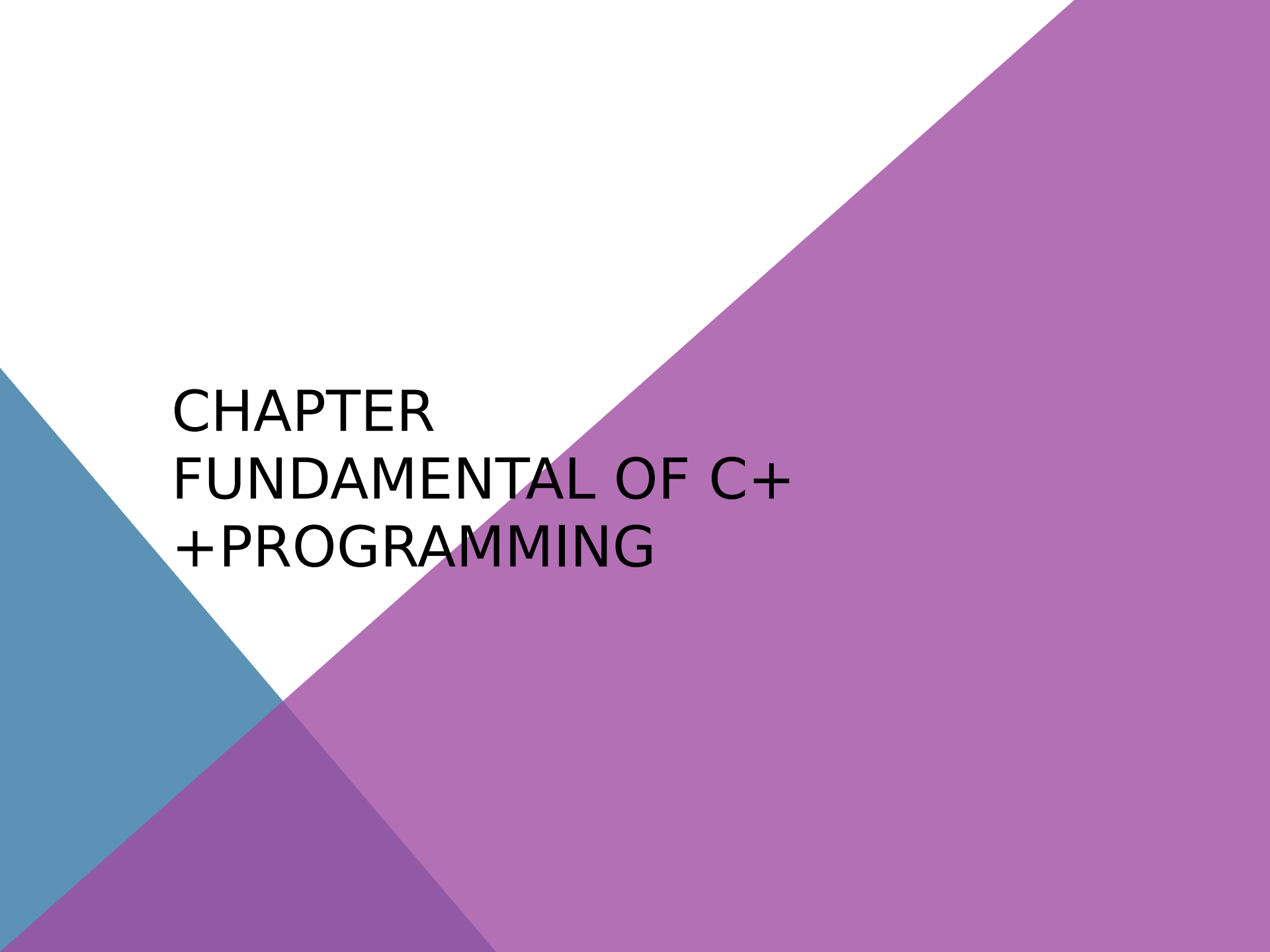


INTRODUCTION TO PROGRAMMING

A LECTURE FOR THE C++ COURSE



CHAPTER FUNDAMENTAL OF C+ +PROGRAMMING

BASICS OF PROGRAMMING

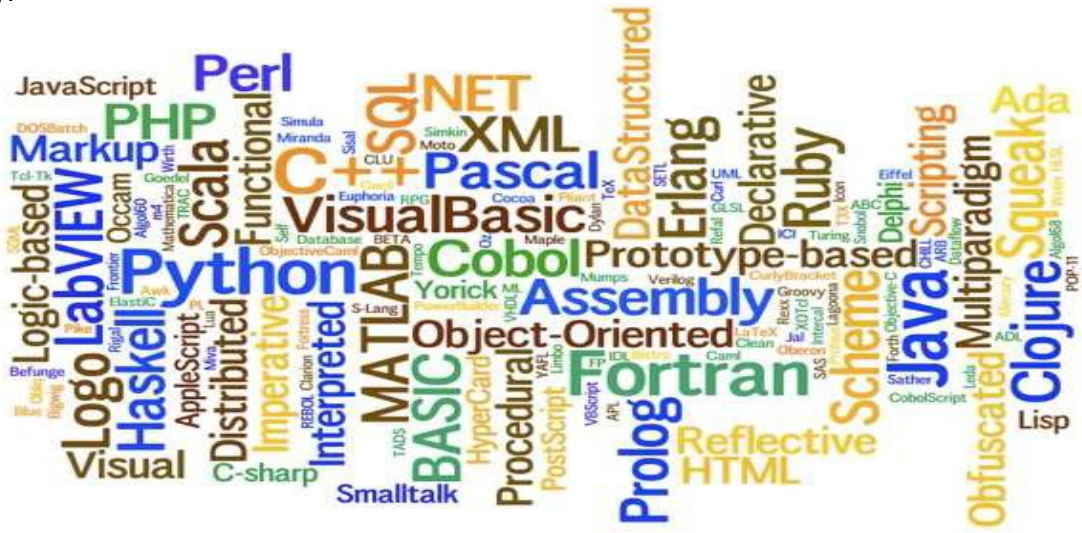
A **computer** is a programmable machine.

