Section 4:Lecture 11

Introduction

TempFunction Templates, Overloading Template Functions, Class Template, Class Templates and Non-Type Parameters, Templates and Inheritance, Templates and Friends, Templates and Static Members.late Parameter Defaults Template Class Method Definition

Template class

Classes (and structs) can be templated:

```
template <class T>
class Array
{
public:
…
protected:
T mArray[MAX_SIZE];
};
Array myArray; // Compiler ERROR
Array<int> myIntArray; // array of int's
Array<void *> myPtrArray; // array of ptr's
```

Class templates with multiple parameter

```
Template <class T1 ,class T2>
Class test
{
T1 a;
T2 b;
Public:
Test(TI x, T2 y)
{
a= x;
b=y;
}

Void show(){  cout<<a;}
};
```

```
Void main()
{
Test<float,int>test1(100,'w');
Test<float,int>test2(1.23,123);

 test1.show();
Test2.show();
}
```

Template Parameter

Template parameters can be non-types:

```
template <class T, int maxSize>
class Array
{
public:
…
protected:
T mArray[maxSize];
};
Array<int, 1024> myIntArray; // 1024 int's
Array<long, 1024> myLongArray1; // 1024 long's
Array<long, 256> myLongArray2; // 256 long's
```

Template Parameter Defaults

Template parameters can have defaults:

```
template <class T = int, int maxSize = 1024>
class Array
{
public:
…
protected:
T mArray[maxSize];
};
Array<> myIntArray; // 1024 int's
Array<long> myLongArray1; // 1024 long's
Array<long, 256> myLongArray2; // 256 long's
```

Template Class Method Definition

```
// Templated Foo< > declaration
template <class T>
class Foo
{
public:
...
T bar();
};
// Definition of bar() method in Foo<>
template <class T>
T Foo<T>::bar()
{ ... }
```

Templated Methods

Classes can have templated methods:

```
class DebugDialog
{
public:
…
template <class T>
void displayToUser(T input)
{ cout << input << endl; }
};
dd.displayToUser(“Hello, World!”);
dd.displayToUser(theErrorNum);
dd.displayToUser(myInvalidPtr);
```

functionTemplate

- Template class <T>
- void swap(T &x, T &y)

```
{
 T temp= x;
 X=y;
 Y=temp;
}
Void main()
{
Swap(10 20);
}
```