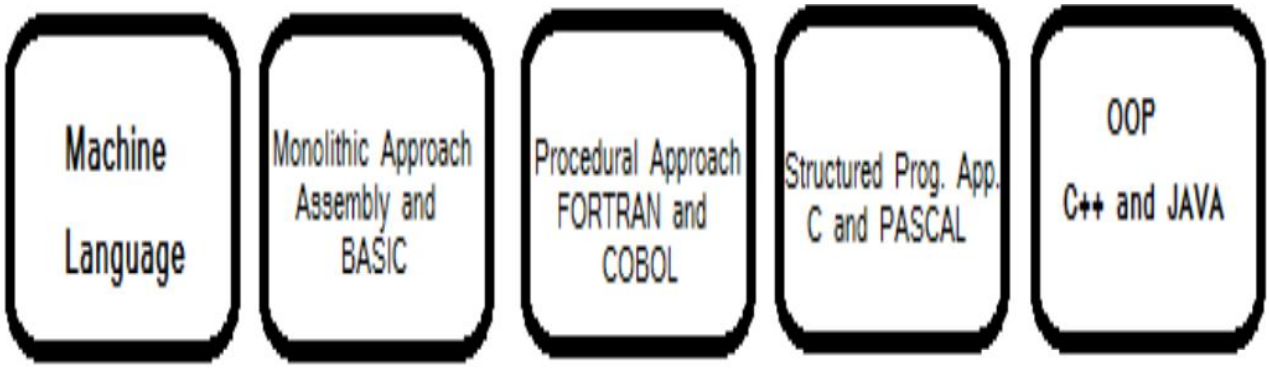
OBJECT ORIENTED PROGRAMMING THROUGH C++

UNIT-1

Evolution of oops:

- Initially for designing small and simple programs, the machine language was used.
- Next came the Assembly Language which was used for designing larger programs.
- Both machine and Assembly languages are machine dependent. Next came Procedural Programming Approach which enabled us to write larger and hundred lines of code.
- Then in 1970, a new programming approach called Structured Programming Approach was developed for designing medium sized programs.
- In 1980's the size of programs kept increasing so a new approach known as OOP was invented.



Object Oriented Programming:

- Oops was introduced to overcome the flaws of procedural programming language such as lack of reusability and maintainability.
- Fundamental idea behind oops language is to combine data and functions that operate on that data into a single unit and that unit is known as object.
- In other words, Object oriented programming is a way of solving complex problems by breaking them into smaller problems using objects.
- In Object oriented programming we write programs using classes and objects utilizing features of OOPs such as abstraction, encapsulation, inheritance and polymorphism.
- In oops problem is divided into number of entities called objects and then build data and functions around these objects.
- It ties data more closely to the functions and avoid any modification.
- Data of an object can be accessed by that function only associated with that object.
- Communication among objects is done through objects.

Advantages of OOPs:

- Emphasis on data rather than procedure.
- Programs are divided into entities known as **objects**.
- Data Structures are designed such that they characterize objects.
- Functions that operate on data of an object are tied together in data structures.
- Data is hidden and cannot be accessed by external functions.
- Objects communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follows bottom up design in program design.

Comparison between Functional programming and oops:

Functional Programming	Object Oriented Programming
This programming paradigm emphasizes on the use of functions where each function performs a specific task.	This programming paradigm is based on object-oriented concept. Classes are used where instance of objects are created
Fundamental elements used are variables and functions. The data in the functions are immutable (cannot be changed after creation).	Fundamental elements used are objects and methods and the data used here are mutable data.
Importance is not given to data but to functions.	Importance is given to data rather than procedures.
It follows declarative programming model.	It follows imperative programming model.
It uses recursion for iteration.	It uses loops for iteration.
It is parallel programming supported.	It does not support parallel programming.
The statements in this programming paradigm does not need to follow a particular order while execution.	The statements in this programming paradigm need to follow an order i.e., bottom-up approach while execution.
Does not have any access specifier.	Has three access specifiers namely, Public, Private and Protected.
To add new data and functions is not so easy.	Provides an easy way to add new data and functions.

