Linked Sheet 5

2. Implement StackADT as a linked list.

LINKEDSTACK.H

```
#ifndef LINKEDSTACK_H_INCLUDED
#define LINKEDSTACK_H_INCLUDED
typedef int stackentry;
typedef struct node{
    stackentry entry ;
 struct node *next;
}Node;
typedef struct {
Node *top;
int Size;
}Stack;

void createstack(Stack *ps);
void push (Stack *ps,stackentry e);
void pop(Stack *ps,stackentry *e);
int stackfull(Stack s);
int stackempty(Stack s);

#endif // LINKEDSTACK_H_INCLUDED
```

LINKEDSTACK.C

```
#include"linkedstack.h"
#include <stdlib.h>
#include <stddef.h>
void createstack(Stack *ps){

    ps->top= NULL;
    ps->Size=0;

}
void push (Stack *ps,stackentry e){
    Node *pn=(Node*)malloc(sizeof (Node));
    pn->entry=e;
    pn->next=ps->top;
    ps->top=pn;
    ps->Size++;

}
void pop(Stack *ps,stackentry *e){
    Node *pn;
    pn=ps->top;
    *e=ps->top->entry;
    ps->top=pn->next;
    free(pn);
    ps->Size--;
}


int stackempty(Stack s){

    return s.top==NULL;
}


int stackfull(Stack s){

    return 0;
}
```

3. Re-solve sheet 2 but for LinkedStackADT.

1. Write a function that returns the first element entered to a stack. (implementation level)= LINKEDSTACK.C

Solution (1)(with size field)

```
stackentry firstelment (Stack s){
Node *pn;
pn=ps.top;
for(int i=0;i<ps.Size-1;i++){

   pn=pn->next;
}
return pn->entry;

}
```

Solution (2)(without size field)

```
stackentry firstelment (Stack s){
 Node *pn;
 pn=s.top;
stackentry e;
while(pn){

     pn=pn->next ;

     if(pn)
         e=pn->entry;
}
return e;
 }
```

2. Write a function that returns a copy from the last element in a stack. (implementation level)= LINKEDSTACK.C

Solution

```
stackentry stacktop(Stack s){

    return s.top->entry;
}
```

3. Write a function to destroy a stack. (implementation level) = LINKEDSTACK.C

Solution

```
void destroystack(Stack *ps ){
    Node *pn ;
    pn=ps->top;
    while(pn){
    pn=pn->next;
    free(ps->top);
    ps->top=pn;
    }
    ps->Size=0;
    ps->top=NULL;
}
```

5. Write a function to return the size of a stack (implementation level)=LINKEDSTACK.C

Solution (1)(with size field)

```
int stacksize(Stack *ps){

    return ps->Size;
}
```

Solution (2)(without size field)

```
int stacksize(Stack s){
Node *pn;
int x=0;
for(pn=s.top;pn;pn=pn->next)
{
    x++;

    }
return x;}
```

4. Write a function to copy a stack to another. (implementation level)
=LINKEDSTACK.C

Solution

```
void copystack (Stack s1,Stack *s2){
Stack temp;
createstack(&temp);
Node *pn,*q;
q=s1.top;
while(q!=NULL){

pn=(Node*)malloc(sizeof(Node));

     pn->entry=q->entry;
     pn->next=temp.top;
     temp.top=pn;

q= q->next;
}
q=temp.top;
while(q!=NULL){

pn=(Node*)malloc(sizeof(Node));

     pn->entry=q->entry;
     pn->next=s2->top;
     s2->top=pn;

q= q->next;

}

}
```

- Traverse function = LINKEDSTACK.C

```
void traversestack(Stack *ps,void (*pf)(stackentry)){
    Node *pn;
    for (pn=ps->top;pn;pn=pn->next){
        (*pf)(pn->entry);

    }
}
```

