THE C PROGRAMING LANGUAGE

Historical Devlopment of C

=> The c language was developed in 1970's at Bell laboratories by a System programmer named Da Dennis Ritchie

=> Ken Thompson another System programmer at Bell laboratories, Ken Thompson adapted it from Basic combined programming language (BCPL) and named it as B language (first letter of BCPL), B language was modified by and improved by Dennie Ritchie and named as C language (the second letter of (BCPL)

version of C

=> Different organization have written and implemented C compilers which differ from one another to a laser or greater extent depending on the compiler being used

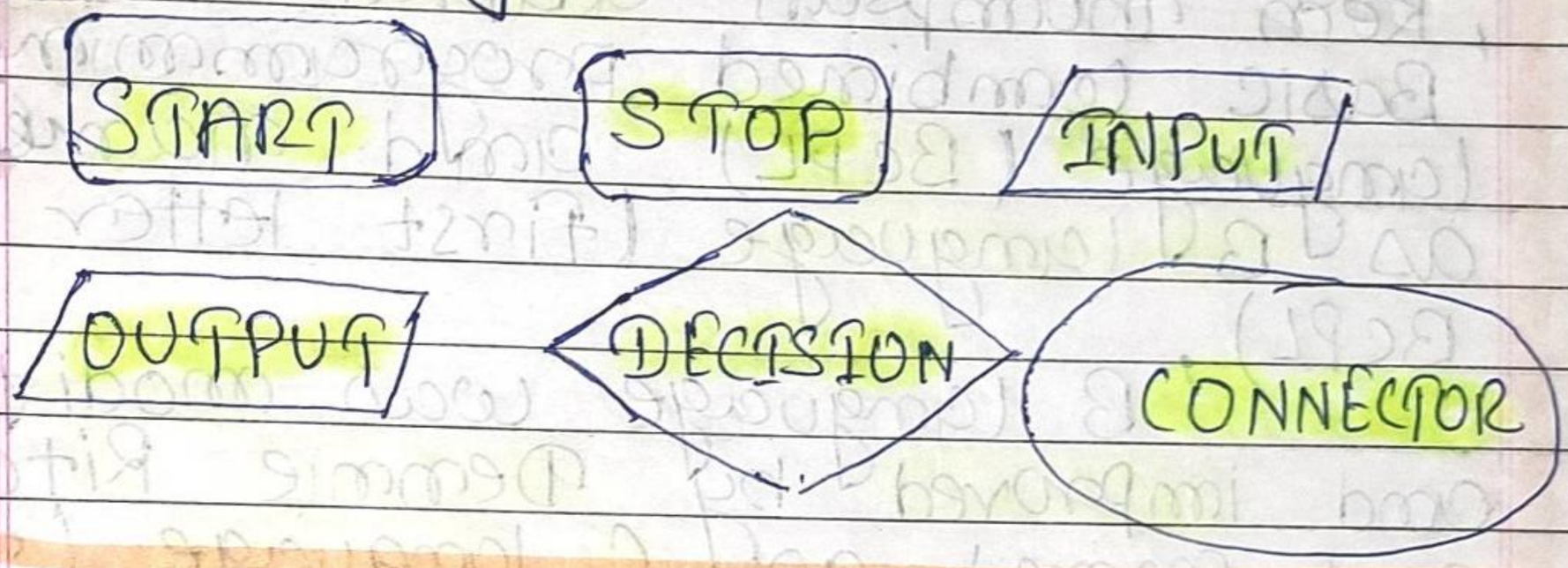
Some of the famous compilers are following

=> MicroSoft C Compiler.
=> Quick C compiler (from MicroSoft)
=> Turbo C Compiler (from Boroland International)

Flow Chart

A flow chart is simply a graphical method of indicating a proped proposed or actual solution of problem.

Symbols of flowchart are following



In C Programming a Program ~~is~~ is consist of

(i) Pre-Processor Comands
(ii) function
(iii) variables
(iv) Statement & expression
(v) Comments

#include <stdio.h> ⇒ Preprocessor command

main() ⇒ function

Printf(" ") ⇒ Gives desire output in our display

NOR In C Programming to execute the Program (Code) the compiler Passes through these the following Process

(i) Pre-processing =>
(ii) compilation => changes in Assemble langs
(iii) Assembly
(iv) Linking

(i) Pre-Processing => expanding macros, muting comment line, #include file are called and saved in(.i) files

(ii) Compilation => Codes are converted into assembly language instructions and saved with (.s) extension

(iii) Assembly => Convert the assembly language instruction into machine level instruction so that our CPU can understand

(iv) Linking => all the Assembly instruction, machine level instruction and all the files) get together in the process of linking to execute the code.

Note ; => Semi Colom is used in the end of statement mainly Semi Colom ~~is~~ Show that the statement is being terminated (@ [a-z], end, thanky (very much :-))

key words => key words are the words whose meaning has already been explained to the JC compiler,

example =>

asm	Signed	for	Const
continue	unsigned	near	enum
extern	break	Struct	if
include	do	volatile	return
Short	float	Char	typedef
union	long	else	
auto	static	goto	
define	void	register	
far	Case	Switch	
int	double	while	

Identifiers (Names) => In C Programming identifiers are the names to identify / ~~the~~ denote the variables, Arrays, Structure union, functions etc.

Shubham ≠ SHUBHAM
A ≠ a

Because C- Language is a Case Sensetive language.

Note

C- Programming is made up of keywords, Identifiers, Constent String liter, Symbols

Variable => The memory in which we first define its name then we store our data in it.

* ~~Dec~~ variable is declared by writing type variable_name.

* variable is initialized and declared by type variable_name = value;

Rules for Deffiming A variable in C

(i) ~~It~~ ~~Can~~ variable can contain alphabates, digits and underscore (_)

(ii) A variable name can start with an alphabet and underscore only

(iii) It can't start with a digit

(iv) No. white space & reserved keywords is allowed

(v) Example

(a) int Shubham ✓

(b) int $Shubham ✗

(c) float SHUBH123 ✓

(d) char 873Shubh ✗

(e) char __Shub@123 ✗

(f) int long ✗

(g) char __Shub__123 ✓

DATA TYPE

=> SIMPLE DATA TYPE

(i) int / long int => (Integer) Ex= 1, 2, 3, ...

