Programming in C++

Lecture Notes

1 History of C++

C++ programming language was developed in 1980 by Bjarne Stroustrup at Bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A. **Bjarne Stroustrup** is known as the **founder of the C++ language.** It was developed to add a feature of **OOP (Object Oriented Programming)** in C without significantly changing the C component. Therefore, C++ was created as an extension of C, and both languages have almost the same syntax. The main difference between C and C++ is that C++ supports classes and objects, while C does not.

2 Why Use C++

1. C++ is one of the world's most popular programming languages.
2. C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems.
3. C++ is an object-oriented programming language that gives a clear structure to programs and allows code to be reused, lowering development costs.
4. C++ is portable and can be used to develop applications that can be adapted to multiple platforms.
5. C++ is fun and easy to learn!

The following are the differences between C and C++

Sno	C	C++
1	C follows the **procedural style of programming.**	C++ is a multi-paradigm. It supports both **procedural and object-oriented.**
2	Data is less secure in C.	In C++, you can use modifiers for class members to make them inaccessible to outside users.
3	C follows the **top-down approach.**	C++ follows the **bottom-up**

	- Top-Down Model is a system design approach where the design starts from the system as a whole. The complete system is then divided into smaller sub-applications with more details. Each part again goes through the top-down approach till the complete system is designed with all the minute detail.	**approach** - **Bottom-Up Model** is a system design approach where a system's parts are defined in detail. Once these parts are designed and developed, then these parts or components are linked together to prepare a bigger component. This approach is repeated until the complete system is built.
4	C does not support function overloading.	C++ supports function overloading. - Two functions can have the same name with different numbers of parameters e.g. calculate () and calculate (int x) are two different functions
5	In C, you can't use functions in Structure.	In C++, you can use functions in Structure.
6	C does not support reference variables.	C++ supports reference variables. - E.g String s=”anova”; String s1=&s -
7	In C, **scanf() and printf()** are mainly used for input/output.	C++ mainly uses stream **cin and cout** to perform input and output operations.
8	Operator overloading is not possible in	Operator overloading is possible in

	C.	C++.
9	C programs are divided into **procedures and modules**	C++ programs are divided into **functions and classes.**
10	C does not provide the feature of the namespace.	C++ supports the feature of the namespace.
11	Exception handling is not easy in C. It has to perform using other functions.	C++ provides exception handling using Try and Catch block.
12	C does not support inheritance.	C++ supports inheritance.

3 C++ Data Types

A data type specifies the type of data that a variable can store, such as integer, floating, character, etc

Types	**Data Types**
Basic Data Type (Built-in/Primitive Data types)	int, char, float, double, void etc
Derived Data Type	array, pointer, reference etc
Enumeration Data Type	enum
User Defined Data Type	Structure, Class, Union, Typedef

4 Basic Data Types

The basic data types are integer-based and floating-point based. C++ language supports both signed and unsigned literals.

The memory size of basic data types may change according to 32 or 64-bit operating systems.

Data Types	Memory Size	Range
char	1 byte	-128 to 127
signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 127
short	2 byte	-32,768 to 32,767
signed short	2 byte	-32,768 to 32,767
unsigned short	2 byte	0 to 32,767
int	2 byte	-32,768 to 32,767
signed int	2 byte	-32,768 to 32,767
unsigned int	2 byte	0 to 32,767
short int	2 byte	-32,768 to 32,767
signed short int	2 byte	-32,768 to 32,767
unsigned short int	2 byte	0 to 32,767
long int	4 byte	
signed long int	4 byte	
unsigned long int	4 byte	
float	4 byte	
double	8 byte	
long double	10 byte	

Note: Other data types will be discussed in the coming sections

5 Operands and Operators

5.1 Operands

Operands are variables or values that operators can manipulate

5.2 Operators

Operators are symbols that perform operations on variables and values. C++ provides a rich set of operators to manipulate variables. We can divide all the C++ operators into the following groups.

i. Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators. Assume integer variable A holds 10 and variable B holds 20, then.

Operator	Description	Example	
* (Multiplication)	Multiplies values on either side of the Operator.	A * B will give 200	It is evaluated first. If there are several operators of this type, they're evaluated from left to right.
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2	
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0	
+ (Addition)	Adds values on either side of the Operator.	A + B will give 30	It is evaluated next. If there are several operators of this type, they are evaluated from left to right.
- (Subtraction)	Subtracts right-hand operand from the left-hand operand.	A - B will give -10	
= (Equal/Assignment)	Assign a value to a variable		Its is evaluated last.

ii. Relational Operators

There are the following relational operators supported by C++ language. Assume variable A holds 10 and variable B holds 20, then.

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

iii. Bitwise Operators

C++ defines several bitwise operators, which can be applied to the integer types long, int, short, char, and byte.

Bitwise Operator works on bits and performs the bit-by-bit operation. Assume if a = 60 and b = 13; now in the binary format, they will be as follows;

```
   a = 0011 1100
   b = 0000 1101
-----------------
a&b = 0000 1100
a|b =  0011 1101
a^b = 0011 0001
~a  = 1100 0011
```

The following table lists the bitwise operators;

Assume integer variable A holds 60 and variable B holds 13 then;

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12, which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61, which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61, which is 1100 0011 in 2's complement form due to

		a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operand value is moved left by the number of bits specified by the right operand.	A << 2 will give 240, which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operand value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

iv. Logical Operators

The following table lists the logical operators.

Assume Boolean variables A holds true and variable B holds false, then.

