Book: Let us C : 16th Edition
(Red Colour) YASHAVANT KANETKAR.

Mid Sem → 30

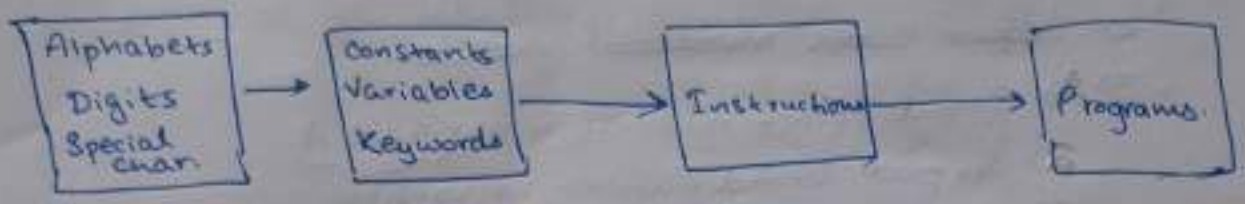
Sem → 50

Class Test → 10

Attendance/Assignment → 10 (5 18)

PROGRAM AND SOFTWARE

Structure of C-Language:



Structure of Normal Language:→ Alphabets → words → Sentences → Paragraphs

※ What is C?

→ Programming Language (HLL)

→ Developed at AT&T's bell Lab, USA in 1972.

→ Designed and developed by Dennis Ritchie.

→ It is reliable, simple, and easy to use.

→ Important Types:

i) Imperative Statements (Action)

ii) Conditional Statement (Decision Making)

iii) Iterative Statement (Loops / Repetation)

※ ALGORITHM:

→ An algorithm is a finite set of unambigious instructions which, when executed, performs a task correctly.

(i) The number of steps required to be performed should be finite

(ii) Each of the instruction in the algorithm should be unambigous in nature, meaning on execution of each such ~~instant~~ instruction the outcome should be definite and predictable.

(iii) finally, the algorithm should solve the program correctly.

Q. Develop an algorithm to find the average of three numbers taken as input from use.

→ Steps: 0) Start

~~2) Define three numerical variables.~~

~~3)~~ 1) Input first number in variable A.

2) Input second number in variable B.

3) Input third number in variable C.

4) Compute SUM = (A+B+C)

5) Compute AVG = $\frac{SUM}{3}$

6) Print AVG.

7) STOP/End.

Q. Develop an algorithm to find the maximum of two numbers input by the user.

→ STEPS:- 0) START

1) Input first number in variable A.

2) Input second number in variable B.

3) ~~Compare~~ If A > B, {print A}
else {print B}.
End If.

4) STOP/END.

Q. Develop an algorithm to arrange given three nos in descending order. ~~input by user.~~

→ STEPS: 0) START

1) DECLARE INT VARIABLES A, B, C WITH values stored in them.

2) DECLARE INT VARIABLE TEMP = 0.

3) FOR INT VARIABLE I = 1 AND I < 3

{

IF A > B AND A > C

{ Do Nothing }

ELSE IF A > B AND A < C

{

TEMP = A

A = C

C = TEMP

}

ELSE

{

TEMP = A

A = B

B = TEMP

}

IF C > B

{ TEMP = C

C = B

B = TEMP

}

END IF { Increas value for I by 1.

END FOR

4) DISPLAY A, B, C.

5) STOP.

Method 1:

Step 0: START

1: INPUT three variables n_1, n_2, n_3.

2: (i) If ($n_1 > n_2$ AND $n_1 > n_3$)

COMPUTE HIGhest $= n_1$

(ii) If ($n_2 > n_1$ AND $n_2 > n_3$)

COMPUTE Highest $= n_2$.

(iii) If ($n_3 > n_2$ AND $n_3 > n_1$)

COMPUTE Highest $= n_3$

3: STOP

Method 2:

Step 0: START

1: INPUT three variables n_1, n_2, n_3.

2: IF ($n_1 > n_2$)

```
{ if (n1 > n3)
      highest = n1
  else
      highest = n3
}
else
{
    if (n2 > n3)
        Highest = n2
    else
        Highest = n3
}
```

3: STOP

Method 3:

Step 0: START

1: INPUT Three variables n_1, n_2, n_3.

2: IF ($n_1 > n_2$ AND $n_1 > n_3$)

Highest $= n_1$.

else if ($n_2 > n_1$ AND $n_2 > n_3$)

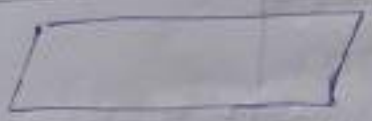
Highest $= n_2$

else

Highest $= n_3$

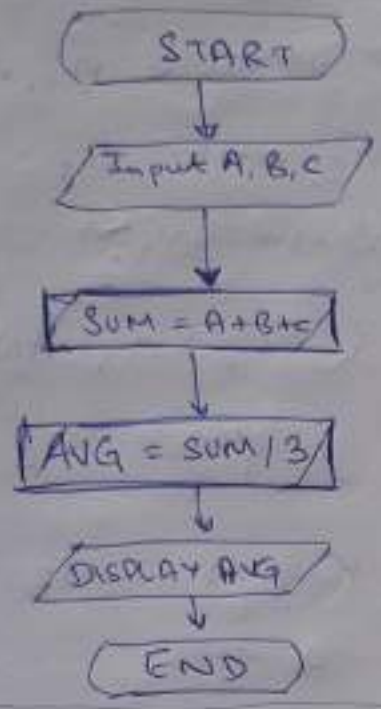
3: STOP.

FLOW CHART

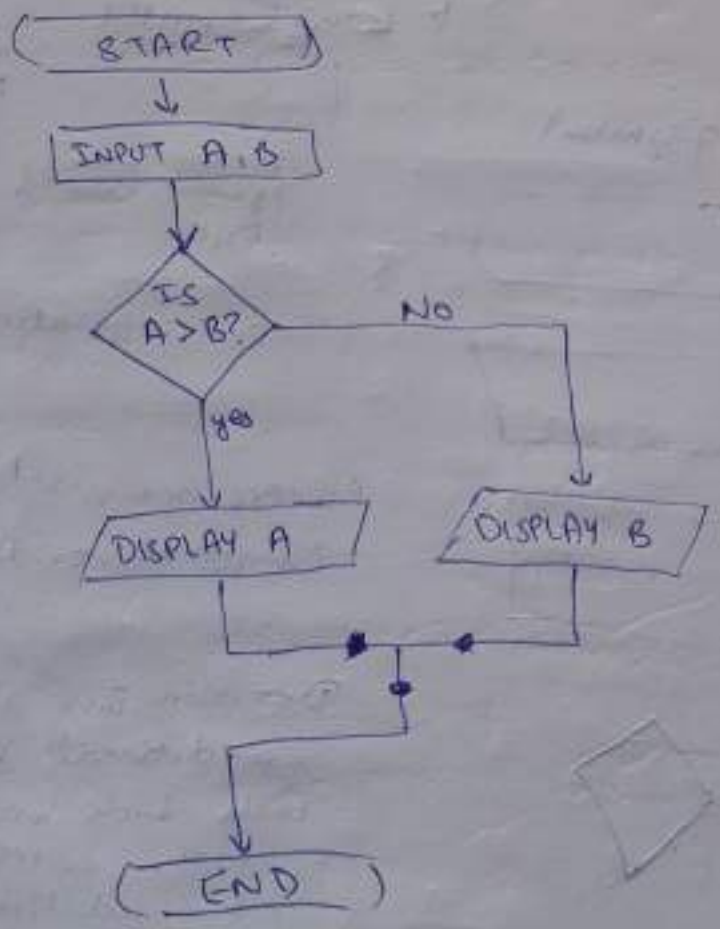
Symbol	Meaning
	Symbol issued to mark START and END
	Input operation / output operation
	Process box used to represent imperative logic construct.
	Decision Box used to represent conditional statement/constructions. Each such box has two exit points which corresponds to evaluation of condition to true / false.
	Sequence of flow.