باقى الشيت اللى هى الفانكشنز اليوزر ليفيل محلولة فى الشيت اللى قبله .....هى هى لأنى احنا بنغير فى الإمبلمنتاشن لكن طريقة استخدام الفانكشنز واحدة لو لاحظت ده ...لو انت فهمت ديه يبقى انت فاهم اهم كونسيبت فى الداتا استركشر وهو ال

Encapsulation

4. Implement QueueADT as a linked list.

LINKEDQUEUE.H

```
#ifndef LINKEDQUEUE_H_INCLUDED
#define LINKEDQUEUE_H_INCLUDED
typedef int queueentry;
typedef struct queuenode{
queueentry entry;
struct queuenode *next;

}Queuenode;
typedef struct {
Queuenode *Front;
Queuenode *Rear;
int Size;
}Queue;

void createqueue(Queue *pq);
void enqueue (Queue *pq,queueentry e);
void dequeue (Queue *pq,queueentry *e);
int queueempty(Queue *pq);

#endif // LINKEDQUEUE_H_INCLUDED
```

LINKEDQUEUE.C

```
#include"linkedqueue.h"
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
void createqueue(Queue *pq){

    pq->Front=NULL;
    pq->Rear=NULL;
    pq->Size=0;
}
void enqueue (Queue *pq,queueentry e){
    Queuenode  *pn = (Queuenode *)malloc(sizeof(Queuenode));
    pn->entry =e;
    pn->next=NULL;
     if(pq->Front==NULL){
    pq->Front=pn;
    }
    else{
    pq->Rear->next=pn;
    }
    pq->Rear=pn;
    pq->Size++;
}
void dequeue (Queue *pq,queueentry *e){
    Queuenode *pn;
    pn=pq->Front;
    *e=pn->entry;
    pq->Front=pn->next;
    free(pn);
    if(pq->Front==NULL){
    pq->Rear=NULL;
    }
    pq->Size--;

}
int queueempty(Queue *pq){
    return pq->Size==0;
}

int queuefull(Queue *pq){

    return 0;
}
```

5. Re-solve sheet 3 but for LinkedQUEUEADT.

2. Write a function that returns the last element in a queue. (implementation level)= LINKEDQUEUE.C

Solution (1)(with size field)

```
queueentry lastelement(Queue q){
Queuenode *p;
queueentry e;
p=q.Front;

for(int i=0;i<q.Size-1;i++){

   p=p->next;

}
return p->entry;
}
```

Solution (2)(without size field)

```
queueentry lastelement(Queue q){
Queuenode *p;
queueentry e;
p=q.Front;
while(p){
     p=p->next;
     if(p)
      e=p->entry;


}

return e;

}
```

3. Write a function that returns a copy from the first element in a queue. (implementation level))= LINKEDQUEUE.C

Solution

```
queueentry first_element(Queue q) {

return q.Front->entry;
}
```

4. Write a function to destroy a queue (implementation level))= LINKEDQUEUE.C

Solution

```
void destroyqueue(Queue *pq){

    for(pq->Rear=pq->Front->next ; pq->Front ;pq->Rear=pq->Rear->next)
    {
       free(pq->Front);
       pq->Front=pq->Rear;
    }
    pq->Size=0;

}
```

6. Write a function to return the size of a queue (implementation level) = LINKEDQUEUE.C

Solution

```
int queu_size(Queue q) {

return q.Size;
}
```

5. Write a function to copy a queue to another. (implementation level) =LINKEDQUEUE.C

Solution

```
void copy_queue(Queue q1,Queue *q2){
Queuenode *pn,*p;
p=q1.Front;

while(p){
pn = (Queuenode *)malloc(sizeof(Queuenode));

        pn->entry=p->entry;
        pn->next=NULL;
         if(q2->Front==NULL){
        q2->Front=pn;
        }
        else{
        q2->Rear->next=pn;
        }
        q2->Rear=pn;

       q2->Size++;

p=p->next;

}
```

- Traverse function = **LINKEDQUEUE.C**

```
void traversequeue(Queue *pq,void (*pf)(queueentry)){
    Queuenode *qn;
    for (qn=pq->Front;qn;qn=qn->next){
        (*pf)(qn->entry);

    }
}
```

باقى الشيت اللى هى الفانكشنز اليوزر ليفيل محلولة فى الشيت اللى قبله

LINKED LIST SHEET FROM MINDERS

Implementation of linked list.

