List without type definition

#include <stdio.h>

#include<stdlib.h>

#include<string.h>

struct list1

{

char str[30];

struct list1 \*link;

};

struct list1 \*start=NULL;

void insert(char \*s)

{

struct list1 \*newnode,\*temp;

newnode=(struct list1 \*)malloc(1\*sizeof(struct list1 ));

printf("insede insert ");

if(start==NULL)

{

strcpy(newnode->str,s);

newnode->link=NULL;

start=newnode;

printf("%s ",start->str);

}

else

{

for(temp=start;temp->link!=NULL;temp=temp->link)

{

}

temp->link=newnode;

strcpy(newnode->str,s);

newnode->link=NULL;

}

}

void delete(char \*s)

{

struct list1 \*temp,\*prev=NULL;

if(start==NULL)

printf("cant delete");

else

{

for(temp=start; temp!=NULL&&strcmp(temp->str,s)!=0;temp=temp->link)

{

prev=temp;

}

if(prev==NULL)

{

if(temp->link==NULL)

{

start=NULL;

free(temp);

}

else

{

start=temp->link;

free(temp);

}

}

else if(temp==NULL) {

printf("string not found\n");

}

else{

prev->link=temp->link;

free(temp);

}

}

}

void traverse()

{

struct list1 \*temp;

for(temp=start;temp!=NULL;temp=temp->link)

printf("%s ",temp->str);

}

int main()

{

printf("Hello World");

char temp[30];

int choice;

do

{

printf("1. Insert node 2. Delet Node 3. Traverse 4. Exit\n");

scanf("%d",&choice);

switch(choice)

{

case 1: printf("enter to the string to be entered\n");

scanf("%s",temp);

insert(temp);

break;

case 2: printf("enter the string too be deleted \n");

scanf("%s",temp);

delete(temp);

break;

case 3: traverse();

break;

}

}while(choice!=4);

return 0;

}

//Fully working list code

#include <stdio.h>

#include<stdlib.h>

typedef struct list \* ptr;

struct list

{

int data;

ptr link;

};

typedef struct list list1;

void insertfront(ptr \*start,ptr \*last,int d)

{

ptr newnode,temp;

newnode=(ptr)malloc(1\*sizeof(list1));

if(\*start==NULL && \*last==NULL)

{

newnode->data=d;

newnode->link=NULL;

\*start=newnode;

\*last=newnode;

}

else

{

newnode->data=d;

newnode->link=\*start;

\*start=newnode;

}

}

void insertlast(ptr \*start,ptr \*last,int d)

{

ptr newnode,temp;

newnode=(ptr)malloc(1\*sizeof(list1));

if(\*start==NULL && \*last==NULL)

{

newnode->data=d;

newnode->link=NULL;

\*start=newnode;

\*last=newnode;

}

else

{

newnode->data=d;

newnode->link=NULL;

(\*last)->link=newnode;

\*last=newnode;

}

}

void insertspecific(ptr \*start,ptr \*last,int d)

{

ptr newnode,temp,prev=NULL;

newnode=(ptr)malloc(1\*sizeof(list1));

for(temp=\*start;temp!=NULL &&temp->data<d;temp=temp->link)

{

prev=temp;

}

if(prev==NULL)

{

newnode->data=d;

newnode->link=\*start;

\*start=newnode;

}

else if(temp==NULL)

{

newnode->data=d;

newnode->link=NULL;

\*last=newnode;

}

else

{

newnode->data=d;

newnode->link=temp;

prev->link=newnode;

}

}

void deletefront(ptr \*start,ptr \*last)

{

ptr temp;

if(\*start==NULL &&\*last==NULL)

{

printf("sorry cant delete\n");

}

else if(\*start==\*last)

{

temp=\*start;

\*start=NULL;

\*last=NULL;

free(temp);

}

else

{

temp=\*start;

\*start=(\*start)->link;

free(temp);

}

}

void deleterear(ptr \*start,ptr \*last)

{

ptr temp,prev;

if(\*start==NULL &&\*last==NULL)

{

printf("sorry cant delete\n");

}

else if(\*start==\*last)

{

temp=\*start;

\*start=NULL;

\*last=NULL;

free(temp);

