

Software Engineering 2

(C++)

CSY2006

Module Overview/Objectives (provisional)

- Introduction to Software Engineering (SE), Programming Languages, Comparison of C++ and Java (all along the course), Visual C++ IDE
- Preprocessors, Data Types, Arrays, Variables, Operators, Expressions, Statements and Blocks ,
- Control Flow Statements: Decision Making, Looping and Branching
- Structures and Functions, Introduction to Classes and Objects
- Inheritance
- Arrays and Strings

Module Overview/Objectives (provisional)

- Polymorphism (Operator Overloading)
- Pointers
- Virtual Functions
- Streams and Files
- Namespaces, Templates and Exceptions
- Searching and Sorting Algorithms
- Data Structures (Linked List, Stacks, Queues, Binary Trees, Graphs, Expression Evaluation)

Software Engineering

- ❑ The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software

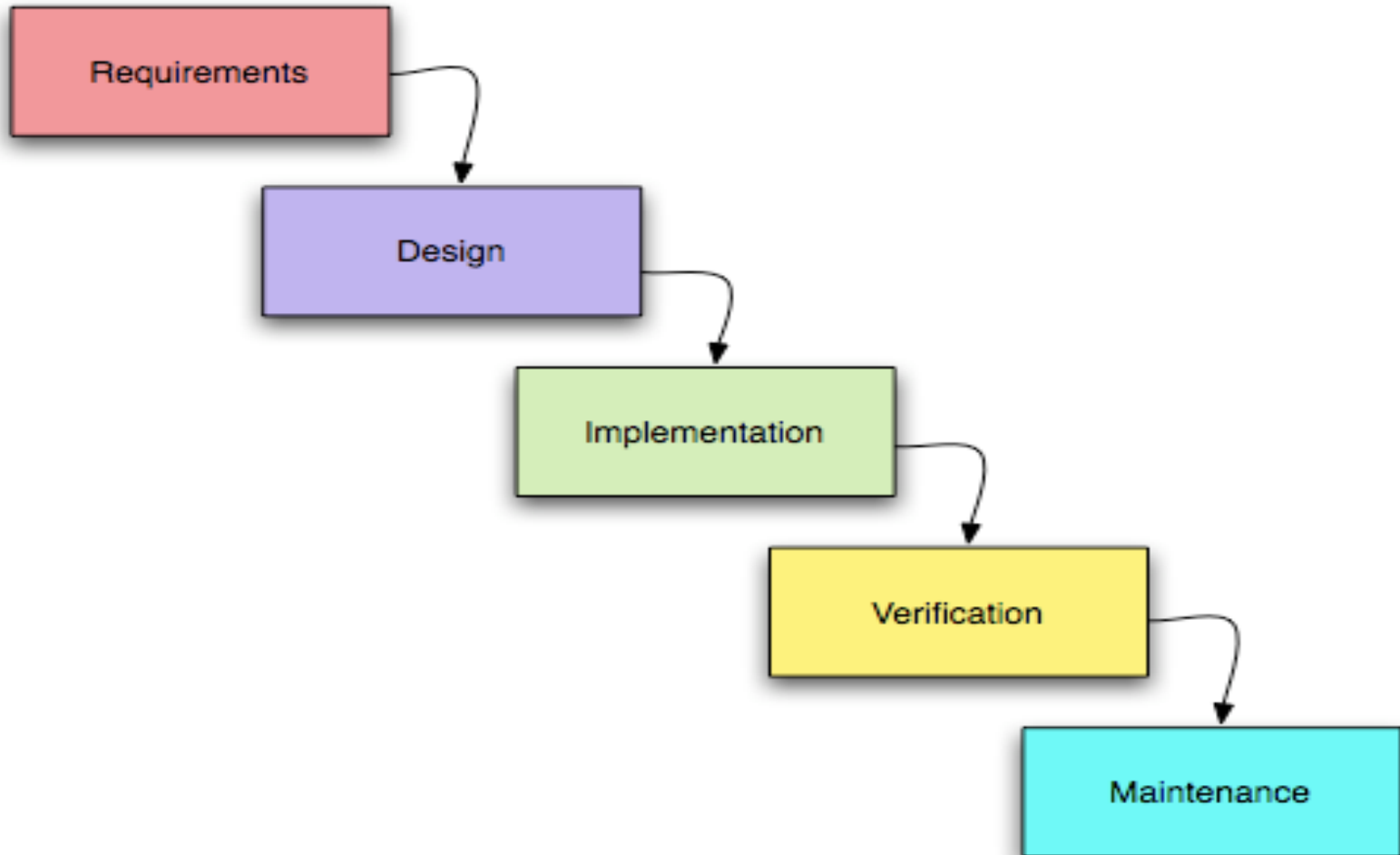
-IEEE

- ❑ It is the application of Engineering to software because it integrates significant mathematics, computer science and practices whose origins are in Engineering

