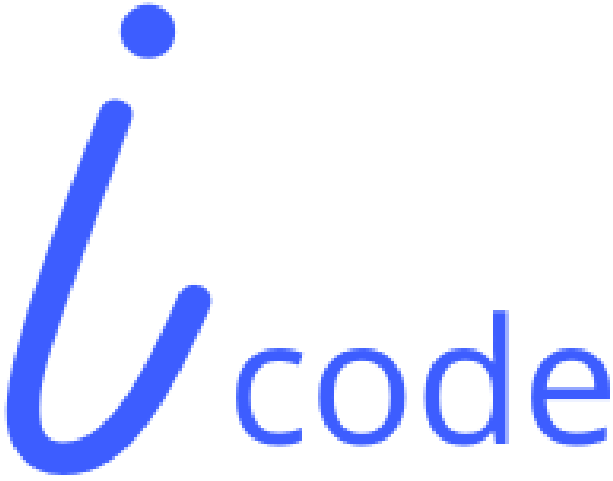



presents

# C programming notes

# "C Programming"

→ What is programming?

Computer Programming is a medium for us to Communicate with Computers Just like we use "Hindi" or "English" to communicate with each other. Programming is a way for us to deliver our instructions to the Computer.

– What is C?

- C is a programming language
- C is one of the oldest and finest language {Programming}
- C was developed by Dennis Ritchie at AT & T's Bell Labs, USA in 1972.

→ Uses of C

C language is used to program a wide variety of systems. Some of the uses of C are as follow:

1) Major part of Windows, Linux and other operating system are written in C

2) C is used to write driver programs for devices like Tablets, printers etc.

3) C language is used to program embedded systems where programs need to run faster in limited memory [Microwave, Cameras etc.]

4) C is used to develop games, an area where latency is very important i.e Computer has to react quickly on user input.

→ Variables.

A Variable is a Container which stores a 'Value'. Ex:- In Kitchen, We have Containers which store rice, dal, sugar, wheat etc. Similarly to that Variables in C stores value of a Constant.

→ Example

```
a = 3;      // a is assigned "3"
b = 4.7;    // b is assigned "4.7"
C = 'A';    // C is assigned 'A'
```

# Rules for Naming in C (Variables)

1.) first character must be an alphabet or underscore (_).

2.) No special symbole other than (_) is allowed.

3.) No Comas, blanks allowed.

4.) Variable names are caseSensitive.

- We must Creak meaningful Variable names in our programs. This ~~enab~~ enhance readability of our programs

## → Constants

- An entity Whose Value does not change is called as a Constant.
- A Variable is an entity Whoes Value changes.

## - Types of Constant.

Primarily there are three types of Constant.

1) Integer Constant → -1, 6, 7, 9

2) Real Constant → -322.1, 2.5, 7.0

3) Character Constant → 'a', '$', '@' [Must be enclosed in single inverted Commas]

## → Key Words.

| | | | | |
|---|---|---|---|---|
| auto | do | Size of | for | unsigned |
| break | double | Static | goto | Void |
| Case | long | int | if | Volatile |
| Char | return | else | struct | While. |
| Const | register | enum | Switch | |
| Continue | Short | extern | typedef | |
| default | Signed | float | union | |

→ Our first C Program

```
#include <stdio.h>
int main(){
    Printf("Hello, I am learning C programming language");
    return 0;
}
```

→ file name: first.c

# Basic structure of a C program

All C programs have to follow a basic structure. A C program starts with a main function and executs function{Instructions} inside it.

- Each instruction is terminated with a semicolon (;).

→ There are some rules which are applicable to all the C programs:

1) Every program's execution starts from main() function.
2) All the statements are terminated with a semicolon.
3) Instructions are Case-Sensitive
4) Instructions are executed in the same order in which they are written.

→ Comments

Comments are used to clarify something about the program in plain language. It is a way for us to add notes to our program There are two types of Comments In C:-

1 Single-line Comment: // This is Comment.

2 Multi-line Comment: /* This is multi-line Comment
Which we use to Comment multiple lines */

- Comments in C program are not executed and are Ignored.

## → Compilation and Execution.

first.c ⇒ [C Compiler] ⇒ first.exe

in V-S code

gcc

A Compiler is a Computer program which converts a C program in to machine language so that it can be easily understood by the Computer.

A C program is written in plain text.

This plain text is Combination of Instructions in a particular sequence The Compiler performs some basic checks and finally converts the program into an executable.

## → Library functions.

C language has a lot of valuable library functions which is used to carryout certain tasks for instance printf function is used to print values on the screen

```
Printf ("This is %d", i);
```

'%d' for integers.

'%.f' for real values

'%.c' for characters

## → Type of Variables.

1) Integer Variables → int a=3;

2) Real Variables → float a = 7.7;

3) Character Variables → char a = 'B';

## → Receiving input from the user

In order to take input from the user and assign it to a variable, We use 'Scanf function'.

Syntax for using Scanf:

```
Scanf ("%d", &i);
```

↳ This & is important!

"&" is the "address of" operator and it means that the supplied Value should be copied to the address which is indicated by Variable i.

## Practice Set.

Ques 1 Write a C program to calculate area of a rectangle

a) using hard coded inputs

b) using input supplied by the user.

Ques 2 Calculate the area of a Circle and modify the same program to calculate the Volume of a Cylinder given its radius & height.

Ques 3 Write a program to Convert Celcius in to farenheit.

$\{[0c \times \frac{9}{5}] + 32 = 32f\}$

Ques 4 Write a program to calculate Simple interest for a set of Values representing principal, no. of years & rate of interest.

$$\text{Simple interest} = \frac{P \times R \times t}{10}$$

Where P = Principal amount
R = Rate of intrest
t = time period

# → Instructions and operators

A C program is a set of instructions. Just like a receipie - which contains instruction to prepare a particular Dish.

→ Types of Instructions.

1) Type declaration Instruction
2) Arithmetic Instruction.
3) Control Instruction.