}

else

{

for(temp=\*start;temp!=\*last;temp=temp->link)

{

prev=temp;

}

prev->link=NULL;

\*last=prev;

free(temp);

}

}

void deletespecific(ptr \*start,ptr \*last,int del)

{

ptr newnode,temp,prev=NULL;

if(\*start==NULL && \*last==NULL)

{

printf("list is empty\n");

}

else if(\*start==\*last && del==(\*start)->data)

{

temp=\*start;

\*start=NULL;

\*last=NULL;

free(temp);

}

else

{

for(temp=\*start;temp!=NULL &&temp->data!=del;temp=temp->link)

{

prev=temp;

}

if(prev==NULL)

{

\*start=(\*start)->link;

free(temp);

}

else if(temp==NULL)

{

printf("data to be deleted not found\n");

}

else

{

prev->link=temp->link;

free(temp);

}

}

}

void display(ptr s,ptr l)

{

ptr temp;

for (temp=s;temp!=NULL;temp=temp->link)

printf("%d ",temp->data);

}

int main()

{

ptr start=NULL;

ptr last=NULL;

int ch,d,del;

do{

printf("1.if 2.ir 3.is 4.df 5.dr 6. ds 7. display 8. exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("enter the element to be added into the list\n");

scanf("%d",&d);

insertfront(&start,&last,d);

break;

case 2: printf("enter the element to be added into the list\n");

scanf("%d",&d);

insertlast(&start,&last,d);

break;

case 3: printf("enter the element to be added into the list\n");

scanf("%d",&d);

insertspecific(&start,&last,d);

break;

case 4: deletefront(&start,&last);

break;

case 5: deleterear(&start,&last);

break;

case 6:printf("enter the data to be deleted from the list\n");

scanf("%d",&del);

deletespecific(&start,&last,del);

break;

case 7: display(start,last);

break;

}

}while(ch!=8);

printf("Hello World");

return 0;

}

// addition of polynomials

#include <stdio.h>

#include<stdlib.h>

typedef struct polynode \*polyptr;

struct polynode

{

int coef;

int expo;

polyptr link;

};

typedef struct polynode pnode;

int compare(int pe,int qe)

{

if(pe>qe)

return 1;

else if(pe==qe)

return 0;

else

return -1;

}

void attach(int c,int e,polyptr \*ptr)

{

polyptr newnode;

newnode=(polyptr)malloc(1\*sizeof(pnode));

newnode->coef=c;

newnode->expo=e;

newnode->link=NULL;

(\*ptr)->link=newnode;

\*ptr=newnode;

}

polyptr padd(polyptr p,polyptr q)

{

polyptr front,rear;

rear=(polyptr)malloc(1\*sizeof(pnode));

front=rear;

int sum;

while(p!=NULL && q!=NULL)

{

switch(compare(p->expo,q->expo))

{

case -1: attach(q->coef,q->expo,&rear);

q=q->link; break;

case 1: attach(p->coef,p->expo,&rear);

p=p->link; break;

case 0: sum=p->coef+q->coef;

if(sum)

attach(sum,q->expo,&rear);

q=q->link; p=p->link;break;

}

}

while(p)

{

attach(p->coef,p->expo,&rear);

p=p->link;

}

while(q)

{

attach(q->coef,q->expo,&rear);

q=q->link;

}

polyptr temp=front;

front=front->link;

free(temp);

return front;

}

void createpoly(polyptr \*p,int c,int e)

{

polyptr newnode,temp;

newnode=(polyptr)malloc(1\*sizeof(pnode));

newnode->coef=c;

newnode->expo=e;

if(\*p==NULL)

{

newnode->link=NULL;

\*p=newnode;

}

else

{

for(temp=\*p;temp->link!=NULL;temp=temp->link)

{

}

temp->link=newnode;

newnode->link=NULL;

}

}

void display(polyptr ptr)

{

polyptr tr;

for(tr=ptr;tr!=NULL;tr=tr->link)

{

if(tr->coef>0)

printf("+%d ",tr->coef);

else

printf("%d",tr->coef);

if(tr->expo>=0)

printf("x+%d ",tr->expo);

else

printf("x%d",tr->expo);

