C PROGRAMMING

BCA TU

WHAT'S INSIDE?

Notes of C Programming

info@notebahadur.com

Note Bahadur

Unit - 1

Introduction.

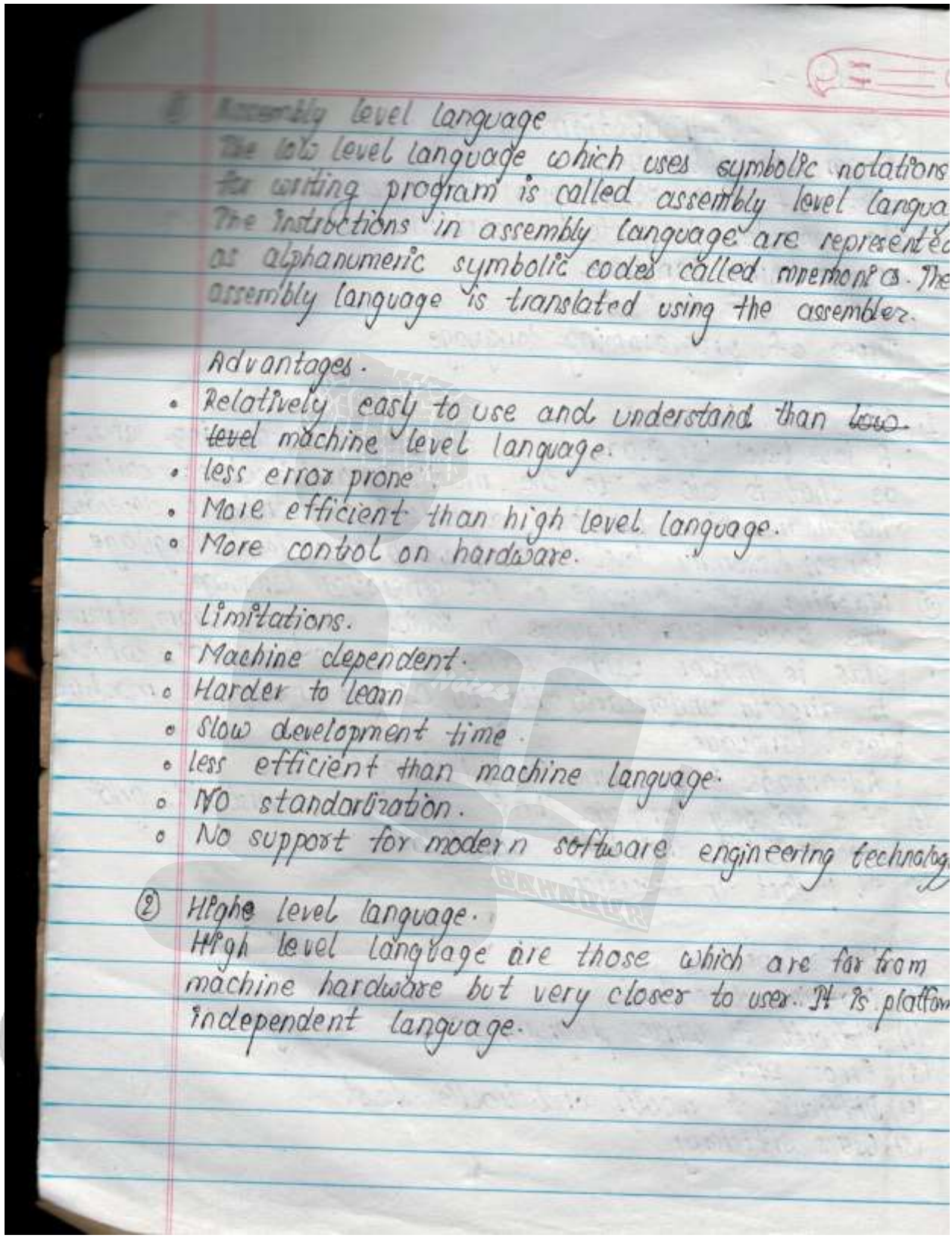
Programming language.

- A set of rules that provides a way of instructing the computer to perform certain tasks is called programming language.

Types of programming language

1. Low level language.
   A low level language is a computer programming language that is closer to the machine architecture or computer hardware. It is platform dependent or hardware dependent.
   For eg: Assembly level language and machine level language
   (a) Machine level language or 1st generation language.
   The lowest level language in which every progam statements is written using series of 1's and 0's which is directly understandable by CPU is known as machine level language.
   Advantages of machine level language.
   (1) It is the only language that computer understands and execute without using the need translator.
   (2) It is fast in execution.

   Limitations.
   (1) Machine dependent.
   (2) Difficult to write program/ complex language.
   (3) Error prone
   (4) Difficult to modify and trouble shoot.
   (5) User's efficiency.



① Assembly level language
The low level language which uses symbolic notations for writing program is called assembly level langua
The instructions in assembly language are represented as alphanumeric symbolic codes called mnemonics. The assembly language is translated using the assembler.

Advantages.
- Relatively easy to use and understand than low level machine level language.
- less error prone
- More efficient than high level language.
- More control on hardware.

Limitations.
- Machine dependent.
- Harder to learn
- Slow development time.
- less efficient than machine language.
- No standardization.
- No support for modern software engineering technolog

② Highe level language.
High level language are those which are far from machine hardware but very closer to user. It is platfor independent language.

Language Processor

A language processor is a special type of computer software that has the capacity of translating the source code or program codes into the machine codes. i.e in the form of 0s and 1s.

The language processor can be any of the following three types :-

(1) Compiler (Higher level language into machine language)
(2) Assembler.

The assembler is used to translate the program written in Assembly language into machine code. The source program is an input of assembler that contains assembly language instruction. The output generated by assemblers is the object code.

(3) Interpreter.

Difference between Compiler and Interpreter.

→	Compiler	Interpreter.
(1)	A compiler is a program which converts the entire source code of a program into executable machine code for a CPU at once.	(1) Interpreter take a source program written in High level language and translate it into machine code line by line.
(2)	Compiler take large amount of time to analyze the entire source code but the overall execution time of the program is comparatively faster.	(2) Interpreter take less amount of time to analyze the source code but the overall execution time of the program is slower.

3. Compiler generates the error message only after scanning the whole program.	3. Its Debugging is easier it continues translating the program until the error is met.
4. Generates intermediate object code.	4. No intermediate object code is generated.
Example: C, C++, Java.	Example: Python, Perl.

Program Errors

An error is anything in the code that prevents a program from compiling and running correctly. It is also known as a bug. There are three types:-

(1) Logical Error

The presence of logical errors leads to undesired or incorrect output and are caused due to error in the logic applied in the program to produce the desired output.

(2) Syntax Error

The error related to the grammer or specific syntax in which code needs to be written is called syntax error.

(3) Run Time Error

Those errors that occur during the execution of a program and occur due to illegal operation performed in the program is called run time error.

Program development lifecycle.

1. Problem identification or problem understanding or analysis
2. Algorithm design
3. Flow chart
4. Coding
5. Compilation and execution.
6. Testing and debugging.
7. Maintenance & support.
8. Documentation.

Algorithm

Features

* Input / output
* Finiteness
* Definiteness
* Correctness
* Efficient
* Generality.

L) Step by step procedur for solving any problem.

4) Statements used in algorithm should be simple, clear and precise

Eg: Algo for Adding two number.

Step 1 : Start

Step 2 : Input : A, B.

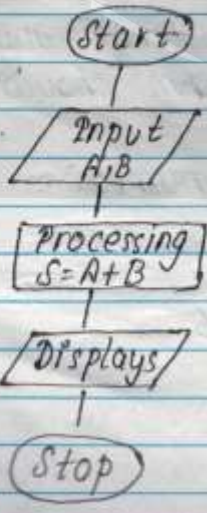
Step 3 : Calculate : S = A + B

Step 4 : Display : S

Step 5 : Stop

③ Flow chart

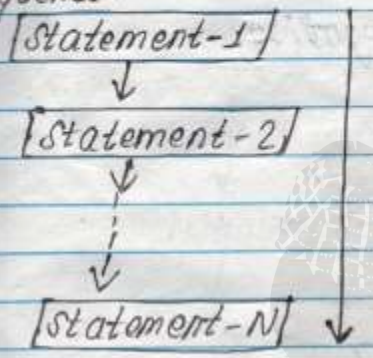
Symbol	Meaning
Oval	Start / Stop
Parallelogram	Input / Output
rectangle	Processing
diamond	Decision making
Circle	Connector
Arrow head	Flow direction



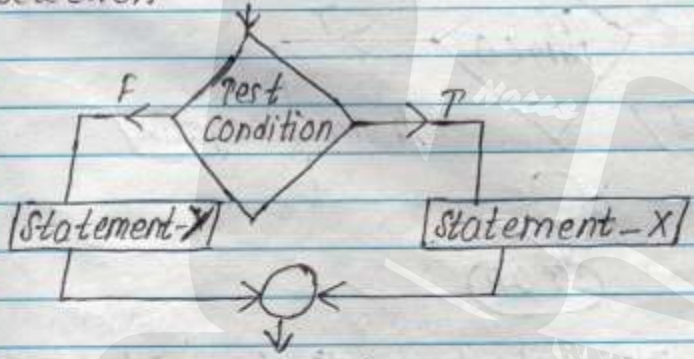
Three basic operation used in flow chart are:-

① Sequence ② Selection ③ Iteration/Repedition/ looping.

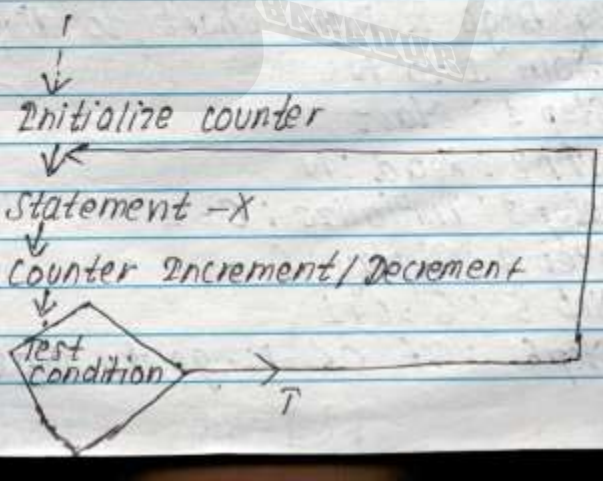
① Sequence



② Selection

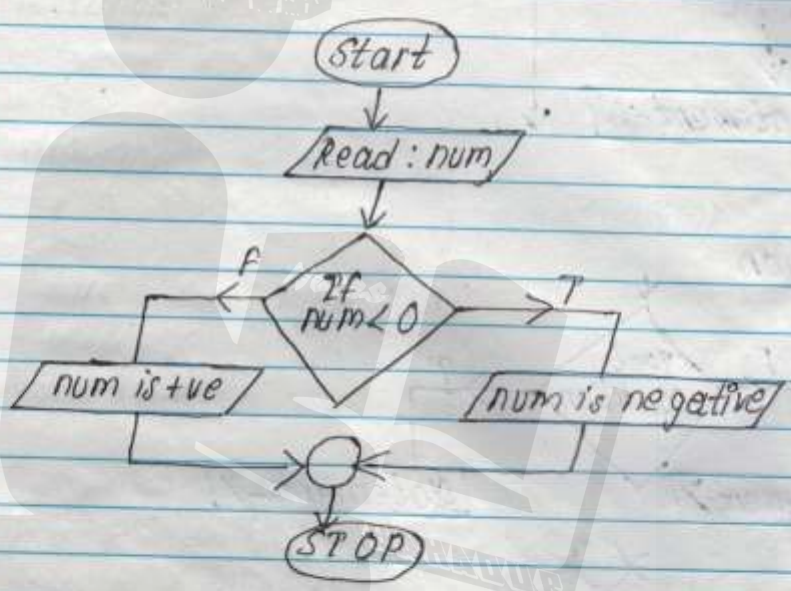


③ Iteration



Eg: Algo and flow chart to determine number is positive or negative.

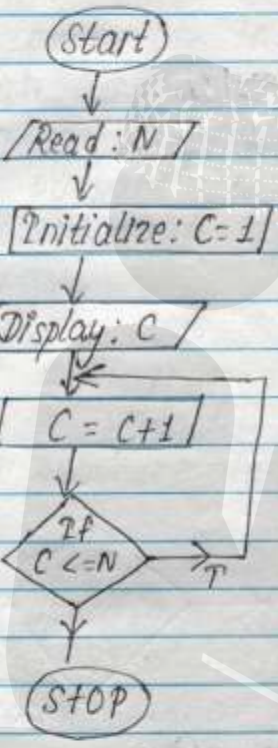
Step 1: Start
Step 2: Read num
Step 3: If num < 0, then
Display: num is negative
ELSE
Display: num is positive
Step 4: Stop



Eg: Algo & Flow chart to display natural numbers from 1 to N

Step 1: Start
Step 2: Read: N
Step 3: Initialize: C = 1
Step 4: Display: C
Step 5: C = C+1
Step 6: If C<=N, go to step (4)

ELSE
go to step (7)
Step 7: Stop

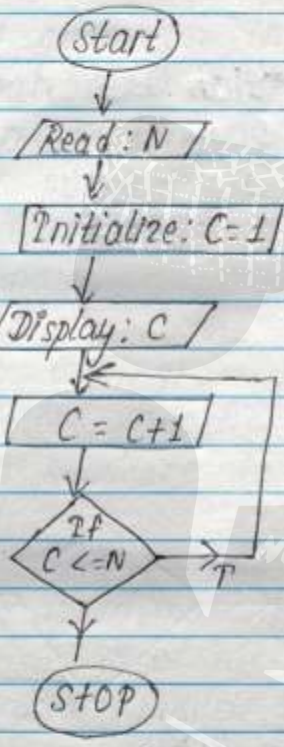


Structure of C-program:

(1) Documentation ~~section~~ section/ Comment section.
/* Program to display Hello world.*

(2) Link Section
include < stdio.h >
include < conio.h >
Void main( )
{

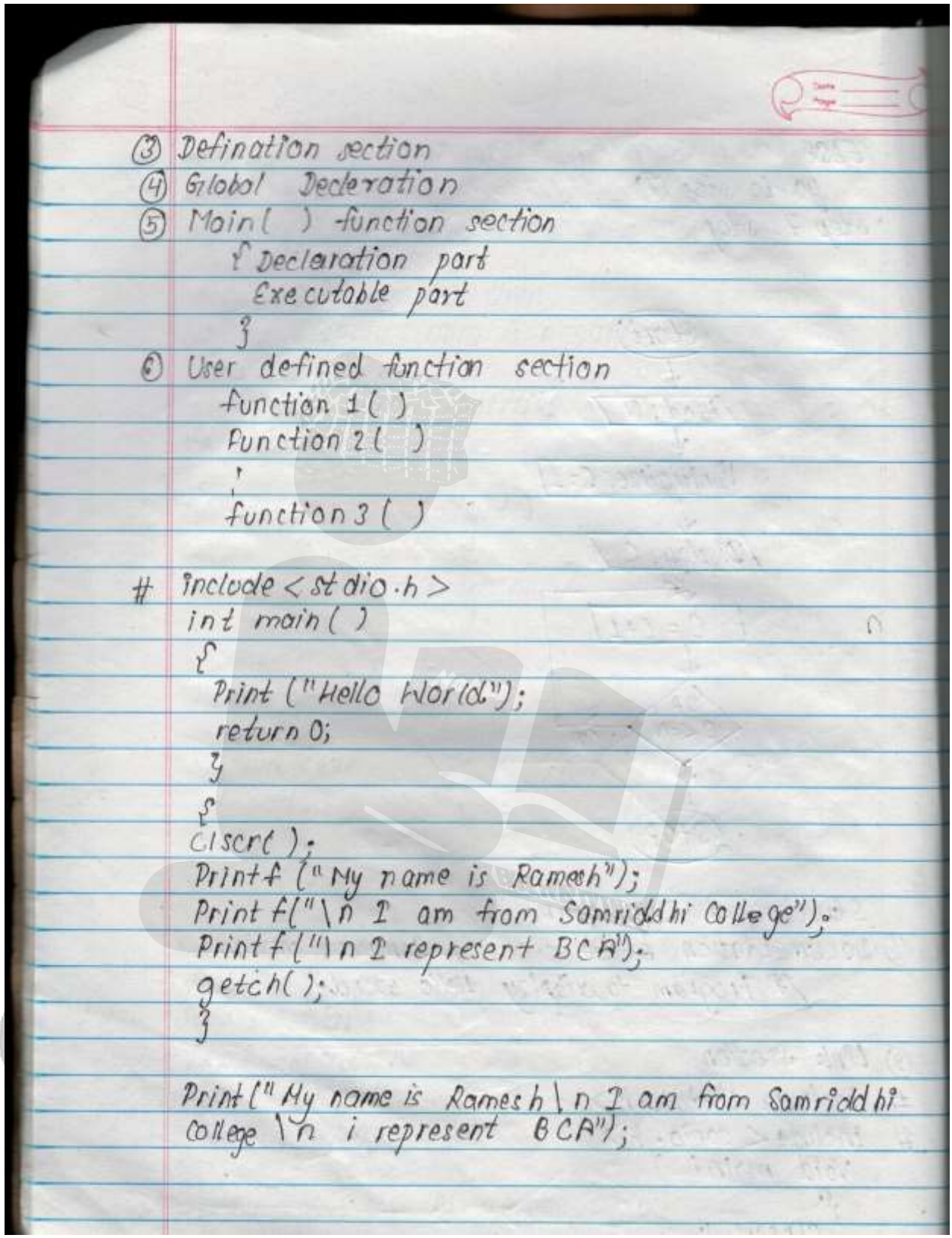
ELSE
go to step (7)
Step 7: Stop



Structure of C-program:

(1) Documentation ~~section~~ section/ Comment section.
/* Program to display Hello world */

(2) Link Section
#include <stdio.h>
#include <conio.h>
Void main( )
{
clrscr( );
Print("Hello world"); → getch( );
}



③ Defination section
④ Global Decleration
⑤ Main( ) function section
{ Declaration part
Executable part
}
⑥ User defined function section
function 1( )
Function 2( )
,
,
function 3( )

```
# include <stdio.h>
int main( )
{
 Print ("Hello World");
 return 0;
}
{
clscr( );
Printf ("My name is Ramesh");
Print f("\n I am from Samriddhi College");
Print f("\n I represent BCA");
getch( );
}

Print("My name is Ramesh\n I am from Samriddhi
College \n i represent BCA");
```

Introduction to C-programming.

C is a general purpose high-level programming language that was originally developed by Dennis Ritchie for the Unix operating system. It was first implemented on the Digital Equipment Corporation PDP-11 computer in 1972.

By defination, C is a general purpose and procedural programming language.

Gieneral purpose: C language is designed for developing software that applies in wide range of application domain.

Procedual: C program is set of functions. Each function performs the specific task in C program. Function are called in sequence to make the program work as designed.

