System Programming
1st Laboratory (week of 27th March 2018)

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 if this array, it should be printed in the screen with a single printf instruction.

Pseudocode:

```
while counter < argc:
     result_len += strlen(argv[argc])
create (result_str)
while counter < argc:
     append argv[argc] to  result_str
printf(result_str
```

Pointers and pointers and functions
II

Compile the program **`pointers.c`** and run it.
Observe the various printed values and fill the following table:

name	value	type
&a		
&b		
v		
&v		
&main		
&exit		
sin		
&sin		
func		
&func		
p		
&p		

- How many types of addresses it is possible to observe?
- Why does in some variables the print of its value and its address is equal? What

type of variables are those?

Main and function and compilation and linking III

Look at the files **`lib1.c`** **`lib2.c`** **`lib.h`** **`prog1.c`**

- Try to compile the file **`lib1.c`** issuing the command **`gcc lib1.c`**
- Try to compile the file **`prog1.c`** issuing 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 **lib1.c** functions.
- Compile the file **prog1.c** (and create a program) to use the **lib2.c** functions.

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

Dynamic linking and pointers to functions IV

Sometime it is necessary to dynamical 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 application just calls one of those functions.
Observe the **`prog2.c`** program.
Create two dynamic libraries:

- **`gcc lib2.c -o lib2.so -ldl -shared -fPIC`**
  - creates lib1.so
- **`gcc lib2.c -o lib2.so -ldl -shared -fPIC`**
  - creates lib2.so

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

Modify prog2.c so that the user selects what library the program will use. Follow the comments on the code.

More information:

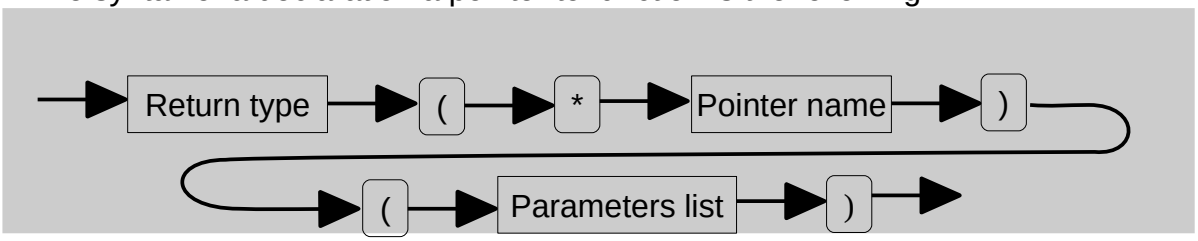
- http://tldp.org/HOWTO/Program-Library-HOWTO/dl-libraries.html
- http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html

One more program
V

Implement a program that generates 10000 random numbers and verifies if those number are multiples of 2, 3, 5 and 7.
When a number is found to be multiple it should be printed on the screen.

Pointers to functions

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



Examples:

- **`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://beej.us/guide/bgc/html/multi/morestuff.html#ptfunc