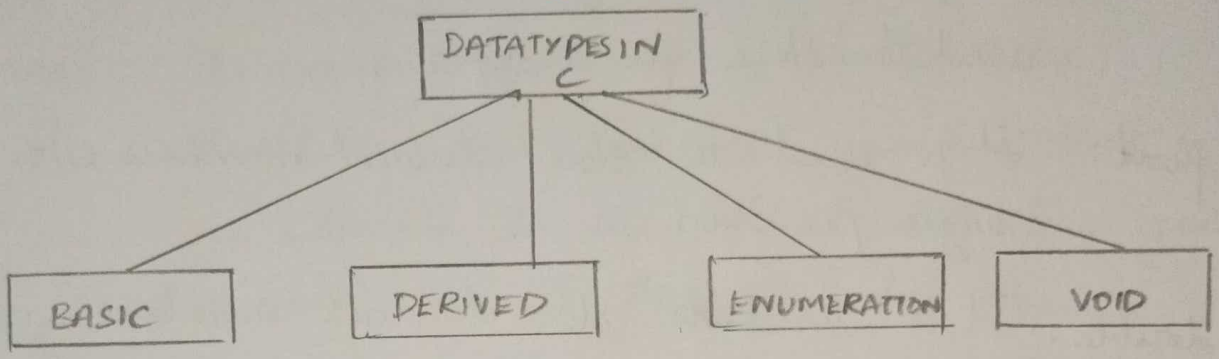
1. Explain the different data types available in C.

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



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 the basic data types may change according to 32 or 64-bit operating system.

DATA TYPES	MEMORY SIZE	RANGE
char	1 byte	-128 to 127
short	2 byte	-32,768 to 32767
int	2 byte	-32,768 to 32767
float	4 byte	
double	8 byte	

* char: The most basic data type in C. It stores a single character and require a single byte of memory in almost all compilers.

* int: As the name suggests, an int variable is used to store an integer.

* float: It is used to store decimal numbers with single precision.

* double: It is used to store decimal numbers with double precision.

Derived Data types in C

→ The derived data types are basically derived out of the fundamental data types.

→ Derived data types add various new functionalities to the existing ones instead.

→ These are used to represent multiple values as well as single values in a program.

Types

* Arrays: The array basically refers to a sequence of a finite number of data items from the same data type sharing one common name.

* Function: A function in C refers to a self contained block of single or multiple statements. It has its own specified name.

* Pointers: The pointers in C language refer to some special form of variables that one can use for holding other variable's addresses.

* Unions: The unions are very similar to the structures. But here, the memory that we allocate to the largest data type gets reused for all the other dat types present in the group.

* Structures: A collection of various different types of data type items that get stored in a contagious type of memory allocation is known as structure in C.

Enumeration Data Type

→ Enumeration is a user defined datatype in C language.

→ It is used to assign names to the integral constants which makes a program easy to read and maintain.

→ The keyword "enum" is used to declare an enumeration.

Syntax: enum enum_name {const1, const2, ... };

The enum keyword is also used to define the variables of enum type. 2 ways to define the variables:

- enum week {sunday, monday, tuesday, wednesday, thursday, friday, saturday};
- enum week day;

Void Data types

→ The void data type is an empty data type that refers to an object that does not have a value of any type.

→ void is used as a function returns type.

```
void myfunction (int i);
```

→ Void returns type specifies that the function does not return a value.

→ When it is used as a function's parameter list:

```
int myfunction (void);
```

→ Void parameter specifies that the function takes no parameters.

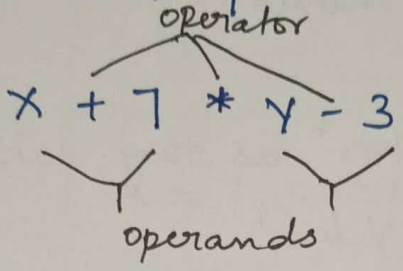
→ When it is used in the declaration of a pointer variable:

```
void *ptr;
```

→ It specifies that the pointer is "universal" and it can point to anything.

2. List the various types of operators in C. Explain Ternary Operator with example.

Operator is a symbol to perform some operations on one or more operands. Operands are variables which have been used with operator to evaluate expressions.



Types of Operators:

→ Arithmetic Operator
→ Relational Operator
→ Logical Operator
→ Assignment Operator
→ Increment and Decrement Operator
→ Conditional Operator
→ Bitwise Operator
→ Special Operator

→ Arithmetic Operator

C provides the following arithmetic operators to perform arithmetic operations.

