Exam 2

April 19th, 2012

Work alone. Do not use any notes or books. You have approximately 75 minutes to complete this exam.

Please write your answers on the exam. More paper is available if you need it. Please put your name at the top of the first page.

There are four questions on this exam, for a total of 80 points.

1 Pointer juggling (20 points)

On which line will the following program fail with a segmentation fault, and why?

```
1  #define N (3)
2  
3  int
4  main(int argc, char **argv)
5  {
6      int i;
7      struct pointy {
8          struct pointy *p[2];
9      } a[N];
10 
11     for(i = 0; i < N; i++) {
12         a[i].p[0] = &a[i];
13         a[i].p[1] = a + (i + 1) % N;
14     }
15 
16     a[0].p[0] = 0;
17 
18     a[2].p[0]->p[1]->p[1] = a[0].p[0];
19     a[1].p[0]->p[0]->p[1] = a[0].p[0];
20     a[0].p[1]->p[1]->p[0] = a[2].p[0];
21     a[0].p[0]->p[1]->p[1] = a[2].p[1];
22     a[0].p[0]->p[1]->p[1] = a[2].p[0];
23 
24     return 0;
25 }
```

2 Choosing a data structure (20 points)

Suppose you are asked to write a program for displaying a leaderboard for a large on-line multiplayer game. The leaderboard consists of the names of the 10 players with the highest scores and their scores. Each player of the game has a name (a string) and a score, equal to the number of games they have won. Your program should provide routines `incrementScore(const char *player)` and `returnWinners(char *winners[10], int scores[10])` where `incrementScore` increases the score of the named player by 1 (or sets the score to 1 if the player has never won a game before) and `returnWinners` fills in the `winners` and `scores` arrays with the names and scores of the players with the 10 highest scores (with ties broken arbitrarily).

Without writing actual code, describe as succinctly as you can what data structure would be a good choice for this task and why.

3 A digital search tree (20 points)

Here is an implementation of the creation and insertion routines for a binary digital search tree. The idea of the tree is that a key of type `unsigned int` is represented by a path through the tree, where each edge in the path is determined by the next bit in the key.

Write a routine `DSTpredecessor` that returns the largest key in the tree less than or equal to the given key, or `0` if there is no such key. We have provided a declaration for this function for you on the next page.

```
#include <stdlib.h>
#include <limits.h>

#define BITS (sizeof(unsigned int) * CHAR_BIT)  /* CHAR_BIT == 8 */

struct node {
    struct node *kids[2];
};

struct node *DSTcreate(void)
{
    struct node *root;

    root = malloc(sizeof(struct node));
    root->kids[0] = root->kids[1] = 0;
    return root;
}

void DSTinsert(struct node *root, unsigned int key)
{
    unsigned int mask;
    int bit;
    for(mask = 1 << (BITS-1); mask != 0; mask >>= 1) {
        bit = (key & mask) != 0;
        if(root->kids[bit] == 0) {
            root->kids[bit] = malloc(sizeof(struct node));
        }
        root = root->kids[bit];
    }
}
```

```
/* return largest key <= key or 0 if there is none */
unsigned int DSTpredecessor(struct node *root, unsigned int key);
```

4 Change this program (20 points)

Here is a rather dangerous implementation of a queue. Turn it into a stack by changing exactly one line of code.

```
#include <stdlib.h>

struct thing {
    int size;         /* size of contents */
    int bot;          /* first used location */
    int top;          /* first unused location */
    int *contents;    /* array of elements */
};

struct thing *thingCreate(int size)
{
    struct thing *t;

    t = malloc(sizeof(*t));

    t->size = size;
    t->bot = t->top = 0;
    t->contents = malloc(sizeof(int) * size);

    return t;
}

void thingPush(struct thing *t, int value)
{
    t->contents[t->top++] = value;
}

int thingPop(struct thing *t)
{
    return t->contents[t->bot++];
}
```