AUTUMN MID SEMESTER EXAMINATION-2017

Data Structure and Algorithms Solution & Evaluation Scheme

[CS-2001]

Full Marks: 25

Answer any five questions including question No.1 which is compulsory.
The figures in the margin indicate full marks.
Candidates are required to give their answers in their own words as far as practicable and all parts of a question should be answered at one place only.

Q1 Answer the following questions: (1 x 5)

a) Find the worst case time complexity of the following C function.

```
int func(int n)
{
  int i, j, k, p, q = 0;
  for (i = 1; i<n; ++i)
  {
    p = 0;
    for (j=n; j>1; j=j/2)
      ++p;
    for (k=1; k<p; k=k*2)
      ++q;
  }
  return q;
}
```

Evaluation Scheme:
1 mark for correct answer. Step mark of 0.5 may be awarded by looking into the partial correctness of the answer.

Solution:

```
int func(int n)
{
  int i, j, k, p, q = 0;

  for (i = 1; i < n; ++i) // This loop runs O(n) time
  {
    p = 0;

    for (j=n; j > 1; j=j/2) // This loop runs O(log n) times.
      ++p;

    // Since above loop runs O(log n) times, p = O(log n)
    // This loop runs O(log p) times which log(log n)
    for (k=1; k < p; k=k*2)
      ++q;
  }
  return q;
}
```

So **T(n) = n * (log n + log (log n)) = O (n log n)**

b) An array X[10…25, 15…40] requires four bytes of storage for each element. Base address of the array X is 1500. Calculate the size of array X and determine the address of X[15][20] considering column-major representation.

Evaluation Scheme:
Correct answer on size of the array: 0.5 mark and address calculation: 0.5 mark

Solution:
Number or rows say m = (Ur – Lr) + 1 = [25 – 10] + 1 = 16
Number or columns say n = (Uc – Lc) + 1 = [40 – 15] +1 = 26
So size of the array is [16] [26]

The given values are: BA (Base Address) = 1500, w (word length) = 4 byte, i = 15, j = 20, Lr = 10, Lc = 15, m = 26

Address of X[i][j] = BA + w * [m * (j – Lc) + (i – Lr)]
= 1500 + 4* [16 * (20 – 15) + (15 – 10)]
= 1500 + 4 * [16 * 5 + 5] = 1500 + 4 * [80 + 5] = 1500 + 4 * 85
= 1500 + 340 = 1840

c) Consider the sparse matrix shown below. Diagrammatically represent it in triplet and single header linked list.

Row index ⟶

	0	1	2	3	4 ← Column Index
0	1	4	0	2	0
1	0	0	0	0	2
2	0	0	7	0	0
3	0	0	4	5	0
4	0	0	4	0	0
5	0	6	0	0	0

Evaluation Scheme:
Correct answer on Triplet representation: 0.5 mark & single header linked list representation: 0.5 mark. Step mark of 0.5 may be awarded by looking into the partial correctness of the answer

Solution:
Triplet Representation:

Row Representation

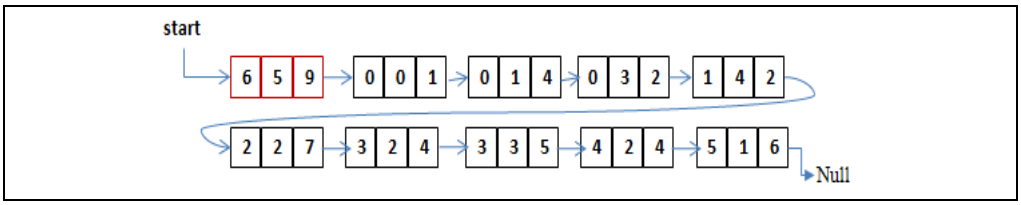
Rows	Columns	Values
6	5	9
0	0	1
0	1	4
0	3	2
1	4	2
2	2	7
3	2	4
3	3	5
4	2	4
5	1	6

OR

Columnar Representation

Rows	6	0	0	0	1	2	3	3	4	5
Columns	5	0	1	3	4	2	2	3	2	1
Values	9	1	4	2	2	7	4	5	4	6

Single Header Link Representation: (node with red border represents the header node and rest represents data nodes.



d) In a single linked list, a node pointer p points to a specific node. The data in p and its previous and next node are same. Write a code segment to delete the duplicate nodes. E.g. if input: 1->3->3->3->5->6 then output: 1->3->5->6

Evaluation Scheme:
1 mark for correct logic. Step mark of 0.5 may be awarded by looking into partial correctness of the logic/code.