LINKEDLIST.H

```
#ifndef LINKEDLIST_H_INCLUDED
#define LINKEDLIST_H_INCLUDED
typedef int listentry;
typedef struct listnode{
listentry entry;
struct listnode *next;
}Listnode;


typedef struct {
Listnode *head;
int Size;
}List;
void createlist(List *pl);
void insert_list(List *pl,listentry e,int pos);
void delete_list(List *pl,listentry *e,int pos);



#endif // LINKEDLIST_H_INCLUDED
```

LINKEDLIST.C

```
#include"linkedlist.h"  #include <stdio.h> #include <stdlib.h> #include <stddef.h>
void createlist(List *pl){
pl->head=NULL;
pl->Size=0;
}
int list_is_empty(List *pl){  return pl->Size==0;}

int  list_is_Full(List *pl){ return 0; }

void insert_list(List *pl,listentry e,int pos){
    Listnode *p,*q;int i;
p=(Listnode*)malloc(sizeof(Listnode));
p->entry=e;
p->next=NULL;
    if(pos==0){
      p->next=pl->head;
      pl->head=p;     }

    else   {
        for(q=pl->head, i=0;i<pos-1;i++){  q=q->next;    }

        p->next=q->next;
        q->next=p;
                    }

pl->Size++;

    }

void delete_list(List *pl,listentry *e,int pos){
Listnode *p,*q; int i;
    if(pos==0){
      *e=pl->head->entry;
      p=pl->head->next;
      free(pl->head);
      pl->head=p;

    }
    else{
    for(q=pl->head, i=0;i<pos-1;i++){     q=q->next;     }
    *e=q->next->entry;
    p=q->next->next;
    free(q->next);
    q->next=p;
                }
pl->Size--;
}
```

1. Write a function that retrieve element from the list. (implementation level) =LINKEDLIST.C

Solution

```
listentry retrievelist(List *pl,int pos){
Listnode *q;int i;
for(q=pl->head, i=0;i<pos-1;i++,q=q->next);

return q->next->entry;
}
```

2. Write a function to destroy a list (implementation level) =LINKEDLIST.C

Solution

```
void destroylist(List *pl){
Listnode *q;
for(q=pl->head->next;pl->head;q=q->next){

     free(pl->head);
     pl->head=q;

}
pl->Size=0;

}
```

3. Write a function to return the size of a list (implementation level) =LINKEDLIST.C

Solution

```
int listsize(List *pl){

return pl->Size;
}
```

4. Write a function to replace element from list (implementation level) =LINKEDLIST.C

Solution

```
void replacelist(List *pl,listentry *e,int pos){

Listnode *q;int i;
for(q=pl->head, i=0;i<pos-1;i++,q=q->next);
q->next->entry=e;

}
```

5. Write a function to copy a list to another. (implementation level) =LINKEDLIST.C

Solution 1

```
void copylist1(List l1,List *l2){
 l2->head=l1.head;
 l2->Size=l1.Size;
}
```

Solution 2

```
void copylist2(List *l1,List *l2){
Listnode *pn,*p,*q;
p=l1->head;

while(p){
     pn=(Listnode*)malloc(sizeof(Listnode));
     pn->entry= p->entry;
     pn->next=NULL;

     if(l2->head==NULL){
          l2->head=pn;
          q=l2->head;
          }

     else{
     q->next=pn;
     q=q->next;

 }
p=p->next;


l2->Size++;
}


}
```