• Type declaration Instruction

```
int a ;    float b ;
int b ;
```

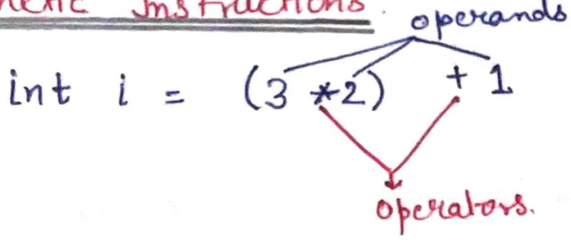
→ other Variations:

```
int i=10 , int j=i ; int a=2
int j1 = a+j-i ;
```

```
float b = a+3 ; float a = 1.1   =>Error! as we are trying to use 'a' before
                                   defining it.
```

```
int a,b,c,d ;
a = b = c = d = 30  => Value of a,b,c & d will be 30 each.
```

→ Arithmetic Instructions.




operands Can be int / float etc.
+, −, *, / are arithmetic operators

```
int b=2 , c=3;
int z ;  z = b*c ;   ✓ This is legal
int z;  b*c = z ;   (X) This is Illegal [Not allowed in C].
```

→ % [Modular division operator]

① Returns the remainder.

② Can not be aplied on float

③ Sign is same as on numerator [ -5 % 2 = -1]

Note :-

① No operator is assumed to present.

int i = ab → Invalid

int i = [a * b] → Valid.

② There is no operator to perform exponentiation in C. However we can use pow (x, y) from <math.h>

# Type Conversion

An Arithmetic operation between.

Int and Int → Int

Int and float → float

float and float → float

•important

[ 5/2 → 2 , 5.0/2 → 2.5 ; 2/5 → 0 , 2.0/5 → 0.4 ]

Note :

→ Int

int a = 3.5 ; In this case 3.5 [float] will be demoted to 3 (int) because a is not able to store floats.

→ float.

float a = 8 ; a will store 8.0

8 → 8.0 [promotion to float]

- Quick quiz

Q int K = 3.0/9 Value of K? & Why?

Solu 3.0/9 = 0.333 but since K is an int, it can not store float & Value 0.333 is demoted to 0.

## # Operator precedence In C

$3 * x - 8y$ is → $(3x) - (8y)$ or → $3(x - 8y)$ ?

In C language simple mathematical rules like BODMas, no longer applies. "The answer to the above question is provided by operator Precedence & associativity."

## # operator precedence

The following table lists the operator priority in C

| Priority | operators. |
|---|---|
| 1st | *, /, % |
| 2nd | +, − |
| 3rd | = |

Operators of higher priority and evaluated first in the absence of parenthesis.

## # Operator Associativity.

When operators of equal priority are present in an expression, the tie is taken care of by associativity.

$x * y / z \Rightarrow (x * y) / z$

$x / y * z \Rightarrow (x / y) * z$

*, / follows left to right associativity.

# Control Instructions.

Determines the flow of Control in a program four types of Control Instruction in C are:

1) Sequence Control Instruction.
2) Decision Control Instruction.
3) Loop Control Instruction
4) Case Control Instruction.

## Practice Set.

**Ques 1** Which of the following is valid in C?

i) int a; b = a; → This is invalid.
[In Case of Power]
ii) int V = 3^3; → This also invalid because in C we use power {Function from math.h}
iii) Char dt = '21 Dec 2020'; → This is Invalid.

**Ques 2** What data type will 3.0/8 - 2 return? → Double. [By Default]

**Ques 3** Write a program to check whether a no. is divisible by 97 or not.

**Ques 4** Explain step by step evaluation of 3*x/y - z + R, where

x = 2, y = 3, z = 3, R = 1

**Solu** ⇒ 3*2/3 - 3+1
⇒ 6/3 - 3 + 1 ⇒ 2 - 3 + 1 ⇒ -1 + 1 ⇒ 0 Ans.

**Ques 5** 3.0 + 1 will be

(X) a) Integer
(✓) b) floating point no. → This is right answer.
(X) c) Character.

## → Conditional Instructions.

- Sometimes We want to Watch Comedy vedios on Youtube if the day is Sunday.
- Sometimes We order junk food if it is our friend's birthday in the hostel.
- You order the meal if dal or your favourite bhindi is listed on the menu.

All this decisions are depends on Condition. In C language too, We must be able to execute instructions on a Condition(s) being met.

## # Decision Making [Statement or Instructions] in C

→ If-else statement
→ Switch statement.

## → If-else statement

The Syntax of an if-else statement in C looks like:

```
if (condition to be checked) {
        Statements-if-Condition-true;
    }
else {
        Statements-if-Condition-false;
    }
```

→ Code Example:

```
int a = 23;
if (a > 18) {
        Printf ("You Can drive \n");
    }
```

# Note :- that else block is not necessary but optional.

## → Relation operators in C

Relational operators are used to evaluate conditions [true or false] of inside the if statements. Some examples of relational operators are:

greater than — less than or equal to

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

== : equals
>= : greater than or equal to
> : greater than
< : less than
<= : less than or equal to
!= : not equal to.

**Note** : '=' is used for assignment where as '==' is used for equality check.

The condition can be any valid expression. In C a non-zero value is considered to be true.

## # Logical operators

&&, || and ! are three logical operators in C. These are read as 'AND', 'OR' and 'NOT'. They are used to provide logic to our C programs.

## → Usage of logical operators.

i) && → AND → is true when both the conditions are true.

"1 and 0" is evaluated as false.
"0 and 0" is evaluated as flase.
"1 and 1" is evaluated as true.

ii) || → OR → is true when at least one of the condition is true.

iii) ! → returns true if given false. & false if given true.

!(3==3) → evaluates to false.

!(3>30) → evaluates to true.

As the no. of condition increases, the level of indentation increases. This reduces readability. logical operators come to reseve in such case.

→ **else if Clause**

Instead of using multiple if statements, we can do use else if along with if thus forming an if-else-if-else ladder.

```
if {
    Statements;
   }
else if {.....
        }
else {.....
        }
```

Using if-else if-else reduces indents. The last "else" is optional. Also there can be any number of "else if"

→ Last else is executed only if all conditions fail.