(ii) float -> (Single Precision Real value)
Ex= 1.25, 3.0, 7.25 etc

(iii) double / long float -> Double Precision Real value ex=

(iv) char (character) = "Shubham"

=> STRUCTURED DATA TYPE

(i) Arrays & Strings

(ii) Structures

(iii) Unions

(iv) void

DATA TYPE = SIZE (Depends upon the (32/64 OS)

int = 2 Byte
long int = 4 Byte
unsigned int / short = 2 Byte
unsigned long int = 4 Byte

Code

	(Precision digit)
float = 4 Byte	7
double = 8 Byte	14

Code => Code to know the size of data type

```
#include <stdio.h>
void main()
{
Printf("%lu", sizeof(int/float/
doub/ long int / Char/ unsigned
int/ double));
}
```

OPERATORS IN C

=) An operator is a symbol used to perform operations in given programming language.

(1) Arthematic operators =)

(a) + → Addition → 2+3 = 5
(b) − → Substraction
(c) / → Division
(d) % → Modulus => Reminder

a=5
a%2 = 3

Relation operators

(a) == => is equal to
(b) != => is not equal to
(c) > => greater than
(d) < => less than
(e) >= => Greater than equal to
(f) <= => Less than equal to

Ex
```
x=34;
y=3;
Printf("%d", x==y);
```
Output = 0

↳ यहाँ Relation सही नही होगा तो 0 output आएगा

Ex
```
x=3;
y=3;
Printf("%d", x==y);
```
Output = 1

↳ यहाँ Relation सही होगा तो output 1 देगा

LOGICAL OPERATORS

(i) && => Logical AND operators if both the operands are non zero (True) then the condition is true and we will get the output as 1 otherwise we will get the output as 0

Ex- { int a, b;
a = 5
b = 4
Printf("%d", a && b);
output = 1

y both a and b are non-zero

Ex { int a, b;
a = 5
b = 0
Printf("%d", a && b);
Output = 0

(2) || => logical OR operators
if any of the operands is non-zero then condition becomes true (A || B) is true and it gives output as 1

Ex { int a, b;
a = 5
b = 0
Printf("%d", a || b);
outPut = 1

Ex { int a, b;
a = 0
b = 0
Printf("%d", a || b);
outPut = 0

(3) ! => logical NOT operator it is used to reverse the logical operator state of its operands. if logic is true then the logical NOT operator will make it false

Ex:- x = 5
y = 6
Printf("%d", !9&&b)
output = 0

BITWISE OPERATORS

a	b	a&b	a|b	a^b
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

and → or

~(a^b) को exclusive कहते है इसको एक 1 और एक zero अर्थात (True-false) चाहिए।

ASSIGNMENT OPERATORS

(i) = => Simple assignment operator assigns value from right side operands to left side operands

(ii) += => Add AND Assignment

operator. it adds the right operand to the left operand and assign the result to the left operands.

(3) `-=` => Substract AND assignment operators. it substract the right operand from the left operand and the result is assigned to the left operand.

(4) `*=` => Multiply, AND assignment operators. it multiplies the right operand with the left operand and the result is assigned to the left operand.

(5) `/=` => Divide, AND assignment operator. it divides the left operand with the right operand and the result is assigned to the left operand

Ex
```
a = 7;
a += 1;   → it will add 1 to 7 and a becomes =8
printf("%d", a)
output : 8
```

Ex a = 8;
    a *= 2;
printf ("%d", a)
output = 16

MISCELLANEOUS OPERATORS

(1) Size of() = It returns the size of a variable

(2) & => Returns the address of a variable

Ex: &a

(3) * => Pointer to a variable

(4) ?: => Conditional Expression

Ex: If Condition is true ? then value x; otherwise value y

OPERATOR PRECEDANCE

Multiplicative	*/%	left to right
Additive	+-	left to right
Relational	<<=>>=	left to Right

Increment & Decrement operators

=) In addition to arthematic operators there are two (2) special operators in C that operate on integer data only

(1) Increment operator (++)

Ex: (k++) or (++k), it will increase the value of integer variable k

(2) Decrement operators (--)

Ex: (k--) or (--k) it will decrease the value of integer variable k

Note If these operators are used as part of other expression, then they work differently

(*) In Prefix notation, the value of the variable is first increased or decreased and then used.

In Sufix notation the value of the variable is

first used and then increased or decreased.

~~If~~ Example

If I = 10 , J = 12

I++;

P = J++ + I => 12+11 = 23

Q = ++I + J++ => 12 + 13 = 25

OPERATOR PRECEDENCE

	Category	Operator	Associativity
(1)	Multiplicative	* / %	left to right
(2)	Additive	+ -	left to right
(3)	Shift	<< >>	left to right
(4)	Relational	<<=>>=	left to right
(5)	Equality	== !=	left to right
(6)	Bitwise AND	&	
(7)			
(8)			
(9)			
(10)			

Ajanta
Page No.
Date

① Example
2 + 3*6 = ?

Solution => * → Multiplicative operator comes before (+) Additive operators hence

~~2+18~~ = 2+3×6 = 2+18
Step2 Step1 = 20 Ans

Example ②
2+3−2 = ?

Solution: (+) Additive operator comes ~~first in the~~ ~~take~~ before the Substraction operator Hence

2+3−2 = 5−2
Step1 Step2 => 3 Ans

FORMAT SPECIFIERS, CONSTANTS & ESCAPE SEQUENCES IN C

FORMAT SPECIFIER => format specifier is a way to tell the compiler what type of data is in variable during taking input displaying output to the user.

Ex
```
int a = 8;
float b = 7.333;
Printf("Hello")
Print("The value of a = %d and the value of b = %f", a, b)
```
Output The v..... a = 8 and the ... b = 7.333

* Slicing of format specifiers *

```
Print("The value of b = 0.4%f", b);
```
output
=> 7.3330

↳ This imply that Decimal के बाद 4 digit ही होंगे।

```
Printf("value of b = %8.4f", b);
```

output = value of b = 7.3330
(12345678)

Note 8.4 => Means Total digit will be 8 and decimal digit will be 4

(1) %8.4 => it will create two space in the begaining

(2) %-10.4 => it will create 4 space after print the value of %f

* SOME USEFULL FORMAT SPECIFIER *

1. %c => Character
2. %d => Integer
3. %f => float
4. %l => long
5. %lf => double
6. %Lf => long double.

CONSTANT IN C

A constant is a value or variable that can't be changed in the program for example 15, 2B, 4.5, "Shubham arora", 8.4, 'a', etc

There are two way to define constant in C programming.

