① Unit 1: Principles of object oriented programming beginning with C++

→ Procedure oriented programming - object oriented programming - Basic concepts of object-oriented programming - Benefits of OOP - Applications of OOP - A simple C++ program - Structure of C++ program - C++ data types - Symbolic constants - Reference by variables - Operators in C++ - Operator precedence - Control structures - Function in C++ - The main function, Function prototyping - Call by reference - Return by reference - Inline function - Default arguments - Function overloading

② Unit 2: Classes and Objects

→ Specifying a class - Defining member functions - Nesting of member functions - Private member functions - Arrays within a class - Memory allocation for objects - Static data members - Static member functions - Arrays of objects - Objects as function arguments - Friendly functions - (Returning objects)

③ Unit 3: Constructors and destructors overloading

→ Constructors - Default constructor - Parameterized constructor - copy constructor - Multiple constructors - constructors with default arguments - Dynamic constructor - Destructors - Operator overloading - Unary and Binary operator overloading -

Overloading using friends - Rules for overloading - Type conversion.

④ Unit 4: Inheritance

→ Inheritance - Defining derived classes - Visibility modes - Single, Multilevel, Multiple, Hierarchical and Hybrid inheritance - Virtual base classes. Abstract classes - Constructors in derived classes - Nesting of classes.

⑤ Unit 5: Pointers, Virtual Functions and Polymorphism.

UNIT - I

26/02/2020

1. Explain the basic concepts of object - oriented programming.

Ans:- Some of the concepts used extensively in object - oriented programming include:-

- objects
- classes
- data abstraction and encapsulation
- inheritance
- polymorphism
- dynamic binding
- message passing

→ Objects

Objects are the basic run - time entities in an object oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle.
They may also represent user - defined data such as vectors, time and lists.
Programming problem is analyzed in terms of objects and the nature of communication between them. Program objects should be chosen such that they match closely with the real - world objects.
Objects take up space in the memory and have an associated address like a record in Pascal, or a structure in C.

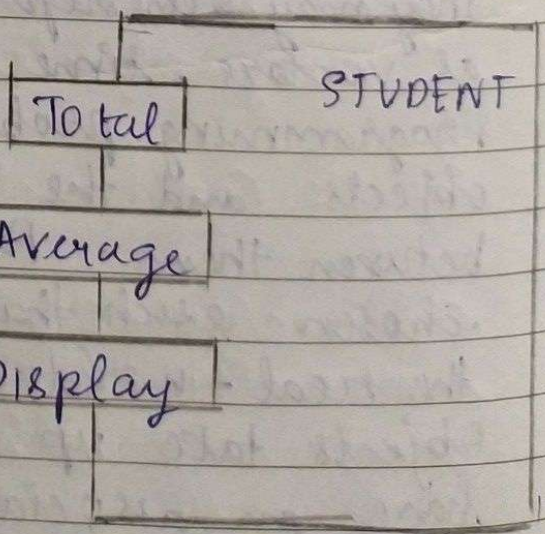
When a program is executed the objects interact by sending messages to one another. For example, if "customer" and "account" are two objects in a program, then the customer object may send a message to the account object requesting for the bank balance.

Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code.
It is sufficient to know the type of message accepted, and the type of response returned by the objects.

Although different authors represent them differently the figure shows two notations that are popularly used in object-oriented analysis and design

fig:-

Object : STUDENT
DATA Name Date-of-birth Marks .....
FUNCTIONS Total Average Display .....



→ • Classes

We know that objects contain data, and code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the help of a class.
In fact, objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created.

A class is thus a collection of objects of similar type.
Classes are user-defined types and behave like the built-in types of a programming language.

The syntax used to create an object is no different than the syntax used to create an integer object in C.

Eg:- Mango, apple and orange are members of the class fruit.

If fruit has been defined as a class then the statement `fruit mango;` will create an object 'mango' belonging to the class fruit

→• Data abstraction and encapsulation

The wrapping up of data and functions into a single unit (called class) is known as encapsulation.

Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object's data and the program. This insulation of the data from direct access by the program is called data hiding or information hiding.

Abstraction refers to the act of representing essential features without including the background details or explanations.

Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost, and functions to operate on these attributes. They encapsulate all the essential properties of the objects that are to be created. The attributes are sometimes called data members because they hold information. The functions that operate on these data are sometimes called methods or member functions.

