المحاضره الأولى

History of C++

- C++, as the name implies, is essentially based on the C programming language.
- The C programming language was devised in the early 1970s at Bell Laboratories by Dennis Ritchie.
- It was designed as a system implementation language for the Unix operating system.
- The history of C and Unix are closely intertwined

- For this reason a lot of Unix programming is done with C.
- To some extent, C was originally based on the typeless language BCPL; however it grew well beyond that.

C++ Fundamentals

- How A computer thinks?

A computer thinks in 1's and 0's. Various switches are either on (1's), or off (0's).

- How a human thinks?

A *programming language* is a language that provides a bridge between the computer and human beings.

- A “**low-level” language** is one that is closer to a computer’s thinking than to a human language.
- Example Assembly language.

A “**high-level”language** is one that is closer to human language

High Level

Cobol, BASIC, Visual Basic, Java, etc.

C and C++

Assembly language

Click To expand

Machine code (1's and 0's)

Low Level

- **That data might be information about**

1. employees,
2. parts to a mathematical computation,
3. scientific data,
4. or even the elements of a game.

No matter what programming language or techniques you use,

What is the goals of programming

1. store,
2. manipulate,
3. and retrieve data.

Why we need to used variables ?

Data must be temporarily stored in the program, in order to be manipulated. This is accomplished via variables

What is the variables ?

A variable is simply a place in memory set aside to hold data of a particular type.

It is a specific section of the computer's memory that has been reserved to hold some data.

- **Why it is called variable ?**
- It is called a variable because its value or content can vary.
- When you create a variable, you are actually setting aside a small piece of memory for storage purposes.
- The name you give the variable is actually a label for that address in memory.

- example, you might declare a variable in the following manner.
- **int j;**
- Then, you have **just allocated** four bytes of memory;
- you are using the variable j to refer to those four bytes of memory.
- You are also stating that the only type of data that j will hold, is whole numbers.
- (The int is a data type that refers to integers, or whole numbers.)
- Now, whenever you reference j in your code, you are actually referencing the contents being stored at a specific address in memory.

Getting Ready to Program

- Programs are written to instruct machines to carry out specific tasks or to solve specific problems.
- *Algorithm ?*
- Is a step-by-step procedure that accomplishes a desired task.
- Thus, programming is the activity of communicating algorithms to computers.
- The programming process is analogous, except that machines have no tolerance for ambiguity and must have all steps specified in a precise language and in tedious detail.

The Programming Process

- 1. Specify the task.
- 2. Discover an algorithm for its solution.
- 3. Code the algorithm in C++.
- 4. Test the code

- A computer is a digital electronic machine composed of three main components:

1. **processor,**
2. **memory,**
3. **input/output devices.**

- The processor is also called the *central processing unit*, or *CPU*.
- The processor carries out instructions that are stored in the memory.
- Along with the instructions, data also is stored in memory.
- The processor typically is instructed to manipulate the data in some desired fashion

- . Input/output devices take information from agents external to the machine and provide information to those agents.

- **. Input devices are typically**

1. terminal keyboards,
2. disk drives,
3. and tape drives.

Output devices are typically

1. terminal screens,
2. printers,
3. disk drives,
4. and tape drives.

The physical makeup of a machine can be quite complicated, but the user need not be concerned with the details.

- When a simple program is compiled , three separate actions occur:

First the preprocessor is invoked, then the compiler, and finally the linker.

- The preprocessor modifies a copy of the source code by including other files and by making other changes.
- The compiler translates this into *object code*, which the linker then uses to produce the final executable file.
- A file that contains object code is called an *object file*.
- Object files, unlike source files, usually are not read by humans.

- When we speak of compiling a program, we really mean invoking the preprocessor, the compiler, and the linker.
- For a simple program, this is all done with a single command.
- After the programmer writes a program, it has to be compiled and tested.
- If modifications are needed, the source code has to be edited again.
- Thus, part of the programming process consists of this cycle:
- When the programmer is satisfied with the program performance, the cycle

edit
compile
execute

A First Program

- A first task for anyone learning to program is to print on the screen.
- Let us begin by writing the traditional first C++ program which prints the phrase *Hello, world!* On the screen. The complete program is

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

- **// my first program in C++**

- **#include <iostream>**
- Lines beginning with a hash sign (#) are directives for the preprocessor.
- They are not regular code lines with expressions but indications for the compiler's preprocessor.
- This specific file (iostream) includes the declarations of the basic standard input-output library in C++

int main ()

- It does not matter whether there are other functions with other names defined before or after it
- The instructions contained within this function's definition will always be the first ones to be executed in any C++ program.

cout<< "Hello World";

- cout is declared in the iostream standard file within the std namespace
- so that's why we needed to include that specific file and to declare that we were going to use this specific **namespace** earlier in our code.
- Notice that the statement ends with a semicolon character (;).

- **return 0;**
- The return statement causes the main function to finish.
- Return may be followed by a return code (in our example is followed by the return code 0).

I/P

```
// my second program in C++
#include <iostream>
using namespace std;
int main ()
{
  cout << "Hello World! ";
  cout << "I'm a C++ program";
  return 0;
}
```

Hello World! I'm a C++ program

- **Ex/write a program to print the following message on the screen .**
- C++ is a programming language
- C++,based on the C programming

```
#include <iostream.h>
Void main ()
{
cout << " C++ is a programming language\n";
cout << " C++ based on the C programming ";
}
```

- O/P
- C++ is a programming language
- C++ based on the C programming

- \n
- This character means newline on the screen.

- #include < iostream.h>
- main ()
- }
- cout<<“smoking is dangerous \n”;
- return0;
- {

- #include <iostream.h>
- main ()
- }
- cout<<“ smoking is dangerous”<<endl;
- return0;
- {

- #include <
  iostream.h>
- Int main ()
- {
- cout<<“matrix”;
- cout<<“matrix \n”;
- cout<<“matrix \n\n”;
- cout<<“matrix \n\n\n
  ”;
- cout<<“matrix”;
- return0;

- matrixmatrix
- matrix
-
- Matrix
-
-
- Matrix

المحاضره الثانيه

What is an algorithm?

- In computer programming, an algorithm is a set of well-defined instructions to solve a particular problem.
- It takes a set of input and produces a desired output.

Step 1: Start

Step 2: Define the variables

Step 3: Read values of variables

Step 4: Process

- math operations
- logic operations
- comparisons

Step 5: Display result

Step 6: End

1. بداية
2. تعريف المتغيرات
3. قراءة قيمة كل متغير
4. معالجة
   - عمليات رياضية
   - عمليات منطقية
   - مقارنات
5. عرض/طباعة الناتج
6. نهاية

Algorithm to add two numbers

- We need to first write the steps (sequence of actins) that lead to performing the task.
- For example, an algorithm to add two numbers:
  - Take two number inputs
  - Add numbers using the + operator
  - Display the result

Step #	Description
Step1	Start
Step2	Declare or define variables num1, num2, sum;
Step3	Read values num1 and num2
Step4	Add num1 and num2 and assign the results to sum sum ← num1 + num2;
Step5	Print sum
Step6	End

Flowchart Symbols

- A Flowchart is a graphical representation that shows the behavior (workflow) of an algorithm.