Computer Programming is the process of developing & implementing various sets of instructions to enable a computer to do a certain task.

Programs are written to solve problems or perform tasks on a computer

Programme
the comp

a language



BASICS OF PROGRAMMING

Machine code- also known as machine language, is the elemental language of computers, comprising a long sequence of binary digital zeros and ones (bits). Machine language is the only language a computer is capable of understanding.

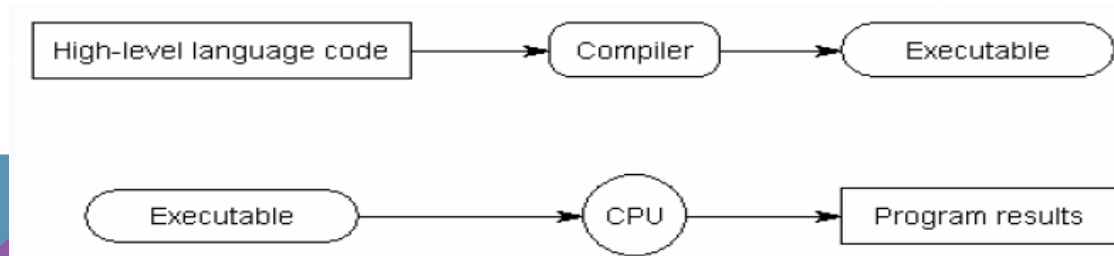
Assembly Language-Sometimes referred to as assembly or ASL, is a low-level programming language used to interface with computer hardware. uses structured commands as substitutions for numbers allowing humans to read program written in a **low-level language** can be extremely efficient, making optimum use of both computer memory and processing time. But takes substantial amount of time, as well as a clear understanding of the inner workings of the processor

BASICS OF PROGRAMMING

High-level languages, Permit faster development of large programs. Not efficient while executed by computer, but the savings in programmer time. This is because the cost of writing a program is nearly constant for each line of code, regardless of the language.

A **compiler** is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses.

After you write a program, your source language statements are compiled into machine code that is stored as an executable file.



BASICS OF PROGRAMMING

Debugger A special program used to find errors (bugs) in other programs. A debugger allows a programmer to stop a program at any point and examine and change the values of variables.

Graphical User Interface (GUI)

A GUI is a graphical (rather than purely textual) user interface to a computer.



ALGORITHM & FLOW CHART

An **algorithm** is a list of instructions, procedures, or formulas used to solve a problem.

To be an algorithm, a set of rules must be unambiguous and have a clear stopping point”.

Type of Algorithms

The algorithm and flowchart, classification to the three types of *control structures*. They are:

1. Sequence
2. Branching (Selection)
3. Loop (Repetition)

The *sequence* is exemplified by sequence of statements place one after the other - the one above or before another gets executed first.

- represented by rectangular box in flow chart

The *branch* refers to a binary decision based on some condition. If the condition is true, one of the two branches is explored;

- represented by diamond shape in flow chart
- There are three selection structures in C++:
 - If
 - If else
 - Switch

The *loop* allows a statement or a sequence of statements to be repeatedly executed based on some loop condition.

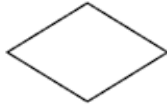
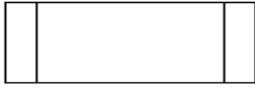
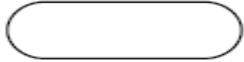
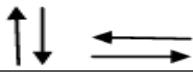
- In the flowcharts, a back arrow hints the presence of a loop
- There are three repetition structures in C:
 - While
 - Do While

For

FLOWCHART SYMBOLS

There are 6 basic symbols commonly used in flowcharting of assembly language

Programs

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

General Rules for drawing flowcharting

- 1. All boxes of the flowchart are connected with Arrows.**
- 2. Flowchart symbols have entry point on the top & exit point on the bottom except for the Decision symbol.**
- 3. The Decision symbol has two exit points; the bottom and one side.**
- 4. Generally a flowchart will flow from top to bottom.**
- 5. Connectors are used to connect breaks in the flowchart. Examples are:**
 - From one page to another page.
 - From the bottom of the page to the top of the same page.
- 7. All flow charts start with a Terminal or Predefined Process symbol.**
- 8. All flowcharts end with a terminal or a contentious loop.**

Pseudo code (or Program Design Language) Consists of natural language-like statements that precisely describe the steps of an algorithm or program.

- Focuses on the *logic* of the algorithm or program
- Avoids language-specific elements
- Written at a level so that the desired programming code can be generated almost automatically from each statement

Pseudo code Language Constructs

- Computation/Assignment
 - Compute** var1 as the sum of x and y
 - Assign** expression to var2
 - Increment** counter1
- Input / Output
 - Input: **Get** var1, var2, ...
 - Output: **Display** var1, var2, ...

Example of pseudocode

Example 1: Find the area of a Circle of radius r.