→ **Operator precedence.**

| Priority | Operator |
|---|---|
| 1st | ! |
| 2nd | *, /, % |
| 3rd | +, - |
| 4th | >, <, <=, >= |
| 5th | ==, != |
| 6th | && |
| 7th | \|\| |
| 8th | = |

→ **Conditional operators**

A short hand "if-else" can be written using the conditional or ternary operators.

Condition ? experession-if-true : expression-if false

(Ternary operator)

## → Switch Case Control Instruction.

Switch-Case is used when we have to make a choice between no. of alternatives for a given variable.

```
Switch (integer-expression)
    {
    Case C1:
        Code;
    Case C2:
        Code;
    Case C3:
        Code;
    default:
        Code;
    }
```

The value of integer-expression is matched against $C_1, C_2, C_3 \ldots$ If it matches any of the cases, that case along with all subsequent "case" and "default" statements are executed.

→ Quick quiz: Write a program to find grade of a student given his marks based on below:

→ 90-100 → A
→ 80-90 → B
→ 70-80 → C
→ 60-70 → D
→ <60 → F

## # Important Notes.

1) We can use Switch-Case statement even by writing cases in any order of our choice [Not necessarily ~~to~~ ascending].

2) Char values are allowed as they can be easily evaluated to an integer.

3) A Switch can occur within another but in practice this is rarely done.

# Practice Set.

**Ques 1** What will be the output of this program

```
int a=10;
if (a=11)
    Printf ("I am 11");
else
    Print ("I am not 11");
```

**Solve** [Here the output will be 'I am not 11'.] — This is wrong because a=11 is assigning & '==' is used to check equality.

**Ques 2** Write a program to find out whether a student is pass or fail, if it requires total 40% and at least 33% in each subject to pass. Assume 3 subjects are taken and marks are as an input from user.

**Ques 3** Calculate income tax paid by an employee to the government as per the slabs mentioned below:

| Income tax Slab | tax |
|---|---|
| 2.5L - 5.0L | 5% |
| 5.0L - 10.0L | 20% |
| Above 10.0L | 30% |

**Note:** There is no tax below 2.5L Take income amount as an input from the user.

**Ques 4** Write a program to find whether a year entered by the user is a leap year entered or not. Take year as an input from user.

Quess Write a program to determine whether a character entered by the user is lowercase or not.

Quess Write a program to find greatest of four no. entered by the user.

## LOOP CONTROL INSTRUCTION

→ Why Loops

Sometimes we want our programs to execute few set of instruction over and over again for example: printing 1 to 100, first 100 even number etc.

Hence loops make it easy for a programmer to tell Computer that a given set of instructions must be executed repeatedly.

→ Types of loops

Primarily there are three types of loops in c language:

1) While-loop
2) Do-while loop
3) for loop

We will look in to these one by one.

# While Loop

```
While (Condition is true) {

    // Code
    // Code

}
```

=> The block keep executing as long as the condition is true.

An Example:

```
int i=0
while (i<10){
        Printf ("The Value of i=%d", i); i++;
            }
```

Note :- If the condition never becomes false, the While loop keeps executed Such a loop is known as an infinite loop.

Quick quiz : Write a program to print natural number from 10 to 20 When initial loop Counter i is initialized to 0.

The loop Counter need not be int, it can be float as well.

# Increment and Decrement operator.

i++ — i is increased by 1

i-- → i is decreased by 1

Printf ("--i = %d", --i);

This first decrements i and then print it.

Printf (" i-- = %d", i--);

This first print i and then decrement it.

— *** Imp +++ operator does not exist

* += is Compound assignment operator just like -=, *=, /= and %=.

→ **Do-While Loop**

The syntax of do-while loop looks like this.

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

do while loop works very similar to while loop.

while → checks the condition & then executs the code.

do while → Executes the code & then checks the condition.

do while loop = while loop which executes at least once.

Quick quiz: Write a program to print first n natural numbers using do-while loop

Input: 4

output: 1
2
3
4

# **for loop**

The syntax of for loop looks like this:

```
for (initialize; test; increment or decrement)
    {
     // code;
     // code;
     // code;
     }
```

→ Initialize → Setting a loop counter to an initial value.
→ Test → Checking a condition.
→ Increment → updating the loop counter.

An Example:

```
for (i=0, i<3, i++)
    { Printf ("%d", i);
      Printf ("\n");
    }
```

Output: 0
1
2

Quick Quiz: Write a program to print first 'n' natural numbers using for loop.

→ A Case of Decrementing for loop.

```
for (i=5, i; i--)
    {
      Printf("%d \n", i);
    }
```

This for loop will keep on running until i becomes 0.

The loop runs in following steps:

1> i is initialized to 5

2> The condition "i" (0 or none) is tested

3> The Code is executed

4> i is decremented

5> Condition i is checked & code is executed if its not 0.

6> & So on until i is non 0

Quick Quiz: Write a program to print 'n' natural no. in reverse order.

## # The break statement in C

The break statement is used to exit the loop irrespective of whether the Condition is true or false.

Whenever a "break" is encountered inside the loop the control is sent outside the loop.

Let us see this with the help of an Example.

```
for (i=0 ; i<1000 ; i++)
    { Printf("%d \n", i);
  if (i==5)
  {
   break;
   }
   }
```

output ⇒ 0 1 2 3 4 5 and not 0 to 100 ☺

## # The Continue statement in C

The Continue statement is used to immidiately move to the next iteration of the loop.

The control is taken to the next iteration thus skipping everything below "Continue" inside the loop for that iteration.

⟶ Let us look on Example

```
int skit = 5;
int i = 0;
while (i<10){
 if (i != skip)
        continue;
 else
    Printf ("%d", i);
```

output ⇒ 5

and not 0 .... 9

Notes:

1) Sometimes, the name of the variable might not indicate the behaviour of the program.

2) break statement completely exist the loop.

3) Continue statement skips the particular iteration of the loops.

