Page No.
Date / /

C Language

What is C?

C is a programming language. It is a medium for us to communicate with computer.

⇒ C is one of the oldest and first programming languages.

Uses of C

C language is used to program a wide variety of system.

Some of the uses are as follows:-

i) Major parts of Windows, Linux and other program are written in C.

ii) It is used to write driver program for devices like tablets, printer etc.

iii) It is used to program embedded system where programs need to run faster in limited memory (Microwave, camera etc)

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

Variables, Constants & Keyword

Variable

A variable is a container which stores a 'value'. In Kitchen, we have containers storing rice, Dal, Sugar etc. Similar to that variables in C stores value of a constant. Eg.

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

Rules for naming variables in C

i) First character must be an alphabet or underscore(_)

ii) No commas, blanks allowed.

Date / /

iii) No special symbol other than (_) allowed.

iv) Variables names are case sensitive

* We must create meaningful variable names in our programs. This enhances readability of our program.

Constants

→ An entity whose value doesn't change is called as a constant.

⇒ A variable is an entity whose value can be changed

Types of Constants

Primarily, there are three types of constants:

i) Integer Constant → -1, 6, 7, 9
ii) Real Constant → -322.1, 2.5, 7.0
iii) Character Constant → 'a', '$', @
↳ Must be enclose within single inverted commas

Date / /

Keywords

These are reserved words, whose meaning is already known to the compiler. There are 32 keywords available in C.

auto	double	int	struct
break	long	else	switch
case	return	enum	typedef
char	register	extern	union
const	short	float	unsigned
continue	signed	for	void
default	size of	go to	volatile
do	static	if	while

Our first C program

```
#include <stdio.h>

int main(){
printf("Hello, I am PiRacoder");
return 0;
}
```

File: first.c

DOMS | Page No.
Date / /

Basic Structure of a C Program

All C programs have to follow a basic structure. A C program start with a main function and executes instructions present inside it.

* Each instruction is terminated with a semicolon (;)

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

i) Every program's execution starts from main() function.

ii) All the statements are terminated with a semicolon.

iii) Instructions are case-sensitive

iv) Instructions are executed in the same order in which they are written.

Comments

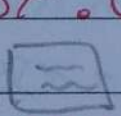
Comments are used to clarify something about the program in plain language. It is a way for us to add notes to our program. There are two types of comments in C.

i) Single-line comment : // This is Comment.
ii) Multi-line : /* This is
a multi line Comment */

* Comments in a C program are not executed and are ignored.

Compilation and Execution

First.c ⇒ C Compiler ⇒ First.exe



in VS Code

A Compiler is a computer program which converts a C program into machine language so that it can be easily understood by the computer

Page No.
Date / /

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 carry out 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

Types of variables

i) Integer variable ⇒ int a=3;

ii) Real variables → ~~int a = 7.7;~~
↳ wrong as 7.7 is real.

So, float a = 7.7;

iii) Character variable → 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.

Page No.
Date / /

Chapter 2: Instructions and operations.

A C program is a set of instruction. It is just like a recepie - which contains instructions to prepare a particular Dish.

Types of Instructions

i) Type declaration Instruction
ii) Arithmetic Instruction
iii) Control Instruction.

Type declaration Instruction

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

Other variations:

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

Date / /

float b=a+3; float a=1.1

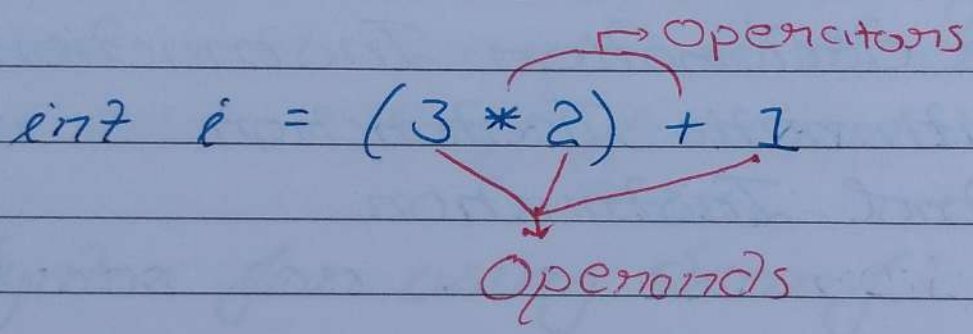
This Gives error! ↵

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; ✓ legal

int z; b*c = z; ✗ Illegal (Not allow)

