Template

Prof. Seokin Hong

SUNGKYUNKWAN UNIVERSITY

Function Template

swapValues for char

- **Here is a version of swapValues to swap character variables:**

```
void swapValues(char& v1, char& v2)
{
      char temp;
      temp = v1;
      v1 = v2;
      v2 = temp;
}
```

double?

A General swapValues

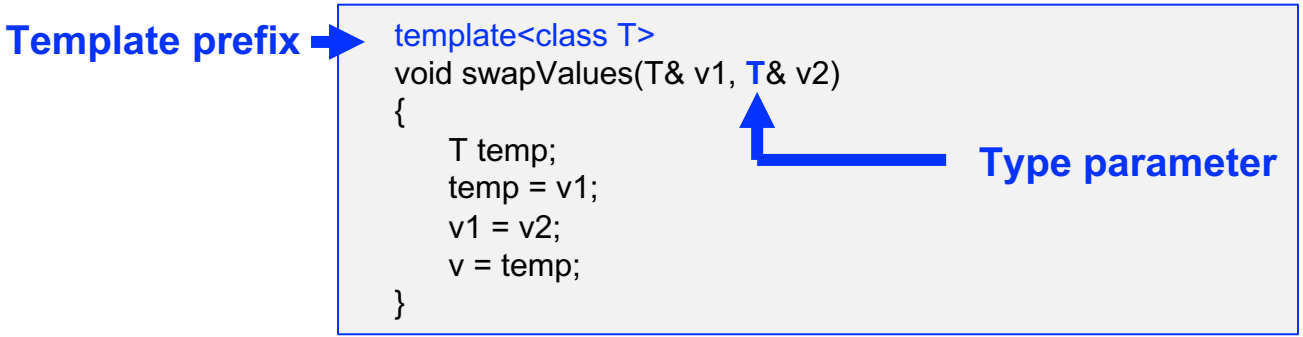
- **A generalized version of swapValues is shown here.**

```
void swapValues(typeOfVar& v1, typeOfVar& v2)
{
      typeOfVar temp;
      temp = v1;
      v1 = v2;
      v2 = temp;
}
```

This function could be used to swap values of any type if typeOfVar could accept any type.

Templates for Functions

- **A C++ function template will allow swapValues to swap values of two variables of the same type**



- **template<class T> is the template prefix**
  - Tells compiler that the declaration or definition that follows is a template
  - Tells compiler that T is a type parameter
    - ***typename*** could replace *class*
    - *T* can be replaced by any type argument

Calling a Template Function

- **Calling a function defined with a template is identical to calling a normal function**
  - Example: To call the template version of swapValues

```
char s1, s2;
int i1, i2;
…
swapValues(s1, s2);
swapValues(i1, i2);
```

- The compiler checks the argument types and generates an appropriate version of swapValues

Type Parameter T

- **T is the traditional name for the type parameter**
  - Any valid, non-keyword, identifier can be used

```
template <typename VariableType>
 void swapValues(VariableType& v1, VariableType& v2)
  {
       VariableType temp;
        …
   }
```

```
template<class T>
void swap_values(T& variable1, T& variable2)
{
      T temp;

      temp = variable1;
      variable1 = variable2;
      variable2 = temp;
}

int main()
{
    int integer1 = 1, integer2 = 2;
    cout << "Original integer values are "
         << integer1 << " " << integer2 << endl;
    swap_values(integer1, integer2);
    cout << "Swapped integer values are "
         << integer1 << " " << integer2 << endl;

    char symbol1 = 'A', symbol2 = 'B';
    cout << "Original character values are "
         << symbol1 << " " << symbol2 << endl;
    swap_values(symbol1, symbol2);
    cout << "Swapped character values are "
         << symbol1 << " " << symbol2 << endl;

    return 0;
}
```

Templates with Multiple Parameters

- **Function templates may use more than one parameter**
  - Example:

```
template<class T1, class T2>
test(T1 a, T2 b)
{
    std::cout<<a<<std::endl;
    std::cout<<b<<std::endl;

}
int a=1; double b=2.2;
test(a, b);
```

Example: A Generic Sorting Function

- **The sort function below uses an algorithm that does not depend on the base type of the array**

```
void sort(int a[], int numberUsed)
{
        int indexOfNextSmallest;
        for (int index = 0; index < numberUsed -1; index++)
        {
            indexOfNextSmallest = indexOfSmallest(a, index, numberUsed);
            swapValues(a[index], a[indexOfNextSmallest]);
        }
}
```

