Basics of File Handling in C

geeksforgeeks.org/basics-file-handling-c

October 27, 2017

So far the operations using C program are done on a prompt / terminal which is not stored anywhere. But in the software industry, most of the programs are written to store the information fetched from the program. One such way is to store the fetched information in a file. Different operations that can be performed on a file are:

1. Creation of a new file (**fopen with attributes as "a" or "a+" or "w" or "w++")**
2. Opening an existing file (**fopen**)
3. Reading from file (**fscanf or fgets**)
4. Writing to a file (**fprintf or fputs**)
5. Moving to a specific location in a file (**fseek, rewind**)
6. Closing a file (**fclose**)

The text in the brackets denotes the functions used for performing those operations.
Functions in File Operations:

Opening or creating file
For opening a file, fopen function is used with the required access modes. Some of the commonly used file access modes are mentioned below.
File opening modes in C:

- **"r" –** Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer which points to the first character in it. If the file cannot be opened fopen( ) returns NULL.
- **"rb" –** Open for reading in binary mode. If the file does not exist, fopen( ) returns NULL.
- **"w" –** Searches file. If the file exists, its contents are overwritten. If the file doesn't exist, a new file is created. Returns NULL, if unable to open file.
- **"wb" –** Open for writing in binary mode. If the file exists, its contents are overwritten. If the file does not exist, it will be created.
- **"a" –** Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the last character in it. If the file doesn't exist, a new file is created. Returns NULL, if unable to open file.
- **"ab" –** Open for append in binary mode. Data is added to the end of the file. If the file does not exist, it will be created.
- **"r+" –** Searches file. If is opened successfully fopen( ) loads it into memory and sets up a pointer which points to the first character in it. Returns NULL, if unable to open the file.
- **"rb+" –** Open for both reading and writing in binary mode. If the file does not exist, fopen( ) returns NULL.
- **"w+" –** Searches file. If the file exists, its contents are overwritten. If the file doesn't exist a new file is created. Returns NULL, if unable to open file.
- **"wb+" –** Open for both reading and writing in binary mode. If the file exists, its contents are overwritten. If the file does not exist, it will be created.
- **"a+" –** Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer which points to the last character in it. If the file doesn't exist, a new file is created. Returns NULL, if unable to open file.
- **"ab+" –** Open for both reading and appending in binary mode. If the file does not exist, it will be created.

As given above, if you want to perform operations on a binary file, then you have to append 'b' at the last. For example, instead of "w", you have to use "wb", instead of "a+" you have to use "a+b". For performing the

File operation	Declaration & Description
fopen() - To open a file	Declaration: FILE *fopen (const char *filename, const char *mode) fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist. FILE *fp; fp=fopen (“filename”, ”‘mode”); Where, fp – file pointer to the data type “FILE”. filename – the actual file name with full path of the file. mode – refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+. Please refer below the description for these mode of operations.

fclose() - To close a file	Declaration: int fclose(FILE *fp);
	fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below.
	fclose (fp);

fgets() - To read a file	Declaration: char *fgets(char *string, int n, FILE *fp) fgets function is used to read a file line by line. In a C program, we use fgets function as below. fgets (buffer, size, fp); where, buffer – buffer to put the data in. size – size of the buffer fp – file pointer
fprintf() - To write into a file	Declaration: int fprintf(FILE *fp, const char *format, …);fprintf() function writes string into a file pointed by fp. In a C program, we write string into a file as below. fprintf (fp, “some data”); or fprintf (fp, “text %d”, variable_name);

operations on the file, a special pointer called File pointer is used which is declared as

```
FILE *filePointer;

So, the file can be opened as

filePointer = fopen(“fileName.txt”, “w”)
```

The second parameter can be changed to contain all the attributes listed in the above table.

