Homework #3
Due Friday, April 8, 11:59 PM
45 points total

Write your programs in C using the environment of your choice. Submit your code via Blackboard as a zip file. We'll discuss what files to include in class. **See the posted documentation guidelines - for this assignment some problems will have points assigned to documentation. Also please give your files meaningful file names (e.g. problem1.c or gameshow.c, etc.) instead of the default such as source.c**

1) (10 points) Inventory using Structs

Start with the inventory code using structs that is posted on the calendar page: inventory-struct.c. First, rewrite the program so it uses functions for each menu choice, the same way we did in lecture. Next, add a new menu option to sort the inventory by ID. When this option is selected the array should be sorted by ID and then printing the inventory should output the products in ascending ID order.

2) (5 points) Double pointers

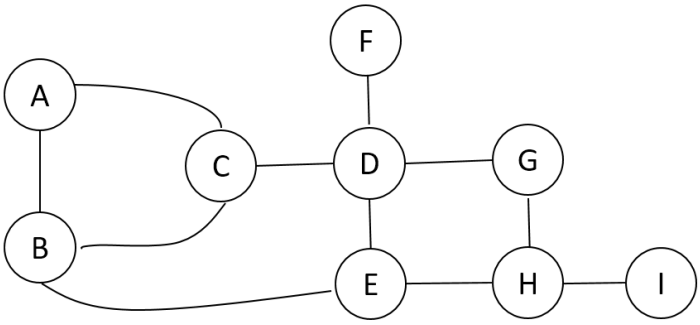
In Lab 8 part 2 you were asked to write a `concatenate` function. Instead of the function allocating memory and returning a pointer to the concatenated string, consider this version where the address of a pointer is passed in and the function should change the pointer so it points to the new memory holding the concatenated string:

```
char s1[3] = "hi";
char s2[6] = "there";
char* p = NULL;

concatenate(&p, s1, s2);
printf("%s\n", p);  // Should print hithere

free(p);
```

As in the lab, your function should work with strings of arbitrary length.

3) (15 points) Lost Underground Adventure

You awaken in a maze of twisty little passages, all alike. Can you find your way out? Here is a map of the maze:



You start in cavern A. Caverns have pathways that can (possibly) go north, south, east, or west. For example, A can go east to C or south to B, but is blocked to the west and north. From C you can go north to A, south to B, or east to D. You escape the maze by making it to cavern I.

Create a typedef Cavern struct that has a char variable that stores the cavern letter and four pointers to Cavern structs that represent the Caverns to the north, south, east, and west. If there is no passageway in one of those directions then the pointer should be NULL. Write a program that links together the Caverns as depicted in the map (when you link objects in such a manner it is called a *graph*). For example, following the `east` pointer from the struct representing Cavern A should point to the struct representing Cavern C. You are free to add any other variables in the struct that you like.

Allow the user to input N,S,E,W (or n,s,e,w) to move in directions that are allowed. Stop if the user makes it to I. Your final program should not output any of the passage letters. Here is a sample output:

```
You awaken in a maze of twisty little passages, all alike.
You can go: South East
S
You are in a maze of twisty little passages, all alike.
You can go: North South East
S
You are in a maze of twisty little passages, all alike.
You can go: North East West
E
You are in a maze of twisty little passages, all alike.
You can go: North East West
E
You found the exit!
```

4) (15 points) ASCII Rendering

Start with the bitmap library and sample code posted on the calendar page. Your job for this program is, instead of processing an image and saving it to a new bmp file, to render the bitmap in ASCII text and output it to the screen. The basic idea is to find the average greyscale value in a 5x8 block of pixels (I didn't use a square block of pixels because characters typically have a 1.5 aspect ratio; they are taller than they are wide). The greyscale value for the 5x8 block should be represented with an ASCII character that has a corresponding amount of brightness to it. The string we use for brightness is the following:

```
char *brightness=" _.,-=+:;cba!?0123456789$W#@N";
```

When used with a black background, the first character in the string is a blank, which is all black. The second character is a _ which is mostly black, and so on. The last character, 'N', is mostly white. So if the average greyscale value in the 5x8 block is 0 then the first character of ' ' should be output to represent it, if the average greyscale value in the 5x8 block is 255 then the last character of 'N' should be output to represent it, and everything in between should be scaled based on the range of 0-255 to one of the characters in the string.

Steps:

- Modify the main code so you loop through the image in 5x8 chunks, i.e. 5 pixels wide and 8 pixels tall. For all 40 pixels in this block, calculate the greyscale value of each pixel as (R+G+B)/3. Then for each greyscale value, calculate the average for all 40 pixels in the block.
- You can ignore extra pixels on the right and bottom that don't form a complete block, for example, if the bitmap is 13 pixels wide and 8 pixels tall then you could process the leftmost two 5x8 blocks and ignore the 3 pixel wide block on the right. You can do the same if there are blocks on the bottom less than 8 pixels tall.
- For each average greyscale value for a 5x8 block output the value from the `brightness` array, mapped proportionally so that the darkest possible color is the first character and the brightest possible color is the last character in the string. In other words, scale the values 0-255 to entries in the array.
- Output a newline after each row to print the image in ASCII. If your console window is not large enough and results in word wrapping then obviously the ASCII image will look jumbled.

This results in a fairly crude rending from the bitmap but you should generally be able to make out key features. Here are some images you can try along with the results from my own program:

- bridge.bmp, ASCII bridge
- dog.bmp, ASCII dog
- smily.bmp, ASCII smily

If you want to experiment you can get higher resolution ASCII if you shrink the block size and zoom out/shrink the font on the ASCII image, which will need to be a higher dimension.