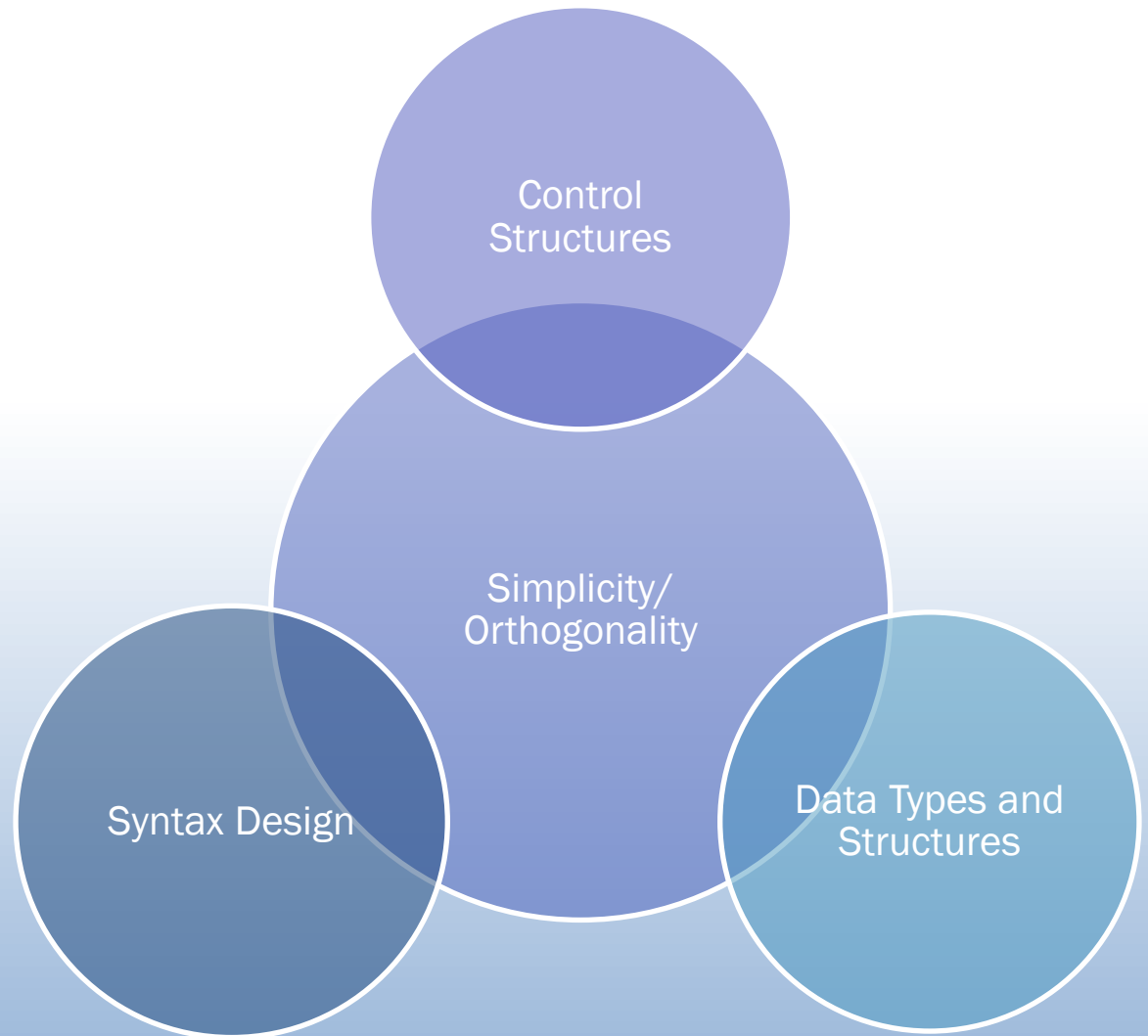





C++: Variables, Datatypes & Operators

Original author: Dr. Ha Viet Uyen Synh.

