COP 3502 – Exam #2 – Spring 2004

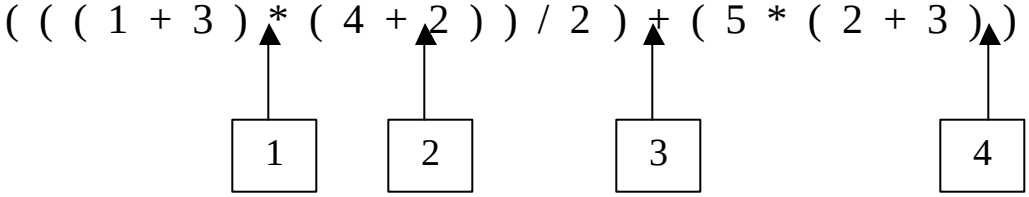
March 18, 2004 (100 points)

NAME: KEY

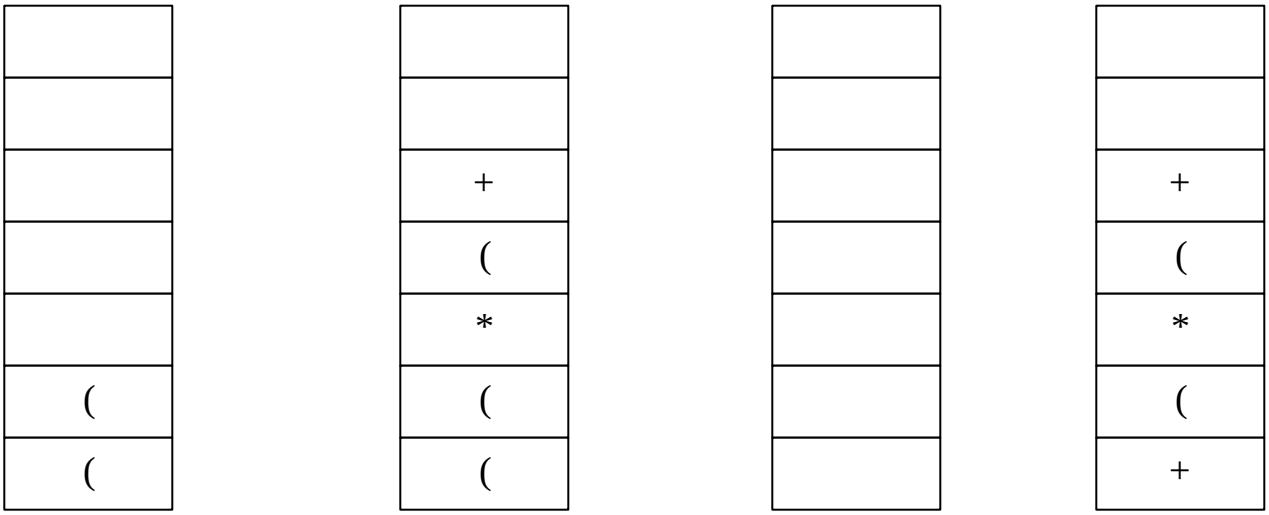
please print clearly

1. (15 points-total)

Convert the following infix expression to its corresponding postfix form. Show the values in the stack at the points indicated in the expression (points marked 1, 2, 3, and 4).



Postfix expression: **1 3 + 4 2 + * 2 / 5 2 3 + * +**



Stack at point #1 **Stack at point #2** **Stack at point #3 (EMPTY)** **Stack at point #4**

2. (10 points)

Given the following array of integers, show the contents of the array after each of the first six passes of the bubble sort on this array. Assume that the bubble sort is to sort the values into ascending order.

Array X

index	0	1	2	3	4	5	6	7	8	9	10	11
value	**41**	**36**	**22**	**13**	**17**	**32**	**25**	**34**	**28**	**3**	**27**	**14**

index	0	1	2	3	4	5	6	7	8	9	10	11
initial	**41**	**36**	**22**	**13**	**17**	**32**	**25**	**34**	**28**	**3**	**27**	**14**
pass 1	**36**	**22**	**13**	**17**	**32**	**25**	**34**	**28**	**3**	**27**	**14**	**41**
pass 2	**22**	**13**	**17**	**32**	**25**	**34**	**28**	**3**	**27**	**14**	**36**	**41**
pass 3	**13**	**17**	**22**	**25**	**32**	**28**	**3**	**27**	**14**	**34**	**36**	**41**
pass 4	**13**	**17**	**22**	**25**	**28**	**3**	**27**	**14**	**32**	**34**	**36**	**41**
pass 5	**13**	**17**	**22**	**25**	**3**	**27**	**14**	**28**	**32**	**34**	**36**	**41**
pass 6	**13**	**17**	**22**	**3**	**25**	**14**	**27**	**28**	**32**	**34**	**36**	**41**

3. (10 points)

Write a recursive function (in C) which will correctly print the first **n** odd integer numbers. You must define this function using only a single numeric parameter to be passed to the procedure. Assume that the initial call has the form: **print_odd(n)** where **n** indicates how many odd numbers are to be printed. For example, the call **print_odd(6)** would produce the following output: 1 3 5 7 9 11. Assume that n > 0.

```
void function print_odd (int n)
{  //print the first n odd integers recursively.
   // Assume n > 0.
   if (n ==1)
      printf("1  \n");
   else  {
      print_odd(n-1);
      printf("%d  ", (2*n)-1);
  }
}//end function print_odd
```

4. (10 points)

Consider the following recursive function.

```
int P4 (int x, int y)
{
      if (x < 0  || y < 0)
         return x–y;
       else
         return (P4(x-1, y) + P4(x, y-1));
}
```

What value is produced by the following function call? SHOW YOUR WORK BELOW!