Reading from a file –
The file read operations can be performed using functions fscanf or fgets. Both the functions performed the same operations as that of scanf and gets but with an additional parameter, the file pointer. So, it depends on you if you want to read the file line by line or character by character.
And the code snippet for reading a file is as:

```
FILE * filePointer;

filePointer = fopen(“fileName.txt”, “r”);

fscanf(filePointer, "%s %s %s %d", str1, str2, str3, &year);
```

Writing a file –:
The file write operations can be performed by the functions fprintf and fputs with similarities to read operations. The snippet for writing to a file is as :

```
FILE *filePointer ;

filePointer = fopen(“fileName.txt”, “w”);

fprintf(filePointer, "%s %s %s %d", "We", "are", "in", 2012);
```

Closing a file –:
After every successful file operations, you must always close a file. For closing a file, you have to use fclose function. The snippet for closing a file is given as :

```
FILE *filePointer ;

filePointer= fopen(“fileName.txt”, “w”);

---------- Some file Operations -------

fclose(filePointer)
```

Example 1: Program to Open a File, Write in it, And Close the File

C

```
// C program to Open a File,
// Write in it, And Close the File
# include <stdio.h>
# include <string.h>
int main( )
{
// Declare the file pointer
FILE *filePointer ;

// Get the data to be written in file
char dataToBeWritten[50]
= "GeeksforGeeks-A Computer Science Portal for Geeks" ;
// Open the existing file GfgTest.c using fopen()
// in write mode using "w" attribute
filePointer = fopen ( "GfgTest.c" , "w" ) ;
// Check if this filePointer is null
// which maybe if the file does not exist
if ( filePointer == NULL )
{
printf (  "GfgTest.c file failed to open." ) ;
}
else
{

printf ( "The file is now opened.\n" ) ;

// Write the dataToBeWritten into the file
if (  strlen ( dataToBeWritten ) > 0 )
{

// writing in the file using fputs()
fputs (dataToBeWritten, filePointer) ;
fputs ( "\n" , filePointer) ;
}

// Closing the file using fclose()
fclose (filePointer) ;

printf ( "Data successfully written in file GfgTest.c\n" );
printf ( "The file is now closed." ) ;
}
return 0;
}
```

Example 2: Program to Open a File, Read from it, And Close the File

C

```
// C program to Open a File,
// Read from it, And Close the File
# include <stdio.h>
# include <string.h>
int main( )
{
// Declare the file pointer
FILE *filePointer ;

// Declare the variable for the data to be read from file
char dataToBeRead[50];
// Open the existing file GfgTest.c using fopen()
// in read mode using "r" attribute
filePointer = fopen ( "GfgTest.c" , "r" ) ;

// Check if this filePointer is null
// which maybe if the file does not exist
if ( filePointer == NULL )
{
printf ( "GfgTest.c file failed to open." ) ;
}
else
{

printf ( "The file is now opened.\n" ) ;

// Read the dataToBeRead from the file
// using fgets() method
while ( fgets ( dataToBeRead, 50, filePointer ) != NULL )
{
```

```
// Print the dataToBeRead
printf (  "%s" , dataToBeRead ) ;
}

// Closing the file using fclose()
fclose (filePointer) ;

printf ( "Data successfully read from file GfgTest.c\n" );
printf ( "The file is now closed." ) ;
}
return 0;
}
```

Read/Write structure to a file in C

geeksforgeeks.org/readwrite-structure-file-c

June 28, 2017

Prerequisite: Structure in C
For writing in file, it is easy to write string or int to file using **fprintf** and **putc**, but you might have faced difficulty when writing contents of struct. **fwrite** and **fread** make task easier when you want to write and read blocks of data.

