CSE 333 Midterm Exam 2/14/14 Sample Solution

Question 1. (36 points) C programming. Recall from CSE143 that a Binary Search Tree (BST) is a binary tree where each node contains a value, and for each node *n*, every value in the left subtree of *n* is less than the value stored in *n*, and every value in the right subtree of *n* is greater than the value stored in *n*.

For this problem we will use the following `typedef` to define the nodes of a BST of C strings:

```
typedef struct bst_node {
  char *str;               // heap-allocated string for this node
  struct bst_node *left;   // left subtree with strings < str
  struct bst_node *right;  // right subtree with strings > str
} BST_Node, *BST_NodePtr;
```

Each node in the BST is allocated on the heap. The string pointer `str` in the node points to an array of characters on the heap that is allocated when the node is created and holds a private copy of the string data associated with that node.

You may make the following simplifying assumptions while answering this question:

- You may assume that `malloc` always succeeds when it is called.
- All string values are properly terminated with a `'\0'` byte and are not `NULL`.
- You may use the unsafe but simpler string functions like `strcpy` and `strcmp`.
- Assume that all necessary header files like `string.h` have been `#include`d.

Reminders about a few possibly useful string functions. All string arguments have type `char*`.

- `strlen(s)` returns the number of characters (bytes) in `s`, not including the `'\0'` at the end.
- `strcpy(dst,src)` copies `src` to `dst`.
- `strcat(dst,str)` appends a copy of `src` to the end of `dst`.
- `strcmp(x,y)` returns 0 if strings `x` and `y` are the same, some negative integer if `x<y`, and some positive integer if `x>y`.

Complete the functions on the following pages. You may remove this page for reference if you wish.

Hint: recursion is your friend (really!).

(continued next page)

Question 1. (cont.) (a) (12 points) Complete the following function so it frees all dynamically allocated storage associated with the BST whose root node is `r`. Note that `r` may be `NULL` if the BST is empty.

Do not be alarmed if there is a lot more space here than you need for your answer.

```
void free_tree(BST_NodePtr r) {
  if (r == NULL)
    return;
  // non-empty tree; delete subtrees first, then
  // delete string and the root node itself
  free_tree(r->left);
  free_tree(r->right);
  free(r->str);
  free(r);
}
```

Question 1. (cont.) (b) (24 points) Complete the implementation of `insert` below so that `insert(str,r)` adds a copy of `str` to the BST with root `r` if `str` is not already present in the tree, and returns a pointer to the root of the modified tree. If `str` is already in the tree, `insert` should just return the pointer to the root of the original tree. For example, the following code adds "`pqr`" to the BST whose root node is `words` if "`pqr`" is not already present.

```
words = insert("pqr", words);
```

Note that the tree (`words` in this example) may be empty (`NULL`) initially. When new words are added, the code should preserve the binary search tree property that left subtrees have values less than their parent node, and right subtree values are greater. For full credit your answer should only traverse the necessary nodes in the tree.

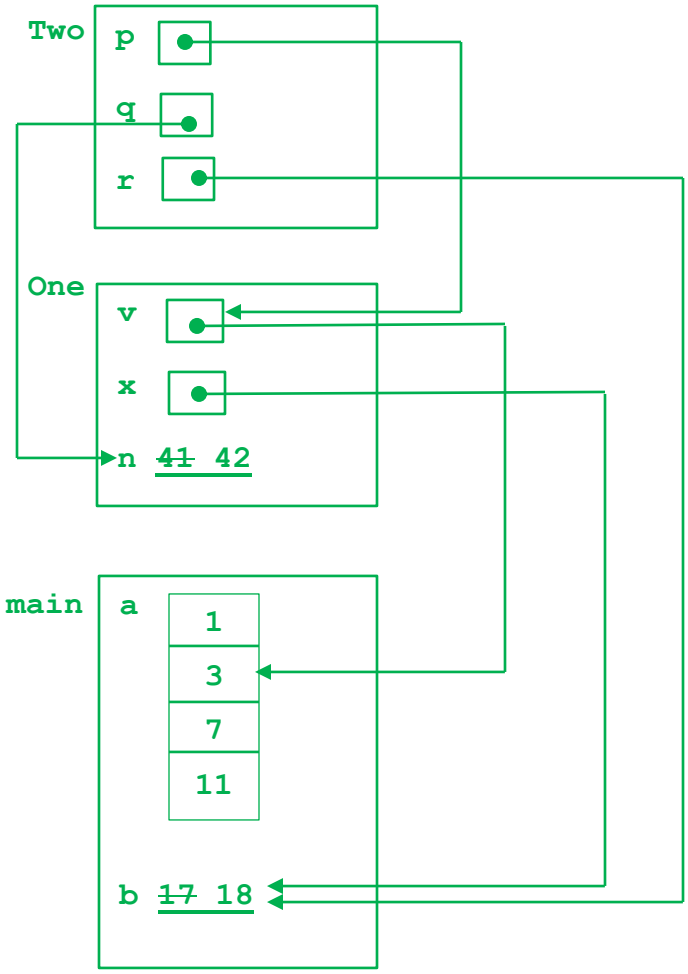
```
BST_NodePtr insert(char *s, BST_NodePtr r) {

  // If tree is empty, return a leaf node as the new tree
  if (r == NULL) {
    // allocate new node
    BST_NodePtr n = (BST_NodePtr)malloc(sizeof(BST_Node));
    n->str = (char*)malloc(strlen(s)+1);
    strcpy(n->str, s);
    n->left = NULL;
    n->right = NULL;
    return n;
  }

  // Tree is not empty.  If s is the same as the string in
  // the root node, return the root node without altering
  // the tree.  Otherwise insert the string in the left or
  // right subtree as appropriate and return the root of
  // the resulting tree.
  int cmp = strcmp(s, r->str);
  if (cmp < 0)
    r->left = insert(s, r->left);
  else if (cmp > 0)
    r->right = insert(s, r->right);
  return r;
}
```

