جامعة المنوفية

MENOUFIA UNIVERSITY
FACULTY OF COMPUTERS AND INFORMATION

First Year (First Semester)

Introduction to Computer

LECTURE Six

Dr. Hamdy M. Mousa

Introduction to C++

Introduction

- ***A computer is a device capable of performing computations and making logical decisions at speeds millions (even billions) of times faster than human beings can.***
- Computers process **data** under the control of sets of instructions called **computer programs**.
  - These programs guide the computer through orderly sets of actions specified by people called **computer programmers**.

Introduction

- Programmers write instructions in various programming languages,
  - some directly understandable by computers and others requiring intermediate **translation** steps.
- Computer languages may be divided into **three general types:**
  - Machine languages
  - Assembly languages
  - High-level languages

Languages

- **Machine language**
  - “Natural language” of computer component
  - Machine dependent
  - Machine-language programming was simply too slow, tedious and error-prone for most programmers.
- **Assembly language**
  - English-like abbreviations represent computer operations
  - Translator programs convert to machine language
- **High-level language**
  - Allows for writing more “English-like” instructions
    - Contains commonly used mathematical operations
  - Compiler convert to machine language
- Interpreter
  - Execute high-level language programs without compilation

Machine Languages

- Machine languages generally consist of strings of numbers (1s and 0s) that instruct computers to perform their most elementary operations one at a time.
- Machine languages are **machine dependent** (i.e., a particular machine language can be used on only one type of computer).
- Any computer can directly understand only its own **machine language**.

Ex.:

```
+1300042774
+1400593419
+1200274027
```

- Machine-language programming was simply too slow, tedious and error-prone for most programmers.

Assembly Languages

- programmers began using **English-like abbreviations** to represent elementary operations.
  - These abbreviations formed the basis of assembly languages .
  - **Translator programs** called **assemblers** were developed to convert early assembly-language programs to machine language at computer speeds.

Ex.:

```
load basepay
add overpay
store grosspay
```

- Although such **code is clearer** to humans, it is **incomprehensible to computers** until translated to machine language.
- Programmers still had to use many instructions to accomplish even the simplest tasks.

High-Level Languages

- To speed the programming process, **high-level languages** were developed in which single statements could be written to accomplish substantial tasks.
- Translator programs called **compilers** convert high-level language programs into machine language.
- High-level languages allow programmers to write instructions that look almost like everyday English and contain commonly used mathematical notations.

Ex.:

```
grossPay = basePay + overTimePay;
```

- The process of compiling a **high-level language program into machine language** can take a considerable amount of computer time.
- **Interpreter** programs were developed to execute high-level language programs directly, although much more slowly.

History of C and C++

- Because C is a standardized, **hardware-independent**, widely available language, applications written in C often can be run with little or no modification on a wide range of computer systems.

- C++, an extension of C ,was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories.
  - It provides capabilities for object-oriented programming.

History of C and C++

- **Objects** are essentially reusable software **components** that model items in the real world.
- Software developers are discovering that using a modular, object-oriented design and implementation approach can make them much more productive than they can be with previous popular programming techniques.
- Object-oriented programs are easier to understand, correct and modify.

C++ Standard Library

- C ++programs consist of pieces called classes and functions .
- most C++ programmers take advantage of the rich **collections of existing classes and functions** in the **C++ Standard Library** .