Operator	**Description**	**Example**
&& (logical and)	Called Logical AND Operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If	(A || B) is true

	any of the two operands are non-zero, then the condition becomes true.	
! (logical not)	Called Logical NOT Operator. Use to reverse the logical state of its operand. If a condition is true, then the Logical NOT operator will make false.	!(A && B) is true

v. Assignment Operators

Following are the assignment operators supported by the C++ language

Operator	Description	Example
=	Simple assignment operator. Assigns values from right-side operands to left-side operands.	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assigns the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C – A

*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A

6 C++ Identifiers

C++ identifiers in a program refer to the name of the variables, functions, arrays, or other user-defined data types created by the programmer. They are the basic requirement of any language. Every language has its own rules for naming the identifiers.

In short, we can say that the C++ identifiers represent the essential elements in a program which are given below:

- **Variables**
- **Constants**
- **Functions**
- **Labels**

- **Defined data types**

6.1 Some naming rules are common in both C and C++. They are as follows:

- Only alphabetic characters, digits, and underscores are allowed.
- The identifier name cannot start with a digit, i.e., the first letter should be alphabetical. After the first letter, we can use letters, digits, or underscores.
- In C++, uppercase and lowercase letters are distinct. Therefore, we can say that C++ identifiers are case-sensitive.
- A declared keyword cannot be used as a variable name.

For example, suppose we have two identifiers named 'FirstName', and 'Firstname'. Both identifiers will be different as the letter 'N' in the first case is in uppercase while lowercase is in the second. Therefore, it proves that identifiers are case-sensitive.

6.2 Valid Identifiers

The following are examples of valid identifiers are:

Result

Test2

_sum

power

6.3 Invalid Identifiers

Sum-1 // containing the special character '-'.
2data // The first letter is a digit.
break // use of a keyword.

7 C++ Variables

A variable is the name of a memory location whose value(data) can change during program execution. The value stored in the variable can be changed when it is reused many times. We name the memory location before storing data so that it will be easy to restore them when needed by referring to the memory location name.

The following is the syntax for declaring a variable.

Data_type variable_name;

e.g.

int x;

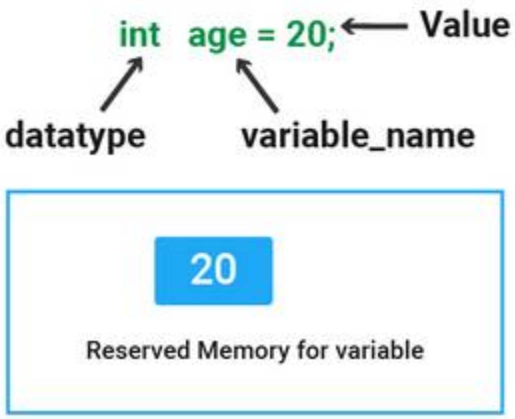
float y;

char z;

Here, x, y, and z are variables, and int, float, and char are data types.

We can also provide values while declaring the variables as given below:

1. int x=5,b=10; //declaring 2 variable of integer type
2. float f=30.8;
3. char c='A';



8 Basic Structure of a C++ Program

A **C++ program** is structured in a specific manner. In C++, a program is divided into the following three sections:

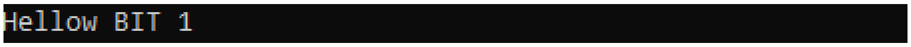
1. Standard Libraries Section
2. Main Function Section
3. Function Body Section

For example, let's look at the implementation of **We are BIT 1** program:

Program

```cpp
#include<iostream>
using namespace std;
int main()
{
    cout<<" We are BIT 1";
}

```

Output

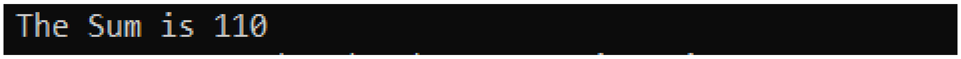


```
Hellow BIT 1
```

Question: Write a C++ Program to add two number

```
#include<iostream>
using namespace std;
int main()
{
    float a=10,b=100,sum;
    sum=a+b;

    cout<<" The Sum is "<<sum;
}

```

Output



```
The Sum is 110
```

9 C++ Constants

A constant is the name of a memory location whose value(data) can not change during program execution.

When you do not want others (or yourself) to change existing variable values, use the const keyword (this will declare the variable as "constant", which means **unchangeable and read-only**):

A constant is declared using the *constant* keyword

The syntax for declaring a constant

constant data_type constant_name;

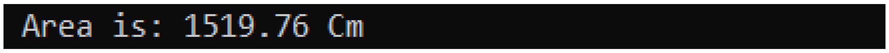
e.g., constant float PI=3.14

Question: Write a C++ program to calculate the area of a circle with a radius of 22cm.

Program

```cpp
#include<iostream>
using namespace std;
int main()
{
    int r=22;
    float PI=3.14,Area;
    Area=PI*r*r;
    cout<<" Area is: "<<Area <<" Cm";
}

```

Output



```
Area is: 1519.76 Cm
```

10 C++ Control Statements | Control Flow in C++

C++ compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, C++ provides statements that can be used to control the flow of C++ code. Such statements are called control flow statements. It is one of the fundamental features of C++, which provides a smooth program flow.

C++ provides three types of control flow statements.