* Average of 3 numbers input by user.

→



* Find highest of 2 numbers.



CONSTANTS

- **Primary**
  - integer
  - real
  - characters
- **Secondary**
  - Array
  - pointer
  - Structure
  - Union

IDE :→ Visual Studio, GCC, Turbo C/C++, etc.

* Rules for Constructing Integer:

(i) Must have at least one digit.
(ii) must not have decimal point.
(iii) can be any zero, positive, negative.
(iv) no commas/blanks are allowed within integer constant.
(v) The allowance range for integer constant.

Visual Studio/GCC → -2147483648 to +2147483647
Turbo C → -32768 to +32767.

* Rules for Constructing Real:

(i) must have at least one digit
(ii) Must have a decimal point
(iii) either positive/negative.
(iv) default sign positive.
(v) No commas/blanks within real constant.

* Rules for constructing Characters:

(i) is a single alphabet, single digit a single special symbol enclosed within single inverted comma (' ').

C-Keywords : (32):

(i) auto
(ii) break
(iii) case
(iv) char
(v) int
(vi) float

---

* Variable:

* A particular type of variable can hold only same type of constant.

RULES:

1) first character in the variable name must be an alphabet / underscore

2) Case Sensitive.

3) Variable name can be any combination of alphabets, digits and underscores.

4) No special symbols are allowed other than underscore

5) Some compilers allow variable name whose length could be upto 247• characters.

6) No commas or blanks are allowed within a variable name

- Keywords in C : (32 in total)

1. auto
2. break
3. case
4. char
5. const
6. continue
7. default
8. do
9. double
10. else
11. enum
12. extern
13. float
14. for
15. goto
16. if
17. int
18. long
19. register
20. return
21. short
22. signed
23. sizeof
24. static
25. struct
26. switch
27. typedef
28. union
29. unsigned
30. void
31. volatile
32. while

ANSI C :

Three types of datatypes:

1) Primary/ fundamental datatypes → int / float / char / double / long
2) Derived datatypes → Arrays / Structures / classes
3) User defined datatypes

i) integer → int
ii) floating type → float
iii) character → char
iv) double precision floating type → double
v) void
vi) long int
vii) long double.

- Integer
  - → Signed → int, short int, long int
  - → Unsigned → unsigned int, unsigned short int, unsigned long int

- Syntax for defining variable name:

  data_type variable_name;

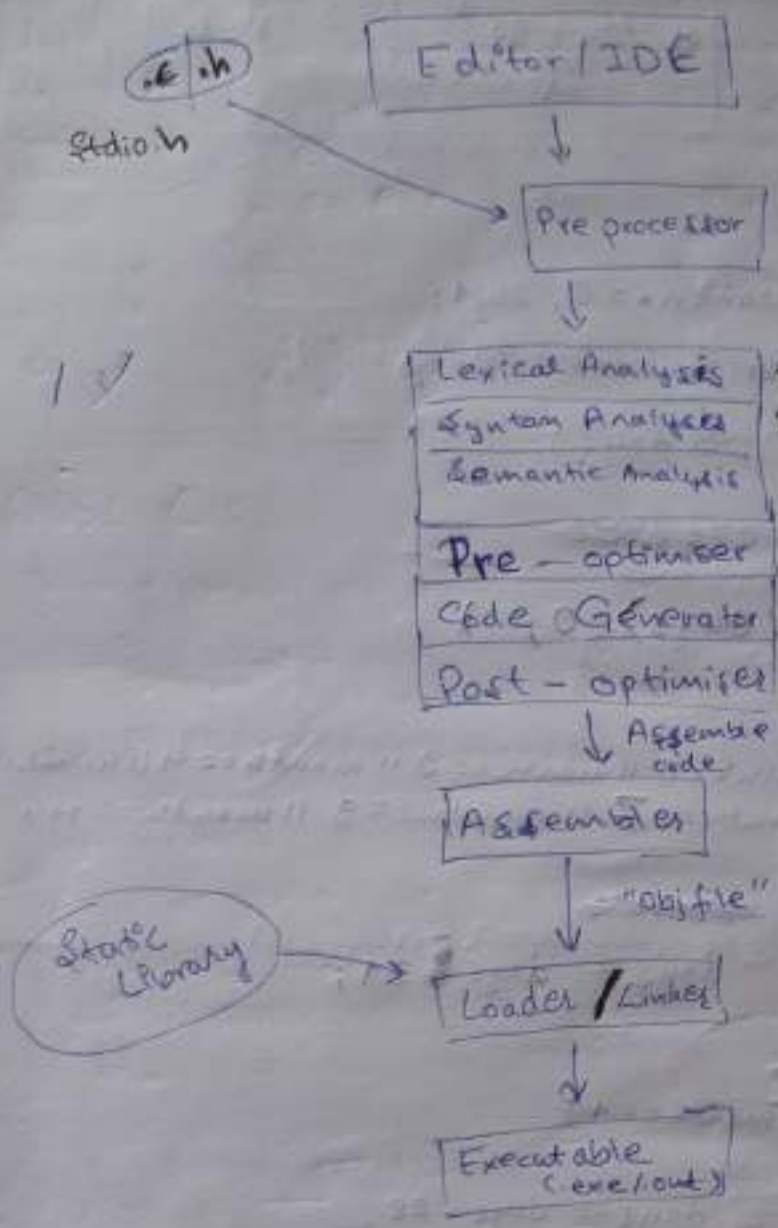
* Every C program consists of one or more functions :
  - one primary function must be called main().
  - the program will always begin by executing the main() function.

* Each function must contain:
  - A function heading, which consists of function name, followed by an optional lists of arguments enclosed in parenthesis.
  - A list of argument declaration
  - compound statements.

