EECE 230 Introduction to Programming, Sections 3, 4, and 12
Quiz II

Dec 11, 2012

- The duration of this exam is 2 hours and 45 minutes.
- It consists of 4 problems. The total number of points is 100.
- The exam is open moodle. You can use all the material on Moodle: lecture notes, programming assignments, and solutions, etc. You are **NO**T allowed to use the **web** (**imail** included). You are not allowed to use **USB's** or files previously stored in your **account**.
- If you violate the above rules or if you communicate with a person other than the exam proctors during the exam, you will immediately get zero and you will be referred to the appropriate disciplinary committee.
- Active cell phones and any other unauthorized electronic devices are absolutely not allowed in the exam rooms. They should be turned off and put away.
- Plan your time wisely. Do not spend too much time on any one problem. Read through all of them first and attack them in the order that allows you to make the most progress.
- Submit your solutions each part in a separate file as indicated in the booklet. Include your name and ID number in each file. Submit the files online in a single zip file called $abcMN.zip$, where $abcMN$ is your AUB user i.d., e.g., $abc01.zip$.
- Good luck!

Problem 1 (20 points). Files

a) **(10 points).** Write a program which creates a new file called "xyz.txt", prompts the user to an enter a string s, and stores s in "xyz.txt" (don't worry about white-spaces in s).

Submit your code in a file called Prob1a.cpp including your name and ID number.

b) **(10 points).** Write a program which asks the user to enter the names of two text files. You program is supposed to check whether or not the files are identical and display the following messages:

- "Identical files" if they two files consist of exactly the same characters
- "Files not identical" if they differ by at least on characters
- "First files does not exist" if the first file does not exist
- "Second file does not exist" if the second file does not exist.

Submit your code in a file called Prob1b.cpp including your name and ID number.

Problem 2 (30 points). Range, Frequency, and fastSort

In this problem we are interested in arrays consisting of integers restricted to the range $0, 1, 2, \ldots, 9$.

a) **(10 points) Check range.** Write a function *checkRange*, which given an array A of integers (and the size n of A), checks whether or not all the integers in A are in the range $0, 1, 2, \ldots, 9$.

The return type of *checkRange* is *bool* (i.e., YES/NO value).

Examples:

- On $\langle 20, 10, 3, 10, 5, 10, 6\rangle$, *checkRange* returns NO
- On $\langle 9, 0, 3, 0, 5, 8, 3, 1, 1, 1, 3, 8, 7, 6, 8, 0, 3, 5, 3, 6, 6, 3\rangle$, *checkRange* returns YES

Test your function on the above examples. Check the test program below.

b) **(10 points) Find frequency.** Write a function $frequency$, which given:

- an array A of integers containing integers restricted to the range $0, 1, 2, \ldots, 9$ (and the size n of A), and
- an array F or size 10.

stores in F the number of times the integers $0, 1, 2, \ldots, 9$ appear in A. That is, the function $frequency$ is supposed to store in $F[k]$ the number of times the integer k appears in A, for $k = 0, 1, 2, \ldots, 9$.

Example: If $A = \langle 9, 0, 3, 0, 5, 8, 3, 1, 1, 1, 3, 8, 7, 6, 8, 0, 3, 5, 3, 6, 6, 3\rangle$, $frequency$ should set F to $\langle 3, 3, 0, 6, 0, 2, 3, 1, 3, 1\rangle$.

Before processing A and building F, the function $frequency$ is supposed to call the function *checkRange* in (a). If *checkRange* returns $false$, $frequency$ is supposed to return $false$ without attempting to store values in F. Otherwise, it is supposed to fill F and return *true* when done.

Test your function on the above example. Check the test program below.

c) **(10 points) Fast sorting of bounded arrays.** In this part we are interested in sorting arrays consisting of integers restricted to the range $0, 1, 2, \ldots, 9$. We want to do it without nested loops (as in Insertion Sort and Selection Sort) or recursion (as in Merge Sort and Quick Sort).

Write a function $fastSort$, which given:

- an array A of integers containing integers restricted to the range $0, 1, 2, \ldots, 9$ (and the size n of A), and
- an array B of size at least n (assume memory is preallocated for B)

sorts A in nondecreasing order and stores the sorted version in B. The function $fastSort$ is not supposed to modify A. Nested loops and recursion are not allowed. (*Hint:* Use Part (b)).

Example: If $A = \langle 9, 0, 3, 0, 5, 8, 3, 1, 1, 1, 3, 8, 7, 6, 8, 0, 3, 5, 3, 6, 6, 3 \rangle$, $fastSort$ should store in B the ordered sequence $\langle 0, 0, 0, 1, 1, 1, 3, 3, 3, 3, 3, 3, 5, 5, 6, 6, 6, 7, 8, 8, 8, 9 \rangle$.

