W 2019 Jan 31, Feb1
LAB 4 C preprocessing, string and other library functions. 2D arrays, Pointer basics.

Due: Feb 9 (Sat) 11:59 pm

In this lab you are going to practice using some C library functions. The functions covered in week 4'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>

double atof(s)
int    atoi(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`.

---

1. Problem A. Pre-processing Macro and system()

Download the file `lab4marcoSys.c`, compile and run it.
Observe that,

- a macro SIZE is defined using `#define`, used as a constant to avoid magic number.
- a parameterized macro `CUBE(x)` is defined, to calculate the cube of parameter `x`;
- the CUBE macro works correctly for argument `i` and `j` but not correctly for argument `i+j`. What went wrong? You may want to examine how the macro CUBE is pre-processed. Issue **`gcc -E lab4marcoSys.c`**, which invokes gcc pre-processor only. Note that it is the code generated by the preprocessor (what you see here), not your original code, that is to be compiled. Look at the end of the screen, observe that:
  - `#include<stdio.h>` is replaced with the content of `stdio.h`, which is inserted before `main()`. Try to find the declarations (prototypes) of `printf, scanf, getchar, putchar.`
    One way to filter an output in Unix is to use command `grep`, which we will cover later in the course. Issue command **`gcc -E lab4marcoSys.c | grep printf`** to search for lines that contain word `printf`. You will also see declarations of `sprintf` and `fprintf` that we mentioned in class. Then issue **`gcc -E lab4marcoSys.c | grep -w printf`** to do a 'whole word only' search. Do the same search for `scanf`.

- the commented code is removed
- macro `#define SIZE` was processed (removed). `SIZE` in main() is replaced with 10;
- `#define CUBE` is processed. `CUBE (i)` is textually replaced with `i * i * i`; and `CUBE (j)` is textually replaced with `j * j * j`;
- macro `CUBE (i + j)` is textually replaced with `i + j * i + j * i + j`;

Modify the macro to fix the problem. You should get 343 for `CUPE(i+j)`. After you made modifications to macro CUBE, you might want to run **`gcc -E lab4marcoSys.c`** again to see how the modified macro CUBE is processed by the preprocessor.
You can also comment out the first line `#include<stdio.h>`, and run preprocessing again. Now the output is much shorter, since now no header file is inserted.

Next, uncomment the 2nd line `#include <stdlib.h>`, and uncomment the commented block at the end of file.
The code block calls a standard library function `system()`, whose prototype is given in `stdlib.h`. Taking as input a string `command`, which is a valid Unix command, `system` executes a Unix shell command specified in `command`.
Compile and run it. Observe that,

- the current directory is listed, and new directories `xxx` and `xxx/xxx2` were created in the current directory, and the current directory is listed again.
- predefined Macro `__FILE__`, `__LINE__`, `__DATE__`, `__TIME__`, which contain the information about the current file, current date and time, are used. These information is useful for debugging programs.

Issue commands in the terminal to verify that `xxx` and `xxx/xxx2` are generated for you. Remove these directories then (how to do that in command line?)

Submit your program using **`submit 2031Z lab4 lab4marcoSys.c`**

2. Problem B Character array, char and math library function

2.1 Specification

Standard library defines a library function `atoi`. This function converts an array of digit characters, which represents a decimal integer literal, into the corresponding decimal integer. For example, given a char array (string) `s` of "134", internally stored as `'1' '3' '4' '\0' ....`, `atoi(s)` returns an integer 134.
Implement your version of `atoi` called `my_atoi`, which does exactly the same conversion.

2.2 Implementation

Download the file `lab4B.c`. For each input, which is assume to be a valid integer literal, the program first prints it as a string, and then call `atoi` and `myatoi` to convert it, and output its numerical value in decimal, hex and oct, followed by double the value and square of the value. The program keeps on reading from the user until `quit` is entered.
Complete the `while` loop in `main()`, and implement function `my_atoi`.