}

}

void erase(polyptr \*ptr)

{

polyptr temp;

while(\*ptr)

{

temp=\*ptr;

(\*ptr)=(\*ptr)->link;

free(temp);

}

}

int main()

{

polyptr p,q,r;

p=NULL;

q=NULL;

printf("Hello World");

int ch,c,e;

do{

printf("press 1. enter a 2. to enter b");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("enter coefficient \n");

scanf("%d",&c);

printf("enter exponent\n ");

scanf("%d",&e);

createpoly(&p,c,e);

break;

case 2: printf("enter coefficient \n");

scanf("%d",&c);

printf("enter exponent\n ");

scanf("%d",&e);

createpoly(&q,c,e); break;

}

}while(ch!=3);

display(p);

printf("\n");

display(q);

printf("\n");

r=padd(p,q);

printf("\n");

display(r);

erase(&p);

erase(&q);

erase(&r);

if(!p&&!q&&!r)

printf("successful release of memory");

else

printf("not released memory");

return 0;

}

Hello Worldpress 1. enter a 2. to enter b1

enter coefficient

2

enter exponent

2

press 1. enter a 2. to enter b1

enter coefficient

3

enter exponent

3

press 1. enter a 2. to enter b1

enter coefficient

4

enter exponent

4

press 1. enter a 2. to enter b2

enter coefficient

2

enter exponent

2

press 1. enter a 2. to enter b2

enter coefficient

3

enter exponent

3

press 1. enter a 2. to enter b2

enter coefficient

4

enter exponent

4

press 1. enter a 2. to enter b3

+2 x+2 +3 x+3 +4 x+4

+2 x+2 +3 x+3 +4 x+4

+4 x+2 +6 x+3 +8 x+4 successful release of memory

// laddition of poly using circular list without a header node

/ addition of polynomials

#include <stdio.h>

#include<stdlib.h>

typedef struct polynode \*polyptr;

struct polynode

{

int coef;

int expo;

polyptr link;

};

polyptr avail=NULL;

typedef struct polynode pnode;

polyptr getnode(void)

{

polyptr newnode;

if(avail)

{

newnode=avail;

avail=avail->link;

}

else

newnode=(polyptr)malloc(1\*sizeof(pnode));

return newnode;

}

void returnnode(polyptr ptr)

{

ptr->link=avail;

avail=ptr;

}

void cerase(polyptr \*ptr)

{

polyptr temp;

if(\*ptr)

{

temp=(\*ptr)->link;

(\*ptr)->link=avail;

avail=temp;

\*ptr=NULL;

}

}

int compare(int pe,int qe)

{

if(pe>qe)

return 1;

else if(pe==qe)

return 0;

else

return -1;

}

void attach(int c,int e,polyptr \*ptr)

{

polyptr newnode;

newnode=getnode();

newnode->coef=c;

newnode->expo=e;

newnode->link=NULL;

(\*ptr)->link=newnode;

\*ptr=newnode;

}

polyptr padd(polyptr p,polyptr q)

{

polyptr front,rear;

char s;

polyptr ps=p;

polyptr qs=q;

rear=getnode();

front=rear;

int sum;

while(1)

{

switch(compare(p->expo,q->expo))

{

case -1: attach(q->coef,q->expo,&rear);

q=q->link; break;

case 1: attach(p->coef,p->expo,&rear);

p=p->link; break;

case 0: sum=p->coef+q->coef;

if(sum)

attach(sum,q->expo,&rear);

q=q->link; p=p->link;break;

}

if(p->link==ps)

{

switch(compare(p->expo,q->expo))

{

case -1: attach(q->coef,q->expo,&rear);

q=q->link; break;

case 1: attach(p->coef,p->expo,&rear);

p=p->link; break;

case 0: sum=p->coef+q->coef;

if(sum)

attach(sum,q->expo,&rear);

q=q->link; p=p->link;break;

}

break;

}

if(q->link==qs)

{

switch(compare(p->expo,q->expo))

{

case -1: attach(q->coef,q->expo,&rear);

q=q->link; break;

case 1: attach(p->coef,p->expo,&rear);

p=p->link; break;

case 0: sum=p->coef+q->coef;

if(sum)

attach(sum,q->expo,&rear);

q=q->link; p=p->link;break;