Solution:
Let the linked list node structure is

```
struct node
{
    int info;
    struct node *next;
} *start;

// let's p is pointing to the node with second occurrence of value e.g. 3 and start is pointing to the 1st node in the list.
struct node *pptr, *p, *nptr;
for (pptr=start; pptr->next!=p;pptr=pptr->next);
nptr=p->next;
pptr->next=nptr->next;
free (nptr);
free (p);
```

e) What is the output of the following function for a single linked list with the values 1 -> 2 -> 3 -> 4 -> 5-> 6? start is pointing to the first node in the list.

```
void  fun(struct node *start)
{
    if (start == NULL) return;
    if (start-> next != NULL) fun (start-> next -> next);
    printf("%d", start->data);
}
```

Evaluation Scheme:
Correct Output:1 mark and incorrect output with some explanation: 0.5 mark

Output: 5 3 1
Explanation: The traversal would takes place till second last node i.e. 5 and since printf is called post to the recursion, intermediate values would be pushed to the stack. The sequence at which data is pushed to the stack - 1, 3 and 5. For printing the values, items has to be popped off and hence program will display 5 3 1.

Q2 a) Define and explain data structure and abstract data type (ADT) with suitable examples. (2.5)

Evaluation Scheme:
- *Definition of Data structure:0.5 mark*
- *Definition of ADT:0.5 mark*
- *Examples:1.5 mark*

Solution:
Data Structure:-
•A data structure is a specialized/way format for organizing and storing data.
•A data structure is the organization of data in computer memory in such a way that any programming language can use it in an efficient way.

Abstract Data Type (ADT):-
•An abstract data type (ADT) is a mathematical model for data types where a data type is defined by its behavior from the point of view of a user of the data. Or
•An ADT is a set of elements with some well defined operation.

The representation of stack ADT

```
struct stacknode
{
    int info[MAX];
    int top;
};
typedef struct stacknode STACK;
```

Here STACK is the stack ADT. We can define different variables of stack as follows.
STACK s1, s2; //Here s1 and s2 are of type STACK.

b) Design a data structure to represent a polynomial and write an algorithm or pseudocode or C code snippet to multiply two polynomials. (2.5)

Evaluation Scheme:
- *Representation of polynomial with example: 1 mark*
- *Multiplication of two polynomials: 1.5 marks. Appropriate step marks may be awarded by looking into the code)*

Solution:
Array Representation:-
A polynomial may be represented using 1-D array. Array representation assumes that the exponents of the given expression are arranged from 0 to the highest value (degree), which is represented by the subscript of the array beginning with 0. The coefficients of the respective exponent are placed at an appropriate index in the array.
E.g. If the polynomial is $5 + 10x^2 + 6x^3$ then it can be represented using array as:

5	0	10	6	Coefficients
0	1	2	3	Exponents

C code segment to multiply two polynomial:-

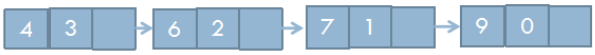
```
// A[] represents coefficients of first polynomial
// B[] represents coefficients of second polynomial
// m and n are sizes of A[] and B[] respectively
void multiply(int A[], int B[], int m, int n)
{
  int prod[]; //Create an array of size m+n-1
  int i,j;

  // Initialize the product polynomial
  for (i = 0; i<m+n-1; i++)
    prod[i] = 0;

  // Multiply two polynomials term by term and Take ever term of first polynomial
  for (i=0; i<m; i++)
  {
    // Multiply the current term of first polynomial with every term of second polynomial.
    for (j=0; j<n; j++)
       prod[i+j] += A[i]*B[j];
  }

  for (i = 0; i<m+n-1; i++)
    printf("%d", prod[i]);
}
```

Linked list Representation:-

```
struct node // Node representation
{
   int coeff;
   int power;
   struct node *next;
};
```

Node of a polynomial:

For example, P(X) = $4X^3 + 6X^2+7X+9$ is represented as



In each node the exponent field will store the corresponding exponent and the coefficient field will store the corresponding coefficient. Link field points to the next item in the polynomial.

