University of Toronto
Faculty of Applied Science and Engineering

Final Examination, December 1997

CSC 181F — Introduction to Computer Programming

Examiner — H. Hayashi

Duration of the exam: 2.5 hours

- Aids Allowed:
  - A non-programmable electronic calculator.
  - A single 8.5 by 11 inch aid sheet. The student may enter on both sides of the aid sheet any information desired, without restriction, except that nothing may be affixed or appended to the aid sheet.
- The exam consists of 7 pages, including this one. Make sure you have all 7.
- The exam consists of 7 questions. **Answer all 7**. The mark for each question is listed at the start of the question. The total mark is 100.
- **Write all your answers in the spaces provided after each question.** There are some extra spaces at the end, if you need scrap paper.
- **If you need to make any assumption in order to answer a question be sure to state those assumptions clearly.**

Write legibly. Unreadable answers are worthless.

Last (family) name:

First (given) name:

Student id:

Question	1	2	3	4	5	6	7	Total
Marks								
Out of	10	20	15	10	15	5	25	100

1. [10 marks.]

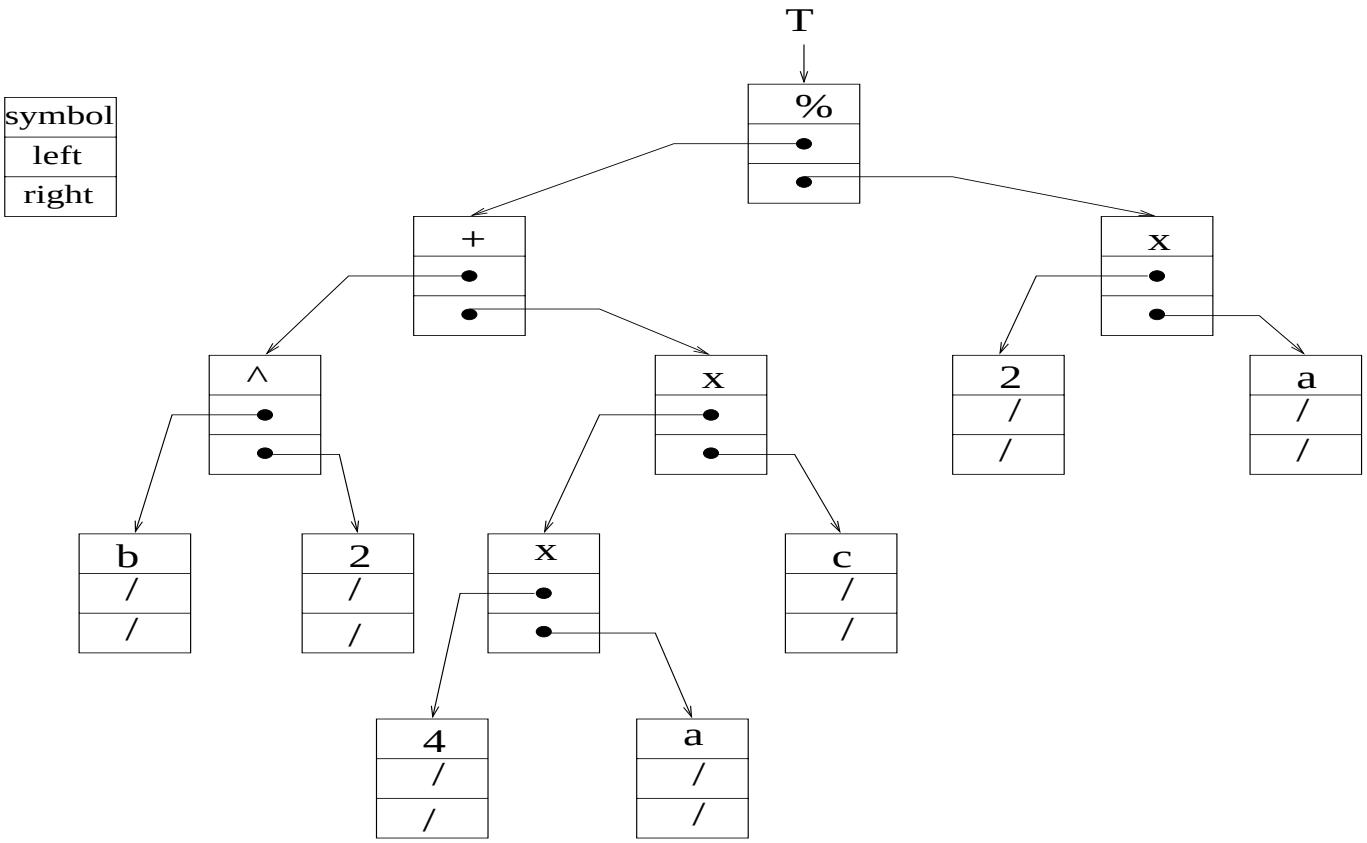

Figure 1: Binary Tree.