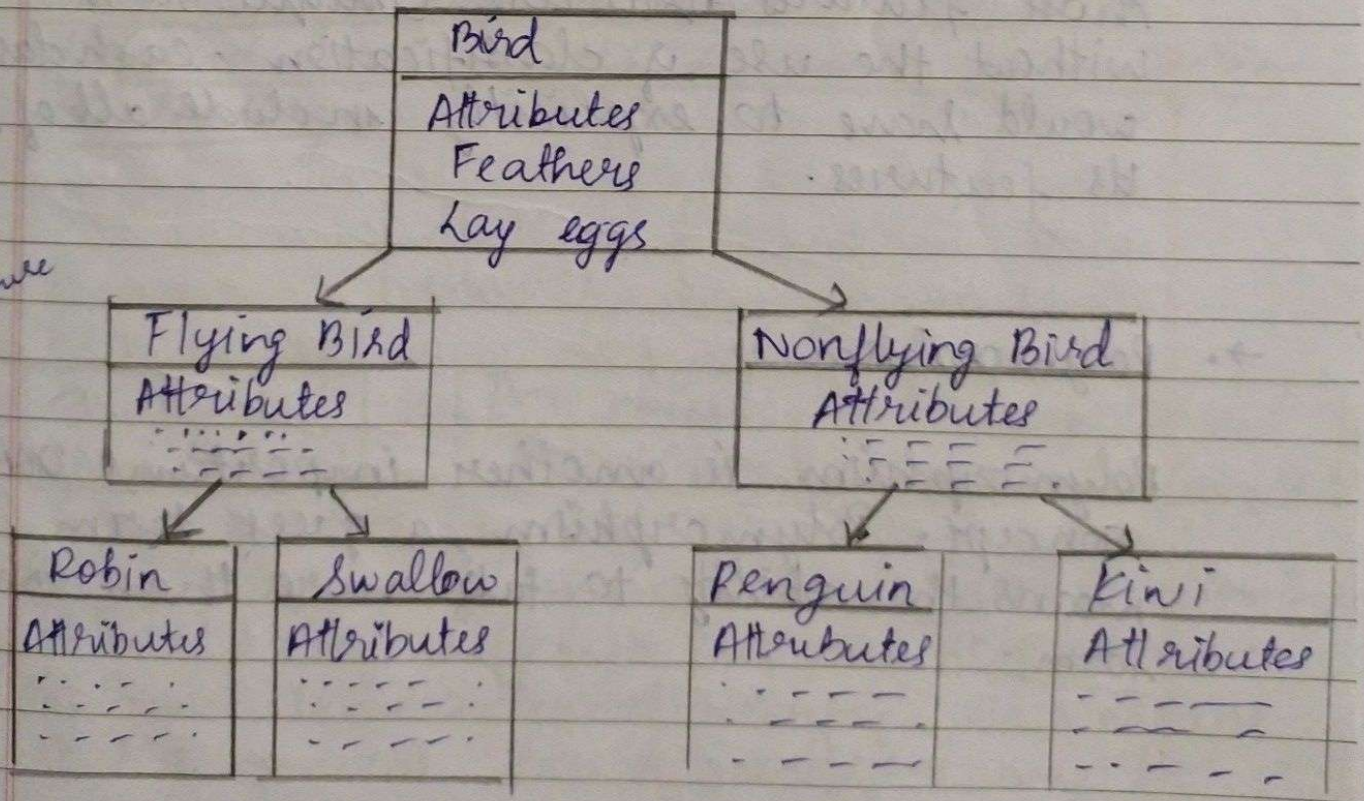
Since the classes use of concept of data abstraction they are known as Abstract Data Types (ADT)

→ Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class.
It supports the concept of hierarchical classification.

For example, the bird 'robin' is a part of the class 'flying bird' which is again a part of the class 'bird'. The principle behind this sort of division is that each derived class shares common characteristics with the class from which it is derived as illustrated in figure.

figure



In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it.

This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes. The real appeal and power of the inheritance mechanism is that it allows the programmer to reuse a class that is almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduce any undesirable side-effects into the rest of the classes.

Note that each sub-classes defines only those features that are unique to it. Without the use of classification, each class would have to explicitly include all of its features.