1. Decision-Making statements
    - if statements
    - switch statement

2. Looping statements

   - for loop
   - do while loop
   - while loop
   - for loop

3. Jump statements

   - break statement
   - continue statement

10.1 Decision-Making Statements

Simple if statement:

It is the most basic statement among all control flow statements in C++. It evaluates a Boolean expression and enables the program to enter a code block if the expression is true.

Syntax

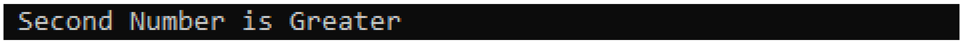
```
if(condition)

{

    Statement 1; //executes when the condition is true.

}
```

Question: Given two numbers**,** a=10 and b=20. Write a C++ Program to **determine** which **number** is greater and display **the output** on the screen**.**

Program

```
#include<iostream>
using namespace std;
int main()
{
    int a=10, b=20;
    if(a>b)
    {
        cout<<" First Number is Greater";
    }
    if(b>a)
    {
        cout<<" Second Number is Greater";
    }
    if(a==b)
    {
        cout<<" The given numbers are equal";
    }
}
```

Output



```
Second Number is Greater
```

if-else- statement

It is an extension to the if-statement, which uses another block of code, i.e., the else block. The else block is executed if the condition of the if-block is evaluated as false. Syntax

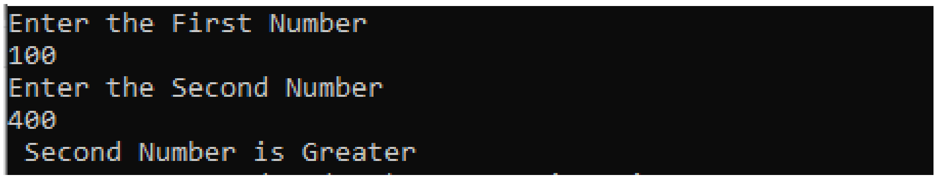
```
if(condition)
{
     Statement 1; //executes when the condition is true.
}
else
{
    Statement 2; //executes when the condition is false
}
```

Question: Write a C++ program to find the largest between two input numbers. Note that a program should receive numbers as input from the user.

Program

```cpp
#include<iostream>
using namespace std;
int main()
{
    int a, b;
    cout<<"Enter the First Number\n";
    cin>>a;
    cout<<"Enter the Second Number\n";
    cin>>b;
    if(a>b)
    {
        cout<<" First Number is Greater";
    }
    if(b>a)
    {
        cout<<" Second Number is Greater";
    }
    else
    {
        cout<<" The given numbers are equal";
    }
}
```

Output



```
Enter the First Number
100
Enter the Second Number
400
 Second Number is Greater
```

if-else-if statement

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that the chain of if-else statements creates a decision tree where the program may enter the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax

```
if (condition 1)
{
      Statement 1; //executes when condition 1 is true
}
else if (condition 2)
{
      Statement 2; //executes when condition 2 is true
}
else
{
       Statement 3; //executes when all the conditions are false
}
```

Question: Assume that CBE always assigns Diploma students' to different Bachelor's Degree Courses based on their GPA qualification, as shown in Table 1. CBE implemented a simple program to automate this task to avoid paperwork and save time. Assume that you are working as a programmer at CBE. Write a simple C++ program that can perform this task.

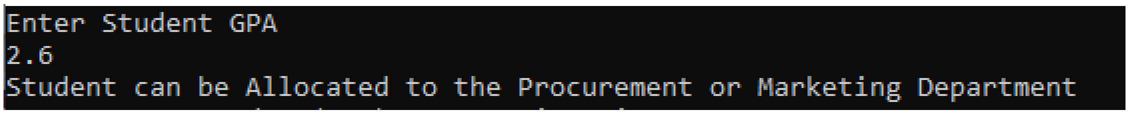
Table 1:

GPA Range	Department
4.5 - 5.0	ICT
4.5 - 5.0	Legal Metrology
3.5 - 4.4	Accounts
2.1 - 3.4	Procurement
2.1 - 3.4	Marketing

Program

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


int main()
{
    float GPA;
    cout<<"Enter Student GPA\n";
    cin>>GPA;
    if(GPA>=4.5 && GPA <=5.0)
    {
        cout<<"Student can be Allocated to the ICT or Legal Metrology Department";
    }
    else if(GPA>=3.5 && GPA<=4.4)
    {
        cout<<"Student is Allocated to the Accounts Department";
    }
    else if(GPA>=2.1 && GPA<=3.4)
    {
        cout<<"Student can be Allocated to the Procurement or Marketing Department";
    }
    else
    {
        cout<<"Student doe not qualify for any department";
    }
}

```

Output