(1) Const keyword

(2) # define ~~process~~ Preprocessor

Example

```
int a = 5
a Const a=88
   a = 7
Printf("%d", a)
```

output = error (because you already mention a=88 as Constant which I can't be Changed.

Example

```
#include <stdio.h>
# define Shu = 3.45

main
{ int Shu = 4.3;
   Printf("%f" Shu);
```

output = error you can't change a pre define value.

ESCAPE SEQUENCES IN C

=) An escape sequence in C programming language is a sequence of characters.

=) It doesn't represent itself when used inside String literal or character.

=) It is composed of two or more characters starting with backslash \ for example \n represent new line.

List of Some useFull Escape Sequence

(1) \a => Alarm or Beep
(2) \b => backspace
(3) \n => New line
(4) \t => Tab (Horizontal)
(5) \v => vertical Tab
(6) \\ => BACK Slash
(7) \' => Single Quote
(8) \" => double Quote
(9) \? => Question Mark

COMMENT

Ajanta
Page No.
Date

COMENTS IN C PROGRAMMING

(1) Single line Commenent = //

// Shubham Arorein has writen, Comment

// This is a one line Commenent

(2) multiLine Commenent

/* This is a multiline Comment
and this is the 2nd line
This is the 3rd line of Comm.. */

CONTROL STATEMENT

Relational Expression ⇒ Relational Expressions Compare the value of two operands. Relational expression are used to test a condition in order to decide whether an action should be taken or not.

The result of the relation expression is either 0 or 1

Relational expression are following

<, <=, > >=, ==, !=

Note ⇒ we can apply relational expression b/w a float and a int but before execution integer operand is converted to float type.

Logical Expression ⇒ A logical expression is used to form a complex test condition in order to take a decision.

OPERATOR	OPERATION	PRECEDANCE
!	Negation Logical NOT	Highest
&&	Logical AND	Middle
||	Logical OR	Lowest

CONTROL STATEMENT

(1) Decision Making Statement
* If Statement
* if-else Statement
* else-if construct / ladder if
* Switch Statement

(2) Looping Statement
* for Statement
* while Statement
* do-while Statement.

(3) Jumping Statement
* break
* continue
* go to

---

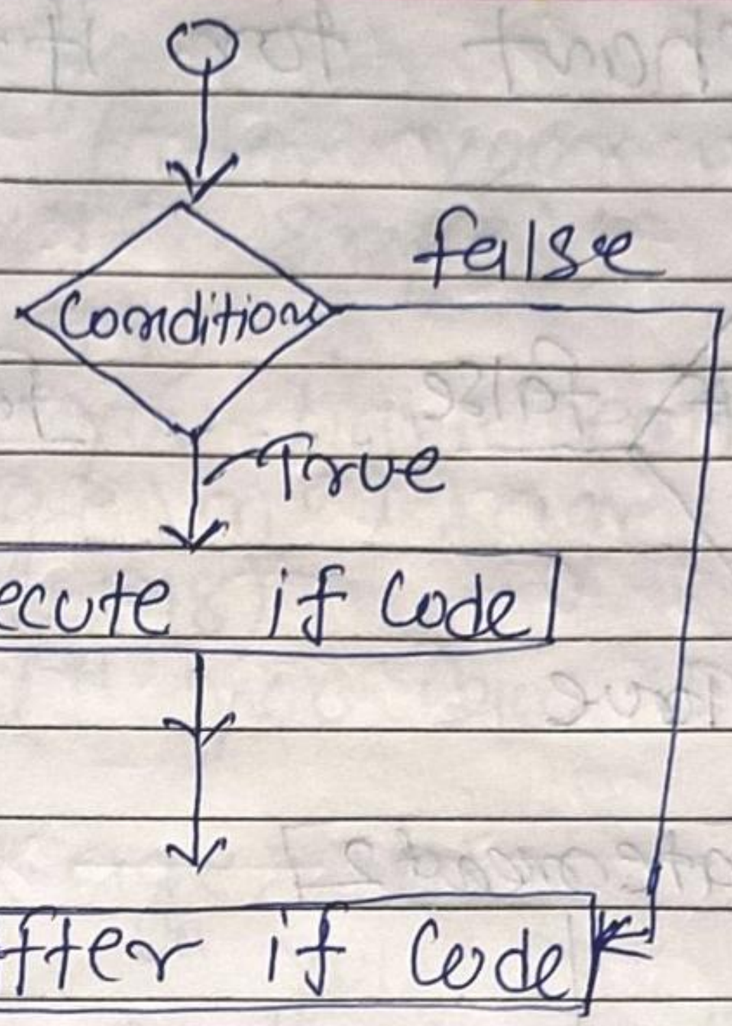
if statement => If statement is used to perform operation based on some condition.