Flowcharts use standard shapes including the following:

Symbol	Name	Description
	Start / End	An oval represents a ***start*** or ***end*** point
	Arrows	A line is a connector that shows flow direction between the representative shapes
	Input /Output	A parallelogram represents ***input*** or ***output***
	Process	A rectangular represents a ***process*** **(calculation)**
No ? Yes	Decision	A diamond indicates a ***decision*** **(comparison)**
	Connector	A circle is used to combine one part of the flowchart with another part

Adding two numbers

✓ Write the pseudo-code for algorithm to sum two numbers. Then draw the equivalent flowchart.

➢ Flowcharts use standard shapes including the following:

Ans:

Pseudo-code Algorithm

```
Step 1: START
Step 2: DECLARE variables num1, num2, sum;
Step 3: READ variables num1, num2;
Step 4: CALCULATE sum
        sum = num1 + num2;
Step 5: PRINT sum
Step 6: END
```

Flowchart

Note:

- The ***pseudo-code*** and ***flowchart*** of an algorithm is not written for a particular programming language.
- They can be used to plan a solution before coding it.

Print if the condition met

- Draw the Flowchart to print the square of a number if the number is less than 5.

Note:

- The lines and arrows in it show the sequence of steps and the relationship among them.

Difference between pseudo-code and flowchart

Main differences between Pseudo-code and Flowchart

- The pseudo-code is a high-level description of an algorithm while the flowchart is a graphical representation of an algorithm.
- An algorithm is a set of instructions for solving a problem or accomplishing a task.
- Every computerized device uses algorithms, which cut the time required to do things manually.

- There are three main ways or scenarios used in the programming. These are:

1) ***Sequencing*** – putting things in the right order
2) ***Conditions*** – performing different things depending on some rules
3) ***Iteration (Looping)*** – repeating operations

Sequencing

- Calculate the average of three input numbers.

Pseudo-code Algorithm

Step 1: Start
Step 2: Declare variables Ave;
Step 3: Read variables num1, num2, num3;
Step 4: Calculate Ave
Ave = (num1 + num2 + num3)/3;
Step 5: Print Ave
Step 6: End

Flowchart

Conditions

- Conditions are very important to understand the flow-control of the code being executed.
- Conditions are used to make a decision (choose between two or more alternatives based on the condition)

  - For example, conditions become very essential to use when facing different directions, so based on the condition should choose the thing to be executed.

Use the *if* statement

```
If (condition) then
  run this statement;
Else
run the other statement;
```

Conditions

1 *if .... then*

```
If (condition) then
 statement(s)

End if
```

2 *if .... then .... else*

```
If (condition) then
 statement 1
Else
 statement 2

End if
```

Nested if structure

3 ***Nested structure of if .... then***

```
If (condition 1) then
 statement 1

If (condition 2) then
 statement 2

If (condition 3) then
 statement 3

Else
  statement 4

End if
End if
End if
```

if .... then .... elseif

```
If (condition 1) then
 statement 1

Elseif (condition 2) then
 statement 2

Elseif (condition 3) then
 statement 3

Else
  statement 4

End if
```

if … then … elseif – Flowchart

- *if .... then .... elseif* Flowchart

Condition – Example

✓ Write the pseudo-code to check if a student has passed the exam or failed. Then draw the equivalent flowchart?

Ans:

Step 1: Start
Step 2: Declare Result
Step 3: Read the student's grade
Step 4: *If* (Grade >= 50)
Result = "passed"
else
Result = "failed"
Step 5: Print result
Step 6: End

Classwork #1 – Student's Grade

✓ Send a message to a student depending on the obtained score?

Ans:

Step 1: Start
Step 2: Declare Result, message
Step 3: Read the student's grade
Step 4: *If* (Grade >= 50)
    Result = "pass"
    message = "Well Done!"
  else
    Result = "fail"
    message = "See me!"
Step 5: Print result
Step 6: End

Draw the equivalent flowchart

Flowchart

Multiple Conditions – Example

✓ Is it possible to have multiple conditions in an algorithm?

❖ **For example, print the result of a student according to the conditions listed in the table below?**

Grade value	Printed result	الـتـقـديـر
Grade < 50	Result = “Failed”	ر ا سب
Grade < 60	Result = “Passed”	مـقـبـو ل
Grade < 70	Result = “Medium”	مـتـو سط
Grade < 80	Result = “Good”	جـيـد
Grade < 90	Result = “Very good”	جـيـد جـد أ
Grade > = 90	Result = “Excellent”	ا مـتـيـا ز

Multiple Conditions – Example solution

Ans:

Step 1: Start

Step 2: Declare Grade, Result

Step 3: Read the student's grade

Step 4: *If* (Grade < 50)

Result = "failed"

else if (Grade < 60)

Result = "passed"

else if (Grade < 70)

Result = "medium"

else if (Grade < 80)

Result = "good"

else if (Grade < 90)

Result = "very good"

else

Result = "excellent"

Step 5: Print result

Step 6: End

تكملة الحل

Classwork #2 – Ticket price

✓ Write the pseudo-code algorithm to display the price of a cinema ticket as follow: (age < 12) then (price = 5K IQD), (age > 65) then (price = 7K IQD); otherwise (price = 10K IQD)?

Ans:

Step 1: Start
Step 2: Declare price
Step 3: Read the customer's Age
Step 4: *If* (Age < 12)
    price = 5K
  Else if (Age > 65)
    price = 7K
  else
    price = 10K
Step 5: Display price
Step 6: End

Draw the equivalent flowchart

Flowchart

Iteration –> Repetition

- Let's say we need to print all the numbers from 1 to 1000.
- Would we do this by sequencing?
- It is much better to do tasks like that using iteration

Repeat the operation or sequence of operations until the condition is met

Sequencing

Step 1: Start

Step 2: display 1

Step 3: display 2

Step 4: display 3

.

.

.

Step 1001: display 1000

Step 1002: END

Iteration

Step 1: Start

Step 2: Declare number =1

Step 3: while the number <= 1000

{

display the number

number = number +1 (increment the number)

}

Step 4: End

go to step 3 if count <= 1000

Loop

Iteration – printing the numbers from 1 to 1000

- Flowchart
  - print the numbers from 1 to 1000

Iteration – Summing five numbers

✓ Write the pseudo-code for algorithm to sum five numbers. Then draw the equivalent flowchart?

