System Programming
1st Laboratory (24 and 26 February 2016)

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

II

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

III

Compile program **`pointers.c`** and run it. observe the various values.

Execute the same program int 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 of the previous register to the values printed in the screen.
Why do the addresses of **`a`** and **`b`** are so different?

IV

Run the command **`man strace`** and understand what it does.
Run the command **`strace ./pointers`**
Where are the printfs?

V

Look at the files **`test1.c`** **`test2.c`** **`test.h`** **`prog1.c`**
Try to compile the file **`lib1.c`** (**`gcc lib1.c`**)
Try to compile the file **`prog1.c`** (**`gcc prog1.c`**) to use the **test1.c** functions.
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.

VI

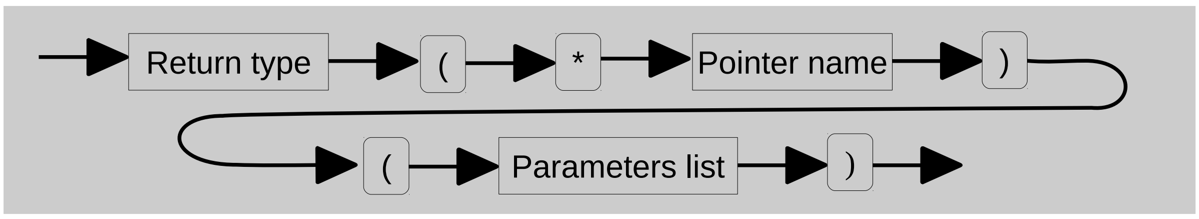
Observe the **`prog2.c`** program.
How to load one of the libraries depending on the user input?
Create two dynamic libraries:

- **`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 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