```
Enter Student GPA
2.6
Student can be Allocated to the Procurement or Marketing Department
```

Question: Write a C++ program to display the student's examination results. Your program should perform the following: Accept the Student name, English Marks, Geography Marks and Biology Marks as inputs from the user; also calculate the average and grade based on the given average marks range as shown in Table 2. Finally, the program should display student names, marks obtained from each subject, average marks, and grades obtained.

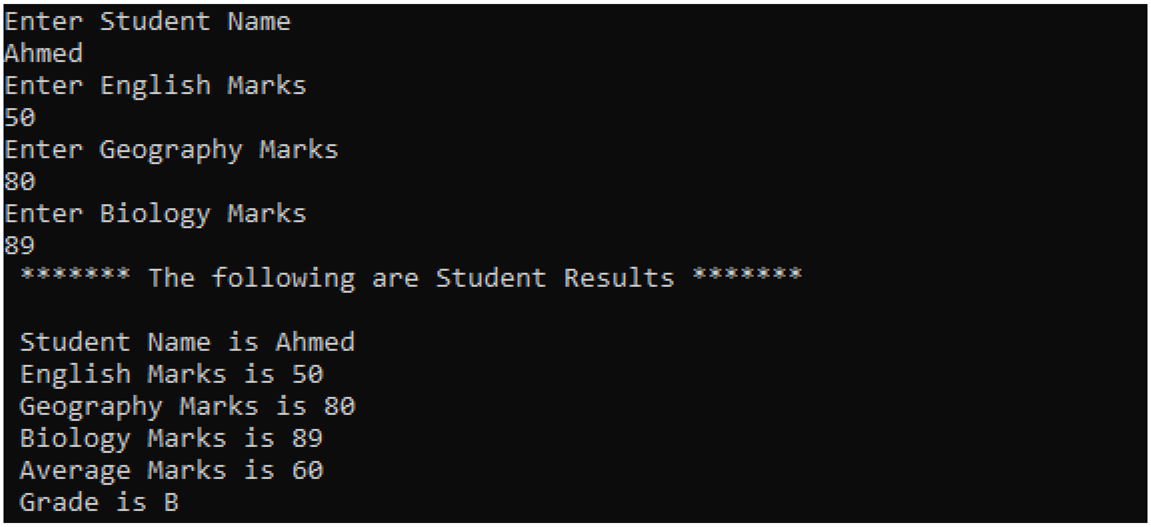
Table 2: Average Scores vs Grades

Average score	Grade
80-100	A
70-79	B+
60-69	B
50-59	C
0-49	F

Program

```
#include<iostream>
using namespace std;
int main()
{
    string studentName;
    float englmarks,geogmarks,biolmarks,averagegmarks;
    cout<<"Enter Student Name\n";
    cin>>studentName;
    cout<<"Enter English Marks\n";
    cin>>englmarks;
    cout<<"Enter Geography Marks\n";
    cin>>geogmarks;
    cout<<"Enter Biology Marks\n";
    cin>>biolmarks;
    averagegmarks =(englmarks+englmarks+geogmarks)/3;

    cout<<" ******* The following are Student Results *******\n";
    cout<<"\n Student Name is "<<studentName;
    cout<<"\n English Marks is "<<englmarks;
    cout<<"\n Geography Marks is "<<geogmarks;
    cout<<"\n Biology Marks is "<<biolmarks;
    cout<<"\n Average Marks is "<<averagegmarks ;

    if(averagegmarks>=80 && averagegmarks<=100)
    {
        cout<<"\n Grade is A";
    }
    else if(averagegmarks>=70 && averagegmarks<=79)
    {
        cout<<"\n Grade is B+";
    }
    else if(averagegmarks>=60 && averagegmarks<=69)
    {
        cout<<"\n Grade is B";
    }
    else if(averagegmarks>=50 && averagegmarks<=59)
    {
        cout<<"\n Grade is C";
    }
    else
    {
        cout<<"\n Grade is F";
    }
}
```

Output



```
Enter Student Name
Ahmed
Enter English Marks
50
Enter Geography Marks
80
Enter Biology Marks
89
 ******* The following are Student Results *******

 Student Name is Ahmed
 English Marks is 50
 Geography Marks is 80
 Biology Marks is 89
 Average Marks is 60
 Grade is B
```

10.2 Switch Case Statement

The C++ switch statement executes one statement from multiple conditions. It is like the if-else-if ladder statement; it tests the expression value against each case value. It executes the case value body if the expression value matches the case value. However, it executes the default body if no match is found. Each case statement can have an optional break statement. When control reaches the break statement, it exits the switch statement. If a break statement is not found, it executes the next case.

Syntax

```
Switch (expression)
{
    case value 1:
        //code to be executed;
        break;  //optional
    case value 2:
       //code to be executed;
        break;  //optional
    ......
    …..
    case value n:
    default:
     Code to be executed if all cases are not matched;
 }
```

Question: Using a switch case statement, write a C++ program to display the module name based on the given module code considering Table 3. Note that your program should accept module code as input from the user (*Also observe the output without using the break statements*).

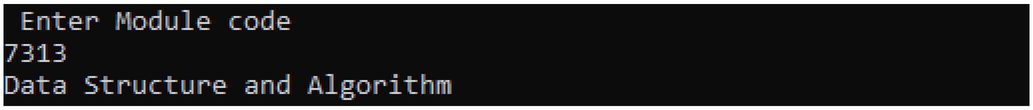
Table 4: Module code against the department

Module code	Department
7313	Data Structure and Algorithm
7312	Programming in Java
7212	Programming in C++

Program

```
#include<iostream>
using namespace std;
int main()
{
    int modulecode;
    cout<<" Enter Module code\n";
    cin>>modulecode;
    switch(modulecode)
    {
        case 7313:
            cout<<"Data Structure and Algorithm";
            break;
        case 7312:
            cout<<"Programming in Java";
            break;
        case 7212:
            cout<<"Programming in C++";
            break;
        default:
            cout<<"Please Select the Correct Code";
    }
}
```

Output



```
 Enter Module code
7313
Data Structure and Algorithm
```

10.3 Looping statements

Looping statements allow certain instructions to be executed repeatedly until a condition has become false. Consider a circumstance where you must print numbers ranging from 1 to 1000. How would you react? Will you type **cout** thousands of times or try to copy/paste it? Hopefully, it will be a tedious job. By using looping statements, this process can be completed efficiently.

for loop

Like a while loop, the for loop executes its internal part only if the condition is valid in each execution. Therefore, before performing any iteration, the for loop tests the condition to see if it is true.

Syntax

