UNIT - II.

C PROGRAMMING BASICS.

History of C.

* Structured, high-level, machine-independent language.
* Root of modern languages is ALGOL (1960, First computer language to use block structure). ↓ used in Europe
  (structured pgming)
* 1967 - Martin Richards developed a lang. BCPL (Basic Combined Programming lang) primarily for writing SS.
* 1970 - Ken Thompson created a language with features of BCPL called B.
* B - used to create early version of UNIX OS @ Bell lab
* B & BCPL - "Typeless" system programming languages.
* Dennis Ritchie (ALGOL, BCPL, B) 1972 ← C language
  @ Bell Lab
* C uses many concepts from these languages & added up datatypes, other powerful concepts.
* Unix is associated with C. C is used in academic environments, it runs under variety of operating sty & H/w platforms.
* 1979, C is evolved into "traditional C".
* 'The C programming language' book - Dennis Ritchie & Brain Kernighan (1978).
  K&R C
* 1983 - American National Standards Institute (ANSI) appointed a technical committee to define a standard for C.
  It was approved in Dec 1989. C89
  approved by ISO in 1990.

* C ++ added new features to C - to make it a Versatile and object-oriented language.

* During this period Sun Microsystems of USA created a new lang Java

* All languages are Dynamic in Nature.

1960	ALGIOL	International Group
↓		
1967	BCPL	Martin Richards
↓		
1970	B	Ken Thompson
↓		
1972	Traditional C	Dennis Ritchie
↓		
1978	K & R C	Kernighan & Ritchie
↓		
1989	ANSI C	ANSI Committee
↓		
1990	ANSI/ISO C	ISO Committee
↓		
1999	C99	Standardization committee

Importance of C

* Robust
* Built-in functions & operators - complex pgm.
* C Compiler - capability of Assembly language + Features of high-level lang

↓

Suited for writing both ss & business package.

- pgms in C - Efficient & fast [Datatypes + operators]

* Faster than BASIC.

Ex: Increment a Variable 0 - 15000 take 1 sec in C
0 - 15000 take more than 50 sec in BASIC (Interpreter)

* 32 keywords in C, Built-in functions
* Highly portable - C pgm written in one computer can run on another with little/no Modification
* Structured programming. - function modules.
* Extends itself. C is a collection of function supported by C library

Example programs.

```
#include <stdio.h>
#include <conio.h>
main()
{
 /* printing */
 printf("C programming");
}
```

C pgm is divided into modules/functions.

functions are written by User, stored in C library.

Library fns are grouped category-wise and stored in different files known as header files.

#include <filename>

preprocessor directive.

* #define ← preprocessor directive - Symbolic constants.

* main:

Different forms of main are:

- main()
- int main()
- Void main()
- main(void) → No arguments.
- Void main(Void)
- int main(Void).

No arguments.

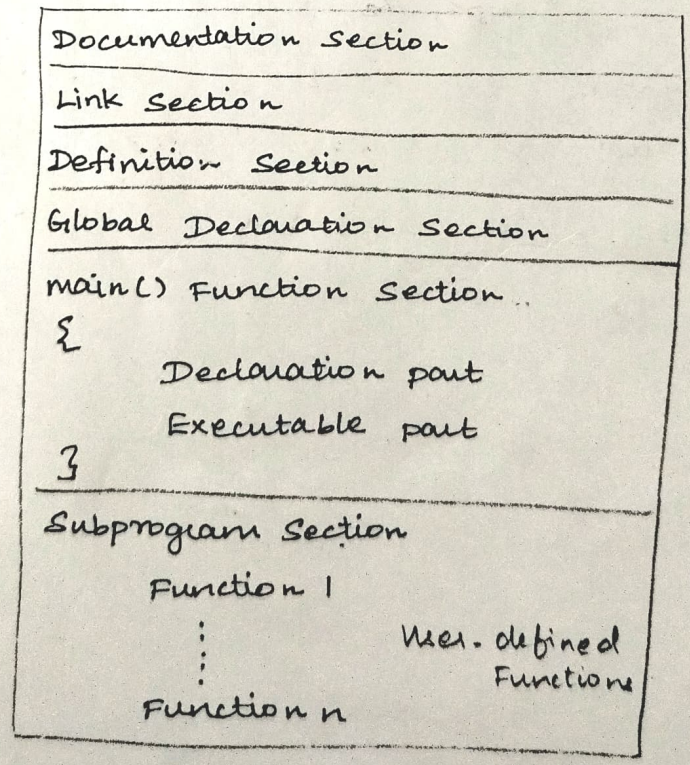
Void - doesn't return any information too

int - returns an integer value to OS.
..... return 0 ← last stm

Structure of C:

* C - group of building blocks called functions.

* Function - Subroutine that may include 1/more stmts to perform specific tasks.



C Structure.

Documentation - Set of comment lines (Name of the pgm, author & other details). /* .... */

* Link - provides instn to the compiler to link functions from system library.

* Definition - defines all symbolic constants.

* Global Variable - Variable that can be used in more than one functions.

(User-defined fns are also included in this section).

* Main function Section
  - Declaration part — a = 5, b = 3.
  - Executable part. — a = b * c

Subpgm → user-defined fn which are called in main fn.

Programming Rules.

* Rules to be followed are:

  ① All statements in C - lower case. Symbolic Constants alone uppercase letter can be used.

  ② Blank space inserted b/w words. Space not allowed in declaration of variable, keyword, constant & function

  ③ Two/three Stmts are allowed in a single line seperated by semicolon.

Executing a C program.

* Executing a pgm written in C involves - series of steps

① Creating a program

② Compiling a program

③ Linking the pgm with functions that are needed by C library

④ Executing the program.

