PEN203

Introducing Object Technology

C How to Program
Deitel & Deitel

Outline

- Introduction
- C++
- A Simple Program: Adding Two Integers
- Inline Functions
- References and Reference Parameters
- Empty Parameter Lists
- Default Arguments
- Unary Scope Resolution Operator
- Function Overloading
- Function Templates

Introduction

- Two additional programming concepts:
  - Object-oriented programming
    - Classes, encapsulation, operator everloading, inheritance and polymorphism
  - Generic programming
    - Function templates and class templates

C++

- C++ was developed by B. Stroustrup
  - Originally called “C with classes”
- C++ improves C’s features
- C++ provides object-oriented programming
- Increase software productivity, quality and reusability
- C++ file names have one of several extensions: .cpp, .cxx or .C

A Simple Program: Adding Two Integers

```
1  // Fig. 18.1: fig18_01.cpp
2  // Addition program that displays the sum of two numbers.
3  #include <iostream> // allows program to perform input and output
4
5  int main()
6  {
7     int number1; // first integer to add
8
9     std::cout << "Enter first integer: "; // prompt user for data
10    std::cin >> number1; // read first integer from user into number1
11
12    int number2; // second integer to add
13    int sum; // sum of number1 and number2
14
15    std::cout << "Enter second integer: "; // prompt user for data
16    std::cin >> number2; // read second integer from user into number2
17    sum = number1 + number2; // add the numbers; store result in sum
18    std::cout << "Sum is " << sum << std::endl; // display sum; end line
19
20    return 0; // indicate that program ended successfully
21 } // end function main
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

A Simple Program: Adding Two Integers

- // comment is maximum one line long
- iostream must be included to use input/output operations
- C++ requires that all functions should have return types
- Declarations can be placed anywhere in a C++ program (you should define a variable before you use it)

A Simple Program: Adding Two Integers

- To read a value from keyboard input stram object cin is used.
- >> is stream extraction operator.
  - Waits for a user to enter a value. Converts value to variable (placed right of operator) data type.
- cin>>number1; reads an integer from keyboard and stores it in variable number1.

A Simple Program: Adding Two Integers

- endl is used to output a newline. It also flushes the buffer.
- cout is output stream object and is used to output on the screen.
- << is stream insertion operator and it knows how to output each data type

Inline Functions

- Inline functions reduce function call overhead.
- Keyword inline before a function's return type in the function definition indicates that compiler generates o copy of the function's code in place.
- The compiler can ignore inline qualifier.

References and Reference Parameters

- Reference parameter is an alias for its corresponding argument in a function call.
- & operator is placed after the parameter data type in function prototype and function header.
- int &aref; aref is a reference to an integer value.
- Parameter name in function actually refers to the original variable .
- Simulate call by reference.

References and Reference Parameters

```
1  // Fig. 18.5: fig18_05.cpp
2  // Comparing pass-by-value and pass-by-reference with references.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int squareByValue( int ); // function prototype (value pass)
8  void squareByReference( int & ); // function prototype (reference pass)
9
10 int main()
11 {
12    int x = 2; // value to square using squareByValue
13    int z = 4; // value to square using squareByReference
14
15    // demonstrate squareByValue
16    cout << "x = " << x << " before squareByValue\n";
17    cout << "Value returned by squareByValue: "
18       << squareByValue( x ) << endl;
19    cout << "x = " << x << " after squareByValue\n" << endl;
20
21    // demonstrate squareByReference
22    cout << "z = " << z << " before squareByReference" << endl;
23    squareByReference( z );
24    cout << "z = " << z << " after squareByReference" << endl;
25    return 0; // indicates successful termination
26 } // end main
```

References and Reference Parameters

```
27
28 // squareByValue multiplies number by itself, stores the
29 // result in number and returns the new value of number
30 int squareByValue( int number )
31 {
32    return number *= number; // caller's argument not modified
33 } // end function squareByValue
34
35 // squareByReference multiplies numberRef by itself and stores the result
36 // in the variable to which numberRef refers in the caller
37 void squareByReference( int &numberRef )
38 {
39    numberRef *= numberRef; // caller's argument modified
40 } // end function squareByReference
```

```
x = 2 before squareByValue
Value returned by squareByValue: 4
x = 2 after squareByValue

z = 4 before squareByReference
z = 16 after squareByReference
```

References and Reference Parameters

- References can also be created and used in a function.
- All operations performed in reference is actually performed in original variable.
- References must be initialized in their declarations.
- Example:

```
int a=1;
int &aref=a;
aref++;
```

The value of a is 2. It is incremented through alias aref.

Empty Parameter Lists

- Specified by writing either void or nothing at all.
- void print(); or void print(void);
- Both print functions do not take any arguments and do not return a value.

Default Arguments

- A default argument is a default value to be passed to a parameter.
- It is used when the function call does not specify an argument for that parameter.
- Must be the rightmost argument(s) in parameter list.

Default Arguments

