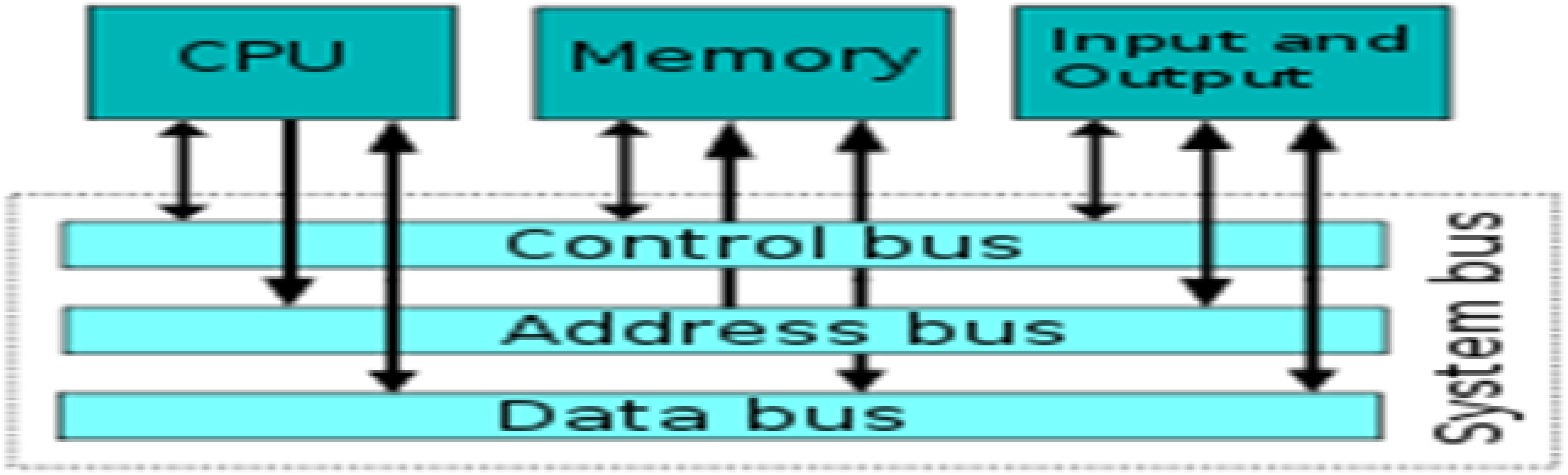
# Basics of C++ Revision

## Von Neumann Model

- **Data** and **Instructions** are stored in Single **R/W Memory**
- Contents of This Memory are addressable by **Location**
- Execution occurs **Sequentially**, from one Instruction to the next



## C++ program

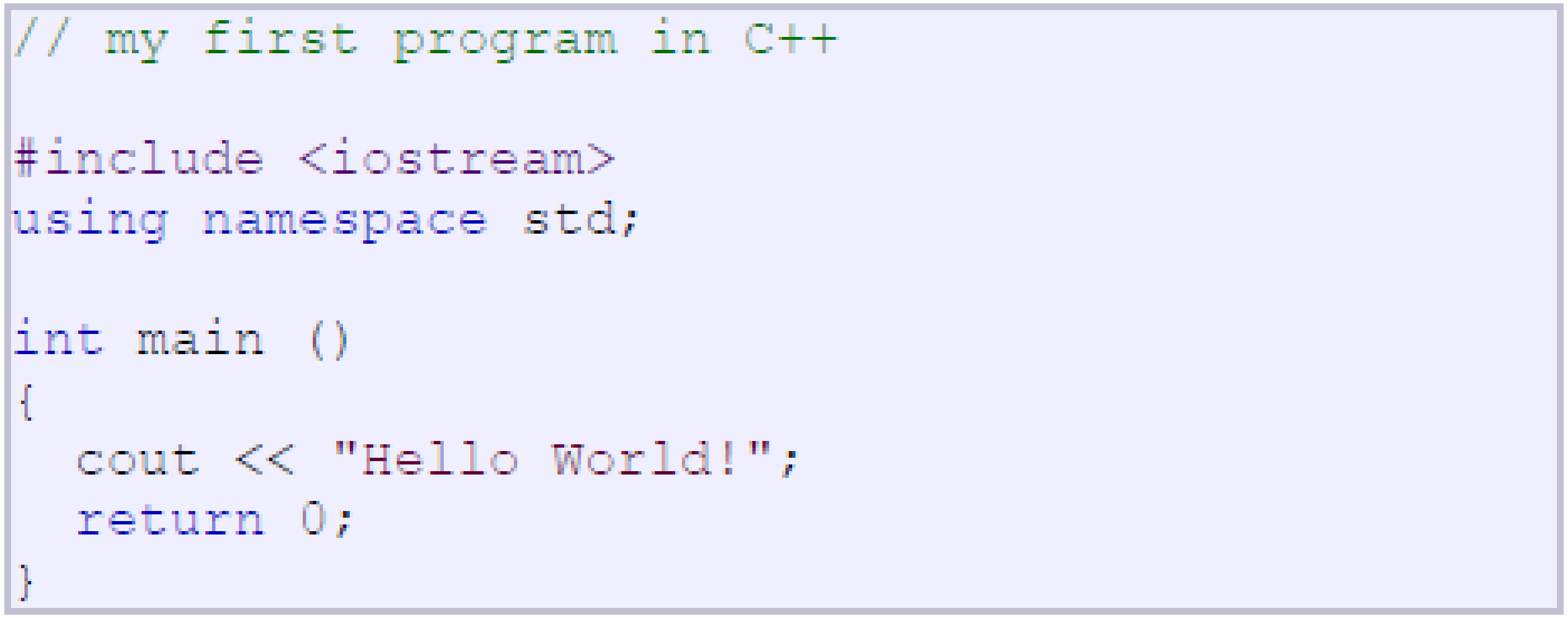
Consists  of :