Types of if Statement :-

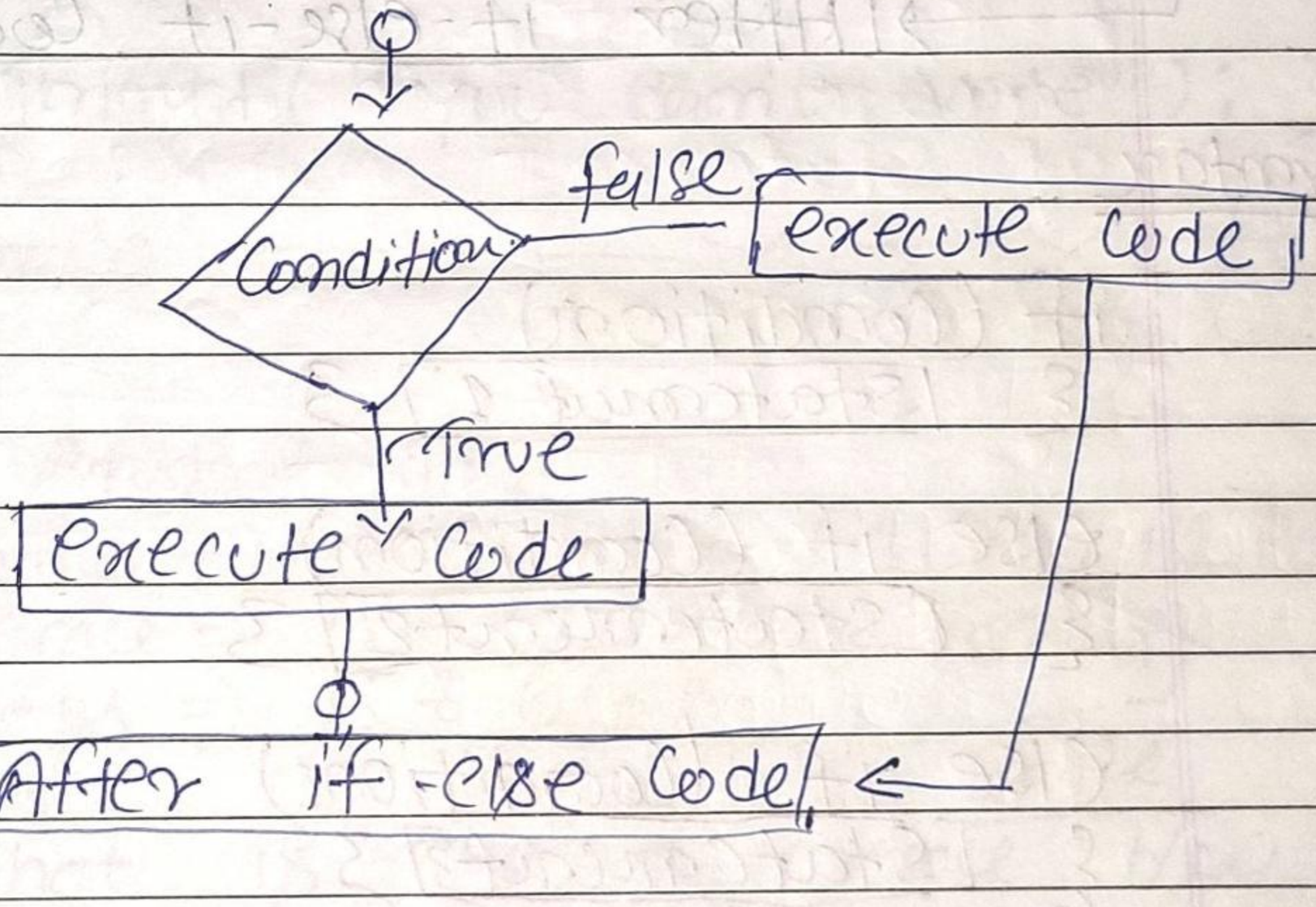
(1) If Syntax for if-else

```
if (3>2)
{ Print("This will execute");
}
```

(1) flow chart for if statement



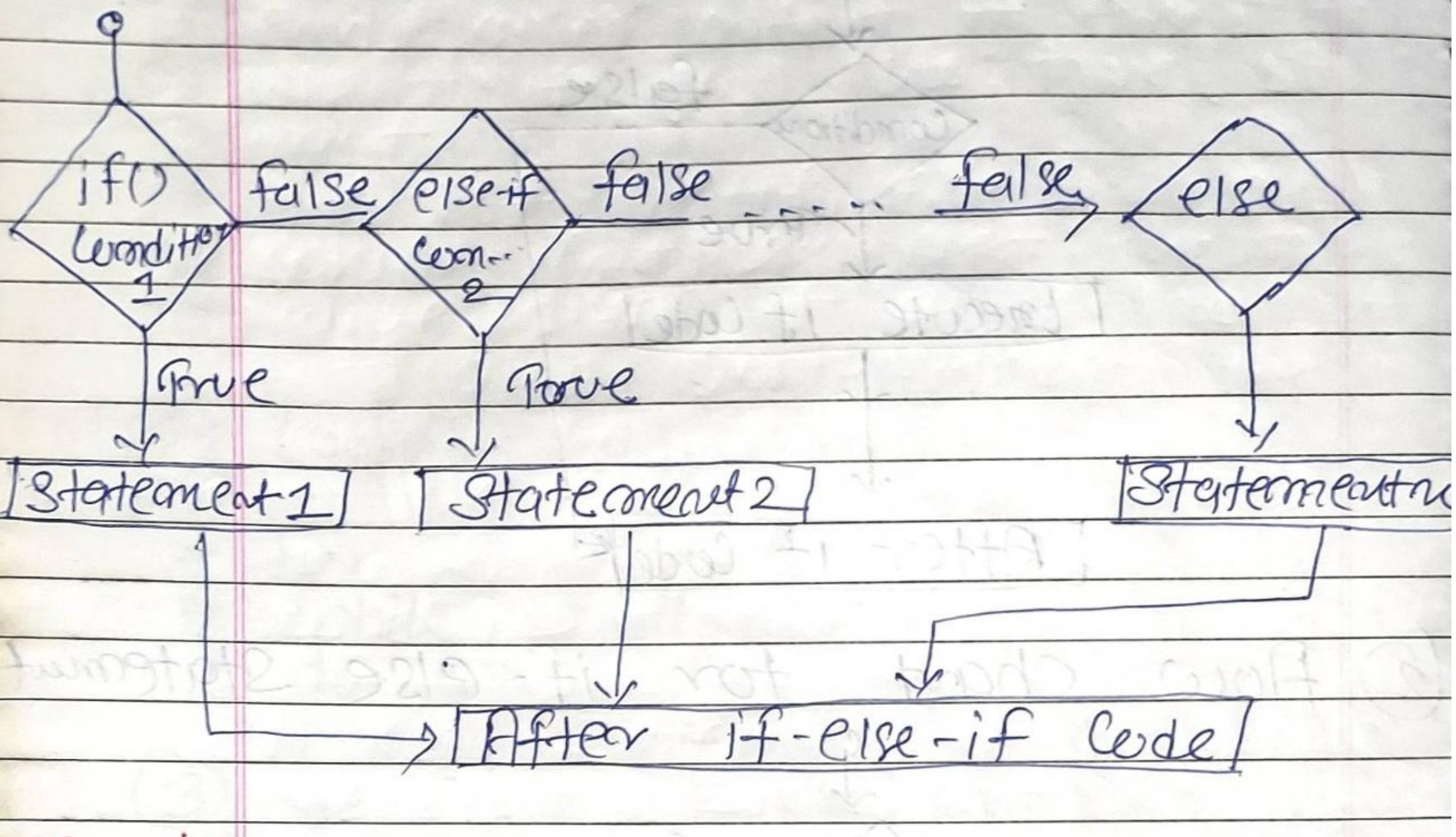
(2) flow chart for if-else statement



Syntax

```
if (condition)
{ [if True]
                }
else
    { [if false] }
```

flow chart for if-else-if



Syntax

```
if (condition)
  { [Statement 1] }

else if (Condition)
  { [Statement 2] }

else if (condition)
  { [Statement 3] }
  .
  .
  .
else { condition }
```

viva question below

Example

```
int age;
Printf("Enter your age");
Scanf("%d", &age);
if (age >=18)
  Printf("you have entered %d as
  your age\n"), age)
if (age >= 18)
{ printf("you Can wait");

else if (age > 18)
{ Printf("you are b/w 10-18
and you Can't vote")

else
{ Printf("you Can't vote"); }
```

Quiz

Student Pass in
Science + Maths = 95 ₹ (1 Copy + 1 Pen)
Science ⇒ 15 ₹ (1 Pen)
Maths ⇒ 15 ₹ (1 Copy)
Prize

(Q) what is the difference b/w
if, if if, if...else
and
if, else if, else if... else ?

