The Hashemite University

Computer Programming (C++)

For Faculty of Engineering

(110400102)

Lecturer: Alaa Abu-Srhan

alaa abu srhan --email: alaaa@hu.edu.jo

Course Syllabus

Hashemite University
College of Engineering
(3 Credit Hours/Fac. Compulsory)

Course Name:	**Computer Programming**
Course Number:	110400102
Prerequisite:	110108099
Textbook:	**C++ Programming: From Problem Analysis to Program** D.S. Malik, 6th Edition
References	**C++ How to Program**, Paul J. Deitel and Harvey Deitel, Pearson, 4th edition.
Course Description:	This course covers main topics of C++ programming including C++ fundamentals, operations, elements, structured methods, variables, assignment, Input/Output, control structures, functions, arrays, pointer, strings and classes.
Course Learning Outcomes (CLOs):	CLO1: **understand** basic programming structures. (a, c) CLO2: **design** C++ program to perform predefined task (c, k). CLO3: **analyze** written C++ program to predict output (c, k). CLO4: **develop**, **debug** and **run** C++ programs on Visual Studio (k)
Important material	- Lecture notes - References - Internet resources

Major Topics Covered and Schedule:

Topic	Chapter	# Lectures
Introduction to computers and programming languages	Chapter 1	2
Basics of C++ – Data types, variables – Arithmetic expressions, operators, assignment, increment, decrement	Chapter 2	6
Input/ Output Basics	Chapter 3	2
Quiz		
Control Structure I (Selection) – Relational and logical operators – "if, if … else" – Switch Structure	Chapter 4	5
Control Structure II (Repetition) – Loops: "while" Loop, "for" Loop and "do… while" Loop. – Nested control structure	Chapter 5	5
Midterm Exam	March 11, 2019	
Arrays and strings	Chapter 7, 8	4

alaa abu srhan --email: alaaa@hu.edu.jo

– One dimensional Arrays creation, initialization and manipulation – Strings – Multidimensional Arrays		
Homework		
User defined functions – Predefined functions, user defined functions – Value returning functions, void functions – Value Parameters – Reference Variables as Parameters – Value and Reference Parameters and Memory Allocation – Reference Parameters and Value-Returning Functions Scope of an Identifier – Global Variables, Named Constants, Static and Automatic Variables – Function Overloading – Functions with Default Parameters – Recursive function – Arrays as a parameter to function	Chapter 6	8
POINTERS – Pointer Data Type and Pointer Variables – Address of Operator (&) and dereferencing Operator (*) – Pointers with arrays – Pointers as a parameter to functions	Chapter 12	4
In-lab Assignment		

Course Policy

- Course Website (Moodle): http://www.mlms.hu.edu.jo/. Students are asked to check the website regularly for announcements.
- Students are responsible for the reading assignments from the text and handouts
- Students are responsible for following up the lecture materials
- If you miss class, there won't be a makeup test, quiz, etc. and you WILL get a zero unless you have a valid excuse.
- Cheating and plagiarism are completely prohibited.
- If you miss more than 15% of classes you will automatically fail the class.
- **Grading policy:**
    - **Midterm exam: 35%**
    - **Quiz, homework and in-lab Assignment: 25%**
    - **Final exam: 40%**

- **Midterm Exam will be held in March 11, 2018**

Chapter 1

Introduction to computers and programming languages

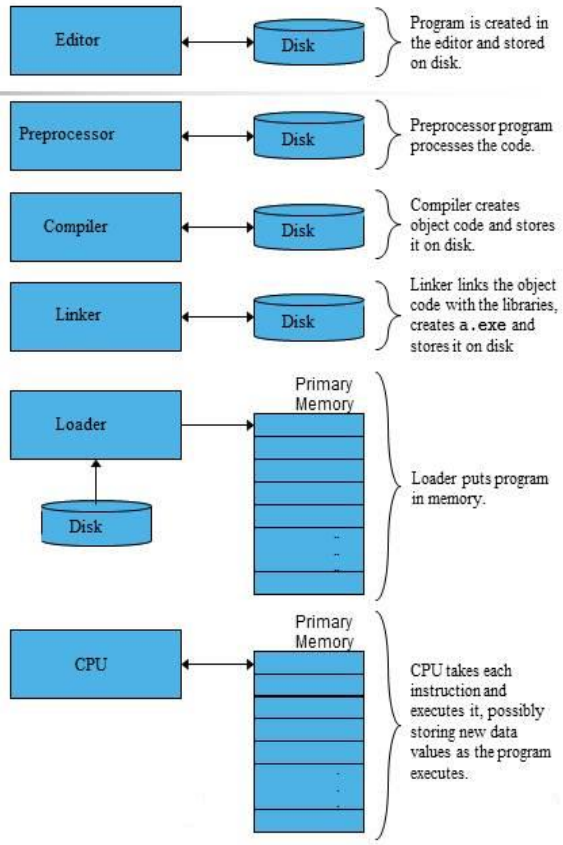
outline:

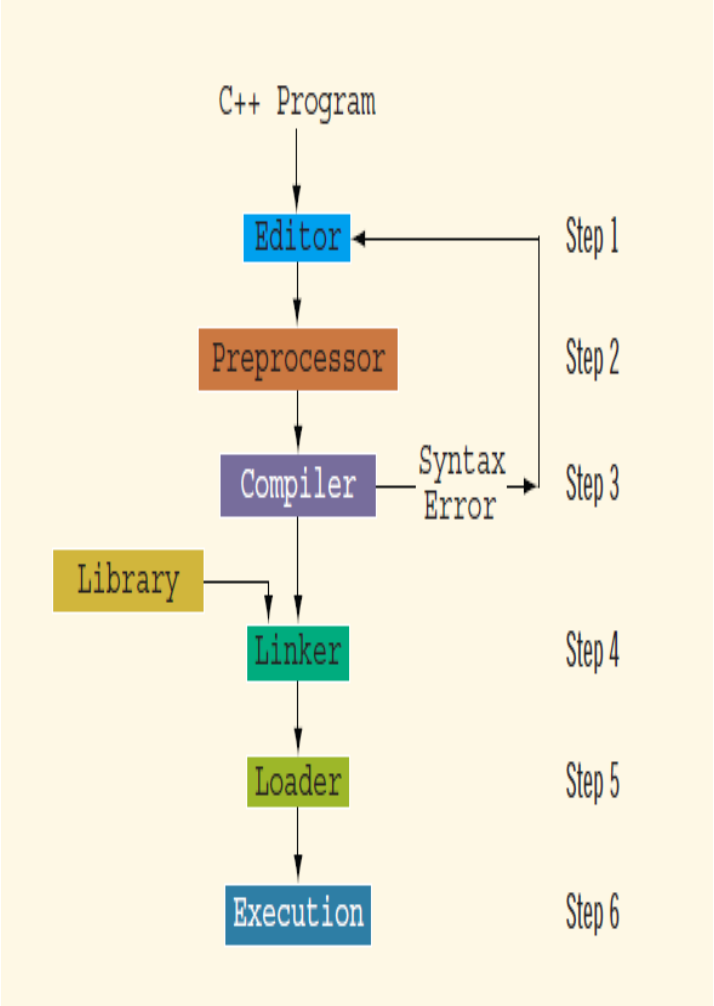
- **The Language of a Computer**
- **The Evolution of Programming Languages**
- **Processing a C++ Program**
- **Programming with the Problem Analysis–Coding–Execution Cycle**

Basics of a Typical C++ Environment

Phases of C++ Programs:

1. Edit (create .h and .cpp files): create the source code or file
2. Preprocess (text replacement, include other files)
3. Compile: creates the object code
4. Link: creates the executable file
5. Load
6. Execute





- **Editor**

Use an editor to create a **source program** in C++.

- **Preprocessor**

Preprocessor directives begin with #, used to include other files.

- **Compiler**

Check that the program obeys the language rules, Translate into machine language (object program)

- **Linker:**

Combines object program with other programs provided by the SDK to create executable code

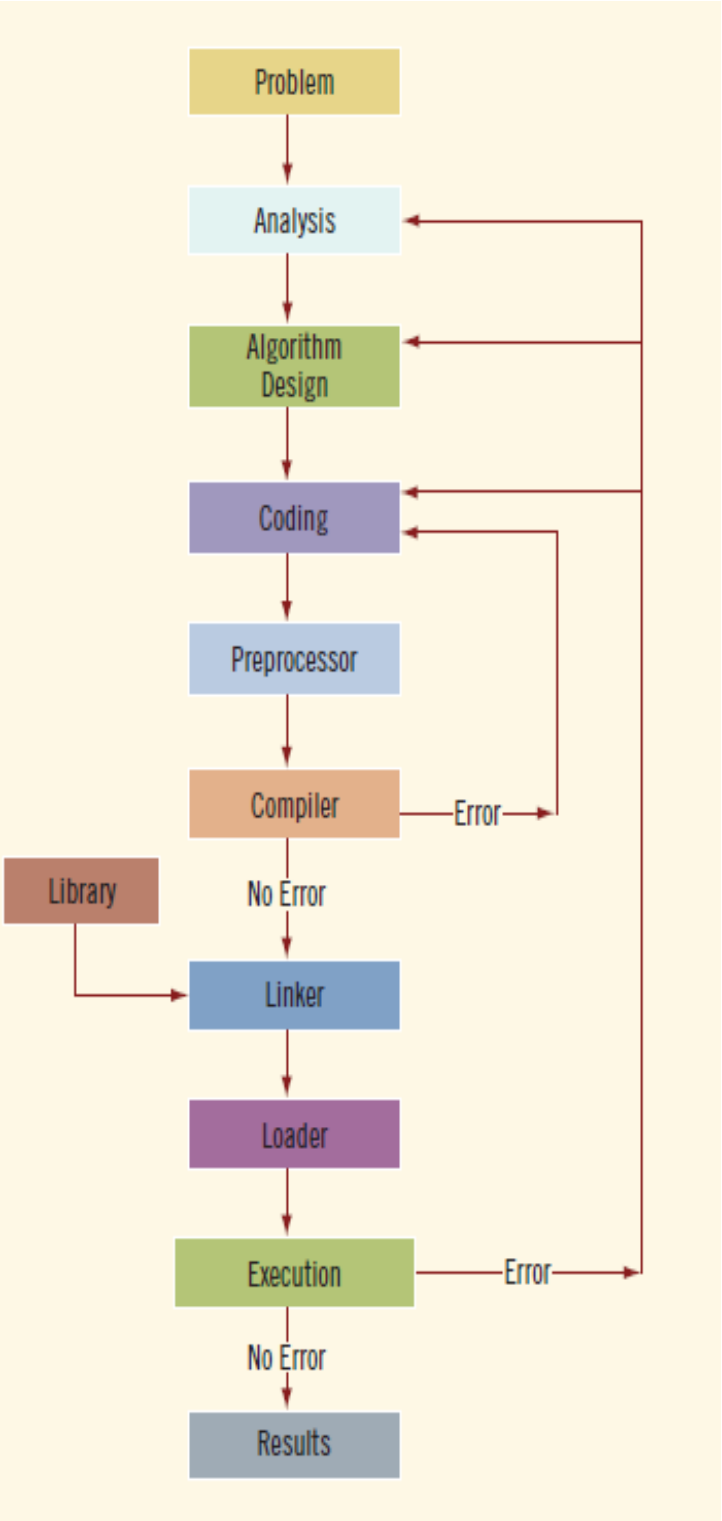
- **Loader:**

Loads executable program into main memory

- **Execute**

The last step is to execute the program

- **Library:** contains prewritten code you can use



Program Development Cycle:

Programming is a process of Problem Solving. Three main steps in good problem solving technique:

1. Analysis: Understanding the problem in depth, determine the inputs, outputs, and operations. Pin points the problems in the current system, suggest solutions and algorithms to solve the problem.
2. Coding: implement the algorithm in a programming language, including:
    a. Editor: writing your programming instructions in an editor, instructions should follow c++ syntax rules. The file saved in extension ".cpp"
    b. Preprocessor: process any statement start with the preprocessor directive # (ex. #include<iostream>)
    c. Compiler: converting (.cpp)code to a machine language (.obj) code
    d. Linker: combine the object file with library object files. The extension of the result file is (.exe)
    e. Loader: A program that loads an executable program into main memory.
    f. Execution: read and execute instruction by the CPU and show the result.
3. Execution: while the program deployed in the environment, maintain the program to solve any new issues and modify it to accommodate with any change.

During this process, errors may appear in different phases. For Examplesyntax errors may appear in the compilation phase, let us to return to the editor in order to fix these errors. Symantec errors (logical errors) may appear during program execution because of wrong ordering of sum operations or errors in the implemented algorithm or there may be some wrong understanding of sub-problem issues and needed to reanalyze.

Program development cycle steps:

1. Problem definition.

- To understand the problem is half the solution.
- Describe it by precise, up to the point statements that will make both analyzing and solving the problem easier and clearer.

2. Problem analysis (understanding).

- Determine the inputs, outputs, and the required operations.
- Explore all possible solutions.
- Pick the easiest, in terms of implementation cost (space, time) one.

3. Algorithm Development

- Algorithm is a procedure that determines the:
  - Actions to be executed.
  - Order in which these actions are to be executed (which is called program control and in industry it is called work flow).
- So, it is a plan for solving the given problem.
- You must validate the developed algorithm, i.e. make sure that it solves the correct problem.
- You must verify the developed algorithm, i.e. make sure that it produces correct results.
- You must check the feasibility (in terms of the needed resources, ease of implementation, ease of understanding and debugging, its expected execution time, etc.) of the developed algorithm.
- **Algorithm Representation:**
  1. Human language
  2. Pseudocode.
  3. Flowcharts (also called UML activity diagram).

4. Coding

- Writing the source code of your solution that is to convert the developed algorithm into code statements of the used language, i.e. C++.
- Some useful tips:
  - Make sure of using correct syntax.
  - Use meaningful identifiers to make your code more readable.
  - Add suitable documentation and comments.
  - Make your code modular or structured as possible.

5.Execution and Testing

- Compilation and debugging.
- Types of errors:
  - Syntax errors (Compile time errors): Errors caught by compiler
  - Logical errors (Runtime errors): Errors which have their effect at execution time
    - Non-fatal logic errors: program runs, but has incorrect output
    - Fatal logic errors: program exits prematurely
- Tracing to verify your program with different sets of inputs.

6. Maintenance

- Not always applicable in education, i.e. highly required in real world jobs.
- Update your code based on:
  - Discovered and reported bugs.
  - Customer feedback to make your application more efficient and flexible.
  - Upgrade the code

Chapter 2 and 3

C++ Basics

Outline:

- Introduction to C++ code.
- Data types.
- Identifiers.
- Casting.
- C++ keywords.

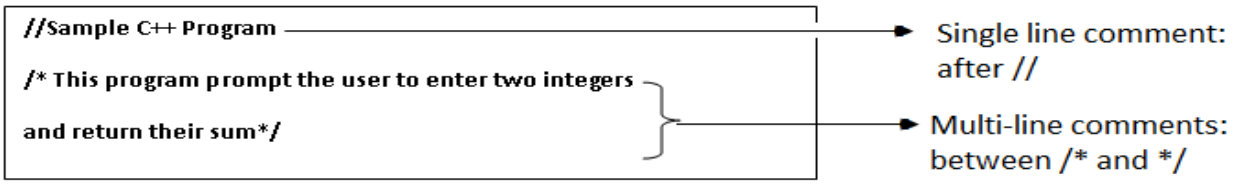
Sample C++ Program

```
//Sample C++ Program
/* This program prompt the user to enter two integers
and return their sum*/

#include <iostream.h>

int  main()
{
    int a1, a2, sum;
    cout<<"Enter the first number:"; //this message will appear to the user
    cout << endl;
    cin >> a1;
    cout<<"Enter the second number: "<< endl;
    cin >> a2;
    sum = a1 + a2;
    cout <<"The sum of " << a1 << " and " << a2 << " = " << sum << "\n";
    return 0;
}
```

```
"C:\Program Files\Microsoft Visual Studio\COMM
Enter the first number:
5
Enter the second number:
6
The sum of 5 and 6 = 11
Press any key to continue
```

Program Explanation:

- Comments:



- You use comments to document programs
- Comments should appear in a program to:
  - Explain the purpose of the program
  - Identify who wrote it
  - Explain the purpose of particular statements

    Note: Comments are not processed by the compiler, feel free to write anything you want.

- Preprocessor directives:
  example:

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

or

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

- Special instructions for the preprocessor.Start with # and usually come at the beginning of the program.
- Tell the preprocessor to perform code substitutions, variables definitions, or conditional compilation in the source code.
- Use **iostream** header file to receive data from keyboard and send output to the screen
  - Contains definitions of two data types:
    - istream: input stream
    - ostream: output stream
  - Has two variables:
    - cin: stands for common input
    - cout: stands for common output

- .h file:
  - Header file which is simply a library that includes the definitions of the used functions within the program, i.e. the frequently used ones to avoid repeating the code.
  - Two types: standard (comes with C++ package) and user defined.
- int main():

```
int  main()
{

    write your code here

 }
```

  - The main part of your program and it represents the entry point of it.
  - The compiler will compile all instructions inside the main().
  - So, place all instruction within the main function, i.e. between its braces { }.
  - main executes when a program is run, other functions execute only when called

- { }:
  - Braces define a block of code.
  - { is the start of this block and } is its end.
- a1, a2, sum:

```
int a1, a2, sum;
```

- Names of variables and they are called identifiers.

- int:
  - Define the data type of the used variable.
  - *int* means an integer variable.
- cout:

```
cout<<"Enter the second number: "<< endl;
```

  - Function defined in the iostream library.
  - An output operator.
  - Tells the compiler to display the string or variable value after the insertion operator << on the screen.
  - cout is always followed by <<.
- cin:

```
cin >> a2;
```

  - An input function defined in the iostream library.
  - Get an input usually from the keyboard.
  - Followed by the extraction operator >> then the variable name in which you want to store the input value.
  - Input type depends on the variable type in which you store the input.
- return 0:
  - Tells the compiler that the main function returns 0 to the operating system in case of successful execution of the program.
- Semicolon ; :

```
int a1, a2, sum;
cout<<"Enter the first number:";
cout << endl;
cin >> a1;
cout<<"Enter the second number: "<< endl;
```

  - Tells the compiler that one instruction line has been terminated.
  - A large set of errors will appear if you forget to put a semicolon at the end of every code line in your program.
  - Any line of code terminated with ; is for the compiler, preprocessor directives *do not* end with ;

C++ Versions:

- Visual C++.
- Visual studio

Identifiers (variable):

- memory location whose content may change during execution

alaa abu srhan --email: alaaa@hu.edu.jo

- Names of variables, constants, and functions.
- Data must be loaded into main memory before it can be manipulated
- Storing data in memory is a two-step process:
    - Instruct computer to allocate memory (define a variable)
    - Include statements to put data into memory (set its value)

Variable declaration:

- A variable is said to be initialized the first time a value is placed into it
- In C++, = is called the assignment operator

Example:

```
#include <iostream>
using namespace std;
int  main()
{
    int x;
    char y;
    float z;
    double h;
    bool g;
    return 0;
}
```

- To declare a variable you must specify **its name** and its **data type.**
  int x ; --> variable name is **x** and its data type is **int**

▪ data type :

1. **int :** Integer Positive and negative integer values (no decimal point is allowed).
2. **floot :** Floating point numbers include integers, decimal point (fractions) and exponents (power of 10).
3. **double:** Same as float but with greater range.
4. **char :** Character is one byte of memory used to save any symbol, alphabet or digit number presented on the keyboard, e.g. :, /, @, d, 5.
5. **bool :** Boolean value is either true or false.

Example:

```
#include <iostream>
using namespace std;
int  main()
{
    int x;
    y=9;
    return 0;
}
```

Error: identifier "y" is undefined

- All variables must be declared before they are being used in the program. If you use a variable not declared the compiler will give you ***a syntax error*.**

Example:

alaa abu srhan --email: alaaa@hu.edu.jo

```
#include <iostream>
using namespace std;
int  main()
{
        y=9;
    int y;
    return 0;
}
```

Error: identifier "y" is undefined

- **syntax error -** make sure to declare variable before the first usage of the variables(declare it then use it)
- You can place declarations in any place within your program, but it is preferred to place them at the beginning of your program.

example:

```
#include <iostream>
using namespace std;
int  main()
{
    int y, z,f;
    float g=9.4,a=5.7;
    return 0;
}
```

- Multiple variables of the same data type can be defined using one statement having the variables name comma separated.

Example:

```
#include <iostream>
using namespace std;
int  main()
{
    int x;
    x=5;
    float x;
    return 0;
}
```

- declare variable twice is **syntax error**.

Example:

```
#include <iostream>
using namespace std;
int main()
{
    int x=10;
    int y =x;
    cout<<y<<endl;
    cout<<x;
return 0;
}
```

```
10
10_
```


alaa abu srhan --email: alaaa@hu.edu.jo

Example:

```
#include <iostream>
using namespace std;
int  main()
{
    int xx;
XX=10;
    return 0;
}
```

Error: identifier "XX" is undefined

- **syntax error** -- C++ is case sensitive, i.e. xx is different from Xx and xX.

You can use anything as an identifier with the following restrictions:

1. Do not use any of C++ keywords, e.g. if, for, int, float, cout, **...**

```
#include <iostream>
using namespace std;
int  main()
{
    int for;
    retu
}
```

Error: expected an identifier

```
#include <iostream>
using namespace std;
int  main()
{
    int if;
    retu
}
```

Error: expected an identifier

```
#include <iostream>
using namespace std;
int  main()
{
    int while   ;
    retu
}
```

Error: expected an identifier

- **syntax error**

2. Never start your identifier with a digit (number) always start it with alphabet or underscore.

```
#include <iostream>
using namespace std;
int  main()
{
    int 2x  ;
    retu
}
```

Error: expected an identifier

variable never start with number !!**(syntax error)**

```
#include <iostream>
using namespace std;
int  main()
{
    int _x  ;
    return 0;
}
```

variable can start with underscore (_) (no error)

```
#include <iostream>
using namespace std;
int  main()
{
    int x2  ;
    return 0;
}
```

variable strat with alphabet and contain number (no error)

3. Do not use white spaces in your identifier, use underscores instead.

```
#include <iostream>
using namespace std;
int  main()
{
    int x g  ;

    return
}
```
Error: expected a ';'

Do not use white spaces
(syntax error)

```
#include <iostream>
using namespace std;
int  main()
{
    int x_g  ;

    return 0;
}
```

use underscores instead of white spaces
(no error)

4. Do not use special symbols in your identifier such as #, $, etc.

```
#include <iostream>
using namespace std;
int  main()
{
    int x@g  ;

    retur
}
```
Error: unrecognized token

```
#include <iostream>
using namespace std;
int  main()
{
    int #xg  ;

    retu
}
```
Error: '#' not expected here

```
#include <iostream>
using namespace std;
int  main()
{
    int xg!  ;

    return
}
```
Error: expected a ';'

- **syntax error**

5. Do not use any of the operators (arithmetic, logical, etc.) in your identifier such as +, =, etc

```
#include <iostream>
using namespace std;
int  main()
{
    int x+g  ;

    retur
}
```
Error: expected a ';'

```
#include <iostream>
using namespace std;
int  main()
{
    int xg>  ;

    return
}
```
Error: expected a ';'

```
#include <iostream>
using namespace std;
int  main()
{
    int =xg ;

    retur
}
```
Error: identifier "xg" is undefined

- **syntax error**

Naming Identifiers

- Identifiers can be <u>self-documenting</u>:
  - CENTIMETERS_PER_INCH
- Avoid <u>run-together words</u> :
  - annualsale
  - Solution:
    - Capitalizing the beginning of each new word: annualSale
    - Inserting an underscore just before a new word: annual_sale

C++ Keywords

- Words reserved by C++.
- Always lower case, should not be used as identifiers.

C++ Keywords

Keywords common to the C and C++ programming languages

auto	**break**	**case**	**char**	**const**
continue	**default**	**do**	**double**	**else**
enum	**extern**	**float**	**for**	**goto**
if	**int**	**long**	**register**	**return**
short	**signed**	**sizeof**	**static**	**struct**
switch	**typedef**	**union**	**unsigned**	**void**
volatile	**while**			

C++ only keywords

asm	**bool**	**catch**	**class**	**const_cast**
delete	**dynamic_cast**	**explicit**	**false**	**friend**
inline	**mutable**	**namespace**	**new**	**operator**
private	**protected**	**public**	**reinterpret_cast**	
static_cast	**template**	**this**	**throw**	**true**
try	**typeid**	**typename**	**using**	**virtual**
wchar_t				

cout example:

- **Prompt lines**: executable statements that inform the user what to do

```
cout << "Please enter a number between 1 and 10 and " << "press the return key" ;
cout<<endl;
cin >> num;
```

- Always include prompt lines when input is needed from users

alaa abu srhan --email: alaaa@hu.edu.jo

Example1

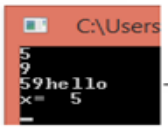
```
#include <iostream>
using namespace std;
int  main()
{
    int x =5;
    int y=9;
    cout<<x<<endl;
    cout<<y<<endl;
    cout<<x<<y;
    cout<<"hello"<<endl;
    cout<<"x=  "<<x<<endl;
    return 0;
}
```

cout use to output x value and then make new line

cout use to output y value and then make new line

cout use to output x value then y value at the same line

cout use to output hello (hello is string so it will output on screen as it is)



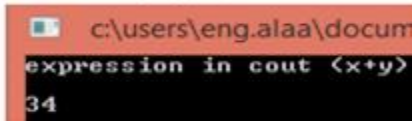
there is no endl between cout<<x<<y; and cout<<"hello"; so it will print in one line

cout use to ouput "x= "string then value of x --> x= 5

we can use one cout sentence to output multi things(strings and variable vlaues) separated with <<

Example2

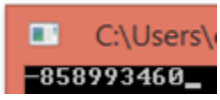
```
#include <iostream>
using namespace std;
void main()
{
    int x;
    x=29;
    int y=5;
    cout<<"expression in cout (x+y)";
    cout<<endl;
    cout<<endl;
    cout<<x+y<<endl;
}
```



- when we found cout<<endl; twice this mean empty line

Example3

```
#include <iostream>
using namespace std;
int  main()
{
    int x;
    cout<<x;
    return 0;
}
```



- use any variable before assignment(give the variable value) is **logical error**

Example5

```
#include<iostream>
using namespace std;
void main()
{
double d3;
d3+=4;
cout<<d3;
}
```

```
C:\Users\eng
-9.25596e+061
```

- logical error

Data Types:

1. int :

Example1:

```
#include <iostream>
using namespace std;
int  main()
{
    int x ;
    x=3;
    int y= -5;
    cout<<x<<endl;
    cout<<y;
    return 0;
}
```

```
C:\Us
3
-5_
```

cout : use to output variable value or string
endl : new line
cout followed always with <<

- Positive and negative integer values (no decimal point is allowed).

Example2:

```
#include <iostream>
using namespace std;
int  main()
{
    int x ;
    x=3;
    int y= 3.5;
    cout<<x<<endl;
    cout<<y;

    return 0;
}
```

```
3
3_
```

you can assign value to a variable when you declare it, or after declaration

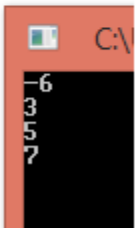
storing 3.5 in int ➔ ignore everything after the decimal point without rounding

- What will happen if you save a float number in an integer?
  result : ignore everything after the decimal point without rounding

Example3

alaa abu srhan --email: alaaa@hu.edu.jo

```
#include <iostream>
using namespace std;
int  main()
{
    int signed  x =-6 ;
    int unsigned y= 3;
    int long z=5;
    int short h=7;
    cout<<x<<endl;
    cout<<y<<endl;
    cout<<z<<endl;
    cout<<h<<endl;
    cin.get();
    return 0;
}
```



short: 2 bytes signed integer.

long: 4 bytes signed integer.

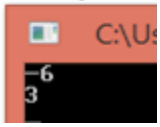
signed: 4 bytes the most significant bit is reserved for sign.

unsigned:4 bytes no negative numbers.

- **note:** The default of int type depends on the used compiler and the operating system under which this compiler is work:
  - int a; // here 'a' is by default signed long integer under VC++ (works under windows).
  - int a; // here 'a' is by default signed short integer under Turbo C++ (works under DOS)

Example4

```
#include <iostream>
using namespace std;
int  main()
{
    int signed long x =-6 ;
    int unsigned  short y= 3;
    cout<<x<<endl;
    cout<<y<<endl;
    return 0;
}
```



- you can combine signed and long or short for the same variable, and you can combine signed and long or short for the same variable.

Example5

```
#include <iostream>
using namespace std;
int  main()
{
    int signed unsigned  x =-6 ;

    cout<<x<<endl;
    cin.get();
    return 0;
}
```

Error: invalid combination of type specifiers

```
#include <iostream>
using namespace std;
int  main()
{
    int short long  x =-6 ;

    cout<<x<<endl;
    cin.get();
    return 0;
}
```

Error: invalid combination of type specifiers

- **syntax error**
- You cannot combine signed and unsigned for the same variable and you cannot combine long and short for the same variable.

2. float :

- Floating point numbers include integers, decimal point (fractions) and exponents (power of 10).
- Memory size: 4 bytes.

Example1

```
#include <iostream>
using namespace std;
int  main()
{
    float x=3.5;
    float y=3;
    cout<<x<<endl;
    cout<<y<<endl;
    cin.get();
    return 0;
}
```

decimal point (fraction)

integer

```
3.5
3
```

Example2

```
#include <iostream>
using namespace std;
int  main()
{
            float x=-5.6;
            cout<<x;

    return 0;
}
```

```
-5.6
```

- float Include both positive and negative values.

example3

```
#include <iostream>
using namespace std;
int  main()
{
    float x=21.34567;
    float y=21.34562;
    cout<<x<<endl;
    cout<<y<<endl;
    return 0;
}
```

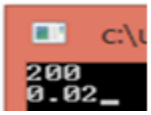
```
21.3457
21.3456
```

- Float has decimal point precision up to 6 digits. What happen if you enter more than 6 digits ? (truncation) 6 digit will store and last digit(in the right) must be rounded.

Example 4

alaa abu srhan --email: alaaa@hu.edu.jo

```
#include <iostream>
using namespace std;
void main()
{
    float x;
    x=2e2;
    cout<<x<<endl;

    float d;
d=2e-2;
    cout<<d;
}
```



Example6

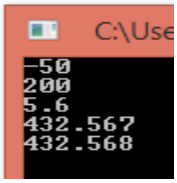
```
#include <iostream>
using namespace std;
int  main()
{
            float x=2.5e2;
            cout<<x;
        return 0;
}
```

- 2.5*100=250

3. double:
   - Same as float but with greater range.
   - Syntax: double a;
   - Its size is 8 bytes.

Example1

```
#include <iostream>
using namespace std;
int  main()
{
            double  x=-50;
            double y=2e2;
            double z=5.6;
            double f=432.5671;
            double ff=432.5677;
            cout<<x<<endl;
            cout<<y<<endl;
            cout<<z<<endl;
            cout<<f<<endl;
            cout<<ff<<endl;
        return 0;
}
```



Note:

- Maximum number of significant digits (decimal places) for float values: 6 or 7
- Maximum number of significant digits for double: 15
- Precision: maximum number of significant digits
  - Float values are called single precision
  - Double values are called double precision

4. char :

- Character is **one byte** of memory used to save any symbol, alphabet or digit number presented on the keyboard, e.g. :, /, @, d, 5.
- Syntax: char a;
- Each character is enclosed in single quotes
  - 'A', 'a', '0', '*', '+', '$', '&'
- A blank space is a character written ' ', with a space left between the single quotes

Example1

```
#include <iostream>
using namespace std;
int  main()
{
    char x='a';
    char y='9';
    cout<<x<<endl;
    cout<<y<<endl;
    return 0;
}
```

- only and only one character can be stored in a char variable

Example2

```
#include <iostream>
using namespace std;
int  main()
{
   char x='abcd';
   cout<<x<<endl;
   return 0;
}
```

- only and only one character can be stored in a char variable--> if you initialize char variable with more than one character, then only the last one will be taken and stored in your variable.

Example3

```
#include <iostream>
using namespace std;
void main()
{
    char d;
cin>>d;
    cout<<d;

}
```

- if you enter it from the keyboard (using cin) you must enter only one character. If you enter more than one character for a char variable only the first one will be taken and stored in your variable.

Example 4

```
#include <iostream>
using namespace std;
int  main()
{
    char x = "a";
                Error: identifier ""a"" is undefined
            cout<<x<<endl;
        return 0;
}
```

- Syntax error since "a" is not only one character (a+null value)

character coding:

- Character coding is used to store the values of characters in char variable in C++ since computers only knows binary numbers.
- Based on this coding every character has a number value.
- Coding types:
  - ASCII (American Standard Code for Information Interchange).
  - EBCDIC (Extended Binary Coded Decimal Information Code).
  - Unicode : used mainly for Internet applications.
  - ASCII is the most dominant.

ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

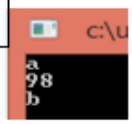
Example5

```
#include <iostream>
using namespace std;
int  main()
{
    char x = 'a';
    char y=97;
            cout<<x<<endl;
            cout<<y<<endl;
        return 0;
}
```

- y has char data type so it will store character, 97 is the value of **'a' in** ASCI.

Example 6

```
#include <iostream>
using namespace std;
void main()
{
    char x;
    x=97;
    cout<<x<<endl;
    cout<<x+1<<endl;
    char d='a';
    d=d+1;
    cout<<d;

    cin.get();
}
```



x is char so it will print a

use any Arithmetic operators(+,-,*,/) with x inside cout statment will print the result as ASKI

d=d+1 the result will save in d , d has char data type so d will be b

note: 'a' has the value of 97 in ASCII, 'b' has the value of 98 in ASCII

5. bool :

- Boolean value is either true or false.
- Size = 1 byte.
- Syntax: bool a;

Example1

```
#include <iostream>
using namespace std;
int  main()
{
    bool x='a';
    bool y=8;
    bool z="hi";
    bool g=true;
    cout<<x<<endl;
    cout<<y<<endl;
    cout<<z<<endl;
    cout<<g<<endl;
    cin.get();
    return 0;
}
```

- True is any non-zero value or simply true keyword.
- By any non-zero value we mean:
  - Any number other than zero, e.g. 9, -2, -100, 3.4, etc.
  - Any character (including white spaces) or string.

  string is any thing between ", character put between ' e.g "hello" and 'a'

- When you print a bool value on the screen, i.e. using cout statement, it will be either 1 when true or 0 when false.

Example2

```
#include <iostream>
using namespace std;
int  main()
{
    bool x=0;
    bool y=false;
    cout<<x<<endl;
    cout<<y<<endl;
    cin.get();
    return 0;
}
```

False is only zero value or `false` keyword

- When you print a bool value on the screen, i.e. using cout statement, it will be either 1 when true or 0 when false.

Example3

```
#include <iostream>
using namespace std;
int  main()
{
    bool y=False;

    cout<<y

    return 0;
}
```

Error: identifier "False" is undefined

- **syntax error** --> **False** is not keyword , also **True** is not keyword (False and True are variables)

Example4

```
#include <iostream>
using namespace std;
int  main()
{
    bool y=b;
    cout<<y

    return 0;
}
```

Error: identifier "b" is undefined

- **syntax error**--> note that b is not character (character between ') b is undefined variable .

Example5

```
#include <iostream>
using namespace std;
int  main()
{
    bool y="false";
    cout<<y<<endl;
    return 0;
}
```

- "false" is string, not false keyword.

Data Types Range

Name	Bytes (Compiler dependent)	Description	Range (depends on number of bytes)
char	1	Character or 8 bit integer.	signed: -128 to 127 unsigned: 0 to 255
bool	1	Boolean type. It takes only two values.	true or false
short	2	16 bit integer.	signed: -32768 to 32767 unsigned: 0 to 65535
long	4	32 bit integer.	signed:-2147483648 to 2147483647
int	2/4	Integer. Length depends on the size of a 'word' used by the Operating system. In MSDOS a word is 2 bytes and so an integer is also 2 bytes.	Depends on whether 2/4 bytes.
float	4	Floating point number.	--
double	8	double precision floating point number.	--

Allocating Memory with Constants and Variables

- Named constant: memory location whose content can't change during execution
- Syntax to declare a named constant:

  const dataType identifier = value;
- In C++, **const** is a reserved word

Example:

```
const double Conversion=2.54;
const int NoOfStudents = 20;
const char Blank = ' ';
```

casting:

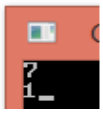
- **Casting** is converting the data type of *the value* of a specified variable to another data type *temporarily*.
- **Two types of casting:**
  - Implicit casting: done by the compiler implicitly.
  - Explicit casting: need to be coded explicitly by the programmer (to force casting).

Implicit casting:

```
#include <iostream>
using namespace std;
int  main()
{
    int y= 7.5;
    cout<<y<<endl;
    bool x= 10;
    cout<<x;
    cin.get();
    return 0;
}
```

implicitly the compiler will convert 7.5 into 7

implicitly the compiler will convert 10 to true(1)



- Implicit casting is done when needed by the compiler not when needed by you.

Explicit casting:

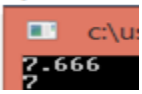
- Explicit casting types:
  1) C-style explicit casting.

```
#include <iostream>
using namespace std;
int  main()
{
    int a = 100;
bool b = (bool)a;
bool c = bool(a);
cout<<a<<endl;
cout<<b<<endl;
cout<<c<<endl;
    return 0;
}
```

- Just put the data type name that you want to convert to it before the variable name (or value) that you want to convert and place the parenthesis correctly.

example:

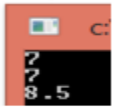
```
#include <iostream>
using namespace std;
void main()
{
float  a=7.666;
int b=(int)a;
    cout<<a<<endl;
    cout<<b<<endl;
}
```



- Note that You cast the type of the value of the variable only. So, you do not change the original data type of the variable through casting.
- data type of a will stay the same (float data type).

Example:

```
#include <iostream>
using namespace std;
void main()
{
int a=7.666;
float b=(float)a;
    cout<<a<<endl;
    cout<<b<<endl;
     b=b+1.5;
    cout<<b<<endl;
}
```



- implicitly the compiler will convert 7.666 into 7.
- b has float data type which contain value of x after casting it to float, and 7 will be stored in b

Example:

```
#include <iostream>
using namespace std;
int  main()
{
    char x='a';
    cout<<x <<"->"<<int(x)<<endl;
    x='b';
    cout<<x <<"->"<<int(x)<<endl;
    x='A';
    cout<<x <<"->"<<int(x)<<endl;
    x='B';
    cout<<x <<"->"<<int(x)<<endl;

    return 0;
}
```

```
a->97
b->98
A->65
B->66
```

2) C++ casting operators.

- There are 4 casting operators in C++, we will take only one of them.

Syntax: static_cast <new_type> (expression)

Where:

- new_type: is the type you want to convert to.
- expression: is the variable name or the expression that you want to cast its value.

Example1

```
#include <iostream>
using namespace std;
void main()
{
int a=7.666;
float b=static_cast<float>(a);
    cout<<a<<endl;
    cout<<b<<endl;
}
```

```
?
?
```

Example2

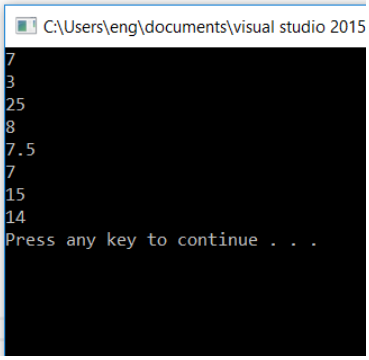
```
#include <iostream>
using namespace std;
int main()
{
    cout<<static_cast<bool>('b');


}
```

```
1
```

Example3

alaa abu srhan --email: alaaa@hu.edu.jo

```cpp
# include <iostream>
using namespace std;
# include <string>
void main()
{
    cout<<static_cast<int>(7.9)<<endl;
    cout << static_cast<int>(3.3)<<endl;
    cout << static_cast<double>(25)<<endl;
    cout << static_cast<double>(5+3)<<endl;
    cout << static_cast<double>(15)/2<<endl;
    cout << static_cast<double>(15/ 2)<<endl;
    cout << static_cast<int>(7.8+ static_cast<double>(15) / 2)<<endl;
    cout << static_cast<int>(7.8 + static_cast<double>(15 / 2))<<endl;
    system("pause");
}
```



Example: Write a program to perform athematic operations (*,+,-,/) for two numbers

```
# include <iostream>
using namespace std;
void main()
{
    int x ;
    int y ;
    int mult, sum, dev, sub;
    cout << "please enter first integer number" << endl;
    cin >> x;
    cout << "please enter second integer number" << endl;
    cin >> y;
    mult = x*y;
    sub = x - y;
    sum = x + y;
    dev = x / y;
    cout << "sum of " << x << " and " << y << "=  " << sum << endl;
    cout << "mult of " << x << " and " << y << "=  " << mult << endl;
    cout << "dev of " << x << " and " << y << "=  " << dev << endl;
    cout << "sub of " << x << " and " << y << "=  " << sub << endl;
    cin>>y;
    cout << "sum of " << x << " and " << y << "=  " << sum << endl;
    cout << "mult of " << x << " and " << y << "=  " << mult << endl;
    cout << "dev of " << x << " and " << y << "=  " << dev << endl;
    cout << "sub of " << x << " and " << y << "=  " << sub << endl;
    system("pause");
}
```

Example: Write a program to compute the area of the rectangle.

```
# include <iostream>
using namespace std;
# include <string>
void main()
{
    double length;
    double width;
    cout << "enter length of the rectangle"<<endl;
    cin >> length;
    cout << "enter width of the rectangle"<<endl;
    cin >> width;
    double area;
    area = length*width;
    cout << " the area of rectangle that has length= " << length;
    cout << " and " << "width=  "<<width<<" is "<<area<<endl;
    system("pause");
}
```

```
C:\Users\eng\documents\visual studio 2015\Projects\firstcode\Debug\firstcode.exe
enter length of the rectangle
12.3
enter width of the rectangle
14.5
 the area of rectangle that has length= 12.3 and width=  14.5 is 178.35
Press any key to continue . . .
```

String Type

- Sequence of zero or more characters enclosed in double quotation marks
- Length of a string is number of characters in it
  - Example: length of "William Jacob" is 13
  - Position of character 'W' is 0
  - Position of character 'J' is 8
- To use the string type, you need to access its definition from the header file string, Include the following preprocessor directive:
  - #include <string>

Example1

```
1  # include <iostream>
2  using namespace std;
3  # include <string>
4  void main()
5  {
6      string name;
7      cin >> name;
8      cout << name<<endl;
9      system("pause");
10 }
```

```
C:\Users\eng\documents\visual studio 2015\P
alaa srhan
alaa
Press any key to continue . . .
```

```
1  # include <iostream>
2  using namespace std;
3  # include <string>
4  void main()
5  {
6      string name;
7      getline(cin, name);
8      cout << name<<endl;
9      system("pause");
10 }
```

```
C:\Users\eng\documents\visual studio 2015\P
alaa srhan
alaa srhan
Press any key to continue . . .
```

- The function **getline** reads until end of the current line
- **cin** skips any leading whitespace characters, reading stops at a whitespace character

Example

```
1   # include <iostream>
2   using namespace std;
3   # include <string>
4   void main()
5   {
6       string FirstName;
7       string LastName;
8       int age;
9       double weight;
10      cout << "enter first name, last name, age, and weight separated by spaces "<<endl;
11      cin >> FirstName>> LastName;
12      cin >> age >> weight;
13
14      cout << "Name: " << FirstName <<"   "<<LastName << endl;
15      cout << "Age: " << age<<endl;
16      cout<< " Weight" << weight << endl;
17      system("pause");
18  }
```

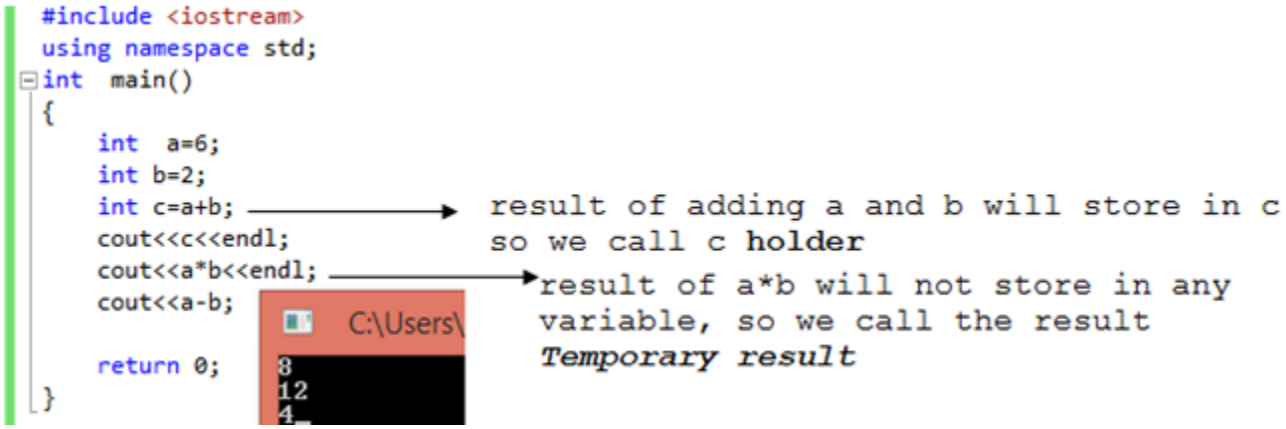
```
C:\Users\eng\documents\visual studio 2015\Projects\firstcode\Debug\firstcode.exe
enter first name, last name, age, and weight separated by spaces
ahmad ali 25 70
Name: ahmad   ali
Age: 25
 Weight70
Press any key to continue . . .
```

Arithmetic Operators

- Addition +
- Subtraction -
- Division /
- Multiplication *
- Modulus %

- Arithmetic expressions: contain values and arithmetic operators
- Operands: the number of values on which the operators will work
  - Operators can be unary (one operand) or binary (two operands)

Example



- the result of any arithmetic operation is either store on variable and we call the variable **holder,** or use directly without store it so we call it **temporary result** .
- Arithmetic operators are binary operators (take two operands).

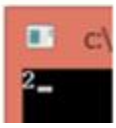
- **Division Operator General Rules:**

we have two case:

1. without store the result **(temporary result).**
   - if one of the operands or both has **float/ double** data type, then the result will be floating point
   - if the two operands has **int** data type, then the result will be integer.
2. using **holder** .
   - if one of the operands or both has **float/ double** data type and the holder data type is **int** , then the result will be **integer.**
   - if one of the operands or both has **float/ double** data type and the holder data type is **float/double**, then the result will be **floating point** .
   - if the two operands has **int** data type, then the result will be integer.

examples:

```
#include<iostream>
using namespace std;
void main()
{
int a=5;
int b=2;
cout<<a/b;

}
```




```
#include<iostream>
using namespace std;
void main()
{
float a=5;
float b=2;
cout<<a/b;

}
```


```
#include<iostream>
using namespace std;
void main()
{
int a=5;
float b=2;
cout<<a/b;

}
```

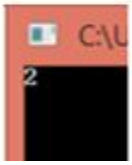

```
#include<iostream>
using namespace std;
void main()
{
float a=5;
int b=2;
cout<<a/b;

}
```


```
#include<iostream>
using namespace std;
void main()
{
int a=5;
int b=2;
float c;
c=a/b;
cout<<c;

}
```


```
#include<iostream>
using namespace std;
void main()
{
int a=5;
int b=2;
int c;
c=a/b;
cout<<c;

}
```

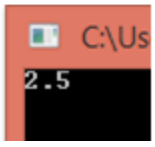

```
#include<iostream>
using namespace std;
void main()
{
int a=5;
float b=2;
float c;
c=a/b;
cout<<c;

}
```


```
#include<iostream>
using namespace std;
void main()
{
int a=5;
float b=2;
int c;
c=a/b;

cin.get();
}
```

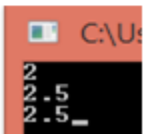



```
#include<iostream>
using namespace std;
void main()
{
float a=5;
int b=2;
int c;
c=a/b;
cout<<c;

}
```

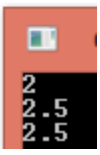

```
#include<iostream>
using namespace std;
void main()
{
float a=5;
int b=2;
float c;
c=a/b;
cout<<c;

}
```




```
#include<iostream>
using namespace std;
void main()
{
float a=5;
float b=2;
int c;
c=a/b;
float d;
d=a/b;
cout<<c<<endl;
cout<<d<<endl;
cout<<a/b;

}
```




```
#include <iostream>
using namespace std;
int  main()
{
    double a = 5/2;
    cout<<a<<endl;
    double b = 5.0/2;
    cout<<b<<endl;
    double c = 5/2.0;
    cout<<c;

    return 0;
}
```



Example

```
#include<iostream>
using namespace std;
void main()
{
int x=9;
int y=4;
cout<<x%y<<endl;
}
```

- % (modulus which finds the remainder)

Example

```
#include<iostream>
using namespace std;
void main()
{
float x=9;
float y=4;
cout<<x%y<<endl;
}
```

float y

Error: expression must have integral or enum type

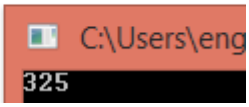
- **syntax error** -->modulus is applied for integer values only.

arithmetic operators precedence:

- *, %, and / have the same priority which are higher than + and – (+ and – have the same priority).
- If multiple operators are combined which have the same priority you start implementation from left to right.
- Remember that parenthesis forces priority.
- For nested parenthesis you start with the most inner parenthesis pair.
- For multiple parenthesis start implementation from left to right.

Example1:

```
#include<iostream>
using namespace std;
void main()
{
int a = 9, b = 30, c = 89, d = 35, x;
     double e = 3, y;
     x = d*a + c%b - b + d/e;
cout<<x<<endl;
}
```



x = d*a + c%b - b + d/e;
x=35*9+89%30-30+35/3
x=315+89%30-30+35/3
x=315+29-30+35/3
x=315+29-30+11.6667
x=325.6667
but x is int so output is 325

Example2

```
#include<iostream>
using namespace std;
void main()
{
int a = 9, b = 30, c = 89, d = 35, x;
     double e = 3, y;
     y= d*a + c%b - b + d/e;
cout<<y<<endl;
}
```

```
C:\Users
325.667
```

y = d*a + c%b - b + d/e;
y=35*9+89%30-30+35/3
y=315+89%30-30+35/3
y=315+29-30+35/3
y=315+29-30+11.6667
y=325.6667
but y is float so output is 325.667

Example3

```
#include<iostream>
using namespace std;
void main()
{
int a = 9, b = 30, c = 89, d = 35, x;
     double e = 3, y;

cout<< d*a + c%b - b + d/e<<endl;
}
```

```
C:\Users\en
325.667
```

d*a + c%b - b + d/e
35*9+89%30-30+35/3
315+89%30-30+35/3
315+29-30+35/3
315+29-30+11.6667
325.6667
since d is int and e is float so output will be float

Example4

```
#include<iostream>
using namespace std;
void main()
{
int a = 9, b = 30, c = 89, d = 35, x;
     double e = 3, y;
    x = d*a + c%b - (b + d/e);
cout<< x<<endl;

}
```

```
C:\Users\eng.ala
302
```

x = d*a + c%b - (b + d/e);
x=35*9+89%30-(30+35/3);
x=35*9+89%30-(30+11.6667)
x=35*9+89%30-41.6667
x=315+89%30-41.6667
x=315+29-41.6667
x=344-41.6667
x=302.333
but x is int so output is 302

Example5

```
#include<iostream>
using namespace std;
void main()
{
int a = 9, b = 30, c = 89, d = 35, x;
     double e = 3, y;
    y = d*a + c%b - (b + d/e);
cout<< y<<endl;
}
```

```
C:\Users\eng.alaa
302.333
```

y = d*a + c%b - (b + d/e);
y=35*9+89%30-(30+35/3);
y=35*9+89%30-(30+11.6667)
y=35*9+89%30-41.6667
y=315+89%30-41.6667
y=315+29-41.6667
y=344-41.6667
y=302.333
but y is float so output is 302.333

Example6

alaa abu srhan --email: alaaa@hu.edu.jo

```
1    # include <iostream>
2    using namespace std;
3    # include <string>
4    void main()
5    {
6        cout<<4%2.5<<endl
7        system("pa
8    }
```

expression must have integral or unscoped enum type

- Use % only with integral data types

Assignment

- = assigns the value on the right hand side to what present on the left hand side.

```
#include <iostream>
using namespace std;
int  main()
{
    int a;
    a=2;
    cout<<a;
    return 0;
}
```

Example1

```
#include <iostream>
using namespace std;
int  main()
{
    int a=2;
    a+=2;
    cout<<a;
    return 0;
}
```

- Assignment expression abbreviations

  a= a +2 ; can be abbreviated as **a+=2 ;** using the addition assignment operator

- Examples of other assignment operators include:

  d -= 4 (d = d - 4)
  n += 4 (n = n + 4)
  e *= 5 (e = e * 5)
  f /= 3 (f = f / 3)
  g %= 9 (g = g % 9)

Example2

```
#include<iostream>
using namespace std;
void main()
{
int a=3,b=7,c=11;
int d=a+=b*c;
cout<<d<<endl;
cout<<a;
}
```

```
80
80
```

d=a+=b*c;
d=a+=7*11
d=a+=77
d=a=3+77
a=80
d=80

- Multiple assignments at the same time are allowed where implementation starts from right to left

Example 3

```
#include<iostream>
using namespace std;
void main()
{
int x=5, y=9,z=2;
int xx=x+=y/z;
cout<<xx<<endl;
xx=x+4.5;
cout<<xx<<endl;
cout<<"y= "<<y<<endl;
cout<<"z= "<<z<<endl;
cout<<"x= "<<x;
}
```

```
9
13
y= 9
z= 2
x= 9
```

xx=x+=y/z;
xx=x+=9/2
xx=x+=4.5
xx=(x=x+4.5)
xx=(x=9.5)--> xx is int
xx=9
x=9
xx=x+4.5
xx=9+4.5=13.5 -->xx is int
so
xx=13

Example4

```
#include <iostream>
using namespace std;
int  main()
{
    int x;
int y;
y=5;
x=10+y=9;
cout<< y<<endl;
    return 0;
}
```

Error: expression must be a modifiable lvalue

- you cannot use arithmetic operators between multiple assignments.

Increment & Decrement Operators

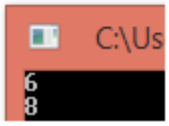
- Increment operator (++) - can be used instead of c += 1 (unary operator)
- Decrement operator (--) - can be used instead of c -= 1 (unary operator)
  - Pre-increment/decrement
    - When the operator is used before the variable (++c or --c)
    - Variable is changed, then the expression it is in is evaluated.
  - Post-increment/decrement
    - When the operator is used after the variable (c++ or c--)
    - Expression the variable is in executes, then the variable is changed.

Example1

```
#include <iostream>
using namespace std;
int  main()
{
int x=5;
    x++;
    cout<<x<<endl;
    int y = 9 ;
        y--;
    cout<<y<<endl;
    return 0;
}
```

x++ => x=x+1 => x=5+1= 6

y-- => y=y-1 => y=9-1= 8

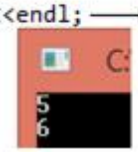


Example2

```
#include <iostream>
using namespace std;
int  main()
{
int x=5;

    cout<<x++<<endl;
    int y = 5 ;
    cout<<++y<<endl;

    return 0;
}
```

prints out 5 (cout is executed before the increment. x now has the value of 6)

prints out 6 (y is changed before cout is executed)



Example3

```
#include <iostream>
using namespace std;
int  main()
{
int x=5;
   x++;
    cout<<"result of x++ is    "<<x<<endl;
    x=5;
   ++x;
    cout<<"result of ++x is    "<<x<<endl;
    cout<<"same result";
    return 0;
}
```

```
C:\Users\eng.alaa\D
result of x++ is    6
result of ++x is    6
same result
```

- When Variable is not in an expression --> Preincrementing and postincrementing have the same effect.

Example4

```
#include <iostream>
using namespace std;
int  main()
{
    int b;
int x=5;
  b= x++;
    cout<<b<<endl;
    x=5;
   b=++x;
    cout<<b<<endl;
    return 0;
}
```

```
C:\U
5
6
```

Example5

```
#include <iostream>
using namespace std;
int  main()
{
int x = 0;
cout << ++ (x + 100);
return 0;
}
```

```
Error: expression must be a modifiable lvalue
```

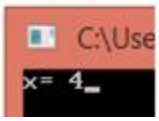
- note that ++ and -- cannot be applied to expressions

Increment and Decrement Operators precedence:

- Post-increment and post-decrement has higher priority than pre-decrement and pre-increment.
- Post-increment and post-decrement associates from left to right.
- Pre-increment and pre-decrement associates from right to left.

Example6

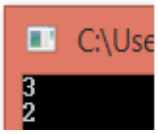
```
#include<iostream>
using namespace std;
void main()
{
     int a=3,b=2,x;
    x=a++ + (a-b);
    cout<<"x= "<<x;

}
```




```
x=a++ +(a-b)
       (3-2)    start with parenthesis
         1
x=a++ + 1
    3
x=4
```

Example7

```
#include<iostream>
using namespace std;
void main()
{  int x=1;
int b=2;
int y=x+++b;
cout<<y<<endl;
cout<<x<<endl;


}
```

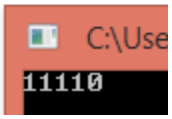



```
y=x++ + b          start with increment then
   1               addition
y=1+b

y=1+2
  y=3    now x = 2
```

more Example:

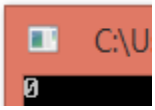
1)

```
#include <iostream>
using namespace std;
void main()
{
            int x = 10, y = 11;
      int z=y * x - ++x;
      cout<< x;
    cout<< z;


}
```



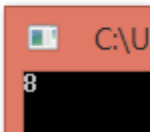
2)

```
#include <iostream>
using namespace std;
int  main()
{
    int x=1;
int y= 1;
int z=!y>x;
cout<<z;
return 0;
}
```



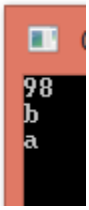
3)

```
#include <iostream>
using namespace std;
int  main()
{
    int x =3 , y=5,z;
z=x+++y;
cout<<z;
return 0;
}
```



4)

```
#include <iostream>
using namespace std;
int main()
{
    int a=3, b=2,c=5;
    char x='c';
        cout<<x-1<<endl;
        x--;
        cout<<x<<endl;
        cout<<--x;

}
```



Operators Precedence table

Precedence	Operator	Description	Associativity
1	::	Scoping operator	None
2	() [] ++ --	Grouping operator Array access Post-increment Post-decrement	left to right
3	! ~ ++ -- - + * & (type) sizeof	Logical negation Bitwise complement Pre-increment Pre-decrement Unary minus Unary plus Dereference Address of Cast to a given type Return size in bytes	right to left
4	* / %	Multiplication Division Modulus	left to right
5	+ -	Addition Subtraction	left to right
6	<< >>	Bitwise shift left Bitwise shift right	left to right
7	< <= > >=	Comparison less-than Comparison less-than-or-equal-to Comparison greater-than Comparison greater-than-or-equal-to	left to right
8	== !=	Comparison equal-to Comparison not-equal-to	left to right
9	&	Bitwise AND	left to right
10	^	Bitwise exclusive OR	left to right
11	|	Bitwise inclusive (normal) OR	left to right
12	&&	Logical AND	left to right

13	||	Logical OR	left to right
14	? :	Ternary conditional (if-then-else)	right to left

Escape Sequences

- An escape sequence begins with a \ (backslash or called escape character) followed by an alphanumeric character.
- Note that the two characters of an escape sequence are construed as a single character and indicates a special output on the screen.
- Also, it is used to allow the usage of some characters within a character constant which are reserved for C++ (e.g. \, “).
- Note that the escape sequence is considered as one character by the compiler. So, writing both of the following is correct:
  cout << “\n”;
  Or
  cout <<‘\n’;

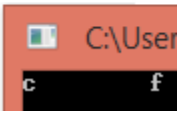
Escape Sequence	Meaning
\n	**new line**
\t	**horizontal tab**
\a	**bell sound (alert)**
****	**Backslash**
\"	**double quotation**
\b	**Backspace (place the cursor one character space back *not deleting characters).***
\r	**Carriage return (place the cursor at the beginning of the current line not a new one)**
\0	**Null character (used in strings)**

Example1

```
#include <iostream>
using namespace std;
int main()
{
	cout<<"one\ttwo\tthree\n";
	cout << "Jordan\b\b   " << endl;
	cout << "Jordan\b\b" << endl;
	cout << "Jordan\b\byy" << endl;
	cout << "\"Jordan\"" << endl;
	cout << "\\Jordan\\" << endl;
	cout << "Hello" << "\r" << "Bye" << endl;


}
```

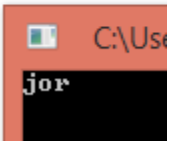
Example2

```
#include <iostream>
using namespace std;
bool main()
{char a='c',b='f',c='\t';
cout<<a<<c<<b;


}
```



Example3

```
#include <iostream>
using namespace std;
int main()

{
    cout<<"jor\0dan";
cin.get();
return 0;

}
```



Example4

```
1    # include <iostream>
2    using namespace std;
3    # include <string>
4    void main()
5    {
6        cout << "one\ntwo" << endl;
7        cout << "one\ttwo" << '\n';
8        cout << "onee\btwo" << endl;
9        cout << "one\rtwo" << endl;
10       cout << "one\\two" << endl;
11       cout << "one\'two" << endl;
12       cout << "one\"two" << endl;
13       system("pause");
14   }
```

```
C:\Users\eng\documents\visual studio 20
one
two
one     two
onetwo
two
one\two
one'two
one"two
Press any key to continue . . .
```

cout Function

- Ostream class.
- Tied to the standard output device (monitor or screen).
- Can display:
  - A string.
  - A variable value.
  - A result of an operation (mathematical, logical, function call, etc.).
- Concatenating or cascading or chaining of stream insertion operators: output many values using one cout and multiple insertion operators.
- The evaluation of the cascaded expressions starts from right to left but the printing on the screen starts from left to right

Example1

```
#include <iostream>
using namespace std;
int main()
{
    int x = 10;
cout << x <<"\t"<< ++x << "\t"<< x++ << "\t" << x << endl;


}
```

```
C:\Users\eng.alaa\Documents\Visual Stuc
12      12      10      10
```

Example2

```
#include <iostream>
using namespace std;
int main()
{
int x = 10; int z=10;
cout << x <<"\t"<< x+1 << "\t"<< x+3 << "\t" << x << endl;
cout << z <<"\t"<< (z+=1) << "\t"<< (z+=3) << "\t" << z << endl;

}
```

```
C:\Users\eng.alaa\Documents\
10      11      13      10
14      14      13      10
```

Example3

Consider the following statements. The output is shown to the right of each statement.

	Statement	Output
1	`cout << 29 / 4 << endl;`	`7`
2	`cout << "Hello there." << endl;`	`Hello there.`
3	`cout << 12 << endl;`	`12`
4	`cout << "4 + 7" << endl;`	`4 + 7`
5	`cout << 4 + 7 << endl;`	`11`
6	`cout << 'A' << endl;`	`A`
7	`cout << "4 + 7 = " << 4 + 7 << endl;`	`4 + 7 = 11`
8	`cout << 2 + 3 * 5 << endl;`	`17`
9	`cout << "Hello \nthere." << endl;`	`Hello` `there.`

cin Function

- Istream class.
- Tied to the standard input device (keyboard).
- When reading a string cin will stop at the first white space encountered in the string or when you press enter.

Example1

```
#include <iostream>
using namespace std;
int main()
{
    int x=0;
    cin>>x;
    cout<<x;

return 0;

}
```

```
5   6
5
```

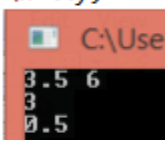
→ user enter 5 6
cin will stop at the first white space so value of x will be 5

- Cascaded extraction operator can be applied to read more than one variable from the keyboard using one statement (after entering each variable value press enter).

Example2

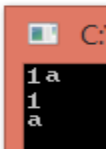
alaa abu srhan --email: alaaa@hu.edu.jo

```
#include <iostream>
using namespace std;
int main()
{
int x;
float y;
cin>>x>>y;
cout<<x<<"\n"<<y;

}
```



Example3

```
#include <iostream>
using namespace std;
int main()
{
int x;
char y;
cin>>x>>y;
cout<<x<<"\n"<<y;


}
```



Valid Input for a Variable of the Simple Data Type

Data Type of a	Valid Input for a
char	One printable character except the blank
int	An integer, possibly preceded by a + or – sign
double	A decimal number, possibly preceded by a + or – sign. If the actual data input is an integer, the input is converted to a decimal number with the zero decimal part.

- Entering a **char** value into an **int** or **double** variable causes serious errors, called input failure
- When reading data into a **char** variable
  - >> skips leading whitespace, finds and stores only the next character
  - Reading stops after a single character
- To read data into an **int** or **double** variable
  - >> skips leading whitespace, reads + or - sign (if any), reads the digits (including decimal)
  - Reading stops on whitespace non-digit character

Example:

Suppose you have the following variable declarations:

```
int a, b;
double z;
char ch;
```

The following statements show how the extraction operator >> works.

