C++ for Science and Engineering COSC3000/6000

2018 Spring Semester

# Part XIV
# Templates

## 1 Templates for Algorithm Abstraction

- Function definitions often use application specific adaptations of more general algorithms
  - For example: The general algorithm used in swap_values could swap variables of any type:

```
void swap_values(type_of_var& v1, type_of_var& v2)
{
        type_of_var temp;
        temp = v1;
        v1 = v2;
        v2 = temp;
}
```

### 1.1 swap_values

- Here is a version of swap_values to swap character variables:

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

- Function overloading: **sample45.cpp**

```
#include <iostream>

using namespace std;
void swap_values(int &v1, int &v2){
  int tmp = v1;
  v1 = v2;
  v2 = tmp;
}

void swap_values(double &v1, double &v2){
  double tmp = v1;
  v1 = v2;
  v2 = tmp;
}
```

```
int main(int argc, const char * argv[])
{
  int m = 5, n = 8;
  cout << "(m,n)=(" << m << "," << n << ")\n";
  swap_values(m,n);
  cout << "(m,n)=(" << m << "," << n << ")\n";

  double x = 1.2, y = 4.7;
  cout << "(x,y)=(" << x << "," << y << ")\n";
  swap_values(x,y);
  cout << "(x,y)=(" << x << "," << y << ")\n";

  return 0;
}
```

### 1.1.1 A General swap_values

- A generalized version of swap_values is shown here.

```
void swap_values(type_of_var& v1, type_of_var& v2)
{
        type_of_var temp;
        temp = v1;
        v1 = v2;
        v2 = temp;
}
```

- This function, if **type_of_var** could accept any type, could be used to swap values of any type

## 1.2 Templates for Functions

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

```
template<class T>
void swap_values(T& v1, T& v2) {
        T temp; temp = v1;
        v1 = v2;
        v = temp;
}
```

- **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
    - class means type in this context (**typename** could replace class but class is usually used)
    - **T** can be replaced by any type argument(whether the type is a class or not)
- A template overloads the function name by replacing T with the type used in a function call

## 1.3 Calling a Template Function

- Calling a function defined with a template is identical to calling a normal function
  - **sample46.cpp**: To call the template version of swap_values

```
#include <iostream>

using namespace std;

template <class T>
void swap_values(T &v1, T &v2){
  T tmp = v1;
  v1 = v2;
  v2 = tmp;
}

int main(int argc, const char * argv[])
{
  int m = 5, n = 8;
  cout << "(m,n)=(" << m << "," << n << ")\n";
  swap_values(m,n);
  cout << "(m,n)=(" << m << "," << n << ")\n";

  double x = 1.2, y = 4.7;
  cout << "(x,y)=(" << x << "," << y << ")\n";
  swap_values(x,y);
  cout << "(x,y)=(" << x << "," << y << ")\n";

  return 0;
}
```

  * The compiler checks the argument types and generates an appropriate version of swap_values

## 1.4 The Type Parameter T

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

```
template <class VariableType>
void swap_values(VariableType& v1, VariableType& v2)
{
        VariableType temp;
        :
        :
}
```

## 1.5 Templates with Multiple Parameters

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

```
template<class T1, class T2>
```

    * All parameters must be used in the template function

## 1.6 Algorithm Abstraction

- Using a template function we can express more general algorithms in C++
- **Algorithm abstraction** means expressing algorithms in a very general way so we can ignore incidental detail
  - This allows us to concentrate on the substantive part of the algorithm

## 1.7 Defining Templates

- When defining a template it is a good idea:
  - To start with an ordinary function that accomplishes the task with one type
    - It is often easier to deal with a concrete case rather than the general case
  - Then debug the ordinary function
  - Next convert the function to a template by replacing type names with a type parameter

**Practice**

Develop a function that finds the maximum value within an array.