The Unix operating system and virtually all unit of application are written in C-language. Chas now become widely used professional language for various reasons:

- Easy to learn.
- Structural language.
- Produce efficient programs.
- It can handle low-level activities.
- It can be compiled on variety of compiler.

An aspect of C which makes it a powerful programming language is the access it provide to the address where variables are stored. These address

are known as pointers. The access to pointers and the operators which can be performed with distinguishes form of other language.

Feature of C.

① Simple.
C is a simple language in the sense that it provides a structured approach, the rich set of library junctions, data types.

② Machine Independent.
Unlike assembly language, C program

Program to find sum of two numbers.

Variable

Variable Decleration.

Syntax:

Data-type [space] variable name;

Eg: int a;
float b;
char remarks; remarks = 'p';

→
```
int main()
{
int a,b, S; // Here a,b and S are variable of type integer.
a = 50;
b = 25;
S = a+b
Printf("Required sum=%d",S); // Print value of S. Here d is conversion character to represent integer
return 0;
}
```

Initialize a ~~sum~~ a, b and S are variable of type integer.

→ Program to print memory address of variable.

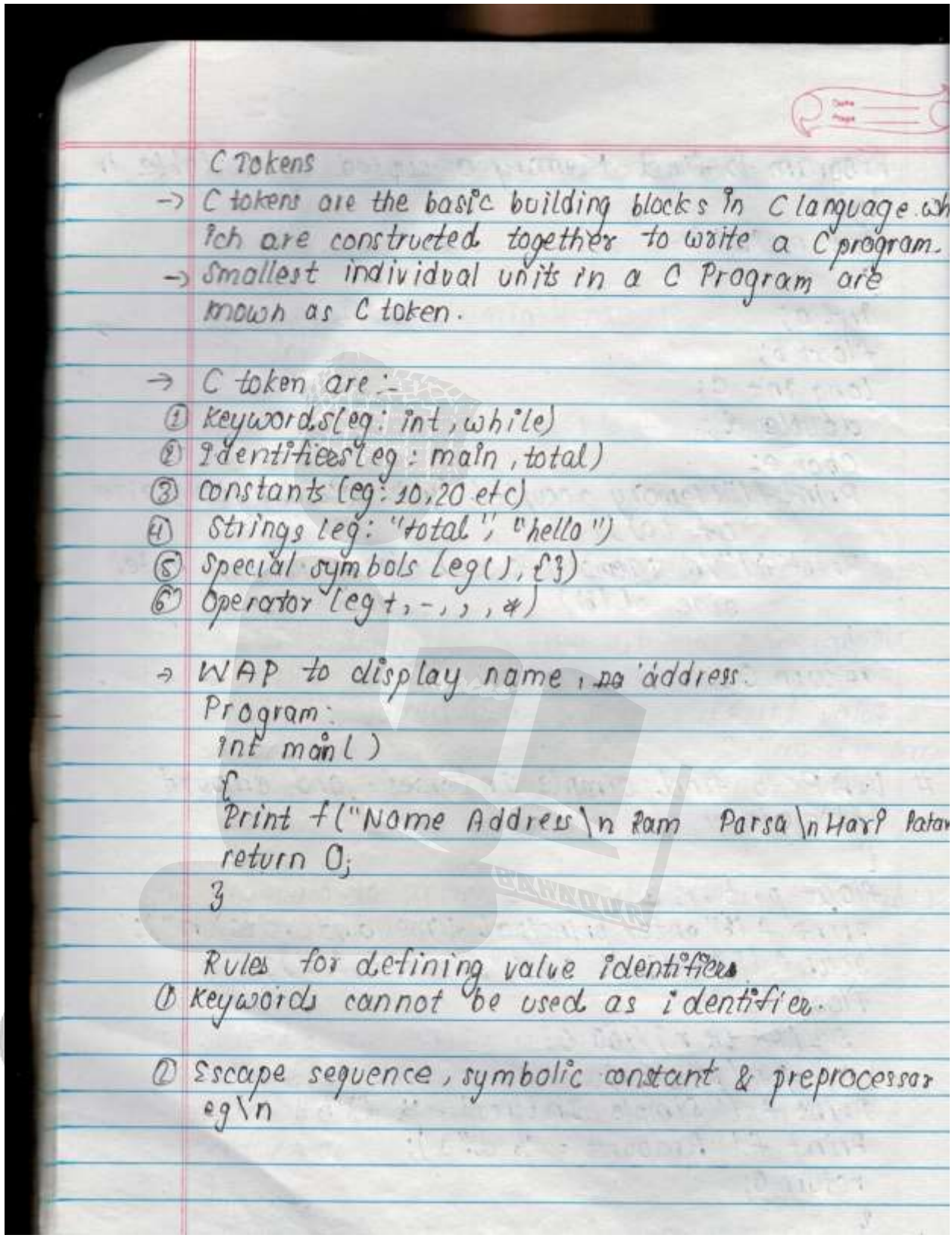
```
int main()
{
  int a;
  float b;
Printf("\n Memory Addres of a is %d",&a)
Printf("\n Memory Address of b is %d",&b)
return 0;
}
```

Program to find Memory occupied by variable in byte.

```
int main()
{
 int a;
 float b;
 long int c;
  double d;
  char e;
  Print f("Memory occupied by a = %d bytes", size
          of (a))
  Print f("/n Memory occupide by b = %d byte,"
          size of (b))
  ;
 return 0;
}
```

WAP to find simple Interset and amount

```
int main()
{
float p, t, r, a;
print f("enter principal, time and rate\n");
scanf ("%d%d%d %d",&p, &t, &r);
float S;
  S = (p*t*r)/100.0;
  a = p+ (p*t*r);
Print f ("Simple Interest = %f\n", S);
Print f("Amount = %d", a);
return 0;
}
```



C Tokens

-> C tokens are the basic building blocks in C language wh
ich are constructed together to write a C program.

-> Smallest individual units in a C Program are
known as C token.

-> C token are :-

① Keywords (eg: int, while)
② Identifiers (eg: main, total)
③ constants (eg: 10, 20 etc)
④ Strings (eg: "total", "hello")
⑤ Special symbols (eg (), {})
⑥ Operator (eg +, -, ,, *)

-> WAP to display name, no address.

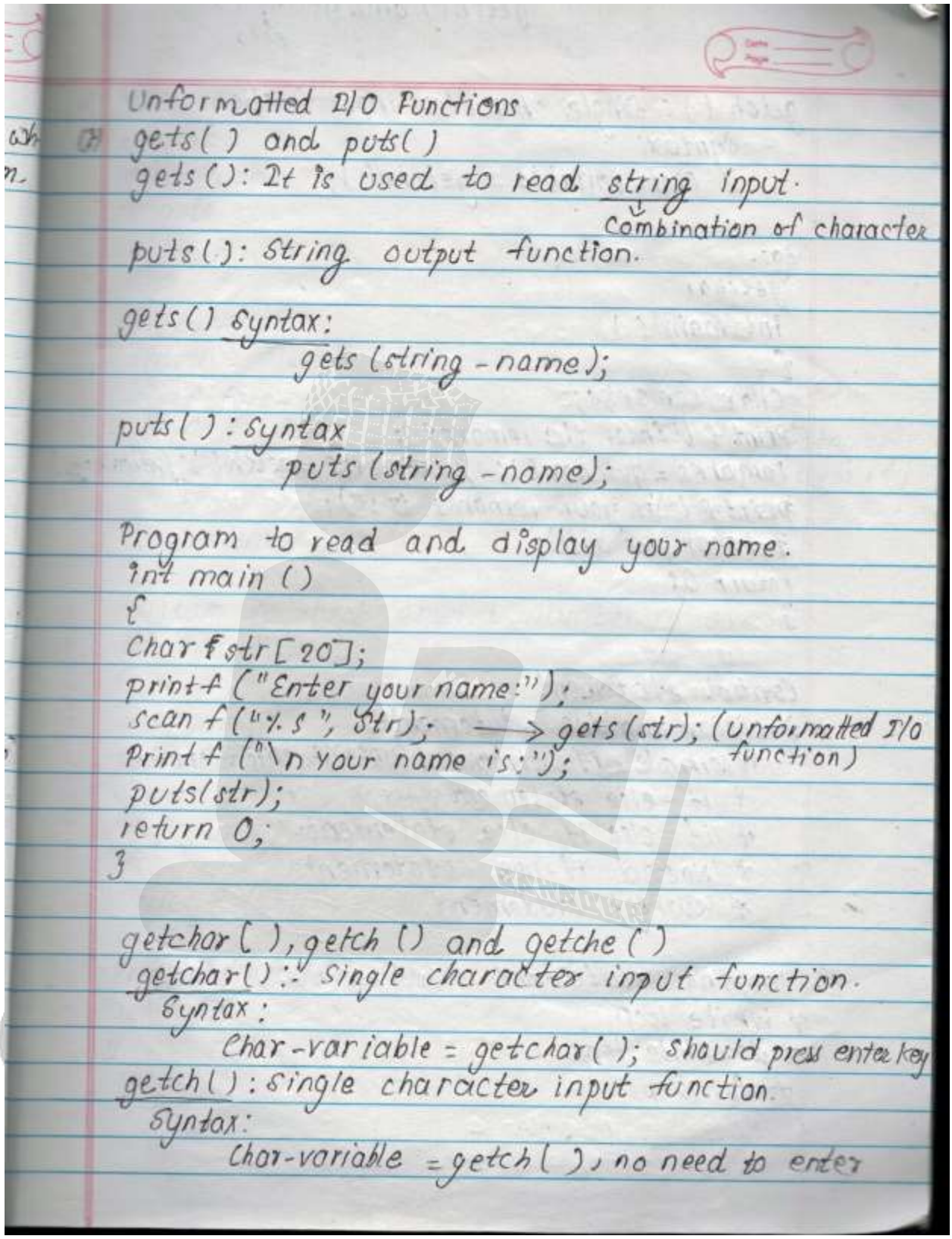
Program:

```
int main()
{
Print f("Name Address\n Ram  Parsa\n Hari Patan
return 0;
}
```

Rules for defining value identifiers

① Keywords cannot be used as identifier.

② Escape sequence, symbolic constant & preprocessor
eg \n



Unformatted I/O Functions

gets( ) and puts( )

gets( ): It is used to read string input.
(combination of character)

puts( ): String output function.

gets( ) Syntax:

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

puts( ): Syntax

```
puts (string-name);
```

Program to read and display your name.

```
int main ()
{
Char str[20];
printf ("Enter your name:");
scanf ("%s", str);  ---> gets(str); (unformatted I/O function)
Printf ("\n your name is:");
puts(str);
return 0;
}
```

getchar( ), getch ( ) and getche ( )

getchar( ): Single character input function.

Syntax:

```
Char-variable = getchar( );
```

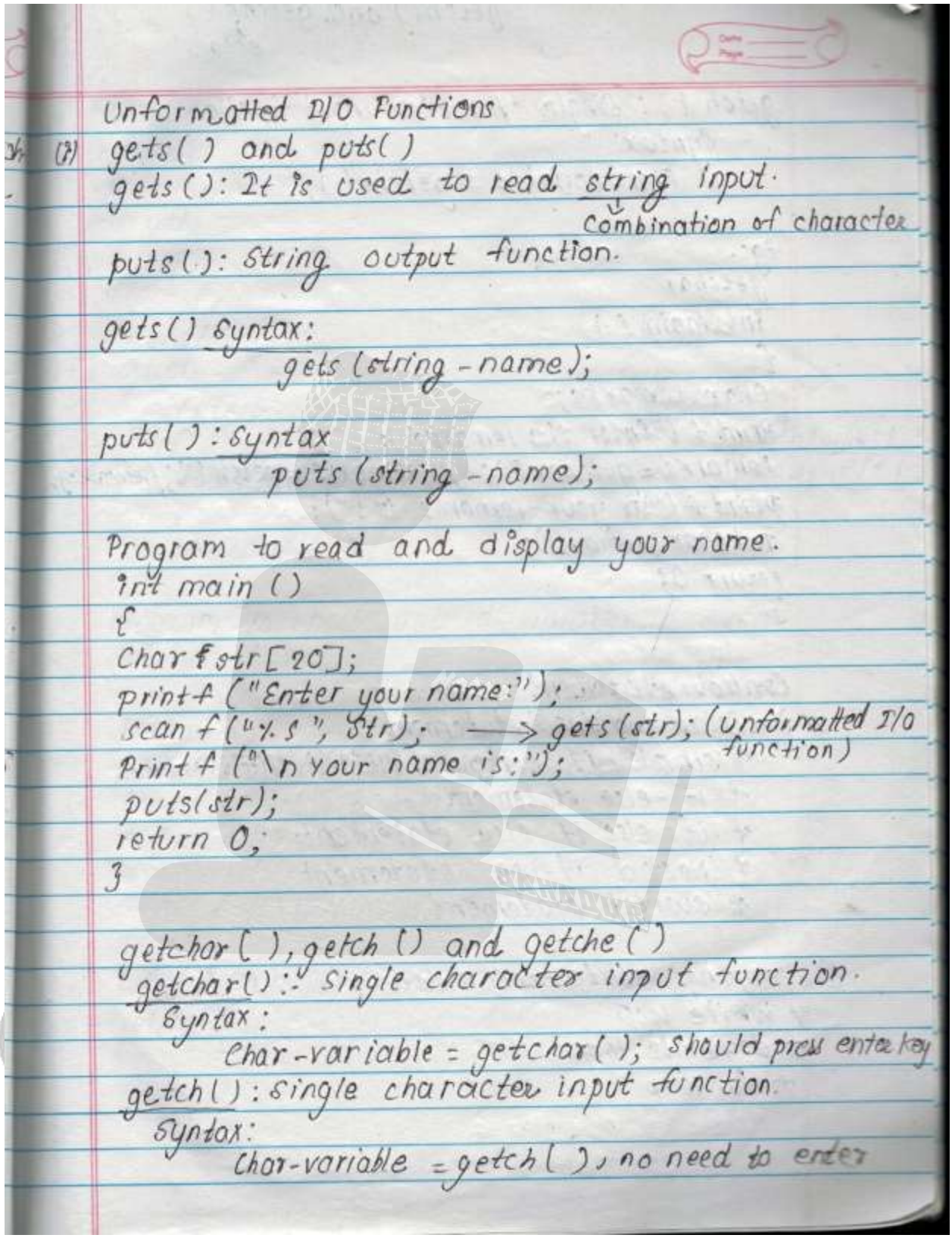
should press enter key

getch( ): Single character input function.

Syntax:

```
Char-variable = getch( );
```

no need to enter



Unformatted I/O Functions

(1) gets( ) and puts( )

gets( ): It is used to read string input.
(combination of character)

puts( ): String output function.

gets( ) Syntax:

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

puts( ): Syntax

```
puts(string-name);
```

Program to read and display your name.

```
int main()
{
char str[20];
printf("Enter your name:");
scanf("%s", str);      ---> gets(str); (unformatted I/O function)
Printf("\n your name is:");
puts(str);
return 0;
}
```

getchar( ), getch( ) and getche( )

getchar( ): single character input function.

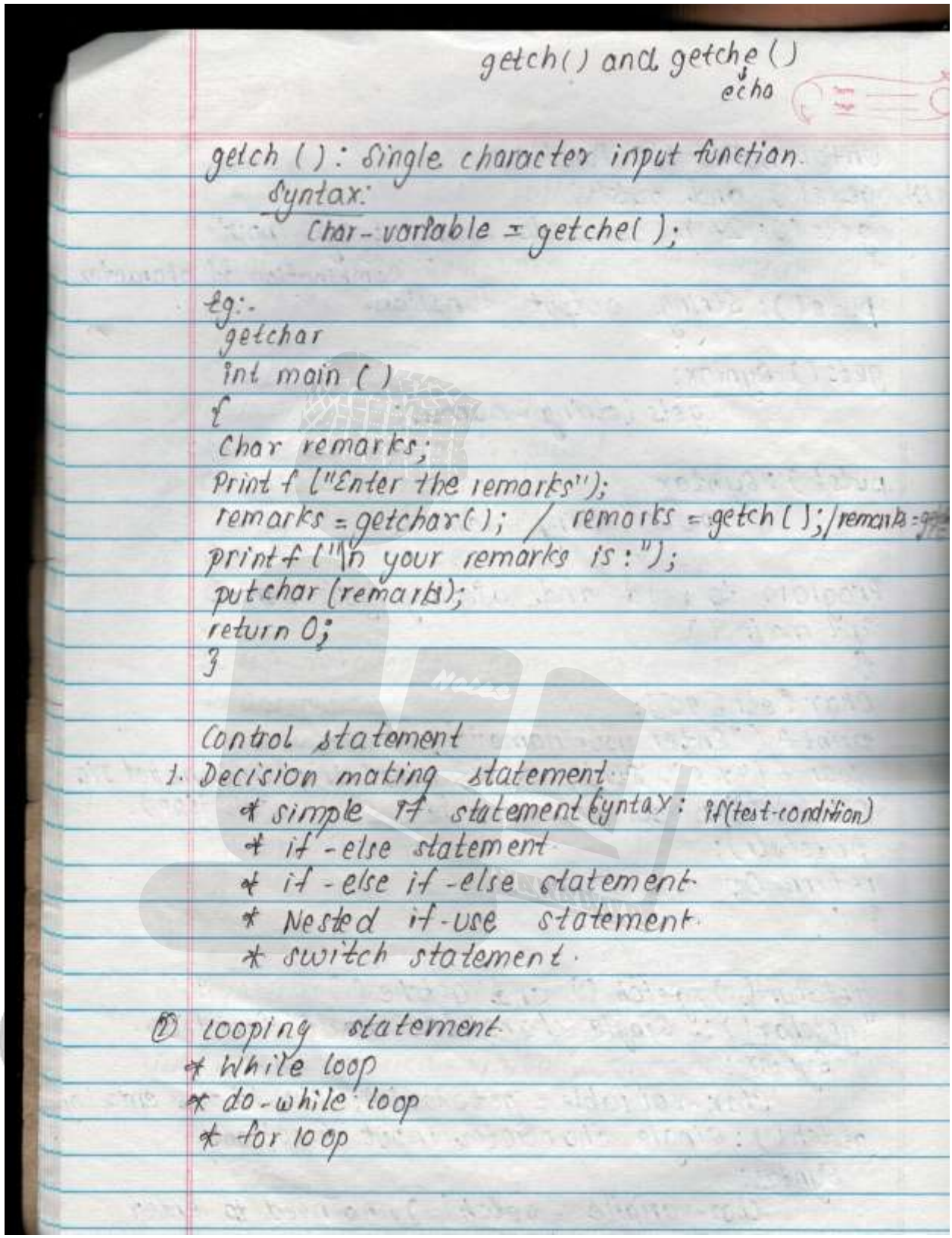
Syntax:

```
char-variable = getchar( );
```
should press enter key

getch( ): single character input function.

Syntax:

```
char-variable = getch( );
```
no need to enter



getch() and getche()
echo

getch(): Single character input function.