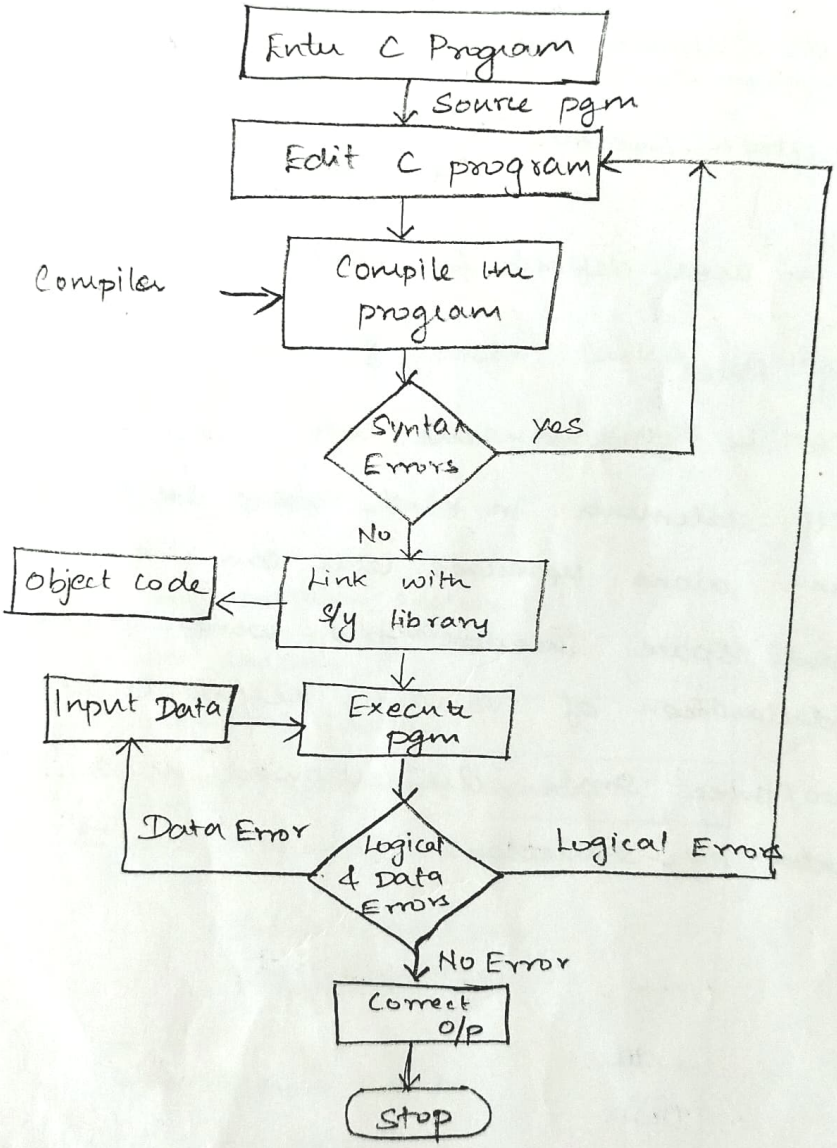


Fig: process of compiling & running a C program.

(i) Creating - writting & editing in c editor. (.c extension)

(ii) compiling

converts high level lang to machine language

ALT + F9 / Compile option

Errors: Syntax errors - Stmts are wrong.

(iii) Linking

C lang - collection of predefined functions. where functions are written in header files.

Linking with s/y library is important. This is automatically done at the time of execution.

(iv) Executing. (Ctrl + F9) / Run

Running & testing the pgm with sample data.

2 Errors: Logical & Data Errors.

↓

error after some sequ. ← cndtn & ctrl stmts execution

Constants, Variables & Data types.

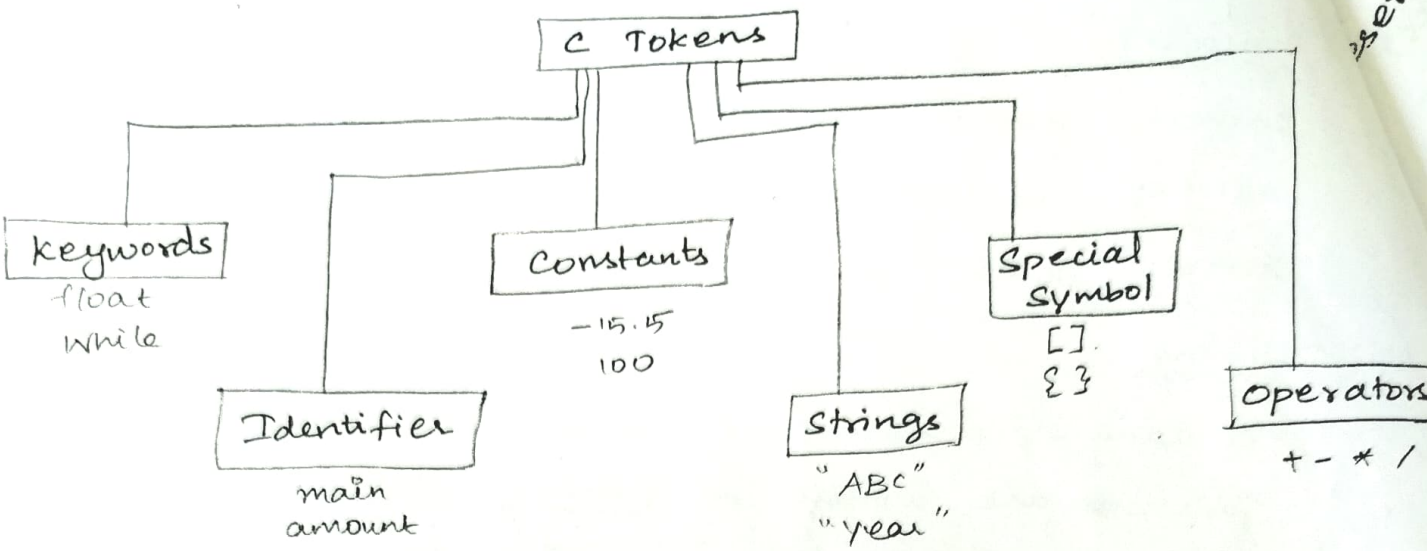
* Pgm - Sequence of instn

- instruction (formed using certain Symbols & words according to some rigid rules (Syntax rules)).

* characters is C are grouped into:

(i) Letters
(ii) Digits
(iii) Special characters
(iv) White Spaces.

* C Tokens. Smallest individual units are known as C Tokens. (words/punctation).



Identifier

* In c language, every word is classified into either a keyword / identifier.

* Names given to various pgm elements such as variables, functions & arrays.

Rules - Identifier

(i) Consists of letters & digits in any order (Considered as letter ↓)

(ii) must begin with a letter / character / Underscore (_)

(iii) Uppercase & lowercase letters are allowed.

(iv) Identifier can be of any length (most 'c' compiler recognizes 1st 31 characters).

(v) No Space & Special Symbols are allowed.

(vi) Identifier cannot be a keyword.

Examples.

STDNAME, TOT_MARKS, _TEMP, Y2K.

Not allowed: IREC, STD NAME.

word.

Reserved words - standard & predefined meaning in 'C' which cannot be changed & they are building blocks for program structure.

Examples.

auto, break, case, default, char, do, float, int, void

Variables

* Variable is an identifier which is used to represent some specific type of information within a particular portion of program.

Variable ← different values at different times during the execution.

Rules for naming the variables.

(i) Variable name - combination of 1to 8 alphabets, digits / underscore.

(ii) first character - Alphabet / Underscore (_).

(iii) Length of variable cannot extend upto 8 characters long, & some can recognize upto 31 characters long.

(iv) No commas / blank spaces allowed within a variable name.

(v) No special symbol, an underscore (_) can be used in variable name.

