Programming in C

By

Dr Ramkumar Krishnamoorthy

kramdharma@gmail.com

Additional Features

Unit V

Contents

File Concepts

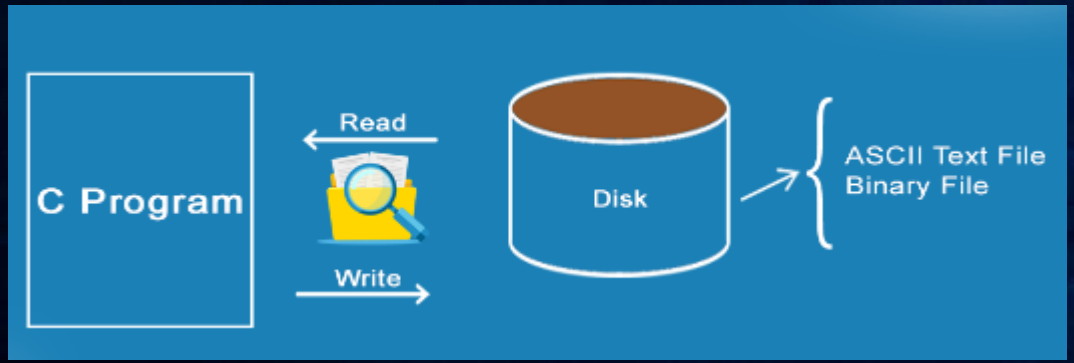
A collection of records (or) group of records.

Record means Collection of fields(or)collection of attributes

NAME	AGE	CLASS	PERCENTAGE
AA	12	1ST	100
BB	13	2ND	98
CC	14	3RD	97
DD	15	4TH	99
EE	16	5TH	100

File Handling

- File Handling is a mechanism for storing of data in a file using a program.
- In C programming language, the programs store results, and other data of the program to a file using file handling in C.
- Also, users can extract/fetch data from a file to work with it in the program.

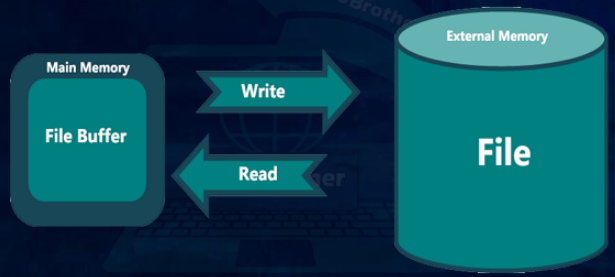


File Concepts

- In programming, users may require some specific input data to be generated several numbers of times. Sometimes, it is not enough to only display the data on the console.
- The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again.

File handling in C enables users to create, update, read, and delete the files stored on the local file system through users C program. The following operations can be performed on a file.

- Creation of the new file
- Opening an existing file
- Reading from the file
- Writing to the file
- Deleting the file



File Accessing (Operations) - Functions

No.	Function	Description
1	fopen()	opens new or existing file
2	fprintf()	write data into the file
3	fscanf()	reads data from the file
4	fputc()	writes a character into the file
5	fgetc()	reads a character from file
6	fclose()	closes the file
7	fseek()	sets the file pointer to given position
8	fputw()	writes an integer to file
9	fgetw()	reads an integer from file
10	ftell()	returns current position
11	rewind()	sets the file pointer to the beginning of the file

File Operation Modes

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode
rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode

File Creation

The **fopen()** function is used to create a new file or open an existing file in C.

The fopen function is defined in the **stdio.h** header file.

The syntax for creation of a new file or opening a file

```
file = fopen(“file_name”, “mode”)
```

This is a common syntax for both opening and creating a file in C

“file_name” − It is a string that specifies the name of the file that is to be opened or created using the fopen method.

mode: It is a string (usually a single character) that specifies the mode in which the file is to be opened.

fopen() – File Open

We must open a file before it can be read, write, or update. The fopen() function is used to open a file. The syntax of the fopen() is given below.