Syntax:

Char-variable = getche();

Eg:-

getchar

```
int main()
{
Char remarks;
Printf("Enter the remarks");
remarks = getchar(); / remarks = getch();/remarks=g
printf("\n your remarks is:");
putchar(remarks);
return 0;
}
```

Control statement

1. Decision making statement
   * simple if statement (syntax: if(test-condition)
   * if-else statement
   * if-else if-else statement
   * Nested if-use statement
   * switch statement.

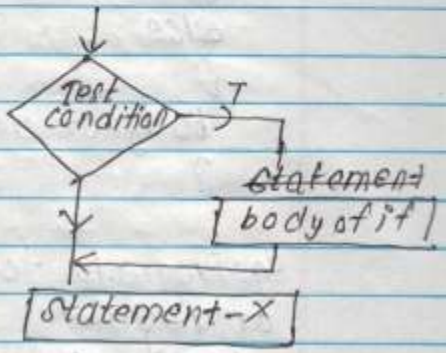
2. looping statement
   * While loop
   * do-while loop
   * for loop

2. Jump statement.
* break
* continue
* goto
* return.

1 If statement.

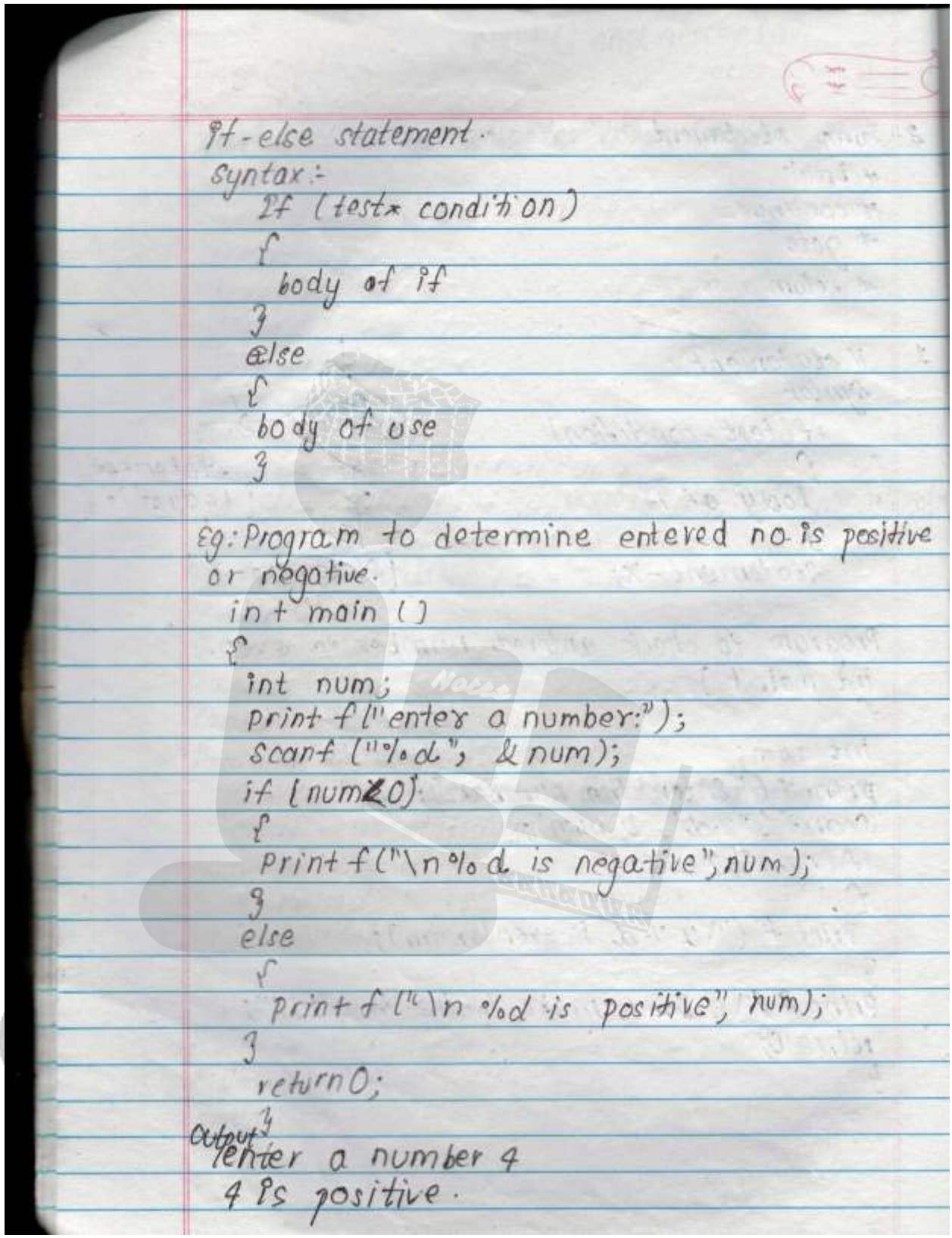
Syntax

```
if(test-condition)
{
 body of if
}
Statement-X;
```



Program to check entered number is even.

```
int main( )
{
int num;
printf("Enter the number:");
scanf("%d", &num);
if(num%2==0);
{
 Print f ("\n %d is even", num);
}
Print f ("\n Dont make noise in class");
return 0;
}
```



If-else statement.

Syntax:-

```
If (test condition)
{
 body of if
}
else
{
 body of use
}
```

Eg: Program to determine entered no is positive or negative.

```
int main ()
{
 int num;
 printf("enter a number:");
 scanf ("%d", &num);
 if (num<0)
 {
  Printf("\n%d is negative", num);
 }
 else
 {
  printf ("\n %d is positive", num);
 }
  return 0;
}
```

Output:

enter a number 4

4 is positive.

Eg: Program to find greater ~~array~~ among two numbers.

```
int main()
{
 int x,y
 Print f("Enter value of x and y:");
 scanf ("%d %d", &x, &y);
 if (x>y)
 {
  Print f("\n %d is greater", x);
 }
 else
 {
  Print f("\n %d is greater", y);
 }
 return 0;
}.
```

WAP that reads coffecients of quadratic equation. $ax^2+bx+c=0$ and find the real and imaginary root.

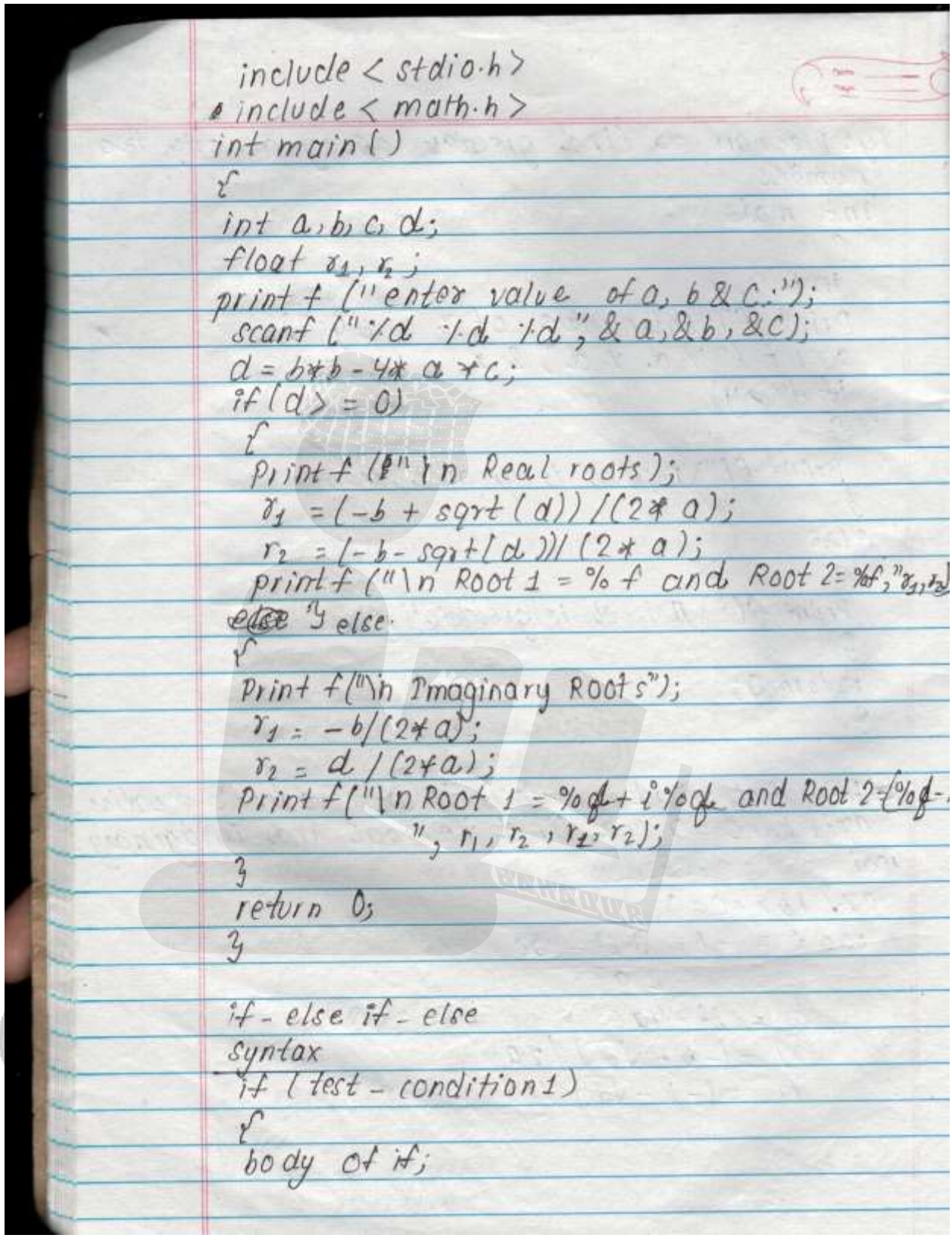
$$ax^2+bx+c=0$$

$$root = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

$$d = b^2-4ac$$

$$r_1 = (-b+\sqrt{d})/2a$$

$$r_2 = (-b-\sqrt{d})/2a$$



```
include <stdio.h>
include <math.h>
int main()
{
int a, b, c, d;
float r1, r2;
printf("enter value of a, b & c:");
scanf("%d %d %d", &a, &b, &c);
d = b*b - 4*a*c;
if(d >= 0)
{
Printf("\n Real roots);
r1 = (-b + sqrt(d)) / (2*a);
r2 = (-b - sqrt(d)) / (2*a);
printf("\n Root 1 = %f and Root 2 = %f", r1, r2
else
{
Print f("\n Imaginary Roots");
r1 = -b/(2*a);
r2 = d/(2*a);
Print f("\n Root 1 = %of + i%of and Root 2 = (%of-
", r1, r2, r2, r2);
}
return 0;
}
```

if - else if - else

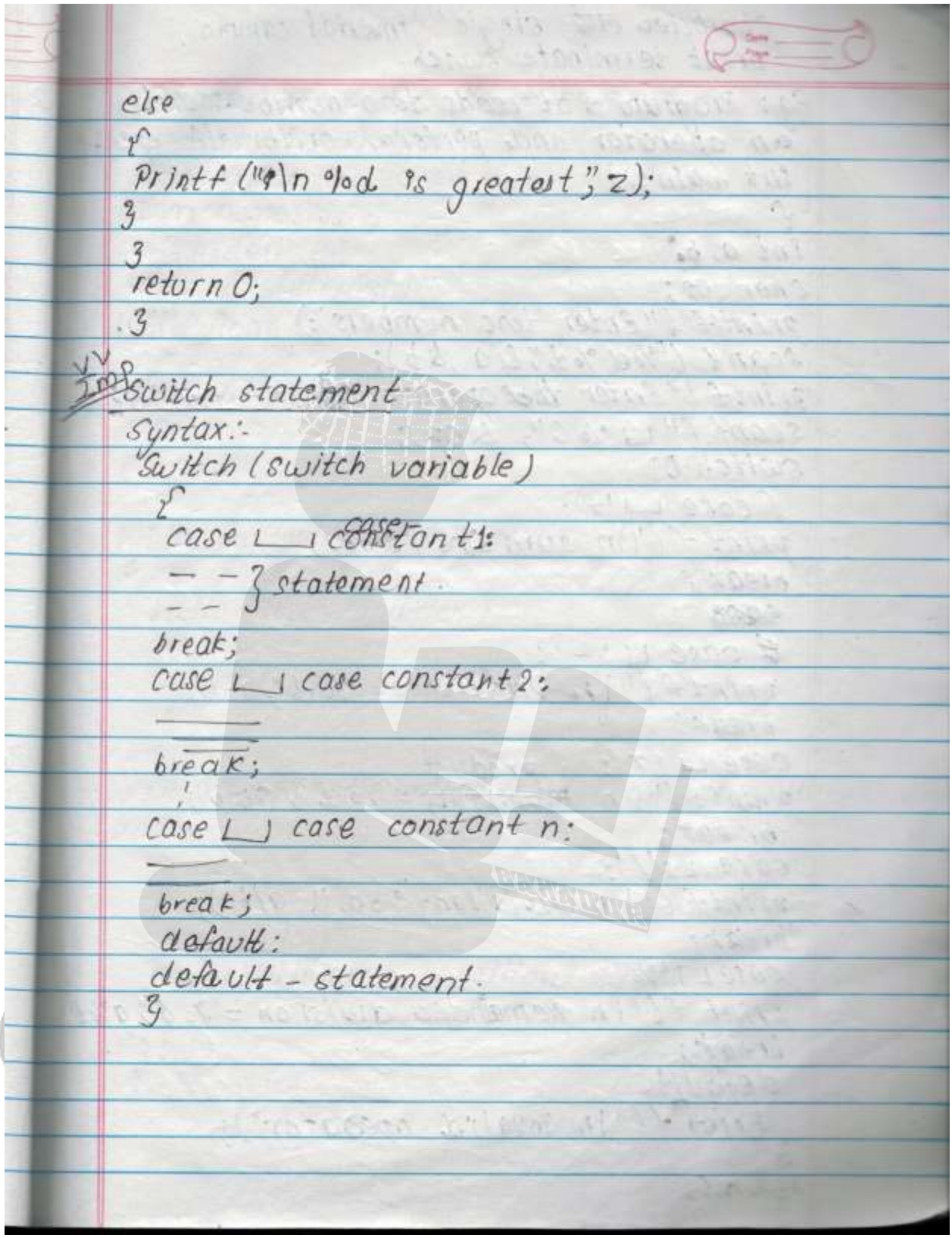
Syntax

```
if (test - condition1)
{
body of if;
```

```
}
else if (test-condition2)
  {
    body of else if;
  }
else if (test-condition3)
  {
   body of else if;
  }
. . . .
. . . .
else
  {
   body of else;
  }
```

Eg: WAP to find Smallest Among three numbers.

```
int main()
{
 int x, y, z;
printf("Enter value of x, and y and z: ");
scanf("%d %d %d", &x, &y, &z);
if (x < y && x < z)
 {
  printf("\n %d is smallest", x);
 }
else if (y < x && y < z)
{
 Printf("\n%d is smallest", y);
}
```

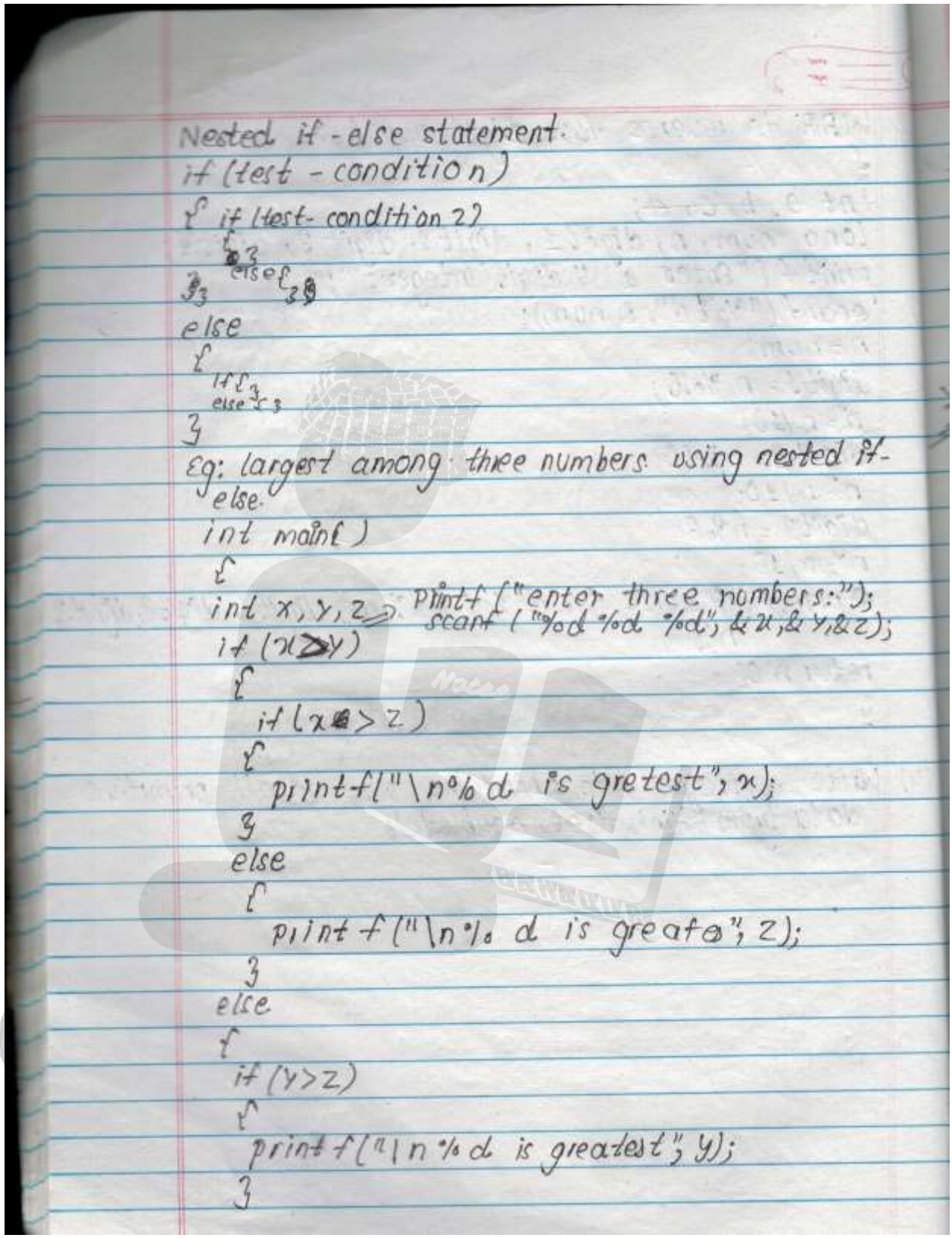


```
else
{
Printf ("\n %d is greatest", z);
}
}
return 0;
}
```

Switch statement

Syntax:-

```
switch (switch variable)
{
case L_I constant1:
- - } statement.
- -
break;
case L_I case constant2:
_____
_____
break;
case L_I case constant n:
_____
_____
break;
default:
default - statement.
}
```

