PES University
Department of Computer Science and Engineering

UE19CS202- Data Structures and its Applications(4-0-0-4-4)
UNIT - 1 : Questions and Answers

1) Consider the file node.h has the following declarations; use this to answer all the sub questions in question no. 1.

```
typedef struct node{
char c;
struct node * next;
} Node;
Node * p, * q, * r;
Node x, y, z;
```

i) **For each of the following statements, either describe its effect, or state why it is illegal.**

(a) p = (Node *) malloc(sizeof(Node));

Answer: This operation is legal as it creates a new dynamic variable whose address is given to the pointer p.

(b) * q = (Node *) malloc(sizeof(Node));

Answer: This operation is illegal as * q denotes the variable itself, and not the pointer assigned to it. The statement should read like the one in a).

(c) x = (Node *) malloc(sizeof(Node));

Answer: This operation is illegal as x is a variable predeclared in the program. malloc returns a pointer to the allocated space.

(d) p = r;

Answer: This operation is legal. The pointer p now points to the same item to which r points.

(e) q = y;

Answer: This operation is illegal as p is a pointer, and y is a variable of type structure node. Therefore, they are of different types and cannot be assigned.

(f) r = NULL;

Answer: This operation is legal as r is now assigned to point to NULL, meaning that it is not pointing to any variable.

(g) z = * p;

Answer: This operation is legal provided that p is not equal to NULL The static variable z now contains the same value as the dynamic variable to which p is pointing.

(h) p = * x;

Answer: This operation is illegal as x is not a pointer.
(i) free(y);
Answer: This statement attempts to free a variable, which is illegal.
(j) free(* p);
Answer: This operation is illegal as * p refers to the actual dynamic variable stored in the address pointed to by p, while p gives the address where this variable is stored. The free function, written properly, frees memory formerly occupied by the variable located at the given address p. free(p); would be proper syntax.
(k) free(r);
Answer: This operation is legal if r is not equal to NULL. The space occupied by the variable pointed to by r is returned to available memory.
(l) * q = NULL;
Answer: This operation is also illegal as only pointers can be assigned the address NULL This statement tries to assign the address NULL to a dynamic variable of type Node pointed to by q.
(m) * p = * x;
Answer: This operation is illegal as x is a regular variable, not a pointer like p, and * x is illegal syntax
(n) z = NULL;
Answer: This operation is illegal as z is also a regular variable and cannot be assigned, as is attempted here, to point to an address.

ii) **Write a C function to interchange pointers p and q, so that after the function is performed, p will point to the node to which q formerly pointed, and vice versa.**
Answer:

```
#include “node.h”
/ * Pre: p and q both point to nodes.
Post: p now points to the node that q previously pointed to, and q now points to the node previously pointed to by p. * /
void Swap(Node ** p, Node ** q)
{
        Node * tmp;
        tmp = * q;
        * q = * p;
        * p = tmp;
}
```

iii) **Write a C function to interchange the values in the dynamic variables to which p and q point, so that after the function is performed * p will have the value formerly in * q and vice versa.**

Answer:

```
#include “node.h”
/ * Pre: Both p and q point to nodes.
Post: The values in * p and * q have been exchanged. */
void SwapValues(Node * p, Node * q)
{
        Node tmp;
        tmp = * q;
        * q =  * p;
        * p = tmp;
}
```

iv) **Write a C function that makes p point to the same node to which q points, and frees the item to which p formerly pointed.**

Answer:

```
 #include <stdlib.h>
#include “node.h”
/ * Pre: p points to a node, not to NULL.
Post: p now points to the node (if any) pointed to by q. The node formerly pointed
to by p has been disposed. * /
void Reassign(Node_type ** p, Node–type * q)
{
        if ( * p)
                free( * p);
        * p = q;
}
```

2) What are the advantages and disadvantages of representing a group of items as an array versus a linear linked list?

Random access is faster in array than in linked list.
Less space taken in array since no pointers.

