

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

```

1  /**
2  * Data Structures Assignment - 2
3  * CS 392
4  * Problem 1:
5  * - Write a C program to create a singly linked list to perform the following operations:
6  *   - Insert element
7  *   - Delete element
8  *   - Search a given element
9  *
10 * Joydeep Mukherjee
11 * CSE,3rd Sem, 11000116030
12 * GCETTS 2017
13 *
14 **/
15
16 #include <stdio.h>
17 #include <stdlib.h>
18
19 typedef struct node_t {
20     /* basic node data type */
21     int val;
22     struct node_t* next;
23 } node;
24
25 void show(node* head,int count) {
26
27     int i;
28     node* next = head;
29     printf("\tLinked List\n" );
30     printf("\t\tHEAD->");

```

```
31  for (i = 0; i < count; i++){
32      printf("[%d,%d]->",i,next->val);
33      fflush(stdout);
34      next = next->next;
35      if(next == NULL)
36          break;
37  }
38  printf("NULL\n");
39
40 }
41
42
43 int insert(node** head, int count) {
44     int val, pos, i;
45     node* next = *head;
46
47     printf("\tInsert Element in Linked List\n");
48     printf("\t\tEnter Value: ");
49     scanf("%d",&val);
50
51     node* newNode = (node*) malloc(sizeof(node));
52
53     if(!newNode) {
54         printf("\t[ERROR] Memory Error\n");
55         return count;
56     }
57
58     newNode->val = val;
59     newNode->next = NULL;
60
61     if(*head == NULL) {
62         *head = newNode;
63         return count+1;
64     }
65     else{
66         printf("\t\tEnter Position: ");
67         scanf("%d",&pos);
68
69         if(!pos) {
70             newNode->next = (*head);
71             *head = newNode;
72             return count + 1;
73         }
74
75         for (i = 0; i < pos -1; i++) {
76             if(next->next == NULL) {
77                 printf("\t[WARN] Value: %d will be inserted at end\n",val);
78                 break;
79             }
80             next = next->next;
81         }
82
83         if(next != NULL)
84             newNode->next = next->next;
85         else
86             newNode->next = NULL;
87
88         next->next = newNode;
89
90         return count+1;
91     }
```

```
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;
```

```
153 }
154
155 printf("\t\tElement: ");
156 scanf("%d",&val);
157
158 for( i = 0; i < count; i++) {
159
160     if(next == NULL) {
161         printf("\tEnd Of List, No Results\n");
162         return;
163     }
164     if(next->val == val) {
165         printf("\tFound %d at index %d\n",val,i);
166         return;
167     }
168
169     next = next->next;
170
171 }
172 if(next == NULL) {
173     printf("\tEnd Of List, No Results\n");
174     return;
175 }
176
177 }
178
179
180 int main(int argc, char const *argv[]) {
181     /* */
182     int menu_option = -1,count = 0;
183     node* head = NULL;
184     printf("Your options are:\n");
185     printf("\t1. Insert Element\n");
186     printf("\t2. Delete Element\n");
187     printf("\t3. Search Element by Value\n");
188     printf("\t4. Show List\n");
189     printf("\t0. Exit\n");
190     while (menu_option) {
191         printf("Enter option: ");
192         fflush(stdin);
193         scanf("%d",&menu_option);
194         switch (menu_option) {
195             case 1:
196                 count = insert(&head,count);
197                 break;
198             case 2:
199                 count = delete(&head,count);
200                 break;
201             case 3:
202                 search(head,count);
203                 break;
204             case 4:
205                 show(head,count);
206                 break;
207             case 0:
208                 return 0;
209                 break;
210             default:
211                 printf("Invalid option\n");
212         }
213     }
```

```
214     return 0;
215 }
216
```

