CpSc 1010 ProgrammingAssignment 1
How Many Calories Am I Burning

.

Overview

This lecture programming assignment is similar to lab 6. With this program, you will gain some more experience with arrays:

- declaring and initializing arrays (1-D and 2-D arrays)
- filling up the arrays (the text and the activity level multipliers) using redirection from a file
- printing out the result of calculations using the values read in from the input file and stored in the arrays

NOTE: This program will be used for the second lecture programming assignment, so you have to get this one to work to be able to use it for asg2.c.

Background Information (same as from lab6 write-up)

- -

Strings
In lecture, you have learned that strings are character arrays that are null terminated. When printing a string (character array), use the `"%s"` format specifier with the `printf()` statement. When reading in a string from the keyboard, `scanf()` can be used with the `"%s"` format specifier as well, but you learned that `scanf()` only reads in characters up to a space, tab, or newline character. As you also learned in lecture, there are other options for printing out and reading in strings besides `printf()` and `scanf()`, some of which come from the `string.h` library file, and some from `stdio.h` library file.

2-D arrays are used to hold lines of text: the first dimension refers to the number of rows (the number of lines) and the second dimension refers to the number of columns (the number of characters) of the longest line.

- -

Redirection and `stdin`
Recall that when you run a program, the system automatically opens up three files: stdin, stdout, and stderr. There are file descriptors, or unique identifiers, for each of these: 0, 1, and 2 respectively. `stdin` refers to the keyboard, with the file descriptor of 0.

Also recall that input can be redirected from a file rather than typing everything by hand on the keyboard. The code in your program doesn't change – it still uses `stdin` as the source of input. Using redirection reassigns the source of input from the keyboard to whatever file name is specified at the command line. When you type something like the following at the command line: `./a.out < input.txt` (or whatever the input file name is), `stdin` is now reassigned to the file called `input.txt` instead of the keyboard. So, when you redirect input, the file descriptor of 0 is assigned to the file specified at the command-prompt when running the program and is no longer associated with the keyboard. This presents a problem when, later in your program, you want to get more input from the user (keyboard) using `scanf()`.

In order to do this, you would need to reset `stdin` so the keyboard will work again. You can Google this and possibly find different ways of accomplishing this, but the following line of code will work:

```
freopen("/dev/tty", "rw", stdin);
```

`/dev/tty` is a Unix file system object that represents the current terminal – think of it as a synonym for the controlling terminal that you are using. Both `stdin` and `stdout` file descriptors point to it. By using `freopen` you are redirecting an open file, or resetting `stdin` as being associated with the file descriptor of 0 for the current terminal (i.e. making `stdin` point to the keyboard again).

This will work fine in our programming environment when logged in to the Linux terminals. HOWEVER, it does not work the same on Gradescope.

So, for this lab, after you develop your program and it works on our system, right before you submit your program to Gradescope, comment out that one line of code that resets `stdin` back to the keyboard.

- -

Good programming practice – not hardcoding values whenever possible
In lecture, you also learned that you can use a `#define` statement or a `const int` variable to hold values that can be used for array sizes (dimensions) and loop limits, among other things. One advantage of doing this is that if changes in these values need to be made, the change only needs to be made on one line in the program, rather than finding every occurrence of the value and changing it throughout the entire program.

Lecture Programming Assignment

For this lecture homework programming assignment, you will write a program, called asg1.c. This program will use information in the input file along with user input to calculate the total number of calories burned for an exercise chosen from a list and based on duration of the workout and the weight of the person. The list of exercises will be read in with redirection from a file, just as you did with lab 6 and reading in the activity levels. The only big difference with this program compared to lab 6 is that the user will be allowed to continue choosing other exercises by entering 1 to go again, or to quit the program by entering a 0, so part of the code will be in a loop.