-Compiler Preprocessor directives

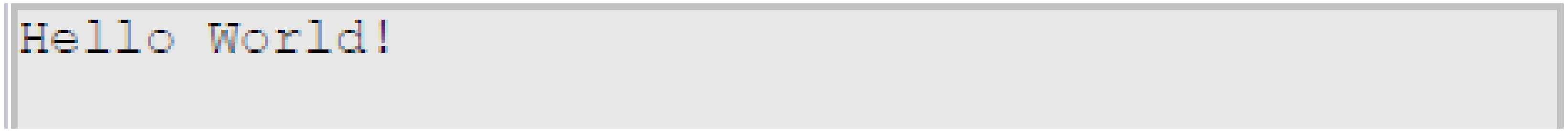
-Constants-Variables

-Standard functions- other functions which
 are declared
 -main functions

## First program



```cpp
// my first program in C++

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```



```
Hello World!
```

## My first program

- // my first program in C++

  This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program.
- #include <iostream>

  Lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor.

- **tells the preprocessor to include the iostream standard file.**
- **This specific file (*iostream*) includes the declarations of the basic standard input-output library in C++,**
- **using namespace std;**
- **All the elements of the standard C++ library are declared within what is called a namespace, the**

**namespace with the name *std*.**

# Header Files

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

## Directives

- The two lines that begin the FIRST 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
- They are instructions to the compiler

- Include Directives add library files to our programs
  - To make the definitions of the cin and cout available to the program:

```
#include <iostream>
```

- Using Directives include a collection of defined names
  - To make the names cin and cout available to our program:

```
using namespace std;
```

## main ()

- This line defines a *function* called main
- This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs *start* their execution the instructions contained within this function's definition will always be the *first* ones to be executed in any C++ program. For that same reason, it is *essential* that all C++ programs have a main function

## Always Start with main()

- The word main is followed in the code by a pair of parentheses (()).
- That is because it is a function declaration: In C++, what differentiates a function declaration from other types of expressions are these parentheses that follow its name. Optionally, these parentheses may enclose a list of *parameters* within them.

## cout

- Right after these parentheses we can find the body of the main function enclosed in braces ({}). What is contained within these braces is what the function does when it is executed.
- cout << "Hello World!";

cout represents the *standard output stream* in C++

## <<  output operator

- A string is any sequence of characters enclosed in double-quotes.
- Cout is the standard output stream in C++

(standard output usually means your computer monitor screen). The symbol

<< is an output operator

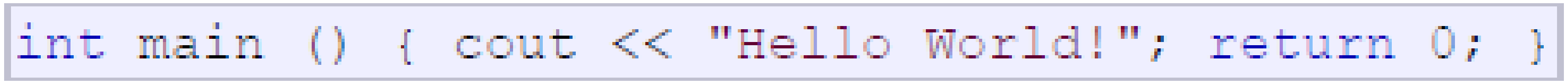
## return

- Notice that the statement ends with a semicolon character (;). This character is used to mark the end of the statement and in fact it must be included at the end of all expression statements in all C++ programs
- return 0;

  The return statement causes the main function to finish

## just one line

- We could have written:

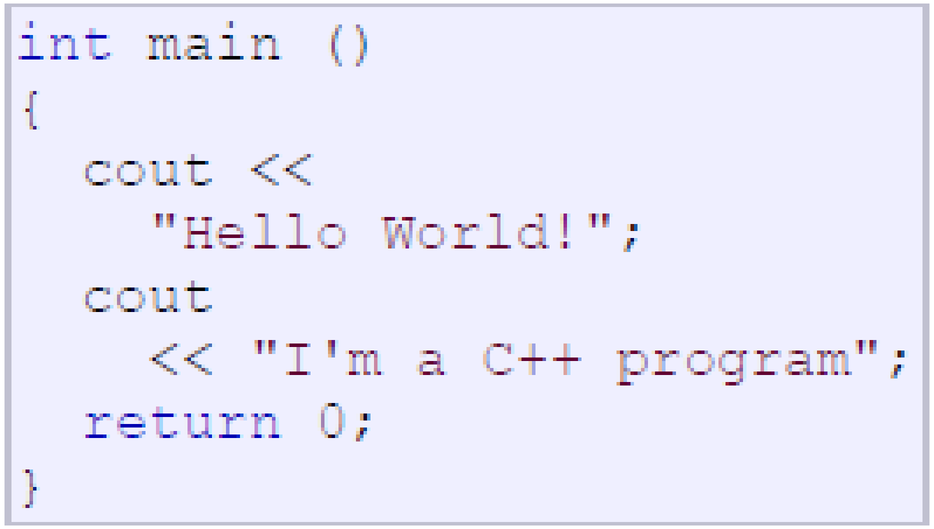


```
int main () { cout << "Hello World!"; return 0; }
```

All in just one line and this would have had exactly the same meaning as the previous code.

## Code in many lines

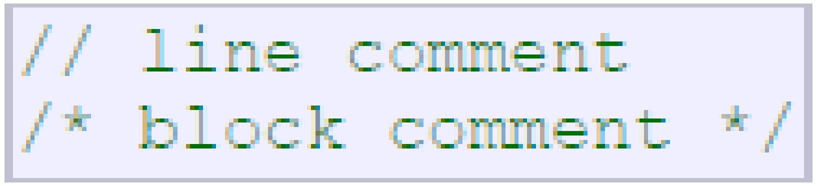
- We were also free to divide the code into more lines if we considered it more convenient:



```
int main ()
{
  cout <<
    "Hello World!";
  cout
    << "I'm a C++ program";
  return 0;
}
```

# Comments

- C++ supports two ways to insert comments:



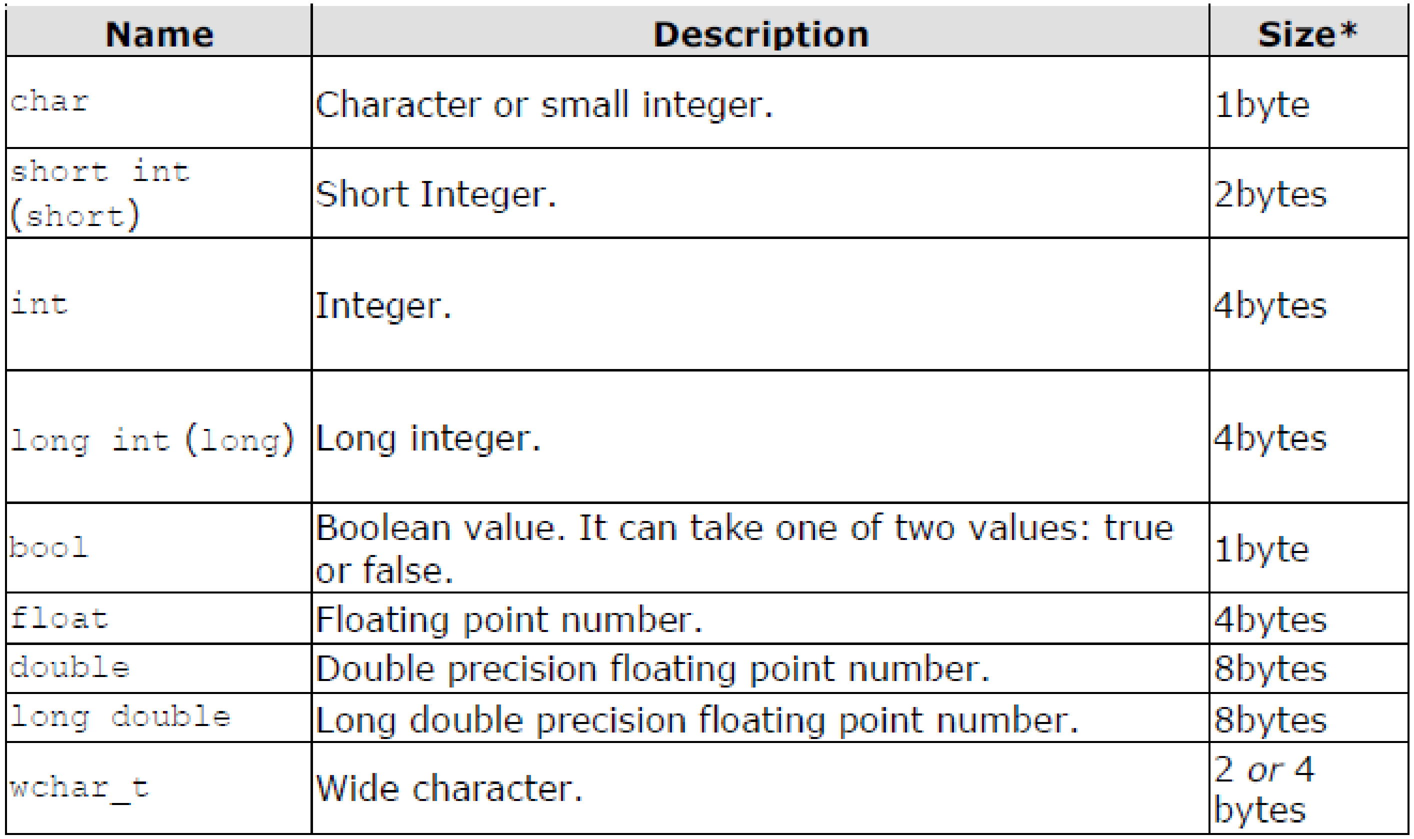
```
// line comment
/* block comment */
```

The second one, known as block comment, discards everything between the /*characters and the first appearance of the */ characters, with the possibility of including more than one line.
We are going to add comments to our second program:

## block comment

```
/* my second program in C++
   with more comments */

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World! ";     // prints Hello
World!
  cout << "I'm a C++ program"; // prints I'm a
C++ program
  return 0;
}
```

# Data types in C++



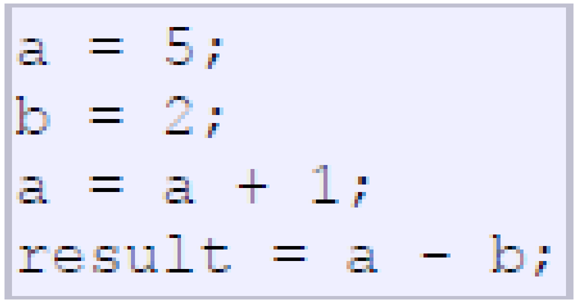
| Name | Description | Size* |
|---|---|---|
| `char` | Character or small integer. | 1byte |
| `short int` (`short`) | Short Integer. | 2bytes |
| `int` | Integer. | 4bytes |
| `long int` (`long`) | Long integer. | 4bytes |
| `bool` | Boolean value. It can take one of two values: true or false. | 1byte |
| `float` | Floating point number. | 4bytes |
| `double` | Double precision floating point number. | 8bytes |
| `long double` | Long double precision floating point number. | 8bytes |
| `wchar_t` | Wide character. | 2 *or* 4 bytes |

# Variables

- Variables are like small blackboards
  - We can  write a number on them
  - We can change the number
- C++ variables are names for memory locations
  - We cannot erase the memory location

# Variables

- we can define a variable as a portion of memory to store a determined value.