- The sign of each coefficient and exponent is stored within the coefficient and the exponent itself
- The storage allocation for each term in the polynomial must be done in descending order of their exponent

```
void multiply(struct node *p1, struct node *p2)
{
   struct node *start3;
   struct node *p2_beg = p2;
   start3=NULL;

   if(p1==NULL || p2==NULL)
   {
      printf("Multiplied polynomial is zero polynomial\n");
      return;
   }

   while(p1!=NULL)
   {
      p2=p2_beg;
      while(p2!=NULL)
      {
         start3=insert(start3,p1->coef*p2->coef,p1->expo+p2->expo);
         p2=p2->link;
      }
      p1=p1->link;
   }
}

struct node *insert(struct node *start,float co,int ex)
{
   struct node *ptr,*tmp;
   tmp=(struct node *)malloc(sizeof(struct node));
   tmp->coef=co;
   tmp->expo=ex;
   /*list empty or exp greater than first one */
   if(start==NULL || ex > start->expo)
   {
      tmp->link=start;
      start=tmp;
   }
   else
   {
      ptr=start;
      while(ptr->link!=NULL && ptr->link->expo >= ex)
         ptr=ptr->link;
      tmp->link=ptr->link;
      ptr->link=tmp;
   }
   return start;
}
```

Q3 a) Write a function rearrange() that exchanges data of every node with its next node, starting from the first node in a single linked list. (2.5)

Example: Input: 1->2->3->4->5->6->7 Output: 2->1->4->3->6->5->7
Input: 1->2 Output: 2->1
Input: 1 Output: 1

Evaluation Scheme:

- *Correct Answer : 2.5 marks*
- *Step marks to be awarded by looking into the correctness of logic.*

Solution:

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

void rearrange(struct node *head) // head is the variable pointing or to point to 1st node.
{
  struct node  *ptr = NULL;
  if (head = = NULL && head -> next = = NULL) // 0 or 1 node
  {
    return;
  }

  for (ptr = head; ptr != NULL || ptr-> next != NULL ; ptr = ptr -> next -> next)
  {
        ptr ->data = ptr ->data + ptr->next->data;
        ptr->next->data = ptr ->data - ptr->next->data;
        ptr->data = ptr ->data - ptr->next->data;
  }
}
```

OR

```
void rearrange(struct node *start)
{
 struct node *ptr, *nptr;
 int temp;
 if (!start || !start->next)
    return;
 ptr = start;
 nptr = start->next;
 while(nptr!=NULL)
 {
   temp = ptr->info;
   ptr->info = nptr->info;
   nptr->info = temp;
   ptr = nptr->next;
   nptr = ptr?ptr->next:NULL;
 }
}
```

b) Write an algorithm or pseudocode or C code snippet to reverse a double linked list by changing node values. (2.5)

Example: Input: 1->2->3->4->5->6->7 Output: 7->6->5->4->3->2->1

Evaluation Scheme:

- *Correct Answer : 2.5 marks*
- *Step marks to be awarded by looking into the correctness of logic.*

Solution:

```
struct node  // node definition
{
  struct node *prev;
  int data;
  struct node *next;
}
// head is the variable pointing to 1st node.
// tail is the variable to point to last node.
void reverse (struct node *head, struct node *tail)
{
  if ((head = = NULL && tail = = NULL)  || head = = tail)
  {
    return;
  }
  for(; head != tail || tail ->prev != head ; head = head->next, tail = tail ->prev)
  {
       head->data = head->data + tail->data;
       tail ->data = head->data - tail->data;
       head ->data = head->data - tail->data;
  }
}
```

Q4 a) Write an algorithm or pseudocode or C code snippet to delete a node with a given data/key and its previous node (if available) in a single linked list. (2.5)

Evaluation Scheme:

- *Logic for traversal to the node: 1mark*
- *Logic for deleting the node and previous(if available):1.5 marks*
- *Appropriate step marks to be awarded by looking into the partial correctness of the logic*

Solution:

```
struct node  // Node definition
{
  int data;
  struct node *next;
};

// head_ref is the variable pointing to 1st node.
```

```
void delete(struct node *head_ref, int itemToDelete)
{
    // Store head node
   struct node *temp = head_ref;
   struct node *prev = NULL;

   // If head node itself holds the key to be deleted
   if (temp != NULL && temp->data == key)
   {
      head_ref = temp->next;   // Changed head
      free(temp);              // free old head
      return;
   }

   // Search for the key to be deleted, keep track of the
   // previous node as we need to change 'prev->next'
   while (temp != NULL && temp->data != itemToDelete)
   {
      prev = temp;
      temp = temp->next;
   }

   // If key was not present in linked list
   if (temp == NULL) return;

   // Unlink the node from linked list
   prev->next = temp->next;

   free(temp);    // Free memory of the node matching the data

   temp = head_ref;

   // Traverse till one node before previous node
   while (temp != NULL && temp->next != prev)
   {
      temp = temp->next;
   }

   if (temp->next = = NULL) return;

    temp -> next = prev -> next;
    free(prev); // Free memory of the previous node
}
```

b) What are advantages and disadvantages of circular linked list over single linked list? (2.5)
Write an algorithm or pseudocode or C code snippet to convert a double linked list

into circular double linked list.

Evaluation Scheme:
- *Advantages and disadvantages: 1.5 marks*
- *Algorithm/C code snippet:1 mark*

Solution:

Advantages:-

1. If we are at a node, then we can go to any node. But in linear linked list it is not possible to go to previous node.
2. It saves time when we have to go to the first node from the last node. It can be done in single step because there is no need to traverse the in between nodes. But in double linked list, we will have to go through in between nodes.

Disadvantages:

1. It is not easy to reverse the linked list.
2. If proper care is not taken, then the problem of infinite loop can occur.
3. If we at a node and go back to the previous node, then we can not do it in single step. Instead we have to complete the entire circle by going through the in between nodes and then we will reach the required node.

Converting a double linked list into circular double linked list:-

Let the double linked list node structure is

```
typedef struct List
{
    int info;
    struct List *prev;
    struct List *next;
}NODE;
```

Code Snippet:-

```
NODE *ptr;
for(ptr=start; ptr!=NULL; ptr=ptr->next);
ptr->next=start;
start->prev=ptr;
```

Q5 Write a function which will traverse the single linked list only once and split the original linked list into two sub-lists, where first and second sub-list contains the even position nodes and odd position nodes of the original list respectively, and then join the second sub-list at the end of the first sub-list. Example: If the list contains 3->4->2->1->7->9->8, then the function need to produce 3->2->7->8->4->1->9. (5)

Evaluation Scheme:
- *Correct Answer : 5 marks*
- *Step marks to be awarded by looking into the partial correctness of logic.*

Solution:

```
struct node  // node definition
{
  int data;
```

```
  struct node *next;
}

void format(struct node *head) // head is pointing to 1st node
{
   struct node *h1=NULL,*ptr,*p,*last;
   if(head==NULL)
      return;
   for(ptr=head;ptr->next!=NULL;ptr=ptr->next)
   {
      p=ptr->next;
      ptr->next=p->next;
      p->next=NULL;
      if(h1==NULL)
      {
         last=p;
         h1=p;
      }
     else
      {
         last->next=p;
         last=p;
      }
      if(ptr->next==NULL)
        break;
   }
   ptr->next=h1;
}
```

Q6 Write a function that will traverse a 1-D array from the beginning to the end and reverse all the increasing and decreasing sequence of components present in the array. Component is the collection of continuous elements either in increasing or decreasing order. Example: Let an array contains 1, 2, 3, 7, 4, 2, 9, 7, 8 elements. Here the components are "1, 2, 3, 7", "4, 2", "9, 7", and "8". The function should produce the array with elements 7, 3, 2, 1, 2, 4, 7, 9, 8. (5)

Evaluation Scheme:

- *Correct Answer : 5 marks*
- *Step marks to be awarded by looking into the partial correctness of logic.*

Solution:

