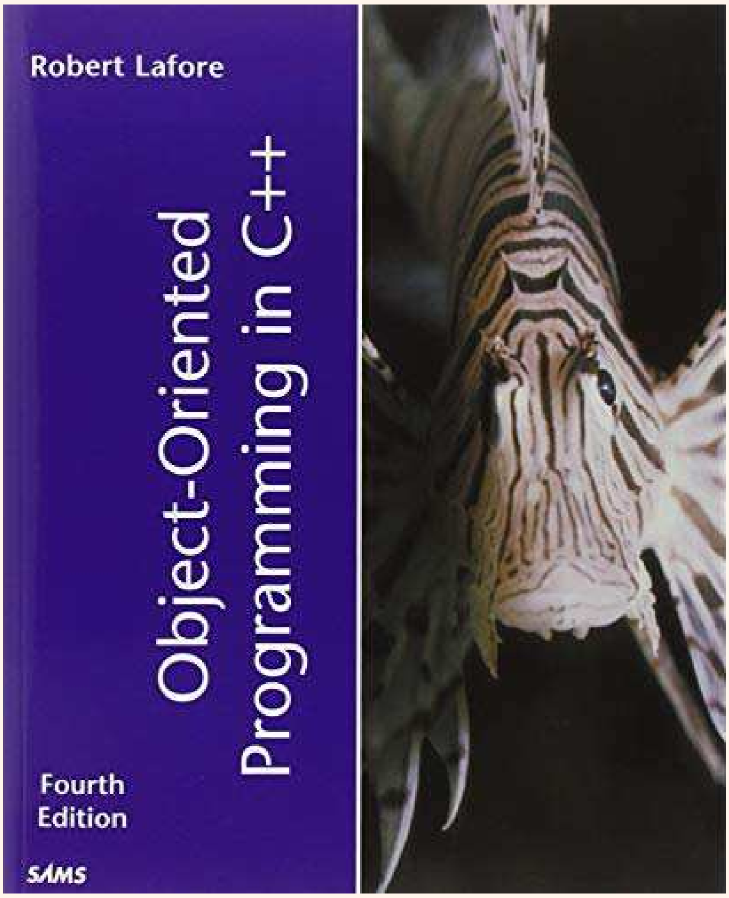


Chapter 2

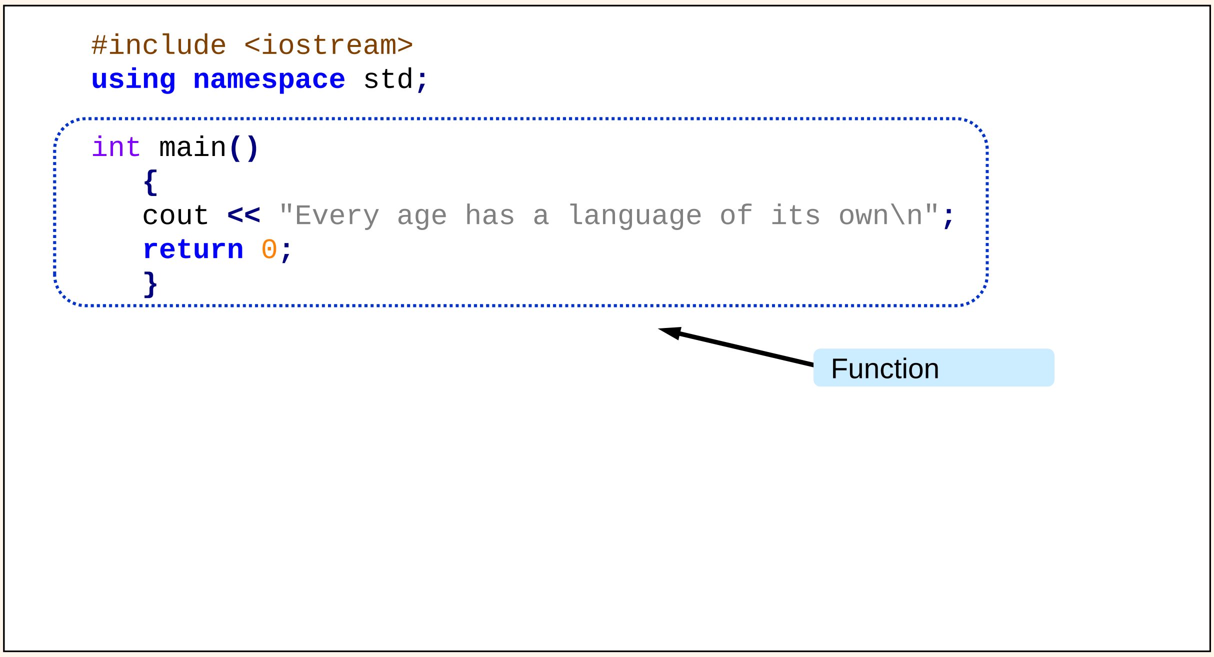
C++ Programming Basics

Animated Version

Topics

- Basic Program Construction
- Output Using *cout*
- Directives
- Comments
- Integer Variables
- Character Variables
- Input with *cin*
- Floating Point Types
- Type *bool*
- The *setw* Manipulator
- Variable Type Summary
- Type Conversion
- Arithmetic Operators
- Library Functions