Variable Declaration.

* After designing variable names, Declare them in pgm & this declaration tells the compiler what variable name & what type of data it can hold.

Syntax: data_type $v_1, v_2, v_3 \ldots v_n$;

data_type is the type of data ←

$v_1, v_2, v_3 \ldots v_n$ — list of variables.

Example: int code; char sex; float price;

char name[10] (array of characters)

Initializing Variables.

* Initialization - Assignment operator (=).
* Initialization can be done while declaring variables.

Syntax: Variable = constant; / datatype variable = const;

Example: int i = 5; char c = 's';

float f = 29.77;

Users-defined variables: C provides a feature to declare a variable of type of user-defined. It allows users to define an identifier that represents existing data type & this can later be used to declare variables.

Type Declaration ⇒ Syntax: typedef data-type identifier;

User defined type declaration.

↳ identifier refers to new name given to the datatype.

Example: typedef int marks;

marks m1, m2, m3;

Enumerated Data type. (User-defined data type) define own datatype where its variable can take value.

Syntax: enum identifier {value1, value2, ... valuen}

User-defined enumerated Datatype

Enumerated Constant

Example: enum day {mon, tue, ... sun}

enum day w-st, w-end;

w-st = mon;

w-end = sun;

enum ... have only one value from constant

ype of Variables.

Availability of variables within the progeam.

Two types of scopes - local & Global

① Local Variables.

Variables defined inside function blk/ compound stmt is called local variables.

Ex:
```
function()
{
    int i, j;      → local variables
    /* body of function */
}
```

② Global / External Variables.

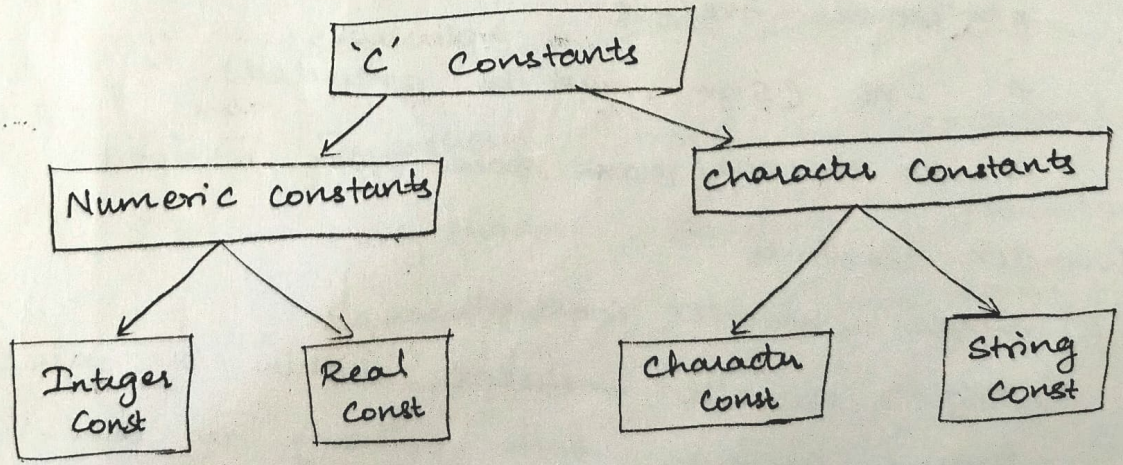
Variables declared before function main(). these variables are available for all functions inside the pgm.

Ex:
```
int a, b = 2;      → Global variables.
main()
{
    ...
    fun();
}

fun()
{
    int sum;
    sum = a+b;
}
```

Constants.

Item whose value cannot be changed during execution is called constants.



Numeric Constants.

① Integer Constant. formed with sequence of digits

3 types: Decimal Number - 0 to 9
(10, 153, -321)
Octal Number - 0 to 7
(052, 0541)
Hexadecimal Number - 0 to 9, A, B, C, D, E, F.
(0x4, 0x8F, 0xBCF)

Ex: marks = 90; discount = 15;

Rules:

* Decimal point is not allowed.
* can either be +ve/-ve. -ve (sign must be preceded) -51
* Must have atleast one digit.
* No commas / Blank spaces are allowed.
* Range (-32,768 to 32,767).

② Real Constants. It is made up of sequence of numeric digits with presence of a decimal point.

To represent quantities that vary continuously such as distance, height, temperature, etc..

Ex: distance = 126.0;
height = 5.6;

Rules:

* Must have one digit
* Must have decimal point
* Either +ve/-ve.
* -ve (sign must be preceded).
* No commas/ Blank spaces are allowed.

Character Constants.

① Single character constants.

Single character enclosed within a pair of single inverted commas both pointing to left.

Ex: 's', 'M', '3'.

(2) String constants.

Sequence of characters enclosed in double quotes, the characters may be letters, numbers, special characters & blank spaces. At the end of string '\0' is automatically placed.

Ex: "HI", "Nuni", "39.77", "50".

(3) Declaring a variable as constant.

When the value of some of the variables may remain constantly during execution of program, in such a situation this is done by using const keyword.

Syntax: const datatype variable = constant

keyword to declare constant

Example: const int dob = 3977;

Delimiters.

Symbols, which has some syntactic meaning & has got significance. Doesn't specify any operation.

#	Hash	pre-processor directive
,	Comma	seperate list of variables.
:	Colon	label Delimiters
;	Semicolon	stmt Delimiters
()	Parenthesis	Used in Expression/fn.
{}	Curly braces	Blocking 'c' structure
[]	Square brats	Used with arrays.

Data types. Type of data, 'C' support various data types. each type may have predefined memory requirement, operators, variables, Storage representation.

① Primary — const, int, float, double

② Userdefined — typedef, enum.

③ Derived — arrays, pointer, structures, union

④ Empty — void → has no value
→ doesn't return any value

① Primary Datatype.

int - 2bytes (-32768 to 32767) 16-bit compiler — 16th bit - sign indication. %d or %i
32-bit compiler -2147483648 to 2147483647

char - 1byte (-128 to 127) %c Ex: char s = 'n';

float - 4bytes (3.4E-38 to 3.4E+38) %f or %g Ex: float f = 29.77;

double - 8bytes (1.7E-308 to 1.7E+308) %lf Ex: double d = 29107511 14

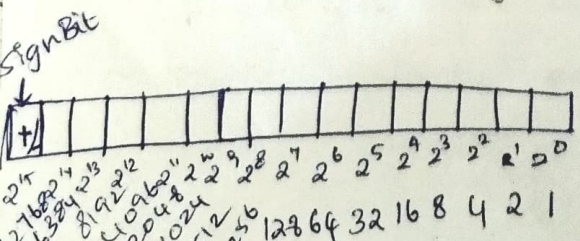
Primary Datatype is dirided into:

Integer — signed / unsigned
[int, short int, long int]

char — char / signed char (-128 to 127) / unsigned char (0 and 255)

Float — float / double / long double

Refer Balaguruswamy Book for reference to various Datatypes



$2^0 = 1$ (0,1)

$2^{15} = 32,768$ (signed)

$2^{16} = 65536$ (unsigned)

0-00 1-01 2-10 3-11

0-000 1- 2- 3- 4- 5- 6- 7-

Operators & Expressions.