Agenda



Structure of a C++ Program



```
1 // Fig. 2.1. fig02_01.cpp
2 // Text-printing program.
3 #include <iostream> // allows program to output data to the screen
4
5 // function main begins program execution
6 int main()
7 {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11
12 } // end function main
```

Single-line comment _ slash

Allows access to an I/O library

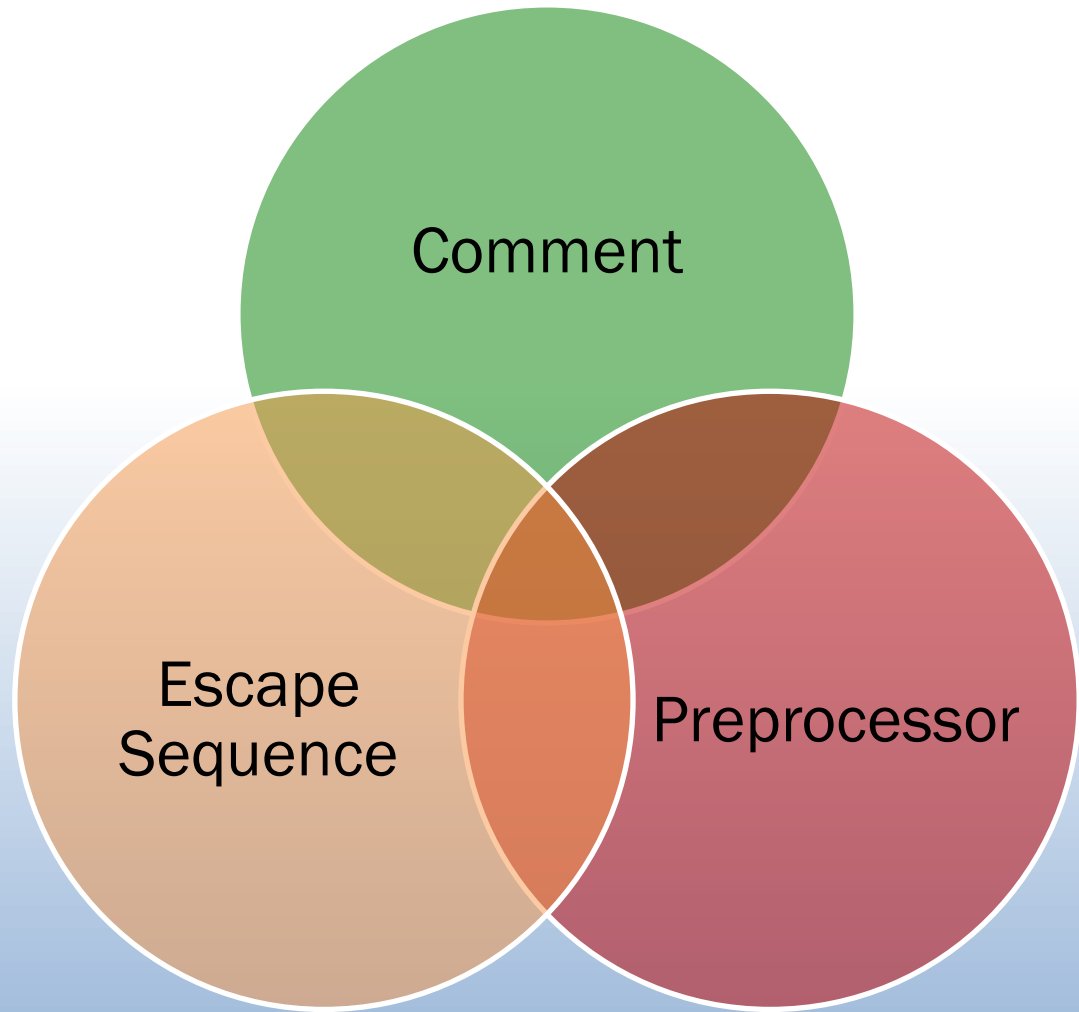
Starts definition of special function main()

output (print) a string

Program returns a status code (0 means OK)

Welcome to C++!

