Epitech Training

Session 6

C - Structures

Arrays allow to define type of variables that can hold several data items of the same kind. Similarly structure is another user defined data type available in C that allows to combine data items of different kinds.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

- Title
- Author
- Subject
- Book ID

C - Structures

Defining a Structure

To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows –

```
struct [structure tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

C - Structures

The structure tag is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure −

```
struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
} book;
```

C - Structures

Accessing Structure Members

To access any member of a structure, we use the member access operator (.). The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword struct to define variables of structure type. The following example shows how to use a structure in a program −

```
struct Books Book1;
strcpy( Book1.title, "My book");
strcpy( Book1.author, "Yoann");
strcpy( Book1.subject, "C is fun");
Book1.book_id = 1234;
```

C - Input & 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.

C - Input & Output

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.

- Stdin
  - Keyboard
- Stdout
  - Screen
- Stderr
  - Screen

C - Input & 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.

C - Input & Output

If you remember the my_write function I have asked you to write last time, you have just implemented the **putchar** by yourself.

The reason why I don’t want you to use printf at all is that I want you to learn how C is working.

If you are able to rewrite printf by yourself, I will allow you to use it into your next program.

For now, you just need to build your own library that you can reuse for your future projects.

C - File I/O

The last chapter explained the standard input and output devices handled by C programming language. This chapter cover how C programmers can create, open, close text or binary files for their data storage.

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. This chapter will take you through the important calls for file management.

C - File I/O

You can use the **open( )** function to create a new file or to open an existing file. This call will initialize a file descriptor where you can read from or write to. Every time you open a file, your program is using resources, that is why you need to close it.

A file descriptor is an integer that is assigned to your program via the kernel, so when you run your program, the system will allocate by default 3 file descriptors (STDIN - 0, STDOUT - 1 and STDERR - 2), so next time you open a file, it file descriptor will have the value 3.

C - File I/O

Writing or reading from a file is really easy, you just need to provide the file descriptor to read or write function.

write(fd, buff, buff_len);

Or

read(fd, buff, buff_len);

For more details, read the manual associated to it. man read, man write, man open, man close.

C - File I/O

As any system call, open, read, write and close all return an integer when you call them.

You NEED to check the value of this integer to see if the function has failed or not. A value lower than 0 will indicate an error during the function call. If above 0, then the function successfully performed the action you wanted.

UNCHECKED returned values of system calls will often make your program unstable and will, by definition, make your program crash.

A crashing program leads to 0 / 20!

Exercises

- Lib
  - int my_putstring(char *)
  - int my_putchar(int c)
  - int my_putnumber(int)
- Program
  - my_cat
    - Same as the command cat which display a file into your terminal, you will have to write a similar program that takes files as arguments and display them into the terminal on STDOUT.
    - If a file provided as argument does not exists or doesn't have read permission, you need to display "**An error has occured while opening <filename>.**" on STDERR.
    - Allowed function are: **open, read, write, close**
    - **Using printf, putchar, getchar will lead to 0 / 20. You need to code those functions by yourself.**