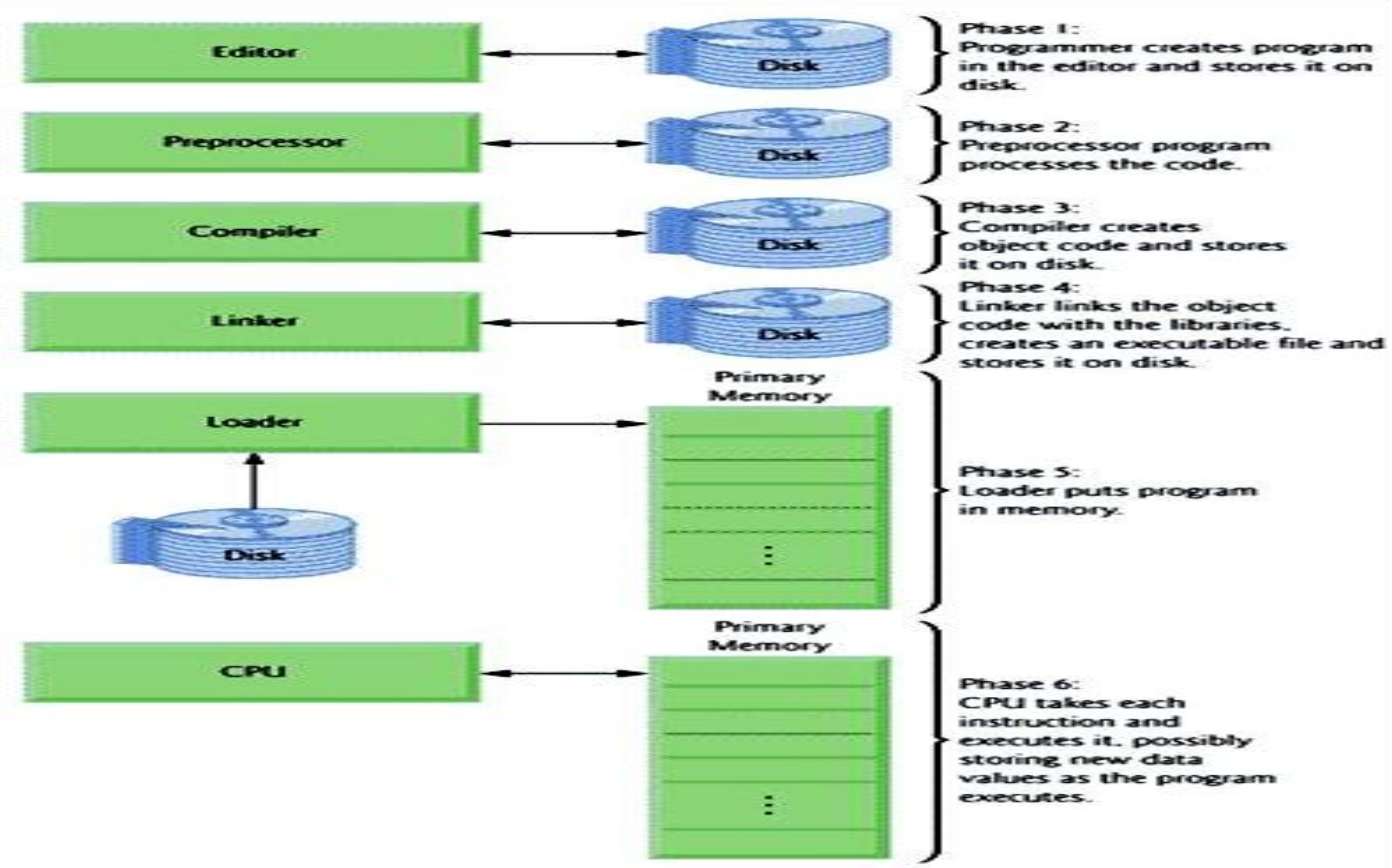
- The standard class libraries generally are provided by compiler vendors.
  - Many special-purpose class libraries are supplied by independent software vendors.

C++ Development Environment

- The steps in creating and executing a C++ application using a C++ development environment.
- C++ systems generally consist of three parts:
  - Program development environment,
  - The language
  - The C++ Standard Library.
- C++ programs typically go through **six phases**:

  E**dit**, preprocess, compile, link, load and execute.

C++ Environment



Phase 1:
Programmer creates program in the editor and stores it on disk.

Phase 2:
Preprocessor program processes the code.

Phase 3:
Compiler creates object code and stores it on disk.

Phase 4:
Linker links the object code with the libraries, creates an executable file and stores it on disk.

Phase 5:
Loader puts program in memory.

Phase 6:
CPU takes each instruction and executes it, possibly storing new data values as the program executes.