Assume the following data structures have been used to define binary trees.

```
typedef struct treeNode *treePtr;
struct treeNode {
    char symbol;
    struct treeNode *left;
    struct treeNode *right;
};
```

The following function traverses binary trees and print out the symbols in the nodes depending on the value of traversalOrder.

```
void traverse( treePtr T, int traversalOrder )
{
    if ( T != NULL ) {
        if ( traversalOrder == 1 ) {
            putchar( T->symbol );
            traverse( T->left, 1 );
            traverse( T->right, 1 );
        } else if ( traversalOrder == 2 ) {
            traverse( T->left, 2 );
            putchar( T->symbol );
            traverse( T->right, 2 );
        } else if ( traversalOrder == 3 ) {
            traverse( T->left, 3 );
            traverse( T->right, 3 );
            putchar( T->symbol );
        }
    }
}
```

What is the output from the function traverse when it is applied to the tree shown in Figure 1 in the following three cases?

(a) The value of traversalOrder is 1.

(b) The value of traversalOrder is 2.

(c) The value of traversalOrder is 3.

2. [20 marks.]

Write a function with header:

char *partition(char *S);

The parameter S is a string containing lower case letter and upper case letter. The value produced by this function is a string containing in order: all of the lower case letter occurring in S, a single blank, all of the upper case letters occurring in S. If an invalid parameter is encountered, the function returns a NULL pointer. The characters in each part of the partition can be given in any order. Examples:

```
partition("Ab")              ->     "b A"
partition("CSCF")            ->     " FCCS"
partition("XyZ")             ->     "y XZ"
partition("abc")             ->     "abc "
```

3. [15 marks.]

Assume the following declarations are used to create singly linked list.

```
struct listNode {
    int val;
    struct listNode *next;
};
typedef struct listNode *listPtr;
```

Write a function with the header

void reverseList(listPtr L);

This function returns a pointer to the list that is the same as the original list L, except that the nodes are in reverse order. Assume that a dummy header node is used.

4. [10 marks.]

Let n be an integer variable. It may have a positive or zero or negative value. As a function of n, how many times is the body of the loop

```
while ( n != 1 )
    n /= 2;
```

executed?

5. [15 marks.]

A string that reads the same either backward or forward is called **palindrome**. Examples are *anna* and *otto*. Write a function that tests whether a given string is a palindrome or not using the header:

int palindrome(char *str);

The return value is 1 if the string str is a palindrome. Otherwise it is 0.

6. [5 marks.]

Archimedes estimated π as follows. He took a circle with diameter 1, and hence circumference π. Inside the circle he inscribed a square, as shown in Figure 2. The length of the perimeter of the square is smaller than the circumference of the circle, and so it is a lower bound on π. Archimedes then considered an inscribed octagon, 16-gon, 32-gon, etc., each time doubling the number of sides of the inscribed polygon. Using this approach, he was able to show $223/71 < \pi < 22/7$.

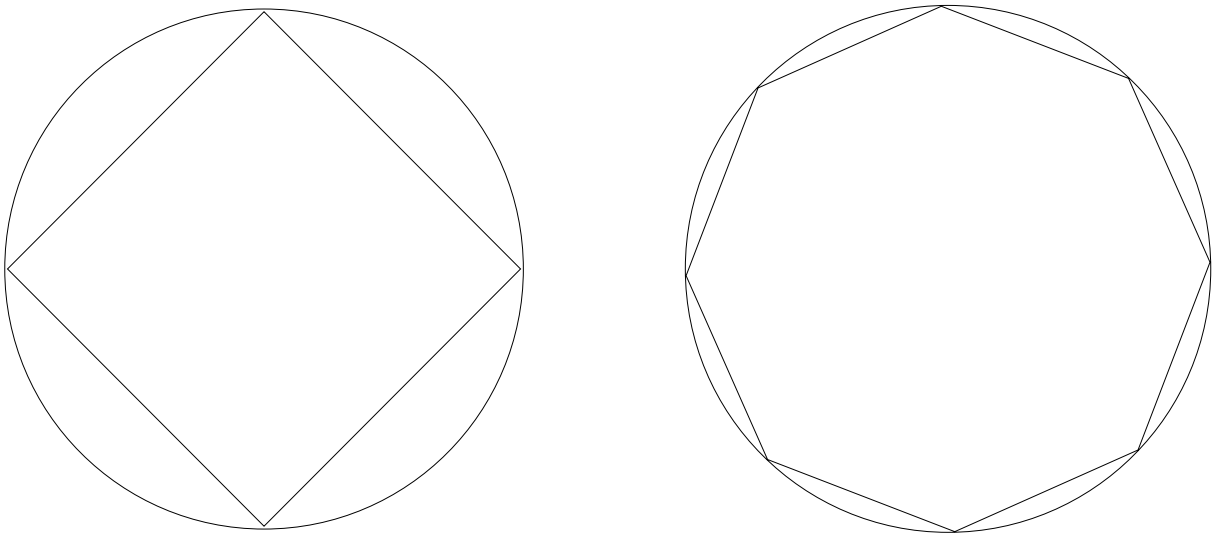

Figure 2: Inscribed square and octagon.

