(/)

Curriculum
SE Foundations
Average: **108.76%**

0x0F. C - Function pointers

C

By: Alexandre Gautier

Weight: 1

Project over - took place from Mar 22, 2023 6:00 AM to Mar 23, 2023 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 24.0/39 mandatory & 7.0/7 optional
- **Altogether: 123.08%**
  - Mandatory: 61.54%
  - Optional: 100.0%
  - Calculation: 61.54% + (61.54% * 100.0%) == **123.08%**

Resources

Read or watch:

- Function Pointer in C (/rltoken/yt8Q9jxzT_gyRAvnNkAgkw)
- Pointers to functions (/rltoken/wP-yWvo9IqbcQsywMmh_iQ)
- Function Pointers in C / C++ (/rltoken/dAN27S1yyBPeBa8RGfvPNA)
- why pointers to functions? (/rltoken/1vvWpH9Ux8axOLc9jPWcMw)
- Everything you need to know about pointers in C (/rltoken/G_0lQzs4LAd1e5tKhNMPiw)

Additional Resources

- Function pointers in C Programming & How to use them (/rltoken/Xt4O3p-a_3S3M82ApwVenA)

Help

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/ITYG4BLMI4_5Unpdwue2tw), **without the help of Google**:

General

- What are function pointers and how to use them
- What does a function pointer exactly hold
- Where does a function pointer point to in the virtual memory

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be compiled on Ubuntu 20.04 LTS using `gcc`, using the options `-Wall -Werror -Wextra -pedantic -std=gnu89`
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project is mandatory
- Your code should use the `Betty` style. It will be checked using betty-style.pl (https://github.com/alx-tools/Betty/blob/master/betty-style.pl) and betty-doc.pl (https://github.com/alx-tools/Betty/blob/master/betty-doc.pl)
- You are not allowed to use global variables
- No more than 5 functions per file
- The only C standard library functions allowed are `malloc`, `free` and `exit`. Any use of functions like `printf`, `puts`, `calloc`, `realloc` etc… is forbidden
- You are allowed to use _putchar (https://github.com/alx-tools/_putchar.c/blob/master/_putchar.c)
- You don't have to push `_putchar.c`, we will use our file. If you do it won't be taken into account
- In the following examples, the `main.c` files are shown as examples. You can use them to test your functions, but you don't have to push them to your repo (if you do we won't take them into account). We will use our own `main.c` files at compilation. Our `main.c` files might be different from the one shown in the examples
- The prototypes of all your functions and the prototype of the function `_putchar` should be included in your header file called `function_pointers.h`
- Don't forget to push your header file
- All your header files should be include guarded

Quiz questions

Great! You've completed the quiz successfully! Keep going! (Show quiz)

Tasks

0. What's my name

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that prints a name.

- Prototype: `void print_name(char *name, void (*f)(char *));`

```
julien@ubuntu:~/0x0e. Function pointers$ cat 0-main.c
#include <stdio.h>
#include "function_pointers.h"

/**
 * print_name_as_is - prints a name as is
 * @name: name of the person
 *
 * Return: Nothing.
 */
void print_name_as_is(char *name)
{
    printf("Hello, my name is %s\n", name);
}

/**
 * print_name_uppercase - print a name in uppercase
 * @name: name of the person
 *
 * Return: Nothing.
 */
void print_name_uppercase(char *name)
{
    unsigned int i;

    printf("Hello, my uppercase name is ");
    i = 0;
    while (name[i])
    {
        if (name[i] >= 'a' && name[i] <= 'z')
        {
            putchar(name[i] + 'A' - 'a');
        }
        else
        {
            putchar(name[i]);
        }
        i++;
    }
}

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    print_name("Bob", print_name_as_is);
    print_name("Bob Dylan", print_name_uppercase);
    printf("\n");
    return (0);
}
julien@ubuntu:~/0x0e. Function pointers$ gcc -Wall -pedantic -Werror -Wextra -std=gn
u89 0-main.c 0-print_name.c -o a
julien@ubuntu:~/0x0e. Function pointers$ ./a
Hello, my name is Bob
Hello, my uppercase name is BOB DYLAN
julien@ubuntu:~/0x0e. Function pointers$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0F-function_pointers`
- File: `0-print_name.c`

Done! Help Check your code Get a sandbox QA Review

1. If you spend too much time thinking about a thing, you'll never get it done

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that executes a function given as a parameter on each element of an array.

- Prototype: `void array_iterator(int *array, size_t size, void (*action)(int));`
- where `size` is the size of the array
- and `action` is a pointer to the function you need to use

(/)

```
julien@ubuntu:~/0x0e. Function pointers$ cat 1-main.c
#include <stdio.h>
#include "function_pointers.h"

/**
 * print_elem - prints an integer
 * @elem: the integer to print
 *
 * Return: Nothing.
 */
void print_elem(int elem)
{
    printf("%d\n", elem);
}

