



Coding-bootcamps.com

Introduction to C Programming Language



Coding Bootcamps

By Tom Brownlee
from [Coding Bootcamps](https://coding-bootcamps.com)

Session 7

Basic Formatted I/O

Recap from Session 6

Arrays

- Purpose of Arrays
- Declaring an Array
- Initializing an Array
- Addressing Elements
- Stepping Through an Array
- Variable Size Arrays
- Arrays of Pointers
- Arrays of Strings
- Passing an Array to a Function
- Dynamic Memory Allocation
- Multidimensional Arrays

Session 7 Outline

Basic Formatted I/O

- Standard I/O Library
- Character Set Encoding
- Standard Input and Output
- Character I/O Functions
- Formatted I/O Functions
- String Constants

Basic File I/O(Standard I/O Library)

A file represents a sequence of bytes, regardless of it being a text file or a binary file. C programming language provides access on high level functions as well as low level (OS level) calls to handle file on your storage devices. All high-level file IO functions deal with FILE* objects. You can't modify these directly, but the functions do it all for you.

You can use the **fopen()** function to create a new file or to open an existing file. This call will initialize an object of the type **FILE**, which contains all the information necessary to control the stream. The prototype of this function call is as follows. Always **fclose()** files when you're done!

```
FILE *fopen( const char * filename, const char * mode );
```

Here, **filename** is a string literal, which you will use to name your file, and access **mode** can have one of the following values –

Mode	Description
r	Opens an existing text file for reading purpose.
w	Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file.
a	Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here your program will start appending content in the existing file content.
r+	Opens a text file for both reading and writing.
w+	Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.
a+	Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

Standard Input and Output

When we say **Input**, it means to feed some data into a program. An input can be given in the form of a file or from the command line. C programming provides a set of built-in functions to read the given input and feed it to the program as per requirement.

When we say **Output**, it means to display some data on screen, printer, or in any file. C programming provides a set of built-in functions to output the data on the computer screen as well as to save it in text or binary files.

The Standard Files

C programming treats all the devices as files. So devices such as the display are addressed in the same way as files and the following three files are automatically opened when a program executes to provide access to the keyboard and screen.

Standard File	File Pointer	Device
Standard input	stdin	Keyboard
Standard output	stdout	Screen
Standard error	stderr	Your screen

The file pointers are the means to access the file for reading and writing purpose. This section explains how to read values from the screen and how to print the result on the screen.

Standard Input and Output

The `getchar()` and `putchar()` Functions

The **`int getchar(void)`** function reads the next available character from the screen and returns it as an integer. This function reads only single character at a time. You can use this method in the loop in case you want to read more than one character from the screen.

The **`int putchar(int c)`** function puts the passed character on the screen and returns the same character. This function puts only single character at a time. You can use this method in the loop in case you want to display more than one character on the screen. Check the following example –

```
#include <stdio.h>
int main( ) {

    int c;

    printf( "Enter a value :");
    c = getchar( );

    printf( "\nYou entered: ");
    putchar( c );

    return 0;
}
```

Character I/O Functions and Formatted I/O Functions

Formatted Output

The function `printf()` is used for formatted output to standard output based on a format specification. The format specification string, along with the data to be output, are the parameters to the `printf()` function.

Syntax:

```
printf (format, data1, data2,.....);
```

In this syntax `format` is the format specification string. This string contains, for each variable to be output, a specification beginning with the symbol `%` followed by a character called the conversion character.

Example:

```
printf ("%c", data1);
```


Character I/O Functions and Formatted I/O Functions

The character specified after % is called a conversion character because it allows one data type to be converted to another type and printed.

See the following table conversion character and their meanings.

Conversion Character	Meaning
d	The data is converted to decimal (integer)
c	The data is taken as a character.
s	The data is a string and character from the string , are printed until a NULL, character is reached.
f	The data is output as float or double with a default Precision 6.
Symbols	Meaning
\n	For new line (linefeed return)
\t	For tab space (equivalent of 8 spaces)

Example

```
printf ("%c\n",data1);
```

The format specification string may also have text.

Example

```
printf ("Character is:""%c\n", data1);
```

The text "Character is:" is printed out along with the value of data1.

Character I/O Functions and Formatted I/O Functions

A number of functions provide for character oriented I/O. Their declarations are:

```
#include <stdio.h>
/* character input */
int fgetc(FILE *stream);
int getc(FILE *stream);
int getchar(void);
int ungetc(int c, FILE *stream);

/* character output */
int fputc(int c, FILE *stream);
int putc(int c, FILE *stream);
int putchar(int c);

/* string input */
char *fgets(char *s, int n, FILE *stream);
char *gets(char *s);

/* string output */
int fputs(const char *s, FILE *stream);
int puts(const char *s);
```

Character Set Encoding

There are tons of different character sets/character encodings out there. Usually, source code is written in ASCII, as are string literals.

If you need non-ASCII encoding, libraries exist to make this as easy as possible for you to work with. Just search for the name of the encoding, plus "C library." C++ has better support for these things.

Under most circumstances, situations where you'd need non-ASCII strings are best handled using localization files (files of strings used in your program translated to different languages and character sets), but these are used in large-scale programs that are beyond the scope of this course.

Summary

NEXT SESSION

Program Debugging

- Problem Analysis
- Instrumenting with printf
- Instrumenting with ctrace
- The Purpose of Debuggers
- How Not to Use Debuggers
- Symbolic Debuggers

Live private coaching sessions for C

- Private tutoring sessions for software design and engineering- Weekly and monthly plans
- C and C++ programming languages- Private tutoring sessions

More software engineering training

- [Introduction to Python Programming](#)
- [Introduction to Java Programming](#)
- [Introduction to Go Programming](#)
- [Learn Kotlin Programming by Examples](#)

Follow up classes

- [Intermediate level C programming language with hands-on examples](#)
- [Learn C++ Programming by Examples](#)



Coding-bootcamps.com

Thank You



Coding
Bootcamps