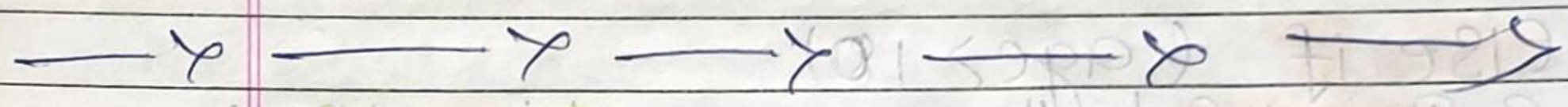
Ans:) In multiple if..if, if else statem
Compilar check each and every

LOOP

Ajanta
Page No.
Date

Statement condition one by one and gives output in every time when it find the true state ment

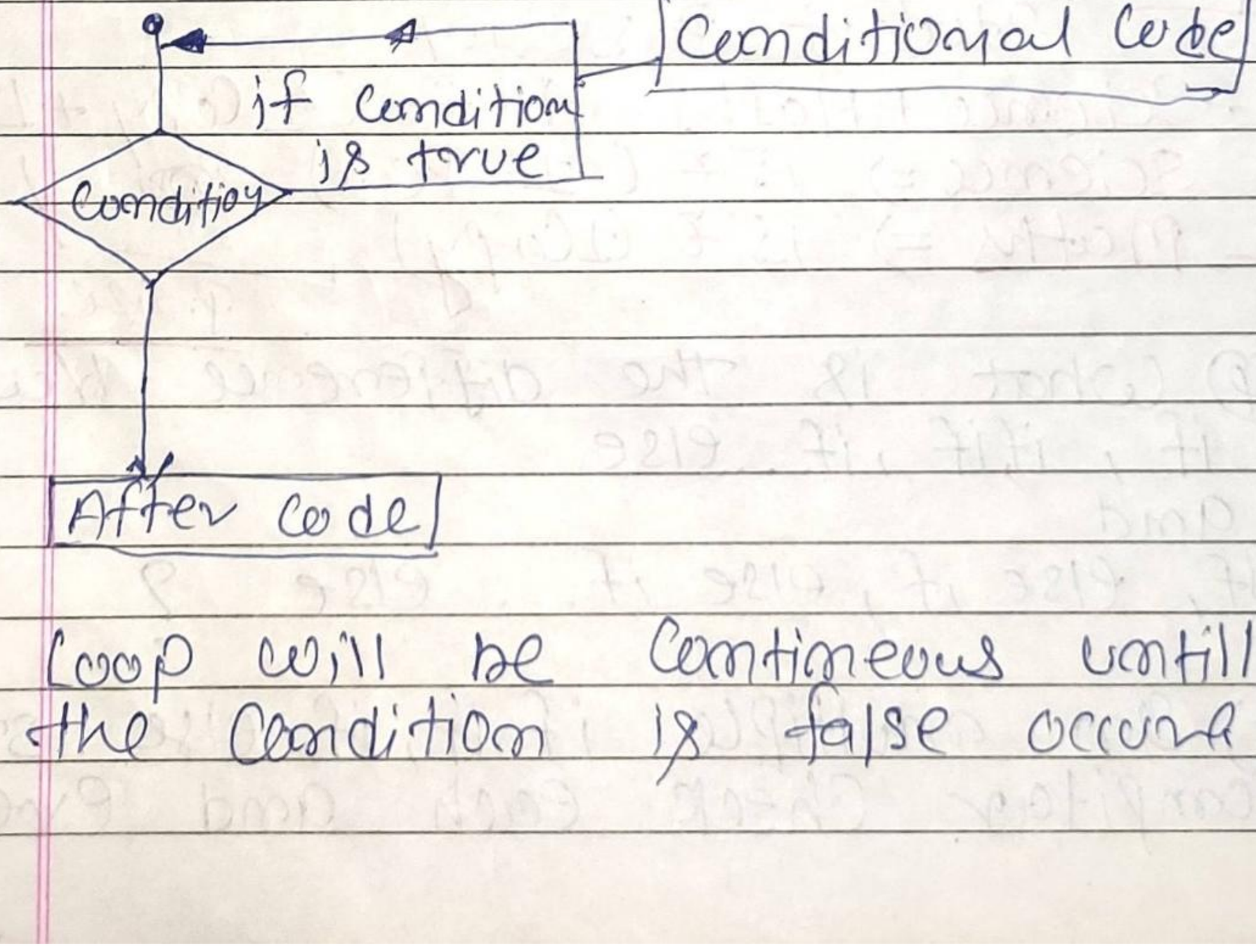
but in if, else if... else when compiler get the true value then it gives the out put and jump from the loop.



LOOPS

A loop statement allows us to execute a statement or grop of statement multiple times

flow chart



Loop will be continuous untill the condition is false occure

Types of loops

1. do while loop
2. While loop
3. for loop

(1) do while loop => The do while loop in C programming check its condition at the bottom of the loop,
do while loop execute at least once

Syntax

```
do {
    code to be executed
} while (condition)
```

Ex

```
int i = 0
do {
    i = i+1;
    Print ("%d", i);
} while (i < 10);
```

output : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Ex:

```
{ int num, index=0;
  int num, last;
  Printf ("Enter the first num\n");
  Scanf ("%d", &num);

  Printf ("Enter the last num...\n")
  Scanf ("%d", &last)

  do
    { Print("%d \n", num);
      num = num + 1;
    } while (num <= last);
```

output:

```
Enter the first no...
1
Enter the last no..
10
1
2
3
4
5
6
7
8
9
10
```

While Loop

While loop

A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition

In other words we can say that The while statement is suited for problems where it is not known in advance that how many times a ~~student~~ statement or a statement block will be executed

Syntax

```
while (condition)
{ (code to be executed) }
```

Example

```
{ int i = 0
  while (i <= 30)
  {
    Print ("%d", i);
    i = i+1
  }
}
```

output
0 13
1 14
2 ,
3 ,
4 ,
5 ,
6
7 .
8 29
9
10
11
12

FOR LOOPS

A ~~for~~ for loop is a repetition control structure that allows us to efficiently write a loop that needs to execute a specific numbers of times. which is know by the user in advance.