Structure of C program:

```
#include <stdio.h>
/*  ____________  */
return type main()
{
    Variable
    Statements...
}
```

```
#include <stdio.h>
main()
{
    int a,b,c;
    a = 10;
    b = 2;
    c = a+b;
    printf("The sum of %d and %d is %d",a,b,c);
    printf("\n Enter two numbers: ");
    scanf("%d %d", &a,&b);
    c = a+b;
    printf("The sum of %d and %d is %d",a,b,c);
}
```



C Operators:

1) Arithmetic Operators
2) Relational operators
3) Logical Operator
4) Assignment Operator
5) Increment / Decrement operator
6) Conditional Operators
7) Bitwise operators
8) Special operators.

0	1	2	3	4	5	6	7	8	9	10	11
Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
31	28	31	30	31	30	31	31	30	31	30	31

```
→ #include<stdio.h>
void main()
{
    int days, month=0, day1;
    printf("Enter number of days: ");
    scanf("%d", &days);

    for(      ; days>=0; )
    {
        if (month==0 || month==2 || month==4 || month==6 ||
            month==7 || month==9 || month==11)
        {
            days = days - 31;
        }
        else if (month==1)
        {
            days = days - 28;
        }
        else
        {
            days = days - 30;
        }

        if (days >=0)
        {
            day1 = days;
            ++month;
        }
    }
    printf(" Month=%d and Day=%d", month, day1
}
```

```
main()
{ int m, d, c;
    printf ("Enter number of days");
    scanf ("%d", &d);
      m = d / 30;
      c = d % 30;
    printf ("Months = %d and Days = %d", m, c);
}
```

---

* Relational operators :

(i) < (v) ==

(ii) <= (vi) !=

(iii) >

(iv) >=

---

* Logical Operators :

(i) && Logical AND

(ii) || Logical OR

(iii) ! Logical NOT

* Assignment Operators :

Variable operator = expression

eg x + = 3

i.e. $\boxed{x = x+3}$

Shorthand operators

* Increment and Decrement Operators:

(i) ++ increment

(ii) -- decrement

Increment:

(i) post increment : x++

(ii) pre increment : ++x

Decrement:

(i) post decrement : x--

(ii) pre decrement : --x

```
main()
{
    int a=5;
    printf("%d %d %d", ++a, a, a++);
}
```

→ output: 5.67

Output: 5 6 7

* Conditional Operator:

(a>b) ? 1 : 0

```
main()
{ int a,b,c,d;
    a = 15; b = 10;
    c = ++a - b;
    printf("a=%d b=%d c=%d", a,b,c);
    d = (b++) + a;
    printf("a=%d b=%d d=%d", a,b,d);
    printf("%d", (c>d)? 1:0);
}
```

→ output: a=16 b=10 c=6 a=16 b=11 d=26 0

※ Evaluation of Expression:

Variable = expression

eg: x = a*b-c

x = a-b/c+d

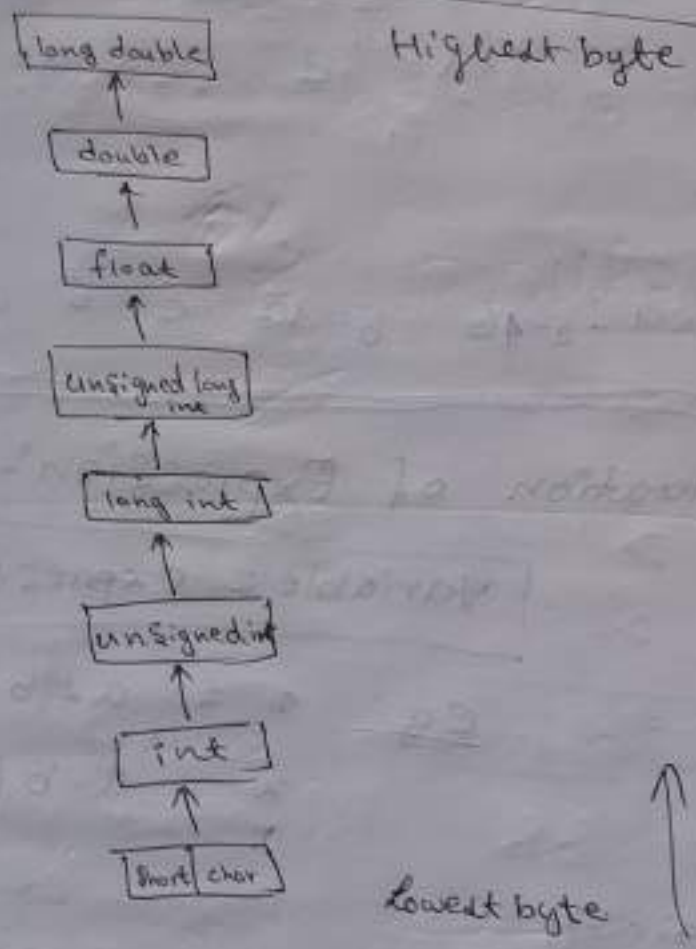

```
main()
{   float a,b,c,x,y,z;
    a=9; b=12; c=3;
    x = a-b/3+c*2-1;
    y = a-b/(3+c)*(2-1);
    z = a-(b/(3+c)*2)-1;
    printf("x=%f y=%f z=%f",x,y,z);
}
```

output: x= 10.00000 y = 7.00000 z = 4.00000

Type Casting:

```
(datatype) expression;
```

```
int x;
float r = 3.0;
x = (int) (2*r);
```



Managing input and output operation.

```
getchar();
```

```
datatype variable_name = getchar();
```

```
main()
{
    char a;
    printf(" Enter Y or y... ");
        a = getchar();
    if ( a=='Y' || a=='y')
            printf(" MyName ");
    else
            printf(" Invalid input... Try again!! ");
}
```

Writing a character:

```
putchar (variable_name);
```

eg.
```
char x = 'Y';
putchar (x);
```

---

include <ctype.h>

1) isalnum (c) : checks if c is alphanumeric character.
2) isalpha (c) : Checks if c is alphabetical character.
3) isdigit (c) : checks if c is numerical value.
4) islower (c) : checks if alphabet is lower case.
5) isupper (c) : checks if alphabet is upper case.
6) ispunct (c) : punctuation mark.
7) isspace (c) : white space.

* Determine the value of each of the following logical expression if a=5 ; b=10; c=-6;

a. a>b && a<c

b. a<b && a>c

c. a==c || b>a

d. b>15 && c<0 || a>0

e. (a/2.0==0.0 && b/2.0!=0.0) || c<0.0

→ (a) 0

(b) 1

(c) 1

(d) 1

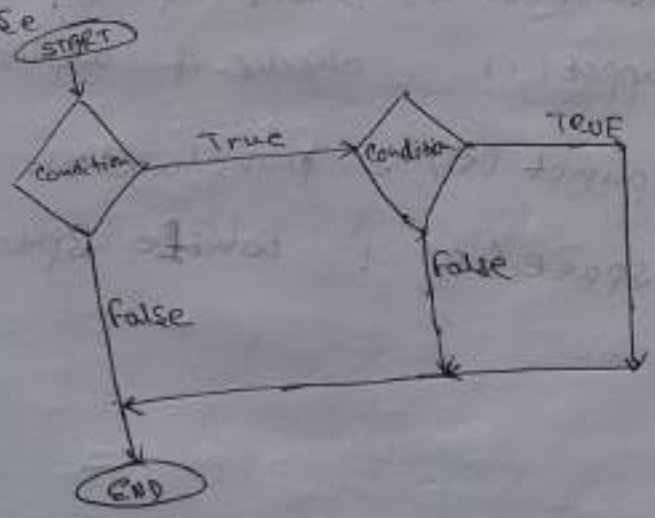
(e) 1

PODMAS

Decision Making and Branchery:-

- Statements:
  - (i) if Statement
  - (ii) Switch Statement
  - (iii) Conditional operator Statement
  - (iv) goto Statement.