Ans:

Step 1: Start
Step 2: Initialization
Initialize: sum = 0 and count = 0
Step 3: Enter n numbers
Step 4: Calculate sum and increment count
sum = sum + n
count = count +1
Step 5: *while* (count < 5)
go to step 3
Step 6: Print sum
Step 7: End

Average of ten numbers – Example

✏️ Write a pseudo-code algorithm to find the sum and average of ten numbers entered by the user. Then draw the equivalent flowchart.

Ans:

Algorithm

Step 1: Start

Step 2: Initialization

Initialize: sum = 0, Ave = 0, and count = 0

Step 3: while count < **10**

{

enter a ***number***

sum = sum + ***number***

count = count +1

}

go to step 3 if count < 10

Step 4: *Ave = sum/count*

Step 6: Print sum

Print Ave

Step 7: End

Flowchart

Qualities of Good Algorithms

1. ***Precision*** – الدقة the steps of input and output should be precisely defined.
2. ***Clarity*** – الوضوح each step in the algorithm should be clear and unambiguous.
3. ***Inputs/Outputs*** – المدخلات والمخرجات has inputs that lead to the production of the output
4. ***Effectiveness*** – الفعالية Algorithms should solve problems in effective ways such that getting an
5. output from each step based on its inputs and outputs of the previous step
6. ***Generality*** – العمومية/ الشمول the algorithm should be written in such a way that it can be used in
7. different programming languages.

THANK YOU

المحاضره الثالثة

Keywords and Identifier

- Keywords are predefined, reserved words used in programming that have special meanings to the compiler.
- Keywords are part of the syntax and they cannot be used as an identifier.
- For example:

```
int money;
```

- Here, int is a keyword that indicates money is a variable of type int (integer).
- Since C++ is a **<u>case sensitive language</u>**, all keywords must be written in lowercase.

A ≠ a
X ≠ x
Case ≠ case

Note:
case-sensitive for both variable names and function names

Keywords and Identifier

❖ The table below lists some of the keywords allowed in ANSI C.

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

Note:

As can be seen in the table above, all the commands (keywords) should be written in lowercase (small letters).

Keywords and Identifier

- Identifier refers to name given to entities such as variables, functions, structures etc.
- Identifiers must be unique. They are created to give a unique name to an entity (variable,const, etc.) to identify it during the execution of the program.
- For example:

```
int money;
double accountBalance;
```

  - Here, money and accountBalance are identifiers.
- Always remember that identifier names must be different from keywords.
  - You ***cannot*** use `int` as an identifier because `int` is a keyword.

Case Sensitivity

- C ++ language is a case-sensitive.
- It differs whether an identifier, such as a variable name, is uppercase or lowercase

Examples:

- area
- Area
- AREA
- ArEa

all are seen different variables by the compiler.

Rules for naming Identifiers

- A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
- Sperate “words” within a variable name using underscores and mixed upper and lower case.

Examples:

- ✓ surfaceArea
- ✓ surface_Area
- ✓ surface_area

- The first letter of a variable name must be either a letter or an underscore, i.e., variable names cannot begin with a number.
- You cannot use any reserved word (keywords) like int, char etc. as a variable name.

Data types in C++ Programming

- In C++ programming, data types are declarations for variables. This determines the type and size of data associated with variables.
- The following table contains some commonly used data types in C++ programming.

Table: Basic types of data in C programming

Type	Size (bytes)	Format Specifier
int	at least 2, usually 4	%d
char	1	%c
float	4	%f
double	8	%lf
short int	2 usually	%hd
unsigned int	at least 2, usually 4	%u
long int	at least 4, usually 8	%ld

Declaring Variables

- Before you use a variable, you should first declare it, so the compiler can know it.
- Declaring a variable means specifying the data type of the variable

***Example** of variable declarations*

```
int myVar;
```

- myVar is a variable of `int` (integer) type.
- the size of `int` is 4 bytes.

Variables

- In programming, a variable is a container (storage area) to hold data.
- To indicate the storage area, each variable should be given a unique name (identifier).
- Variable names are just the symbolic representation of a memory location.

- For example:

```
int playerScore = 95;
```

  - In the above definition, playScore is a variable of `int` type.
  - The variable is assigned to an integer value 95
  - As the name indicates, the value of a variable can be changed through the program.

```
char ch = 'a';
// in somewhere in the code, you write
ch = '1';
```

Rules for naming Variables

Note:

- You should always try to give meaningful names to variables.
- For example:
  `firstName` is a better variable name than `fn`.

- C++ is a strongly typed language. This means that the *variable* ***type*** **cannot** be changed once it is declared.
- For example:

```
int  number = 5; // integer variable
number = 5.5; // error
double  number ; // error
```

- In the above, we define the type of number as `int`.
- Thus, we *cannot* assign a *floating-point* (decimal (عد عشري value 5.5 to an `int` variable.
- Additionally, we cannot redefine the same variable with different data type (i.e., double).
- A *floating-point* variable should be declared (defined) either as `double` or `float` to store its decimal value in C++

Constants

- If you want to define a variable whose value cannot be changed, you can use the const keyword.
- This will create a constant, for example:

```
const double PI = 3.14;
```

Note:

- the keyword `const` have added
- Also, PI is a symbolic constant; its value *cannot be changed*.

```
const double PI = 3.14 ; // define PI as a constant with value 3.14
```

- Then somewhere in your code you write the blow

```
PI = 2.9; // error
```

Example of Declarations and Assignments

- When you declare a variable with a data type, a space inside the memory is set (reserved) by a unique address to hold the value of the defined variable.

- The address of the space is associated with the variable name

Example:

```
1  #include <stdio.h>
2
3  int main() {
4  int inches, feet, fathoms;
5  fathoms = 7;
6  feet = 6 * fathoms;
7  inches = 12 * feet;
```

C output – printf

- In C++ programming, printf() is one of the main output function.
- The function sends formatted output to the screen.
- For example

```
#include <stdio.h>
int main()
{
    //display the string inside quotations
    printf("C++ Programming");
    return 0;
}
```

Output

```
C++ Programming
```

Integer Output – printf

❖ The following C++ program outputs an integer number on the screen.

```
#include <stdio.h>
int main()
{
    int testInteger = 5;
    printf("Number = %d", testInteger);
    return 0;
}
```

Output

```
Number = 5
```

Note:

- The %d format specifier is used to print `int` data types.
- Here, the %d inside the quotations will be replaced by the value of `testInteger`.

Print (Output) float and double

❖ The following C++ program outputs a float and a double number on the screen.

```
#include <stdio.h>
int main()
{
    float num1 = 13.5;
    double num2 = 12.4;
    printf("num1 = %f\n", num1);
    printf("num2 = %lf", num2);
    return 0;
}
```

Output

```
num1 = 13.500000
num2 = 12.400000
```

Note:

- To print a float number, `%f` format specifier is used.
- Similarly, we use `%lf` to print double values.