Pseudocode

Step1: Start

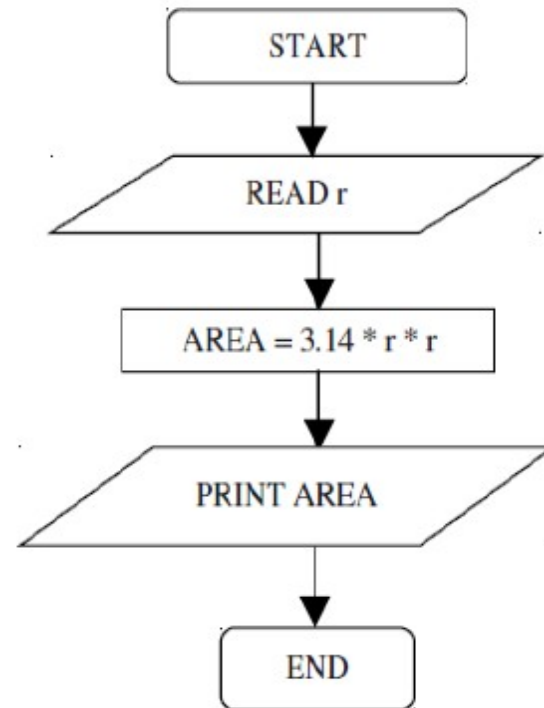
Step2: Read\input the Radius r of the Circle

Step3: Area $\pi * r * r$ // calculation of area

Step4: Print Area

Step6: End

Example of flow chart



Example 2: Write an Pseudocode to read two numbers and find their sum.

Step1: Start

Step2: Read\input the first num1.

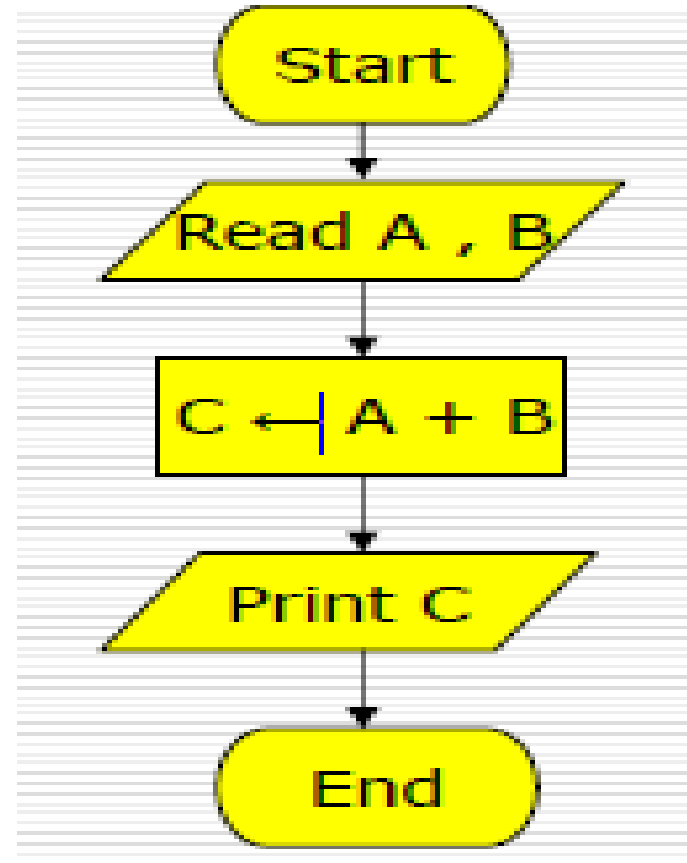
Step3: Read\input the second num2.

Step4: Sum num1+num2 // calculation of sum

Step5: Print Sum

Step6: End

Example of flow chart



Example 3: write pseudo code to find the greater number between two numbers

Step1: Start

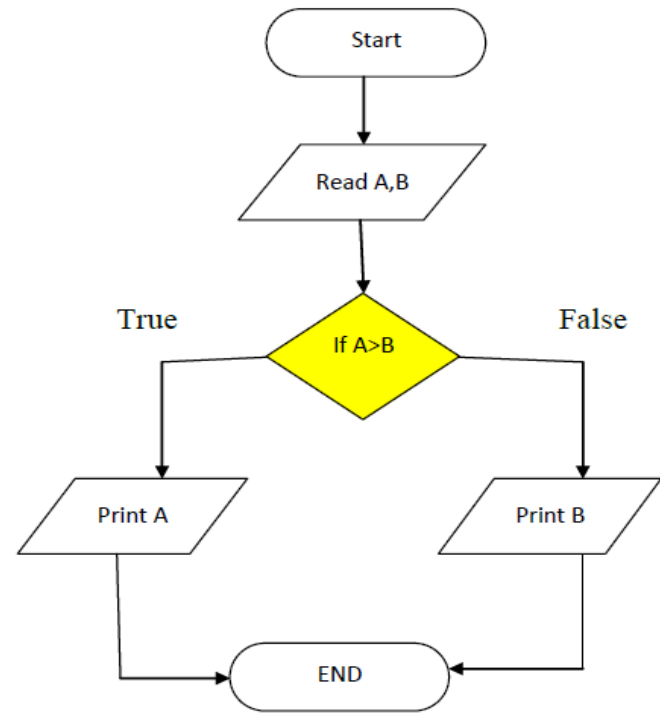
Step2: Read/input A and B

Step3: If A greater than B then $C=A$

Step4: if B greater than A then $C=B$

Step5: Print C

Step6: End



Example 4: write pseudo code to find the result: $f(x) = \begin{cases} -x, & x < 0 \\ x, & x \geq 0 \end{cases}$

- Step1: Start
- Step2: Read/input x
- Step3: If X Less than zero then F=-X
- Step4: if X greater than or equal zero then F=X
- Step5: Print F
- Step6: End

Problem 1: A pseudo code &flow chart to find the largest value of any three numbers.

