احمد حسن احمد عماره – 20220017

Sheet #3

Q1 :

```
188
189  int Count(List * pl , ListEntry e) // Question 1
190  {
191      ListNode * pn = pl->head;
192      int counter = 0;
193      for (int i =0; i < pl->size; i++)
194      {
195          if(pn->entry == e)
196              return 1;
197
198          pn = pn->next;
199      }
200
201      return 0;
202
203  }
204
```

Q2 :

main.cpp | ArrayList.cpp | ArrayList.h

```
1   #ifndef ARRAYLIST_H_INCLUDED
2   #define ARRAYLIST_H_INCLUDED
3
4   #define MaxList 10
5
6   typedef int ListEntry;
7
8   typedef struct List
9   {
10      ListEntry entry[MaxList];
11      int size;
12  };
13
14  void CreateList(List *pl);
15  void InsertList(List *pl , ListEntry e , int pos);
16  void DeleteList(List *pl , ListEntry *pe , int pos);
17  int ListEmpty(List *pl);
18  int ListFull(List *pl);
19  int ListSize(List *pl);
20  void DestroyList(List *pl);
21  void RetriveList(List *pl , ListEntry *pe , int pos);
22  void ReplaceList(List *pl , ListEntry e , int pos);
23  void TraverseList(List *pl , void (*pf)(ListEntry));
24  void Display(ListEntry e);
25  void CopyList(List *pl1 , List *pl2);
26  #endif // ARRAYLIST_H_INCLUDED
27
```

main.cpp | *ArrayList.cpp | ArrayList.h

```cpp
#include<stdlib.h>
#include<iostream>
#include "ArrayList.h"

void CreateList(List *pl)
{
    pl->size=0;
}

void InsertList(List *pl , ListEntry e , int pos)
{
    for(int i = pl->size-1; i >= pos ; i--)
        pl->entry[i+1] = pl->entry[i];

    pl->entry[pos] = e;
    pl->size++;
}

void DeleteList(List *pl , ListEntry *pe , int pos)
{
     *pe = pl->entry[pos];

    for(int i = pos + 1; i <=pl->size-1; i++)
        pl->entry[i-1] = pl->entry[i];

    pl->size--;
}

int ListEmpty(List *pl)
{
   if(pl->size == 0)
    return 1;
    else
        return 0;
}

```

Q3 :

main.cpp | *LinkedList.cpp | LinkedList.h

```cpp
void CopyList(List *pl, List *pcl)
{
    ListNode *pn1 = pl->head, *pn2;

    while (pn1)
    {
        ListNode *tmp = (ListNode *)malloc(sizeof(ListNode));
        if (!tmp)
        {
            std::cout << "Couldn't allocate memory\n";
            exit(0);
        }

        tmp->entry = pn1->entry;
        tmp->next = NULL;

        if (pcl->head == NULL)
        {
            pcl->head = tmp;
            pn2 = tmp;
        }
        else
        {
            pn2->next = tmp;
            pn2 = tmp;
        }

        pn1 = pn1->next;
        pcl->size++;
    }
}

```

Q4 :

```
91  void RetrieveList(List *pl , ListEntry *pe , int pos) // Question 4
92  {
93      ListNode *ln;
94      int i;
95      for(ln = pl->head , i=0; i < pos; i++)
96          ln= ln->next;
97      *pe = ln->entry;
98  }
99
```

Q5 :

main.cpp | *LinkedList.cpp | LinkedList.h

```
66  void DeleteList(List *pl , ListEntry *pe , int pos) // Question 5
67  {
68      ListNode *ln , *tmp;
69      int i ;
70
71      if(pos==0)
72      {
73          *pe = pl->head->entry;
74          tmp = pl->head->next;
75          free(pl->head);
76          pl->head = tmp;
77      }
78
79      else{
80          for(ln= pl->head , i=0; i < pos-1; i++)
81              ln = ln->next;
82          *pe = ln->next->entry;
83          tmp = ln->next->next;
84          free(ln->next);
85          ln->next = tmp;
86      }
87
88      pl->size--;
89  }
90
```

Q6 :

```
204 void Insertbeginning(List *pl , ListEntry e) // Question 6
205 {
206     ListNode * pn = (ListNode*)malloc(sizeof(ListNode));
207     pn->entry = e;
208     pn->next = pl->head;
209     pl->head = pn;
210     pl->size++;
211 }
212
213
```

Q8 :

```
229
230  void RemoveDuplicates(List *pl) {
231      if (pl == NULL || pl->head == NULL) {
232          exit(1);
233      }
234
235      ListNode *pn = pl->head;
236
237      while (pn != NULL && pn->next != NULL) {
238
239          if (pn->entry == pn->next->entry) {
240              ListNode *temp = pn->next;
241              pn->next = temp->next;
242              free(temp);
243              pl->size--;
244          }
245
246          else {
247              pn = pn->next;
248          }
249      }
250  }
251
```

Q9 :

```
214  void ReverseList(List *pl) // Question 9
215  {
216      ListNode *prev = NULL;
217      ListNode *pn = pl->head;
218      ListNode *next;
219
220      while (pn != NULL) {
221          next = pn->next;
222          pn->next = prev;
223          prev = pn;
224          pn = next;
225      }
226
227      pl->head = prev;
228  }
```