## Practice Set

1 Write a program to print multiplication table of a given number n.

2 Write a program to print multiplication table of 10 in reverse order.

3 A do While loop is executed:

1) at least once

2) at least twice

3) at most once

4 What can be done using one type of loop can also be done using the other type of loops – True or false?

5 Write a program to sum first ten natural no. using While loop.

6 Write a program to calculate the factorial of a given no. using a for loop.

7 Write a program to calculate the sum of the no. occuring in the multiplication table of 8. (Consider 8x1 to 8x10)

8 Write a program to sum first ten natural no. using a for and do-While loop.

9 Reapt 8 using While loops.

10 Write a program to check whether a given no. is prime or not using loops.

11 Implement 10 using other types of loops.

# function And Recursion.

Sometimes our program gets bigger in size and its not possible for a programmer to track which piece of code is doing what.

functions is a way to break our code into chunks so that it is possible for a programme to reuse them.

→ What is a function?

- A function is a block of code which performs a particular task.
- A function can be reused by the programmer in a given program any no of times.

→ Example & Syntax of a function.

```
# include <stdio.h>
  Void display();        ⇒ function prototype.

  int main(){
    int a;
    display();           ⇒ function Call.
    return 0;
    }

  Void display(){
                         ⇒ function defination
     Prinf("Hi I am display");
     }
```

# function Prototype: function prototype is a way to tell the Compiler about the function we are going to define in the program.

⟶ Here the Void indicates that the function returns nothing.

# function Call : function Call is a way to tell the Compiler to execute the function body at the time the Call is made.

Note that the program execution starts from the main function in the sequence the instructions are written.

# function definetion : This part Contains the exact set of instructions which are executed during the function Call. When the function is Called from main(), the main function falls asleep & gets temporarily suspended. During this time the Control goes to the function being Called. When the function body is done executing main() resumes.

Quick quiz : Write a program with three functions.

1> Good morning function which prints "Good Morning".
2> Good afternoon function which prints "Good Afternoon".
3> Good night function which prints "Good night".

main() should call all of those in order 1 → 2 → 3

→ Important Points

① Execution of a C program starts from main()
② A C program can have more than one function
③ Every function gets called directly or indirectly from main()
④ There are two type of function in C. Let's see them.

- Type of functions

① Library function - Commonly required functions grouped together in a library file or disk.

② User defined function - These are the functions declared and defined by the user.

Q Why we use functions ?

1) To avoid rewriting the same logic again and again.

2) To keep track of what we are doing in a program.

3) To test and check logic independently.

# Passing Values to functions.

We can pass values to a function and get a value in return from a function.

```
int Sum (int a , int b)
```

The above prototype means that Sum is a function which takes values a (of type int) and b (of type int) and returns a value of type int

→ function definition of Sum can be:

```
int Sum (int a, int b) {
    int c;
    int c = a+b;          ⇒ a & b are parameters.
    return c;
}
```

Now we can call Sum (2,3) ; from main to get 5 in return.

⇒ Here 2 & 3 are arguments.

```
int d = Sum(2,3) ;    d becomes 5
```

→ Note :

1) Parameters are the values or variables placeholders in the function definition. Ex: a & b

2) Arguments are the actual values passed to the function to make a call
Ex: 2,3

3) A function can return only one value at a time.

4) If the passed variable is changed inside the function, the function call doesn't change the value in the calling function.

```
int change (int a){
        a = 77;          ⇒ Misnomer
        return 0;
        }
```

Change is a function which changes a to 77. No if we call it from main like this.

```
int b=22
change (b);          ⇒ The value of b remains 22
Printf ("b is %d", b);   ⇒ print "b is 22".
```

This happens because a copy of b is passed to the change function.

Quick quiz - use the library functions to calculate the area of a square with side a.

## # Recursion.

- A function defined in C can call itself. This is called recursion.
- A function calling itself is also called 'recursive function'

### → Example of Recursion.

- A very good example of Recursion is factorial.

factorial (n) = 1 x 2 x 3 . . . x n

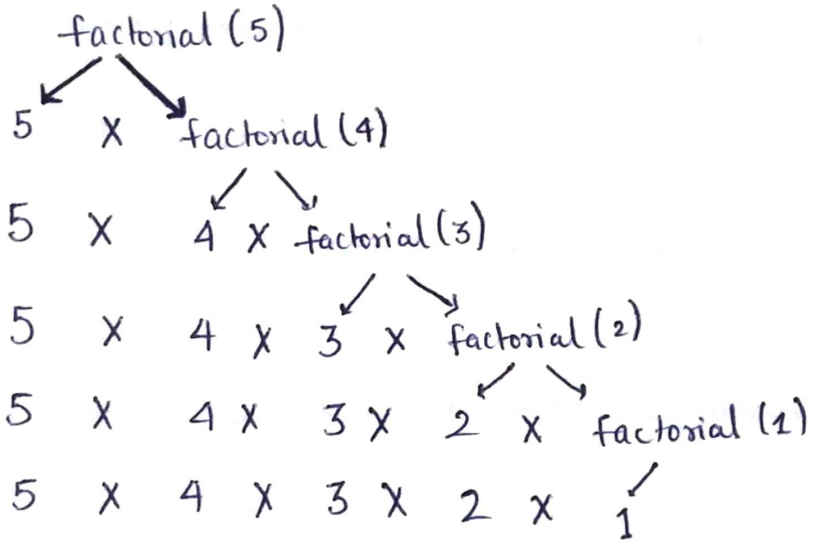
factorial (n) = 1 x 2 x 3 . . . n-1 x n

factorial (n) = factorial (n-1) x n

Since we can write factorial of a no. in terms of itself, We can program it using recursion.

```
int factorial (int x){
    int f;
    if (x==0 || x==1)
        return 1;
    else
        f = x* factorial(x-1);
        return f;
        }
```

→ How does it Works?




→ Important notes :

1) Recursion is sometimes the most direct way to code an algorithm.

2) The condition which doesn't call the function any further in a recursive function is called as the base condition.

3) Sometimes, due to a mistake made by the programmer, a recursive function can keep running without returning resulting in a memory error.

# Practice Set.



1) Write a program using functions to find average of three numbers.

2) Write a function to Convert Celcius temperature into farenheit.

