1. 函数模板
   1. 占位符T

template <class T>

void foo(T param)

{

//do something

}

注意：

1. template <class T> 和 template <typename T>是一样的
2. swap<int>(a ,b)明确告诉编译器它应该使用哪种类型
3. 多种占位符

template <class T,class U>

class MyClass

{

//.........

}

实例化时：MyClass<int,float> myClass;

1. 类模板

template <class T>

class MyClass

{

MyClass();

void swap(T &a,T &b);

}

1. 构造器

MyClass<T>::MyClass()

{

//初始化操作

}

#include <iostream>

#include <string>

template <class T>

class Stack

{

public:

Stack(unsigned int size = 100);

~Stack();

void push(T value);

T pop();

private:

unsigned int size;

unsigned int sp;

T \*data;

};

template <class T>

Stack<T>::Stack(unsigned int size)

{

this->size = size;

data = new T[size];

sp = 0;

}

template <class T>

Stack<T>::~Stack()

{

delete []data;

}

template <class T>

void Stack<T>::push(T value)

{

data[sp++] = value;

}

template <class T>

T Stack<T>::pop()

{

return data[--sp];

}

int main()

{

Stack<int> intStack(100);

intStack.push(1);

intStack.push(2);

intStack.push(3);

std::cout<<intStack.pop()<<"\n";

std::cout<<intStack.pop()<<"\n";

std::cout<<intStack.pop()<<"\n";

return 0;

}

1. 内联模板
2. 内联函数

inline int add(int x,int y,int z)

{

return x+y+z;

}

在程序中，调用其函数的时候，该函数在编译时被替代，而不像一般函数那样是在运行时被调用。

1. 内联方法

class Person

{

Person(std::string name)

{

this->name = name;

}

//......

}

#include <iostream>

#include <string>

template <class T>

class Stack

{

public:

Stack(unsigned int size = 100)

{

this->size = size;

data = new T[size];

sp = 0;

}

~Stack()

{

delete []data;

}

void push(T value)

{

data[sp++] = value;

}

T pop()

{

return data[--sp];

}

private:

unsigned int size;

unsigned int sp;

T \*data;

};

int main()

{

Stack<int> intStack(100);

intStack.push(1);

intStack.push(2);

intStack.push(3);

std::cout<<intStack.pop()<<"\n";

std::cout<<intStack.pop()<<"\n";

std::cout<<intStack.pop()<<"\n";

return 0;

}

注意：

1. 如果打算在自己的程序里面是有Stack模板，一定要给它增加副本构造器和一个赋值操作符重载