```
for(init; condition, incr/decr)
{
   //Statements to be executed or body
}
```

In the above syntax:

- init: The init expression is used for initializing a variable and is executed only once.
- Condition: It executes the condition statement for every iteration. It executes the loop's body if it evaluates the condition as true. The loop will continue to run until the condition becomes false.

- incr/decr: The increment or decrement statement is applied to the variable to update the initial expression.

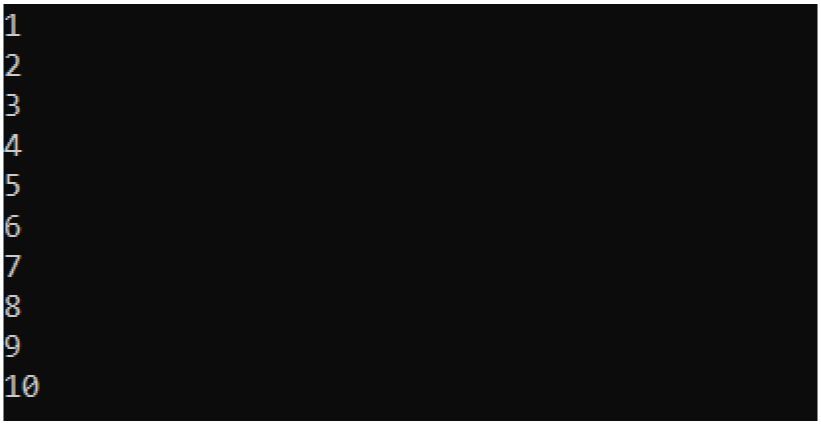
Note that: *The "init, condition and incr/decr" parts are enclosed inside the brackets in the for loop syntax*

Question: Write a C++ program to display 1 to 10 using for loop

Program

```
#include<iostream>
using namespace std;
int main ()
{
    int i;
    for (i=1;i<=10;i++)
    {
        cout<< i <<"\n";
    }
}
```

Output



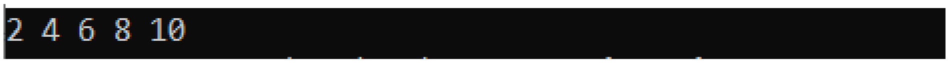
```
1
2
3
4
5
6
7
8
9
10
```

Question: Write a C++ program to display 2, 4, 6, 8, 10 using a for loop

Program

```
#include<iostream>
using namespace std;
int main()
{
    int i,n;
    for(i=1;i<=5;i++)
    {
        n=i*2;
        cout<<n<<" ";
    }
}
```

Output



```
2 4 6 8 10
```

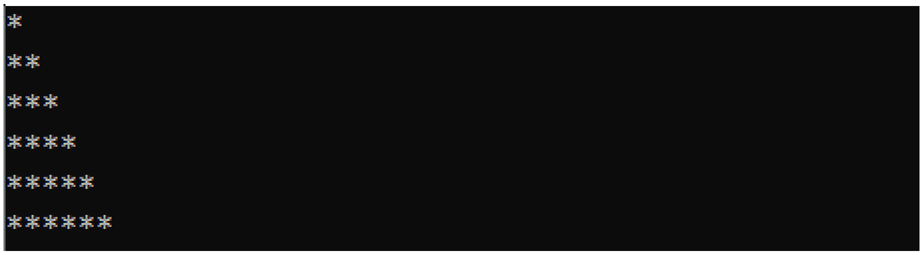
Question: Write a C++ program to display the following shape using for loop

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

Program

```
#include<iostream>
using namespace std;
int main()
{
    for(int i=1; i<=6; i++)
    {
        for(int j=1;j<=i;j++)
        {
          cout<<"*";
        }
        cout<<"\n";
    }
}
```

Output



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

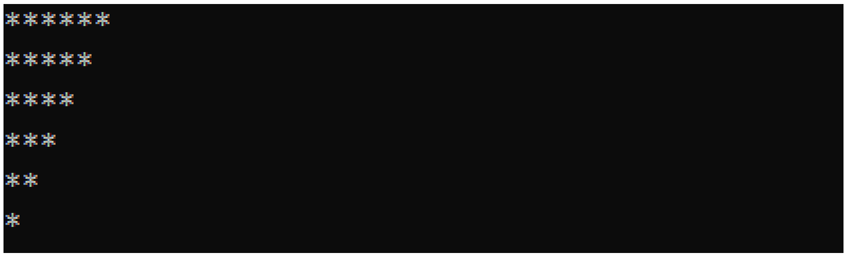
Question: Write a C++ program to display the following shape using for loop

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

Program

```
#include<iostream>
using namespace std;
int main()
{
    for(int i=1; i<=6; i++)
    {
        for(int j=1;j<=7-i;j++)
        {
          cout<<"*";
        }
        cout<<"\n";
    }
}
```

Output



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

Question: Write a C++   program to display the following shape using for loop

