PROGRAMMING

Subject Title: Programming in 'C'
Subject Code: CAM203-1C

Unit-4 File Management in C

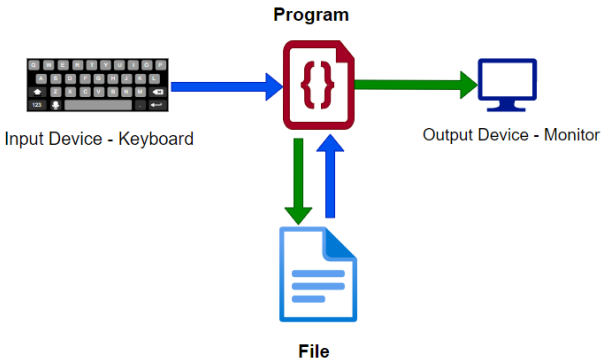
Prepared by Ankit Rami(AR)

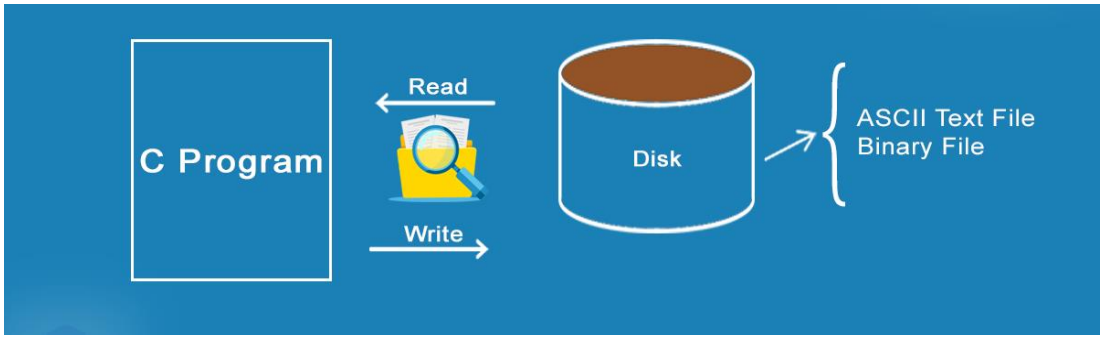
Any Query Contact – 8460467193

Email – ankitramiblog@gmail.com

Introduction for File Management

- ✓ File handing in C is the process in which we create, open, read, write, and close operations on a file.
- ✓ The process of file handling refers to how we store the available data or information in a file with the help of a program.
- ✓ The C language stores all the data available in a program into a file with the help of file handling in C. This data can be fetched/extracted from these files to work again in any program.
- ✓ C language provides different functions such as fopen(), fwrite(), fread(), fseek(), fprintf(), etc. to perform input, output, and many different C file operations in our program.





