EX-1, BY 10 Nov 2021

How to do the exercises

We expect that after doing the exercises your skills in handling Linux OS and programming in C will increase. What it takes, is memorizing commands by typing the commands on the Linux terminal and observing the result. This is what the first part is about. This part is for the self-learning.

The second part is about writing a C program. At the end of the course you should be able to translate your research task into a C program based on libraries of different functions found in the Linux environment.

In this first assignment our task is to make a program that mimics the behavior of standard Linux commands by accepting the flags, parametrized flags and other arguments from the command line. To accomplish this task you will use a function from standard library. Part of the exercise is to study the manual pages related to the suggested function[1].

You write the program yourself based on the guidance we give in the assignment. You can use the C book of Kernighan & Ritchie (see Pages/Textbooks on canvas[2]). In canvas Files you have a summary of C language in slides file 3.1_Linux-C[3]

You compile and run the program as in the instructions in this assignments and solving error messages from the compiler. After you convince yourself that the program is doing what it supposed to do you copy it to directory for us to check (see below).

You pass successfully if the program is running and produces the desired result. We will give you feedback on your programming style, as this is important going forward. In program text, we ask you to comment the statements in the programs. Also, the header in form of a comment should contain your name and date of creation and a summary of the intended program behavior.

On the delivery of your results: When you are ready with your program assignment, please create a directory with your name and copy the files there:

```
mkdir /trinity/home/LINUX_SUPERCOMPUTERS/yourname/

mkdir /trinity/home/LINUX_SUPERCOMPUTERS/yourname/exe1

cp yourprogram /trinity/home/LINUX_SUPERCOMPUTERS/yourname/exe1/yourprogram
```

After you have finished copying and ready to report, please send us e-mail.If there are any problems or questions, please email me or Andrey:

"Igor Zacharov" I.Zacharov@skoltech.ru

"Andrey Kardashin" Andrey.Kardashin@skoltech.ru

[1] On Zhores it is “man 3 getopt” or on the web type into the browser “man 3 getopt”.

[2] http://cslabcms.nju.edu.cn/problem_solving/images/c/cc/The_C_Programming_Language_(2nd_Edition_Ritchie_Kernighan).pdf

[3] https://skoltech.instructure.com/files/257760/download?download_frd=1

Files and Directories manipulations

Login to your sandbox account: ssh –p 2200 yourname@sandbox.zhores.net

Get used to frequent commands in the file system

```
pwd
ls
ls –l -t
ls –a
man ls
touch
env
cat
more
less
mkdir
cd
rm
rmdir
```

Find out in which directory you are: pwd, list files: ls

Use flags on the commands, eg. ls –l or ls -ltr

See man ls to discover all flags, Especially ls –a

The –a flag lists the "hidden" files: filenames starting with "." (dot).

What hidden files do you have in your home directory (report back with it)

Create (empty) file xx: touch xx And to remove a file: rm xx

The creation with single name (eg. xx) happen in the current directory

Try absolute file path name: get it first with pwd then the result of pwd (eg. /my/result/of/pwdcmd) make file: touch /my/result/of/pwdcmd/myxx

```
xxx=`pwd`
echo $xxx
touch $xxx
```

Alternatively, try this: (the backquotes `cmd` give result of cmd)

The xxx is the environment variable

Discover all environment variables: env (report back with it) you can redirect the output with > to a file. Get the contents of a file: cat myfile

Create directory "yy": mkdir yy And to remove an empty directory yy: rmdir yy

Change your position to the newly created directory: cd yy

Change your position to one hierarchy up: cd .. (two dots)

Change to your home directory: cd (no arguments)

Editor vi (vim)

```
vi fname
```

Create an empty file with name xx (or your choice) or just type in: vi xx

Editor vi commands to enter the insert mode: i

Exit insert mode: key **Esc** on the keyboard

To write out the changed file: type in : (colon), then type in w (letter w)

To exit the editor: ZZ (Capitals) or via the command line: type :q

Watch (7m) https://www.youtube.com/watch?v=pU2k776i2Zw to learn vi.