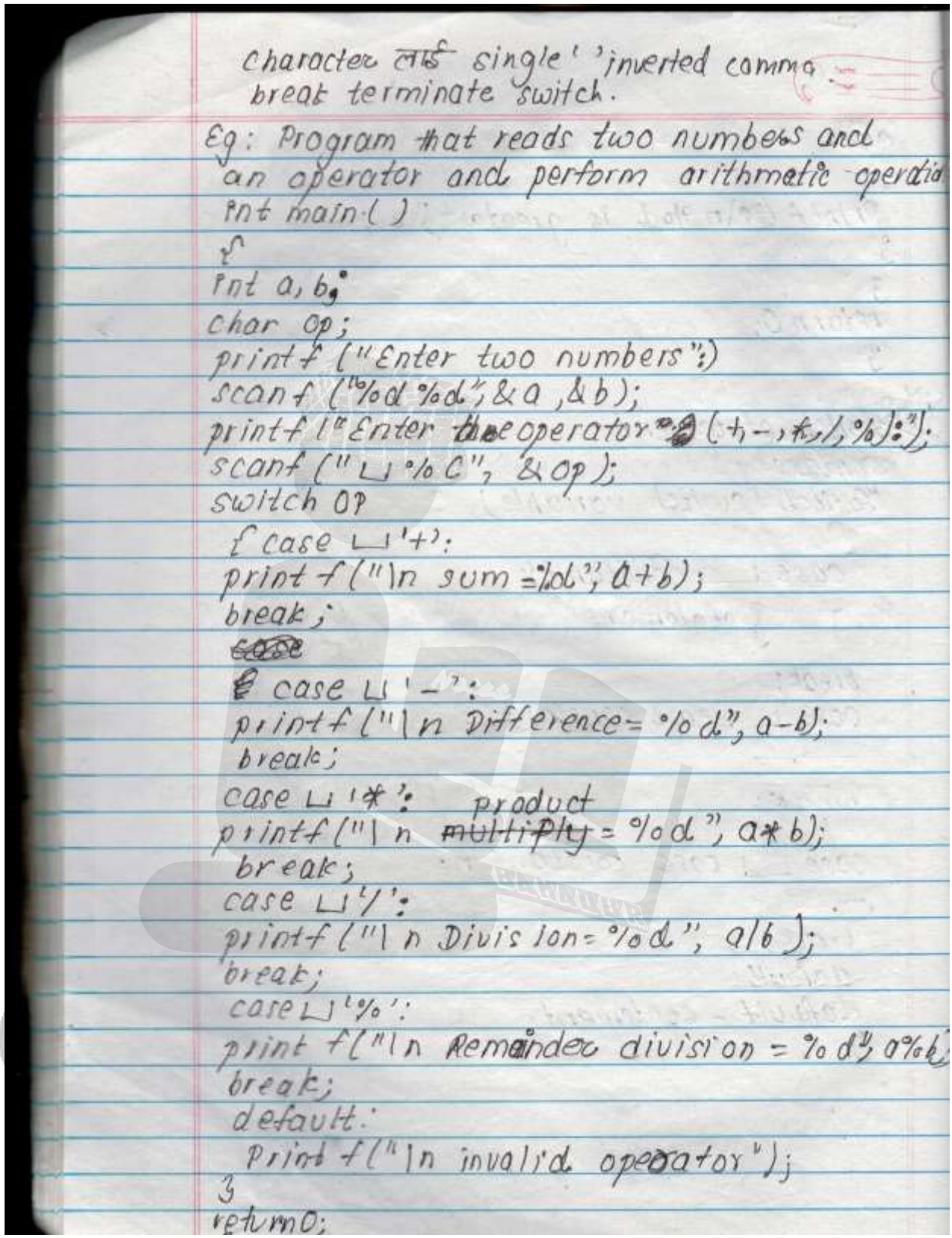


Nested if-else statement.

```
if (test - condition)
{ if (test- condition 2)
  {
  }
  else{
  }
}
else
{
  if{
  }
  else{
  }
}
```

Eg: largest among three numbers using nested if-else.

```
int main()
{
int x, y, z;  printf("enter three numbers:");
              scanf("%d %d %d", &x, &y, &z);
if (x>y)
 {
  if (x>z)
  {
   printf("\n%d is gretest", x);
  }
 else
  {
   printf("\n%d is greato", z);
  }
 else
 {
  if (y>z)
  {
   printf("\n%d is greatest", y);
  }
```



character लाई single ' ' inverted comma
break terminate switch.

Eg: Program that reads two numbers and an operator and perform arithmetic operation

```
int main()
{
int a, b;
char op;
printf("Enter two numbers");
scanf("%d %d", &a, &b);
printf("Enter the operator (+,-,*,/,%):");
scanf(" %c", &op);
switch op
 { case '+':
printf("\n sum = %d", a+b);
break;
 case '-':
printf("\n Difference = %d", a-b);
 break;
case '*': product
printf("\n multiply = %d", a*b);
 break;
case '/':
printf("\n Division = %d", a/b);
break;
case '%':
print f("\n Remainder division = %d", a%b);
 break;
 default:
  Print f("\n invalid operator");
}
return 0;
```

Declaration of variable

int a, b, c; → integer

float a, b, c; → decimal

Char name[20];

→ repetation गर्नु

Looping statement.

1. While loop
2. Do while loop
3. For loop

⇒ While loop (entry control loop)

Syntax

```
While (test-condition)
  {
  body of loop;
  }
```

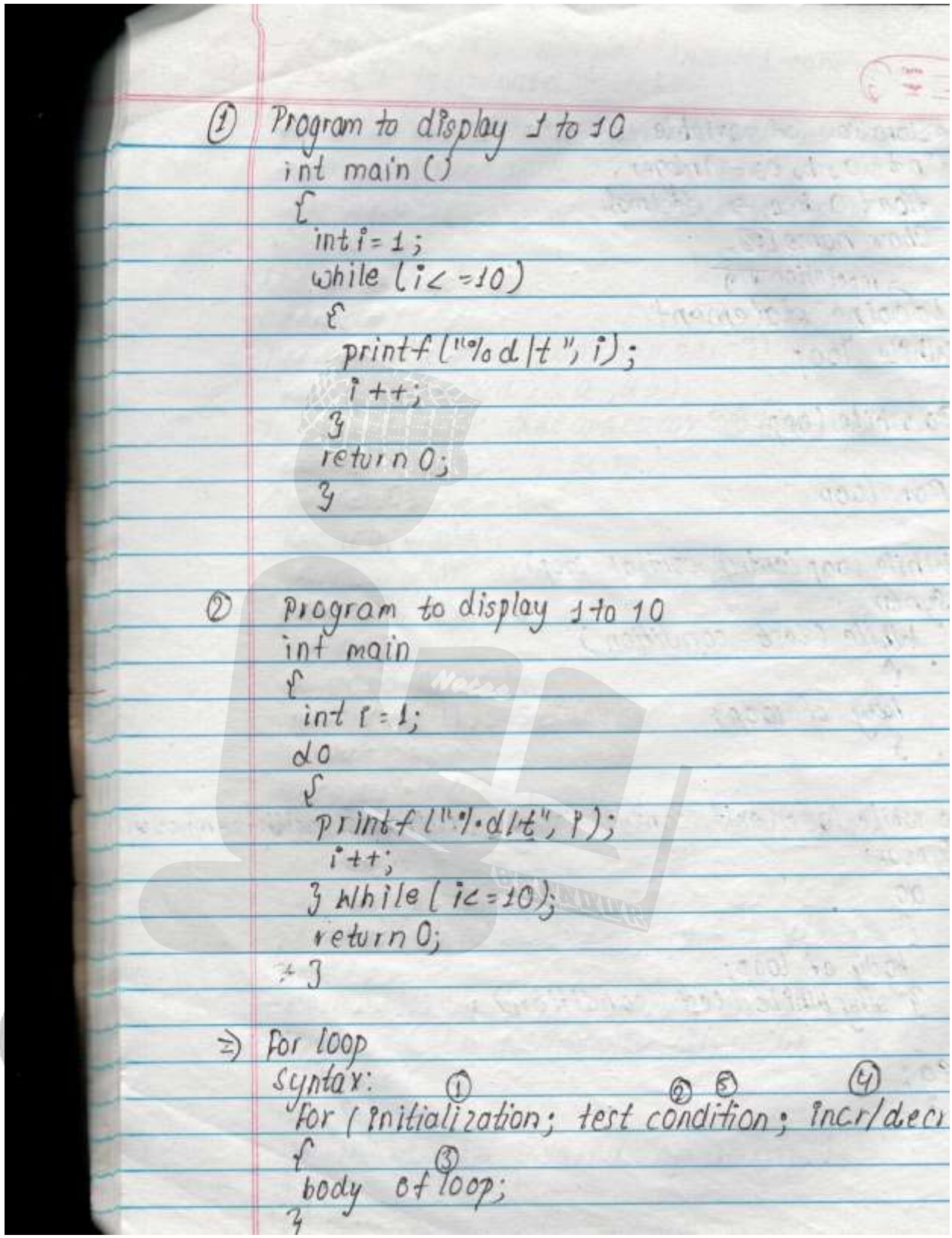
⇒ Do while loop (exit control loop), terminated with semi-colon.

Syntax:

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

Eg:

{



① Program to display 1 to 10

```
int main ()
{
 int i=1;
 while (i<=10)
  {
   printf("%d\t", i);
   i++;
  }
 return 0;
}
```

② Program to display 1 to 10

```
int main
{
 int i=1;
 do
  {
   printf("%d\t", i);
   i++;
  } while (i<=10);
 return 0;
}
```

⇒ For loop

Syntax:

```
for (initialization; test condition; incr/decr
{
 body of loop;
}
```

WAP to determine entered number is palindrom or not

Palindrom.

If reverse of any number is equal to number itself.

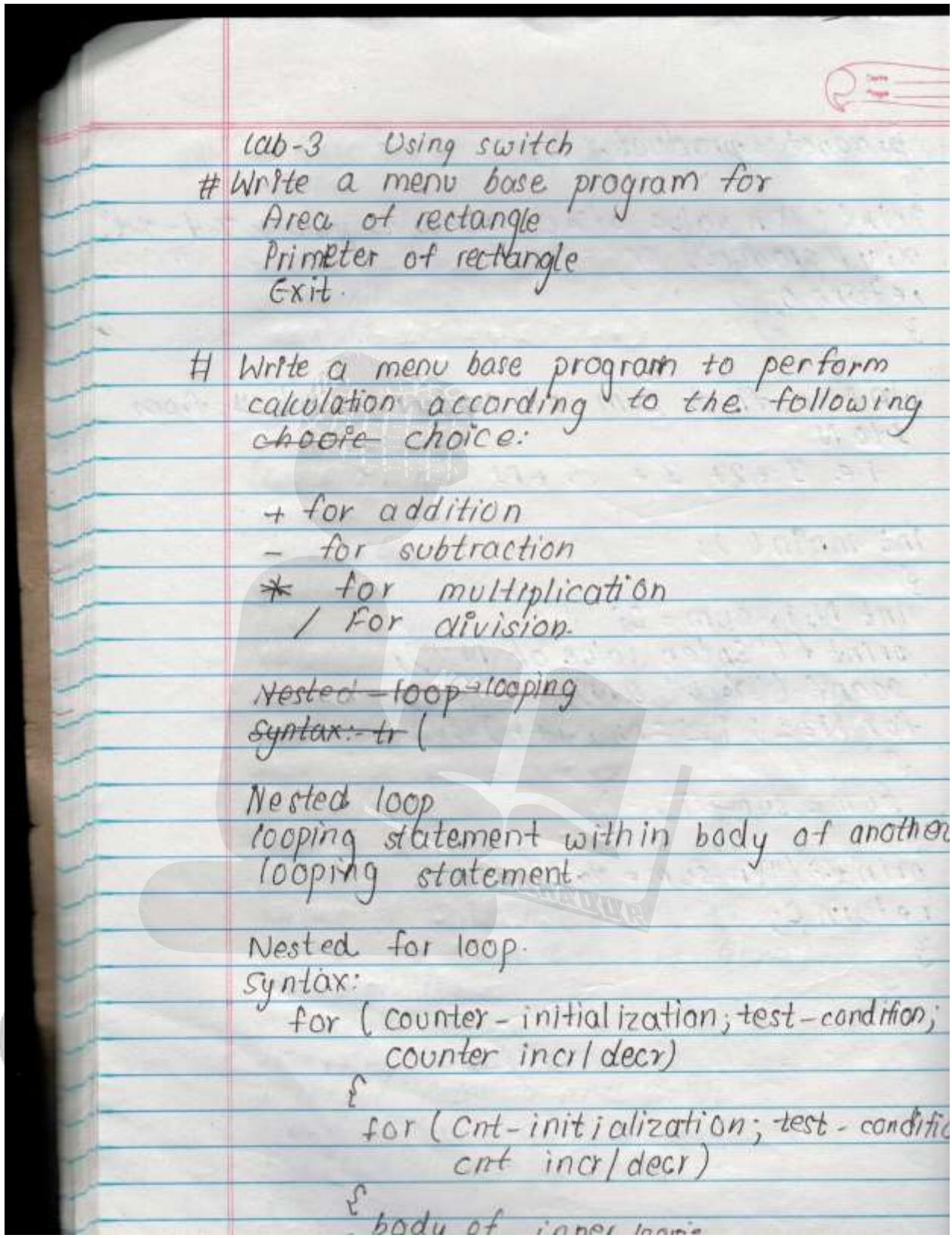
N = 143 rev = 341

≠ Not palindrome.

N = 151 rev = 151

= palindrome.

```
int main()
{
int N, rem, rev = 0;
printf("Enter value of N:");
scanf("%d", &N);
if (rev == t) t = N
While(N > 0)
{ rem = N%10 rev = revX10+rem N = N/10; } if(rev==t
printf("\n%d is palindrome", t);
}
else
{
printf("\n %d is not palindrome", t);
}
return 0;
}
```



Lab-3 Using switch

Write a menu base program for
Area of rectangle
Primeter of rectangle
Exit.

Write a menu base program to perform calculation according to the following ~~choose~~ choice:

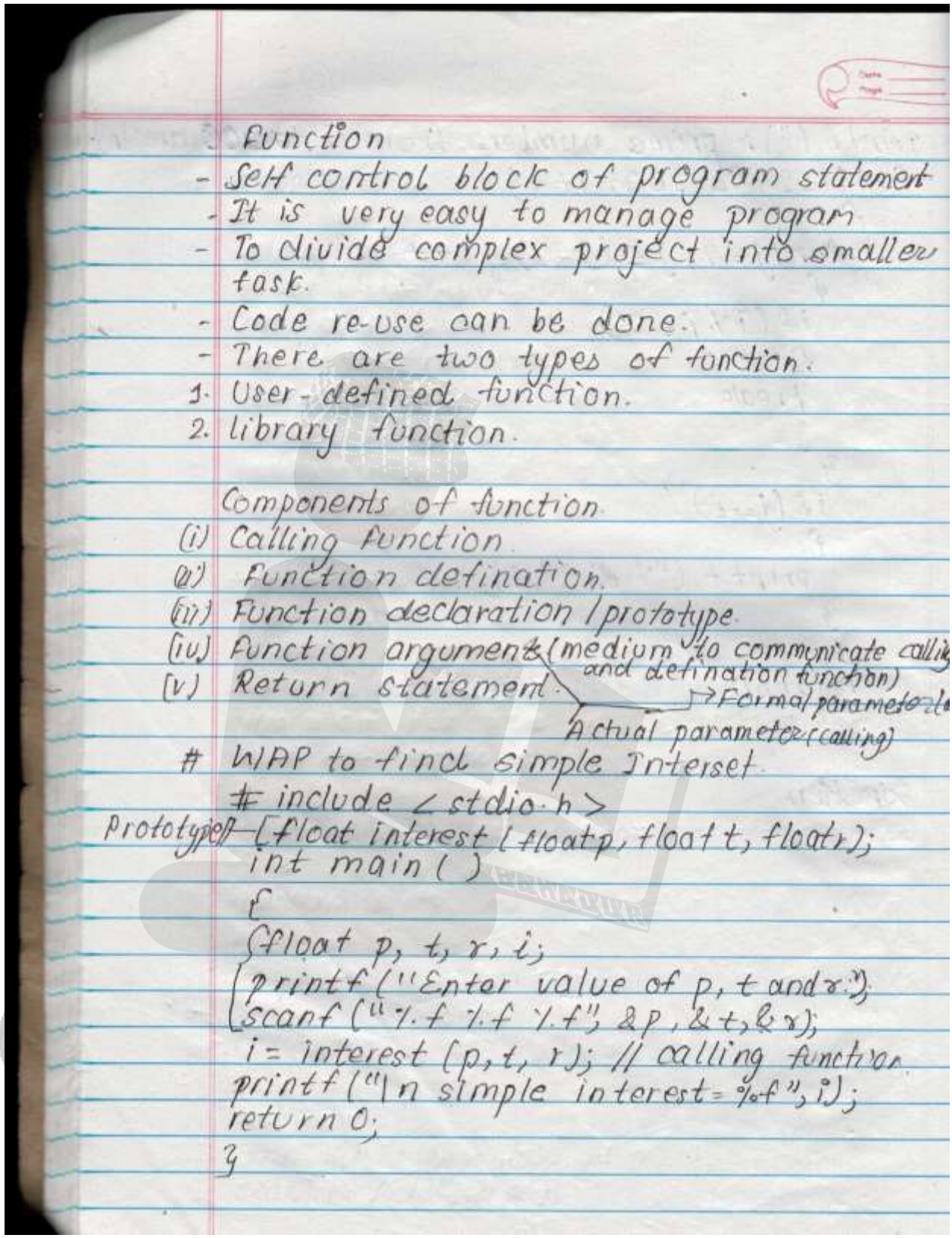
+ for addition
- for subtraction
* for multiplication
/ for division.

~~Nested - loop looping~~
~~Syntax: tr~~ (

Nested loop
looping statement within body of another looping statement.