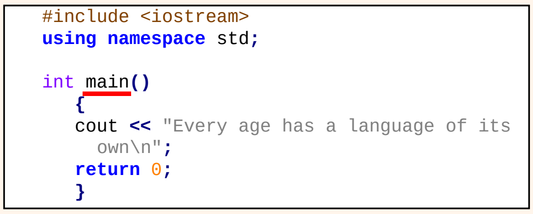
Basic Program Construction



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

int main()
   {
   cout << "Every age has a language of its own\n";
   return 0;
   }
```

Function

Functions (1)



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

int main()
   {
   cout << "Every age has a language of its
     own\n";
   return 0;
   }
```

- One of the fundamental building blocks of C++
- a function can be part of a class (member function) or can also exist independently of classes.
- Function Name
  - parentheses following the word main are the distinguishing feature of a function.
  - parentheses are used to hold function arguments
  - word `int` preceding the function name indicates that this particular function has a return value of type `int`.

Functions (2)

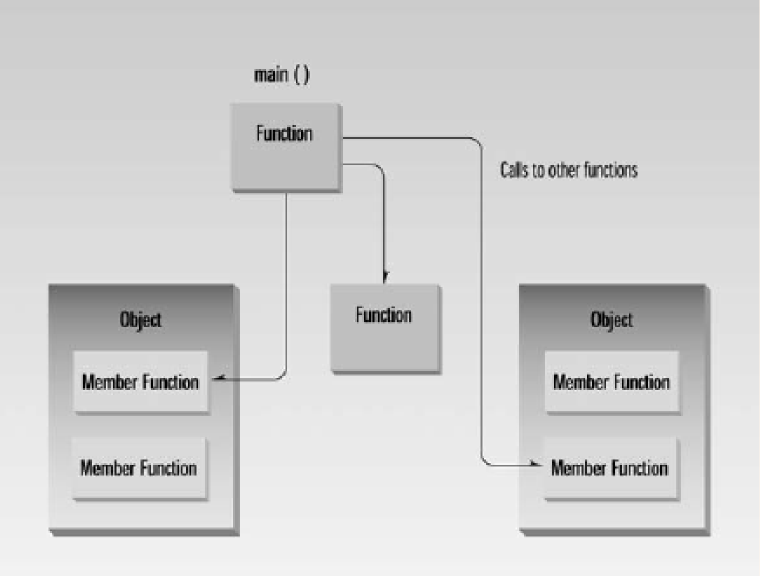
```
#include <iostream>
using namespace std;

int main()
  {
  cout << "Every age has a language of its own\n";
  return 0;
  }
```

- Braces and the Function Body
  - Every function must use pair of braces around the function body
  - a function body can consist of many statements

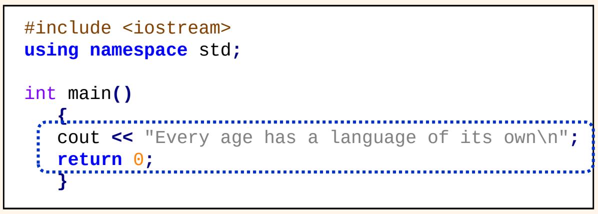
Functions (3)

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

int main()
  {
  cout << "Every age has a language of its own\n";
  return 0;
  }
```

- Always Start with `main()`
  - When you run a C++ program, the first statement executed will be at the beginning of a func-tion called main()



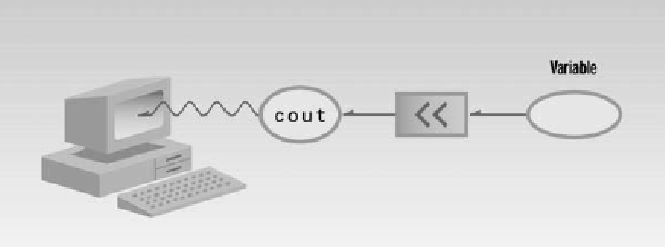
Program Statements

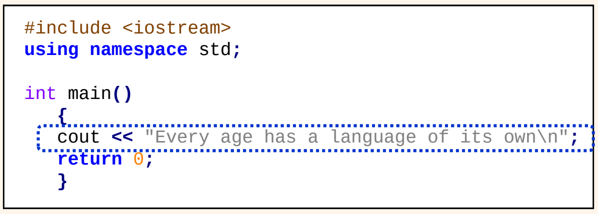


```
#include <iostream>
using namespace std;

int main()
   {
   cout << "Every age has a language of its own\n";
   return 0;
   }
```

- The majority of statements in C++ are identical to statements in C
- main() to return the value 0 to the operating system or compiler.
- return type of void is incorrect in Standard C++
- Whitespace is defined as spaces, carriage returns, linefeeds, tabs, vertical tabs, and formfeeds. These characters are invisible to the compiler.

Output Using `cout`

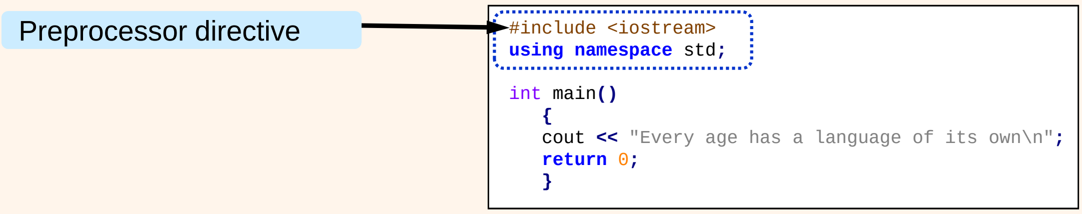




