S 2019
LAB 4 C Local and Global variables, recursions, string and other library functions. 2D arrays, Pointer basics.

Due: June 3 (Monday) 11:59 pm

In this lab you are going to practice using some C library functions. The simplified prototypes of the functions covered in this week's lecture and earlier are listed below:

```
<stdio.h>

printf()
scanf()

getchar()
putchar()

sscanf()
sprintf()

fgets()
fputs()
```

```
<string.h>

int strlen(s)
s   strcpy(s,s)
s   strcat(s,s)
int strcmp(s,s)
```

```
<ctype.h>

int islower(int)
int isupper(int)
int isalpha(int)
int isdigit(int)
int isxdigit(int)

int tolower(int)
int toupper(int)
```

```
<stdlib.h>

int    atoi(s)
double atof(s)
long   atol(s)
int    rand()
int    abs(int)
system(s)
exit()
```

```
<math.h>

sin() cos()
double exp(x)
double log(x)
double pow(x,y)
double sqrt(x)
double ceil(x)
double floor(x)
```

For exact prototypes of these functions, you can 1) issue '**man 3 function_name**' in the terminal. 2) look at Appendix B of the textbook.

You are encouraged to use these functions when appropriate, especially string functions declared in `<string.h>` as well as string-related IO functions declared in `<stdio.h>`. **Don't forget to include the corresponding header files**. Moreover, if you use functions declared in `<math.h>`, you need to link the library by using `-lm` flag of `gcc`.

0.0 Problem A0 scope, life time and initialization of global variables, local variables and static global/local variables

Download the files `lab4A0.c` and `cal.c`, compile them together (the order does not matter), and run the `a.out` file.

[Scope and initialization of global variables] Observe that global variables `x` and `y`, which are defined in `cal.c`, can be accessed in the main file (`x y` have global scope), and in order to access `x` and `y`, the main file needs to declare them using keyword **`extern`**.
Moreover, the output `0 0` and `1 11` shows that gobal variable `x` and `y`, which were not initialized explicitly, all got initialized to 0 by the complier. Also observe how the function `func_1`, which was defined in `cal.c`, was declared and used in the main file.

[Scope of local variables] Next, uncomment the commented block, and compile the files again. Observe the error message. The problem is that local variable `counter`'s scope is the

block/function in which it is defined, so it is not accessible to the `main` function. In our case its scope is within function `aFun`. Comment out the `printf` line, compile and run it.

[Lifetime of local variables] Observe that function `aFun` is called several times. Local variable `counter` in the function, which has life time 'automatic' – comes to life when `aFun` is called and vanishes when `aFun` returns – is created and initialized each time the function is called. Thus it always has value 100.

[Initialization of local variables] Observe the initial values of local variable `a` in `aFun`. In C and Java, if a local variable is not explicitly initialized, it is not initialized to 0 (or, more precisely, are initialized with some garbage values).

[Lifetime of static local variables] Next, make `counter` a **static** local variable, compile and run again. Observe that the value of `counter` is different in each call and its value are maintained during function calls, due to the fact that in C a static local variable has persistent life time over function calls. (Note that, a static local variable's scope is still within the block where it is defined. So `counter` is still not accessible outside the function.) Also observe that compound operator `+=` is used.

[Initialization of static local variables] Next, remove the initial value 100 for `counter`, compile and run again, and observe that in the first time call `counter` gets an initial value 0. As discussed in class, global variables and static local variables get initial value 0 if not initialized explicitly. ('Regular' local variables, as we observed above, are not initialized to 0, or, more precisely, are initialized with some garbage values).

[Scope of static global variables] Finally, make `y` in `cal.c` to be static and compile again. Observe that global variable `y` becomes inaccessible in main. (But it is still accessible later in file `cal.c` where it is defined.)

No submission for this question.

1. Problem A

1.1 Specification

In lab3 we practiced two approaches to calculating running averages, one using local variables, and one using global variables. By using global variables, communications between functions are simplified. Here we explore the 3rd approach.

1.2 Implementation

- Download file `runningAveLocal2.c`.
- Complete the function `double runningAverage(int currentInput)`, which, given the current input `currentInput`, computes and returns the running average in `double`. Notice that compared against the `runningAveLocal` function in lab3, this function takes only one argument current input and does not take current sum and input count as its arguments. In such an implementation, current sum and input count are not maintained in `main`. Instead, `main` just pass `current input` to `runningAverage()`, assuming that `runningAverage()` somehow maintains the current sum and input count info.

- do not modify or add to the code of `main()`.
- do not use any global variable. How can `runningAverage()` maintain the current sum and input count info?
  Hint: **static** can be used to local variables to make their lifetime permanent.

1.3 Sample Inputs/Outputs

Same as that for lab3 problem D1.

```
red 309 % a.out
enter number (-1 to quit): 10
running average is 10.000

enter number (-1 to quit): 20
running average is 15.000

enter number (-1 to quit): 33
running average is 21.000

enter number (-1 to quit): 47
running average is 27.500

enter number (-1 to quit): 51
running average is 32.200

enter number (-1 to quit): 63
running average is 37.333

enter number (-1 to quit): -1
red 310 %
```

Submit your program using **`submit 2031 lab4 runningAveLocal2.c`**

2. Problem B Math Library functions, simple recursions

2.1 Specification

Write an ANSI-C program that reads input from the standard input about base *b* and power *p*, and then calculate b^p.