Part #1





Comments

Comments contain text that is not converted to machine language (it's just there for humans).

Everything after "//" is ignored by the compiler.

Everything between "/*" and "*/" is ignored

```
// Dave's Homework #1
// This program is awesome!
#include <iostream>
/* This program computes the coefficient of expansion of
the universe to 27 decimal places.
*/
int main() {
    std::cout << 1.000000000000000000000001;
    return 0;
}
```

A decorative image on the left side of the slide showing a stack of smooth, dark stones on a calm body of water, with their reflections visible. The stones are stacked in a slightly offset manner, creating a sense of balance and tranquility.

Includes

The statement: `#include <foo.h>` inserts the contents of the file `foo.h` inside your file before the compiler starts.

Definitions that allow your program to use the functions and classes that make up the standard C++ library are in these files.

You can include your own file(s):

```
#include "myfile.h"
```



C++ Preprocessor

C++ Compilers automatically invoke a *preprocessor* that takes care of `#include` statements and some other special directives.

You don't need to do anything special to run the preprocessor - it happens automatically.

Lines that start with the character '`#`' are special to instructions to a *preprocessor*.

The preprocessor can replace the line with something else:
include: replaced with contents of a file

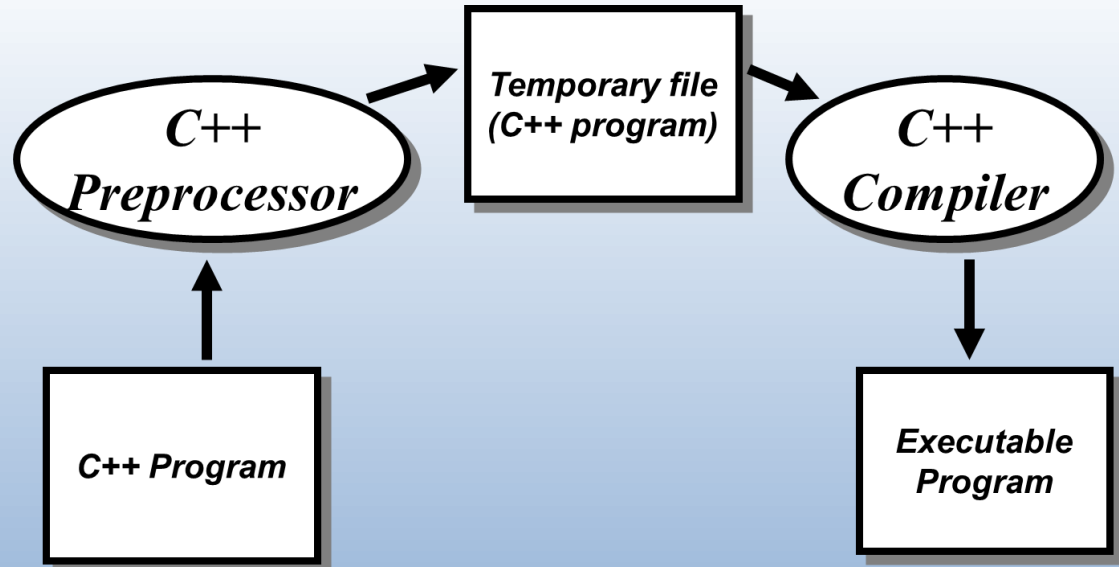
Other *directives* tell the preprocessor to look for patterns in the program and do some fancy processing.

#define (macro) Example

```
#define square(a) (a * a)
```

`y = square(x) ;` ← *becomes* `y = (x * x);`

`z = square(y*x) ;` ← *becomes* `z = (y*x * y*x);`



Escape sequences.

Escape sequence	Description
<code>\n</code>	Newline. Position the screen cursor to the beginning of the next line.
<code>\t</code>	Horizontal tab. Move the screen cursor to the next tab stop.
<code>\r</code>	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
<code>\a</code>	Alert. Sound the system bell.
<code>\\</code>	Backslash. Used to print a backslash character.
<code>\'</code>	Single quote. Use to print a single quote character.
<code>\"</code>	Double quote. Used to print a double quote character.