Nested if-else



GRIM ~ x SCYTHE

1) →
```
main()
{ int a=5, b=8, c=12;
    if(a>b && a>c)
        printf("%d is largest", a);
    if (b>a && b>c)
        printf("%d is largest", b);
    if ( c>a && c>a)
        printf("%d is largest", c);
}
```

→
```
main()
{ int a=5, b=8, c=12;
    if (a>b && a>c)
        printf("%d is largest", a);
    else if (b>c)
        printf("%d is largest", b);
    else
        printf("%d is largest", c);
}
```

→
```
main()
{ int a=5, b=8, c=2;
    if (a>b)
        if (a>c)
            printf("%d is largest", a);
        else
            printf("%d if largest", c);
```

(5/20 == 0.088 && 10/20 != 0.0)
|| -6 < 0.0

(2.50000 == 0.088 && 0.00001 != 0.0)
|| -6 < 0.0

(0 && 1) || 1

0 || 1

1

Q. WAP :

Gender	Years of Service	Qualification	Salary
M	>= 10	PG	15000
	>= 10	G	10000
	< 10	PG	10000
	< 10	G	7000
F	>= 10	PG	12000
	>= 10	G	9000
	< 10	PG	10000
	< 10	G	6000

```
#include<stdio.h>
main()
{
    int yos, sal=0;
    char gen, quali;
    printf("Enter your gender. M/F ? ");
    scanf("%c",&gen);
    printf("Enter your years of service : ");
    scanf("%d", &yos);
    printf("Enter your qualification : ");
    scanf("%c", &quali);

    if( gen=='m' || gen=='M')
    {
        if( yos >= 10)
        {
            if(quali=='P' || quali=='p')
                sal = 15000;
            else if(quali=='G' || quali=='g')
                sal = 10000;
        }
        else if (yos < 10)
            if (quali=='P' || quali=='p')
                sal = 10000;
            else if (quali=='G' || quali=='g')
                sal = 7000;
    }
```

```
else if (gen=='f' || gen=='F')
    if (yos >= 10)
        if (quali=='p' || quali=='P')
            sal = 12000;
        else if (quali=='G' || quali=='g')
            sal = 9000
    else
        if (quali=='P' || quali=='P')
            sal = 10000;
        else if (quali=='G' || quali=='g')
            sal = 6000;
printf(" Your salary is INR %d ", sal);

}
```

Switch:

1. When one of the many alternatives ~~to~~ is to be selected, we can use an if statement to control ~~the~~ selection.
2. Complexity increases when number of an alternative increases.

SYNTAX:

```
switch (expression)
{
    case value1:
        statements 1;        // Block-1
        break;
    case value 2:
        statements-2;        // Block-2
        break;
    default:
        statements-;         // Default Block
        break;               // If default, break is not required.
}
```

(1) Expression is integer / character.

(2) Value 1, Value 2, ... Value n are constants.

(3) Each value should be unique.

(4) Block-1, Block-2 ... Block n are statements.

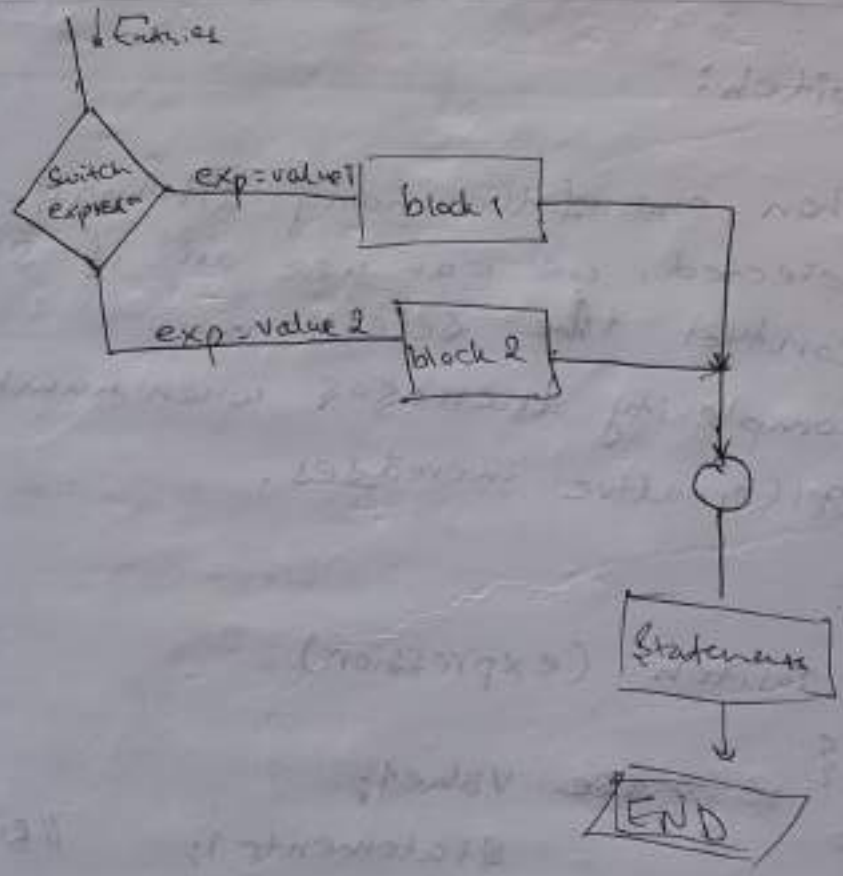
(5) There is no need to put braces around these blocks

(6) Case labels end with colon (:)

(7) The break statement at the end of the block signals the end of particular case and causes an exit from switch statement transferring control to the statement-x following the switch.

(8) default case and statements are optional.