```
1  // Fig. 18.8: fig18_08.cpp
2  // Using default arguments.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  // function prototype that specifies default arguments
8  int boxVolume( int length = 1, int width = 1, int height = 1 );
9
10 int main()
11 {
12    // no arguments--use default values for all dimensions
13    cout << "The default box volume is: " << boxVolume();
14
15    // specify length; default width and height
16    cout << "\n\nThe volume of a box with length 10,\n"
17       << "width 1 and height 1 is: " << boxVolume( 10 );
18
19    // specify length and width; default height
20    cout << "\n\nThe volume of a box with length 10,\n"
21       << "width 5 and height 1 is: " << boxVolume( 10, 5 );
22
23    // specify all arguments
24    cout << "\n\nThe volume of a box with length 10,\n"
25       << "width 5 and height 2 is: " << boxVolume( 10, 5, 2 )
26       << endl;
27    return 0; // indicates successful termination
28 } // end main
```

Default Arguments

```
29
30 // function boxVolume calculates the volume of a box
31 int boxVolume( int length, int width, int height )
32 {
33    return length * width * height;
34 } // end function boxVolume
```

```
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
```

Unary Scope Resolution Operator

- Unary scope resolution operator (::)
- The unary scope resolution operator is used to access a global variable if in local scope there is a varible with the same name.

```
1  // Fig. 18.9: fig18_09.cpp
2  // Using the unary scope resolution operator.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int number = 7; // global variable named number
8
9  int main()
10 {
11    double number = 10.5; // local variable named number
12
13    // display values of local and global variables
14    cout << "Local double value of number = " << number
15       << "\nGlobal int value of number = " << ::number << endl;
16    return 0; // indicates successful termination
17 } // end main
```

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

Function Overloading

- Overloaded functions have same name but different sets of parameters.
- Compiler selects the correct function based on the number, types and order of arguments.
- They usually are used perform similar tasks on different data types.

Function Overloading

```
1  // Fig. 18.10: fig18_10.cpp
2  // Overloaded functions.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  // function square for int values
8  int square( int x )
9  {
10    cout << "square of integer " << x << " is ";
11    return x * x;
12 } // end function square with int argument
13
14 // function square for double values
15 double square( double y )
16 {
17    cout << "square of double " << y << " is ";
18    return y * y;
19 } // end function square with double argument
20
21 int main()
22 {
23    cout << square( 7 ); // calls int version
24    cout << endl;
25    cout << square( 7.5 ); // calls double version
26    cout << endl;
27    return 0; // indicates successful termination
28 } // end main
```

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

Function Templates

- Template functions are defined once and used for different data types.
- Identical program logic and operations for each datat type
- Begins with the template keyword.

```
1  // Fig. 18.12: maximum.h
2  // Definition of function template maximum.
3
4  template < class T >  // or template< typename T >
5  T maximum( T value1, T value2, T value3 )
6  {
7     T maximumValue = value1; // assume value1 is maximum
8
9     // determine whether value2 is greater than maximumValue
10    if ( value2 > maximumValue )
11       maximumValue = value2;
12
13    // determine whether value3 is greater than maximumValue
14    if ( value3 > maximumValue )
15       maximumValue = value3;
16
17    return maximumValue;
18 } // end function template maximum
```

Function Templates

```
1  // Fig. 18.13: fig18_13.cpp
2  // Function template maximum test program.
3  #include <iostream>
4  using std::cout;
5  using std::cin;
6  using std::endl;
7
8  #include "maximum.h" // include definition of function template maximum
9
10 int main()
11 {
12    // demonstrate maximum with int values
13    int int1, int2, int3;
14
15    cout << "Input three integer values: ";
16    cin >> int1 >> int2 >> int3;
17
18    // invoke int version of maximum
19    cout << "The maximum integer value is: "
20       << maximum( int1, int2, int3 );
21
22    // demonstrate maximum with double values
23    double double1, double2, double3;
24
25    cout << "\n\nInput three double values: ";
26    cin >> double1 >> double2 >> double3;
27
```

Function Templates

```
28    // invoke double version of maximum
29    cout << "The maximum double value is: "
30       << maximum( double1, double2, double3 );
31
32    // demonstrate maximum with char values
33    char char1, char2, char3;
34
35    cout << "\n\nInput three characters: ";
36    cin >> char1 >> char2 >> char3;
37
38    // invoke char version of maximum
39    cout << "The maximum character value is: "
40       << maximum( char1, char2, char3 ) << endl;
41    return 0; // indicates successful termination
42 } // end main
```

```
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
```