```
#include <iostream>
using namespace std;

int main()
   {
   cout << "Every age has a language of its own\n";
   return 0;
   }
```

- Pronounced “C out” - causes the phrase in quotation marks to be displayed on the screen
- Actually an *object* - predefined in C++ to correspond to the standard output stream.
  - stream is an abstraction that refers to a flow of data
- The operator `<<` is called the *insertion* or *put to* operator.
  - directs the contents of the variable on its right to the object on its left.

Directives (1)

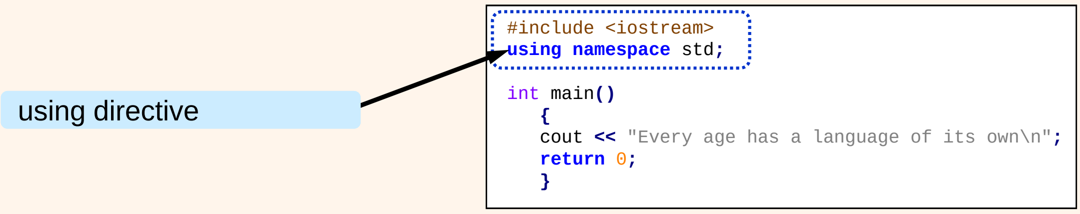


Preprocessor directive

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

int main()
   {
   cout << "Every age has a language of its own\n";
   return 0;
   }
```

- Preprocessor Directives
  - starts with a number sign (#)
  - An instruction to the compiler. A part of the compiler called the *preprocessor* deals with these directives before it begins the real compilation process.
  - `#include` tells the compiler to insert another file into your source file - header file
    - IOSTREAM - concerned with basic input/output. Without these declarations, the compiler won't recognize `cout` and will think << is being used incorrectly.

Directives (2)

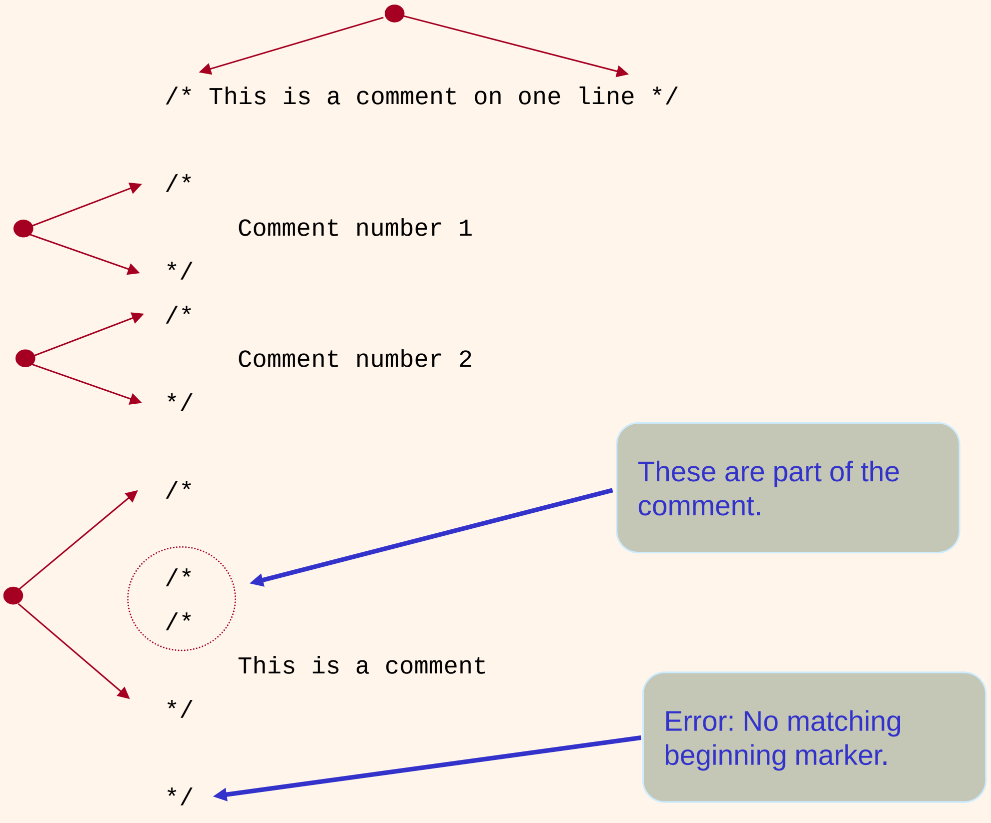


using directive

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

int main()
   {
   cout << "Every age has a language of its own\n";
   return 0;
   }
```

- Using Directives
  - C++ program can be divided into different namespaces.
    - A namespace is a part of the program in which certain names are recognized; outside of the namespace they're unknown.
  - `using namespace std;`
    - says that all the program statements that follow are within the `std` namespace.
    - If we didn't use the using directive, we would need to add the std name to many program elements.
      - `std::cout << "Every age has a language of its own.";`

Matching Comment Markers



```
/* This is a comment on one line */

/*
     Comment number 1
*/
/*
     Comment number 2
*/

/*

/*
/*
     This is a comment
*/

*/
```

These are part of the comment.

Error: No matching beginning marker.

Two Types of Comments

