CS312 Lab 2

August 14, 2024

- Part A has to be shown in the lab on the same day and has to be submitted on Moodle before 4 pm. A plagiarism check may be done on the submissions.
- Part B is take-home. This will not be evaluated. You do not have to submit on Moodle.

Part A

1. You are given code for basic linked list operations. Make the following changes.

   (a) Modify the code to have a list of student records. Each record should be a struct having the firlds Name (string), Roll number (string) and Cgpa (float).

   (b) Make a separate library `listlib.o`, having the corresponding `listlib.c` and `listlib.h`.

   (c) Make a separate main program `listmain.c` resulting in an executable `listmain`. Make a menu-driven interface to add, delete, search and view records.

   (d) Write a Makefile that will build your program.

   (e) **Optional:** Save and read the records into secondary storage.

   (f) **Submission:** Create an archive called `lab2_rollnum.tgz` with a directory named `lab2_rollnum/`. Put all your files inside that directory before archiving. Submit the archive into Moodle.

Part B

2. Run the three assembly language programs discussed in class. Understand the C calling convention. See the reference uploaded on Moodle.

3. This question asks you to create an executable with different binaries. A toy program with strings is given. Try that first.

   You will implement a stack with an array or with a linked list. Make sure that the behavior of the stack is independent of its implementation. In other words, **the main program must be unchanged while using either type of stack.**

   Create the following:

   (a) Header file `stack.h`
   (b) Source file `arrstack.c`
   (c) Source file `llstack.c`
   (d) Source file `stackmain.c`
   (e) Makefile.

   See the below definition of a stack abstract data type (ADT).

Stack ADT

```
// create stack
void create(stack *s);
// push a char into stack
void push(stack *s, char x);
// pop the top of the stack
char pop(stack *s);
// return the top of stack, without popping
char peek(stack *s);
// is the stack empty?
int isEmpty(stack *s);
// return the size of the stack
int getSize(stack *s);
```

Implement two libraries `arrstack.o` and `llstack.o`. You have to code up `arrstack.c` and `llstack.c` as per the interface specified in the ADT.

```
# This will create arrstack.o and llstack.o
$ gcc -c arrstack.c
$ gcc -c llstack.c
```

Implement a main program `stackmain`, which can be built with either `arrstack.o` or `llstack.o`.

```
# One of the below should be used
$ gcc -o stackmain stackmain.c arrstack.o # build with array stack
$ gcc -o stackmain stackmain.c llstack.o # build with ll stack
```

The Makefile provided in the `strings` example automates these tasks. Modify it to suit your code.

The main program should work as follows.

```
$ ./stackmain
Usage: stackmain -p c|-o|-i
where -p c means push character c into the stack,
-o means pop the top of the stack, and
-i means print stack info including top of the stack, and the size.
```

The example run below pushes the characters `a,b,c,d` into the stack, performs two pops, prints info, pushes `e` and again prints info. Here we assume that the stack was built with `llstack.o`.

```
$ ./stackmain -p a -p b -p c -p d -o -o -i -p e -i
Created stack. [0] # the number in [] shows the size.
Pushed a. [1]
Pushed b. [2]
Pushed c. [3]
Pushed d. [4]
Pop, returning d [3]
Pop, returning c [2]
[Stack info: ll-based stack, top=b, size=2]
Pushed e. [3]
```

```
  [Stack info: ll-based stack, top=e, size=3]
```

Main program skeleton.

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

   /* check command line args */

   // create stack
   Stack* sp;
   create(sp);

   while there are args {
      // case push
      push(sp,x);
      // case pop
      pop(sp);
      // case info
      // you need to use peek() and getSize()
   }
}
```