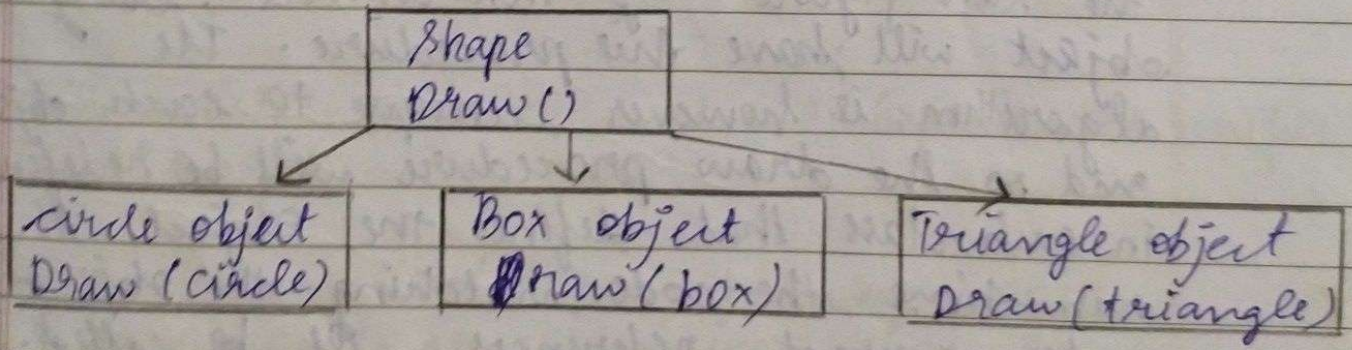
→. Polymorphism

Polymorphism is another important OOP concept. Polymorphism, a greek term, means the ability to take more than one form.

An operation may exhibit different behaviors in different instances. The behavior depends upon the types of data used in the operation.

For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation. The process of making an operator to exhibit different behaviors in different instances is known as operation overloading.

Figure illustrates that a single function name can be used to handle different number and different types of arguments. This is something similar to a particular word having several different meanings depending on the context. Using a single function name to perform different types of tasks is known as function overloading



Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. This means that a general class of operations may be

accessed in the same manner even though specific actions associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.

→ Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding (also known as late binding) means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.

A function call associated with a polymorphic reference depends on the dynamic type of that reference.

Consider the ~~produce~~ procedure "draw" in the last figure. By inheritance, every object will have this procedure. Its algorithm is however, unique to each object and so the draw procedure will be redefined in each class that defines the object. At run-time, the code matching the object under current references will be called.

7. Message Passing

An object - oriented program consists of set of objects that communicate with each other. The process of programming in an object-oriented language, therefore, involves the following basic steps:-

1. Creating classes that define objects and their behavior
2. Creating objects from class definitions and
3. Establishing communication among objects.

Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another. The concept of message passing makes it easier to talk about building systems that directly model or stimulate their real-world counterparts.

2. Differentiate between POP and OOP. Explain the benifits and applications of OOP.

Ans:- Procedure oriented programming (POP) is a conventional programming using high level languages such as COBOL, FORTRAN and C.

Object oriented programming (OOP) is defined as an approach that provides a way of

modulazing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

Procedure oriented programming (POP)	Object-oriented programming (OOP)
1. • Emphasis is on doing things (algorithms)	Emphasis is on data rather than procedure
2. • Large programs are divided into smaller programs known as functions	Programs are divided into what are known as objects
• Most of the functions share global data.	Data structures are designed such that they characterize the objects
• Data move openly around the system from function to function.	Data is hidden and cannot be accessed by external functions.
• Functions transform data from one form to another	Objects may communicate with each other through functions
• Employs top-down approach in program design.	Follows bottom-up approach in

program design

Functions that operate on the data of an object are tied together in the data structure.

New data and functions can be easily added whenever necessary

Benefits of OOP

OOP offers several benefits to both the program designer and the user.
Object-orientation contributes to the solution of many problems associated with the development and quality of software products

The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost.

The principal advantages are:-

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.

- We can build programs from the standard working modules that communicate with

one another, rather than having to write the code from scratch. This leads to saving of development time and higher productivity.

- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map objects in the problem domain to those in the program.
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more details of a model in implementable form.
- Object-oriented system can be easily upgraded from small to large systems.
- Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

Applications of OOP

The most popular application of object-oriented programming, up to now, has been in the area of user interface design such as windows.

OOP is useful in real-business system types of application because it can simplify a complex problem.

The application of OOP include:-

- Real-time systems
- Stimulation and modeling
- Object-oriented databases
- Hypertext, hypermedia and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems.