- ACM

... much more than just programming

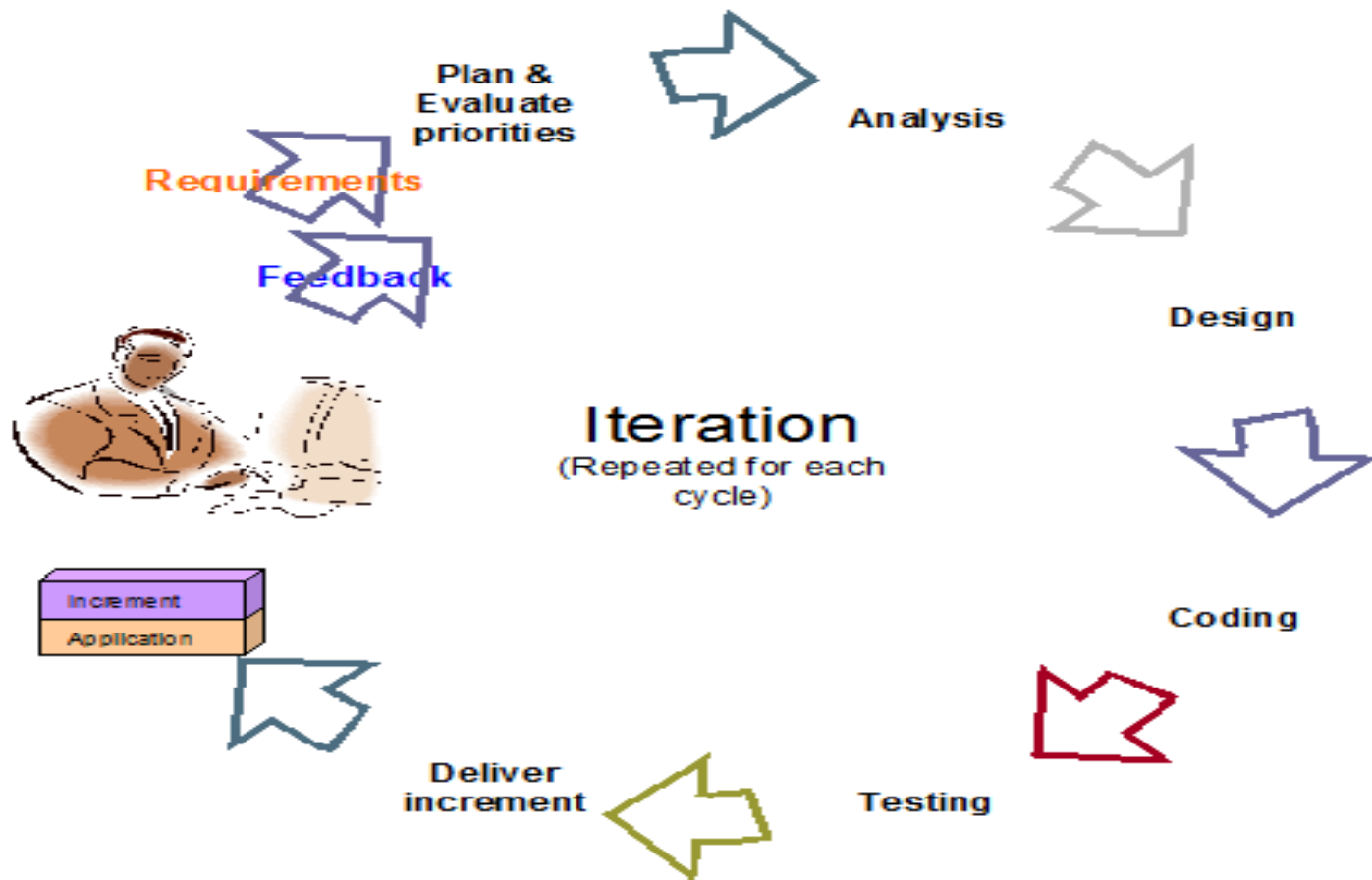
Traditional Software Development Process



