Ans_Sheet_5

1) Implement the List using array (not linked).

Ans

List .c

```c
#include "list.h"

void Create(ListType *l)
{
    l->size = 0;
}

int IsEmpty(ListType *l)
{
    return l->size == 0;
}

int IsFull(ListType *l)
{
    return l->size == Max;
}

void Insert(ListType *l , int p , EntryType e)
{
    int i;

    for(i = l->size ; i > p ; i --)
    {
        l->entry[i] = l->entry[i - 1];
    }
    l->entry[p] = e;
    l->size ++;
}

void Retrieve(ListType *l , int p , EntryType *e)
{
    *e = l->entry[p];
    int i;

    for(i = p ; i < l->size - 1 ; i++)
    {
        l->entry[i] = l->entry[i + 1];
    }
    l->size --;

}
```

List .h

```c
#ifndef LIST_H_INCLUDED
#define LIST_H_INCLUDED
#define Max 10
typedef int EntryType;

typedef struct
{
    int size;
    EntryType entry[Max];

}ListType;

void Create(ListType *l);
int IsEmpty(ListType *l);
int IsFull(ListType *l);
void Insert(ListType *l , int p , EntryType e);
void Retrieve(ListType *l , int p , EntryType *e);

#endif // LIST_H_INCLUDED
```

2) Implement Stack as a linked list.

Ans

Stack .c

```c
#include "stack.h"

void Create(Stack *s)
{
    s->top = '\0';
}

int IsEmpty(Stack *s)
{
    return (s->top == '\0');
}

int IsFull(Stack *s)
{
    return 0 ;
}

void Push(Stack *s , EntryType e)
{
    StackNode *a = (StackNode *)malloc(sizeof(StackNode));
    a->info = e;
    a->next = s->top;
    s->top = a;
}

void Pop(Stack *s , EntryType *e)
{
    *e = s->top->info;
    StackNode *q;
    q = s->top;
    s->top = q->next;
    free(q);
}
```

```c

void Clear(Stack *s)
{
    StackNode *b;
    while(b)
    {
        b = s->top;
        s->top = b->next;
        free(b);
    }

}

```

Stack .h

```
#ifndef STACK_H_INCLUDED
#define STACK_H_INCLUDED

typedef int EntryType;

typedef struct st
{
    EntryType info;
    struct st * next;
}StackNode;

typedef struct
{

StackNode *top;

}Stack;

void Create(Stack *s);
int IsEmpty(Stack *s);
int IsFull(Stack *s);
void Push(Stack *s , EntryType e);
void Pop(Stack *s , EntryType *e);
void Clear(Stack *s);


#endif // STACK_H_INCLUDED

```

---

3) Re-solve sheet 2 but for Linked Stack .

i. Write a function that returns the first element entered to a stack. (implementation level)

Ans

```
EntryType First(StackType *s)
{
    StackNode *a;
    a = s->top;
    while(a->next != '\0')
    {
        a = a->next;
    }

    return (a->info);
}
```

ii. Write a function that returns a copy from the last element in a stack. (implementation level)

Ans

```
EntryType Last(StackType *s)
{
    EntryType item = s->top->info;
    return item;
}
```

iii. Write a function to destroy a stack. (implementation level)

Ans

```
void Destroy(StackType *s)
{
    StackNode *q;
    q = s->top;
    while(q)
    {
        s->top = q->next;
        free(q);
        q = s->top;
    }
}
```

iv. Write a function to copy a stack to another. (implementation level)

Ans

```
void Copy(StackType *s1 , StackType *s2)
{
    StackNode *q;
    StackType a;
    Create(&a);
    q = s1->top;
    EntryType holder;
    while(q)
    {
        holder = q->info;
        Push(&a,holder);
        q = q->next;
    }
    while(!IsEmpty(&a))
    {
        Pop(&a,&holder);
        Push(s2,holder);
    }

}
```

v. Write a function to return the size of a stack (implementation level)

Ans

```
int Size(StackType *s)
{
    StackNode *b = s->top;
    int count = 0;
    while(b)
    {
        b = b->next;
        count ++;
    }
    return count;
}
```

vi. Write a function that returns the first element entered to a stack. (user level)

