

Solution of Second Test – Data Structure

1. Write a function to add two polynomials represented using single linked list.

```
char compare(int a, int b)
{
    if (a>b) return '>';
    else if (a<b) return '<';
    else return '=';
}

void Poly :: add(int c, int e)
{
    PolyNode *temp = new PolyNode();
    temp->coef = c;
    temp->exp = e;
    if(first == NULL)
    {
        temp -> link = first;
        first = temp;
    }
    else
    {
        PolyNode *a = first;

        if ( e > a -> exp)
        { //adding at begining
            temp->link = a;
            first = temp;
        }
        else
        { //Adding in between
            while(e < a->link->exp && a->link != NULL) a = a ->link;
            temp -> link = a -> link;
            a -> link = temp;
        }
    }
}

Poly Poly :: operator +(Poly p1)
{
    Poly temp;
    PolyNode *a = first, *b = p1.first;
    while( a != NULL && b != NULL )
    {
        switch(compare(a->exp,b->exp))
```

```

        {
            case '>':temp.add(a->coef,a->exp);
                a = a -> link;
                break;
            case '<':temp.add(b->coef,b->exp);
                b = b -> link;
                break;
            case '=': temp.add(a->coef + b->coef, a->exp);
                a = a -> link; b = b -> link;
                break;
        }

    }
    for(;a!=NULL;a=a->link) temp.add(a->coef,a->exp);
    for(;b!=NULL;b=b->link) temp.add(b->coef,b->exp);
    return temp;
}

```

2. Write a function to sort a given circular linked list using selection sort.

```

void List :: sort()
{
    ListNode *a, *temp, *b, *c;
    int t;
    a = first;
    for(b=first;b->link !=a;b=b->link)
    {
        temp = b;
        for(c=b->link; c !=a ; c = c -> link)
        {
            if(temp->data > c ->data)
                temp = c;
        }
        t = temp->data;
        temp -> data = b -> data;
        b -> data = t;
    }
}

```

3. Write a function to create linked list in the sorted order. Consider string as data type in list.

```

void List :: add(char *str)

```

```

{
    ListNode *temp = new ListNode();
    strcpy(temp->s,str);
    ListNode *a = first;
    if ( first == NULL )
    {
        first = temp;
        temp -> link = NULL;
    }
    else
    {
        if ( first -> link == NULL )
        {
            if (strcmp(first->s,str) > 0)
            {
                temp -> link = first;
                first = temp;
            }
        }
        else
        {
            while ( a -> link != NULL && strcmp(a -> link -> s , str) < 0) a = a
-> link;

            if ( a == first)
            {
                temp -> link = first;
                first = temp;
            }
            else
            {
                temp -> link = a -> link;
                a -> link = temp;
            }
        }
    }
}

```

3. Write all four functions (insert and delete at front and rear) required to manage double ended queue using array.

```

void DQueue :: insertFront(int data)
{
    if (front == 0)
        cout << "Can not insert at front";
}

```

```
        else
            q[--front] = data;
    }
void DQueue :: insertRear(int data)
{
    if ( rear == size -1)
        cout << "Can not insert at rear";
    else
        q[++rear] = data;
}
int DQueue :: deleteFront()
{
    if ( front == rear )
    {
        cout << "List empty";
        return -9999;
    }
    else
    {
        return q[front++];
    }
}
int DQueue :: deleteRear()
{
    if ( front == rear )
    {
        cout << "List Empty";
        return -9999;
    }
    else
    {
        return q[rear--];
    }
}
```