12/7/22

CHAPTER-1 -(Variable's, Keywords, constants)

Compile → write gcc file name in terminal which create's a executable file name a.exe → executable file

To run → write ".\a.exe" or first write a.exe and press tab in the terminal

Variables

* Name of a memory location which stores some data.

Example:- Memory

here 25 is some data and a is the name of memory location of that data.

Rules for naming a variable

- Variables are case sensitive (ABHI and abhi are different)
- 1st character is either alphabet or '_' → underscore
- no comma , blank space should be involved.
- No other symbol apart from '_'
- The data stored in variable can be changed.

Data types

Data type	Size in bytes
char or signed char	1
unsigned char	1
int or signed int	2 or 4
unsigned int	2 or 4
short int or unsigned short int	2
signed short int	2
long int or signed long int	8 or (4 bytes for 32-bit os) → gcc
~~using~~ unsigned long int	8 bytes → 4 → gcc
float	4
double	8
long double	8 / 12 → gcc

NOTE → • Boolean and string are not data types

Constants :

Value's that dont change (fixed)

- Types

Integer constants
1, 2, 3, 0, -1, -2

Real constants
1.0, 2.0, 3.14, -2.14

character constants
'a', 'b', 'A', '#', '8'.

Characters constants should be written in single quotes → 6 9

Keywords :

Reserved words that have special meaning to the compiler

↓

32 keywords in C

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

Program structure

```
#include <stdio.h>      → pre processive direct *
int main() {            → The execution always starts from main *
return 0;               → here 0 indicate zero errors
}                       → statement terminator
```

Comments

lines that are not part of program.

- Single line //
- multiple line /* */

Output

- printf ("Hello world");

new line

printf ("Hello world \n");

cases

1. integers.
   printf ("age is %od", age);
2. real numbers
   %of
3. characters
   %c

%d, %f, %c are Format Specifiers

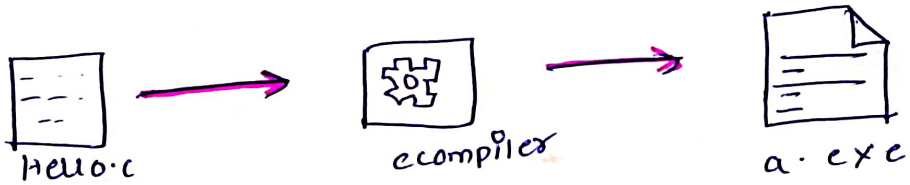
Input

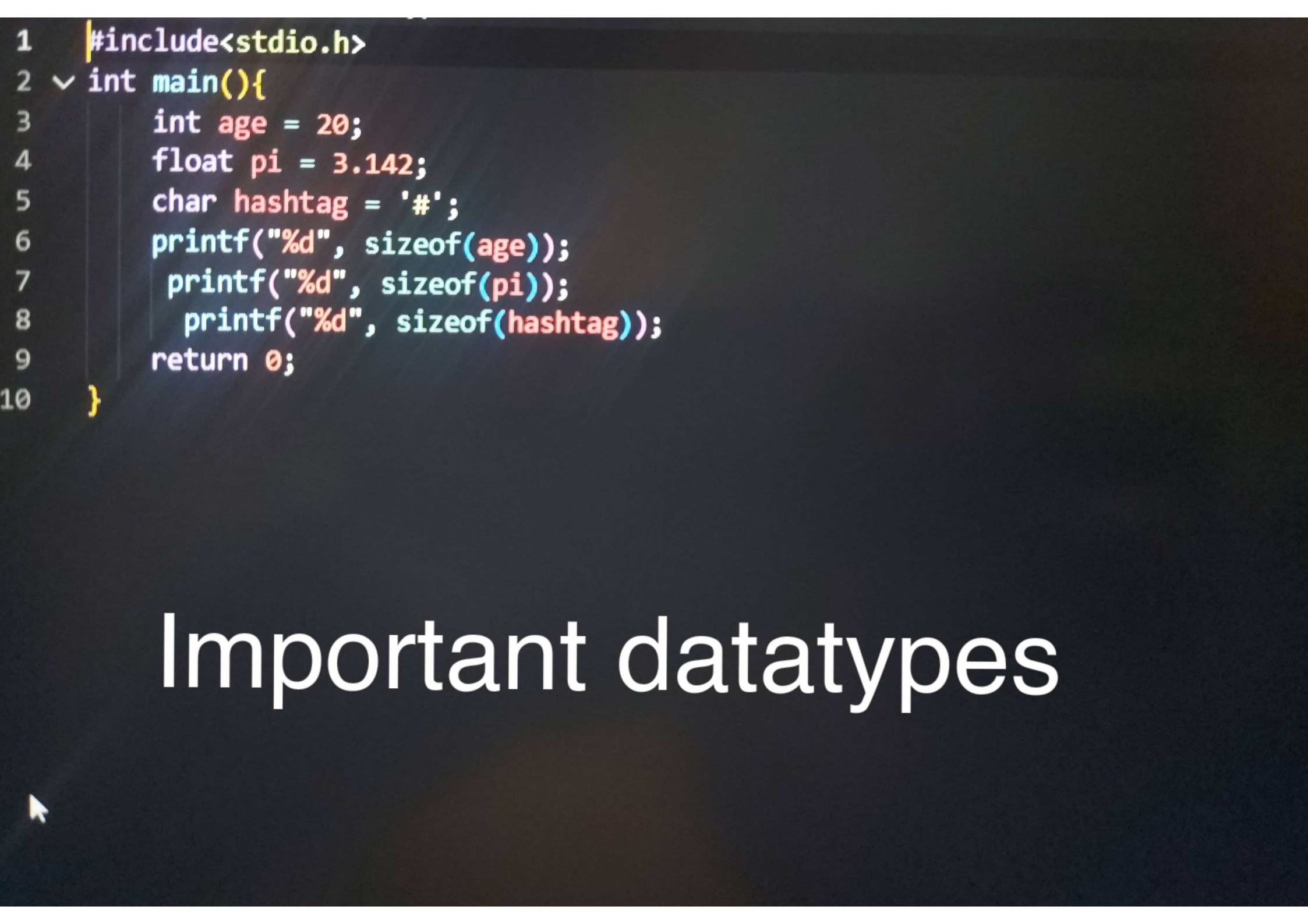
scanf ("%d", &age);

- &age → address of
- %d → int format specifier

Compilation

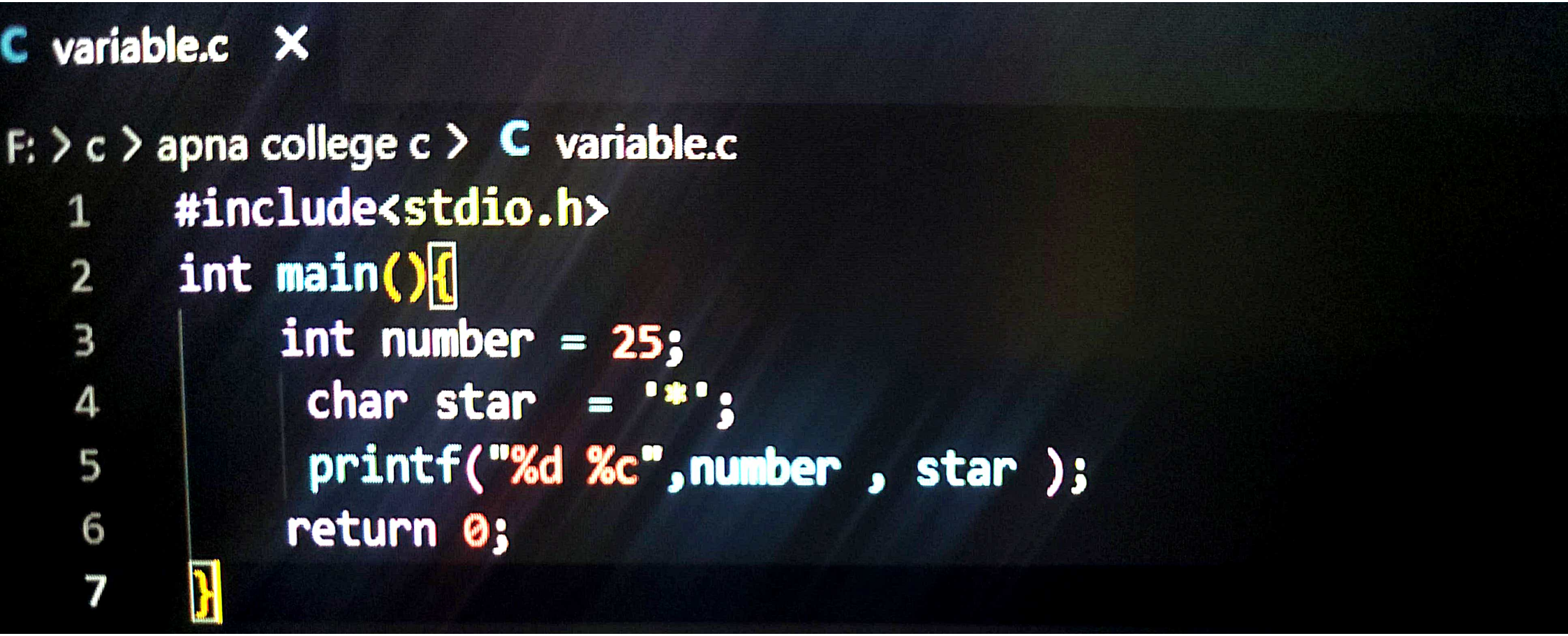
A computer program that translates c code into machine code




#include<stdio.h>
int main(){
int age = 20;
float pi = 3.142;
char hashtag = '#';
printf("%d", sizeof(age));
printf("%d", sizeof(pi));
printf("%d", sizeof(hashtag));
return 0;
}
Important datatypes

```c
#include<stdio.h>
int main(){
    int age;
    printf("enter your age ");
    scanf("%d",&age);
    printf("%d",age);
    return 0;
}
```

Input from a user and print


variable.c
F: > c > apna college c > variable.c
#include<stdio.h>
int main(){
int number = 25;
char star = '*';
printf("%d %c",number , star);
return 0;
}

12/7/22

CHAPTER-2

Instructions and operators

Instructions

These are statements in a program.