Let p_n be the length of the perimeter of the polygon with 2^n edges inscribed in the circle of diameter 1. For example, $p_2 = 2\sqrt{2}$ is the length of perimeter of the inscribed square. Using trigonometry, one can show that

$$p_{n+1} = 2^n \sqrt{2\left(1 - \sqrt{1 - \left(\frac{p_n}{2^n}\right)^2}\right)} \qquad \text{for } n \geq 2.$$

One can also show that $p_n = q_n$, where $q_n = 2^{n-1}\sqrt{r_n}$, $r_2 = 2$ and

$$r_{n+1} = \frac{r_n}{2 + \sqrt{4 - r_n}} \qquad \text{for } n \geq 2.$$

You do not need to prove the validity of any of these formulae — just accept them.

If the formulae above are evaluated in exact arithmetic, then $p_n = q_n$ for all $n \geq 2$. However, if the formulae are evaluated in double precision floating-point arithmetic in a C program, then you get that $q_{40} = 3.141592653589793 \approx \pi$, but $p_{40} = 0$.

Explain why the formula for p_n produces such a poor approximation to π in floating-point arithmetic, while the mathematically equivalent formula for q_n produces such a good one.

7. [25 marks.]

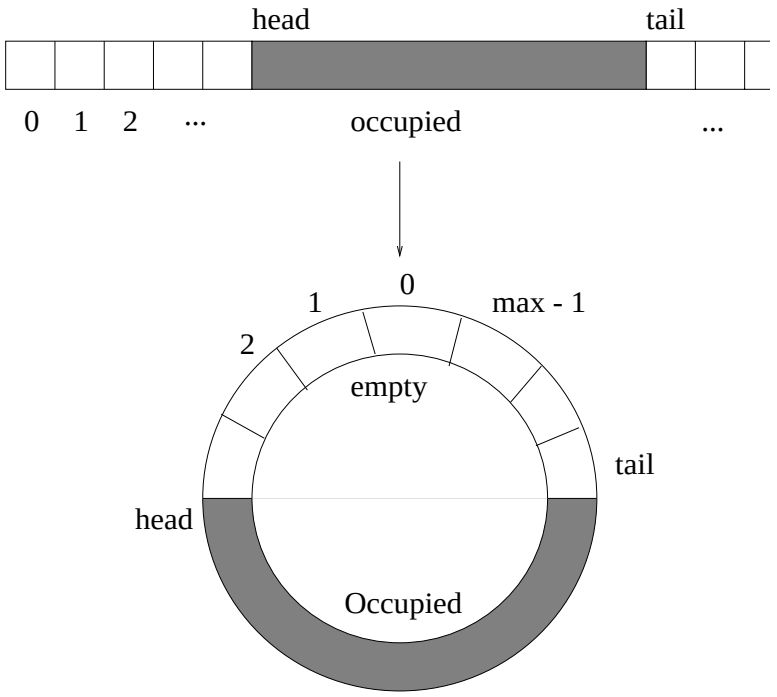

Figure 3: Circular Array

Queue can be implemented in many different ways. One way is to use an array and to keep track of the head and the tail of the queue. One problem is that as the queue moves down the array, the storage space at the beginning of the array is discarded and never used again. To overcome this inefficient use of space, we can think of the array as a circle rather than a straight line. See Figure 3. In this way, as items are added and removed from the queue, the head will continually chase the tail around the array. At different times, the queue will occupy different parts of the array, but we never worry about running out of space unless the array is fully occupied.

The following declarations are used to implement Queue using a "circular array", just described.

```
#define MAXQSIZE 100
typedef int itemType;
typedef struct {
    int count, head, tail;
    itemType items[MAXQSIZE];
} Queue;
```

where `count` keeps track of the number of items in the queue, `head` is an array index which keeps track of the index of the item that can next be removed from the queue, and `tail` is an array index that designates the place to insert the next queue item.

Write the following two functions using a circular array, *i.e.*, you must reuse empty space when necessary.

void enqueue(Queue *Q, itemType R);

This function inserts the new item R at the tail of the Queue Q.

void dequeue(Queue *Q, itemType *H);

This function removes an item from the head of the Queue Q and returns the item as the value of H.