Software Development Process

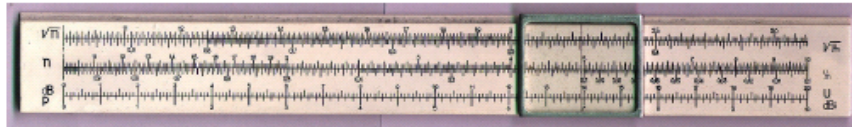
- *Software requirements* specify *what* a program must accomplish. Requirements are expressed in a document called a *Software Requirements Specification*
- A *software design* indicates how a program will accomplish its requirements
- *Implementation* is the process of writing the source code that will solve the problem
- *Verification/Testing* is the act of ensuring that a program will solve the intended problem given all of the constraints under which it must perform
- *Maintenance* is the act of improving software programs after delivery for reusing it in the future

Agile Software Development



Augustine's Law – Growth of Software: Order of Magnitude Every 10 Years

In The Beginning



1960's



**F-4A
1000
LOC**



1970's



**F-15A
50,000
LOC**



1980's



**F-16C
300K
LOC**



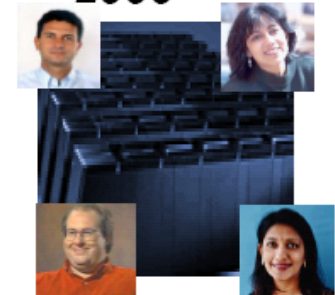
1990's



**F-22
1.7M
LOC**



2000+



**F-35
>6M
LOC**



Programming Languages

- **Machine Language** consisting of binary code (0s and 1s). Computers understand only this. It is processor dependent.
- **Assembly Language** consisting of mnemonics (symbols) to make programming less tedious and faster. (e.g. Load A, Add B, Store C). An *Assembler* converts assembly language into machine language. It is also processor dependent.
- **High-level Language** consisting of English-like instructions to make programming simpler and faster. (e.g. $C = A + B$). A compiler/interpreter helps convert high-level language into machine language. It is not processor dependent.

Some High-level languages

- *procedural languages*: programs are a series of commands
 - **Pascal** (1970): designed for education
 - **C** (1972): operating systems and device drivers
- *functional programming*: functions map inputs to outputs
 - **Lisp** (1958) / **Scheme** (1975), **Haskell** (1990)
- *object-oriented languages*: programs use interacting "objects"
 - **Smalltalk** (1980): first major object-oriented language
 - **C++** (1985): "object-oriented" improvements to C
 - **Java** (1995): general purpose language and world's most widely used computer programming language.
(Deitel & Deitel)

C/C++ vs Java

- C++:
 - Power and control: What to do? How to do it?
- Java:
 - “Do it my way and I’ll do more of the work for you”
 - But it may be less efficient than doing it yourself
 - Some things cannot be done in Java alone (JNI)
- **Java hides many things from you**
 - And decides how you will do things
- **Java prevents you doing some things and checks others**
 - **C++ trusts that you know what you are doing**
 - If you do not, then you can REALLY break things
- Do you want/need the power/control of C/C++?

What is C++?

Procedural C

Global Functions
File-specific functions
Structs
Pointers (addresses)
Low-level memory access
C Preprocessor

Variables
Arrays
Loops
Conditionals

Classes

- Grouping of related data together
- With associated methods (functions)

'new' for object creation
'delete' for object destruction
Constructors, Destructors
Operator Overloading
Assignment operators
Conversion operators
Inheritance (sub-classing)
Virtual functions & polymorphism
Access control (private/public/protected)

Function Libraries

Standard functions
Custom libraries
O/S functions

Templates

(Generic classes)

Non-C features
e.g. References

Class Libraries

(+templated classes)
Standard library
Custom libraries
Platform specific libraries

What about Java?

Procedural C

~~Global Functions~~
~~File-specific functions~~
~~Structs~~
~~Pointers (addresses)~~
~~Low-level memory access~~
~~C Preprocessor~~

Variables
Arrays
Loops
Conditionals

Classes

- Grouping of related data together
- With associated methods (functions)

'new' for object creation
~~'delete' for object destruction~~
Constructors, ~~Destructors~~
~~Operator Overloading~~
~~Assignment operators~~
~~Conversion operators~~ (toString())
Inheritance (sub-classing)
(ONLY) Virtual functions & polymorphism
Access control (private/public/protected)

Function Libraries

~~Standard functions~~
~~Custom libraries~~
~~O/S functions~~
Java Native Interface

Templates

'Generics' (weaker)