Output

```
1 Your options are:
2     1. Insert Element
3     2. Delete Element
4     3. Search Element by Value
5     4. Show List
6     0. Exit
7 Enter option: 1
8     Insert Element in Linked List
9     Enter Value: 17
10 Enter option: 1
11     Insert Element in Linked List
12     Enter Value: 1
13     Enter Position: 0
14 Enter option: 1
15     Insert Element in Linked List
16     Enter Value: 34
17     Enter Position: 5
18     [WARN] Value: 34 will be inserted at end
19 Enter option: 4
20     Linked List
21     HEAD->[0,1]->[1,17]->[2,34]->NULL
22 Enter option: 1
23     Insert Element in Linked List
24     Enter Value: 85
25     Enter Position: 3
26 Enter option: 1
27     Insert Element in Linked List
28     Enter Value: 51
29     Enter Position: 3
30 Enter option: 4
31     Linked List
32     HEAD->[0,1]->[1,17]->[2,34]->[3,51]->[4,85]->NULL
33 Enter option: 1
34     Insert Element in Linked List
35     Enter Value: 68
36     Enter Position: 4
37 Enter option: 4
38     Linked List
39     HEAD->[0,1]->[1,17]->[2,34]->[3,51]->[4,68]->[5,85]->NULL
40 Enter option: 3
41     Search Element in List:
42     Element: 85
43     Found 85 at index 5
44 Enter option: 3
45     Search Element in List:
46     Element: 1
47     Found 1 at index 0
48 Enter option: 2
49     Delete Element in Linked List
50     Linked List
51     HEAD->[0,1]->[1,17]->[2,34]->[3,51]->[4,68]->[5,85]->NULL
52     Enter Index: 5
53 Enter option: 4
```

```

54     Linked List
55     HEAD->[0,1]->[1,17]->[2,34]->[3,51]->[4,68]->NULL
56 Enter option: 2
57     Delete Element in Linked List
58     Linked List
59     HEAD->[0,1]->[1,17]->[2,34]->[3,51]->[4,68]->NULL
60     Enter Index: 2
61 Enter option: 4
62     Linked List
63     HEAD->[0,1]->[1,17]->[2,51]->[3,68]->NULL
64 Enter option: 2
65     Delete Element in Linked List
66     Linked List
67     HEAD->[0,1]->[1,17]->[2,51]->[3,68]->NULL
68     Enter Index: 0
69 Enter option: 4
70     Linked List
71     HEAD->[0,17]->[1,51]->[2,68]->NULL
72 Enter option: 2
73     Delete Element in Linked List
74     Linked List
75     HEAD->[0,17]->[1,51]->[2,68]->NULL
76     Enter Index: 0
77 Enter option: 2
78     Delete Element in Linked List
79     Linked List
80     HEAD->[0,51]->[1,68]->NULL
81     Enter Index: 0
82 Enter option: 4
83     Linked List
84     HEAD->[0,68]->NULL
85 Enter option: 2
86     Delete Element in Linked List
87     Linked List
88     HEAD->[0,68]->NULL
89     Enter Index: 0
90 Enter option: 4
91     Linked List
92     HEAD->NULL
93 Enter option: 1
94     Insert Element in Linked List
95     Enter Value: 987
96 Enter option: 1
97     Insert Element in Linked List
98     Enter Value: 456
99     Enter Position: 0
100 Enter option: 4
101     Linked List
102     HEAD->[0,456]->[1,987]->NULL
103 Enter option: 0
104