Print Characters – printf

❖ The following C++ program prints a character on the screen.

```
#include <stdio.h>
int main()
{
  char chr ='a';
  printf("Character = %c", chr);
  return 0;
}
```

Output

```
Character = a
```

Note:

➢ To print char, `%c` format specifier is used.

C input – scanf

- In C++ programming, scanf() is one of the commonly used input function
- The scanf() function is used to read input variables entered by users from the keyboard.

```
#include <stdio.h>
int main()
{
    int testInteger;
    printf("Enter an integer: ");
    scanf("%d", &testInteger);
    printf("Number = %d", testInteger);
    return 0;
}
```

Output

```
Enter an integer: 4
Number = 4
```

C input – scanf

```
scanf("%d", &testInteger);
printf("Number = %d", testInteger);
```

- Here, the `%d` format specifier is used inside the `scanf()` function to take **int** input from the user.
- When the user enters an integer, it is stored in the **testInteger** variable.
- Also, notice, that we have used **&testInteger** inside `scanf()`

  → because **&testInteger** gets the *address* of **testInteger**, and the value entered by the user is stored in that address.

Format Specifiers for I/O

- As can noticed from the previous examples, we have used the following format specifiers:
  - **%d** for **int**
  - **%f** for **float**
  - **%lf** for **double**
  - **%c** for **char**

Scan (Input) float and double

- The following C++ program inputs a float and double number from the keyboard and the display them on the screen.

```c
#include <stdio.h>
int main()
{
    float num1;
    double num2;
    printf("Enter a number: ");
    scanf("%f", &num1);
    printf("Enter another number: ");
    scanf("%lf", &num2);
    printf("num1 = %f\n", num1);
    printf("num2 = %lf", num2);
    return 0;
}
```

Output

```
Enter a number = 12.523
Enter another number = 10.2
num1 = 12.523000
num2 = 10.200000
```

Input/Output a character

- The below c++ program inputs a char from the keyboard, then displays (outputs) its ASCII on the screen

```
#include <stdio.h>
int main()
{
    char chr;
    printf("Enter a character: ");
    scanf("%c", &chr);
    // When %c is used, a character is displayed
    printf("You entered %c.", chr);
    // When %d is used, ASCII value is displayed
    printf("ASCII value is %d", chr);
    return 0;
}
```

Output

```
Enter a character: w
You entered w.
ASCII value is 120
```

I/O multiple values

- Take multiple values form the user and display them on the screen

```
#include <stdio.h>
int main()
{
    int a;
    float b;
    printf("Enter an integer and then a float: ");
    scanf("%d%f", &a, &b);
    printf("You entered %d and %f", a, b);
    return 0;
}
```

Output

```
Enter an integer and then a float: -3
3.4
You entered -3 and 3.400000
```

THANK YOU

المحاضره الرابعة

LECTURE

4&3

DR.AZHAAR A.HUSSAN

STRUCTURE OF A PROGRAM

- A computer program is a sequence of instructions that tell the computer what to do.

- **1-1-2Statements and expressions**
- The most common type of instruction in a program is the **statement**.
- A statement in C++ is the smallest independent unit in the language.
- In C++, we write statements in order to convey to the compiler that we want to perform a task.
- Statements in C++ are terminated by a semicolon.
- There are many different kinds of statements in C.++

- The following are some of the most common types of simple statements:
- int x;
- x = 5
- cout << x;

- An **expression** is a mathematical entity that evaluates to a value.
- For example, in math, the expression 2+3 evaluates to the value 5
- **For example,** the statement x = 2 + 3; is a valid assignment statement.
- The expression 2+3 evaluates to the value of .5This value of 5 is then assigned to x.
-

2-2 L KEYWORDS

- *Keywords* in C++ are explicitly reserved words that have a strict meaning and may not be used in any other way. They include words used for type declarations, such as int, char, and float; words used for statement syntax, such as do, for, and if; and words used for access control, such as public, protected, and private. Table 2.2 shows the keywords in current C++ systems.

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

IDENTIFIERS, AND NAMING THEM

- The name of a variable, function, class, or other entity in C++ is called an **identifier**
- there are a few rules that must be followed when naming identifiers:

1. The identifier cannot be a keyword. Keywords are reserved.
2. The identifier can only be composed of letters, numbers, and the underscore character. That means the name can not contains symbols (except the underscore) nor whitespace.
3. The identifier must begin with a letter or an underscore. It can not start with a number.
4. C++ distinguishes between lower and upper case letters.

- **nValue** is different than **NVALUE.**
- int value; *// correct*
- int Value; *// incorrect (should start with lower case letter(*
- int VALUE; *// incorrect (should start with lower case letter(*
-
- int VaLuE; *// incorrect;*

- If the identifier is a multiword name, there are two common conventions: separated by
- **underscores**, or **intercapped**.
- int my_variable_name; *// correct (separated by underscores(*
- int myVariableName; *// correct (intercapped(*
- int my variable name; *// incorrect (spaces not allowed(*
- int MyVariableName; *// incorrect (should start with lower case letter*

- **Very important:** The C++ language is a "case sensitive" language.
- That means that an identifier written in capital letters is not equivalent to another one with the same name but written in small letters.
- Thus, for example, the **RESULT** variable is not the same as the **result** variable or the **Result** variable.
- These are three different variable identifiers.

VARIABLES2-5 L

- Variables are a way of reserving memory to hold some data and assign names to them so that we 46735 don't have to remember the numbers like and instead we can use the memory location by .simply referring to the variable
- Every variable is mapped to a unique memory .address
- .Every variable in C++ can store a value
- However, the type of value which the variable can .store has to be specified by the programmer

- -:C++ supports the following inbuilt data types

int (to store integer values) There are three types of integer variables.1
,++in C

- **short**, **int** and **long int**
- **;Short a**
- ;int i
- ;25int i=
- ;299long b=
- ;int s,j,k

float (to store decimal values) Floating point data types comes in **.2**
.three sizes, namely float, double and long double

-
- ;float a
- ;4.84float b=
- ;6e2float a=
- ;double sum

(char (to store characters.3

- ;char a
- ;’char name=’s

(1or 0 bool (to store Boolean value either .4

- A variable of type bool can store either value true or false and is mostly used in comparisons and logical operations. When converted to integer, true is any non zero .value and false corresponds to zero

.To see what variable declarations look like in action within a program

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

SCOPE OF 2-5-1 L VARIABLES

- All the variables that we intend to use in a program must have been declared with its type specifier in an earlier point in the code,
- like we did in the previous code at the beginning of the body of the function main when we declared that a, b, and result were of type int.
- A variable can be either of global or local scope.