3) Write a function to calculate force of attraction on a body of mass 'm' exerted by earth ($g = 9.8\ m/s^2$)

4) Write a program using recursion to calculate $n^{th}$ element of fiborachi series.

5) What will the following line produce in a C program:

```
Printf ("%d %d %d \n", a, ++a, a++);
```

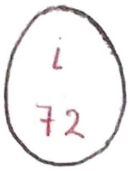
6) Write a recursive function to calculate the sum of first n natural numbers.

7) Write a program using functions to print the following pattern. [First 4 lines]

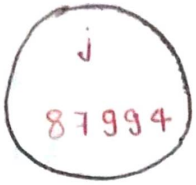
```
*
* **
* * * *
* * * * *
```

# POINTERS

A Pointer is a variable which stores the address of another variable.




address → 87994




address → 87998

j is a pointer

j points to i

The "address of" (&) operator is used to obtain the address of a given variable

If you refer to the diagrams above

$\&i \Rightarrow 87994$

$\&j \Rightarrow 87998$

formate specifier for printing address is '%u'

→ The 'Value at address' operator (*)

→ The Value at address or * operator is used to obtain the Value present at a given memory address. It is denoted by (*)

$*(\&i) = 72$

$*(\&j) = 87994$

— How to declare a pointer?

A pointer is declared using the following syntax.

```
int *j;   => declare a variable j of type int-pointer
j = &i    => store address of i in j.
```

Just like pointer of type integer, we also have pointers to char, float etc.

```
int *ch_ptr;      → pointer to integer
Char * ch_ptr;    → pointer to character
float * ch_ptr;   → pointer to float.
```

- Although its a good practice to use meaningful Variable names, we should be very Careful while reading & working on programs from fellow programmers.

→ A Program to demostrate pointers.

```
#include <stdio.h>
int main(){
int i=8;
int *j;
j = &i;
Printf("Add i = %u \n", &i);
printf("Add i = %u \n", &j);
printf("Add j = %u \n", &j);
printf("Value i = %d \n", i);
printf("Value j = %d \n", *(&i));
printf("Value i = %d \n", *j);
Return 0
}
```

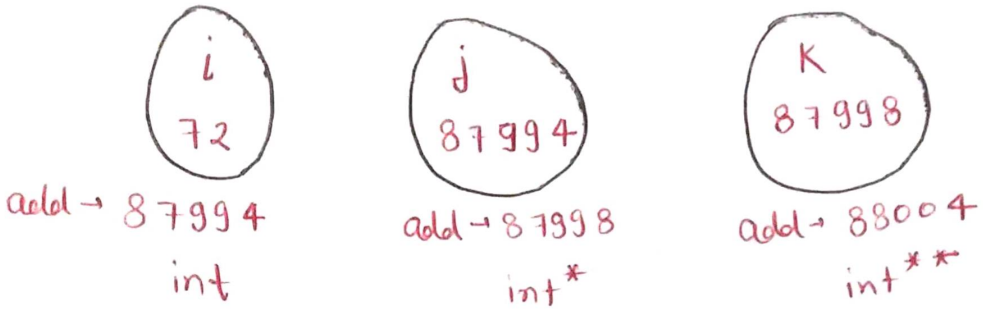
- Output:
```
Add i = 8799 4
Add i = 87994
Add j = 8799 8
Value i = 8
Value i = 8
Value i = 8
```

" This program sums it all. If you understand it, you have got the idea of pointer."

## — Pointer to a Pointer.

Just like 'j' is pointing to 'i' or storing the address of i, we can have another variable k which can further store the address of j. What will be the type of k.

```
int **k;
k = &j;
```




We can even go further one level and create a variable & of type int*** to store the address of K. We mostly use int * and int** sometimes in real world programs.

## — Types of function Calls.

Based on the way we pass arguments to the function, function calls are of two types.

1) Call by value → Sending the values of arguments
2) Call by reference → Sending the address of arguments.

## Call by Value.

Here the value of the arguments are passed to the function. Consider this example.

```
int c = sum(3, 4);      ⇒ assume x=3 and y=4
        x, y
```

If Sum is defined as sum(int a, int b), the value 3 & 4 are copied to a and b. Now even if we change a and b, nothing happens to the variable x and y.

This is call by value.

In C we usually make a call by value.

## Call by Reference.

Here the address of the variables is passed to the function as arguments.

Now since the address are passed to the functions, the functions can now modify the value of a variable in calling function using * and & operators.

### Example

```
Void swap (int*x, int *y)
{
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
}
```

This function is capable of swapping the values passed to it.

if a = 3 and b = 4 before a call to swap(a,b), a = 4 and b = 3 after calling swap.

```
int main(){
int a = 3;
int b = 4;
swap(a,b);
return 0;
}
```

→ a = 3 & b = 4

→ Now a = 4 & b = 3

# Practice Set.

1) Write a program to print the address of a variable. Use this address to get the value of this variable.

2) Write a program having a variable i. Print the address of i. Pass this variable to a function and print its address. Are the address same? Why?

3) Write a program to change the value of a variable to ten times of its current value. Write a function and pass the value by reference.

4) Write a program using a function which calculates the sum and average of two numbers. Use pointers and print the value of sum and average in main().

5) Write a program to print the value of a variable i by using "pointer to pointer" type of variable.

6) Try problem 3 using call by value and verify that it doesn't change the value of the said variable.

# Arrays

An array is a Collection of similar elements
(imp.)

One Variable ⇒ Capable of storing multiple Values.

## Syntax

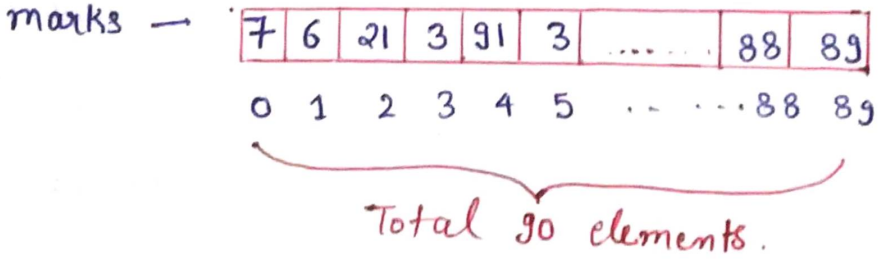
The syntax of declaring an Array looks like this:

```
int marks[90];    ⇒ Integer array
char name[20];    ⇒ Character array or string
float percentile[90]; ⇒ float array.
```

The Value now be assigned to marks array like this:

```
marks[0] = 33;
marks[1] = 12;
```

Note: It is very important to note that the array index starts with 0.