Characteristics of oops:

1. Objects:

An Object is an identifiable entity with some characteristics and behaviour. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e., an object is created) memory is allocated.

2. Class:

The building block of C++ that leads to Object-Oriented programming is a Class. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

A Class is a user-defined data-type which has data members and member functions.

3. Encapsulation:

Encapsulation is a process of combining data and function into a single unit like capsule. In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them. This is to avoid the access of private data members from outside the class. To achieve encapsulation, we make all data members of class private and create public functions, using them we can get the values from these data members or set the value to these data members.

4. Abstraction:

- Data abstraction is one of the most essential and important features of object-oriented programming in C++.
- Abstraction means displaying only essential information and hiding the details.
- Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

5. Polymorphism:

The word polymorphism is a Greek term which means having more than one form.

In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Operator Overloading:

The process of making an operator to exhibit different behaviours in different instances is known as operator overloading.

Function Overloading:

Function overloading is using a single function name to perform different types of tasks.

Polymorphism is extensively used in implementing inheritance.

6. Inheritance:

- The capability of a class to derive properties and characteristics from another class is called Inheritance.
- Inheritance is one of the most important features of Object-Oriented Programming.
- **Sub Class:** The class that inherits properties from another class is called Sub class or Derived Class.
- **Super Class:** The class whose properties are inherited by sub class is called Base Class or Super class.

7. Reusability:

Inheritance supports the concept of "reusability", i.e., when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

8. Dynamic Binding:

In dynamic binding, the code to be executed in response to function call is decided at runtime. C++ has virtual functions to support this.

9. Message Passing:

Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

User-Defined Datatypes:

The data types that are defined by the user are called the derived datatype or user-defined derived data type.
These types include:

1. Class
2. Structure
3. Union
4. Enumeration
5. Typedef defined Datatype

1. Class:

The building block of C++ that leads to Object-Oriented programming is a **Class**. It is a user-defined data type, which holds its own data members and member functions, which can be

accessed and used by creating an instance of that class. A class is like a blueprint for an object.

2. Structure:

A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.

3. Union:

Like Structures, union is a user defined data type. In union, all members share the same memory location. For example, in the following C program, both x and y share the same location. If we change x, we can see the changes being reflected in y.

4. Enumeration:

Enumeration (or Enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

5. Typedef:

C++ allows you to define explicitly new data type names by using the keyword typedef. Using typedef does not actually create a new data class, rather it defines a name for an existing type. This can increase the portability (the ability of a program to be used across different types of machines; i.e., mini, mainframe, micro, etc.; without much changes into the code) of a program as only the typedef statements would have to be changed. Using typedef one can also aid in self-documenting code by allowing descriptive names for the standard data types.

INTRODUCTION TO C++:

C++ Tokens:

A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

1. Keywords
2. Identifiers
3. Constants
4. Strings
5. Operators

1. Keywords:

Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed. You cannot redefine keywords. However, you can specify the text to be substituted for keywords before compilation by using C/C++ preprocessor directives.

While in C++ there are 31 additional keywords other than C Keywords they are:

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

2. Identifiers:

Identifiers are used as the general terminology for the naming of variables, functions and arrays. These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore (_) as a first character. Identifier names must differ in spelling and case from any keywords. You cannot use keywords as identifiers; they are reserved for special use. Once declared, you can use the identifier in later program statements to refer to the associated value. A special kind of identifier, called a statement

label, can be used in goto statements.
There are certain rules that should be followed while naming c identifiers:

- They must begin with a letter or underscore(_).
- They must consist of only letters, digits, or underscore. No other special character is allowed.
- It should not be a keyword.
- It must not contain white space.
- It should be up to 31 characters long as only the first 31 characters are significant.

