Main Args Lab

Use the file `MainArgLabAnswers.docx` to answer questions in this lab.

Introduction

The entry point of C programs is the function `main`, which has the following prototype.

```
int main(int argc, char *argv[]);
```

When a C program is run from a shell, C strings are passed as arguments in the `argv` parameter, which is an array of `char *`. In C a `char *` is a string, which is a null terminated sequence of `char`, where a `char` is a single byte. C uses 7-bit ASCII to encode `char`s. The statement `char c = 'a';` places 97 into `c`, which is the ASCII value for `'a'`. The `argc` parameter contains the number of elements in the `argv` array. The return code is passed to the shell. A 0 indicates success. A non 0 indicates failure.

Given a C program in the file `my_program` that is invoked as follows.

```
$ ./my_program arg_a 23 another_arg
```

The values of `argc` and `argv` are as follows.

- `argc` is a `4`
- `argv[0]` is `"./my_program"`
- `argv[1]` is `"arg_a"`
- `argv[2]` is `"23"`
- `argv[3]` is `"another_arg"`

Note that the `argv[2]` parameter is the string. It is not an integer. The string can be converted to an `int` as follows.

```
int i = atoi(argv[2]);
```

Linux Commands

Linux commands such as `cat`, `ls`, `diff` are programs written in C, compiled/linked into an executable program, placed in a specific directory. For example, you may find the `ls` program in `/usr/bin/ls`. A Linux terminal is a program that draws a window with an interactive text box. The Linux terminal program executes a specific shell program such as `BASH`, `csh`, and `zsh`. The shell program maintains various environment variables. For example in my MacOS `zsh`, the environment variable `PATH` provides a list of directories that are searched for programs. When I

type `% ls` in my `zsh` terminal window, the `ls` program is found in the directory `/usr/bin/ls`. Following this paragram is a list of Linux commands demonstrating various concepts. You see the initial value of `PATH`, which is a list of system directories. You see where `zsh` finds various commands. I have created a directory `/Users/gusty/gustyscommands` in which I placed the `linkedlist` executable. I add `gustyscommands` to `PATH`, which allows me to type `linkedlist` as a Linux command. `zsh` searches the path in order. If I create a program named `cat` in `gustyscommands`, `zsh` will find it before finding the system `cat`. Note that `zsh` hashes commands creating key-value pairs. After using `cat`, `zsh` associates `cat` with `/usr/bin/cat`. When adding `/Users/gusty/gustyscommands` to the first position in `PATH`, `zsh` continues to show `cat` as `/usr/bin/cat`. The `rehash` command resets the command associations, allowing `cat` to locate the `cat` in `/Users/gusty/gustyscommands`

```
% echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
% type ls
ls is /bin/ls
% which ls
/bin/ls
% type pwd
pwd is a shell builtin
% which pwd
pwd: shell built-in command
% type cat
cat is /bin/cat
% which cat
/bin/cat
% export PATH=/Users/gusty/gustyscommands:$PATH
% echo $PATH
/Users/gusty/gustyscommands:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
% ls /Users/gusty/gustyscommands
linkedlist
% linkedlist
Print list values using a while loop.
1
111
123
… more follows this
```

See https://www.howtogeek.com/658904/how-to-add-a-directory-to-your-path-in-linux/ for more information.

Linux Command Options

Options are passed to Linux commands using `argc` and `argv`. Options that begin with a `-` are often called switches or flags. For example, the `ls` command can be invoked with the long listing switch `-l` as follows.

```
$ ls -l
```

`ls` is simply a program that a systems programmer wrote. The `ls` program uses `argc` and `argv` to process the switch.

Linux Command Options - Take One

Our first program processes switches using a for loop that examines each of the `argv` values. Consider the following code for `mainargs.c`.

```
#include <stdio.h>

int main(int argc, char **argv) {
    int nflg, sflg, uflg;
    int c = 0;
    for( ; argc>1 && argv[1][0]=='-'; argc--,argv++) {
        c++;
        switch(argv[1][1]) {
        case 0:
            break;
        case 'u':
            printf("-u flag\n");
            uflg++;
            continue;
        case 'n':
            printf("-n flag\n");
            nflg++;
            continue;
            continue;
        case 's':
            printf("-s flag\n");
            sflg++;
            continue;
        default:
            printf("invalid flag\n");
        }
        break;
    }
    printf("Loops: %d\n", c);
```

```
}
```

Experiments and Questions

Study the code and answer the following questions. You submit your answers.

1. What is printed when invoked as `$ ./mainargs -s`
2. What is printed when invoked as `$ ./mainargs -sun`
3. What is printed when invoked as `$ ./mainargs -u -u`
4. What is printed when invoked as `$ ./mainargs -u -ussr`
5. What is printed when invoked as `$ ./mainargs -a`
6. What is printed when invoked as `$ ./mainargs -a -u`
7. What is printed when invoked as `$ ./mainargs -u -a`
8. Run the program to verify your answers to the questions above. Copy/paste your run log here.
9. Experiment with other combinations of switches.