```
2
2 4
2 4 6
2 4 6 8
```

Program

```
#include<iostream>
using namespace std;
int main()
{
    for(int i=1; i<=4; i++)
    {
        for(int j=1;j<=i;j++)
        {
          int n=2*j;
          cout<<n;
        }
        cout<<"\n";
    }
}
```

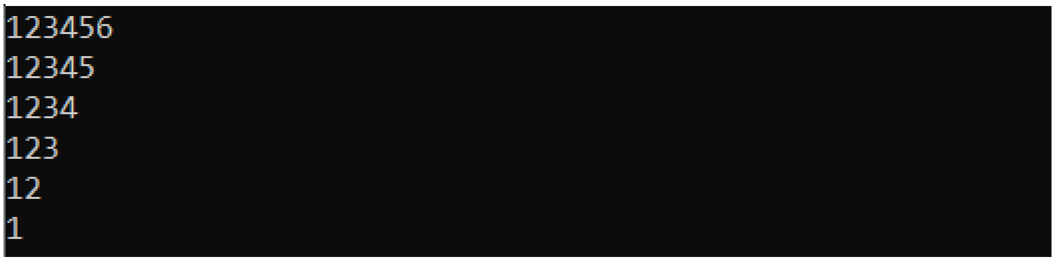
Question: Write a C++ program to display the following shape using a for loop

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

Program

```
#include<iostream>
using namespace std;
int main()
{
    for(int i=0; i<=6; i++)
    {
        for(int j=1;j<=6-i;j++)
        {
          cout<<j;
        }
        cout<<"\n";
    }
}

```

Output



```
123456
12345
1234
123
12
1
```

while loop

Like a for loop, the while loop executes its internal part only if the condition is valid in each execution. Therefore, before performing any iteration, the while looptests the condition to see if it is true. The while loop is similar to for loop; however, **the init, condition and the incr/decr** part are separate (not enclosed inside the bracket)

Syntax

```
init;
while (condition)
{
   //Statements to be executed or body
    incr/decr;
}
```

In the above syntax:

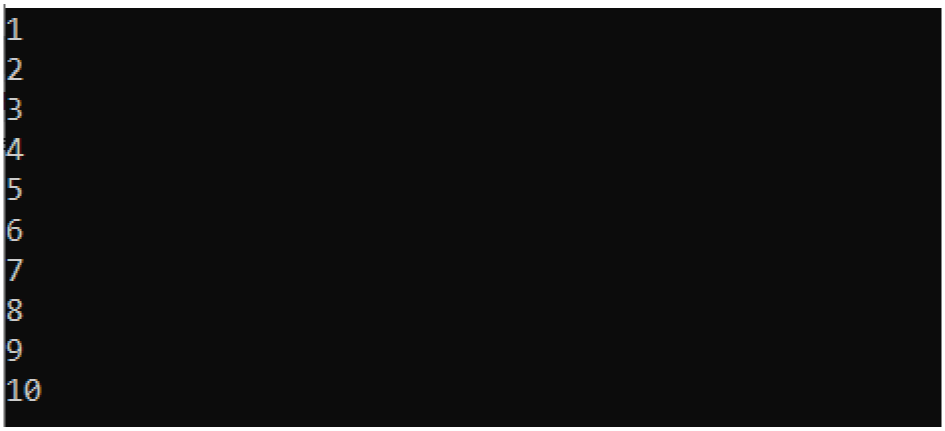
- init: The init expression is used for initializing a variable and is executed only once.
- condition: It executes the condition statement for every iteration. It executes the loop's body if it evaluates the condition as true. The loop will continue to run until the condition becomes false.
- incr/decr: The increment or decrement statement is applied to the variable to update the initial expression.

Question: Write a C++ program to display 1 to 10 using a while loop

Program

```
#include<iostream>
using namespace std;
int main()
{
    int i=1;
    while(i<=10)
    {
        cout<<i<<"\n";
        i=i+1;
    }

}
```

Output



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

Question: Attempt all programs completed using for loop statement above using a while loop

do-while loop

Unlike Like a **for loop** and **while loop**,the **do-while** executes its internal part only if the condition is valid in each execution proceeding the first one. It means the first execution is a must for the **do-while** loop. Like while loop, the **init, condition and the incr/decr** part are separate (not enclosed inside the bracket)

Syntax

```
init;
do
{
   //Statements to be executed or body
   incr/decr;
}
while(condition);
```

In the above syntax:

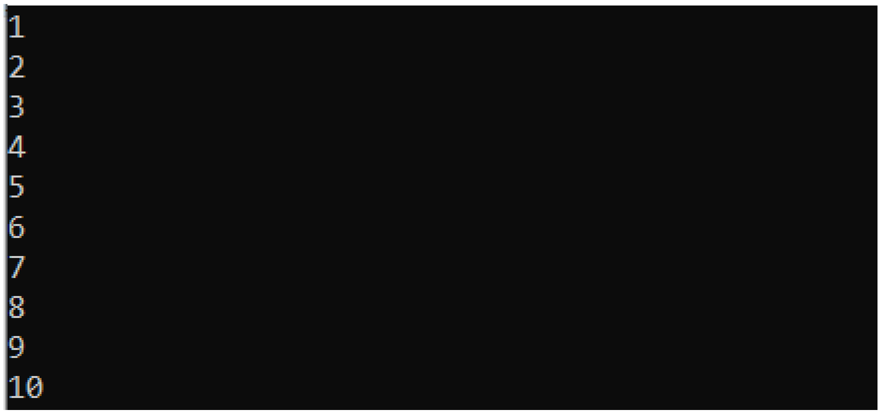
- init: The init expression is used for initializing a variable andis executed only once.
- condition: It executes the condition statement for every iteration. It executes the loop's body if it evaluates the condition as true. The loop will continue to run until the condition becomes false.
- incr/decr: The increment or decrement statement is applied to the variable to update the initial expression

Question: Write a C++ program to display 1 to 10 using a do-while loop

Program