	Statement	Input	Value Stored in Memory
1	`cin >> ch;`	`A`	`ch = 'A'`
2	`cin >> ch;`	`AB`	`ch = 'A', 'B'` is held for later input
3	`cin >> a;`	`48`	`a = 48`
4	`cin >> a;`	`46.35`	`a = 46, .35` is held for later input
5	`cin >> z;`	`74.35`	`z = 74.35`
6	`cin >> z;`	`39`	`z = 39.0`
7	`cin >> z >> a;`	`65.78 38`	`z = 65.78, a = 38`

Example:

Suppose you have the following variable declarations:

```
int a;
double z;
char ch;
```

The following statements show how the extraction operator >> works.

	Statement	Input	Value Stored in Memory
1	`cin >> a >> ch >> z;`	`57 A 26.9`	`a = 57, ch = 'A', z = 26.9`
2	`cin >> a >> ch >> z;`	`57 A` `26.9`	`a = 57, ch = 'A', z = 26.9`

3	`cin >> a >> ch >> z;`	`57`
		`A`
		`26.9`
			`a = 57, ch = 'A', z = 26.9`

4	`cin >> a >> ch >> z;`	`57A26.9`	`a = 57, ch = 'A', z = 26.9`

Example:

Suppose you have the following variable declarations:

```
int a, b;
double z;
char ch, ch1, ch2;
```

The following statements show how the extraction operator >> works.

	Statement	Input	Value Stored in Memory
1	`cin >> z >> ch >> a;`	`36.78B34`	`z = 36.78, ch = 'B', a = 34`
2	`cin >> z >> ch >> a;`	`36.78` `B34`	`z = 36.78, ch = 'B', a = 34`
3	`cin >> a >> b >> z;`	`11 34`	`a = 11, b = 34,` computer waits for the next number
4	`cin >> a >> z;`	`78.49`	`a = 78, z = 0.49`
5	`cin >> ch >> a;`	`256`	`ch = '2', a = 56`
6	`cin >> a >> ch;`	`256`	`a = 256,` computer waits for the input value for `ch`
7	`cin >> ch1 >> ch2;`	`A B`	`ch1 = 'A', ch2 = 'B'`

Using Predefined Functions in a Program

- Header file may contain several functions
- To use a predefined function, you need the name of the appropriate header file
    - You also need to know:
        - Function name

- Number of parameters required
- Type of each parameter
- What the function is going to do

Example:

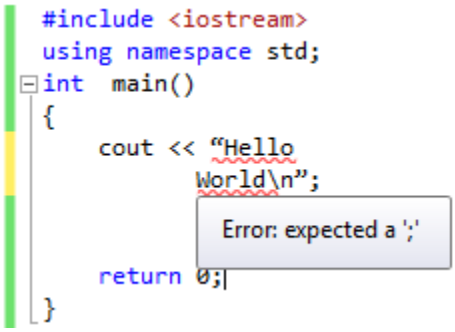
```
1    # include <iostream>
2    using namespace std;
3    # include <string>
4    #include<cmath>
5    void main()
6    {
7        const double PI=3.1416;
8        double SphareRadius;
9        double SphareVolume;
10       double Point1x, Point2x, Point1y, Point2y;
11       cout << "Enter the raduis of the sphare: " << endl;
12       cin >> SphareRadius;
13       SphareVolume = (4 / 3)*PI*pow(SphareRadius,3);
14       cout << "the volume of the sphare is : " << SphareVolume << endl;
15       cout << endl;
16       cout << "enter the coordinates of two points int the X-Y plane(x1,y1,x2,y2)" << endl;
17       cout << "point1 --> x1,y1, spearted by space" << endl;
18       cin >> Point1x >> Point1y;
19       cout << "point1 --> x2,y2, spearted by space" << endl;
20           cin>>Point2x>> Point2y;
21           double distance =sqrt(pow(Point2x-Point1x,2)+pow(Point2y-Point1y,2)) ;
22           cout << "the distance between points" << "( " << Point1x << " , " << Point2x << ") and ";
23           cout << "( " << Point1y << " , " << Point2y << ") is :"<<distance<<endl;
24           cout << endl;
25           string str;
26           getline(cin, str);
27           cout << "the number of characters, including blanks in \"str\"" << str.length() << endl;
28           system("pause");
29   }
```

- To use **pow** (power), include cmath
  - Two numeric parameters
- Syntax: pow(x,y)
  - = x^y
  - x and y are the arguments or parameters

- To use **sqrt** include cmath
  - Two numeric parameters
- Syntax: sqrt(x,y)
  - x and y are the arguments or parameters

Code Lines Breakage:

- You can split a long line of code among multiple lines by pressing enter. However, you must be careful when selecting the break locations of the line of code:
  - After or before an operator.
  - After a comma in a comma separated list.
  - After the end of a string (do not break at the middle of a string).



Syntax Error Example

- Errors in syntax are found in compilation

int x; //Line 1

int y //Line 2: error

double z; //Line 3

y = w + x; //Line 4: error

Chapter 4

Control Structure

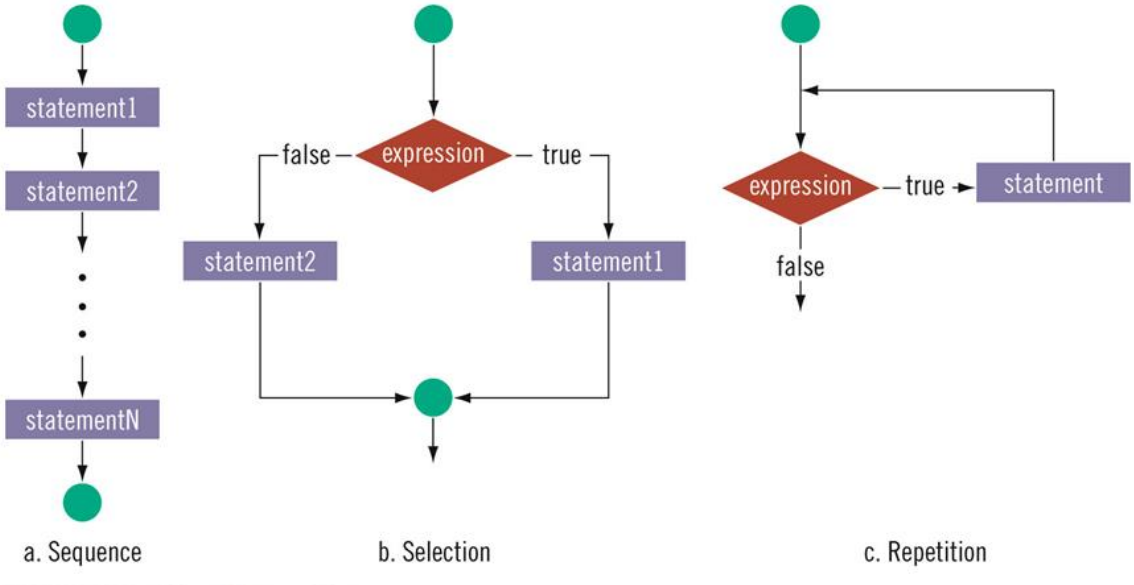
Selection control

Outline :

- Control Structure Introduction.
- **if** statement.
- **if/else** statement.
- Nested **if/else** statements.

Control Structure Introduction.

- **Sequential execution:** Statements executed one after the other in the order written
- **Transfer of control:** When the next statement executed is not the next one in sequence
- All programs written in terms of **3 control structures**:
  1. **Sequence structure:** Built into C++, programs executed sequentially by default.
  2. **Selection structures:** C++ has three types - **if**, **if/else**, and **switch**
  3. **Repetition structures:** C++ has three types - **while**, **do/while**, and **for**



Equality and Relational Operators

- Relational Operators:
  - Greater than >
  - Less than <
  - Greater than or equal >=
  - Less than or equal <=
- Equality operators:
  - Equal to ==
  - Not equal to !=

Operator	Operation Performed
x>y	Is x greater than y?
x<y	Is x less than y?
x>=y	Is x greater than or equal to y?
x<=y	Is x less than or equal to y?
x==y	Is x equal to y?
x!=y	Is x not equal to y?

Example1

```
using namespace std;
void main()
{
        int a = 9, b = 30, c = 89, d = 35;
      bool e, f;
      e = a > b;
      cout <<"e = a > b is    "<< e << endl;
}
```

```
C:\Users\eng.alaa\Doc
e = a > b is    0
```

e=a>b
mean is a>b and store result on e--> 9>30 no so its false then e = false, and the output will then 0

- Equality and relational operators are binary operators, used for comparison between two operands.
- Their result is either “true” --> output will be 1 or “false” --> output will be 0, i.e. boolean data type.

Example2

```
using namespace std;
void main()
{
        int a = 9, b = 30, c = 89, d = 35;
     bool e , f;
      f = d == b;
      e = d == d;
      cout <<"f = d == b is      "<< f << endl;
      cout <<"e = d == d is      "<< e << endl;
}
```

```
C:\Users\eng.alaa\Docum
f = d == b is      0
e = d == d is      1
```

f=d==b -->
d== b mean is d equal to b --->false
f=false --> will print as 0

Relational and equality Operators precedence:

- Relational operators have the same priority which is higher than the equality operators.
- If relational operators are associated, precedence is implemented from left to right.
- If equality operators are associated, precedence is implemented from left to right.
- Again, parenthesis forces priority.

Example3

```
using namespace std;
void main()
{
        int a = 9, b = 30, c = 89, d = 35;
    bool e , f;
    e = a>=b == d;
    cout<<"e = a>=b == d  is  "<<e<<endl;
    f = a<=b!=a>=d;
    cout <<"f = a<=b!=a>=d is "<<f<<endl;
}
```

```
C:\Users\eng.alaa\Docu
e = a>=b == d  is  0
f = a<=b!=a>=d is 1
```

e = a>=b == d;
note that Relational operators have the same priority which is higher than the equality operators
so start with a>=b it's false then 0
we get e=0==d; then e = 0 since d !=0

f = a<=b!=a>=d;
start with Relational operators from left to right then with equality from left to right
a<=b --> true then 1--> f=1=!a>=d
a>=d false then 0--> f=1!=0
1!=0 true then 1 --> f=1

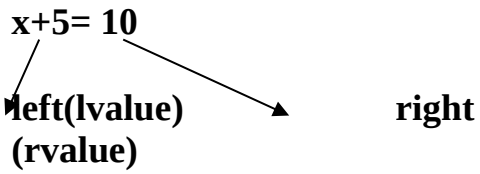
lvalues and rvalues:

- **lvalues or l-values(left value):** What can appear on the left side of an equation
- **rvalues or r-values(right value):** What can appear on the right side of an equation
- **lvalues** can be just **variable** but **rvalues** Can be **Constants, such as numbers, Variables,Or expressions**
- **note** lvalues can be used as rvalues, but not vice versa

Example4

```
using namespace std;
void main()
{
    int x;
    x+5=10;
}
```

```
int x

Error: expression must be a modifiable lvalue
```



- **syntax error --> lvalues** can be just **variable** (expressions cannot be used as l-values)

Example5

```
#include <iostream>
using namespace std;
int  main()
{

int y;
5=y;
cout<< y<<endl;
    return 0;
}
```

Error: expression must be a modifiable lvalue

- **syntax error --> lvalues** can be just **variable** (constant cannot be used as l-values)

- ## Relational Operators and thestring Type
  - Relational operators can be applied to strings
    - Strings are compared character by character, starting with the first character
    - Comparison continues until either a mismatch is found or all characters are found equal
    - If two strings of different lengths are compared and the comparison is equal to the last character of the shorter string
      - The shorter string is less than the larger string

Example

```
#include <string>
#include<iostream>
using namespace std;
void main()
{
    string str1 = "Hello";
    string str2 = "Hi";
    string str3 = "Air";
    string str4 = "Bill";
    string str5 = "Big";
    cout << (str1 < str2)<<endl;
    cout<< (str1 >"Hen" )<<endl;
    cout << (str3 <"An")<<endl;
    cout << (str1 == "hello")<<endl;
    cout << (str3<=str4)<<endl;
    cout << (str2 > str4) << endl;
    cout << (str4 >= "Billy") << endl;
    cout << (str5 <= "Bigger") << endl;
    system("puse");
}
```

```
c:\users\eng\documents\visual studio 2
1
0
1
0
1
1
0
1
```


alaa abu srhan --email: alaaa@hu.edu.jo

Expression	Value /Explanation
`str1 < str2`	**true** `str1 = "Hello"` and `str2 = "Hi"`. The first characters of `str1` and `str2` are the same, but the second character `'e'` of `str1` is less than the second character `'i'` of `str2`. Therefore, `str1 < str2` is **true**.
`str1 > "Hen"`	**false** `str1 = "Hello"`. The first two characters of `str1` and `"Hen"` are the same, but the third character `'l'` of `str1` is less than the third character `'n'` of `"Hen"`. Therefore, `str1 > "Hen"` is **false**.
`str3 < "An"`	**true** `str3 = "Air"`. The first characters of `str3` and `"An"` are the same, but the second character `'i'` of `"Air"` is less than the second character `'n'` of `"An"`. Therefore, `str3 < "An"` is **true**.
`str1 == "hello"`	**false** `str1 = "Hello"`. The first character `'H'` of `str1` is less than the first character `'h'` of `"hello"` because the ASCII value of `'H'` is 72, and the ASCII value of `'h'` is 104. Therefore, `str1 == "hello"` is **false**.
`str3 <= str4`	**true** `str3 = "Air"` and `str4 = "Bill"`. The first character `'A'` of `str3` is less than the first character `'B'` of `str4`. Therefore, `str3 <= str4` is **true**.
`str2 > str4`	**true** `str2 = "Hi"` and `str4 = "Bill"`. The first character `'H'` of `str2` is greater than the first character `'B'` of `str4`. Therefore, `str2 > str4` is **true**.

Expression	Value/Explanation
`str4 >= "Billy"`	**false** `str4 = "Bill"`. It has four characters, and `"Billy"` has five characters. Therefore, `str4` is the shorter string. All four characters of `str4` are the same as the corresponding first four characters of `"Billy"`, and `"Billy"` is the larger string. Therefore, `str4 >= "Billy"` is **false**.
`str5 <= "Bigger"`	**true** `str5 = "Big"`. It has three characters, and `"Bigger"` has six characters. Therefore, `str5` is the shorter string. All three characters of `str5` are the same as the corresponding first three characters of `"Bigger"`, and `"Bigger"` is the larger string. Therefore, `str5 <= "Bigger"` is **true**.

- # Logical Operators

Logical operators allows the programmer to combine more than one condition with each other to form more complex conditions.

- **&&** (logical **AND**)
  - It is a binary operator.
  - Returns **true** if both conditions are **true.**
- || (logical **OR**)
  - It is a binary operator.
  - Returns **true** if either of its conditions is **true.**
- **!** (logical **NOT** or logical negation)
  - Reverses the truth/falsity of its condition (reverse the meaning of a condition).
  - Returns **true** when its condition is **false.**
  - It is a unary operator, only takes one condition.
- Logical operators used as conditions in loops, e.g. for and while, and conditional statements, e.g. if/else.

Truth tables

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false

A	!A
true	false
false	true

- ## Logical (Boolean) Operators and Logical Expressions

1. Not (!)

Expression	!(Expression)
`true` (nonzero)	`false` (0)
`false` (0)	`true` (1)

Expression	Value	Explanation
`!('A' > 'B')`	`true`	Because `'A' > 'B'` is `false`, `!('A' > 'B')` is `true`.
`!(6 <= 7)`	`false`	Because `6 <= 7` is `true`, `!(6 <= 7)` is `false`.

alaa abu srhan --email: alaaa@hu.edu.jo

2. AND(&&)

Expression1	Expression2	Expression1 && Expression2
true (nonzero)	true (nonzero)	true (1)
true (nonzero)	false (0)	false (0)
false (0)	true (nonzero)	false (0)
false (0)	false (0)	false (0)

Expression	Value	Explanation
(14 >= 5) && ('A' < 'B')	true	Because (14 >= 5) is true, ('A' < 'B') is true, and true && true is true, the expression evaluates to true.
(24 >= 35) && ('A' < 'B')	false	Because (24 >= 35) is false, ('A' <'B') is true, and false && true is false, the expression evaluates to false.

3. OR(||)

Expression1	Expression2	Expression1 || Expression2
true (nonzero)	true (nonzero)	true (1)
true (nonzero)	false (0)	true (1)
false (0)	true (nonzero)	true (1)
false (0)	false (0)	false (0)

Expression	Value	Explanation
(14 >= 5) || ('A' > 'B')	true	Because (14 >= 5) is true, ('A' > 'B') is false, and true || false is true, the expression evaluates to true.
(24 >= 35) || ('A' > 'B')	false	Because (24 >= 35) is false, ('A' > 'B') is false, and false || false is false, the expression evaluates to false.
('A' <= 'a') || (7 != 7)	true	Because ('A' <= 'a') is true, (7 != 7) is false, and true || false is true, the expression evaluates to true.

Example1

```
#include <iostream>
using namespace std;
int  main()
{
int a=9; int b=1; int d=0;bool c;
    c= a&&b; cout<<c<<endl;
    c=a||b;  cout<<c<<endl;
    c=!b;  cout<<c;
    return 0;
}
```

```
C:\User
1
1
0
```

Example2

```
#include <iostream>
using namespace std;
int  main()
{
int a=9, b=0, d=0, c;
    c= a&&b; cout<<c<<endl;
    c=a||b;  cout<<c<<endl;
    c=!b;  cout<<c;
    return 0;
}
```

```
0
1
1
```

- Note that logical operators are applied to Boolean values only, if other types are given as operands implicit casting will work to convert them to Boolean.

Logical Operators precedence:

- **!** has the highest priority followed by **&&** and then **||** (i.e. || has the lowest priority).
- When any of these operators are combined, implementation starts from left to right.

• Order of Precedence

Operators	Precedence
!, +, – (unary operators)	first
*, /, %	second
+, –	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

Example3

```
#include<iostream>
using namespace std;
void main()
{
    bool a=true;bool b=false; bool c=false;
    bool result = a&&!c ||a&&a;
    cout<<"the result of a&&!c ||a&&a is "<<result;
}
```

```
C:\Users\eng.alaa\Documents\Visu
the result of a&&!c ||a&&a is 1
```

```
a&&!c||a&&a                !--> &&-->||

start with !        !c --> !0--> 1

then && from left to right

     a&&1||a&&a
      [1]  [2]
     1&&1  1&&1
     =1    =1
       \___/
     1||1 =1
```

Example4

```
#include<iostream>
using namespace std;
void main()
{
    int a=3,b=0,c=100;
    bool d;
    int f;
    d=!a&&!b||!c&&(a||b);
    cout<<"the result of !a&&!b||!c&&(a||b) is "<<d<<endl;
}
```

```
C:\Users\eng.alaa\Documents\Visual Stuc
the result of !a&&!b||!c&&(a||b) is 0
```

```
d=!a&&!b||!c&&(a||b)  start with parenthesis()
                3||0  note true is any non-zero
                 =1   value so 3 is true(1)

d=!a&&!b||!c&&1       then ! from left to right
  [1] [2] [3]
 !3=0 !0=1 !100=0

d=0&&1||0&&1          then && from left to right
   0       0

d=0||0                then ||

d=0
```

Example5

```
#include <iostream>
using namespace std;
int  main()
{
int a=9, b=1, d=0;
    bool c;
    c= a>=b||b!=b;
    cout<<c<<endl;

    return 0;
}
```

```
C:\U
1
```

Logical Negative (!) → Relational operator → Equality (==,!=) → && → ||

```
c = a>=b || b != b
    [1]     [2]
        [3]
```

start with a>=b

is a >= b ?? 9>= 1 → true

c = 1 || b != b then b != b

1 != 1 → false (0)

c = 1 || 0 = 1

C=1

Example6

```
#include <iostream>
using namespace std;
int  main()
{
int a=9, b=1, d=0;
    bool c;
    c= !(a&&b<d);
    cout<<c<<endl;
    return 0;
}
```

```
C:
1
```

```
c=!(a&&b<d)   start with parenthesis

   (a&&b<d)   start with relational operator
              then &&
     is b<d ?? 1<0 false(0)
   (a&&0)
    9&&0 -->1&&0 =0

  c=!0 =1

  c=1
```

Example7

```
#include<iostream>
using namespace std;
void main()
{
    int a=3,b=2,c=100;
    int d;
    int f;
    d=!a+2;
    f=!(a+2);
    cout<<"the result of !a+2 is "<<d<<endl;
     cout<<"the result of !(a+2) is "<<f<<endl;

}
```

```
C:\Users\eng.alaa\Documer
the result of !a+2 is 2
the result of !(a+2) is 0
```

Logical Negative (!) → Arithmetic operators → Relational operators → Equality (==,!=) → && → ||

d=!a+2 start with ! then addition (+)

d=!3+2

0

d=0+2=2

d=2

---

f=!(a+2) start with parenthesis ()

3+2=5

f=!5=0 then !

f=0

Example8

```
#include<iostream>
using namespace std;
void main()
{
    int a=3,b=0,c=100;
    int d;
    int f;
    d=a&&b+2;

    cout<<"the result of a&&b+2 is "<<d<<endl;

}
```

```
C:\Users\eng.alaa\Docume
the result of a&&b+2 is 1
```

Logical Negative (!) → Arithmetic operators → Relational operators → Equality (==,!=) → && → ||

d=a&&b+2 start with addition then &&

0+2=2

d= a&&2

3&&2 -->1

d=1

Exapmle

```
1   #include <string>
2   #include<iostream>
3   using namespace std;
4   void main()
5   {
6       bool found = true;
7       int age = 20;
8       double hours = 45.30;
9           double overtime = 15.00;
10          int count = 20;
11          char ch = 'B';
12      cout << !found << endl;
13      cout << (hours>40) << endl;
14      cout << (!age) << endl;
15      cout << (!found&&(age>=18)) << endl;
16      cout << (!(found && (age >= 18))) << endl;
17      cout << (hours+overtime<=75.00) << endl;
18      cout << ((count>=0)&&(count<=100)) << endl;
19      cout << ('A' <= ch&&ch <= 'Z') << endl;
20      system("puse");
21  }
```

```
c:\users\eng\doc
0
1
0
0
0
1
1
1
```

Expression	Value / Explanation
`!found`	`false` Because `found` is `true`, `!found` is `false`.
`hours > 40.00`	`true` Because `hours` is `45.30` and `45.30 > 40.00` is `true`, the expression `hours > 40.00` evaluates to `true`.
`!age`	`false` `age` is `20`, which is nonzero, so `age` is `true`. Therefore, `!age` is `false`.
`!found && (age >= 18)`	`false` `!found` is `false`; `age > 18` is `20 > 18` is `true`. Therefore, `!found && (age >= 18)` is `false && true`, which evaluates to `false`.
`!(found && (age >= 18))`	`false` Now, `found && (age >= 18)` is `true && true`, which evaluates to `true`. Therefore, `!(found && (age >= 18))` is `!true`, which evaluates to `false`.
`hours + overTime <= 75.00`	`true` Because `hours + overTime` is `45.30 + 15.00 = 60.30` and `60.30 <= 75.00` is `true`, it follows that `hours + overTime <= 75.00` evaluates to `true`.
`(count >= 0) && (count <= 100)`	`true` Now, `count` is `20`. Because `20 >= 0` is `true`, `count >= 0` is `true`. Also, `20 <= 100` is `true`, so `count <= 100` is `true`. Therefore, `(count >= 0) && (count <= 100)` is `true && true`, which evaluates to `true`.
`('A' <= ch && ch <= 'Z')`	`true` Here, `ch` is `'B'`. Because `'A' <= 'B'` is `true`, `'A' <= ch` evaluates to `true`. Also, because `'B' <= 'Z'` is `true`, `ch <= 'Z'` evaluates to `true`. Therefore, `('A' <= ch && ch <= 'Z')` is `true && true`, which evaluates to `true`.

- **Two types of control structures combination:**

a. **Control structure stacking:** use them one after the other.
b. **Control structure nesting:** use one control structure inside the body of another control structure.

- Stacking control structure

```
int main()

{
    int grade=0;
   cout<<"enter grade";
   cin>>grade;
   if(grade>90)
    {
      cout<<"high grade";
    }
  else
    {
      cout<<"low grade";
    }
}
```

- Nesting control structure

```
int main()
{
    int grade=0;
   cout<<"enter grade";
   cin>>grade;
   if (grade<90)
   {
        if (grade<60)
        {
            if(grade<50)
            {
               cout<<"fail";
            }
            else
            {
                cout<<"pass";
            }
        }
    }
}
```

Selection structure

- used to choose among alternative courses of action
- If the condition is **true**: print statement executed and program goes on to next statement
- If the condition is **false**: print statement is ignored and the program goes onto the next statement
- Indenting makes programs easier to read
    - C++ ignores white-space characters

if and if...else statement

- **if and if...else statements can be used to create:**
  - **One-way selection**

```
if (expression)
    statement
```

    - Statement is executed if the value of the expression is true
    - Statement is bypassed if the value is false; program goes to the next statement
    - Expression is called a decision maker
  - Only performs an action if the condition is true (single selection structure)

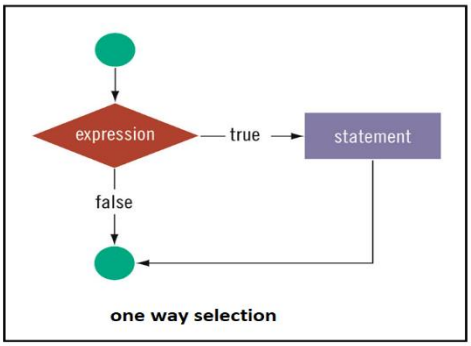
  - **Two-way selection**

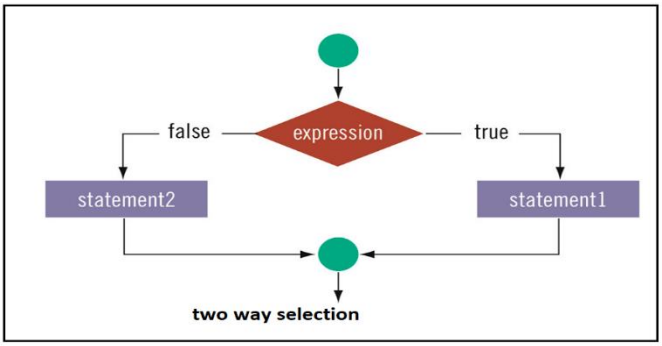
```
if (expression)
     statement1
else
     statement2
```

- If expression is true, statement1 is executed; otherwise, statement2 is executed
  - statement1 and statement2 are any C++ statements

– **Multiple selections(nested if)**

- Nesting: one control statement is located within another
- An else is associated with the most recent if that has not been paired with an else





Example1

```
#include <iostream>
using namespace std;
void main()
{
int x=2 ;
if (x<5)
{
    x=x+1;
    cout<<x<<endl;
}
cout<<"good"<<endl;

}
```

```
3
good
```

condition: is x<5 ??

the body of if is between barces

- If there are braces, then the body will be the compound statement.

if the condition is true then execute the body and then continue reading the code

- A decision can be made on any expression(zero - **false,** nonzero - **true).**

Example2

```
#include <iostream>
using namespace std;
int main()
{
int grade=60;
if (grade>50)
    cout<<"pass";

}
```

```
pass
```

is grade >50?? true then execute the body of if

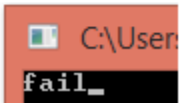
If no braces after if then the body will be only the first statement after it

Example3

```
#include <iostream>
using namespace std;
int main()
{
int grade=40;
if (grade>50)
    cout<<"pass";
else
    cout<<"fail";
}
```

if the condition is true then execute body of if

if false then execute body of else

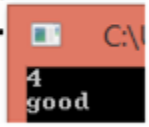


Example4

```
#include <iostream>
using namespace std;
void main()
{
int x=6;
if (x<5)
{
    x=x+1;
    cout<<x<<endl;
}
else
    cout<<x+1<<endl;
cout<<"good"<<endl;

}
```

body of else is just one sentence


```
#include <iostream>
using namespace std;
void main()
{
int x=3;
if (x<5)
{
    x=x+1;
    cout<<x<<endl;
}
else
    cout<<x+1<<endl;
cout<<"good"<<endl;

}
```



this sentence would be automatically executed and the body of the else will be the first statement after it only

- If no braces after **else** then the body will be only the first statement after it .

Example5

```
#include <iostream>
using namespace std;
int main()
{
int grade=40;
if (grade>50)
    cout<<"pass";
cout<<" what then";
else
    cout<<"fail";
}
```

Error: expected a statement

- Placing lines of codes between else and the body of its if is ***syntax error***.

Example6

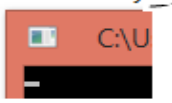
```
#include <iostream>
using namespace std;
int main()
{
int grade=40;
if ( )
        Error: expected an expression

    cout<<"pass";
else
    cout<<"fail";

}
```

- Leaving the parenthesis after the if empty (you have not put an expression for the condition) is ***syntax error***.

Example7

```
#include <iostream>
using namespace std;
void main()
{
int x=7;
if (x<5)
   if (x==5)
    cout<<x+1<<endl;
   else
   cout<<"good"<<endl;

}
```

body of if(x<5)




```
#include <iostream>
using namespace std;
void main()
{
int x=7;
if (x<5)
{
   if (x==5)
    cout<<x+1<<endl;
}
   else
   cout<<"good"<<endl;
}
```

body of if (x<5)

- What about if x =3 ??
- What about if x=5 ??

Example8

```
#include <iostream>
using namespace std;
void main()
{
int x=7;
if (x<5)
    cout<<x<<endl;
else
   if (x==5)
    cout<<x+1<<endl;
   else
   cout<<"good"<<endl;

}
```

body of if(x<5)

body of else

- Nested **if/else** structures:

  Test for multiple cases by placing **if/else** selection structures inside **if/else** selection structures.

- What about if x =3 ??
- what about if x=5 ??

Example

```
#include <iostream>
using namespace std;
void main()
{
    double creditCardBalance;
    double payment;
    double balance;
    double penalty = 0;
    cout << "Enter credit card balance: "<<endl;
    cin >> creditCardBalance;
    cout << "enter the payment:  " << endl;
    cin >> payment;
    balance = creditCardBalance - payment;
    if (balance > 0)
        penalty = balance*0.5;
    cout << "the balance is : " << balance<<endl;
    cout << "the penalty to be added to your next month bill is : " << penalty << endl;
    system("puse");
}
```

Example

```
#include <iostream>
using namespace std;
void main()
{
    int grade;
    cin >> grade;
    if (grade >= 90)
        cout << "the grade is A" << endl;
    else if (grade >= 80)
        cout << "the grade is B" << endl;
    else if (grade >= 70)
        cout << "the grade is C" << endl;
    else if (grade>=60)
        cout << "the grade is D" << endl;
    else
        cout << "the grade is F" << endl;
    system("puse");
}
```

Example

```
if (gender == 'M')                  //Line 1
    if (age < 21 )                  //Line 2
        policyRate = 0.05;          //Line 3
    else                            //Line 4
        policyRate = 0.035;         //Line 5
else if (gender == 'F')             //Line 6
    if (age < 21 )                  //Line 7
        policyRate = 0.04;          //Line 8
    else                            //Line 9
        policyRate = 0.03;          //Line 10
```

In this code, the `else` in Line 4 is paired with the `if` in Line 2. Note that for the `else` in Line 4, the most recent incomplete `if` is the `if` in Line 2. The `else` in Line 6 is paired with the `if` in Line 1. The `else` in Line 9 is paired with the `if` in Line 7. Once again, the indentation does not determine the pairing, but it communicates the pairing.

Example

```
#include <iostream>
using namespace std;
int main()
{
    int x = 6, y = 2;
    if (x > y)
        cout << "x is greater than y\n";
    else if (y > x)
        cout << "y is greater than x\n";
    else
        cout << "x and y are equal\n";
    return 0;
}
```

The output of this program is :
x is greater than y.

If we assign the values of x & y as follow: **int x = 2; int y = 6;**
then the output is:
y is greater than x.

If we assign the values of x & y as follow: **int x = 2; int y = 2;**
then the output is:
x and y are equal.

Short-circuit evaluation

- **Short-circuit evaluation**: evaluation of a logical expression stops as soon as the value of the expression is known

Consider the following expressions:

```
(age >= 21) || ( x == 5)        //Line 1
(grade == 'A') && (x >= 7)      //Line 2
```

For the expression in Line 1, suppose that the value of age is 25. Because (25 >= 21) is true and the logical operator used in the expression is ||, the expression evaluates to true. Due to short-circuit evaluation, the computer does not evaluate the expression (x == 5). Similarly, for the expression in Line 2, suppose that the value of grade is 'B'. Because ('B' == 'A') is false and the logical operator used in the expression is &&, the expression evaluates to false. The computer does not evaluate (x >= 7).