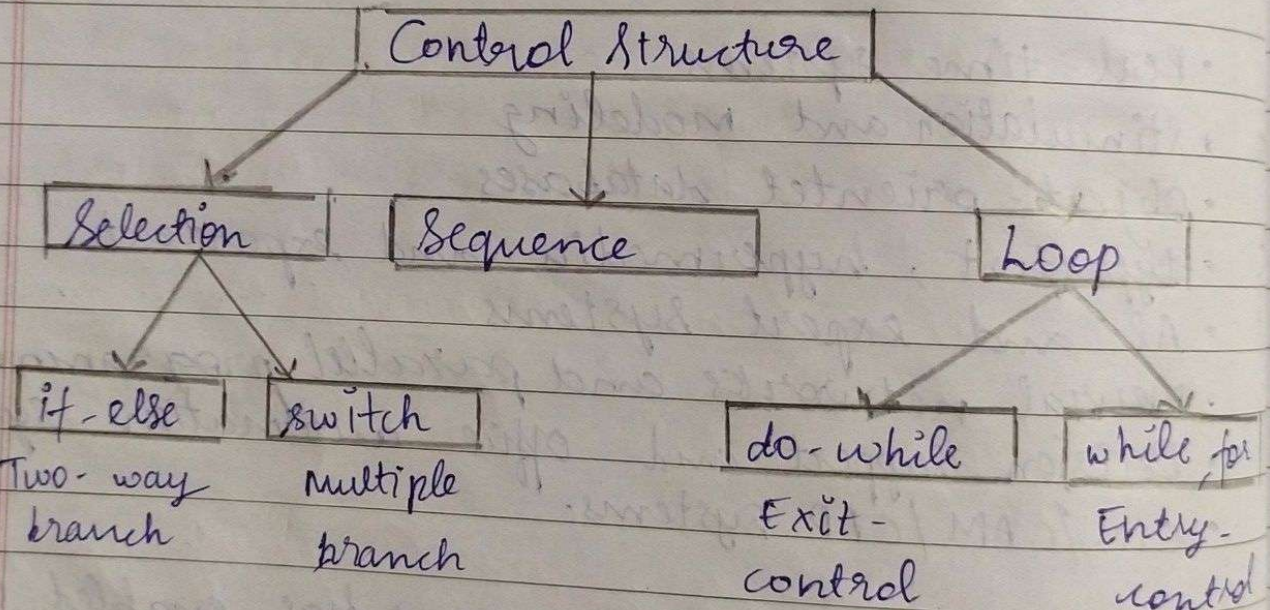
The richness of OOP environment has enabled the software industry to improve not only the quality of software system but also its productivity.

Object-Oriented ~~per~~ technology is certainly changing the way the software engineers think, analyze, design and implement systems.

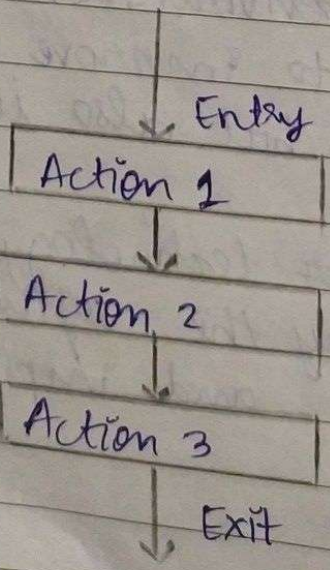
3. What are control structures? List and explain each of them with code segments.

Ans:- The method of achieving the objective of an accurate, error-resistant and maintainable code is to use one or any combination of the following three control structures:-