## # Accessing elements.

Elements of an array can be accessed using:

```
Scanf("%d", &marks[0]); ⇒ Input first Value
printf("%d", marks[0]); ⇒ output first Value of the array.
```

Quick quiz: Write a program to accept marks of five students in an array and print them to the Screen.

# Initialization of an Array

There are many other ways in which an array can be initialized.

```
int cgpa[3] = {9,8,8}
Float marks[] = {33,40}
```

= Arrays can be initialized while declaration.

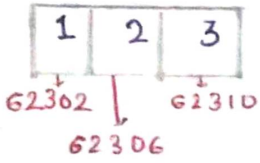
# Arrays in memory.

Consider this array:

```
int arr[3] = {1,2,3}
```

⟹ 1 integer = 4 bytes.

This will reserve 4×3 = 12 bytes in memory.
4 bytes for each integer.




⇒ arr in memory

# Pointer Arithmetic

A Pointer can be incremented to point to the next memory location of that type.

→ Consider this example.

```
int i = 32;
int *a = &i;
a++;
```

⇒ a = 87994

⇒ Now a = 87998

i

32

address → 87994

```
char a = 'A';
char *b = &a;
b++;
```

⇒ b = 87994

⇒ now b = 87998

```
float i = 1.7;
float *a = &i;
a++;
```

⇒ a = 87994

⇒ Now a = 87998

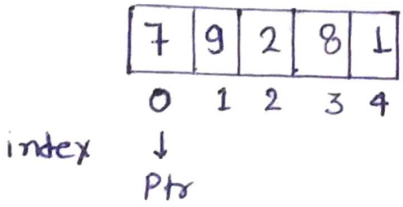
# following operations can be performed on a pointer:

1) Addition of a number to a pointer
2) Subtraction of a number from a pointer
3) Subtraction of two pointer from another
4) Comparision of two pointer variables.

Quick quiz - Try these operations on another variable by creating pointers in a separate program. Demonstrate all the four operation.

# Accessing Arrays using pointers

Consider the arrays.




If ptr points to index 0, ptr ++ will point to index 1 & so on ...

This way we can have an integer pointer pointing to first element of the array like this:

```
int * ptr = &arr[0];
ptr++;
  * ptr => will have 9 as its value
```

# Passing arrays to functions.

Arrays can be passed to the functions like this.

```
Print Array (arr, n);          => function Call
Void PrintArray (int* i, int n); => function prototype.
                or
Void printArray (int i[], int n);
```

# Multidimensional Arrays.

An array can be of 2 dimension / 3 dimension / n dimensions

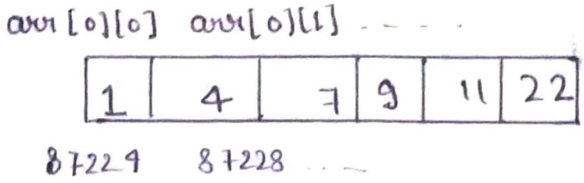
A 2 dimensional array can be defined as:

```
int arr[3][2] = { {1,4}
                  {7,9}
                  {11,22} };
```

We can access the elements of this array as arr[0][0], arr[0][1] & so on..
Value 1 Value 4

## 2-D Arrays in Memory

A 2d array like a 1-d array is stored in contiguous ~~memory~~ memory blocks like this:




Quick quiz : Create a 2-d array by taking input from the user. Write a display function to print the content of this 2-d array on the screen.

# Practice Set

1. Create an array of 10 numbers. Verify using pointer arithmetic that (ptr + 2) points to the ~~first~~ third element where ptr is a pointer pointing to the first element of the array.

2. If S[3] is a 1-D array of integer then *(S+3) refers to the third element :
   i) True
   ii) False
   iii) Depends.

3. Write a program to create an array of 10 integers and store multiplication table of 5 in it.

4. Repeat Problem 3 for a general input provided by the user using scanf.

5. Write a program containing a function which reverse the array passed to it

6. Write a program containing functions which counts the number of Positive integers in a array.

7. Create an array of size 3x10 containing multiplication tables of the number 2, 7 and 9 respectively.

8. Repeat problem 7 for a Custom input given by the user.

9. Create a three-dimentional array and print the address of its elements in increasing order.

# String

A String is a 1-D character array terminated by a null ('\0')

† This is null character

null character is used to denote string termination characters

are stored in contiguous memory locations

## → Initializing Strings

Since string is an array of characters, it can be intialized as follows

```
Char S[] = {'H','A','R','R','Y','\0'};
```

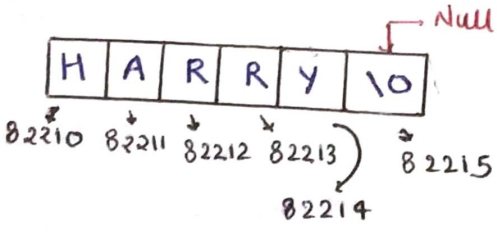
There is another shortcut for initializing strings in c language.

```
Char S[] = "HARRY";
```

⇒ In this case C adds a null character automatically.

## – Strings In Memory.

A String is stored just like an array in the memory as shown below




"Contiguous blocks in memory"

**Quick quiz** – Create a ~~using~~ string using " " and print its content using a loop.

## # Printing Strings

A String can be printed character by character using printf and %c

But there is another convenient way to print strings in C.

```
Char St[ ] = "HARRY";
printf ("%S", St);  ⇒ prints the entire string.
```

## # Taking string input from the user.

We can use %s with Scanf to take string input from the user:

```
Char st[50];
Scanf ("%s", &st);
```

Scanf automatically adds the null character when the enter key is pressed.