Nested for loop.
Syntax:

```
for (counter-initialization; test-condition;
     counter incr/decr)
{
   for (cnt-initialization; test-condi
        cnt incr/decr)
   {
     body of inner loop
```

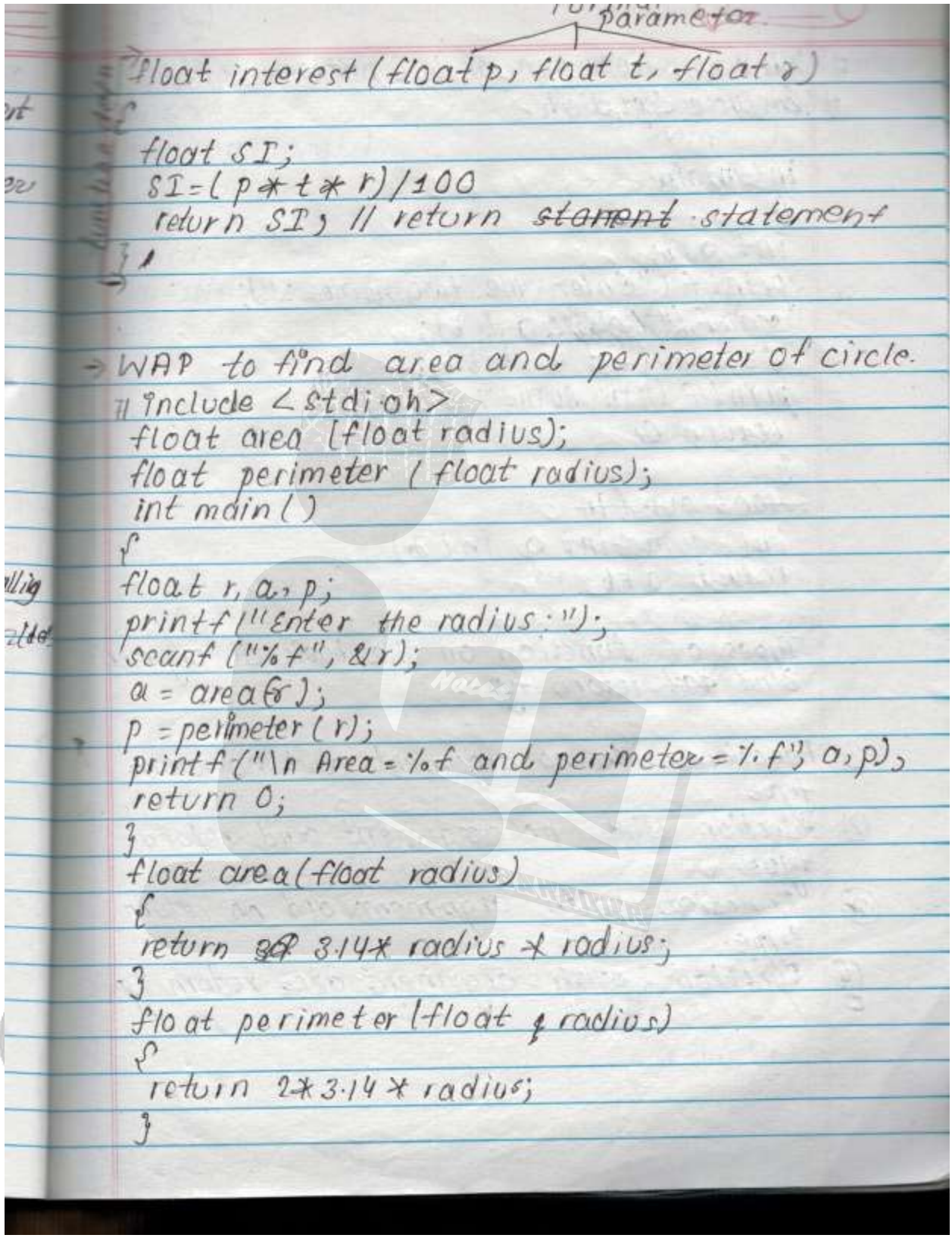


Function
- Self control block of program statement
- It is very easy to manage program
- To divide complex project into smaller task.
- Code re-use can be done.
- There are two types of function.
1. User-defined function.
2. library function.

Components of function
(i) Calling function
(ii) Function defination.
(iii) Function declaration / prototype
(iv) Function arguments (medium to communicate calling and defination function)
(v) Return statement.
- Formal parameter
- Actual parameter (calling)

WAP to find simple Interset

```
#include <stdio.h>
Prototype: float interest (float p, float t, float r);
int main()
{
float p, t, r, i;
printf("Enter value of p, t and r:");
scanf("%f %f %f", &p, &t, &r);
i = interest (p, t, r); // calling function
printf("\n simple interest = %f", i);
return 0;
}
```

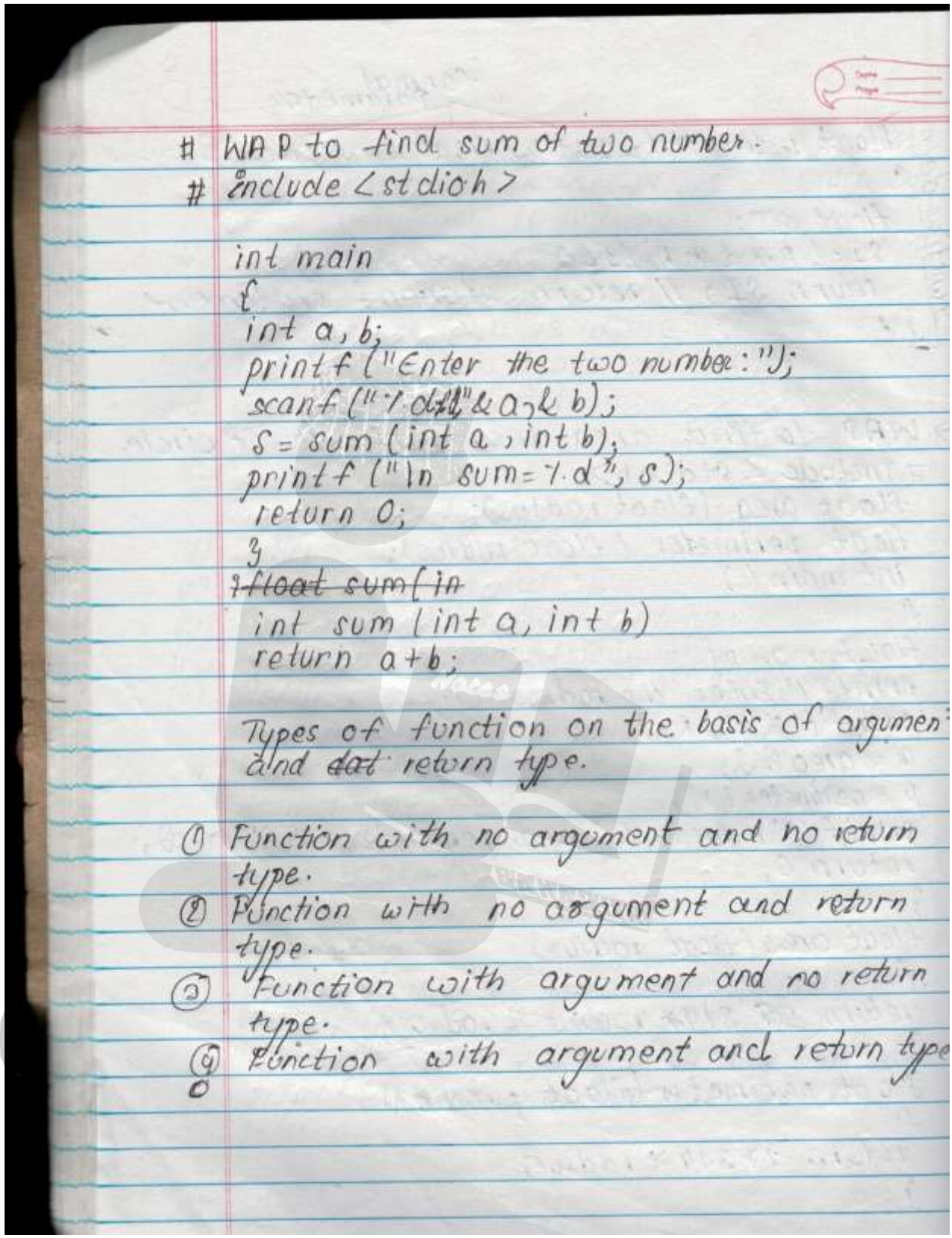


parameter

```
float interest (float p, float t, float r)
{
 float SI;
 SI=(p*t*r)/100
 return SI; // return ~~stament~~ statement
}
```

→ WAP to find area and perimeter of circle.

```
#include <stdio.h>
float area (float radius);
float perimeter (float radius);
int main()
{
float r, a, p;
printf("Enter the radius:");
scanf("%f", &r);
a = area(r);
p = perimeter(r);
printf("\n Area = %f and perimeter = %f", a, p);
return 0;
}
float area(float radius)
{
 return 3.14* radius * radius;
}
float perimeter (float radius)
{
 return 2*3.14*radius;
}
```



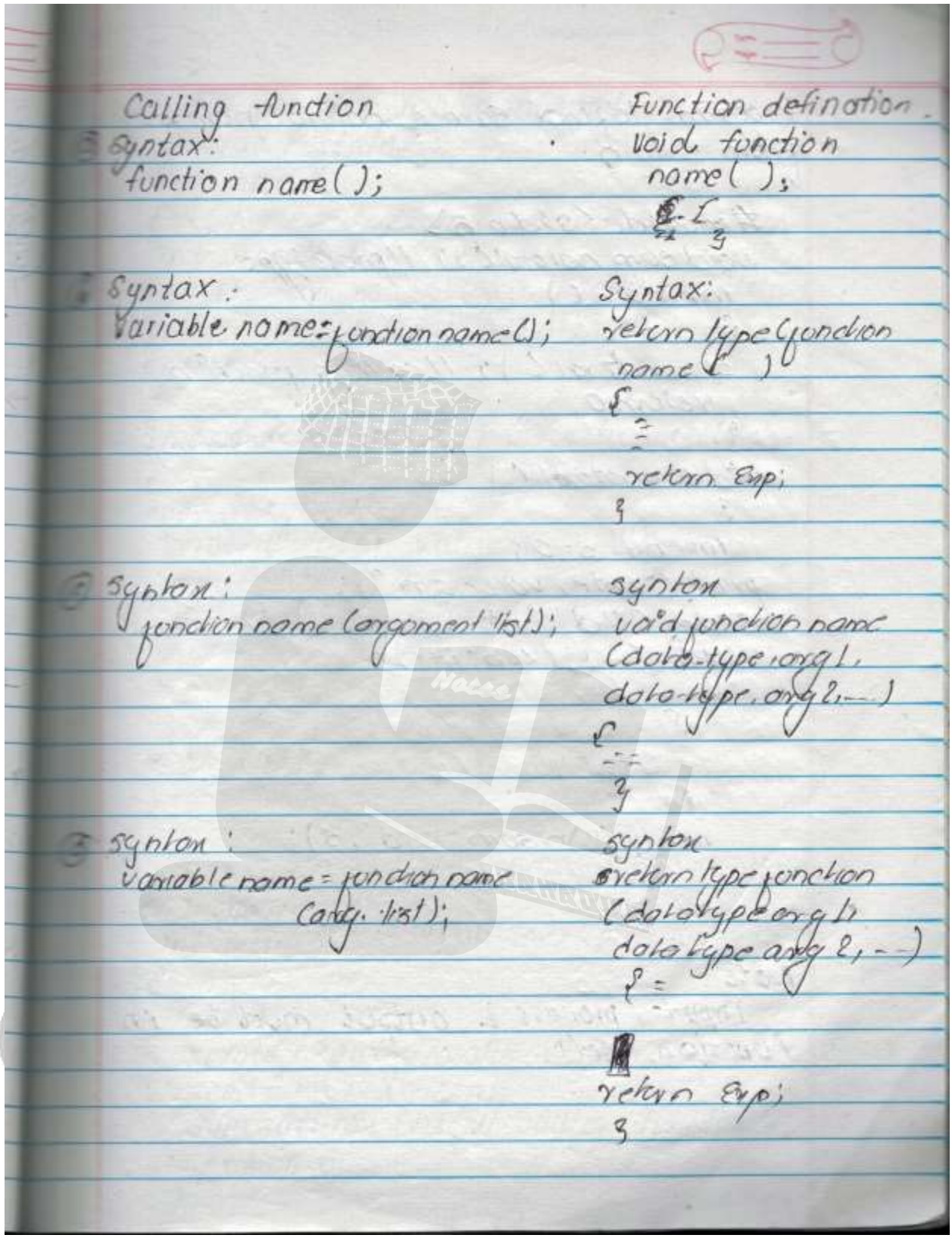
WAP to find sum of two number.

```
#include <stdio.h>

int main
{
int a, b;
printf("Enter the two number:");
scanf("%d%d",&a,&b);
s = sum(int a, int b);
printf("\n sum=%d", s);
return 0;
}
int sum(int a, int b)
return a+b;
```

Types of function on the basis of argumen and ~~dat~~ return type.

1. Function with no argument and no return type.
2. Function with no argument and return type.
3. Function with argument and no return type.
4. Function with argument and return type



Calling function

1. Syntax:
   function name();

2. Syntax:
   variable name = function name();

3. Syntax:
   function name(argument list);

4. Syntax:
   variable name = function name (arg. list);

Function definition

1. Void function name();
   {
   }

2. Syntax:
   return type (function name())
   {
   return Exp;
   }

3. Syntax
   void function name (data-type arg1, data-type arg2, --)
   {
   }

4. Syntax
   return type function (data type arg1, data type arg2, --)
   {
   return Exp;
   }

(1) What is function? Explain its advantages.

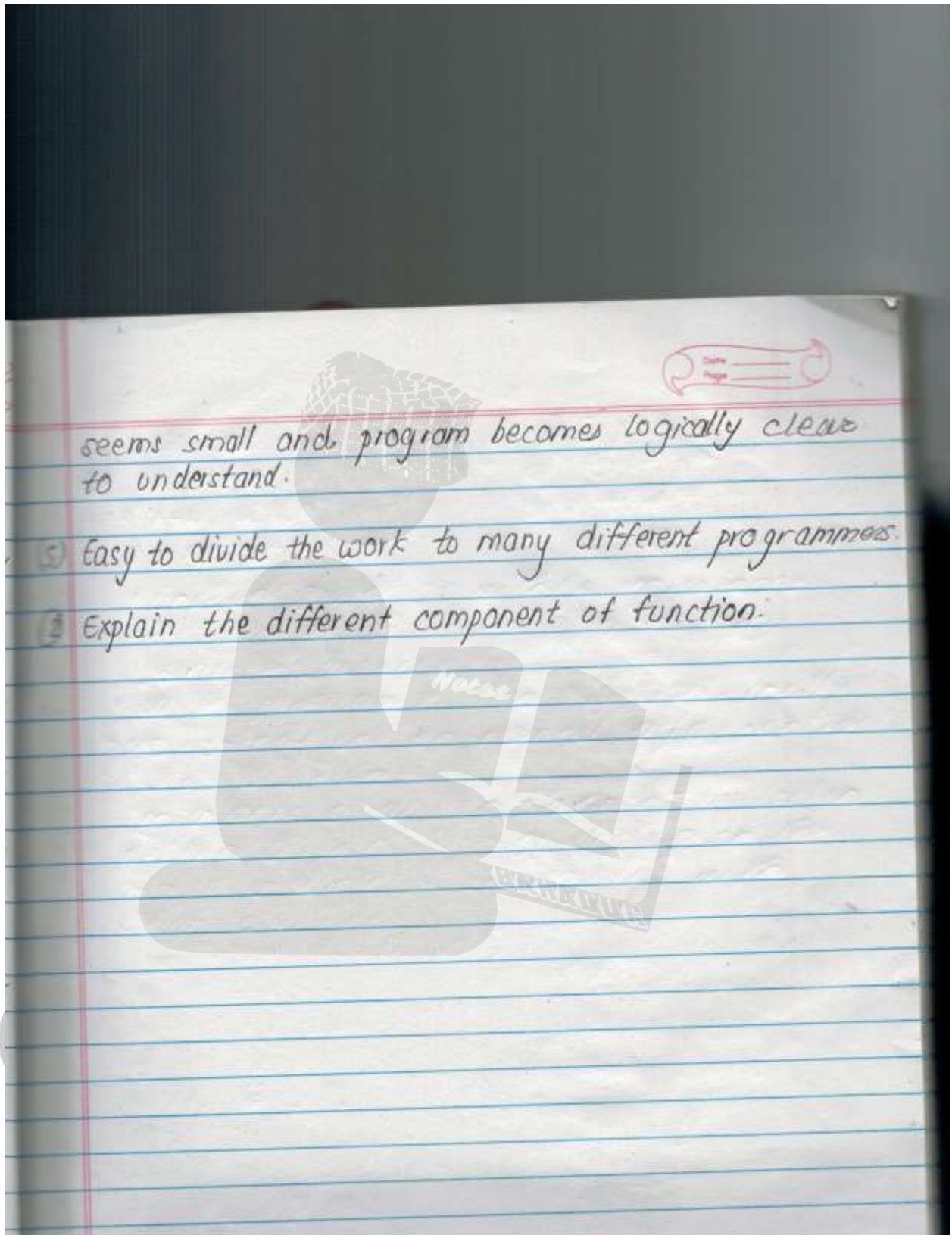
→ A function is a self ~~control~~ contained block of statement that performs a particular task or job. It is a logical unit composed of a number of statements into a unit and give it a name. The advantages of functions are:-

(1) Manageability
Functions are used to specify job. It makes programs significantly edsier to understand, debug, testing and ~~mat~~ maintain by breaking them up into easily manageable chunks.

(2) Non-redundant programming.
The activity that is to be ~~a~~ accessed repeatdly multiple times from several different places with or outside the program is written within function. If the function is not used in such situation, the code of same activity is to be re-written every time.

(3) Code Reusability.
A single function can be used multiple times by a single program from different places.

(4) Logical Clarity.
When a single program is decomposed into various well defined function the main program consists of a series of function calls rather than countless line of code so that the size of main progra



seems small and program becomes logically clear to understand.

5. Easy to divide the work to many different programmes.

3. Explain the different component of function.

Types of function call.

1. Function call by value
2. Function call by Reference / Address

1. Function call by value.

Eg: Program to ~~so~~ swap the content of two variables.

```
#include <stdio.h>

int main()
{
 int a, b;
 printf("enter value of a and b:");
 scanf("%d", &a, &b);
 printf("\n before swap : a=%d and b=%d", a, b);
 swap(a, b); // calling function
 printf("\n After swap : a = %d and b=%d", a, b);
 return 0;
}
void swap(int a, int b)
{
 int temp;
 temp = a;
 a = b;
 b = temp;
printf("\n value of a = %d & b = %d", a, b);
}
```

Pointer

Pointer is a special type of variable that stores memory address of another variable of similar type.

Declaration:

data type └─┘ * Ptr-name;

Eg: int * p, * q;

Initializing a pointer.