```
/*
     This is a comment with
     three lines of
     text.
*/
```

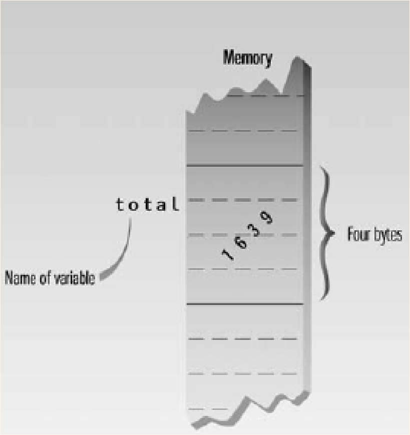
Multiline Comment

```
// This is a comment
// This is another comment
// This is a third comment
```

Single line Comments

Variables

- The most fundamental part of any language
- A variable has a symbolic name and can be given a variety of values.
- When a variable is given a value, that value is actually placed in the memory space assigned to the variable.
- Most popular languages use the same general variable types, such as integers, floating-point numbers, and characters
- The amount of memory occupied by the integer types is system dependent.
- On a 32-bit system such as Windows, an int occupies 4 bytes (which is 32 bits) of memory. This allows an int to hold numbers in the range from –2,147,483,648 to 2,147,483,647.



Integer Variables

```
// intvars.cpp
// demonstrates integer variables
#include <iostream>
using namespace std;

int main()
   {
   int var1;                   //define var1
   int var2;                   //define var2

   var1 = 20;                  //assign value to var1
   var2 = var1 + 10;           //assign value to var2
   cout << "var1+10 is ";      //output text
   cout << var2 << endl;       //output value of var2
   return 0;
   }
```

- Declaration and definition
- Variable names

-Assignment statements

Integer constant

Output:

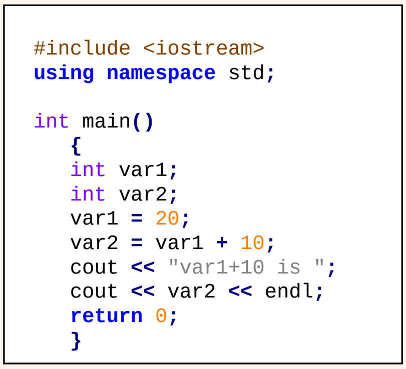
```
var1+10 is 30
```

Declarations and Definitions

- You must declare a variable before using it
  - you can place variable declarations anywhere in a program. It’s not necessary to declare variables before the first executable statement
- A declaration introduces a variable’s name (such as var1) into a program and specifies its type (such as int)
- If a declaration also sets aside memory for the variable, it is also called a definition

Variable Names

- names given to variables (andother program features) are called identifiers.
  - can use upper- and lowercase letters, and the digits from 1 to 9. You can also use the underscore (_). The first character must be a letter or underscore.
  - Length depends on compilers
  - Case sensitive

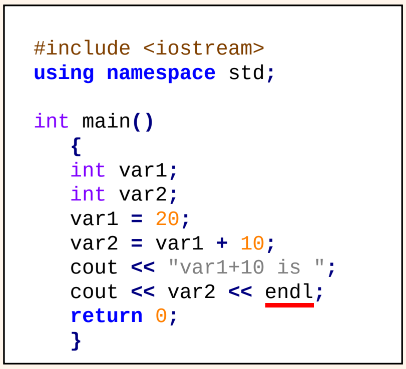


```
#include <iostream>
using namespace std;

int main()
   {
   int var1;
   int var2;
   var1 = 20;
   var2 = var1 + 10;
   cout << "var1+10 is ";
   cout << var2 << endl;
   return 0;
   }
```

- A keyword is a predefined word with a special meaning.
  - `int , return , if , while`

`endl` manipulator

- It has the same effect as sending the ‘\n’ character, but is somewhat clearer.
- It has the same effect as sending the ‘\n’ character, but is somewhat clearer.
- endl (unlike ‘\n’) also causes the output buffer to be flushed, but this happens invisibly so for most purposes the two are equivalent.



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

int main()
   {
   int var1;
   int var2;
   var1 = 20;
   var2 = var1 + 10;
   cout << "var1+10 is ";
   cout << var2 << endl;
   return 0;
   }
```

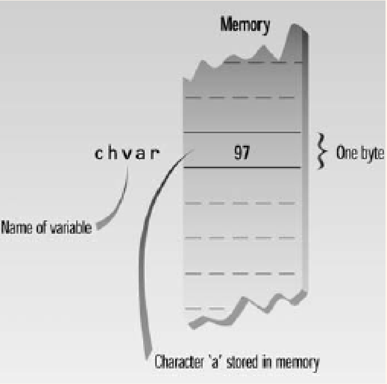
Other Types

– Size of type `int` is system dependent.

- 4 bytes in 32-bit system

– Types `long` (4 bytes) and short (2 bytes) no matter what system is used.

– `char` (1 byte) stores integers that range in to 127.



– Standard C++ provides a larger character type called `wchar_t` to handle foreign languages.

– Character constants use single quotation marks around a character, like ‘a’ and ‘b’.

- When the C++ compiler encounters such a character constant, it translates it into the corresponding ASCII code.