Comparing Floating-Point Numbers for Equality: A Precaution

- Comparison of floating-point numbers for equality may not behave as you would expect
  - Example:
    - 1.0 == 3.0/7.0 + 2.0/7.0 + 2.0/7.0 evaluates to false
    - Why? 3.0/7.0 + 2.0/7.0 + 2.0/7.0 = 0.99999999999999989
- Solution: use a tolerance value
  - Example: if fabs(x – y) < 0.000001

Associativity of Relational Operators: A Precaution

```
#include <iostream>

using namespace std;

int main()
{
     int num;

     cout << "Enter an integer: ";
     cin >> num;
     cout << endl;

     if (0 <= num <= 10)
         cout << num << " is within 0 and 10." << endl;


    else
        cout << num << " is not within 0 and 10." << endl;

    return 0;
}
```

- num = 5

0 <= num <= 10	= 0 <= 5 <= 10	
	= (0 <= 5) <= 10	(Because relational operators are evaluated from left to right)
	= 1 <= 10	(Because 0 <= 5 is true, 0 <= 5 evaluates to 1)
	= 1 (true)	

- num = 20

0 <= num <= 10	= 0 <= 20 <= 10	
	= (0 <= 20) <= 10	(Because relational operators are evaluated from left to right)
	= 1 <= 10	(Because 0 <= 20 is true, 0 <= 20 evaluates to 1)
	= 1 (true)	

Ternary conditional operator (?:)

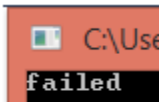
- Conditional operator (?:)
  - Ternary operator: takes 3 arguments
- Syntax for the conditional operator:

Expression1 ? Expression2 : expression3

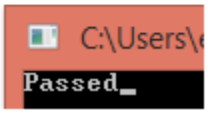
- If expression1 is true, the result of the conditional expression is expression2
  - Otherwise, the result is expression3
- Example: max = (a >= b) ? a : b;

Example1

```
#include <iostream>
using namespace std;
int main()

{
    int grade;
    grade=50;
    cout<<(grade>=60 ? "Passed" : "failed");

}
```



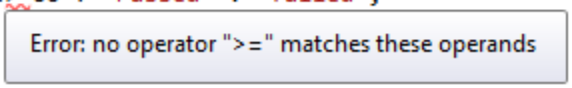

```
#include <iostream>
using namespace std;
int main()

{
    int grade;
    grade=70;
    grade >= 60 ? cout << "Passed" : cout << "Failed";

}
```



- Ternary conditional operator (**?:**) (the only C++ ternary operator)
    - Takes three arguments (condition, value if **true**, value if **false**)

Example2

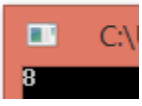
```
#include <iostream>
using namespace std;
int main()

{
    int grade;
    grade=50;
    cout<<grade>=60 ? "Passed" : "failed";

}
```



- omitting ( ) is **syntax error**
- Remember that the precedence of ?: is low, so do not forget the parenthesis that is used to force its priority

Example3

```
#include <iostream>
using namespace std;
void main()
{
int x=7;
x >= 6 ? x++ : x--;
cout<<x;

}
```



Example4

```
#include <iostream>
using namespace std;
bool main()
{
    int x=5;
    (x>=60 ? cout<<"pased";

}
```

Error: expected a ':'

- You can't use ? without :

Example5

Conditional Operator	Equivalent if else	Output

int A = 15, B = 2;
cout << (A > B ? A : B)
  << " is greater \n";
	int A = 15, B = 2;
	if(A>B)
	  cout << A << " is greater \n“; else
	  cout<<B<<<< " is greater\n";
		15 is greater

int x, y = 15;
x = (y < 10) ? 100 : -40;
cout << "value of x: " << x ;
	int x, y = 15;
	if (y < 10)
	  x=100;
	else
	  x= -40;
	cout << "value of x: " << x;
		value of x: -40

int n;
cout << "Enter a number : ";
cin >> n;
(n% 2 == 0) ? cout << n <<
“:Even number\n" : cout << n <<
“:Odd number\n";
	int n;
	cout << "Enter a number : ";
	cin >> n;
	if(n% 2 == 0)
	  cout<<n<<“ :Even number\n" ;
	else
	  cout<<n<<“ :Odd number\n";	

More Example

Example 1

```
#include <iostream>
using namespace std;
int main()
{
    int a=3, b=3,c=5;
    if (!c||!b&&a)
        cout<<"true";
    else
        cout<<"false";
}
```

```
!c||!b&&a   !--> && --> ||
 0   0

0||0&&a
      3
0||0&&3
     0
 0||0 --> 0
      0 or false mean dont
      execute if body
```

```
C:\U
false
```

Example2

```
#include <iostream>
using namespace std;
int main()
{
    int a=3 ;
    if (a=2) ----------> Sets a to 2 so the
        cout<<"you are great "; condition is always true
    else
        cout<<"you have to do better";
}
```

```
C:\Users\eng.alaa\Docum
you are great
```

```
#include <iostream>
using namespace std;
void main()
{
int a=3;
if (a==2)  ----------> is a=2 ?? --> false
    cout<<"you are great ";
else
    cout<<"you have to do better";
}
```

```
C:\Users\eng.alaa\doc
you have to do better_
```

- C++ allows you to use any expression that can be evaluated to either true or false as an expression in the if statement:

```
if (x = 5)
   cout << "The value is five." << endl;
```

- The appearance of = in place of == resembles a *silent killer*
  - It is not a syntax error
  - It is a logical error

Example3

```
#include <iostream>
using namespace std;
void main()
{
int a=3,b=2;
if (a==b++)
    cout<<"value of a : "<<a;
else
    cout<<"value of b : "<<b;
}
```

```
C:\Users\eng.ala
value of b : 3
```

b=2 a=3
is a= b++?? (b++ --> post increment)
3 = 2 ?? false

Now after we get the result **b** will be incremented by one

Example4

```
#include <iostream>
using namespace std;
void main()
{
int a=3,b=2;
if (--a==b)
    cout<<"value of a : "<<a;
else
    cout<<"value of b : "<<b;
}
```

```
C:\Users\eng.
value of a : 2
```

Is --a=b ???
2 = 2 ?? true --> now a= 2
Then Execute if body

Example5

```
#include <iostream>
using namespace std;
bool main()
{int x=1;
if (x)
{
    cout<<"ok";
}
}
```

```
C:\Use
ok
```

- A decision can be made on any expression(zero - **false,** nonzero - **true).**
- if x= 5 then it mean true --> print ok , if x=2 then it mean false ---> print nothing

Example6

```
#include <iostream>
using namespace std;
bool main()
{int x=0;
if (x)
{
    cout<<"ok";
}
else
    cout<<"no";
}
```

```
C:\Users
no
```

Example7

```
#include <iostream>
using namespace std;
bool main()
{int x=0;
int y =2;
if (x>5|| y<3)
{
     cout<<"ok";
}
else
     cout<<"no";

}
```

Example8

```
#include <iostream>
using namespace std;
bool main()
{int x=0;
int y =2;
if (sizeof('a'))

     if (sizeof y==4)
         cout<<x+5;
      else
     cout<<x+2;

}
```

Example9: what is the output of the following code ??

```
#include <iostream>
using namespace std;
bool main()
{int a=0;
cout<<a++<<"\t"<<++a<<endl;
if (++a==2)
{
     cout<<a;
}
else
     cout<<a+1;

}
```

Example10

```
#include <iostream>
using namespace std;
bool main()
{char a='a';
char c='c';
if (a-c==2)
{
     cout<<"ok";
}
else
     cout<<"no";

}
```

Note that a-c = 97-99=-2

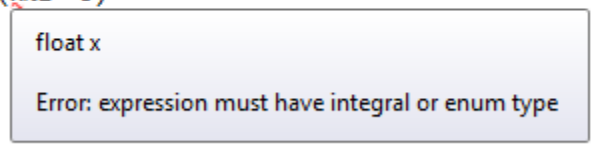
Example11

```
#include <iostream>
using namespace std;
bool main()
{
    int x=3.5;
    if(x==3)
        cout<<x*x;
}
```

implicitly the compiler will convert 3.5 into 3

Example12

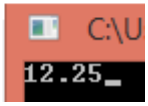
```
#include <iostream>
using namespace std;
bool main()
{
    float x=3.5;
    if(x%2==3)
        cout<<x*x;
}
```



- **syntax error** --> x must be int

Example13

```
#include <iostream>
using namespace std;
bool main()
{
    float x=3.5;
    if(int(x)%3==0)
        cout<<x*x;
}
```



Int(x)%3 --> 3%3 = 0
But
X will stay 3.5

Example14

```
#include <iostream>
using namespace std;
void main()
{
int x=5;
if(x<4)
    cout<<x;
else


}
```

no body for else and the condition is false so it will print nothing

Example15

```
#include <iostream>
using namespace std;
void main()
{
    int a=0,b=2,c=3;
    if (!a>b)
    {
        c++;
        cout<<c<<endl;
        cout<<b++;
    }
    else
    {
        ++c;
        cout<<c<<endl;
        cout<<++b;
    }
}
```

Example16

```
#include <iostream>
using namespace std;
bool main()
{
    int x=5;
    if(x%2==0)
        cout<<"even";
    else
    cout<<"odd";
}
```

Example17

```
#include <iostream>
using namespace std;
bool main()
{
    int x=5;
    if (x++==0);
        cout<<x;
}
```

- if we put ; after () of if this mean that if sentence is end (no body for if), but the compiler will read the condition of if; x++==0 so x will be 6 not 5. cout<<x ; will be executed always

Example18

```
#include <iostream>
using namespace std;
bool main()
{
    int x=5;
    if (x++==0);
        cout<<x;
    else
        cout<<x+1;
}
```

Error: expected a statement

- there is ; after () of if
- **syntax error** -->if statement has been ended so else now without if

Example19

```
#include <iostream>
using namespace std;
bool main()
{
    int x=5;
    if (x++==5)
        cout<<x<<endl;
    else;
        cout<<x+1;
}
```

```
6
7
```

- cou<<x+1 will be executed always, since else statement has been ended (else; mean that else ended , and it has no body).

Example20

```
#include <iostream>
using namespace std;
bool main()
{
    if (false)
        cout<<"true";
    else
        cout<<"false";
}
```

```
false
```

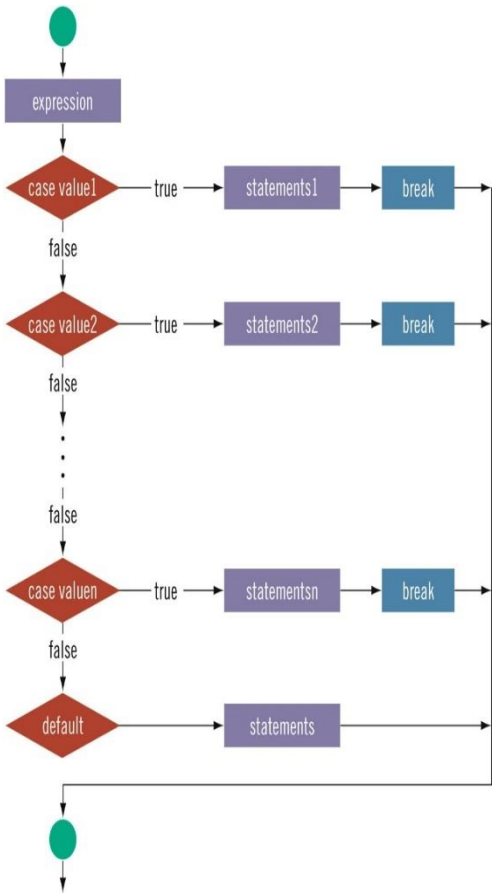
The *switch* Multiple-Selection Structure

switch:

- Useful when variable or expression is tested for multiple values.
- Consists of a series of **case** labels and an optional **default** case.
- Used instead of nested **if/else** statements to make the code more readable and easier to trace.
- switch (integral) expression is evaluated first
- Value of the expression determines which corresponding action is taken
- Expression is sometimes called the selector

Syntax:

```
switch (variable or expression)
{
case value1: //do something;
break;
case value2: //do something;
break;
case value3: //do something;
break;
case value4: //do something;
break;
default: // do something
break;
}
```



- One or more statements may follow a case label
- Braces are not needed to turn multiple statements into a single compound statement
- When a case value is matched, all statements after it execute until a break is encountered
- The break statement may or may not appear after each statement
- switch, case, break, and default are reserved words

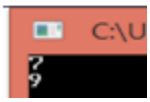
Example1

```
#include <iostream>
using namespace std;
void main()
{
    int x=7;
switch(x)
{
    case 7 :
        cout<<x++<<endl;
        break;
    case 8:
        cout<<++x<<endl;
        break;
}
}
```

- In this Examplevariable **x** is called the controlling expression**.**
- x will determine which case will be executed
- *break* determine the end of the *case* code block

Example2

```
#include <iostream>
using namespace std;
void main()
{
    int x=7;
switch(x)
{
    case 7 :
        cout<<x++<<endl;

    case 8:
        cout<<++x<<endl;
}

}
```



- If no *break* statement is included, all *case* statements will be implemented once a match has been found**.**
- Forgetting break statement is logical error.

Example3

```
#include <iostream>
using namespace std;
void main()
{
    int x=7;
switch(x)
{
    case 8 :
        cout<<x<<endl;

    case 7 :
        cout<<++x<<endl;
}

}
```

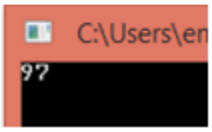
Example4

```
#include <iostream>
using namespace std;
void main()
{
    int x=1;
switch(x)
{
    case1:
        cout<<x<<endl;
        break;
case 2 :
            cout<<x<<endl;
}

}
```

- Forgetting white space between case and the value to test against it. **logical error**

Example5

```
#include <iostream>
using namespace std;
void main()
{
    char x='a';
switch(x)
{
    case 'a':
    case 'A':
        cout<<int(x)<<endl;

}

}
```



- what about if x ='A' ???

Example6

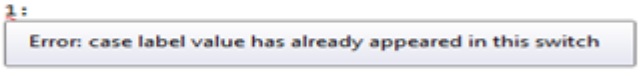
```
#include <iostream>
using namespace std;
void main()
{
    int x=1;
switch(x++)
{
    case 1:
        cout<<x<<endl;
        break;
    case 2:
        cout<<--x;
        break;
}

}
```

X=1
X++--> mean case 1 then x will be 2
Case 1 will print value of x which is 12

- what about if switch(++x)???

Example7

```
#include <iostream>
using namespace std;
void main()
{
    int x=1;
switch(x+++x++)
{
    case 1:
        cout<<x<<endl;
        break;
    case 1:


        cout<<--x;
        break;
}

}
```



- Identical *case* labels in switch is a syntax error.

Example8

```
#include <iostream>
using namespace std;
void main()
{
    int x=1;
switch(x+++x++)
{
    case 1:
        cout<<x<<endl;
        break;
    case 3:
        cout<<--x;
        break:
}
}
```

```
C:\Use
```

X++ + x++;
1 + 1 =2 ---> case 2
There is no case 2 so nothing will appear on screen

Example9

```
#include <iostream>
using namespace std;
void main()
{
    int x=1;
switch(x+++x++)
{
    case 1:
        cout<<x<<endl;
        break;
    case 3:
        cout<<--x;
        break;
    default :
        cout<<"there is no case value";
}
}
```

```
C:\Users\eng.alaa\docum
there is no case value
```

- if no case matches then default statement will be executed
- *default* statement is optional in *switch*

Example10

```
#include <iostream>
using namespace std;
void main()
{
    int x=1;
switch(x)
{
default :
    cout<<"ok";
    case 1:
        cout<<x<<endl;
        break;
    case 2 :
            cout<<x<<endl;
}
}
```

```
C:\User
1
```

```
#include <iostream>
using namespace std;
void main()
{
    int x=4;
switch(x)
{
default :
    cout<<"ok"<<endl;
    break;
    case 1:
        cout<<x<<endl;
        break;
    case 2 :
            cout<<x<<endl;
}
}
```

```
C:\Us
ok
```

```
#include <iostream>
using namespace std;
void main()
{
    int  x=1;
switch(x)
{
default :
    cout<<"ok"<<endl;
case 3: cout<<x;
    break;
case 4 : cout<<x;
    break;
}
}
```

```
C:\Users\e
ok
1
```

- *default* and *case* statements can be placed in any order inside the switch structure

Example11

```
#include<iostream>
using namespace std;
int main()
{
int grade=0;
cin>>grade;
switch (grade+10)
{
default: cout<<"no
matching found";
case 50: cout<<"50";
        break;
case 40: cout<<"40";
        break;
case 60: cout<<"60";
case 70: cout<<"70";
case 80: cout<<"80";
}
return 0;
}
```

Placing case 80: cout<<"hi"; is syntax error (never place similar

If you enter 50 , the program will display 607080
If you enter 60 , the program will display 7080
If you enter 10 , the program will display no

Remember that writing the switch or the break like this Switch, Break will give a syntax error

Example12

```
#include <iostream>
using namespace std;
void main()
{
int x=1;
int y=1;
switch (x)
{
case x+5 : cout<<x;
}

}
```

int x

Error: expression must have a constant value

Syntax error

```
#include <iostream>
using namespace std;
void main()
{
int x=1;
int y=1;
switch (x)
{
case y : cout<<x;
}

}
```

int y

Error: expression must have a constant value

Syntax error

```
#include <iostream>
using namespace std;
void main()
{
int x=1;
switch (x)
{
case 1.2 : cout<<x;
}

}
```

Error: expression must be an integral constant expression

Syntax error

- For case statements only constants integer values are allowed (**either integer or single characters**). No expressions, float/double values, and variables are allowed.

Example13

```
#include <iostream>
using namespace std;
void main()
{
int x=1;
int y=1;
switch ("x")
{
case 1 :
}

}
```

Error: expression must have integral or enum type

Syntax error

```
#include <iostream>
using namespace std;
void main()
{
float x=1;
int y=1;
switch (x)
{
case 1 :
}

}
```

float x

Error: expression must have integral or enum type

Syntax error

```
#include <iostream>
using namespace std;
void main()
{
int x=1;
float y=1;
switch (x/y)
{
case 1 :
}

}
```

int x

Error: expression must have integral or enum type

Syntax error

```
#include <iostream>
using namespace std;
void main()
{
int x=2;
float y=2;
switch (x+y)
{
case 1 :
    brea
case 0 :
}

}
```

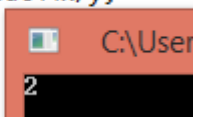
int x

Error: expression must have integral or enum type

Syntax error

alaa abu srhan --email: alaaa@hu.edu.jo

```
#include <iostream>
using namespace std;
void main()
{
int x=2;
int y=2;
switch (x/y)
{
case 1 : cout<<x;
    break;
case 0 : cout<<x/y;
}

}
```



x and y have int data type --> no error

- *switch* is used for testing constant integral expressions only, i.e. float, arrays, strings are not allowed. Only integer values or single character values. These values can be the result of an expression, variable, or a constant value.

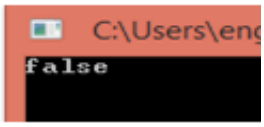
Example14

```
#include <iostream>
using namespace std;
void main()
{
    int x=1;
switch(x)
{
default :
    cout<<"ok"<<endl;
    break;
    case 1:
        cout<<x<<endl;
        break;
}


}
```

- putting ; after switch is **syntax error**

Example15

```
#include <iostream>
using namespace std;
void main()
{
    int x=5;
switch(x%2==0)
{
case 0:
    cout<<"false";
    break;
case 1:
    cout<<"true";
    break;
}


}
```

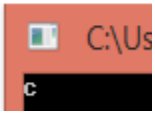


Example16

```
#include <iostream>
using namespace std;
void main()
{
    int x=3;
switch(x%5)
{
case 3:
        cout<<"3";

case 0:
    cout<<"0";
    break;
case 1:
    cout<<"1":
    break;

}
```

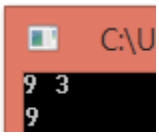
Example17

```
#include <iostream>
using namespace std;
void main()
{// cout<<x;
char x ='abc';
switch(x)
{
case 'c' :
    cout<<x;
}

}
```



Example18

```
#include <iostream>
using namespace std;
void main()
{
char x;
int y;
cin >>x>>y;
switch (x)
{
case '9': cout<<x; break;
case 9 : cout<<y; break;
    }
}
```



Example19

What will be happened if we didn't put braces between body of **switch ??**

```
#include <iostream>
using namespace std;
void main()
{
    int x=4;
switch(x)
default :
    cout<<"ok"<<endl;
    break;
}
```

Error: a break statement may only be used within a loop or switch

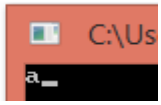
```
#include <iostream>
using namespace std;
void main()
{
    int x=4;
switch(x)
case 5 :
    cout<<"ok"<<endl;
    case 4
}
```

Error: a case label may only be used within a switch

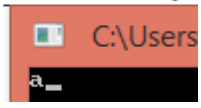
```
#include <iostream>
using namespace std;
void main()
{int A=4;
switch (A)
case 4 :
    cout<<A;
}
```

Example 20

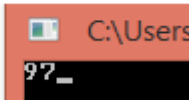
```
#include <iostream>
using namespace std;
void main()
{
    char  x=97;
    switch (x)
    {
        case 'a' :
            cout<<x;
    }
}
```




```
#include <iostream>
using namespace std;
void main()
{
    char  x=97;
    switch (x)
    {
        case 97 :
            cout<<x;
    }
}
```




```
#include <iostream>
using namespace std;
void main()
{
    int  x='a';
    switch (x)
    {
        case 97 :
            cout<<x;
    }
}
```




```
#include <iostream>
using namespace std;
void main()
{
    char  x=97;
switch(x)
{
case 97: cout<<x;
    break;
case 'a' : cout<<x;
    break;
}
}
```

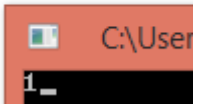
Error: case label value has already appeared in this switch

Syntax error

- case 97 == case 'a' , case 98 ==case 'b' and so on

Example21

```cpp
#include <iostream>
using namespace std;
void main()
{
bool   x =1;
switch (x)
{
case true: cout<<x; break;
}

}
```

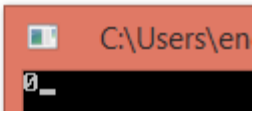



```cpp
#include <iostream>
using namespace std;
void main()
{
int   x =1;
switch (x)
{
case true: cout<<x; break;
case 1 : cout <<x; break;
}

}
```

Error: case label value has already appeared in this switch

syntax error

```cpp
#include <iostream>
using namespace std;
void main()
{
int   x =0;
switch (x)
{
case false: cout <<x; break;
}

}
```




```cpp
#include <iostream>
using namespace std;
void main()
{
int   x =0;
switch (x)
{
case false: cout<<x; break;
case 0: cout <<x; break;
}

}
```

Error: case label value has already appeared in this switch

Syntax error

- case true == case 1 , case false = case 0

Example22

```cpp
#include <iostream>
using namespace std;
void main()
{int x=1;
char y = 'a';
switch (x+y)
{
case 98 :
    cout<<x<<endl;
    cout<<y;
}

}
```


```cpp
#include <iostream>
using namespace std;
void main()
{int x=1;
char y = 'a';
switch (x+y)

case 97:
    cout<<x<<endl;
    cout<<y;
}
```

this sentence doesn't belong to switch

- **note that** there are no braces for switch statement

Example23

```cpp
#include<iostream>
using namespace std;
void main()
{
    int x=1;
    int y='a';
    switch(x+y)
    {
    case 97 :
        cout<<x<<endl;
        cout<<y<<endl;
    }

}
```

- **case 97** has two statements , and switch doesn't match this case. so it **will print nothing**.

Example24

```
#include<iostream>
using namespace std;
void main()
{
    int x=1;

    switch(x)
    {
        cout<<x<<endl;
    }

}
```

- There is no **case 1** so it will **print nothing**

More Example

Example

```
switch (grade)
{
case 'A':
     cout << "The grade point is 4.0.";
     break;
case 'B':
     cout << "The grade point is 3.0.";
     break;
case 'C':
     cout << "The grade point is 2.0.";
     break;
case 'D':
     cout << "The grade point is 1.0.";
     break;
case 'F':
     cout << "The grade point is 0.0.";
     break;
default:
     cout << "The grade is invalid.";
}
```

Example

```
switch (score / 10)
{
case 0:
case 1:
case 2:
case 3:
case 4:
case 5:
     grade = 'F';
     break;
case 6:
     grade = 'D';
     break;
case 7:
     grade = 'C';
     break;
case 8:
     grade = 'B';
     break;
case 9:
case 10:
     grade = 'A';
     break;
default:
     cout << "Invalid test score." << endl;
}
```

Example

Switch statement

```
int i, n;
cin >> i;
switch (i)
{
case 0:
case 1: n = 10;
    break;

case 2: n = 500;
    break;

default:n = 0;
    break;
}
cout << n << endl;
```

Equivalent nested if else

```
int i, n;
cin >> i;

if (i == 0 || i == 1)
    n = 10;
else if (i == 2)
    n = 500;
else
    n = 0;
cout << n << endl;
```

Chapter 5

Control Structure

(Repetition)

■ **In this chapter, you will study:**

- Why Is Repetition Needed?
- while Repetition Structure.
- for Repetition Structure.
- do/while Repetition Structure.
- continue and break Statements.
- Nested Control Structures
- Debugging loops

■ **Why Is Repetition Needed?**

- Repetition allows efficient use of variables
- Can input, add, and average multiple numbers using a limited number of variables
- For example, to add five numbers:
  - Declare a variable for each number, input the numbers and add the variables together
  - Create a loop that reads a number into a variable and adds it to a variable that contains the sum of the numbers

■ **Repetition (or looping) control structures:**

1. while.
2. for.
3. do/while.

■ **Two types of repetition or looping exist:**

1. Sentinel-Controlled Repetition.

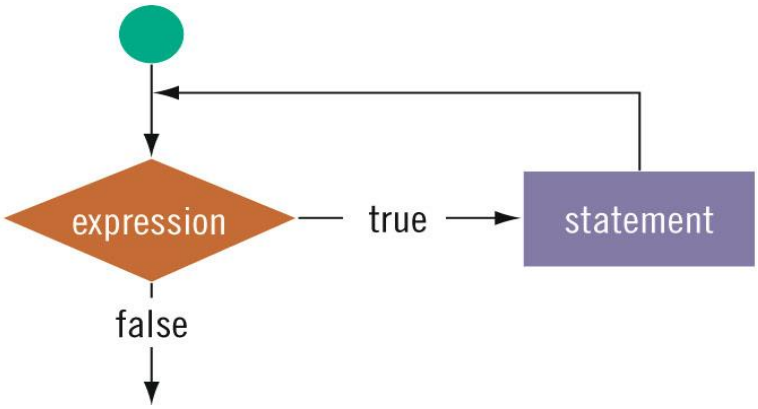
- In this type you do not know the number of times the body of the loop must be repeated, i.e. do not know the number of loop iterations.
- Mainly you use while, and do/while control structures for this type of looping.

2. Counter-Controlled Repetition.
   - In this type you know the number of times the body of the loop must be repeated, i.e. the number of loop iterations is defined in advance.
   - Mainly you use for control structures for this type of looping.

- ***while*** **Repetition Structure:**
  - Programmer specifies an action to be repeated while some condition remains true.
  - Also called looping or simply loop.

syntax :

```
while ( condition )
{
  ...  body of while

}
```



- statement can be simple or compound
- expression acts as a decision maker and is usually a logical expression
- statement is called the body of the loop
- The parentheses are part of the syntax

Example1

```
#include <iostream>
using namespace std;
void main()
{
int   counter =1 ;
while  (counter<=10)
{
cout<<counter<<endl;
counter++;
}

}
```

```
1
2
3
4
5
6
7
8
9
10
```

counter	condition	output
1	1<=10 ?? yes	1
2	2<=10 ?? yes	2
3	3<=10 ?? yes	3
4	4<=10 ?? yes	4
5	5<=10 ?? yes	5
6	6<=10 ?? yes	6
7	7<=10 ?? yes	7
8	8<=10 ?? yes	8
9	9<=10 ?? yes	9
10	10<=10 ?? yes	10
11	11<=10 ?? no ->stop	

- **while loop** repeated until condition becomes false where the next line of code after while loop will be executed
- **counter** in the Exampleis called the loop control variable (LCV)

Example2

```
#include <iostream>
using namespace std;
void main()
{
int   counter =1 ;
while  (counter<=10)
{
cout<<counter<<endl;
counter++;
}

}
```

```
1
2
3
4
5
6
7
8
9
10
```

```
#include <iostream>
using namespace std;
void main()
{
int   counter =1 ;
while  (counter<=10)
cout<<counter++<<endl;
cout<<"ok"<<endl;

}
```

```
1
2
3
4
5
6
7
8
9
10
ok
```

- The body of the **while** loop is the code block contained within the braces after the while, otherwise it is the first statement after the **while** only.

Example3

```
#include <iostream>
using namespace std;
void main()
{
int   counter =1 ;
while  (counter<=10)
cout<<counter<<endl;
counter ++;

}
```

iteration	counter		output
1	1	1<=10 ?? true	1
2	1	1<=10 ?? true	1
3	1	1<=10 ?? true	1
	....		

- The body of the **while** loop is the first statement after the **while** only.
- The condition of the **while** is always true, i.e. the body of the **while** loop ***does not*** modify the condition value.
- the execution will not finish (the condition is always true) --->**Infinite loop**
- **Infinite loop** is **Logical error** in the **while** structure.
- **Infinite loop**: continues to execute endlessly
- Avoided by including statements in loop body that assure the exit condition is eventually false

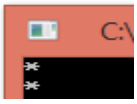
Example4

```
#include <iostream>
using namespace std;
void main()
{
    int x =1;
    while(   )
    {
                Error: expected an expression

    cout<<x <<endl;

    }
}
```

- Leaving the parenthesis after the while empty (i.e. you do not specify any condition) is **syntax error**

Example5

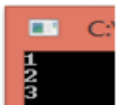
```
#include <iostream>
using namespace std;
void main()
{
int   x =1 ;
while  (x<3)
{
    x++;
cout<<"*"<<endl;
}

}
```



iteration	x		output
1	1	1<3 yes	*
2	2	2<3 yes	*
3	3	3<3 no-->stop	

Example6

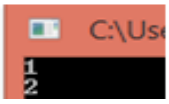
```
#include <iostream>
using namespace std;
void main()
{
    int x = 1;
    int y=1;
    int z;
    while(x<=3)
    {
    z=(++x)+(y-2);
    cout<<z<<endl;
    }
}
```



iteration	x		z	output
1	1	1<=3 ?? yes	2+(1-2)=1	1
2	2	2<=3 ?? yes	3+(1-2)=2	2
3	3	3<=3 ?? yes	4+(1-2)=3	3
	4	4<=3 ?? no	stop	

Example7

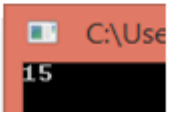
```
#include <iostream>
using namespace std;
void main()
{
    int x =5;
    int y=2;
    int z=1;
while (!(!x||!y))
{
        x=x-1;
        y=y-1;
    cout<<z++<<endl;
}

}
```



```
#include <iostream>
using namespace std;
void main()
{
    int x =5;
    int y=2;
    int z=1;
while (!x||!y)
{
        x=x-1;
        y=y-1;
    cout<<z++<<endl;
}

}
```

Example8

```
#include <iostream>
using namespace std;
void main()
{
 int c = 1, sum = 0;
while (c <= 5)
{
sum=sum+c;
c++;
}
cout<<sum<<endl;

}
```



c			sum	output
1	1<=5	?? true	0+1=1	
2	2<=5	?? true	1+2=3	
3	3<=5	?? true	3+3=6	
4	4<=5	?? true	6+4=10	
5	5<=5	?? true	10+5=15	
6	6<=5	?? no	stop	15

Example9

```
#include <iostream>
using namespace std;
void main()
{
int   x =1 ;
while  (x<=3)
{
    if (x%2 !=0)
        cout<<x<<endl;
    x++;
}

}
```

```
C:\Users\eng
1
3
```

x		output
1	1<=3 ?? true	
	1%2 !=0 true	1
2	2<=3 ?? true	
	2%2 !=0 false	
3	3<=3 ?? true	
	3%2 !=0 true	3
4	3<=3 ?? false	stop

Example10

```
#include <iostream>
using namespace std;
void main()
{
    int z=2;
    int i=0,c=2;
while(i<10)
{i+=z;
switch (i)
{
case 2: c=c+1;
cout<<"add 1"<<endl;
break;
case 4: c=c+2;
cout<<"add 2"<<endl;
default : c=c+3;
cout<<"add 3"<<endl;
}
}
cout<<"result: "<<c;

}
```

```
C:\Users
add 1
add 2
add 3
add 3
add 3
add 3
result: 17
```

Example11

```
#include <iostream>
using namespace std;
void main()
{
    int x =5;
    int y=2;
    int z=1;
while (!x||!y);
{
        x=x-1;
        y=y-1;
    cout<<z++<<endl;
}

}
```

```
C:\Us
1
```

!x||!y
!5||!2
0||0 --> 0 false stop while
X=4
Y=1
Will print z++ --> **1**
Z=2

- putting ; after **while** mean no body for **while**

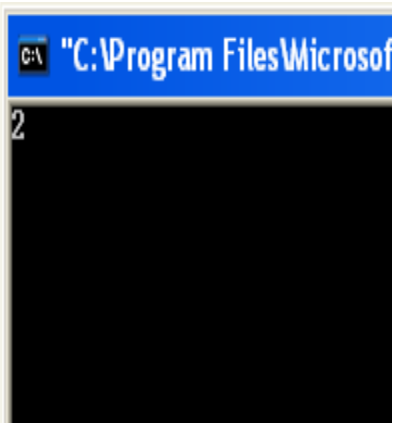
Example12

```
#include<iostream>
using namespace std;
void main()
{int x=1;
    while(x++<3);
    cout<<x<<endl;
}
```

x			
1	1<3	true	do nothing
2	2<3	true	do nothing
3	3<3	false	stop loop
4			

Example13

```
#include<iostream>
using namespace std;
void main()
{int x=1;
    while(x++>3);
    cout<<x<<endl;
}
```



Example14

```
#include <iostream>
using namespace std;
void main()
{
int   ff=5;
while  (ff<=10);
{
    ff++;
    cout<<ff++<<endl;
}
```

- Infinite loop-->logical error

■ While has three cases:

- **Case 1 :Counter-Controlled while Loops**
    - When you know exactly how many times the statements need to be executed
    - Use a counter-controlled while loop

```
counter = 0;          //initialize the loop control variable

while (counter < N) //test the loop control variable
{
     .
     .
     .
     counter++;       //update the loop control variable
     .
     .
     .
}
```

Example(test your self)

- 10 Students at a local middle school volunteered to sell fresh baked cookies to raise funds to increase the number of computers for the computer lab. Each student reported the number of boxes he/she sold. We will write a program that will do the following:
    - Ask each student about the total number of boxes of cookies he/she sold
    - Output the total number of boxes of cookies sold
    - Output the total revenue generated by selling the cookies
    - Output the average number of boxes sold by each student
- Assume the cost of each box of cookies = 5$.