Note: 1) The string should be short enough to fit into the array

2) Scanf cannot be used to input multi-word strings with spaces

## # gets() and puts()

gets() is a function which can be used to receive a multi-word string.

```
Char St[30];
get (St); ⇒ The entered string is stored in St!
```

Multiple gets() calls will be needed for multiple strings

Likewise, puts can be used to output a string.

puts(st); ⇒ prints the string places the cursor on the next line

# Declaring a String using pointers.

We can declare string using pointers

```
Char* ptr = "Prathamm";
```

This tells the compiler to store the string in memory and assigned address is stored in a char pointer.

Note:

1) Once a string is defined using char st[ ] = "Prathamm", it cannot be initialized to something else.

2) A string defined using pointers can be reinitialized.

```
ptr = "Rohan";
```

# Standard library functions for strings

C provides a set of standard library functions for string manipulation.

Some of the most commonly using string functions are:

→ Strlen()

This function is used to count the number of character in the string excluding the null ('\0') character.

```
int length = strlen(st);
```

These functions are declared under <string.h> header file.

→ Strcpy()

This function is used to copy the content of second string into first string passed to it.

```
Char source[] = "Harry";
Char target[30];
Strcpy (target, source);  ⇒ target now Contain "Harry"
```

Target string should have enough capacity to store the source string.

→ **Strcat()**

This function is used to Concatenate two strings.

```
Char S1[11] = "Hello";
Char S2[] = "Pratham";

Strcat (S1, S2);  ⇒ S1 now contains "Hello Pratham"
                    <no space in between>
```

→ **Strcmp()**

This function is used to compare two strings.

It returns: 0 if strings are equal.

Negative value if first string's mismatching character's ASCII value is not greater than second string's corresponding mismatching character. It returns positive values otherwise.

```
Strcmp ("for", "Joke");  - positive value
Strcmp ("Joke", "for");  - negative value.
```

# Practice Set

1. Which of the following is used to appropriately read a multi-word string
   a) gets()  b) puts()  c) printf()  d) Scanf()

2. Write a program to take string as an input from the user using I.C and I.S confirm that the string are equal.

3. Write your own version of strlen function from <String.h>

4. Write a program to encrypt a string by adding 1 to the ascii value of its characters.

5. Write a function slice() to slice a string. It should change the original string such that it is now the sliced string. Take m and n as the start and ending position for slice

6. Write your own version of strcpy function from <string.h>

7. Write a program to decrypt the string encrypted using encrypt function is problem 6.

8. Write a program to count the occurence of a given character in a string.

9. Write a program to check whether a given character is present in a string or not.

# Structures.

Arrays and Strings ⇒ Similar data (int, float, char)

Structure can hold ⇒ dissimilar data

## ⇒ Syntax for Creating Structures.

A C structure can be created as follows:

```
Struct employee {
    int code;
    float Salary;
    char name[10];
};
```

⇒ This declares a new user defined data type!

→ Semi colon is important.

We can use this user defined data type as follows:

```
Struct employee e1;
```

⇒ Creating a structure variable

```
strcpy(e1.name, "Pratham");
e1.code = 100;
e1.Salary = 71.22;
```

So a structure in c is a collection of variables of different types under a single name.

Quick Quiz: Write a program to store the details of 3 employees from user defined data. Use the structure declared above.

## # Why use structures?

We can ~~use~~ create the data types in the employee structure separately but when the number of properties in a structure separately but when the no. of properties in a structure increases, it becomes difficult for us to create data variables without structures. In a nut shell:

a) Structure keep the data organized.

b) Structures make data management easy for the programmer.

— Array Structures.

Just like an array of integers, an array of floats and an array of Characters, we can create an array of Characters, We can create an array of Structures.

Struct employee facebook [100]; ⇒ An array of Structures

We can access the data using:

facebook [0].code = 100;
facebook [1].code = 101;
.... & so on

# Initializing Structures.

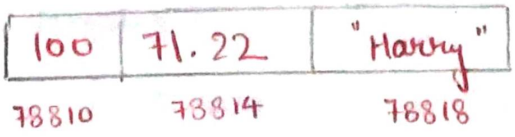
Structures can also be initialized as follows:

Struct employee harry = {100, 71.22, "Harry"};

Struct employee shubh = {0}; ⇒ All elements set to 0.

# Structures in memory:

Structures are stored in contiguous memory locations for the ~~same~~ structures e1 of type Struct employee, memory




In a Array of Structure, these employee instances are stored adjacent to each other.

→ Pointers to Structures.

A pointer to Structures can be created as follow

```
Struct employee *ptr;
ptr = &e1;
```

Now we can print Structure element using:

```
Printf ("%d", *(ptr).code);
```

→ Arrow operator

Instead of writing *(ptr).code, we can use arrow operator to access Structure properties as follows

*(ptr).code or ptr → code

Here → is known as the arrow operator.

→ passing structure to a function

A Structure can be passed to a function just like any other data type.

```
Void show(struct employee e);
```
⇒ function prototype.

Quick quiz: Complete this show function to display the content of employee

→ Typedef keyword.

We can use the typedef keyword to create an alias name for data types in c. typedef is more commonly used with structures.

```
Struct complex {
    float real;
    float img;
};
```
⇒ Struct complex $C_1$, $C_2$;
for defining complex numbers.

```
typedef Struct Complex {
        float real;
        float img;
        } Complex NO;
```

⇒ Complex NO $C_1$, $C_2$ ;
for defining Complex numbers

## Practice Set.

1. Create a two dimensional Vector using structures in C.
2. Write a function SumVector which returns the Sum of two Vectors passed to it. The Vectors must be two-dimensional.
3. twenty integers are to be Stored in memory. What will you prefer - Array or Structure ?
4. Write a program to illustrate the use of arrow operator → in C.
5. Write a program with a Structure representing a Complex number.
6. Create an array of 5 Complex numbers created in problem '5' and display them with the help of a display function. The Values must be taken as an input form the user.
7. Write problem 5's Structure using typedef Keyword.
8. Create a Structure representing a bank account of a Customer. What fields did you use and Why?
9. Write a Structure Capable of ~~String~~ storing data. Write a function to Compare those datas.
10. Solve problem a for time using typedef Keyword.