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)
ptr - This is pointer to array of elements to be written
size -  This is the size in bytes of each element to be written
nmemb - This is the number of elements, each one with a size of size bytes
stream - This is the pointer to a FILE object that specifies an output stream
```

C

```
// C program for writing
// struct to file
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// a struct to read and write
struct person
{
int id;
char fname[20];
char lname[20];
};
int main ()
{
FILE *outfile;

// open file for writing
outfile = fopen ( "person.dat" ,  "w" );
if (outfile == NULL)
{
```

```
fprintf (stderr, "\nError opened file\n" );
exit (1);
}
struct person input1 = {1, "rohan" , "sharma" };
struct person input2 = {2, "mahendra" , "dhoni" };

// write struct to file
fwrite (&input1, sizeof ( struct person), 1, outfile);
fwrite (&input2, sizeof ( struct person), 1, outfile);

if ( fwrite != 0)
printf ( "contents to file written successfully !\n" );
else
printf ( "error writing file !\n" );
// close file
fclose (outfile);
return 0;
}
```

Output:

```
gcc demowrite.c
./a.out
contents to file written successfully!
```

fread : Following is the declaration of fread function

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
ptr - This is the pointer to a block of memory with a minimum size of size*nmemb bytes.
size - This is the size in bytes of each element to be read.
nmemb - This is the number of elements, each one with a size of size bytes.
stream - This is the pointer to a FILE object that specifies an input stream.
```

C

```
// C program for reading
// struct from a file
#include <stdio.h>
#include <stdlib.h>
```

```
// struct person with 3 fields
struct person
{
int id;
char fname[20];
char lname[20];
};
// Driver program
int main ()
{
FILE *infile;
struct person input;

// Open person.dat for reading
infile = fopen ( "person.dat" , "r" );
if (infile == NULL)
{
fprintf (stderr, "\nError opening file\n" );
exit (1);
}

// read file contents till end of file
while ( fread (&input, sizeof ( struct person), 1, infile))
printf ( "id = %d name = %s %s\n" , input.id,
input.fname, input.lname);
// close file
fclose (infile);
return 0;
}
```

Output:

```
gcc demoread.c
./a.out
id = 1   name = rohan sharma
id = 2   name = mahendra dhoni
```

Standard Library Functions for Character & String Manipulation in C++

study.com/academy/lesson/standard-library-functions-for-character-string-manipulation-in-c.html

String and Character Libraries

If you've ever written a line of code in C++ and it won't compile because something wasn't defined, the chances are you didn't include the right library. When you work with strings, that is, one or more characters strung together, there are a few key libraries you will need to know.

C++ provides a lot more support for strings and string functions than C. In C, a string is really just an array of characters, which can sometimes make it more challenging to work with. However, you can still work with individual characters. Let's look at the C-style character libraries that you can use.

First, let's look at the old C-style string:

C String

The library file **<cstring>** contains the functions for the old-fashioned C-string, which is a **char** array. If you create a string using the C standard, you would use the following:

```
char * banner2 = "Welcome";
```

However, you will get a compiler warning! This is because the functionality is deprecated, meaning it is no longer fully supported. You can get around it by making the character a constant. If this doesn't look like much fun, don't worry. We'll cover the C++ string libraries shortly.

```
char * banner3 = const_cast<char *>("Welcome");
```

The other option is to create an array of characters:

```
char[] banner = "Welcome to the Middle of the Film";
```

Let's look at some of the functions available in <cstring>.

Functions in C-String

The table below shows some common functions in the <cstring> library:

Function	Description
strlen	Displays the length of a string
strcpy (char * destination, const char * source)	Copies the source string to the destination

strcat (char * destination, const char * source)	Concatenates the source string to the destination

C Standard Library

Another C-style library is **<cstdlib>**. The string tools in this library are mainly used to manipulate strings and change them into other data types. Below is a sample of some of the functions.

Function	Description
atoi (char * cstring)	Convert C string to an int
atof (char * cstring)	Convert C string to a double
atol (char * cstring)	Convert C string to a long integer

You can use all of these tools in C++ but remember that your strings must be in the C format. That is, an array of characters! It must be declared as follows:

```
char * banner3 = const_cast<char *>("Welcome");
```

On the other hand, if you want to use C++ string functions, there are libraries for that, too. Let's take a look.