1. A global variable is a variable declared in the main body of the source code, outside all functions,
2. while a local variable is one declared within the body of a function or a block.

-

```
#include <iostream>

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

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

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

Global variables

Local variables

Instructions

INITIALIZATION OF 2-6 L VARIABLES

- When declaring a regular local variable, its value .is by default undetermined
- But you may want a variable to store a concrete .value at the same moment that it is declared
- In order to do that, you can initialize the variable. :++There are two ways to do this in C

.1 The first one, known as c like, is done by appending an equal sign followed by the value :to which the variable will be initialized

- **; type identifier = initial_value**
- For example
- **;0int a =**

.2 The other way to initialize variables, known as constructor initialization, is done by enclosing the initial value between parentheses (()):

- **; (type identifier (initial_value**
-
- For example:
- **;(0int a (**

- Both ways of initializing variables are valid and equivalent in C++
-
- **//initialization of variables**
- **#include <iostream<**
- **int main()**
- **}**
- **int a=5; // initial value = 5**
- **int b(2); // initial value = 2**
- **int result;// initial value undetermined**
- **a = a + ;3**
- **result = a - b;**
- **cout << result;**
- **return ;0**
- **{**
- 6

5 LECTURE 6&

LITERALS

- Literals are used to express particular values .within the source code of a program
- We have already used these previously to give concrete values to variables or to express messages we wanted our programs to print out, :for example, when we wrote
- ;5a =
- .in this piece of code was a literal constant5 The
- **Literal constants** can be divided in Integer Numerals, Floating-Point Numerals, Characters, .Strings and Boolean Values

INTEGER NUMERALS

- They are numerical constants that identify .integer decimal values
- Notice that to express a numerical constant we do not have to write quotes (") nor any special character
- **1776**
- **707**
- **273-**

- In addition to decimal numbers ,C++ allows the use as) and 8literal constants of octal numbers (base .(16hexadecimal numbers (base
- to havewe numberoctalan expressto wantwelf .((zero character0 precede it with a
- And in order to express a hexadecimal number we have .(x (zero, x0to precede it with the characters
- all are constantsliteralfollowingtheexample,For :equivalent to each other
- **decimal // 75**
- **octal // 0113**
- **b // hexadecimal4x0**

- (seventy- 75 All of these represent the same number: numeral, octal numeral 10 five) expressed as a base- .and hexadecimal numeral, respectively
- Literal constants, like variables, are considered to have .a specific data type
- By default, integer literals are of type int. However, we can force them to either be unsigned by appending the :u character to it, or long by appending l
- **int // 75**
- **u // unsigned int75**
- **l // long75**
- **ul // unsigned long75**

- In both cases, the suffix can be specified using either upper or lowercase letters.

FLOATING POINT NUMBERS

- They express numbers with decimals and/or .exponents
- They can include either a decimal point, an e character (that expresses "by ten at the Xth height", where X is an integer value that follows the e character), or both a :decimal point and an e character
- **3.14159 // 3.14159**
- **23^10x 6.02 // 23 e6.02**
- **19^-10x 1.6 // 19 e-1.6**
- **3.0 // 3.0**

- decimals with numbersvalidfourareThese .++expressed in C

1. ,The first number is PI
2. ,the second one is the number of Avogadro
3. the third is the electric charge of an electron (an extremely small number) **(all of them (approximated**
4. and the last one is the number three .expressed as a floating point numeric literal

- The default type for floating point literals is .double
- If you explicitly want to express a float or long double numerical literal, you can use the f or l :suffixes respectively

- **L // long double3.14159**
- **f // float23e6.02**

- Any of the letters than can be part of a floating-point numerical constant (e, f, l) can be written using either lower or uppercase letters without any difference in their .meanings

CHARACTER AND STRING LITERALS

- There also exist non-numerical constants, like:
- 'z'
- 'p'
- "Hello world"
- "How do you do?"
- The first two expressions represent single character constants.
- and the following two represent string literals composed of several characters.

Notice that to represent a single character we enclose it between single quotes (') and to express a string (which generally consists of more than one character) we enclose it between double quotes (").

- Character and string literals have certain peculiarities, like the escape codes.
- These are special characters that are difficult or impossible to express otherwise in the source code of a program,
- (like newline (\n) or tab (\t).
- All of them are preceded by a backslash (\). Here you have a list of some of such escape codes:

- n\ newline
- r\ carriage return
- t\ tab
- b\ backspace
- a\ alert (beep)
- '\ (') single quote
- "\ double quote (")
- ?\ question mark (?)
- \\ (\) backslash

- For example:
- **'n\'**
- **'t\'**
- **"Left \t Right"**
- **"one\ntwo\nthree"**

- String literals can extend to more than a single line of code by putting a backslash sign (\) at the end of each unfinished line. "string “expressed in \ two lines

DEFINED CONSTANTS ((#DEFINE

- You can define your own names for constants that you use very often without having to resort to memory-consuming variables, simply by using the
- :define preprocessor directive. Its format is#
- **define identifier value#**
- :For example
- **3.14159265define PI #**
- **'define NEWLINE '\n#**
- This defines two new constants: PI and NEWLINE. Once they are defined, you can use them in the rest of the code as if they were any other regular constant, for :example

WRITE A PROGRAM TO DETERMINE CIRCUMFERENCE ?OF THE CIRCLE

- defined constants //
- <include <iostream.h#
- 3.14159define PI #
- 'define NEWLINE '\n#
- () int main
- }
- ; // radius5.0double r=
- ;double circle
- ;* PI * r2 circle =
- ;cout << circle
- ;cout << NEWLINE
- { ;0return
- 31.4159O/P =

- **Notes**
- The #define directive is not a C++ statement but a directive for the preprocessor;
- therefore it assumes the entire line as the directive and does not require a semicolon (;) at its end.
- If you append a semicolon character (;) at the end, it will also be appended in all occurrences within the body of the program that the preprocessor replaces.

(DECLARED CONSTANTS (CONST

- With the const prefix you can declare constants with a specific type in the same way as you would do with a variable:
- **const int pathwidth = 100;**
- **const char tabulator = '\t';**

- Here, path width and tabulator are two typed constants. They are treated just like regular variables except that their values cannot be modified after their definition.

المحاضرة الخامسة

Operators in C++

- The table below lists all possible operators that can be executed in C++ programming.

#	Types of C operator	Symbols of operators
1	Arithmetic operators	+ → Addition - → Subtraction * → Multiplication / → Division % → Modulo (remainder after division)
2	Logical operators	&& → Logical AND || → Logical OR ! → Logical NOT
3	Assignment operators	= → Assigns value of the left side to the right side += → a+=b is same as a = a+ b -= → a-=b is same as a = a- b *= → a*=b is same as a = a* b /= → a/=b is same as a = a/ b %= → a%=b is same as a = a% b

Operators in C++ (Cont…)

#	Types of C operator	Symbols of operators
4	Relational operators	< → Less than > → Greater than <= → Less than or equal to >= → Greater than or equal to == → is equal to != → is not equal to
5	Increment and Decrement operators	++ → increment value by 1 -- → decrement value by 1

6	Conditional operators	? → Example: x = (a>b)?a:b;
		it means
		If (a>b)
		x = a;
		else
		x = b;

7	Bitwise operators	& → bitwise AND | → bitwise OR ^ → bitwise exclusive-OR << → Shift Left (it multiples number by 2) >> → Shift Right (it divides number by 2)

Operators in C++ programming – Explanation

- Arithmetic operators are two types
  - Unary operators: operators that operate on a single operator

    Example

    b = a++;
  - Binary operators: operators that operate with two operators

    Example

    c = a+b;
    - '+' is the operator known as addition operator,
    - 'a' and 'b' are operands
    - The addition operator tells the compiler to add both of the operands 'a' and 'b' and put them in c variable

Logical operators:

- The result of the operation of a logical operator is a **Boolean** value either *true* or *false*.
- used to test one or more condition or make decisions

Operators in C++ programming – Explanation

- Relational operators are used to compare two numbers and taking decisions based on their relation.

- Relational expressions are used in decision statements such as ***if*** , ***for***.

- Conditional operator: It takes three arguments, **condition? exp1:exp2** if condition is true then execute exp1 otherwise exp2 will be executed.

- **Bitwise Operators**

  - Bitwise operators are used to perform operation bit by bit.

  - Bitwise operators may not be applied to float or double.

Assignment Operators in C++

- The Assignment operators in C++ are some of the C Programming Operator, which are useful to assign the values or the result of an expression to the declared variables.
- The equals (=) operator is the most commonly used assignment operator in C++.
- For example:

```
int i = 10;
```

Arithmetic operators – Example

```
#include <stdio.h>

int main() {

   int a = 9, b = 4, c;

   c = a+b;                          //addition + → (a+b=13)
   printf("a+b = %d \n", c);

   c = a-b;                          //subtraction - → (a-b=5)
   printf("a-b = %d \n", c);

   c = a*b;                          //multiplication * → (a*b=36)
   printf("a*b = %d \n", c);

   c = a/b;                          //division / → (a/b=2)
   printf("a/b = %d \n", c);

   c = a%b;                          //modulo % → (a%b=1)
   printf("Remainder when a divided by b = %d \n", c);

return 0;
}
```

Assignment Operators in C

- The below table displays all the assignment operators present in C Programming with an example.

C Assignment Operators	Example	Explanation
=	x = 25	Value 25 is assigned to x
+=	x += 25	This is same as → x = x + 25
–=	x –= 25	This is same as → x = x – 25
∗=	x ∗= 25	This is same as → x = x * 25
/=	x /= 25	This is same as → x = x / 25
%=	x %= 25	This is same as → x = x % 25

Assignment Operators in C++ – Example

- In following C++ assignment operators Program, two integer variables a and Total are used. And their values are 7 and 21, respectively. It shows the working functionality of all the Assignment Operators in C++ Programming Language.

```
/* Program for Assignment Operators in C*/
#include <stdio.h>

int main()
{
  int a = 7;
  int Total = 21;

   printf(" Value of the Total = %d \n", Total += a );
   printf(" Value of the Total = %d \n", Total -= a );
   printf(" Value of the Total = %d \n", Total *= a );
   printf(" Value of the Total = %d \n", Total /= a );
   printf(" Value of the Total = %d \n", Total %= a );

    return 0;
  }
```

Assignment Operators in C++ – Example

```c
/* Program for Assignment Operators in C*/
#include <stdio.h>

int main()
{
  int a = 7;
  int Total = 21;

   printf(" Value of the Total = %d \n", Total += a );
   printf(" Value of the Total = %d \n", Total -= a );
   printf(" Value of the Total = %d \n", Total *= a );
   printf(" Value of the Total = %d \n", Total /= a );
   printf(" Value of the Total = %d \n", Total %= a );

    return 0;
  }
```

output

```
Value of the Total = 28
Value of the Total = 21
Value of the Total = 147
Value of the Total = 21
Value of the Total = 0
```

Note:

- `\n` : means new line
- `\t` : means take a tab space then write

Assignment Operators in C++ – Example

- The ***printf*** statements will perform C++ Programming Assignment operations on **a** and **Total** and then display the output (result).
- Let us see the C Programming Operator functionality in this C Program

```
printf(" Value of the Total = %d \n ", Total += a );
```

Total += a
means → Total = Total + a = 21 + 7 = 28

```
printf(" Value of the Total = %d \n ", Total -= a );
```

Total -= a
means → Total = Total - a = 28 - 7 = 21

```
printf(" Value of the Total = %d \n ", Total *= a );
```

Total *= a
means → Total = Total * a = 21 * 7 = 147

Assignment Operators in C – Example

```
printf(" Value of the Total = %d \n ", Total /= a );
```

Total /= a

means →Total = Total / a = 147 / 7 = 21

```
printf(" Value of the Total = %d \n ", Total %= a );
```

Total %= a

means →Total = Total % a = 21 % 7 = 0 (the remainder of 21/7 is 0)

Increment and Decrement – Explanation

❖ Increment → ++

- **a++** is postfix, the expression is evaluated first and then the value is incremented.

  Example

  a = 10;

  b = a++; // after this statement, a = 11, b = 10

- **++a** is prefix, the value is incremented first and then the expression is evaluated.

  Example

  a = 10;

  b =++a; // after this statement, a = 11, b = 11

Increment and Decrement – Explanation

- **Decrement →** **–**
  - **a--** is postfix, the expression is evaluated first and then the value is decremented.

    Example

    a = 10;

    b = a--; // after this statement, a = 9, b = 10

  - **--a** is prefix, the value is decremented first and then the expression is evaluated.

    Example

    a = 10;

    b = --a; // after this statement, a = 9, b = 9

Class #2 – Adding two double numbers

```c
#include <stdio.h>
 int main()
{
    double a, b, Sum = 0;

    printf("Enter the 1st number: ");
    scanf("%lf", &a);


    printf("Enter the 2nd number: ");
    scanf("%lf", &b);


    Sum = a + b;
    printf("%lf + %lf = % lf", a, b, Sum);

     return 0;

}
```

Output:

?

Build and run

```
Faris.Alghareb@Users-MacBook-Pro ~ % '/Users/user/Documents/
Enter the 1st number: 5.5
Enter the 2nd number: 3.2
5.500000 + 3.200000 = 8.700000
Faris.Alghareb@Users-MacBook-Pro ~ %
```

Area calculation of a circle

Write a C program that reads the radius of a circle and and prints its area and circumference.

Ans:

```
#include <stdio.h>

int main() {

?

return 0;
}
```

THANK YOU

Algorithm:

algorithm can be defined as a finite sequence of effect statements to solve a problem. An effective statement is a clear, unambiguous instruction that can be carried out.

Algorithm properties:

•**Finiteness:** the algorithm must terminate a finite number of steps.
• **Non-ambiguity:** each step must be precisely defined. At the completion of each step the nest step should be uniquely determined .
•**Effectiveness:** the algorithm should solve the problem in a reasonable amount of time.

Ex1:Write algorithm to find maximum number from two numbers.

1.start
2. input two numbers(num1 and num2)
3.if num1 greater than num2 print num1
4.if num2 greater than num1 print num2
5. end

*****

Ex3 :Write algorithm to find the division two number

1.Start
2.Input first number
3.Input second number
4.If second number < > 0 Goto 6
5.If second number = 0 Goto 9
6.Calculate division
7.Print the result
8.Goto 10
9.Print error
10.end

****

A flowchart

is a graphical representation of an algorithm or of a portion of an algorithm. Flowcharts are drawn using symbols. The main symbols used to draw a flowchart are shown in following figure.

Symbol	Name	Function
	Start/end	An oval represents a start or end point.
	Arrows	A line is a connector that shows relationships between the representative shapes.
	Input/Output	A parallelogram represents input or ouptut.
	Process	A rectangle represents a process.
	Decision	A diamond indicates a decision.

Ex1:Draw the flowchart of maximum number between two numbers?

Ex2: Draw the flowchart of sum two numbers?

Ex3: Draw the flowchart of the division of two numbers?

ex4 :draw the flowchart to find the sum of negative numbers among 50 numbers.

Homework:

1)Draw a flowchart to read 3 numbers: x , y and z and print the largest number of them.

2)Write an algorithm that finds the average of three degrees. If the average is successful, printed word "success" ,opposite of that printed word "fail"?

Arrays:

An array is a variable that can store multiple values of the same type. Suppose a class has 27 students, and we need to store the grades of all of them. Instead of creating 27 separate variables, we can simply create an array.

An array is a consecutive group of homogeneous memory locations. Each element (location) can be referred to using the array name along with an integer that denotes the relative index of that element within the array. The data items grouped in an array can be simple types like int or float; or can be user-defined types like structures and objects.

1. Array of one dimension:

It is a single variable that specifies each array element. The general declaration of one- dimensional arrays is:

dataType arrayName[arraySize];

For example,

```
int     age [10];
int     num
[30]; float
degree[5];
char  a [15];
```

The item in an array are called elements (in contrast to the items in a structure which are called members). The elements in an array are of the

.same type only the values vary

Initializing array elements:

The initializing of array can be done in several methods such as:

- The first element of array named **age**: **age** [0] = 18;
- The last element of array **age**: **age** [9] = 19;
- All elements of array **age**:

  age [9] = { 18, 17, 18 ,18 ,19, 20 ,17, 18 ,16,19 };
- int x [ ] = { 12, 3, 5, 0, 11, 7, 30, 100, 22 };
- int y [10] = { 8, 10, 13, 15, 0, 1, 17, 22};

Accessing array elements:

We can access each array element by written the name of array, followed by brackets delimiting a variable (or constant) in the brackets which are called the array index.

- Accessing the first element of array **num** to variable x.

  x = num [0];
- Print the second element of the array **age**:
- cout<<age[1];
- Accessing the last element of array **num** to variable y:

  y = num [9];
- cout << **num** [0] + **num** [9];
- **num** [0] = **num** [1] + **num**[2];
- **num** [7] = **num** [7] + 3;

Input/print /process array elements:

Herein several C++ codes illustrate how to input, print, and processing of **num** array elements:

Input	Processing	Printing

cin>>num[10];	sum=0;
	for (int i=0;i<10;i++){
	sum=sum+num[i];
	}
		cout << num [4];
		cout>>num[9];
		cout>>num[0];
		cout>>num[2]+4;

cin>>num[0];
cin>>num[2];
	for (int i=0; i<10; i++){
	num[i]=num[i]+1;
	}
		for (int i=0; i<10; i++) {
		cout << num[i];
		}

for (int i=0<10; i++){
cin>>num[i];
}
	num[0]=num[2]+num[3];
	num[1]=num[2]*num[3];
	num[2]=num[2]*2;
	num[3]=num[0]+num[5];
		if (num [5] > 5)
		{
		cout << "greater";
		}

Another examples of C++ code:

int mark[5] = {19, 10, 8, 17, 9};

marks	19	10	8	17	9	Value
	0	1	2	3	4	Index

Now, change 4th element to 50

mark[3] = 50;

- **The result is:**

marks	19	10	8	**50**	9	Value
	0	1	2	3	4	Index

Then, take input from the user and store the input at the third index. Suppose that the input value is **150**.

cin >> mark[2];

- **The result is:**

marks	19	10	**150**	50	9	Value
	0	1	2	3	4	Index

And, take input from the user and insert it into ith index. Suppose that the value of **i** is 2 and the input value is **20**.

```
cin >> mark[i-1];
```

- **The result is:**

marks	19	**20**	150	50	9	Value
	0	1	2	3	4	Index

Print first element of the array

```
cout << mark[0];
```

- **The result is:**

19

Print ith element of the array. Suppose that the **i** is 5.

```
cout >> mark[i-1];
```

- **The result is:**

9

Ex 1:

- Write a C++ program to read 1-D array and print it.

```
#include <iostream.h>
#include <conio.h>
int main()
{
        int numbers[5] = {7, 5, 6, 12, 35};
       cout << "The numbers are: ";
      for (int i = 0; i < 5; i++)
      {
              cout << numbers[i] << "  ";
      }
  return (0);
}
```

Output

```
The numbers are: 7  5  6  12  35
```

Ex 2:

- Write a C++ program to take **five** inputs from user and store them in an array called number. After that, print them.

```
#include <iostream.h>
#include <conio.h>
void main()
{
   int numbers[5];
   cout << "Enter 5 numbers: " << endl;
   for (int i = 0; i < 5; i++)
     {
          cin >> numbers[i];
     }
   cout << "The numbers are: ";
   for (int n = 0; n < 5; n++)
     {
          cout << numbers[n] << "  ";
     }
}
```

Output

```
Enter 5 numbers:
12
13
14
15
16
The numbers are: 12  13  14  15  16
```

Ex 3:

- Write a C++ program to add **1** to all elements of array called **number**. After that, print the result.

