Escola de Engenharia
Universidade do Minho

Complementos de Programação de Computadores – Aula 1

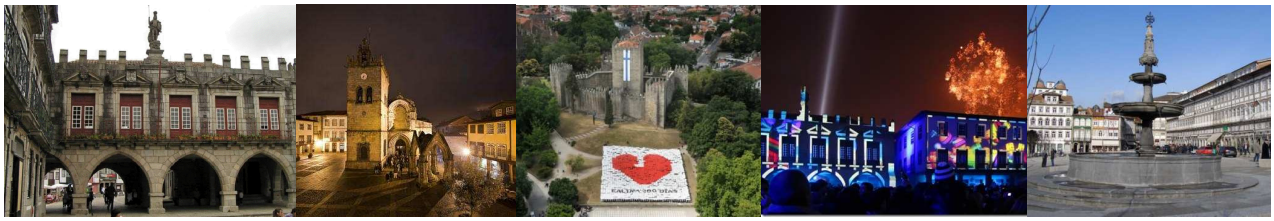
Introdução ao C++

Luís Paulo Reis

lpreis@dsi.uminho.pt

Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em P.Deitel e H.Deitel 2010 e Paulo Cortez, 2011)



Escola de Engenharia
Universidade do Minho

Outline

- 15.1 Introduction to C++
- 15.2 A Simple Program: Adding Two Integers
- 15.3 C++ Standard Library
- 15.4 Header Files
- 15.5 Inline Functions
- 15.6 References and Reference Parameters
- 15.7 Default Arguments and Empty Parameter Lists
- 15.8 Unary Scope Resolution Operator
- 15.9 Function Overloading
- 15.10 Function Templates

Escola de Engenharia
Universidade do Minho

Objectives

- To become familiar with the C++ enhancements to C
- To become familiar with the C++ standard library
- To understand the concept of inline functions
- To be able to create and manipulate references
- To understand the concept of default arguments
- To understand the role the unary scope resolution operator has in scoping
- To be able to overload functions
- To be able to define functions that can perform similar operations on different types of data

Escola de Engenharia
Universidade do Minho

15.1 C++ Language

- **C++ Language**
  - Improves on many of C's features
  - Has object-oriented capabilities
    - Increases software quality and reusability
  - Developed by Bjarne Stroustrup at Bell Labs
    - Called "C with classes"
    - C++ (increment operator) - enhanced version of C
  - Superset of C
    - Can use a C++ compiler to compile C programs
    - Gradually evolve the C programs to C++
- **ANSI C++**
  - Final version at http://www.ansi.org/
  - Free, older version at http://www.cygnus.com/misc/wp/

Escola de Engenharia
Universidade do Minho

15.2 A Simple Program: Adding Two Integers

- **File extensions**
  - C files: `.c`
  - C++ files: `.cpp` (which we use), `.cxx`, `.C` (uppercase)
- **Differences**
  - C++ allows you to "comment out" a line by preceding it with `//`
  - For example: `// text to ignore`
  - `<iostream>` - input/output stream header file
  - Return types - all functions must declare their return type
    - C does not require it, but C++ does
  - Variables in C++ can be defined almost anywhere
    - In C, required to defined variables in a block, before any executable statements

Escola de Engenharia
Universidade do Minho

15.2 A Simple Program: Adding Two Integers (II)

- **Input/Output in C++**
  - Performed with streams of characters
  - Streams sent to input/output objects
- **Output**
  - `std::cout` - standard output stream (connected to screen)
  - `<<` stream insertion operator ("put to")
  - `std::cout << "hi";`
    - Puts `"hi"` to `std::cout`, which prints it on the screen
- **Input**
  - `std::cin` - standard input object (connected to keyboard)
  - >> stream extraction operator ("get from")
  - `std::cin >> myVariable;`
    - Gets stream from keyboard and puts it into `myVariable`

