Q. Why c is called Structured Programming Language?
The basic structure of C programming.

Sol. C is called a structured programming language because it organized code into functions, uses clear control structures like if-else and loops, and follows a modular approch. This makes programs easier to read, maintain and debug, promoting a logical and organized flow of instructions.

Basic Structure of C Language

- Documentation Section : // Name of program
- Link Section : #include <stdio.h> (header file)
- Define Section: #define PI 3.14
- Global Declaration Section: int a=20
- Main() Function Section: main(){ }
- Subprogram section : Function 1, Function 2,.....

Or,

```
// simple c program  → Documentation Section
#include <stdio.h>  → Link Section (Header File)
#define MIN 99  → Define Section
void add();
int x = 100;  } → Globla Declaration Section
int main(){  → Main() Function
    int a = 100;
    printf("Hello World!");
    return 0;
}  } → Body of main() Function
void add(){
  printf("Hello Add");  } → Function Define
}
```

Fundamental Rules for naming a variable in C:

→ Start with a letter or underscore: A variable name must begin with a letter (a-z, A-Z) or an underscore(_) but not a number. Example: Aa2, _Aa, _a etc.

→ only letter, digits and underscores: After the first characters, it can contain letters, digit (0-9) or underscores. Example: Five_5, One_1 etc

→ Case sensitive: C is a case sensitive language means uppercase and Lowecase are different variable. So, var and Var are different variable.

→ No space or spacial character: spacial characters like #, @, %, etc and space can not be a variable

→ can not be a keyword: Keywords like if, return, int etc can not be a variable.

Program : Swap of Two Numbers with third variable :-

Code:

```
#include <stdio.h>
int main () {
  int num1, num2, temp;
    num1 = 10, num2 = 20;
        num1 = num2;
        temp = num1;
        num2 = temp;
    printf (" Swap num1 = %d \n ", num1);
    printf (" Swap num2 = %d \n ", num2);
    return 0;
}
```

Swap of two numbers without third variable :

Code:

```
#include <stdio.h>
int main () {
    int num1 = 10, num2 = 20;
        num1 = num1 + num2
        num2 = num1 - num2
        num1 = num1 - num2
   printf (" Swap num1=' %d \n ", num1);
   printf (" Swap num2 = %d \n ", num2);
   return 0;
}
```

Increment Operator and Decrement Operator:

- Prefix increment – ++a – Increments a by one before it used.
- postfix increment – a++ – Increment a by one after it used.
- Prefix Decrement – --a – Decrements a by one before it used.
- Postfix Decrement – a-- – Decrement a by one after it used.

Why conditional Operator is called a ternary operator:

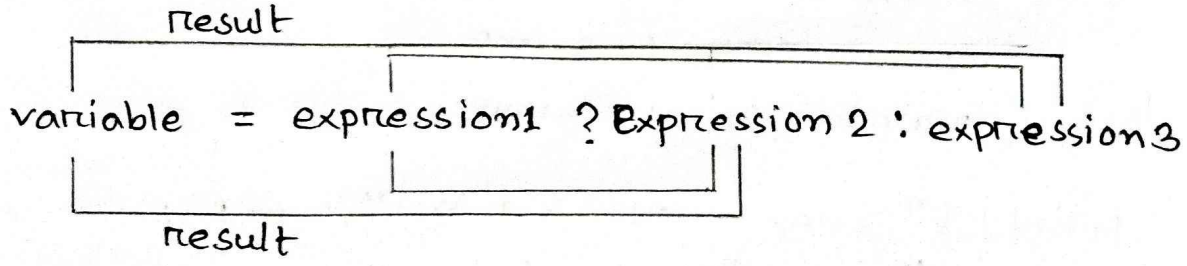
→

01. As conditional operator works with three operands so it is called ternary operator.

02. It is represented by two symbols → '?' and ':'

03. It works similarly as if-else statement.

The flowchart below shows how it works.



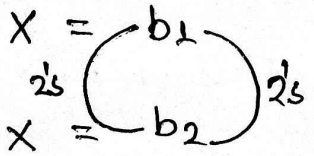
Ⓐ For any integer n, bitwise complement of n will be -(n+1) —

- Bit wise complement operator is a unary operator (works on only one operand).
- It changes 1 to 0 and 0 to 1.
- It is denoted by ~

35 = 00100011 (in Binary)

~35 = 11011100

```
  00100011
+         1
----------
  00100100 (36)
```



So, ~35 = -36

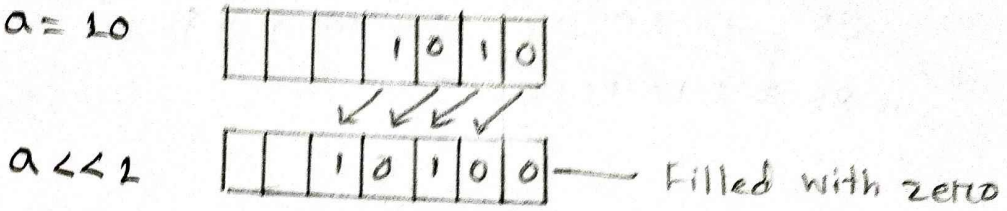
So, For any integer n, bitwise complement of n will be

= -(n+1)

Explane the left shift operator:

1. Left shift is a bitwise operator, denoted by <<
2. It shift all of the bits toward left by a certain number of specific value.
3. The left bits filled with zero.
4. If a variable left shifted one time the value becomes double of the original.

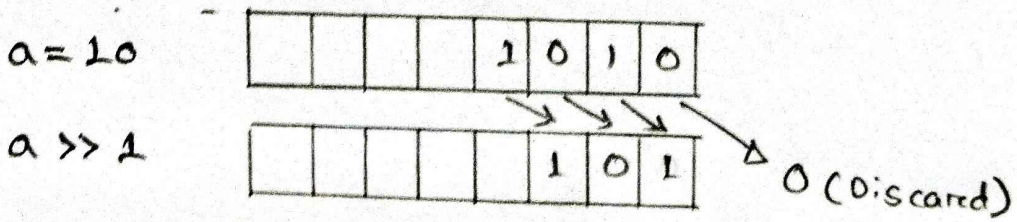
For instance, a = 10, then, a << 1 = 20



Explane right shift operator:

1. Right shift is a bitwise operator, denoted by >>
2. It shifts all of the bits toward right by a certain number of specific value.
3. The most right shifted values are discarded.
4. If a variable right shifted one time the value becomes half of the original.

For instance, a = 10 then, a >> 1 = 5



Differences between if-else and switch Statement:

→ If-else:

- Flow of Execution: Can have two possible paths of execution. Either if or else.
- Type of outcome: Always evaluates an expression to a boolean value.
- Readability: less readable in case of multiple if else.
- Efficiency: It becomes less efficient in case of multiple if-else. Since it checks for every if/else and evaluates the corresponding statements.

→ Switch:

- Flow of Execution: Can have multiple paths of execution in the form of different cases.
- Type of outcome: Can evaluate to an expression to enum, short, char and int type values.
- Readability: More readable in case of multiple cases.
- Efficiency: More efficient working because once a valid match is found other cases are bypassed if handled carefully with break statement.