```

Problem 2

C Code

```

1  /**
2  * Data Structures Assignment - 2
3  * CS 392

```

```
4 * Problem 2:
5 * - Write a C program to create a singly circular linked list to perform the following
  operations:
6 *   - Insert element
7 *   - Delete element
8 *   - Search a given element
9 *
10 * Joydeep Mukherjee
11 * CSE,3rd Sem, 11000116030
12 * GCETTS 2017
13 *
14 **/
15 #include <stdio.h>
16 #include <stdlib.h>
17
18 typedef struct node_t {
19     /* basic node data type */
20     int val;
21     struct node_t* next;
22 } node;
23
24 void show(node* head,int count) {
25
26     int i;
27     node* next = head;
28     printf("\tLinked List\n" );
29     printf("\t\tHEAD->");
30     for (i = 0; i < count; i++){
31         printf("[%d,%d]->",i,next->val);
32         fflush(stdout);
33         next = next->next;
34         if(next == head)
35             break;
36     }
37     printf("HEAD\n");
38
39 }
40
41
42 int insert(node** head, int count) {
43     int val, pos, i;
44     node* next = *head;
45
46     printf("\tInsert Element in Linked List\n");
47     printf("\t\tEnter Value: ");
48     scanf("%d",&val);
49
50     node* newNode = (node*) malloc(sizeof(node));
51
52     if(!newNode) {
53         printf("\t[ERROR] Memory Error\n");
54         return count;
55     }
56
57     newNode->val = val;
58     newNode->next = (*head);
59
60     if(*head == NULL) {
61         *head = newNode;
62         newNode->next = (*head);
63         return count+1;
64     }
```

```
64 }
65 else{
66     printf("\t\tEnter Position: ");
67     scanf("%d",&pos);
68
69     if(!pos) {
70         newNode->next = (*head);
71         *head = newNode;
72         return count + 1;
73     }
74
75     for (i = 0; i < pos - 1; i++) {
76         if(next->next == (*head)) {
77             printf("\t[WARN] Value: %d will be inserted at end\n",val);
78             break;
79         }
80         next = next->next;
81     }
82
83     newNode->next = next->next;
84
85     next->next = newNode;
86
87     return count+1;
88 }
89
90 }
91
92 int delete(node** head,int count) {
93     int pos, i;
94     node *next = *head;
95     node *tmp = NULL;
96
97     if(next == NULL) {
98         printf("\t[ERROR] List is Empty\n");
99         return 0;
100     }
101
102     printf("\tDelete Element in Linked List\n");
103     show(*head,count);
104     printf("\t\tEnter Index: ");
105     scanf("%d",&pos);
106
107     if(!pos) {
108         tmp = (*head);
109         //printf("%d\n",(*head)->next->val );
110         if((*head)->next != (*head))
111             (*head) = (*head)->next;
112         else
113             (*head) = NULL;
114
115         free(tmp);
116         return count - 1;
117     }
118     else {
119         tmp = next;
120         for ( i = 0; i < pos; i++) {
121
122             if(next->next == (*head)) {
123                 printf("\t[ERROR] Index out of range\n");
124                 return count;
```



```
125     }
126     tmp = next;
127     next = next->next;
128 }
129 // next is at index to be removed
130 // tmp is at previous index
131
132 if(next->next == (*head))
133     tmp->next = (*head);
134 else
135     tmp->next = next->next;
136
137 free(next);
138
139 return count -1;
140 }
141 }
142
143 void search(node *head, int count) {
144     int i,val;
145     node* next = head;
146
147     printf("\tSearch Element in List:\n");
148
149     if(next->next == head) {
150         printf("\t[ERROR] List is Empty\n");
151         return;
152     }
153
154     printf("\t\tElement: ");
155     scanf("%d",&val);
156
157     for( i = 0; i < count; i++) {
158
159         if(next->next == head) {
160             printf("\tEnd Of List, No Results\n");
161             return;
162         }
163
164         if(next->val == val) {
165             printf("\tFound %d at index %d\n",val,i);
166             return;
167         }
168
169         next = next->next;
170     }
171
172     if(next == head) {
173         printf("\tEnd Of List, No Results\n");
174         return;
175     }
176 }
177
178 }
179
180 int main(int argc, char const *argv[]) {
181     /* */
182     int menu_option = -1,count = 0;
183     node* head = NULL;
184     printf("Your options are:\n");
185     printf("\t1. Insert Element\n");
```

```

186 printf("\t2. Delete Element\n");
187 printf("\t3. Search Element by Value\n");
188 printf("\t4. Show List\n");
189 printf("\t0. Exit\n");
190 while (menu_option) {
191     printf("Enter option: ");
192     fflush(stdin);
193     scanf("%d",&menu_option);
194     switch (menu_option) {
195         case 1:
196             count = insert(&head,count);
197             break;
198         case 2:
199             count = delete(&head,count);
200             break;
201         case 3:
202             search(head,count);
203             break;
204         case 4:
205             show(head,count);
206             break;
207         case 0:
208             return 0;
209             break;
210         default:
211             printf("Invalid option\n");
212     }
213 }
214 return 0;
215 }
216