15.3 A Simple Program: Adding Two Integers (III)

- `std::endl`
  - "end line"
  - Stream manipulator - prints a newline and flushes output buffer
    - Some systems do not display output until "there is enough text to be worthwhile"
    - `std::endl` forces text to be displayed
- `using` statements
  - Allow us to remove the `std::` prefix
  - Discussed later
- **Cascading**
  - Can have multiple **<<** or **>>** operators in a single statement

```
std::cout << "Hello " << "there" << std::endl;
```


Outline

fig15_01.cpp

```
1  // Fig. 15.1: fig15_01.cpp
2  // Addition program
3  #include <iostream>
4
5  int main()
6  {
7     int integer1;
8
9     std::cout << "Enter first integer\n";
10    std::cin >> integer1;
11
12    int integer2, sum;          // declaration
13
14    std::cout << "Enter second integer\n";
15    std::cin >> integer2;
16    sum = integer1 + integer2;
17    std::cout << "Sum is " << sum << std::endl;
18
19    return 0;   // indicate that program ended successfully
20 } // end function main
```

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

Escola de Engenharia
Universidade do Minho

15.3 C++ Standard Library

- **C++ programs built from**
  - Functions
  - Classes
    - Most programmers use library functions
- **Two parts to learning C++**
  - Learn the language itself
  - Learn the library functions
- **Making your own functions**
  - Advantage: you know exactly how they work
  - Disadvantage: time consuming, difficult to maintain efficiency and design well

Escola de Engenharia
Universidade do Minho

15.4 Header Files

- **Header files**
  - Each standard library has header files
    - Contain function prototypes, data type definitions, and constants
  - Files ending with `.h` are "old-style" headers
- **User defined header files**
  - Create your own header file
    - End it with `.h`
  - Use `#include "myFile.h"` in other files to load your header

15.4 Header Files

Standard library header file	Explanation
<cassert>	Contains macros and information for adding diagnostics that aid program debugging. The old version of this header file is <assert.h>.
<cctype>	Contains function prototypes for functions that test characters for certain properties, that can be used to convert lowercase letters to uppercase letters and vice versa. This header file replaces header file <ctype.h>.
<cfloat>	Contains the floating-point size limits of the system. This header file replaces header file <float.h>.
<climits>	Contains the integral size limits of the system. This header file replaces header file <limits.h>.
<cmath>	Contains function prototypes for math library functions. This header file replaces header file <math.h>.
<cstdio>	Contains function prototypes for the standard input/output library functions and information used by them. This header file replaces header file <stdio.h>.
<cstdlib>	Contains function prototypes for conversions of numbers to text, text to numbers, memory allocation, random numbers and various other utility functions. This header file replaces header file <stdlib.h>.
<cstring>	Contains function prototypes for C-style string processing functions. This header file replaces header file <string.h>.
<ctime>	Contains function prototypes and types for manipulating the time and date. This header file replaces header file <time.h>.
<iostream>	Contains function prototypes for the standard input and standard output functions. This header file replaces header file <iostream.h>.

Fig. 15.2 Standard library header files. (Part 1 of 3)

15.4 Header Files

Standard library header file	Explanation
<iomanip>	Contains function prototypes for the stream manipulators that enable formatting of streams of data. This header file replaces <iomanip.h>.
<fstream>	Contains function prototypes for functions that perform input from files on disk and output to files on disk. This header file replaces header file <fstream.h>.
<utility>	Contains classes and functions that are used by many standard library header files.
<vector>, <list>, <deque>, <queue>, <stack>, <map>, <set>, <bitset>	These header files contain classes that implement the standard library containers. Containers are used to store data during a program's execution.
<functional>	Contains classes and functions used by standard library algorithms.
<memory>	Contains classes and functions used by the standard library to allocate memory to the standard library containers.
<iterator>	Contains classes for accessing data in the standard library containers.
<algorithm>	Contains functions for manipulating data in standard library containers.
<exception>, <stdexcept>	These header files contain classes that are used for exception handling (discussed in Chapter 23).