Writing C programs

a) The first program is the "Hello, World!" program. Make yourself a new directory, change to it and edit the hello.c file in that new directory.

```
[i.zacharov@an LS_1]$ cat hello.c
#include <stdio.h>

int main()
{
        printf("Hello, World!\n");
}
[i.zacharov@an LS_1]$
```

Exit the editor and compile the program: gcc –o hello hello.c

Run the program: ./hello

TYPE IN THE PROGRAM TEXT YOURSELF! Use the editor and the compiler. Exercise your fingers.

b) Change the program to include and analyze the command line flags. The goals is to get a program which can be run as: ./hello –a –b –c where you decide what letters should be used for the (one or more) flags.

To find out about the getopt function -- Read: **man 3 getopt**

(linux or web browser)

Read the "C book" (in Canvas Pages/Textbooks)

Also in canvas Files: 3.1_Linux-C

```
[i.zacharov@an LS_1]$ cat hello2.c
/*
 * First serious assignment in the Linux & Supercomputers class
 *
 * Revision history
 * me    version 1       today
 *
 */
#include <stdio.h>
#include <stdlib.h>           //definitions for exit()
#include <unistd.h>           //definitions for getopt()

int main(int argc, char *argv[])
{
        const char flags[] = "abc";
        int opt;
        int flag = 0;

        while ( (opt = getopt(argc, argv, flags)) != -1 ) {
                switch (opt) {
                case 'a': flag = 1;
                          break;
                case 'b': flag = 2;
                          break;
                case 'c': flag = 3;
                          break;
                default :
                          printf("invalid command flags: %s\n",*argv);
                          exit(3);
                }
        }

        printf("Hello, World!  Flag = %d\n", flag);
}
```

Choose your own flags

Arithmetic assignement

Logical comparison

Convert integer

Compiler and run this program, control that flag value is printed correctly, eg. ./hello2 –a should print flag value1, etc. When flag is not part of the letters (i.e. the default selection) see the command status like this:

What is the status when no error?