Non-C features

(ONLY) references

Class Libraries

(Standardised)
Collections
Networking
Graphics

What Sun changed for Java

- Remember: C++ came first
 - The Java changes were deliberate!
- Java is cross-platform
 - Interpreted intermediate byte-code (.class files)
 - Standard cross-platform class libraries
 - Libraries include graphics (AWT, SWING, ...), networking, ...
 - Platform independent type sizes
 - *Cannot take advantage of platform-specific features*
- Java prevents things which are potentially dangerous
 - Pointer arithmetic (but it can be fast)
 - Writing outside arrays (checks take time)
 - Low-level access to memory (dangerous per powerful)
 - Uninitialised data (initialisation takes time)
- Java forces you to use objects
 - Even when it would be quicker not to
- Java does garbage collection for you
 - Safer(?), but may execute slower than freeing memory yourself

So which is better: Java or C++?

- What does 'better' mean?
- What are you trying to do?
- Do you need the power and control that C++ gives you?
- With less options, things may seem simpler
 - Potentially harder to make mistakes
 - But you lose the flexibility to optimise
- If you know both, then you have more options (and the basics are very similar)

Basic Data Types - Summary

Type	Minimum size (bits)	Minimum range of values (Depends upon the size on your platform)
<code>char</code>	8	-128 to 127 (Java chars are 16 bit!)
<code>short</code>	16	-32768 to 32767
<code>long</code>	32	-2147483648 to 2147483647
<code>float</code>	Often 32	Single precision (implementation defined) e.g. 23 bit mantissa, 8 bit exponent
<code>double</code>	Often 64	Double precision (implementation defined) e.g. 52 bit mantissa, 11 bit exponent
<code>long double</code>	\geq double	Extended precision, implementation defined
<code>int</code>	\geq short	varies

ints, bools and booleans

- In C++ integer types (char, short, long, int) can be used in conditions
 - (In C++ the value is silently converted to a C++ **bool** type)
- When using integer types:
 - true is equivalent to non-zero (or 1), false is equivalent to zero
- Example:

```
int x = 6;
if ( x )
{
    x = x - 2;
}
```

- In Java this would be an error (x not boolean)
- In C++ this is valid (it means 'if (x != 0)')

The `bool` Data Type

- Represents values that are `true` or `false`
- `bool` variables are stored as small integers
- `false` is represented by 0, `true` by 1:

```
bool allDone = true;    allDone finished
bool finished = false;  

|   |
|---|
| 1 |
|---|



|   |
|---|
| 0 |
|---|


```

Boolean Variables

Program 2-17

```
1  // This program demonstrates boolean variables.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      bool boolValue;
8
9      boolValue = true;
10     cout << boolValue << endl;
11     boolValue = false;
12     cout << boolValue << endl;
13     return 0;
14 }
```