- **Case 2: Sentinel-Controlled while Loops**
    - Sentinel variable is tested in the condition
    - Loop ends when sentinel is encountered

```
cin >> variable;              //initialize the loop control variable

while (variable != sentinel)  //test the loop control variable
{
     .
     .
     .
     cin >> variable;       //update the loop control variable
     .
     .
     .
}
```

- **Case 3: Flag-Controlled while Loops**
    - Flag-controlled while loop: uses a bool variable to control the
    - loop

```
found = false;        //initialize the loop control variable

while (!found)        //test the loop control variable
{
    .
    .
    .
    if (expression)
       found = true; //update the loop control variable
    .
    .
    .
}
```

Example(Number Guessing Game)

- implementing a number guessing game using a flag-controlled while loop
- Uses the function rand of the header file cstdlib to generate a random number
    - rand() returns an int value between 0 and 32767
    - To convert to an integer >= 0 and < 100:
        - rand() % 100

```
1   # include <iostream>
2   using namespace std;
3   #include <stdlib.h>
4   #include<ctime>
5   void main()
6   {
7       int num;
8       int guess;
9       srand(time(0));
10      num = rand() % 100;
11      bool isGuessed = false;
12      while (!isGuessed)
13      {
14          cout << "enter an integer greater than or equal 0 and less than 100 " << endl;
15              cin >> guess;
16              if (guess == num)
17              {
18                  cout << "you guessed the correct number" << endl;
19                  isGuessed = true;
20              }
21              else if (guess < num)
22              {
23                  cout << "your guess is lower than the number" << endl;
24                  cout << "guess again!" << endl;
25
26              }
27              else
28              {
29                  cout << "your guess is higher than the number" << endl;
30                  cout << "guess again!" << endl;
31
32              }
33      }
34  }
35
```

Example

- Consider the following sequence of numbers:
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- Called the Fibonacci sequence
- Given the first two numbers of the sequence (say, a1 and a2)
  - n^{th} number a_n, n >= 3, of this sequence is given by: $a_n = a_{n-1} + a_{n-2}$ Fibonacci sequence

- nth Fibonacci number
  - $a_2 = 1$
  - $a_1 = 1$
  - Determine the n^{th} number a_n, n >= 3
- Suppose $a_2 = 6$ and $a_1 = 3$
  - $a_3 = a_2 + a_1 = 6 + 3 = 9$
  - $a_4 = a_3 + a_2 = 9 + 6 = 15$
- Write a program that determines the n^{th} Fibonacci number, given the first two numbers

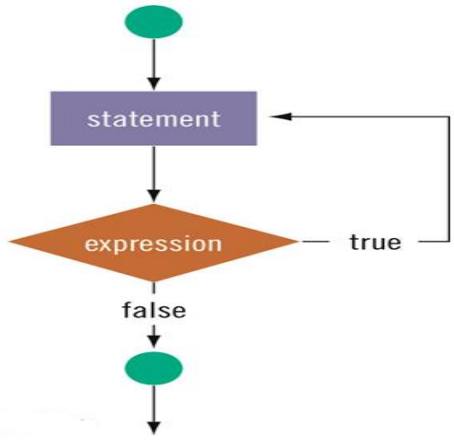
- **Algorithm:**
  - Get the first two Fibonacci numbers
  - Get the desired Fibonacci number
    - Get the position, *n*, of the number in the sequence
  - Calculate the next Fibonacci number
    - Add the previous two elements of the sequence
  - Repeat Step 3 until the n^{th} Fibonacci number is found
  - Output the n^{th} Fibonacci number

■ do while repetition structure

The do/while repetition structure is similar to the while structure except that Condition for repetition tested after the body of the loop is executed.

Syntax:

```
do
    statement
while (expression);
```



- The statement executes first, and then the expression is evaluated
  - As long as expression is true, loop continues
- To avoid an infinite loop, body must contain a statement that makes the expression false
- Loop always iterates at least once

Example1

body of while will execute once before condition for repetition has been tested

```
#include <iostream>
using namespace std;
void main()
{
int   x =1 ;
do
{
cout<<x<<endl; x++;
}
while  (x<=3);
    cout<<"the end";

}
```

```
C:\
1
2
3
the end
```

- All actions are performed ***at least once***.

Example2

```
#include <iostream>
using namespace std;
void main()
{
int   i=2,x =2 ;
do
{
    if (x<5) x-=1;
    else x+=2;
cout<<x<<endl;
}while  (++i<4);
    cout<<i;
}
```

```
1
0
4
```

```
#include <iostream>
using namespace std;
void main()
{
int   i=2,x =2 ;
do
{
    if (x<5) x-=1;
    else x+=2;
cout<<x<<endl;
}while  (i++<4);
    cout<<i;
}
```

```
1
0
-1
5
```

Example3

```
#include <iostream>
using namespace std;
void main()
{
int   x =5 ;
do
{
    if (x+5<10) x+=5;
    else x+=10;
}while  (++x<4);
    cout<<x<<endl;
}
```

```
16
```

Example4

```
#include <iostream>
using namespace std;
void main()
{
int   z=17 ;
do
{cout<<z<<endl;
z--;
}while  (z%5!=0);
    cout<<"ok"<<endl;
}
```

```
17
16
ok
```

Example5

```
#include <iostream>
using namespace std;
void main()
{
    int x=4,y=7;
do
{
    cout<<x<<endl;
}
while  ((x--,y--));

}
```

```
4
3
2
1
0
-1
-2
-3
```


alaa abu srhan --email: alaaa@hu.edu.jo

Example6

- Always be careful with the pre or post condition when applied to the while or do/while repetition

```
int num=3;
while(++num<7)
cout<<"loop"<<endl; //
    loop will be printed
    3 times on the
    screen
```

```
int num=3;
while(num++<7)
cout<<"loop"<<endl;
    // loop will be
    printed 4 times on
    the screen
```

```
int count=-3;
do {
cout<<"loop"<<endl;
}while(count++);//
    loop will be
    printed 4 times on
    the screen
```

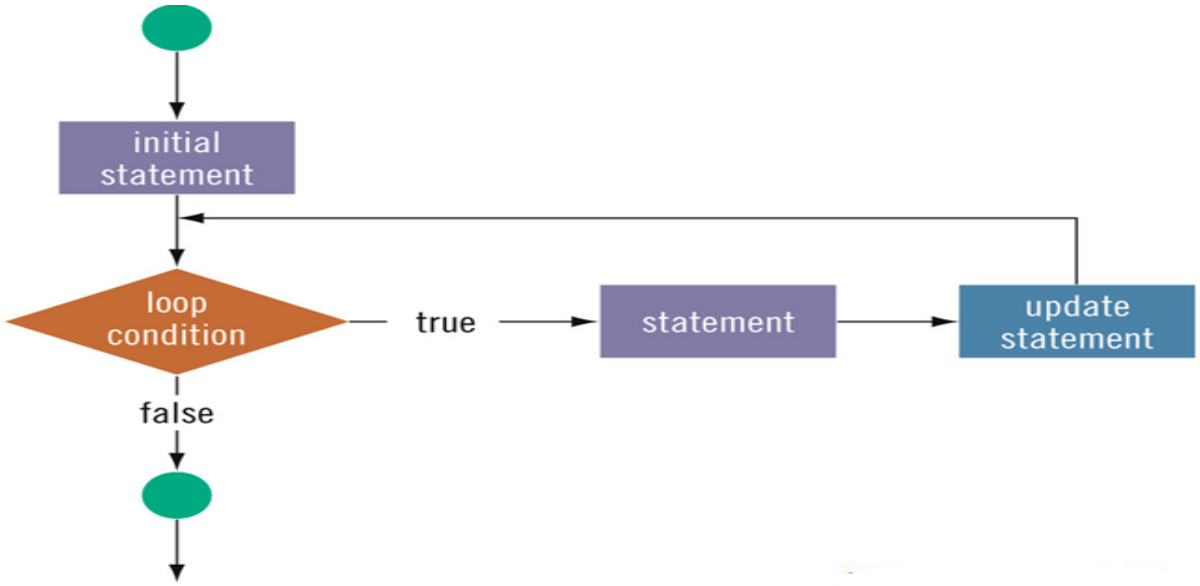
```
int count=-3;
do {
cout<<"loop"<<endl
    ;
}while(++count);//
    loop will be
    printed 3 times on
    the screen
```

■ for repetition structure

- ■ Handles all the details of counter-controlled repetition in a concise way.

syntax :

```
for (initial statement; loop condition; update statement)
    statement
```



Example1

```
#include <iostream>
using namespace std;
void main()
{
for (int i =0 ;i<=4; i++)
{cout<<i<<"  ";
}

}
```

1 2 4 3

C:\Users\en
0 1 2 3 4

steps:

1. initialization
2. condition
3. body of for
4. increment or decrement
   repeat from step 2

stop when the condition become false

Example2

```
#include <iostream>
using namespace std;
void main()
{
for (int i = 0, j = 0; j + i <= 10; j++, i++)
   cout << j + i << endl;

}
```

initialization — conditions — increment/decrement

0
2
4
6
8
10

■ **for loop** has three part separated by semicolon.

Example3

```
#include <iostream>
using namespace std;
void main()
{
    int i =0;
for ( ;i<=4; i++)
{cout<<i<<"  ";
}

}
```

```
0  1  2  3  4
```

```
#include <iostream>
using namespace std;
void main()
{

for ( ;i<=4; i++)
{cout<<
}

}
```

Error: identifier "i" is undefined

Syntax error

- The three parts of the **for** loop are optional , don't omit **;** inside for if you omit any part.
- we have to declare variable i outside **for** in this case , otherwise its **syntax error**.

Example4

```
#include <iostream>
using namespace std;
void main()
{

for (int i = 0 ; ; i++)
{cout<<i<<endl;
}

}
```

```
24910
24911
24912
24913
24914
24915
24916
24917
24918
24919
24920
24921
```

- The three parts of the **for** loop are optional, if the condition is omitted this will create an ***infinite loop*** since the compiler assumes that the **for** condition is true.

Infinite loop -->logical error

Example5

```
#include <iostream>
using namespace std;
void main()
{

for (int i = 0 ;i<=4; )
{cout<<i<<"  ";
i++;
}

}
```

```
0  1  2  3  4
```

```
#include <iostream>
using namespace std;
void main()
{

for (int i = 0 ;i<=4; )
{cout<<i<<endl;

}

}
```

infinite loop-->logical error

- if you omit increment or decrement from **for** sentence don't forget to put it inside the **for** body, otherwise it will give you **infinite loop**.

Example6

```
#include <iostream>
using namespace std;
void main()
{
        for (int h=5; h<10)
        {
       cout<<h<<endl;
        }
}
```

Error: expected a ';'

- omitting any part of the three parts of **for** doesn't mean to omit any **;** . omitting ; give us **<u>syntax error</u>**.

Example7

```
#include <iostream>
using namespace std;
void main()
{

for (int i = 97 ;i<100; )
{i++;
    cout<< char(i)<<endl;

}
}
```

```
b
c
d
```

```
#include <iostream>
using namespace std;
void main()
{

for (int i = 97 ;i<100; )
{
    cout<< char(i)<<endl;
    i++;
}
}
```

```
a
b
c
```

- be careful where you put increment or decrement of for .

Example8

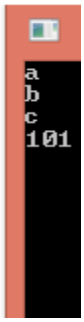
```
#include<iostream>
using namespace std;
#include <math.h>
void main()
{for (int i =0 ;i<5 ;i++)
cout<<i<<endl;
cout<<i<<endl;

            }
```

```
0
1
2
3
4
5
```

- The body of the for loop is the code block contained within the braces after the while, otherwise it is the first statement after the for only.
- the second cout sentence will execute once since it is outside for body .
- variable i can used inside and outside for body.

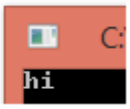
Example9

```
#include <iostream>
using namespace std;
void main()
{

for(int i = 97; i<100 ;i++)
    cout<<char(i)<<endl;
i++;
cout<<i;

}
```



Example10

```
#include <iostream>
using namespace std;
void main()
{

for (int i = -21 ;i<3;i*=-2 )
{
    cout<<"hi"<<endl;
    i++;
}
}
```



I

output

-21

is -21<3 ?? --> true

hi

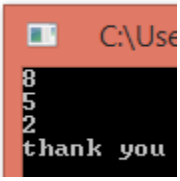
-20

-20*-2=40

is 40<3 ??--> false stop

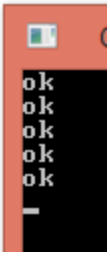
Example11

```
#include <iostream>
using namespace std;
void main()
{

for (int z = 8 ;z>0;z-=3 )
    cout<<z<<endl;
 cout<<"thank you";
}
```



Example12

```
#include <iostream>
using namespace std;
void main()
{

for (int ok = 5;ok>=5&&ok<10;ok++ )
    cout<<"ok"<<endl;
}
```



Example13

```
#include <iostream>
using namespace std;
void main()
{

for (int counter = 30;counter>0;counter/=2 )
    cout<<counter<<endl;
}
```

```
30
15
7
3
1
```

Example14

```
#include <iostream>
using namespace std;
void main()
{
for(int i=1;i<5;++i)
cout<<"loop"<<endl;

}
```

```
loop
loop
loop
loop
```

```
#include <iostream>
using namespace std;
void main()
{
for(int i=1;i<5;i++)
cout<<"loop"<<endl;

}
```

```
loop
loop
loop
loop
```

- The two for loops above will print loop 4 times. There is no difference if the increment/decrement is pre or post inside the third part of the for loop statement.

Example15

```
#include <iostream>
using namespace std;
void main()
{
    char com='a';

for (char i = 'a';i<'c';i++ )
{
    cout<<com<<"\t "<<i<<endl;
    com+=2;
    switch(com)
        case 'c' :
            cout<<"you get c"<<endl;
}
}
```

```
a        a
you get c
c        b
```

Example16

```
#include <iostream>
using namespace std;
void main()
{
for (int e = 1;e<=3;cout<<++e<<endl )

    cout<<e<<endl;


}
```

```
1
2
2
3
3
4
```

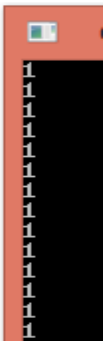
```
#include <iostream>
using namespace std;
void main()
{
for (int e = 1;e<=3;cout<<e++<<endl )

    cout<<e<<endl;


}
```

```
1
1
2
2
3
3
```


alaa abu srhan --email: alaaa@hu.edu.jo

Example17

```
#include <iostream>
using namespace std;
void main()
{
for (int e = 1;e<=3;cout<<e<<endl )

    cout<<e<<endl;


}
```



- **infinite loop --> logical error**

Example18

```
#include <iostream>
using namespace std;
void main()
{
    int i, c=0;
    for(i=12;i>5;i-=3)
    {
      c++; }
    cout<<c;
}
```

Example19

```
#include <iostream>
using namespace std;
void main()
{
    for(int e =1; e<=3,cout<<e<<endl)
        cout<<e<<endl;
}
```

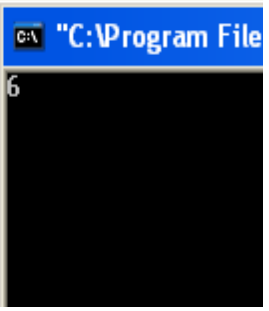
Error: expected a ';'

- **syntax error**

Example19

```
#include<iostream>
using namespace std;
void main()
{
    for(int x=1;x<5;x++);
        cout<<x<<endl;
}
```


```
#include<iostream>
using namespace std;
void main()
{
    for(int x=1;x<5;x++);
    x++;
        cout<<x<<endl;
}
```



- putting ; after **for** mean no body for **for**

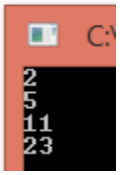
Example20

```
#include <iostream>
using namespace std;
void main()
{
    int e =7;
for (int e = 1;e<=3;e++ )
        cout<<e<<endl;
}
```

- **syntax error,** if we declare variable outside and inside for sentence

Example21

```
#include <iostream>
using namespace std;
void main()
{
for(i=1;i++<12; i*=2 )

cout<<i<<endl;
cout<<i<<endl;

}
```

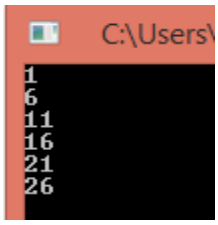


Example22

how many time **cout<<e<<endl;** will execute??

```
#include <iostream>
using namespace std;
void main()
{
for (int e = 1;e<=30;e+=5 )
        cout<<e<<endl;
}
```

answer : 6



Example23

what this code do ??

```
#include <iostream>
using namespace std;
void main()
{
for (int e = 1;e<100;e++ )
        if (e%2==0)
            cout<<e<<endl;

}
```

0

answer: print even numbers between **1 and 99** on screen .

Example24

```
#include <iostream>
using namespace std;
void main()
{
for(float i=5.5;i<10;i++)
cout<<i<<endl;

}
```

```
5.5
6.5
7.5
8.5
9.5
```

Note that even if variable ***i*** is float but it is used as integer counter in the for loop, however it's value is printed as float inside the body.

■ Choosing the Right Looping Structure

- ■ All three loops have their place in C++
  - ○ If you know or can determine in advance the number of repetitions needed, the for loop is the correct choice
  - ○ If you do not know and cannot determine in advance the number of repetitions needed, and it could be zero, use a while loop
  - ○ If you do not know and cannot determine in advance the number of repetitions needed, and it is at least one, use a do...while loop

■ break and continue Statements

break:

- ■ Causes immediate exit from a **while**, **for**, **do/while** or **switch** structure

alaa abu srhan --email: alaaa@hu.edu.jo

- Program execution continues with the first statement after the structure
- Common uses of the **break** statement:
    - Escape early from a loop
    - Skip the remainder of a **switch** structure
- break statement is used for two purposes:
- To exit early from a loop
    - Can eliminate the use of certain (flag) variables
- To skip the remainder of a switch structure
- After break executes, the program continues with the first statement after the structure

Example1

```
#include <iostream>
using namespace std;
void main()
{
for (int x=1;x<10;x++)
{
    cout<<x;
    break;
}
}
```

```
1
```

```
#include <iostream>
using namespace std;
void main()
{
for (int x=1;x<10;x++)
{
    cout<<x<<endl;
    break;
}
cout<<"ok";
}
```

```
1
ok
```

- break mean exit **loop** and continue with the first statement after **loop(loop: for, while, do while)**

Example2

```
#include <iostream>
using namespace std;
void main()
{
    int x=1;
    while(x<=7)
    {
        if(x%2==0)
            break;
        x++;
        cout<<"ok";
    }
}
```

```
ok
```

Example3

```
#include <iostream>
using namespace std;
void main()
{
    int x=-2;
    while(x<2)
    {
        if(!x)
            break;
        cout<<x++;
    }
}
```

C:\Users\

-2-1

Example4

```
#include <iostream>
using namespace std;
void main()
{
for (int x=1;x<3;x++)
{
    cout<<x<<endl;
}
break;
}
```

Error: a break statement may only be used within a loop or switch

- Using **break** outside a loop or **switch** (e.g. inside if/else) statement is ***a syntax error***.

Example5

```
#include <iostream>
using namespace std;
void main()
{
    int x=1;
for ( ; ; )
    cout<<x;

}
```

C:\Users\eng.alaa\do

```
#include <iostream>
using namespace std;
void main()
{
for ( ; ; )
    break;
}
```

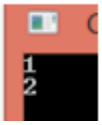
C:\

continue:

- Skips the remaining statements in the body of a **while**, **for** or **do/while** structure and proceeds with the next iteration of the loop. Also, can be used with **switch**.
- In **while** and **do/while**, the loop-continuation test is evaluated immediately after the **continue** statement is executed.
- In the **for** structure, the increment/decrement expression is executed, then the loop-continuation test is evaluated.

Example1

```
#include <iostream>
using namespace std;
void main()
{
for (int x=1;x<3;x++)
{
    cout<<x<<endl;
    continue;
    cout<<"no";

}

}
```



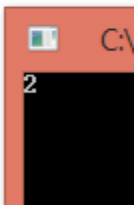
Example2

```
#include <iostream>
using namespace std;
void main()
{

for (int x=1;x<10;x++)
{
if (x%2==0)
continue;
cout<<x<<endl;
}
}
```

- In the **for** structure, the increment/decrement expression is executed, then the loop-continuation test is evaluated

Example3

```
#include <iostream>
using namespace std;
void main()
{
    int x=1;
while(x<10)
{
if (x%2==0)
continue;
x++;
cout<<x<<endl;
}
}
```

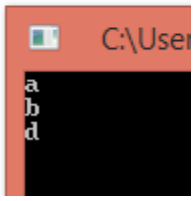


infinate loop --> logical error

- In **while** and **do/while**, the loop-continuation test is evaluated immediately after the **continue** statement is executed

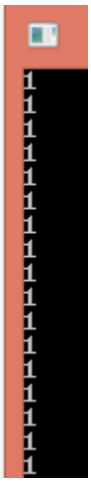
Example3

```
#include <iostream>
using namespace std;
void main()
{

for (char e='a';e<='d';e++)
{
if (e=='c')
continue;
cout<<e<<endl;
}
}
```



Example4

```
#include <iostream>
using namespace std;
void main()
{

for (int x=1;x<3;)
{
cout<<x<<endl;
continue;
x++;
}
}
```



- infinite loop-->logical error

Example 5

what this C++ code do?

```
#include <iostream>
using namespace std;
void main()
{
int c = 2, sum = 0;
while (c <= 10)
{
    c++;
    if (c%2==0)
        continue;
    sum += c;
}
cout<<sum<<endl;


}
```

answer : sum of odd numbers between 3 and 11 [3,11]

Example6

```
#include <iostream>
using namespace std;
void main()
{
for (int x=1;x<3;x++)
{
    cout<<x<<endl;
     cout<<"no";

}
continue;
```

Error: a continue statement may only be used within a loop

- Using continue outside a loop or switch (e.g. inside if/else) statement is a **syntax error.**

Example:

Write the pseudocode to create the following multiplication table:

```
1   2   3   4   5   6   7   8   9  10
2   4   6   8  10  12  14  16  18  20
3   6   9  12  15  18  21  24  27  30
4   8  12  16  20  24  28  32  36  40
5  10  15  20  25  30  35  40  45  50
```

Example:

- To create the following pattern:

```
*
**
***
****
*****
```

- We can use the following code:

```
for (i = 1; i <= 5 ; i++)
{
        for (j = 1; j <= i; j++)
                cout << "*";
        cout << endl;
}
```

The Hashemite University

Computer Programming (C++)

Part 2

For Faculty of Engineering

(110400102)

Lecturer: Alaa Abu-Srhan

alaa abu srhan --email: alaaa@hu.edu.jo

Course Syllabus

Hashemite University

College of Engineering

(3 Credit Hours/Fac. Compulsory)

Course Name:	**Computer Programming**
Course Number:	110400102
Prerequisite:	110108099
Textbook:	**C++ Programming: From Problem Analysis to Program** D.S. Malik, 6th Edition
References	**C++ How to Program**, Paul J. Deitel and Harvey Deitel, Pearson, 4th edition.
Course Description:	This course covers main topics of C++ programming including C++ fundamentals, operations, elements, structured methods, variables, assignment, Input/Output, control structures, functions, arrays, pointer, strings and classes.
Course Learning Outcomes (CLOs):	CLO1: **understand** basic programming structures. (a, c) CLO2: **design** C++ program to perform predefined task (c, k). CLO3: **analyze** written C++ program to predict output (c, k). CLO4: **develop**, **debug** and **run** C++ programs on Visual Studio (k)
Important material	- Lecture notes - References - Internet resources

Major Topics Covered and Schedule:

Topic	Chapter	# Lectures
Introduction to computers and programming languages	Chapter 1	2

alaa abu srhan --email: alaaa@hu.edu.jo

Basics of C++ – Data types, variables – Arithmetic expressions, operators, assignment, increment, decrement	Chapter 2	6
Input/ Output Basics	Chapter 3	2
Quiz		
Control Structure I (Selection) – Relational and logical operators – "if, if … else" – Switch Structure	Chapter 4	5
Control Structure II (Repetition) – Loops: "while" Loop, "for" Loop and "do… while" Loop. – Nested control structure	Chapter 5	5
Midterm Exam	March 11, 2019	
Arrays and strings – One dimensional Arrays creation, initialization and manipulation – Strings – Multidimensional Arrays	Chapter 7, 8	4
Homework		
User defined functions – Predefined functions, user defined functions – Value returning functions, void functions – Value Parameters – Reference Variables as Parameters – Value and Reference Parameters and Memory Allocation	Chapter 6	8

– Reference Parameters and Value-Returning Functions Scope of an Identifier – Global Variables, Named Constants, Static and Automatic Variables – Function Overloading – Functions with Default Parameters – Recursive function – Arrays as a parameter to function		
POINTERS – Pointer Data Type and Pointer Variables – Address of Operator (&) and dereferencing Operator (*) – Pointers with arrays – Pointers as a parameter to functions	Chapter 12	4
In-lab Assignment		

Course Policy

- Course Website (Moodle): http://www.mlms.hu.edu.jo/. Students are asked to check the website regularly for announcements.
- Students are responsible for the reading assignments from the text and handouts
- Students are responsible for following up the lecture materials
- If you miss class, there won't be a makeup test, quiz, etc. and you WILL get a zero unless you have a valid excuse.
- Cheating and plagiarism are completely prohibited.
- If you miss more than 15% of classes you will automatically fail the class.
- **Grading policy:**
    - **Midterm exam: 35%**
    - **Quiz, homework and in-lab Assignment: 25%**

- **Final exam: 40%**

- **Midterm Exam will be held in March 11, 2018**

Chapter 8
Array

Outline:

- Arrays
- Searching an Array for a Specific Item
- C-Strings (Character Arrays)
- Parallel Arrays
- Two- and Multidimensional Arrays

- **Simple data type:** variables of these types can store only one value at a time
- **Structured data type:** a data type in which each data item is a collection of other data items
- An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.
- That means that, for example, five values of type int can be declared as an array without having to declare 5 different variables (each with its own identifier). Instead, using an array.
- **Format:**

```
dataType arrayName[intExp];
```

  - **intExp: any constant expression that evaluates to a positive integer**

Example1:

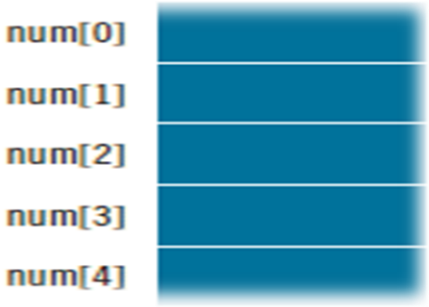
```
#include <iostream>
using namespace std;
void  main()
{
    int a[10];

}
```

- we have array a that has 10 element
  - First element at position 0
  - last element at position n-1. where n is number of elements
  - a[0] :first element , a[1] : is 2nd element … a[9] : last element.

Example 2

int num[5];

- declares an array num of five components. Each component is of type int. The components are num[0], num[1], num[2], num[3], and num[4].



- **Basic operations on a one-dimensional array:**
  - Initializing
  - Inputting data
  - Outputting data stored in an array

- Finding the largest and/or smallest element

- Declaring arrays - specify: Name, Type of array, Number of elements

Example 1

```
#include <iostream>
using namespace std;
void  main()
{
    int a[10], b[5];
}
```

- Declaring multiple arrays of same type

Example 2

```
int list[10];
```

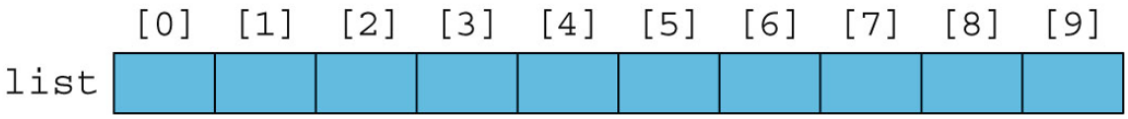
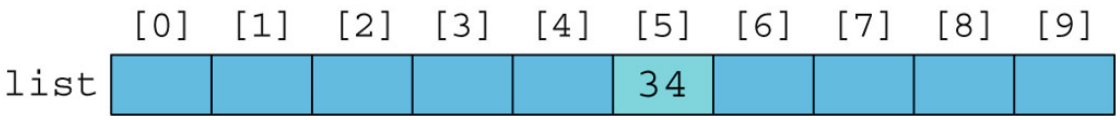


FIGURE 8-3 Array list

```
list[5] = 34;
```




```
list[3] = 10;
list[6] = 35;
list[5] = list[3] + list[6];
```

- Value of the **index** is the position of the item in the array
- []: array subscripting operator
- Array index always starts at 0

Example3:

```
#include <iostream>
using namespace std;
void  main()
{
    int  b[5]={ 1, 2, 3, 4, 5 };

}
```

- initialize an array (b) Using an initializers list.
- Arrays can be initialized during declaration
- Values are placed between curly braces
- Size determined by the number of initial values in the braces

Example 4

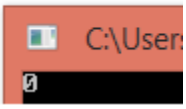
```
#include <iostream>
using namespace std;
void  main()
{
    int  b[5]={ 1, 2, 3, 4, 5,6 };

}
```

Error: too many initializer values

- If too many initializers, a **syntax error** is generated

Example 5

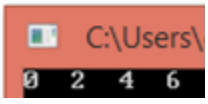
```
#include <iostream>
using namespace std;
void  main()
{
    int  b[5]={ 1, 2, 3, 4};
        cout<<b[4];

}
```



- If not enough initializers, rightmost elements become 0.
- Cout<<b[4]; → array element that have index = 4.

Example 6

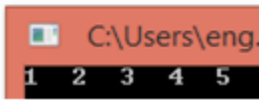
```
#include <iostream>
using namespace std;
void  main()
{
    int  b[5];
    for (int i=0;i<=3;i++)
        b[i]=i*2;
    for (int i=0;i<=3;i++)
        cout<<b[i]<<"  ";

}
```



- We can initialize an array Using a loop.

Example 7

```
#include <iostream>
using namespace std;
void  main()
{
    int  b[5]={ 1, 2, 3, 4, 5};
    for (int i=0;i<=4;i++)
        cout<<b[i]<<"  ";

}
```



- to output all array elements use loop (for or while)

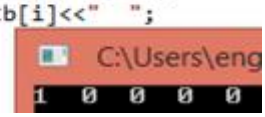
Example 8

```
#include <iostream>
using namespace std;
void  main()
{
    int  b[5]={0};
    for (int i=0;i<=4;i++)
        cout<<b[i]<<"  ";

}
```

- Sets all the elements to 0 since the first element is initialized to 0 and the rest are implicitly initialized to 0.