/**
 * print_elem_hex - prints an integer, in hexadecimal
 * @elem: the integer to print
 *
 * Return: Nothing.
 */
void print_elem_hex(int elem)
{
    printf("0x%x\n", elem);
}

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    int array[5] = {0, 98, 402, 1024, 4096};

    array_iterator(array, 5, &print_elem);
    array_iterator(array, 5, &print_elem_hex);
    return (0);
}
julien@ubuntu:~/0x0e. Function pointers$ gcc -Wall -pedantic -Werror -Wextra -std=gn
u89 1-main.c 1-array_iterator.c -o b
julien@ubuntu:~/0x0e. Function pointers$ ./b
0
98
402
1024
4096
0x0
0x62
0x192
0x400
```

```
0x1000
julien@ubuntu:~//0x0e. Function pointers$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0F-function_pointers`
- File: `1-array_iterator.c`

Done! Help Check your code Get a sandbox QA Review

2. To hell with circumstances; I create opportunities

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that searches for an integer.

- Prototype: `int int_index(int *array, int size, int (*cmp)(int));`
- where `size` is the number of elements in the array `array`
- `cmp` is a pointer to the function to be used to compare values
- `int_index` returns the index of the first element for which the `cmp` function does not return `0`
- If no element matches, return `-1`
- If size <= `0`, return `-1`

(/)

```
julien@ubuntu:~/0x0e. Function pointers$ cat 2-main.c
#include <stdio.h>
#include "function_pointers.h"

/**
 * is_98 - check if a number is equal to 98
 * @elem: the integer to check
 *
 * Return: 0 if false, something else otherwise.
 */
int is_98(int elem)
{
    return (98 == elem);
}

/**
 * is_strictly_positive - check if a number is greater than 0
 * @elem: the integer to check
 *
 * Return: 0 if false, something else otherwise.
 */
int is_strictly_positive(int elem)
{
    return (elem > 0);
}


/**
 * abs_is_98 - check if the absolute value of a number is 98
 * @elem: the integer to check
 *
 * Return: 0 if false, something else otherwise.
 */
int abs_is_98(int elem)
{
    return (elem == 98 || -elem == 98);
}

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    int array[20] = {0, -98, 98, 402, 1024, 4096, -1024, -98, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 98};
    int index;

    index = int_index(array, 20, is_98);
    printf("%d\n", index);
    index = int_index(array, 20, abs_is_98);
    printf("%d\n", index);
    index = int_index(array, 20, is_strictly_positive);
    printf("%d\n", index);
    return (0);
}
julien@ubuntu:~/0x0e. Function pointers$ gcc -Wall -pedantic -Werror -Wextra -std=gn
u89 2-main.c 2-int_index.c -o c
julien@ubuntu:~/0x0e. Function pointers$ ./c
2
1
2
julien@ubuntu:~/0x0e. Function pointers$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0F-function_pointers`
- File: `2-int_index.c`

Done! Help Check your code Get a sandbox QA Review

3. A goal is not always meant to be reached, it often serves simply as something to aim at

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Write a program that performs simple operations.

- You are allowed to use the standard library
- Usage: `calc num1 operator num2`
- You can assume `num1` and `num2` are integers, so use the `atoi` function to convert them from the string input to `int`
- `operator` is one of the following:
  - `+` : addition
  - `-` : subtraction
  - `*` : multiplication
  - `/` : division
  - `%` : modulo
- The program prints the result of the operation, followed by a new line
- You can assume that the result of all operations can be stored in an `int`
- if the number of arguments is wrong, print `Error`, followed by a new line, and exit with the status `98`
- if the `operator` is none of the above, print `Error`, followed by a new line, and exit with the status `99`
- if the user tries to divide (`/` or `%`) by `0`, print `Error`, followed by a new line, and exit with the status `100`

This task requires that you create four different files.

(/)

3-calc.h

This file should contain all the function prototypes and data structures used by the program. You can use this structure:

```
/**
 * struct op - Struct op
 *
 * @op: The operator
 * @f: The function associated
 */
typedef struct op
{
    char *op;
    int (*f)(int a, int b);
} op_t;
```

3-op_functions.c

This file should contain the 5 following functions (not more):

- `op_add` : returns the sum of `a` and `b` . Prototype: `int op_add(int a, int b);`
- `op_sub` : returns the difference of `a` and `b` . Prototype: `int op_sub(int a, int b);`
- `op_mul` : returns the product of `a` and `b` . Prototype: `int op_mul(int a, int b);`
- `op_div` : returns the result of the division of `a` by `b` . Prototype: `int op_div(int a, int b);`
- `op_mod` : returns the remainder of the division of `a` by `b` . Prototype: `int op_mod(int a, int b);`

3-get_op_func.c

This file should contain the function that selects the correct function to perform the operation asked by the user. You're not allowed to declare any other function.

- Prototype: `int (*get_op_func(char *s))(int, int);`
- where `s` is the operator passed as argument to the program
- This function returns a pointer to the function that corresponds to the operator given as a parameter. Example: `get_op_func("+")` should return a pointer to the function `op_add`
- You are not allowed to use `switch` statements
- You are not allowed to use `for` or `do ... while` loops
- You are not allowed to use `goto`
- You are not allowed to use `else`
- You are not allowed to use more than one `if` statement in your code
- You are not allowed to use more than one `while` loop in your code
- If `s` does not match any of the 5 expected operators (`+` , `-` , `*` , `/` , `%`), return `NULL`
- You are only allowed to declare these two variables in this function:

(/)
```
    op_t ops[] = {
        {"+", op_add},
        {"-", op_sub},
        {"*", op_mul},
        {"/", op_div},
        {"%", op_mod},
        {NULL, NULL}
    };
    int i;
```

3-main.c

This file should contain your `main` function only.

- You are not allowed to code any other function than `main` in this file
- You are not allowed to directly call `op_add`, `op_sub`, `op_mul`, `op_div` or `op_mod` from the `main` function
- You have to use `atoi` to convert arguments to `int`
- You are not allowed to use any kind of loop
- You are allowed to use a maximum of 3 `if` statements

Compilation and examples