The formula is based on:

1. weight of the person (outside of the loop so that it is entered only once at the beginning of the program)
2. MET value for the chosen exercise, which is a standardized value that represents the amount of energy the body uses during the performance of a certain exercise. MET stands for metabolic equivalent for a task.
3. duration of the exercise

The formula for total number of calories burned:

```
duration * (MET * 3.5 * (weight / 2.205))/200
```

NOTE: (weight / 2.205) converts lbs to kg as the formula uses kg

The values for the formula and the MET values for different exercises were found at this web page:

https://betterme.world/articles/calories-burned-calculator/#:~:text=The%20equation%20for%20the%20Exercise,%2F%20200%20%3D%20Total%20calories%20burned

The steps below are similar to those for lab 6:

1. Using good programming practice, you can use a `#define` statement or a `const int` variable to hold the values of the number of exercises and the length of a line. Use 37 for the number of exercises (**`NUM_EXERCISES`**) and 105 for the maximum length of the lines (**`MAX_LINE`**) as there are 37 exercises in the input file, and the length of the longest line is 102 characters. By using a `#define` statement or a `const int` variable, you will not be hard-coding any array dimensions or loop conditions – it is typically preferred to not hard-code values like these, especially when that value will be used in multiple locations in your code. The `#define` statements or `const int` variables should be place right below the `#include` statements and above the `main()` function.

2. After setting up NUM_EXERCISES and MAX_LINE, the next thing your program will do is read in text from the given file `METs.txt` ***using redirection***. Copy that file from the following location:

   ```
   /home/chochri/101/Assignments/F23/asg1/METs.txt
   ```

   If you look at the contents of that input file, you will see the data in the file alternates starting with a line of text, and then a floating point value for the MET value. Each item will be read and saved into an array: one read for the activity that will be put into a 2-D array and another read for the MET value that will be put into the 1-D array. (Just as with lab 6, this will likely be the hardest part of the program, but will be done very similarly to the way you did it in lab 6.)

   Because `scanf()` with `%s` only reads in strings (characters) up to a space, tab, or newline character, and the activity levels are not all single word lines of text, something other than `scanf()` will need to be used to read these input lines into the 2-D array. There are many different ways to read in text from a file. You will want to choose one that reads the entire line up to the newline character (you can refer to the lecture slides for a hint on which one). There are different things that you have to take into consideration depending on which function you use. For example, does the function you choose copy the newline character into the array or not? What about the `\0` character?

The lines in the input file with the activity level multiplier values can be read into a 1-D array using a `scanf()` with `“%f\n”` or `“%lf\n”` (`f` or `lf` depending on which type you use for the array). The `\n` character is included in the format specifier because it must be read in to get past it so that the next read for the next line of text starts on the next line.

3. As you are working on reading in the data from the input file, you should add print statements to verify that the arrays have the information in the arrays correctly before proceeding with the rest of the program.

4. After verifying both arrays contain the correct information, add the following line of code:

```
freopen("/dev/tty", "rw", stdin);
```

   This is the line of code that you will comment out after your program works right before you submit it to Gradescope.

5. The next part of the program is where you ask the user for certain values and do the calculations, and then show the results for the total number of calories burned for the chosen exercise and duration. The first thing the program asks for is the user’s weight. Then it will show the list of exercises and allow the user to choose one of them. After the user chooses the exercise, then they will enter the duration of the exercise that they intend to do. The **sample output on the next page** will give more insight as to what values you will be asking for and the output your program should present. Some of the print statements that you will need are shown below – you can copy and paste these to use in your program.

Outside of the loop to go again:

```
printf("\nYour weight (pounds):  ");
```

All of the following would be inside the loop to go again:

```
printf("\n\n\n");
```

Other loops for various things:

```
printf("\n\nChoose exercise:  ");
```

```
printf("Duration (minutes):  ");
```

```
printf("\nThe total calories burned:  %.1lf\n\n”, totalCal);
```

```
printf(“Would you like to choose another exercise?”
       “  1 for yes, 0 for no:  ”);
```