}

break;

}

}

while(p!=ps)

{

attach(p->coef,p->expo,&rear);

p=p->link;

}

while(q!=qs)

{

attach(q->coef,q->expo,&rear);

q=q->link;

}

polyptr temp=front;

front=front->link;

rear->link=front;

returnnode(temp);

return front;

}

void createpoly(polyptr \*s,int c,int e)

{

polyptr newnode,temp;

newnode=getnode();

newnode->coef=c;

newnode->expo=e;

if(\*s==NULL)

{

newnode->link=newnode;

\*s=newnode;

//\*r=newnode;

}

else

{

for(temp=\*s;temp->link!=\*s;temp=temp->link)

{

}

temp->link=newnode;

newnode->link=\*s;

}

}

void display(polyptr ptr)

{

polyptr tr=ptr;

while(tr)

{

if(tr->coef>0)

printf("+%d ",tr->coef);

else

printf("%d",tr->coef);

if(tr->expo>=0)

printf("x+%d ",tr->expo);

else

printf("x%d",tr->expo);

tr=tr->link;

if(tr==ptr)

break;

}

}

void erase(polyptr \*ptr)

{

polyptr temp;

while(\*ptr)

{

temp=\*ptr;

(\*ptr)=(\*ptr)->link;

free(temp);

}

}

int main()

{

polyptr ps,qs,pr,qr,r;

ps=NULL;

qs=NULL;

//pr=NULL;

//qr=NULL;

printf("Hello World");

int ch,c,e;

do{

printf("press 1. enter a 2. to enter b");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("enter coefficient \n");

scanf("%d",&c);

printf("enter exponent\n ");

scanf("%d",&e);

createpoly(&ps,c,e);

break;

case 2: printf("enter coefficient \n");

scanf("%d",&c);

printf("enter exponent\n ");

scanf("%d",&e);

createpoly(&qs,c,e); break;

}

}while(ch!=3);

display(ps);

printf("\n");

display(qs);

printf("\n");

r=padd(ps,qs);

printf("\n");

display(r);

cerase(&ps);

cerase(&qs);

cerase(&r);

if(!ps&&!qs&&!r)

printf("successful release of memory");

else

printf("not released memory");

return 0;

}

Hello Worldpress 1. enter a 2. to enter b1

enter coefficient

5

enter exponent

4

press 1. enter a 2. to enter b1

enter coefficient

4

enter exponent

3

press 1. enter a 2. to enter b1

enter coefficient

2

enter exponent

2

press 1. enter a 2. to enter b2

enter coefficient

4

enter exponent

4

press 1. enter a 2. to enter b2

enter coefficient

2

enter exponent

3

press 1. enter a 2. to enter b2

enter coefficient

2

enter exponent

1

press 1. enter a 2. to enter b3

+5 x+4 +4 x+3 +2 x+2

+4 x+4 +2 x+3 +2 x+1

+9 x+4 +6 x+3 +2 x+2 +2 x+1 successful release of memory

//polynomials using circular list with a header node

#include<stdio.h>

#include<stdlib.h>

typedef struct polynode \*poly\_pointer;

struct polynode {

int coef;

int expon;

poly\_pointer link;

};

#define IS\_FULL(poly\_pointer) (!poly\_pointer)

poly\_pointer avail=NULL;

int compare(int ac,int bc)

{

if(ac<bc)

return -1;

else if(ac>bc)

return 1;

else

return 0;

}

void attach(int coef,int expon,poly\_pointer \*rear)

{

poly\_pointer temp;

temp=calloc(1,sizeof(struct polynode));

if(IS\_FULL(temp))

{

printf("cant allocate memory\n");

exit(1);

}

temp->coef=coef;

temp->expon=expon;

(\*rear)->link=temp;

\*rear=temp;

}

poly\_pointer getNode()

{

poly\_pointer node;

if(avail)

{

node=avail;

avail=avail->link;

printf("allocating node from avail list\n");

}

else

{

node=calloc(1,sizeof(struct polynode));

printf("allocating node using calloc\n");

}

return node;

