Data Structure Assignment - 2

Problems

- Write a C program to create a singly linked list to perform the following operations:
  - Insert element
  - Delete element
  - Search a given element
- Write a C program to create a singly circular linked list to perform the following operations:
  - Insert element
  - Delete element
  - Search a given element
- Write a C program to create a doubly linked list to perform the following operations:
  - Insert element
  - Delete element
  - Search a given element
- Write a C program to add two polynomial using linked lists

Problem - 1

C Code

```
/**
* Data Structures Assignment - 2
* CS 392
* Problem 1:
* - Write a C program to create a singly linked list to perform the following operations:
*    - Insert element
*    - Delete element
*    - Search a given element
*
* Joydeep Mukherjee
* CSE,3rd Sem, 11000116030
* GCETTS 2017
*
**/

#include <stdio.h>
#include <stdlib.h>

typedef struct node_t {
  /* basic node data type */
  int val;
  struct node_t* next;
} node;

void show(node* head,int count) {

  int i;
  node* next = head;
  printf("\tLinked List\n" );
  printf("\t\tHEAD->");
```

```
  for (i = 0; i < count; i++){
    printf("[%d,%d]->",i,next->val);
    fflush(stdout);
    next = next->next;
    if(next == NULL)
      break;
  }
  printf("NULL\n");

}


int insert(node** head, int count) {
  int val, pos, i;
  node* next = *head;

  printf("\tInsert Element in Linked List\n");
  printf("\t\tEnter Value: ");
  scanf("%d",&val);

  node* newNode = (node*) malloc(sizeof(node));

  if(!newNode) {
    printf("\t[ERROR] Memory Error\n");
    return count;
  }

  newNode->val = val;
  newNode->next = NULL;

  if(*head == NULL) {
    *head = newNode;
    return count+1;
  }
  else{
    printf("\t\tEnter Position: ");
    scanf("%d",&pos);

    if(!pos) {
      newNode->next = (*head);
      *head = newNode;
      return count + 1;
    }

    for (i = 0; i < pos -1; i++) {
      if(next->next ==  NULL) {
        printf("\t[WARN] Value: %d will be inserted at end\n",val);
        break;
      }
      next = next->next;
    }

    if(next != NULL)
      newNode->next = next->next;
    else
      newNode->next = NULL;

    next->next = newNode;

    return count+1;
  }
```

```
92
93 }
94
95 int delete(node** head,int count) {
96   int pos, i;
97   node *next = *head;
98   node *tmp = NULL;
99
100  if(next == NULL) {
101    printf("\t[ERROR] List is Empty\n");
102    return 0;
103  }
104
105  printf("\tDelete Element in Linked List\n");
106  show(*head,count);
107  printf("\t\tEnter Index: ");
108  scanf("%d",&pos);
109
110  if(!pos) {
111    tmp = (*head);
112    if((*head)->next != NULL)
113      (*head) = (*head)->next;
114    else
115      (*head) = NULL;
116
117    return count -1;
118  }
119  else {
120    tmp = next;
121    for ( i = 0; i < pos; i++) {
122
123      if(next->next == NULL) {
124        printf("\t[ERROR] Index out of range\n");
125        return count;
126      }
127      tmp = next;
128      next = next->next;
129    }
130    // next is at index to be removed
131    // tmp is at previous index
132
133    if(next->next == NULL)
134      tmp->next = NULL;
135    else
136      tmp->next = next->next;
137
138    free(next);
139
140    return count -1;
141  }
142 }
143
144 void search(node *head, int count) {
145   int i,val;
146   node* next = head;
147
148   printf("\tSearch Element in List:\n");
149
150   if(next == NULL) {
151     printf("\t[ERROR] List is Empty\n");
152     return;
```

```
  }

  printf("\t\tElememt: ");
  scanf("%d",&val);

  for( i = 0; i < count; i++) {

    if(next == NULL) {
      printf("\tEnd Of List, No Results\n");
      return;
    }
    if(next->val == val) {
      printf("\tFound %d at index %d\n",val,i);
      return;
    }

    next = next->next;

  }
  if(next == NULL) {
    printf("\tEnd Of List, No Results\n");
    return;
  }

}


int main(int argc, char const *argv[]) {
  /* */
  int menu_option = -1,count = 0;
  node* head = NULL;
  printf("Your options are:\n");
  printf("\t1. Insert Element\n");
  printf("\t2. Delete Element\n");
  printf("\t3. Search Element by Value\n");
  printf("\t4. Show List\n");
  printf("\t0. Exit\n");
  while (menu_option) {
    printf("Enter option: ");
    fflush(stdin);
    scanf("%d",&menu_option);
    switch (menu_option) {
      case 1:
        count = insert(&head,count);
        break;
      case 2:
        count = delete(&head,count);
        break;
      case 3:
        search(head,count);
        break;
      case 4:
        show(head,count);
        break;
      case 0:
        return 0;
        break;
      default:
        printf("Invalid option\n");
    }
  }
```

```
  return 0;
}