Syntax

```
for( exp1 ; exp2 ; exp3)
{ Statement block }
```

where exp1 and exp3 are assignment expression and exp 2 is a relational expression.

Example

```
int i;
For (i=0; i<=5; i++)
{ printf ("%d", i); }
```

Output 0, 1, 2, 3, 4, 5

Note.

we can ~~int~~ initialize more then one variable in Expression

Ex:- int i=0, J=0;

Note = exp3 is always optional

```
for(;  i<3, i<5,  i++, j++)
{ Printf("%d %d \n", i, j;
  i++; j++; }
```

output
0 0
1 1
2 2
3 3
4 4
5 5

Note ⇒

BREAK STATEMENT

⇒ Break statement is used to bring the program control out of the loop

⇒ Break statement is used inside loops or switch statement.

Break can be used with

(1) loops
(2) switch case expressions

SWITCH STATEMENT

Ajanta
Page No.
Date

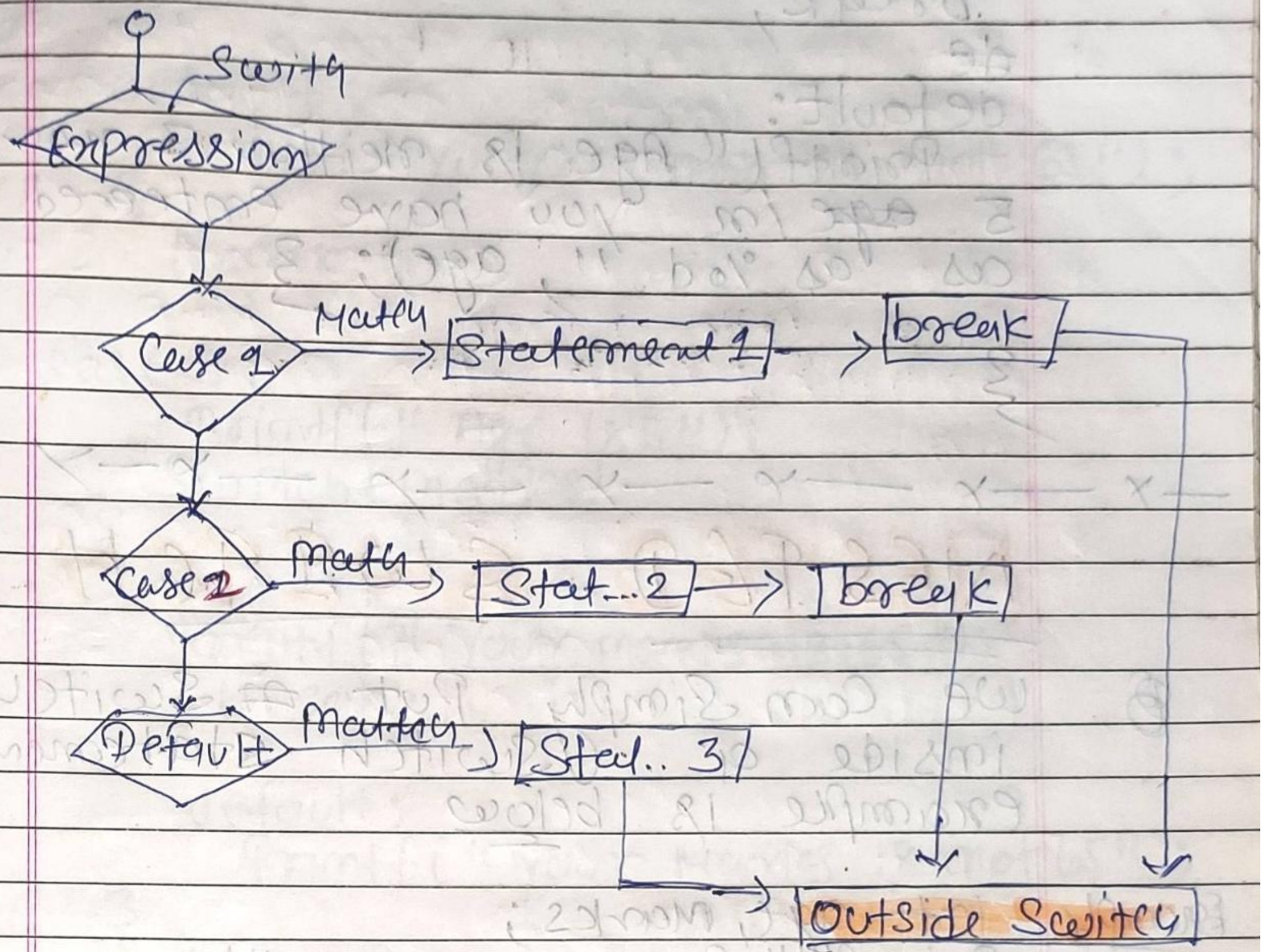
⇒ Switch Statement allows a variable to be tested for equality against a list of value. Each value is called a case and the variable being switch on is checked for each switch case.

Syntax

```
Switch (var name/expression)
{
    Case var value
        Printf(" ") /Statement
        break (optional)

    Case var value
      Printf(" ") / Statement
       break (optional)

      .
      .
      .

    default : (optional)
      Printf(" ") / Statement

}
```

Note Default is the last case of switch statement.