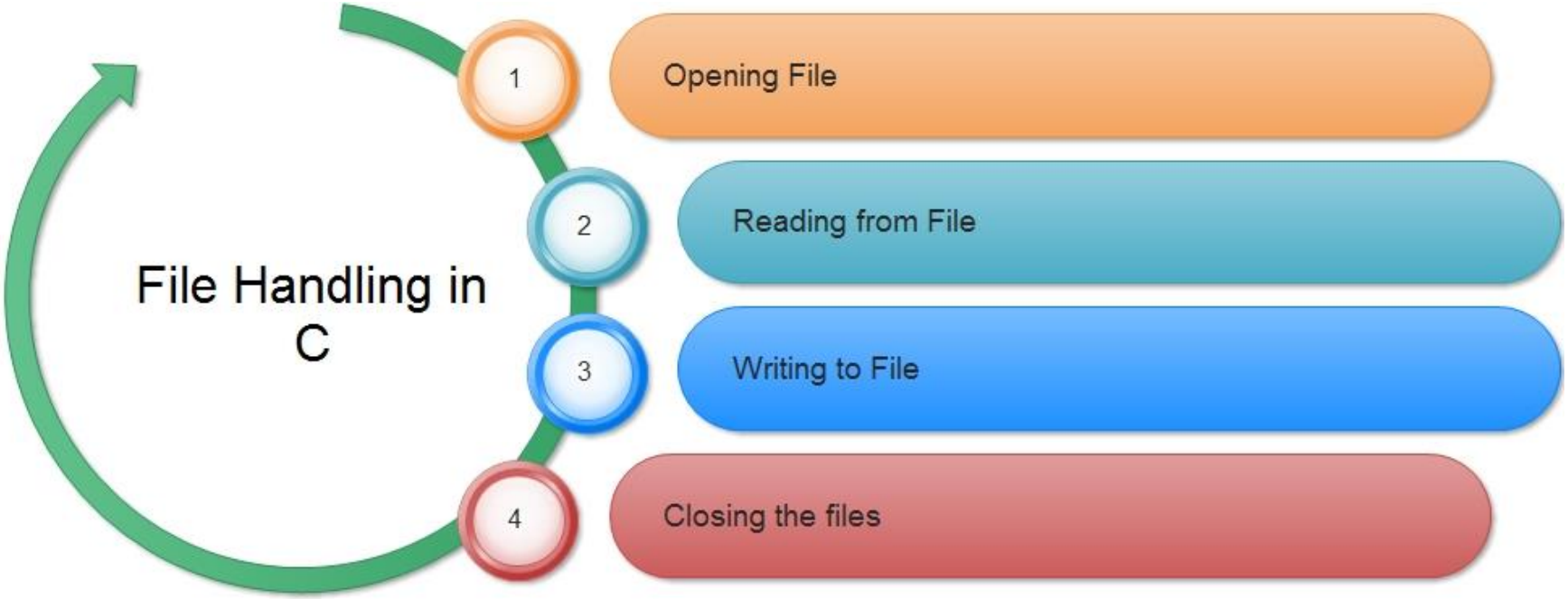
Why do we need File Handling in C?

- ✓ So far the operations using the C program are done on a prompt/terminal which is not stored anywhere. The output is deleted when the program is closed. But in the software industry, most programs are written to store the information fetched from the program. The use of file handling is exactly what the situation calls for.
- ✓ In order to understand why file handling is important, let us look at a few features of using files:

1. **Reusability:** The data stored in the file can be accessed, updated, and deleted anywhere and anytime providing high reusability.
2. **Portability:** Without losing any data, files can be transferred to another in the computer system. The risk of flawed coding is minimized with this feature.
3. **Efficient:** A large amount of input may be required for some programs. File handling allows you to easily access a part of a file using few instructions which saves a lot of time and reduces the chance of errors.
4. **Storage Capacity:** Files allow you to store a large amount of data without having to worry about storing everything simultaneously in a program.

File Handling in C

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



Functions for file Handling in C

Function	Operation
fopen()	Creates a new file / opens an existing file
fclose()	Closes a file which has been opened for use
getc()	Reads a character from the file
putc()	Writes a character to the file
fprintf()	Write data values to a file
fscanf()	Reads a set of data values from a file
getw()	Reads an integer from the file
putw()	Writes an integer to a file
fseek()	Sets the position to the desired point in the file
ftell()	Gives the current position in the file
rewind()	Sets the position to the beginning of the file

File Handling Mode in C

Mode	Description
"w"	We use this mode to open or create a text file in writing mode. The data existing in the file is erased.
"r"	We use this mode for opening an existing file in reading mode. The file to be opened must exist.
"a"	We use this mode to open a text file in append mode. The existing data of the file is not erased as in “w” mode.
"w+"	We use this mode to open a text file in both reading and writing mode.
"a+"	We use this mode to open a text file in both reading and writing mode.
"r+"	We use this mode to open a text file in both reading and writing mode.
"wb"	We use this mode to open or create a binary file in writing mode.
"rb"	We use this mode to open a binary file in reading mode.
"ab"	We use this mode to open a binary file in append mode.
"wb+"	We use this mode to open a binary file in both reading and writing modes.
"rb+"	We use this mode to open a binary file in both reading and writing modes.
"ab+"	We use this mode to open a binary file in both reading and writing modes.