Character Variables

```
// charvars.cpp
// demonstrates character variables
#include <iostream>         //for cout, etc.
using namespace std;

int main()
   {
   char charvar1 = 'A';    //define char variable as character
   char charvar2 = '\t';   //define char variable as tab

   cout << charvar1;       //display character
   cout << charvar2;       //display character
   charvar1 = 'B';         //set char variable to cl
   cout << charvar1;       //display character
   cout << '\n';           //display newline character
   return 0;
   }
```

- Declaration and initialization

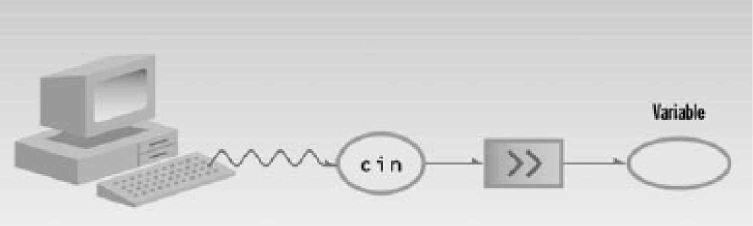
Escape Sequebces

```
Output:
A    B
```

Escape Sequences

Escape Sequence	Character
\a	Bell (beep)
\b	Backspace
\f	Formfeed
\n	Newline
\r	Return
\t	Tab
\\	Backslash
\’	Single quotation marks
\”	Double quotation marks
\xdd	Hexadecimal notation

Input Using `cin`



- Pronounced “C in” - causes the phrase in quotation marks to be displayed on the screen
- Actually an *object* - predefined in C++ to correspond to the standard input stream.
  - stream is an abstraction that refers to a flow of data
- The operator `>>` is called the *extraction* or *get from* operator
  - takes the value from the stream object on its left and places it in the variable on its right.

Demonstrating `cin` and newline

```
// fahren.cpp
// demonstrates cin, newline
#include <iostream>
using namespace std;

int main()
   {
   int ftemp;  //for temperature in fahrenheit

   cout << "Enter temperature in fahrenheit: ";
   cin >> ftemp;
   int ctemp = (ftemp-32) * 5 / 9;
   cout << "Equivalent in Celsius is: " << ctemp << '\n';
   return 0;
   }
```

- Expression
- precedence

-Variable defined at point of use

Output:
```
Enter temperature in fahrenheit: 212
Equivalent in Celsius is: 100
```

Floating Point Types (1)

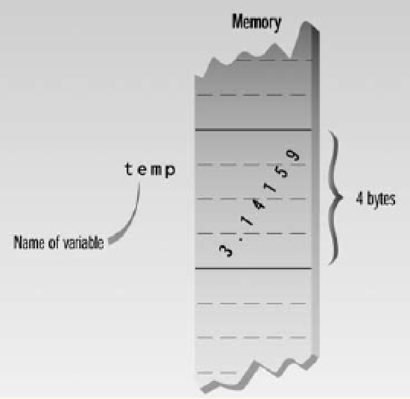
- represent numbers with a decimal place (real numbers) - like 3.1415927, 0.0000625, and –10.2.
- Three kinds: float , double , and long double
- float: 4 bytes (32 bits), range about 3.4x10^{-38} to 3.4x10 ^{38}

```
// circarea.cpp
// demonstrates floating point variables
#include <iostream>                         //for cout, etc.
using namespace std;

int main()
   {
   float rad;                               //variable of type
     float
   const float PI = 3.14159F;               //type const float

   cout << "Enter radius of circle: ";      //prompt
   cin >> rad;                              //get radius
   float area = PI * rad * rad;             //find area
   cout << "Area is " << area << endl;      //display answer
   return 0;
   }
```

- const qualifier
- floating point constant

Output:

```
Enter radius of circle: 0.5
Area is 0.785398
```



Floating Point Types (2)

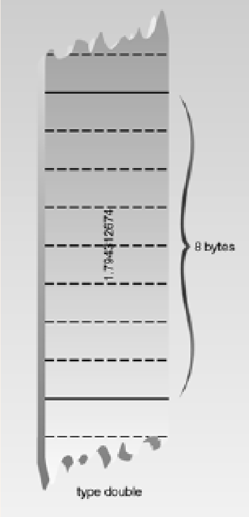
- double: requires 8 bytes of storage; range from 1.7x10 $^{-308}$ to 1.7x10 ^{308}; 15 digits precision.
- long double: compiler-dependent but is often the same as double.
- The exponential number

  6.35239E–5 is equivalent to

  0.0000635239 in decimal notation.

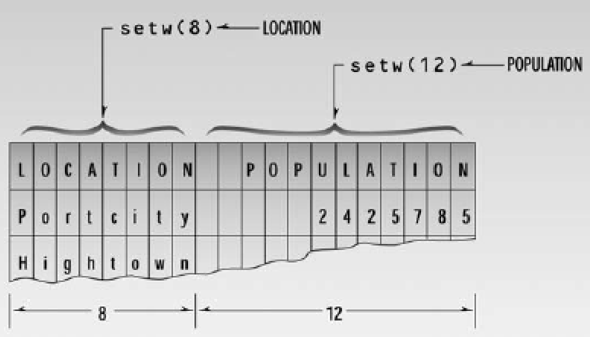
  This is the same as 6.35239

  times 10^{-5} .