FLOW CHART :



Q. WAP ~~where~~ to take a value from the user, where if user inputs value 1, addition of 2 numbers occur, for second value, subtraction occurs

→

```
# include <stdio.h>
void main()
{
    int a, b, c;
    printf(" Enter two values: ");
    scanf("%d %d", &a, &b);
    printf(" *** MENU *** ");
    printf("\n 1. Addition ");
    printf("\n 2. Subtraction");
    printf("\n Select your option: ");
    scanf(" %d ", &c);
    switch (c)
    {
        case 1:
            printf(" Sum is : %d ", (a+b));
            break;
        case 2:
            printf(" Difference is : %d", (a-b));
            break;
        default:
            printf(" Invalid Option! Try again.");
    }
}
```

```
void main()
{
    int num = 1;
    switch (num)
    {
        case 1:
            print ("\nIn block 1");
        case 2:
            print ("\nIn block 2");
        default:
            printf("\nXYZ");
    }
}
```

→ ~~Output: In block 1 In block 2 XYZ~~

→ output: In block 1
In block 2
XYZ

```
void main()
{
    int x=2, y=1, z=0;
    switch (x)
    {
        case 2:
            x=1; y=x+1;
            printf("In block 2");
        case 1:
            x=0;
            printf("In case 1 Block");
            break;
        default: x=1; y=0;
            printf("In Default");
    }
}
```

→ OUTPUT: In block 2 In case 1 Block

---

* Decision Making and Looping:

A loop is used for execution of a ~~boolean~~ block of statements repeatedly until a given condition returns false.

STEPS: ① Setting and initialisation of a condition variable.

② Executing statements in the loop.

③ Test for a specified value of condition variable for execution of the loop.

④ Incrementing/updating the condition variable.

⑤ Execution of a block of statements repeatedly until a given condition returns false.

→ TYPES OF LOOP:

(i) The while state method.

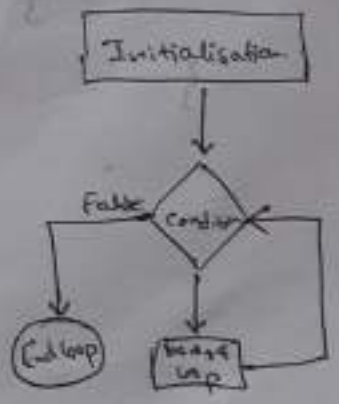
(ii) The do while statement.

(iii) for statement.

While Loop SYNTAX:

```
while (test condition)
{
    body of the loop;
}
```

Flow chart



Q. WAP to check

→
```
#include <stdio.h>
void main()
{
    int i=0;
    while (i<5)
    {
        printf("Looping %d", (i+1));
        printf("\n");
        ++i;
    }
}
```

Q. WAP to identify odd numbers input by the users.

→
```
#include <stdio.h>
void main()
{ int a, i=0;
    while (i <20)
    {
        printf("\n");
        printf("Enter Number: ");
        scanf("%d", &a);
        if( a % 2 != 0)
            printf("odd");
        ++i;
    }
}
```

```
main()
{
    int var=6;
    while (var >=5)
    {   printf("%d\t", var);
        ++var;
    }
}
```

→ output: 6 7 8 9 10 11 ....

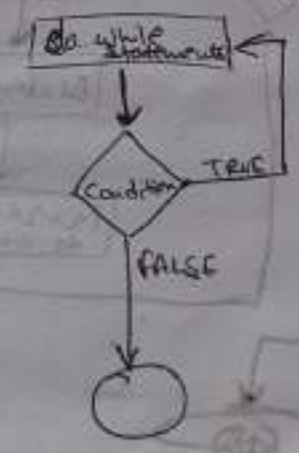
infinite loop

do... while loop:

Syntax:

```
do{
    // statements inside the body of the loop.
}while (test condition);
```

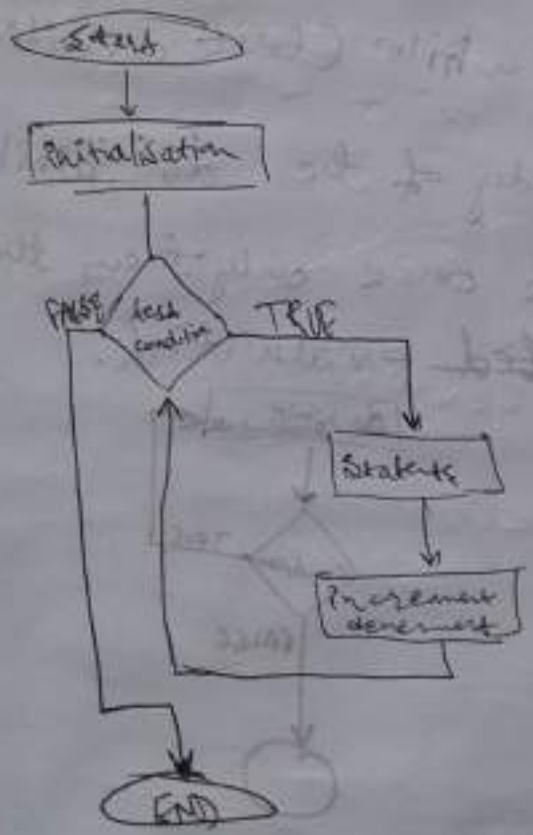
* The body of the do while loop is executed at least once only then the test expression is ~~executed~~ evaluated.



```
# include<stdio.h>
void main()
{
    int num, sum=0;
    printf(" Enter numbers: ");
    do
    {
        scanf("%d", &num);
        sum = sum + num;
    } while (num!=0);
    printf("sum is: %d", sum);
}
```

For Loop:

Syntax: for (initialisation; condition; increment/decrement)
{ Statements; }



```
#include <stdio.h>
void main()
{
    for (int i=1, i<=10;++i)
        printf(" %d ", i);
}
```

```
# include <stdio.h>
void main()
{
    for (int i=10; i>0; --i)
        printf("%d", i);
}
```

for	while	do ... while

for(initialization;
test condition;
increment/decrement)
{ statements;
}
	initialisation;
	while (test condition)
	{
	statements;
	increment/decrement;
	}
		initialisation;
		do
		{
		statements;
		increment/decrement;
		} while (test condition);

Q. WAP to give out-put in following manner:

```
0   0
0   1
0   2
0   3
1   0
1   1
1   2
1   3
```

→

```
#include <stdio.h>
void main()
{
    int i, j;
    for (i=0; i<2; ++i)
    {
        for (j=0; j<4; ++j)
        {
            printf("%d %d", i, j)
            printf("\n");
        }
    }
}
```

ARRAY:

Syntax:

datatype variable_name[size];

1) int a[5]; // Garbage values stored.

