**The University of Chicago**
Department of Computer Science

***CMSC 15200 – Introduction to Computer Science 2***
***Summer Quarter 2006***
***Homeword #3 (08/02/2006)***
***Due: 08/04/2006 @ 5pm***

*Name:*

*Student ID:*

*Instructor:* Borja Sotomayor

*Do not write in this area*

| 1 | 2 | 3 | 4 | 5 | 6 | **TOTAL** |
|---|---|---|---|---|---|---|
| | | | | | | |

Maximum possible points: 45 + 10

## Exercise 1 <<10 points>>

[Use C-Strings in this exercise]

Write a program that asks the user to enter two strings (max length: 25 characters). First, the program will check if the first 10 characters are the same (if the strings have length <=10, then you simply have to check if the strings are equal). If they are not, then check if the second string is contained in the first string. Clue: There are `cstring` functions that will do this for you.

```
Enter string #1: Hello, world of C!
Enter string #2: Hello, world of C++!

The first 10 characters of the strings are equal


Enter string #1: This toffee is scrumptious.
Enter string #2: scrump

The first 10 characters are not equal, but string #2 is contained in string #1.
```

## Exercise 2 <<5 points>>

Rewrite exercise 1 using STL strings.

## Exercise 3 <<15 points>>

[Use C-Strings in this exercise]

Write a program that asks the user to enter a number *x*. The program will then ask the user for *x* words (max length: 25 characters). Next, the program will show the words with all characters in uppercase (without modifying the original strings). Finally, the program will concatenate all the strings into a single string (with a blank character between each word) and count the number of uppercase and lowercase characters.

Hint: `cctype` library.

```
How many words do you wish to enter? 4
Enter word #1: Nitwit
Enter word #2: Blubber
Enter word #3: Oddment
Enter word #4: Tweak
```

```
Uppercase:
NITWIT
BLUBBER
ODDMENT
TWEAK

Concatenated:
Nitwit Blubber Oddment Tweak
Uppercase: 4
Lowercase: 21
```

## Exercise 4 <<5 points>>

Rewrite exercise 3 using STL strings.

## Exercise 5 <<10 points>>

Write a program that asks the user to enter a number *x*. The program will then ask the user for *x* numbers. Next, the program will ask the user to enter a position from 1 to *x.* If the specified position is valid, the program will print out the value in that position. Otherwise, an error message will be displayed.

```
How many numbers do you wish to enter? 5
Enter number #1: 10
Enter number #2: 56
Enter number #3: 34
Enter number #4: 5
Enter number #5: 103

What position do you wish to access (1-5)? 3
Number #3 is 34

What position do you wish to access? 50
50 is not a valid position.
```

Note on implementation: You *must* write this program implementing the following function:

```
int getValue(??? array, int numElements, int pos, ??? value);
```

You will need to decide what the parameter type should be for `array` and `value`.

Parameters:

- `array`: The array specified by the user.
- `numElements`: The number of elements in the array (number *x* specified by the user)
- `pos`: Array position to access
- `value`: Output parameter where the value is to be deposited.

Return:

- 0: If the specified position is valid.
- 1: If the specified position is not valid.

## Exercise 6 <<Extra credit: 10 points>>

Take exercise 2 from homework #2, and rewrite the number-guessing part of the program as a standalone program. Once you've done that, add the following (5 points each):

- The program must accept a command-line parameter to specify the upper bound for the range of possible numbers. For example, if the user runs the program like this...

  ```
  ./guess -n 150
  ```

  ... a number will be picked between 1 and 150.

- It is always possible to guess the number in at most $\lceil \log_2(n) \rceil$ guesses, where *n* is the number of possible numbers (+3 bonus points if you can tell me why this is so). So, it is possible to control the game's level of difficulty by limiting the number of guesses to some fraction of $\lceil \log_2(n) \rceil$. You must add a command-line parameter to specify the difficulty of the game: easy ( $\lceil 0.8 \cdot \log_2(n) \rceil$ guesses), medium ( $\lceil 0.6 \cdot \log_2(n) \rceil$ guesses), hard ( $\lceil 0.4 \cdot \log_2(n) \rceil$ guesses), or unlimited guesses:

  ```
  ./guess -d easy
  ./guess -d medium
  ./guess -d hard
  ./guess -d unlimited
  ```