P4(1,1) **0**

Solution :

P4(1,1) = P4(0,1) + P4(1,0)
= P4(-1,1) + P4(0,0) + P4(1,0)
= P4(-1,1) + P4(-1,0) + P4(0,-1) + P4(1,0)
= P4(-1,1) + P4(-1,0) + P4(0,-1) + P4(0,0) + P4(1,-1)
= P4(-1,1) + P4(-1,0) + P4(0,-1) + P4(-1,0) + P4(0,-1) + P4(1,-1)
= [-1-1] + [-1-0] + [0-(-1)] + [-1-0] + [0-(-1)] + [1-(-1)]
= -2 + (-1) + 1 + (-1) + 1 + 2
= 0

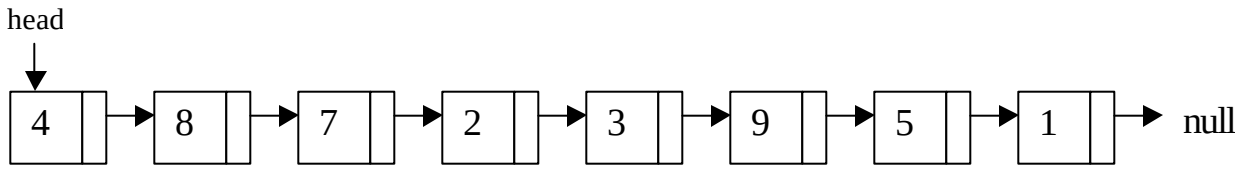
5. (10 points)

Trace the effect of executing the code segment shown below on the linked list structure, also shown below, by drawing the linked list as it would look after the code has been executed.

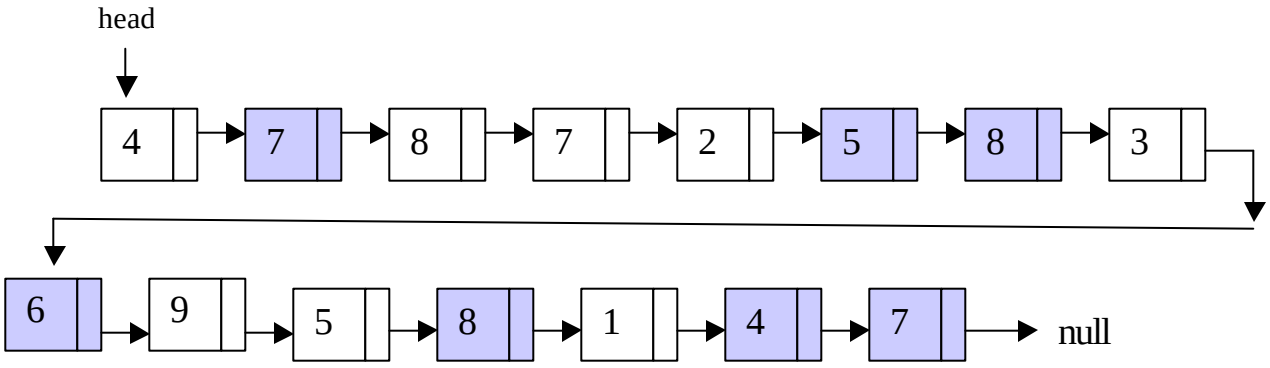
```
struct node{
   int data;
   struct node *next;
};

p = head;
while (p != NULL)
{  if (p->data <= 5)
   {  newp = (struct node *) malloc(sizeof(struct node));
      newp->data = p->data + 3;
      newp->next = p->next;
      p->next = newp;
      p = newp;
   }
   else
      p = p->next;
}
```

The linked list:



Solution: The code adds a new node after every node whose data value is less than or equal to 5. The new node has a data value equal to its predecessor plus 3. (New nodes are shaded.)

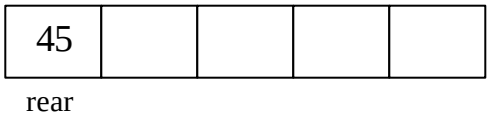


6. (15 points)

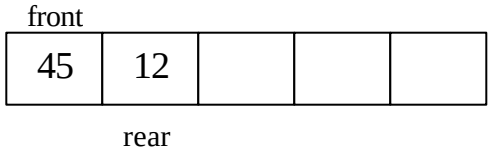
Consider the circular array implementation of a queue in which the array is declared to have a size of 5. Trace the status of the queue by showing the elements in the queue and the position of the front and rear pointers after each of the operations shown below. Assume that the queue is initially empty.

