

## Ans\_Sheet\_5

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

Ans

List .c

```
1  #include "list.h"
2
3  void Create(ListType *l)
4  {
5      l->size = 0;
6  }
7
8  int IsEmpty(ListType *l)
9  {
10     return l->size == 0;
11 }
12
13 int IsFull(ListType *l)
14 {
15     return l->size == Max;
16 }
17
18 void Insert(ListType *l , int p , EntryType e)
19 {
20     int i;
21
22     for(i = l->size ; i > p ; i --)
23     {
24         l->entry[i] = l->entry[i - 1];
25     }
26     l->entry[p] = e;
27     l->size ++;
28 }
29
30 void Retrieve(ListType *l , int p , EntryType *e)
31 {
32     *e = l->entry[p];
33     int i;
34
35     for(i = p ; i < l->size - 1 ; i++)
36     {
37         l->entry[i] = l->entry[i + 1];
38     }
39     l->size --;
40 }
41 }
```

List .h

```
1  #ifndef LIST_H_INCLUDED
2  #define LIST_H_INCLUDED
3  #define Max 10
4  typedef int EntryType;
5
6  typedef struct
7  {
8      int size;
9      EntryType entry[Max];
10 }ListType;
11
12 void Create(ListType *l);
13 int IsEmpty(ListType *l);
14 int IsFull(ListType *l);
15 void Insert(ListType *l , int p , EntryType e);
16 void Retrieve(ListType *l , int p , EntryType *e);
17
18 #endif // LIST_H_INCLUDED
```

## 2) Implement Stack as a linked list.

Ans

### Stack .c

```
1  #include "stack.h"
2
3  void Create(Stack *s)
4  {
5      s->top = '\0';
6  }
7
8  int IsEmpty(Stack *s)
9  {
10     return (s->top == '\0');
11 }
12
13 int IsFull(Stack *s)
14 {
15     return 0 ;
16 }
17
18 void Push(Stack *s , EntryType e)
19 {
20     StackNode *a = (StackNode *)malloc(sizeof(StackNode));
21     a->info = e;
22     a->next = s->top;
23     s->top = a;
24 }
25
26 void Pop(Stack *s , EntryType *e)
27 {
28     *e = s->top->info;
29     StackNode *q;
30     q = s->top;
31     s->top = q->next;
32     free(q);
33 }
34
35 void Clear(Stack *s)
36 {
37     StackNode *b;
38     while(b)
39     {
40         b = s->top;
41         s->top = b->next;
42         free(b);
43     }
44 }
45
46
```

## Stack.h

```
1  #ifndef STACK_H_INCLUDED
2  #define STACK_H_INCLUDED
3
4  typedef int EntryType;
5
6  typedef struct st
7  {
8      EntryType info;
9      struct st * next;
10 } StackNode;
11
12 typedef struct
13 {
14     StackNode *top;
15 } Stack;
16
17
18
19 void Create(Stack *s);
20 int IsEmpty(Stack *s);
21 int IsFull(Stack *s);
22 void Push(Stack *s , EntryType e);
23 void Pop(Stack *s , EntryType *e);
24 void Clear(Stack *s);
25
26
27 #endif // STACK_H_INCLUDED
28
```

### 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**

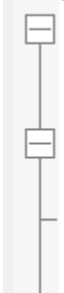
```

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



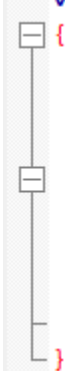
```
EntryType last(StackType *s)
{
    EntryType item;

    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

```
1  #include "queue.h"
2
3  void Create(QueueType *q)
4  {
5      q->front = '\0';
6      q->rear = '\0';
7      q->size = 0;
8  }
9
10 int IsEmpty(QueueType *q)
11 {
12     return (q->front == '\0');
13 }
14
15 int IsFull(QueueType *q)
16 {
17     return 0;
18 }
19
20 void Enqueue(QueueType *q , EntryType e)
21 {
22     Node *a = (Node *)malloc(sizeof(Node));
23     a->info = e;
24     a->next = '\0';
25     if(q->front == '\0')
26     {
27         q->front = a;
28     }
29     else
30     {
31         q->rear->next = a;
32     }
33     q->rear = a;
34     q->size ++;
35 }
36
37
38 void Dequeue(QueueType *q , EntryType *e)
39 {
40     if(q->front->next == '\0')
41     {
42         *e = q->front->info;
43         q->front = '\0';
44         q->rear = '\0';
45     }
46     else
47     {
48         Node *s;
49         s = q->front;
50         q->front = s->next;
51         *e = s->info;
52         free(s);
53     }
54 }
55
56
57
```

```

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 int Size(QueueType *q)
109 {
110     Node *s;
111     s = q->front;
112     int x = 0;
113     while(s)
114     {
115         s = s->next;
116         x ++;
117     }
118     return x;
119 }
120

```

6) Write the function void Join List(List \*p1, List \*p2) that copies all entries from p1 onto the end of p2. (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