Lab 1: An Introduction to C Programming

Exercise: Geometry Calculator Program

In addition to `main()`, you will write 11 separate functions:

```
area_rectangle()
perimeter_rectangle()
diagonal_rectangle()
area_circle()
circumference()
area_triangle()
hypotenuse()
perimeter_triangle()
exterior_angle()
interior_angle()
area_regular_polygon()
```

Sample Output:

```
Welcome to Wile E. Coyotes Geometry Calculator!

Enter the height of a rectangle as a whole (integer) number: 2
Enter the width of a rectangle as a whole (integer) number: 3

Rectangle Calculations---

The area of a rectangle with height 2 and width 3 is 6

The perimeter of a rectangle with height 2 and width 3 is 10

The length of the diagonal of a rectangle with height 2 and width 3 is
3.605551

Enter the radius of a circle as a floating point number: 2.0

Circle Calculations---

The area of a circle with radius 2.0 is 12.566371

The circumference of circle with radius 2.0 is 12.566371

Enter the height of a right triangle as a floating point number : 1.0

Enter the base of a right triangle as a floating point number : 1.0

Triangle Calculations---
```

```
The area of a right triangle with height 1.0 and base 1.0 is 0.500000

The perimeter of a triangle with height 1.0 and base 1.0 is 3.414214

Enter the number of sides of a regular polygon as an integer: 8

Enter the length of the side of a regular polygon as a floating point
number: 5.0

Regular Polygons---

The exterior angle of a regular polygon with 8 sides is 45.000000 degrees.

The interior angles of a regular polygon with 8 sides sums to 1080.000000
degrees.

Each interior angle of a regular polygon with 8 sides is 135.000000 degrees.

The area of a regular polygon with sides, each 5.0 long is 120.710678
```

Important: Your program should perform the geometry operations based on the numerical values read in from the keyboard as the bold blue text in the example.

Lab 2: An Introduction to Conditional Execution in C

Exercise: Geometry Calculator Program 2

Requirement: your program must be able to execute all the geometry calculations from Lab 1. You must:

- use a nested `switch` statement to determine which calculation to perform. You will first ask what general type of calculation (Circle, Regular Polygon, Rectangle, or Triangle) the user wants. And based on that input, determine which operation to perform, get user input to perform the calculation, and carry out the calculation.
- error check user input at every step of the way. You should accept both upper and lower case values for menu options, and since this is geometry the values the user entered should be greater or equal to zero. If an error occurs, print a message to the screen and exit the program. Depending on the error, you will either handle it with a switch or a if statement. See the sample code (below) for an example.

Sample output:

```
Welcome to Wile E. Coyote's Geometry Calculator!
Guaranteed to Pythagorize the Roadrunner in his tracks!

Please select a geometry calculation:
C. Circles
```

```
P. Regular Polygons
R. Rectangles
T. Right Triangles

Please enter your choice (C, P, R, T): C
A. Area of a circle
C. Circumference of a circle
Please enter your choice (A, C): A
Enter the radius of the circle: 2
The area of a circle with radius 2.0 is 12.566371
```

or if the user made an error in the value of the radius entered:

```
Welcome to Wile E. Coyote's Geometry Calculator!
Guaranteed to Pythagorize the Roadrunner in his tracks!

Please select a geometry calculation:
C. Circles
P. Regular Polygons
R. Rectangles
T. Right Triangles

Please enter your choice (C, P, R, T): C
A. Area of a circle
C. Circumference of a circle
Please enter your choice (A, C): A
Enter the radius of the circle: -2
Error: radius has to be greater or equal to zero
Goodbye.
```

Additional Problems

Chapter 5 of *C Programming: A Modern Approach*: Exercise #11, Programming Projects #1, #7, #8, #11.

Lab 3: Fundamental Algorithms

Exercise: Array

In addition to `main()`, you will write the following separate functions:

- `find_max()`: Find the maximum value of an array
- `find_min()`: Find the minimum value of an array
- `midpoint()`: Find the midpoint of an array

- `get_count()`: Given a value finds the number of elements in the array that are less than, less than or equal, equal, greater than or equal to, or greater than the value in the array.
  Example: counting elements which is less than or equal to 7 in an array `a` with `size` length
  ```
  int le7 = get_count(a, size, LE, 7);
  ```
- `linear_search()`: determine if a given value if in the list or not. The linear search function returns the index of the element if found; -1 if not.
- `find_avg()`: Find the average value of an array.
- `bubble_sort()`: sort the elements of the array using bubble sort algorithm. The function returns void, but your array will com back sorted.
- `insertion_sort()`: sort the element of the array using insertion sort algorithm. The function returns void, but your array will com back sorted.
- `reverse()`: reverse the array.
- `find_median()`: find the median of the elements in the array and return its value.
- `even_count()`: find the count of all even numbers in the array.
- `odd_count()`: find the count of all odd numbers in the array.
- `divisible_count()`: find the number of elements of an array that are divisible by a given number.