```

Output

```
Your options are:
    1. Insert Element
    2. Delete Element
    3. Search Element by Value
    4. Show List
    0. Exit
Enter option: 1
    Insert Element in Linked List
        Enter Value: 17
Enter option: 1
    Insert Element in Linked List
        Enter Value: 1
        Enter Position: 0
Enter option: 1
    Insert Element in Linked List
        Enter Value: 34
        Enter Position: 5
    [WARN] Value: 34 will be inserted at end
Enter option: 4
    Linked List
        HEAD->[0,1]->[1,17]->[2,34]->NULL
Enter option: 1
    Insert Element in Linked List
        Enter Value: 85
        Enter Position: 3
Enter option: 1
    Insert Element in Linked List
        Enter Value: 51
        Enter Position: 3
Enter option: 4
    Linked List
        HEAD->[0,1]->[1,17]->[2,34]->[3,51]->[4,85]->NULL
Enter option: 1
    Insert Element in Linked List
        Enter Value: 68
        Enter Position: 4
Enter option: 4
    Linked List
        HEAD->[0,1]->[1,17]->[2,34]->[3,51]->[4,68]->[5,85]->NULL
Enter option: 3
    Search Element in List:
        Elememt: 85
    Found 85 at index 5
Enter option: 3
    Search Element in List:
        Elememt: 1
    Found 1 at index 0
Enter option: 2
    Delete Element in Linked List
    Linked List
        HEAD->[0,1]->[1,17]->[2,34]->[3,51]->[4,68]->[5,85]->NULL
        Enter Index: 5
Enter option: 4
```

```
    Linked List
        HEAD->[0,1]->[1,17]->[2,34]->[3,51]->[4,68]->NULL
Enter option: 2
    Delete Element in Linked List
    Linked List
        HEAD->[0,1]->[1,17]->[2,34]->[3,51]->[4,68]->NULL
        Enter Index: 2
Enter option: 4
    Linked List
        HEAD->[0,1]->[1,17]->[2,51]->[3,68]->NULL
Enter option: 2
    Delete Element in Linked List
    Linked List
        HEAD->[0,1]->[1,17]->[2,51]->[3,68]->NULL
        Enter Index: 0
Enter option: 4
    Linked List
        HEAD->[0,17]->[1,51]->[2,68]->NULL
Enter option: 2
    Delete Element in Linked List
    Linked List
        HEAD->[0,17]->[1,51]->[2,68]->NULL
        Enter Index: 0
Enter option: 2
    Delete Element in Linked List
    Linked List
        HEAD->[0,51]->[1,68]->NULL
        Enter Index: 0
Enter option: 4
    Linked List
        HEAD->[0,68]->NULL
Enter option: 2
    Delete Element in Linked List
    Linked List
        HEAD->[0,68]->NULL
        Enter Index: 0
Enter option: 4
    Linked List
        HEAD->NULL
Enter option: 1
    Insert Element in Linked List
        Enter Value: 987
Enter option: 1
    Insert Element in Linked List
        Enter Value: 456
        Enter Position: 0
Enter option: 4
    Linked List
        HEAD->[0,456]->[1,987]->NULL
Enter option: 0

