Templates

Function Templates

- Consider the following function

```
int max (int x, int y)
{
      return (x > y? x : y);
}
```

It only works for the `int` type of variables!

How can we make it work for other types of variables?

Function Templates

Use function overloading!

```
float max (float x, float y)
{
      return (x > y? x : y);
}

double max (double x, double y)
{
      return (x > y? x : y);
}

char max (char x, char y)
{
      return (x > y? x : y);
}

 ……
```

Function Templates

Use function overloading!

```
float max (float x, float y)
{
      return (x > y? x : y);
}

double max (double x, double y)
{
      return (x > y? x : y);
}

char max (char x, char y)
{
      return (x > y? x : y);
}

 ……
```

Do we have a smarter way to do that?

An Example Function Template

Indicates a template is being defined

Indicates T is our formal template parameter

```
template <class T>
  T Min(const T &a, const T &b) {
    if (a < b)
        return a;
    else
        return b;
}
```

Instantiated functions will return a value whose type is the actual template parameter

Instantiated functions require two actual parameters of the same type. Their type will be the actual value for T

How Does this Work?

Consider the following example

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

template<typename T>
T abs(T x) {
    return x < 0? -x : x;
}

int main() {
    int n = -5;
    double d = -5.5;
    cout << abs(n) << endl;
    cout << abs(d) << endl;
    return 0;
}
```

If not using template, at least two `abs` functions need to be defined:
One for `int` type, the other for `double` type

Output is
5
5.5

How Does this Work?

Consider the following example

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

template<typename T>
T abs(T x) {
    return x < 0? -x : x;
}

int main() {
    int n = -5;
    double d = -5.5;
    cout << abs(n) << endl;
    cout << abs(d) << endl;
    return 0;
}
```

In **run-time,** compiler determines the actual type for **T** based on the type of the input variable

For instance, `abs(n)`, `n` is `int` type

The compiler then creates the following function based on the template

```
int abs(int x) {
    return x < 0? -x : x;
}
```

Difference Between Template Functions and Normal Functions

- **Compiler won't generate instance for template functions during compiling**. It generates when it is called (see abs example).

- If a template function is used by several other places in different .cpp files, **the template function and its body need to be put in the .h file not only the declaration**.

- The function pointer can only point to instance of the template function.

Class Templates

We can also define template class, which can be considered as the “generalized” classes (i.e., can be used to generate actual classes in run-time).

```
template<class T>
class Pair
{
public:
        Pair();
        Pair(T firstVal, T secondVal);
        void setFirst(T newVal);
        void setSecond(T newVal);
        T getFirst() const;
        T getSecond() const;
private:
        T first;
        T second;
};
```

Two Types of Binding

- Static Binding (the default in C++)
  - `px->print()` uses `X`'s print
  - this is known at compile time

- Dynamic Binding
  - `px->print()` uses the `print()` in the object pointed at
  - this is **only known at run time**
  - coded in C++ with *virtual functions*

Practice!!!!

Predict the output?

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

template <typename T>
void fun(const T&x)
{
   static int count = 0;
   cout << "x = " << x << " count = " << count << endl;
   ++count;
   return;
}

int main()
{
   fun<int> (1);
   cout << endl;
   fun<int>(1);
   cout << endl;
   fun<double>(1.1);
   cout << endl;
   return 0;
}
```

Output of following program? Assume that the size of char is 1 byte and size of int is 4 bytes, and there is no alignment done by the compiler.

```
#include<iostream>
#include<stdlib.h>
using namespace std;

template<class T, class U>
class A  {
    T x;
    U y;
    static int count;
};

int main()  {
   A<char, char> a;
   A<int, int> b;
   cout << sizeof(a) << endl;
   cout << sizeof(b) << endl;
   return 0;
}
```

Output of following program?

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

template <class T>
class Test
{
private:
   T val;
public:
   static int count;
   Test() {   count++;   }
};

template<class T>
int Test<T>::count = 0;

int main()
{
   Test<int> a;
   Test<int> b;
   Test<double> c;
   cout << Test<int>::count   << endl;
   cout << Test<double>::count << endl;
   return 0;
}
```

Write a Program to display largest among two numbers using function templates

```
#include <iostream>
using namespace std;
// template function
template <class T>
T Large(T n1, T n2)
{
return (n1 > n2) ? n1 : n2;
}
int main()
{
int i1, i2;
float f1, f2;
char c1, c2;
cout << "Enter two integers:\n";
cin >> i1 >> i2;
cout << Large(i1, i2) <<" is larger." << endl;
cout << "\nEnter two floating-point numbers:\n";
cin >> f1 >> f2;
cout << Large(f1, f2) <<" is larger." << endl;
cout << "\nEnter two characters:\n";
cin >> c1 >> c2;
cout << Large(c1, c2) << " has larger ASCII value.";
return 0;
}
```

Write a Program to swap data using function templates.

```
#include <iostream>
using namespace std;
template <typename T>
void Swap(T &n1, T &n2)
{
T temp;
temp = n1;
n1 = n2;
n2 = temp;
}
int main()
{
int i1 = 1, i2 = 2;
float f1 = 1.1, f2 = 2.2;
char c1 = 'a', c2 = 'b';
cout << "Before passing data to function template.\n";
cout << "i1 = " << i1 << "\ni2 = " << i2;
cout << "\nf1 = " << f1 << "\nf2 = " << f2;
cout << "\nc1 = " << c1 << "\nc2 = " << c2;
Swap(i1, i2);
Swap(f1, f2);
Swap(c1, c2);
cout << "\n\nAfter passing data to function template.\n";
cout << "i1 = " << i1 << "\ni2 = " << i2;
cout << "\nf1 = " << f1 << "\nf2 = " << f2;
cout << "\nc1 = " << c1 << "\nc2 = " << c2;
return 0;
}
```

Write a Program to add, subtract, multiply and divide two numbers using class template

```
template <class T>
class Calculator
{
private:
T num1, num2;
public:
Calculator(T n1, T n2)
{
num1 = n1;
num2 = n2;
}
void displayResult()
{
cout << "Numbers are: " << num1 << " and " << num2 << "." << endl;
cout << "Addition is: " << add() << endl;
cout << "Subtraction is: " << subtract() << endl;
cout << "Product is: " << multiply() << endl;
cout << "Division is: " << divide() << endl;
}
T add() { return num1 + num2; }
T subtract() { return num1 - num2; }
T multiply() { return num1 * num2; }
T divide() { return num1 / num2; }
};
int main()
{
Calculator<int> intCalc(2, 1);
Calculator<float> floatCalc(2.4, 1.2);
cout << "Int results:" << endl;
intCalc.displayResult();
cout << endl << "Float results:" << endl;
floatCalc.displayResult();
return 0;
}
```