# COS10007 - Developing Technical Software

## Week 10

## Features of C++

**Dr Prince Kurumthodathu Surendran**

# More features of C++

- References can also be used as aliases for other variables within a function, e.g.

```
int count =  1;
int &cRef  =  count;
cRef++ ;   // increment  count  (using  its alias cRef)
```

- Reference variables must be initialized in their declarations.
- It cannot be reassigned as aliases to other variables.
- All operations done on the alias (i.e., the reference) are actually done to the original variable.
- The alias is actually another name for the original variable.
- A reference argument can be a variable name, but not a constant or expression that returns a value (e.g., the result of a calculation) .

# Alias

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

int main()
{
   int x = 3;
   int &y = x; // y refers to (is an alias for) x

   cout << "x = " << x << endl << "y = " << y << endl;
   y = 7; // actually modifies x
   cout << "x = " << x << endl << "y = " << y << endl;
} // end main
```

# Output

```
x = 3
y = 3
x = 7
y = 7
```

# Default Arguments

- Sometime a program need to invoke a function repeatedly with the same argument value for a particular parameter.
- In such cases, the programmer can specify a default argument for those parameter, i.e., a default value to be passed to that parameter.
- When a program omits an argument for a parameter with a default argument, the compiler inserts the default value of that argument to be passed as an argument in the function call.
- Default arguments must be the rightmost arguments in a function's parameter list.

# Default Arguments - example

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

// function prototype that specifies default arguments
int boxVolume( int length = 1, int width = 1, int height = 1 );

int main()
{
   // no arguments--use default values for all dimensions
   cout << "The default box volume is: " << boxVolume();

   // specify length; default width and height
   cout << "\n\nThe volume of a box with length 10,\n"
      << "width 1 and height 1 is: " << boxVolume( 10 );

   // specify length and width; default height
   cout << "\n\nThe volume of a box with length 10,\n"
      << "width 5 and height 1 is: " << boxVolume( 10, 5 );
```

# Default Arguments - example

```
   // specify all arguments
   cout << "\n\nThe volume of a box with length 10,\n"
      << "width 5 and height 2 is: " << boxVolume( 10, 5, 2 )
      << endl;
} // end main

// function boxVolume calculates the volume of a box
int boxVolume( int length, int width, int height )
{
   return length * width * height;
} // end function boxVolume
```

Output

```
The default box volume is: 1

The volume of a box with length 10,
width 1 and height 1 is: 10

The volume of a box with length 10,
width 5 and height 1 is: 50

The volume of a box with length 10,
width 5 and height 2 is: 100
```

# Default Arguments – explained

- The first call to boxVolume specifies no arguments, it's using all three default values of 1.
- The second call passes a length argument , so it's using default values of 1 for the width and height arguments.
- The third call passes arguments for length and width, so a default value of 1 for the height argument is used.
- The last call passes arguments for all three parameters of length, width and height. Therefore no default values are used.

# Default Arguments – explained

- Any arguments passed to the function explicitly are assigned to the function's parameters from left to right.
- When boxVolume receives one argument, the function assigns the value of that argument to its length parameter (i.e., the leftmost parameter in the parameter list).
- When boxVolume receives two arguments, the function assigns the values of those arguments to its length and width parameters in that order.
- Finally , when box Volume receives all three arguments, the function assigns the values of those arguments to its length , width and height parameters, respectively.
- If the default values for a function change, all client code must be recompiled.

# Unary Scope Resolution Operator (: :)

- You actually can declare local and global variables of the same name. This will hide the global variable by the local variable in  the local scope.
- Use (: :) to access a global variable when there is a local variable of the same name.
- But (: :) cannot be used to access a local variable of the same name in an outer block.
- A global variable can be accessed directly without the (: :) if there is no local variable in scope with the same name.

# Unary Scope Resolution Operator - example

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

int number = 7; // global variable named number

int main()
{
   double number = 10.5; // local variable named number

   // display values of local and global variables
   cout << "Local double value of number = " << number
      << "\nGlobal int value of number = " << ::number << endl;
} // end main
```

**Output**

```
Local double value of number = 10.5
Global int value of number = 7
```

# Function Overloading

- This is a feature of C++ (not in C).
- Function Overloading allows several functions of the same name to be defined.
- These functions must have different sets of parameters, e.g. different
  - Parameter types
  - Number of parameter
  - The order of parameter types

# Function overloading

- It is used to create several functions of the same name that perform similar tasks, but on data of different types.
- For example, many functions in the math library are overloaded for different numeric data types.
- Overloading functions that perform closely related tasks can make programs more readable and understandable.

# Square functions

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

int square( int x ) { // function square for int values
   cout << "square of integer " << x << " is ";
   return x * x;
}

double square( double y ) { // function square for double values
   cout << "square of double " << y << " is ";
   return y * y;
} // end function square with double argument

int main() {
   cout << square( 7 ); // calls int version
   cout << endl;
   cout << square( 7.5 ); // calls double version
   cout << endl;
} // end main
```

**Output**

```
square of integer 7 is 49
square of double 7.5 is 56.25
```

# Function Templates

- Function templates is useful when a program logic and operations are identical for each data type.
- Defining a single function template would defines a whole family of overloaded functions.
- Given the argument types provided in calls to this function, C++ will generates separate function template specializations to handle each type of call appropriately.
- Function templates can be defined in a header file.

# Definition of function template

```
// maximum.h

template < class T >  // or template< typename T >
T maximum( T value1, T value2, T value3 )
{
   T maximumValue = value1; // assume value1 is maximum

   // determine whether value2 is greater than maximumValue
   if ( value2 > maximumValue )
      maximumValue = value2;

   // determine whether value3 is greater than maximumValue
   if ( value3 > maximumValue )
      maximumValue = value3;

   return maximumValue;
} // end function template maximum
```

# Definition of function template

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

#include "maximum.h" // include definition of function template maximum

int main()
{
   // demonstrate maximum with int values
   int int1, int2, int3;

   cout << "Input three integer values: ";
   cin >> int1 >> int2 >> int3;

   // invoke int version of maximum
   cout << "The maximum integer value is: "
      << maximum( int1, int2, int3 );
```

# Definition of function template

```
   // demonstrate maximum with double values
   double double1, double2, double3;

   cout << "\n\nInput three double values: ";
   cin >> double1 >> double2 >> double3;

   // invoke double version of maximum
   cout << "The maximum double value is: "
      << maximum( double1, double2, double3 );

   // demonstrate maximum with char values
   char char1, char2, char3;

   cout << "\n\nInput three characters: ";
   cin >> char1 >> char2 >> char3;

   // invoke char version of maximum
   cout << "The maximum character value is: "
      << maximum( char1, char2, char3 ) << endl;
} // end main
```

# Output

```
Input three integer values: 1 2 3
The maximum integer value is: 3

Input three double values: 3.3 2.2 1.1
The maximum double value is: 3.3

Input three characters: A C B
The maximum character value is: C
```

# Function template

- Three functions are created as a result of the function calls, expecting three int values, three double values and three char values , respectively .
- For example, the function template specialization created for type int replaces each occurrence of T with int.

# Thank You