The setw Manipulator

- The setw manipulator cause string) that follows it in the st within a field n characters wi argument to setw(n).


setw(8) LOCATION
setw(12) POPULATION
L O C A T I O N P O P U L A T I O N
P o r t c i t y 2 4 2 5 7 8 5
H i g h t o w n
8
12

```
// width1.cpp
// demonstrates need for setw manipulator
#include <iostream>
using namespace std;

int main()
   {
   long pop1=2425785, pop2=47, pop3=9761;

   cout << "LOCATION " << "POP." << endl
        << "Portcity " << pop1 << endl
        << "Hightown " << pop2 << endl
        << "Lowville " << pop3 << endl;
   return 0;
   }
```

Output:

```
LOCATION POP.
Portcity 2425785
Hightown 47
Lowville 9761
```

```
// width2.cpp
// demonstrates setw manipulator
#include <iostream>
#include <iomanip>     // for setw
using namespace std;

int main()
   {
   long pop1=2425785, pop2=47, pop3=9761;

   cout << setw(8) << "LOCATION" << setw(12)
        << "POPULATION" << endl
        << setw(8) << "Portcity" << setw(12) << pop1 << endl
        << setw(8) << "Hightown" << setw(12) << pop2 << endl
        << setw(8) << "Lowville" << setw(12) << pop3 << endl;
   return 0;
   }
```

IOMANIP header file

Output:

```
LOCATION POPULATION
Portcity     2425785
Hightown          47
Lowville        9761
```

Variable Type Summary

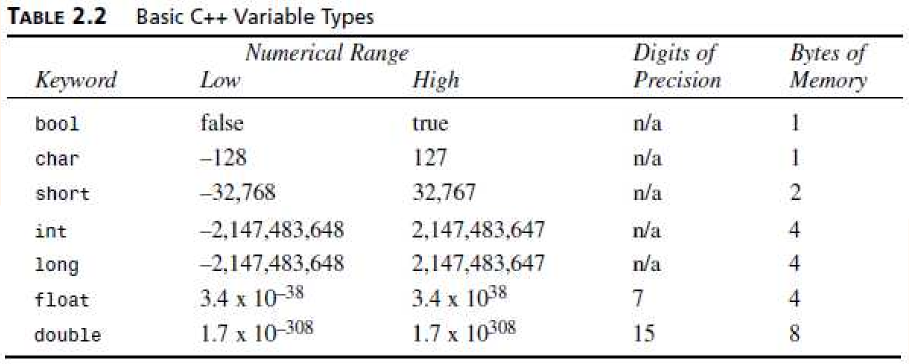


TABLE 2.2 Basic C++ Variable Types

	Numerical Range		*Digits of*	*Bytes of*
Keyword	*Low*	*High*	*Precision*	*Memory*
`bool`	false	true	n/a	1
`char`	–128	127	n/a	1
`short`	–32,768	32,767	n/a	2
`int`	–2,147,483,648	2,147,483,647	n/a	4
`long`	–2,147,483,648	2,147,483,647	n/a	4
`float`	3.4×10^{-38}	3.4×10^{38}	7	4
`double`	1.7×10^{-308}	1.7×10^{308}	15	8

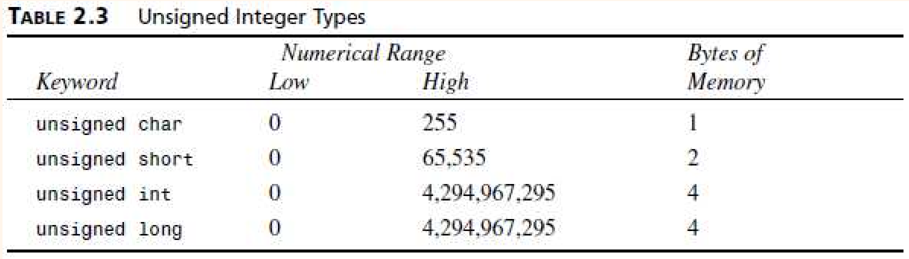


TABLE 2.3 Unsigned Integer Types

	Numerical Range		*Bytes of*
Keyword	*Low*	*High*	*Memory*
`unsigned char`	0	255	1
`unsigned short`	0	65,535	2
`unsigned int`	0	4,294,967,295	4
`unsigned long`	0	4,294,967,295	4

Testing Signed and Unsigned Integers

```
// signtest.cpp
// tests signed and unsigned integers
#include <iostream>
using namespace std;

int main()
   {
   int signedVar = 1500000000;           //signed
   unsigned int unsignVar = 1500000000;  //unsigned

   signedVar = (signedVar * 2) / 3;  //calculation exceeds range
   unsignVar = (unsignVar * 2) / 3;  //calculation within range

   cout << "signedVar = " << signedVar << endl;  //wrong
   cout << "unsignVar = 
   return 0;
   }
```

Output:

```
signedVar = -431,655,765
unsignVar = 1,000,000,000
```

Type Conversion (1)

```
// mixed.cpp
// shows mixed expressions
#include <iostream>
using namespace std;

int main()
   {
   int count = 7;
   float avgWeight = 155.5F;

   double totalWeight = count * avgWeight;
   cout << "totalWeight=" << totalWeight << endl;
   return 0;
   }
```

A variable of type int is multiplied by a variable of type float to yield a result of type double <- automatic conversion

Type Conversion (2)

- Automatic (implicit) conversion:
  - A lower order type can be automatically converted to a higher one

