CS 1723, Data Structures
Fall Semester, 1998
First Examination

1. (a) Give C declarations that will declare an array of 26 `structs`, where each `struct` has a `char` field and an `int` field.

   (b) Using the declarations from part (a), write a loop that will store the characters `'a'` through `'z'` in successive `char` fields of the array of `structs`, and will store the integers 1 through 26 in successive `int` fields of the array of `structs`. (You *must* use a loop to do this problem.)

2. This problem is concerned with RPN. (RPN stands for "Reverse Polish Notation.")

   (a) Give the RPN that corresponds to the following ordinary arithmetic expression:

   ```
   a + b * c - d #
   ```

   (b) Consider the following input to a program (like the one written for Assignment 2) that processes items of an RPN sequence, from left to right.

   ```
   3 6 4 + 2 * + #
   ```

   Assume that individual digits are treated as separate operands and that the other characters (except for the final #) are operators with two operands. Show step-by-step how a stack is used to evaluate this RPN sequence, and give the final value. (You should not write any C code for this problem.)

3. Suppose we use an array to implement a circular queue of characters as follows:

```
#define Q_SIZE 4   /* maximum queue size */
char delete(void); /* delete (remove) from front */
void insert(char); /* insert at read */
int empty(void);   /* check if empty */
int full(void);    /* check if full */

char q[Q_SIZE];  /* actual queue */
int fp = 0;      /* front pointer */
int rp = 0;      /* rear pointer */
int qs = 0;      /* size of queue */
```

   (a) Give C code for the function `insert` that inserts a `char` (the parameter of `insert`) at the rear of the queue. You may assume a `full()` function exists. Don't forget to make this a *circular* queue.

   (b) Give C code for the function `empty()`.

4. Consider the code

```
#include <stdio.h>
#include <stdlib.h>
void strcpy1(char *, char *);
void main(void)
{
   char s[] = "UTSA";
   char *t;
   t = ???;
   strcpy1(t, s);
   ...
}
```

The function `strcpy1` is supposed to behave like the real `strcpy` except that the latter returns a string (the result of the copy operation).

(a) Draw a diagram of `s` showing exactly what characters are stored in `s` and at which locations of `s`.

(b) Fill in the `???` above so as to allocate exactly enough storage to hold the string.

(c) Write code for the `strcpy1` function so that you do not use any square brackets (`[` or `]`), and do not use any C string functions (such as `strcpy` itself). You may use `strlen` if you wish.

(d) For each of the following expressions, say whether they represent a string or a character, and what string or character is represented (assuming that `s` is copied to `t`):

i. `t[1]`
ii. `*t`
iii. `&t[0]`
iv. `t+2`
v. `*(t+2)`