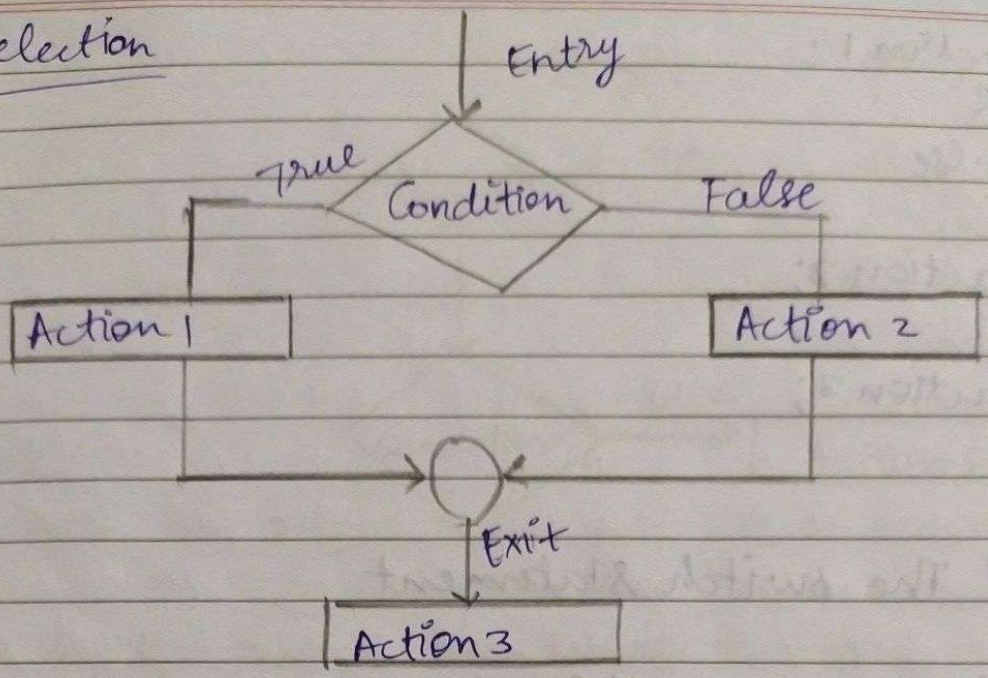
1. Sequence structure (straight line)
2. Selection structure (branching)
3. Loop structure (iteration or repetition)



1) Sequence



2) Selection



(a) If Statement

The if statement is implemented in two forms:-

- Simple if statement
- If... else statement

syntax of simple if statement

```
if (expression is true)
{
action 1;
}
action 2;
action 3;
```

Syntax of if.. else statement

```
if( expression is true)
{
```

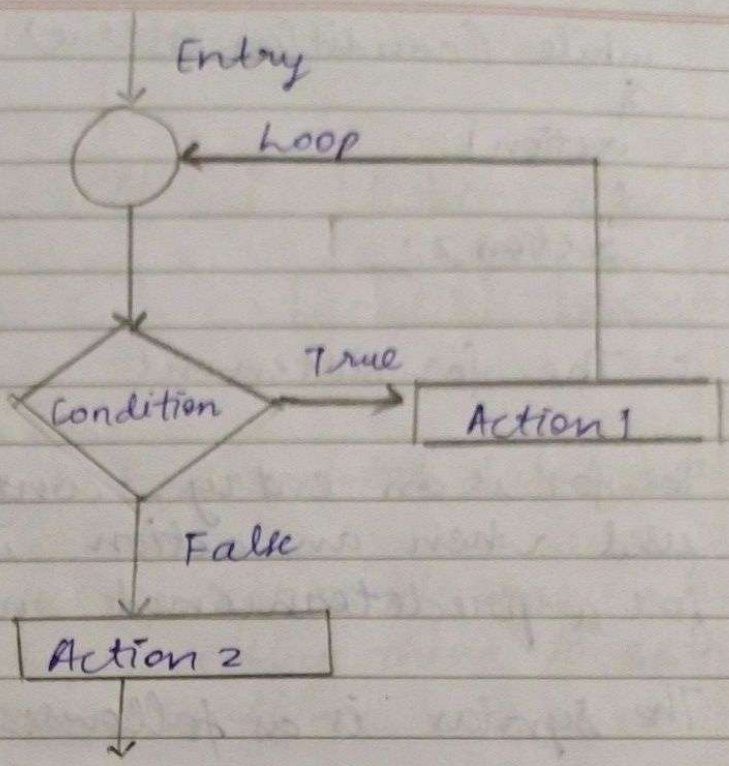
```
action 1;
}
else
{
action 2;
}
action 3;
```

(b) The switch statement

This is a multiple-branching statement where, based on a condition, the control is transferred to one of the many possible points. This is implemented as follows:-

```
switch (expression)
{
case 1:
{
 action 1;
}
case 2;
{
action 2;
}
case 3:
{
action 3;
}
default:
{ action 4;
}
```

3) Loop



(a) The do-while statement

The do-while is an exit-controlled loop. Based on a condition, the control is transferred back to a particular point in the program. The syntax is as follows:-

```
do
{ action;
}
while (condition is true);
action 2;
```

(b) The while statement

This is also a loop structure, but is an entry-controlled one. The syntax is as follows:-

```
while (condition is true)
{
action 1;
}
action 2;
```