```
#include<iostream>
using namespace std;
int main()
{
    int i=1;
    do
    {
        cout<<i<<"\n";
        i=i+1;
    }
    while(i<=10);
}
```

Output



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

Question: Attempt all programs completed using for loop statement above using a do-while loop

11 Arrays

An array is a variable that stores a collection of data with the same data type or

An array is a collection of elements with the same data type

Arrays are used to store multiple values in a single variable instead of declaring separate variables for each value.

To declare an array, define the variable type, specify the name of the array followed by **square brackets,** and specify the number of elements it should store:

The syntax for declaring an array:

Data_type   Array_name[Array_size];

Example

int A[10];

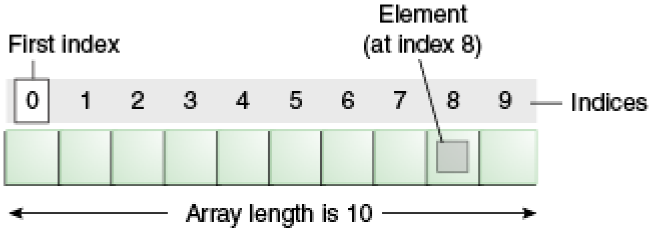
From the above declaration

int: Indicates the data type

A: Is an array name

10: Array size

The elements of an array are stored in a contiguous memory location. We can store only a fixed set of elements in an array. Array in C++ is index-based; the first element of the array is stored at the 0th index, 2nd element is stored on the 1st index, and so on. Consider a description of an array A[10] below;



The elements 10,20,30,40,50,60,70,80,90,100 of an Array "A" can be stored as follows in the computer memory.

Values	10	20	30	40	50	60	70	80	90	100
Index	0	1	2	3	4	5	6	7	8	9

Actions that can be performed to an Array "A" above

1) Storing Elements 10,20,30,40,50,60,70,80,90,100 to the array "A"

   Sample of storing elements to an Array "**A**"

   int A[10]={10,20,30,40,50,60,70,80,90,100} or

```
A[0]=10;
A[1]=20;
A[2]=30;
A[3]=40;
A[4]=50;
A[5]=60;
A[6]=70;
A[7]=80;
A[8]=90;
A[9]=100;
```

2) Storing 100 into index 1 of an array "A"

```
A[1]=100;
```

3) Retrieving 40 from Array "A"

```
cout<<A[3];
```

Question:

a) Define an array (*Refer to the definition array on the previous page*)

b) Write a C++ program to perform the following to an array named "A".

   i. Storing elements 10,20,30,40,50,60,70,80,90,100 into an array "A " as integers

   ii. Retrieving the element 30 from an array "A"

   iii. Retrieving the 3rd element from an array "A"

   iv. Updating the element 50 to 100 from an array "A"

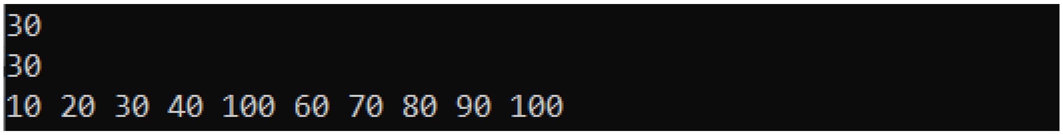
   v. Retrieving all Elements using a for loop

Program for part(b)

Program

```
#include<iostream>
using namespace std;
int main()
{
    int A[10]={10,20,30,40,50,60,70,80,90,100}; //Storing of Elements
    cout<<A[2]<<"\n"; // Retrieving Element 30
    cout<<A[2]<<"\n";  // Retrieving Element 30
    A[4]=100; //Updating 50 to 100

    //Displaying All Elements Using for Loop
    for(int i=0;i<=9;i++)
    {
        cout<< A[i]<<" ";
    }
}
```

Output



```
30
30
10 20 30 40 100 60 70 80 90 100
```

12 Functions/Methods

A C++ method/Function is a collection of statements grouped to perform a certain operation when being called in a program. For example, a section of a bank information system dealing with customers' deposit may contain several lines of code that communicate with the database, which transfer and store the desired amount in the customer's account. All program statements that perform deposits can be grouped into a function.

The syntax for creating a function

return_type method name (list of arguments)

{

    //Function Body

}

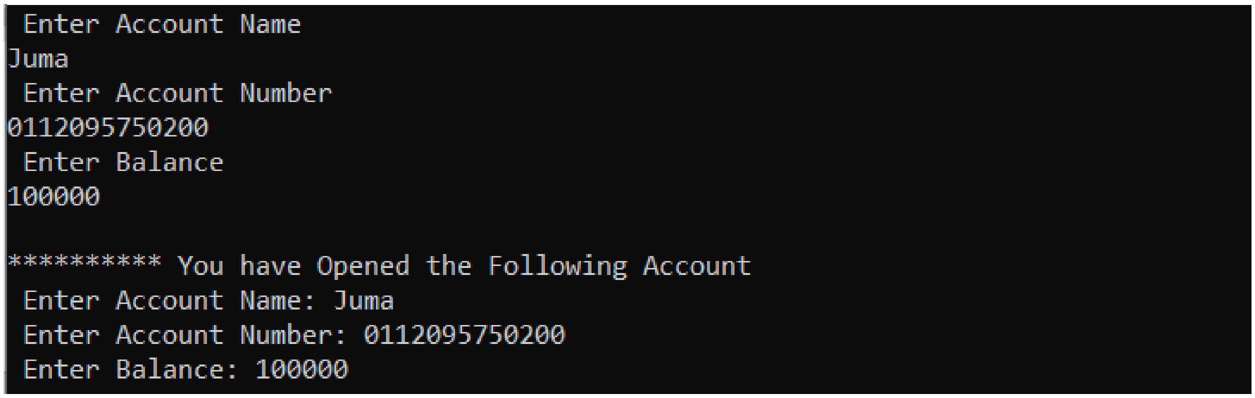
Note that: The list of arguments part is optional.

Example: A C++ program implementing a function that enables a banker to open a bank account for a particular customer and display account details on the screen.

Case 1: A program without parameters

