Computer Science Foundation Exam

Aug. 6, 2004

COMPUTER SCIENCE I

Section I A

No Calculators!

KEY

Name: ____________________

SSN: ____________________

Score: / **50**

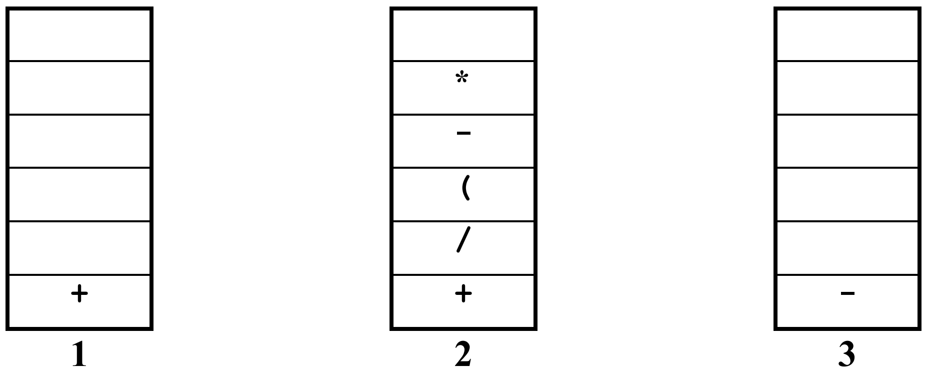
In this section of the exam, there are Three (3) problems

<u>You must do all of them.</u>

The weight of each problem in this section is indicated with the problem. Partial credit cannot be given unless all work is shown. As always, be complete, yet concise, and above all <u>be neat.</u>

1. **[12 pts]** Convert the following infix expression into postfix expression using a stack. Trace the state of the stack as each character of the infix expression is processed. Show the contents of the stack at the indicated points in the infix expressions (points 1, 2 and 3), and also the final postfix expression.

```
                     1                           2          3
     D  *  F  + H       / ( K – M / R *    B ) –    E
```



Final postfix expression **D F * H K M R / B * – / + E –**

GRADING: Allocate 3 points for each stack and 3 for the final answer

2. **[12 pts]** Indicate the time complexity for each of the following operations in terms of Big-O notation, assuming that efficient implementations are used. Give the worst case complexities in terms of n, assuming that each data structure holds n elements.

a) Searching for an element in a sorted array **O(log n)**

b) A push operation in an array implemented stack **O(1)**

c) Finding the sum of elements of an array **O(n)**

d) A Dequeue operation in a circular array implementation of queue **O(1)**

e) Inserting an element in a linked list containing sorted values **O(n)**

f) Searching for an element in a balanced binary search tree **O(log n)**

g) Deleting 10th node of a linked list **O(1)**

h) A Push operation in a linked list implementation of stack **O(1)**

Grading: Two points each (all or nothing).

3. **[12 pts]** A list containing the values 20, 48, 57, 60, 66, 69, 75, 77, and 82 (in that order) is stored in a linked list **front (struct node *)** having the node structure

```
struct node {
  int data;
  struct node* next;
};
```

right BEFORE the following code is executed. Find the new contents of the list right AFTER this code terminates.

$$\begin{tikzcd} 20 \arrow[r] & 48 \arrow[r] & 57 \arrow[r] & 60 \arrow[r] & 66 \arrow[r] & 69 \arrow[r] & 75 \arrow[r] & 77 \arrow[r] & 82 \end{tikzcd}$$

```
struct node *current;
struct node *temp;
current  =  front;
while (current->next !=NULL) {

      while (   (current->next->data    –    current->data)      >     10) {
            temp  =  (struct node*) (malloc(sizeof(struct node)));
            temp->data  =  current->data  +  10;
            temp->next = current->next;
            current->next =  temp;
            current  =  current->next;
      }
      if (      (current->next->data    –    current->data)        <      5){
            temp = current-> next;
            current->next  =  current->next ->next;
            free( temp );
      }
      current  =  current->next;

 }
```

20 30 40 48 57 66 75 82

Grading:
If only one node added between 20 and 48, then deduct 5 points.
If the output includes 60, deduct 3 points.
If new list contains 69 instead of 66 , deduct 3 points.

4. [14 pts] A queue is implemented using a linked list. The access to the queue is only through a single node called *qfront*. The nodes of the linked list have the same structure as shown in Question 3**.** Develop an enqueue function to add an item with value *id*. You may choose to develop the function

[Soln1]

`struct node* enqueue(struct node *qfront, int id)` returns the new value of qfront.

or the function

[Soln2]

`void enqueue( struct node **qfront, int id)`

```
/* Soln1 */struct node* enqueue( struct node *qfront, int id)
/* Soln2 */void enqueue( struct node **qfront, int id)
{
  struct node  *pCur, *pNew;
  pNew = (struct node*)  malloc(sizeof(struct node));
  pNew-> data = id;
  pNew->next  = NULL;
  /*Soln1*/
  if ( qfront == NULL)

       qfront = pNew;

  else{
        pCur = qfront;
        while( pCur->next != NULL) pCur = pCur->next;
        pCur->next=pNew;
  }/*else*/
  return qfront;
 }/*Soln1*/

  /*Soln2*/
  if ( *qfront == NULL)

        *qfront = pNew;

  else{
        pCur = *qfront;
        while( pCur->next != NULL) pCur = pCur->next;
        pCur->next=pNew;
  }/*else*/

 }/*Soln2*/
```

GRADING: if malloc not used (-3)
If the case of empty queue not considered (-2)
[Soln1] does not return qfront (-3)
[Soln2] Ignores double pointer (uses qfront instead of *qfront in the code) (-3)
adds the items at front of the list (-3)