Fig. 15.2 Standard library header files. (Part 2 of 3)

Escola de Engenharia
Universidade do Minho

15.4 Header Files

Standard library header file	Explanation
`<string>`	Contains the definition of class `string` from the standard library.
`<sstream>`	Contains prototypes for functions that perform input from strings in memory and output to strings in memory.
`<locale>`	Contains classes and functions normally used by stream processing to process data in the natural form for different languages (e.g., monetary formats, sorting strings, character presentation, etc.).
`<limits>`	Contains classes for defining the numerical data type limits on each computer platform.
`<typeinfo>`	Contains classes for run-time type identification (determining data types at execution time).

Fig. 15.2 Standard library header files. (Part 3 of 3)

Escola de Engenharia
Universidade do Minho

15.5 Inline Functions

- **Function calls**
  - Cause execution-time overhead
  - Qualifier `inline` before function return type "advises" a function to be inlined
    - Puts copy of function's code in place of function call
  - Speeds up performance but increases file size
  - Compiler can ignore the `inline` qualifier
    - Ignores all but the smallest functions

```
inline double cube( const double s )
   { return s * s * s; }
```

- **Using statements**
  - By writing `using std::cout;` we can write `cout` instead of `std::cout` in the program
  - Same applies for `std::cin` and `std::endl`

```
1  // Fig. 15.3: fig15_03.cpp
2  // Using an inline function to calculate
3  // the volume of a cube.
4  #include <iostream>
5
6  using std::cout;
7  using std::cin;
8  using std::endl;
9
10 inline double cube( const double s ) { return s * s * s; }
11
12 int main()
13 {
14    double side;
15
16    for ( int k = 1; k < 4; k++ ) {
17       cout << "Enter the side length of your cube:  ";
18       cin >> side;
19       cout << "Volume of cube with side "
20            << side << " is " << cube( side ) << endl;
21    } // end for
22
23    return 0;
24 } // end function main
```

Outline

fig15_03.cpp


```
Enter the side length of your cube:  1.0
Volume of cube with side 1 is 1
Enter the side length of your cube:  2.3
Volume of cube with side 2.3 is 12.167
Enter the side length of your cube:  5.4
Volume of cube with side 5.4 is 157.464
```

Outline

Program Output

Escola de Engenharia
Universidade do Minho

15.5 Inline Functions (II)

- `bool`
  - Boolean - new data type, can either be `true` or `false`

C++ Keywords

Keywords common to the C and C++ programming languages

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++ only keywords

asm	bool	catch	class	const_cast
delete	dynamic_cast	explicit	false	friend
inline	mutable	namespace	new	operator
private	protected	public	reinterpret_cast	
static_cast	template	this	throw	true
try	typeid	typename	using	virtual
wchar_t				

Escola de Engenharia
Universidade do Minho

15.6 References and Reference Parameters

- **Call by value**
  - Copy of data passed to function
  - Changes to copy do not change original
- **Call by reference**
  - Function can directly access data
  - Changes affect original
- **Reference parameter alias for argument**
  - Use &

```
void change(int &variable)
  {
       variable += 3;
  }
```

    - Adds 3 to the original variable input
  - `int y = &x`
    - Changing `y` changes `x` as well

15.6 References and Reference Parameters (II)

- **Dangling references**
  - Make sure to assign references to variables
  - If a function returns a reference to a variable, make sure the variable is `static`
    - Otherwise, it is automatic and destroyed after function ends
- **Multiple references**
  - Like pointers, each reference needs an &

```
int &a, &b, &c;
```


Outline

fig15_05.cpp (Part 1 of 2)

```
1  // Fig. 15.5: fig15_05.cpp
2  // Comparing call-by-value and call-by-reference
3  // with references.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  int squareByValue( int );
10 void squareByReference( int & );
11
12 int main()
13 {
14    int x = 2, z = 4;
15
16    cout << "x = " << x << " before squareByValue\n"
17         << "Value returned by squareByValue: "
18         << squareByValue( x ) << endl
19         << "x = " << x << " after squareByValue\n" << endl;
20
```

```
21    cout << "z = " << z << " before squareByReference" << endl;
22    squareByReference( z );
23    cout << "z = " << z << " after squareByReference" << endl;
24
25    return 0;
26 } // end function main
27
28 int squareByValue( int a )
29 {
30    return a *= a;   // caller's argument not modified
31 } // end function squareByValue
32
33 void squareByReference( int &cRef )
34 {
35    cRef *= cRef;    // caller's argument modified
36 } // end function squareByReference
```