3. Constants:

Constants are also like normal variables. But, the only difference is, their values can not be modified by the program once they are defined. Constants refer to fixed values. They are also called literals.
Constants may belong to any of the data type

Syntax:
const data_type variable_name; (or) const data_type *variable_name;

Types of Constants:

1. **Integer constants –** Example: 0, 1, 1218, 12482
2. **Real or Floating-point constants –** Example: 0.0, 1203.03, 30486.184
3. **Octal & Hexadecimal constants –** Example: octal: (013)8 = (11)10, Hexadecimal: (013)16 = (19)10
4. **Character constants** -Example: ‘a’, ‘A’, ‘z’
5. **String constants** -Example: “GeeksforGeeks”

4. Strings:

Strings are nothing but an array of characters ended with a null character (‘\0’). This null character indicates the end of the string. Strings are always enclosed in double-quotes. Whereas, a character is enclosed in single quotes in C and C++.Declarations for String:

when we declare char as “string[20]”, 20 bytes of memory space is allocated for holding the string value.

When we declare char as “string[]”, memory space will be allocated as per the requirement during the execution of the program.

5. Special Symbols:

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose.[] () {}, ; * = #

Brackets[]: Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.

Parentheses(): These special symbols are used to indicate function calls and function parameters.

Braces{}: These opening and ending curly braces mark the start and end of a block of code containing more than one executable statement.

Comma (,): It is used to separate more than one statements like for separating parameters in function calls.

Colon(:): It is an operator that essentially invokes something called an initialization list.

Semicolon(;): It is known as a statement terminator. It indicates the end of one logical entity. That's why each individual statement must be ended with a semicolon.

Asterisk (*): It is used to create a pointer variable and for the multiplication of variables.

Assignment operator(=): It is used to assign values and for the logical operation validation.

Pre-processor (#): The preprocessor is a macro processor that is used automatically by the compiler to transform your program before actual compilation.

6. Operators:

Operators are symbols that trigger an action when applied to C variables and other objects. The data items on which operators act upon are called operands.
Depending on the number of operands that an operator can act upon, operators can be classified as follows:

1. Unary Operators:

Those operators that require only a single operand to act upon are known as unary operators. For Example, increment and decrement operators

2. Binary Operators:

Those operators that require two operands to act upon are called binary operators. Binary operators are classified into:

1. Arithmetic operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Bitwise Operator

3. Ternary Operator:

The operator that require three operands to act upon are called ternary operator. Conditional Operator(?) is also called ternary operator.
Syntax: (Expression1)? expression2: expression3;

C++ Data Types

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

Data Types in C++ are Mainly Divided into 3 Types:

1. Primitive Data Types:

These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char, float, bool, etc. Primitive data types available in C++ are:

1. **Integer:** The keyword used for integer data types is int. Integers typically require 4 bytes of memory space and range from -2147483648 to 2147483647.
2. **Character:** Character data type is used for storing characters. The keyword used for the character data type is char. Characters typically require 1 byte of memory space and range from -128 to 127 or 0 to 255.
3. **Boolean:** Boolean data type is used for storing Boolean or logical values. A Boolean variable can store either true or false. The keyword used for the Boolean data type is bool.
4. **Floating Point:** Floating Point data type is used for storing single-precision floating-point values or decimal values. The keyword used for the floating-point data type is float. Float variables typically require 4 bytes of memory space.
5. **Double Floating Point:** Double Floating Point data type is used for storing double-precision floating-point values or decimal values. The keyword used for the double floating-point data type is double. Double variables typically require 8 bytes of memory space.
6. **void:** Void means without any value. void data type represents a valueless entity. A void data type is used for those function which does not return a value.

2. Derived Data Types:

Derived data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:

1. Function
2. Array
3. Pointer
4. Reference

3. Abstract or User-Defined Data Types:

Abstract or User-Defined data types are defined by the user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:

1. Class
2. Structure
3. Union
4. Enumeration
5. Typedef defined Datatype

Type Conversion in C++

Type conversion is the process that converts the predefined data type of one variable into an appropriate data type. The main idea behind type conversion is to convert two different data type variables into a single data type to solve mathematical and logical expressions easily without any data loss.

For example, we are adding two numbers, where one variable is of int type and another of float type; we need to convert or typecast the int variable into a float to make them both float data types to add them.