Operator	Action	Example
+	Addition	A+B
-	Subtraction	10-5

*	Multiplication	x*y
/	Division	8/2
%	Modulus	9%4
+	Unary plus	
−	Unary minus	

* These are "Binary Operators" as they take two operands must be the "Numeric Value".
* If both operands take integer value, then the result is also in integer.
* If any one of operand is floating point and other is integer means, then the result is in floating point value.

→ Relational Operator

* A relational operator checks the relationship between two operands.
* If the relation is true, it returns 1.
* If the relation is false, it returns 0.
* Relational Operators are used in decision making and loops.
* Expression:

Ae1 relational operator Ae2;

Operator	Action	Example
==	Is operand 1 equal to operand 2.	y == z
>	Is operand 1 greater than operand 2	5 > 8
<	Is operand 1 less than operand 2.	4 < 2
>=	Is op 1 greater than or equal to op 2.	10 >= 4
<=	Is op1 lesser than or equal to op2	A <= B
!=	Is op1 not equal to op 2.	8 != 4

→ Logical Operator

* Logical operators are used to perform logical operations on the given expressions.
* It is also used to combine one or more relational expression, constants.

Operator	Action	Example
&&	Logical AND	(a&&b) is false
||	Logical OR	(x||y) is true
!	Not	!(x) is true

→ Assignment Operator.

* Assignment operator (=) used to assign a value to a variable.
* It can be combined with arithmetic operator to perform arithmetic assignment operation.

Operator	Action	Example
+ =	Addition assignment	A+=1
− =	Subtraction assignment	A-=2
* =	Multiplication Assignment	A *=b
/ =	Division Assignment	a/=n+1
%. =	Modulo Assignment	A%.=b

→ Increment and Decrement Operator

* By using these operators, we can do either increase or decrease a value of a variable.
* These operators are "unary" as they take only one operand.
* These operand can be written before or after the operator.

Increment/Decrement Operators		Let us assume x is a variable.
Operator	Expression	Description.
+ +	++x	Pre-increement
	x++	Post- increement
− −	−−x	Pre-decrement
	x−−	Post − decrement

→ Conditional Operator

* We use the ternary operator in C to run one code when the condition is true and another code when the condition is false. For example,

```
(age >= 18) ? printf ("Can vote") : printf ("Cannot Vote");
```

Here, when the age is greater than or equal to 18, 'Can Vote' is printed. Otherwise, 'Cannot Vote' is printed.

Syntax:

```
test condition ? expression1 : expression 2
```

The test condition is a boolean expression that results in either true or false. If the condition is

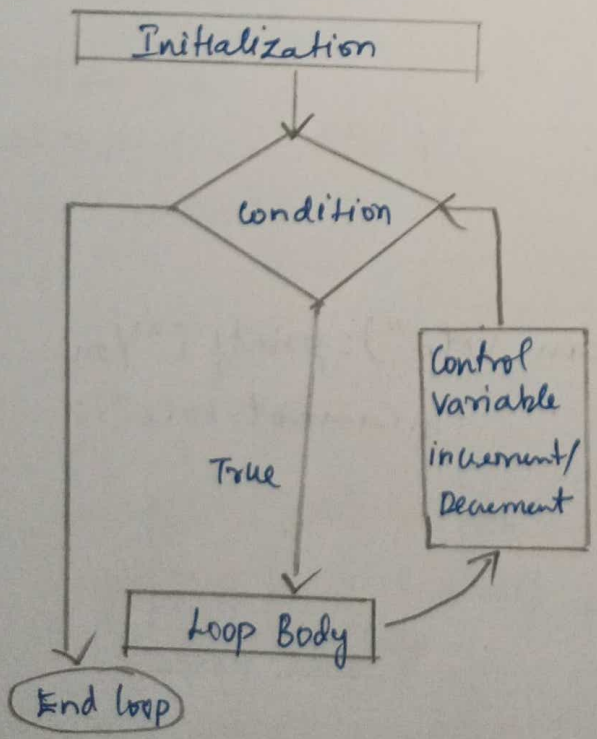
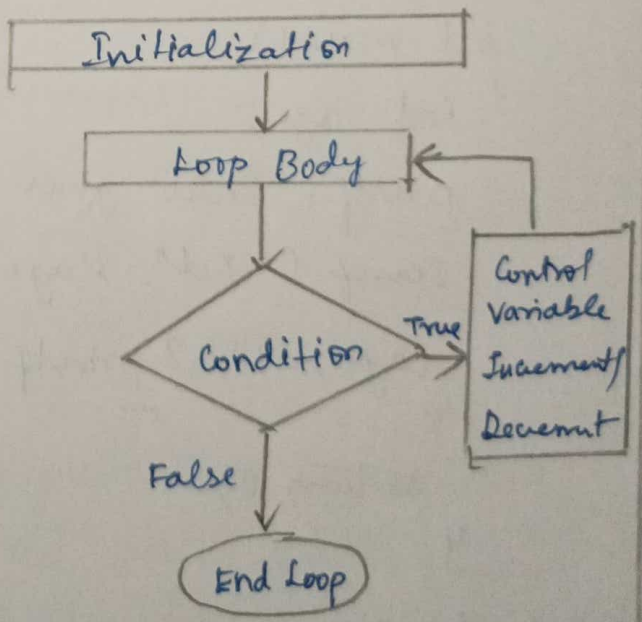
- true - expression 1 is executed
- false - expression 2 is executed