```
a = 5;
b = 2;
a = a + 1;
result = a - b;
```

Each variable needs an identifier that distinguishes it from the others, for example, in the previous code the variable identifiers were a, b and result

# Identifiers

- A valid identifier is a sequence of one or more letters, digits or underscore characters (_). Neither spaces nor punctuation marks or symbols can be part of an identifier. Another rule that you have to consider when inventing your own identifiers is that they cannot match any *reserved keywords*

## Identifiers

- First character must be
  - a letter
  - the underscore character
- Remaining characters must be
  - letters
  - numbers
  - underscore character

## Identifiers

- an identifier written in capital letters is not equivalent to another one with the same name but written in small letters.
- Identifiers refer to the names of variables, functions, arrays, classes , etc. created by the programmer.

# Declaration of variables

- In order to use a variable in C++, we must first declare it specifying which data type we want it to be. The syntax to declare a new variable is to write the specifier of the desired data type (like int, bool, float...) followed by a valid variable identifier.
- Examples:

```
int     number_of_students;
double  student_weight;
```

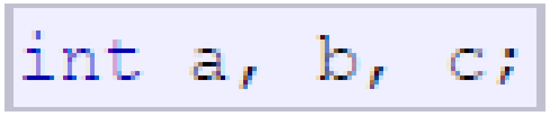
# Declaration of variables

- **int** is an abbreviation for integer.
- **double** represents numbers with a fractional component

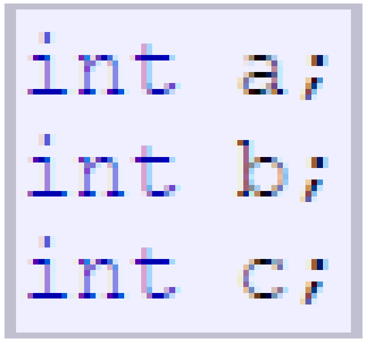
- Before use, variables must be <u>declared</u>
  - Tells the compiler the type of data to store
- Declaration syntax:
  - **Type_name  Variable_1 ,  Variable_2, . . . ;**
  -

**If you are going to declare more than one variable of the same type, you can declare all of them in a single statement by separating their identifiers with commas. For example**:

## Declaration of variables

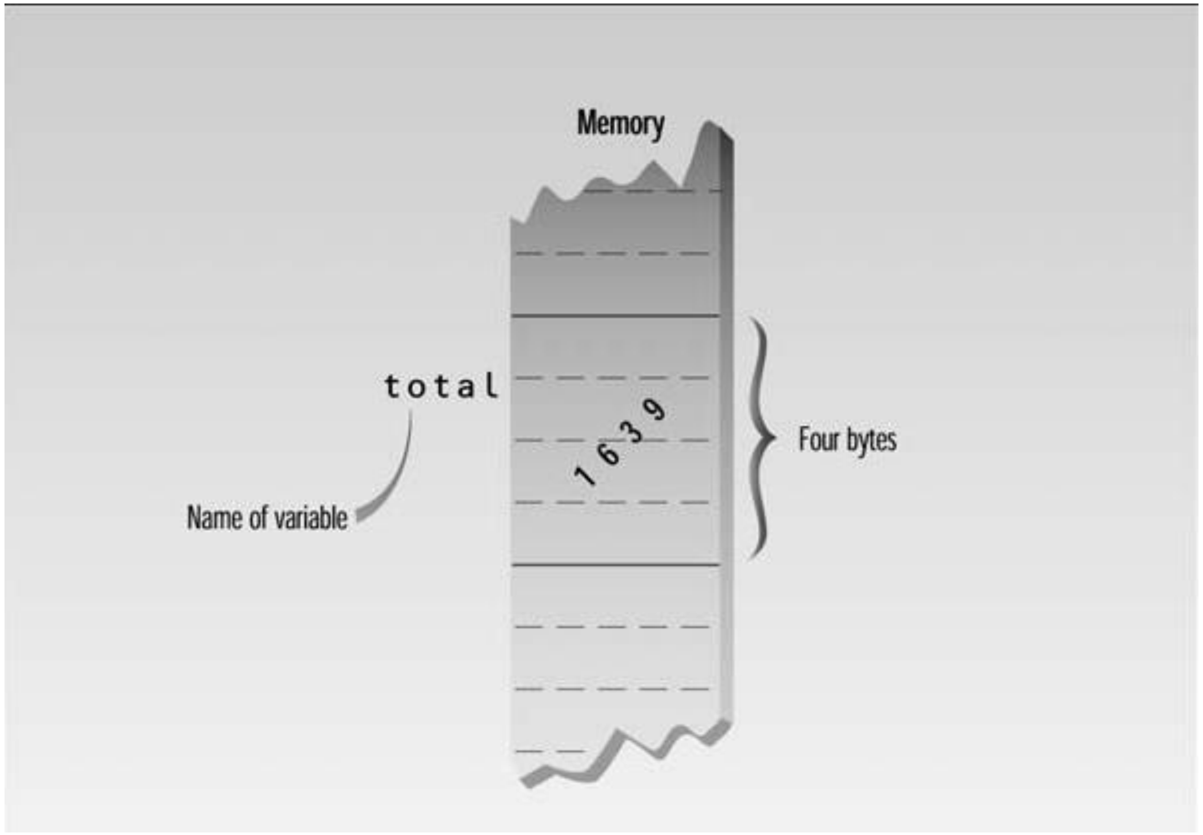