Printing multiple lines of text with a single statement

```
1  // Fig. 2.4: fig02_04.cpp
2  // Printing multiple lines of text with a single statement.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8      std::cout << "Welcome\nto\n\n C++!\n";
9
10     return 0; // indicate that program ended successfully
11
12 } // end function main
```

```
Welcome
to

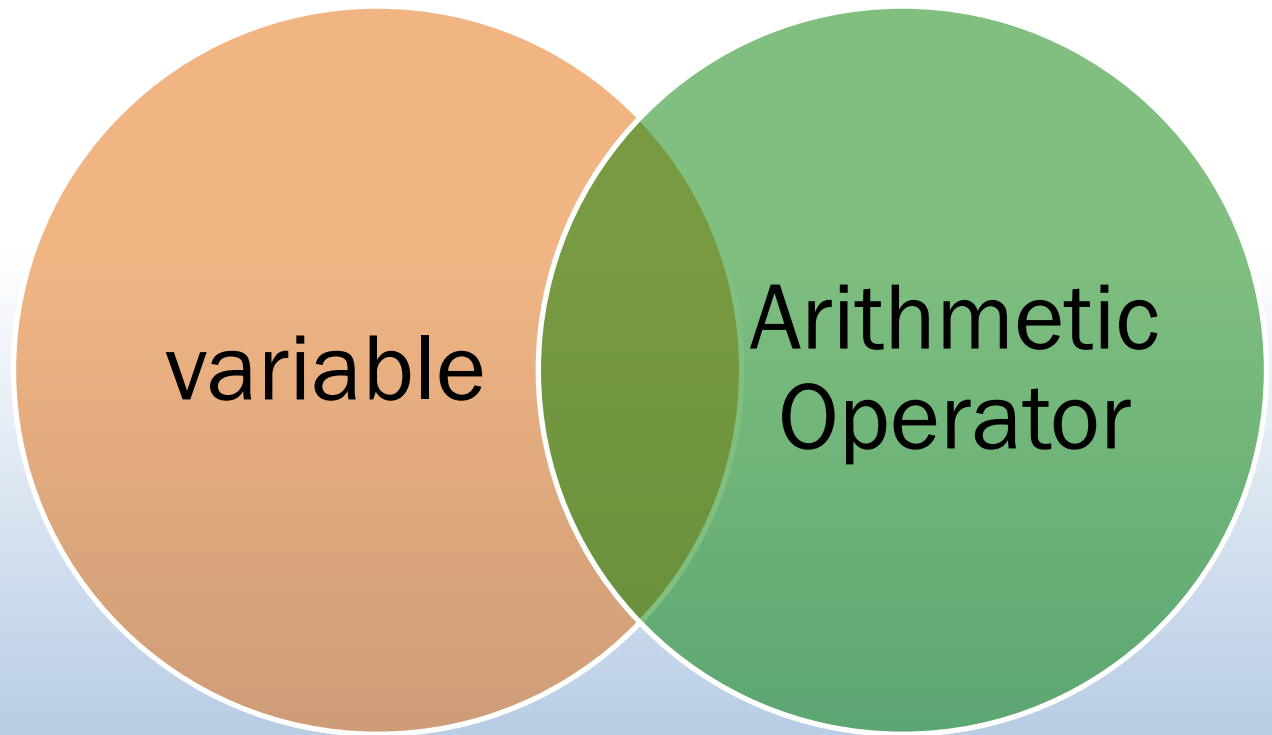
C++!
```

Another C++ Program

```
1 // Fig. 2.5: fig02_05.cpp
2 // Addition program that displays the sum of two numbers.
3 #include <iostream> // allows program to perform input and output
4
5 // function main begins program execution
6 int main()
7 {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
12
13    std::cout << "Enter first integer: "; // prompt user for data
14    std::cin >> number1; // read first integer from user into number1
15
16    std::cout << "Enter second integer: "; // prompt user for data
17    std::cin >> number2; // read second integer from user into number2
18
19    sum = number1 + number2; // add the numbers; store result in sum
20
21    std::cout << "Sum is " << sum << std::endl; // display sum; end li
22
23    return 0; // indicate that program ended successfully
24
25 }
```

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

Part #2



Variables

The program now uses *variables*:

```
int number1, number2, sum;
```

Variables are just names for locations in memory.