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

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

Outline

fig15_05.cpp (Part 2 of 2)

Program Output


```
1  // Fig. 15.6: fig15_06.cpp
2  // References must be initialized
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int main()
9  {
10    int x = 3, &y = x;  // y is now an alias for x
11
12    cout << "x = " << x << endl << "y = " << y << endl;
13    y = 7;
14    cout << "x = " << x << endl << "y = " << y << endl;
15
16    return 0;
17 } // end function main
```

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

Outline

fig15_06.cpp

```cpp
// Fig. 15.7: fig15_07.cpp
// References must be initialized
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   int x = 3, &y;   // Error: y must be initialized

   cout << "x = " << x << endl << "y = " << y << endl;
   y = 7;
   cout << "x = " << x << endl << "y = " << y << endl;

   return 0;
} // end function main
```

Borland C++ command-line compiler error message

```
Error E2304 Fig15_07.cpp 10: Reference variable 'y' must be initialized
in function main()
```

Microsoft Visual C++ compiler error message

```
Fig15_07.cpp(10) : error C2530: 'y' : references must be initialized
```

Outline

fig15_.07.cpp

Escola de Engenharia
Universidade do Minho

15.7 Default Arguments and Empty Parameter Lists

- **If function parameter omitted, gets default value**
  - Can be constants, global variables, or function calls
  - If not enough parameters specified, rightmost go to their defaults

- **Set defaults in function prototype**

```
int myFunction( int x = 1, int y = 2, int z = 3 );
```

- **Empty parameter lists**
  - In C, empty parameter list means function takes any argument
    - In C++ it means function takes no arguments
  - To declare that a function takes no parameters:
    - Write `void` or nothing in parentheses
    - Prototypes:

```
void print1( void );
void print2();
```

```
1  // Fig. 15.8: fig15_08.cpp
2  // Using default arguments
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int boxVolume( int length = 1, int width = 1, int height = 1 );
9
10 int main()
11 {
12    cout << "The default box volume is: " << boxVolume()
13         << "\n\nThe volume of a box with length 10,\n"
14         << "width 1 and height 1 is: " << boxVolume( 10 )
15         << "\n\nThe volume of a box with length 10,\n"
16         << "width 5 and height 1 is: " << boxVolume( 10, 5 )
17         << "\n\nThe volume of a box with length 10,\n"
18         << "width 5 and height 2 is: " << boxVolume( 10, 5, 2 )
19         << endl;
20
21    return 0;
22 } // end function main
23
```

Outline

fig15_08.cpp (Part 1 of 2)


```
24 // Calculate the volume of a box
25 int boxVolume( int length, int width, int height )
26 {
27    return length * width * height;
28 } // 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
```

Outline

fig15_08.cpp (Part 2 of 2)

15.8 Unary Scope Resolution Operator

- **Unary scope resolution operator (`::`)**
  - Access global variables if a local variable has same name
  - Instead of `variable` use `::variable`
- `static_cast<newType> (variable)`
  - Creates a copy of `variable` of type `newType`
  - Convert `int`s to `float`s, etc.
- **Stream manipulators**
  - Can change how output is formatted
  - `setprecision` - set precision for `float`s (default 6 digits)
  - `setiosflags` - formats output
  - `setwidth` - set field width
  - Discussed in depth in Chapter 21