FILE *fopen(const char * filename, const char * mode);
The fopen() function accepts two parameters:

The file name (string). If the file is stored at some specific location, then we must mention the path at which the file is stored. For example, a file name can be like "c://some_folder/some_file.ext".
The mode in which the file is to be opened. It is a string.

getc() – Function in File

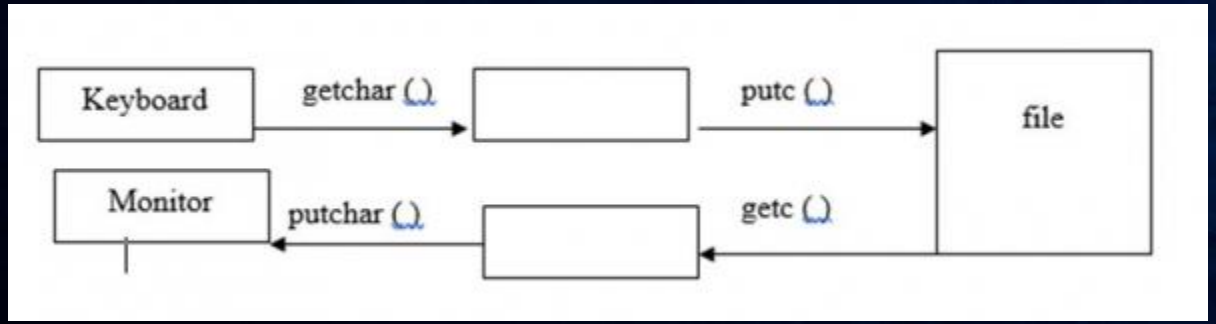
getc( ) function is used to read a character from file

The syntax for getc() function is as follows −

char getc (FILE *fp);

For example,

```
FILE *fp;
char ch;
ch = getc(fp);
```



putc() – Function in File

putc( ) function is used for writing a character into a file.

The syntax for putc() function is as follows −

putc (char ch, FILE *fp);

For example,

```
FILE *fp;
char ch;
putc(ch, fp);
```

Example

```
#include<stdio.h>
void main ()
{
   char c;
   printf("Enter character: ");
   c = getc(stdin);
   printf("Character entered: ");
   putc(c, stdout);
}
```

Output

Enter character: a
Character entered: a

Example for getc() and putc()

```
#include<stdio.h>
void main()
{
   char ch;
   FILE *fp;
   fp=fopen("std1.txt","w");   //opening file in write mode
   printf("enter the text.press cntrl Z:");
   while((ch = getchar())!=EOF)
   {
      putc(ch,fp);       // writing each character into the file
   }
   fclose(fp);

   fp=fopen("std1.txt","r");
   printf("text on the file:");
   while ((ch=getc(fp))!=EOF) // reading each character from file
   {
      putchar(ch);              // displaying each character on to the screen
   }
   fclose(fp);
}
```

Output

```
enter the text.press cntrl Z:
Hi Welcome to C World
Here I am Presenting Question and answers in C Programming
Language
^Z
text on the file:
Hi Welcome to C World
Here I am Presenting Question and answers in C Programming
Language
```

fprintf() function

Writing File : **fprintf() function**
The fprintf() function is used to write set of characters into file. It sends formatted output to a stream.

Syntax:

int fprintf(FILE *stream, const char *format [, argument, ...])

Example

```
#include <stdio.h>
main()
{
   FILE *fp;
   fp = fopen("file.txt", "w");
   fprintf(fp, "Hello file by fprintf...\n");
   fclose(fp);
}
```

fscanf() function

Reading File : **fscanf()** function
The fscanf() function is used to read set of characters from file.
It reads a word from the file and returns EOF at the end of file.

Syntax:

int fscanf(FILE *stream, const char *format [, argument, ...])