```
int a, b, c;
```



```
int a;
int b;
int c;
```

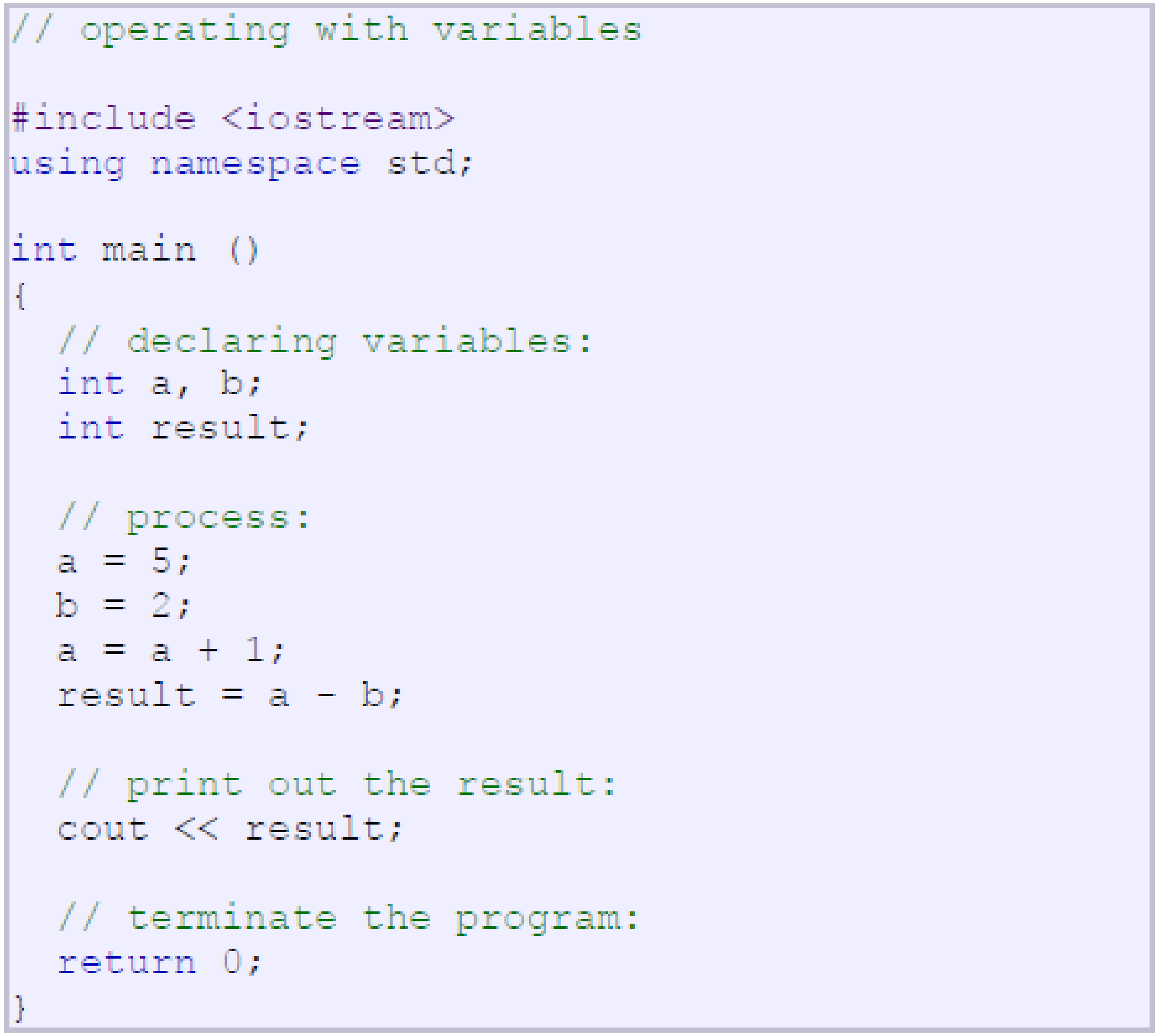
## Variable of type int in memory



**an int occupies 4 bytes (which is 32 bits) of memory**

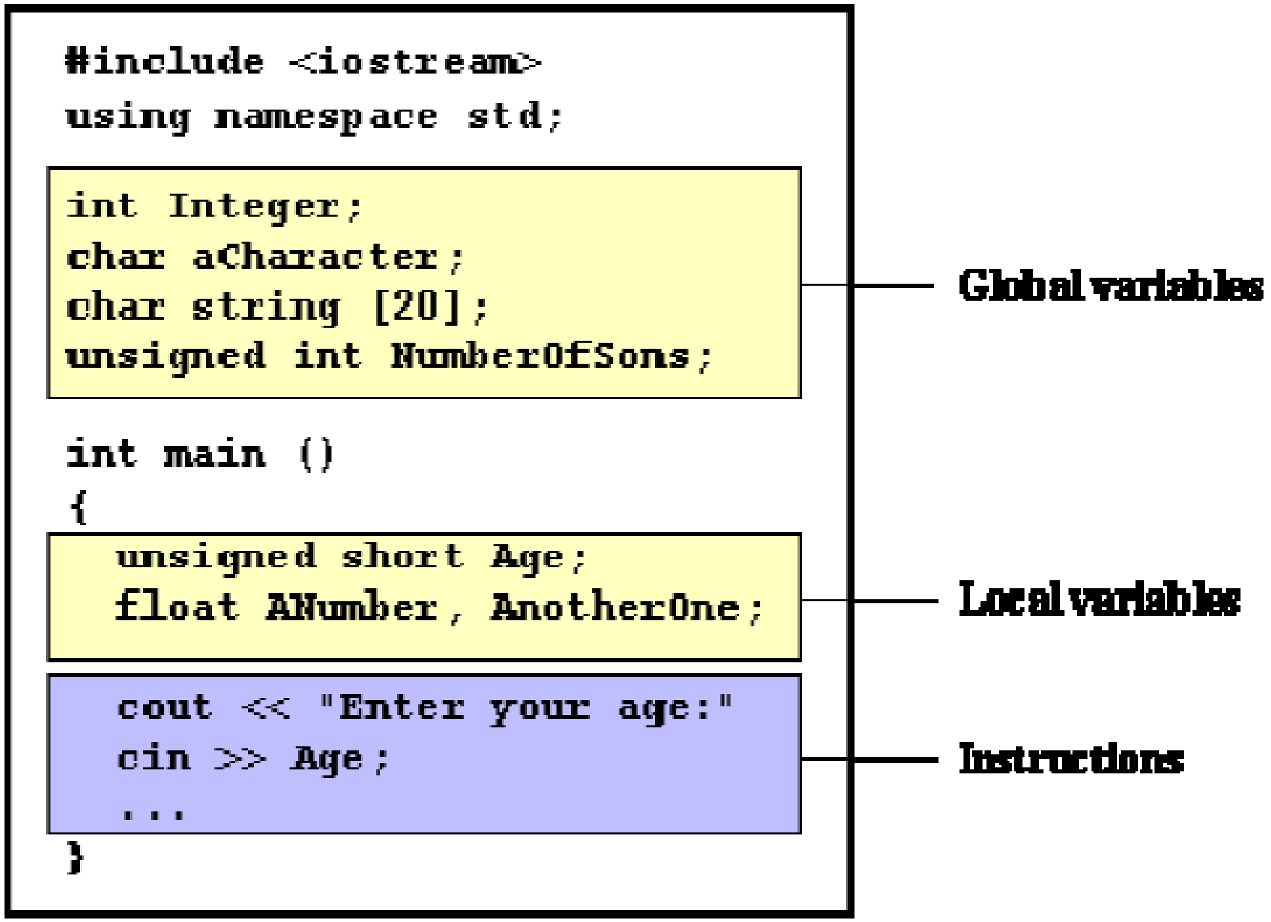
## Assignment Statements

- The statements
- var1 = 5;
- var2 = 2;
- assign values to the two variables. The equal sign (=), as you might guess, causes the value on the right to be assigned to the variable on the left.