```

Problem 2

C Code

```
/**
* Data Structures Assignment - 2
* CS 392
```

```
/**
* Problem 2:
* - Write a C program to create a singly circular linked list to perform the following operations:
*    - Insert element
*    - Delete element
*    - Search a given element
*
* Joydeep Mukherjee
* CSE,3rd Sem, 11000116030
* GCETTS 2017
*
**/
#include <stdio.h>
#include <stdlib.h>

typedef struct node_t {
  /* basic node data type */
  int val;
  struct node_t* next;
} node;

void show(node* head,int count) {

  int i;
  node* next = head;
  printf("\tLinked List\n" );
  printf("\t\tHEAD->");
  for (i = 0; i < count; i++){
    printf("[%d,%d]->",i,next->val);
    fflush(stdout);
    next = next->next;
    if(next == head)
      break;
  }
  printf("HEAD\n");

}


int insert(node** head, int count) {
  int val, pos, i;
  node* next = *head;

  printf("\tInsert Element in Linked List\n");
  printf("\t\tEnter Value: ");
  scanf("%d",&val);

  node* newNode = (node*) malloc(sizeof(node));

  if(!newNode) {
    printf("\t[ERROR] Memory Error\n");
    return count;
  }

  newNode->val = val;
  newNode->next = (*head);

  if(*head == NULL) {
    *head = newNode;
    newNode->next = (*head);
    return count+1;
```

```
  }
  else{
    printf("\t\tEnter Position: ");
    scanf("%d",&pos);

    if(!pos) {
      newNode->next = (*head);
      *head = newNode;
      return count + 1;
    }

    for (i = 0; i < pos -1; i++) {
      if(next->next ==  (*head)) {
        printf("\t[WARN] Value: %d will be inserted at end\n",val);
        break;
      }
      next = next->next;
    }

    newNode->next = next->next;

    next->next = newNode;

    return count+1;
  }

}

int delete(node** head,int count) {
  int pos, i;
  node *next = *head;
  node *tmp = NULL;

  if(next == NULL) {
    printf("\t[ERROR] List is Empty\n");
    return 0;
  }

  printf("\tDelete Element in Linked List\n");
  show(*head,count);
  printf("\t\tEnter Index: ");
  scanf("%d",&pos);

  if(!pos) {
    tmp = (*head);
    //printf("%d\n",(*head)->next->val );
    if((*head)->next != (*head))
      (*head) = (*head)->next;
    else
      (*head) = NULL;

    free(tmp);
    return count -1;
  }
  else {
    tmp = next;
    for ( i = 0; i < pos; i++) {

      if(next->next == (*head)) {
        printf("\t[ERROR] Index out of range\n");
        return count;
```

```
      }
      tmp = next;
      next = next->next;
    }
    // next is at index to be removed
    // tmp is at previous index

    if(next->next == (*head))
      tmp->next = (*head);
    else
      tmp->next = next->next;

    free(next);

    return count -1;
  }
}

void search(node *head, int count) {
  int i,val;
  node* next = head;

  printf("\tSearch Element in List:\n");

  if(next->next == head) {
    printf("\t[ERROR] List is Empty\n");
    return;
  }

  printf("\t\tElememt: ");
  scanf("%d",&val);

  for( i = 0; i < count; i++) {

    if(next->next == head) {
      printf("\tEnd Of List, No Results\n");
      return;
    }

    if(next->val == val) {
      printf("\tFound %d at index %d\n",val,i);
      return;
    }

    next = next->next;

  }

  if(next == head) {
    printf("\tEnd Of List, No Results\n");
    return;
  }

}

int main(int argc, char const *argv[]) {
  /* */
  int menu_option = -1,count = 0;
  node* head = NULL;
  printf("Your options are:\n");
  printf("\t1. Insert Element\n");
```

```
  printf("\t2. Delete Element\n");
  printf("\t3. Search Element by Value\n");
  printf("\t4. Show List\n");
  printf("\t0. Exit\n");
  while (menu_option) {
    printf("Enter option: ");
    fflush(stdin);
    scanf("%d",&menu_option);
    switch (menu_option) {
      case 1:
        count = insert(&head,count);
        break;
      case 2:
        count = delete(&head,count);
        break;
      case 3:
        search(head,count);
        break;
      case 4:
        show(head,count);
        break;
      case 0:
        return 0;
        break;
      default:
        printf("Invalid option\n");
    }
  }
  return 0;
}