Flow Chart of Switch Statement



Simple example of Switch Statement

```
{ int age;
  Printf("Enter your age");
  Scanf("%d", &age);

Switch(age)
{
  Case 3:
    Printf("Age is 3");
      break;
  Case 5:
    Printf("Age is 5");
```

```
    break;
de
default:
    Printf("Age is neither 3 nor
5. age \n you have entered
as as %d ", age); 3
}
```

NESTED SWITCH

- We can Simply Put ~~##~~ Switch inside a Switch statement example is below

Example

```
int age, marks;
Printf("Enter your age");
Scanf("%d", &age);

Printf("Enter your marks ~~age~~");
Scanf("%d", &marks);

Switch (age)
{
  Case 3:
  Printf("Age is 3\n");
  Switch (marks)
  {
    Case 65:
    Printf("your marks is 65");
```

```
        break;
    default:
        Printf("your marks is not 65");
    }
        break;
Case 5:
    Printf("Age is 5");
    Switch(marks)
{
    Case 65:
    Printf("your marks is 65");
        break;
    default:
        Printf("your marks is not 65");
}
    break;
default:
    Printf("Age is neither 5 nor 3
you have entered age as %d", age);
    }
    }
```

CONTINUE STATEMENT

* It is used to bring the program control to the next iteration of the loop.

* The continue statement skips some code inside the loop and continue with the next iteration

* it is mainly used for a condition so that we can skip some lines of code for a particular condition.

Ex

```
int num = 1;
while (num <= 5)
{
    if (num == 3)
    {
        num = num+1;
        continue;
    }
    printf("%d", num);
    num = num+1
```

GOTO STATEMENT (JUMP)

* Goto (goto) is also called Jump Statement in C Programming.

* It is used to transfer program control to a predefine label

* it is use to its use is avoided since it cause confusion for the fellow programmers in understanding the code.

* Goto Statement is preferable when we need to break multiple loops using a single statement at the same time

```
#include <stdio.h>
main()
{
goto first;
third:
printf("my name is
```

```
#include <stdio.h>
main()
{
   goto first;

   third:
     printf("My last name is Singh\n");
   go to end;

    second:
   printf("My middle name is Kumar\n");
    go to third;

    first:
   printf("my first name is Shubham\n");
    go to second;

    end:
    printf("Done :-)");
```

Type Casting Syntax

Converting one datatyp into another is known as type casting or type conversion

Ex
```
int a = 3;
float b = 54/5;
Printf("%f\n", b);
```
Output = 10.000000

Ex
```
int b = (float) 54/5;
Printf("%f\n", b);
```
Output = 10.800000

Ex
```
int a = 3;
Printf("The value of a = %f", a);
```
output: The value of a = 0.000000

Ex
```
float a = 54.256;
Printf("The value of a = %d", (int) a);
```
output: The value of a = 54

FUNCTION

(1) What is function

→ function are used to divide a large C program into smaller pieces.

→ A function can be called multiple times to provide reusablity and modularity to the C- program

→ A function is called procedure or subroutine.

FUNCTION SYNTAX

The basic syntax is following

```
return_type function name( data_type Parameter 1, data_type Parameter 2.....)
{
    code to be written
}
```

Ex
```
int Print Star()
{ Print f("*");
   return 0;
}
```

ADVANTAGE of FUNCTION

→ We can avoid rewriting same logic through function

→ We can divide work among programmers using function.

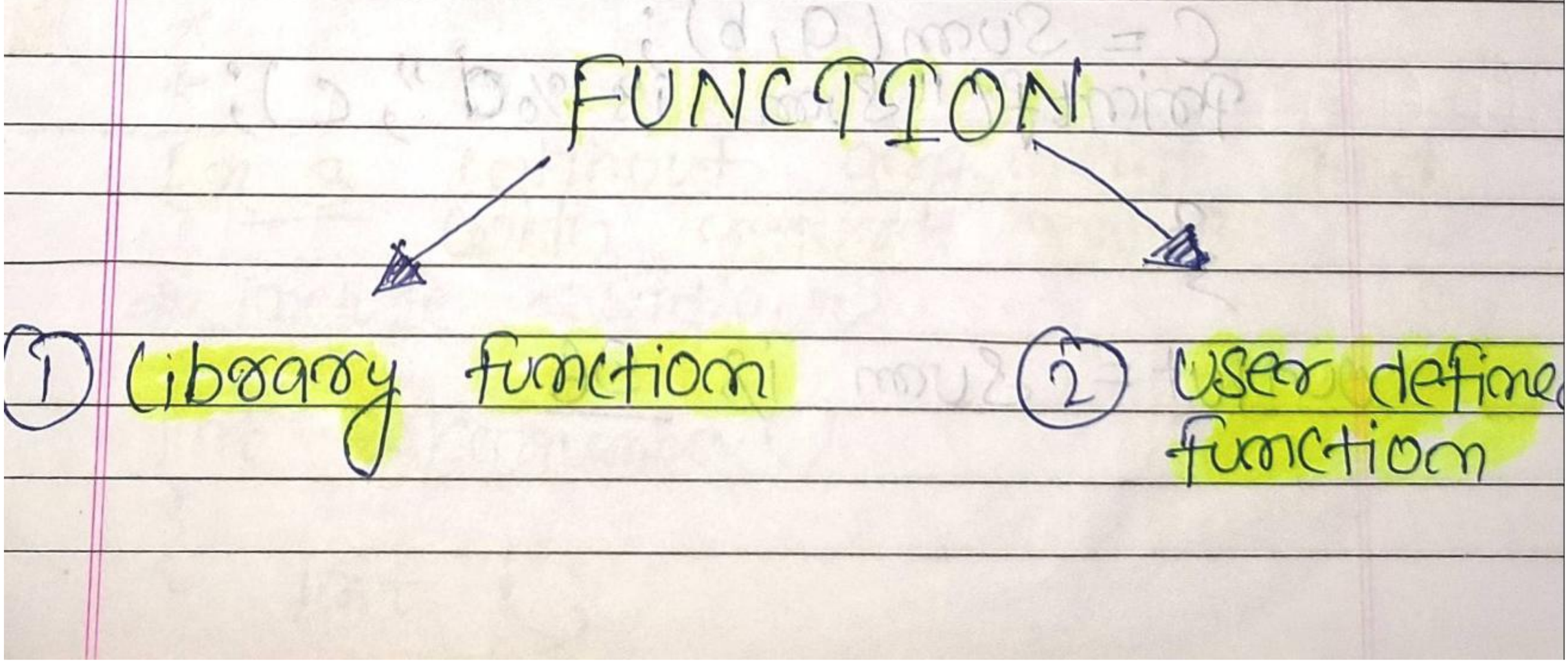
→ We can easily debug a program using function.

DECLARATION, DEFFINATION AND CALL

→ A function is declared to tell a compiler about its existence.

→ A function is defined to get some task done.

→ A function is called in order to be used.



* Library function => function include in c header files

* User define function => function created by c programmer to reduce complexity of a program.