% → Modular division operator

% → Returns the remainder

% → Cannot be applied on float

% → Sign is same as of

Page No.
Date / /

Numerator (-5 % 2 = -1)

i.e +5 % 2 = 1 and -5 % 2 = -1

+ → Positive, - → -ve

Note :-

i) No operator is assumed to be present.

int i = ab → Invalid

int i = a * b → valid

ii) There is no operator to perform exponentiation in C

⇒ However we can use pow(x,y) from <math.h>

→ (Explained later)

Type Conversion

In arithmetic operation between -

Int and Int → Int

Int and float → float

float and float → float

$\frac{5}{2} \rightarrow 2$ $\quad\quad$ $\frac{5.0}{2} \rightarrow 2.5$

$\frac{2}{5} \rightarrow 0$ $\quad\quad$ $\frac{2.0}{5} \rightarrow 0.4$

Important

Note

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 a = 8; a will store 8.0
8 → 8.0 (promotion to float)

Quick Quiz

Q. int k = 3.0/9 value of k? and why?

Ans 3.0/9 = 0.333 but since k is an int, it cannot store floats & value 0.33 is demoted to 0.

Page No.
Date / /

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 and associativity.

Operator precedence
The following table lists the operator priority in C

Priority	Operators
1st	* / %
2nd	+ -
3rd	=

- Operators of higher priority are evaluated first in the absence of parenthesis.

Date / /

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 instructions in C are:

i) Sequence Control Instruction
ii) Decision Control Instruction
iii) Loop Control Instruction
iv) Case Control Instruction

Page No.
Date / /

Ch:3 - Conditional Instructions

- Sometimes we want to watch comedy videos on Youtube if the day is Sunday
- Sometimes we order junk food if it is our friend's birthday in hostel.

⇒ All these are decisions which depends on a condition being met.

⇒ In C language too, we must be able to execute instructions on a condition being met.

Decision making Instructions in C
→ If-else statement
→ Switch statement

If-Else statement.

Syntax:

```
if(condition to be checked){
    statements-if-condition-true;
}
Else{
```

Statements - if condition - false;
}

Example:

```
int a = 23;

if (a > 18) {
    printf("You can drive\n");
}
```

• Note that Else block is not necessary but optional.

Relational Operators in C
Relational operators are used to evaluate conditions (true or false) inside the if statements.

⇒ Some example of relational operators are:-

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

== : Equals
>= : Greater than or equal to
!= : not equal to

Date / /

Important 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" respectively.

⇒ They are used to provide logic to our C program.

Usage of logical operators

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

Eg.: Let '1' be true & 0 be false then,
i) "1 and 0" is evaluated as false
ii) "1 and 1" is evaluated as true.

Eg. if {
// statements ; }
Else if {
// statement ; }
Else { // statement }

- It keeps on checking condition & where it is satisfied, then that statement is evaluated.

=) The last Else is executed only if all conditions fail.

=) Using if-else-if-else reduces indents

=) There can be any number of 'Else if'.

Operator precedence

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

ii) || → or → is true when at least one of the conditions is true.

Eg. (1 or 0 → 1) (1 or 1 → 1)

True False True

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

Eg.

! (3 == 3) → evaluates to false

! (3 > 30) → evaluates to true

As the number of conditions increases the level of indentation increases. This reduces readability. Logical operators come to rescue in such cases.

Else-if Clause

Instead of using multiple if statement, we can also use Else-if along with if thus forming an if-else-if-else ladder.

Date / /

Conditional Operators

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

condition? Express-if-true : expression-if-false

→ Ternary operators.

- ## Switch Case Control Instruction

Switch case is used when we have to make a choice between number of alternatives for a given variable.

```
Switch(integer-expression)
  {
    Case C1:
       Code;

    Case C2:
       Code;

    Case C3:
       Code;
```

⇒ C1, C2, C3 → Constant

⇒ Code → Any valid C code.

default :
code ;
}

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

- Points to Remember

i) We can use switch-case statements even by writing cases in any order of our choice (not necessarily ascending).

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

iii) A switch can occur within another but in practice this is rarely done.

Date / /

Ch-4 Loop Control Instruction

Why loops?

Sometimes we want our programs to execute few set of instructions over and over again.

Eg. Printing 1 to 100, Print first 100 even numbers 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:

i) While loop
ii) do-while loop
iii) for loop

We will look into these one by one.

Page No.
Date / /

Do-while loop

Syntax looks like this:

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

do-while loop works very similar to while loop.