Sample Output

```
chochri@joey4:[34]  ./a.out < METs.txt

Your weight (pounds): 145


1. Basketball, game
2. Bicycling: 12-13.9 mph (20-22km/h)
3. Bicycling: BMX or mountain
4. Bicycling: 14-15.9 mph (22-25 km/h)
5. Bicycling: 16-19 mph (25-30 km/h)
6. Bicycling: > 20 mph (>32 km/h)
7. Boxing, in the ring, general
8. Calisthenics (e.g. pushups, sit-ups, pullups, jumping jacks), heavy, vigorous
effort
9. Circuit training, including some aerobic movement with minimal rest, general
10. Fast running, 10.9 mph (17.5 km/h)
11. Football, competitive
12. Jog/walk combination (jogging component of less than 10 minutes)
13. Jogging, in place or 5 mph (8 km/h)
14. Judo, jujitsu, karate, kickboxing, taekwondo
15. Mild stretching
16. Moderate walking, 3.5 mph, (5.6 km/h) uphill
17. Rope jumping, fast
18. Rope jumping, moderate, general
19. Running, 10 mph (16 km/h)
20. Running, 8 mph (around 13 km/h)
21. Running, stairs, up
22. Soccer, casual, general
23. Softball or baseball, fast or slow pitch, general
24. Stair-treadmill ergometer, general
25. Swim. laps, freestyle, slow, moderate or light effort
26. Swimming laps, freestyle, fast, vigorous effort
27. Swimming, leisurely, not lap swimming, general
28. Swimming, sidestroke, general
29. Tennis, general
30. Volleyball
31. Volleyball, beach
32. Walking for pleasure
33. Walking, 4.0 mph, (around 6.5 km/h) level, firm surface, very brisk pace
34. Walking, 5.0 mph (8 km/h)
35. Weightlifting (free weight, nautilus or universal-type), powerlifting or
bodybuilding, vigorous effort
36. Wrestling (one match = 5 minutes)
37. Yoga


Choose exercise:  39


Choose exercise:  42


Choose exercise:  9
Duration (minutes):  45

The total calories burned:  414.3

Would you like to choose another exercise?  1 for yes, 0 for no:  1
```

```
1. Basketball, game
2. Bicycling: 12-13.9 mph (20-22km/h)
3. Bicycling: BMX or mountain
4. Bicycling: 14-15.9 mph (22-25 km/h)
5. Bicycling: 16-19 mph (25-30 km/h)
6. Bicycling: > 20 mph (>32 km/h)
7. Boxing, in the ring, general
8. Calisthenics (e.g. pushups, sit-ups, pullups, jumping jacks), heavy, vigorous
effort
9. Circuit training, including some aerobic movement with minimal rest, general
10. Fast running, 10.9 mph (17.5 km/h)
11. Football, competitive
12. Jog/walk combination (jogging component of less than 10 minutes)
13. Jogging, in place or 5 mph (8 km/h)
14. Judo, jujitsu, karate, kickboxing, taekwondo
15. Mild stretching
16. Moderate walking, 3.5 mph, (5.6 km/h) uphill
17. Rope jumping, fast
18. Rope jumping, moderate, general
19. Running, 10 mph (16 km/h)
20. Running, 8 mph (around 13 km/h)
21. Running, stairs, up
22. Soccer, casual, general
23. Softball or baseball, fast or slow pitch, general
24. Stair-treadmill ergometer, general
25. Swim. laps, freestyle, slow, moderate or light effort
26. Swimming laps, freestyle, fast, vigorous effort
27. Swimming, leisurely, not lap swimming, general
28. Swimming, sidestroke, general
29. Tennis, general
30. Volleyball
31. Volleyball, beach
32. Walking for pleasure
33. Walking, 4.0 mph, (around 6.5 km/h) level, firm surface, very brisk pace
34. Walking, 5.0 mph (8 km/h)
35. Weightlifting (free weight, nautilus or universal-type), powerlifting or
bodybuilding, vigorous effort
36. Wrestling (one match = 5 minutes)
37. Yoga


Choose exercise:  35
Duration (minutes):  90
The total calories burned:  621.4

Would you like to choose another exercise?  1 for yes, 0 for no:  2
Would you like to choose another exercise?  1 for yes, 0 for no:  5
Would you like to choose another exercise?  1 for yes, 0 for no:  0
```

Reminder About Formatting and Comments

- The top of your file should have a header comment, which should contain:
  - Your name
  - Course and semester
  - Lab number
  - Brief description about what the program does
  - Any other helpful information that you think would be good to have.
- Variables should be declared at the top of the main function, and should have meaningful names.
- Always indent your code in a readable way. Some formatting examples may be found here: https://people.cs.clemson.edu/~chochri/Assignments/Formatting_Examples.pdf
- Don't forget to use the `-Wall` flag when compiling, for example: `gcc -Wall asg1.c`

Turn In Work

After testing your program with multiple test cases, submit your `asg1.c` file to Gradescope.

Grading Rubric

For this lab, points will be based on the following:

Functionality	45	(clean compile, and works according to the instructions like lab 6, and enforces valid input)
Uses a 2-D array for strings	15	(using something other than `scanf()`)
Uses a 1-D array for METs values	10	
Allows the user to continue choosing different exercises	10	(for reading in the strings)
Use of #define or const variables	10	(for NUM_EXERCISES and MAX_LINE)
Code formatting	10	