Type conversion can be done in two ways in C++, one is implicit type conversion, and the second is explicit type conversion. Those conversions are done by the compiler itself, called the implicit type or automatic type conversion. The conversion, which is done by the user or requires user interferences called the explicit or user define type conversion.

Implicit Type Conversion

The implicit type conversion is the type of conversion done automatically by the compiler without any human effort. It means an implicit conversion automatically converts one data type into another type based on some predefined rules of the C++ compiler. Hence, it is also known as the automatic type conversion.

Order of the typecast in implicit conversion

The following is the correct order of data types from lower rank to higher rank:

bool -> char -> short int -> int -> unsigned int -> long int -> unsigned long int -> long long int -> float -> double -> long double

Explicit type conversion

Conversions that require user intervention to change the data type of one variable to another, is called the explicit type conversion. In other words, an explicit conversion allows the programmer to manually changes or typecasts the data type from one variable to another type.

Hence, it is also known as typecasting. Generally, we force the explicit type conversion to convert data from one type to another because it does not follow the implicit conversion rule.

The explicit type conversion is divided into two ways:

1. Explicit conversion using the cast operator
2. Explicit conversion using the assignment operator

Variables in C++

- Variable is a name given to a memory location. It is the basic unit of storage in a program.
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In C++, all the variables must be declared before use.

Rules For Declaring Variable

- The name of the variable contains letters, digits, and underscores.
- The name of the variable is case sensitive (ex Arr and arr both are different variables).
- The name of the variable does not contain any whitespace and special characters (ex #,$,%,*, etc).
- All the variable names must begin with a letter of the alphabet or an underscore(_).
- We cannot used C++ keyword(ex-float, double, class) as a variable name.

Variable Declaration:

A typical variable declaration is of the form:

```
// Declaring a single variable
type variable_name;

// Declaring multiple variables:
type variable1_name, variable2_name, variable3_name;
```

Reference Variable:

- When a variable is declared as a reference, it becomes an alternative name for an existing variable. A variable can be declared as a reference by putting ‘&’ in the declaration.
- Reference variable cannot be updated.
- Reference variable is an internal pointer.
- Declaration of a Reference variable is preceded with the ‘&’ symbol (but do not read it as “address of”).

Array:

- It is a group of variables of similar data types referred to by a single element.
- Its elements are stored in a contiguous memory location.
- The size of the array should be mentioned while declaring it.
- Array elements are always counted from zero (0) onward.
- Array elements can be accessed using the position of the element in the array.
- The array can have one or more dimensions.

An array in C/C++ or be it in any programming language is a collection of similar data items stored at contiguous memory locations and elements can be accessed randomly using indices of an array. They can be used to store the collection of primitive data types such as int, float, double, char, etc. of any particular type. To add to it, an array in C/C++ can store derived data types such as structures, pointers etc. Given below is the picture representation of an array.

Advantages of an Array in C/C++:

- Random access of elements using the array index.
- Use of fewer lines of code as it creates a single array of multiple elements.
- Easy access to all the elements.
- Traversal through the array becomes easy using a single loop.
- Sorting becomes easy as it can be accomplished by writing fewer lines of code.

Disadvantages of an Array in C/C++:

- Allows a fixed number of elements to be entered which is decided at the time of declaration. Unlike a linked list, an array in C is not dynamic.
- Insertion and deletion of elements can be costly since the elements are needed to be managed in accordance with the new memory allocation.

C++ Expression:

C++ expression consists of operators, constants, and variables which are arranged according to the rules of the language. It can also contain function calls which return values. An expression can consist of one or more operands, zero or more operators to compute a value. Every expression produces some value which is assigned to the variable with the help of an assignment operator.

Types of Expression:

1. Constant expressions

A constant expression is an expression that consists of only constant values. It is an expression whose value is determined at the compile-time but evaluated at the run-time. It can be composed of integer, character, floating-point, and enumeration constants.

2. Integral Expressions

An integer expression is an expression that produces the integer value as output after performing all the explicit and implicit conversions.