Examples: A pseudo code & flow chart to display even numbers between 0 and 99

Step 1. Start

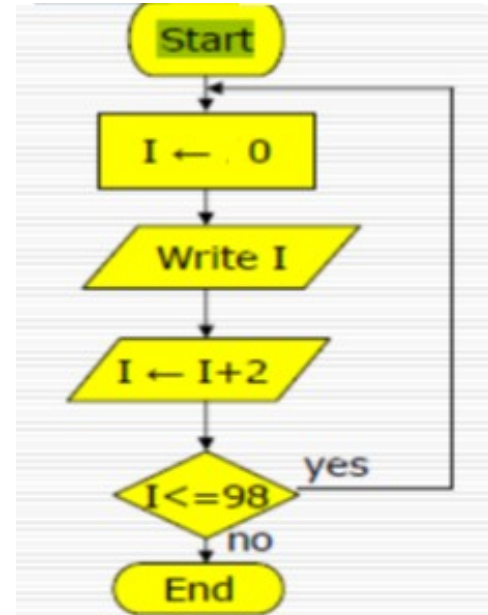
Step 2. $I \leftarrow 0$

Step 3. Write I in standard output

Step 4. $I \leftarrow I+2$

Step 5. If ($I \leq 98$) then go to line 3

Step 6. End



Problem 2 Pseudo code & Flowchart for the problem of printing odd numbers less than a given number. It should also calculate their sum and count.

BASICS OF C++

Structure of a C++ program

```
// my first program in  
C++  
#include <iostream.h>  
int main ()  
{  
cout << "Hello World!";  
return 0;  
}
```

- This is a comment line. All the lines beginning with two slash signs (//) are considered comments
- pound sign (#) are directives for the preprocessor
 - They are not executable code lines but indications for the compiler. [iostream standard header file](#).
- Beginning of the **main** function declaration. The **main** function is the point by where all C++ programs begin their execution.
 - In C++ all functions are followed by a pair of parenthesis ()
- **cout** is the standard output stream in C++
 - Notice that the sentence ends with a semicolon character (;).
- terminates the function main() and also the program, returning a value of 0 as an exit code to the calling program.
- last ones were all included within the block delimited by the curly brackets ({}) of the **main** function.

Comments.

Comments are pieces of source code discarded from the code by the compiler. They do nothing.

Their purpose is only to allow the programmer to insert notes or descriptions embedded within the source code.

- C++ supports two ways to insert comments:
 - `//` line comment -only single line
 - `/*` block comment `*/`- more than one line

```
/* my second program in C++  
with more comments */  
#include <iostream.h>  
int main ()  
{  
cout << "Hello World! "; //  
says //Hello World!  
cout << "I'm a C++ program";  
//says I'm a C++ program  
return 0;  
}
```

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

block comment

```
//Hello World!  
// says  
//says I'm a C++ program
```

line comment

VARIABLE

Is a portion of memory to store a determined value.

variable needs an identifier that distinguishes it from the others.

- Example

- `a = 5;`

- `b = 2;`

a & b were variable identifiers

but we could have called the variables any names we wanted to invent, as long as they were valid identifiers.

A valid identifier is a sequence of one or more letters, digits or underline symbols (`_`).

- The length of an identifier is not limited
- Neither spaces nor marked letters can be part of an identifier.
- should always begin with a letter & underline character (`_`).
- In no case they can begin with a digit.

Another rule that you have to consider when inventing your own *identifiers* is that they cannot match any key word of the C++ language

Here some key words

asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t

Additionally, alternative representations for some operators do not have to be used as identifiers since they are reserved words under some circumstances:

and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq

Very important: The C++ language is "case sensitive", that means that an identifier written in capital letters is not equivalent to another one with the same name but written in small letters.

- example
- Variable result is not RESULT or Result

DATA TYPES

We store the variables in our computer's memory, the computer must know what we want to store in them

- Storing a simple number,
- Single character
- A letter
- A large number is not going to occupy the same space in memory.

Our computer's memory is organized in bytes.

A byte is the minimum amount of memory that we can manage.

A byte can store a relatively small amount of data the computer can manipulate more complex data types that come from grouping several *bytes*,

Name	Bytes*	Description	Range*
char	1	character or integer 8 bits length.	signed: -128 to 127 unsigned: 0 to 255
short	2	integer 16 bits length.	signed: -32768 to 32767 unsigned: 0 to 65535
long	4	integer 32 bits length.	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
int	*	Integer. Its length traditionally depends on the length of the system's Word type , thus in MSDOS it is 16 bits long, whereas in 32 bit systems (like Windows 9x/2000/NT and systems that work under protected mode in x86 systems) it is 32 bits long (4 bytes).	See short, long
float	4	floating point number.	3.4e + / - 38 (7 digits)

long double	10	long double precision floating point number.	1.2e + / - 4932 (19 digits)
bool	1	Boolean value. It can take one of two values: <code>true</code> or <code>false</code> NOTE: this is a type recently added by the ANSI-C++ standard. Not all compilers support it. Consult section bool type for compatibility information.	true or false
wchar_t	2	Wide character. It is designed as a type to store international characters of a two-byte character set. NOTE: this is a type recently added by the ANSI-C++ standard. Not all compilers support it.	wide characters

DECLARATION OF VARIABLES

The syntax to declare a new variable is to write the data type specifier that we want followed by a valid variable identifier.

syntax ----- **type identifier;**

- **int** a;
 - **float** mynumber;
- Are valid declarations of variables.

If you need to declare several variables of the same type,

int a, b, c;	or	Char a, b, c;	or	float a, b, c;
int a;		Char a;		float a;
int b;		Char b;		float b;
int c;		Char c;		float c;

To specify an integer data type whether it is **signed** or **unsigned** put before the data type itself

unsigned short NumberOfSons;

signed int MyAccountBalance;

By default, if we do not specify **signed** or **unsigned** it will be assumed that the type is **signed**, therefore in the second declaration we could have written:

int MyAccountBalance;

INITIALIZATION OF VARIABLES

When declaring a local variable, its value is undetermined by default.