Types

- Type declaration instructions
- Arithmetic instructions
- control instructions.

1) Type declaration instructions : Declare variable before using it
ex: int var

NOTE
- VALID
  - int a, b, c ;
    a = b = c = 1 ;
  - int a = 22 ;
  - ~~int b = a ;~~
  - int c = b + 1 ;
  - int d = 1 , e ;
- IN VALID
  - int a, b, c = 1 ;
  - int b = a ;
    int a = 1 ;

2) Arithmetic Instructions

NOTE : single variable on LHS

example: LHS VAR = RHS a + b - c * d

Arguements will be passed from ~~left~~ right to ~~right~~ left

Valid

a = b + c
a = b * c
a = b / c

Invalid

b + c = a
a = bc
a = b ^ c

NOTE - pow(x,y) for x to the power y
use #include<math.h> preprocessor directive.

Types

+ → addition operator
− → substraction
* → multiplication
/ → division

% → modular operator
ex:- 3%2 = 1

* modular operator doesnt work on float → Invalid operand
* If numerator is negative the output will be negative

Type conversion

- int operation int → int
- int operation float → Float
- float operation float → float

ex:- 2x2 = 4
2.0x4 = 8.0
2.0x2.0 = 4.0

NOTE :- 2/3 will not be some decimal value.
it will be zero
2.0/3 = Decimal number

Conversion are of Two types

1) Implicit → compiler will do the conversion on its own
Integer value will be stored in float, double
double cannot be stored but in integer we have to do it by explicit
2) explicit → The user has to convert it.

ex:- int a = 1.9999 → error. (Implicit).

int a = (int) 1.9999 → 1 (explicit conversion)
all decimal will be removed
it will not round off.

* Operator precedence

priority ~~oder~~ order of operators is called as operator precedence.

1st priority — *, /, %

↓

2nd priority — +, −

↓

3rd priority — = (assignment operator).

example: $X = 4+9*10$

1st priority

2nd priority

$= 4+90$

$X = 94$

$X = 4*3/6*2$

?

- In case of same ~~Assoc~~ precedence operator (associativity) comes into picture

Left to right.

i.e $X = 4*3/6*2.$

1 2 3

$12/6 * 2.$

$2*2$

$X = 4$

- In case of paranthesis it will be solved first.

example $5*(2/2)*3$

1 2 3

3) Control Instructions.

used to determine flow of program.

a. Sequence control :- The program statements (Instructions) has sequential flow i.e one by ~~after~~ one

b. Decision control :- if - else

c. loop control :- while, do while, for.

d. case control :- switch case

Operators

a. Arthimetic operators → +, −, /, *, %

b. Relational operators → * ==

example

a == b means is a and b equal ?
a = b means the value of b will be stored in a

4 == 4 = 1 → means True (any non zero number means True)
4 == 3 = 0 → False

* > → ~~equal to~~ greater than
* >= → greater than equal to
* <= → less than equal to
* < → less than.
* != → Not equal to

C. logical operator

* && → And

False True = False
True False = False
True True = True
False False = False

* || → OR

False False = False , True False = True , True True = True

* ! NOT

True = False
False = True

Operator precedence

Priority	operator
1	!
2	*, /, %
3	+, -
4	<, <=, >, >=
5	==, !=
6	&&
7	||
8	=

a == 5

a = 5

D. Assignment operator (short hand operators).

- =
- += which means a = a+b.] same.
  a += b
- *= a = a*b] same.
  a *= b
- -=
- /=
- %=

Important notes

a. int a = 8^8 → valid.
↓ bitwise XOR operator.

b. int x; int y = x; → valid

c. int x, y = x; → Not valid because we can't declare and use variable in the same line.

d. char stars = '**'; ⇒ In valid because char we cannot create multi character constant we should use only single value.

↓

why not valid?

because it cannot fit in 1 byte character memory.

E. Bitwise operators

F. conditional operator → Ternary operator

```c
#include <stdio.h>
int main()
{
    int a;
    int b;
    printf("enter the value of a ");
    scanf("%d",&a);
    printf("enter the value of b ");
    scanf("%d",&b);

    printf("the sum of a and b is %d", a + b);

    return 0;
}
```

```c
#include<stdio.h>

int main(){
    float pi = 3.142;
    float radius;
    printf("enter the radius of the circle-->");
    scanf("%f", &radius);
    printf("the area of circle is --->%f",pi*radius*radius);
}
```

```c
#include<stdio.h>
int main(){
    float length , breadth;
    printf("enter length in cm\t");
    scanf("%f", &length );
    printf("enter breadth in cm\t");
    scanf("%f", &breadth );
    printf("the area of rectangle is %f square cm " , length*breadth);

    return 0;
}
```

CHAPTER-3 Conditional Statements *

Types

- if - else
- switch.

1) If else (SYNTAX)

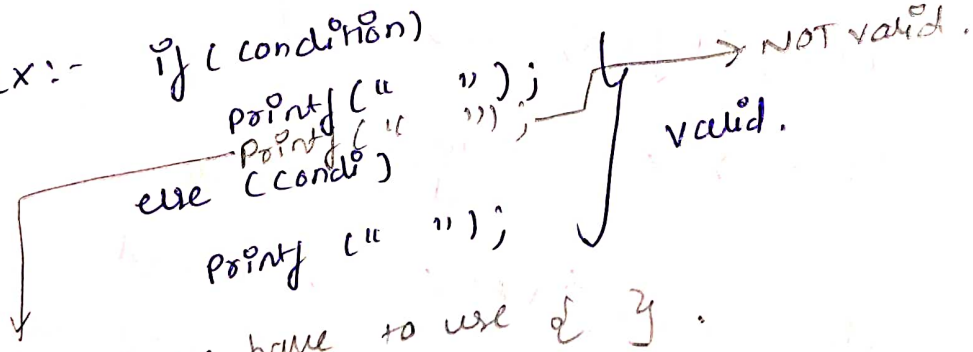
```
if (condition) {
    // code
}
else {
    // code
}
```

else is optional

we can not use { } if there is only one statement

ex:-

```
if (condition)
    printf (" ");
    Printf (" ");
else (condi)
    print (" ");
```



In this case we have to use { }.

- else if

```
if (condition 1) {
    // do something if True
}
else if (condition 2) {
    // do something if 1st is false and 2nd is True
}.
```

NOTE

if
else if
else if
else if

V/S

if
if
if

here if the above conditions is false then if will execute

example
if [False] x
else if [False] x
else if [True] x

If the first if is True then the below lines will not execute.

here irrespective of the above conditions all the decision control statements will execution

example
if [True] ;
if [False] ;
if [True] ;

It will not stop if above condition is True.

Conditional operator

ternary operator.

syntax

condition ? do something if True : do something if false.

example

```
#include<stdio.h>
int main() {
    int age;
    printf("enter age : ");
    scanf("%d", &age);
    age>=18 ? Printf("adult") : Printf("child");
}
```

Input → output
18 → adult
7 → child

NOTE :- Whenever an else block is not present, the conditional statement is known as the simple if statement.

2) SWITCH

Syntax :

```
Switch ( number) {
  case c1 : // dosomething
        break;
  case c2 :  // do something
        break;
  default :  // do something
  }
```

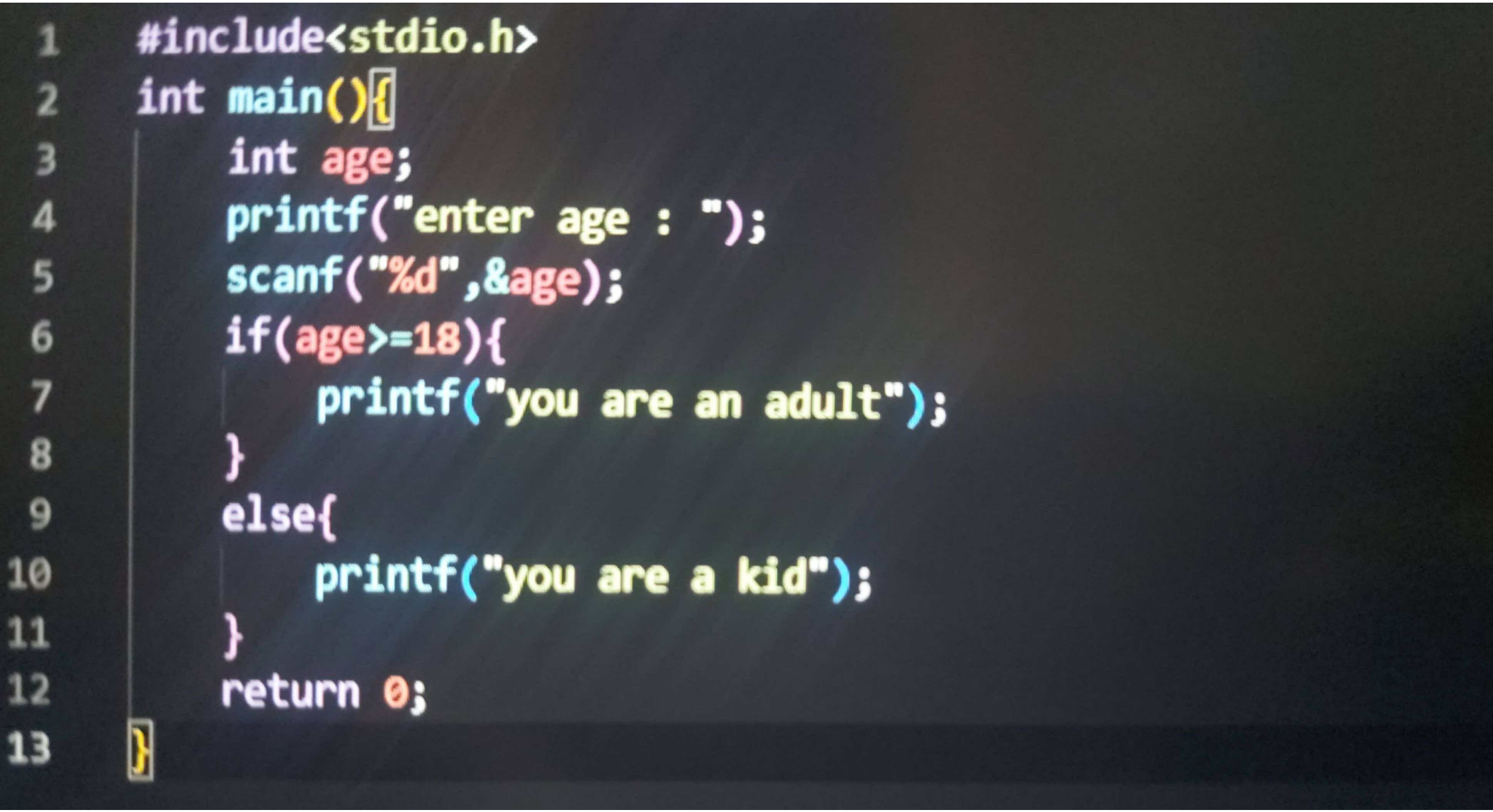
```
Switch (characters) {
  case 'a' : // do something   → Important
        break;
  case 'b' : // dosomething
        break;
  default :  // do something
  }
```