```

Output

```

1 Your options are:
2     1. Insert Element
3     2. Delete Element
4     3. Search Element by Value
5     4. Show List
6     0. Exit
7 Enter option: 1
8     Insert Element in Linked List
9     Enter Value: 8
10 Enter option: 1
11     Insert Element in Linked List
12     Enter Value: 16
13     Enter Position: 16
14     [WARN] Value: 16 will be inserted at end
15 Enter option: 1
16     Insert Element in Linked List
17     Enter Value: 1
18     Enter Position: 0
19 Enter option: 4
20     Linked List
21     HEAD->[0,1]->[1,8]->[2,16]->HEAD
22 Enter option: 2
23     Delete Element in Linked List
24     Linked List
25     HEAD->[0,1]->[1,8]->[2,16]->HEAD

```

```

26      Enter Index: 0
27 Enter option: 4
28      Linked List
29      HEAD->[0,8]->[1,16]->HEAD
30 Enter option: 1
31      Insert Element in Linked List
32      Enter Value: 64
33      Enter Position: 0
34 Enter option: 4
35      Linked List
36      HEAD->[0,64]->[1,8]->[2,16]->HEAD
37 Enter option: 2
38      Delete Element in Linked List
39      Linked List
40      HEAD->[0,64]->[1,8]->[2,16]->HEAD
41      Enter Index: 2
42 Enter option: 4
43      Linked List
44      HEAD->[0,64]->[1,8]->HEAD
45 Enter option: 1
46      Insert Element in Linked List
47      Enter Value: 69
48      Enter Position: 3
49 Enter option: 4
50      Linked List
51      HEAD->[0,64]->[1,8]->[2,69]->HEAD
52 Enter option: 3
53      Search Element in List:
54      Element: 8
55      Found 8 at index 1
56 Enter option: 1
57      Insert Element in Linked List
58      Enter Value: 48
59      Enter Position: 1
60 Enter option: 4
61      Linked List
62      HEAD->[0,64]->[1,48]->[2,8]->[3,69]->HEAD
63 Enter option: 3
64      Search Element in List:
65      Element: 8
66      Found 8 at index 2
67 Enter option: 0
68

```

Problem 3

C Code

```

1  /**
2  * Data Structures Assignment - 2
3  * CS 392
4  * Problem 3:
5  * - Write a C program to create a doubly linked list to perform the following operations:
6  *   - Insert element
7  *   - Delete element
8  *   - Search a given element
9  *
10 * Joydeep Mukherjee
11 * CSE,3rd Sem, 11000116030