Operators are used in programs to manipulate data & variables. (Mathematical / logical manipulations).

* Data item that operators act upon are called operands.

* Unary operator, Binary operator.

Ex: a+b

a, b - operands

+ - operator.

* Types.

(i) Arithmetic
(ii) Relational
(iii) Logical
(iv) Assignment
(v) Increment & Decrement
(vi) Conditional
(vii) Bitwise
(viii) Special

(1) Arithmetic operator.

+	Addition / Unary plus	2+3 = 5
-	Subtraction / Unary Minus	3-2 = 1
*	Multiplication	3*2 = 6
/	Division	6/3 = 2
%	Modulo Division	7%3 = 1

* Classification:

① Unary arithmetic - +x, -y

② Binary arithmetic - x+y

③ Integer arithmetic - a = 5, b = 4

2 operands should be integers.

Ex: a/b = 5/4 = 1

Here the decimal part is truncated.

④ Real arithmetic - operands are real.

Ex: x = 6.0/7.0 = 0.857143.

Mixed-mode Arithmetic: one operand is real & other integer. If any of the operand is real, result is real

Ex: 2/5 = 0

Involves Real as 1 operand:
- 5.0/2 = 2.5
- 5/2.0 = 2.5
- 5.0/2.0 = 2.5

→ Result will be Real.

Example.

```
#include <stdio.h>
#include <conio.h>
Void main()
{
    /*Local definitions*/
    int a,b,c,d;
    int sum, sub, mul, rem;
    float div;
    /*stmts*/
    clrscr();
    printf("Enter values of b,c,d:");
    scanf("%d %d %d", &b, &c, &d);
    Sum = b+c;
    Sub = b-c;
    mul = b*c;
    div = b/c;
    rem = b%d;
    Printf("\n Sum = %d, Sub = %d, mul = %d, div = %f",
                    Sum, Sub, mul, div);
    getch();
}
```

Output: Enter values of b, c, d: 3, 5, 10

Sum = 8 Sub = -2 mul = 15 div = 0.3

3/10

Relational Operators.

Used to compare 2/more operands. operands - variables, constants or expressions.

Ex: Compare age of 2 persons.

			RESULT
<	less than	2 < 9	1 (Return value;
>	Greater than	2 > 9	0 False
<=	less than or equal	2 <= 2	1 True
>=	Greater than or equal to	2 >= 3	0
==	Equal to	2 == 3	0
!=	Not Equal to	2 != 3	1

Relational operator complements:

> is complement of <=

< is complement of >=

== is complement of !=

Ex: ! (x < y) ⇒ x >= y.

! (x > y) ⇒ x <= y.

③ Logical operators.

* Used when we want to test more than one cndtn & take a decision.

&& logical AND (exp1) && (exp2).

|| logical OR ! (exp) ⇒ if(! ($c_1 < c_2$))

! logical NOT.

(A) Assignment operator.

* Assign a value to an variable, value of a variable to another variable.

Syntax: Variable = expression (or) value;

Example: x = 10;

y = a+b;

(a) Compound Assignment - assign a value to a variable

Shorthand operator

x += y	x = x+y
x -= y	x = x-y.

(b) Nested / Multiple Assignments.

Var1 = Var2 = ... = Varn = Single Variable or expression;

i = j = k = 1;

x = y = z = (i+j+k);

(5) Increment & Decrement operators (Unary Operators).

++ → Add 1 to variable.

-- → Sub 1 from variable

Hence it is called

++x	pre increment	
--x	pre decrement	
x++	post increment	11, 11, 10, 10
y--	post decrement.	

Example: a = 10

		Now value of a is
a++	10	10+1 = 11
a--	11	11-1 = 10
--a	9	10-1 = 9
++a	10	9+1 = 10

Conditional / Ternary operator.

checks the condition & executes the statement depending on the condition.

Syntax: condition ? exp1 : exp2; ← ternary operator

↓

True exp1 is evaluated

False exp2 is evaluated.

```
Example: main()
{
    int a=5, b=3, big;
    big = a>b ? a : b;
    printf("Big is ... %d", big);
}
```

O/P : Big is ... 5

④ Bitwise operator. used to manipulate data at bit level. It operates on integers only. It may not be applied to float/real

a = 5 & 6;

a = 4

5 - 0101

6 - 0110

0 1 0 0

8 4 2 1

&	Bitwise AND	5 & 6 → 0100
|	Bitwise OR	
^	Bitwise XOR	
<<	Shift left	a = 7; a << 2. → 10000
>>	Shift right.	14 a >> 2
~	one's complement.	

7 << 1 → 1110

a << 2 → 10000

a >> 2

Example: Bitwise AND (&).

x = 7 = 0000 0111

y = 8 = 0000 1000

x & y = 0000 0000.

&	0	1
0	0	0
1	0	1

Bitwise OR

|	0	1
0	0	1
1	1	1

Bitwise XOR

^	0	1
0	0	1
1	1	0

x	y	x^y
0	0	0
0	1	1
1	0	1
1	1	0

1000 ^ 0100 → 1100

⑧ Special operator.

Comma operators ,

Sizeof operators sizeof

Address of variable ← & * → value of a variable

pointer operators

. --> Member selection operator.

↳ Access the elements from the structure.

(i) Comma operator. used to seperate the stmt elements such as variables, constants or expression.

Link the related expressions together.

Ex: Val=(a = 3, b = 9, c = 77, a+c); 3 is assigned to a

9 is assigned to b

77 is assigned to c

a+c = 80.

(ii) Sizeof() is a unary operator, returns length in bytes of the specified variable. Helps in finding the bytes of a variable.

Syntax: sizeof(var);

Example:

```
main()
{
    int a;
    printf("size of a is %d", sizeof(a));
}
```

<u>O/P</u>: size of a is 2.

naging I/o Functions (I/p, process, O/p)
└ essential features of computer.

2 ways to give input to pgm:

① Assigning data to variable

② I/o stmts. C I/o operations --fn. call stdio).

- Unformatted I/o stmts
- formatted I/o stmts.

① Unformatted I/o statements.

* Cannot specify the type of data.
* I/o a single/group of characters from/to I/o devices.