Page 43 of the textbook describes an approach to convert a character array into decimal value, this approach traverses the array from left to right.
A more intuitive approach, **which you should implement here, is to calculate by traversing the array from right to left** and convert following the traditional concept of $\ldots 10^3\ 10^2\ 10^1\ 10^0$

Hint: the loop body you are going to write is different from that in the textbook.

2.2 Sample Inputs/Outputs:

```
red 127 % a.out
Enter a word of positive number or 'quit': 2
2
atoi:    2 (02, 0X2)    4       4
my_atoi: 2 (02, 0X2)    4       4

Enter a word of positive number or 'quit': 4
4
atoi:    4 (04, 0X4)    8       16
my_atoi: 4 (04, 0X4)    8       16

Enter a word of positive number or 'quit': 5
5
atoi:    5 (05, 0X5)    10      25
my_atoi: 5 (05, 0X5)    10      25

Enter a word of positive number or 'quit': 9
9
atoi:    9 (011, 0X9)   18      81
my_atoi: 9 (011, 0X9)   18      81

Enter a word of positive number or 'quit': 12
12
atoi:    12 (014, 0XC)  24      144
my_atoi: 12 (014, 0XC)  24      144

Enter a word of postive number or 'quit': 75
75
atoi:    75 (0113, 0X4B)        150     5625
my_atoi: 75 (0113, 0X4B)        150     5625

Enter a word of positive number or 'quit': 100
100
atoi:    100 (0144, 0X64)       200     10000
my_atoi: 100 (0144, 0X64)       200     10000

Enter a word of positive number or 'quit': quit
red 128 %
```

Submit your program using **`submit 2031Z lab4 lab4B.c`**

Once you finish, think about how to convert arrays that represent Oct or Hex integer literals. For example "0124" (internally stored as 0 1 2 4 \0 .....) and "0X12F" (stored as 0 X 1 2 F \0 ……).

3. Problem C String Library functions, operators, multiple files

3.1 Specification

Write an ANSI-C program that reads input from the standard input about date, and then calculate the number days that has elapsed in the year.

The program keeps on prompting user to enter `Month-Date Year` (three words separated by a dash and a space). For each input your program displays the days that have elapsed from the start of the year. Program terminates when user enters `quit`.

3.2 Implementation

Download file `lab4C.c` and start from there. The program uses `fgets()` to read in a whole line. **Don't modify the existing codes.**

- Implement the function `int countDays(int year, int month, int day)` which calculates how many days have elapsed since the start of the year. For simplicity, assume that there are 31 days in Jan, Mar, May, July, Sep and Nov, and there are 30 days in April, June, Aug, Oct and Dec. There are 29 days in Feb if the year is a leap year, and 28 days in Feb if the year is not leap year.
- Define function `isLeap(int year)` as you did for lab 2. Put the definition of this function in another file named `leap.c`.
- Implement the `while` conditions, checking if input is `quit`.
- Display the output as `xx days of year xxx have elapsed`

Thoughts and Hints:

- How to get year, month and date information from the line of string? That is, how to tokenize a string? How about `sscanf` mentioned in class?
- In implementing `countDays()`, instead of checking month with lots `if… elseif… elseif… elseif….` Or `if (||…||…||…||…||…)`, can you use an arithmetic operator to simplify the code?
- When `fgets` reads in a line, it appends a new line character `\n` at the end (before `\0`). For example, input `a line` is stored as `'a' ' ' 'l' 'i' 'n' 'e' '\n' '\0' ……` where …. are some other values (`'\0'` or random values). Be careful when checking if the input is `quit`.
- **You should not put `#include "leap.c"` in your main program in order to use isLeap() function.** The proper way is to declare the function and compile the programs together.