After reading base and power from the user, the program first calls the math library function `pow()`, and then call function `my_pow()`, which is a **recursive** function that you are going to implement here.

The program keeps on prompting user and terminates when user enters `-100` for base (followed by any number for power).

2.2 Implementation

Download file `lab4pow.c` and start from there.

Your function `my_pow()` should be RECURSIVE, not Iterative. That is, the function should be implemented using RECURSION, not loop. In a recursive solution, you call the function itself with different inputs, until a base case is reached.

2.3 Sample Inputs/Outputs

```
red 117% a.out
enter base and power: 10 2
```

```
pow:    100.0000
my_pow: 100.0000

enter base and power : 10 4
pow:    10000.0000
my_pow: 10000.0000

enter base and power: 2 3
pow:    8.0000
my_pow: 8.0000

enter base and power: 2 5
pow:    32.0000
my_pow: 32.0000

enter base and power: -2 4
pow:    16.0000
my_pow: 16.0000

enter base and power: -2 5
pow:    -32.0000
my_pow: -32.0000

enter base and power: 2 -3
pow:    0.1250
my_pow: 0.1250

enter base and power: 2 -5
pow:    0.0312
my_pow: 0.0312

enter base and power: -2 -6
pow:    0.0156
my_pow: 0.0156

enter base and power: -2 -5
pow:    -0.0312
my_pow: -0.0312

enter base and power: -100 4
red 118%
```

Submit your program using **`submit 2031 lab4 lab4pow.c`**

3. Problem C String manipulations, Library functions

3.1 Specification

Develop an ANSI-C program that reads user information from the standard inputs, and outputs the modified version of the records.

3.2 Implementation

Download file `lab4C.c` and start from there. Note that the program

- uses loop to read inputs (from standard in), one input per line, about the user information in the form of `name age wage`, where `name` is a word (with no space), `age` is an integer literal, and `wage` is a floating point literal. See sample input below.

- uses `fgets()` to read in a whole line at a time. A bit introduction to `fgets` is given in problem D2.

The program should,

- after reading each line of inputs, creates a char [40] string `resu` for the modified version of the input. In the modified version of input, the first letter of `name` is capitalized, `age` becomes `age` + 10, and `wage` has 100% increases with 3 digits after decimal point, followed by the floor and ceiling of the increase wage. The values are separated by dashes and brackets as shown below.

- then duplicate/copy `resu` to `resu2` using a library function declared in `<string.h>` (how about `strcpy` or `strcat`)
- then duplicate/copy `resu` to `resu3` using a library function declared in `<stdio.h>` (how about `sprintf?`)
- then output the resulting strings `resu`, `resu2` and `resu3`.
- continue reading input, until a line of `exit` is entered.

Hints:

- When `fgets` reads in a line, it appends a new line character `\n` at the end (before `\0`). Be careful when checking if the input is `exit`.
- To tokenize a string, consider `sscanf`
- To create `resu` from several variables, consider `sprintf`.
- If you use math library functions, be aware that the return type is `double`.

3.3 Sample Inputs/Outputs:

```
red 118 % a.out
Enter name, age and wage (xxx to quit): sue 22 33.3
Sue-32-66.600-[66,67]
Sue-32-66.600-[66,67]
Sue-32-66.600-[66,67]

Enter name, age and wage (xxx to quit): john 60 1.0
John-70-2.000-[2,2]
John-70-2.000-[2,2]
John-70-2.000-[2,2]

Enter name, age and wage (xxx to quit): lisa 30 1.34
Lisa-40-2.680-[2,3]
Lisa-40-2.680-[2,3]
Lisa-40-2.680-[2,3]

Enter name, age and wage (xxx to quit): judy 40 3.2
Judy-50-6.400-[6,7]
Judy-50-6.400-[6,7]
Judy-50-6.400-[6,7]

Enter name, age and wage (xxx to quit): quit
red 119 %
```

Submit your program using **`submit 2031 lab4 lab4C.c`**

4. Problem D. 2D array, Library functions.

4.1 Specification

Write an ANSI-C program that reads user information from the standard inputs, and outputs both the original and the modified version of the records.

4.2 Implementation

A file `lab4D.c` is for you to get started. The program should:

- use a table-like **2-D array** (i.e., an array of 'strings') to record the inputs.
- use loop and `scanf("%s %s %s")` to read inputs (from standard in), one input per line, about the user information in the form of `name age wage`, where `age` is an integer literal, and `wage` is a floating point literal. See sample input below.
- store each input string into the current available 'row' of the 2D array, starting from row 0.
- create a modified string of the input, and store it in the next row of the 2D array. In the modified version of input, all letters in `name` are capitalized, `age` becomes `age` + 10, and `wage` has 50% increases and is formatted with 2 digits after decimal point.
  Hint: for converting `name` to upper cases, you might need a small loop to convert character by character. `name[i] = touppper(name[i])`
- continue reading input, until a name `xxx` is entered.
- after reading all the inputs, output the 2-D array row by row, displaying each original input followed by the modified version of the input.
- display the current date and time and program name before generating the output, using predefined macros such as __FILE__, __TIME__ (implemented for you).