Types of File in C

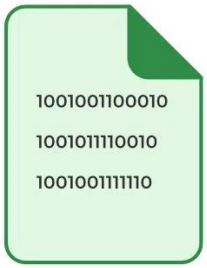
✓ Text Files

- A text file contains data in the **form of ASCII characters** and is generally used to store a stream of characters.
- Each line in a text file ends with a new line character ('\n').
- It can be read or written by any text editor.
- They are generally stored with **.txt** file extension.
- Text files can also be used to store the source code.

file.txt

✓ Binary Files

- A binary file contains data in **binary form (i.e. 0's and 1's)** instead of ASCII characters. They contain data that is stored in a similar manner to how it is stored in the main memory.
- The binary files can be created only from within a program and their contents can only be read by a program.
- More secure as they are not easily readable.
- They are generally stored with **.bin** file extension.



file.bin

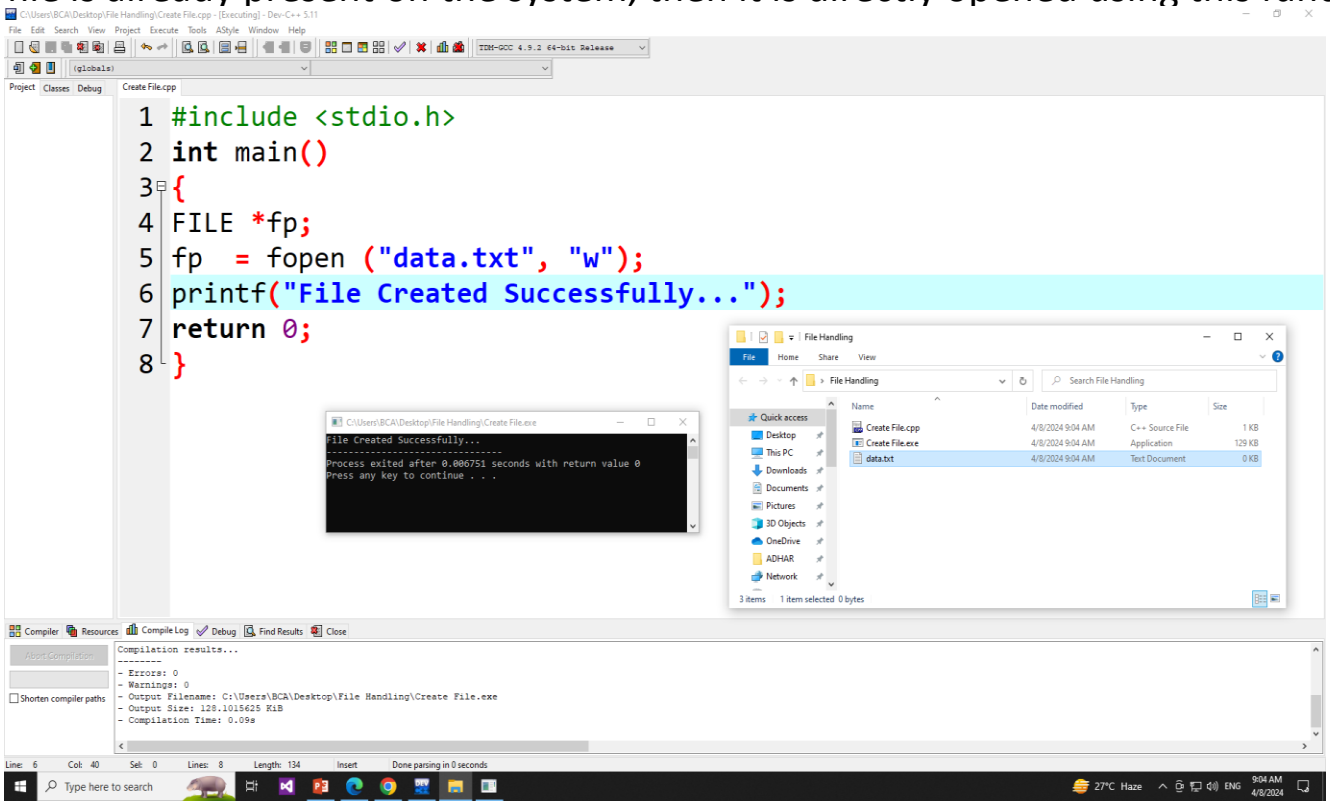
📣 Create .txt File Using C