Phase 1: Creating a Program

- Phase 1 consists of editing a file with an editor program (normally known simply as an **editor**).
- You type a C++ program (typically referred to as **source code**) using the editor, make any necessary corrections and save the program on a secondary storage device, such as your hard drive.
- C++ source code file names often end with the .cpp, .cxx, .cc or .C extensions (note that C is in uppercase) which indicate that a file contains C++ source code.

Phases 2 and 3: Preprocessing and Compiling a C++ Program

- In phase 2, the programmer gives the command to **compile** the program.
  - In a C++ system, a **preprocessor** program executes automatically before the compiler's translation phase begins.
  - The C++ preprocessor obeys commands called **preprocessor directives**, which indicate that certain manipulations are to be performed on the program before compilation. These manipulations usually **include other text files to be compiled and perform various text replacements**.
- In phase 3, the compiler translates the C++ program into machine-language code (also referred to as object code).

Phase 4: Linking

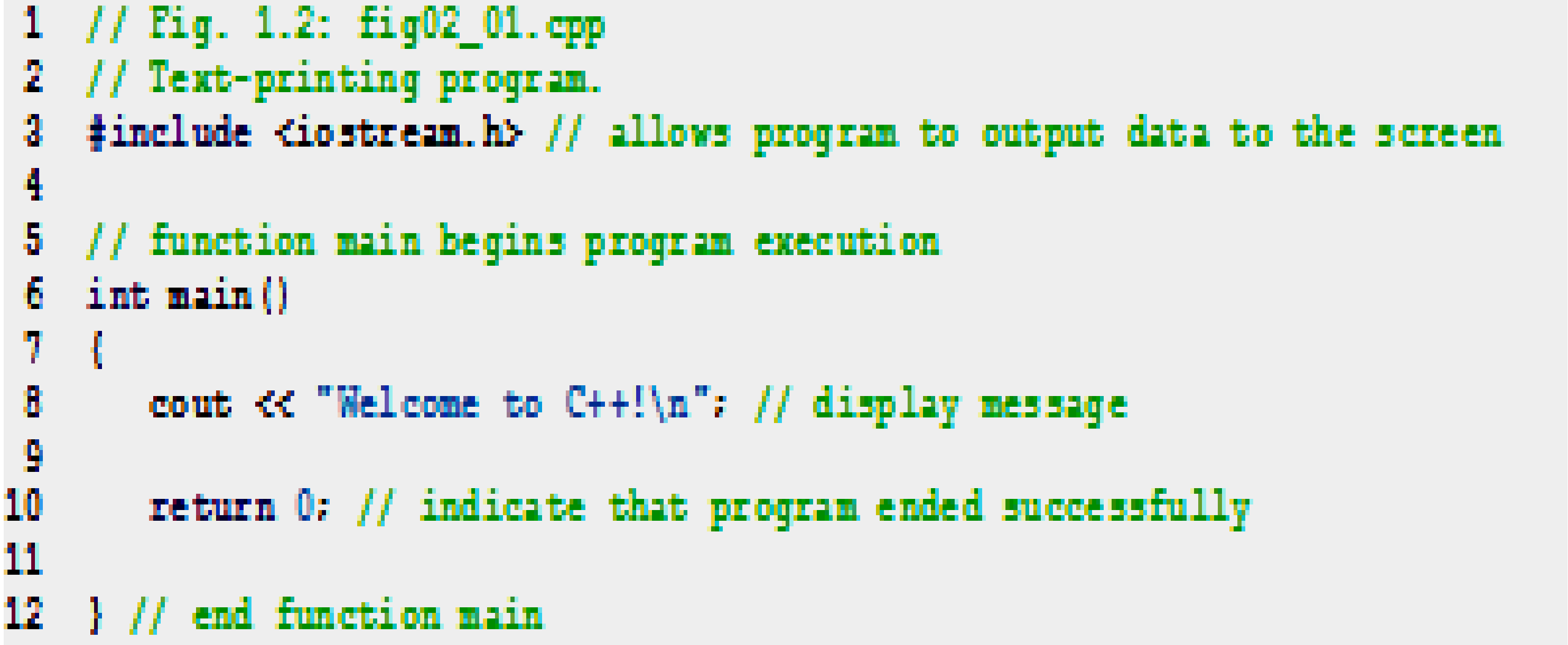
- Phase 4 is called linking. C++ programs typically contain references to functions and data defined elsewhere, such as in the standard libraries or in the private libraries of groups of programmers working on a particular project.
- The object code produced by the C++ compiler typically contains "holes" due to these missing parts.
- A **linker** links the object code with the code for **the missing functions to produce** an **executable image** (with no missing pieces).
- If the program compiles and links correctly, an executable image is produced.