Input	output
getc()	putc()
getchar()	putchar()
gets()	puts()

(i) getchar() Single character input. char var = getchar();

```
char x;
x = getchar();
```

(ii) putchar() Single character output. putchar(char var);
displays one character at a time.

```
char x; putchar(x);
```

(iii) getc() accept a single character from standard input to a character variable.

```
char c = getc();
```

(iv) putc() display single character variable to std. o/p

```
putc(c);
```

getc() & putc() are used in file processing.

v) gets() & puts()

used to read the string, display/write string to o/p screen

gets(s); puts(s);

character test functions.

check the character taken as input.

isalpha(ch)	isupper(ch)
isdigit(ch)	tolower(ch)
islower(ch)	toupper(ch).

Example:

```
char x;
printf("\n Enter any alphabet");
x=getchar();
if (islower(x))
        putchar (toupper(x));
else
        putchar (tolower(x));
```

(2) Formatted I/O stmts.

Input & output arranged in a particular format

I/P	O/P
scanf()	printf()
fscanf()	fprintf()

(i) scanf()

scanf("control string", &var1, &var2 ... &varn);

control string
- character groups
- % (conversion character).

%c, %f, %d, %s

Example

getchar() & putchar()

```
Void main()
{
    char a;
    printf ("\n Enter a character");
    a = getchar();
    printf ("\n Displaying that character");
    putchar(a); putchar ('A');
}
```

gets() & puts()

```
main()
{
    char a[10];
    puts ("Enter string");
    gets (a);
    puts (a);
}
```

While loop (print n numbers).

```
# include <stdio.h>
# include <conio.h>
Void main() {
int i=1, n; clscr();
Printf ("Enter n");
Scanf ("%d", &n);
While (i<=n)
{
   printf ("%d", i);
   i++;
}
getch();
}
```

do-while loop

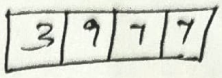
```
# include <stdio.h>
# include <conio.h>
Void main() {
int i=1, n; clscr();
printf ("Enter n");
Scanf ("%d", &n);
do {
 printf ("%d", i);
 i++;
} while (i<=n);
getch();
}
```

printf ()

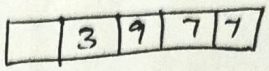
printf ("control string", var1, ... varn);

field width

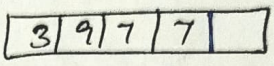
printf("%d", 3977)



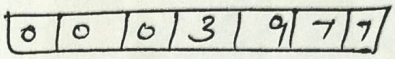
%5d, 3977



%-5d, 3977

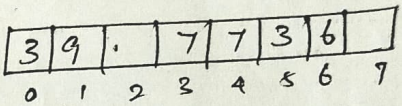


%07d, 3977

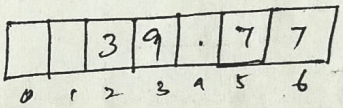


a = 39.7736

%7.4f, a



%7.2f, a



Decision Making

Control statements.

* pgm - all stmts are executed sequentially.
* No repetition cases.
* Repetition / Execution order of stmts is changed based on conditions.
  "conditional / ctrl stmts".

4 types of ctrl structures:

1. Sequential
2. Selection
3. Iteration
4. Encapsulation

① Sequential structure. instn executed in a sequence

```
i = i+1 ; j = j+1 ;
```

② Selection structure sequence of instn are executed by deciding upon the condition.

```
if (x > y)
        i = i+1;
else.
        j = j+1;
```

③ Iteration structure stmts are repeatedly executed.

```
for (i=1; i<=5 ; i++)
{
    i = i+1;
}
```

④ Encapsulation structure Compound structure.

Decision Making Statements:

① if stmt

Control the flow of execution of stmts. Condition

Syntax:

```
if (Condition)
{
    Stmts ;
}
```

(iv) check equivalance of two nos. if (m-n==0).

Example: (i) Check whether the number is less than 5

(ii) Swap two values. when first no. is greater than two (2nd no). (iii) Print no. b/w 10 & 15

2 variables

```
a = a+b
b = a-b
a = a-b
```

```
int a, b, c;
if (a > b)
{
    c = a;
    a = b;
}
b = c;
```

	a	b	c
	10	5	
	5		10
	5	10	

Managing I/O Functions (I/p, process, O/p)

└ essential features of computer.

2 ways to give input to pgm:

① Assigning data to Variable

② I/o stmts. (I/o operations - fn call stdio)

- Unformatted I/o stmts
- formatted I/o stmts.

① Unformatted I/o statements.

* Cannot specify the type of data.
* I/o a single/group of characters from/to I/o devices.

Input	Output
getc()	putc()
getchar()	putchar()
gets()	puts()

(i) getchar() Single character input. char Var = getchar();

```
char x;
x = getchar();
```

(ii) putchar() Single character output. putchar(char Var);

displays one character at a time.

```
char x; putchar(x);
```

(iii) getc() accept a single character from standard input to a character variable.

```
char c = getc();
```

(iv) putc() display single character variable to Std. O/p

```
putc(c);
```

getc() & putc() are used in file processing.

v) gets() & puts()

used to read the string, display/write string to dp dev

gets(s); puts(s);

printf

character test functions:

check the character taken as input.

isalpha(ch)	isupper(ch)
isdigit(ch)	tolower(ch)
islower(ch)	toupper(ch).

Example:

```
char x;
printf("\n Enter any alphabet");
x=getchar(x);
if (islower(x))
        putchar (toupper(x));
else
        putchar (tolower(x));
```

② Formatted I/O stmts.

Input & output arranged in a particular format.

I/P	O/P
scanf()	printf()
fscanf()	fprintf()

(i) scanf()

scanf("control string", &var1, &var2 ... &varn);

control string = character groups + % (conversion character).

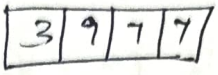
%c, %f, %d, %s

printf ()

printf ("control string", var1, ... varn);

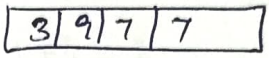
field width

printf("%d", 3977)

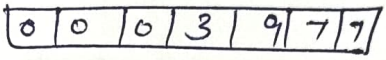


%5d, 3977

%-5d, 3977

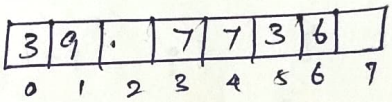


%07d, 3977

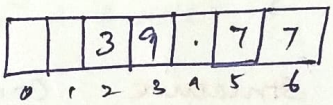


a = 39.7736

%7.4f, a



%7.2f, a



Decision Making

Control statements.

* pgm - all stmts are executed sequentially.
* No repetition cases.
* Repetition / Execution order of stmts is changed based on conditions.

"Conditional / ctrl stmts".