✓ Create File Syntax

FILE *fp;

fp = fopen ("file_name", "mode");

- fopen is a standard function which is used to open a file.
- If the file is not present on the system, then it is created and then opened.
- If a file is already present on the system, then it is directly opened using this function.

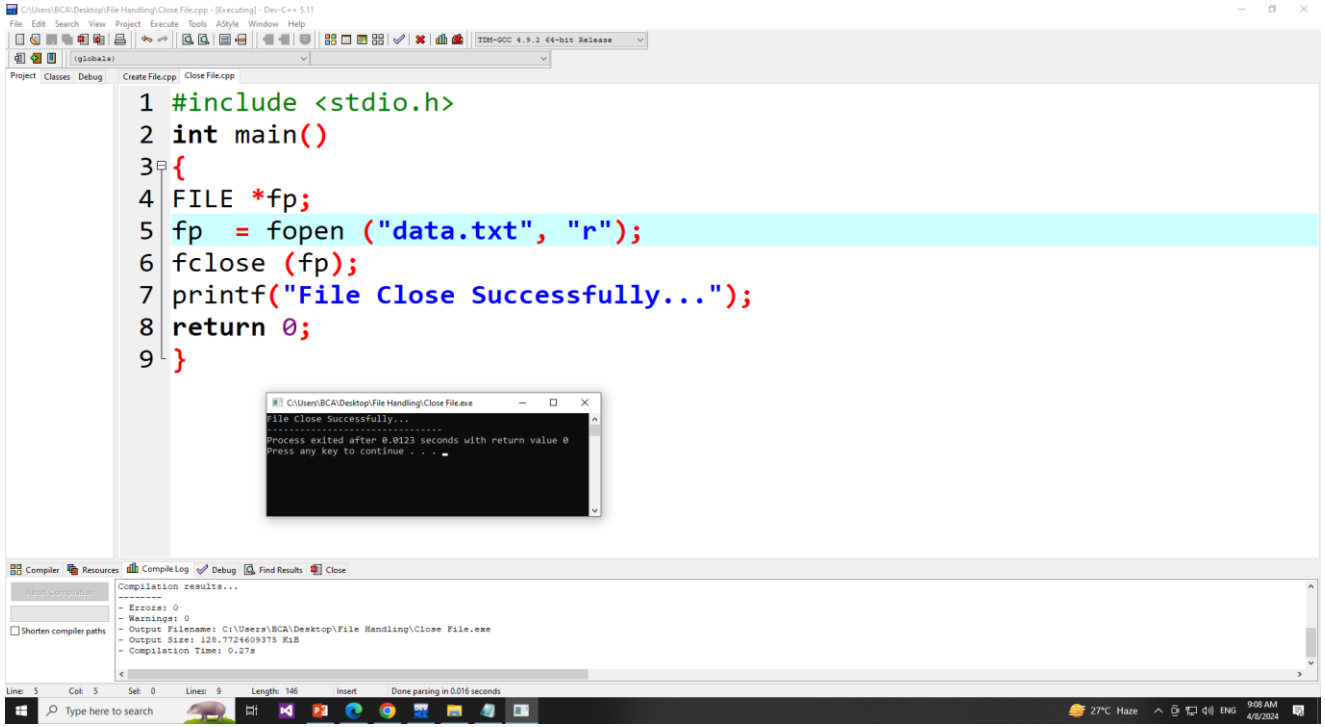


📣 Close .txt File Using C

✓ Close File Syntax

fclose (file_pointer);

- The fclose function takes a file pointer as an argument. The file associated with the file pointer is then closed with the help of fclose function. It returns 0 if close was successful and EOF (end of file) if there is an error has occurred while file closing.
- After closing the file, the same file pointer can also be used with other files.
- In ‘C’ programming, files are automatically close when the program is terminated. Closing a file manually by writing fclose function is a good programming practice.