Example (1) with argument & return value

```
#include <stdio.h>

int sum(int a, int b)
{
    return a+b;
}

void main()
{
    int a, b, c;
    a = 9
    b = 87
    C = sum(a, b);
    Printf("Sum is %d", c);
}
```

output = Sum is 96

Ex 2 ~~to~~

with argument and without return value.

```
# include <stdio.h>

void PrintStar (int x)
{
    for (int i=0; i<x; i++)
    {
        Printf("%c\n", '*');
    }
}

code
    ~~Main~~ void main()
    {
        Print_Stars (5);
    }
```

↳ Calling function.

FUNCTION

output

```
*
*
*
*
*
```

Ex 3 without argument and with return value.

```
# include <stdio.h>

int telecomnumber()
{
    int i;
```

```
    Printf("Enter a number");
    Scanf("%d", i);

    return i;
}
```

Code

```
int main()
{  int c

   c = takenumber();
   Printf("The entered number
      is %d", c);

   return 0;
}
```

RECURSION IN C

What is a recursive function ?

=> Recursive function or recursion is a process when a function calls a copy of itself to work on a smaller problem.

=) Any function which call itself is called recursive function

=> This makes the life of Programmer easy by dividing a given problem into easier problem

=) A termination condition is imposed on such function to stop them executing of themselves forever.

=) Any problem that can be ~~saved~~ solved recursively. Can also be solved iteratively.

Eg:- 1 FACTORIAL CALCULATION

=) The case at which the function doesn't recur is called the base case

=> The instance where the function

Calling itself to perform a subtask, is called the recursive case.

⇒ for factorial calculation the base case occurs at the Parameter value 0 and 1

Ex factorial(5)
5 x factorial(4)
5x4x factorial(3)
5x4x3 x factorial(2)
5x4x3x2 x factorial(1)

Code
```
#include <stdio.h>
int factorial (int number)
{
  if (number==1 || number ==0)
  {
    return 1;
  }
  else {
    return (number * factorial (
number -1));
  }
}
int main()
{ int num;
printf("Enter the no to find
its factorial");
scanf("%d", &num);
```

ARRAY

printf ("The factorial of %d is %d \n", num, factorial(num));

return 0;
}

ARRAY

What is an ARRAY

=) An array is a collection of data item of the same type.

=) items are stored at contiguous memory locations.

=) it can also store the collection of derived data type, such as pointers, structures etc

=) A one dimentional array is like a list

=) A two dimentional array is like a table.

=) The C language place no limits on the number of dimensions in an array.

=) Some texts refers to one dimensional arrays as vector, two-dimentional arrays as matrices and use the general term arrays when the number of dimensions is unspecified or unimported

ADVANTAGE OF ARRAYS

(1) It is used to represent multiple data items of same type by using only single name.

(2) Accessing an item in a given array is very fast.

(3) 2 Dimentional arrays makes it easy in mathematical application as it is used to represent a matrix

SYNTAX FOR DECLARING AND INITIALIZING AN ARRAY.

=) Data_type name[size];

=) Data_type name[size] = {x, y, z, ...}; // size is not required in this case.

=) data_type name [row] [colum];
// for 2D arrays.

#) We can also initialize the array one by one by accessing it using its index.

=) name [0] = 0;

Ex ~~in~~

```
#include <stdio.h>
void main()
{
    int Marks [5] ;
    marks [0] = 44 ;
    marks [1] = 30 ;
    marks [2] = 85 ;
    marks[3] = 90 ;
    marks [4] = 78 ;
printf(" marks of 1st studet = %d", marks[0]);
  ,,          ,,      ,, 2nd   ,  ,  ,  ,   marks[1]);
  ,,          ,,      ,, 3rd       ,,       marks[2]);
  ,,          ,,      ,, 4th       ,,       marks[3]);
  ,,          ,,      ,, 5th       ,,       marks[5]);
```

Ex ② int marks[4];

for (int i=0; i<4; i++);
{
printf("Enter the value of %d element of array \n", i);
scanf("%d", &marks[i]);
}

for (int i=0 i<4; i++);

{ printf("The value of %d element of the array is %d \n", i, marks[i]);

Ex 3

#include <stdio.h>
void main()
{
int marks[4]
float sum=0

printf("Subject Code \n 1 Maths \n 2 Physics \n 3 Chemistry \n 4 Python

for(int i=0; i<4; i++)
{
printf("Enter the marks for sub Code %d \n", i+1);
scanf("%d", &marks[i]);
sum=sum+marks[i];
}

```
for (int i=0; i<4; i++)
{
  printf("marks of code %d sub=
  %d \n", i+1, marks[i]);
}
  printf("Sum= %0.2f", Sum);
}
```

DISADVANTAGES OF USSING ARRAYS

→ Poor time complexity of insertion and deletion operation

→ Wastage of memory since array are fixed size.

→ if there is enough space present in the memory but not in contiguous form, you will not be able to initialize your array.

→ it not possible to ~~init~~ increase the size of the array, once you have declared the array.

2-D Array

```
#include <stdio.h>
void main()
{ int marks [2] [4] = {{2,4,5,6},
{7,3,77,88}};
```

```
for (int i=0; i<2 ; i++)
{
    for (int j=0; j<4 ; j++)
    {
        Print ("%d", marks[i][j]);
    }
    Print("\n");
}
```

Unformmated Input function

(1) getchar() => It is a non-Standard function whose meaning is already defined in the Stdio.h (header file, It accept Singe input from the user and it requires enter key to complete its function.

Ex

```
main()
   Char  Shu;
  Printf(" Enthe the value");
   Shu = getchar();
  Printf ("you have Entered %c", Shu);
```

(2) getch() => It is a nonStemdard Fumction and it is present in Conio.h header file. It does not use any buffer to store the input Charecter, The entered Charecter is immediatly returned without waiting for the enter key. It is used to accept hidden inputs like password, ATM Pin etc.

```
main()
{   Char  Str;
 Str = getch();
 Print f ("you Entered %c", Str);

}
```

(3) getche() => It is a non-Standard function in C which is used to enter ~~the~~ a single character from the Standard input device, It shows the entered character and ~~to~~ ~~immediately~~ immediately execute the next line.

```
main()
{ char str;
  str = getche();
  Printf("you entered %c", str);
}
```

(4) gets() => It reads a line and store in the given variable and it store string instead of single char

```
main()
{ char str[5]
   gets(str);
  Printf("you entered %s", str);
```