4 types of ctrl structures:

1. Sequential
2. Selection
3. Iteration
4. Encapsulation

① Sequential Structure. instn executed in a sequence

```
i = i+1 ; j = j+1 ;
```

② Selection structure sequence of instn are executed by deciding upon the condition.

```
if (x > y)
        i = i+1;
else
        j = j+1;
```

③ Iteration structure stmts are repeatedly executed.

```
for (i=1; i<=5 ; i++)
{
    i = i+1;
}
```

④ Encapsulation structure Compound structure.

Decision Making Statements:

① if stmt

Control the flow of execution of stmts. Condition

Syntax:

```
if (condition)
{
    stmts;
}
```

Example: (i) Check whether the number is less than 5 (ii) check equivalance of two nos. if (m-n==0).

(ii) Swap two values when first no. is greater than two (2nd no). (iii) Print no. b/w 10 & 15

```
int a, b, c;
if (a > b)
{
    c = a;
    a = b;
    b = c;
}
```

2 variables

```
a = a+b
b = a-b
a = a-b
```

a	b	c
10	5	
5		10
5	10	

if - else stmt.

Two way decision making used in conjunction with Condtn.

* Test & then take decision

Syntax:

```
If (condition)
{
    stmts ; }
else { stmts ; }
```

Example: Even / odd, Leap year

③ Nested if ... else stmt.

Example:

```
Void main()
{
    int a;
    printf (" Enter a number ");
    Scanf ("%d", &a);
    if (a == 10)
        printf (" A Grade ");
    else
    {
        if (a == 8)
            printf (" B Grade ");
        else
            printf (" c Grade ");
    } getch();
}
```

Looping & Branching.

Repetition - set of instn in specified no. of times

↳ loop control structure.

"Block of stmts which are repeatedly executed for certain no. of times".

* Body of loop
* Control stmt

looping stmts : Initialization
Test the ctl stmt
Executing the body of loop
Updating Condtn Variable.

op structures: while, do-while, for.

① While loop.

Repetitive control structure, executes the stmts until the cndtn becomes false.

Syntax:
```
while (condition)
{
    ...
    body of loop;
    ...
}
```

Example: Sum of n numbers,

```
main()
{
    int i=1, n, sum=0;
    printf("Enter n");
    scanf("%d", &n);
    while (i <= n)
    {
        sum = sum + i;
        i++;
    }
    printf("%d", sum);
}
```

Find SI using while loop.

```
int p, n, count = 1;
float r, si;
while (count <= 3)
{ pf
  sf
  si = (p*n*r)/100;
  pf("%f", si);
  count++;
}
getch();
```

② do-while loop

Syntax:
```
do
{
    ...
    body of loop;
    ...
} while (condition);
```

Example:
```
Void main()
{   int i = 1;
    clrscr();
    do
    {
        printf("Sample");
        i++;
    } while (i <= 5);
    getch();
}
```

Difference b/w while & do-while loop.

while	do-while
① Top tested loop	① Bottom tested loop.
② loop will not be executed when cndtn is false	② loop is executed even though cndtn is false.

③ for loop.

Syntax:

```
for ( intialization ; condition ; incr/dec counter )
{
    . . .
    body of the loop;
    . . .
}
```

Example:

(i) print n numbers.

```
#include <stdio.h>
#include <conio.h>
Void main()
{   int i,s;
    clscr();
    Printf("\n Enter s ");
    scanf ("%d", &s);
    for (i=1 ; i<=s; i++)
    {
        printf ("%d", i);
    }
    getch();
}
```

(ii) Sum of n numbers.

```
Void main()
{
    int i, n, Sum = 0;
    clrscr();
    printf (" Enter n ");
    scanf ("%d", &n);
    for (i = 1 ; i <= n ; i++)
    {
        Sum = Sum + i ;
        printf ("%d", Sum);
    }
    getch();
}
```

Example 1: for, break — print numbers in ascending & descending order

```
Void main ()
{
    int n,a,d; clscr();
    printf("\n Enter the number");
    scanf ("%d", &n);
    printf("\n\t Asc \t Des \n");
    a = d = n;
    for ( ; ; (a++, d--))
    {
        printf ("\n\t %d \t %d \n", a, d);
        if (d == 0)
            break;
    }
    getch();
}
```

Example 2: Calculate the total & avg of 3 students, 5 subj mark.

```
#include <stdio.h>
Void main()
{
    int m1,m2,m3,m4,m5,t,i;
    float avg;
    clrscr();
    for (i=1 ; i<=3; i++)
    {
        printf ("Enter the marks of student %d", i);
        scanf ("%d %d %d %d %d", &m1, &m2, &m3, &m4, &m5);
        t = m1+m2+m3+m4+m5;
        avg = t/5;
        printf ("The tot & avg of std %d is %d %f ", i, t, avg);
    }
    getch();
}
```

Switch statement

Execute a particular group of stmts from several available group of stmts. Decide upon multiple choices.

Multiway decision making. checks the values against switch cases & decides upon it which set of stmts is to be executed.

Syntax:

```
Switch (expression)
{
    case  const1 :
          block of stmt ;
          break ;
    :
    default :
          block of stmts ;
          break ;
}
```

Example:

```
Void main()
{
    int a,b, c, d ;
    clrscr() ;
    printf ("In Enter two numbers") ;
    scanf ("%d %d", &a, &b) ;
    printf ("In Enter your choice of manipulation") ;
    scanf ("%d", d) ;
    switch (d)
    {
        case 1 :
            c = a+b ; break ;
        case 2 : c = a-b ; break ;
        case 3 : c = a*b ; break ;
        case 4 : c = a/b ; break ;
    }
    printf (" The result is %d", c) ;
    getch() ;
}
```

Using switch - to check whether the no is odd/even.

```
Void main()
{   int n; clscr();
    printf("Enter the no"); scanf("%d", &n);
    switch (n%2) {
     case 0 : printf("It is an Even Number"); break;
     case 1 : printf("It is an odd Number"); break;
    } getch();
}
```

Comparison between Switch & Nested-if stmt.

Switch	Nested -if
① can test only constant values	① can test relational/ logical expressions.
② Nested -if can be used within switch stmt	② Switch cannot be used within Nested-if.

break, continue & goto stmts.

break - used to terminate the loop. skip the block of loop & goes to first stmt after loop.

continue - Continue with the next iteration of loop when we want to continue the pgm w/o executing other parts of pgm.

goto - doesn't require any condition passes the ctrl to anywhere in the progeam

goto label;

ample

getchar() & putchar()

```
Void main()
{
    char a;
    printf ("\n Enter a character");
    a = getchar();
    printf ("\n Displaying that character");
    putchar(a); putchar('A');
}
```

gets() & puts()

```
main()
{
    char a[10];
    puts ("Enter string");
    gets (a);
    puts (a);
}
```

While loop (print n numbers).