📣 Write Data in .txt File Using C

- In C, when you write to a file, newline characters '\n' must be explicitly added.
- The Stander Library offers the necessary functions to write to a file:
- fputc(char, file_pointer): It writes a character to the file pointed to by file_pointer.
- fputs(str, file_pointer): It writes a string to the file pointed to by file_pointer.
- fprintf(file_pointer, str, variable_lists): It prints a string to the file pointed to by file_pointer. The string can optionally include format specifiers and a list of variables variable_lists.
- The **End of the File (EOF)** indicates the end of input.

✓ **Reference Link –**

- https://www.guru99.com/c-file-input-output.html
- https://www.tutorialspoint.com/explain-the-end-of-file-eof-with-a-c-program

📣 Write Data in .txt File Using C

✓ fputc() Function File Syntax

fputc(Character Data, Open File Details and File Mode Permission);

```c
#include <stdio.h>
int main() {
        int i;
        FILE * fptr;
        char fn[50];
        char str[]="Learn with Ankit Rami\n";
        fptr=fopen("fputc_test.txt", "w"); // "w" defines "writing mode"
        for(i=0;str[i]!='\n';i++)
        {
            /* write to file using fputc() function */
            fputc(str[i],fptr);
        }
        printf("Data Write in File-");
        fclose(fptr);
        return 0;
    }
```

```
Data Write in File
--------------------------------
Process exited after 0.005059 seconds with return value 0
Press any key to continue . . .
```

fputc_test.txt - Notepad

Learn with Ankit Rami

📣 Write Data in .txt File Using C

✓ fputs () Function File Syntax

fputs(Write Your Message, Open File Details and File Mode Permission);

```c
#include <stdio.h>
int main()
{
    FILE * fp;
    fp = fopen("fputs_test.txt", "w+");
    fputs("Learn With Ankit Rami", fp);
    fputs("Using - amit.arinfoway.com Site\n", fp);
    printf("Data Write in File");
    fclose(fp);
    return (0);
}
```

```
Data Write in File
--------------------------------
Process exited after 0.01075 seconds with return value 0
Press any key to continue . . .
```

fputs_test.txt - Notepad

Learn With Ankit RamiUsing - amit.arinfoway.com Site

📣 Write Data in .txt File Using C

✓ fprintf()Function File Syntax

fprintf(Open File Details and File Mode Permission,Write Your Message);

```c
#include <stdio.h>
int main()
{
    FILE *fptr;
    fptr = fopen("fprintf_test.txt", "w"); // "w" defines "writing mode"
    /* write to file */
    fprintf(fptr, "Learning C with Ankit Rami\n");
    printf("Data Write in File");
    fclose(fptr);
    return 0;
}
```

```
Data Write in File
--------------------------------
Process exited after 0.009105 seconds with return value 0
Press any key to continue . . .
```

fprintf_test.txt - Notepad

Learning C with Ankit Rami

📢 Read Data in .txt File Using C

- There are three different functions dedicated to reading data from a file
- **fgetc(file_pointer):** It returns the next character from the file pointed to by the file pointer. When the end of the file has been reached, the EOF is sent back.
- **fgets(buffer, n, file_pointer):** It reads n-1 characters from the file and stores the string in a buffer in which the NULL character '\0' is appended as the last character.
- **fscanf(file_pointer, conversion_specifiers, variable_adresses)**: It is used to parse and analyze data. It reads characters from the file and assigns the input to a list of variable pointers variable_adresses using conversion specifies. Keep in mind that as with scanf, fscanf stops reading a string when space or newline is encountered.