Phase 5 & 6 : Loading & Execution

- ***Phase 5: Loading***
  - Phase 5 is called loading. Before a program can be executed, it must first be placed in memory.
  - This is done by the **loader**, which takes the executable image from disk and transfers it to memory. Additional components from shared libraries that support the program are also loaded.
- ***Phase 6: Execution***
  - Finally, the computer, under the control of its CPU, **executes** the program one instruction at a time.

First Program in C++

- ***Printing a Line of Text***



```
// Fig. 1.2: fig02_01.cpp
// Text-printing program.
#include <iostream.h> // allows program to output data to the screen

// function main begins program execution
int main()
{
   cout << "Welcome to C++!\n"; // display message

   return 0; // indicate that program ended successfully

} // end function main
```

- Output:

```
Welcome to C++!
```

Comment

- // fig02_01.cpp// Text-printing program.each begin with ***//***, indicating that the remainder of each line is a **comment**.
  - Programmers insert comments to document programs and also help people read and understand them.
  - Comments do not cause the computer to perform any action when the program is run they are ignored by the C++ compiler and do not cause any machine-language object code to be generated.

#include

- #include <iostream> // allows program to output data to the screen
  - is a **preprocessor directive**, which is a message to the C++ preprocessor Lines that begin with # are processed by the preprocessor before the program is compiled.
  - This line notifies the preprocessor to include in the program the contents of the **input/output stream header file <iostream>**.
  - This file must be included for any program that outputs data to the screen or inputs data from the keyboard using C++-style stream input/output.

int main()

- int main() is a part of every C++ program.
  - The parentheses **{ }** after main indicate that main is a program **building block** called a **function**.
  - C++ programs typically consist of one or more functions and classes
  - C++ programs begin executing at function main, even if main is not the first function in the program.
  - The keyword **int** to the left of main indicates that main "returns" an integer value.

cout <<

- **cout << "Welcome to C++!\n"; // display message**
  - instructs the computer to **perform an action** to print the **string** of characters contained between the double quotation marks.
  - The **<<** operator is referred to the **stream insertion operator**.
  - The backslash (****) is called an **escape character**.

    It indicates that a "special" character is to be output. When a backslash is encountered in a string of characters, the next character is combined with the backslash to form an **escape sequence**.

Escape Sequence

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

return

return 0; ***// indicate that program ended successfully***

- is one of several means we will use to **exit a function**.
- When the return statement is used at the end of main, as shown here, the value 0 indicates that the program has terminated successfully.

Whitespace

- We mentioned that the end of a line isn’t important to a C++ compiler.
- Actually, the compiler ignores whitespace almost completely.
  - ***Whitespace is defined as spaces, carriage returns, linefeeds,* tabs, vertical tabs, and form feeds.**
  - **These characters are invisible to the compiler.**

```
#include <iostream>
using namespace std;
int main()
{
cout << “Every age has a language of its own\n”;
return 0;
}
```

```
#include <iostream>
using
namespace std;
int main () { cout
<<
“Every age has a language of its own\n”
; return
0;}
```

String Constants

- The phrase in quotation marks, “Every age has a language of its own\n”, is an example of a *string constant.*

Directives

- The two lines that begin the program are *directives.*
- *The first is a preprocessor directive*, and the second is a using *directive.*
  - They're **not part** of the basic C++ language, but they're **necessary** anyway

Preprocessor Directive

- The preprocessor directive **#include** tells the compiler to insert another file into your source file.
- In effect, the **#include** directive is replaced by the contents of the file indicated.
  - Using an #include directive to insert another file into your source file
    - is similar to pasting a block of text into a document with your word processor.