```cpp
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

# Scope of variables



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

int Integer;
char aCharacter;
char string [20];
unsigned int NumberOfSons;

int main ()
{
   unsigned short Age;
   float ANumber, AnotherOne;

   cout << "Enter your age:"
   cin >> Age;
   ...
}
```

## Global  and Local variables

- A variable can be either of global or local scope. A global variable is a variable declared in the main body of the source code, outside all functions, while a local variable is one declared within the body of a function or a block.
- A global variable ( External variable) is visible to all functions so you put their decleration at the beginning of listing.

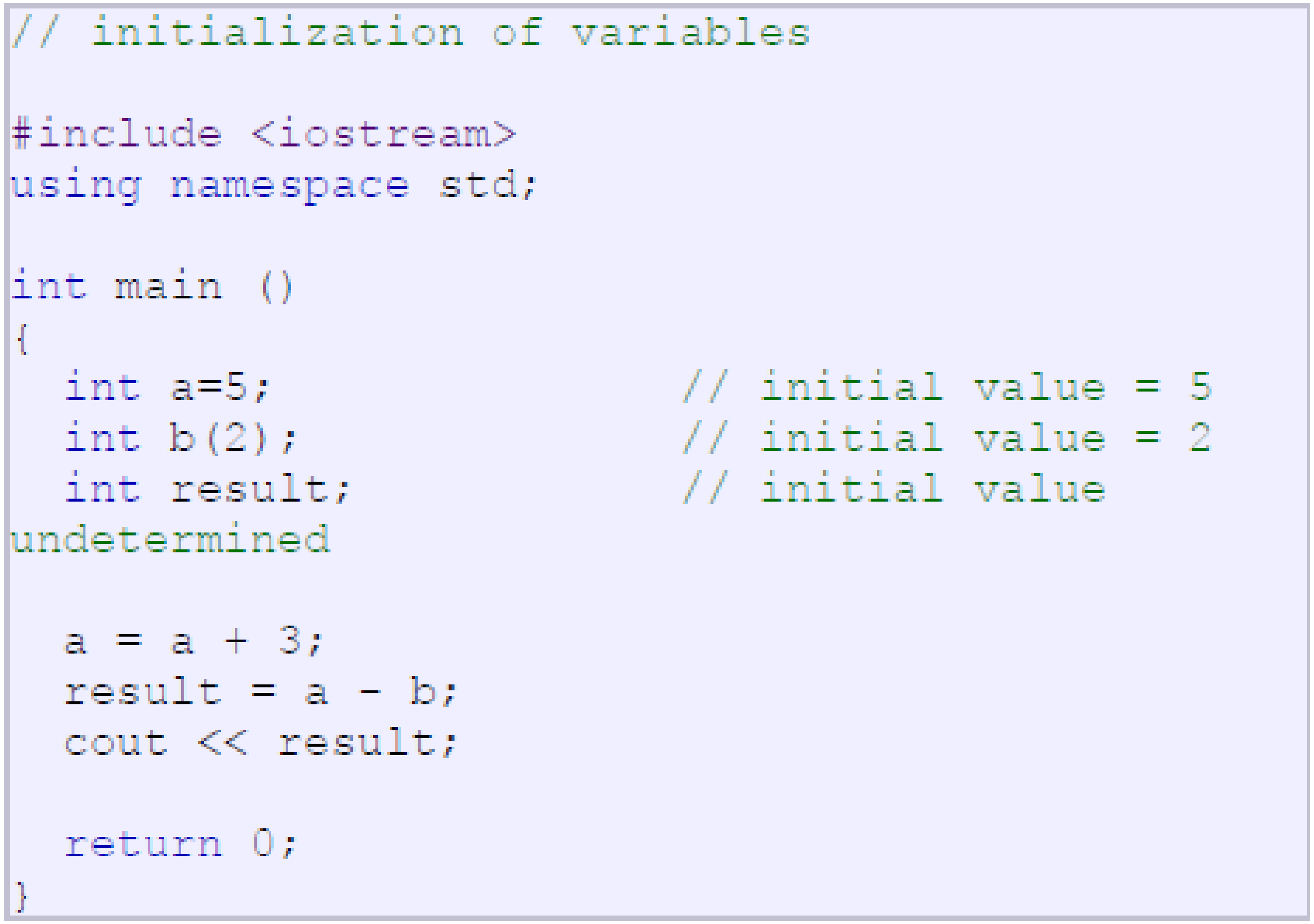
## Initialization of variables

- This is done by appending an equal sign followed by the value to which the variable will be initialized:
- type identifier = initial_value ;
- For example, if we want to declare an int variable called a initialized with a value of 0 at the moment in which it is declared, we could write: int a = 0;

## constructor initialization

- The other way to initialize variables, known as constructor initialization, is done by enclosing the initial value between parentheses (()):
- type identifier (initial_value) ;

  For example: int a (0);