3.3 Sample Inputs/Outputs: (ONE blank link between each interaction/iteration):

```
red 339 % a.out
Enter 'month-date year': 1-1 2010
1 days of year 2010 have elapsed

Enter 'month-date year': 8-8 2011
220 days of year 2011 have elapsed

Enter 'month-date year': 8-8 2012
221 days of year 2012 have elapsed

Enter 'month-date year': 8-8 2016
221 days of year 2016 have elapsed

Enter 'month-date year': 10-1 2010
274 days of year 2010 have elapsed

Enter 'month-date year': 10-1 2012
275 days of year 2012 have elapsed

Enter 'month-date year': 5-4 2017
124 days of year 2017 have elapsed

Enter 'month-date year': 5-4 2012
125 days of year 2012 have elapsed
```

```
Enter 'month-date year': 2-12 2012
43 days of year 2012 have elapsed

Enter 'month-date year': 2-12 2011
43 days of year 2011 have elapsed

Enter 'month-date year': quit
red 340 %
```

Assume inputs are valid.

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

4. Problem D String manipulations, Library functions

4.1 Specification

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

4.2 Implementation

Download file `lab4D.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 `scanf("%s %s %s", name, age, wage)` to read in three input 'strings', where `name`, `age` and `wage` are of type char[20].

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.
  Hint: 1) to convert to a float with 3 digits after decimal point, review the code of `runningAveLocal.c` provided in lab3. 2) To create `resu` from several variables, consider `sprintf`. 3) if you use math library functions, be aware that the return type is `double`.
- 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 name `xxx` is entered (followed by any two words).

4.3 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): xxx 2 2
red 119 %
```

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

5. Problem E. 2D array, Library functions.

5.1 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.

5.2 Implementation

A file `lab4E.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 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 do `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__ .

Note that as the partial implementation shows, you should read in the three inputs in three separate variables, but you have the choice of how they are read in: they can be read in as three 'strings', like in problem D, using `scanf("%s %s %s", ….)`, or, you can use `scanf("%s %d %f", ….)` to read in the three inputs as string, int, float respectively.
In the next question, you will practice reading in the whole line as a string (and then '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.

5.3 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 lab4E.c on Jan 26 2019 14:58:47
john 60 1.00
JOHN 70 1.50
eric 30 1.30
ERIC 40 1.95
lisa 22 2.20
LISA 32 3.30
judy 40 3.22
JUDY 50 4.83
red 308 %
```

You should not hard-code the file name and date time.

Submit your program using **`submit 2031Z lab4 lab4E.c`**

6. Problem E2. 2D array, library functions.

Same question as problem E 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 `lab4E2.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 should be `lab4E2.c`, and the time is different.

Submit your program using **`submit 2031Z lab4 lab4E2.c`**

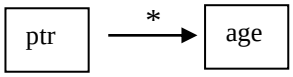
7. Problem F Pointer 101

7.1 Specification

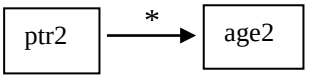
Write your first (short) program that uses pointers.

7.2 Implementation

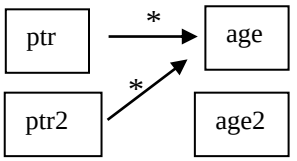
- define an integer `age` which is initialized to 10, define another integer `age2` which is initialized to 100;
- define an integer pointer `ptr` that points to `age`
- display the value of `age`, both via `age`, and via pointer `ptr`
- use `ptr` to change the value of `age` to 14;
- display the value of `age`, both via `age`, and via pointer `ptr`



- define another pointer `ptr2`, which points to `age2`
- copy/assign `age`'s value to `age2` via pointer `ptr` and `ptr2`;
- display the value of `age2`, both via `age2`, and via pointer `ptr2`



- let `prt2` points to `age` by getting the address of `age` from the first pointer `ptr`.
- 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);`

7.3 Sample Inputs/Outputs:

```
red 305 % a.out
age: 10 10
age: 14 14
age2:14 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.

7.4 Submission:

Name your program `lab4F.c` and submit using

`submit 2031Z lab4 lab4F.c`

In summary, you should submit

`lab4marcoSys.c lab4B.c lab4C.c leap.c lab4D.c lab4E.c lab4E2.c lab4F.c`

Common Notes

All submitted files should contain the following header:

```
/*************************************
* CSE2031Z – 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
*************************************/
```