Properties of switch

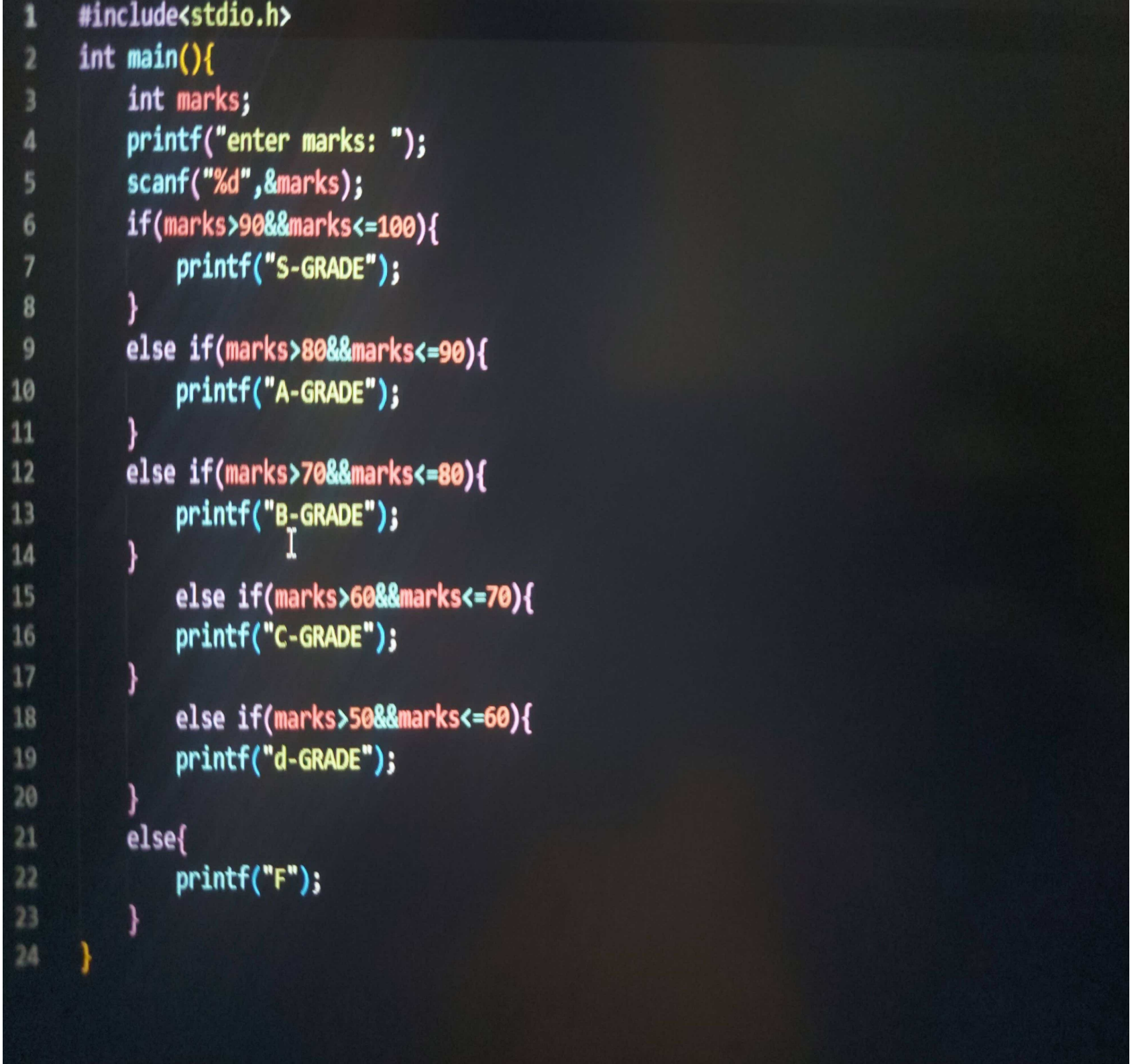
a. cases can be in any order.

ex :-
```
case 20 : code
    breale;
case 1 : code
    breale;
```

b. Nested switch (switch inside switch) are allowed.



```c
#include<stdio.h>
int main(){
    int age;
    printf("enter age : ");
    scanf("%d",&age);
    if(age>=18){
        printf("you are an adult");
    }
    else{
        printf("you are a kid");
    }
    return 0;
}
```



```c
#include<stdio.h>
int main(){
    int marks;
    printf("enter marks: ");
    scanf("%d",&marks);
    if(marks>90&&marks<=100){
        printf("S-GRADE");
    }
    else if(marks>80&&marks<=90){
        printf("A-GRADE");
    }
    else if(marks>70&&marks<=80){
        printf("B-GRADE");
    }
        else if(marks>60&&marks<=70){
        printf("C-GRADE");
    }
        else if(marks>50&&marks<=60){
        printf("d-GRADE");
    }
    else{
        printf("F");
    }
}
```

```c
#include<stdio.h>
int main(){
    int age;
    printf("enter age : ");
    scanf("%d",&age);
    age>=18?printf("adult"):printf("child");
}
```

```c
#include <stdio.h>
int main()
{
    int day;
    printf("enter the day (example 7 - SUNDAY) ");
    scanf("%d", &day);
    switch (day)
    {
    case 1:
        printf("MONDAY");
        break;
    case 2:
        printf("TUESDAY");
        break;
    case 3:
        printf("WEDNESDAY");
        break;
    case 4:
        printf("THURSDAY");
        break;
    case 5:
        printf("FRIDAY");
        break;
    case 6:
        printf("SATURDAY");
        break;
    case 7:
        printf("SUNDAY");
        break;
    default:
        printf("ENTER VALID DAY");
    }
    return 0;
}
```

IMPORTANT

1) Difference between declaring and defining a variable

Declaration of variable hints the compiler about type and size.
No space is reserved in memory for any variable in case of declaration
ex:- int a

Defining a variable means declaring a variable and also allocating space to hold it.

Defination = Declaration + space.
ex:- int a = 10

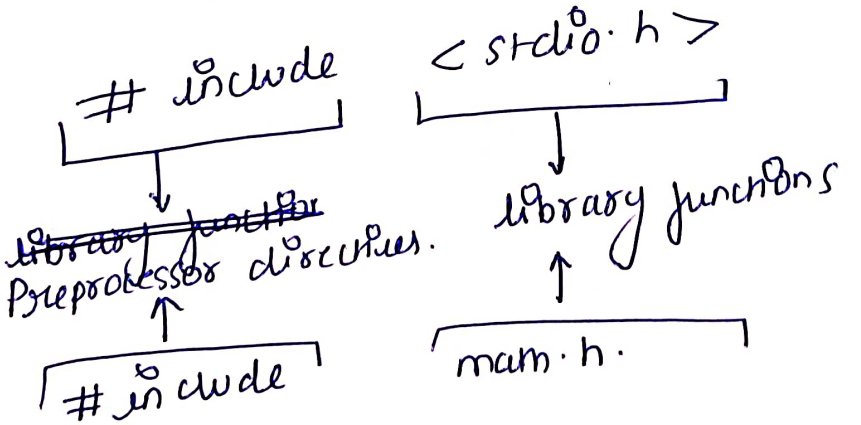
a is described as int to the compiler and also memory is allocated to hold value 10.

2) Pre processor directives

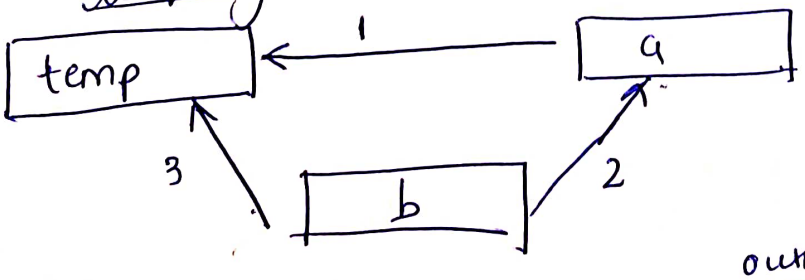
Library function on a file. such as <stdio.h> must be included in the begning to get functions like printf & scanf. such inclusions are made using statements called preprocessor directives.

Here #include is used as pre processor directive.

distinguishes them from other lines of Text.



3) Swapping technique.



a=5 b=6

1) temp = a = 5
temp = 5.