While → Checks the condition & then executes the code.

Do-while → Executes the code & then checks the condition.

- Do-while loop 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

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 prints it.

* printf("i-- = %d", i--);

This first prints i and then decrements it

⇒ +++ Operator doesn't exist
↳ (Important)

⇒ += is compound assignment Operator just like -=, *=, /= & %=
↳ (Also important)

DOMS | Page No.
Date / /

i) While loop

```
while (condition is true) {
        // code
        // code
}
```

⇒ The block keeps executing as long as the condition is true

Eg.

```
int i = 0

while (i < 10) {
    printf("The value of i is %d", i);
    i++;
}
```

Note: If the condition never become false, the while loop keeps getting executed. Such a loop is known as an infinite loop.

Quiz.
Write a program to print natural numbers from 10 to 20 when initial loop counter i is initialized to 0.

Date / /

For Loop

- Syntax

```
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.

⇒ Example:

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

Output:
0
1
2

Date / /

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.

This loop runs in following steps:

i) i is initialized to 5.
ii) The condition "i" (0 or non0) is tested.
iii) The code is executed
iv) i is decremented
v) Condition i is checked and code is executed if its not 0.
vi) And so on until i is non 0.

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.

Date / /

Example-

```
for(i=0; i<100; i++){
    printf("%d\n", i);
    if(i==5){ break; }
}
```

Output: 0 1 2 3 4 5
and not 0 to 100

- The execution of loop stops when i is equal to 5

Continue Statement

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

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

Example

```
int skip = 5;
int i=0;
```

```
while(i<10) {
    if ( i != skip)
        continue;
Else
    printf(" %d", i);
    }
```

Output ⇒ 5
and not 0 - - 9

Note

i) Sometimes the name of the variable might not indicate the behaviour of the program.
ii) Break statement completely exits the loop.
iii) Continue statement skips the particular iteration of the loop.

Quick Quiz

i) Write a program to check whether a given number is prime or not using loops.

Date / /

Ch-5 Functions 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.
Function is a way to break our code into chunks so that it is possible for a programmer to reuse them.

What is a function

⇒ A function is a block of code which performs a particular task.
⇒ It can be used any number of times in a given program.

Example and Syntax of function

```
#include<stdio.h>

i) void display();  ⇒ Function Prototype

ii) int main(){
      int a;        ⇒ Function call
      display();
```

DOMS | Page No.
Date / /

Function definition
This part contains the exact set of instructions which are executed during the function call. When a function is called from main(), the main function falls asleep and gets temporily 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 3 function:
i) Goodmorning function which prints "Good morning"
ii) Goodafternoon function which prints "Good afternoon".
iii) Good night function which prints "Good night".

main() should call all these in order 1 → 2 → 3.

```
return 0;
}
```

iii)
```
void display(){
    printf("Hello, I am Pikacoder");
}
```
→ Function definition

Function Prototype
Function prototype is a way to tell the computer compiler about the function we are going to define in the program.
Here 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.

Date / /

Important points

→ Execution of a C program starts from main().

→ A C program can have more than one function.

→ Every function gets called directly or indirectly from main().

→ There are two types of functions in C. Lets talk about them.

Types of function

i) Library functions → Commonly require function grouped together in a library file on disk.

ii) User defined function → These are the functions declared and defined by the user.

Page No.
Date / /

Passing values to functions

We can pass values to a function and can 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;
    c = a+b;
    return c;
}
```

⇒ 'a' and 'b' are parameters.

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

Here 2 and 3 are arguments.

Date / /

int d = Sum(2,3); ⇒ d becomes 5

Note

i) Parameters are the values or variable placeholders in the function definition. Ex a & b.

ii) Arguments are the actual values passed to the function to make a call. Eg. 2 and 3.

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

iv) 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;
    return 0;
}
```

⇒ Misnomer

Change is a function which changes a to 77 number if we call it from main

Date / /

like this.

```
int b = 22
change(b);
printf("b is %d", b);
```

⇒ The value of b remains 22.

↳ prints "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 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:
A very good example of recursion is factorial.

Page No.
Date / /

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 number 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;
    }
```

This is a program to calculate factorial using recursion.

Note: Recursion is sometimes the most direct way to code an algorithm.