Ans

```
EntryType first(StackType *s)
{
     EntryType item;
     while(!IsEmpty(s))
     {
          Pop(s,&item);
     }
     return item;
}
```

vii. Write a function that returns a copy from the last element in a stack. (user level)

Ans

```
EntryType last(StackType *s)
{
     EntryType item;

     Pop(s,&item);

     return item;
}
```

viii. Write a function to destroy a stack. (user level)

Ans

```
void Destroy(StackType *s)
{
     StackNode *q;
     q = s->top;
     while(q)
     {
          s->top = q->next;
          free(q);
          q = s->top;
     }
}
```

ix. Write a function to copy a stack to another. (user level)

Ans

```
void copy(StackType *s1 , StackType *s2)
{
     StackType s;
     Create(&s);
     EntryType a;
     while(!IsEmpty(s1))
     {
          Pop(s1,&a);
          Push(&s,a);
     }
     while(!IsEmpty(&s))
     {
          Pop(&s,&a);
          Push(s2,a);
     }

}
```

x. Write a function to return the size of a stack (user level)

Ans

```
int size(StackType s)
{
     int count = 0;
     EntryType i;
     while(!IsEmpty(s))
     {
          Pop(s,&i);
          count ++;
     }
     return count;
}
```

xi. Write a function to print on the screen the contents of a stack without changing the stack (user level)

Ans

```
void Print(StackType *s)
{
     StackType a;
     Create(&a);
     EntryType t;

     while(!IsEmpty(s))
     {
          Pop(s,&t);
          printf("%d\n",t);
          Push(&a,t);
     }
     while(!IsEmpty(&a))
     {
          Pop(&a,&t);
          Push(s,t);
     }
}
```

4) Implement Queue as a linked list.

Ans

Queue .c

```c
#include "queue.h"

void Create(QueueType *q)
{
    q->front = '\0';
    q->rear = '\0';
    q->size = 0;
}

int IsEmpty(QueueType *q)
{
    return (q->front == '\0');
}

int IsFull(QueueType *q)
{
    return 0;
}

void Enqueue(QueueType *q , EntryType e)
{
    Node *a = (Node *)malloc(sizeof(Node));
    a->info = e;
    a->next = '\0';
    if(q->front == '\0')
    {
        q->front = a;
    }
    else
    {
        q->rear->next = a;
    }
    q->rear = a;
    q->size ++;

}

void Dequeue(QueueType *q , EntryType *e)
{
    if(q->front->next == '\0')
    {
        *e = q->front->info;
        q->front = '\0';
        q->rear = '\0';
    }
    else
    {

        Node *s;
        s = q->front;
        q->front = s->next;
        *e = s->info;
        free(s);

    }
}

```

```
57
58  EntryType Last(QueueType *q)
59  {
60      Node *s;
61      s = q->front;
62      while(s->next)
63      {
64          s = s->next;
65      }
66
67      return (s->info);
68  }
69
70  EntryType First(QueueType *q)
71  {
72      return (q->front->info);
73  }
74
75  void Destroy(QueueType *q)
76  {
77      Node *x = q->front;
78      while(x)
79      {
80          q->front = x->next;
81          free(x);
82          x = q->front;
83      }
84  }
```

```
85
86  void Copy(QueueType *q1 , QueueType *q2)
87  {
88      Node *a = q1->front;
89      while(a)
90      {
91          Node *s = (Node *)malloc(sizeof(Node));
92          s->info = a->info;
93          s->next = '\0';
94          if(q2->front == '\0')
95          {
96              q2->front = s;
97          }
98          else
99          {
100             q2->rear->next = s;
101         }
102         q2->rear = s;
103         q2->size ++;
104         a = a->next;
105     }
106
107 }
108
109 int Size(QueueType *q)
110 {
111     Node *s;
112     s = q->front;
113     int x = 0;
114     while(s)
115     {
116         s = s->next;
117         x ++;
118     }
119     return x;
120 }
```

6)Write the function void Join List(List *pl1, List *pl2) that copies all entries from pl1 onto the end of pl2. (in both levels).

Ans

implementation level

```
void JoinList(ListType *l1, ListType *l2)
{
    int i;
    for(i = 0 ; i < l1->size ; i++ )
    {
        l2->L[l2->size] = l1->L[i];
        l2->size ++;


    }

}
```

7)Try Your Own

8) Try Your Own