But you may want a variable to store a concrete value the moment that it is declared.

syntax

For example:

`int a (10);` or
`int a=10;`

type identifier = *initial_value* ; or
type identifier (*initial_value*)

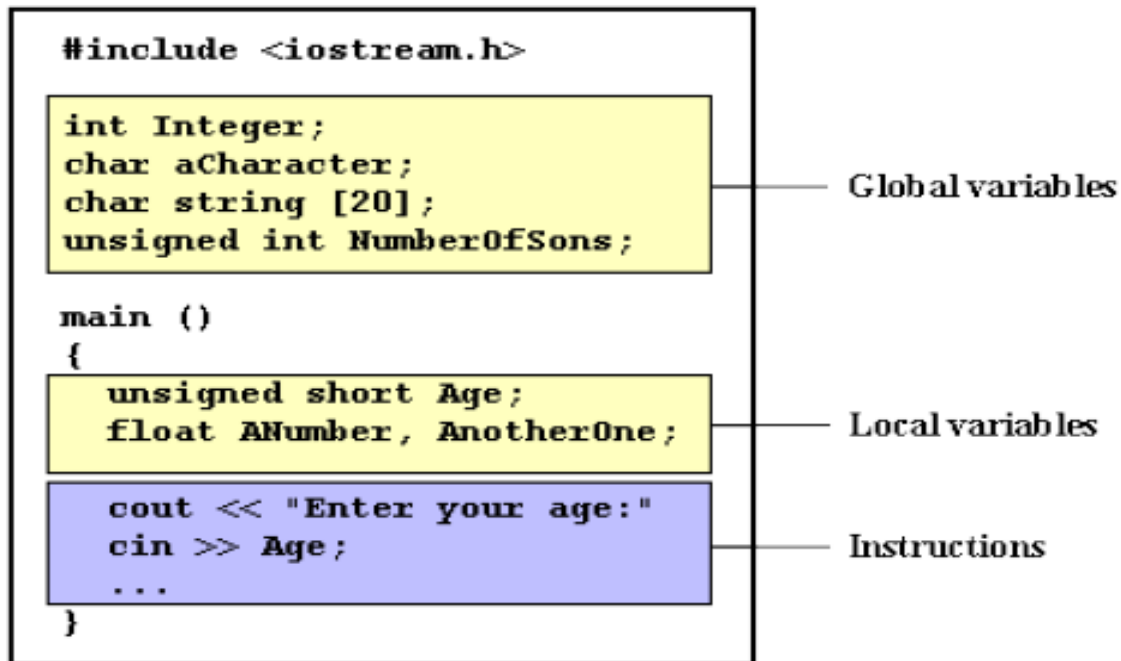
```
#include <iostream.h>
int main ()
{
// declaring variables without
initialization :
int a, b;
int result;
// process:
a = 5;
b = 2;
    result = a - b;
// print out the result:
cout << result;
// terminate the program:
return 0;
}
```

```
#include <iostream.h>
int main ()
{
// declaring variables initialization :
int a=5, b=2;
int result;
// process:
a = a + 1;
result = a - b;
// print out the result:
cout << result;
// terminate the program:
return 0;
}
```

Scope of variables

All the variables that we are going to use must have been previously declared.

- local variables -variables include their declaration at the beginning of each function



the program outside any

Global variables can be referred to anywhere in the code, within any function, whenever it is after its declaration.

The scope of the **local variables** is limited to the code level in which they are declared. If they are declared at the beginning of a function (like in **main**) their scope is the whole **main** function.

FLOATING POINT NUMBERS

They express numbers with decimals and/or exponents.

They can include a decimal point, an e character (that expresses "by ten at the Xth height", where X is the following integer value) or both.

- `float pi = 3.14159 // 3.14159`
- `float Avogadro = 6.02e23 // 6.02 x 1023`
- `float electron = 1.6e-19 // 1.6 x 10-19`
- `float a = 3.0 // 3.0` expressed as a floating point numeric literal.

Character & string

There also exist non-numerical constants, like:

`Char a='z'`

`Char x='p'`

`String s="Hello world"`

`String d="How do you do?"`

Char can hold only single character where string is hold any word that is in side quotation mark (" ")

to declare char we need single quotation mark (' ')

For string quotation marks (" ")

Character constants and string constants have certain peculiarities, like the escape codes.

These are special characters that cannot be expressed otherwise in the source code of a program

<code>\n</code>	newline
<code>\r</code>	carriage return
<code>\t</code>	tabulation
<code>\v</code>	vertical tabulation
<code>\b</code>	backspace
<code>\f</code>	page feed
<code>\a</code>	alert (beep)
<code>\'</code>	single quotes (')
<code>\"</code>	double quotes (")
<code>\?</code>	question (?)
<code>\\</code>	inverted slash (\)

DEFINED CONSTANTS (*#DEFINE*)

You can define your own names for constants that you use quite often without having to resort to variables, simply by using the `#define` preprocessor directive.

This is its format(syntax :

`#define` *identifier value*

`#define` PI 3.14159265

`#define` NEWLINE '\n'

`#define` WIDTH 100

OPERATORS

Assignment (=) operator serves to assign a value to a variable.

`a = 5;` //assigns the integer value **5** to variable **a**.

`S = 'w'` //assigns the character **w** to variable **S**.

`See = "have u work your homework today"` // assigns string to variable **See**

`w=y` // this means assign the value of **y** to **w**

example if `int w, y=10;`

`w=y;` // this means **w=10**

`a = b = c = 5;` //assigns **5** to the three variables **a**, **b** and **c**.