Date / /

How does the above code work?

factorial(5)

5 x factorial(4)

5 x 4 x factorial(3)

5 x 4 x 3 x factorial(2)

5 x 4 x 3 x 2 x factorial(1)

→ 5 x 4 x 3 x 2 x 1

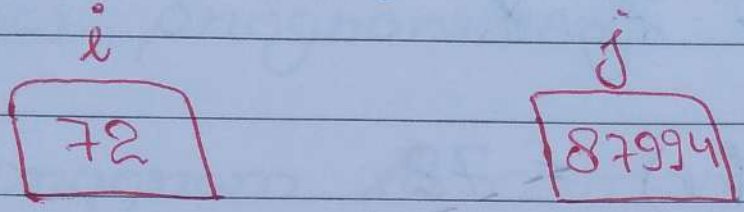
Important point

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

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

Pointers

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



Address: 87994 Address: 87998

Here, j is a pointer
j points to i

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

If you refer to the diagram above:-

&i ⇒ 87994
&j ⇒ 87998

Format specifier for printing pointer address is '%u'

The 'value at address' operator (*)
The value at address or * operator is used to obtain the value present at 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 pointer of char, float etc.

int * ch_ptr; ⇒ Pointer of integer
char * ch_ptr; ⇒ Pointer of character
float *ch_ptr; ⇒ Pointer to float

* Although its a good practice to use meaningful variable names, we should be very careful while reading and working on programs from fellow programmers.

A program to demonstrate pointer

```
#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 i = %d\n", *(&i));
printf("Value i = %d\n", *j);
  return 0;
}
```

Output:

```
Add i = 87994
Add i = 87994
Add j = 87998
Value i = 8
```

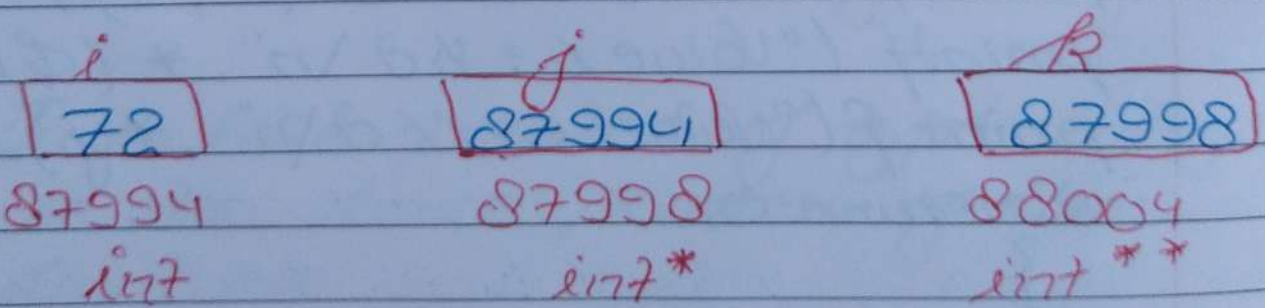
Date / /

value i = 8
Value i = 8

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

Pointer to a pointer
Just like j is pointing to i on 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 l of type int*** to store the address of k. We mostly use int* and int**

Date / /

sometimes in real world programs.

Types of function call

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

i) call by value ⇒ Sending the value of arguments

ii) call by reference ⇒ Sending the address of arguments

Call by value

Here the value of the arguments are passed to the function. Consider the 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 values 3 and 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

Date / /

In C we usually make a call by value.

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

Now. Since the address are passed to the function, the function 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          => a is 3 and b is 4
  ...
  Swap(a, b)
  return 0;        => Now a is 4 and
  }                     b is 3
```

Quick Quiz

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

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

Date / /

Ch-7 Arrays

An array is a collection of similar element.

one variable: Capable of storing multiple values.

Syntax

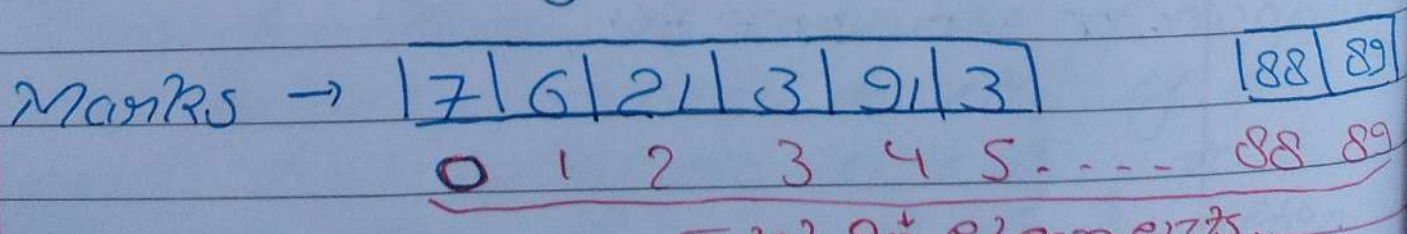
The syntax of declaring an array looks like this.