```

```
12 * GCETTS 2017
13 *
14 **/
15
16 #include <stdio.h>
17 #include <stdlib.h>
18
19 typedef struct node_t {
20     /* basic node data type */
21     int val;
22     struct node_t* next;
23     struct node_t* prev;
24 } node;
25
26 void show(node* head,int count) {
27
28     int i;
29     node* next = head;
30     printf("\tLinked List\n" );
31     printf("\t\tHEAD->");
32     for (i = 0; i < count; i++){
33         printf("[%d,%d]",i,next->val);
34         fflush(stdout);
35         next = next->next;
36         if(next == NULL)
37             break;
38         else
39             printf("<->");
40     }
41     printf("->NULL\n");
42
43 }
44
45
46 int insert(node** head, int count) {
47     int val, pos, i;
48     node* next = *head;
49
50     printf("\tInsert Element in Linked List\n");
51     printf("\t\tEnter Value: ");
52     scanf("%d",&val);
53
54     node* newNode = (node*) malloc(sizeof(node));
55
56     if(!newNode) {
57         printf("\t[ERROR] Memory Error\n");
58         return count;
59     }
60
61     newNode->val = val;
62     newNode->next = NULL;
63     newNode->prev = NULL;
64
65     if(*head == NULL) {
66         *head = newNode;
67         return count+1;
68     }
69     else{
70         printf("\t\tEnter Position: ");
71         scanf("%d",&pos);
72
```

```
73     if(!pos) {
74         newNode->next = (*head);
75         *head = newNode;
76         return count + 1;
77     }
78
79     for (i = 0; i < pos - 1; i++) {
80         if(next->next == NULL) {
81             printf("\t[WARN] Value: %d will be inserted at end\n",val);
82             break;
83         }
84
85         next = next->next;
86     }
87
88     if(next != NULL) {
89         newNode->next = next->next;
90         newNode->prev = next;
91     }
92     else {
93         newNode->next = NULL;
94         newNode->prev = next;
95     }
96
97     next->next = newNode;
98
99     return count+1;
100 }
101
102 }
103
104 int delete(node** head,int count) {
105     int pos, i;
106     node *next = *head;
107     node *tmp = NULL;
108
109     if(next == NULL) {
110         printf("\t[ERROR] List is Empty\n");
111         return 0;
112     }
113
114     printf("\tDelete Element in Linked List\n");
115     show(*head,count);
116     printf("\t\tEnter Index: ");
117     scanf("%d",&pos);
118
119     if(!pos) {
120         tmp = (*head);
121         if((*head)->next != NULL)
122             (*head) = (*head)->next;
123         else
124             (*head) = NULL;
125
126         return count - 1;
127     }
128     else {
129         tmp = next;
130         for ( i = 0; i < pos; i++) {
131
132             if(next->next == NULL) {
133                 printf("\t[ERROR] Index out of range\n");
```

```
134     return count;
135 }
136 tmp = next;
137 next = next->next;
138 }
139 // next is at index to be removed
140 // tmp is at previous index
141
142 if(next->next == NULL)
143     tmp->next = NULL;
144 else {
145     tmp->next = next->next;
146     next->next->prev = tmp;
147 }
148
149 free(next);
150
151 return count -1;
152 }
153 }
154
155 void search(node *head, int count) {
156     int i,val;
157     node* next = head;
158
159     printf("\tSearch Element in List:\n");
160
161     if(next == NULL) {
162         printf("\t[ERROR] List is Empty\n");
163         return;
164     }
165
166     printf("\t\tElement: ");
167     scanf("%d",&val);
168
169     for( i = 0; i < count; i++) {
170
171         if(next == NULL) {
172             printf("\tEnd Of List, No Results\n");
173             return;
174         }
175
176         if(next->val == val) {
177             printf("\tFound %d at index %d\n",val,i);
178             return;
179         }
180
181         next = next->next;
182
183     }
184
185     if(next == NULL) {
186         printf("\tEnd Of List, No Results\n");
187         return;
188     }
189
190 }
191
192 int main(int argc, char const *argv[]) {
193     /* */
194     int menu_option = -1,count = 0;
```

```

195 node* head = NULL;
196 printf("Your options are:\n");
197 printf("\t1. Insert Element\n");
198 printf("\t2. Delete Element\n");
199 printf("\t3. Search Element by Value\n");
200 printf("\t4. Show List\n");
201 printf("\t0. Exit\n");
202 while (menu_option) {
203     printf("Enter option: ");
204     fflush(stdin);
205     scanf("%d",&menu_option);
206     switch (menu_option) {
207         case 1:
208             count = insert(&head,count);
209             break;
210         case 2:
211             count = delete(&head,count);
212             break;
213         case 3:
214             search(head,count);
215             break;
216         case 4:
217             show(head,count);
218             break;
219         case 0:
220             return 0;
221             break;
222         default:
223             printf("Invalid option\n");
224     }
225 }
226 return 0;
227 }
228

```

Output

```

1 Your options are:
2     1. Insert Element
3     2. Delete Element
4     3. Search Element by Value
5     4. Show List
6     0. Exit
7 Enter option: 1
8     Insert Element in Linked List
9     Enter Value: 16
10 Enter option: 1
11     Insert Element in Linked List
12     Enter Value: 32
13     Enter Position: 1
14 Enter option: 1
15     Insert Element in Linked List
16     Enter Value: 1
17     Enter Position: 0
18 Enter option: 4
19     Linked List
20     HEAD->[0,1]<->[1,16]<->[2,32]->NULL
21 Enter option: 1
22     Insert Element in Linked List