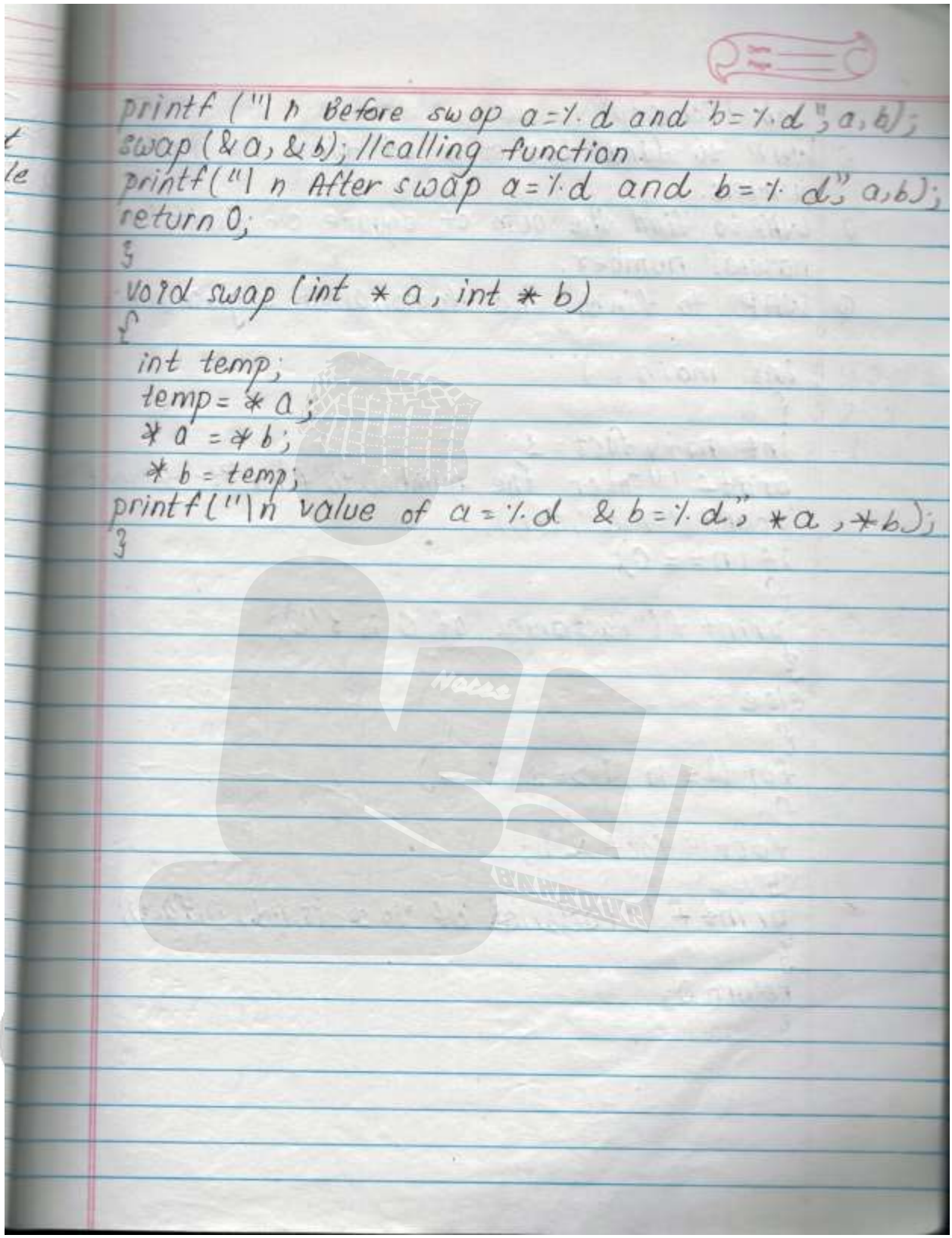
Ptr-name = & variable-name;

Eg: int a, * p;

p = & a; // Address of a is assigned to pointer p.

```
int main()
{
 int a = 10, * p;
   p = &a;
 printf ("\n value of a = %d", a);
 printf ("\n value of a = %d", *p);
```

```
# include <stdio.h>
 void swap (int *, int *);
 int main()
 {
  int a, b;
 printf ("Enter value of a and b");
 scanf ("%d %d", &a, &b);
```



```
printf ("\n Before swap a=%d and b=%d", a, b);
swap (&a, &b); //calling function
printf("\n After swap a=%d and b=%d", a,b);
return 0;
}
void swap (int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
printf("\n value of a=%d & b=%d", *a, *b);
}
```

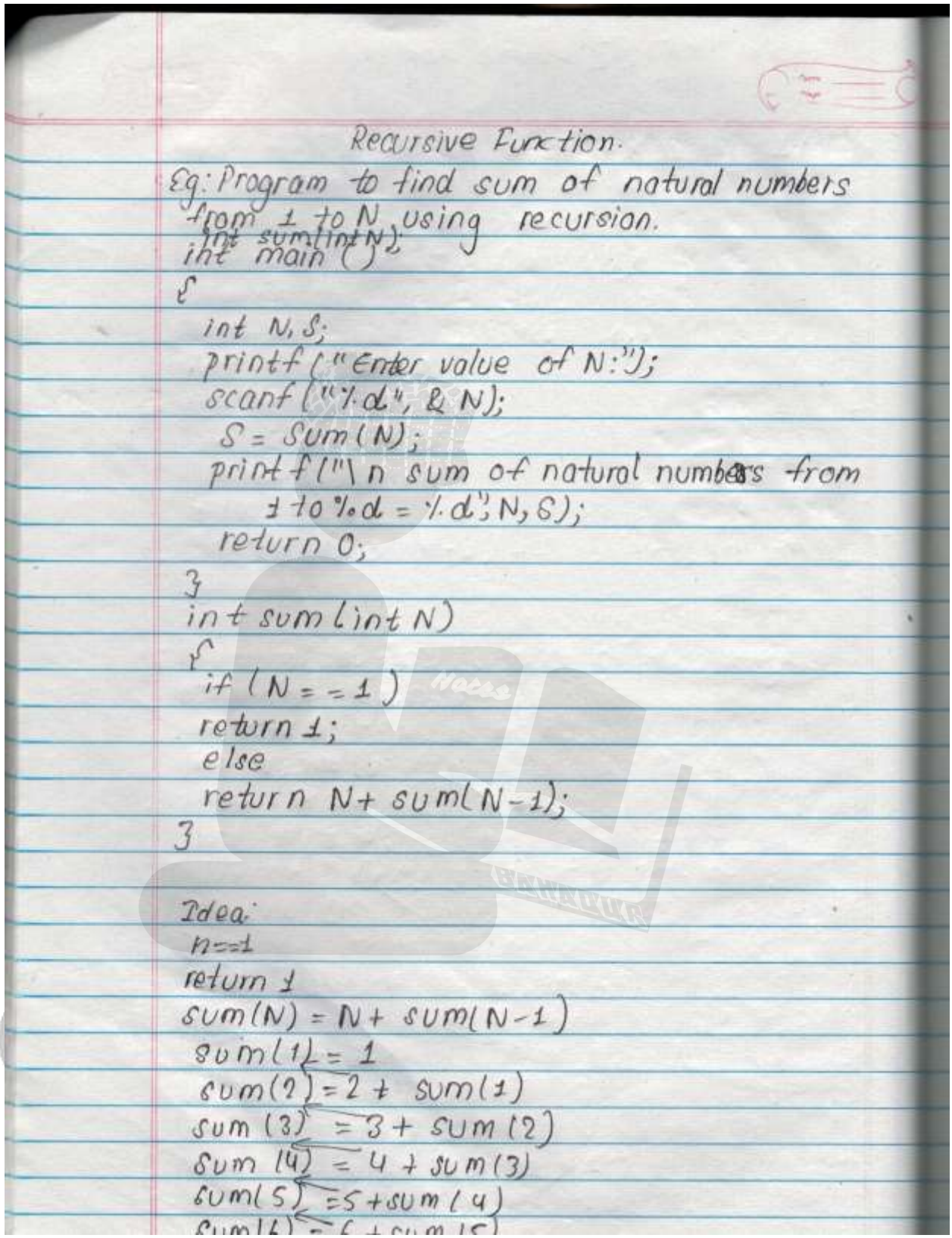
Lab (For loop)

Q WAP to find the sum of first 'n' natural numbers.

Q WAP to find the sum of square of first 'n' natural number.

Q WAP to find the factorial of a given numb

```
int main()
{
int n, i, fact = 1;
printf("Enter the number to find its factorial");
scanf("%d", &n);
if (n == 0)
{
print f("Factorial of 0 is 1");
}
else
{
for (i = n, i >= 1, i--)
{
fact = fact * i;
}
printf("Factorial of %d is %d", n, fact);
}
return 0;
}
```



Recursive Function.

Eg: Program to find sum of natural numbers from 1 to N using recursion.

```
int sum(int N);
int main()
{
  int N, S;
  printf("Enter value of N:");
  scanf("%d", &N);
  S = sum(N);
  printf("\n sum of natural numbers from
      1 to %d = %d", N, S);
  return 0;
}
int sum(int N)
{
  if (N == 1)
  return 1;
  else
  return N + sum(N-1);
}
```

Idea:

```
n==1
return 1
sum(N) = N + sum(N-1)
 sum(1) = 1
 sum(2) = 2 + sum(1)
 sum(3) = 3 + sum(2)
 sum(4) = 4 + sum(3)
 sum(5) = 5 + sum(4)
 sum(6) = 6 + sum(5)
```

Fibonaccii Sequence

1, 1, 2, 3, 5, 8, 13, ...

```
int fibo(int N);
int main()
{
if(N==1||N==2)
return 1;
else
return(fibo(N-1)+fibo(N+2);
```

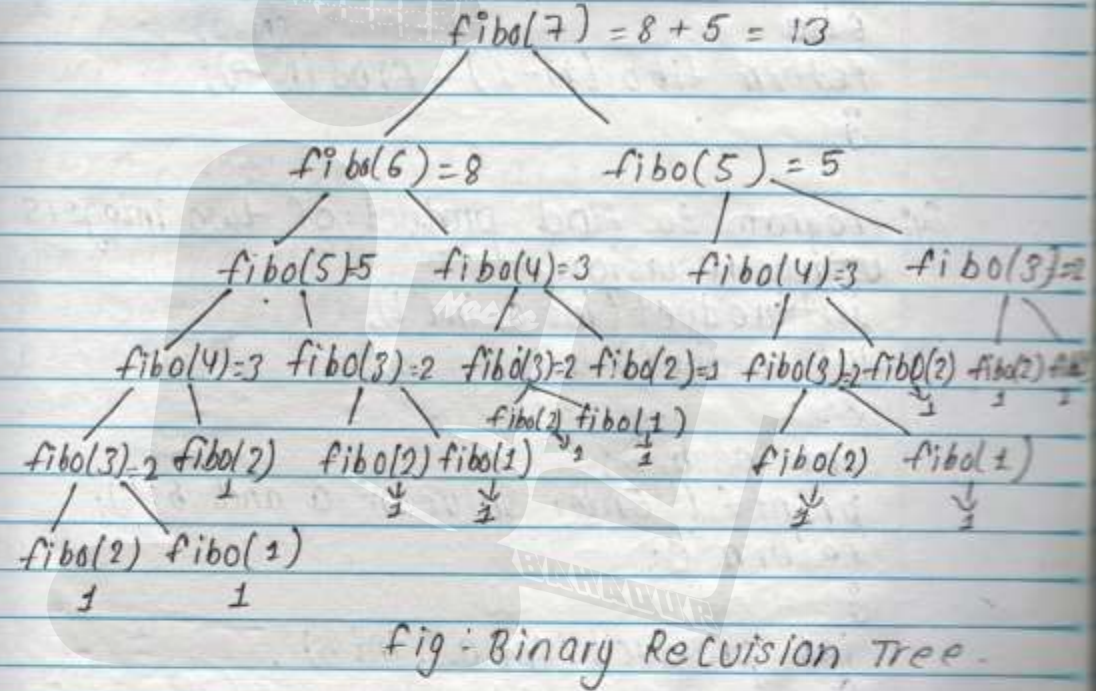
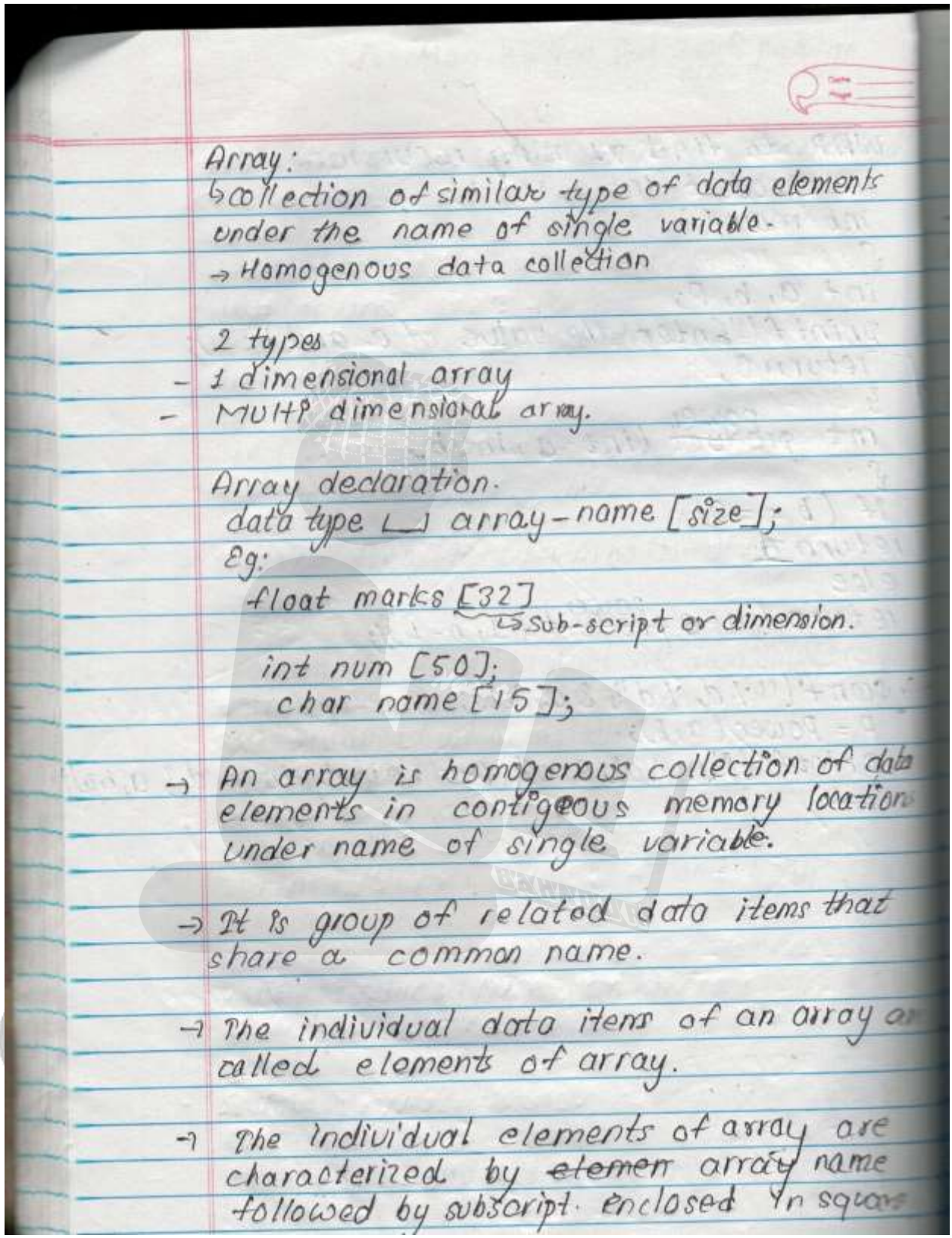


fig: Binary Recursion Tree.

```
int main()
{
 int N, i;
 printf("Enter value of N:");
 scanf("%d", &N);
```



Array:
↳ collection of similar type of data elements under the name of single variable.
→ Homogenous data collection

2 types
- 1 dimensional array
- Multi dimensional array.

Array declaration.
data type ⌴ array-name [size];
Eg:
float marks [32]
↳ sub-script or dimension.

int num [50];
char name [15];

→ An array is homogenous collection of data elements in contigeous memory location under name of single variable.

→ It is group of related data items that share a common name.

→ The individual data items of an array are called elements of array.

→ The individual elements of array are characterized by ~~elemen~~ array name followed by subscript enclosed in squa

bracket.

→ Characteristic of array.

1. All the elements of array share the same name and distinguish from one another with the help of index number.

2. Memory for array elements is allocated at program at array decleration time rather than run time.

3. Once the memory size is fixed, it cannot grow or ~~srin~~ shrink during program execution.

4. Array elements are always stored in contigeou s array locations.

Types

1. One dimensional array
An array name followed by only one subscri- ption(dimension) is called one dimensional array.

- In 1 dimensional array, the array index starts at 0 and end at size-1

Decleration of 1-D array.

Syntax: data type [ ] array-name [size],

Eg:

```
int num [5];
```

num: [num[0] | num[1] | num[2] | num[3] | num[4]] → (each occupy 2-byte)

0 1 2 3 4

1000 1002 1004 1006 1008

Note:- Array size must be numeric constant or symbolic constant but never be a variable
for eg:
(i) int a[50]; // valid.
(ii) # define size 10
int a[size]; //valid
(iii) int n
printf("Enter n:");
scanf("%d", &n);
int num[n]; //Invalid.

@ Initializing one dimensional array.
Syntax:
data type ~~array~~ array name [size] = {val1, val2, ... valn}

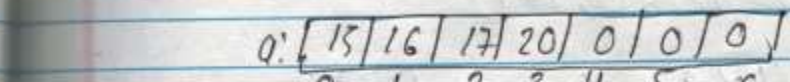
eg:
int a[5] = {46, 72, 91, 60, 62};

a: | 46 | 72 | 91 | 60 | 62 |
0 1 2 3 4

- Assigning ~~indi~~ specific value to the individual value at the time of array decleration is known as array initialization.