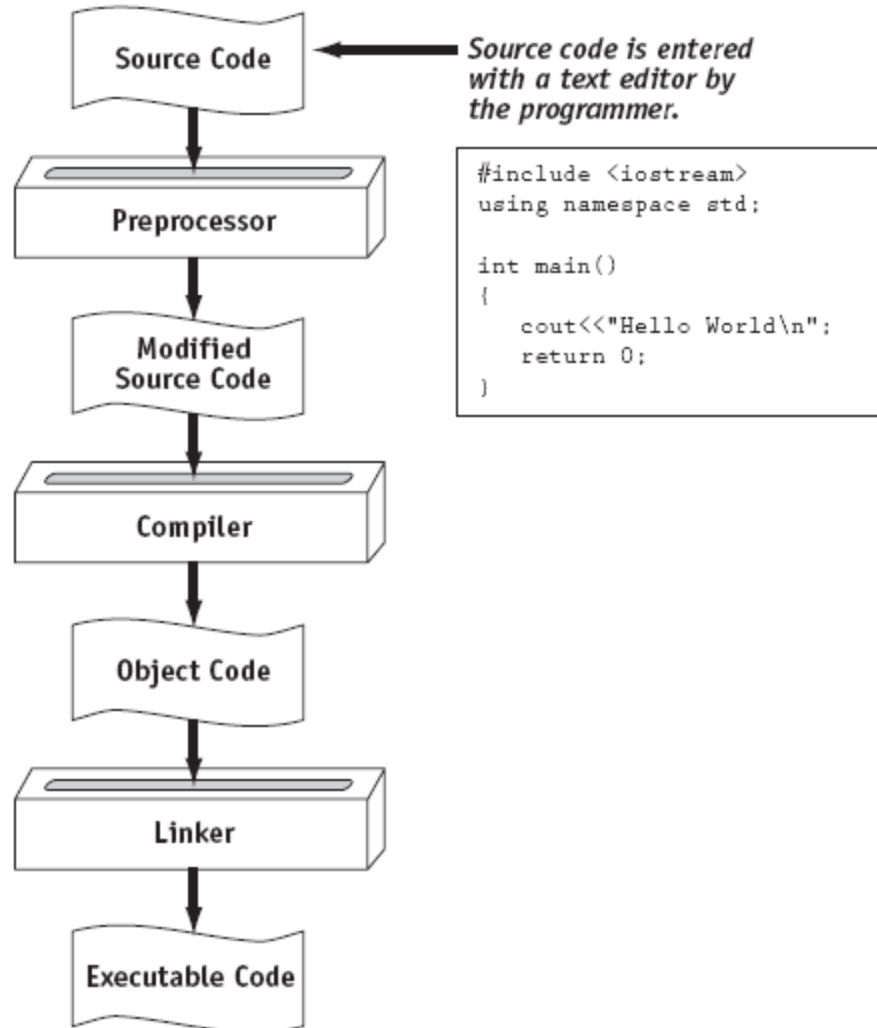
Program Output

1
0

From a High-Level Program to an Executable File

- a) Create file containing the program with a text editor.
 - b) Run preprocessor to convert source file directives to source code program statements.
 - c) Run compiler to convert source program into machine instructions.
 - d) Run linker to connect hardware-specific code to machine instructions, producing an executable file.
- Steps b–d are often performed by a single command or button click.
 - Errors detected at any step will prevent execution of following steps.

From a High-Level Program to an Executable File



The Parts of a C++ Program

```
// sample C++ program ← comment
#include <iostream> ← preprocessor directive
using namespace std; ← which namespace to use
int main() ← beginning of function named main
{ ← beginning of block for main
    cout << "Hello, there!"; ← output statement
    return 0; ← send 0 to operating system
} ← end of block for main
```

Note: For Visual Studio and Visual C++ IDE, use `cin.get()` or `system("PAUSE")` to prevent console from disappearing!

Hello World Program

```
//*****  
// This program prints Hello World!  
//*****  
  
#include <iostream>  
using namespace std;  
int main ()  
{  
    cout << "Hello World!";  
    cin.get(); //system("PAUSE");  
    return 0;  
}
```

Special Characters

Character	Name	Meaning
//	Double slash	Beginning of a comment
#	Pound sign	Beginning of preprocessor directive
< >	Open/close brackets	Enclose filename in #include
()	Open/close parentheses	Used when naming a function
{ }	Open/close brace	Encloses a group of statements
" "	Open/close quotation marks	Encloses string of characters
;	Semicolon	End of a programming statement

The cout Object

- Displays output on the computer screen
- You use the stream insertion operator << to send output to cout:

```
cout << "Programming is fun!";
```

The cout Object

- Can be used to send more than one item to cout:

```
cout << "Hello " << "there!";
```

Or:

```
cout << "Hello ";
```

```
cout << "there!";
```

The cout Object

- This produces one line of output:

```
cout << "Programming is ";  
cout << "fun!";
```

The `endl` Manipulator

- You can use the `endl` manipulator to start a new line of output. This will produce two lines of output:

```
cout << "Programming is" << endl;  
cout << "fun!";
```

The endl Manipulator

```
cout << "Programming is" << endl;  
cout << "fun!";
```



The `endl` Manipulator

- You do NOT put quotation marks around `endl`
- The last character in `endl` is a lowercase L, not the number 1.

`endl` ← This is a lowercase L

The `\n` Escape Sequence

- You can also use the `\n` escape sequence to start a new line of output. This will produce two lines of output:

```
cout << "Programming is\n";  
cout << "fun!";
```

Notice that the `\n` is **INSIDE**
the string.

The `\n` Escape Sequence

```
cout << "Programming is\n";  
cout << "fun!";
```



The `cin` Object

- Standard input object
- Like `cout`, requires `iostream` file
- Used to read input from keyboard
- Information retrieved from `cin` with `>>`
- Input is stored in one or more variables

The cin Object

Program 3-1

```
1  // This program asks the user to enter the length and width of
2  // a rectangle. It calculates the rectangle's area and displays
3  // the value on the screen.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9      int length, width, area;
10
11      cout << "This program calculates the area of a ";
12      cout << "rectangle.\n";
13      cout << "What is the length of the rectangle? ";
14      cin >> length;
15      cout << "What is the width of the rectangle? ";
16      cin >> width;
17      area = length * width;
18      cout << "The area of the rectangle is " << area << ".\n";
19      return 0;
20 }
```

Program Output with Example Input Shown in Bold

This program calculates the area of a rectangle.
What is the length of the rectangle? **10 [Enter]**
What is the width of the rectangle? **20 [Enter]**
The area of the rectangle is 200.

