

System Programming

1st Laboratory (1st and 3rd March 2017)

C programming revision

Pointers and pointers and arrays

I

Implement a program that concatenates all its arguments into a single string. Do not use any string manipulation function.

The program arguments are passed through the **main** parameters **argc** and **argv**:

- **int main(int argc, char * argv)**
- **argv** is a vector of strings. The first string in the name of the program
- **argc** in the number of elements of argv

Argv is an array of strings. The result of this program should be stored in a single array of characters (result_str). After the construction of this array, it should be printed in the screen with a single instruction.

II

Implement a program that creates a copy of **argv** with all its contents in uppercase.

In this program the student should create a copy of the argv array and make copies of all the arguments (argv[0], argv[1], ...)

before exiting the new array (and its contents should be printed in the screen).

Pointers and pointers and functions and DDD

III

Compile the program **pointers.c** and run it.

Execute the same program in the debugger to compare the printed values with the various relevant CPU registers:

- compile with the **-g** option
- run **ddd**
- place a breakpoint (for instance in the last line)
- run the program
- printf the Program Counter (**print \$pc**)
- print the Stack Pointer (**print \$sp**)

Compare the value of the previous register to the values printed in the screen.

Why do the addresses of **a** and **b** are so different?

Observe the various printed values and fill the following table (also use the observations from DDD):

name	type	address	Segment/location
a			
&a			
b			
&b			
v			

&v			
p			
&p			
*p			
main			
&main			
exit			
&exit			

Main and function and compilation and linking IV

Look at the files **test1.c test2.c test.h prog1.c**

Try to compile the file **lib1.c** issueing the command **gcc lib1.c**

Try to compile the file **prog1.c** issueing the command **gcc prog1.c**

What happened?

How to just compile **lib1.c**?

How to create a program?

Compile the file prog1.c (and create a program) to use the **test2.c** functions.

Read <https://www.cs.swarthmore.edu/~newhall/unixhelp/compilecycle.html> to understand how compilation works.

Dynamic linking and pointers to functions V

Sometime it is necessary to dynamically select the library to load depending on some input. In this class of applications several libraries offer functions with the same interface (name and arguments), and the aplication just call one of those functions.

Observe the **prog2.c** program.

How to load one of the libraries depending on the user input?

Create two dynamic libraries:

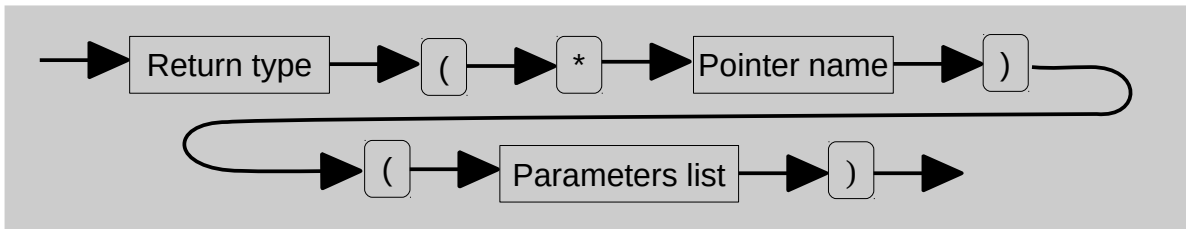
- **gcc test1.c -o test1.so -ldl -shared -fPIC**
- **gcc test2.c -o test1.so -ldl -shared -fPIC**
- two new files were created

These new libraries (and the internal functions) can be loaded using another special library:

- **man dlopen**
- **man dlsym**

The use of this library is straightforward, but requires the knowledge of pointers to functions. A pointer to function is a variable that stores the address of a function (remember the exercise III).

The syntax of a declaration a pointer to function is the following:



Example:

- `int (*compare_cb)(int a, int b)` is compatible with function `int callme(int a, int b)`
 - `compare_cb = callme;`
- and is called by `compare_cb(10, 12)`
- if preceded by `typedef` pointer name is replaced by the new type name
 - `typedef int (*type_pf)(int a, int b);`
 - `type ptr_f;`
 - `ptr_f = callme;`
- the creation of arrays of pointer of function is easy:
 - `int (*array_ptr[2])(int a, int b)`
 - `array_ptr[0] = array_ptr[1] = callme;`
 - `calling`

More information:

- <http://c.learncodethehardway.org/book/ex18.html>
- <http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html>