Arithmetic operators (+, -, *, /, %)

+ addition
- subtraction
* multiplication
/ division
% module

Module is the operation that gives the remainder of a division of two integer values.

Example `f=13%5` // **f=?**

COMPOUND ASSIGNATION OPERATORS

(+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

+= is assign with addition

value += increase; is equivalent to **value = value +increase;**

s+=1; means **s=s+1;**

a -= 5; is equivalent to **a = a - 5;**

a /= b; is equivalent to **a = a / b;**

price *= units + 1; is equivalent to **price = price * (units+ 1);**

and the same for all other operations.

increment and decrement

increment operator (++) and the *decrement* operator (--).

They increase or reduce by 1 the value stored in a variable.

They are equivalent to +=1 and to -=1, respectively.

`a++;` OR `++a;`

`a+=1;`

`a=a+1;`

are all equivalent in its functionality: the three increase by 1 the value of **a**.

`a--;` OR `--a;`

`a-=1;`

`a=a-1;`

are all equivalent in its functionality: the three decrease by 1 the value of **a**.

Additional operators (`==`, `!=`, `>`, `<`, `>=`, `<=`)

evaluate a comparison between two expressions

Is which replies bool data type (true or false)

`==` Equal

`!=` Different or not equal

`>` Greater than

`<` Less than

`>=` Greater or equal than

`<=` Less or equal than

Suppose that **a=2**, **b=3** and **c=6**,

`(a == 5)` return **false**.

`(a*b >= c)` return **true**

`(b+4 > a*c)` return **false**

`((b=2) == a)` return **true**.

`(7 == 5)` return **false**.

`(5 > 4)` return **true**.

`(3 != 2)` return **true**.

`(6 >= 6)` return **true**.

`(5 < 5)` return **false**.

Operator =(one equal sign) is not the same as operator ==(two equal signs),

Logic operators (!, &&, ||).

Operator ! is equivalent to Boolean operation NOT

- **!(5 == 5)** returns **false**
- **!(6 <= 4)** returns **true**

&& and || are used when evaluating two expressions to obtain a single result.

First Operand a	Second Operand b	result a && b	result a b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

((5 == 5)&&(3 > 6))returns **false**(*true && false*).

((5 == 5)|| (3 > 6))returns **true**(*true || false*).

Conditional operator (?).

Evaluates an expression and returns a different value according to the evaluated expression, depending on whether it is *true* or *false*.

Its format is: ***condition ? result1 : result2***

If ***condition*** is true the expression will return ***result1***, if not it will return ***result2***.

7==5 ? 4 : 3 returns **3** since **7** is not equal to **5**.

7==5+2 ? 4 : 3 returns **4** since **7** is equal to **5+2**.

5>3 ? a : b returns **a**, since **5** is greater than **3**.

a>b ? a : b returns the greater one, **a** or **b**.

sizeof()

This operator accepts one parameter, that can be either a variable type or a variable itself and returns the size in bytes of that type or object:

a = sizeof (char);

This will return **1** to **a** because **char** is a one byte long type.

Output (cout)

The **cout** stream is used in conjunction with the overloaded operator `<<` (a pair of "less than" signs).

```
cout << "Output sentence"; // prints Output sentence on screen
```

```
cout << 120; // prints number 120 on screen
```

```
cout << x; // prints the content of variable x on screen
```

The `<<` operator is known as *insertion operator* since it inserts the data that follows it into the stream that precedes it.

```
cout << "Hello"; // prints Hello on screen
```

```
cout << Hello; // prints the content of Hello variable on screen
```

```
cout << "Hello, " << "I am " << "a C++ sentence"; // print Hello, I  
am a C++ sentence
```

INPUT (CIN).

Handling the standard input in C++ is done by applying the overloaded operator of *extraction* (>>) on the `cin` stream

For example:

```
int age; // first declare memory
```

```
cin >> age;
```

`cin` can only process the input from the keyboard once the RETURN key has been pressed.

//Take number form outside then print it with its double

```
#include <iostream.h>
```

```
int main ()
```

```
{
```

```
int i; //local variable
```

```
cout << "Please enter an integer value: ";
```

```
cin >> i;
```

```
cout << "The value you entered is " << i;
```

```
cout << " and its double is " << i*2 << ".\n";
```

```
return 0;
```

```
}
```

//Take number form outside then print it and display a doubled value in global variable

#include <iostream.h>

int i; //global variable

int main ()

{

cout << "Please enter an integer value: ";

cin >> i;

cout << "The value you entered is " << i;

cout << " and its double is " << i*2 << ".\n";

return 0;

}

CONTROL STATEMENT

The if Statement

- The complete form of the **if** statement is
 - If(*expression*)
statement; //single statements.
- If we have multiple statement or block of statement we use {}
 - If(*expression*)
 {
 statement1;
 statement2;
 statementn;
 }

The nested if Statement

- The complete form of the **if** statement is

- If(*expression*)

```
{  
    If(expression)  
    {  
        If(expression)  
        {  
            statement1;  
            statement2;  
            statementn;  
        }  
    }  
}
```

EXAMPLE OF IF STATEMENT

//example for if statement

```
#include<iostream.h>
Int main ()
{
    int x ;
    cout<<"enter the
number\n";
    cin>>x;
    if (x>=0)
        cout<<"the number you
enter is above zero and";
    cout<<" the number is \t"<<x;
    return 0;
}
```

• //example for. nested if statement

```
#include<iostream.h>
Int main ()
{
    int x ;
    cout<<"enter the number\n";
    cin>>x;
    if (x>=0)
        {if (x>10)
            cout<<"the number you
enter is above ten and the number is
\t"<<x;
        }
    return 0;
}
```