2) a = b = 6
a = 6

3) b = temp = 5
b = 5

output a=6 b=5

Format specifiers

- Char → %c
- Integer → %d (signed integer)
- Float → %f.
- signed Integer short → %hi
- unsigned short → %hu
- %i → signed integer
- long signed integer
  - → %li, %ld or %li
- %lf → double.
- %LF → long double.
- %lli / %lld → long long integer
- %llu → unsigned long long.

We can directly use %c for char

and also if (char >= 'A')

instead of using

if (char >= '97')

→ ASCII

Chapter 4 :- Loop control instructions

Loop control instructions → To repeat some parts of the program

Types

- For
- while
- do while.

For loop

```
For (initialization ; condition ; updation ) {
    // do something
}
```

ex:-
```
For (int i=0 ; i<=100 ; i=i+1 ) {
    Printf ("%d", i) ;
}
```
→ here i is iterator / counter variable.

- **Increment operators**

1) i++ → use, then increase. (Post increment operator)

means first print then increase

ex:-
```
int i=1
~~i++;~~ ~~for (i; i<10; i++);~~
~~Printf (i) → 1~~
printf ("%d", i++) → 1
printf ("%d", i) → 2.
```

2) ++i → increment than use (Pre increment operator).

incrementing i before using / printing it.

- **Decrement operators**

1) i-- → (post decrement operator).

2) --i → (pre decrement operator)

similar to increment operator

NOTE :- loop counter can be float or integer or char

ex :- • for (float i=1.0; i<=5.0; i++) {
printf ("%f", i);
y i → 1.0, 2.0, 3.0 4.0, 5.0.

• for (char i='a'; i<=z; i++)
{ printf ("%f", i);
y → a, b, c, d - - -- x, y, z.

- Infinite loop → loop doesn't end until computer memory is full

2) while loop

```
while( condition) {  // do something
    }
```

here • initialization is done outside loop.
• updation is done inside loop.

ex:
```
int i= 1;  ⟶ initialization
while ( i<=5 ) {   ⟶ condition
    printf( " %d", i );
    i++;  ⟶ updation.
    }
```

here condition will be checked first therefore if it is false it will not execute.

---

3) do while loop

```
do {
// do something
} while(condition);
```

in do while loop even if condition is false the program will execute at least once.

ex:-
```
int a=0 ; i= 1 ;
printf (" enter a number :" ) ;
scanf ( " %d", & a) ;

do {
    printf( " %d ", i);
    i++ )
} while ( i<= a);
return 0;
}
```

- Sum of First n natural numbers

```
#include <stdio.h>  ⟶ Preprocessor directive
int main() {
  int n=0, sum, i;
  printf("enter number ::");
  scanf("%d", &n);
  for(int i=0; i<=n; i++)
      initialization condition updation
  {
    sum = sum+i; // sum+=i
  }
  printf("%d", sum);
  return 0;
}
```

Here in this scope if any variable is created it will immediately be destroyed after its scope.

NOTE :- IN FOR LOOP we can create multiple variable's

example :- `For( int i=0, j=1, x=4 ; i<10 ; j++);`

- example2 :- `For(int i=1, j=n ; i<=n, j>=1 ; i++, j--);`

Break Statement

Break is used to exit the loop

Take input from user until user enters an odd number

ex:-

```
int n;
printf("
Pain scanf("int n; n%2!=0;
for(int n; n%2==0; n++)
  { scanf("%d", n);
    if (n%2!=0)
      { break;
      }
  }
return 0;
}.
```

NOTE !- Break can exit us from nested loops as well

Continue Statement

Skip to the next iteration.

ex:-

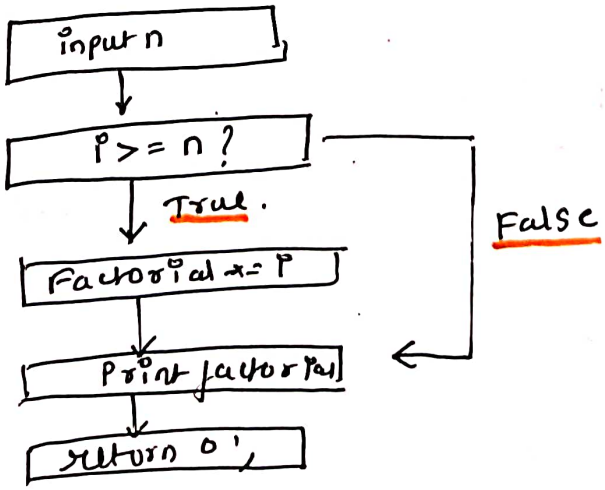
```
For (int i=1 ; i<=5 ; i++)
  { if (i==3) {
      continue ;     // it will skip 3 and go to the next execution of loop.
    printf("%d", i);
  return 0;
  }
```

output
1 2 4 5
↑ skip.

Factorial (example)

```
# include <stdio.h>
int main() {
int factorial=1; int n;

for (int i=1; i<=n; i++)
  {
    Factorial *= i ;  // Factorial = Factorial * i
  }
printf("%d", factorial);
return 0;
}.
```

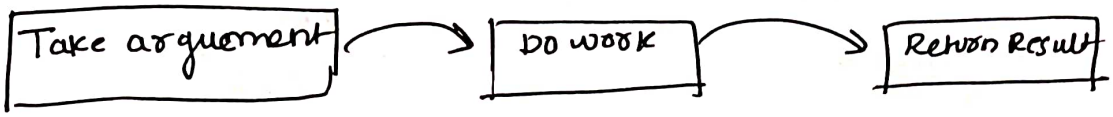
Flowchart



Chapter 5 :- Functions and Recursions

Functions

- block of code that performs particular task.



- It can be used multiple times
- It increases code reusability

Syntax

- Function prototype / Function delaration

```
void printhello();
```

1st step

void: it doesnt return any value

- Function Defination

```
void printhello(){
    printf("Hello");
}
```

2nd step

- Function call

```
int main(){
    printhello();
    return 0;
}
```

3rd step.

Properties of Functions

- execution always starts from main() ,
- Functions get called directly or indirectly
- There can be multiple functions in a program

- **Function Types**

* **library functions**
special functions inbuilt in C
example : scanf() , printf()

* **user defined functions**
declared and defined by programmer.

* Passing arguements

Parameter : functions can take value

Return value : functions give some value

```
int sum(int a, int b)
        ↑ parameters
↓
return value
```

- void printHello(); ← No arguements , returns nothing
- void printTable (int n); ← integer n argument , returns nothing
- ~~void~~ int sum (int a, int b); → Takes two arguements and returns integer value.

Arguement

- values that are passed in function call
- used to send value
- actual parameters

Parameters

values in function declaration and definition.

used to recieve value

formal parameters

NOTE: ★★★★★

- Function can only return one value at a time.
- Changes to parameters in a function dont change values in calling function.

(Because a copy of arguement is passed to the function)

Formal parameters are copy of actual parameters.

Recursion's

when a function calls itself, its called Recursion

ex:-
```
#include<stdio.h>
void hello(int n);        → Function prototype declaration
                          → Parameter (Formal parameters)
int main(){
  int n=5;                → argument (actual parameters)
  hello(n);               ← Function call
  return 0;
}
```

Function definition →
```
void hello(int n){
  if(n==0){ break; }
  Printf("hello /n");
  hello(n-1);             → Parameter
}
```

Function calling itself (Recursive function)

output:
hello
hello
hello
hello
hello

Understanding Recursion using math.

Take $x=2$

: $f(x) = x^2$

$2^2 = 4$

- $f(fx)$ → $f(x) = 2^2 = 4$

$f(4) = 4^2 = 16$

$\therefore f(fx) = 16$

$f(fx) = (2^2)^2$

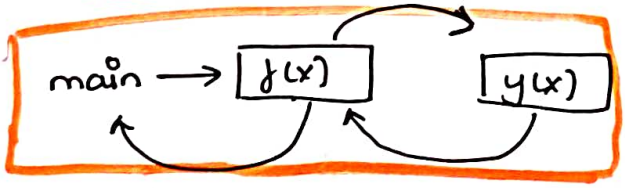
- $f(f(fx))$ gives 2^2 → $2^2 = 4$ returns 4

gives 4^2 → 4^2 16 returns $4^2 = 16$

gives 16^2 → $16^2 = 256$

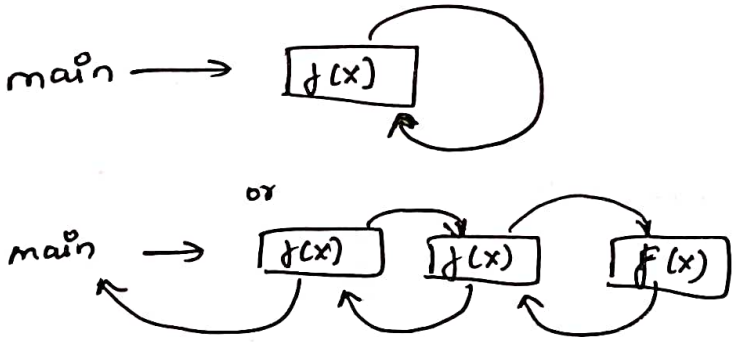
$f(f(fx)) = ((2^2)^2)^2$

Normal function call



- main is asking f(x) for help.
- f(x) is asking y(x) for help.
- y(x) will help f(x) by giving some value.
- this value is used by f(x).
- and the ~~val~~ output of f(x) is returned to main