Preprocessor Directive

- the preprocessor directive **#include** tells the compiler to add the source file **IOSTREAM** to the source file before compiling.
- **IOSTREAM** is an example of a ***header file*** *(sometimes called an include file).*
  - *It's concerned with basic input/output operations*, and
  - contains declarations that are needed by the **cout** identifier and the **<<** operator.

Directive

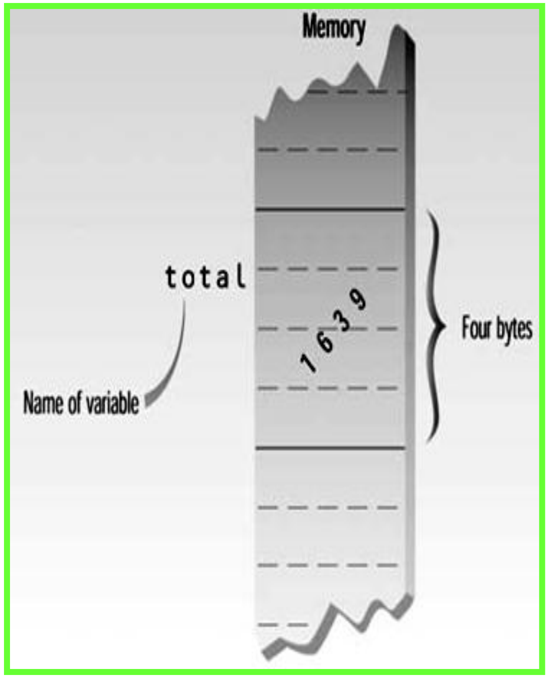
- A **namespace** is a part of the program in which certain names are recognized; outside of the namespace they're unknown.

The directive 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.
  - For example, in the program we'd need to say
  - **std::cout <<** "Every age has a language of its own.";

Variables

- Variables are the most fundamental part of any language.
  - A variable has a symbolic name and can be given a variety of values.
  - Variables are located in particular places in the computer's memory.
  - When a variable is given a value, that value is actually placed in the memory space assigned to the variable.



Identifiers

- The names given to variables (and other program features) are called *identifiers.*

- **Rules for writing identifiers:**
  - You 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**.
  - You can't use a C++ **keyword** as a variable name.
    - A *keyword is a predefined word with a special* meaning.

Statements

Assignment statements:

```
var1 = 20;      // The number 20 is an integer constant.
var2 = var1 + 10;
```