Linux Command Options - Take Two

Our second program - `maingetopts.c` - processes switches using a while loop that calls the function `getopt` to process the `argv` values. Notice how this function gives the two parameters to `main` different names - `count` and `args`. The names `argc` and `argv` are conventions, but you can choose to use different names.

```
#include <getopt.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>

#define FN_LEN 256
struct Options {
    bool using_h; // -h, human readable
    bool using_a; // -a, print all
    bool using_d; // -d, list dirs only
    bool using_f; // -f, has a file optarg
    char filename[FN_LEN]; // -f optarg
};

static void init_opts(struct Options* opts) {
    opts->using_h = false;
    opts->using_a = false;
    opts->using_d = false;
    opts->using_f = false;
    for (int i = 0; i < FN_LEN; i++)
```

```
        opts->filename[i] = 0;
}

struct Options opts;

struct Options get_opts(int count, char* args[]) {
    init_opts(&opts);
    int opt;

    while ((opt = getopt(count, args, ":f:had")) != -1) {
        switch (opt) {
            case 'h': opts.using_h = true; break;
            case 'a': opts.using_a = true; break;
            case 'd': opts.using_d = true; break;
            case 'f':
                opts.using_f = true;
                strcpy(opts.filename, optarg);
                break;
            case ':':
                printf("-f needs a value\n");
                break;
            case '?':
                printf("Unknown option\n");
                exit(-1);
        }
    }

    return opts;
}

int main(int argc, char *argv[]) {
    struct Options o = get_opts(argc, argv);
    printf("-h: %s\n", opts.using_h ? "true" : "false");
    printf("-a: %s\n", opts.using_a ? "true" : "false");
    printf("-d: %s\n", opts.using_d ? "true" : "false");
    if (opts.using_f) {
        printf("-f: %s\n", opts.filename);
    }
}
```

Experiments and Questions

Study the code and answer the following questions. You submit your answers.

10. What is printed when invoked as `$ ./maingetopts -h`
11. What is printed when invoked as `$ ./maingetopts -had`
12. What is printed when invoked as `$ ./maingetopts -hadf`
13. What is printed when invoked as `$ ./maingetopts -hadf gusty`

14. Run the program to verify your answers to the questions above. Copy/paste your run log here.
15. Experiment with other combinations of switches.

Write Program my_kill

Write a program - `my_kill.c` - where `main` processes two options (arguments with a `-`) and one argument. The two options are the following.

- `-h` - When this option is present `my_kill` prints `Hello World` to the terminal.
- `-f file` - When this option is present, `my_kill` prints the contents of the `file` to the terminal. You can assume that file is a text file.
- The argument that is not an option is a process id.

Your program is executed via the following.

```
./my_kill -h -f tiny.txt 1234
./my_kill 1234
./my_kill -h 1234
./my_kill -ftinytxt 1234
```

- `-h` - this informs `my_kill` to print `Hello World` to the terminal.
- `-f tiny.txt` - this informs `my_kill` to print the contents of `tiny.txt` to the terminal.
- `1234` is a process id.
- You can use either of the techniques for processing your command options, take one or take two for processing your options. I used command take two.
- Your program shall terminate with a -1 if it is invoked without a process id to terminate. You will have to determine this. If you use command options take two, `getopt` has a global variable `optind` that can be used.

```
if ((argc - optind) != 1) {
    printf("Error - command format is $ my_kill -options pid\n");
    exit(-1);
}
```

- Your program shall print its process id to standard output.

```
  printf("my_kill pid: %d\n", getpid()); // get my_kill's pid
```

- Your program shall call `kill` sending a `SIGINT` signal to the process id passed as an argument to `my_kill`. Include the following lines of code in your program.

```
  int status = kill(pid_to_kill, SIGINT);
  int errnum = errno;
  if (status == -1) {
      fprintf(stderr, "Value of errno: %d\n", errno);
      perror("Error printed by perror");
      fprintf(stderr, "Error killing process: %s\n", strerror( errnum
));
```

```
}
```

- Your program shall return 0 for normal completion.

Experiments and Questions

Copy this run log and submit it.

16. Run your program without arguments.
    ```
    $ ./my_kill
    Error - command format is $ my_kill pid
    ```
    Copy/paste your running here.
17. Run your program with one integer argument.
    ```
    $ ./my_kill 1234
    my_kill pid: 4823
    Value of errno: 3
    Error printed by perror: No such process
    Error killing process: No such process
    ```
    Copy/paste your running here.
18. Run your program with the `-h` option and a pid.
    ```
    $ ./my_kill -h 1234
    Hello World
    my_kill pid: 4823
    Value of errno: 3
    Error printed by perror: No such process
    Error killing process: No such process
    ```
    Copy/paste your running here.
19. Run your program with the `-h` option, `-f` option, and a pid.
    ```
    $ ./my_kill -h -f tiny.txt 1234
    Hello World
    Printing file tiny.txt
    a
    small
    file
    my_kill pid: 4823
    Value of errno: 3
    Error printed by perror: No such process
    Error killing process: No such process
    ```
    Copy/paste your running here.
20. Run your program with some combinations of options.
    Copy/paste your running here.
21. You will use `my_kill` in our Signals Lab. No answer for this question.

Questions

22. Explain `argc` and `argv` in your own words.
23. True/False - The parameters `argc` and `argv` must be those names. Justify your answer.

24. What are switches/flags?
25. What are two ways to process switches/flags?
26. What is `errno`? When do you examine `errno`? What purpose does `errno` serve?

Submissions

Be sure your submission uses the `MainArgsLabAnswers.docx` to answer the questions so I can easily decipher your answers in your submission.

1. Submit the file `MainArgsLabAnswers.docx` with your answers to the questions and your run logs from the Experiments and Questions sections.
2. Submit your code created for `my_kill.c`.