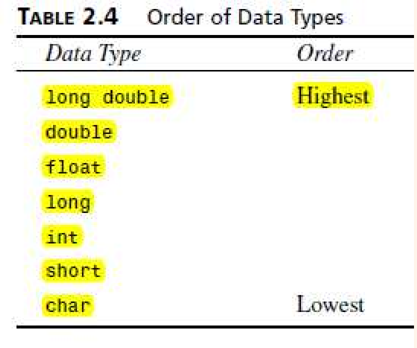
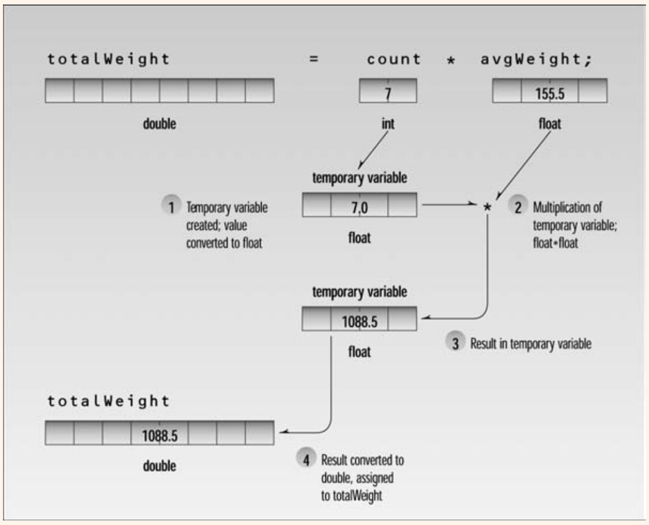


TABLE 2.4 Order of Data Types

Data Type	*Order*
`long double`	Highest
`double`	
`float`	
`long`	
`int`	
`short`	
`char`	Lowest


totalWeight = count * avgWeight;
7
155.5
double
int
float
temporary variable
1 Temporary variable created; value converted to float
7.0
float
*
2 Multiplication of temporary variable; float*float
temporary variable
1088.5
float
3 Result in temporary variable
totalWeight
1088.5
double
4 Result converted to double, assigned to totalWeight

Type Conversion (3)

- Type casting (explicit conversion):
  - convert a value from one type to another in a situation where the compiler will not do it automatically or without complaining.
  - C++ style:
    - `aCharVar = static_cast<char>(anIntVar);`
  - Old styles:
    - `aCharVar = (char)anIntVar;`
    - `aCharVar = char(anIntVar);`

Type Conversion (4)

```
// cast.cpp
// tests signed and unsigned integers
#include <iostream>
using namespace std;

int main()
   {
   int intVar = 1500000000;                 //1,500,000,000
   intVar = (intVar * 10) / 10;             //result too large
   cout << "intVar = " << intVar << endl;   //wrong answer

   intVar = 1500000000;                     //cast to double
   intVar = (static_cast<double>(intVar) * 10) / 10;
   cout << "intVar = " << intVar << endl;   //right answer
   return 0;
   }
```

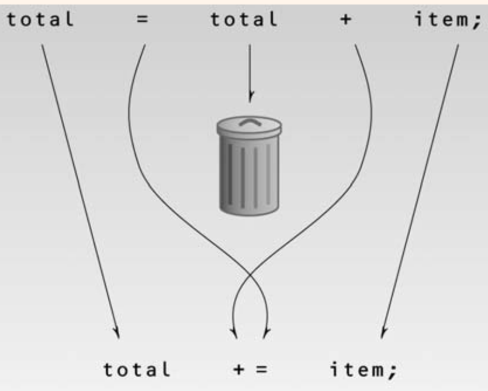
Output:

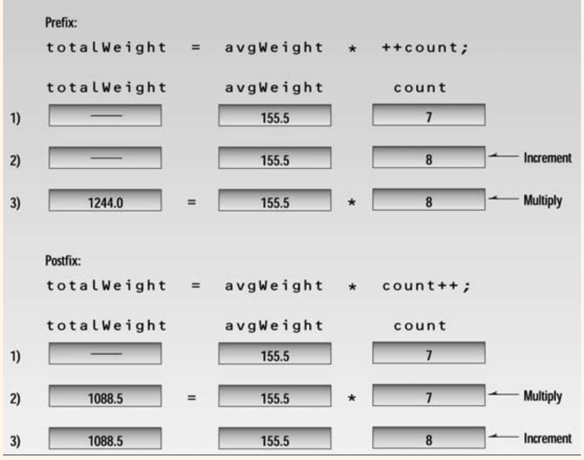
```
intVar = 211509811
intVar = 1500000000
```

Operators (1)

- Arithmetic Operators:
- Common operators: + , - , * , and /
- Remainder Operator: % (works only on integer)
- Arithmetic Assignment Operators:

  `+=,  -=,  *=,  /=,  and  %=`
- Increment and decrement operators: ++, --





Remainder and Arithmetic Assignment Operator (example)

```
// remaind.cpp
// demonstrates remainder operator
#include <iostream>
using namespace std;

int main()
   {
   cout <<  6 % 8 << endl    // 6
        <<  7 % 8 << endl    // 7
        <<  8 % 8 << endl    // 0
        <<  9 % 8 << endl    // 1
        << 10 % 8 << endl;   // 2
   return 0;
   }
```

```
// assign.cpp
// demonstrates arithmetic assignment operators
#include <iostream>
using namespace std;

int main()
   {
   int ans = 27;

   ans += 10;              //same as: ans = ans + 10;
   cout << ans << ", ";
   ans -= 7;               //same as: ans = ans - 7;
   cout << ans << ", ";
   ans *= 2;               //same as: ans = ans * 2;
   cout << ans << ", ";
   ans /= 3;               //same as: ans = ans / 3;
   cout << ans << ", ";
   ans %= 3;               //same as: ans = ans % 3;
   cout << ans << endl;
   return 0;
   }
```

