Week 6 Exercises : For Loops

Feb 17th 2012

1 Approach

Always read *all* of this document before you start the first exercise. Do this before the Tuesday lab session so that you know what your workload for the week will be. If you are aiming for a high grade then do both the basic exercises and those marked with a star. If you are having difficulty with the pace on the course then skip the starred exercises. The exercises are now roughly divided into three; these match the Tuesday / Wednesday / Thursday lab sessions. Make sure you approach the work in these batches: on a given day ask the questions that you need to understand how to do those exercises — finish them between classes.

2 Additional definitions

In this week we are going to introduce some additional notation to ease the input and manipulation of several numerical values into one single variable. This topic (Arrays) will be covered in detail in the future. In the previous weeks, you have used the [ ] operator to access individual characters of strings, now we will do exactly the same to access individual numbers from arrays of numbers. The definition for an array of numbers has some new syntactic elements:

```
datatype variableName[N];
```

This variable declaration will create an array of N elements of type datatype that can be accessed in the same way as the characters can be accessed in strings, using the [ ] operator. Please note that N is a constant expression (a constant number known before compiling the program).

Example for integers:

```
int list[10];
list[0] = 1;
list[9] = 4;
cout << list[9];
```

In C/C++, the contents of variables defined inside main() or other functions are undefined, meaning that until we assign some data to these variables, it is not safe to assume what is the value stored in them.

Additionally, if we need to create an array but we do not know in compile time how many elements are going to be needed, we can use a slightly different notation to create this array. This technique is called *dynamic memory allocation*.

```
datatype *variableName = new datatype[size];
```

The new syntax elements to note here are the *(star) and the word *new*, and we will cover them in detail in the future. There is no difference in the way we use these two types of variables after they have been created.

```
int size;
cin>>size;
int list1[10];
int *list2 = new int[size];
list1[5] = 123;
list2[5] = 123;  // assuming size is >5
```

Arrays can be used with other datatypes, such as *char* and string, and they can be accessed in a similar way. The following code snippet will define one string and one array of 10 strings. Then it will read a line using the function getline() into the single string, and then assign the first 4 characters of the string *line* to the first string in the array of strings.

```
string line;
string arrayStrings[10];

getline(cin,line); // user typed: "Hello world"
arrayStrings[0] = line.substr(0,5);
cout << arrayStrings[0]; // output: "Hello"
```

3 Exercises

In this first part of the lab we will work with small problems that can be put together to construct a sorting algorithm. An algorithm that can process an array of values unordered and reorder them based on a particular ordering criteria. Think about the tasks that repeat on each problem, and try to use a for-loop to implement them instead of repeating code as in the previous weeks.

1. Input ten integers from the standard input into an array, then find the smallest number in the array using a for-loop and print it to the standard output.

2. Input ten numbers from the standard input into an array, find the smallest number as in the previous task, and swap (interchange) this number with the first number in the array. Print to the standard output all the elements in the array in one single line separated by a space and verify the changes.

   Example:

   ```
   input: 33 42 87 12 4 90 105 34 54 99
   output: 4 42 87 12 33 90 105 34 54 99
   ```

3. Input ten numbers as the previous exercise, and now find the smallest number considering only the positions from 1 to 9 in the array (remember that 0 is the first position). Swap the number found with the number in position 1, and print again all the values to verify the changes.

   Example:

   ```
   input:  33 42 87 12 4 90 105 34 54 99
   input:  33 4 87 12 42 90 105 34 54 99
   ```

4. Think carefully about the previous 3 tasks, and write a program that inputs 10 numbers as before, and sorts in ascending order the whole array of numbers by repeating the process of finding/swapping the minimum number. Before coding your solution, work out in paper with a small array of 4 or 5 numbers, what are the steps that your program should make in order to rearrange the numbers to produce the result.

5. *In the previous exercises you have worked with small arrays, and input the values one by one by hand. Create a program that:
   - reads an arbitrary size n for the array from the user input.
   - creates a variable array using dynamic memory allocation of size n.
   - fills in the array with random integers.
   - computes the arithmetic mean and standard deviation of the elements of the array.

In this second part of the lab we will work with variable sized problems, and we will reuse the sorting algorithm that we developed in exercise 4. Review the introduction examples using readline() for these exercises.

6. Input a line of text using the function getline() into a string as explained in section 2. Scan the text character by character, and output each word in the sentence and its length. Example:

```
input: CPP is getting more interesting!
output:
CPP,3
is,2
getting,9
more,4
interesting!,12
```

7. Input a line of text into a string using the function getline(). Use 26 counters to count the frequency (number of occurrences) of each letter ('a' and 'A' counts as the same). Output to the standard output each letter and its frequency in one line each.

8. *Input a line of text into a string and split the line of text into words. Reuse the logic of the exercise 3 to sort the array of strings that you have created.
   a) Use the criteria of word length to sort them in shortest to largest order, and output the list of words reordered.
   b) Modify the ordering criteria, to order the words using alphabetical ordering ('stack' goes before 'student') and output the list of words reordered.

The last part of this lab will introduce you to conceptual mappings between data, and different ways of representing knowledge. For example, if we want to represent if a particular number has a property or not, we can use a boolean variable to represent such knowlege. Storing the booleans for such property in the positions of an array that correspond to the actual numbers, creates this implicit connection with the numbers and the property.

9. In this exercise we are going to write a simple algorithm to find all the prime numbers from 0 to n. We will use an auxiliary array of bools, to hold additional information about each number (prime or not). This is establishing an implicit relationship between numbers and positions in the array. If the position i in the array of bools is true it means that the number i is prime, otherwise is not prime.

   These are the steps of the algorithm:
   - Read the maximum number n from the user input.
   - Create a dynamic array of *bools* (bool datatype) of size n.
   - Set all the values in the array to the value *true*.
   - For each number x between 1 and n, set *every multiple* of x to false in the array of bools.
   Print the prime numbers between 1 and n.

10. *Reuse your previous program to create a new program that prints out the prime decomposition of numbers. The program should read the maximum number n, and then stay in an infinite loop performing the following:
    - read a number ($< n$).
    - if the number is -1: exit the program.
    - else: print out its prime decomposition.

    Use the following algorithm to find the prime factors (not the full prime decomposition):

    ```
    0. Reuse the code from the previous exercise to produce the array of n bools
    1. Add an array of n empty strings
    2. For each number x from 1 to n do:
       a. loop through the prime numbers p in the array of bools
       b. if x can be divided by p then append p to the string in the position of x
    ```

    The result should be the array of strings containing the prime factors for each number in a string: Example:

    ```
    For the numbers 1,2,3,4,5,6,7,8,9,10
    ["1", "2", "3", "2", "5", "2.3","7", "2", "3", "2.5"]
    ```

    Now, if the user inputs 8, the output should be: 2
    if the user inputs 6, the output should be: 2.3