Question 2. (34 points) The traditional twisted C program. This code does compile and execute without warnings or errors.

```
#include <stdio.h>

void Two(int **p, int *q, int *r) {
  (*p)++;
  (*q)++;
  (*r)++;
  // HERE (see instructions for part (a))
  (**p)++;
  printf("Two: **p = %d, *q = %d, *r = %d\n", **p, *q, *r);
}

void One(int *v, int *x, int n) {
  Two(&v, &n, x);
  (*v)++;
  (*x)++;
  printf("One: *v = %d, *x = %d, n = %d\n", *v, *x, n);
}

int main() {
  int a[] = { 1, 3, 7, 11 };
  int b = 17;
  One(a, &b, 41);
  for (int k = 0; k < 4; k++) {
    printf("%d ", a[k]);
  }
  printf("%d\n", b);
  return 0;
}
```

Answer questions about this program on the next page. You may remove this page for reference if you wish.

Question 2. (cont.) (a) (20 points) Draw a boxes 'n arrows diagram showing state of memory when control reaches the comment containing `HERE` in the middle of function `Two`. Your diagram should have three boxes showing the stack frames for functions `main`, `One`, and `Two`. The stack frames should include values of integer variables and an arrow from each pointer to the location that it references. Then answer part (b) at the bottom of the page.



(b) (14 points) What output does this program produce when it is executed?

```
Two: **p = 4, *q = 42, *r = 18
One: *v = 5, *x = 19, n = 42
1 5 7 11 19
```

Question 3. (28 points) Dept. of Computational Entomology. The following code declares a C++ class that acts like a C++ `vector` containing `string`s. New strings can be added to the end of a `StringList` using the `push_back` function, and the `StringList` is supposed to expand as needed to hold new values. (This class omits many useful operations so we can keep it fairly small for this exam. However, the code on this page is correct and compiles successfully.)

```
#include <string>
using namespace std;

class StringList {
 public:
  // Default constructor: Initialize a new, empty StringList
  // with default capacity
  StringList();

  // copy constructor
  StringList(const StringList &other);

  // destructor
  ~StringList();

  // add new string s to the end of this StringList, expanding
  // the list as needed
  void push_back(const string &s);

 private:
  // StringList representation:
  // strings_ : an array allocated on the heap holding strings.
  // capacity_ : the number of elements in the array strings_.
  // size_ : the number of items currently in the list, where
  //         0 <= size_ <= capacity_.  The items are stored in
  //         strings_[0..size_-1].
  string *strings_;
  int capacity_;
  int size_;

  // initial capacity of a new empty StringList
  static const int default_capacity_ = 4;
};
```

(Continued on the next page. You may remove this page for reference.)