(c) The for statement

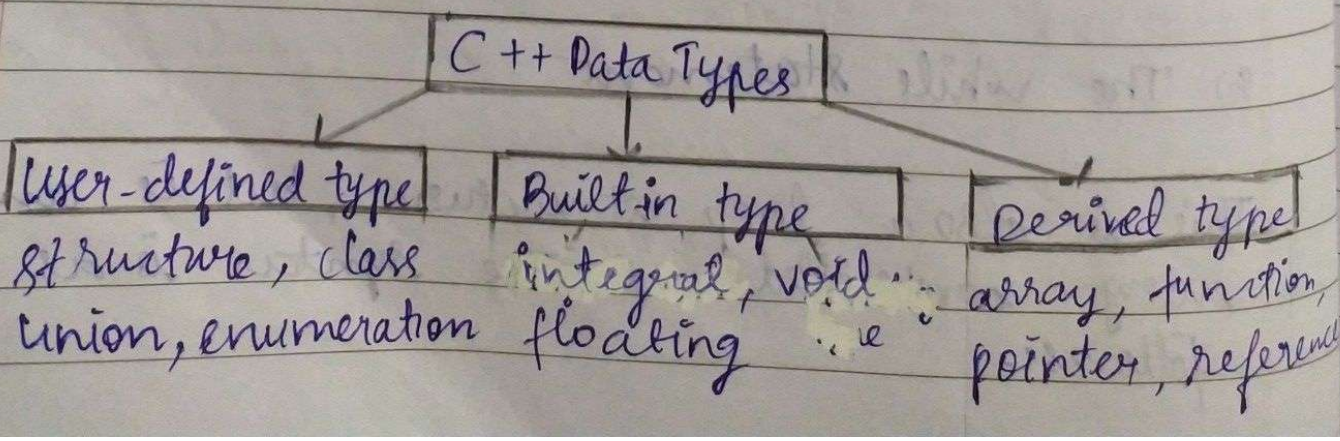
The for is an entry - controlled loop and is used when an action is to be repeated for a predetermined number of times.

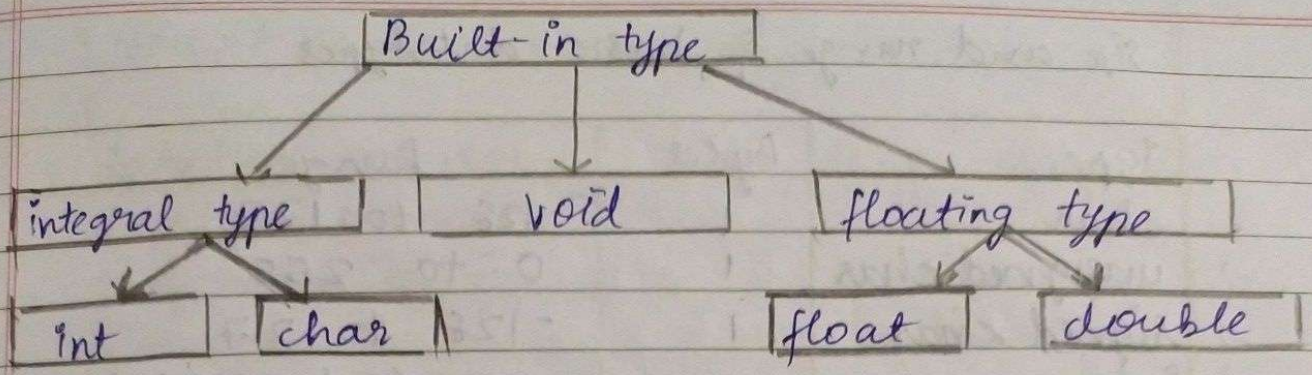
The syntax is as follows:-

```
for (initial value; test; increment)
{
action 1;
}
action 2;
```

4. Explain different datatypes and operators used in C++ with examples.

ns:- Data types in C++ can be classified under various categories as shown:-





(a) Built-in type (Basic or Fundamental datatype)

The basic data types except 'void' may have several modifiers preceding them to serve the needs of various situations.

The modifiers signed, unsigned, long and short may be applied to character and integer basic data types. The modifier long may also be applied to double.

The two normal uses of void are:-
- to specify the return type of function when it is not returning any value.
- to indicate an empty argument list to a function. eg:- void funct1 (void);
- To declaration of generic pointers. eg:- void *gp;

A generic pointer can be assigned a pointer value of any basic data type, but it may not be deferenced

eg:- int *ip; gp = ip; are valid statements but *ip = *gp is illegal. It would not make sense to deference a pointer to a void value.