```
#include<iostream>
using namespace std;
void openbankaccout()
{
    string accountname;
    string accountnumber;
    float balance;
    cout<<" Enter Account Name\n";
    cin>>accountname;
    cout<<" Enter Account Number\n";
    cin>>accountnumber;
    cout<<" Enter Balance\n";
    cin>>balance;

    cout<<"\n********** You have Opened the Following Account\n";
    cout<<" Enter Account Name: "<<accountname<<"\n";
    cout<<" Enter Account Number: "<<accountnumber<<"\n";
    cout<<" Enter Balance: "<< balance<<"\n";
}
int main()
{
    openbankaccout();
}
```

Output

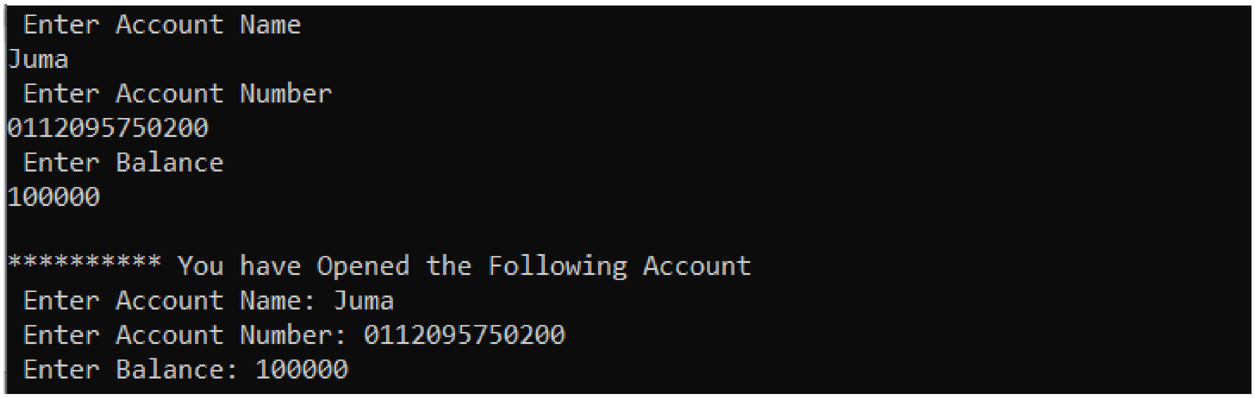


```
 Enter Account Name
Juma
 Enter Account Number
0112095750200
 Enter Balance
100000

********** You have Opened the Following Account
 Enter Account Name: Juma
 Enter Account Number: 0112095750200
 Enter Balance: 100000
```

Case 2: A program with parameters/Using message passing concept

```
#include<iostream>
using namespace std;
void openbankaccout (string a,string b, float c)
{
    cout<<"\n********** You have Opened the Following Account\n";
    cout<<" Enter Account Name: "<<a<<"\n";
    cout<<" Enter Account Number: "<<b<<"\n";
    cout<<" Enter Balance: "<< c <<"\n";
}
int main()
{
    string accountname;
    string accountnumber;
    float balance;
    cout<<" Enter Account Name\n";
    cin>>accountname;
    cout<<" Enter Account Number\n";
    cin>>accountnumber;
    cout<<" Enter Balance\n";
    cin>>balance;

    openbankaccout (accountname, accountnumber, balance);
}
```

Output



```
 Enter Account Name
Juma
 Enter Account Number
0112095750200
 Enter Balance
100000

********** You have Opened the Following Account
 Enter Account Name: Juma
 Enter Account Number: 0112095750200
 Enter Balance: 100000
```

13 C++ Classes and Objects

What is a class in C++?

A Class is an object constructor or a "blueprint" for creating objects. Or Class is a group of variables and methods of different data types.

What is an object in C++?

An object is any real-world entity that can be converted into a computer program. The object can be physical or logical (tangible and intangible). An example of an intangible object is the banking system. Examples of tangible objects include a chair, bike, marker, pen, table, and car.

Or

An object is an instance of a class

An object has three characteristics (Imagine our object is a bus)

State: These are properties of an object. For example, a bus as an object state will be colour, current speed, number of wheels, etc. In programming, object states are represented by variables.

Behaviour: These are all actions that an object can perform. For example, for a bus, object behaviour will be slow down, speed up and turn around. In programming, objects' behaviours are represented by functions.

Identity: It is a unique name for an object which differentiates an object from the rest

Imagine our object is a bank account

State: current balance, opening date, account number

Behaviour: withdraw, deposit, transfer, and check the balance

Identity: Account Name

The syntax for declaring a class

To create a class, use the class keyword

```
class class_name {
 Access specifier:
            Data_type variable 1;
            Data_type variable 2;
            ................................
            ................................

            Data_type variable n;
            Data_type function1();
            Data_type function2();
            ................................
            ................................
            Data_type function n();
}
```