```
int marks[90];  ⇒ Integer array
Char name[20];  ⇒ Character array
float percentile[90];  ⇒ float array
```

The values can 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



* Numbering starts from 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 array.

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

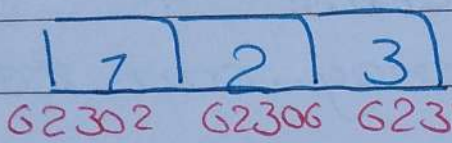
int cgpa[3] = {9, 8, 8}
⇒ Arrays can be initialized while declaration

Float marks[] = {33, 40}

Arrays in memory
Consider this array:

int arr[3] = {1, 2, 3} ⇒ 1 integer is equal to 4 bytes.

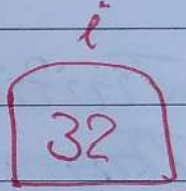
This will reserve 4x3=12 bytes in memory 4 bytes for each integer



⇒ arr in memory

Pointer Arithmetic

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



Example:

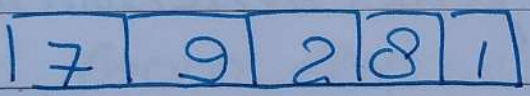
```
int i = 32;
int *a = &i        ⇒ a = 87994
a++;               ⇒ now a = 87998

Char a = 'A';
Char *b = &a;      ⇒ b = 87994
b++;               ⇒ Now b = 87995

float i = 1.7;
float *a = &i;     ⇒ Address of i or a = 87994
a++;               ⇒ Now a = 87998
```

Date / /

Following operations can be perform on a pointer;

i) Addition of a number to a pointer
ii) Subtraction of a number from a pointer.
iii) Subtraction of one pointer from another.
iv) Comparison of two pointer variables.

Accessing Arrays using Pointers
Consider the array



index: 0 1 2 3 4

↑
ptr

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]; ⇒ or simply arr
ptr++;
*ptr ⇒ will have 9 as its value.

Page No.
Date / /

Passing arrays to functions

Arrays can be passed to the function like this

```
printArray(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 dimension.

A two-dimension 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....

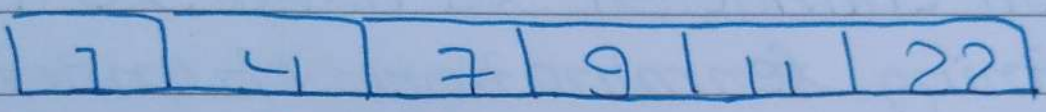
arr[0][0] ⇓ Value = 1

arr[0][1] ⇓ Value = 4

2-D arrays in Memory

⇒ A 2d array, like a 1-d array is stored in contiguous memory block like this:

arr[0][0] arr[0][1] ----



87224 87228 ....

Quick Quiz

Q.1 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.

Q.2 Write a program containing function which counts the number of positive integers in an array.

Date / /

Ch-8 Strings

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 location.

Initializing String

Since string is an array of character, it can be initialized as follows:

```
char S[] = { 'P', 'I', 'K', 'A', '\0' };
```

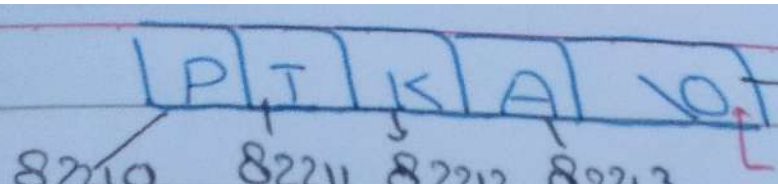
There is an another shortcut for initializing strings in C language.

```
char S[] = "PIKA";
```

⇒ 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

Quiz → Create a 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 strings

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:

i) The string should be short enough to fit into the array.

ii) 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];

gets(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.

Date / /

Declaring a string using pointers

We can declare strings using pointers.

```
Char *ptr = "Harry";
```

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

Note

i) Once a string is defined using Char st[] = "Harry", it cannot be reinitialized to something Else.

ii) 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 used string functions are:

Date / /

Strlen()

This function is used to count the number of characters 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 contains "Harry"

Target string should have enough capacity to store the source string

strcat()

This function is used to concatenate two string.