The `cin` Object

- `cin` converts data to the type that matches the variable:

```
int height;  
cout << "How tall is the room? ";  
cin >> height;
```

Displaying a Prompt

- A prompt is a message that instructs the user to enter data.
- You should always use `cout` to display a prompt before each `cin` statement.

```
cout << "How tall is the room? ";  
cin >> height;
```

The `cin` Object

- Can be used to input more than one value:

```
cin >> height >> width;
```

- Multiple values from keyboard must be separated by spaces
- Order is important: first value entered goes to first variable, etc.

The cin Object Gathers Multiple Values

Program 3-2

```
1  // This program asks the user to enter the length and width of
2  // a rectangle. It calculates the rectangle's area and displays
3  // the value on the screen.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9      int length, width, area;
10
11      cout << "This program calculates the area of a ";
12      cout << "rectangle.\n";
13      cout << "Enter the length and width of the rectangle ";
14      cout << "separated by a space.\n";
15      cin >> length >> width;
16      area = length * width;
17      cout << "The area of the rectangle is " << area << endl;
18      return 0;
19  }
```

Program Output with Example Input Shown in Bold

This program calculates the area of a rectangle.

Enter the length and width of the rectangle separated by a space.

10 20 [Enter]

The area of the rectangle is 200

The cin Object Reads Different Data Types

Program 3-3

```
1 // This program demonstrates how cin can read multiple values
2 // of different data types.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     int whole;
9     double fractional;
10    char letter;
11
12    cout << "Enter an integer, a double, and a character: ";
13    cin >> whole >> fractional >> letter;
14    cout << "Whole: " << whole << endl;
15    cout << "Fractional: " << fractional << endl;
16    cout << "Letter: " << letter << endl;
17    return 0;
18 }
```

Program Output with Example Input Shown in Bold

```
Enter an integer, a double, and a character: 4 5.7 b [Enter]
Whole: 4
Fractional: 5.7
Letter: b
```


The `#include` Directive

- Inserts the contents of another file into the program
- This is a preprocessor directive, not part of C++ language
- `#include` lines not seen by compiler
- Do not place a semicolon at end of `#include` line

Variable Definition

Program 2-7

```
1  // This program has a variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int number;
8
9      number = 5;
10     cout << "The value in number is " << number << endl;
11     return 0;
12 }
```



Program Output

The value in number is 5

Named Constants

- Named constant (constant variable): variable whose content cannot be changed during program execution
- Used for representing constant values with descriptive names:

```
const double TAX_RATE = 0.0675;
```

```
const int NUM_STATES = 50;
```

- Often named in uppercase letters

Named Constants

Program 2-28

```
1 // This program calculates the circumference of a circle.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     // Constants
8     const double PI = 3.14159;
9     const double DIAMETER = 10.0;
10
11     // Variable to hold the circumference
12     double circumference;
13
14     // Calculate the circumference.
15     circumference = PI * DIAMETER;
16
17     // Display the circumference.
18     cout << "The circumference is: " << circumference << endl;
19     return 0;
20 }
```

Program Output

The circumference is: 31.4159

Programming Style

- The visual organization of the source code
- Includes the use of spaces, tabs, and blank lines
- Does not affect the syntax of the program
- Affects the readability of the source code

Programming Style

Common elements to improve readability:

- Braces { } aligned vertically
- Indentation of statements within a set of braces
- Blank lines between declaration and other statements
- Long statements wrapped over multiple lines with aligned operators

Standard and Prestandard C++

Older-style C++ programs:

- Use `.h` at end of header files:
- `#include <iostream.h>`
- Use `#define` preprocessor directive instead of `const` definitions
- Do not use `using namespace` convention
- May not compile with a standard C++ compiler

#define directive

Program 2-31

```
1 // This program calculates the circumference of a circle.
2 #include <iostream>
3 using namespace std;
4
5 #define PI 3.14159
6 #define DIAMETER 10.0
7
8 int main()
9 {
10     // Variable to hold the circumference
11     double circumference;
12
13     // Calculate the circumference.
14     circumference = PI * DIAMETER;
15
16     // Display the circumference.
17     cout << "The circumference is: " << circumference << endl;
18     return 0;
19 }
```