3. Float Expressions

A float expression is an expression that produces floating-point value as output after performing all the explicit and implicit conversions.

4. Pointer Expressions

A pointer expression is an expression that produces address value as an output.

5. Relational Expressions

A relational expression is an expression that produces a value of type bool, which can be either true or false. It is also known as a Boolean expression. When arithmetic expressions are used on both sides of the relational operator, arithmetic expressions are evaluated first, and then their results are compared.

6. Logical Expressions

A logical expression is an expression that combines two or more relational expressions and produces a bool type value. The logical operators are '&&' and '||' that combines two or more relational expressions.

7. Bitwise Expressions

A bitwise expression is an expression which is used to manipulate the data at a bit level. They are basically used to shift the bits.

Conditional statements:

Conditional statements, also known as selection statements, are used to make decisions based on a given condition. If the condition evaluates to True, a set of statements is executed, otherwise another set of statements is executed.

The if Statement:

The if statement selects and executes the statement(s) based on a given condition. If the condition evaluates to True then a given set of statement(s) is executed. However, if the

condition evaluates to False, then the given set of statements is skipped and the program control passes to the statement following the if statement. The syntax of the if statement is

```
1  if (condition) {
2    statement 1;
3    statement 2;
4    statement 3;
5  }
```

The if-else Statement:

The if – else statement causes one of the two possible statement(s) to execute, depending upon the outcome of the condition.

The syntax of the if-else statements is

```
1  if (condition) // if part {
2    statement1;
3    statement2;
4  }
5  else // else part
6    statement3;
```

Nested if-else Statement:

A nested if-else statement contains one or more if-else statements. The if else can be nested in three different ways, which are discussed here.

• The if – else statement is nested within the if part.

The syntax is

```
1  if (condition1) {
2    statement1;
3  if(condition2)
4    statement2;
5  else
6    statement3;
7  }
8  else
9    statement4;
```

The switch Statement:

The switch statement selects a set of statements from the available sets of statements. The switch statement tests the value of an expression in a sequence and compares it with the list of integers or character constants. When a match is found, all the statements associated with that constant are executed.

The syntax of the switch statement

```
1  switch(expression) {
2  case <constant1>: statement1;
3   [break;]
4  case <constant2>: statement2;
5  [break;]
6  case <constant3>: statement3;
7  [default: statement4;]
8  [break;]
9  }
10 Statement5;
```

Jump statements in C++

Jump statements are used to manipulate the flow of the program if some conditions are met. It is used to terminating or continues the loop inside a program or to stop the execution of a function. In C++ there is four jump statement: continue, break, return, and goto.

Continue:

It is used to execute other parts of the loop while skipping some parts declared inside the condition, rather than terminating the loop, it continues to execute the next iteration of the same loop. It is used with a decision-making statement which must be present inside the loop. This statement can be used inside for loop or while or do-while loop.

Break:

It is used to terminate the whole loop if the condition is met. Unlike the continue statement after the condition is met, it breaks the loop and the remaining part of the loop is not executed. Break statement is used with decision-making statements such as if, if-else, or switch statement which is inside the for loop which can be for loop, while loop, or do-while loop. It forces the loop to stop the execution of the further iteration.

Return:

It takes control out of the function itself. It is stronger than a break. It is used to terminate the entire function after the execution of the function or after some condition. Every function has a return statement with some returning value except the void() function. Although void() function can also have the return statement to end the execution of the function.

Goto:

This statement is used to jump directly to that part of the program to which it is being called. Every goto statement is associated with the label which takes them to part of the program for which they are called. The label statements can be written anywhere in the program it is not necessary to use before or after goto statement. This statement makes it difficult to understand the flow of the program therefore it is avoided to use it in a program.

Syntax:

```
goto label_name;
.
.
label_name:
```

Loops in C++

Loops help us automate repetitive work that we may have to do over and over again.

Loops discussed

- While loop
- For loop
- Do while
- Nested loops

While Loop

Imagine we had to print “Hello World” 100 times or n-number of times. Would it be wise to write cout << “Hello World\n” 100 times. While loops help us automate this.