IF ELSE STATEMENT

The complete form of the if else statement is

- For single statement in both case

```
if(expression) statement;
```

```
    else statement;
```

- For block statement only in if case

```
if(expression)
```

```
{
```

```
    statement;
```

```
    statement1;
```

```
    statement n;
```

```
}
```

```
else statement;
```

For block statement in both case

```
if(expression)
```

```
{
```

```
    statement;
```

```
    statement1;
```

```
    statement n;
```

```
}
```

```
Else
```

```
{
```

```
    statement;
```

```
    statement1;
```

```
    statement n;
```

```
}
```

NESTED IF ELSE STATEMENT

```
if(condition1)
{
    if(condition2)
        statement1;
    if(condition3)
        statement2; // this if
    else
        statement3; // is associated
with this else
}
else
    statement4; // associated with
if(expression1)
```

The if-else-if Ladder

A common programming construct. It looks like this:

```
if(condition)
    statement;
else if(condition)
    statement;
else if(condition)
    statement;
.
.
.
else
    statement;
```

```
// Demonstrate an if-else-if ladder.  
#include <iostream.h>  
int main()  
{  
    int x;  
    for(x=0; x<6; x++) {  
        if(x==1)  
            cout << "x is one\n";  
        else if(x==2)  
            cout << "x is two\n";  
        else if(x==3)  
            cout << "x is three\n";  
        else if(x==4)  
            cout << "x is four\n";  
        else  
            cout << "x is not between 1 and 4\n";  
    }  
    return 0;  
}
```

THE SWITCH STATEMENT

The general form of the switch statement is

```
switch(expression) {  
    case constant1:  
        statement sequence  
    break;  
    case constant2:  
        statement sequence  
    break;  
    case constant3:  
        statement sequence  
    break;  
    .  
    .  
    .}  
default:  
    statement sequence
```

The **switch** expression must evaluate to either a character or an integer value. (Floating-point expressions, are not allowed.)

break stops the execution of code within a **switch**.

The **default** statement sequence is performed if no matches are found. & is optional

There are four important things to know about the switch statement:

- I. The switch differs from the if in that switch can test only for equality (i.e., for matches between the switch expression and the case constants), whereas the if conditional expression can be of any type.
- II. No two case constants in the same switch can have identical values. Of course, a switch statement enclosed by an outer switch may have case constants in common.
- III. A switch statement is usually more efficient than nested ifs.
- IV. The statement sequences associated with each case are *not* blocks. However, the entire switch statement *does* define a block.

```
// Demonstrate the switch using a simple "help" program.
#include <iostream>
int main()
{int choice;
  cout << "Help on:\n\n 1. for\n";
  cout << "2. if\n";
  cout << "3. switch\n\n";
  cout << "Enter choice (1-3): ";
  cin >> choice;
  cout << "\n";
  switch(choice)
  {
    case 1: cout << "for is C++'s most versatile loop.\n";
            break;
    case 2: cout << "if is C++'s conditional branch statement.\n";
            break;
    case 3: cout << "switch is C++'s multiway branch statement.\n";
            break;
  default:
    cout << "You must enter a number between 1 and 3.\n";
  }
return 0;
}
```

THE FOR LOOP

The general form of the for loop for repeating a single statement is

for(*initialization*; *expression*; *increment*)
***statement*;**

initialization is usually an assignment that sets the initial value of the *loop control variable*,

a block, the general form is

for(*initialization*; *expression*; *increment*)
{
statement sequence
}

expression is a conditional expression that determines whether or not the loop will repeat.

increment defines the amount by which the loop control variable will change each time the loop is repeated.

Notice that these three major sections of the loop must be separated by semicolons.

```
/*example in for loop calculate sqrt of number b/n 1 to 100 */  
#include <iostream>  
#include <cmath>  
using namespace std;  
int main()  
{  
int num;  
double sq_root;  
for(num=1; num < 100; num++) {  
sq_root = sqrt((double) num);  
cout << num << " " << sq_root << '\n';  
}  
return 0;  
}
```

```
/*example in for loop display the number b/n  
100 & -100 with 5 gaps */
```

```
#include <iostream>  
using namespace std;  
int main()  
{  
int i;  
for(i=100; i >= -100; i = i-5)  
cout << i << '\n';  
return 0;  
}
```

THE WHILE LOOP

Another loop is the while. The general form of the while loop is
while(*expression*)

{ *statement*; }

```
//print all character
#include <iostream>
using namespace std;
int main()
{
    unsigned char ch;
    ch = 32;
    while(ch)
    {
        cout << ch<<"\t";
        ch++;
    }
    return 0;
}
```

The *expression* defines the condition that controls the loop, and it may be any valid expression.

where *statement* may be a single statement or a block of statements.

Block statement

THE DO-WHILE LOOP

The general form of the do-while loop is // The following program loops until the number 100 is entered.

```
do
{
    statements;
}
while(expression);
```

```
#include <iostream>
using namespace std;
int main()
{
    int num;
    do {
        cout << "Enter a number (100 to stop): ";
        cin >> num;
    }
    while(num != 100);
    return 0;
}
```