Example 9

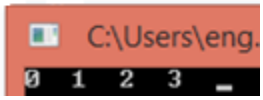
```
#include <iostream>
using namespace std;
void  main()
{
    int  b[5]={1};
    for (int i=0;i<=4;i++)
        cout<<b[i]<<"  ";

}
```



- the first element is initialized to 1 and the rest are implicitly initialized to 0.

Example 10

```
#include <iostream>
using namespace std;
void  main()
{
    int  b[]={0,1,2,3};
    for (int i=0;i<=3;i++)
        cout<<b[i]<<"  ";

}
```



- If size omitted, the initializers determine it
- 4 initializers, therefore **b** is a 4 elements array

Example 11

```
#include <iostream>
using namespace std;
void main()
{
    int b[4] = {1,2,,9};
    cout<<b[0];

}
```

Error: expected an expression

- **syntax error**

Example 12

```
#include <iostream>
using namespace std;
void  main()
{
    int  b[5];
    for (int i=0;i<=3;i++)
        b[i]=i*2;
    cout<<"index"<<"  "<<"element"<<"  "<<endl;
    for (int i=0;i<=3;i++)
        cout<<i<<"      "<<b[i]<<"      "<<endl;

}
```

```
C:\Users\eng.
index   element
0       0
1       2
2       4
3       6
```

- i^{th} element in an array has the index (or subscript) of i – 1.
- The subscript must be an integer value, where it could be: Constant, Variable, Expression, and A result of a function call.

Subscript is always int variable or value

Example 13

```
#include <iostream>
using namespace std;
void main()
{
    int b[6]={0} ;
    for(int i=0;i<6;i++)
    {
        if (i%2==0)
            b[i]=i*2;
        cout<<b[i]<<endl;
    }

}
```

```
0
0
4
0
8
0
```

More Examples

Example 1

- **Write the required code to do the following:**

1. **Define an array sales of 10 components of type double.**

```
double sales[10];
```


alaa abu srhan --email: alaaa@hu.edu.jo

2. **initializes every component of the array sales to 0.0**

```
for (int index = 0; index <
10; index++)
   sales[index] = 0.0;
```

3. **Reading data from user into an array:**

```
for (index = 0; index < 10;
index++)
   cin >> sales[index];
```

4. **Printing an array**

```
for (index = 0; index < 10;
index++)
   cout << sales[index] << " ";
```

5. **Finding the sum and average of an array**

```
double sum = 0;
for (index = 0; index < 10;
index++)
   sum = sum + sales[index];
double average = sum / 10;
```

6. Largest element in the array:

alaa abu srhan --email: alaaa@hu.edu.jo

```
double maxIndex = 0;
for (index = 1; index < 10;
index++)
   if (sales[maxIndex] <
   sales[index])
      maxIndex = index;
largestSale = sales[maxIndex];
```

- Example:

```
int myList[5] = {0, 4, 8, 12, 16};  //Line 1
int yourList[5];  //Line 2

yourList = myList;  //illegal

cin >> yourList; //illegal
```

- Solution:

```
for (int index = 0; index < 5; index ++)
     yourList[index] = myList[index];
```

- **Aggregate operation**: any operation that manipulates the entire array as a single unit is Not allowed on arrays in C++

Example 2

```
#include <iostream>
using namespace std;
void main()
{
    int b[6]={1} ;
    for(int i=0;i<5;i++)
    {
            b[i+1]=b[i];
        cout<<b[i]<<endl;
    }

}
```

```
1
1
1
1
1
```

Example 3

```
#include <iostream>
using namespace std;
void  main()
{ const int x=10;
cout<<x;

}
```

```
C:
10_
```

- const : also called named constants or read-only variables, mean that x has fixed value

Example 14

```
#include <iostream>
using namespace std;
void  main()
{ const int x=10;
x=5;

}
```

```
const int x = 10

Error: expression must be a modifiable lvalue
```

- constant variable cannot be modified throughout the program after it is being declared.
- modified it is syntax error.

Example 15

```
#include <iostream>
using namespace std;
void  main()
{ const int x=3;
int z[x]={1,2,3};
for (int i=0 ;i<3 ;i++)
    cout<<z[i]<<" ";

}
```

```
C:\U
1 2 3
```

- Arrays sizes are usually declared with type const since they are static (fixed).

Example 16

```
#include <iostream>
using namespace std;
void  main()
{  int x=3;
int z[x]={1,2,3};
for (int i=0 ;i<3 ;i++)
    cout<<z[i]<<" ";
}
```

int x

Error: expression must have a constant value

- If you want to declare the size of an array using a variable this variable must be declared const, otherwise you will get a ***syntax error***.

Example 17

```
#include <iostream>
using namespace std;
void main()
{
    int x[1]={1};
    for(int i =0;i<4;i++)
    cout<<x[i]<<endl;

}
```

```
C:\Users
1
-858993460
16775104
20388319
```

- Going outside the range of an array is a ***logical error*** in C++.
- Index of an array is in bounds if the index is >=0 and <= ARRAY_SIZE-1 Otherwise, the index is out of bounds
- In C++, there is no guard against indices that are out of bounds

Example 18

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

void main()
{
    double x[]={5.2,4.5,5.2,6.3,7.5};
    cout<<x[2.0];

}
```

Error: expression must have integral or enum type

- **syntax error** (index is always positive integer number)

Base Address of an Array and Array in Computer Memory

- Base address of an array: address (memory location) of the first array component
- Example:
  - If list is a one-dimensional array, its base address is the address of list[0]

alaa abu srhan --email: alaaa@hu.edu.jo

- **What is the output of the following statements?**

int myList[4]={1,5,3,2};
cout << myList[0];
cout<< myList;

output:
1
1000

Size of Array

example 1

```
#include <iostream>
using namespace std;
void main()
{
    int myArray[10];
cout << sizeof(myArray);

}
```

```
C:\
40
```

- For arrays, sizeof returns

(the size of 1 element) * (number of elements)

example 2

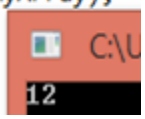
```
#include<iostream>
using namespace std;
void main()
{
    int x[3][4];
    cout<<sizeof(x);

}
```

```
48
```

- number of element =3*4 =12, size of one element =4, so result will be 4*12=48

example3

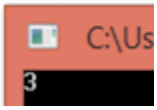
```
#include <iostream>
using namespace std;
void main()
{
    int myArray[]={1,2,3};
cout << sizeof(myArray);

}
```



note that : number of element = 3

example 4

```
#include <iostream>
using namespace std;
void main()
{
    int myArray[]={1,2,3};
int d= sizeof(myArray);
int f = sizeof (int);
cout<<d/f;

}
```



- To get the size of an array (number of elements) using sizeof operator do the following:

number of element = **sizeof(myArray)/ sizeof(int);**

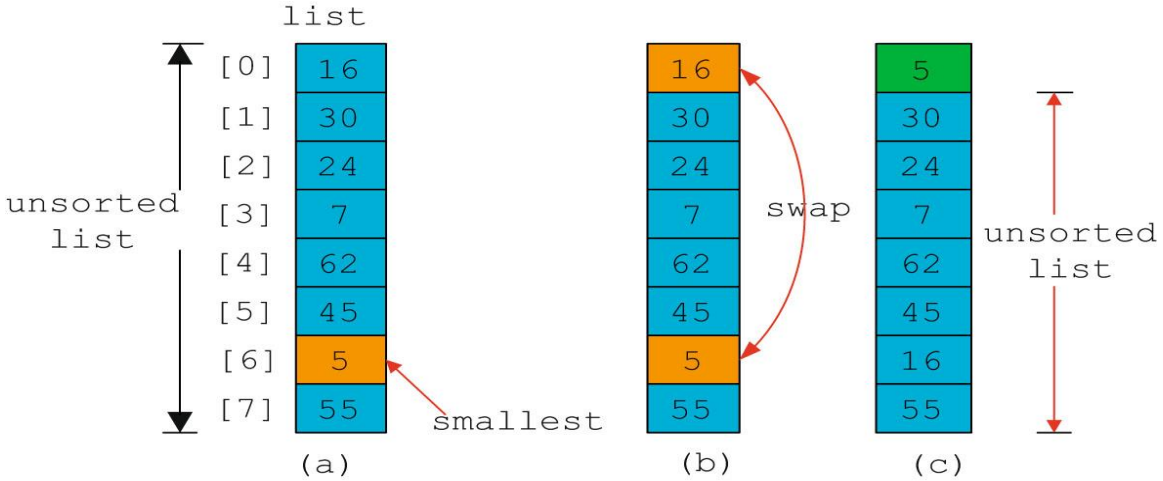
Searching an Array for a Specific Item

- Sequential search (or linear search):
  - Searching a list for a given item, starting from the first array element
  - Compare each element in the array with value being searched for
  - Continue the search until item is found or no more data is left in the list

```cpp
#include <iostream>
using namespace std;
# include<cstdlib>
#include<ctime>
void main()
{
    int list[10];
    int searchitem;
    cout << "please enter a number between 0 to 50 you want to search for it" << endl;
    cin >> searchitem;
    int listlength = sizeof(list)/sizeof(int);
    srand(time(0));
    for (int i=0;i<=9;i++)
    {
        list[i] =rand()%50;
    }
        int loc = 0;
        bool found = false;
        while (loc < listlength && !found)
        {
            if (list[loc] == searchitem)
                found = true;
            else
                loc++;
            if (found)
                cout <<"the location of your search item is : " <<loc << endl;
            else cout << "not found"<<endl;
        }
        system("pause");
}
```

Selection Sort

- Selection sort: rearrange the list by selecting an element and moving it to its proper position
- Steps:
  - Find the smallest element in the unsorted portion of the list
  - Move it to the top of the unsorted portion by swapping with the element currently there
  - Start again with the rest of the list



```cpp
#include <iostream>
using namespace std;
void main()
{
    int list[10];
    int listlength = sizeof(list)/sizeof(int);
    for (int i=0;i<listlength;i++)
    {
        cout << "enter the "<<i+1 <<" element"<<endl;
        cin >> list[i];
    }
    int index;
    int smallestindex;
    int location;
    int temp;
    for (index = 0; index < listlength; index++)
    {
        smallestindex = index;
        for (location = index + 1; location < listlength; location++)
            if (list[location] < list[smallestindex])
                smallestindex = location;
        temp = list[smallestindex];
        list[smallestindex] = list[index];
        list[index] = temp;
    }

    for (int x = 0; x<listlength; x++)
    {
        cout << list[x] << "\t";

    }
    system("pause");
}
```


alaa abu srhan --email: alaaa@hu.edu.jo

C-Strings (Character Arrays)

- Character array: an array whose components are of type char
- Strings is the same as array of characters.
- C-strings are null-terminated ('\0') character arrays

Example:

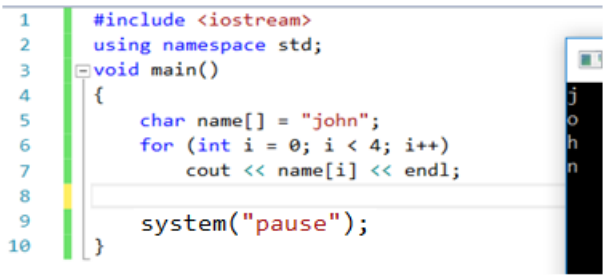
- 'A' is the character A
- "A" is the C-string A
- "A" represents two characters, 'A' and '\0'

Example:

```
char name[16];
```

- Since C-strings are null terminated and name has 16 components, the largest string it can store has 15 characters
- If you store a string whose length is less than the array size, the last components are unused

Example:



```
#include <iostream>
using namespace std;
void main()
{
    char name[] = "john";
    for (int i = 0; i < 4; i++)
        cout << name[i] << endl;

    system("pause");
}
```

```
j
o
h
n
```

- Size of an array can be omitted if the array is initialized during declaration
- Declares an array of length 5 and stores the C-string "John" in it.

Useful string manipulation functions

- **strcpy, strcmp, and strlen**

Function	Effect
`strcpy(s1, s2)`	Copies the string `s2` into the string variable `s1` The length of `s1` should be at least as large as `s2`
`strcmp(s1, s2)`	Returns a value < `0` if `s1` is less than `s2` Returns `0` if `s1` and `s2` are the same Returns a value > `0` if `s1` is greater than `s2`
`strlen(s)`	Returns the length of the string `s`, excluding the null character

alaa abu srhan --email: alaaa@hu.edu.jo

Example

```
#include <iostream>
using namespace std;
void main()
{
    char studentname[21];
    char myname[16];
    char yourname[16];
    strcpy_s(myname, "john Robinson");
    cout<<strlen("john Robinson")<<endl;
    strcpy_s(yourname, "Lisa Miller");
    strcpy_s(studentname, yourname);
    cout << strcmp("Bill", "lisa") << endl;
    cout << strcmp("Air", "Boat") << endl;
    cout << strcmp("Air" , "An" )<< endl;
    cout << strcmp("Bill" , "Billy") << endl;
    cout << strcmp("Hello" , "hello") << endl;
    cout << studentname<<endl;
    cout << myname << endl;
    cout << yourname << endl;

    system("pause");
}
```

```
c:\users\eng\documents\visual studio 201
13
-1
-1
-1
-1
-1
Lisa Miller
john Robinson
Lisa Miller
Press any key to continue . . .
```

String Input

Example:

```
#include <iostream>
using namespace std;
void main()
{
    char name[50];
    cin.get(name, 50);
    cout << name << endl;
    system("pause");
}
```

```
c:\users\eng\documents\visual studio 2015\P
ahmad ali
ahmad ali
Press any key to continue . . .
```

cin >> name;

- Stores the next input C-string into name

- To read strings with blanks, use get function:

cin.get(str, m+1);

- Stores the next m characters into str but the newline character is not stored in str
- If input string has fewer than m characters, reading stops at the newline character

String Output

Example:

cout << name;

- Outputs the content of name on the screen
- << continues to write the contents of name until it finds the null character

alaa abu srhan --email: alaaa@hu.edu.jo

- If name does not contain the null character, then strange output may occur
  - << continues to output data from memory adjacent to name until a '\0' is found

String VS character array

String	Character array
string n1="ahmad"; string n2="ali"; string n3; n3=n2;	char n1[10]={'a','h','m','a','d'}; char n2[10]="ali"; char n3[10]; strcpy(n3,n2);

If(n1<n2)
cout<<n1.length();
else{
n3=n1+n2;
cout<<n3;}
	if(strcmp(n1,n2)<0)
	cout<<strlen(n1)
	else
	{
	strcpy(n3,n1);
	strcat(n3,n2);
	cout<<n3

cin>>n1;
getline(cin,n2);
	cin>>n1;
	getline(n2);
	cin.get(n6,10);

Parallel Arrays

- Two (or more) arrays are called parallel if their corresponding components hold related information

- Example:

```
int studentId[50];
char courseGrade[50];
```

```
23456 A
86723 B
22356 C
92733 B
11892 D
.
.
.
```

Example

```
#include <iostream>
using namespace std;
void  main()
{
char z[5]={'h','e','l','l','o'};
for (int i=0 ;i<5 ;i++)
    cout<<z[i]<<" ";

}
```

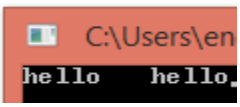
- Character array initialization With initializers

Example

```
#include <iostream>
using namespace std;
void  main()
{
char z[6]="hel";
for (int i=0 ;i<5 ;i++)
    cout<<z[i]<<" ";

}
```

- If you initialize a string with smaller size than the array size it will automatically add spaces at the end of the string within the array .
- in the example size = 6 and we enter 3 char so the output will be hel and three spaces

```
#include <iostream>
using namespace std;
void main()
{
    char b[8]="hello";

    for(int i=0;i<8;i++)
        cout<<b[i];
    cout<<"hello";

}
```



Example

```
#include <iostream>
using namespace std;
void  main()
{
char z[6]="hello";
cout<<z[1]<<endl;
cout<<z[3];


}
```

```
e
l
```

Example

```
#include <iostream>
using namespace std;
void  main()
{
char z[6]="hello";

for (int i=0 ;z[i]!='\0';i++)
    cout<<z[i];

}
```

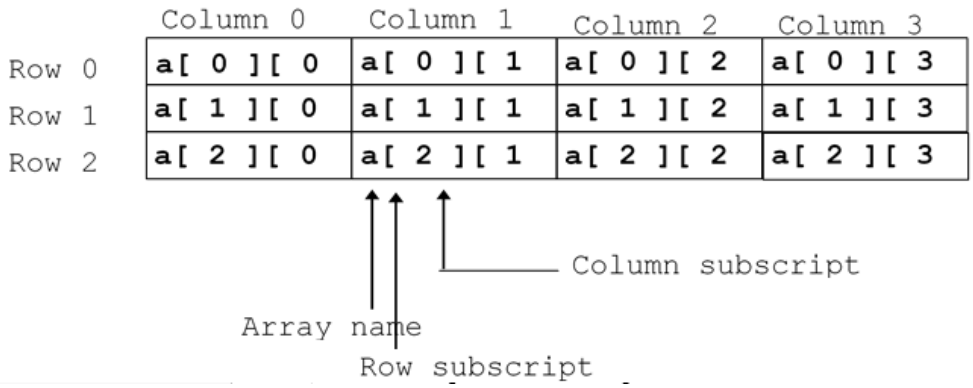
```
C:\Users
hello
```

- cout stop printing an array of character when it reaches the '\0'.

Multiple-Subscripted Arrays

- Two-dimensional array: collection of a fixed number of components (of the same type) arranged in two dimensions

Sometimes called matrices or tables



- Multiple subscripts - tables with rows, columns, Like matrices: specify row, then column.
- **Declaration syntax:**

```
dataType  arrayName[intExp1][intExp2];
```

  - intExp1 and intExp2 are expressions with positive integer values specifying the number of rows and columns in the array

- **Accessing components in a two-dimensional array:**

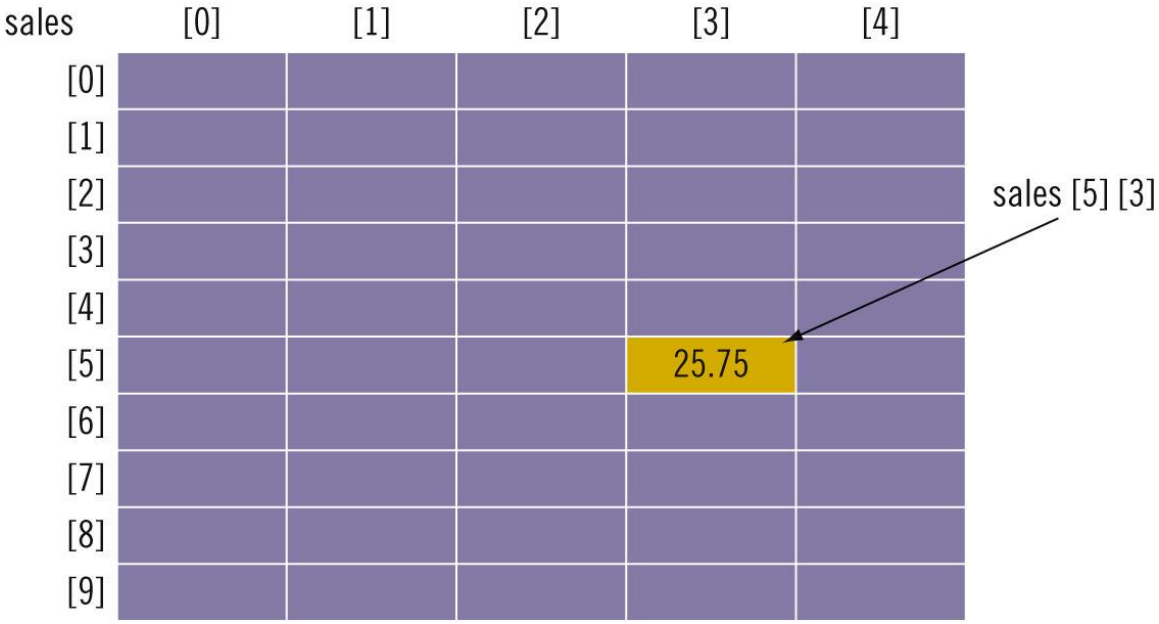
```
arrayName[indexExp1][indexExp2]
```

- Where indexExp1 and indexExp2 are expressions with positive integer values, and specify the row and column position

- **Processing Two-Dimensional Arrays**
  - Ways to process a two-dimensional array:
    - Process entire array
    - Row processing: process a single row at a time
    - Column processing: process a single column at a time
  - Each row and each column of a two-dimensional array is a one-dimensional array
    - To process, use algorithms similar to processing one-dimensional arrays

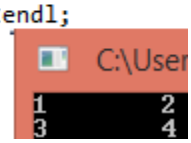
- Example:

sales[5][3] = 25.75;



Example

```
#include <iostream>
using namespace std;
void main()
{
 int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };
cout<<b[0][0]<<"\t"<<b[0][1]<<endl;
cout<<b[1][0]<<"\t"<<b[1][1]<<endl;

}
```



- Using initializers list: Initializers grouped by row in braces

Example

```
int board[4][3] = {{2, 3, 1},
                   {15, 25, 13},
                   {20, 4, 7},
                   {11, 18, 14}};
```

board	[0]	[1]	[2]
[0]	2	3	1
[1]	15	25	13
[2]	20	4	7
[3]	11	18	14

Example

```
#include <iostream>
using namespace std;
void main()
{
 int b[ 2 ][ 3 ] ;
 for(int i=0;i<2;i++)
 {
     for(int j=0;j<3;j++)
     {
         b[i][j]=i+1;
     }
 }
  for(int i=0;i<2;i++)
 {
     for(int j=0;j<3;j++)
     {
         cout<<b[i][j]<<"\t";
     }
     cout<<endl;
 }
}
```

Initialization:
Using two nested for loops to

show output:

Using two nested for loops.

```
C:\Users\eng.alaa
1       1       1
2       2       2
```

Example

```
#include <iostream>
using namespace std;
void main()
{
 int b[ 2 ][ 3 ] ;
cout<<b[1][1];

}
```

```
C:\Users\e
-858993460_
```

```
#include <iostream>
using namespace std;
void main()
{
 int b[ 2 ] ;
cout<<b[1];

}
```

```
C:\Users\eng.
-858993460_
```

- If you declare an array (onesubscript or multisubscript) and don't initialize it the elements will have garbage numbers.

Example

```
#include <iostream>
using namespace std;
void main()
{
    int b[ 2 ][ 3 ] ={{1,2},{3}};

  for(int i=0;i<2;i++)
{
    for(int j=0;j<3;j++)
    {
        cout<<b[i][j]<<"\t";
    }
cout<<endl;
}

}
```

```
C:\Users\eng.ala
1       2       0
3       0       0
```

- If you declare an array (onesubscript or multisubscript) and initialize some elements and leave the others uninitialized , the uninitialized elements will be given the value 0 if it is a integer array and assigned a space if it is an character array.

Example

```
#include <iostream>
using namespace std;
void main()
{
    int b[ 2 ][ 3 ] ={{1,2,7,4},{3}};

  for(int i=0;i<2;i++)
{
    for(int j=0;j<3;j++)
    {
        cout<<b[i][j]<<"\t";
    }
cout<<endl;
}

}
```

Error: too many initializer values

- If you try to fill a location (**using initializer list**) that is outside the array boundary syntax error will be generated

Example

board	[0]	[1]	[2]
[0]	2	3	1
[1]	15	25	13
[2]	20	4	7

- To find the sum of each individual column

```
#include <iostream>
using namespace std;
void main()
{
    int bound[3][3] = { {2,3,1},{15,25,13},{20,4,7} };
    int sum;
    for (int col = 0; col < 3; col++)
    {
        sum = 0;
        for (int row = 0; row < 3; row++)
            sum = sum + bound[row][col];
        cout << "sum of column " << col + 1 << "  = " << sum<<endl;
    }
    system("pause");
}
```

```
c:\users\eng\documents\visual studio 2015\Pr
sum of column 1  = 37
sum of column 2  = 32
sum of column 3  = 21
Press any key to continue . . .
```

- To find the sum of each individual row

```
#include <iostream>
using namespace std;
void main()
{
    int bound[3][3] = { {2,3,1},{15,25,13},{20,4,7} };
    int sum;
    for (int row = 0; row < 3; row++)
    {
        sum = 0;
        for (int col = 0; col < 3; col++)
            sum = sum + bound[row][col];
        cout << "sum of row " << row + 1 << "  = " << sum<<endl;
    }
    system("pause");
}
```

```
c:\users\eng\documents\visual studio 201
sum of row 1  = 6
sum of row 2  = 53
sum of row 3  = 31
Press any key to continue . . .
```

- To find the largest element in each row:

```
#include <iostream>
using namespace std;
void main()
{
    int bound[3][3] = { {2,3,1},{15,25,13},{20,4,7} };
    int max;
    for (int row = 0; row < 3; row++)
    {
        max= bound[row][0];
        for (int col = 0; col < 3; col++)
            if (max<bound[row][col])
             max=bound[row][col];
        cout << "the largest elements in row  " << row + 1 << "  = " << max<<endl;
    }
    system("pause");
}
```

```
c:\users\eng\documents\visual studio 2015\Pro
the largest elements in row  1  = 3
the largest elements in row  2  = 25
the largest elements in row  3  = 20
Press any key to continue . . .
```

-

Example

1)
```
#include <iostream>
using namespace std;
void main()
{
    int b[ 2 ][] = { 1, 2, 5 };
}
```
Error: too many initializer values

2)
```
#include <iostream>
using namespace std;
void main()
{
    int b[][] = { 1, 2, 5 };
}
```
Error: an array may not have elements of this type

3)
```
#include <iostream>
using namespace std;
void main()
{
    int b[ 2 ][] = { 1, 2, 5 };
}
```
Error: an array may not have elements of this type

4)
```
#irclude <iostream>
using namespace std;
void main()
{
    int b[][ 2 ][] = { 1, 2, 5 }; ] = { 1, 2, 5 };
}
```
Error: an array may not have elements of this type

- **Note that only the first subscript (or dimension size) is allowed to be empty.**

Example 7

```
#include <iostream>
using namespace std;
void main()
{
    int b[][3] = {{-1,1,2,4}};
}
```
Error: too many initializer values

- If you try to fill a location (**<u>using initializer list</u>**) that is outside the array boundary syntax error will be generated(we have 3 columns and we fill 4 !!!)

Example 8

correct initializations examples :

1. int b[][3] = {{-1,1,2},{4}};

2. int b[2][2] = { 1, 2, 5 };--> we have two rows and two columns, in this case we start read the elements and put 1 and 2 at the first row (since we have 2 columns) and 5 in the first column of second row, and since there are no more elements so it will be 0 in b[1][1](second row, second column).

Example 9

```
#include <iostream>
using namespace std;
void main()
{
    int b[ ][ 2 ] = { 1, 2, 5 };
    for(int i=0;i<2;i++)
    {
        for(int j=0;j<2;j++)
    cout<<b[i][j]<<" ";
    cout<<endl;
    }
}
```
C:\User
1 2
5 0

```
#include <iostream>
using namespace std;
void main()
{
    int b[3 ][ 2 ] = { 1, 2, 5 };
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<2;j++)
    cout<<b[i][j]<<" ";
    cout<<endl;
    }
}
```
C:\User
1 2
5 0
0 0

Arrays of Strings

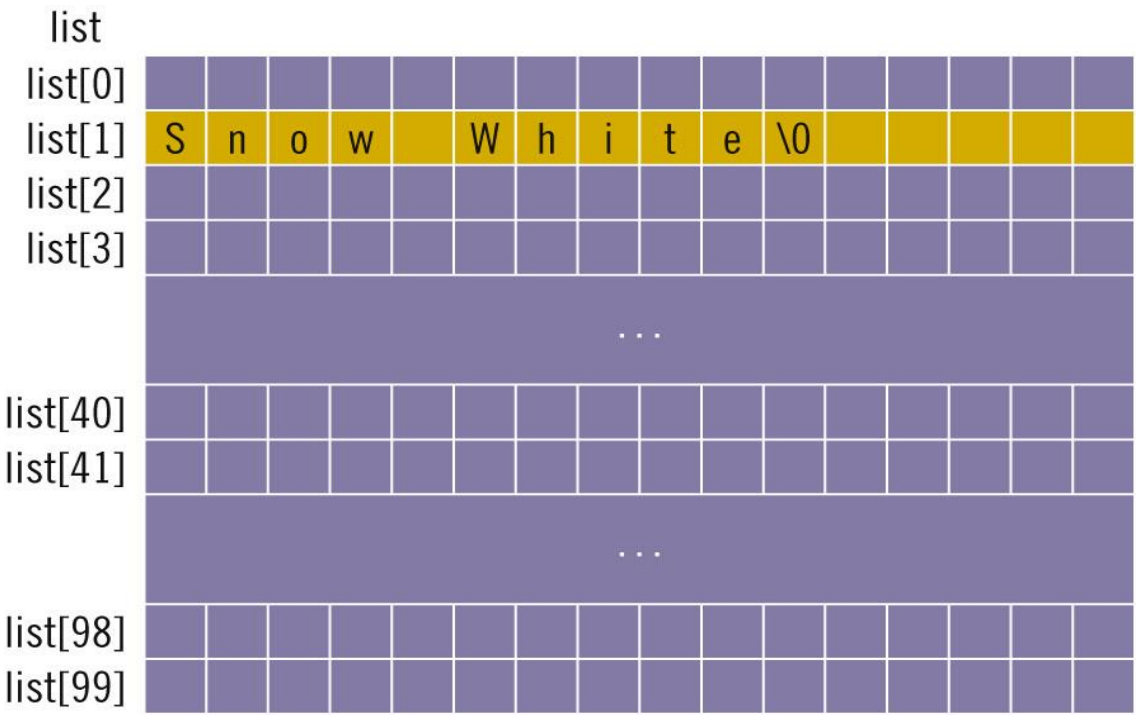
- Strings in C++ can be manipulated using either the data type string or character arrays (C-strings)
- On some compilers, the data type string may not be available in Standard C++ (i.e., non-ANSI/ISO Standard C++)
- To declare an array of 100 components of type string:

```
#include <iostream>
using namespace std;
#include<string.h>
void main()
{
    string list[100];
    system("pause");
}
```

- Basic operations, such as assignment, comparison, and input/output, can be performed on values of the string type
- The data in list can be processed just like any one-dimensional array

Example

```
#include <iostream>
using namespace std;
#include<string.h>
void main()
{
    char list[100][16];
    strcpy(list[1], "snow white");
    system("pause");
}
```



- The following for loop is used to read and store string in each row:

```
#include <iostream>
using namespace std;
#include<string.h>
void main()
{
    char list[100][16];

    for (int j = 0; j < 100; j++)
        cin.get(list[j], 16);
    system("pause");

}
```

The following for loop outputs the string in each row:

```
#include <iostream>
using namespace std;
#include<string.h>
void main()
{
    char list[100][16];

    for (int j = 0; j < 100; j++)
        cin.get(list[j], 16);
    for (int j = 0; j < 100; j++)
        cout<<list[j]<<endl;
    system("pause");

}
```

Chapter 6
Function

In this chapter, you will study:

- Predefined Functions
- User-Defined Functions
- Value-Returning Functions
- Void Functions
- Value Parameters
- Reference Variables as Parameters
- Value and Reference Parameters and Memory Allocation
- Reference Parameters and Value-Returning Functions
- Scope of an Identifier
- Global Variables, Named Constants, and Side Effects
- Static and Automatic Variables
- Function Overloading: An Introduction
- Functions with Default Parameters

Function

- Functions are often called modules
- They are like miniature programs that can be combined to form larger programs
- They allow complicated programs to be divided into manageable pieces
- used when the same code block is used many times within the program.
- int main() is function (main function tells the compiler that you have to start here).
-

function definition :

