University of Washington

Outreach Program

Name_____

1. For this design, write a C++ program that includes a function *prompt()* to request the user to enter an arbitrary (C style) text string. The prototype for the prompt() function must be as follows,

   void prompt(char* &anAddress);

   The function will use a temporary buffer to store the data from the user, prompt for the data, dynamically allocate a buffer large enough to hold the entered data, transfer the data to that buffer, and return the address of the dynamic buffer through the parameter anAddress.

   Why are we passing the pointer into the function by reference?

   Why are we dynamically allocating the buffer in the prompt function?

2. Using your *prompt()* function and the operators new and delete, create a set of C++ functions for constructing dynamic strings. Memory for a dynamic string is to be allocated as required. Include functions to concatenate, append, and replace one string with a larger or smaller one. Values are to be passed and returned by reference.

   Once again, for this problem, we are working with C style strings – that is, null terminated char arrays.

   The function concatenate () combines two strings while append () adds a character or substring to the end of a named string. Replace simply replaces one C style string with another. For example,

   concat: S3 ← S1 + S2

   S1: "Bon Jour "

   S2: "mon petit frere"

   S3: "Bon Jour mon petit frere"

   S1, S2 remain unchanged

   append: S1 ← S1 + " Ca va, mon ami"

   S1: "Bon Jour Ca va, mon ami"

   S1 is changed, S2 remains unchanged

   replace: S1 ← S2

   S1: "Bon Jour "

   S1 is changed to "mon petit frere", S2 remains unchanged

3. An array is a common structure for storing information. In many languages, one must specify the size of the array in advance of its use. Frequently, however, one does not know how much data will need to be stored. A common solution is to declare the array to be much larger than is expected will be necessary in hopes that it will be large enough. This is certainly not the most efficient solution.

   In this problem, you are to design an array that expands as data is entered, contracts when it is removed, and supports the following access methods.

   void put (int aValue) - puts aValue at array index 0.
   void atPut (int anIndex, int aValue) - puts aValue at array index anIndex – the array must expand when the index is beyond the size of the array. We have overwrite semantics when the index is in bounds and insert semantics otherwise.
   void atGet (int anIndex, int &aValue) - returns a copy the data at array index anIndex.
   void atRemove (int anIndex) - removes the data at array index anIndex – the array must contract.
   void showArray - displays the contents of the array.

   a. Write the use cases for the array. For each use case, be certain to give the textual description of the use case and identify any exceptions.
   b. Draw a class diagram for the array. Be sure to identify all public and private data and function members.