```

Output

```
Your options are:
    1. Insert Element
    2. Delete Element
    3. Search Element by Value
    4. Show List
    0. Exit
Enter option: 1
    Insert Element in Linked List
        Enter Value: 8
Enter option: 1
    Insert Element in Linked List
        Enter Value: 16
        Enter Position: 16
    [WARN] Value: 16 will be inserted at end
Enter option: 1
    Insert Element in Linked List
        Enter Value: 1
        Enter Position: 0
Enter option: 4
    Linked List
        HEAD->[0,1]->[1,8]->[2,16]->HEAD
Enter option: 2
    Delete Element in Linked List
    Linked List
        HEAD->[0,1]->[1,8]->[2,16]->HEAD
```

```
        Enter Index: 0
Enter option: 4
    Linked List
        HEAD->[0,8]->[1,16]->HEAD
Enter option: 1
    Insert Element in Linked List
        Enter Value: 64
        Enter Position: 0
Enter option: 4
    Linked List
        HEAD->[0,64]->[1,8]->[2,16]->HEAD
Enter option: 2
    Delete Element in Linked List
    Linked List
        HEAD->[0,64]->[1,8]->[2,16]->HEAD
        Enter Index: 2
Enter option: 4
    Linked List
        HEAD->[0,64]->[1,8]->HEAD
Enter option: 1
    Insert Element in Linked List
        Enter Value: 69
        Enter Position: 3
Enter option: 4
    Linked List
        HEAD->[0,64]->[1,8]->[2,69]->HEAD
Enter option: 3
    Search Element in List:
        Elememt: 8
    Found 8 at index 1
Enter option: 1
    Insert Element in Linked List
        Enter Value: 48
        Enter Position: 1
Enter option: 4
    Linked List
        HEAD->[0,64]->[1,48]->[2,8]->[3,69]->HEAD
Enter option: 3
    Search Element in List:
        Elememt: 8
    Found 8 at index 2
Enter option: 0

```

Problem 3

C Code

```
/**
* Data Structures Assignment - 2
* CS 392
* Problem 3:
* - Write a C program to create a doubly linked list to perform the following operations:
*    - Insert element
*    - Delete element
*    - Search a given element
*
* Joydeep Mukherjee
* CSE,3rd Sem, 11000116030
```

```
 * GCETTS 2017
 *
 **/

#include <stdio.h>
#include <stdlib.h>

typedef struct node_t {
  /* basic node data type */
  int val;
  struct node_t* next;
  struct node_t* prev;
} node;

void show(node* head,int count) {

  int i;
  node* next = head;
  printf("\tLinked List\n" );
  printf("\t\tHEAD->");
  for (i = 0; i < count; i++){
    printf("[%d,%d]",i,next->val);
    fflush(stdout);
    next = next->next;
    if(next == NULL)
      break;
    else
      printf("<->");
  }
  printf("->NULL\n");

}


int insert(node** head, int count) {
  int val, pos, i;
  node* next = *head;

  printf("\tInsert Element in Linked List\n");
  printf("\t\tEnter Value: ");
  scanf("%d",&val);

  node* newNode = (node*) malloc(sizeof(node));

  if(!newNode) {
    printf("\t[ERROR] Memory Error\n");
    return count;
  }

  newNode->val = val;
  newNode->next = NULL;
  newNode->prev = NULL;

  if(*head == NULL) {
    *head = newNode;
    return count+1;
  }
  else{
    printf("\t\tEnter Position: ");
    scanf("%d",&pos);