Size and range of basic datatypes

Type	Bytes	Range
char	1	-128 to 127
unsigned char	1	0 to 255
signed char	1	-128 to 127
int	2	-32768 to 32767
unsigned int	2	0 to 65535
signed int	2	-32768 to 32767
short int	2	-32768 to 32767
unsigned short int	2	0 to 65535
signed short int	2	-32768 to 32767
long int	4	-2147483648 to 2147483647
signed long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

(5) User-defined data type

Structures and Unions

Structures are used to group together ~~similar type data~~ elements with dissimilar types.

The Syntax of structure:-

```
struct name
{
datatype member 1;
datatype member 2;
. . . . . .
. . . . . .
. . . . . .
};
```

Unions are conceptually similar to structures as they allow us to group ~~together~~ together dissimilar-type elements inside a single unit. But there are significant difference between structures and unions as far as their implementation is concerned.
The size of structure is equal to the sum of the sizes of individual member types.
The size of union is equal to the size of its largest member element.

eg:-
The ~~syntax~~ of union:-

```
union result
{
int marks;
char grade;
float percent;
};
```

classes

Classes are user-defined datatype which can be used to declare variables. The

class variables are known as objects.

Enumerated Data Type

An enumerated data type is another user-defined type which provides a way for attaching names to numbers thereby increasing comprehensibility of the code. The enum keyword automatically enumerates a list of words by assigning them values 0, 1, 2 and so on.

(c) Derived data type

Arrays

Arrays are used to group together similar data type elements.

The syntax is:-

datatype array name (size);

When initialising a character array in ANSI C, the compiler will allow us to declare the array size as the exact length of the string constant.

eg:- char string[3] = "xyz"; But in C++ the size should be one larger than the number of characters in the string eg:- char string[4] = "xyz";

Functions

Functions have undergone major changes in C++. While some of these changes are simple

others require a new way of thinking when organising our programs. Many of these modifications and improvements were driven by the requirements of object oriented concept of C++.

Pointers

They are declared and initialised as in C

eg:- int* ip; ip = &x; *ip = x;

C++ adds the concepts of constant pointer and pointer to a constant.

char const ptr1 = "GOOD";

We cannot modify the address that ptr1 is initialized to

Pointers are extensively used in C++ for memory management and achieving polymorphism.

Operators in C++

Some new operators in C++ are

<< insertion operator
>> extraction operator
:: scope resolution operator
::* Pointer to member declarator
->* or .* Pointer to member operator
delete Memory release operator
endl Line feed operator
new memory allocation operator
setw Field width operator

Operators in C++

An operator is a symbol that tells the computer to perform certain mathematical and logical manipulation. Operators are used in programs to manipulate data and variables. C++ operators can be classified into the number 3 categories:-

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and decrement operators
6. Conditional operators
7. Bitwise operator
8. Special operator

① Arithmetic operators

C++ provides all the basic arithmetic operators

Operators	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo Division

Integer division truncates any functional part. The modulo division operator produces the remainder of an integer division

Integer arithmetic:- when both the operands in a single arithmetic expression such as a+b are integers the expression is called an integer value.

Real arithmetic:- an arithmetic operation involving only real operand may assume values either in decimal or exponential notation.

Mixed mode arithmetic:- when one of the operand is real and the other ~~the or~~ is integer.

② Relational Operator

We often compare two quantities and depending on their relation, takes certain decision. An expression such as A<B or 1<20 containing a relational operation is turned as a relational expression. The value of a relational expression is either 1 or 0.
It is 1 if the specified relation is true and zero the relation is false

C supports 6 relational operators

Operator	meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to.

eg:- 4.5 <= 10 T
4.5 <= -10 F

Relational operator compliments
> is a complement of <=
< is a complement of >=
== is a complement of !=

③ Logical operators

In addition to the relational operators, C has the following three logical operators

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

The logical operators AND and OR are used when we want to test more than one condition and make decision

eg:- a > b && x == 10

OP-1	OP-2	Value of expression OP-1 && OP2	OP1 || OP2
Nonzero	Nonzero	one	one
Nonzero	zero	zero	one
Zero	Nonzero	zero	one
Zero	zero	zero	zero

④ Assignment Operator

Assignment Operators are used to assignment the result of an expression to a variable. C has a set of short hand assignment operators of the form

v op = exp;

where v is a variable exp is the expression and op is the binary arithmetic operation