```
// a[] represents the input array with data elements and n is the size of A[]
void format(int a[], int n)
{
   int i,j,order=-1;
   i=0;
   for(j=0; j<n-1; )
```

```
    {
      if(order==-1)
         order=(a[j] > a[j+1])?1:0;
      if((a[j] <= a[j+1] && order==0) || (a[j]>a[j+1] && order==1))
      {
         j++;
      }
      else if((a[j] <= a[j+1] && order==1) || (a[j]>a[j+1] && order==0))
      {
         reverse(a,i,j);
         j++;
         i=j;
         order=-1;
      }
   }
   if(i<j)
      reverse(a,i,j);
}
// reverse the data elements between tow indexes
void reverse(int a[],int low, int up)
{
   int tmp;
   while(low<up)
   {
      tmp=a[low];
      a[low]=a[up];
      a[up]=tmp;
      low++;
      up--;
   }
}
```

Q7 Write Stack ADT. Write Push and Pop operations in the stack. Convert the infix expression (a + b * c / (d * e +f) / g ^ h) to postfix expression using stack. (5)

Evaluation Scheme:

- *Stack ADT: Definition:1 mark*
- *Push and pop operation:2 marks*
- *Infix to postfix conversion for the given expression: 2 marks*

Solution:

The stack abstract data type is defined by the following structure and operations.

Structure: Its is an ordered collection of items where items are added to and removed from the end called the "top." Stacks are ordered LIFO.

Operation:

- push(item) adds a new item to the top of the stack. It needs the item and returns nothing.

- pop() removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.
- peek() returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.
- isEmpty() tests to see whether the stack is empty. It needs no parameters and returns a boolean value.
- size() returns the number of items on the stack. It needs no parameters and returns an integer.

Push and Pop operations in the stack:
Let the stack is defined with static 1-D array with following specification

```
#define N 10;
struct stack
{
  int stackContainer[N];
  int top;
};

void push(struct stack s, int element)
{
  if (s.top = = n-1)
  {
    printf("Overflow");
    return;
   }
   s. stackContainer[s. top++] = element;
}


int pop(struct stack s)
{
  if (s.top = = -1)
  {
    printf("Underflow");
    return;
   }
   return s.stackContainer[s. top--];
}
```

Infix to Postfix Conversion: (a + b * c / (d * e +f) / g ^ h)
Postfix = a b c * de*f+ gh^ / +

Pos	Symbol Scanned	STACK	Postfix Expression
1	(	((	
2	a	((	a
3	+	((+	a
4	b	((+	a b
5	*	((+*	a b
6	c	((+*	a b c
7	/	((+/	a b c *
8	(	((+/(	a b c *

9	d	((+/(	a b c * d
10	*	((+/(*	a b c * d
11	e	((+/(*	a b c * d e
12	+	((+/(+	a b c * d e *
13	f	((+/(+	a b c * d e * f
14	)	((+/	a b c * d e * f +
15	/	((+/	a b c * d e * f + /
16	g	((+/	a b c * d e * f + / g
17	^	((+/^	a b c * d e * f + / g
18	h	((+/^	a b c * d e * f + / g h
19	)		a b c * d e * f + / g h ^ / +

NOTE: Solutions can be written in a different way as well.

COURSE COMMITTEE SIGNATURE

Sr #	Name	Signature
1	Suchismita Das	30/8/17
2	Anil kumar swain	
3	Arup sahoo	
4	Arup A. Acharya	Arup Achya 30/8/17
5	Satanupa Mohanty	
6	Adyasha Das	30/8/17
7	M. Nazma	30/8
8	Pradhy	
9	Nishita Kindo	
10	D. Swain	
11	Amulya Ratna Swain	
12	SUJOY DATTA	
13	Santosh Kumar Pani	
14	Alok	
15	Rajat Kumar Behera	

End of the document