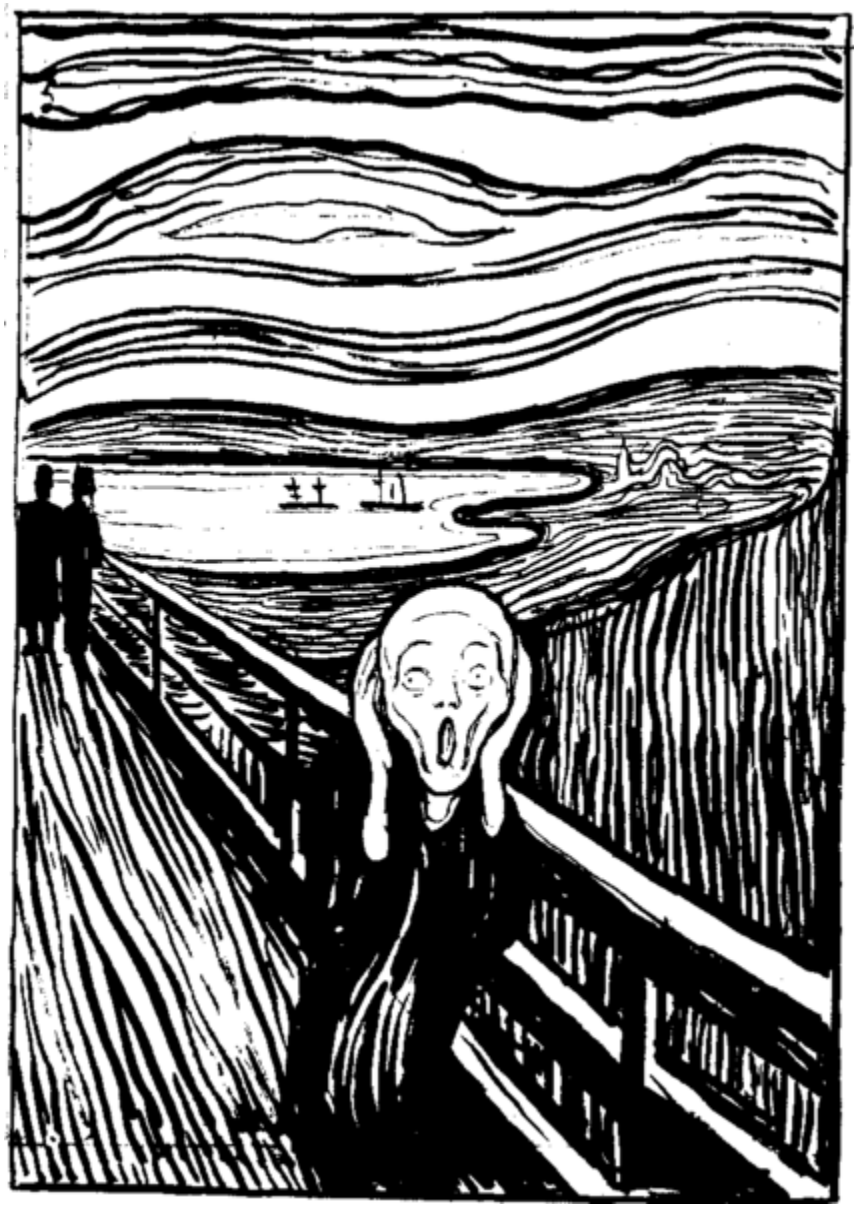
Question 3. (cont.) A separate file contains the following implementations of the functions declared on the previous page. Unfortunately there are bugs – lots of bugs. Mark the code below to identify the problems, and write in corrections so the code will work as intended. Keep your notes brief and to the point. You can write additional comments or changes on the next page, but *please* help the graders by making it easy to follow your changes and figure out where they fit. You may cross out some of the code on this page and rewrite it on the next page if that is easier.

Corrections and additions are shown in bold green.

```
// Default constructor
StringList::StringList(): capacity_(default_capacity_),size_(0) {
  strings_ = new string[capacity_];
}

// copy constructor
StringList::StringList(const StringList &other)
  : capacity_(other.capacity_), size_(other.size_)
    , strings_(other.strings_) {
  strings_ = new string[capacity_];
  for (int k = 0; k < size_; k++)
    strings_[k] = other.strings_[k];
}

// destructor
StringList::~StringList() {
  delete [] strings_;            // wrong form of delete
}

// add string s to the end of this StringList
void StringList::push_back(const string &s) {
  // expand list by default_capacity_ if already full (poor
  // (heuristic in general, but any amount is ok for the test)
  if (size_ == capacity_ ) {
    capacity_ += default_capacity_;
    string *new_strings = new string[capacity_];
    for (int k = 0; k < size_; k++)
      new_strings[k] = strings_[k];
    delete [] strings_;
    strings_ = new_strings;
  }
  // add new string (and we are guaranteed there is room now)
  strings_[size_] = s;
  size_++;
}
```

Question 4. (2 points) Draw something interesting below.

By definition, all drawings made in answer to this question are interesting and will receive the free points.



Grader after reading midterm exams all day