```

```

23      Enter Value: 24
24      Enter Position: 1
25 Enter option: 4
26      Linked List
27      HEAD->[0,1]<->[1,24]<->[2,16]<->[3,32]->NULL
28 Enter option: 2
29      Delete Element in Linked List
30      Linked List
31      HEAD->[0,1]<->[1,24]<->[2,16]<->[3,32]->NULL
32      Enter Index: 1
33 Enter option: 4
34      Linked List
35      HEAD->[0,1]<->[1,16]<->[2,32]->NULL
36 Enter option: 2
37      Delete Element in Linked List
38      Linked List
39      HEAD->[0,1]<->[1,16]<->[2,32]->NULL
40      Enter Index: 0
41 Enter option: 4
42      Linked List
43      HEAD->[0,16]<->[1,32]->NULL
44 Enter option: 1
45      Insert Element in Linked List
46      Enter Value: 1
47      Enter Position: 1
48 Enter option: 4
49      Linked List
50      HEAD->[0,16]<->[1,1]<->[2,32]->NULL
51 Enter option: 3
52      Search Element in List:
53      Element: 32
54      Found 32 at index 2
55 Enter option: 3
56      Search Element in List:
57      Element: 16
58      Found 16 at index 0
59 Enter option: 1
60      Insert Element in Linked List
61      Enter Value: 48
62      Enter Position: 44
63      [WARN] Value: 48 will be inserted at end
64 Enter option: 4
65      Linked List
66      HEAD->[0,16]<->[1,1]<->[2,32]<->[3,48]->NULL
67 Enter option: 0
68

```

Problem 4

C Code

```

1  /**
2  * Data Structures Assignment - 2
3  * CS 392
4  * Problem 4:
5  * - Write a C program to add two polynomial using linked lists
6  *
7  * Joydeep Mukherjee
8  * 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         if((*head) == NULL) {
43             *head = newNode;
44             next = *head;
45         }
46         else{
47             next->next = newNode;
48             next = next->next;
49         }
50
51         (!base && !power)?(iszero = 0):(iszero = 1);
52     }
53     return 0;
54 }
55
56 void show(node *head) {
57     node *this = head;
58     while(this != NULL) {
59         printf("%dx^%d ",this->base,this->power);
60         this = this->next;
61         if(this != NULL)
62             printf("+ ");
63         else
64             break;
65     }
66     printf("\n");
67 }
```

```
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 || pb != 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[]) {
```

```
131  /* code */
132  node *pa,*pb,*pc;
133
134  printf("For polynomial:1 enter the multiplier, power in descending order\n");
135  printf("Enter '0 0' to quit\n");
136  printf("Format:[mult][pow]\n");
137  pa = NULL;
138  insert(&pa);
139
140  printf("For polynomial:2 enter the multiplier, power in descending order\n");
141  pb = NULL;
142  insert(&pb);
143
144  pc = NULL;
145  add(pa,pb,&pc);
146  printf("polynomial 1: ");
147  show(pa);
148  printf("polynomial 2: ");
149  show(pb);
150  printf("Result: ");
151  show(pc);
152
153  return 0;
154 }
155
```

Output

```
1 For polynomial:1 enter the multiplier, power in descending order
2 Enter '0 0' to quit
3 Format:[mult][pow]
4 9 9
5 7 7
6 5 5
7 0 0
8 For polynomial:2 enter the multiplier, power in descending order
9 10 10
10 9 9
11 8 8
12 14 7
13 2 2
14 0 0
15 polynomial 1: 9x^9 + 7x^7 + 5x^5 + 0x^0
16
17 polynomial 2: 10x^10 + 9x^9 + 8x^8 + 14x^7 + 2x^2 + 0x^0
18 Result: 10x^10 + 18x^9 + 8x^8 + 21x^7 + 5x^5 + 2x^2 + 0x^0
```