1. enqueue(Q, 45);
2. enqueue(Q, 12);
3. enqueue(Q, 69);
4. enqueue(Q, 54);
5. x = dequeue(Q);
6. enqueue(Q, 98);
7. y = dequeue(Q);
8. enqueue(Q, y);
9. enqueue(Q, x);
10. x = dequeue(Q);

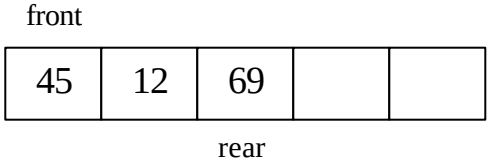
After stmt #1:



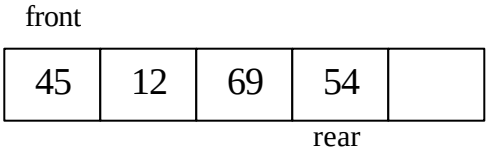
After stmt #2:



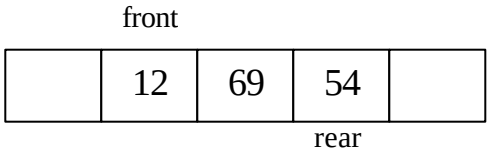
After stmt #3:



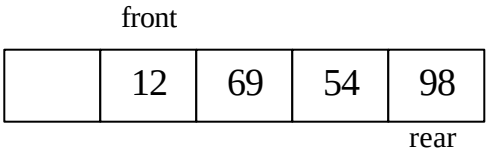
After stmt #4:



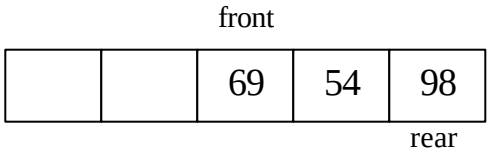
After stmt #5:



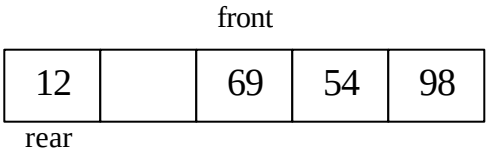
After stmt #6:



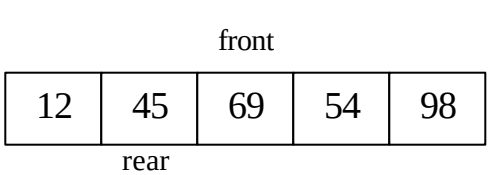
After stmt #7:



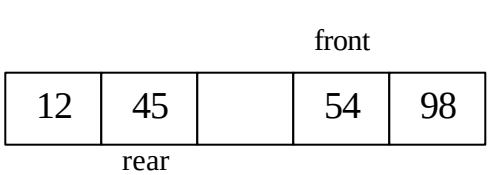
After stmt #8:



After stmt #9:



After stmt #10:



7. (10 points)

You have developed an algorithm which is known to be $O(n^2)$ and can solve a problem instance of size n = 40 in 4 minutes. Your boss has just given you a big problem to solve by the time the board of directors meeting begins at 3:00pm. If it is 2:30pm now and the big problem to be solved is of size n = 120, will you solve the problem in time to ask for a big raise?

$$\frac{(40)^2}{4} = \frac{(120)^2}{t} \quad \Rightarrow \quad t = \frac{(120)^2(4)}{(40)^2} = 4(3)^2 = 36 \text{ min}$$

Sorry! No raise today.

8. (10 points)

For the following code segment give (a) the Big-Oh run-time of the code and (b) the value of the variable x after the loop ends.

```
x = 0;
for (i = 1; i <= 3*n+2;  i++)
   for (j = 1; j <= 2*n-1;  j++)
        x = x + i;
```

(a) $$\sum_{i=1}^{3n+2} \sum_{j=1}^{2n-1} 1 = (2n-1) \sum_{i=1}^{3n+2} 1 = (2n-1)(3n+2) = (6n^2 + 4n - 3n - 2) = 6n^2 + n - 2 = O(n^2)$$

(b) $$\sum_{i=1}^{3n+2} \sum_{j=1}^{2n-1} i = (2n-1) \sum_{i=1}^{3n+2} i = (2n-1)\frac{(3n+2)(3n+3)}{2} = \frac{18n^3 + 21n^2 - 3n - 6}{2}$$

9. (10 points – 5 points each)

Consider the merge-sort algorithm shown below:

```
void Mergesort( int list[ ], int start, int end)
{   int mid;
    if (start < end)
    {     mid = (start + end) /2;
          Mergesort (list, start, mid);
          Mergesort (list, mid+1, end);
          Merge(list, start, mid+1, end);
    }
}
```

The following array is to be sorted using this algorithm.

30	21	40	15	13	8	4	6

(a) How many recursive calls to the Mergesort function will be made to sort this array? (Do not count the original call from main which is not a recursive call.)

Answer: 14

(b) How many calls to the Merge function are made in total?

Answer: 7