Example

```
#include <stdio.h>
void main()
{
   FILE *fp;
   char buff[255];
   fp = fopen("file.txt", "r");
   while(fscanf(fp, "%s", buff)!=EOF)
    {
          printf("%s ", buff );
     }
   fclose(fp);
}
```

Output: Hello file by fprintf...

Example program for File Read

```
#include<stdio.h>
#include<conio.h>
void main()
{
        FILE *ptr;
        int c;
        clrscr();
        ptr=fopen(“total.c”,”r”);
        c=getc(ptr);
        while(c!=EOF)
        {
                putchar(c);
                c=getc(ptr);
        }
        fclose(ptr);
        getch();
}
```

Example program for File Write

```
#include<stdio.h>
#include<conio.h>
void main()
{
        FILE *ptr;
        clrscr();
        ptr=fopen(“sai.txt”,”w”);
        fprintf(ptr,”\n %s”,”HELLO HOW ARE YOU!”);
        fprintf(ptr,”\n%d”,15000);
        fprintf(ptr,”\n%f”,12.9090);
        fclose(ptr);
        getch();
}
```

C Preprocessor

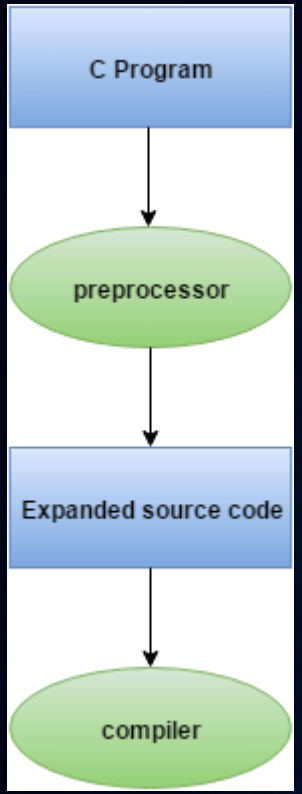
C Preprocessor is just a text substitution tool and it instructs the compiler to do required pre-processing before the actual compilation.

It preprocesses instructions before it sends to the compiler

All preprocessor commands begin with a hash symbol (#).

#define Substitutes a preprocessor macro.	**#ifndef** Returns true if this macro is not defined.
#include Inserts a particular header from another file.	**#if** Tests if a compile time condition is true.
#undef Undefines a preprocessor macro.	**#else** The alternative for #if.
#ifdef Returns true if this macro is defined.	**#elif** #else and #if in one statement.

C Preprocessor



#endif

Ends preprocessor conditional.

#error

Prints error message on stderr.

#pragma

Issues special commands to the compiler, using a standardized method

C Preprocessor

In programming terminology, a preprocessor is nothing but a System Software that performs the processing of a high-level language before compilation by translating the source code written in a high-level language to object code written in the machine-level language, that is easily understood by the compiler

#inlcude

The **#include** preprocessor directive is used to paste code of given file into current file.

It is used include system-defined and user-defined header files. If included file is not found, compiler renders error.

By the use of #include directive, user provides information to the preprocessor where to look for the header files.

There are two variants to use #include directive.

#include <filename>

#include "filename“

The #include <filename> tells the compiler to look for the directory where system header files are held.

The #include "filename" tells the compiler to look in the current directory from where program is running.

#inlcude

Example

```
#include<stdio.h>
 int main()
{
   printf("Hello C");
   return 0;
 }
```

Output

Hello C

#define

The **#define** preprocessor directive is used to define constant or micro substitution.

It can use any basic data type.

Syntax

```
#define token value
```

Example

```
#include <stdio.h>
#define PI 3.14
void main()
 {
      printf("%f", PI);
 }
```

Output

3.140000

```
#include <stdio.h>
#define MIN(a,b) ((a)<(b)?(a):(b))
void main()
{
   printf("Minimum between 10 and 20 is: %d\n", MIN(10,20));
}
```

Output: Minimum between 10 and 20 is: 10

#undef

The #undef preprocessor directive is used to undefine the constant or macro defined by #define.