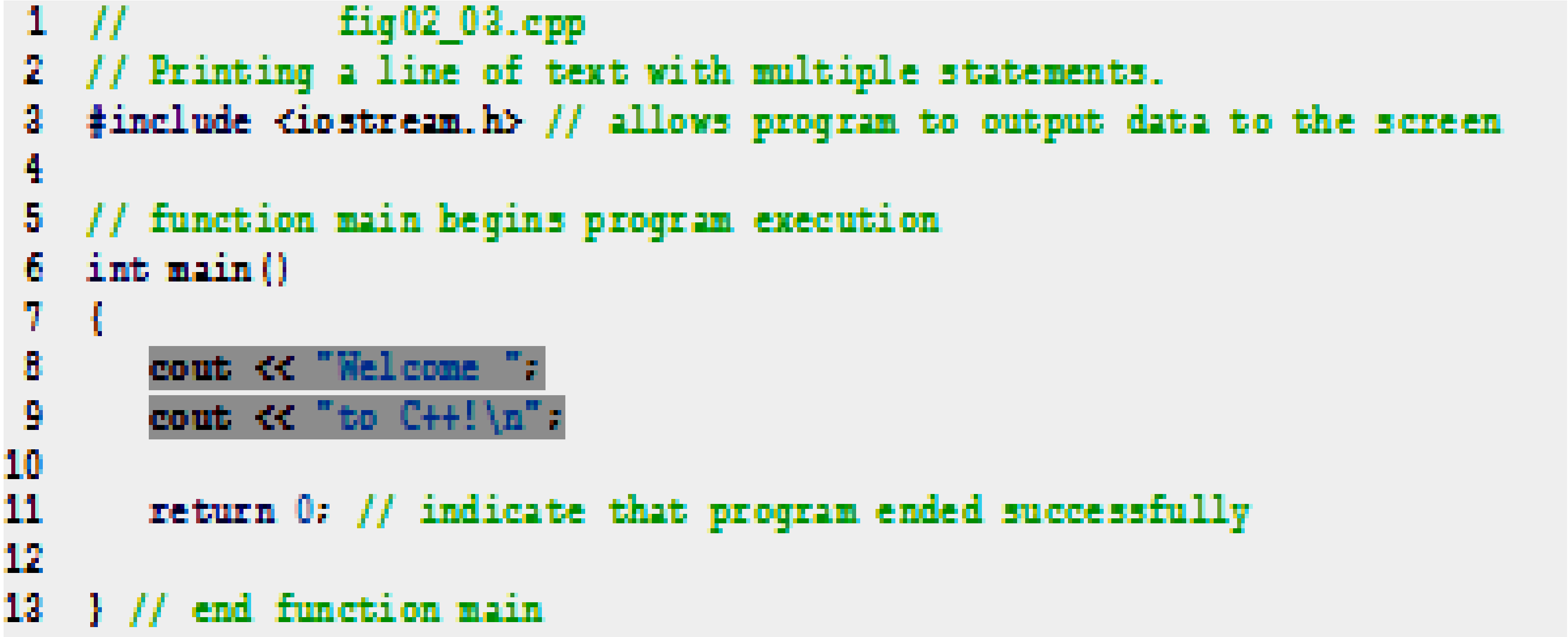
Expressions

- Any arrangement of variables, constants, and operators that specifies a computation is called an ***expression.***

alpha+12

(alpha-37)*beta/2

Printing Multiple Statements



```
//              fig02_03.cpp
// Printing a line of text with multiple statements.
#include <iostream.h> // allows program to output data to the screen

// function main begins program execution
int main()
{
    cout << "Welcome ";
    cout << "to C++!\n";

    return 0; // indicate that program ended successfully

} // end function main
```