Syntax:

```
while(condition(s)){
   // execute statement(s)
}
```

For Loop

While loop uses an external variable to control the execution. A for loop takes into account the

- Initialization
- Condition checking
- Incrementation

In its syntax itself. The syntax is shown below

```
for(intialization ; condition(s); incrementation)
{
   // execution statements(s)
}
```

Do While Loop

- Do while is very similar to while loop with one difference that it is exit control loop.
- That is the decision to run the loop again or not happens at the time of exit of the current loop.
- While in the while loop it happens before entry in the loop.

Syntax:

```
do{
   // execute statement(s)
}
while(condition(s))
```

C++ Functions:

The function in C++ language is also known as procedure or subroutine in other programming languages.

To perform any task, we can create function. A function can be called many times. It provides modularity and code reusability.

Advantage of functions in C:

1. Code Reusability

By creating functions in C++, you can call it many times. So we don't need to write the same code again and again.

2. Code optimization

It makes the code optimized, we don't need to write much code.

Suppose, you have to check 3 numbers (531, 883 and 781) whether it is prime number or not. Without using function, you need to write the prime number logic 3 times. So, there is repetition of code.

But if you use functions, you need to write the logic only once and you can reuse it several times.

Types of Functions:

There are two types of functions in C programming:

1. Library Functions: are the functions which are declared in the C++ header files such as ceil(x), cos(x), exp(x), etc.

2. User-defined functions: are the functions which are created by the C++ programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code.

Declaration of a function:

The syntax of creating function in C++ language is given below:

return_type function_name(data_type parameter...)

{

//code to be executed

}

Inline function in C++

One of the key features of C++ is the inline function. Therefore, let's first examine the utilization of inline functions and their intended application. If make a function is inline, then the compiler replaces the function calling location with the definition of the inline function at compile time.

Any changes made to an inline function will require the inline function to be recompiled again because the compiler would need to replace all the code with a new code; otherwise, it will execute the old functionality.

1. If a function contains a loop. (for, while, do-while)
2. if a function has static variables.
3. Whether a function recurses.
4. If the return statement is absent from the function body and the return type of the function is not void.
5. Whether a function uses a goto or switch statement.

Advantages of inline function:

- In the inline function, we do not need to call a function, so it does not cause any overhead.
- It also saves the overhead of the return statement from a function.
- It does not require any stack on which we can push or pop the variables as it does not perform any function calling.
- An inline function is mainly beneficial for the embedded systems as it yields less code than a normal function.

Disadvantages of inline function

The following are the disadvantages of an inline function:

- If we use many inline functions, then the binary executable file also becomes large.
- The use of so many inline functions can reduce the instruction cache hit rate, reducing the speed of instruction fetch from the cache memory to that of the primary memory.
- It also increases the compile-time overhead because whenever the changes are made inside the inline function, then the code needs to be recompiled again to reflect the changes; otherwise, it will execute the old functionality.

C++ Pointers

The pointer in C++ language is a variable, it is also known as locator or indicator that points to an address of a value.

The symbol of an address is represented by a pointer. In addition to creating and modifying dynamic data structures, they allow programs to emulate call-by-reference.

Syntax

datatype *var_name;

int *ptr;

Advantage of pointer:

1. Pointer reduces the code and improves the performance, it is used to retrieving strings, trees etc. and used with arrays, structures and functions.
2. We can return multiple values from function using pointer.
3. It makes you able to access any memory location in the computer's memory.

void pointer:

This unique type of pointer, which is available in C++, stands in for the lack of a kind. Pointers that point to a value that has no type are known as void pointers. This indicates that void pointers are very flexible because they can point to any data type.

Structure:

Structure in c++ is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures ca; simulate the use of classes and templates as it can store various information

The, struct keyword is used to define the structure. Let's see the syntax to define the structure in c.

```
1. struct structure_name
2. {
3.     data_type member1;
4.     data_type member2;
5.     .
6.     .
7.     data_type memeberN;
8. };
```

CODECHAMP
CREATED WITH ARBOK