✓ **Reference Link –**

- https://www.guru99.com/c-file-input-output.html
- https://www.tutorialspoint.com/explain-the-end-of-file-eof-with-a-c-program

Read Data in .txt File Using fgets(),fscanf(),fgetc() in C

C:\Users\BCA\Desktop\File Handling\Read Data From File In C.cpp - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 64-bit Release

(globals)

Project Classes Debug [*] Read Data From File In C.cpp

```c
#include <stdio.h>
int main()
{
        FILE * file_pointer;
        char buffer[30], c;

        file_pointer = fopen("fputc_test.txt", "r");
        printf("----read a line----\n");
        fgets(buffer, 50, file_pointer);
        printf("%s\n", buffer);

        printf("----read and parse data----\n");
        file_pointer = fopen("fprintf_test.txt", "r"); //reset the pointer
        char str1[10], str2[10], str3[10], str4[10];
        fscanf(file_pointer, "%s %s %s %s", str1, str2, str3, str4);
        printf("Read String1 |%s|\n", str1);
        printf("Read String2 |%s|\n", str2);
        printf("Read String3 |%s|\n", str3);
        printf("Read String4 |%s|\n", str4);

        printf("----read the entire file----\n");
        file_pointer = fopen("fprintf_test.txt", "r"); //reset the pointer
        while ((c = getc(file_pointer)) != EOF)
        {
            printf("%c", c);
        }
        fclose(file_pointer);
        return 0;
}
```

C:\Users\BCA\Desktop\File Handling\Read Data From File In C.exe

```
----read a line----
Learn with Ankit Rami
----read and parse data----
Read String1 |Learning|
Read String2 |C|
Read String3 |with|
Read String4 |Ankit|
----read the entire file----
Learning C with Ankit Rami

--------------------------------
Process exited after 0.01513 seconds with return
 value 0
Press any key to continue . . .
```

fprintf_test.txt - Notepad

File Edit Format View Help

Learning C with Ankit Rami

Ln 1, Col 27 | 100% | Windows (CRLF) | UTF-8

Compiler Resources Compile Log Debug Find Results

Line: 30 Col: 1 Sel: 0 Lines: 31 Length: 975 Insert Done parsing in 0 seconds

Type here to search 27°C Haze ENG 10:47 AM 4/8/2024

Read Data in .txt File Using fgets(),fscanf(),fgetc() in C

- In the above program, we have opened the file called “fprintf_test.txt” which was previously written using fprintf() function, and it contains **“Learning C with Ankit Rami”** string. We read it using the fgets() function which reads line by line where the buffer size must be enough to handle the entire line.
- We reopen the file to reset the pointer file to point at the beginning of the file. Create various strings variables to handle each word separately. Print the variables to see their contents. The fscanf() is mainly used to extract and parse data from a file.
- Reopen the file to reset the pointer file to point at the beginning of the file. Read data and print it from the file character by character using getc() function until the EOF statement is encountered
- After performing a reading operation file using different variants, we again closed the file using the fclose function.

Interactive File Read and Write with getc and putc in C

```c
#include<stdio.h>
int main(){
    char ch;
    FILE *fp;
    fp=fopen("ar.txt","w"); //opening file in write mode
    printf("Write Text in FIle And Press ctrl+z:");
    while((ch = getchar())!=EOF)
    {
        putc(ch,fp); // writing each character into the file
    }
    fclose(fp);
    fp=fopen("ar.txt","r");
    printf("Read Text in File:");
    while ((ch=getc(fp))!=EOF)
    { // reading each character from file
        putchar(ch); // displaying each character on to the screen
    }
    fclose(fp);
    return 0;
}
```

```
Write Text in FIle And Press ctrl+z:Ankit Rami From Rupal
Jay Maa Vardayini
^Z
Read Text in File:Ankit Rami From Rupal
Jay Maa Vardayini

--------------------------------
Process exited after 16.18 seconds with return value 0
Press any key to continue . . .
```

ar.txt - Notepad

```
Ankit Rami From Rupal
Jay Maa Vardayini
```

📣 Command Line Arguments in C

- The arguments passed from command line are called command line arguments. These arguments are handled by main() function.
- To support command line argument, you need to change the structure of main() function as given below.
- **int** main(**int** argc, **char** *argv[])
- Here, **argc** counts the number of arguments. It counts the file name as the first argument.
- The **argv[]** contains the total number of arguments. The first argument is the file name always.