In C++ all variables must have a *type* (not all languages require this).

In C++ all variables must be *declared* before they can be used.

C++ variables are declared like this:

```
type var_name;
```

type indicates what kind of variable.

C++ built in types include:

```
int char float double bool
```

You can also create new types!

number1	45
number2	72
sum	117



Variable Names

C++ variable names:

Made up of *letters, digits and underscore.*

Must start with a non-digit.

Case sensitive

`foo` is not the same name as `Foo`

Can be any length

Good variable names tell the reader what the variable is used for!



Literals (Constants)

Literals are fixed values used by a program.

Some examples of literals:

<code>22</code>	<code>3.14159</code>	<code>0x2A</code>
<code>false</code>	<code>"Hi Dave"</code>	<code>'c'</code>

You can initialize a variable in the declaration by *assigning* it a value:

```
int foo = 17;  
double PI = 3.14159;  
char newline = '\n';
```



Expressions

C++ *expressions* are used to express computation.

Expressions include operators and the *operands* on which the operations are applied.


Operands can be variables, literals or function calls.

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	<code>f + 7</code>
Subtraction	-	$p - c$	<code>p - c</code>
Multiplication	*	bm or $b \cdot m$	<code>b * m</code>
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	<code>x / y</code>
Modulus	%	$r \bmod s$	<code>r % s</code>

Precedence of arithmetic operators

Operator(s)	Operation(s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
* / %	Multiplication Division Modulus	Evaluated second. If there are several, they are evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.





Precedence and associativity of the operators

Operators				Associativity	Type
()				left to right	parentheses
*	/	%		left to right	multiplicative
+	-			left to right	additive
<<	>>			left to right	stream insertion/extraction
<	<=	>	>=	left to right	relational
==	!=			left to right	equality
=				right to left	assignment

Algebraic and C++ Expressions

Algebra: $m = \frac{a + b + c + d + e}{5}$

C++: `m = (a + b + c + d + e) / 5;`

Algebra: $y = mx + b$

C++: `y = m * x + b;`

Algebra: $z = pr \% q + w / x - y$

C++: `z = p * r % q + w / x - y;`

6

1

2

4

3

5

Enter two integers to compare: 3 7

3 != 7

3 < 7

3 <= 7

Enter two integers to compare: 22 12

22 != 12

22 > 12

22 >= 12

Enter two integers to compare: 7 7

7 == 7

7 <= 7

7 >= 7

```
1 // Fig. 2.13: fig02_13.cpp
2 // Comparing integers using if statements, relational operators
3 // and equality operators.
4 #include <iostream> // allows program to perform input and output
5
6 using std::cout; // program uses cout
7 using std::cin; // program uses cin
8 using std::endl; // program uses endl
9
10 // function main begins program execution
11 int main()
12 {
13     int number1; // first integer to compare
14     int number2; // second integer to compare
15
16     cout << "Enter two integers to compare: "; // prompt user for data
17     cin >> number1 >> number2; // read two integers from user
18
19     if ( number1 == number2 )
20         cout << number1 << " == " << number2 << endl;
21
22     if ( number1 != number2 )
23         cout << number1 << " != " << number2 << endl;
24
25     if ( number1 < number2 )
26         cout << number1 << " < " << number2 << endl;
27
28     if ( number1 > number2 )
29         cout << number1 << " > " << number2 << endl;
30
31     if ( number1 <= number2 )
32         cout << number1 << " <= " << number2 << endl;
33
34     if ( number1 >= number2 )
35         cout << number1 << " >= " << number2 << endl;
36
37     return 0; // indicate that program ended successfully
38
39 } // end function main
```

Any Questions?