Syntax:

```
#undef token
```

Simple example to define and undefine a constant.

Example

```
#include <stdio.h>
#define PI 3.14
#undef PI
void main()
{
    printf("%f",PI);
}
```

Output

Compile Time Error: 'PI' undeclared

#undef Example

```
#include <stdio.h>
#define number 15
int square=number*number;
#undef number
void main()
{
   printf("%d",square);
}
```

Output

225

Conditional Directives

The conditional directives are:

#ifdef - If this macro is defined

#ifndef - If this macro is not defined

#if - Test if a compile time condition is true

#else - The alternative for #if

#elif - #else an #if in one statement

#endif - End preprocessor conditional

#if directive

The #if preprocessor directive takes condition in parenthesis, **if condition is true, then the statements given between #if and #else will get execute.**

If condition is false, then statements given between #else and #endif will get execute

Syntax

```
#if(condition)
    ----------
    ----------
#else
    ----------
    ----------
#endif
```

```
#include<stdio.h>
#define MAX 45
void main()
{
    #if MAX > 40
        printf("Yes, MAX is greater then 40.");
    #else
        printf("No, MAX is not greater then 40.");
    #endif
}

Output :

    Yes, MAX is Greater then 40.
```

#elif directive

The #elif preprocessor directive is similar to if-else ladder statement. It is used for checking multiple conditions, if the first condition will not satisfy, compiler will jump to #else block and check other condition is true or not and so on.

Syntax

```
#if(condition)
    ----------
    ----------
#elif(condition)
    ----------
    ----------
#elif(condition)
    ----------
    ----------
#else
    ----------
    ----------
#endif
```

#elif directive

Example

```
#include <stdio.h>
#define checker 5
void main()
{
   #if checker <= 3
   printf("This is the if directive block.\n");

   #elif checker > 3
   printf("This is the elif directive block.\n");

   #endif
}
```

Output

This is the elif directive block

#ifdef directive

The #ifdef preprocessor directive is used to check whether the macro-name is previously defined or not, if defined then the statements given between #ifdef and #else will get execute

Syntax

```
#ifdef macro-name
    ----------
    ----------
#else
    ----------
    ----------
#endif
```

Example

```
#include<stdio.h>
#define MKS 65
void main()
{

    #ifdef MONTH
        printf("\nMONTH is defined.");

    #else
        printf("\nMONTH is not defined.");

     #endif

 }
```

Output :

```
    MONTH is not defined.
```

#ifndef directive

The #ifndef preprocessor directive is used to check whether the macro-name is previously defined or not, if not defined then the statements given between #ifndef and #else will get execute.

Syntax

```
#ifndef macro-name
    ----------
    ----------
#else
    ----------
    ----------
#endif
```

Example

```
#include<stdio.h>
#define MKS 65
void main()
{
    #ifndef MAX
        printf("\nMAX is not defined.");
    #else
        printf("\nMAX is defined.");
    #endif
}
```

Output :

MAX is not defined.

Standard Library Header Files in C

stdio.h
Input/Output functions

conio.h
Console Input/Output functions

stdlib.h
General utility functions

math.h
Mathematics functions

string.h
String functions

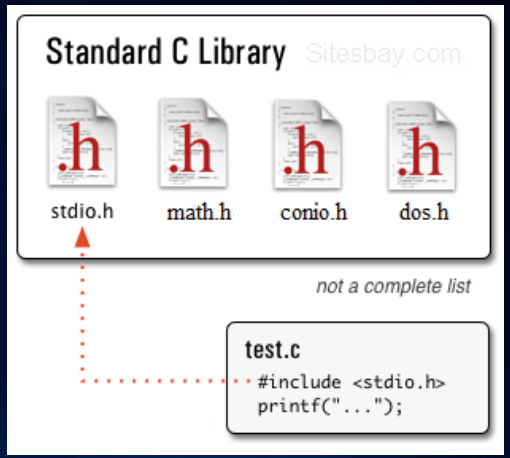
ctype.h
Character handling functions

time.h
Date and time functions

float.h
Limits of float types

limits.h
Size of basic types

wctype.h
Functions to determine the type contained in wide character data.