The operator op = is known as the short hand assignment operator.

v op = exp;

is equalent to v = v op (exp);

x + = y + 1; = x = x + y + 1

Statement with simple assignment operator	Statement with short hand Operator
a = a + 1	a + = 1
a = a - 1	a - = 1
a = a * (n + 1)	a * = (n + 1)
a = a / (n + 1)	a / = (n + 1)
a = a % b	a % = b

⑤ Increment and decrement operators

C allows two very useful operators not generally formed in other language. These are the increment and decrement

operators ++ and -- . The operator ++ adds 1 to the operands while -- subtracts 1. Both are unary operators and takes the following form:-

++m ; or m++ ;
--m ; or m--;

⑤ Conditional Operator

A ternary operator pair "? :" is available in C to construct the conditional expression of the form:

Exp 1 ? Exp 2 : exp 3
where exp 1, exp 2 and exp 3 are expression

The operator ? : works are follows:-
exp 1 is evaluated first if it is non-zero (true), then the expression exp 2 is evaluated and becomes the value of the expression. If exp 1 is false, exp 3 is evaluated and its value becomes the value of the expression.

eg:- a = 10;
b = 15;
x = (a > b) ? a : b;
x will be assigned the value of b
i.e x = 15;

④ Bitwise operators

C has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level. These operators are used for testing the bits, or changing shifting them right or left

operators	meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

⑤ Special operators

C supports some special operators such as comma operator, size of operater pointer operator and * and member selection operaters.

Comma operator

The comma operator can be used to link the related expressions together. A comma linked list of expressions are evaluated left to right and the value of the right most expression is the value of the combined

expression

size of operator

The size of is a compile time operater and when used with an operand, it returns the number of bits the operand occupies. The operands may be a variable, the constant or a datatype qualifiers.

eg:- m = size of (sum)*;
n = size of (long int);
x = size of (235L);

some new operators in C++ are.
<< insertion operater
>> extraction operater

5. What is user defined function? Explain its components and operators.

Ans:- A user defined function is a programmed routine that has its parameters set by the user of the system. User defined functions often are seen as programming shortcuts as they define functions that perform specific tasks within a larger system such as database or spreadsheet program

User defined or spreadsheet program user defined functions can be categorised as:-

- Function with no argument and no return value
- Function with no argument ~~and~~ but return value
- Function with argument but no return value
- Function with argument and return value.

For eg:- consider a situation to check prime number. This can be done by user defined function in 4 different methods as mentioned above

eg:- No arguments passed and no return value

```
#include <iostream.h>
    void prime();
    int main()
    {
    prime(); return 0;
    }
```

```
void prime ()
{
 int num, i, flag = 0;
cout << " Enter a positive number ";
 cin >> num;
for (i = 2; i <= num/2 ; i++)
 {
 if (num % i == 0)
 {
 flag = 1;
 break;
 }}
if (flag == 1)
{
cout << num << " not prime ";
}
else
{ cout << num << " is a prime ";
}}
```

eg2: NO arguments passed but a return value

```
#include <iostream.h>
 int prime();
 int main()
{
int num, i, flag=0;
 num = prime();
for (i=2; i <= num/2; i++)
{
if (num % i == 0)
{ flag = 1;
```

```
break;
}}
if (flag == 1)
{ cout << num << " is not prime";
} else {
cout << num << " is prime";
}
return 0; }
int prime ()
{
int n.
cout << "Input number "; cin >> n;
return n;
}
```

9) Arguments passed but no return value

```
#include <iostream.h>
void prime (int n)
int main()
{
int num;
cout << "Enter a number.";
cin >> num;
prime (n);
return 0; }
void prime (int n)
{ int i, flag = 0;
for (i = 2; i <= n/2; i++)
{ if (n % i == 0)
{ flag = 1; break;
```

```
}
}
if (flag == 1)
{ cout << n << " is not a prime";
}
else {
cout << n << " is prime";
}}
```

eg 4:- Arguments passed and a return value

```
#include <iostream.h>
int prime (int n);
 int main()
{
int num, flag = 0;
cout << "Enter a number";
 cin >> num;
flag = prime (num);
if (flag == 1)
cout << num << " is not prime";
else
 cout << num << " is prime";
return 0;
}
 int prime (int n)
 { int i;
for (i = 2; i <= n/2; i++)
{ if (n%i == 0)
 return 1; }
 return 0; }
```