Recursion function call



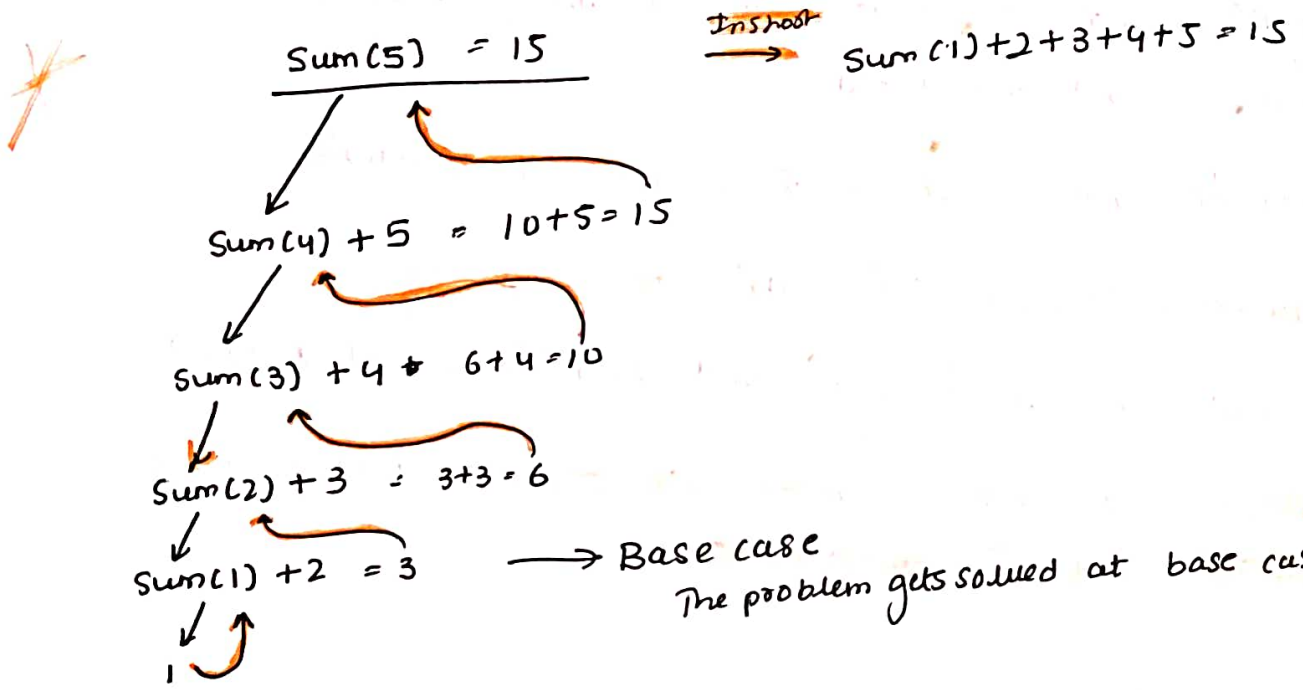
---

ex:-2: sum of n natural numbers

```
#include <stdio.h>
int sum_natural (int n);
int main () {
  int n;
  Printf (" enter a number: "); scanf ("%d", &n);
Printf ("%d ", sum_natural (n));
  return 0;
  }

Int sum_natural (int n) {
                    if (n==1) { return 0; }
  ~~sum_natural +=~~
  return   n + sum_natural (n-1);
  }.
```

Flow diagram

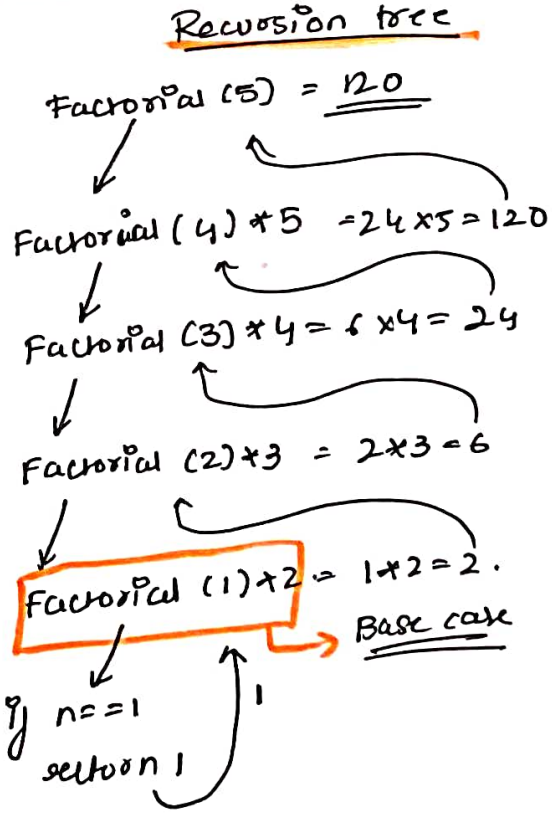
Recursion tree



ex:-3 Factorial.

```
#include<stdio.h>
int Factorial(int n);
int main() {
  int n;
  printf("enter n: ");
  scanf("%d",&n);
  Factorial
  printf("%d", Factorial(n));
  return 0;
  }
int Factorial(int n) {
   if(n==1){
     return 1;
     }
   return n*Factorial(n-1);
  }
```

Recursion tree



Base case: You have to define a value for example in the above case we defined if n==1 return 1;

Calculation. Start from base case.

Properties of Recursion

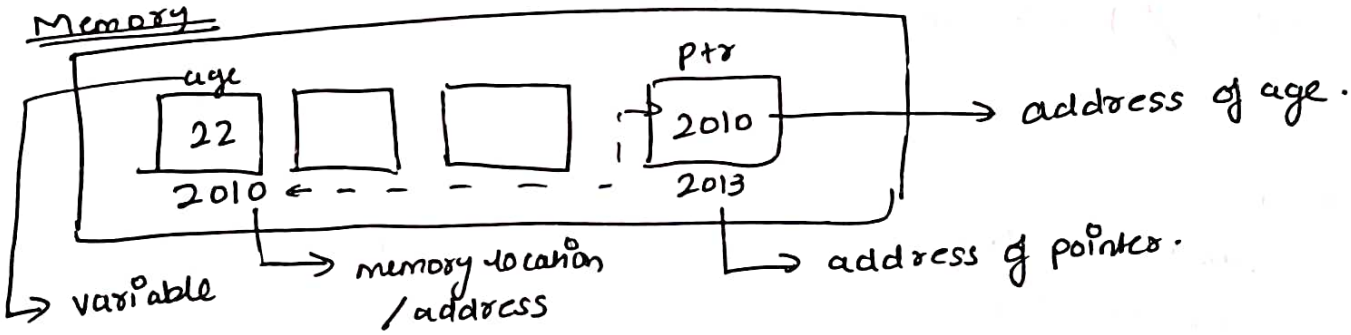
- Anything that can be done with iteration, can be done with recursion and vice-versa.
- Recursion can sometimes give the most simple solution.
- Base-case is the condition which stops recursion.
- Iteration has infinite loop and Recursion has stack overflow
- A Recursive function is incomplete or it will stack overflow if there isn't base case provided.

Chapter-6 Pointers

Pointers → A variable that stores memory address of another variable.

As we know a variable is a name of memory location which stores some data.

the memory location refers to address



variable name can be changed but memory location remains the same.

Syntax

```
int age=22;
int *ptr= &age;
int age = *ptr;
```

* = value at address operator.

& = address of

→ value at address which is stored in pointer i.e value at 2010 = 22.

NOTE!

- int* ptr and int *ptr both are same

- *ptr → value at address operator.

suppose *ptr= &age.

*ptr is equal to *&age.

Declaring pointers

int *ptr ; ⟶ For storing the address of integer var

char *ptr ; ⟶ " " " " character var

float *ptr ; ⟶ " " " " float var

Format specifier

%.p ⟶ hexadecimal value (pointer address)

%.u , %.d ⟶ can be used for just number address

---

ex:- 1

```
#include <stdio.h>
int main () {
int x; *ptr = &x
int *ptr = &x ;
*ptr = 0;
printf (" %.d ", x );
printf (" %.d", *ptr);

 *ptr +=5 ;
printf (" %.d", x );
printf (" %.d", *ptr);

(*ptr)++ ;
printf (" %.d", x );
printf (" %.d", *ptr);

return 0;
}
```

output

x = 0

*ptr = 0

x = 5

*ptr = 5

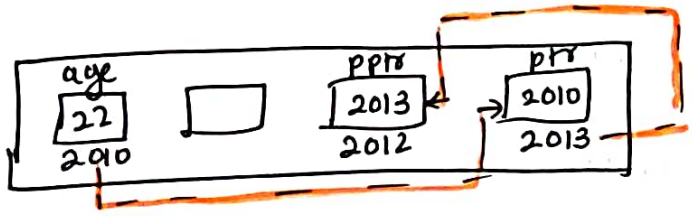
x = 6

*ptr = 6

Here we are indirectly changing the value stored in x using * operator :

Pointer to pointer

A variable that stores the memory address of another pointer



Syntax

```
int **ptr;
char **ptr;
float ** ptr;
```

ex:-
```
#include <stdio.h>
int main() {
  int a=55;
  int *ptr= &a;
  int**pptr= &ptr;
```

Imp

→ here to get the value at a using pointer to pointer we have to write

```
printf("%d", **pptr);
```

****pptr means**

- Value at address of (value at address of pptr).

~~value at d~~ Suppose

& a= 2010
& ptr= 2014
& pptr= 2018

value at address of (value at address of 2018)

↓

value at address of (2010)

↓

because 2018 is storing 2014 and the value at address of 2014 is 2016

↓

~~a=55~~ **pptr= 55;

NOTE :

- value at address of (value at address of (what pptr is storing)

pptr is storing 2014

∴ value at address of 2014

Point to remember:

* = value at address of what ptr is storing
and what ptr is storing?
⇒ address of some variable

& = address of some variable

Pointers in Function Call

call by value : we pass value of variable as arguement.
↓already learnt.

call by reference : we pass address of variable as arguement

ex:-

```
#include<stdio.h>
void square(int n); // passing values as arguements
void -square(int *n);
int main{
    int n=2;
    square(n);          ---> call by value
    printf("-/.d",n);
   -square(&n);         ---> call by reference.
    printf("%d", n);
    return 0;
}
void square(int n){
    printf("%d", n*n);
}
void -square(int *n){
    *n = *n * *n.
    printf("%d", *n);
}
```

output

4] values didnt change
2] permanently

4] value's changed
4] due to changing the actual parameter by accessing its address.

Imp : · In call by value the address of a variable and the address of copy of var is not the same.

· In call by reference both the address are same.