Example: A Generic Sorting Function

```
#include <iostream>

template<class T>
void swap_values(T& variable1, T& variable2)
{
    T temp;

    temp = variable1;
    variable1 = variable2;
    variable2 = temp;
}

template<class BaseType>
int index_of_smallest(const BaseType a[], int start_index, int number_used)
{
    BaseType min = a[start_index];
    int index_of_min = start_index;

    for(int index=start_index+1; index< number_used; index++)
    {
        if(a[index]<min)
        {
            min=a[index];
            index_of_min=index;
        }
    }
    return index_of_min;
}

template<class BaseType>
void sort(BaseType a[], int number_used)
{
    int index_of_next_smallest;
    for(int index=0; index<number_used-1; index++)
    {
        index_of_next_smallest = index_of_smallest(a, index, number_used);
        swap_values(a[index], a[index_of_next_smallest]);
    }
}
```

```
int main()
{
    using namespace std;

    int i;
    int a[10] = {9,8,7,6,5,1,2,3,4,0};
    cout << "Unsorted integers:\n";
    for(i =0; i<10; i++)
        cout << a[i] <<" ";
    cout <<endl;

    sort(a, 10);
    cout<< "In sorted order the integers are:\n";
    for(i=0;i<10;i++)
        cout << a[i] << " ";

    double b[5] = {5.5, 4.4, 1.1, 3.3, 2.2};
    cout<< "Unsorted doubles:\n";
    for(i=0; i< 5; i++)
        cout<<b[i] << " ";
    cout<<endl;

    sort(b,5);
    cout << "In sorted order the doubles are:\n";
    for(i=0;i<5; i++)
        cout<<b[i]<<" ";

    cout <<endl;
}
```

Class Template

```
std::vector<int> int_vec;
```

Templates for Data Abstraction

- **Class definitions can also be made more general with templates**
  - The syntax for class templates is basically the same as for function templates
    - template<class T> comes before the template definition
    - Type parameter T is used in the class definition just like any other type
    - Type parameter T can represent any type

A Class Template

- **The following is a class template**

```
template <class T>
class Pair
{
        public:
                Pair( );
                Pair( T firstValue, T secondValue);
                void setElement(int position, T value);
                T getElement(int position) const;
        private:
                T first;
                T second;
};
```

Declaring Template Class Objects

- **Once the class template is defined, objects may be declared**
  - Declarations must indicate what type is to be used for T
  - Example: To declare an object so it can hold a pair of integers:

```
Pair<int> score;
```

For a pair of characters:

```
Pair<char> seats;
```

Using the Objects

- **After declaration, objects based on a template class are used just like any other objects**
  - Continuing the previous example:

```
score.setElement(1,3);
score.setElement(2,0);
seats.setElement(1, 'A');
```

Defining a Pair Constructor

- **This is a definition of the constructor for class Pair that takes two arguments**

```
template<class T>
Pair<T>::Pair(T firstValue, T secondValue) : first(firstValue), second(secondValue)
  {
    //No body needed due to initialization above
  }
```

The class name includes <T>

Defining setElement

- **Here is a definition for setElement in the template class Pair**

```
void Pair<T>::setElement(int position, T value)
{
    if (position == 1)
       first = value;
    else if (position == 2)
       second = value;
    else
      …
}
```

Example: An List Class

```
#ifndef GENERICLIST_H
#define GENERICLIST_H
#include <iostream>
#include <cstdlib>

namespace mylist
{
    template<class ItemType>
    class GenericList
    {
        public:
            GenericList(int max);
            ~GenericList();

            int length() const;

            void add(ItemType new_item);
            bool full() const;

            void erase();

            friend std::ostream& operator <<(std::ostream& outs,
                    const GenericList<ItemType>& the_list)
            {
                for(int i=0; i< the_list.current_length;i++)
                    outs<<the_list.item[i]<<std::endl;
                return outs;
            }

        private:
            ItemType *item;
            int max_length;
            int current_length;
    };
```

```
    template<class ItemType>
    GenericList<ItemType>::GenericList(int max): max_length(max), current_length(0)
    {
        item = new ItemType[max];
    }

    template<class ItemType>
    GenericList<ItemType>::~GenericList()
    {
        delete[] item;
    }

    template<class ItemType>
    int GenericList<ItemType>::length()const
    {
        return (current_length);
    }

    template<class ItemType>
    void GenericList<ItemType>::add(ItemType newItem)
    {
        if(full())
        {
            std::cout<<"Error\n";
            exit(1);
        }
        else
        {
            item[current_length]=newItem;
            current_length = current_length+1;
        }
    }

    template<class ItemType>
    bool GenericList<ItemType>::full() const
    {
        return (current_length == max_length);
    }

    template<class ItemType>
    void GenericList<ItemType>::erase()
    {
        current_length =0;
    }

}

#endif
```

Example: An List Class

```
#include "genericlist.h"

int main()
{
    using namespace std;
    using namespace mylist;

    GenericList<int> first_list(2);
    first_list.add(1);
    first_list.add(2);
    cout << "first_list = \n"
         << first_list;

    GenericList<char> second_list(10);
    second_list.add('A');
    second_list.add('B');
    second_list.add('C');
    cout << "second_list = \n"
         << second_list;

    return 0;
}
```

typedef and Templates

- **You specialize a class template by giving a type argument to the class name such as Pair<int>**
  - The specialized name, Pair<int>, is used just like any class name

- **You can define a new class type name with the same meaning as the specialized name: typedef Class_Name<Type_Arg> New_Type_Name;**

  For example:

```
typedef Pair<int> PairOfInt;
PairOfInt  pair1, pair2;
```

NEXT ?

STL