Exercise: Rock-Paper-Scissors-Lizard-Spock

Requirement:
Use **`switch`** statements to determine the winner of Rock-paper-scissors-lizard-Spock. See `http://en.wikipedia.org/wiki/Rock-paper-scissors-lizard-Spock` for the rules of the game.

Exercise: Luhn's Algorithm

Luhn's algorithm (`http://en.wikipedia.org/wiki/Luhn_algorithm`) provides a quick way to check if a credit card is valid or not. Write a program that implements Luhn's Algorithm for validating credit cards.

Lab 4: Arrays and Structures

Reading

C Programming: A Modern Approach. Sections 7.6, 8.1, 16.1 - 16.3

Exercise: Array Problems

In addition to `main()`, you will write the following separate functions:

- A function that multiplies each element of an array by an integer and stores the new value in the same array
- A function that adds an integer to each element of an array and stores the new value in the same array.
- A function that copies the contents of one array into a new array. Both arrays have to be the same size.
- A function that finds the sum of two arrays and writes the results to a third array.
- A function that finds the product of two arrays and writes the results to a third array. Since functions can't return arrays, you have to pass in three arrays to your function.
- A function that finds the "inverted" product of two arrays and writes the results to a third array. Since functions can't return arrays, you have to pass in three arrays to your function.
- A function that reverses the elements in a array. You will have to swap the array elements.
- A function that generates a random array of 10 numbers that are less than a given value (please use 50 as this value). Please output the array you generated.

Exercise: Array of Structures Problems

The following codes are defined in `array_struct.h`.

```
#ifndef ARRAY_STRUCT_H_
#define ARRAY_STRUCT_H_


#define SIZE 5

struct data_t {

      int age;  /* age of the subject */
      int height;  /* height of subject in inches */
      char subject;     /* one capital letter id for subject */

};

void init_array(struct data_t data[], int index, char id, int years, int
inches);


#endif
```

Please implement the following functions:

- Write the body of the function `init_array()`. The prototype is given in `array_struct.h`.
- Write a function that finds the min of the ages. Return the index.
- Write a function that finds the min of the heights. Return the index.
- Write a function that finds the max of the ages. Return the index.
- Write a function that finds the max of the heights. Return the index.
- Write a function to find the average age of the test subjects. Return the average age.
- Write a function to find the average height of the test subjects. Return the average Height.

- Write a function to print an individual structure. Returns nothing.

Exercise: Distance

Write a program that calculates the Euclidean distance and the Manhattan or taxicab distance between two two-dimensional points. Make sure you 1) use a point structure to store the points, 2) use functions to calculate the two distances and 3) ask the user for the point data.

Lab 5: Structures and Enumerations

Additional Problems in Project Euler

Problem 1, 5, 6

Lab 7: Strings, Pointers, and Dynamic Memory

Readings

Chapter 11, 12, and 13 in *C Programming: A Modern Approach*

Exercise

Write a program that reads in the following 11 integers dynamically: 4, 6, 2, 4, 9, 11, 14, 16, 1, 15, 3. Find the min, max, average and median of the list. You already did this in Lab 3, but now rewrite those functions using pointers. Print the array, the min, the max, the average and median to stdout.

Lab 8: Linked List

Exercise: Linked List

Implement a linked list of real numbers. Your program should have a menu interface and the ability to store an arbitrary number of **`double`**s - limited only by how much free memory is available for your computer.