Program Output

The circumference is: 31.4159

Binary Arithmetic Operators

SYMBOL	OPERATION	EXAMPLE	VALUE OF ans
+	addition	<code>ans = 7 + 3;</code>	10
-	subtraction	<code>ans = 7 - 3;</code>	4
*	multiplication	<code>ans = 7 * 3;</code>	21
/	division	<code>ans = 7 / 3;</code>	2
%	modulus	<code>ans = 7 % 3;</code>	1

A Closer Look at the / Operator

- / (division) operator performs integer division if both operands are integers

```
cout << 13 / 5;      // displays 2
```

```
cout << 91 / 7;      // displays 13
```

- If either operand is floating point, the result is floating point

```
cout << 13 / 5.0;    // displays 2.6
```

```
cout << 91.0 / 7;    // displays 13.0
```

A Closer Look at the % Operator

- % (modulus) operator computes the remainder resulting from integer division

```
cout << 13 % 5;    // displays 3
```

- % requires integers for both operands

```
cout << 13 % 5.0; // error
```

Mathematical Expressions

- Can create complex expressions using multiple mathematical operators
- An expression can be a literal, a variable, or a mathematical combination of constants and variables
- Can be used in assignment, `cout`, other statements:

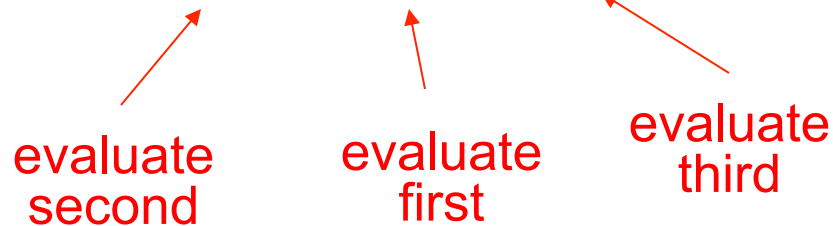
```
area = 2 * PI * radius;  
cout << "border is: " << 2*(1+w);
```

Order of Operations

In an expression with more than one operator, evaluate in this order:

- (unary negation), in order, left to right
- * / %, in order, left to right
- + –, in order, left to right

In the expression $2 + 2 * 2 - 2$



evaluate second

evaluate first

evaluate third

Order of Operations

Table 3-2 Some Simple Expressions and Their Values

Expression	Value
$5 + 2 * 4$	13
$10 / 2 - 3$	2
$8 + 12 * 2 - 4$	28
$4 + 17 \% 2 - 1$	4
$6 - 3 * 2 + 7 - 1$	6

Associativity of Operators

- $-$ (unary negation) associates right to left
- $*$, $/$, $\%$, $+$, $-$ associate right to left
- parentheses $()$ can be used to override the order of operations:

$$2 + 2 * 2 - 2 = 4$$

$$(2 + 2) * 2 - 2 = 6$$

$$2 + 2 * (2 - 2) = 2$$

$$(2 + 2) * (2 - 2) = 0$$

Grouping with Parentheses

Table 3-4 More Simple Expressions and Their Values

Expression	Value
$(5 + 2) * 4$	28
$10 / (5 - 3)$	5
$8 + 12 * (6 - 2)$	56
$(4 + 17) \% 2 - 1$	0
$(6 - 3) * (2 + 7) / 3$	9

Type Casting

- Used for manual data type conversion
- Useful for floating point division using ints:

```
double m;  
m = static_cast<double>(y2-y1)  
                        / (x2-x1);
```

- Useful to see `int` value of a `char` variable:

```
char ch = 'C';  
cout << ch << " is "  
      << static_cast<int>(ch);
```


Type Casting

Program 3-9

```
1  // This program uses a type cast to avoid integer division.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int books;           // Number of books to read
8      int months;          // Number of months spent reading
9      double perMonth;     // Average number of books per month
10
11     cout << "How many books do you plan to read? ";
12     cin >> books;
13     cout << "How many months will it take you to read them? ";
14     cin >> months;
15     perMonth = static_cast<double>(books) / months;
16     cout << "That is " << perMonth << " books per month.\n";
17     return 0;
18 }
```

Program Output with Example Input Shown in Bold

How many books do you plan to read? **30 [Enter]**

How many months will it take you to read them? **7 [Enter]**

That is 4.28571 books per month.