Ex:-

```
int n=5.
printf ("%d", &n);
address (int n);
}
address (int n) {
  printf ("%d", &n);
  }
```

2010 — 213 — Not the same

We know we cannot return multiple values from a function in that case we have to use call by reference

ex:-

```
#include <stdio.h>
void dowork (int a, int b, int *sum, int *prod, int *avg);
int main () {
  int a, b, sum, prod, avg;
  a=5; b=3;
  dowork (a, b, &sum, &prod, &avg);
  printf ("prod is %d, sum is %d, avg is %d", prod, sum, avg);
  return 0;
}
void dowork (int a, int b, int *sum, int *prod, int *avg) {
  *prod = a*b;
  *sum = a+b;
  *avg = (a+b)/2;
}
```

Chapter 7 — Arrays

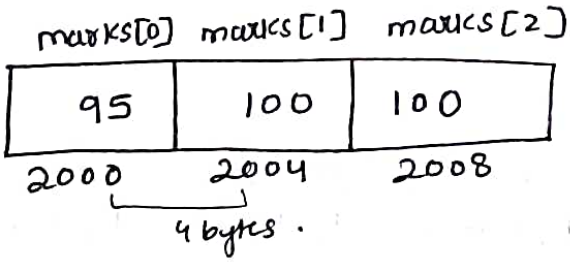
Arrays : collection of similar data types stored at contiguous memory location.

Syntax

- `int marks[3] = {95, 100, 100};`
- `Float marks[3] = {95.0, 100.0, 100.0};`
- `char Aphabets[3] = {'A', 'B', 'C'};`

`Printf("%.d", marks[0])` ⟶ 95

↳ In C indexing / count always starts from 0.



Input and output.

```
scanf("%.d", &marks[0]);

Printf("%.d", marks[0]);
```

Initialization of Array

```
int marks[] = {97, 98, 89};
```

~~here there isn't ~~

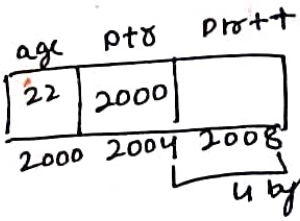
here we donot compulsory require to enter the number of elements in array it will automatically fill itself by seeing the number of elements on RHS.

Pointer Arithmetic

- Pointer can be incremented and decremented

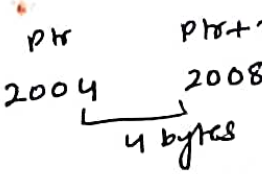
case1:

```
int age = 22;
int *ptr = &age
ptr++;
```



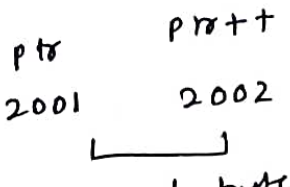
case 2:

```
Float price = 22.0;
float *ptr = &price;
ptr++;
```



case 3:

```
char star = '*'
char *ptr = &star;
ptr++;
```



NOTE: If `*ptr = &age` age is integer.

```
ptr = 2000;
ptr++;
```

doesnt give ptr+1 i.e 2000+1 ✗

it gives ptr+size of data type i.e 2000+4 = 2004 ✓

- we can also subtract a pointer from a pointer
- we can also compare 2 pointers.

---

ex:-
```
#include<stdio.h>
int main() {
 int a, b;
 int *ptr=&a; int *ptr1 = &b;
 ptr++; ptr1++
```

Take ptr = 2004
ptr1 = 2008.

2008 ← ptr++ ; ptr1++ → 2012.

```
Printf("%d", ptr1-ptr);
```

→ 1

1 because it divides by size of data type

```
Printf("%d", ptr1==ptr);
```

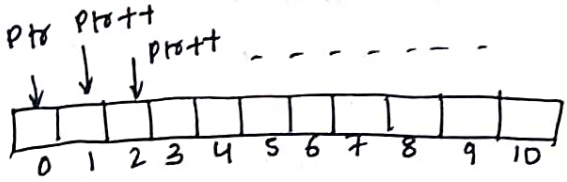
↓ 0 because false

Array is a pointer

```
int *ptr = &arr[0]
```
or
```
Pnt *ptr = arr;
```

The name of array is actually a pointer which points to the 0th element of it.
in this case arr is pointer pointing address of arr[0]

Traverse an Array




```
int aadhar[10];
int *ptr = &aadhar[0];
for (int i=0; i<10; i++){
    printf("enter aadhar:");
    scanf("%d",(ptr+i));
}
printf(int i=0; i<10; i++){
    printf("%d", *(ptr+i));
}
return 0;
}.
```

Arrays as Function Arguement

```
// Function declaration
void printNumbers (int arr[], int n)
        or
void printNumbers (int *arr, int n)

// Function call
printNumber (arr, n);
              └→ = &arr[0].
```

here as you can see we need not write &arr because arr itself is a pointer.

For ex:
```
#include<stdio.h>
void number (int *ptr, int n);
int main() {
int n = 5;
int arr[n];
number (arr, n);
return 0;
}
void number (int *ptr, int n);
*ptr = 1;
for (int i=0; i<n; i++) {
*(ptr[i] = i+1;
printf ("%d", ptr[i]);
}
}
```
pointer

1
2
3
4
5

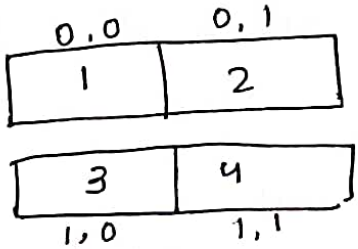
Multidimensional Arrays

ex: There can be n dimensional array

ex:

2D arrays

```
int arr[][] = {{1,2},{3,4}}; // Declare.
```



// Access

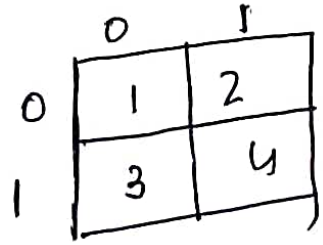
Arr [0][0] → 1

Arr [0][1] → 2

Arr [1][0] → 3

Arr [1][1] → 4

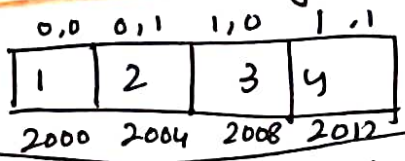
how ? → MATRIX



but how is multi dimensional array stored ?

it is stored in Normal format

ex:-



NOTE : In 2D array we have to give at least one Dimension or it will throw an error : arr[][] ✗ - arr[][10] ✓

ex:-

```
#include<stdio.h>
int main(){
    int subjects, students; subjects=3; students=2;
    int arr[students][subjects];
    for(int i=0; i<students; i++){
        for(int j=0; j<subjects; j++){
            scanf("%d", arr[i][j]);
        }
        printf("\n");
    }
    for(int i=0; i<students; i++){
        for(int j=0; j<subjects; j++){
            printf("%d", arr[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

output

		marks 1 (0)	marks 2 (1)	marks 3 (2)
student 1	0	95	88	100
student 2	1	100	95	88

Suppose printf("%d", arr[1][0]);

↳ 100

Practice Set

1) write a function to count the number of odd numbers in an array

⇒

```
#include <stdio.h>
int oddcount( int *ptr);
int main(){
int arr[10];
for(int i=0; i<10; i++){
    printf("arr[%d] = ", i+1);
    scanf("%d", arr[i]);
}
printf("%d", oddcount(arr));
return 0;
}
int oddcount(int *ptr){
int count=0;
for(int i=0; i<10; i++){
    if(*(ptr+i) % 2 != 0){
        count++;
    }
}
return count;
}
```

NOTE: we cannot define and increment a variable inside a loop

ex:-

```
for(int i=0; i<5; i++){
    int j=1;
    j++;
}
```

Cause the loop runs on again and again and new value of j will be 1 again and again. after incrementing

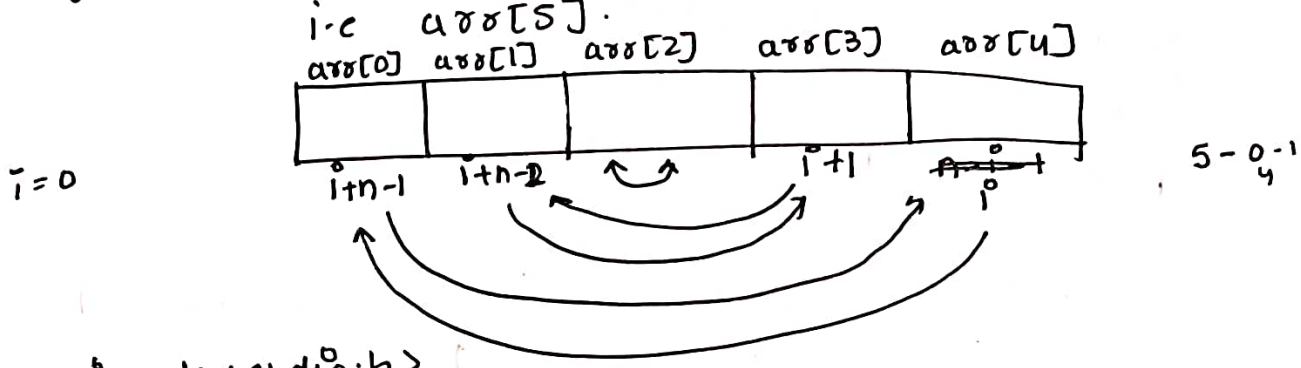
★ If we pass an array as arguement is it call by reference or call by value?

⇒ when we pass an array as an arguement it is always call by reference because array itself is a pointer

2) write a function to reverse the elements of an array

logic ⇒ suppose size of array is 5

i.e arr[5].



⇒ #include<stdio.h>
void rev (int arr[], int n); → call by reference.
int main() {
int n; printf("enter length of array :"); scanf("%d",&n);
int arr[n];
for(int i=0; i<n; i++){
printf("enter arr[%d] :", i+1); scanf("%d",&arr[i]);
}
// Before reversing
for(int i=0; i<n; i++){
printf("arr[%d] is %d\n", i+1, arr[i]); }
rev(arr, n); // after reversing
for(int i=0; i<n; i++) {
printf("arr[%d] is %d\n", i+1, arr[i]); } return 0; }

void rev (int arr[], int n){
for(int i=0; i< n/2; i++) {
firstvalue = arr[i];
Second value = arr[n-i-1]
arr[i] = second value;
arr[n-i-1] = first value; }
}

output
1
2
3
4
5
after reversing
5
4
3
2
1

- write a program to store the first n fibonacci numbers using array

