 8.4 6.7 3.8
8.4在arr2中的位置:3
排序后:1.5 2.3 3.8 6.1 6.7 8.4

```

3、代码填空：设计一个能存储任意类型数据的顺序堆栈，即设计一个堆栈模板，然后用某种类型的数据进行验证，并进行进栈和出栈操作。

背景知识：堆栈是一种存储结构，它的特点是先进后出结构，堆栈中有一个栈顶指针，始终指向栈顶上的元素。程序中，建立的一个顺序栈，采用数组。

```
#include <iostream>
```

```
#define MAXSIZE 50
```

```
using namespace std;
```

```
template <class T> {
```

```
private:
```

```
    T s[MAXSIZE];
```

```
    int top;
```

```
public:
```

```
    stack() {top=-1;}

```

```
    void push(T newvalue); //进栈

```

```
T pop(); //出栈

```

```
};
```

```
void stack<T>::push( T newvalue) {  
    if ( )  
        { top=top+1;  
          s[top]=newvlaue;  
        }  
    else  
        cout<<"栈满，无法进栈！"<<endl;  
}
```

```
template <class T>  
T      ::pop() {  
    if (top>=1) {  
        cout<<s[top]<<endl;  
        top=top-1;  
    }  
    else  
        cout<<"堆栈已空!无法出栈！"<<endl;  
}
```

```
void main( ) {  
    stack<      > args;  
    args.push(10);  
    args.push(20);  
    args.pop( );  
    args.pop( );  
    args.pop( );  
}
```

请回答问题：

```

#include <iostream>
#define MAXSIZE 50
using namespace std;


template <class T>
class stack{
private:
    T s[MAXSIZE];
    int top;
public:
    stack() { top = -1; }
    void push(T newvalue); //进栈
    T pop(); //出栈
};

template <class T>
void stack<T>::push(T newvalue) {
    if (top<MAXSIZE-1) {
        top = top + 1;
        s[top] = newvalue;
    }
    else cout << "栈满, 无法进栈!" << endl;
}

template <class T>
T stack<T>::pop() {
    if (top > -1) {
        cout << s[top] << endl;
        top = top - 1;
    }
    else cout << "堆栈已空!无法出栈!" << endl;
    return s[top];
}

int main() {
    stack<int> args;
    args.push(10);
    args.push(20);
    args.pop();
    args.pop();
    args.pop();
}

```

 Microsoft Visual Studio 调试控制台

```

20
10
堆栈已空!无法出栈!

```

(1) 为何要定义栈？

答：为了方便内存/数据的管理。堆和栈都只是一种抽象结构，较为高级的语言运行需要栈，栈用来存储局部变量，返回地址以及参数，堆用来存储动态分配的数据。

(2) 栈的代码实现是依赖数组，是如何实现数组“先进后出”特性的？

答：我们把允许插入和删除的一端称为栈顶，另一端称为栈底，栈又称为后进先出的线性表，简称LIFO（Last In First Out）结构。元素从栈顶进入，先进入的元素出于底端，后进入的元素处于顶端。而取数据的时候同样从栈顶取出，这样实现了

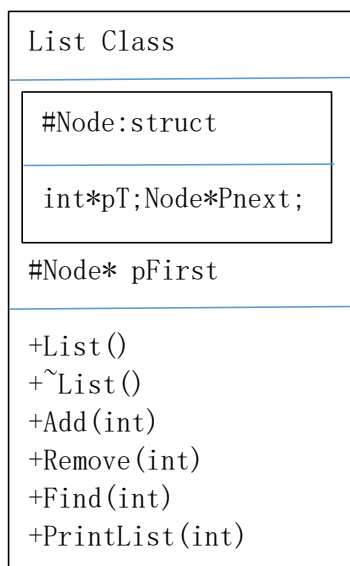
元素的先进后出。

4、以下是一个整数链表类的定义

```
const int maxqueue = 10;
class List {
public:
    List();
    ~List();
    void Add(int);
    void Remove(int);
    int* Find(int);
    void PrintList();
protected:
    struct Node {
        Node* Pnext;
        int* pT;
    };
    Node* pFirst;
};
```

(1) 请图形表示：类描述下的链表形式；并与C++版的链表描述做对比。

答：C++ STL 库的 list 容器是一个双向链表。包含在头文件 <list> 中。而类描述下的链表形式如下图：



(2) 编写一个链表的类模板（包括其成员函数定义），让任何类型的对象提供链表结构数据操作；

```

#include<iostream>
using namespace std;

template<class Type>
class Node {
public:
    Type Data;
    Node* next;
};

template<class Type> <T>
class List{
private:
    Node<Type>* head;
    int Length;
public:
    void Get() {cout << head->next->Data << endl;}
    int length() { return Length; }

    List() {
        head = new Node<Type>;
        head->Data = (Type) 0;
        head->next = NULL;
        Length = 1;
    }
}

```

```
int Append(const Type& t) {
    Node<Type>* p, * q;
    p = head;
    while (p->next != NULL) {
        p = p->next;
    }
    q = new Node<Type>;
    q->Data = t;
    q->next = NULL;
    p->next = q;
    Length++;
    return Length;
}

int Insert(const Type& t, int i) {
    Node<Type>* p, * q;
    p = head;
    for (int loc = 0; loc < i ; loc++) {
        p = p->next;
    }
    q = new Node<Type>;
    q->Data = t;
    q->next = p->next;
    p->next = q;
    Length++;
    return Length;
}
```

```

int Search(Type& t) {
    Node<Type>* p;
    p = head;
    int i = 0;
    while (p != NULL) {
        i++;
        if (p->Data == t) return i-1;
        p = p->next;
    }
    return 0;
}

void Delete(int t) {
    Node<Type>* p;
    p = head;
    if (t != 1) {
        for (int j = 0; j < t-1; j++) p = p->next;
        p->next = p->next->next;
    }
    else {
        head = p->next;
    }
}

Type& operator [ ] (int i) {
    Node<Type>* p;
    p = head;
    for (int j = 0; j < i; j++) p = p->next;
    return p->Data;
}

void Display() {
    Node<Type>* p;
    p = head->next;
    while (p != NULL) {
        cout << p->Data << " ";
        p = p->next;
    }
    cout << endl;
}
};


```

(3) 在应用程序中创建整数链表、字符链表和浮点数链表，并提供若干数据能插入链表；在链表中删除一个节点和打印链表中所有节点元素，遍历整个链表查找给定对应节点等操作。


```

int main() {
    //整数链表
    List<int> L;
    int n=70;
    L.Append(70);L.Append(3);L.Append(5);
    L.Append(1);L.Append(8);
    L.Insert(55, 3);
    L.Delete(2);
    L.Display(); //链表头赋值为-1，方便排序的进行，也更方便数据的存取
    cout<< L.Search(n) <<endl;
    cout << L[3]<<endl;
    //字符链表
    List<char> Lc;
    Lc.Append('a'); Lc.Append('b'); Lc.Append('c');
    Lc.Insert('d', 3);
    Lc.Delete(3);
    Lc.Display();
    //浮点数链表
    List<double> Ld;
    Ld.Append(1.1); Ld.Append(2.2); Ld.Append(3.3);
    Ld.Insert(4.4, 3);
    Ld.Delete(4);
    Ld.Display();
}

```



```

Microsoft Visual Studio 调试控制台
70 5 55 1 8
1
55
a b d
1.1 2.2 3.3

```

三、 附录

源程序文件项目清单：src/homework