2) int a[5] = { 10, 20, 30, 40, 50};
printf("%d", a[1]); // prints 20,

3) int a[5] = { 20, 30};
a[2], a[3], a[4], // all will be zero.

```
main()
{
    int a[5] = {10,20,30,40,50};
    for (int i=0; i<5; ++i)
        printf("%d ", a[i]);
}
```

Q. WAP to take 5 input from the user and output them.

→
```
#include <stdio.h>
void main()
{
    int a[5], i;
    printf("Enter 5 numbers");
    for (i=0; i<5; ++i)
        scanf("%d", &a[i]);
    printf("Your inputs are: ");
    for (i=0; i<5; ++i)
        printf("%d \t", a[i]);
}
```

```
→ void main()
  {
      int a[5], i, sum=0;
      for( i=0; i<5; ++i)
          scanf("%d", &a[i]);

      for( i=0; i<5; ++i)
      {
          sum = sum+a[i];
      }
      printf("sum is %d", sum);
  }
```

Break and Continue:

```
while (test exp)
{ if (condition)
  {
      Statements;
      break; //continue
  }
}
Out of the loop
```

```
do
{
  if(condition)
  {
      break; //continue
  }
}while(test exp);
out of the loop
```

```
for(init; test exp; incr/decr)
{
  if(condition)
  {
      break; //continue
  }
}
out of the loop
```

```
#include <stdio.h>
void main()
{
    int psum=0, i, num;

    for (i=0 ; i<5 ; ++i)
    {
        printf(" Enter integer: ");
        scanf("%d", &num);
        if(num<0)
            continue;

        psum = psum+num;
    }
    printf("sum is: ", psum);
}
```

Using array, WAP to:

i/p : A, B, C, D, E

o/p : E, D, C, B, A.

```
void main()
{
    char inp[5], out[5];
    int i;
    printf(" Enter 5 charaters:\n");
    for(i=0; i<5; ++i)
        scanf("%c", inp[i]);
    for (i=0, j=4; i<5; ++i, --j)
    {
        out[i] = inp[j];
    }
    for (i=0; i<5; ++i)
        printf("%c", out[i]);
}
```

NUMBER SYSTEM:

Bit : 1 byte = 8 bits

1 KB = 1024 Bytes
= 2^{10} Bits

1 MB = 1024 KB
= 2^{20} Bits

1 GB = 1024 MB
= 2^{30} Bits

1 TB = 1024 GB
= 2^{40} Bits

1 PB = 1024 TB
= 2^{50} Bits.

1 ~~Eta~~ EB a Bytes = 1024 ~~Bytes~~ PB
= 2^{60} Bits.

$(25)_{10} = (?)_2$

2	25	1
2	12	0
2	6	0
2	3	1
2	1	1

$\therefore (25)_{10} = (11001)_2$

$(111111)_2 = (?)_{10}$

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64	32	16	8	4	2	0
×1	×1	×1	×1	×1	×1	×1

$64 + 32 + 16 + 8 + 4 + 2 + 1$

$(127)_{10}$

320
− 61
259

8 6 4 2 1
1 0 1

$(205)_8 = (?)_{10}$

205

010000101

10000101

128 0000401

133

Hexadecimal

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

16
×16
96
16×
256
×7
1792

13
×16
78
13×
208
+14
222

$(7DE)_{16} = (?)_{10}$

$7 \times 16^2 + D \times 16^1 + E \times 16^0$

$\rightarrow 7 \times 256 + 13 \times 16 + 14 \times 1$

$\rightarrow 1792 + 208 + 14$

$\rightarrow 1792 + 222$

$\rightarrow \boxed{2014}$

$\therefore (7DE)_{16} = (2014)_{10}$

$= (111101110)_2$

7 D E

0111 1101 1110

$(205)_8 = (?)_{10}$

205

010 000 101

$\underbrace{2\times8^2}_{2\times64} + \underbrace{0\times8^1}_{} + \underbrace{5\times8^0}_{5}$

$\boxed{128 + 5}$

$\rightarrow (133)_{10}$

$\rightarrow (1000010)_2$

21.53

2	21	1
2	10	0
2	5	1
2	2	0
	1	

$\rightarrow (10101)_2$

$0.53\times2 \longrightarrow 1.06$	1
$0.06\times2 \longrightarrow 0.12$	0
$0.12\times2 \longrightarrow 0.24$	0
$0.24\times2 \longrightarrow 0.48$	0
$0.48\times2 \longrightarrow 0.96$	0
$0.96\times2 \longrightarrow 1.92$	1

$\rightarrow (100001)$

$\rightarrow (21.53)_{10} \rightarrow (10101.100001)_2$

---

int

16 bit (2 byte) $\rightarrow -2^{16-1}$ to $2^{16-1}-1 = -2^{15}$ to $2^{15}-1$

$\rightarrow -32768$ to 32767

32 bit (4 byte) $\rightarrow -2^{32-1}$ to $2^{32-1}-1$

$\rightarrow -2147483648$ to 21

$\boxed{-2^{n-1} \text{ to } 2^{n-1}-1}$

unsigned int → 2 byte → ~~$(0 \to 2^{16})$ or = 2 * 32767 → 0 to 2^{16}~~

0 to $\boxed{65536}$ 0 to $2^{16}-1$

→ 4 byte → ~~$(2^{32}$ or = [illegible]~~

→ 0 to $2^{32}-1$

→ 0 to 4294967295

Q Difference between signed and unsigned character.

Signed char	unsigned char

---

* How to see size of our machine by knowing int values.

Use: sizeof().

```
printf(" %d ", sizeof(int));
```

TRY OUT

```
scanf("%4d", &x);
printf("%d", x);
```

x = 123456

$\boxed{\text{output: } 1234}$

Pf,

scanf	output
%3d;	123
%2d;	12

i.e, 3d, 4d here, 3,4 means print upto that digit

PROGRAM ERROR.

1) Syntax Error.
2) Semantic Error.
3) Logical Error.

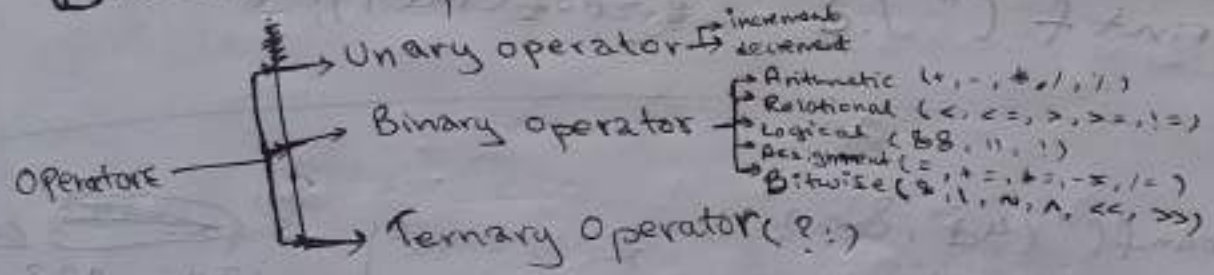
Compile-time error
Runtime error

Tokens:

1) Keywords: eg: int,
2) Identifiers: -> variables [eg (a)]
3) Constants: eg (a=5) 5 is constant value
4) Strings. array of characters
5) Special Symbols
6) Operators.

X ——— X ——— X ——— X ———

Bitwise Operators:



Bitwise Operators:

(i) Bitwise AND → &
(ii) Bitwise OR → |
(iii) Bitwise NOT → ~
(iv) XOR → ^
(v) Left Shift → <<
(vi) Right Shift → >>

① first operand << Second operand

whose bits are shifted

decides the number of places to shift bits

what if second operand is in negative? 3 << -2 ?