Insertion and deletion of elements is faster in Linked list.
Linked list is used when the number of elements is not known ahead and is variable.

3) Explain how polynomial arithmetic of single variable can be implemented using singly linked list. (i) Specify structure of each node of the list.(ii) Show with an example of how polynomial addition and multiplication operations can be performed using linked list.

```
// Node structure containing power and coefficient of variable
struct Node
{
int coeff;
int pow;
struct Node *next;
};
```

Input:
1st number = 5x^2 + 4x^1 + 2x^0
2nd number = 5x^1 + 5x^0
Addition Output:
5x^2 + 9x^1 + 7x^0
Multiplication Output:
25x^3 + 45x^2 + 30x^1 + 10x^0

Addition of two polynomials using linked list requires comparing the exponents, and wherever the exponents are found to be same, the coefficients are added up. For terms with different exponents, the complete term is simply added to the result thereby making it a part of addition result.

Multiplication of two polynomials however requires manipulation of each node such that the exponents are added up and the coefficients are multiplied. After each term of first polynomial is operated upon with each term of the second polynomial, then the result has to be added up by comparing the exponents and adding the coefficients for similar exponents and including terms as such with dissimilar exponents in the result.

4) Specify the node structure of an integer doubly-linked list (dll). Write a function to delete a node based on the specified index position in the dll. (0 should delete the head node, 1 should delete node after the head and so on). Make sure you handle the boundary conditions.

```
struct dnode
{
```

```
  int data;
struct dnode *llink;
struct dnode *rlink;
};
typedef struct dnode d_node;
struct list
{
d_node* head;
d_node* tail;
int number_of_nodes;
};

typedef struct list d_list;

int delete_At_Position(d_list* ptr_list, int index)
{
    d_node* curr = ptr_list -> head;
    if(index < 0 || index > (ptr_list -> number_of_nodes - 1) || ptr_list -> number_of_nodes    == 0)
    {
       return -1;
     }
    for(int i = 0; i < index; ++i)
    {
           curr = curr -> rlink;
     }
    d_node* prev = curr -> llink;
    d_node* next = curr -> rlink;
    if(prev != NULL)
    {
           prev -> rlink = next;
      }
     else
      {
          ptr_list -> head = next;
       }
      if(next != NULL)
       {
          next -> llink = prev;
        }
       else
        {
           ptr_list -> tail = prev;
        }
       free(curr);
```

```
      --ptr_list -> number_of_nodes;
}
```

5. Write C functions to perform the following operations using Singly Linked List

   a. Append an element to the end of the list
   b. Concatenate two lists
   c. Free all the nodes in a list

```
void insert_tail(struct node **p,int x)
 {
   struct node *temp,*q;
   //create a a node for x
   temp=(struct node*)malloc(sizeof(struct node));
   temp->data=x;
   temp->next=NULL;

   //check if this is the first element
   if(*p ==NULL)
     *p=temp;
  else
  {
    //go to end of list
   q=*p;
   //keep moving forward till you reach end of the list
  while(q->next!=NULL)
     q=q->next;
  q->next=temp;//link new node to the last node
 }
}



concatenate(struct node *first, struct node *second)
{
   while(first->next!=NULL)//go to the end of the first list
     first=first->next
   first->next=second;append the second to the first
}


//function to free all the nodes
free_nodes(struct node *first)
```

```
{
 struct node *prev;
prev=first;
while(first!=NULL)
{
   free(prev);
   first=first->next;
   prev=first;
}
 free(prev);
}
```

6. Write C functions to perform the following operations using Singly Linked List

   a. Reverse the list, so that the last element becomes the first, and so on
   b. Delete the last element from a list
   c. Delete the n^{th} element from a list

//reversing the list

```
void reverse(struct node **p)
  {
    struct node *curr, *prev, *temp;
    prev=NULL;
    curr=*p;
    while(curr!=NULL)
    {
       temp=curr->next;
       curr->next=prev;
       prev=curr;
        curr=temp;
     }
    *p=prev;
   }
```