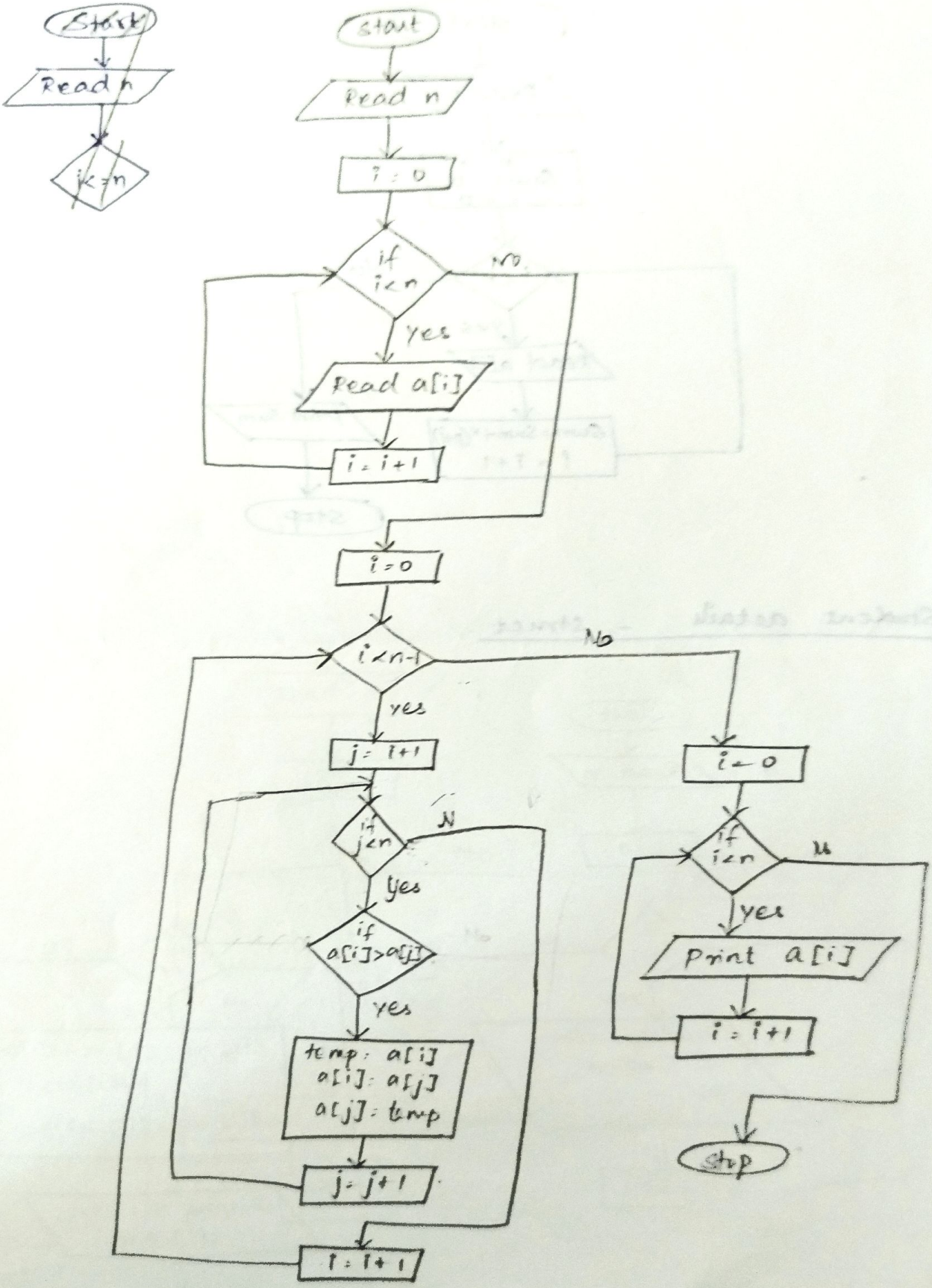
```
#include <stdio.h>
#include <conio.h>
Void main() {
int i=1, n; clscr();
Printf ("Enter n");
Scanf ("%d", &n);
while (i<=n)
{
   printf ("%d", i);
   i++;
}
getch();
}
```

do-while loop

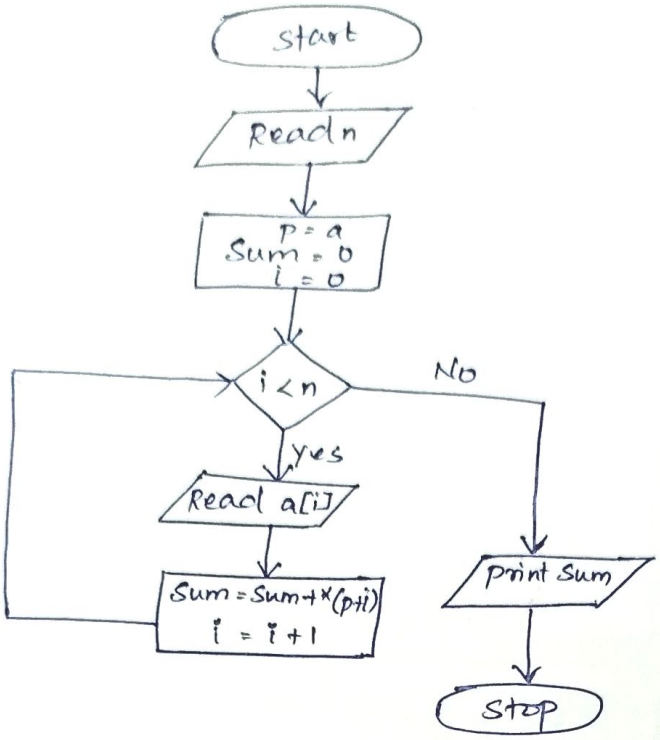
```
#include <stdio.h>
#include <conio.h>
Void main() {
int i=1, n; clscr();
printf ("Enter n");
Scanf ("%d", &n);
do {
 printf ("%d", i);
 i++;
} while (i<=n);
getch();
}
```