String Functions

No.	Function	Description
1)	strlen(string_name)	returns the length of string name.
2)	strcpy(destination, source)	copies the contents of source string to destination string.
3)	strcat(first_string, second_string)	concats or joins first string with second string. The result of the string is stored in first string.
4)	strcmp(first_string, second_string)	compares the first string with second string. If both strings are same, it returns 0.
5)	strrev(string)	returns reverse string.
6)	strlwr(string)	returns string characters in lowercase.
7)	strupr(string)	returns string characters in uppercase.

Mathematical Functions

No.	Function	Description
1)	ceil(number)	rounds up the given number. It returns the integer value which is greater than or equal to given number.
2)	floor(number)	rounds down the given number. It returns the integer value which is less than or equal to given number.
3)	sqrt(number)	returns the square root of given number.
4)	pow(base, exponent)	returns the power of given number.
5)	abs(number)	returns the absolute value of given number.

Date and Time Functions

Date and Time predefined functions are

No.	Macro	Description
1	_DATE_	represents current date in "MMM DD YYYY" format.
2	_TIME_	represents current time in "HH:MM:SS" format.
3	_FILE_	represents current file name.
4	_LINE_	represents current line number.
5	_STDC_	It is defined as 1 when compiler complies with the ANSI standard.

Date and Time Functions

Example

```c
#include<stdio.h>
int main(){
  printf("File :%s\n", __FILE__ );
  printf("Date :%s\n", __DATE__ );
  printf("Time :%s\n", __TIME__ );
  printf("Line :%d\n", __LINE__ );
  printf("STDC :%d\n", __STDC__ );
  return 0;
}
```

```
File :simple.c
Date :Dec 6 2015
Time :12:28:46
Line :6
STDC :1
```

Date and Time Functions

Example

```
#include <dos.h>
#include <stdio.h>
void main()
{
    struct date dt;
     getdate(&dt);
     printf("System's current date\n");
     printf("%d/%d/%d", dt.da_day, dt.da_mon, dt.da_year);
}
```

```
System's current date
18/4/2019_
```

Credits

- https://www.javatpoint.com/file-handling-in-c
- https://www.tutorialspoint.com/cprogramming/c_preprocessors.htm
- https://www.javatpoint.com/c-preprocessor-include
- https://www.javatpoint.com/c-preprocessor-undef
- https://www.geeksforgeeks.org/getdate-and-setdate-function-in-c-with-examples/
- https://www.javatpoint.com/c-string-functions
- https://www.javatpoint.com/c-math

Roadmap for Unit V

1. File Handling
2. File Pointers and Operations
3. Preprocessor Directives
4. Standard Library and Header Files
5. String, Math and Date Time Functions
6. Summary