}

poly\_pointer padd(poly\_pointer a,poly\_pointer b)

{

poly\_pointer rear, front,temp,starta=a,startb=b;

rear=getNode();

int sum;

a=a->link;

b=b->link;

if(IS\_FULL(rear))

{

printf("menory full cant allocate memory\n");

exit(0);

}

else

{

front=rear;

while(a!=starta && b!=startb)

{

switch(compare(a->expon,b->expon))

{

case -1:

attach(b->coef,b->expon,&rear);

b=b->link; break;

case 1: attach(a->coef,a->expon,&rear);

a=a->link; break;

case 0: sum=a->coef+b->coef;

attach(sum,a->expon,&rear);

a=a->link;

b=b->link;

break;

}

}

while(a!=starta)

{

attach(a->coef,a->expon,&rear);

a=a->link;

}

while(b!=startb){

attach(b->coef,b->expon,&rear);

b=b->link;

}

front->coef=-999;

front->expon=-999;

rear->link=front;

return front;

}

}

void cerase(poly\_pointer \*ptr)

{

printf("erasing poly\n");

poly\_pointer temp;

if(\*ptr)

{

avail=\*ptr;

\*ptr=NULL;

}

printf(" avail is %d\n",avail->coef);

}

void create(poly\_pointer \*aa,int c, int e)

{

poly\_pointer newnode,tr;

newnode=calloc(1,sizeof(struct polynode));

if(IS\_FULL(newnode))

{

printf("menory full cant allocate memory\n");

exit(0);

}

/\* if(\*aa==NULL)

{

newnode->coef=-999;

newnode->expon=-999;

newnode->link=\*aa;

\*aa=newnode;

}\*/

//else{

for(tr=\*aa;tr->link!=\*aa;tr=tr->link)

{

}

newnode->coef=c;

newnode->expon=e;

newnode->link=\*aa;

tr->link=newnode;

//}

}

void display(poly\_pointer ptr)

{

poly\_pointer tr=ptr->link;

while(tr)

{

if(tr->coef>=0)

printf("+%d ",tr->coef);

else

printf("%d",tr->coef);

if(tr->expon>=0)

printf("x+%d ",tr->expon);

else

printf("x%d",tr->expon);

tr=tr->link;

if(tr==ptr)

break;

}

}

int main()

{

poly\_pointer a,b,cc,d;

a=NULL;

b=NULL;

cc=NULL;

a=calloc(1,sizeof(struct polynode));

a->coef=-999;

a->expon=-999;

a->link=a;

b=calloc(1,sizeof(struct polynode));

b->coef=-999;

b->expon=-999;

b->link=b;

cc=calloc(1,sizeof(struct polynode));

cc->coef=-999;

cc->expon=-999;

cc->link=cc;

printf("Hello World");

int ch,c,e;

do{

printf("press 1. enter a 2. to enter b");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("enter coefficient \n");

scanf("%d",&c);

printf("enter exponent\n ");

scanf("%d",&e);

create (&a,c,e);

break;

case 2: printf("enter coefficient \n");

scanf("%d",&c);

printf("enter exponent\n ");

scanf("%d",&e);

create (&b,c,e); break;

case 3: printf("enter coefficient \n");

scanf("%d",&c);

printf("enter exponent\n ");

scanf("%d",&e);

create (&cc,c,e); break;

}

}while(ch!=4);

display(a);

printf("\n");

display(b);

printf("\n");

display(cc);

printf("\n");

cerase(&cc);

d=padd(a,b);

printf("\n");

display(d);

return 0;

}

Hello Worldpress 1. enter a 2. to enter b1

enter coefficient

4

enter exponent

4

press 1. enter a 2. to enter b1

enter coefficient

3

enter exponent

3

press 1. enter a 2. to enter b1

enter coefficient

2

enter exponent

2

press 1. enter a 2. to enter b1

enter coefficient

1

enter exponent

1

press 1. enter a 2. to enter b2

enter coefficient

5

enter exponent

5

press 1. enter a 2. to enter b2

enter coefficient

4

enter exponent

4

press 1. enter a 2. to enter b2

enter coefficient

2

enter exponent

2

press 1. enter a 2. to enter b3

enter coefficient

5

enter exponent

5

press 1. enter a 2. to enter b3

enter coefficient

4

enter exponent

4

press 1. enter a 2. to enter b3

enter coefficient

2

enter exponent

2

press 1. enter a 2. to enter b4

+4 x+4 +3 x+3 +2 x+2 +1 x+1

+5 x+5 +4 x+4 +2 x+2

+5 x+5 +4 x+4 +2 x+2

erasing poly

avail is -999

allocating node from avail list

+5 x+5 +8 x+4 +3 x+3 +4 x+2 +1 x+1t

//linked stacks (working full)