Delete the last element from a list

```
delete_last(struct node **p)
{
   struct node *q;
   q=p;
  while(q->link!=NULL)//go to the last node
  {
```

```
    prev=q;//keep track of the previous node
    q=q->link;
 }
 free(q);
 prev->link=NULL; //make next of prev node as NULL.
}
```

Delete the nth element from a list

```
void delete_pos(struct node **p, int pos)
  {
    int i;
    struct node *prev,*q;

    i=1;
    q=*p;

   //move forward till the position is reached
    while((q!=NULL) && (i<pos))
    {
      prev=q;
       q=q->next;
      i++;
     }
    if(q==NULL)
      printf("Invalid position..\n");
    else if(prev==NULL)//first position
     *p=q->next;
    else
      prev->next=q->next;//other positions
    free(q);
   }
```

7. Write C functions to perform the following operations using Singly Linked List

    a. Combine two ordered lists into a single ordered list
    b. Form a list containing the union of the elements of two lists
    c. Form a list containing the intersection of the elements of two lists

```
void merge(struct node* p,struct node* q,struct node** t)
{
  while((p!=NULL)&&(q!=NULL))
  {
    if(p->data <= q->data)
    {
      insert_tail(t,p->data);
      p=p->next;
    }
   else
   {
     insert_tail(t,q->data);
      q=q->next;
   }
  }
  if(p==NULL)//end of the first list
  {
    while(q!=NULL)//copy all the elements of the second list
    {
      insert_tail(t,q->data);
      q=q->next;
    }
  }
  else// q==NULL end of the second list
  {
    while(p!=NULL)//copy all the elements of the first list
    {
      insert_tail(t,p->data);
      p=p->next;
     }
   }
}
```

b.Form a list containing the union of the elements of two lists

```
struct Node* getUnion( struct Node* head1, struct Node* head2)
{
    struct Node* result = NULL;
    struct Node *t1 = head1, *t2 = head2;

    // Insert all elements of list1 to the result list
    while (t1 != NULL) {
```

```
        push(&result, t1->data);
        t1 = t1->next;
    }

    // Insert those elements of list2
    // which are not present in result list
    while (t2 != NULL) {
        if (!isPresent(result, t2->data))
            push(&result, t2->data);
        t2 = t2->next;
    }

    return result;
}
```

C.Form a list containing the intersection of the elements of two lists

```
struct Node* getIntersection(struct Node* head1,struct Node* head2)
{
    struct Node* result = NULL;
    struct Node* t1 = head1;

    // Traverse list1 and search each element of it in list2. If the element is present in list 2,
then insert the element to result
    while (t1 != NULL) {
        if (isPresent(head2, t1->data))
            push(&result, t1->data);
        t1 = t1->next;
    }

    return result;
}
```

8. Write C functions to perform the following operations using Singly Linked List

    a. Insert an element after the n^{th} element of a list
    b. Delete every second element from a list
    c. Place the elements of a list in increasing order

Place the elements of a list in increasing order

```
void insert_order(struct node**p,int x)
 {
   struct node *temp, *prev, *q;

   //create a node
  temp=(struct node*)malloc(sizeof(struct node));
  temp->data=x;
  temp->next=NULL;

  q=*p;
  prev=NULL;

  //move forward until the position is reached

  while((q!=NULL)&&(x>q->data))
  {
    prev=q;
    q=q->next;
   }
   if(q!=NULL)//position found
   {
     if(prev==NULL)//insert at first position(smallest number)
     {
       temp->next = q;
       *p=temp;
     }
    else
     {
       temp->next=q;
        prev->next=temp;
     }
   }
  else//q=NULL
   {
     if(prev==NULL)//empty list, first node inserted
      *p=temp;
    else
      prev->next=temp;//insert at end(largest number)
   }
}
```

Insert an element after the n^{th} element of a list