# File I/O

The Random access Memory is Volatile and its Content is lost once the program terminates In order to persist the data forever we use files.

A file is data stored in a storage device. A 'C' program can talk to the file by reading Content from it and writing Content to it.


→ FILE POINTER

The file is a structure which needs to created for opening the file.

A file pointer is a pointer to this structure of the file.

FILE pointer is needed for Communication between the file and the program.

A File pointer can be Created as follows:

```
FILE *ptr
ptr = fopen ("Filename.ext","mode");
```

→ File opening modes in C

C offers the programmers to select a mode for opening a file.

following modes are primarily used in C File I/O.

- "r" ⟶ open for reading
- "rb" ⟶ open for reading in binary

→ If the file does not exist, fopen returns NULL

"w" — open for writing → If the file exists, the contents will be overwritten

"wb" — open for writing in binary → If the file exists, the contents will be overwritten

"a" — open for append → If the file does not exists, it will be created.

→ Types of files.

There are two types of files:

1> Text files (.txt, .c)

2> Binary files (.jpg, .data)

— Reading a file.

A file can be opened for reading as follows:

```
FILE * ptr;
ptr = fopen("Pratham.txt","r");
int num;
```

Let us assume that "~~Hotey~~ Pratham.txt" contains an integer.

We can read that integer using:

```
fscanf(ptr, "%d", &num);
```

⇒ "fscanf is file counterpart of scanf."

This will read an integer from file in num variable.

Quick quiz: Modify the program above the check whether the file exists or not before opening the file.

→ Closing the file

It is very important to close the file after read or write. This is acheived using fclose as follows:

```
fclose(ptr);
```

This will tell the Compiler that we are done working with this file and the associated resources could be freed.

— Writing a file

We can write to a file in a ~~way~~ very similar manner like we read the file

```
FILE *fptr
fptr = fopen("Harry.text", "w");
int num = 432;
fprintf(fptr, "%d", num);
fclose(fptr);
```

→ fgetc() and fputc()

fgetc and fputc are used to read and write a character from/to a file

```
fgetc(ptr)          ⇒ used to read a character from file.
fputc('c', ptr);    ⇒ used to write character 'c' to the file.
```

— EOF : End of file

fgetc returns EOF when all the characters from a file have been read. So we can write a check link below to detect end of file.

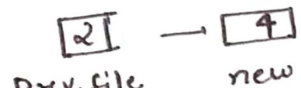
```
while(1){
    ch = fgetc(ptr);
    if(ch == EOF){          ⇒ When all the content of a file
        break;                has been read, break the loop!
    }
    // Code
}
```

# Practice Set

1. Write a program to read three integers from a file.
2. Write a program to generate multiplication table of a given number in text format. Make sure that the file is readable and well formatted.
3. Write a program to read a text file character by character and write its content twice in a separate file.
4. Take name and salary of two employees as input from the user and write them to a text file in the following format.

```
name 1 , 3300
name 2 , 7700
```

5. Write a program to modify a file containing an integer to double its value.




# Dynamic Memory Allocation

C is a language with some fixed rules of programming. for example: Changing the size of an array is not allowed.

→ **Dynamic Memory Allocation**

Dynamic memory allocation is a way to allocate memory to a data structure during the runtime. We can use DMA functions available in C to allocate and free memory during runtime.

→ **Function for DMA in C**

following function are available in C to perform Dynamic memory allocation.

1) malloc ()
2) calloc ()
3) free ()
4) realloc ()

→ malloc() function.

malloc stands for memory allocation. It takes number of bytes to be allocated as an input and returns a pointer of type Void.

Syntax:

```
ptr = (int *) malloc (30 * Size of (int))
```

(int *): Casting Void pointer to int
30 *: Space for 30 ints
Size of (int): returns Size of 1 int.

The expression returns a null pointer if they memory cannot be allocated.

Quick Quiz: Write a program to create a dynamic array of 5 floats using malloc().

→ Calloc() function.

Calloc stands for Continuous allocation. It initializes each memory block with a default value of 0.

Syntax:

```
ptr = (float *) calloc (30, Size of (float));
```

Allocates Contiguous space in memory for 30 blocks (floats)

If the space is not sufficient, memory allocation fails and a Null pointer is returned.

Quick Quiz: Write a program to create an array of Size n using Calloc where n is an integer entered by the user.

→ free() function.

We can use free() function to de allocated the memory.

The memory allocated using Calloc/malloc is not deallocated automatically.

Syntax:

```
free (ptr);
```

⇒ Memory of ptr is released.

Quick Quiz: Write a program to demonstrate the usage of free() with malloc()

## → realloc() function

Sometimes the dynamically allocated memory is insufficient or more than required.

realloc is used to allocate memory of new size using the previous pointer and size.

Syntax:

```
ptr = realloc (ptr, newsize);
ptr = realloc (ptr, 3 * sizeof(int));
```

⇒ ptr now points to this new block of memory capable of storing 3 integers.

## Pratice Set.

1. Write a program to dynamically create an array of size 6 capable to storing 6 integers.
2. Use the array in problem 1 to store 6 integers entered by the user.
3. Solve problem 1 using calloc()
4. Create an array dynamically capable of storing 5 integers. Now use realloc so that it can now store 10 integers.
5. Create an array of multiplication table of 7 upto 10 ($7 \times 10 = 70$). use realloc to make it store 15 numbers (from $7 \times 1$ to $7 \times 15$).
6. Attempt problem 4 using calloc().

**Project 1** Number Guessing Game

We will write a program that generates a random number and asks the player to guess it. If the player's guess is higher than the equal no., the program display "Lower no please". Similarly if the user guess is too low, the program print "Higher number please". When the user guess the correct no., the program display the no. of guesses the player used to arrive at the no.

Hint: use loops
use a random no. generator.

**Project 2** → Snake, Water, Gun.

Snake water, gun or Rock, paper Scissors is a game most of us have played during school time. Write a C program capable of playing this game with you.

Your program should be able to print the result after you choose Snak, water or gun.