#include <stdio.h>

#include<stdlib.h>

#define MAX\_STACKS 3 /\*maximum number of stacks \*/

typedef struct {

int key;

} element;

typedef struct stack1 \*stack\_pointer;

struct stack1 {

element item;

stack\_pointer link;

};

typedef struct stack1 stack;

//add an item

#define IS\_EMPTY(stack\_pointer) (!(stack\_pointer))

#define IS\_FULL(stack\_pointer) (!(stack\_pointer))

void add(stack\_pointer \* top, element item)

{

stack\_pointer temp = (stack\_pointer) malloc(sizeof (stack));

if (IS\_FULL(temp)) {

fprintf(stderr,"The memory is full\n");

exit(1);

}

temp->item = item;

temp->link =\*top;

\*top=temp;

}

element delete(stack\_pointer \*top)

{

stack\_pointer temp=\*top;

element item;

if (IS\_EMPTY(temp)) {

fprintf(stderr,"The stack is empty\n");

item.key=-999;

}

else{

item=temp->item;

\*top=temp->link;

free(temp);

}

return item;

}

int main()

{

printf("Hello World");

int ch,i,k;

element item;

stack\_pointer temp;

stack\_pointer top[MAX\_STACKS];

do

{

printf("1.Push on to the ith stack 2. Pop from ith stack 3. Display the stacks\n");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("enter the stack number on to which you have to push the item");

scanf("%d",&i);

if(i>=0 && i<=2)

{

printf("enter the key value to be pushed on to the stack\n");

scanf("%d",&k);

item.key=k;

add(&top[i],item);

}

break;

case 2: printf("enter the stack number from which you have to delete\n");

scanf("%d",&i);

if(i>=0 && i<=2)

{

item=delete(&top[i]);

if(item.key!=-999)

printf("the deleted value of the key is %d\n",item.key);

else

printf("the stack number you were attempting to delete was empty\n");

}

break;

case 3: printf("the stacks are \n");

for(i=0;i<3;i++)

{

printf("printing %d stack",i+1);

for(temp=top[i];temp!=NULL;temp=temp->link)

printf("%d ",temp->item.key);

printf("\n");

}

break;

}

}while(ch!=4);

return 0;

}

//linked Queues

#include <stdio.h>

#include<stdlib.h>

#define MAXQ 3

typedef struct {

int key;

} element;

typedef struct lq1 \*qptr;

struct lq1 {

element item;

qptr link;

};

typedef struct lq1 lq;

//add an item

#define IS\_EMPTY(qptr) (!(qptr))

#define IS\_FULL(qptr) (!(qptr))

qptr front[MAXQ],rear[MAXQ];

void addq(qptr \*rear,qptr \*front,element item)

{

qptr temp=(qptr)malloc(1\*sizeof(lq));

temp->item=item;

if(IS\_FULL(temp))

{

printf("memory is full\n");

}

else

{

if(\*rear!=NULL)

{

temp->link=NULL;

(\*rear)->link=temp;

\*rear=temp;

}

else

{

temp->link=NULL;

\*rear=temp;

\*front=temp;

}

}

}

element deleteq(qptr \*rear,qptr \*front)

{

qptr temp=\*front;

//temp->item=item;

element e;

if(IS\_EMPTY(temp))

{

printf("Q empty\n");

e.key=-999;

}

else

{

if(\*rear==\*front)

{

\*rear=NULL;

\*front=NULL;

e=temp->item;

free(temp);

}

else

{

\*front=temp->link;

e=temp->item;

free(temp);

}

}

return e;

}

int main()

