Problem Set 2: Looping, Arrays, Strings

Please send back via NYU Brightspace

- A zip archive named as
  `PS02_<your name as Last_First>.zip`
  containing all C code files and answers to all questions.

For each problem of this set, you are asked to write a C program and answer some questions.

Look up how to use any of the functions mentioned here online if needed. Look at the example code to determine what #include <>'s are needed (e.g. #include <string.h>) in addition to #include <stdio.h>.

Total points: 100

Problem 1 (20 points): lexical scoping

`scope1.c, scope2.c`

1 Program **`scope1.c`**. In your `main()` function, put the following code:

```
int i;
for (i = 1; i <= 10; i++) {
    printf("%d\n", i);
}
printf("%d\n", i);
```

a What does it print on the last line?
b Why does it print that number?

2 Program **`scope2.c`** as above. This time, however, replace the second line with the following code (notice the added `int`):

```
for (int i = 1; i <= 10; i++) {
```

a What does it print on the last line this time?
b Is this number different? Why (or why not)? What does inserting the `int` within the loop initialization do?

Problem 2 (20 points): String Operations and Buffer Overrun

`buf1.c, buf2.c`

Buffer overrun, or buffer overflow, refers to the situation where an array is accessed at an index that is out of bounds, and is a typical example of "illegal" code. This problem uses variations of an example C program to show how buffer overrun can happen using the `string copy` and `concatenation` functions.

1 Program **`buf1.c`**. The `strcpy()` function accepts two strings and copies the content of the second at the start position of the first, thus over-writing anything in the first. The `strcat()` function accepts two strings and copies the content of the second to end of the first. In a C the end of a string is always indicated by the terminating "NULL" (i.e. zero-value) character. Create **`buf1.c`** from the following code and verify that the program runs as expected.

```
#include <stdio.h>
#include <string.h>
#define DLEN 32
#define _FORTIFY_SOURCE 0

int main(void) {
    char dest[DLEN];

    strcpy(dest, "C programming");
    printf("dest: %s\n", dest);

    strcat(dest, " is so much fun");
    printf("dest: %s\n", dest);

    return 0;
}
```

2 Program **`buf2.c`**. Copy buf1.c to a new program **buf2.c** and change the second string to `" is really, really fun"` and test if the program runs as expected.

2.a What happens when you execute **buf2.c**?

2.b When you concatenate the second string and the first, what it the length of the new string? How does that compare to the length of the dest[] array?

Hint: Iteration can be useful to draw a "picture" of the character array `buf[]` and how it is filled as your program executes. Don't forget that it always needs a NULL character at the end in order to be a valid C-language string. For example, after executing the `strcpy()` statement in buf1.c, the character array `dest[]` would contain the following, where the "SP" is the space character and "NULL" is the Null character.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…	29	30	31
C	SP	P	r	o	g	r	a	m	m	i	n	g	NULL		…			

Problem 3 (25 points): ASCII value conversion and place value

my_atoi.c
Write a function that converts a string of numerals to an integer value.

Type the following lines into your **my_atoi.c.**

```
#include <stdio.h>
#include <string.h>

int my_atoi(char *s) {
    int n = 0;
    // Your code here


    return n;
}

int main() {
    char *s1 = "153";
    char *s2 = "23";
    printf("The sum of %s and %s is %i\n", s1, s2,
         my_atoi(s1) + my_atoi(s2));
    return 0;
}
```

Add additional lines of code in the function `my_atoi()`, after the "Your code here" comment in order to convert the string `s` into an integer value `n`. Remember that a numeral string representing an integer has "place value" i.e. the rightmost in "units", one position left is "tens", two positions left is "hundreds" and so on. Consider:

- If the input string is "153" then the numerical value is

```
n = 1*100 + 5*10 + 3;
```

Which is to say, each "place" has a different place-value factor (i.e. 100, 10, 1).

- The length of the string "153" is 3

```
len = strlen("153");
```

- The numerical value of "5" is

```
v = '5' – '0';
```

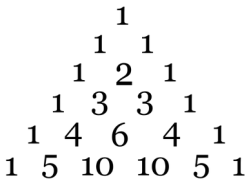
Since `'5'` and `'0'` are ASCII encoded character values, the computation in this statement is

```
v = 53 - 48;
```

or

```
v = 5; // (which is what we want)
```

Problem 4 (35 points): Pascal's Triangle and Multi-Dimensional Arrays



pascal1.c, pascal2.c

The figure above shows Pascal's Triangle, where the leftmost and rightmost entries of each row are 1 and the others are the sum of the two numbers that are the nearest upper left and upper right neighbors. We are going to make a program that prints a skewed version of Pascal's Triangle:

```
1
1  1
1  2  1
1  3  3  1
```
(... etc)

In this problem, we will build a Pascal's Triangle, using a two-dimensional array:

```
int pascal[16][16];
```

You can build nested for-loops that calculating the numbers using:

```
pascal[i][j] = pascal[i-1][j-1] + pascal[i-1][j]
```

In order to align the numbers, use format string `"%5d"` to print the number with padded spaces.

1 Program **`pascal1.c`**. Write a program that uses nested for-loops to print Pascal's triangle from row 0 to 15. The numbers should be aligned to the left margin, as shown above.

2 Program **`pascal2.c`**. Instead of printing numbers, modify your program to print `"*"` if the number is odd, and `" "`, a space, if even. Print row 0 to 31.

2.a What patterns do you see?

HINTS:

- An outer `for`-loop will iterate over the row index `i`, from `0 up to N, the number of rows in the triangle`.
- An inner for-loop index will iterate over the column index `j`, from `0` up to and including `i`.
- Inside the inner loop, check
    - If j is 0 or j is equal to i, then set to 1.
    - Otherwise, set equal to the sum of values above left and above right.
- In (2), make use of the modulus operator (i.e. % 2) and the conditional operator.