```
functionType functionName(formal parameter list)
{
     statements
}
```

Where:

- -**functionType** also called the data type or return type: is the type of the value returned by the function.

alaa abu srhan --email: alaaa@hu.edu.jo

- **functionName** is the identifier by which the function can be called.
- **Formal parameter list** (as many as needed):

- Each parameter consists of a type followed by an identifier, with each parameter being separated from the next by **a comma**.
- Each parameter looks very much like a regular variable declaration (for example:int x), and in fact acts within the function as a regular variable which is local to the function.
- The purpose of parameters is to allow passing arguments to the function from the location where it is called from.

-Syntax:

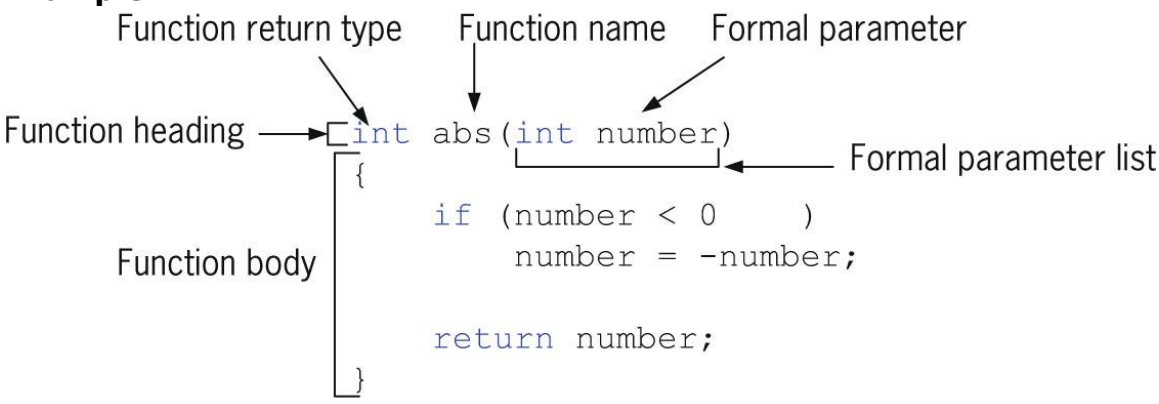
```
dataType identifier, dataType identifier, ...
```

- **Statements** is the function's body. It is a block of statements surrounded by braces { } that specify what the function actually does.

Note: Functions are invoked by a function call

- A function call specifies the function name and provides information (as arguments) that the called function needs.

Example:



Call the function inside main() :

- **Syntax to call a value-returning function:**

```
functionName(actual parameter list)
```

- Syntax of the actual parameter list:

```
expression or variable, expression or variable, ...
```

- Formal parameter list can be empty:

```
functionType functionName()
```

- A call to a value-returning function with an empty formal parameter list is:

```
functionName()
```

- **Function returns its value via the return statement**
  - It passes this value outside the function

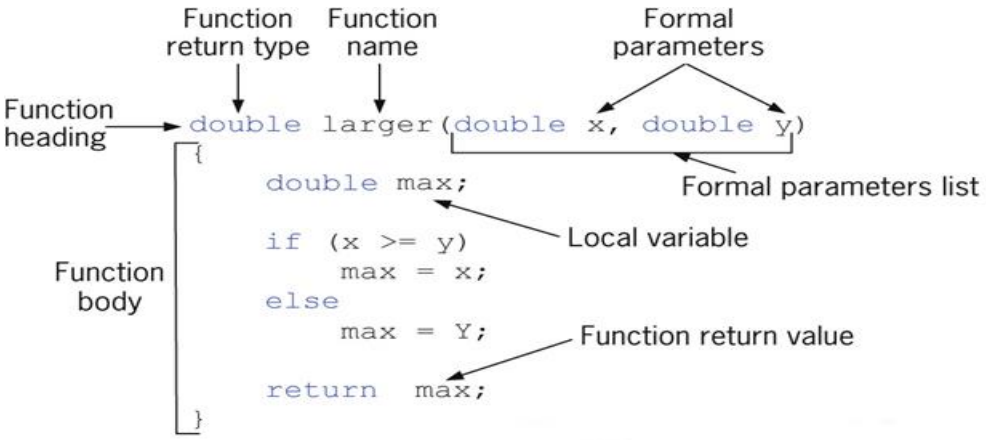
```
return expr;
```

- In C++, return is a reserved word
- When a return statement executes
  - Function immediately terminates
  - Control goes back to the caller
- When a return statement executes in the function main, the program terminates

Local variables

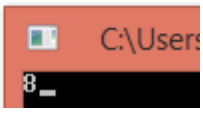
- Known only in the function in which they are defined.
- All variables declared in function definitions are local variables.

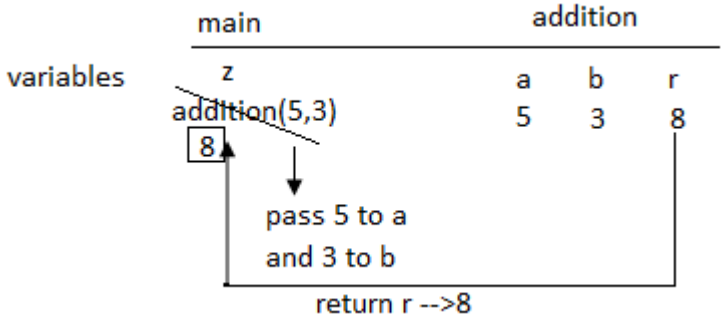
Example



Example

```
#include <iostream>
using namespace std;
int addition (int a,int b)
{
    int r ;
    r=a+b;
    return r ;
}
void main()
{
    int z ;
    z=addition(5,3);
    cout<<z;

}
```





- This program is divided in two functions: addition and main. Remember that no matter the order in which they are defined, a C++ program always **starts by calling main function**.
- the program started from main()
- In fact, main is the only function called automatically.
- the code in any other function is only executed if its function is **called from main or another function** (directly or indirectly).
- addition(5,3); --> function call (call function addition and pass 5 and as parameters or arguments addition), **call by value**

```
int addition (int a, int b)
               ↑     ↑
z = addition (  5  ,  3  );
```

- In a function call, you specify only the actual parameter, not its data type.

Example

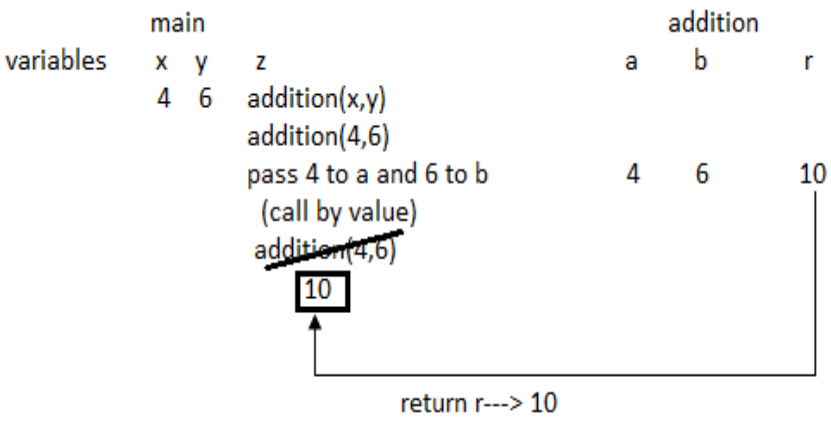
- Once a function is written, you can use it anywhere in the program. Even as a parameter to another function

```
double compareThree(double x, double y, double z)
{
        return larger(x, larger(y, z));
}
```

Example

```
#include <iostream>
using namespace std;
int addition (int a,int b)
{
    int r ;
    r=a+b;
    return r ;
}
void main()
{
    int z ;
    int x=4;
    int y=6;
    z=addition(x,y);
    cout<<z;

}
```

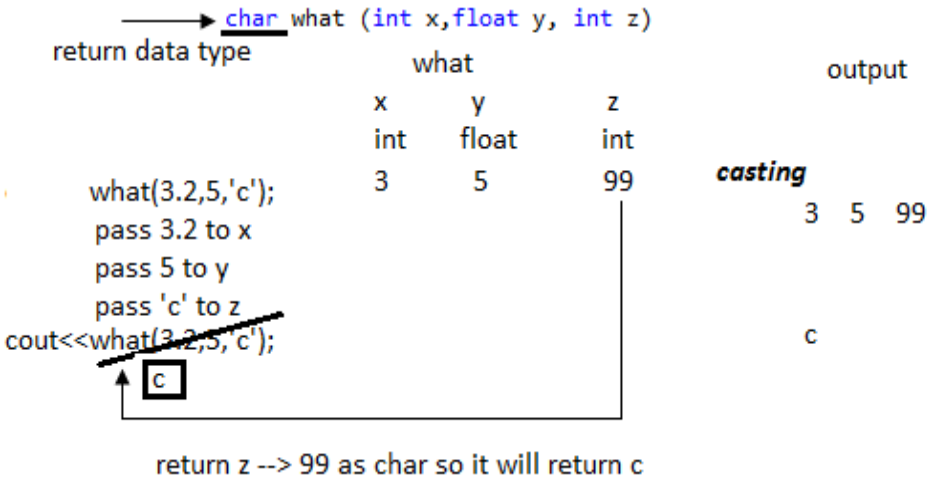
```
10
```



Example

```
#include <iostream>
using namespace std;
char what (int x,float y, int z)
{
    cout<<x<<"   "<<y<<"  "<<z<<endl;
    return z ;
}
void main()
{
    cout<<what(3.2,5,'c');

}
```

```
3   5   99
c
```



Example

```
#include <iostream>
using namespace std;
void  print( )
{
    cout<<"I'am function";
}
int  main()
{
    print();

    return 0;
}
```

```
I'am function
```

- **Print** Function has **void** return data type
- **void** : function return nothing.
- don't use **return** inside **print** function, since this function return nothing .
- **return 0; :** the function returns nothing.
- we can use **return;** : the function returns nothing.

alaa abu srhan --email: alaaa@hu.edu.jo

Example

```
#include <iostream>
using namespace std;
void  print( )
{
    cout<<"I'am function";
}
int  main()
{
    cout<<print();

    retu
}
```

Error: no operator "<<" matches these operands

```
#include <iostream>
using namespace std;
void  print( )
{
    cout<<"I'am function";
}
int  main()
{
    int x=print();

    return
}
```

void print()

Error: a value of type "void" cannot be used to initialize an entity of type "int"

- Calling a function that does not return any value inside cout statement or stored in a variable is **syntax error** (cout<<print() ➔ syntax error)

  void printer1 () function that return nothing (data type is void) so when we call it from main() **we can't put printer1(); in cout statement or store it in variable.**

  Example

```
#include <iostream>
using namespace std;
void  print( )
{
    cout<<"I'am function";
    return 0;
}
int  main()
{
    print();

    return 0;
}
```

Error: return value type does not match the function type

- **syntax error**; we can't use return inside function that return nothing (its data type is void)

Example

```
#include <iostream>
using namespace std;
int sub(int x, int y )
{
    return x-y;

}
int  main()
{
    cout<<sub(5,6)<<endl;
    int z=sub(10,4);
    int x=3;
    int y=1;
    cout<<sub(x,y)<<endl;
    cout<<z;

    return 0;
}
```

```
C:\U
-1
2
6
```

- A function can actually be **called multiple times** within a program, and its argument is naturally not limited just to literals.
  1. store result of function call in variable then use cout to output this variable.
  2. call inside cout statement , The arguments passed to **sub** are numbers (value not variable).
  3. call inside cout statement , The arguments passed to **sub** are variables That is also valid, and works fine. The function is called with the values x and y have at the moment of the call: 3 and 1 respectively, returning 2 as result.
  4. use function as operand of any arithmetic operations
     cout<<sub(4,5)+sub(x,y)+4 ; (what is the output ??)

Example

```
#include <iostream>
using namespace std;
int add (int ,int );
int main()
{
    cout <<"addition       is "<< add (20,4) << '\n';
        cin.get();
    return 0;
}
int add (int a, int b=2)
    {
  int r;
  r=a+b;
  return (r);
    }
```

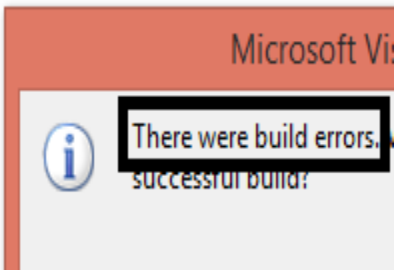
- **function prototype** :Used by the compiler to check the validity of the function call within the main program (function name, its return data type, number of arguments, their data types, and their order).
- **Function prototype:** function heading without the body of the function
- int add(intx, int y); is prototype (same as function but end with ;)
- *Function prototype is **Only needed*** if function definition comes after the function call in the program (after the main()).
- *Function prototype can used to define default value for function variable (will discuss later !!)*
- **Syntax**:

```
functionType functionName(parameter list);
```

  - Not necessary to specify the variable name in the parameter list
  - Data type of each parameter must be specified

alaa abu srhan --email: alaaa@hu.edu.jo

```
#include <iostream>
using namespace std;
int main()
{
    cout <<"addition       is "<< add (20,4) << '\n';
        cin.get();
    return 0;
}
int add (int a, int b=2)
    {
  int r;
  r=a+b;
  return (r);
    }
```



- if function definition comes after the function call in the program (after the main()) then you have to use prototype.
- error because main() come before add function so we have to use prototype and there is no prototype in this code .

Example

What is the function prototype for the following function:

1)

```
int sub(int x, int y )
{
    return x-y;
}
```

answer :

int sub(int ,int);

or

int sub(int x,int y);

or

int sub(int x1, int z1); we can use any variable

2)

```
int add(int x,int z,int y)
{  int g=x+y+z;
    return g;
}
```

answer:

int add(int ,int,int);

or

int add(int x,int z, int y);

or

int add (int x1, int y, int z); we can use any variable

3)

```
float mult (float x,  float z)
{
    return x*z;
}
```

answer:

float mult(float ,float);

or

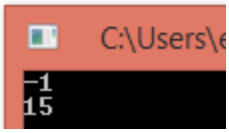
float mult(float x, float z);

or

float mult(float x1, float z1); we can use any variable

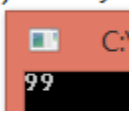
Example

```
#include <iostream>
using namespace std;
int sub(int x, int y )
{
    return x-y;
}
int add(int x,int z,int y)
{  int g=x+y+z;
    return g;
}
int  main()
{
    cout<<sub(5,6)<<endl;
    cout<<add(4,5,6)<<endl;

    return 0;
}
```



Example

```
#include <iostream>
using namespace std;
int fun(int x, int y )
{
    return 'c';
}

int  main()
{
    cout<<fun(5,6)<<endl;

    return 0;
}
```



- remember that **fun** return data type is **int.**

Example

```
#include <iostream>
using namespace std;
void first(int);
int end(int);
void  main()
{
    first(8);
}
void first (int x)
{
    if (x<10) x++;
    cout<<end(x)<<endl;
}
int end(int y )
{
    switch(y) {
        case 9: ++y;
            case 8:  ++y; }
        return y ;
}
```

```
11
```

main | first | end

first(8) --> call function first

and send 8 to this function

x

8

x< 10 ?? yes

9

cout<<end(x)<<endl;

send value of x (9) to end and wait until end return something and then print it

y

9 switch(x)-->switch(9)

10 from case 9

11 from case 8 (no break)

return y --> return 11

cout<<end(x)<<endl;

11

Example

```
#include <iostream>
using namespace std;
int  test (int first, int end )
{
    first=first+2;
    end=first+end;
    return end;
}
void  main()
{
    int x=5, y=9;
    cout<<test(x,y)<<endl;

}
```

```
C:\Use
16
```

main | test

x y

5 9

cout<<test(x,y)<<endl;

test(5,9) first end

pass 5 to first 5 9

and 9 to end 7

first =first+2

16 end=first+end=9+7

return end --> return 16

cout<<test(x,y)<<endl;

16

Example

```
#include <iostream>
using namespace std;
void  While (int x, int y )
{
    x++;
    cout<<x+y;
}
void  main()
{
    int x=5;
While(x,x);

}
```

```
C:\Users\eng.ala
11
```

main | While

x

5

While(x,x)

While(5,5) x y

5 5

6

cout<<x+y; --> 11

- **we can pass the same variable more than one time**
- **Do not use any of C++ keywords(while ,do, for ,...) as function name**
- **While is not C++ keywords While =! while**

Example

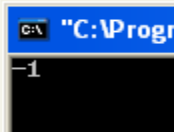
```
#include <iostream>
using namespace std;
void  while (int x, int y )
{
    x+
    cout<<x+y;
}
void  main()
{
    int x=5;
while(x,x);
        cin.get();
}
```

Error: expected an identifier

- **Syntax error -- function name has the same rule as identifier.**

Example

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

void While (int x, int y)
{

cout << x+y<< endl;

}
void main()
{
int x = 5;
while(x--);
cout<<x<<endl;
}
```



- you can write function without any need to use it --> ***While*** will not be called from main !!

Example

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

char ok  ( int ff )
{
if( ff == 99)
{
   ff+=5;
 return ff+1;
}
else
   return ff;
}
void main()
{
    int x=97;
int y = x+++1;
cout << ok (x) << endl;

}
```

main ok

x y

97

99

ok(x)

ok(98) ff

98

ff==99 ?? no

return ff --> return b (return data type is char)

cout<<ok(x);

b

Example

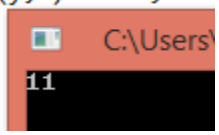
```
#include<iostream>
using namespace std;

char ok  ( int ff )
{
if( ff == 99)
   ff+=5;
 return ff+1;
else
   return ff;
}
void main()
{
    int x=97;
int y = x+++1;
cout << ok (x) << endl;
}
```

Error: expected a statement

- **syntax error**

Example

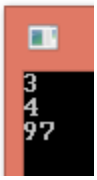
```
#include <iostream>
using namespace std;
int  x (int x, int y )
{
    return x+y;
}
void  main()
{
    int y=5;
cout<<x(y,6)<<endl;

}
```



- name of function could be the same name of one of its parameters

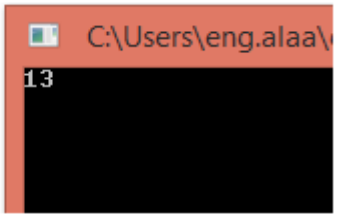
Example

```
#include <iostream>
using namespace std;
int  For (int a )
{
    for ( ;a<5;a++)
        cout<<a<<endl;
    return 'a';
}
void  main()
{
    int y=5;
cout<<For(3)<<endl;

}
```



Example

```
float Switch(int a)
{
return a++;
}

void main()
{
    int x = 13;
    switch(x%4)
    {
    case 1:
        cout << Switch(x++) << endl; break;
    case 2:
        cout << Switch(x+2) << endl; break;
    default:
        cout << Switch(x) << endl;
    }
}
```



Example

```
#include<iostream>
#include <math.h>
using namespace std;
float S (int a)
{
return a;
}
void main()
{
int x= 5;
cout<<S(++x)<<endl;
}
```


```
#include<iostream>
#include <math.h>
using namespace std;
float S (int a)
{
return a;
}
void main()
{
int x= 5;
cout<<S(x++)<<endl;
}
```


```
#include<iostream>
#include <math.h>
using namespace std;
float S (int a)
{
return ++a;
}
void main()
{
int x= 5;
cout<<S(x)<<endl;
}
```

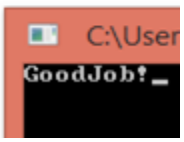

```
#include<iostream>
#include <math.h>
using namespace std;
float S (int a)
{
return a++;
}
void main()
{
int x= 5;
cout<<S(x)<<endl;
}
```

Example

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

int integer()
{
    return 10;
}
char character(){
    return '7';
}
void main(){
    int x=1;
    x = integer()-5;
    char a=character();
    cout<<a<<endl;
    if(a == '7')
        cout<<x;


}
```

Example

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

int bar(int a , int b)
{if (a>b) return b;
else return a;
}
char foo(int n)
{
    switch(n)
    {
    case 0 : return 'b';
    case 1 : return 'a';
    case 2 : return 'd';
    case 3 : return '!';
    case 4 : return 'G';
    case 5 : return 'J';
    case 6 : return 'o';
    default : return '*';
    }
}
void main()
{
    int x;
    for(x=4;x!=6;x=(x+3)%11)
        cout<<foo(bar(x,6));

}
```



Value-Returning Functions: Some Peculiarities

1)

```
int secret(int x)
{
    if (x > 5)                  //Line 1
        return 2 * x;           //Line 2
}
```

A correct definition of the function secret is:

```
int secret(int x)
{
     if (x > 5)              //Line 1
          return 2 * x;      //Line 2

     return x;               //Line 3
}
```

2)

```
return x, y;   //only the value of y will be returned
```

3)

```
int funcRet1()
{
     int x = 45;

     return 23, x;   //only the value of x is returned
}

int funcRet2(int z)
{
     int a = 2;
     int b = 3;

     return 2 * a + b, z + b; //only the value of z + b is returned
}
```

Predefined Functions

Math Library Functions

- Allow the programmer to perform common mathematical calculations
- Are used by including the header file <cmath> or <math.h>
- Functions called by writing *functionName* (*argument*)
- All math library functions **return double** values (as a result).

math functions:

- *acos(x)* inverse cosine, -1 <= x <= +1, returns value in radians in range 0 to PI
- *asin(x)* inverse sine, -1 <= x <= +1, returns value in radians in range 0 to PI
- *atan(x)* inverse tangent, returns value in radians in range -PI/2 to PI/2
- *cos(x)* returns cosine of x, x in radians
- *sin(x)* returns sine of x, x in radians
- *tan(x)* returns tangent of x, x in radians
- *exp(x)* exponential function, e to power x
- *log(x)* natural log of x (base e), x > 0
- *sqrt(x)* square root of x, x >= 0

- *fabs(x)* absolute value of x
- *floor(x)* largest integer not greater than x
- *ceil(x)* smallest integer not less than x.
- *pow(x, y)* returns x^y.
- *fmod(x, y)* computes the modulus of floating point numbers.

Function	Header File	Purpose	Parameter(s) Type	Result
abs(x)	<cmath>	Returns the absolute value of its argument: abs(-7) = 7	int (double)	int (double)
ceil(x)	<cmath>	Returns the smallest whole number that is not less than x: ceil(56.34) = 57.0	double	double
islower(x)	<cctype>	Returns 1 (true) if x is a lowercase letter; otherwise, it returns 0 (false); islower('h') is 1 (true)	int	int
isupper(x)	<cctype>	Returns 1 (true) if x is an uppercase letter; otherwise, it returns 0 (false); isupper('K') is 1 (true)	int	int
pow(x, y)	<cmath>	Returns x^y; if x is negative, y must be a whole number: pow(0.16, 0.5) = 0.4	double	double
sqrt(x)	<cmath>	Returns the nonnegative square root of x; x must be nonnegative: sqrt(4.0) = 2.0	double	double

Example 1

```
#include<iostream>
using namespace std;
#include <math.h>
void main()
{double  x=sqrt(36);
    cout<<x<<endl;
    double y= log10(1000);
    cout<<y<<endl;
    cout<<fabs(-2.3)+pow(2,2)<<endl;

    return ;}
```

```
"C:\Prog
6
3
6.3
```

- **log10(10)=1 ,log10(100)=2, log10(1000)=3**
- **pow(2,2) --> 2 to the power 2 =4**
- **sqrt(36) -->square root of 36 =6**
- **fabs(-1)=|-1|=1**

Example 2

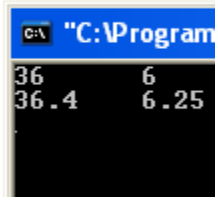
```
#include<iostream>
using namespace std;
#include <math.h>
void main()
{double  x=fmod(36.4,2);
    cout<<x<<endl;
cout<<floor(13.5)<<"\t"<<floor(13.2)<<"\t"<<floor(13)<<endl;
cout<<ceil(13.5)<<"\t"<<ceil(13.2)<<"\t"<<ceil(13)<<endl;
    return ;}
```

```
"C:\Program Files\Microsoft Visu
0.4
13      13      13
14      14      13
```


alaa abu srhan --email: alaaa@hu.edu.jo

Example 3

```
#include<iostream>
using namespace std;
#include <math.h>
void main()
{int  x=fabs(36.4);
int y=pow(2.5,2);
    cout<<x<<"\t"<<y<<endl;
cout<<fabs(36.4)<<"\t"<<pow(2.5,2)<<endl;
    return ;}
```



Example 4

```
#include<iostream>
using namespace std;
#include <math.h>
void main()
{int  x=sqrt(-25);

    cout<<x<<endl;

    return ;}
```

- **logical error**

Example 5

```
#include<iostream>
using namespace std;
#include <math.h>
void main()
{int y=ceil(-2.5);
    cout<<y<<endl;

    }
```

Example 6

```
#include<iostream>
#include <math.h>
using namespace std;

void main()
{
double i= pow(fabs(fabs(-10)),2)+pow(10,1)+ceil(0.3);
cout<<i<<endl;

}
```

Example 7

```
#include<iostream>
#include <math.h>
using namespace std;

void main()
{
double i= sqrt(25)+pow(5,floor(0.3));
cout<<i<<endl;

}
```

Example 8

```
#include <iostream>
using namespace std;
#include<math.h>
void main ()
{
        int x=2;
    switch(sizeof(atan(-1)))
    {
    case 2 :
          cout<<pow(x,2)<<endl;
                    break;
    case 4 :  cout<<x<<endl;  break;
    case 8 :
          cout<<fabs(x)<<endl; break;
    default :
          cout<<pow(x,3)<<endl;
    }
}
```

```
"C:\
2
```

Example 9

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


void main()
{
    int y = ceil(-5.4);
    switch(y)
    {
    case -5: y--; break;
    case -4: y++; break;
    }
    cout<<y;

}
```

```
C:\U
-6
```

Example 10

```
6   void main()
7   {
8       char x;
9       x = 'a';
10      cout << islower(x) << endl;
11      cout << isupper(x) << endl;
12      x = 'A';
13      cout << islower(x) << endl;
14      cout << isupper(x) << endl;
15
16      system("pause");
17  }
18
```

```
C:\Users\eng\documents\visual studio 2015\P
2
0
0
1
Press any key to continue . . .
```

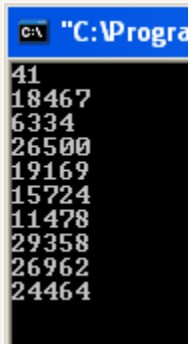
- **islower returns non zero value if x is lowercase letter.**
- **Isupper returns nonzero value if x is uppercase letter.**

Random Number

rand() function:

- is used in C++ to generate random integer numbers between 0 and a maximum specified value.
- rand() function **takes nothing** (i.e. void) as its arguments and **returns an unsigned integer.**
- In order to use this function you must Load **<cstdlib> or <stdlib.h>**
- **rand** function syntax:
  int i = rand();
- Generates a pseudorandom number between 0 and **RAND_MAX** (usually 32767)
- RAND_MAX is a symbolic constant defined in the stdlib header file.
- 0 <= rand() <= RAND_MAX.
- A pseudorandom number is a preset sequence of "random" numbers.
- The same sequence is generated upon every program execution, is this preferred?.
- This repeated behavior is essentially in programming debug and verification in simulation and other random-based applications.

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

void main ()
{
    for(int i=0;i<10;i++)
    {
    cout<<rand()<<endl;
    }
}
```



- the sequence is repeated every time

srand function:

- Jumps to a seeded location in a "random" sequence.
- Similar to rand() function, srand function is defined in the <stdlib.h> library.
- Takes an unsigned integer as a seed (i.e. as an argument).
- It does not return any value (returns void), it just change the random sequence (randomizing the rand() function).
- Can be called more than once within the same program.
- Still you need to use the rand() function to get the random numbers.
- **srand** syntax:

  srand(seed);
- seed can be any unsigned integer entered manually be the user or initialized through the program.
- If the same seed is used every time the program is run we will get the same random sequence (i.e. the same without seed).

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

void main ()
{
    srand(3);
    for(int i=0;i<10;i++)
    {
    cout<<rand()<<endl;
    }
}
```

```
48
7196
9294
9091
7031
23577
17702
23503
27217
12168
```

- The random sequence is changed , however if you run this code more than one time, the same set of random numbers will be displayed every time

So that…

41
18467
6334
26500
19169

The sequence before calling srand()

48
7196
9294
9091

The sequence after calling srand()

Thus, calling srand() will change the sequence of random numbers given that you change the seed value passed ti it, otherwise if the same seed value the same random numbers will appear over and over again.

- To initialize seed value automatically use the following syntax:

srand(time(0));

- **time(0)**
  - Returns the current calendar time in seconds.
  - time() function takes a pointer as an argument and returns unsigned integer.
- Changes the seed every time the program is run, thereby allowing **rand()** to generate random numbers. So, it is much better than manual seeding.

- Need to include the <ctime> or <time.h> library to use the time() function.

Reduces random number to a certain range:

Number = offset (shift value) + rand() % scaling_factor

Lets assume the range is [min, max], then:
min=offset .
max =scaling factor +min-1
[min, max]=(min-1,max+1) =[min,max+1]=(min-1,max]

example 1

what is the output of the following program:

```
#include <iostream>
using namespace std;
#include <cstdlib>
void main()
{
    for (int i=2 ; i<=5 ;i++)
    {
        cout<<(1 + rand() % 6);
}
    cout<<endl;
}
```

1)153426

2)1153

3)115

4)1761

offset =1
scalling factor = 6
[min, max]
min =offset =1
max =scalling factor +min-1=6+1-1=6

the program will print 4 random number [1,6] so the answer is (2)

example 2

```
#include <iostream>
using namespace std;
#include <math.h>
void main()
{
    for ( int i = 1; i <= 5; i++ )
    {
        cout << ( 1 + rand() % 6 );
    }
}
```

syntax error (where <csdlib> or <stdlib.h>??)

example 3

what is the output of the following program:

```
#include <iostream>
using namespace std;
#include <cstdlib>
void main()
{
    for (int i=5 ; i>=1 ;i--)
    {
        cout<<(1 + rand() % 4);
}
    cout<<endl;
}
```

1)24321

2)2432

3)24325

4)2435

solution:

offset =1

scalling factor = 4

[min, max]

min =offset =1

max =scalling factor +min-1=4+1-1=4

The program will print 5 random number [1,4] so the answer is (1)

example 4

what is the output of the following program:

```
#include <iostream>
using namespace std;
#include <cstdlib>
void main()
{
    int x=1;
    while(x<3)
    {
        cout<<(2 + rand() % 6);
        x++;
}
    cout<<endl;
}
```

1)78

2)743

3)77

4)783

solution:

offset =2

scalling factor = 6

[min, max]

min =offset =2
max =scalling factor +min-1=6+2-1=7
the program will print 2 random number [2,7] so the answer is (3)
example 5
what is the range that the following rand equation generate?
return 8+rand()%17;
offset =8
scalling factor = 17
[min, max]
min =offset =8
max =scalling factor +min-1=17+8-1=24
[8,24] or (7,25) or [8,25) or (7,24]

example 6
what is the range that the following rand equation generate?
```
(6+ rand()%12)*.01
```
offset =6
scalling factor = 12
[min, max]
min =offset =6
max =scalling factor +min-1=12+6-1=17
[6,17] or (5,18) or [6,18) or (5,17]
then multiply it with 0.01
[0.06,0.17] or (0.05,0.18) or [0.06,0.18) or (0.05,0.17]

example 7
what is the range that the following rand equation generate?
```
(3+ rand()%3)
```
offset =3
scalling factor = 3
[min, max]
min =offset =3
max =scalling factor +min-1=3+3-1=5
[3,5] or (2,6) or [3,6) or (2,5]

example 8
what is the range that the following rand equation generate?
```
 rand()%3
```
answer : [0, 2]

example 9
Generate random numbers in the following ranges:

- **100 <= n <= 200 → int n = 100 + rand()%101**
- **100 <= n < 500 → int n = 100 + rand()%400**

- **50 < n <= 200 → int n = 51 + rand()%150**
- **100 < n < 200 → int n = 101 + rand()%99**
- **0.01 <= n <= 0.08 →**
- **double n = (1 + rand()%8)/100 -- with step width = 0.01**

Or double n = (10+rand()%71)/1000

- **0.02 <= n <= 0.9 →**

double n = (20 + rand()%881)/1000 -- with step width = 0.001

example 9

what is the output of the following program:

```
#include <iostream>
using namespace std;
#include <cstdlib>
void main()
{
    for ( int i = 1; i <3; i++ )
    {
        cout << (3 + rand() % 20)*.01;
    }
    cin.get();
}
```

1) 3 7

2) 0.1 0.04

3) 0.4 0.5 0.6

4) 0.02 0.22

the program will print 2 random number [0.03,0.22] so the answer is (2)

Reference Variables

- Reference variable is an alias to some variable.
  - **& (ampersand)** is used to signify a reference

example 1

```
#include <iostream>
using namespace std;
void  main()
{
int x=10;
int &y=x;
cout<<x<<endl;
cout<<y<<endl;

}
```

```
10
10
```

```
x
10          y
            10
```

- Here y is an alias to the variable x since the address of y is equal to the address of x
- x and y have the same value, since the address of y is equal to the address of x.

example 2

```
#include <iostream>
using namespace std;
void  main()
{
int x=10;
int &y=x;
x=x+5;
cout<<x<<"\t"<<y<<endl;
y=y+3;
cout<<x<<"\t"<<y<<endl;

}
```

```
C:\Users\e
15      15
18      18
```

```
x
10          y
            10
15
            15
            18
18
```

- modifying either x or y both variables will have the same modified value since both of them refer to the same memory location or address.

example 3

```
#include <iostream>
using namespace std;
void  main()
{
int x=10;
int &y;
y=x;
cout<<
cout<<y<<endl;

}
```

Error: reference variable "y" requires an initializer

- Reference variables must be initialized within the same statement that defines them, if not it will be a ***syntax error*** *(reference must be initialized).*

example 4

```
#include <iostream>
using namespace std;
void  main()
{
int x=10;
int &y=1;

cout<<x
cout<<y<<endl;

}
```

Error: initial value of reference to non-const must be an lvalue

```
#include <iostream>
using namespace std;
void  main()
{
int x=10;
int &y=x+1;

cout<<x
cout<<y

}
```

int x

Error: initial value of refer

- Reference variables must be initialized with a variable only, constants and allowed → ***syntax error.***

example 5

```
#include<iostream>
using namespace std;
int main()
{   int x;
double & y=x;

}
```

int x

Error: a reference of type "double &" (not const-qualified) cannot be initialized with a value of type "int"

- **syntax error** >>>>x and &y must have the same data type

example 6

```
#include<iostream>
using namespace std;
int main()
{   int x;
int &x=x;
}
```

- **syntax error** >>>> we can't use the same name for the reference variable and what the variable refer to.

alaa abu srhan --email: alaaa@hu.edu.jo

example 7

```
#include <iostream>
using namespace std;
void  main()
{
int x=10;
int &y=x;
int z=3;
y=z;
y++;
cout<<x<<endl;
cout<<y<<endl;
cout<<z<<endl;

}
```

```
4
4
3
```

• You cannot reassign the reference variable to another variable since you simply copy the value of the new variable in the old one and you still working on the old one and this is considered as a ***logical error.***

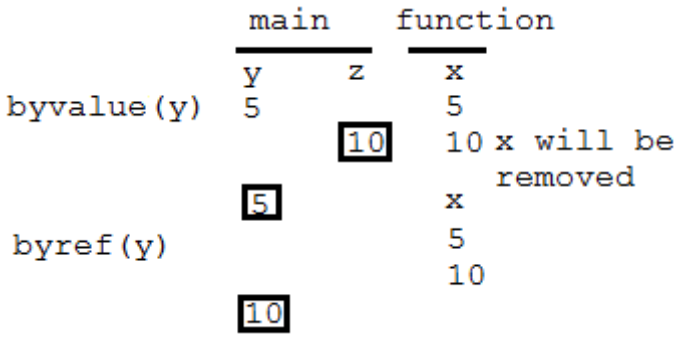
Call By Reference

- Two types of function call:
    - Call by value
        - Copy of data passed to function.
        - Changes to copy do not change the original found in the caller.
        - Used to prevent unwanted side effects.
    - Call by reference
        - Function can directly access data.
        - Changes affect the original found in the caller.
        - No copy exist (reduce overhead), however, it is dangerous since the original value is overwritten.
- Reference parameters are useful in three situations:
    - Returning more than one value
    - Changing the actual parameter
    - When passing the address would save memory space and time

Example 1

```
#include <iostream>
using namespace std;
int byvalue(int );
void byref(int & x);
void  main()
{
int y=5;
int z=byvalue(y);
cout<<"z= "<<z<<endl;
cout<< " y after call by value: "<<y<<endl;
byref(y);
cout<< " y after call by ref : "<<y<<endl;
}
int byvalue(int x)
{x*=2;
    return x;  }
void byref(int &x)
{  x*=2;   }
```

```
C:\Users\eng.alaa\documer
z= 10
 y after call by value: 5
 y after call by ref : 10
```



- Function arguments can be passed by reference.