- In an uninitialized array, the individual array elements content garbage value.
i.e
int a[4];

Eg: int a[7] = {15, 16, 17, 20};



WAP to create an array of 10 integers and display the array element.

soln

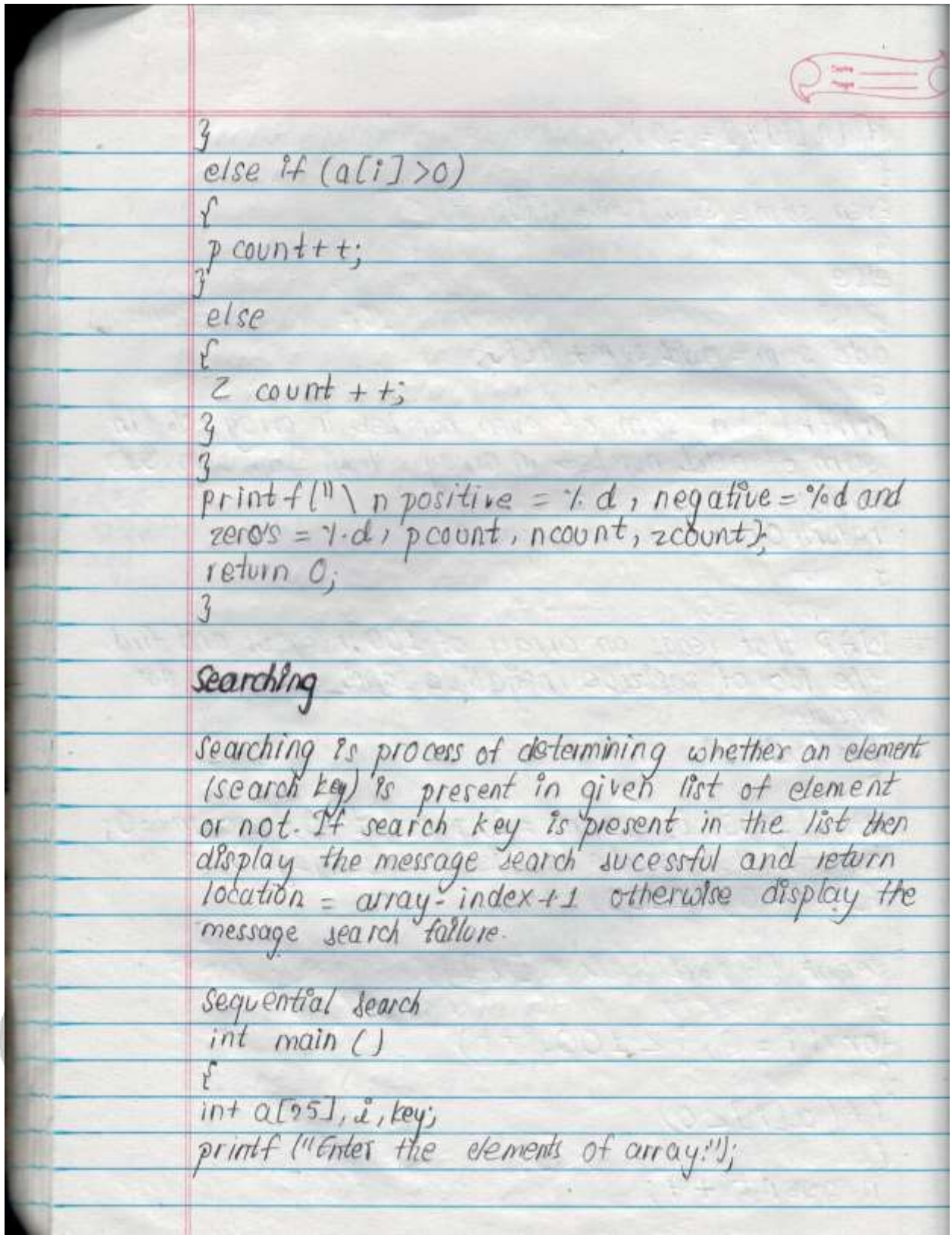
```
int main ()
{
int a[10] = { 54, 67, 1, 35, 64, 18, 25, 17, 5};
int i;
printf("Array elements are:\n");
for (i=0; i<10; i++)
  {
  printf("a[%d] = %d\t", i, a[i]);
  }
return 0
}
```

Output: Array elements are

a[0]=54 a[1]=67 a[2]=1 ... a[9]=0

WAP that reads marks obtained by 50 students in a class and find the average marks.

```
int main ()
{
float marks[50], total=0, Avg;
int i;
```



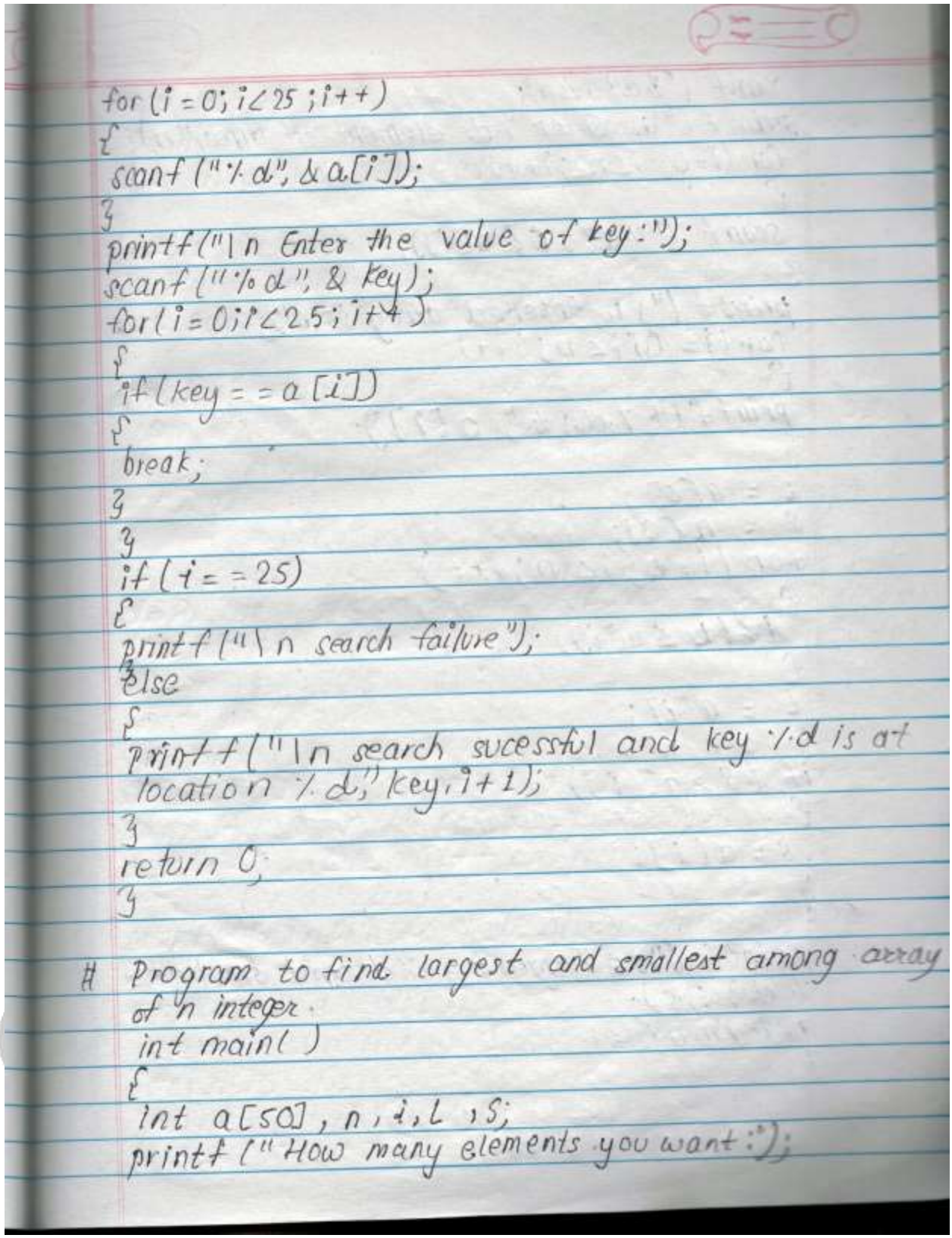
```
}
else if (a[i]>0)
{
 p count++;
}
 else
{
 z count ++;
}
}
printf("\n positive = %d, negative = %d and
 zero's = %d, pcount, ncount, zcount);
return 0;
}
```

Searching

Searching is process of determining whether an element (search key) is present in given list of element or not. If search key is present in the list then display the message search sucessful and return location = array-index+1 otherwise display the message search failure.

Sequential Search

```
int main ()
{
int a[95], i, key;
printf ("Enter the elements of array:");
```



```
for (i = 0; i < 25 ; i++)
{
 scanf ("%d", &a[i]);
}
printf("\n Enter the value of key:");
scanf ("%d", & key);
for(i = 0;i<25; i++)
{
 if(key = = a [i])
{
 break;
}
}
if (i = = 25)
{
 printf ("\n search failure");
}else
{
 printf ("\n search sucessful and key %d is at
 location %d", key, i+1);
}
return 0;
}
```

Program to find largest and smallest among array of n integer.

```
 int main( )
 {
 int a[50], n, i, L , S;
 printf (" How many elements you want:");
```

Sorting

Sorting is the process of ~~m~~ arranging elements of list in ascending or descending order

Bubble sort:

In this ~~shorting~~ sorting, we always compare the adjacent elements of list.

(a) [25 | 18 | 57 | 31 | 10]
0 1 2 3 4

Pass 1 i = 0, j = 0

25 18 57 31 10 ∵ 25 > 18 (swap)
j j+1

j = 1, 18 25 57 31 10 ∵ 25 < 57 (no swap)
j = 2, 18 25 57 31 10 ∵ 57 > 31 (swap)
j = 3, 18 25 31 57 10 ∵ 57 > 10 (swap)
18 25 31 10 57

```
int main()
{
int a[50], n, i, j, temp;
printf("Enter value of n: ");
scanf("%d", &n);
printf("\n Enter %d array elements", n);
for(i=0; i<n; i++)
scanf("%d", a[i]);
printf("\n Array before sorting \n");
for(i=0; i<n; i++)
printf("%d \t", a[i]);
```

```
for(i=0; i<(n-1); i++)
{
for(j=0; j<(n-i-1); j++)
{
if(a[j]>a[j+1])
{
temp = a[j];
a[j] = a[j+1];
a[j+1] = temp;
}
}
}
printf("\n After sorting \n");
for(i=0; i<n; i++)
printf("%d \t", a[i]);
return 0;
}
```

Selection sort

In this sorting first element of list is compared with all ~~sort~~ remaining elements, smallest one is selected and placed at first position of list and so on.

a: [47 | 11 | 54 | 23 | 5]
0 1 2 3 4

Pass 1: i = 0, j = 1

47 11 54 23 5
11 47 54 23 5 no change
11 47 54 23 5 no change

```
void simpleinterest (float P, float T, float R)
{
float SI;
 SI = P*T*R/100;
printf("simple interest= %f", SI);
}
```

Multi Dimensional Array

An array name followed by more than one dime
is called multidimensional array.

2-D array

↳ An array name followed by two subscripts (d
is called 2-D array. The first dimension repres
row-size and second subscript represent
col-size.

↳ 2-D array is also called matrix.

Declaration of 2-D array

Syntax:

data-type [ ][ ] array-name[Row-size][col-s

Eg: int a [3] [3]

$$a: \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

a: [a00 | a01 | a02 | a10 | a11 | a12 | a20 | a21 | a2

2-D array is an array of 1-D

gm → graphics mode

Graphics → 5 marks.

The program initializes graphics mode and then closes it after a key is pressed. To begin with we have declared two variable of int type gd and gm for graphics mode respectively. We can choose any other variable names as well. DETECT is a macro defined in "graphics.h" header file, then we have passed three ~~eg~~ arguments to initgraph functions first is the address of gd, second is the address of gm and third is the path where your BGI files are present. Initgraph function automatically decides an appropriate graphics driver mode such that the maximum resolution is set, getch helps us to wait until a key is pressed, closegraph function closes graphic mode.

Graphics in C

In C, display can be normally used in either text mode or graphic mode. In text mode, the screen is divided into 80 columns and 25 rows. In case of graphics mode, the screen is divided int 640 pixles width and 480 pixels height latest displays are even rich in resolution.

Graphically, text display is faster than graphic display because in graphic mode, we have to program by pixel.

Graphics Initialization in C/S.

In a C program, first step is to initialize the graphics drivers in the computer. This is done using the initgraph() method provided in graphics.lib library. The initgraph function has the following parameter.

void initgraph (int *graphdriver, int *graphmode, char *pathdriver);

The method initgraph() initializes the graphic system by loading driver from disk, then put the system into graphic mode. The method initgraph() also resets all graphic setting (color, palette, current position, viewport etc) to their defaults. The initialization result is set to 0 which can be retrived by calling graph result().

Graphics in C

* graphdriver:

This is an integer values that specifies the graphics driver to be used. A graphdriver is given a value using a constant of the graphics driver enumeration type which is listed in graphics.h, normally use value as 0 "Example auto detect". The graphic driver enumeration type defined in graphic.h are

Graphics hardware initialization.

listed below.

graphic driver control	Numeric value
DETECT	0 (requests auto-detect)
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

Graph mode:

This is an integer value that specifies the initial graphics mode (unless * graph driver = DETECT). If * graph driver = DETECT then initgraph() method sets * graph mode to the higher resolution available for the DETECT graph driver. We can give * graph mode a value using a constant of a graphics mode enumeration type ~~* path to driver~~.

* path to driver:

. It specifies the directory path where initgraph() looks for a graph driver (*.BGI) first. If they are not there, initgraph

() method looks in the current ~~the~~ library. If the pathdriver is null, the driver files must be in the current directo

Circle

Decleration : void circle (int x, int y, int radius)

Circle function is used to draw a circle with center (x,y) and third parameter specifies the radius of the circle. The code gives below draws a circle.

```
# include <graphics.h>
# include <conio.h>
void main()
{
 int gd =DETECT, gm;
 initgraph (&gd, &gm, "C:\\TC\\BGI");
 circle (100, 100, 50);
 getch();
 closegraph();
}
```

String:
↳String is an array of characters.
↳ A string is always terminated by a special character called NULL character (\0)

Declaring a string
Syntax:
data-type name [SIZE]; //Here data-type is character only.

Eg:
```
Char name[10];
Char Post[15];
char address[15];
```

Initializing a string variable.
Eg:
```
Char name[10]={'R','A','E','S','H','\0'};
Char name[10]='Raesh';
```

#Reading and displaying string.
```
#include<stdio.h>
int main()
{
char name[15];
printf("Enter the name:");
scanf("%s",name);
printf("\n you name is: %s",name);
return 0;
}
```

→read →display.
gets( ) and puts( ) function] → Only for words or string.

gets ( ):
Syntax:
gets (string-name);

```
#include <stdio.h>
int main ()
{
 char Post [10];
 printf ("Enter your post:");
 gets (post);
 printf ("\n you post is:");
 puts(post);
 return 0;
}
```

Compairision of scanf( ) and gets( )
scanf में blank space allow नहीं,
gets allow blank and white space.

Ramesh ␣ singh.

String Handling Functions (library function)
① strlen( ):-
This function is accept a string as input and find the length of input string. The length of string is defined as number of character in that string excluding null character. (library function)
Note: string handling function are defined in header file. <string.h>

Syntax:-
integer_variable = strlen(string);
Eg: WAP to read a string and find it's length.

```
#~~induced~~
#include<stdio.h>
#include<string.h>
int main()
{
 char str[50];
 int l;
 printf("Enter the input string:");
 gets(str);
 l=strlen(str);
 printf("\n length of string=%d",l);
 return 0;
}
```

strcpy():- This function is used to copy the content of one string into another string.

Syntax:-
strcpy(string1, string2)
Eg: WAP to read a string and copy it's content to another string.

```
#include<stdio.h>
int main()
{
 char source[30], destination[30];
 printf("Enter the input string:");
 gets(source);
```

```
strcpy(destination ,source);
printf("\n The copied string is:");
puts(destination);
return 0;
}
```

③ strrev():- This function accept a string as input & return the reverse of input string.

syntax: strrev(input-string);

Eg: WAP to read a string as input and find its reverse.

```
#include <stdio.h>
#include <string.h>
int main()
{
char S1[20], S2[20];
printf("Enter the input string:");
gets(S1);
strcpy(S2, S1);
strrev(S1);
printf("\n reverse of string %s is
        %s", S2, S1);
return 0;
}
```

④ strcat():- This function is used to concatinate two strings.

जोड्नु

This function accept two string as input and append (थप्नु) the content of second string at the end of first string.

Syntax:

```
strcat (S1 S2);
```

Note: This function insert content S_2 at end of S_1 and store the result in S_1.

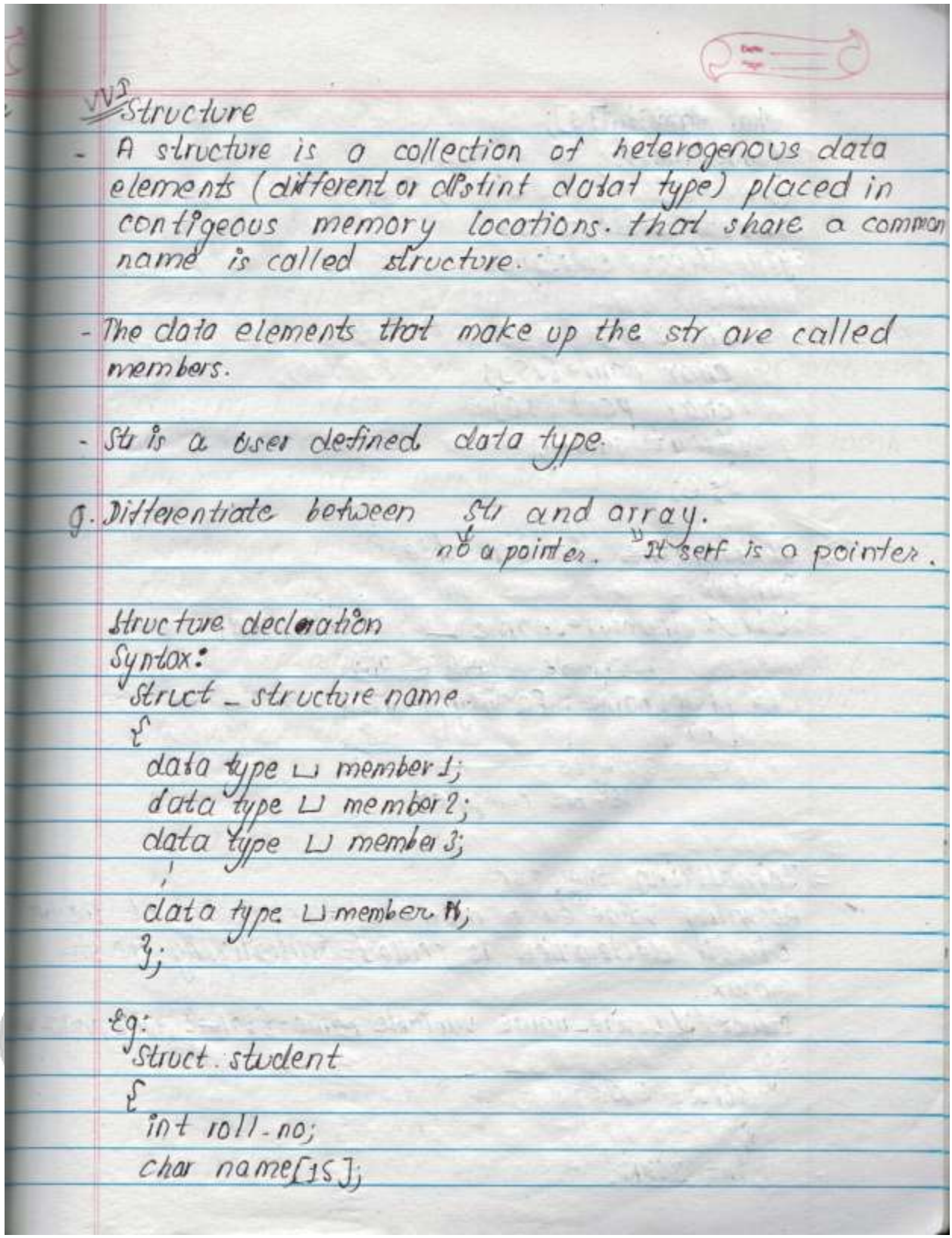
→ WAP that reads two strings as input and concatenate them.

```
#include <stdio.h>
#include <string.h>
int main()
{
Char S1[50], S2[20];
printf ("Enter the string S1 :");
gets (S1);
printf ("\n Enter the string S2 :");
gets (S2);
strcat (S1, S2);
printf ("\n The concatenated string is:");
        puts (S1);
return 0;
}
```

strcmp( ):- string comparision.
The function strcmp( ) accept two strings as input and compare them. Function returns ~~zero~~ (0), if both strings are same, less than zero(<0) if first string is smaller then second string in terms of ASCII value and greater than zero if second string is larger than first string.