```

```
    if(!pos) {
      newNode->next = (*head);
      *head = newNode;
      return count + 1;
    }

    for (i = 0; i < pos -1; i++) {
      if(next->next ==  NULL) {
        printf("\t[WARN] Value: %d will be inserted at end\n",val);
        break;
      }

      next = next->next;
    }

    if(next != NULL) {
      newNode->next = next->next;
      newNode->prev = next;
    }
    else {
      newNode->next = NULL;
      newNode->prev = next;
    }

    next->next = newNode;

    return count+1;
  }

}

int delete(node** head,int count) {
  int pos, i;
  node *next = *head;
  node *tmp = NULL;

  if(next == NULL) {
    printf("\t[ERROR] List is Empty\n");
    return 0;
  }

  printf("\tDelete Element in Linked List\n");
  show(*head,count);
  printf("\t\tEnter Index: ");
  scanf("%d",&pos);

  if(!pos) {
    tmp = (*head);
    if((*head)->next != NULL)
      (*head) = (*head)->next;
    else
      (*head) = NULL;

    return count -1;
  }
  else {
    tmp = next;
    for ( i = 0; i < pos; i++) {

      if(next->next == NULL) {
        printf("\t[ERROR] Index out of range\n");
```

```
        return count;
      }
      tmp = next;
      next = next->next;
    }
    // next is at index to be removed
    // tmp is at previous index

    if(next->next == NULL)
      tmp->next = NULL;
    else {
        tmp->next = next->next;
        next->next->prev = tmp;
    }

    free(next);

    return count -1;
  }
}

void search(node *head, int count) {
  int i,val;
  node* next = head;

  printf("\tSearch Element in List:\n");

  if(next == NULL) {
    printf("\t[ERROR] List is Empty\n");
    return;
  }

  printf("\t\tElememt: ");
  scanf("%d",&val);

  for( i = 0; i < count; i++) {

    if(next == NULL) {
      printf("\tEnd Of List, No Results\n");
      return;
    }

    if(next->val == val) {
      printf("\tFound %d at index %d\n",val,i);
      return;
    }

    next = next->next;

  }

  if(next == NULL) {
    printf("\tEnd Of List, No Results\n");
    return;
  }

}

int main(int argc, char const *argv[]) {
  /* */
  int menu_option = -1,count = 0;
```

```
  node* head = NULL;
  printf("Your options are:\n");
  printf("\t1. Insert Element\n");
  printf("\t2. Delete Element\n");
  printf("\t3. Search Element by Value\n");
  printf("\t4. Show List\n");
  printf("\t0. Exit\n");
  while (menu_option) {
    printf("Enter option: ");
    fflush(stdin);
    scanf("%d",&menu_option);
    switch (menu_option) {
      case 1:
        count = insert(&head,count);
        break;
      case 2:
        count = delete(&head,count);
        break;
      case 3:
        search(head,count);
        break;
      case 4:
        show(head,count);
        break;
      case 0:
        return 0;
        break;
      default:
        printf("Invalid option\n");
    }
  }
  return 0;
}

```

Output

```
Your options are:
    1. Insert Element
    2. Delete Element
    3. Search Element by Value
    4. Show List
    0. Exit
Enter option: 1
    Insert Element in Linked List
        Enter Value: 16
Enter option: 1
    Insert Element in Linked List
        Enter Value: 32
        Enter Position: 1
Enter option: 1
    Insert Element in Linked List
        Enter Value: 1
        Enter Position: 0
Enter option: 4
    Linked List
        HEAD->[0,1]<->[1,16]<->[2,32]->NULL
Enter option: 1
    Insert Element in Linked List
```

```
        Enter Value: 24
        Enter Position: 1
Enter option: 4
    Linked List
        HEAD->[0,1]<->[1,24]<->[2,16]<->[3,32]->NULL
Enter option: 2
    Delete Element in Linked List
    Linked List
        HEAD->[0,1]<->[1,24]<->[2,16]<->[3,32]->NULL
        Enter Index: 1
Enter option: 4
    Linked List
        HEAD->[0,1]<->[1,16]<->[2,32]->NULL
Enter option: 2
    Delete Element in Linked List
    Linked List
        HEAD->[0,1]<->[1,16]<->[2,32]->NULL
        Enter Index: 0
Enter option: 4
    Linked List
        HEAD->[0,16]<->[1,32]->NULL
Enter option: 1
    Insert Element in Linked List
        Enter Value: 1
        Enter Position: 1
Enter option: 4
    Linked List
        HEAD->[0,16]<->[1,1]<->[2,32]->NULL
Enter option: 3
    Search Element in List:
        Elememt: 32
    Found 32 at index 2
Enter option: 3
    Search Element in List:
        Elememt: 16
    Found 16 at index 0
Enter option: 1
    Insert Element in Linked List
        Enter Value: 48
        Enter Position: 44
    [WARN] Value: 48 will be inserted at end
Enter option: 4
    Linked List
        HEAD->[0,16]<->[1,1]<->[2,32]<->[3,48]->NULL
Enter option: 0