- Lesson
- Quiz
- Course

6.3K views

C++ String Class

It is important to understand how C provided the foundation for C++ strings and string manipulation. When you are working in C++, you have access to string tools that are a little more straightforward.

The **string** class is part of the header file **<string>**. When you import this header file into your program, you can then declare variables as strings instead of character arrays.

```
#include <iostream>
#include <string>
using namespace std;
int main() {
  string banner;
  banner = "Welcome to the Middle of the Film!"
  cout << banner << endl;
}
```

When run, the output should be:

```
Welcome to the Middle of the Film!
```

Now let's look at some of the functions we have available for strings.

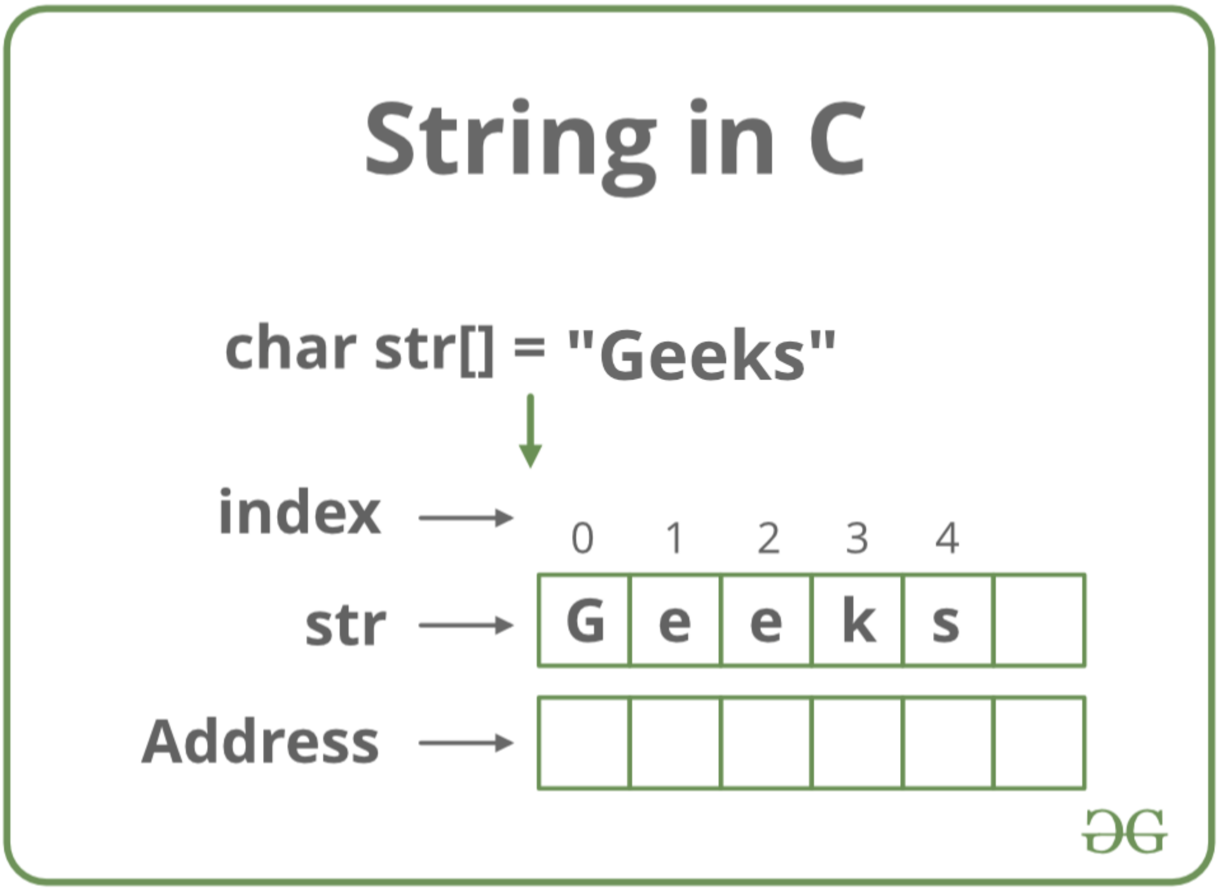
C++ String Functions

Function	Description	Example
size(); length()	Same functions, both return size/length of the string	`cout << banner.size();`
begin(); end();	Starting and ending index of the string	`cout << banner.end();`
clear();	Blanks out the string	`banner.clear();`
empty();	If a string is empty, returns true	`if(banner.empty()) { }`
replace(start, length, string);	Replaces a string from starting position to a set number	`banner.replace(1, 5, "Hi");`
substr(start, end);	Finds substring in range indicated	`string s = banner.substr(0, 5);`
compare(string& str);	Compares strings and returns 0 if equal	`if(banner.compare(1, 3, "Wel") == 0) {}`

Strings in C

geeksforgeeks.org/strings-in-c-2

August 1, 2017



Initializing a String: A string can be initialized in different ways. We will explain this with the help of an example. Below is an example to declare a string with name as str and initialize it with “GeeksforGeeks”.

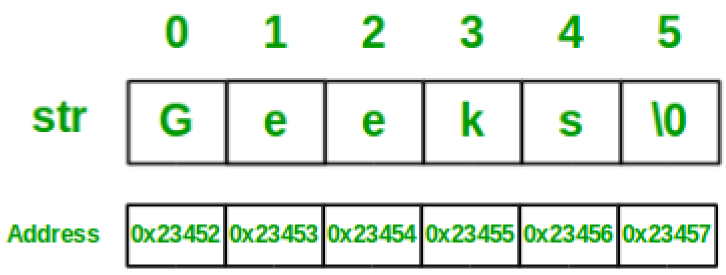
```
1. char str[] = "GeeksforGeeks";

2. char str[50] = "GeeksforGeeks";

3. char str[] = {'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};

4. char str[14] = {'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
```

Below is the memory representation of a string “Geeks”.