- Traverse function = **LINKEDLIST.C**

```
void traverselist(List *pl,void (*pf)(listentry)){
     Listnode *q;
     for (q=pl->head;q;q=q->next){
          (*pf)(q->entry);


     }
}
```

6. Write the function void JoinList(List *pl1, List *pl2) that copies all entries from pl1 onto the end of pl2(implementation level) =LINKEDLIST.C

Solution

```
void JoinList(List*l,List*l2){

Listnode *p;
p=l->head;

while(p->next){
p=p->next;
}
p->next=l2->head;

}
```

7. Think of the list ADT modified using the following strategy. Whenever an element is located using the isPresent() operation, that particular element is deleted from the current position and reinserted at the beginning of the list. The motivation behind this relocation is that in many situations an element accessed in a list is expected with high probability to be accessed several times in the future. So keeping the element near the beginning of the list reduces average search time. Modify the list ADT implementations to incorporate this modification.

Solution

```
int  ispresent(List *pl,listentry e){

Listnode *p,*q;
q=pl->head;
int i=0,check=0;
for (i;i<pl->Size;i++){
     if(q->entry==e){
                              check=1;
                              break;
                         }

     q=q->next;
}

if(check!=0){
     q=pl->head;
     for(int z=0;z<i-1;z++){   q=q->next;   }
     p=q;
     q=q->next;
     p->next=q->next;
     q->next=pl->head;
     pl->head=q;

     return 1;
}
else{
    return 0;
}

}
```

8. Write a function to destroy a list (user level) =MAIN.C

Solution

```
void destroy_list (List *l ){
     listentry e;
while(!list_is_empty(l))
     {    delete_list(l,&e,0);  }

}
```

9. Write a function to copy a list to another. (user level) =MAIN.C

Solution

```
void cpy_list(List l,List *l2){
listentry e ;
int i=0;
while(!list_is_empty(&l)){
    delete_list(&l,&e,0);
    insert_list(l2,e,i);
i++;
}
}
```

10. Write a function to return the size of a list (user level) =MAIN.C

Solution

```
int  size_list (List l ){
     listentry e;
     int x=0;
while(!list_is_empty(&l))
     {
          delete_list(&l,&e,0);
          x++;

     }
return x;
}
```

11. Write the function void JoinList(List *pl1, List *pl2) that copies all entries from pl1 onto the end of pl2. (user level) =MAIN.C

Solution

```
void join_list (List *l,List *l2){
List l3;
createlist(&l3);
listentry e;
int z=0;

while(!list_is_empty(l)){
delete_list(l,&e,0);
insert_list(&l3,e,z);
z++;
}

while(!list_is_empty(l2)){
delete_list(l2,&e,0);
insert_list(&l3,e,z);
z++;
}


for(int i=0;i<z;i++){
delete_list(&l3,&e,0);
insert_list(l,e,i);

}
}
```

QUEZ 3 QUESTIONS

1. Write a function that reverse the list (implementation level) =LINKEDLIST.C

Solution(1)

```
void reverselist(List *pl){
Listnode *p,*q,*z;
z=pl->head;

int i,s=pl->Size-1;

while(s>=0){
for(q=z,i=0;i<s-1;i++){
     q=q->next;
}
 p=q;
q=q->next;

     if(i==pl->Size-2)
          pl->head=q;

   q->next=p;

s--;
}
z->next=NULL;


}
```

Solution(2)

```
void revirse(List *l){
Listnode *curr=l->head,*next=NULL ,*prev=NULL;
while(curr){
    next=curr->next;
    curr->next=prev;
    prev=curr;
    curr=next;
}

l->head=prev;


}
```

2.Write a function to return the sum of the elements of a list(implementation level) =LINKEDLIST.C

IF THE TYPE Definition is :

```
typedef int listentry;
typedef struct listnode{
listentry entry;
struct listnode *next;
}Listnode;
typedef Listnode *List;
```