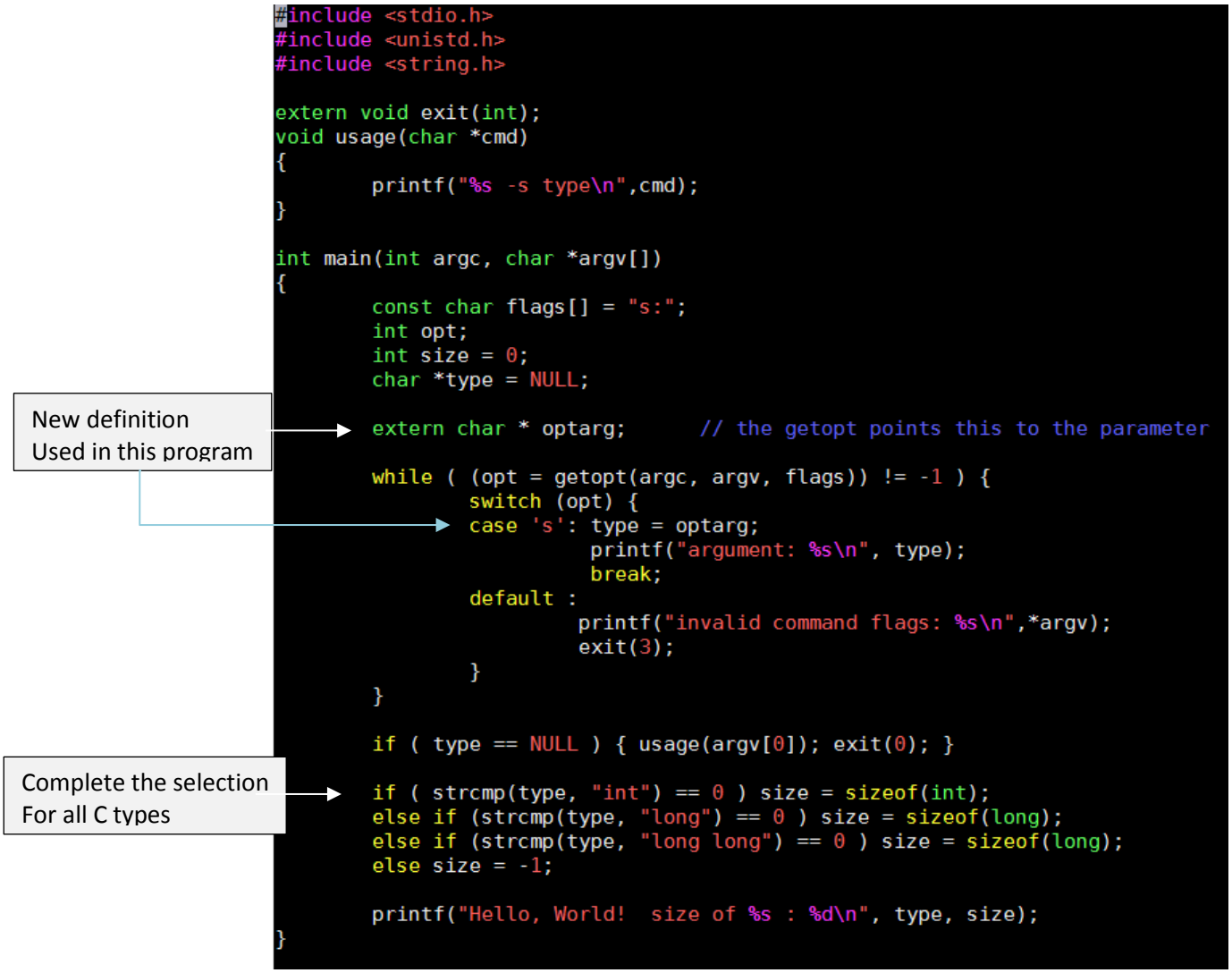
```
./hello2 –x
echo $?
```

Status of the command

c) Change program to include a parameter for a flag. The program will be run as follows:

./hello3 –a param

The parameter is used to provide information on the allocation size of the C types. For example the run ./hello3 –a long would give size of the long type.




```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

extern void exit(int);
void usage(char *cmd)
{
        printf("%s -s type\n",cmd);
}

int main(int argc, char *argv[])
{
        const char flags[] = "s:";
        int opt;
        int size = 0;
        char *type = NULL;

        extern char * optarg;       // the getopt points this to the parameter

        while ( (opt = getopt(argc, argv, flags)) != -1 ) {
                switch (opt) {
                case 's': type = optarg;
                          printf("argument: %s\n", type);
                          break;
                default :
                         printf("invalid command flags: %s\n",*argv);
                         exit(3);
                }
        }

        if ( type == NULL ) { usage(argv[0]); exit(0); }

        if ( strcmp(type, "int") == 0 ) size = sizeof(int);
        else if (strcmp(type, "long") == 0 ) size = sizeof(long);
        else if (strcmp(type, "long long") == 0 ) size = sizeof(long);
        else size = -1;

        printf("Hello, World!  size of %s : %d\n", type, size);
}
```

Complete the program for all C types, such that you can verify the table of sizes given in 3.1_Linux-C file page 3 (ILP64 model).

d) Extend the program to collect all other arguments, such as when program is running. Thus should work independent of the order, in which the arguments are given:

./hello4 –a xx yy zz And also ./hello4 yy –a xx zz Or ./hello4 yy zz -a xx

Where xx is a parameter, the yy, zz are additional names that should be printed by the program. This behaviou will be facilitated by getopt! Add the following definitions:

```
extern int      optind;
extern char * optarg;
```

Add the code to print the additional arguments:

```
if(optind < argc) printf("other arguments:");
      for ( int i = optind; i < argc; i++ ) printf(" %s",argv[i]);
      printf("\n");
```