As in the previous parts, $fastSort$ returns $false/true$ depending on whether or not A contains elements outside the range $0, 1, 2, \ldots, 9$ (without processing B if $false$).

Test your function on the above example. Check the test program below.

Use the following test program

```
... printArray prototype
... checkrange prototype
... frequency prototype
... fastSort prototype

int main()
{

int A1[] = {9, 0, 3 , 0, 5,8,3,1,1,1,3,8, 7 ,6, 8, 0, 3, 5, 3,6,6,3}; // size of A1 is 22
int A2[]={ 20, 10, 3 , 10, 5,10, 6};// size of A2 is 7

cout<<"Testing checkRange:"<<endl;
... call the function checkRange  on A1 and print its return value
... call the function checkRange  on A2 and print its return value
cout<<endl;

cout<<"Testing frequency:";
int F1[10];
int F2[10];
... call the function frequency  on A1 and F1
... if it returns true, print F1
... call the function frequency  on A2 and F2
... if it returns true, print F2

cout<<" Testing fastSort:"<<endl;
int B1[100];
... call the function fastSort on A1 and B1
... print B1

return 0;
}

... implement the functions
```

Submit your code in a file called Prob2.cpp including your name and ID number.

Problem 3 (25 points). Quotient

Recall the definition of quotient and remainder. If x and y are nonnegative integers, dividing x by y results in the quotient q and remainder r, where q and r are nonnegative integer such that $0 \leq r < y$ and $x = q * y + r$.

Examples:

- the quotient of 20 divided by 3 is $q = 6$ $(20 = 6 * 3 + 2)$,
- the quotient of 20 divided by 30 is $q = 0$ $(20 = 0 * 30 + 20)$
- the quotient of 20 divided by 19 is $q = 1$ $(20 = 1 * 19 + 1)$.

In this problem you are asked to compute the quotient without using the modulo operator % or the division operator / *except for dividing by* 2. Using functions in the the header file *cmath* is also not allowed. Use loops.

Write a function

```
int quotient(int x, int y)
```

which, given x and y, returns q. If $x < 0$ or $y < 0$, your function is supposed to return -1 and exit without processing x and y.

(*Hint:* by the definition of the quotient, q is the only integer satisfying: $q * y \leq x$ and $(q+1) * y > x$).

Any correct solution is worth 10/25 points. To get full grade, your function should be very efficient. Think about huge values of x and y, e.g., $x = 320098$ and $y = 33$ (hence $q = 9699$) and try to achieve tens of steps instead of thousands of steps. (*Hint:* binary search idea)

Submit your code in a file called Prob3.cpp including your name and ID number.

Problem 4 (25 points). Substrings and concatenation

a) **(10 points) Substrings.** Write a function *substringAtPosition*, which takes three input parameters: a C-string s, an integer i, and another C-string t. The function *substringAtPosition* should check whether or not t is a substring of s starting at position i, i.e., whether or not $t[0, 1, \ldots, n-1]$ is equal to $s[i, i+1, \ldots, i+n-1]$, where n is the length of t.

Examples:

- *substringAtPosition*("*abcabc*", 2, "*cab*") returns YES
- *substringAtPosition*("*abcabc*", 1, "*cab*") returns NO
- *substringAtPosition*("*abcabc*", 4, "*bcd*") returns NO
- *substringAtPosition*("*abcabc*", -1, "*cab*") returns NO
- *substringAtPosition*("*abcabc*", 0, "*abca*") returns YES

Write a program to test your function.

Submit your code in a file called Prob4a.cpp including your name and ID number.

b) **(15 points) (difficult) Concatenated strings**

Write a function which given a C-string s, checks whether or not s can constructed by concatenating the strings "*abaa*", "*bbab*", and/or "*aaaabb*". What makes this problem difficult is that you are allowed use each of the three strings *more than once* (e.g., once, 100 times, 0 times) and *in arbitrary order*.

Examples:

- "*abaaabaa*" is decomposable: *abaa abaa*.

- "*abaaabaaaaaabbabaabbabbbababaa*" is decomposable: *abaa abaa aaaabb abaa bbab bbab abaa*.
- "*aaaabbbbababaabbabbbabbbababaa*" is decomposable: *aaaabb bbab abaa bbab bbab bbab abaa*
- "*abaaaba*" is not decomposable
- "*aaaabbbababababbabab*" is not decomposable

Any correct solution is worth 7/15 points (*Hint:* use the function in Part (a) and use recursion). To get full grade your solution should be efficient, i.e., it shouldn't take forever on strings of size above 100 (*Hint:* think in terms of smaller subproblems but don't use recursion).

Write a program to test your function. Submit your code in a file called Prob4b.cpp including your name and ID number.