Let us now look at a sample program to get a clear understanding of declaring and initializing a string in C and also how to print a string.

```
// C program to illustrate strings


#include<stdio.h>


int main()
{
// declare and initialize string
char str[] = "Geeks" ;


// print string
printf ( "%s" ,str);


return 0;
}
```

Output:

```
Geeks
```

We can see in the above program that strings can be printed using normal printf statements just like we print any other variable. Unlike arrays, we do not need to print a string, character by character. The C language does not provide an inbuilt data type for strings but it has an access specifier “**%s**” which can be used to directly print and read strings.

Below is a sample program to read a string from user:

```
// C program to read strings

#include<stdio.h>

int main()
{
// declaring string
char str[50];

// reading string
scanf ( "%s" ,str);

// print string
printf ( "%s" ,str);

return 0;
}
```

You can see in the above program that string can also be read using a single scanf statement. Also, you might be thinking that why we have not used the ‘&’ sign with string name ‘str’ in scanf statement! To understand this you will have to recall your knowledge of scanf. We know that the ‘&’ sign is used to provide the address of the variable to the scanf() function to store the value read in memory. As str[] is a character array so using str without braces ‘[‘ and ‘]’ will give the base address of this string. That’s why we have not used ‘&’ in this case as we are already providing the base address of the string to scanf.

Passing strings to function: As strings are character arrays, so we can pass strings to function in a same way we pass an array to a function. Below is a sample program to do this:

```
// C program to illustrate how to
// pass string to functions
#include<stdio.h>

void printStr( char str[])
{
printf ( "String is : %s" ,str);
}

int main()
{
// declare and initialize string
char str[] = "GeeksforGeeks" ;

// print string by passing string
// to a different function
printStr(str);

return 0;
}
```

Output:

```
String is : GeeksforGeeks
```

Related Articles:

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.