Example:

```
# include <stdio.h>
int main()
{ int age;
  printf("Enter your age:");
  scanf ("%d", &age);
  (age >= 18) ? printf ("You can vote") : printf ("You
                                          cannot vote");
  return 0;
}
```

Output: Enter your age: 12
You cannot vote.

3. Differentiate between Entry and Exit controlled loop with example.

Entry Controlled Loop	Exit Controlled Loop.
* An entry control loop checks condition at entry level (at beginning), thats why it is termed as entry control loop.	* An exit control loop checks condition at exit level (in the end), thats why it is termed as exit control loop.
* It is a type of loop in which the condition is checked first and then after the loop body executed.	* Opposite to entry controlled loop, it is a loop in which condition is checked after the execution of the loop body.
* For loop and while loop fall in this category.	* Do-while loop is the example.
* 	* 

* Execution:

→ The control variable is initialized first and acts as a base for comparison.

→ This variable is then evaluated with a conditional statement.

→ If the condition is evaluated as false, the loop would be terminated, otherwise.

→ The loop body would be executed and go to the next iteration, where the variable would get updated with new incremented/decremented value.

→ Repeat 2nd to 4th step until the condition is said to be false.

* Example:

```
# include <stdio.h>
void main ()
{
int i = 10;
while (i<10)
{
printf ("I will not be executed as it is entry controlled loop");
```

* Execution:

→ The control variable is initialized first and acts as a base for comparison.

→ Then, the loop body is executed and the variable would get also updated with new incremented / decremented value.

→ This variable is then evaluated with a conditional statement.

→ If the condition is evaluated with a conditional statement.

→ If the condition is evaluated as false, the loop would be terminated, otherwise.

→ It repeats, 2nd to 4th steps until the condition is said to be false.

* Example:

```
# include <stdio.h>
void main ()
{int i = 10;
do
{
printf ("I will be executed at once as it is exit controlled loop");
i++;
```

```
    i++;
    }
    getch();
}
```

* In the above code, no output would be displayed as the condition is false.

```
    } while (i<10);
    getch();
}
```

* In the above code, print statement would be executed at least once even though condition is false.

4. Explain call by value and call by reference with example.

Call by Value:

→ The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function.

→ In this case, changes made to the parameter inside the function have no effect on the argument.

→ By default, C programming uses call by value to pass arguments.

→ In general, it means the code within a function cannot alter the arguments used to call the function.

→ swap() definition:

```
void swap(int x, int y)
{ int temp;
  temp = x;
   x = y;
   y = temp;
   return;
}
```

Program:

```
# include <stdio.h>
 void swap (int x, int y);
 int main()
 { int a = 100;
   int b = 200;
   printf ("Before swap, value of a and b are:
          %d %t %d, a, b);
    swap (a, b);
    printf ("After swap, value of a and b are:
          %d %t %d, a, b);
     return 0;
}
```

Output:

Before swap, value of a and b are: 100 200

After swap, value of a and b are: 200 200

It shows that there are no changes in the values, though they had been changed inside the function.

Call by Reference

→ The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter.

→ Inside the function, the address is used to access the actual argument used in the call.

→ It means the changes made to the parameter affect the passed argument.

→ To pass a value by reference, argument pointers are passed to the functions just like any other value.

→ So accordingly you need to declare the function parameters as pointer types as in the following function swap(), which exchanges the values of the two integer variables pointed to, by their arguments.

Program:

```
#include <stdio.h>
int main()
{ int a = 100;
  int b = 200;
  printf("Before swap, value of a and b are:
          %d \t %d , a , b);
  swap(&a, &b);
```

```
    printf("After swap, value of a and b are : %d \t %d",
                                                a, b);

    return 0;
}
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
    return;
}
```

Output:

Before swap, value of a and b are: 100 200

After swap, value of a and b are : 200 100

It shows that the change has reflected outside the function as well, unlike call by value where the changes do not reflect outside the function.