```
#include <stdio.h>
int main (){
  int n; printf("enter a number : "); scanf("%d", &n);
  int Fib[n];
  Fib[0]= 0;
  Fib[1]=1;
  printf("%d", Fib[0]); printf("%d", Fib[1]);
  for(int i=2; i<n; i++){
    Fib[i]= Fib[i-1] + Fib[i-2];
  printf("%d \t" Fib[i]);
  }
  return 0;
  }
```

output

0 1 1 2 3 5 8 13 21

34 55 89 - - - - -

- write a program to get multiples of 2,3 using 2D array

```
#include <stdio.h>
int main (){
  int arr[2][10];
  for(int i=0; i<2; i++){
      for(int j=0; j<10; j++){
        arr[i][j] = (i+2) * (j+1);
      }
  }
  for(int i=0; i<2; i++){
    for(int j=0; j<10; j++){
      printf("%d \t", arr[i][j]);
    }
    printf("\n");
  }
  return 0;
  }
```

output

2 4 6 8 10 12 14 16 18 20

3 6 9 12 15 18 21 24 27 30

• write a program to find out largest number in an array

```
#include<stdio.h>
int main()
{
int n;
printf("enter the number of elements:");
scanf("%d",&n);
int arr[n];
for(int i=0; i<n;i++){
    printf("arr[i] =",i+1);
    scanf("%d", arr[i]);
  }

//sorting
for(int i=0; i<n; i++) {
    if( arr[i]>arr[0]) {
        arr[0]=arr[i]
      }
  }
printf("%d",arr[0]);
return 0;
}.
```

Chapter 8 : Strings

Strings : A character array terminated by a '\0' (null character)

null character denotes string termination

ex: char name[] = {'S', 'H', 'R', 'A', 'D', 'H', 'A', '\0'};

Char name[] = {'A', 'B', ' ', 'K', 'O', 'P', 'A', 'R', 'D', 'E', '\0'};

Imp :

- If we dont add '\0' (null character) at the end it will be treated as character array.

ex:- char arr[] = {'A', 'B', 'C'};

output A
B
C

- and if we add null character at the end it will be treated as string

ex:- char arr[] = {'A', 'B', 'C', '\0'};

output ABC

Initialization of strings

char name[] = {'A', 'B', 'H', 'I', '\0'};

char name[] = "ABHI";

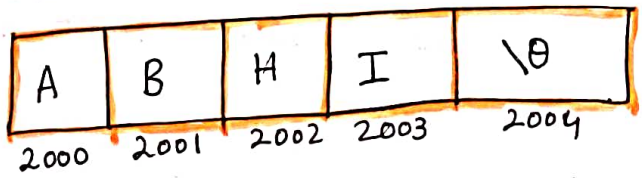
↓

here we need not add "\0" because if we put characters in double quotes ~~if~~ compiler will automatically add the null character.

What happens in memory?

```
char name[] = { 'A', 'B', 'H', 'I', '\0' };
char name[] = " ABHI ";
```

name



here null character is also ~~store~~ stored

Imp:

here while reciving the parameters ~~when~~ we dont need a new variable which gives length of variable because unlike normal arrays which dont know when to end array string knows when null character comes it has to end:

Normal array	string
#include <stdio.h> int arr [arr[], int n] ↳ number of elements	#include <stdio.h> void name [char arr[]);

ex:-

```
#include <stdio.h>
void Name [ char arr[] );
int main() {
    char ~~or~~ firstname [] = " ABHI ";
    char last name [] = " Koparde ";

    ~~void~~ Name ( firstname );
        Name ( lastname );
    return 0;
    }
void Name ( char arr[] ){

    for (int i=0; arr[i] != '\0' ; i++) {
        printf (" %. c ", arr[i] );
      }

    }
```

String Format Specifier

"%.s" → saves a lot of time.

```
char name[] = "ABHI";
printf("%.s", name);
```

in this case we need not print the name character by character using loops.

```
scanf("%.s", name);
```

→ why & is not present
because as we already know name of array itself is a pointer which stores address of 0th element of that array.

Very Important

Try this

```
#include <stdio.h>
int main() {
    char arr[100];
    scanf("%.s", arr); → suppose Input Abhishek Kaparde
    printf("%.s", arr); → output Abhishek.
}
```

** scanf() cannot input multiword strings with spaces

Here

gets() & puts() come into picture

String Functions

★ gets(str) ⟶ Dangerous and outdated.

- input a string (even multi word)
- Dangerous because the softwares might get hacked because it doesn't specify maximum size.

★ Fgets(str, n, file)

- stops when n-1 chars input or new line is entered
- n-1 because the last char is '\0'.

here . str is name of array, n is number of characters,
file = stdin. Std Input.

★ puts(str)

- output a string
- once executed it will automatically go to next line
  no need of \n.

ex:-

```
#include <stdio.h>
int main(){
  char fullname[100];   // → max characters
  // puts(fullname)
  fgets(fullname, 100, stdin);
  puts(fullname);
  return 0;
}
```

Input

Abhishek Koparde

output

Abhishek Koparde

Not possible if we use scanf.

Strings using pointers

Char *str = "Hello World";

Store String in memory and the assigned address is stored in the char pointer 'str'

```
char *str = "Helloworld";  // can be reinitialized.
      str = "world";
char str[ ] = "Hello world";  // cannot be reinitialized.
      str = "world";  ✗
```

Standard Library Functions

<string.h>.

1) strlen(str) → count number of characters excluding '\0'

2) strcpy (newstr, oldstr)

It copies value of old string to new string

3) strcat (firststr, secstr)

- concatenates first string with second string
- the second string always will concatenate with first string but second string will remain same.
- The first string should have space in order to concatenate else there will be an error.

ex:- char first [100] = "Abhi" ← max size

char second [ ] = "Koparde" ← not required

strcat (first, second) = AbhiKoparde.

4) strcmp (firststr, secondstr)

compares two strings and returns a value.

0 → string equal

positive → First > second (ASCII)

negative → First < second (ASCII).

ex:- strcpm (firststr, secstr)

where firststr = Banana

second str = Apple.

In ASCII B = 66.

A = 65

66 > 65 = +

if firststr = HAT

second = HAD.

. first H is compared both has H it will jump to next element

. A is compared both have A it will jump to next element

. T is compared with D.

T > D ∴ positive value.

Practice

Take input from a user character by character.

```
#include<stdio.h>
int main(){
    char str[100];
    int i=0;
    char ch;
    while(ch!='\n'){
      scanf("%c",&ch);
      str[i]=ch;
      i++;
    }
str[i]='\0';
puts(str);
return 0;
}.
```

Salting

Find the salted form of a password entered by user if the salt is "123" and added at the end.

```
#include<stdio.h>
void salting(char arr[]);
int main(){
    char arr[100];
    scanf("%s",&arr);
    salting(arr);
    return 0;
    }
void salting(char arr[]){
char Newpass[200];
char salt[]="123";
strcpy(Newpass,arr);
strcat(Newpass,salt);
puts(Newpass);
}.
```

3) Create a function slice, which takes string and return slice string from index n to m:

```
#include<stdio.h>
void slice (char str[], int n, int m, char newstr[]);
int main(){
    int n, m;
    scanf("%d", &n);
    scanf("%d", &m);
    char str[100], newstr[100];
    scanf("%s", str);
    slice (str, n, m, newstr);

    return 0;
}
void slice (char str[], int n, int m, char newstr[]){
    int j = 0;
    for (int i=n; i<=m; i++){
        printf("%c", str[i]);
        newstr[j] = str[i];
        j++;
    }
    newstr[j] = '\0';
}
```

← very important

NOTE: This is very important the reason being we have mentioned newstr length as 100 (newstr[100]); if we do not put the null character at last all the other spaces will print some garbage value and also it will not be a string but a character array.

- count the occorance of vowels

```
#include <stdio.h>
void vowels ( char arr[] );
int main() {
    char str[100];
    scanf ("%s", str);
    vowels (str);
    return 0;
}
void vowels (char arr[]) {
    int count=0;
char occurence[100];
    for ( int i=0 ; arr[i] < '\0'; i++)
        { if (arr[i]=='a','e','i','o','u' ) {
            count++;
        }
        if (arr[i]=='A','E','I','O','U' ) {
            count++;
        }
    }
    printf (" The occurence of vowels is %d", count);
int j= 0;
for (int i=0 ; arr[i] < '\0' ; i++ ) {
    if (arr[i]== 'a','e','i','o','u','A','E','I','O','U')
        {
            arr[j] = arr[i];
        }
    printf (" The repeating vowels are %c", arr[j] );
    j++;
}
arr[j] = '\0';
}
```

Structures

CHAPTER _9

Structure : a collection of values of different data types

Example :

name (string)
roll no (Integer)
cgpa (Float)
} to store this we can use structure

Syntax

```
Struct Student {
    char name [100];
    int roll;
    float cgpa;
};
```

name

using

```
Struct Student S1;
S1.cgpa = 7.5;
S1.roll = 120;
S1.name;
```

Data types

library data types / In built

- int
- char
- Float
- arr - - - - -

↓

These are the data types which are already available in C

ex:- for int compiler will reserve 4 bytes

User defined data type

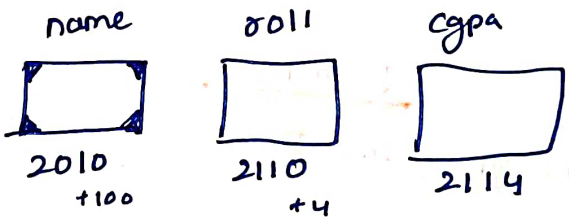
Structures

Ex:-
```
#include <stdio.h>
#include <string.h>