```

Problem 4

C Code

```
/**
* Data Structures Assignment - 2
* CS 392
* Problem 4:
* - Write a C program to add two polynomial using linked lists
*
* Joydeep Mukherjee
* CSE,3rd Sem, 11000116030
```

```
 9 * GCETTS 2017
10 *
11 **/
12
13
14 #include <stdio.h>
15 #include <stdlib.h>
16
17 typedef struct node_t {
18   int base;
19   int power;
20   struct node_t* next;
21 }node;
22
23 int insert(node **head) {
24   int base,power,i,iszero = 1;
25   node *newNode = NULL;
26   node *next = *head;
27
28   while(iszero) {
29     scanf("%d %d",&base,&power);
30
31     newNode = (node*) malloc(sizeof(node));
32
33     if(!newNode){
34       printf("\nMemory Error\n");
35       return -1;
36     }
37
38     newNode->base = base;
39     newNode->power = power;
40     newNode->next = NULL;
41
42
43     if((*head) == NULL) {
44
45       *head = newNode;
46       next = *head;
47     }
48     else{
49       next->next = newNode;
50       next = next->next;
51     }
52
53     (!base && !power)?(iszero = 0):(iszero = 1);
54
55   }
56   return 0;
57 }
58 void show(node *head) {
59   node *this = head;
60   while(this != NULL) {
61     printf("%dx^%d ",this->base,this->power);
62     this = this->next;
63     if(this != NULL)
64       printf("+ ");
65     else
66       break;
67   }
68   printf("\n");
69 }
```

```
 70
 71 void add(node *pa,node *pb,node **pc) {
 72   node *newNode = NULL;
 73   node *next = *pc;
 74
 75   int pc_power = 0;
 76   int pc_base = 0;
 77
 78   while(pa != NULL || pa != NULL){
 79     if(pa->power == pb->power) {
 80
 81       pc_base = pa->base + pb->base;
 82       pc_power = pa->power;
 83
 84       pa = pa->next;
 85       pb = pb->next;
 86
 87     }
 88     else if(pa->power > pb->power){
 89
 90       pc_base = pa->base;
 91       pc_power = pa->power;
 92
 93       pa = pa->next;
 94
 95
 96     }
 97     else {
 98
 99       pc_base =  pb->base;
100       pc_power = pb->power;
101
102       pb = pb->next;
103
104     }
105
106     newNode = (node*) malloc(sizeof(node));
107
108     if(!newNode){
109       printf("\nMemory Error\n");
110       return;
111     }
112
113     newNode->base = pc_base;
114     newNode->power = pc_power;
115     newNode->next = NULL;
116
117     if((*pc) == NULL) {
118       *pc = newNode;
119       next = *pc;
120     }
121     else{
122       next->next = newNode;
123       next = next->next;
124     }
125
126   }
127   return;
128 }
129
130 int main(int argc, char const *argv[]) {
```

```
  /* code */
  node *pa,*pb,*pc;

  printf("For polynomial:1 enter the multiplier, power in descending order\n");
  printf("Enter '0 0' to quit\n");
  printf("Format:[mult][pow]\n");
  pa = NULL;
  insert(&pa);

  printf("For polynomial:2 enter the multiplier, power in descending order\n");
  pb = NULL;
  insert(&pb);

  pc = NULL;
  add(pa,pb,&pc);
  printf("polynomial 1: ");
  show(pa);
  printf("polynomial 2: ");
  show(pb);
  printf("Result: ");
  show(pc);

  return 0;
}

```

Output

```
For polynomial:1 enter the multiplier, power in descending order
Enter '0 0' to quit
Format:[mult][pow]
9 9
7 7
5 5
0 0
For polynomial:2 enter the multiplier, power in descending order
10 10
9 9
8 8
14 7
2 2
0 0
polynomial 1: 9x^9 + 7x^7 + 5x^5 + 0x^0

polynomial 2: 10x^10 + 9x^9 + 8x^8 + 14x^7 + 2x^2 + 0x^0
Result: 10x^10 + 18x^9 + 8x^8 + 21x^7 + 5x^5 + 2x^2 + 0x^0
```