{

printf("Hello World");

int ch,i,k;

element item;

qptr temp;

qptr front[MAXQ],rear[MAXQ];

do

{

printf("1.add an on to the ith Q 2. Pop from ith Q 3. Display the Qs\n");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("enter the Q number on to which you have to push the item");

scanf("%d",&i);

if(i>=0 && i<=2)

{

printf("enter the key value to be add on to the Q\n");

scanf("%d",&k);

item.key=k;

addq(&rear[i],&front[i],item);

}

break;

case 2: printf("enter the Q number from which you have to delete\n");

scanf("%d",&i);

if(i>=0 && i<=2)

{

item=deleteq(&rear[i],&front[i]);

if(item.key!=-999)

printf("the deleted value of the key is %d\n",item.key);

else

printf("the q number you were attempting to delete was empty\n");

}

break;

case 3: printf("the Qs are \n");

for(i=0;i<3;i++)

{

printf("printing %d Q",i+1);

for(temp=front[i];temp!=NULL;temp=temp->link)

printf("%d ",temp->item.key);

printf("\n");

}

break;

}

}while(ch!=4);

return 0;

}

//linked Qs with availability lists

#include <stdio.h>

#include<stdlib.h>

#define MAXQ 3

typedef struct {

int key;

} element;

typedef struct lq1 \*qptr;

struct lq1 {

element item;

qptr link;

};

typedef struct lq1 lq;

//add an item

#define IS\_EMPTY(qptr) (!(qptr))

#define IS\_FULL(qptr) (!(qptr))

qptr avail=NULL;

qptr getNode()

{

qptr node;

if(avail)

{

node=avail;

avail=avail->link;

printf("allocating node from avail list\n");

}

else

{

node=calloc(1,sizeof(lq));

printf("allocating node using calloc\n");

}

return node;

}

void retnode(qptr node)

{

qptr temp;

node->link=avail;

avail=node;

printf("in retnode displaying return node\n");

for(temp=avail;temp!=NULL;temp=temp->link)

printf("%d ",temp->item.key);

}

void addq(qptr \*rear,qptr \*front,element item)

{

qptr temp=(qptr)malloc(1\*sizeof(lq));

temp->item=item;

if(IS\_FULL(temp))

{

printf("memory is full\n");

}

else

{

if(\*rear!=NULL)

{

temp->link=NULL;

(\*rear)->link=temp;

\*rear=temp;

}

else

{

temp->link=NULL;

\*rear=temp;

\*front=temp;

}

}

}

element deleteq(qptr \*rear,qptr \*front)

{

qptr temp=\*front;

//temp->item=item;

element e;

if(IS\_EMPTY(temp))

{

printf("Q empty\n");

e.key=-999;

}

else

{

if(\*rear==\*front)

{

\*rear=NULL;

\*front=NULL;

e=temp->item;

retnode(temp);

}

else

{

\*front=temp->link;

e=temp->item;

retnode(temp);

}

}

return e;

}

int main()

{

printf("Hello World");

int ch,i,k;

element item;

qptr temp;

qptr front[MAXQ],rear[MAXQ];

for(i=0;i<3;i++)

{

rear[i]=NULL;

front[i]=NULL;

}

do

{

printf("1.add an on to the ith Q 2. Pop from ith Q 3. Display the Qs\n");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("enter the Q number on to which you have to push the item");

scanf("%d",&i);

if(i>=0 && i<=2)

{

printf("enter the key value to be add on to the Q\n");

scanf("%d",&k);

item.key=k;

addq(&rear[i],&front[i],item);

}

break;

case 2: printf("enter the Q number from which you have to delete\n");

scanf("%d",&i);

if(i>=0 && i<=2)

{

item=deleteq(&rear[i],&front[i]);

if(item.key!=-999)

printf("the deleted value of the key is %d\n",item.key);

else

printf("the stack number you were attempting to delete was empty\n");

}

break;

case 3: printf("the Qs are \n");

for(i=0;i<3;i++)

{

printf("printing %d Q",i+1);

for(temp=front[i];temp!=NULL;temp=temp->link)

printf("%d ",temp->item.key);

printf("\n");

}

break;

}

}while(ch!=4);

return 0;

}