```
#include <iostream.h>
void main()
{       int numbers[5] = {7, 5, 6, 12, 35}, i ;
        cout << "The numbers are: ";
        for (i = 0; i < 5; i++)
        {
                cout << numbers[i] << "  ";
        }
```

```
    for (i = 0; i < 5; i++)
    {
            numbers[i]=numbers[i]+1;
    }
    cout<<"\n";
    cout << "The numbers after adding are: ";
    for (i = 0; i < 5; i++)
    {
        cout << numbers[i] << " ";
    }
}
```

Output

```
The numbers are: 7  5  6  12  35
The numbers after adding are: 8  6  7  13  36
```

Ex 4:

- Write a C++ program to read **4** numbers and print it in reverse order.

```
#include <iostream>
void main()
  {
        int a [4],i;
        cout << "Enter 5 numbers: \n";
        for (  i =0; i <4; i++ )
        {
                cout << i << ": ";
                cin >> a [ i ];
                cout << "\n";
        }
        cout << "The reverse order is: \n";
        for ( i =3; i >=0; i-- )
        {
                cout << i << ": " << a [ i ] << endl;
        }
}
```

Output

```
Enter 4 numbers:
0: 10
1: 20
2: 30
3: 40
The reverse order is:
3: 40
2: 30
1: 20
0: 10
```

Ex 5:

- Write a C++ program, to find the summation of array elements. Suppose that the size of array is 3.

```
#include <iostream.h>
void main()
{
  int a [3], sum = 0;
  cout << "Enter 3 numbers\n";
  for ( int i =0; i <3; i++ )
  {       cout << "Enter the number:";
          cin >> a [ i ];
          sum = sum+a [ i ];
  }
  cout << "The sum is: " << sum << endl;
}
```

Output

```
Enter 3 numbers
Enter the number:7
Enter the number:5
Enter the number: -1
The sum is: 11
```

Ex 6:

- Write a C++ program, to split the odd numbers and even numbers of one array into two arrays.

```
void main()
 {      int a [ 10 ]= { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, odd[10], even [10], i , o=0, e=0;
        cout<<"The number of array are:"<<endl;
        for ( i=0 ; i<10; i++ )
        {       cout<<a[i]<<"  ";
        }
        for ( i=0 ; i<10; i++ )
        {       if (a[i] % 2 !=0)
                {       odd[o]=a[i];
                        o=o+1;
                }
                else
                {       even[e]=a[i];
                        e=e+1;
                }
        }
        cout<<endl;
```

Output

```
The number of array are:
1  2  3  4  5  6  7  8  9  10
The odd numbers are:
1  3  5  7  9
The even numbers are:
2  4  6  8  10
```

```
        cout<<"The odd numbers are:"<<endl;
        for ( i=0 ; i<o; i++ )
        {
                cout<<odd[i]<<"  ";
        }
        cout<<endl;
        cout<<"The even numbers are:"<<endl;
        for ( i=0 ; i<e; i++ )
        {
                cout<<even[i]<<"  ";
        }
    }
```

Ex 7:

- Write a C++ program, to add two arrays. Then, print the result.

```
#include<iostream.h>
int main()
{
  int first[20], second[20], sum[20], c, n;
  cout << "Enter the number of elements:";
  cin >> n;
  cout << "Enter elements of first array" << endl;
  for (c = 0; c < n; c++)
   {      cin >> first[c];
   }
  cout << "Enter elements of second array" << endl;
  for (c = 0; c < n; c++)
  {      cin >> second[c];
   }
  cout << "Sum of elements of the arrays:" << endl;
  for (c = 0; c < n; c++)
  {      sum[c] = first[c] + second[c];
         cout << sum[c] << endl;
  }
}
```

Output

```
Enter the number of elements: 3
Enter elements of first array
1
2
3
Enter elements of second array
4
5
6
Sum of elements of the arrays:
5
7
9
```

Homework:

1. **Which command shall we write in order to print the second element of array distance [] = { 23, 7, 1, 4, 6} ?**
2. **What is the output of the following C++ Program?**

```
#include<iostream.h>
void main ( )
 {
      int number[3]=[10,2,33]
      number [0]= number[0]+3;
      cout << number [1];
 }
```

3. **What are the first and last Indexes of the below array?**

```
int num [9];
```

4. **What is the output of the following C++ Program?**

```
#include<iostream.h>
int main()
{
        int ary[4];
        ary[4] = {1,2,3,4};
        cout << ary[2];
       return 0;
 }
```

5. **What is the output of the following C++ Program?**

```
#include<iostream.h>
int main()
{         int ary[] = {10,20,30},
          for (int i=0;i<3;i++)
              {      cout<< ary[i]+1<<" ";  }
}
```

6. What is the output of the following C++ Program?

```
int main ()
{        int array[] = {0, 2, 4, 6, 7, 5, 3};
         int n, result = 0;
         for (n = 0 ;n < 8 ;n++) {
             result += array[n];
         }
         cout << result;
}
```

7. Write a c++ code to print the third element of the array age.
8. Suppose that you have the following array.

   int mark [5] = {19, 10, 8, 17, 9};

   change 2nd element to 17.
9. What is the output of the following C++ Program?

```
int main ()
{       int mark[] = {4, 2, 4, 6, 7};
        mark[0]=mark[0]+mark[0];
        mark[1]=mark[1]+mark[4];
        mark[3]=1+mark[4];
        cout<< mark[0]+11<<endl;
        cout<< mark[1] -1<<endl;
        cout<< mark[3]+2 <<endl;
        cout<< mark[4]*2;
}
```

10. Write a C++ program to subtract 1 from all elements of array called degree. After that, print the result. Suppose that the array has 6 elements.
11. Write a C++ program, to sum the positive numbers of array called degree. Suppose that the size of array is 5.

12. Write a C++ program, to sum the negative numbers of array called degree. Suppose that the size of array is 5.
13. Write a C++ program, to sum the odd numbers of array called degree. Suppose that the size of array is 5.
14. Write a C++ program, to sum the even numbers of array called degree. Suppose that the size of array is 5.
15. Write a C++ program, to split the positive numbers and negative numbers of one array into two arrays (one for positive number and the other for the negative number)
16. Write a C++ program, to find the average of the student. Suppose that the student has 7 degrees.
17. Write a C++ program, to subtract one array from the other array. Then, print the result.
18. Check whether each of the following statements is true or false and correct the false statement.
    - The dimension of array num [2] is 2.
    - In C++ programming language, in order to print the second element of array named BOOK we must write the command cout<<BOOK [2];
    - In C++ programming language, we can write the command int age [3] = {1,2,3];
    - Suppose that we have array called num [4] = {1,2,3,4}. The result of num [0] +num [2] is 4.
    - Suppose that we have array age [3] = {20,19,12}, we can write a C++ command:

      cin >> age [2];
    - Suppose that we have array salary [3] = {10,20,30} and sum=1. The result of sum=sum+a[1] is 21.
    - Suppose that we have array called money [4] = {1,2,4}. The result of money [0] +money [2] +1 is 5.