Roadmap to Programming in C

Unit I
Computer Based Problem Solvingand Overview of C

1

Unit – II
Branching, Looping, Arrays, Strings

2

Unit III
Functions and Recursions

3

Unit IV
Dynamic Data Structures, Pointers, Union

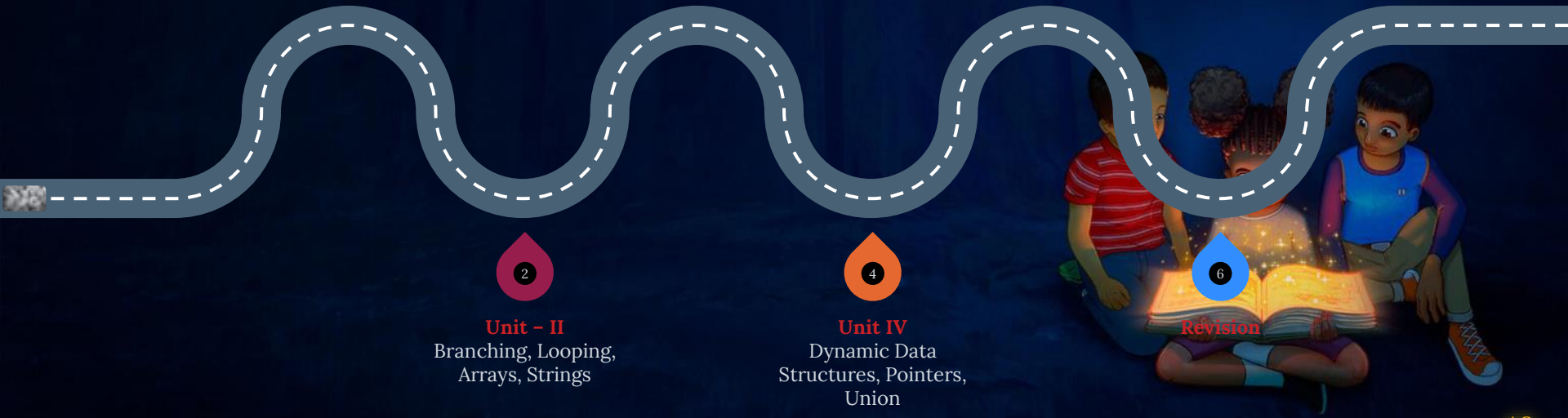
4

Unit V
File Handling, Preprocessor, Standard Library and Header Files

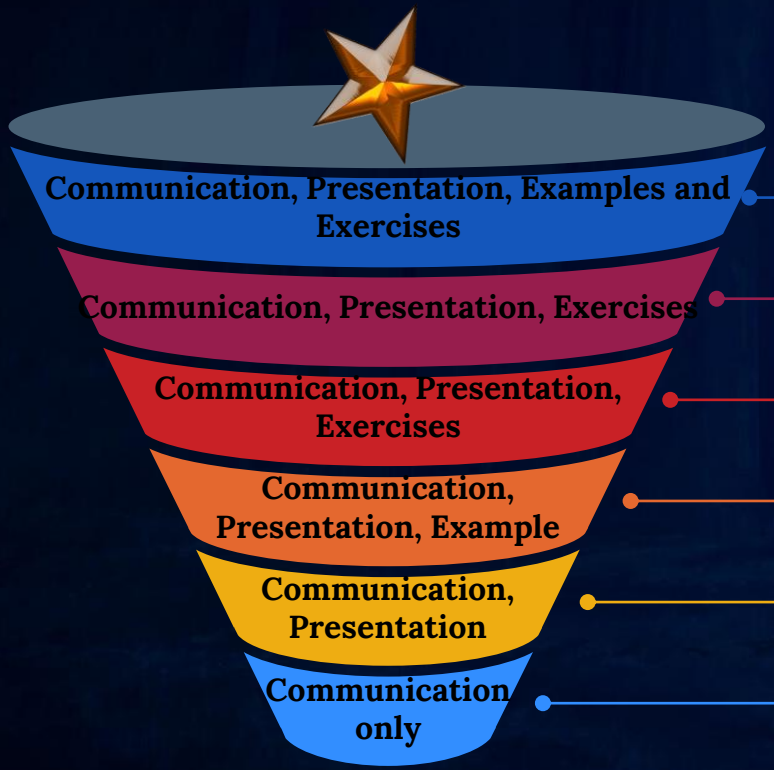
5

6

Revision



Feedback – Rating Star



$90

$80

$60

$50

$40

$30

ALL THE BEST

I'm so glad to have been your teacher,
I've watched you learn and grow,
and change from day to day.
I hope that all the little things
I've done,
Have helped in some small way!
With love and best wishes!

Thanks!

Any questions?

You can find me at:

- kramdharma@gmail.com