```
// initialization of variables

#include <iostream>
using namespace std;

int main ()
{
  int a=5;                // initial value = 5
  int b(2);               // initial value = 2
  int result;             // initial value
undetermined

  a = a + 3;
  result = a - b;
  cout << result;

  return 0;
}
```

- When you declare a variable, you create a named storage location.
- When you make an assignment to a variable, you give it a value.
- you cannot store a string in an int variable. The following statement generates a compiler error.

-
  ```
  int hour;
  hour = "Hello.";        // WRONG !!
  ```

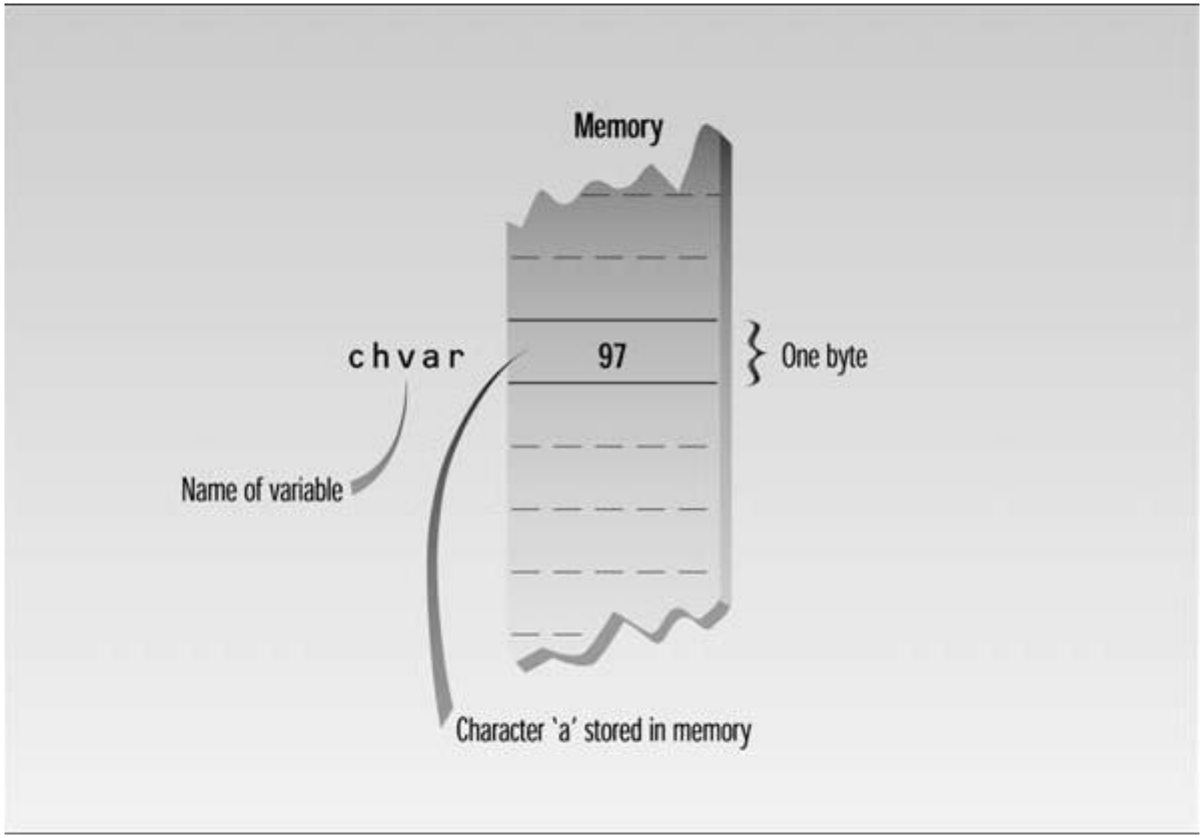
## Character Variables

- Type char stores integers that range in value from –128 to 127. Variables of this type occupy only 1 byte (eight bits) of memory. Character variables are sometimes used to store numbers that confine themselves to this limited range, but they are much more commonly used to store ASCII characters.

## Character Constants

- 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. The constant ‘a’ appearing in a program, for example, will be translated into 97,

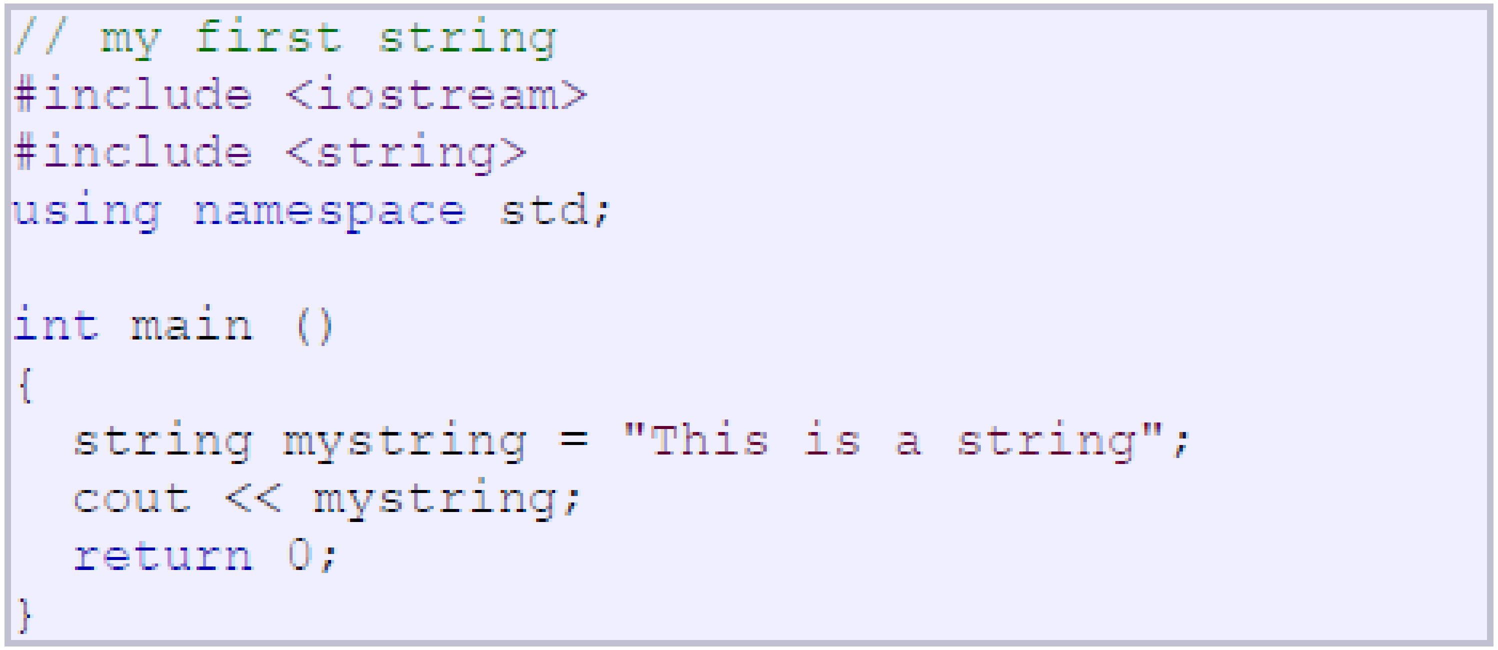
## Character Constants



***Variable of type char in memory***

## Introduction to strings

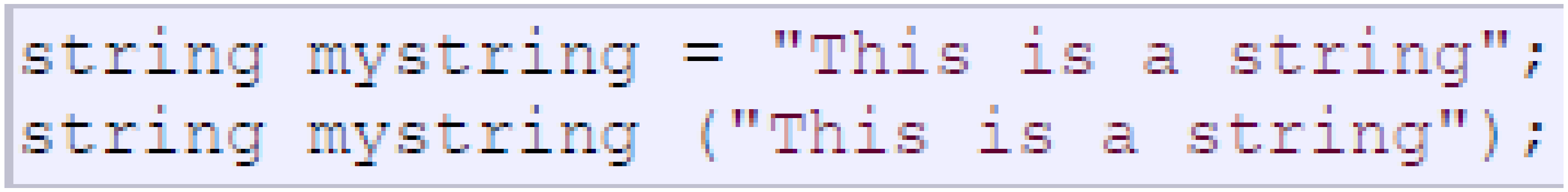
- Variables that can store non-numerical values that are longer than one single character are known as strings.