Output:

```
37, 30, 60, 20, 2
```

Increment Operator (example)

```
// increm.cpp
// demonstrates the increment operator
#include <iostream>
using namespace std;

int main()
   {
   int count = 10;

   cout << "count=" << count << endl;    //displays 10
   cout << "count=" << ++count << endl;  //displays 11 (prefix)
   cout << "count=" << count << endl;    //displays 11
   cout << "count=" << count++ << endl;  //displays 11 (postfix)
   cout << "count=" << count << endl;    //displays 12
   return 0;
   }
```

Output:

```
count=10
count=11
count=11
count=11
count=12
```

Library Functions and Header Files (1)

- Perform file access, mathematical computations, and data conversion, among other things.
- To use a library function like `sqrt()`, you must link the library file that contains it to your program.
- The functions in your source file need to know the names and types of the functions and other elements in the library file. They are given this information in a header file. Each header file contains information for a particular group of functions.

Library Functions and Header Files (2)

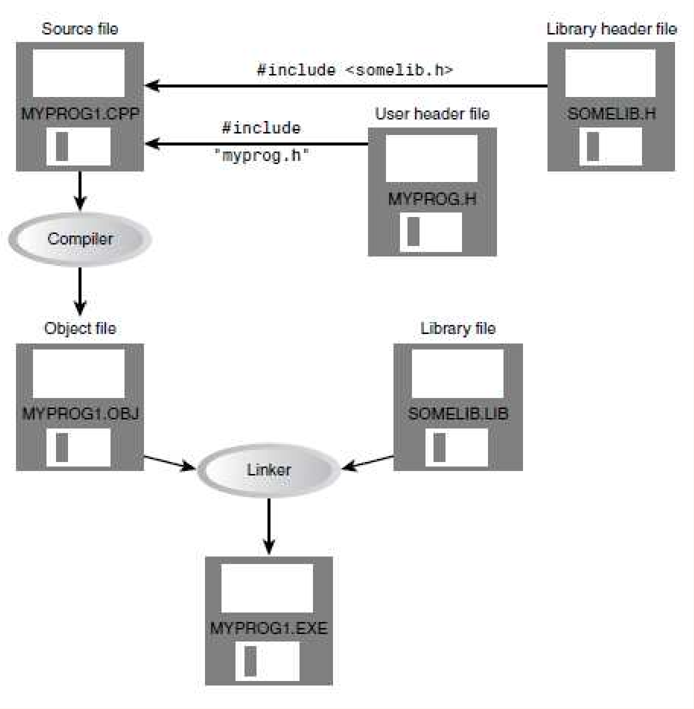
- Two Ways to Use # include:
- `#include <iostream>`
  - files in the standard INCLUDE directory
- `#include “myheader.h”`
  - instruct the compiler to begin its search for the header file in the current directory

```
// sqrt.cpp
// demonstrates sqrt() library function
#include <iostream>             //for cout, etc.
#include <cmath>                //for sqrt()
using namespace std;

int main()
   {
   double number, answer;       //sqrt() requires type double

   cout << "Enter a number: ";
   cin >> number;               //get the number
   answer = sqrt(number);       //find square root
   cout << "Square root is "
     << answer << endl;           //display it
   return 0;
   }
```

Output:

```
Enter a number: 1000
Square root is 31.622777
```

Library Functions and Header Files (3)



Summary (1)

- Function is a major building block of C++ programs
- A function named main() is always the first one executed when a program is executed.
- A function is composed of statements, which tell the computer to do something.
  - Each statement ends with a semicolon. A statement may contain one or more expressions, which are sequences of variables and operators that usually evaluate to a specific value.
- Output is most commonly handled in C++ with the cout object and << insertion operator, which together cause variables or constants to be sent to the standard output device—usually the screen.
- Input is handled with cin and the extraction operator >> , which cause values to be received from the standard input device—usually the keyboard.

Summary (2)

- Various data types are built into C++:
  - char, int, long, and short are the integer types and
  - float , double , and long double are the floating-point types. All of these types are signed.
- Unsigned integer types, signaled by the keyword unsigned, don’t hold negative numbers - hold positive ones twice as large.
- Type bool is used for Boolean variables and can hold only true or false .
- The const keyword stipulates that a variable’s value will not change in the course of a program.
- C++ employs the usual arithmetic operators + , - , * , and / . In addition, the remainder operator, % returns the remainder of integer division.
- The arithmetic assignment operators += , +- ,
  - perform an arithmetic operation and an assignment simultaneously.
- The increment and decrement operators ++ and -- increase or decrease a variable by 1.

Summary (3)

- Preprocessor directives consist of instructions to the compiler, rather than to the computer.
  - The #include directive tells the compiler to insert another file into the present source file, and
  - the #define directive tells it to substitute one thing for another.
  - The using directive tells the compiler to recognize names that are in a certain namespace.
- If you use a library function in your program, the code for the function is in a library file, which is automatically linked to your program.
  - A header file containing the function’s declaration must be inserted into your source file with an #include statement.