Solution

```
int sum_element(List *l){
Listnode *p;
p=*l;
int sum=0;
while(p)
{
sum+=p->entry;
p=p->next;
}

return sum;
}
```

3.Balance point function (is a point when the sum of numbers before it are equal the sum of numbers after it ,,we called it balance point) =**LINKEDLIST.C**

Solution

```
int balancepoint  (List *l){
Listnode *p,*q;
int sum1,sum2=0;
p=l->head;

for(int i=0;i<l->Size;i++){
        q=l->head;

      while(q!=p){
            sum1+=q->entry;
          q=q->next;

      }

      q=p->next;

      while(q){

            sum2+=q->entry;
            q=q->next;

      }

      if(sum1==sum2)
            return p->entry;

      sum1=0;
      sum2=0;
      p=p->next;
}

return 0;
}
```

4.write a function that merge 2 list into another list =LINKEDLIST.C

Solution

```
void merge_list (List l1,List l2,List *l3){
Listnode *z,*q,*p;

p=l1.head;
q=l2.head;
l3->head=l1.head;
z=l3->head;

while(z){
     l1.head=p->next;

    l2.head=q->next;

     p->next=q;

     z=z->next;

     p=l1.head;

     z->next=p;

     z=z->next;

     q=l2.head;

     l3->Size+=2;

}

}
```

5.

Implement the List ADT using array (not linked)

LISTARRAY.H

```
#ifndef LISTARRAY_H_INCLUDED
#define LISTARRAY_H_INCLUDED
#define maxlist 10
typedef int l_entry;
typedef struct {

l_entry entry[maxlist];
int Size;

}listtype;

void create_list(listtype *l);
int  listempty(listtype l);
int listfull(listtype l);
void insertlist(int pos ,l_entry e,listtype *l);
void deletelist(int pos,l_entry *pe,listtype *l);



#endif // LISTARRAY_H_INCLUDED
```

LISTARRAY.C

```
#include"listarray.h"

void create_list(listtype *l){

l->Size=0;

}
int  listempty(listtype l){  return l.Size==0; }

int listfull(listtype l){ return 0; }

void insertlist(int pos ,l_entry e,listtype *l){

for (int i=l->Size-1;i>=pos;i--){
    l->entry[i+1]=l->entry[i];

}
l->entry[pos]=e;
l->Size++;


}

void deletelist(int pos,l_entry *pe,listtype *l){

*pe=l->entry[pos];

for(int i=pos+1;i<=l->Size-1;i++){

    l->entry[i-1]=l->entry[i];
}
l->Size--;

}
```

1. Write a function that retrieve element from the list. (implementation level) = LISTARRAY.C

Solution

```
l_entry retrieve_list(int pos,listtype l){

return l.entry[pos];
}
```

2. Write a function to destroy a list (implementation level) = LISTARRAY.C

Solution

```
void destroy_list(listtype *l){

l->Size=0;
}
```

3. Write a function to return the size of a list (implementation level) = LISTARRAY.C

Solution

```
int list_size(listtype l){
return l.Size;
}
```

4. Write a function to replace element from list (implementation level) = LISTARRAY.C

Solution

```
void replace_list(int pos,l_entry e,listtype *l){

l->entry[pos]=e;

}
```

5. Write a function to copy a list to another. (implementation level) = LISTARRAY.C

Solution

```
void cpy_list(listtype l1,listtype *l2){

for( int i=0;i<l1.Size;i++){
     l2->entry[i]=l1.entry[i];
   l2->Size++;

}

}
```

- Traverse function = **LISTARRAY.C**

```
void traverse_list(listtype *l,void (*pf)(l_entry)){
for (int i=0;i<l->Size;i++){
     (*pf)(l->entry[i]);

}

}
```

DO NOT LET YOUR DREAMS BE DREAMS

MINDERS