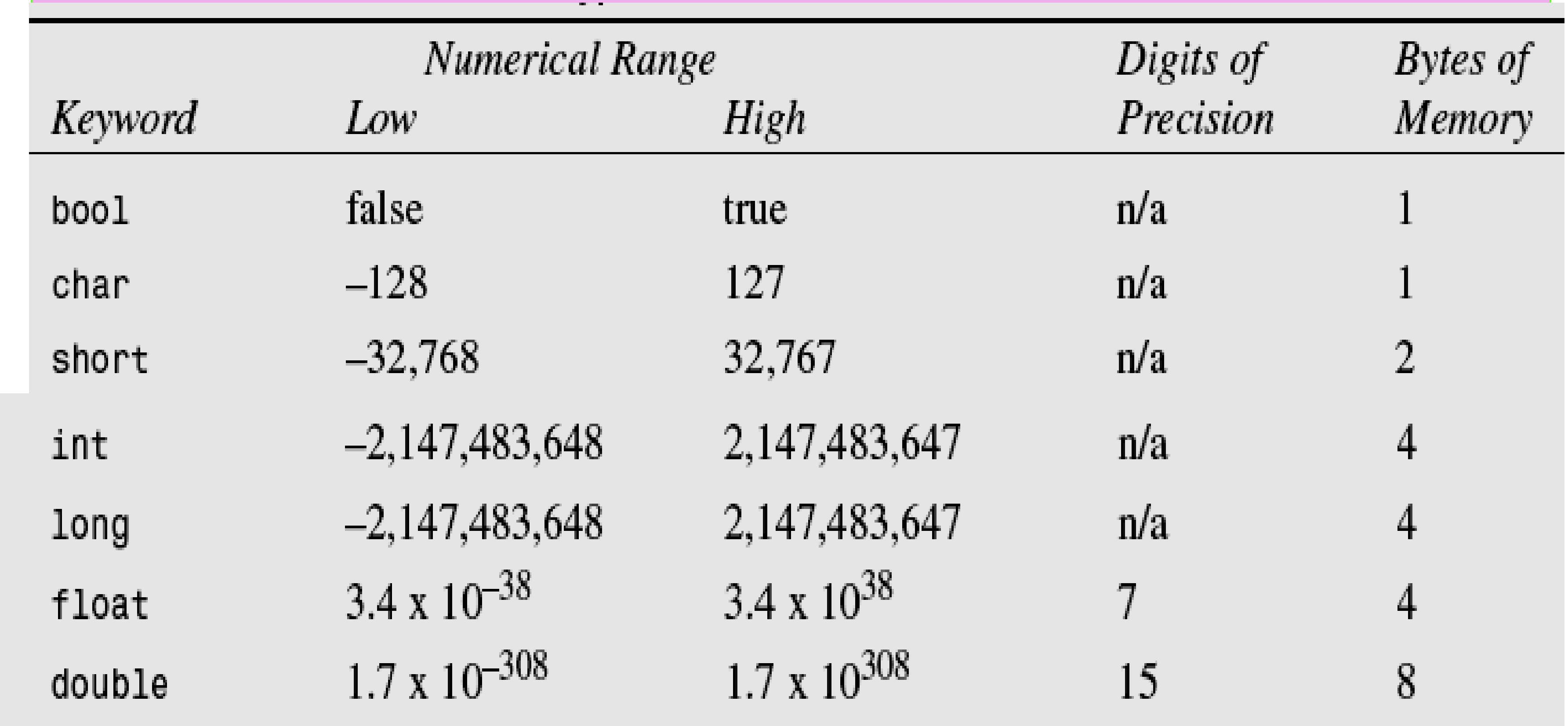
Output:

```
Welcome to C++!
```

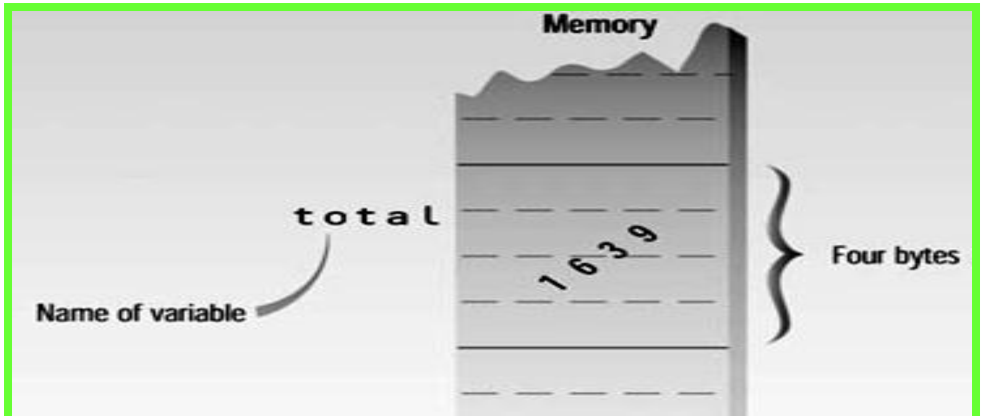
Declarations

```
// 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;
}
```

