# Basics

Basic syntax and functions from the C programming language.

## Boilerplate Code

#include<stdio.h> int main() { return(0); }

## printf function

It is used to show output on the screen

printf("Hello World!")

## scanf function

It is used to take input from the user

scanf("placeholder", variables)

# Comments

A comment is the code that is not executed by the compiler, and the programmer uses it to keep track of the code.

**Single line comment**

// It's a single line comment

## Multi-line comment

/\* It's a multi-line comment

\*/

# Data types

The data type is the type of data

## Character type

Typically a single octet(one byte). It is an integer type

char variable\_name;

## Integer type

The most natural size of integer for the machine

int variable\_name;

## Float type

A single-precision floating-point value

float variable\_name;

## Double type

A double-precision floating-point value

double variable\_name;

## Void type

Represents the absence of the type

void

# Escape Sequences

It is a sequence of characters starting with a backslash, and it doesn't represent itself when used inside string literal.

## Alarm or Beep

It produces a beep sound

\a

## Backspace

It adds a backspace

\b

**Form feed**

\f

## Newline

Newline Character

\n

**Carriage return**

\r

## Tab

It gives a tab space

\t

## Backslash

It adds a backslash

\\

## Single quote

It adds a single quotation mark

\'

## Question mark

It adds a question mark \?

**Octal No.**

It represents the value of an octal number

\nnn

**Hexadecimal No.**

It represents the value of a hexadecimal number

\xhh

## Null

The null character is usually used to terminate a string

\0

# Conditional Instructions

Conditional statements are used to perform operations based on some condition.

## If Statement

if (/\* condition \*/) {

/\* code \*/

}

## If-else Statement

if (/\* condition \*/) {

/\* code \*/ } else{ /\* Code \*/

}

## if else-if Statement

if (condition) { // Statements; }

else if (condition){ // Statements; } else{ // Statements

}

## Switch Case Statement

It allows a variable to be tested for equality against a list of values (cases).

switch (expression) { case constant-expression:

statement1; statement2; break; case constant-expression:

statement; break; ...

default: statement;

}

# Iterative Statements

Iterative statements facilitate programmers to execute any block of code lines repeatedly and can be controlled as per conditions added by the programmer.

## while Loop

It allows execution of statement inside the block of the loop until the condition of loop succeeds.

while (/\* condition \*/) {

/\* code \*/

}

## do-while loop

It is an exit controlled loop. It is very similar to the while loop with one difference, i.e., the body of the do-while loop is executed at least once even if the expression is false

do {

/\* code \*/

} while (/\* condition \*/);

## for loop

It is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

for (int i = 0; i < count; i++) {

/\* code \*/

}

## Break Statement

break keyword inside the loop is used to terminate the loop

break;

## Continue Statement

continue keyword skips the rest of the current iteration of the loop and returns to the starting point of the loop

continue;

# Functions & Recursion

Functions are used to divide an extensive program into smaller pieces. It can be called multiple times to provide reusability and modularity to the C program.

## Function Definition

return\_type function\_name(data\_type parameter...){ //code to be executed

}

## Recursion

Recursion is when a function calls a copy of itself to work on a minor problem. And the function that calls itself is known as the Recursive function.

void recurse() { ... .. ... recurse(); ... .. ...

}

# Pointers

Pointer is a variable that contains the address of another variable,

**Declaration**

datatype \*var\_name;

# Arrays

An array is a collection of data items of the same type.

**Declaration**

data\_type array\_name[array\_size];

**Accessing element**

int variable\_name = array[index];

# Strings

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

**Declaration**

char str\_name[size];

## gets() function

It allows you to enter multi-word string

gets("string");

## puts() function

It is used to show string output

puts("string");

## String Functions strlen()

It is used to calculate the length of the string

strlen(string\_name);

## strcpy() function

It is used to copy the content of second-string into the first string passed to it

strcpy(destination, source);

## strcat() function

It is used to concatenate two strings

strcat(first\_string, second\_string);

## strcmp() function

It is used to compare two strings

strcmp(first\_string, second\_string);

# Structures

The structure is a collection of variables of different types under a single name. Defining structure means creating a new data type.

## Structure syntax

struct structureName {

dataType member1; dataType member2; ...

};

## typedef keyword

typedef function allows users to provide alternative names for the primitive and user-defined data types.

typedef struct structureName { dataType member1; dataType member2;

...

}new\_name;

# File Handling

A set of methods for handling File IO (read/write/append) in C language

**FILE pointer**

FILE \*filePointer;

## Opening a file

It is used to open file in C.

filePointer = fopen(fileName.txt, w)

## fscanf() function

It is used to read the content of file.

fscanf(FILE \*stream, const char \*format, ...)

## fprintf() function

It is used to write content into the file.

fprintf(FILE \*fptr, const char \*str, ...);

## fgetc() function

It reads a character from a file opened in read mode. It returns EOF on reaching the end of file.

fgetc(FILE \*pointer);

## fputc() function

It writes a character to a file opened in write mode

fputc(char, FILE \*pointer);

## Closing a file

It closes the file.

fclose(filePointer);

# Dynamic Memory Allocation

A set of functions for dynamic memory allocation from the heap. These methods are used to use the dynamic memory which makes our C programs more efficient **malloc() function**

Stands for 'Memory allocation' and reserves a block of memory with the given amount of bytes.

ptr = (castType\*) malloc(size);

## calloc() function

Stands for 'Contiguous allocation' and reserves n blocks of memory with the given amount of bytes.

ptr = (castType\*)calloc(n, size);

## free function

It is used to free the allocated memory.

free(ptr);

## realloc() function

If the allocated memory is insufficient, then we can change the size of previously allocated memory using this function for efficiency purposes

ptr = realloc(ptr, x);