chart

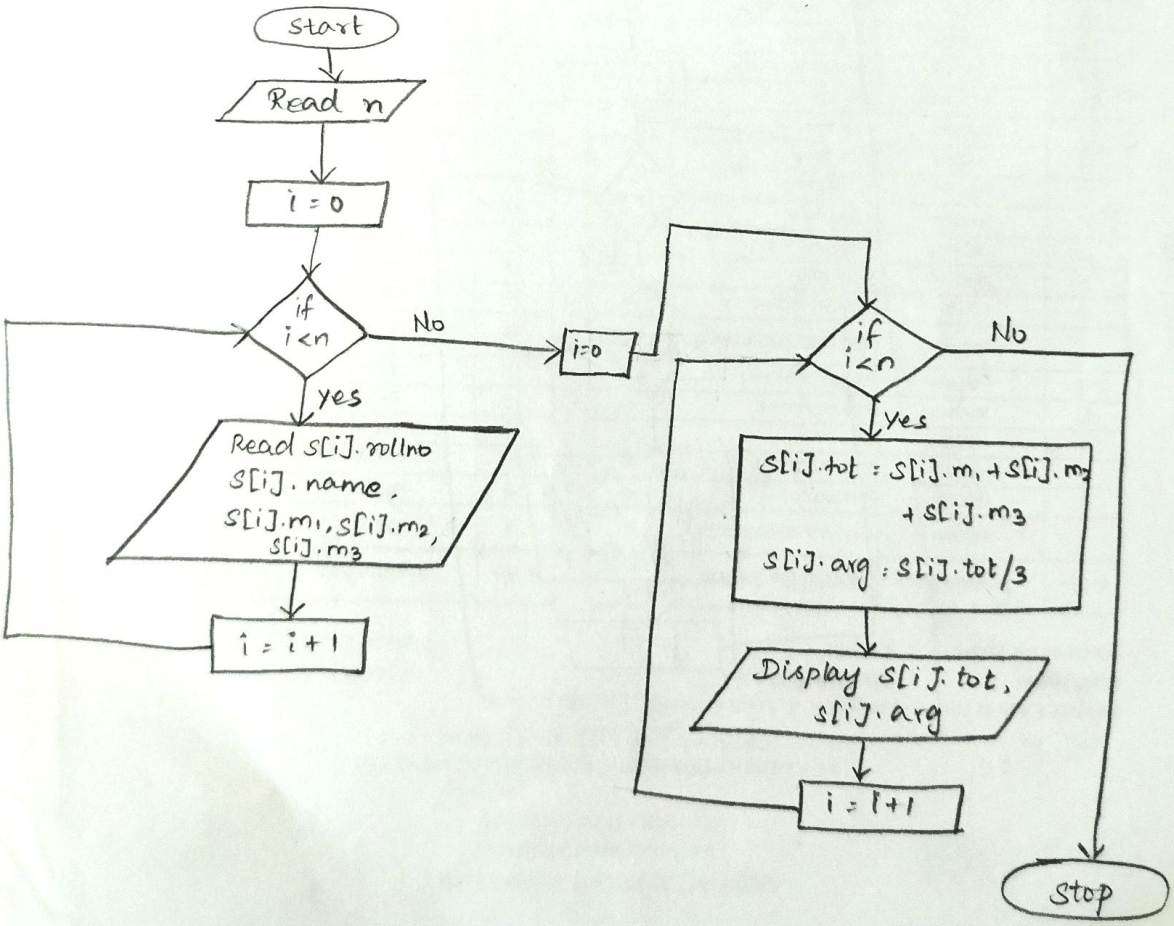
sorting using one dimensional array.



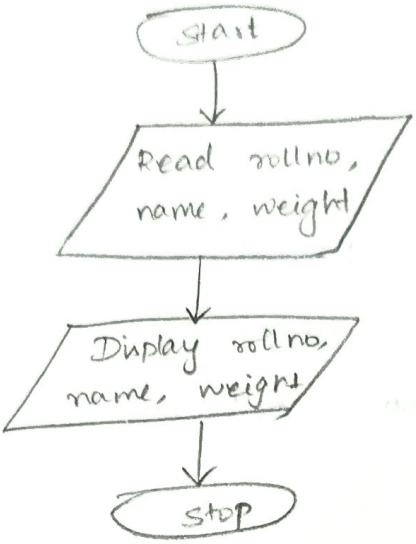
② Sum of n numbers



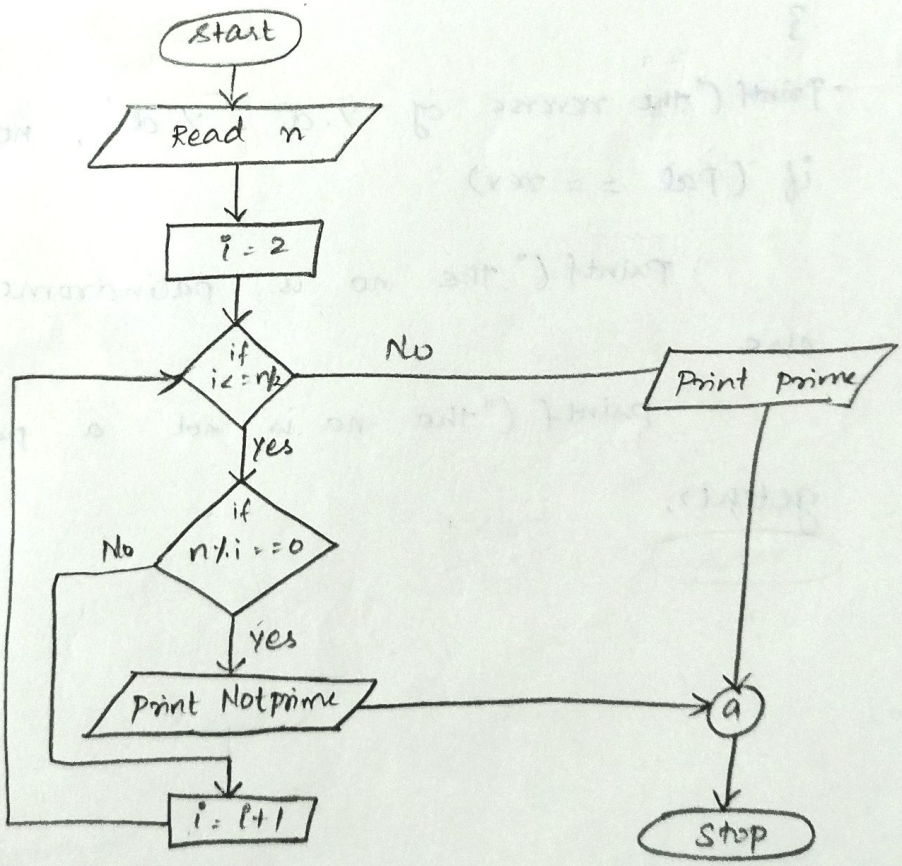
③ Student details - struct.



Student details - union.



(5) prime or not.



⑥ Reverse the number - palindrome

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int no, rev=0, r, pal;
    printf("\n Enter number");
    scanf("%d", &no); pal = no;
    while (no > 0)
    {
        r = no %10;
        rev = (rev *10) + r;
        no = no/10;
    }
    Printf("the reverse of %d = %d", no, rev);
    if (pal == rev)
        Printf("the no is palindrome");
    else
        printf("the no is not a palindrome");
    getch();
}
```