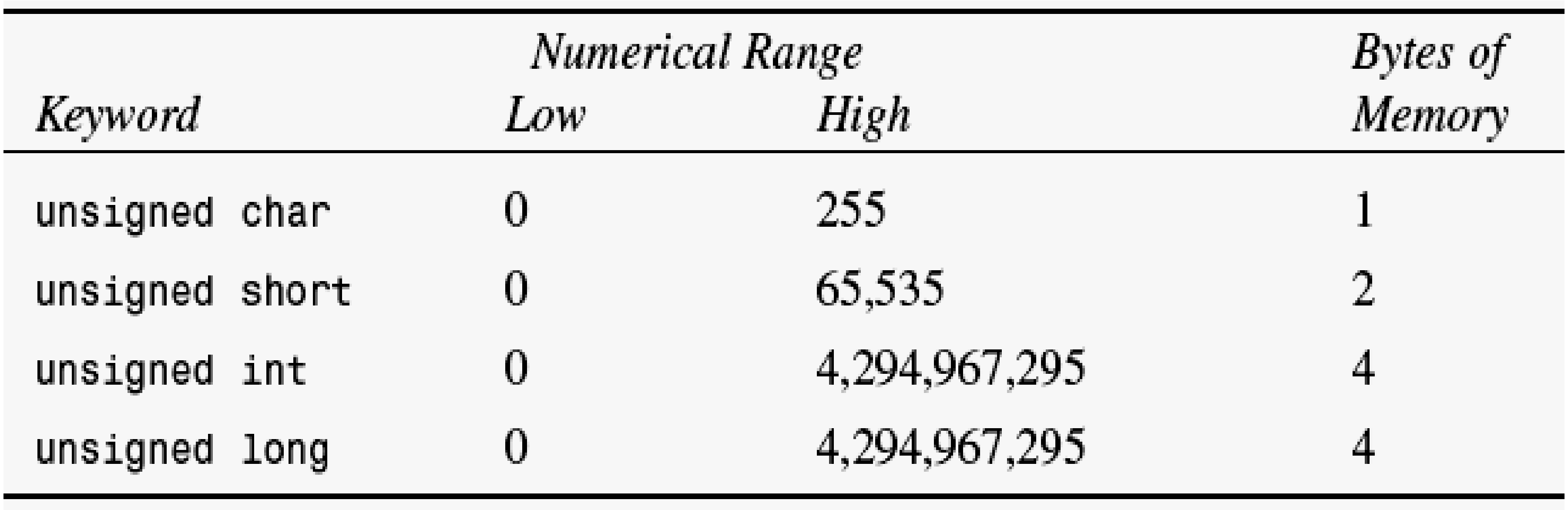
Basic C++ Variable Types



	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`	$3.4 \text{ x } 10^{-38}$	$3.4 \text{ x } 10^{38}$	7	4
`double`	$1.7 \text{ x } 10^{-308}$	$1.7 \text{ x } 10^{308}$	15	8



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

- To change an integer type to an unsigned type, precede the data type keyword with the keyword unsigned. For example, an unsigned variable of type char would be defined as:

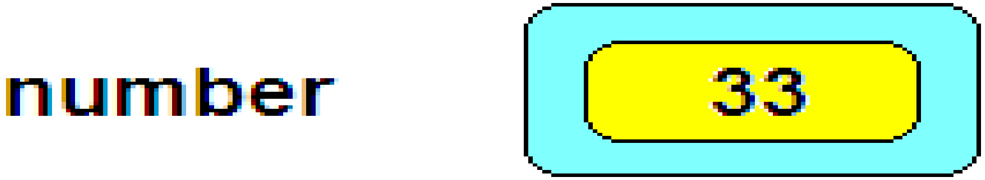
```
unsigned char ucharvar;
```

Arithmetic

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	f + 7	**f + 7**
Subtraction	-	p - c	**p - c**
Multiplication	*	bm or b · m	**b * m**
Division	/	x ÷ y or x/y	**x / y**
Modulus	%	r mod s	**r % s**

Memory Concepts

- Variable names such as number1, number2 and sum actually correspond to locations in the computer's memory.
- Every variable has a name, a type, a size and a value.



Precedence of arithmetic operators

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Braces (Parentheses)	Evaluated first. If the Braces are nested, the expression in the inner most pair is evaluated first.
* / %	Multiplication Division Modulus	Evaluated second.
+ -	Addition Subtraction	Evaluated last.

Decision Making

Standard algebraic equality or relational operator	C++ equality or relational operator	Sample C++ condition	Meaning of C++ condition
Relational operators			
> <	> < >= <=	x > y x < y x >= y x <= y	x is greater than y x is less than y x is greater than or equal to y x is less than or equal to y
Equality operators			
=	== !=	x == y x != y	x is equal to y x is not equal to y

Character Variables

```
// 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 char constant
  cout << charvar1;              //display character
  cout << ‘\n’;                  //display newline character
  return 0;
}
```

Example: Fahrenheit to Celsius

```
// 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;
}
```