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

Strcat(S1, S2);
```

⇒ S1 now contains "HelloHarry"
(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 ASII value is not greater than second string's corresponding mismatching character.
- ⇒ It returns positive value otherwise.

```
Strcmp("For", "Joke");  ⇒ Positive value
Strcmp("Joke", "For");  ⇒ Negative value
```

Page No.
Date / /

Ch-9 Structures

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

Structures 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!

↳ Semicolon is important

We can use this user defined data type as follows:

```
struct employee e1;
strcpy(e1.name, "Harry");
e1.code = 100;
e1.salary = 71.22;
```

↑ creating a structure variable

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

Why use Structure?
We can create the data types in the employee structure separately but when the number of properties in a structure increases; it becomes difficult for us to create data variables without structures.

In Short
i) It keep the data organised.
ii) It make data management simple for a programmer.

Array of Structures
To create an array of structures

```
struct employee facebook[100];
```
↳ An array of structures

We can access data by using:
```
facebook[0].code=100;
facebook[1].code=101;
```
... and so on

Page No.
Date / /

Initializing Structures

Structures can also be initialized as follows:

```
Struct employee Harry = {100, 71.22,
                         "Harry"};
Struct employee Pika = {0};
```

→ All elements set to 0.

Structures in Memory

Structures are stored in contiguous memory locations. For the structure e1 of type Struct employee, memory layout looks like this:

100	71.22	"Harry"
Address → 78810	78814	78818

In an array of structures these employee instances are stored adjacent to each other.

Pointer to Structures

A pointer to structure can be created as follows:

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

Now we can print structure elements 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 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

DOMS | Page No.
Date / /

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 C1, C2; for defining Complex numbers

```
typedef struct Complex {
    float real;
    float img;
} Complex number;
```

⇒ Complex no. C1, C2; for defining Complex number

Quiz

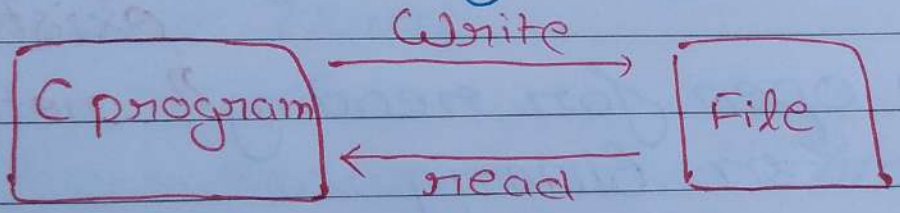
Q.1 Write a structure capable of storing date. Write a function to compare those dates.

Page No.
Date / /

Ch-10 File I/O

The RAM is volatile and its content is lost once the program terminate 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
A file is a structure which needs to be 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
Following modes are primarily used in C file I/O

"r" → open for reading → If the file doesn't exist, fopen returns NULL

"rb" → open for reading in Binary

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

"wb" → Open for writing in Binary

"a" → Open for append → If the file does not exist, it will be created

Page No.
Date / /

Types of files

There are two type of files:

i) Text files (.txt, .c)
ii) Binary files (.jpg, .dat)

Reading a file

A file can be opened for reading as follows:

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

Let us assume that "Harry.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.

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 to a file

We can write to a file in a very similar manner like we read file.

```
FILE *fptr;
fptr = fopen("Manny.txt", "w");

int num = 432;
fprintf(fptr, "%d", num);

fclose(fptr);
```

Date / /

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 like below to detect end of file.

```
while(1){
  ch = fgetc(ptr);
  if(ch == EOF){
      break;
  }
  //code
}
```

⇒ When all the content of a file has been read, break the loop.

Chapter 11 - 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

It 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.

Functions for DMA in C

i) malloc()
ii) calloc()
iii) free()
iv) realloc()

Malloc() function

Malloc stands for memory allocation It keep number of byte to be allocated as an input and returns a pointer of type void.

DOMS | Page No.
Date / /

Syntax:

```
ptr = (int*) malloc (30 * sizeof(int))
```

- (int*): casting void pointer to int
- 30: space for 30 ints
- sizeof(int): returns size of 1 int

The expression returns a null pointer if the memory can't be allocated.

Calloc() function

=) Calloc stands for continuous allocation.

=) It initializes each memory block with a default value of 0

Syntax:

```
ptr = (float*) calloc (30, sizeof (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.

DOMS | Page No.
Date / /

Free() function

We can use free() function to deallocate the memory.

Syntax:

free(ptr); ⇒ Memory of ptr is released

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, new size);

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

↓

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