✓ **Reference Link –**

- https://www.javatpoint.com/command-line-arguments-in-c

Command Line Arguments in C

D:\Programming-C AR\Unit-4\File Handling\Command Line Arguments in C.cpp - [Executing] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

Command Line Arguments in C.cpp

```c
#include <stdio.h>
int main(int argc, char *argv[] )
{
    printf("Program name is: %s\n", argv[0]);
    if(argc < 2)
    {
    printf("No argument passed through command line.\n");
    }
    else
    {
      printf("First argument is: %s\n", argv[1]);
    }
}
```

D:\Programming-C AR\Unit-4\File Handling\Command Line Arguments in C.exe

```
Program name is: D:\Programming-C AR\Unit-4\File Handling\Command Line Arguments in C.exe
No argument passed through command line.

--------------------------------
Process exited after 0.005992 seconds with return value 0
Press any key to continue . . .
```

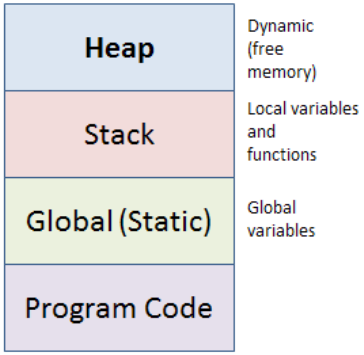
Static V/S Dynamic Memory Allocation in C

Static Memory Allocation	Dynamic Memory Allocation
Memory is allocated before the execution of the program begins. (During Compilation)	Memory is allocated during the execution of the program.
No memory allocation or deallocation actions are performed during Execution.	Memory Bindings are established and destroyed during the Execution.
Variables remain permanently allocated.	Allocated only when program unit is active.
Implemented using stacks and heaps	Implemented using data segments.
Pointer is needed to accessing variables.	No need of Dynamically allocated pointers.
Faster execution than Dynamic	Slower execution than static
More memory Space required.	Less Memory space required.

Dynamic Memory Allocation in C

- ✓ Array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it.
- ✓ Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming.
- ✓ To allocate memory dynamically, library functions are malloc(), calloc(), realloc() and free() are used. These functions are defined in the <stdlib.h> header file.
- ✓ Dynamic Memory Allocation in C language *enables the C Programmer to allocate memory at runtime*.
- ✓ Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

malloc()	allocates single block of requested memory.
calloc()	allocates multiple block of requested memory.
realloc()	reallocates the memory occupied by malloc() or calloc() functions.
free()	frees the dynamically allocated memory.



📣 malloc() function in C

- The malloc() function allocates single block of requested memory.
- It doesn't initialize memory at execution time, so it has garbage value initially.
- It returns NULL if memory is not sufficient.
- syntax of malloc() function - ptr=(cast-type*)malloc(byte-size)

```c
#include<stdio.h>
#include<stdlib.h>
int main(){
  int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)malloc(n*sizeof(int));  //memory allocated using malloc
    if(ptr==NULL)
    {
        printf("Sorry! unable to allocate memory");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
    return 0;
}
```

```
Enter number of elements: 3
Enter elements of array: 10
20
30
Sum=60
--------------------------------
Process exited after 4.704 seconds with return value 0
Press any key to continue . . .
```

📣 calloc() function in C

- The calloc() function allocates multiple block of requested memory.
- It initially initialize all bytes to zero.
- It returns NULL if memory is not sufficient.
- Syntax of calloc() function - ptr=(cast-type*)calloc(number, byte-size)