EXAMPLE

```
#include <iostream.h>
    char char1; // first character
    char char2; // second character
    char char3; // third character
main()
{
    char1 = 'A';
    char2 = 'B';
    char3 = 'C';
    cout << char1 << char2 << char3 << " reversed is " <<
    char3 << char2 << char1 << "\n";
    return (0);
}
```

```
// Computing the average of numbers
#include <iostream>
using namespace std;
int main()
{
    int x, count = 0;
    float sum = 0.0;
    cout << "Please enter some integers:\n"
    "(Break with any letter)"
    << endl;
    while( cin >> x )
    {
        sum += x;
        ++count;
    }
    cout << "The average of the numbers: " << sum / count << endl;
    return 0;
}
```

```
// Definition and use of variables
#include <iostream>
using namespace std;
int gVar1; // Global variables,
int gVar2 = 2; // explicit initialization
int main()
{
char ch('A'); // Local variable being initialized
// or: char ch = 'A';
cout << "Value of gVar1: " << gVar1 << endl;
cout << "Value of gVar2: " << gVar2 << endl;
cout << "Character in ch: " << ch << endl;
int sum, number = 3; // Local variables with
// and without initialization
sum = number + 5;
cout << "Value of sum: " << sum << endl;
return 0;
}
```

```
// Circumference and area of a circle with //radius 2.5
#include <iostream>
using namespace std;
const double pi = 3.141593;
int main()
{
double area, circuit, radius = 1.5;
area = pi * radius * radius;
circuit = 2 * pi * radius;
cout << "\nTo Evaluate a Circle\n" << endl;
cout << "Radius: " << radius << endl
<< "Circumference: " << circuit << endl
<< "Area: " << area << endl;
return 0;
}
```

```
// Demonstration of compound assignments
#include <iostream.h>
#include <iomanip.h>
int main()
{
float x, y;
cout << "\n Please enter a starting value: ";
cin >> x;
cout << "\n Please enter the increment value: ";
cin >> y;
x += y;
cout << "\n And now multiplication! ";
cout << "\n Please enter a factor: ";
cin >> y;
x *= y;
cout << "\n Finally division.";
cout << "\n Please supply a divisor: ";
cin >> y;
x /= y;
cout << "\n And this is " << "your current lucky number: " << x << endl;
return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
cout << "\nSize of Fundamental Types\n"
<< " Type Number of Bytes\n"
<< "-----" << endl;
cout << " char: " << sizeof(char) << endl;
cout << " short: " << sizeof(short)<< endl;
cout << " int: " << sizeof(int) << endl;
cout << " long: " << sizeof(long) << endl;
cout << " float: " << sizeof(float)<< endl;
cout << " double: " << sizeof(double)<<endl;
cout << " long double: " << sizeof(long double)
<< endl;
return 0;
}
```

```

#include <iostream>
#include <climits> // Definition of
//INT_MIN,CHAR_MIN ...
using namespace std;
int main()
{
cout << "Range of types int and unsigned int"<< endl;
cout << "Type          Minimum          Maximum"
<< endl
<< "-----" <<
endl;
cout << "int " << INT_MIN << " "
<< INT_MAX << endl;
cout << "unsigned int " << " 0 "
<< UINT_MAX << endl;
return 0;
}

```

```
#include <iostream>
#include <climits> // Definition of //INT_MIN,CHAR_MIN ...
using namespace std;
int main()
{
cout << "Range of types char and unsigned char"<< endl;
cout << "Type      Minimum      Maximum"
<< endl
<< "-----"
<< endl;
cout << "int " << CHAR_MIN << " "
<< CHAR_MAX << endl;
cout << "unsigned int " << " 0 "
<< UCHAR_MAX << endl;
return 0;
}
```

```

#include <iostream.h>
    int result; // the result of the calculations
    char oper_char; // the user-specified operator
    int value; // value specified after the operator
int main ()
{
    result = 0; // initialize the result
    // loop forever (or until break reached)
while (1)
{
    cout << "Result: " << result << '\n';
    cout << "Enter operator and number: ";
    cin >> oper_char;
    if ((oper_char == 'q') || (oper_char == 'Q'))
        break;
    cin >> value;
    if (oper_char == '+')
    {
        result += value;
    }
    else if (oper_char == '-')
    {

```

```
result -= value; }
else if (oper_char == '*')
result *= value;
else if (oper_char == '/')
{
if (value == 0)
{
cout << "Error: Divide by zero\n";
cout << " operation ignored\n";
}
else
result /= value;
}
else
{
cout << "Unknown operator " << oper_char << '\n';
break ;
}
}
return 0;
}
```

```
#include <iostream.h>
int total; // Running total of all numbers so far
int item; // next item to add to the list
int minus_items; // number of negative items
main ()
{
total = 0;
minus_items = 0;
while (1) {
cout << "Enter number to add\n";
cout << " or 0 to stop:";
cin >> item;
if (item == 0)
break;
if (item < 0) {
++minus_items;
continue;
}
total += item;
cout << "Total: " << total << '\n';
}
cout << "Final total " << total << '\n';
cout << "with << minus_items << " negative items omitted\n";
return (0);
}
```

```
//List Fibonacci series
#include <iostream.h>
    int old_number; // previous Fibonacci number
    int current_number; // current Fibonacci number
    int next_number; // next number in the series
main()
{
    // start things out
    old_number = 1;
    current_number = 1;
    cout << "1\n"; // Print first number
    while (current_number < 100)
    {
        cout << current_number << '\n';
        next_number = current_number + old_number;
        old_number = current_number;
        current_number = next_number;
    }
    return 0 ;
}
```

Write a program to tell whether a number is prime.

- A serial transmission line can transmit 960 characters a second. Write a program that will calculate how long it will take to send a file, given the file's size. Try it on a 400MB (419,430,400 byte) file. Use appropriate units. (A 400MB file takes days)
- A leap year is any year divisible by 4 unless it is divisible by 100, but not 400. Write a program to tell whether a year is a leap year.
- Write a program that converts kilometers per hour to miles per hour. $\text{miles} = (\text{kilometers} \times 0.6213712)$
- Write a program that converts Celsius to Fahrenheit.
- Write a C++ program that two defines variables for floating-point numbers and initializes them with the values 123.456 and 76.543
- Then display the sum and the difference of these two numbers on screen.

• 10/18/2026