Name: ___________________________________

CSE 303, Autumn 2006, Midterm Examination

Solutions

1 November 2006

Please do not turn the page until everyone is ready.

Rules:

- The exam is closed-book, closed-note, except for one side of one 8.5x11in piece of paper.
- Please stop promptly at 3:20.
- You can rip apart the pages, but please write your name on each page.
- There are **58 points** total, distributed **unevenly** among 5 questions (all of which have multiple parts).
- When writing code, style matters, but don't worry about indentation.

Question	Max	Grade
1	8	
2	6	
3	12	
4	24	
5	8	
Total	58	

Advice:

- Read questions carefully. Understand a question before you start writing.
- **Write down thoughts and intermediate steps so you can get partial credit.**
- The questions are not necessarily in order of difficulty. **Skip around.**
- If you have questions, ask.
- Relax. You are here to learn.

Name: ___________________________________

1. (8 Points)

   What does each of the following bash commands do? You must be specific about information that appears on the terminal, changes to the shell environment, and the names of any files that are created. You do not need to be specific about the contents of files that are created.

   (a) `mkdir data{3,4}`

   Creates two directories: data3 and data4

   (b) `tmp=`pwd`; cd ..; rm –r “$tmp”; unset tmp`

   Removes the current directory and all its contents and moves to the parent directory. Also erases the variable “tmp” if it was set.

   (c) `foo=whoami; alias foo=jobs; $foo`

   Prints the current user’s name

   (d) `rmdir . 2>>log || echo Hello`

   Appends an error message to the file “log” and prints “Hello” on the terminal.

Name: ___________________________________

2. (6 Points)

   For each of the following, give a regular expression suitable for grep (or egrep) that matches the lines described. You may assume that the regular expression is wrapped in single quotes (as in `'exp'`) when passed to grep or egrep on the command line.

   (a) Lines that contain `March` or `May`.

   ```
   March|May
   ```

   (b) Lines that start with one or more `a` characters, followed by zero or more spaces, followed by an optional `?` character.

   ```
   Many answers, including the following:
      ^a+ *\??
      ^a+[ ]*[?]?
      ^a\+ *?\?
   ```

   (c) Lines that contain a year between `2000` and `2009`, excluding `2006`.

   ```
   200[0-57-9]
   ```

Name: ___________________________________

3. (12 Points)

   (a) Write a bash command that uses `sed` to make a new file called `book2.txt`, which is a modified version of the file `book.txt`. In `book2.txt`, lines from `book.txt` that start with “`Section n`” (where “`n`” is any integer) should start with “`Chapter n -`”.

```
sed 's/^Section \([0-9]\+\)/Chapter \1 -/' book.txt >book2.txt
```

   (b) If your task were to modify lines that start with “`Section n`” to contain “`Section n+1`” instead, could you use `sed` to do it? If so describe how. If not, explain why not. (Note: `n+1` is the numeric result of adding one to `n`, not the string “`+1`” appended to the original number.)

No. sed cannot be used to do arithmetic on strings. You’d need a more powerful scripting language to do this, like awk, perl, python, or ruby.

Name: ___________________________________

4. (24 Points)

The following lines are part of the definition of a doubly-linked list data type. Users of this data type keep pointers to `List` structures. `List` structures contain a count of the number of elements in the list, as well as pointers to the first and last elements in the list (these pointers are `NULL` if the list is empty). `ListElt` structures contain the list elements themselves, as well as pointers to the elements before and after them in the list.

```
typedef struct _ListElt {
   void *data;
   struct _ListElt *next;
   struct _ListElt *prev;
} ListElt;

typedef struct _List {
   ListElt *head;
   ListElt *tail;
   int count;
} List;


// Return a new ListElt containing a copy of data.
ListElt *LNewElement(void *data, int size) {
   ListElt *e = (ListElt*)malloc(sizeof(ListElt));
   e->next = NULL;
   e->prev = NULL;
   e->data = malloc(size);
   memcpy(e->data, data, size);
   return e;
}

// Free "e" and the data inside
void LFreeElement(ListElt *e) {
   free(e->data);
   free(e);
}
```

The following problems make use of this data structure.

Name: ___________________________________

4(a) The following functions use this data type. Below each function, indicate if it is correct or if there is a bug. If there is a bug, explain what the problem is.

```
// f1: Should print the third element of array i
void f1() {
   int i[] = {1, 2, 3};
   ListElt *e1 = LNewElement(i, sizeof(int));
   int *j = (int*)e1->data;
   printf("%d\n", *(j+2));
   LFreeElement(e1);
}
```

Bug: LNewElement copies only the first element of array i. The reference *(j+2) is outside the bounds of what LNewElement has allocated. The call should be: LNewElement(i, sizof(int) * 3)

```
// f2: Should print "6"
void f2() {
   int i=5, *p = &i;
   if (1) {
      int j=6;
      p = &j;
   }
   ListElt *e1 = LNewElement(p, sizeof(int));
   printf("%d\n", *((int*)e1->data));
}
```

Bug: p is set to the address of a variable that goes out of scope when the if block ends. There is no guarantee that this memory will contain what we expect when LNewElement is called. We could fix this by changing "p = &j;" to "i = j;"

Bug: Leaks the memory allocated by LNewElement.

```
// f3: Returns a new ListElt containing the string "12345"
ListElt *f3() {
   char *str = "12345";
   return LNewElement(str, 5);
}
```

Bug: LNewElement needs to copy 6 bytes of data instead of 5 in order to get the string's NULL terminator. Without this, the data inside the ListElt cannot be treated as a valid string.

Name: ___________________________________

4(b) Write the following function which operates on doubly-linked lists.

```
void LInsertBefore(List *list, ListElt *e, ListElt *eNew);
```

Use the following definition and constraints:

- This fuction inserts the element `eNew` just before element `e` in list `list`.
- You may assume that `list` and `eNew` are not `NULL`.
- If `e` is `NULL`, then insert `eNew` at the end of `list`.
- If `e` is not `NULL`, then you may assume that it has already been inserted into `list` using this function.
- You may also assume `eNew` has never been inserted into any list.
- You may assume that all non-`NULL` arguments have been allocated and initialized.
- Your implementation should not search through the elements of `list`. In other words, your implementation should run in constant time.

A word of advice: Try drawing a schematic of the various parts to help you reason about the various cases you need to handle.

```
void LInsertBefore(List *list, ListElt *e, ListElt *eNew) {
   if (list->count == 0) {
      list->head = eNew;
      list->tail = eNew;
   } else if (e == NULL) {
      list->tail->next = eNew;
      eNew->prev = list->tail;
      list->tail = eNew;
   } else {
      if (e->prev == NULL) {
         list->head = eNew;
      } else {
         e->prev->next = eNew;
         eNew->prev = e->prev;
      }
      eNew->next = e;
      e->prev = eNew;
   }
   list->count += 1;
}
```

Name: ___________________________________

5. (8 Points)

Indicate the output of the program using the provided blanks.

```
#include <stdio.h>

#define foo(x) 1+x*3

int main()
{
   int a = 1;
   int b = 2;

   printf("%d\n", foo(a));       // Output is: ____________4


   printf("%d\n", foo(a+b));     // Output is: ____________8


   printf("%d\n", 2*foo(b));     // Output is: ____________8


   printf("%d\n", -foo((b)));    // Output is: ____________5


   return 0;
}
```