C-Style and Prestandard Type Cast Expressions

- C-Style cast: data type name in ()
`cout << ch << " is " << (int)ch;`
- Prestandard C++ cast: value in ()
`cout << ch << " is " << int(ch);`
- Both are still supported in C++, although `static_cast` is preferred

Multiple Assignment and Combined Assignment

- The = can be used to assign a value to multiple variables:

`x = y = z = 5;`

- Value of = is the value that is assigned
- Associates right to left:

`x = (y = (z = 5)) ;`

↑
value
is 5

↑
value
is 5

↑
value
is 5

Combined Assignment

- The combined assignment operators provide a shorthand for these types of statements.
- The statement

`sum = sum + 1;`

is equivalent to

`sum += 1;`

Combined Assignment Operators

Table 3-9

Operator	Example Usage	Equivalent to
<code>+=</code>	<code>x += 5;</code>	<code>x = x + 5;</code>
<code>-=</code>	<code>y -= 2;</code>	<code>y = y - 2;</code>
<code>*=</code>	<code>z *= 10;</code>	<code>z = z * 10;</code>
<code>/=</code>	<code>a /= b;</code>	<code>a = a / b;</code>
<code>%=</code>	<code>c %= 3;</code>	<code>c = c % 3;</code>

Formatting Output

- Can control how output displays for numeric, string data:
 - size
 - position
 - number of digits
- Requires `ioomanip` header file

Stream Manipulators

- Used to control how an output field is displayed
- Some affect just the next value displayed:
 - `setw(x)` : print in a field at least `x` spaces wide. Use more spaces if field is not wide enough

The setw Stream Manipulator

Program 3-13

```
1  // This program displays three rows of numbers.
2  #include <iostream>
3  #include <iomanip>          // Required for setw
4  using namespace std;
5
6  int main()
7  {
8      int num1 = 2897, num2 = 5,    num3 = 837,
9          num4 = 34,   num5 = 7,    num6 = 1623,
10         num7 = 390,  num8 = 3456, num9 = 12;
11
12     // Display the first row of numbers
13     cout << setw(6) << num1 << setw(6)
14          << num2 << setw(6) << num3 << endl;
15
16     // Display the second row of numbers
17     cout << setw(6) << num4 << setw(6)
18          << num5 << setw(6) << num6 << endl;
19
20     // Display the third row of numbers
21     cout << setw(6) << num7 << setw(6)
22          << num8 << setw(6) << num9 << endl;
23     return 0;
24 }
```

Continued...

The setw Stream Manipulator

Program Output

```
2897      5    837
   34      7  1623
  390 3456    12
```

Stream Manipulators

- Some affect values until changed again:
 - `fixed`: use decimal notation for floating-point values
 - `setprecision(x)`: when used with `fixed`, print floating-point value using `x` digits after the decimal. Without `fixed`, print floating-point value using `x` significant digits
 - `showpoint`: always print decimal for floating-point values

More Stream Manipulators

Program 3-17

```
1  // This program asks for sales figures for 3 days. The total
2  // sales are calculated and displayed in a table.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9      double day1, day2, day3, total;
10
11     // Get the sales for each day.
12     cout << "Enter the sales for day 1: ";
13     cin >> day1;
14     cout << "Enter the sales for day 2: ";
15     cin >> day2;
16     cout << "Enter the sales for day 3: ";
17     cin >> day3;
18
19     // Calculate the total sales.
20     total = day1 + day2 + day3;
```

Continued...

More Stream Manipulators

```
21
22     // Display the sales figures.
23     cout << "\nSales Figures\n";
24     cout << "-----\n";
25     cout << setprecision(2) << fixed;
26     cout << "Day 1: " << setw(8) << day1 << endl;
27     cout << "Day 2: " << setw(8) << day2 << endl;
28     cout << "Day 3: " << setw(8) << day3 << endl;
29     cout << "Total: " << setw(8) << total << endl;
30     return 0;
31 }
```

Program Output with Example Input Shown in Bold

Enter the sales for day 1: **1321.87** [Enter]

Enter the sales for day 2: **1869.26** [Enter]

Enter the sales for day 3: **1403.77** [Enter]

Sales Figures

Day 1: 1321.87

Day 2: 1869.26

Day 3: 1403.77

Total: 4594.90

Stream Manipulators

Table 3-12

Stream Manipulator	Description
<code>setw(<i>n</i>)</code>	Establishes a print field of <i>n</i> spaces.
<code>fixed</code>	Displays floating-point numbers in fixed point notation.
<code>showpoint</code>	Causes a decimal point and trailing zeroes to be displayed, even if there is no fractional part.
<code>setprecision(<i>n</i>)</code>	Sets the precision of floating-point numbers.
<code>left</code>	Causes subsequent output to be left justified.
<code>right</code>	Causes subsequent output to be right justified.