② when bits are shifted left, the trailing positions are filled with zeros.

Exclusive OR (XOR)

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	0

7 - 0000 0111
^
4 - 0000 0100
3 - 0000 0011
(3)

Character Arrays and Strings

1) Reading and Writing String
2) Combining Strings together
3) Copying string to another
4) Comparing Strings for equality
5) Extracting a portion of a string.

Syntax: char String-name [size];

Input: char str[10]; scanf("%s", str); //No & sign

printf("%s", str); // Prints upto blank character.

#
```
char str[80];
scanf("%[^\n]", str);
printf("%s", str);          //Print whole sentence.
```

#
```
gets(string-name);
```

#
```
void main()
{
    int i;
    char str[80];
    printf("Enter a string:\n");
    gets(str);

    for (i=0; str[i]!='\0'; ++i);
    printf("No. of characters= %d", i);
}
```

#
```
void main()
{
    char str1[80], str2[80];
    int i;
    printf("Enter a string:\n");
    gets(str1);
    for(i=0; str1[i]!='\0'; ++i)
        str2[i]= str1[i];
    printf("String Copied");
}
```

```
#include<stdio.h>
void main()
{
  char str[80];
    gets(str);
  int  vow=0, cons=0, i;

  for (i=0 ; str[i]!='\0' ;++i)
   { if (isalpha(str[i]))
     {if ( tolower(str[i])=='a' || tolower(str[i])=='e' ||
          tolower(str[i])=='i' || tolower(str[i])=='o' ||
           tolower(str[i])=='u')
              ++vow;
        else
              ++cons;
     }
   }
  printf(" Vowels = %d and Consonants = %d ", vow, cons);
}
```

(ctype.h) → tolower 'A' 'a'

toupper 'a' 'A'

a A

include <String.h>

→ atoi (str) : string converts to integer.

→ strcat()

→ strcmp() / strcmpi()

strcpy()

strlen()

Function Call:

- Actual Parameters / Parameters
- Formal Parameters / Arguments.

```
main()
{
    function( a1, a2 );   // Parameters - function call
}

function (f1, f2);   // Arguments.
```

Category of Function:

Assignment

1) Function with no arguments and no returntype.
2) Function with arguments but no return value
3) Function with arguments and one return value
4) Function with no arguments but one return value
5) Return multiple values.

Nested Function: When a called function in turn calls another function and 'chaining' occurs, it is known as nested function.

Recursion: When a called function in turn calls itself, it is known as recursive function.

Monday
Submission Date

Q. Factorial using only recursion:

→
```
int fact ( int num)
{
    int factorial=1;
    if ( num > 0)
    {
        factorial = factorial * num;
        --num;
        fact (num);
    }
    else return factorial;
}
```

Q. Fibonacci Series upto 50.

→ ~~void fibo ( )~~

Passing Arrays to function:

→
```
# include <stdio.h>
float ~~low~~ largest ( float arr[ ], int size)
{
    int i;
    float larg;
    larg = arr[0];
    for (i=1 ; i<size ; ++i)
    {
        if( arr[i] >larg){
            larg = arr[i];
    }
    return (larg);
}
void main()
{
    ~~float~~ arr[ ]; ~~size~~ int size,
    // Takes size and values from user
    float larg;
    larg = ~~fo~~ largest (arr , size);
    printf("Largest value is %f ", larg);
}
```

Three Rules to pass an array to a function

1. The function must be called by passing only the name of array.
2. In the function definition of the formal parameter must be an array type, the size of the array does not need to be specified.
3. The function prototype must show that argument is an array.

STRUCTURE

```
struct tag_name
{
    data_type member;
    data_type member;

};
```

e.g.:

```
struct book_bank
{
    char title[20];
    char author[15];
    int page;
    float price;
};
```

size of struct: sizeof(struct tag-name);

→ printf("%d", sizeof(struct-t

```
void main()
{
    struct book_bank book1, book2;
}
```

```
struct personal
{
    char name[20];
    int day;
    char month[10];
    int year;
    float salary;
};

main()
{
    struct personal p;
    scanf("%s %d %s %d %f", p.name, &p.day,
            p.month, &p.year, &p.salary);

    printf("%s %d %s %d %f", p.name, p.day,
            p.month, p.year, p.salary);

    sizeof(p);          //prints size of structure.
```

→ Struct st_record

```
{
    int weight;
    float height;
};
void main()
{
    struct st_record student = {60, 180};
    struct st_record stud2 = {70, 170};
    printf("%f", stud2.height);
}
```

```
for (i=0; i < 60 ; ++i)
{
    printf("Enter details of Student %d", (i+1));
    printf("Name- ");
     gets (stu[i].name);
    printf(" Roll No.- ");
     scanf ("%d", stu[i].roll);
}
```

Assignment

60

Struct

1) Name
2) Reg. No.
3) Mobile No.
4) Semester
5) Course [will be another struct]
   3 subs

NESTED STRUCTURE

```
→ # include <stdio.h>
  # include <string.h>
  struct  student_college_details
  {
      int college_id;
      char  college_name[50];

  };
  struct  student_details
  {
        int id;
        char name[20];
        float percentage;
        struct student_college_details clg_data;
  } stu_data;
```

Passing structure to function

1. Passing structure to a function by value
2. Passing structure to a function by address (reference)
3. No need to pass a structure
   - Declare structure as global variable.

1) → Passing structure to a function by value.

```
struct student
{
    int id;
    char name[20]
    float percentage;
};

void func (struct student record);

main()
{
    struct student record;
    record.id = 1;
    strcpy(" record.name, "Raju");
    record.percentage = 80.00;
    func (record);
}

void func (struct student record)
{
    printf("Id is %d", record.id);
    printf("Name is %s", record.name);
    printf("Percentage is %f", record.percentage);
}
```

Q. WAP to store details of 60 students using structure.

```
# include <stdio.h>
struct course_plan
{
    char course [30];
};
struct student_details
{
    char name [30];
    int regno, mobile, sem;
    struct course_plan C[3];
};

void main()
{
    Struct student_details S[60];
    int i, j;
    for( i=0 ; i< 60 ; ++i)
    {
        printf("Enter details\n");
        printf(" Enter name ");
        gets ( S[i].name);
        printf(" Enter reg No. : ");
        scanf ("%d ", & S[i].regno);
        printf("Enter mobile number : ");
        scanf ("%d ", & S[i].mobile);
        printf("Enter semester : ");
        scanf ("%d ", & S[i].sem);
        printf("Enter course plan\n");
        for( j=0 ; j<3 ; ++j)
        {
            printf("Course %d ", j);
            gets( S[i].C[j].course);
        }
    }
}
```

UNION.

