第九章 泛型程序设计与模板

一、函数模板

(1)函数模板的原理

```
int max(int a,int b){return (a>b)?a:b;}
float max(float a,float b){return (a>b)?a:b;}
char max(char a,char b){return (a>b)?a:b;}
```

以上程序处理类型一模一样，引入函数模板

1.类型作为参数，有三个关键字：template， typename，class

2.定义：函数模板用于生成函数

```
template<class 类型参数1，class类型参数2,...>
返回值类型  模板名（形参表）
{
    函数体
}
//class关键字也可以用typename代替

//swap模板
template<class T>
void swap(T & x,T & y)
{
   T tmp=x;
   x=y;
   y=tmp;
}

//函数模板（比较大小）
template<class T>
T max(T a,T b)
{
   return(a>b)?a:b;
}
```

```
//模板函数
int ival=max(100,99);
char ival=max<char>('A','B');

//swap模板(用typename来定义)
template<typename T>
void swap(T & x,T & y)
{
   T tmp=x;
   x=y;
   y=tmp;
}

//模板函数
int x=20,y=30;
swap<int>(x,y);
```

3.变量作为模板参数

```
template<int size>
void display()
{
   cout<<size<<endl;
}

display<10>();
```

4.多参数函数模板

```
template<typename T,typename C>
void display(T a,C b)
{
   cout<<a<<""<<b<<endl;
}

int a=1024;
string str="hello world!";
disply<int,string>(a,str);
```

5.typename和class可以混用

```
template<typename T,class U>
T minus(T *a,U b);

template<typename T,int size)
void display(T a)
{
```

```
    for(int i=0;i<size;i++)
    cout<<a<<endl;
  }
  dispaly<int,5>(15);
```

(2)函数模板与重载

```
  template<typename T>
  void display(T a);//只有一个参数
  template<typename T>
  void display(T a,T b);//有两个参数，个数不同
  template<typename T,int size>
  void display(T a);//一个参数，一个变量
```

实例化为：

```
  dispaly<int>(10);
  dispaly<int>(10, 20);
  dispaly<int,5>(30);//在定义出函数模板的时候，需要注意，函数模板本身并不是相互重载的关系，因
```

(3)函数模板的编码实现

```
#include <iostream>
#include <stdlib.h>
using namespace std;
//函数模板，要求定义函数模板display
template<typename T>
void display(T a)
{
  cout<<a<<endl;
}
template<typename T,class S>
void display(T t,S s)
{
  cout<<t<<endl;
  cout<<s<<endl;
}
template<typename T,int KSize>
void display(T a)
{
  for(int i=0;i<KSize;i++)
  {
    cout<<a<<endl;
  }
}
int main()
```

```
{
  //display<double>(10.89);
  //display<int,double>(5,8.3);
  display<int,5>(6);//输出五个6
  return 0;
}
```

二、类模板

(1)类模板的原理

```
template<class T>
class MyArray
{
   public:
      void display()
      {......}
   private:
      T *m_pArr;
};
```

类模板的成员函数放到类模板定义外面写时：

```
template<class T>
void MyArray<T>::display()
{
   ......
};

int main()
{
   MyArray<int>arr;
   arr.display();
   return 0;
}
```

类模板中有多个参数时

```
template<typename T>
class Container
{
   public:
      void display;
   private:
      T m_obj;
};
```

类外定义时：

```
template<typename T, int KSize>
void Container<T,KSize>::display()
{
   for(int i=0;i<KSize;i++)
   {
       cout<<m_obj<<endl;
   }
}
int main()
{
  Container<int,10>ct1;
  ct1.display();
  return 0;
}
```

特别提醒：因为IDE环境问题，以及其他问题，模板代码不能分离编译

(2)类模板的编码实现

```
#include<iostream>
#include<string>
using namespace std;
//类模板，定义类模板MyArray,成员函数：构造函数，析构函数，display函数
                        //数据成员：m_pArr
template<typename T,int KSize,int KVal>
class MyArray
{
public:
  MyArray();
  ~MyArray()
  {
    delete[]m_pArr;
    m_pArr=NULL;
  }
  void display();
private:
  T *m_pArr;//模板参数T
};
template<typename T,int KSize,int KVal>
MyArray<T,KSize,KVal>::MyArray()
{
  m_pArr=new T[KSize];
  for(int i=0;i<KSize;i++)
  {
```

```
    m_pArr[i]=KVal;
  }
}
template<typename T,int KSize,int KVal>
void MyArray<T,KSize,KVal>::display()
{
  for(int i=0;i<KSize;i++)
  {
    cout<<m_pArr[i]<<endl;
  }
}
int main()
{
  MyArray<int,5,6>arr;//一个int型数组，数组长度为5，数组元素都是6，有5个6的数组
  arr.display();
  return 0;
}
```

练习题：定义一个矩形类模板，该模板中含有计算矩形面积和周长的成员函数，数据成员为矩形的长和宽。

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

/**
 * 定义一个矩形类模板Rect
 * 成员函数：calcArea()、calePerimeter()
 * 数据成员：m_length、m_height
 */
template <typename T>
class Rect
{
public:
   Rect(T length, T height);
   T calcArea();//面积
   T calePerimeter();//周长
public:
        T m_length;//长
        T m_height;//宽
};

/**
 * 类属性赋值
 */
template <typename T>
Rect<T>::Rect(T length, T height)
```

```
{
        m_length = length;
        m_height = height;
}

/**
 * 面积方法实现
 */
template <typename T>
T Rect<T>::calcArea()
{
        return m_length * m_height;
}

/**
 * 周长方法实现
 */
template <typename T>
T Rect<T>::calePerimeter()
{
        return ( m_length + m_height) * 2;
}

int main(void)
{
        Rect<int> rect(3, 6);
        cout << rect.calcArea() << endl;
        cout << rect.calePerimeter() << endl;
        return 0;
}
```