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)calloc(n,sizeof(int));  //memory allocated using calloc
    if(ptr==NULL)
    {
        printf("Sorry! unable to allocate memory");
        exit(0);
    }
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
        scanf("%d",ptr+i);
        sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
    return 0;
}
```

```
Enter number of elements: 3
Enter elements of array: 10 20 30
Sum=60
--------------------------------
Process exited after 4.009 seconds with return value 0
Press any key to continue . . .
```

📣 free() function in C

- “free” function in C is used to dynamically de-allocate the memory.
- The memory allocated using functions malloc() and calloc() is not de-allocated on their own.
- Hence the free() function is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.
- Syntax of free(ptr) – free(Pointer Variable Name)

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *ptr, *ptr1;
    int n, i;
    // Get the number of elements for the array
    n = 5;
    printf("Enter number of elements: %d\n", n);
    // Dynamically allocate memory using malloc()
    ptr = (int*)malloc(n * sizeof(int));
    // Dynamically allocate memory using calloc()
    ptr1 = (int*)calloc(n, sizeof(int));
    // Check if the memory has been successfully
    // allocated by malloc or not
    if (ptr == NULL || ptr1 == NULL)
    {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else
    {
        printf("Memory successfully allocated using malloc.\n");
        // Free the memory
        free(ptr);
        printf("Malloc Memory successfully freed.\n");

        printf("\nMemory successfully allocated using calloc.\n");
        // Free the memory
        free(ptr1);
        printf("Calloc Memory successfully freed.\n");
    }
    return 0;
}
```

```
Enter number of elements: 5
Memory successfully allocated using malloc.
Malloc Memory successfully freed.

Memory successfully allocated using calloc.
Calloc Memory successfully freed.

--------------------------------
Process exited after 0.005275 seconds with return value 0
Press any key to continue . . .
```

📣 realloc() function in C

- If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.
- **“realloc”** or **“re-allocation”** method in C is used to dynamically change the memory allocation of a previously allocated memory.
- Syntax of realloc– ptr=realloc(ptr, **new**-size)

C:\Users\BCA\Desktop\File Handling\DMA\realloc.cpp - [Executing] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

Project Classes Debug [*] calloc.cpp [*] free.cpp realloc.cpp

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int* ptr;
    int n, i;
    n = 5;
    printf("Enter number of elements: %d\n", n);
    ptr = (int*)calloc(n, sizeof(int));
    if (ptr == NULL)
    {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else
    {
        printf("Memory successfully allocated using calloc.\n");
        for (i = 0; i < n; ++i)
        {
            ptr[i] = i + 1;
        }
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i)
        {
            printf("%d, ", ptr[i]);
        }

        // Get the new size for the array
        n = 10;
        printf("\n\nEnter the new size of the array: %d\n", n);
        // Dynamically re-allocate memory using realloc()
        ptr = (int*)realloc(ptr, n * sizeof(int));
        // Memory has been successfully allocated
        printf("Memory successfully re-allocated using realloc.\n");
        // Get the new elements of the array
        for (i = 5; i < n; ++i)
        {
            ptr[i] = i + 1;
        }
        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i)
        {
            printf("%d, ", ptr[i]);
        }
        free(ptr);
    }
    return 0;
```

C:\Users\BCA\Desktop\File Handling\DMA\realloc.exe

```
Enter number of elements: 5
Memory successfully allocated using calloc.
The elements of the array are: 1, 2, 3, 4, 5,

Enter the new size of the array: 10
Memory successfully re-allocated using realloc.
The elements of the array are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
--------------------------------
Process exited after 0.008062 seconds with return value 0
Press any key to continue . . .
```

Any Query Contact Us

Faculty Name- Ankit Rami

Email – ankitramiblog@gmail.com

Contact No – +91 8460467193

Website - amit.arinfoway.com

Like Comment Share

Subscribe

Subscribe Our YouTube Channel

https://www.youtube.com/channel/UCWbJh2iQ8w-8nrU0Xpjpw7g