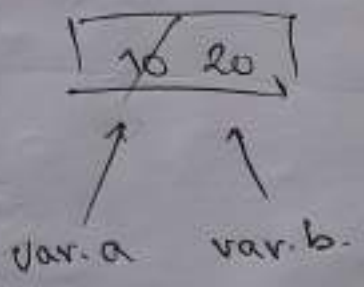
```
struct identity
    {   data_type ele1;
        data_type ele2;
        :
    };
```

Allocates memory location for each data type.

```
Union  Un
    {
        data_type ele1;
        data_type ele2;
        :
    };
```

Allocates memory size for only one & allocates only one memory size equal to largest datatype.

```
union un
    {
        short a;
        short b;
    };
main()
    {
        union un var;
        var.a = 10;
        printf("%d", var.b);        // print 10
        var.b = 20;
        printf("%d ", var.a);       // print 20.
    }
```



* At a time, we can use only one variable in the entire program. (Union).

Storage Class:

* Variable Characteristics:

(i) datatype. – int / float / ...

(ii) Memory – RAM / Registers ...

(iii) Storage Class – auto, register, static, extern.

(iv) Scope

(v) Lifetime

STORAGE CLASS

- auto
- register
- static (imp)
- extern

* Scope: Region of the program where variable can be used. Basically, it is the region enclosed between any two curly braces.

* Lifetime: How long variable holds the memory

* SYNTAX:

storage_class datatype variable;

- auto int a;
- register int a;
- static int a;
- extern int a;

* AUTO STORAGE CLASS:

Scope: In the same block in which variable has been declared.

Lifetime: Only in the block in which variable has been declared.

default value: garbage value.

```
auto int a = int a
```

* STATIC STORAGE CLASS:

Scope: In the same block in which the variable has been declared.

Lifetime: Until the completion of the program the variable will be alive.

Default value: Zero.

---

①
```
inc();
main()
{
    inc();
    inc();
    inc();
}
inc()
{
    auto int a=0;
    a = a+1;
    printf("%d\n", a);
}
```

②
```
inc();
main()
{
    inc();
    inc();
    inc();
}
inc()
{
    Static int a=0;
    a= a+1;
    printf("%d\n",a);
}
```

Q. output: 1
1
1

② output: 1
2
3

~~Static declaration~~

Pg 190, 191, 192 : Let us C

Register STORAGE CLASS:

The compiler will allocate one register to store the variable (if available), else the values will be sent to RAM or other memory location).

Purpose : fast access during the run-time.

Scope : within the block in which variable has been declared

Lifetime : only within the block.

default value: garbage.

Extern STORAGE CLASS : (Similar to global variable).

extern int i;

Scope : Throughout the program.

Lifetime: Alive until the program completion.

default value: zero.

Q. Difference between extern global variable and normal global variable.

```
int x = 10;
main()
{
    int a=3;
    extern int b;
    printf("%d", a);  → 3
    printf("%d", b);  → 10
    printf("%d", x);  → 10
}
b = 10;
```

→ output → | 3 10 10 |

Pointers

- derived datatype in C
- contain memory location/addresses as their values.
- Syntax:

  data-type *pt_name;

- Types:
  - typed: int *p;
  - untyped: void *p

* ⟶ value operator [pointer variable returns value].

& ⟶ Address of a particular variable.

#

```
int a=2;
int *p; // declaration of pointer
p=&a; // initialisation of pointer
```

#

```
int a, b, sum=0;
int *p1, *p2;
a=5; b=6;
p1=&a;
p2=&b;
int sub=p2-p1; // = 1 (why?)
```

Summation of addresses, multiplication and division of addresses are not possible/valid.

only subtraction is valid.

#

```
main()
{   int i=100;
    int *ptr;
    ptr=&i;
    printf("%d", i);      // → 100
    printf("%d", ptr);    // → address value
    printf("%d", &i);     // → address value
}
```

%u

// %u to access unsigned address (always +ve)

Chaining of Pointer:

```
int x, *p1, **p2;

x = 10;
p1 = &x;
p2 = &p1;
```

Q WAP to swap 2 numbers using func

→

```
# include <stdio.h>
void swap( int* a , int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
void main()
{
    int p = 10, q = 20;
    swap(&p, &q);
}
```

address

```
void main()
{ int a=10, b=20;
swap (a,b);
```

→

```
# include <stdio.h>
void swap ( int a , int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf("%d %d", a, b);
}
main()
{ int a=10 , b=20;
  swap(10, 20);
}
```

C Preprocessor :

* Pre-processor Directives :
  - → file inclusion → e.g: #include<stdio.h>
  - → conditional compilation → Eg: #if, #ifdef
  - → Macros → Eg: #define MAX 100

Macro :
- → Simple Macro Substitution
- → Argumented Substitution
- → Nested Macro Substitution.

* Simple Macro Substitution:

E.g.:
```
#include <stdio.h>
#define VAL 10;
main()
{
    int d;
    d=VAL-5;
    printf("%d", d);
}
```

* Argumented Substitution.

E.g.:
```
#define PRODUCT(x,y) (x*y)
main()
{
    int result = PRODUCT (4,5);
    printf("%d", result);        // Ans: 20
    int rst = PRODUCT (2+2, 3+2);
    printf("%d", rst);           // Ans: 10
}
```

To overcome this, we can define like:
#define PRODUCT(x,y)((x)*(y))

→ Reason: 2+2*3+2 = 2+(2*3)+2 = 10

* Nested Macro Substitution:

Eg: #define M 2
#define N M+2

Q. #define MAX(a,b) ((a)>(b)?(a):(b))

```
main()
{
    int i=5, j=6;
    int y = MAX(i++, j++);
    printf("%d", y);
}
```

117

Q: #define INC(x) ++x

```
main()
{
    char *ptr="Hello";
    int x=10;
    printf("%s", INC(ptr));
    printf("%d", INC(x));
}
```

FILE HANDLING:

- → Create
- → Open
- → Input/output /Operations
- → Close.

* FILE * <identity>;

eg. FILE *fp;

* fopen ("path", "Mode");
  - → "r"
  - → "w"
  - → "a" → for append mode, Pointer points last character.

* If file doesn't exist, pointer will be assigned NULL value.

```
* while((ch = fgetc (fp)) != EOF);
        printf ("%c", ch);
```

* fputc (ch, fp);

Q. WAP to copy texts from one file to another.

→

```
main()
{
    FILE *p1, *p2;

    p1 = fopen("abc.txt", "r");
    p2 = fopen("copy.txt", "w");

    char c;
    while((c = fgetc(p1)) != EOF)
    {
        fputc(c, p2);
    }

    p1 = fclose();
    p2 = fclose();
}
```

fputc(p1, p2);

Q. ~~WAP to count frequency of each word in a file.~~

→

```
main()
{
    FILE *f1;
    f1 = fopen("input.txt", "r");

    char word[10], c;

    int i;
    while((c = fgetc(p1)) != EOF)
    {
        i = 0;
```

Q. WAP to count frequency of each word in a file.

→

```
main()
{
    FILE *f1, *f2;
    char * str;
    char word[10];
    f2 = fopen
    f1 = fopen ("input.txt", "r");

    while( !feof( f1))
    {
```