Your menu interface must have at least the following options:

1. Enter a number
   A submenu - enter item at the head, tail, or middle of the list?
2. Delete a number
3. Print all numbers
4. Tell how many items are in the list
5. Find if a number in the list
6. Quit

In implementing your linked list, you must meet (at a minimum) the following requirements:

- When your program ends, you must remember to correctly free all dynamic memory.
- You must use Valgrind to make sure there are no memory leaks.
- You must implement the linked list correctly: it cannot have a builtin limit on the number of elements it can hold. System limits, such as available memory, will of course cap the length of your linked list.
- You must use `while`-loops to traverse the list in an iterative fashion. No recursion.
- You must have a *head* pointer that points to the first node in the list. You have to keep track of this pointer as insertions and deletions in your list may change the head of the list.
- You must have a *node* structure that holds a pointer to the next node in the list and a `double`.
- You must use `NULL` to designate the end of the list.
- You must check that `malloc` allocates memory successfully.
- You must initialize all pointers to `NULL`, except for pointers that are initialized at the time of their declaration.
- You must implement **functions** (which means passing your *head* pointer around) in order to do the following. Required function names are in parenthesis. Function parameters and return types are given below.
  - Create a new node (`create_node`)
  - Print a node (`print_node`). Both the **`double`** it stores, the address of the node, and the address `next` points to. If `next` points to `NULL`, print the string "NULL".
    The ability to print a node is useful for debugging. After you create a node using `create_node()` you can print it out to make sure the node was successfully created. You can also use it to debug your linked list, i.e. is `next` pointing to the correct node in the list?
  - Add an element to a list at the head of the list (`insert_head`)
  - Add an element to the \middle" of the list based on position. Insert the node at the desired position. A position of 1 indicates the head of the list. If position is greater than the number of nodes in the list, then the node is inserted at the tail of the list. Error check that position is greater than or equal to one. (`insert_middle`)
  - Add an element to the tail of the list. Walk down the list to find the tail. (`insert_tail`)

    - Display the contents of the list. For each node print out the **`double`**, the address of the node, and the address of the next node on a single line. This function should call `print_node` to do the printing of a node. (`print_list`)
    - Delete a node by its value and correctly free the memory occupied by that node. If there is more than one node in the list with the same value, it will delete the first one it finds. This has a number of cases: the found node is at the head, the tail, somewhere in the middle, or not found at all. You must account for cases. (`delete_node`)
    - Delete all the nodes in the list and correctly free the memory occupied by those nodes. This functions must account for the fact that it may be called when the list is empty. (`delete_list`)
    - Counts elements in the list by traversing the list. (`count_nodes`)
    - Find if a given number is in the list by searching the list by traversal and returning a pointer to the node if the number is found or `NULL` if not. If the node is found let the user know the number was found and print the node. If it is not found, let the user know it wasn't found in the list.
      For the comparison of floating point numbers use the `isgreaterequal()` and `islessequal()` functions provided in `math.h` rather than `==`. (`find_node`)
- You must return the head pointer from all functions where the head location may change.
- You must account for the fact that the list may be empty in all insert functions and you are adding the first node, so the head pointer will change.

Exercise: Collatz Conjecture

It is conjectured that if you start with any natural number n and perform the following operations for odd and even numbers you will generate a sequence of numbers that always ends in 1:

1. if n is even divide it by 2 to get n/2
2. if n is odd multiply by 3 and add 1 to get 3n + 1
3. repeat with the new value and stop once you have reached 1.

The sequence of numbers generated in this fashion is known as the hailstone sequence. See http://en.wikipedia.org/wiki/Collatz_conjecture for more info.

For example, if n = 6 the hailstone sequence is: 6, 3, 10, 5, 16, 8, 4, 2, 1. The cycle length of 6 is 9, where the cycle-length is the number of terms in the sequence.

For instance if n = 3, the output would be

```
n = 3
cycle length = 8
3, 10, 5, 16, 8, 4, 2, 1
```

Lab 9: Doubly Linked Lists

Revise Lab 8 to doubly linked list.

Lab 11: File I/O

Exercise: Hardware Inventory

You're the owner of a hardware store and need to keep an inventory that can tell you what tools you have, how many you have and the cost of each one. Write a program that read the file "hardware.dat". Use the following information to start your file:

```
Record#, Tool name, Quantity, Cost
3, Electric sander, 7, 57.98
17, Hammer, 76, 11.99
24, Jig saw, 21, 11.00
```

You could use `fgets` to read the file and store the data in the data structure you defined. Since you don't know the number of tools you are reading in, you will use `realloc` and a doubling schema rather than `malloc` to allocate memory for the array of tool structures. To parse the data of the file, you will use the functions strchr and strtok.