- In both the function header and prototype you must proceed the reference variable by &.
- In the function call just type the name of the variable that you want to pass.
- Inside the function body use the reference variable by its name without &.

example 2

```
#include <iostream>
using namespace std;
void byref(int &x)
{  x*=2;   }
void  main()
{
int x=5;
byref(x);
cout<< " x after call by ref 1st time  : "<<x<<endl;
byref(x);
cout<< " x after call by ref 1st time  : "<<x<<endl;
}
```

```
C:\Users\eng.alaa\documents\visual stuc
x after call by ref 1st time  : 10
x after call by ref 1st time  : 20
```

	main	byref
	x	
byref(x)	5	x
		5
	10	10 x will be removed
byref(x)		x
		10
		20
	20	

example 3

```
#include <iostream>
using namespace std;
void  change(int &var)
{  var+=3;   }
void  main()
{
int x=5;
change(3);

cout<<
}
```

Error: initial value of reference to non-const must be an lvalue

```
#include <iostream>
using namespace std;
void  change(int &var)
{  var+=3;   }
void  main()
{
int x=5;
change(x+3);
cout<<
}
```

int x

Error: initial value of reference to non-const must be an lvalue

- the reference argument must be a variable (constants or expressions are not allowed → ***syntax error***

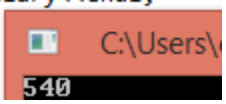
example 4

```
#include <iostream>
using namespace std;
void  change(int &var)
{  var+=3;   }
void  main()
{
int x=5;
change(x+=3);
cout<< " x=  "<<x<<endl;
}
```

```
C:\Users\e
x=  11
```

Example 5

```
#include <iostream>
using namespace std;
void  fsalary(int &sal)
{ int y=10;
sal=sal+sal*10/100-y;
}
void  main()
{
int salary=500;
fsalary(salary);
cout<<salary<<endl;
}
```



Example 6

```
#include <iostream>
using namespace std;
#include<math.h>
void ceilfun(double & y)
{y=ceil(y);
}
void  main()
{
    double x=-5.9;
    ceilfun(x);
    cout<<x<<endl;
}
```

Example 7

```
#include <iostream>
using namespace std;
void switchfun(int &num)
{
    switch(num)
    {
    case -4: num=num+10;
   break;
    case -2: num=num+=20;
          break;
    }
}

void main()
{int x= 2;
    int y = x-3*x;
    switchfun (y);
    cout << y << endl;


}
```

Write a program example

- Write a program that takes a course score (a value between 0 and 100) and determines a student's course grade. This program has three functions: main, getScore, and printGrade, as follows:
- **main**
  - Get the course score.
  - Print the course grade.
- **getScore**
  - Prompt the user for the input.
  - Get the input.
  - Print the course score.
- **printGrade**
  - Calculate the course grade.
  - Print the course grade.

```
void getScore(int& score);
void printGrade(int score);

int main()
{
    int courseScore;

    cout << "Line 1: Based on the course score, \n"
         << "    this program computes the "
         << "course grade." << endl;                    //Line 1

    getScore(courseScore);                              //Line 2

    printGrade(courseScore);                            //Line 3

    return 0;
}

void getScore(int& score)
{
    cout << "Line 4: Enter course score: ";             //Line 4
    cin >> score;                                       //Line 5
    cout << endl << "Line 6: Course score is "
         << score << endl;                              //Line 6
}

void printGrade(int cScore)
{
    cout << "Line 7: Your grade for the course is ";    //Line 7

    if (cScore >= 90)                                   //Line 8
        cout << "A." << endl;
    else if (cScore >= 80)
        cout << "B." << endl;
    else if(cScore >= 70)
        cout << "C." << endl;
    else if (cScore >= 60)
        cout << "D." << endl;
    else
        cout << "F." << endl;
}
```

Identifiers

The main attributes attached with any variable or identifier include:

1. Name, type, size, value (as taken before).
2. Storage class: Determines the period during which the variable exists in memory

storage class types

Types :	Automatic storage		Static storage		
Keywords:	auto	register	extern	static	mutable

3. Scope: Where the identifier can be referenced in program

   Scope Types:

   1. File scope:
      - Defined outside a function, known in all functions
      - Known in all functions from the point at which the identifier is declared until the end of the file
      - Examples include, global variables, function definitions and functions prototypes
   2. Function scope:
      - Can only be referenced inside a function body
   3. Block scope:
      - Declared inside a block. Begins at declaration, ends at }.
      - Includes variables, function parameters (local variables of function).
      - If two nested blocks have the same variable, outer blocks "hidden" from inner blocks.
   4. Function prototype scope:
      - Identifiers in parameter list
        Names in function prototype optional, and can be used anywhere

alaa abu srhan --email: alaaa@hu.edu.jo

Identifiers scope

- Scope of an identifier:where in the program the identifier is accessible.
- Local identifier: identifiers declared within a function (or block)
- Global identifier: identifiers declared outside of every function definition
- **C++ does not allow nested functions**
    - Definition of one function cannot be included in the body of another function
- Rules when an identifier is accessed:
    - Global identifiers are accessible by a function or block if:
        - Declared before function definition
        - Function name different from identifier
        - Parameters to the function have different names
        - All local identifiers have different names
    - Nested block
        - Identifier accessible from declaration to end of block in which it is declared
        - Within nested blocks if no identifier with same name exists
    - Scope of function name similar to scope of identifier declared outside any block
        - i.e., function name scope = global variable scope

Example 1:

```
#include <iostream>
using namespace std;
int  out (int x )
{

    return x*5;
}
void  main()
{
    int y=-5;
cout<<out(y)<<endl;

}
```

```
C:\
-25
```

- x can be accessed only by calling ***out*** function or by using **default value** (will discuss later !!)
- x is **local variable within (out) function** so it just used inside this function, z is **local variable** within main function so it just used inside main function .
- we have two **block** (anything between { } is called block).

alaa abu srhan --email: alaaa@hu.edu.jo

Example 2

outer block

inner block

```
#include <iostream>
using namespace std;
void  main()
{
    int z=-5;
    cout<<z<<endl;
    {
        int z=5;
        cout<<z<<endl;
    }
    cout<<z<<endl;
}
```

```
-5
5
-5
```

inner
z
5

used only in inner block

outer
z
-5

used in outer block, it can be used in inner block if it's not declare in it

- we have two nested blocks (inner and outer block)
- If two nested blocks have the same variable, outer blocks "hidden" from inner blocks.

Example 3

```
#include <iostream>
using namespace std;
void  main()
{
    int z=-5;
    cout<<z<<endl;
    {
        cout<<z<<endl;
    }
    cout<<z<<endl;
}
```

```
-5
-5
-5
```

Example 4

```
#include <iostream>
using namespace std;
void  main()
{
    {
        int z=-5;
        cout<<z<<endl;
    }
    cout<<z<<endl;
}
```

Error: identifier "z" is undefined

- **syntax error-->** z declare in inner block, it is only used within this block

Example 5

```
#include <iostream>
using namespace std;
int z=5;
void  main()
{
    cout<<z<<endl;
    {
        cout<<z<<endl;
    }
    int z=3;
    cout<<z<<endl;
}
```

global variable --> can be used anywhere in the code (under it)

it will print the value of global variable z

```
5
5
3
```

- If a global variable and a local variable share the same name then local value will used.

alaa abu srhan --email: alaaa@hu.edu.jo

Example 6

```
#include <iostream>
using namespace std;
int z=5;
void  main()
{
    {

        z=3;
        cout<<z<<endl;
    }
    cout<<z<<endl;
}
```

global
z
5
3

3
3

global value used

global value used

```
#include <iostream>
using namespace std;
int z=5;
void  main()
{
    {
        int z;
        z=3;
        cout<<z<<endl;
    }
    cout<<z<<endl;
}
```

gloabal
z
5

inner
z
3

local value used

3
5

global value used

Example 7

```
#include <iostream>
using namespace std;
void fun1 (int x);
void  main()
{
    fun1(5);

}
void fun1( int x1)
{   cout<<x1;
    x=5;
}
```

Error: identifier "x" is undefined

- syntax error --> x has a **prototype scope**, you can't use it from main or any function .
- function name is **file scope** you can access it anywhere in your code <u>after the prototype.</u>
- Inside any scope (function scope or block scope) you can't define two variables with the same names.

Example 8

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

void main()
{
    int x=5;
for( int i=3;i<4;i++)
{
int x=7;
    cout<<x<<endl;
}
}
```

7

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

void main()
{
    int x=5;
for( int i=3;i<4;i++)
int x=7;
    cout<<x<<endl;

}
```

5

- we have two block main block and for block, so we can define x in both blocks.(**block scope)**

Example 9

```
#include <iostream>
using namespace std;
void fun1 (int x);
void  main()
{
    fun1(5);
    cout<<x1;
}
void fun1( int x1)
{   cout<<x1;
}
```

Error: identifier "x1" is undefined

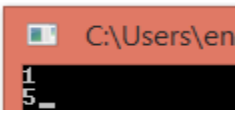
- **x1** has **function scope**, it just used in **fun1**

Example 10

```
#include <iostream>
using namespace std;
int x=1;
void fun1(void );
int main ()
{
    fun1( );
    return 0;
}
void  fun1(void)
{
     x=5;
cout<<x;

}
```

Example 11

```
#include <iostream>
using namespace std;
int x=1;
void fun1(int );
int main ()
{
    cout<<x<<endl;
    fun1(5);
    return 0;
}
void  fun1(int x)
{
cout<<x;

}
```



- Global variables are always accessible. Function **fun1** references the global **x.**

- To access a global variable declared after the definition of a function, the function must not contain any identifier with the same name
    - Reserved word **extern** indicates that a global variable has been declared elsewhere

Storage class

Storage class: Determines the period during which the variable exists in memory

storage class types

Types :	Automatic storage		Static storage	
Keywords:	**auto**	register	extern	**static**	mutable

1. Automatic storage:
    - Variables created and destroyed within its block (created when entering the block and destroyed when leaving this block).
    - Can only be used with local variables and parameters of a specified function.

        auto :Default for local variables.
2. Static Storage:
    - Variables exist from the point of declaration for entire program execution.
    - Static variables are created and initialized ***once*** when the program begins execution.
    - Numeric variables are initialized to 0 by default unless the programmer initialize them explicitly

        static:
        - Usually, used with local variables defined in functions
        - Permanent storage for the static variable is allocated, so that it is not removed from memory when leaving the block it is defined on.
        - Keep value after function ends
        - Only known in their own function (known inside its function like auto and register but it is not removed from memory when the function exit).

```
static dataType identifier;
```

Example 1

```
#include <iostream>
using namespace std;
void  main()
{
    static int a;
    a+=2;
    cout<<a<<endl;
}
```

```
2
```

- static Numeric variables are initialized to 0 by default unless the programmer initialize them explicitly.

Example 2

```
#include <iostream>
using namespace std;
void  main()
{
    register static int a;
    a+=2;
    cout<<a<<

}
```

Error: more than one storage class may not be specified

- **syntax error** -->Do not use multiple storage specifiers at the same time.

Example 3

```
#include <iostream>
using namespace std;
bool fun()
{
static int var=-1; //line 3
}

int main()
{
cout<<var;
}
```

Error: identifier "var" is undefined

- syntax error **var** is local to fun()
- var is static so it will be stored in memory from line 3 to the end of the code, but you can't print it outside fun().

Example 4

```
#include <iostream>
using namespace std;
static int var=-1; //line 1
void  fun()
{
cout<<var;// print -1
}
void main()
{
cout<<var; //print -1

}
```

C:\Users\e
-1

- var is static and global so it will be stored in memory from line 1 to the end of the code, and you can print it anywhere in your code.

Example 5

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

void fun()
{
cout<<var;
}
static int var=-1; //line 5
int main()
{
cout<<var;//print -1}
}
```

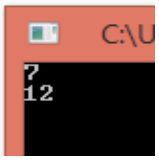
- syntax error (var inside fun() is not defined yet)
- var is static so it will be stored in memory from line 5 to the end of the code, and you can print it from line 5 to the end of the code.
- Local static variables are not destroyed when the function ends.

notes:

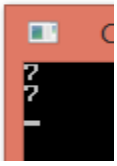
- Global variables are by default static
- Local variable are by default auto
- You can't reference (print/modify) local variable outside its scope

Example 6

```
#include <iostream>
using namespace std;
void twice()
{
    static int x=2;
    x+=5;
    cout<<x<<endl;
}
void  main()
{
    twice();
    twice();

}
```




```
#include <iostream>
using namespace std;
void twice()
{
int x=2;
    x+=5;
    cout<<x<<endl;
}
void  main()
{
    twice();
    twice();

}
```



- Permanent storage for the static variable is allocated, so that it is not removed from memory when leaving the block it is defined on.
- each time you call the function it will print last value of x +5 (not 2+5=7)--> because x is **static** , if we use int x=2 then its **auto** so every time you call the function it will print 7.

Example 7

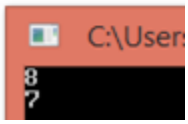
```
#include <iostream>
using namespace std;
void staticfun();
int x=5;
void main() {
int x=1;
for (int i=1;i<=3;i++)
staticfun();
cout<<x;
cout<<endl;
}
void staticfun() {
    static int test=1;
    cout<<test++<<endl;
}
```

Unary Scope Resolution Operator

- Unary scope resolution operator (::)
- If a global variable and a local variable share the same name, unary scope resolution operator is used to access the global variables.
- Not needed if names are different
- Instead of variable use ::variable
- Very important → Cannot be used to access a local variable of the same name in an outer block.

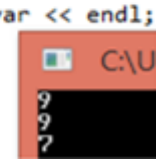
Example 1

```
using namespace std;
int var = 7;
void unary(int var)
{

    var++;
    cout << var<<endl;
}

void main()
{
    unary(7);
    cout << ::var << endl;

}
```



Example 2

```
using namespace std;
int var = 7;
void unary()
{
    int var=6;
    var=var+3;
    cout << var<<endl;
}

void main()
{  int var=6;
    unary();
    unary();
    cout << ::var << endl;

}
```

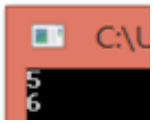



```
#include<iostream>
using namespace std;
int var =7;
void fun()
{
    var+=3;
    cout<<var<<endl;
}
void main()
{
    int var=6;
    fun();
    fun();
    cout<<::var<<endl;

}
```

- Global variables are by default static
- Local variable are by default auto

Example 3

```
using namespace std;
int z = 5;

int main()
{
int  z = 6;
cout<< ::z<<endl;
cout << z<<endl;

return 0;
}
```



Example 4

```
using namespace std
int z = 5;

int main()
{
int  z = 6;
cout<< ::z<<endl;
int z= 2;

cout << z<<endl;

return 0;
}
```

- **syntax error -->**z declare twice within the same block

Example 5

```
using namespace std;
int y = 5;

int main()
{
    int z = 5;
    if(z==5)
    {
   int z = 11;
   cout<< ::z<<endl;
    }

return 0;
}
```

Error: the global scope has no "z"

- **syntax error-->** z undefined as a global variable.

Example 6

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

void main()
{
    int z= -5;
    cout<<z<<endl;
    {
        int z=5;
        cout<<::z;
    }

}
```

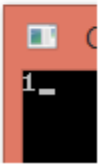
Error: the global scope has no "z"

- unary operator Cannot be used to access a local variable of the same name in an outer block.

Functions with empty parameter lists

Example 1

```
#include <iostream>
using namespace std;
void voidfun(void);
void main() {
voidfun();

}
void voidfun(void) {
    int a=1;
    cout<<a;
}
```



- Either writing void or leaving a parameter list empty indicates that the function takes no arguments.
- Function print takes no arguments and returns no value.

Example 2

```
#include <iostream>
using namespace std;
void voidfun(void);
void main() {
voidfun(1);

}
void voidfun(void) {
    int a=1;
    cout<<a;
}
```

Error: too many arguments in function call

- Passing parameter to a function that does not take any parameter is **syntax error.**

Example 3

```
#include <iostream>
using namespace std;
void voidfun(void);
void main() {
cout<<voidfun();

}
void voidfun(void) {
    int a=1;
    cout<<a;
}
```

Error: no operator "<<" matches these operands

alaa abu srhan --email: alaaa@hu.edu.jo

- Calling a function that does not return any value inside cout statement is **syntax error.**

Functions with default arguments

Example 1

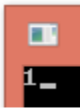
```
#include <iostream>
using namespace std;
void Default(int a);
void main() {
Default();

}
void  Default(int a) {
    cout<<a;
}
```

Error: too few arguments in function call

- **syntax error**, **Default** function has one argument, so when we call it we have to pass value to this function. We can solve the problem using default argument(see next example).

Example 2

```
#include <iostream>
using namespace std;
void Default(int a=1);
void main() {
Default();

}
void  Default(int a) {
    cout<<a;
}
```



- In the **function prototype** give all or some of the arguments **default values**
- When you call the function, you can omit one or more of the arguments values. The omitted arguments will take their values from the default values in the function prototype.

Example 3

```
#include <iostream>
using namespace std;
void Default(int );
int a=1;
void main() {
Default();

}
void  Default(int a) {
    cout<<a;
}
```


alaa abu srhan --email: alaaa@hu.edu.jo

- Set defaults in function prototype (only) where the variables names are provided just for readability

Example 4

```
#include <iostream>
using namespace std;
void Default(int a ,int b=1);
int a=1;
void main() {
Default();

}
void  Default( int a ,int b) {
    cout<<a;
}
```

Error: too few arguments in function call

- In a function call you can omit the parameters that have default values only.
- **syntax error** because a does not have value in the prototype

Example 5

```
#include <iostream>
using namespace std;
void Default(int a=1 ,int b);
int a
void
Defau

}
void  Default( int a ,int b) {
    cout<<a;
}
```

void Default(int a = 1, int b)

Error: default argument not at end of parameter list

```
#include <iostream>
using namespace std;
void Default(int a=1 ,int b, int c=9);
int a


void main() {
Default(1,2,3);

}
void  Default( int a ,int b,int c) {
    cout<<a;
    cout<<b;
    cout<<c;
}
```

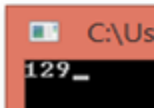
Error: default argument not at end of parameter list

- Not setting all the rightmost parameters after a default arguments to default is a ***syntax error***.
- This means that no argument can take a default value unless the one on its right has a default value.

alaa abu srhan --email: alaaa@hu.edu.jo

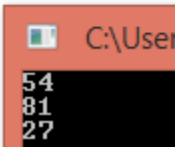
Example 6

```
#include <iostream>
using namespace std;
void Default(int a=1 ,int b=2, int c=9);
int a=1;
void main() {
Default();

}
void  Default( int a ,int b,int c) {
     cout<<a;
     cout<<b;
     cout<<c;
}
```



Example 7

```
#include <iostream>
using namespace std;
void Default(int a=1 ,int b=2, int c=9);
int a=1;
void main() {
Default(3);
Default(3,3);
Default(3,3,3);

}
void  Default( int a ,int b,int c) {
    cout<<a*b*c<<endl;

}
```



Functions overloading

- Function overloading means having functions with same name and different parameters **(different number of parameters, or different data types, or different order, or all of these issues at the same time, *different name of arguments is not function overloading***)

```
#include<iostream>
using namespace std;
int fun1(int ,float ,char
        );
int fun1(int , float,
        char);
int main()
{
        cout<<fun1(5,3.
        3,'a');
return 0;
}
int fun1(int x1,float
        y1,char z1)
```

Syntax error. Two functions are the same

```
#include<iostream>
using namespace
        std;
int fun1(int ,float
        ,char );
float fun1(int , float,
        char);
int main()
{
        cout<<fun1(5
        ,3.3,'a');
return 0;
}
int fun1(int x1,float
```

Syntax error. changing return type is not enough

```
#include<iostream>
using namespace
        std;
int fun1(int ,float
        ,char );
int fun1(float, int ,
        char);
int main()
{
        cout<<fun1(5
        ,3.3,'a');
return 0;
}
int fun1(int x1,float
```

Parameters order is different so it is ok

```
#include<iostream>
using namespace
        std;
int fun1(int ,float
        ,char );
int fun1(int , float);
int main()
{
        cout<<fun1(5
        ,3.3,'a');
return 0;
}
int fun1(int x1,float
        y1,char z1)
```

# of parameters is different so it is ok

- Overloading a function with another version that have the same parameters numbers, types, and order with just the return result data type is different is a ***syntax error***.
- The parameter list supplied in a call to an overloaded function determines which function is executed
- Two functions are said to have different formal parameter lists if both functions have either:
  - A different number of formal parameters
  - If the number of formal parameters is the same, but the data type of the formal parameters differs in at least one position

Example 1

```
#include <iostream>
using namespace std;
int  fun1(int x ,int y , int z);
double fun1(double x ,double y , double z);
void main()
{
    cout<<fun1(2,3,1)<<endl;
    cout<<fun1(2.5,3.8,1.4)<<endl;

}
int  fun1(int x ,int y , int z)
{
    return x*y*z;
}
double fun1(double x ,double y , double z)
{
    return x+y+z;
}
```

```
6
7.7
```

```
#include <iostream>
using namespace std;
int  fun1(int x ,int y , int z);
int fun1(double x ,double y , double z);
void main()
{
    cout<<fun1(2,3,1)<<endl;
    cout<<fun1(2.5,3.8,1.4)<<endl;

}
int  fun1(int x1 ,int y1 , int z1)
{
    return x1*y1*z1;
}
int fun1(double x ,double y , double z)
{
    return x+y+z;
}
```

```
C:\User
6
7
```

- Program chooses function by signature: signature is determined by function name and parameter types.
- Can have the same return types.

Example 2

```
#include <iostream>
using namespace std;
int fun1(int x1=1 ,int y1=2,int y3=4);
int fun1( );
void  main()
{
    cout<<fun1(2,3,1)<<endl;
    cout<<fun1()<<endl;
}
int fun1(int x1,int y1,int z1)
{
return x1*y1*z1;
}
int fun1( )
{
    int r =1;
return r;
}
```

```
fun1

Error: more than one instance of overloaded function "fun1" matches the argument list:
```

- You cannot overload a function with default arguments with another version that takes no arguments → ***syntax error.***

Example 3

```
int fun1(int x1=1 ,int y1=2,int y3=4);
int fun1(double x, double y );
double fun1(int x, double y );

void  main()
{
     cout<<fun1(2,3,1)<<endl;
     int x=1;
     double z=9;
     cout<<fun1(z,z)<<endl;
     cout<<fun1(x,z)<<endl;

}
int fun1(int x1,int y1,int z1)
{return x1*y1*z1;}
int fun1(double x,double y )
{return x*y;}
double fun1(int x,double y )
{return x+y;}
```

```
6
81
10
```

Example 4

```
#include <iostream>
using namespace std;
int fun1(double x, double y );
double fun1(int x, double y );

void  main()
{

     int x=1;
     double z=9;
     cout<<fun1(z,z)<<endl;
     cout<<fun1(x)<<endl;

}
int fun1(double x,double y )
{return x*y;}
double fun1(int x,double y )
{return x+y;}
```

Error: too few arguments in function call

example 5

```
#include <iostream>
using namespace std;
int  fun1(int x1=1 ,int y=2 , int z=3);

void main()
{
    cout<<fun1( )<<endl;
}
int  fun1(int x1 ,int y1 , int z1)
{
    return x1*y1*z1;
}
```

```
6
```

- name of parameters in prototype is not necessary to be the same as name of parameters on the corresponding function
- x1 will take its default value from the first variable on prototype whatever its name,
- y1 will take its default value from the first variable on prototype (y).

Example 6

```
#include <iostream>
using namespace std;
int  fun1(int x1,int y1 , int z1);
int fun1 (int x ,int  y , int z);
void main()
{
    cout<<fun1(1,2,3)<<endl;
}
int  fun1(int x1 ,int y1 , int z1)
{
    return x1*y1*z1;
}
int  fun1 (int  x ,int y , int z)
{
    return x*y+z;
}
```

- **syntax error**
- change the name of parameters is not function overloading. remember ---> different parameters (different number of parameters, or different data types, or different order, or all of these issues at the same time) is ok.

Example 7

```
#include <iostream>
using namespace std;
int  fun1 (int x1=7,int y1=5 , int z1=6);
int fun1 (double x=8,int  y=4 , int z=9);
void main()
{
    cout<<fun1( )<<endl;

    }
int  fun1(int x1 ,int y1 , int z1)
{
    return x1*y1*z1;
}
int  fun1 (double  x ,int y , int z)
{
    return x*y+z;
}
```

fun1

Error: more than one instance of overloaded function "fun1" matches the argument list:

- syntax error >>>> which one will be used when we call the function

Example 8

```
#include <iostream>
using namespace std;
int  fun1 (int x1=7,int y1=5 , int z1=6);
double fun1 (double x,int  y, int z);
void main()
{
    cout<<fun1( 1.2,1.5,1.6)<<endl;

}
int  fun1(int x1 ,int y1 , int z1)
{
    return x1*y1*z1;
}
double  fun1 (double  x ,int y , int z)
{
    return x+y+z;
}
```

```
C:\Users\eng.ala
3.2
```

- all the parameters passed to function are double so the suitable function is fun(double ,int ,int)

Example 9

```
#include <iostream>
using namespace std;
int  fun1 (float x1=7,int y1=5 , int z1=6);
double fun1 (double x,int  y, int z);
void main()
{
    cout<<fun1( 1.2,1.5,1.6)<<endl;
cin.get();
}
int  fun1(float x1 ,int y1 , int z1)
{
    return x1*y1*z1;
}
double  fun1 (double  x ,int y , int z)
{
    return x+y+z;
}
```

```
C:\Users\
3.2
```

- **1.2 is double**

alaa abu srhan --email: alaaa@hu.edu.jo

Example 10

```
#include <iostream>
using namespace std;
int  fun1 (float x1=7,int y1=5 , int z1=6);
double fun1 (double x,int  y, int z);
void main()
{
    cout<<fun1( 1,1,1)<<endl;
}
int  fun1(float x1 ,int y1 , int z1)
{
    return x1*y1*z1;
}
double  fun1 (double  x ,int y , int z)
{
    return x+y+z;
}
```

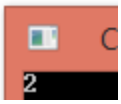
fun1

Error: more than one instance of overloaded function "fun1" matches the argument list:

- ambiguous call to overloaded function ; 1 is integer so which function we will use !!!

Example 11

```
#include <iostream>
using namespace std;
int fun(int x=3,int y=4);
int fun(int ,int );
int fun(int x,int y)
{
    return x;
}
int main()

{
    cout<<fun(2,5);

return 0;

}
```



Passing Arrays to Functions

Example 1

```
#include <iostream>
using namespace std;
void Array( int [], int arraySize );
void main()
{
    int b[10];

}
void  Array(int b[] ,int arraySize)
{

}
```

```
#include <iostream>
using namespace std;
void Array( int [], int  );
void main()
{
    int b[10];

}
void  Array(int b[] ,int arraySize)
{

}
```

- Function prototype:

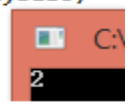
void Array(int b[], int arraySize);

- Parameter names optional in prototype
  - int b[] could be simply int []
  - int arraysize could be simply int.

- If the size of the array is passed as, e.g. int b[5], the compiler will ignore it.

Example 2

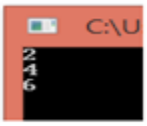
```
#include <iostream>
using namespace std;
void Array( int [], int  );
void main()
{
    int a[3]={1,2,3};
 Array(a,3);

}
void  Array(int b[] ,int arraySize)
{
    cout<<b[1];
}
```



- Function Call:
  - Specify the name without any brackets
  - Array size is usually passed to the function to allow correct processing of the array elements.

Example 3

```
#include <iostream>
using namespace std;
void Array( int [], int  );
void main()
{
    int a[3]={1,2,3};
 Array(a,3);
 for (int i = 0 ; i<3 ;i++)
     cout<<a[i]<<endl;

}
void  Array(int b[] ,int arraySize)
{
    for (int i = 0 ; i<arraySize ;i++)
    b[i]=b[i]*2;


}
```



- Arrays are passed as call-by-reference by default

Example 4

```
#include <iostream>
using namespace std;
void Array ( int ,int &  );
void main()
{
    int a[3]={1,2,3};
 Array(a[1], a[2] );
 for (int i = 0 ; i<3 ;i++)
     cout<<a[i]<<endl;

}
void  Array(int b,int & c)
{   b=5;
c=c*2;
}
```

```
1
2
6
```

- Individual array elements are passed by call-by-value

Example 5

```
#include <iostream>
using namespace std;
void Array ( const int  [],int );
void main()
{
    int a[3]={1,2,3};
 Array(a,3 );
 for (int i = 0 ; i<3 ;i++)
     cout<<a[i]<<endl;
cin.get();
}
void  Array(const int b[],int size)
{   b[1]+=1;
}
```

```
const int *b

Error: expression must be a modifiable lvalue
```

```
#include <iostream>
using namespace std;
void Array ( const int  [],int );
void main()
{
    int a[3]={1,2,3};
 Array(a,3 );
 for (int i = 0 ; i<3 ;i++)
     cout<<a[i]<<endl;

}
void  Array(const int b[],int size)
{   cout<<b[1]<<endl;
}
```

```
2
1
2
3
```

- To prevent a function from modifying an array declare the parameter array within both the function definition and the function prototype as a const (i.e. pass it as read-only variable).
- so, any modification will be reported as a *syntax error*.