Struct student
{
  int roll; float cgpa; char name[100]; }; → Imp

int main() {
  Struct student s1;   → variable name
    s1.roll = '354;
    s1.cgpa = 8.47;
    Strcpy (s1.name, "Abhishek");
    return 0;
  }.
```

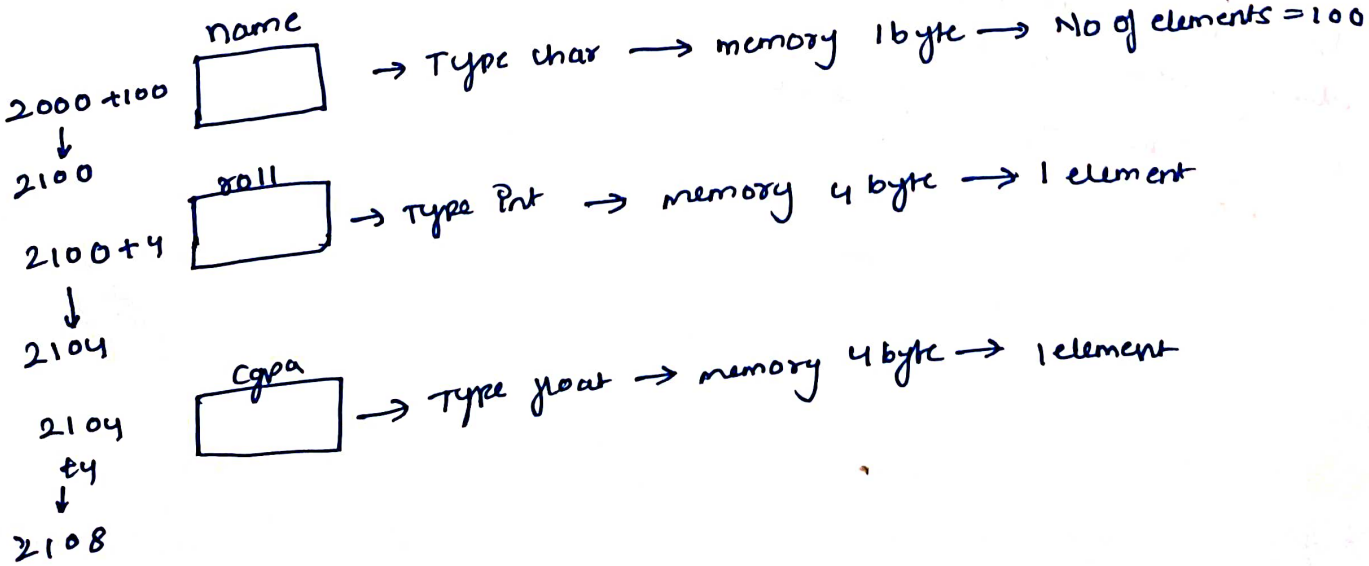
here . is dot operator when ever we want to access the property of structure.

Structures in memory

```
Struct student {
  char name [100];
  int roll;
  float cgpa;
  }
```



Structures are stored in contiguous memory location.

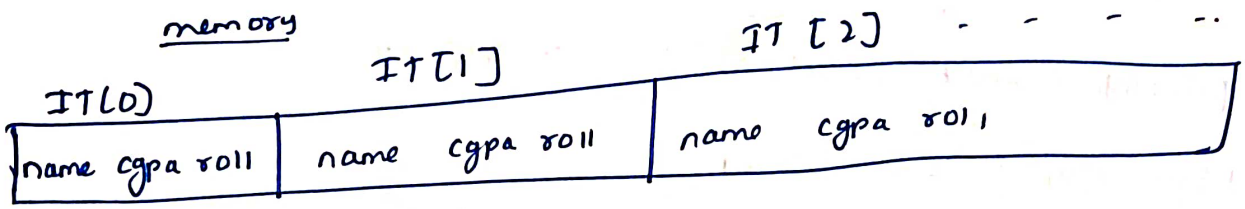


• Array of structures

```
Struct Student IT [100];
```

Access

```
IT [0].roll = 200;
IT[0]. cgpa = 7.6;
```



• Initializing Structures

```
Struct student S1 = { "Abhi", 354, 8.47 };
Struct student S2 = { "Shradha", 355, 8.9 };
Struct Student S3 = { 0 };
```

• Pointers to structures

```
Struct student S1;
Struct Student *ptr;
ptr = &S1;
```

Ex:

```
#include<stdio.h>
struct student { int roll;
    float cgpa;
  };
int main() {
  struct student S1 = { 354, 8.47 }
  Struct student *ptr = &S1;
  printf ("%d ", (*ptr).roll );
  return 0;
  }
```

- Arrow Operator

(*ptr).code ⟷ ptr → code

- Passing structure to function

void printInfo (struct student s1); → Proto type

printInfo (s1); → call

void printInfo (struct student s1); → defination.

here unlike array structure's in function is always call by value;

* typedef keyword

used to create alias for data type.

↓
nick name

```
typedef struct Information_technology {
    int roll;
    float cgpa;
    char name[100];
} IT;
IT student1;
```

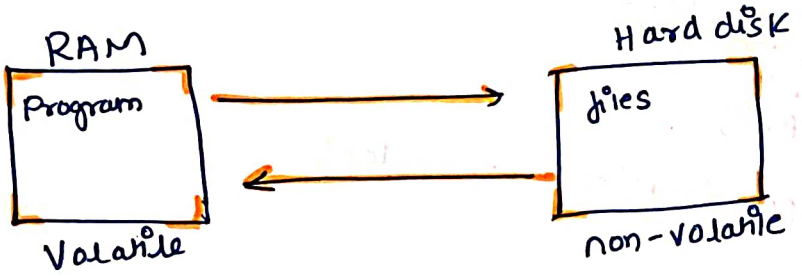
CHAPTER - 10

File input / output

- There are two memories in a computer

1) Volatile memory → as soon as power is disconnected this memory gets erased. ex: unsaved ppt.

2) Non-volatile memory → this memory doesn't get erased: example : movies.



File → container in a storage device to store data

- Ram is volatile
- contents are lost when program terminates
- Files are used to persist the data

Operations on File

- create a file
- open a file
- close a file
- Read from a file
- write in a file.

Types of Files

Text Files
textual data
→ .txt, .c

Binary Files
binary data
.exe, .mp3, .jpg

File pointer

- File is a (hidden) structure that needs to be created for opening a file.

A File ptr that points to this structure and is used to access the file.

```
FILE *fptr;
```

Opening a File

~~File *~~ FILE *fptr;

fptr = Fopen ("Filename", mode);

Closing a File

fclose(fptr);

we need to close the file inorder to save the resources.

File opening modes

- "r" ⟶ open to read
- "rb" ⟶ open to read in binary
- "w" ⟶ open to write
- "wb" ⟶ open to write in binary
- "a" ⟶ open to append.

If we use "w" or "wb" it will overwrite whole file and previous file will not be available, so in order to write something in previous file we use "a".

Reading from a file

```
char ch;
Fscanf (fptr, "%c", &ch);
```

Writing to a file

```
char ch='A';
fprintf(fptr, "%c", ch);
```

Read and write a char

- fgetc(fptr) ⟶ ~~PRINT~~ Read whats in file
- fputc('A', fptr) ⟶ SCAN and put input in file

NOTE: For other data types we can use fscanf or fprintf.

example :- How to create and write in a file.

```
#include <stdio.h>
int main() {
    FILE *ptr;
    ptr= Fopen("Hello.txt","w");
    fprintf(ptr, "%d", 31);
    Fputc('H', ptr);
    Fputc('I', ptr);
    return 0;
}
```

No file named Hello ⟶ Hello.txt file created.

⟶ open file you will see 31 HI

EOF (End Of File)

fgetc returns EOF to show that file has ended.

ex:- Random.txt

This is a random string.

EOF.c

```
#include <stdio.h>
int main() {
    FILE *ptr;
    ptr = fopen("Random.txt", "r");
    char ch = fgetc(ptr);
    while (ch != EOF) {
        printf("%c", ch);
        ch = fgetc(ptr);
    }
    fclose(ptr);
    return 0;
}
```

output → This is a random string.

examples:

1) write a program to write n integers in a file

```
#include <stdio.h>
int main () {
    FILE *inte;
    inte= fopen( "integers.txt", "w");  → creates integers.txt file
    int n;  printf ("enter n:"); scanf("%d", &n");
    for (int i=0 ; i<=n; i++) {
        fprintf (inte, "%d", i);
    }
    fclose (inte);
    return 0;
}
```

Input = 5

integers.txt

1 2 3 4 5.

2) write a program to write info of a student in a file.

```
#include <stdio.h>
Int main () {
    FILE *student;
    student= fopen ( "student.txt", "w");  → creates file.
    Char name[100];
    int age;  Float cgpa;
    Printf ("enter name :"); scanf ("%s", name);
    Printf ("enter age :"); scanf ("%d", &age);
    printf ("enter cgpa :"); scanf ("%f", &cgpa);
    Fprintf (student, "%s", name);
    fprintf (student, "%d", age);
    fprintf (student, "%f", cgpa);
    fclose (student);
    return 0;
}
```

student.txt

Abhishek 20 8.47.

Input

Abhishek
20
8.47

3) write a program to use 2 integers in a file and replace it with its sum;

Sum.txt
1
2

```
#include <stdio.h>
int main() {
    FILE *sum;
    Sum = Fopen("sum.txt", "r");
    int a, b;
    fscanf(sum, "%d", &a);
    fscanf(sum, "%d", &b);
    fclose(sum);
    Fopen("sum.txt", "w");
    fprintf(sum, "%d", a+b);
    fclose(sum);
    return 0;
}
```

Sum.txt
3