```
julien@ubuntu:~/0x0e. Function pointers$ gcc -Wall -pedantic -Werror -Wextra -std=gn
u89 3-main.c 3-op_functions.c 3-get_op_func.c -o calc
julien@ubuntu:~/0x0e. Function pointers$ ./calc 1 + 1
2
julien@ubuntu:~/0x0e. Function pointers$ ./calc 97 + 1
98
julien@ubuntu:~/0x0e. Function pointers$ ./calc 1024 / 10
102
julien@ubuntu:~/0x0e. Function pointers$ ./calc 1024 '*' 98
100352
julien@ubuntu:~/0x0e. Function pointers$ ./calc 1024 '\*' 98
Error
julien@ubuntu:~/0x0e. Function pointers$ ./calc 1024 - 98
926
julien@ubuntu:~/0x0e. Function pointers$ ./calc 1024 '%' 98
44
julien@ubuntu:~/0x0e. Function pointers$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0F-function_pointers`
- File: `3-main.c, 3-op_functions.c, 3-get_op_func.c, 3-calc.h`

☐ Done? Help Check your code Ask for a new correction Get a sandbox QA Review

4. Most hackers are young because young people tend to be adaptable. As long as you remain adaptable, you can always be a good hacker

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Write a program that prints the opcodes (/rltoken/5eSu8Ohx0ddeNGmaeDo_zQ) of its own main function.

- Usage: `./main number_of_bytes`
- Output format:
  - the opcodes should be printed in hexadecimal, lowercase
  - each opcode is two char long
  - listing ends with a new line
  - see example
- You are allowed to use `printf` and `atoi`
- You have to use `atoi` to convert the argument to an `int`
- If the number of argument is not the correct one, print `Error`, followed by a new line, and exit with the status `1`
- If the number of bytes is negative, print `Error`, followed by a new line, and exit with the status `2`
- You do not have to compile with any flags

Note: if you want to translate your opcodes to assembly instructions, you can use, for instance udcli (/rltoken/jUyzrqbp0AUZBdiTKdVExA).

```
julien@ubuntu:~/0x0e. Function pointers$ gcc -std=gnu89 100-main_opcodes.c -o main
julien@ubuntu:~/0x0e. Function pointers$ ./main 21
55 48 89 e5 48 83 ec 30 89 7d dc 48 89 75 d0 83 7d dc 02 74 14
julien@ubuntu:~/0x0e. Function pointers$ objdump -d -j.text -M intel main
[...]
00000000004005f6 <main>:
  4005f6:   55                      push   rbp
  4005f7:   48 89 e5                mov    rbp,rsp
  4005fa:   48 83 ec 30             sub    rsp,0x30
  4005fe:   89 7d dc                mov    DWORD PTR [rbp-0x24],edi
  400601:   48 89 75 d0             mov    QWORD PTR [rbp-0x30],rsi
  400605:   83 7d dc 02             cmp    DWORD PTR [rbp-0x24],0x2
  400609:   74 14                   je     40061f <main+0x29>
[...]
julien@ubuntu:~/0x0e. Function pointers$ ./main 21 | udcli -64 -x -o 4005f6
00000000004005f6 55               push rbp
00000000004005f7 4889e5           mov rbp, rsp
00000000004005fa 4883ec30         sub rsp, 0x30
00000000004005fe 897ddc           mov [rbp-0x24], edi
0000000000400601 488975d0         mov [rbp-0x30], rsi
0000000000400605 837ddc02         cmp dword [rbp-0x24], 0x2
0000000000400609 7414             jz 0x40061f
julien@ubuntu:~/0x0e. Function pointers$
```

- *Note 0:* `je` *is equivalent to* `jz`

(/)

- *Note 1: depending on how you write your `main` function, and on which machine you compile your program, the opcodes (and by extension the assembly code) might be different than the above example*

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x0F-function_pointers`
- File: `100-main_opcodes.c`

Done! Help Check your code Get a sandbox QA Review