Note that as the partial implementation shows, each input line is read in as three 'strings', using `scanf("%s %s %s", ….)`. In the next question, you will practice reading in the whole line as a string, as in `lab4C` (and hen 'tokenize' the string). Each approach has it pros and cons.

Note that you will lose all marks if, instead of a 2D-array, you use 3 parallel 1-D arrays -- one of names, one of ages, one for wages -- to store and display information.

4.3 Sample Inputs/Outputs:

```
red 307 % a.out
Enter name, age and wage: john 60 1.0
Enter name, age and wage: eric 30 1.3
Enter name, age and wage: lisa 22 2.2
Enter name, age and wage: judy 40 3.22
Enter name, age and wage: xxx 2 2

Records generated in lab4D.c on May 26 2019 14:58:47
john 60 1.0
JOHN 70 1.50
eric 30 1.3
ERIC 40 1.95
lisa 22 2.2
LISA 32 3.30
judy 40 3.22
JUDY 50 4.83
red 308 %
```

Submit your program using **`submit 2031 lab4 lab4D.c`**

5. Problem D2. 2D array, library functions.

Same question as problem D but now you read each line of input as a whole line of string.
Note that as discussed earlier, using `scanf("%s", inputArr)` does not work here, as `scanf` stops at the first blank (or new line character). Thus, if you enter `Hi there`, only `Hi` is read in.
As mentioned in week4's class, in order to read a whole line of input which may contain blanks, you can use `scanf("%[^\n]s",inputsArr)`, or, `gets(inputsArr)`, but a much more common approach is to use function `fgets()`. Both the functions are declared in `stdio.h`. `fgets(inputsArr, n, stdin)` reads a maximum of `n` characters from `stdin` (Standard input) into `inputsArr`.

A file `lab4D2.c` is created for you to get started.
As the code shows, reading a whole line allows the input to be read into a table row directly. So you don't need to store the original input into the table manually. The disadvantage, however, is that you have to tokenize the line in order to get the name, age and wage information.

Same output as above, except that the generated file name is `lab4D2.c` now, and the time is different.

Submit your program using **`submit 2031 lab4 lab4D2.c`**

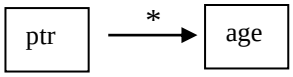
6. Problem E Pointer 101

6.1 Specification

Write your first (short) program that uses pointers.

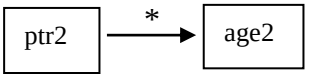
6.2 Implementation

- define an integer `age` which is initialized to 10, define another integer `age2` which is initialized to 100;
- define an integer pointer `ptr`, and make it point to `age`
- display the value of `age`, both via `age` (direct access), and via pointer `ptr` (indirect access).

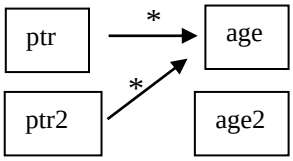


- use `ptr` to change the value of `age` to 14;
- confirm by displaying the value of `age`, both via `age` and via its pointer `ptr`

- define another pointer variable `ptr2`, and make it point to `age2`
- copy/assign `age`'s value to `age2` via pointer `ptr` and `ptr2`;
- display the value of `age2`, both via `age2`, and via its pointer `ptr2`



- now let `ptr2` point to `age` by getting the address of `age` from pointer variable `ptr` (i.,e., without using `&age`)
- confirm by displaying the value of `ptr2`'s pointee `via ptr2`
- display value of `age`, both from `age`, and via `ptr` and `ptr2`.



- use `ptr2` to decrease the value of `age` by 1.
- display value of `age`, both from `age`, and via `ptr` and `ptr2`.

- finally, display the address of `age`, using `printf("%p %p %p\n",&age,ptr,ptr2);` Notice that here we print `prt` and `ptr2` directly, which display the content of the pointer variables, which is the address of `age` (in hex).

6.3 Sample Inputs/Outputs:

```
red 305 % a.out
age: 10 10
age: 14 14
age2:14 14
ptr2's pointee: 14
age: 13 13 13
0x7ffd04a92bcc 0x7ffd04a92bcc 0x7ffd04a92bcc
red 306
```

You will get different numbers but they should be identical to each other. This is the memory address of variable `age`, in Hex.

6.4 Submission:

Name your program `lab4E.c` and submit using

`submit 2031 lab4 lab4E.c`

In summary, you should submit

`lab4runningAveLocal2.c lab4pow.c lab4C.c lab4D.c lab4D2.c lab4E.c`

You may want to issue **`submit -l 2031 lab4`** to view the list of files that you have submitted.

Common Notes

All submitted files should contain the following header:

```
/**************************************
* CSE2031 – Lab4 *
* Filename: Name of file *
* Author: Last name, first name *
* Email: Your email address *
* eecs_username: Your eecs login username *
* York num: Your York student number
**************************************/
```