Outline

fig15_09.cpp (Part 1 of 2)

```
1  // Fig. 15.9: fig15_09.cpp
2  // Using the unary scope resolution operator
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::ios;
8
9  #include <iomanip>
10
11 using std::setprecision;
12 using std::setiosflags;
13 using std::setw;
14
15 const double PI = 3.14159265358979;
16
17 int main()
18 {
19    const float PI = static_cast< float >( ::PI );
20
21    cout << setprecision( 20 )
22         << "  Local float value of PI = " << PI
23         << "\nGlobal double value of PI = " << ::PI << endl;
24
```

```
25    cout << setw( 28 ) << "Local float value of PI = "
26         << setiosflags( ios::fixed | ios::showpoint )
27         << setprecision( 10 ) << PI << endl;
28    return 0;
29 } // end function main
```

Borland C++ command-line compiler output

```
  Local float value of PI = 3.141592741012573242
Global double value of PI = 3.141592653589790007
Local float value of PI = 3.1415927410
```

Microsoft Visual C++ compiler output

```
  Local float value of PI = 3.1415927410125732
Global double value of PI = 3.14159265358979
  Local float value of PI = 3.1415927410
```

Outline

fig15_09.cpp (Part 2 of 2)

Escola de Engenharia
Universidade do Minho

15.9 Function Overloading

- **Function overloading:**
  - Functions with same name and different parameters
  - Overloaded functions should perform similar tasks
    - Function to square `int`s and function to square `float`s

```
int square( int x) {return x * x;}
float square(float x) { return x * x; }
```

  - Program chooses function by signature
    - Signature determined by function name and parameter types
    - Type safe linkage - ensures proper overloaded function called

```
1  // Fig. 15.10: fig15_10.cpp
2  // Using overloaded functions
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int square( int x ) { return x * x; }
9
10 double square( double y ) { return y * y; }
11
12 int main()
13 {
14    cout << "The square of integer 7 is " << square( 7 )
15         << "\nThe square of double 7.5 is " << square( 7.5 )
16         << endl;
17
18    return 0;
19 } // end function main
```

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

Outline

fig15_10.cpp

Escola de Engenharia
Universidade do Minho

15.10 Function Templates

- **Function templates**
  - Compact way to make overloaded functions
  - Keyword `template`
  - Keyword `class` or `typename` before every formal type parameter (built in or user defined)

```
template < class T >  //or template< typename T >
T square( T value1)
{
   return value1 * value1;
}
```

  - T replaced by type parameter in function call

```
int x;
int y = square(x);
```

    - If `int` parameter, all `T`'s become `int`s
    - Can use `float`, `double`, `long`...

```
1  // Fig. 15.11: fig15_11.cpp
2  // Using a function template
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  template < class T >
10 T maximum( T value1, T value2, T value3 )
11 {
12    T max = value1;
13
14    if ( value2 > max )
15       max = value2;
16
17    if ( value3 > max )
18       max = value3;
19
20    return max;
21 } // end function template maximum
22
```

Outline

fig15_11.cpp (Part 1 of 2)


```
23 int main()
24 {
25    int int1, int2, int3;
26
27    cout << "Input three integer values: ";
28    cin >> int1 >> int2 >> int3;
29    cout << "The maximum integer value is: "
30         << maximum( int1, int2, int3 );          // int version
31
32    double double1, double2, double3;
33
34    cout << "\nInput three double values: ";
35    cin >> double1 >> double2 >> double3;
36    cout << "The maximum double value is: "
37         << maximum( double1, double2, double3 ); // double version
38
39    char char1, char2, char3;
40
41    cout << "\nInput three characters: ";
42    cin >> char1 >> char2 >> char3;
43    cout << "The maximum character value is: "
44         << maximum( char1, char2, char3 )        // char version
45         << endl;
46
47    return 0;
48 } // end function main
```

Outline

fig15_11.cpp (Part 2 of 2)

```
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
```


Program Output

Escola de Engenharia
Universidade do Minho

Complementos de Programação de Computadores – Aula 1

Introdução ao C++

Luís Paulo Reis

lpreis@dsi.uminho.pt

Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em P.Deitel e H.Deitel 2010 e Paulo Cortez, 2011)