```cpp
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring = "This is a string";
  cout << mystring;
  return 0;
}
```

## Strings

- strings can be initialized with any valid string literal just like numerical type variables can be initialized to any valid numerical literal. Both initialization formats are valid with strings



```
string mystring = "This is a string";
string mystring ("This is a string");
```

## Input and Output

- A **data stream** is a sequence of data
  - Typically in the form of characters or numbers

- An input stream is data for the program to use
  - Typically originates
    - at the keyboard
    - at a file

- An output stream is the program's output
  - Destination is typically
    - the monitor
    - a file

## Output using cout

- cout is an output stream sending data to the monitor
- The insertion operator "<<" inserts data into cout
- Example:

  cout << number_of_bars << " candy bars\n";
  - This line sends two items to the monitor
    - The value of number_of_bars
    - The quoted string of characters " candy bars\n"

## Output Variations

- The statement
- cout << “var1+10 is “;
- displays a string constant, as we’ve seen before.
- The next statement
- cout << var2 << endl;
- displays the value of the variable var2.

## Input with cin

- how a program accomplishes input. The next example program asks the user for a temperature in degrees Fahrenheit, converts it to Celsius, and displays the result. It uses integer variables.
- The statement
- cin >> ftemp;
- causes the program to wait for the user to type in a number.

// program to calculate temp in Celsius

```
// 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;
   }
```

```
  // program to calculate area of circle
#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;
    }
```

## >> is the *extraction or get from operator*

- The keyword cin is an object, predefined in C++ to correspond to the standard input stream. This stream represents data coming from the keyboard.

## Cascading <<

- The insertion operator << is used repeatedly in the second cout statement in FAHREN.
- The program first sends the phrase *Equivalent in Celsius is: to cout, then it* sends the value of ctemp, and finally the newline character '\n'.

## ++ Increment Operators

- count = count + 1; // adds 1 to “count”
- Or you can use an arithmetic assignment operator:
- count += 1; // adds 1 to “count”
- But there’s an even more condensed approach:
- ++count; // adds 1 to “count”
- The ++ operator increments (adds 1 to) its argument.

## The Decrement (--) Operator

- The decrement operator, --, behaves very much like the increment operator, except that it subtracts 1 from its operand.

## Prefix and Postfix

- the increment operator can be used in two ways: as a *prefix,* meaning that the operator precedes the variable; and as a *postfix, meaning that the operator follows*
- the variable.
- totalWeight = avgWeight * ++count;
- In this case count is incremented first.
- Because prefix notation is used:++count.

- If we had used postfix notation, count++, the multiplication would have been performed first, then count would have been incremented.

## Library Functions

- Many activities in C++ are carried out by *library functions. These functions perform file* access, mathematical computations, and data conversion, among other things.