Syntax:-
integer_variable = strcmp(s1, s2);
Eg: WAP to read two strings as input and compare them.

```
#include <stdio.h>
#include <string.h>
int main( )
{
 char s1[15], s2[15];
 int d;
 printf("Enter the first string:");
 gets(s1);
 printf("Enter the second string:");
 gets(s2);
 d = strcmp(s1, s2);
 if(d <0)
 {
  printf("\n string %s is greater T
        %s", s2, s1);
}
```



VVI Structure

- A structure is a collection of heterogenous data elements (different or distint datat type) placed in contigeous memory locations. that share a common name is called structure.

- The data elements that make up the str are called members.

- Str is a user defined data type.

Q. Differentiate between str and array.

not a pointer.	itself is a pointer.

Structure declaration

Syntax:

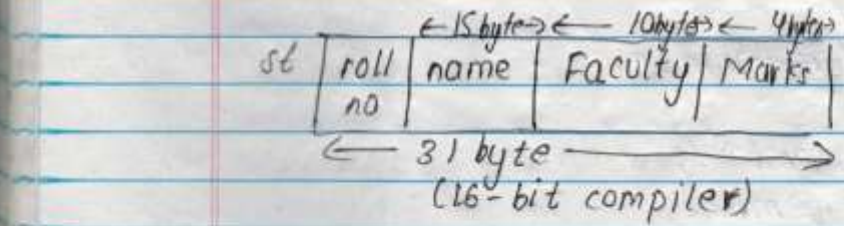
```
struct _ structure name
{
  data type ␣ member1;
  data type ␣ member2;
  data type ␣ member3;
  ,
  data type ␣ member n;
};
```

Eg:

```
struct student
{
  int roll_no;
  char name[15];
```

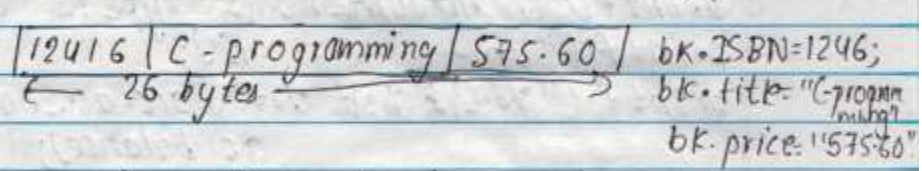
char faculty[10];
float marks;
}; or, }.st;
struct student st;
Eg:# struct employee
{
int ID;
char name[15];
char post[10];
float salary;
}; e;

⇒ Defining a ~~the~~ str variable.
Syntax:-
Strud structure_name ␣ variable name;



⇒ Initializing the str.
Assigning value to each members of structure at the ~~to~~ of str. decleration is called initializing the str.
Syntax:
Struct structure_name variable name={value1, val 2, val
Eg:
struct Book.
{
int ISBN;

```
    char title[20];
    float price;
};
struct book bk = {12416, "C-programming", 575.60};
```



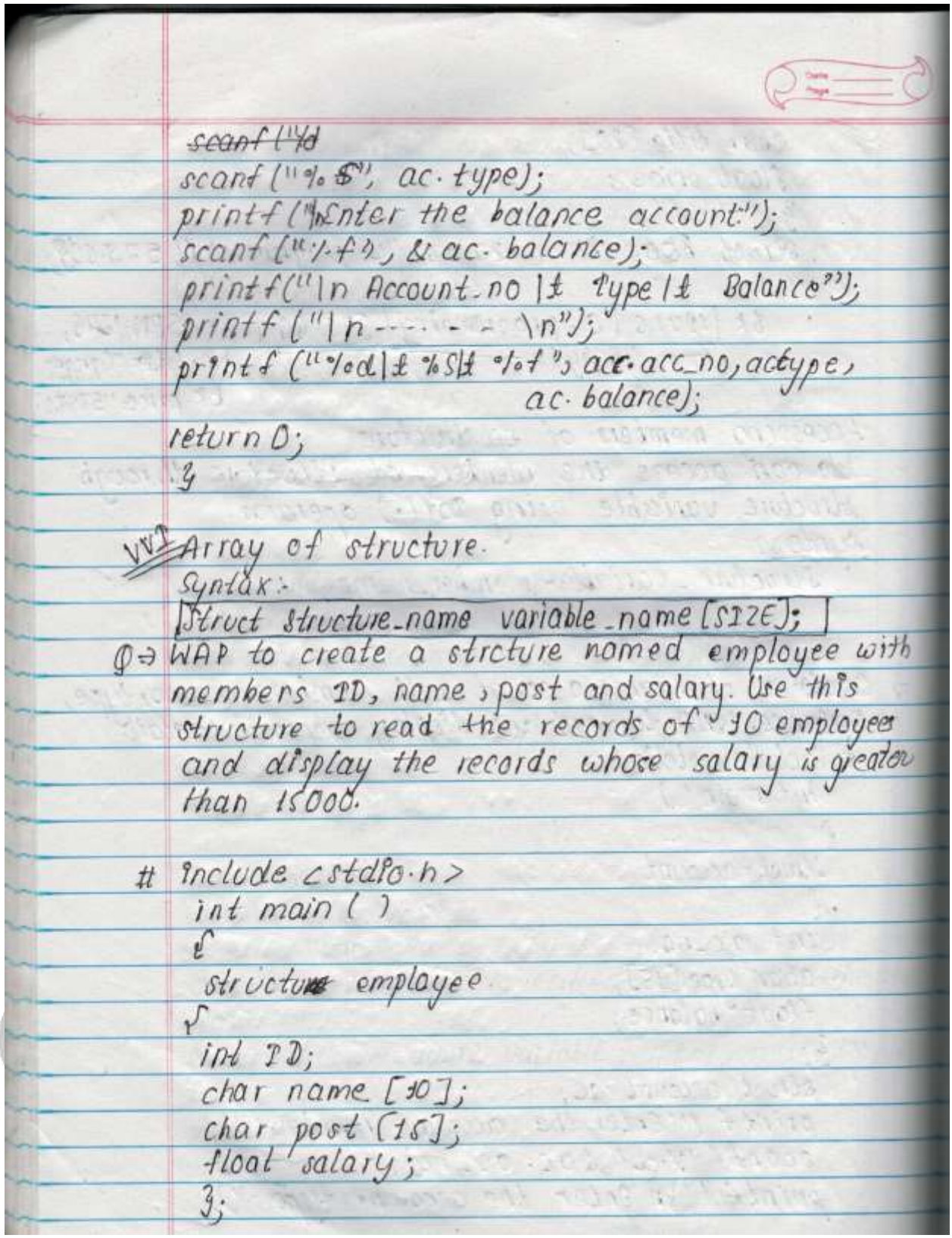
Accessing members of ~~a~~ structure.
We can access the members of structure through structure variable using Dot(.) operator.

Syntax:

```
structure_variable.member_name
```

⇒ Create a str. name account with members acc_no, type, Balance. WAP to read and display account detail.

```
#include<stdio.h>
int main()
{
 struct account
 {
  int acc_no;
  char type[15];
  float balance;
 };
 struct account ac;
 printf("Enter the account number:");
 scanf("%d", &ac.ac_no);
 printf("\n Enter the account type:");
```



```
scanf("%d
scanf("%s", ac.type);
printf("\nEnter the balance account:");
scanf("%f", &ac.balance);
printf("\n Account_no \t Type \t Balance");
printf("\n---------\n");
printf("%d\t %s\t %f", acc.acc_no, actype,
                        ac.balance);
return 0;
}
```

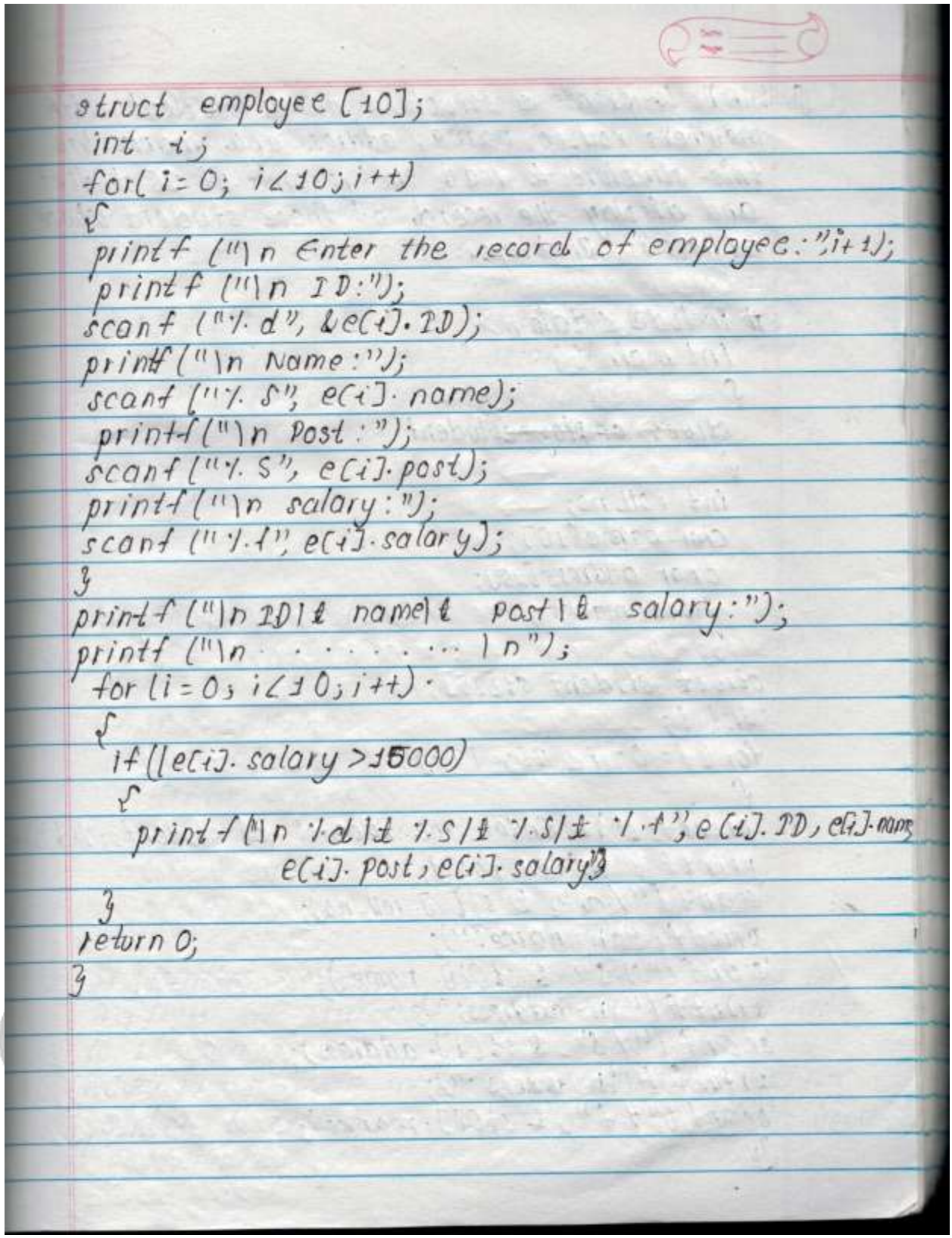
VVI Array of structure.

Syntax:

Struct structure_name variable_name [SIZE];

Q⇒ WAP to create a strcture named employee with members ID, name, post and salary. Use this structure to read the records of 10 employee and display the records whose salary is greater than 15000.

```
# include <stdio.h>
 int main ( )
 {
  structure employee
 {
  int ID;
  char name [10];
  char post [15];
  float salary;
  };
```



```
struct employee [10];
 int i;
for( i=0; i<10;i++)
{
 printf ("\n Enter the record of employee:",i+1);
 printf ("\n ID:");
scanf ("%d", &e[i].ID);
printf("\n Name:");
scanf ("%s", e[i].name);
 printf("\n Post:");
scanf("%s", e[i].post);
printf("\n salary:");
scanf ("%f", e[i].salary);
}
printf ("\n ID\t name\t post\t salary:");
printf ("\n ..... \n");
 for (i=0; i<10;i++)
 {
  if((e[i].salary >15000)
   {
    printf("\n %d\t %s\t %s\t %f", e[i].ID, e[i].name,
            e[i].post, e[i].salary);
 }
return 0;
}
```

```
printf("\n roll_no\t  name\t  address\t marks");
printf("\n ..... \n");
for(i=0; i<25 ; i++)
{
if(strcmp(st[i].address,"Bhaktapur")==0)
 {
 printf("\n %d\t %s\t %s\t %f", st[i].roll_no,
                                  st[i].name,
      st[i].post, st[i].salary);
      e[i].address, e[i].marks);
}
}
return 0;
}
```

Pointer to structure

```
struct student
  {
    int roll_no;
    char name[10];
    float marks;
  };
struct student st, *p;
  p = &st;
```

↳ A pointer of type structure that stores the address of structure variable is called pointer to structure.

Note:- To access the members of structure using pointer to structure, arrow operator [→] is used.

st.roll no.	p → roll_no	(*p).roll_no
st.name	p → name	(*p).name
st.mark	p → marks	(*p).marks.

Nested Structure

Employee
- → ID
- → name
- → Post
- → DOB
  - → Year
  - → Month
  - → Day
- → Salary

- If a structure is defined as member of another structure than that is called nested structure.

```
#include<stdio.h>
  struct employee
  {
   int ID;
   char name[15];
   Char post[10];
   Struct DOB
   {
    int year, month, day;
   }d;
   float salary;
  }emp;
```

OR

```
struct DOB
{
int year;
int month;
int day;
};
struct employee
{
int ID;
char name[10];
char post[10];
float salary;
struct DOB d;
```

```
printf("\n enter the year of DOB : ");
scanf("%d", &emp.d.year);
```

Union

Difference between structure and union

1. Struct student { int roll_no; char name[15]; char faculty[10]; float marks; }st	1. Union student { int roll_no; char name[
2. St. variable occupy the size of memory equal to the sum of memory size of each ~~variable~~ member.	2. Union variable occupy the memory size of largest ~~variable~~ members.
3. Sperate memory space for all members	3. All the members of union share common address space.
4. Access at same time.	4. ~~Donot access~~ Only one member access at any given time.
5. Take more memory	5. less.

Passing structure to the function.

Methods.

1. Passing each members of structure individually to the function.
2. Passing whole structure to the function (Passing structure variable)
3. Passing structure pointer to the function (passing address of structure variable).
4. Passing array of structure to the function.

(File handling) Data File

Opening and closing file

```
FILE *file-pointer;
```

Eg:

```
FILE *fp;
file_pointer = fopen("file_name", "file-mode");
```

File opening modes.

1. read mode ("r")
2. write mode ("w")
3. Append mode ("a") (existing file को last मा थपने)
4. read and write ("r+")
5. write and read ("w+")
6. Append and read ("a+")

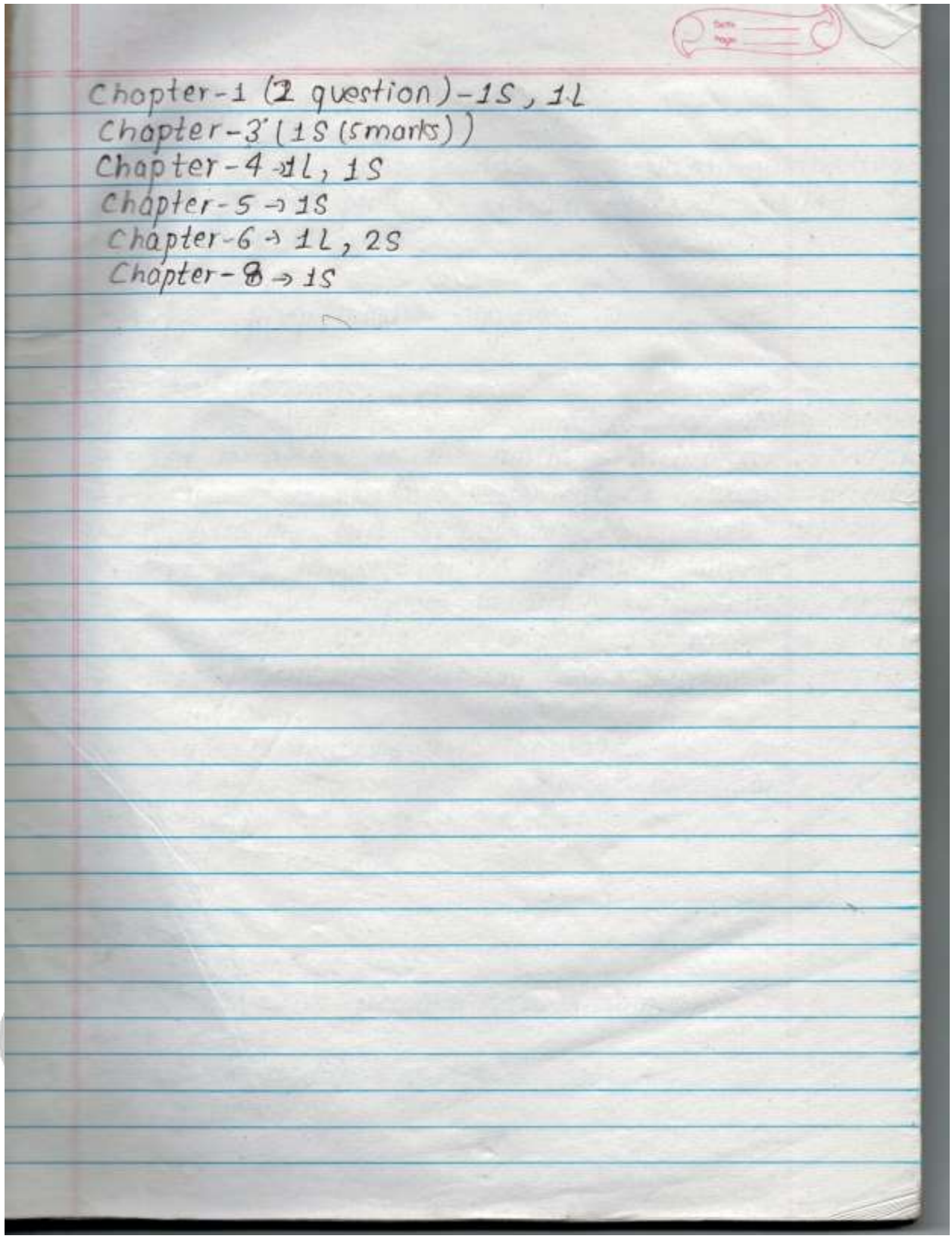
Eg:

```
FILE *fp;
fp = ("C:\\abc.txt", "r");
```

Closing a file

```
fclose(file-pointer);
```

Eg: `fclose(fp);`



Chapter-1 (2 question)-1S, 1L
Chapter-3 (1S (5marks))
Chapter-4 → 1L, 1S
Chapter-5 → 1S
Chapter-6 → 1L, 2S
Chapter-8 → 1S