מכללת אורט בראודה
מת"מ
מבוא לתכנות מערכות
61745

שאלות ממבחנים הקודמים

שאלה 1

השאלה עוסקת במימוש רשימה מקושרת ב-ADT .

מבנה של צומת ברשימה מקושרת נראה בצורה הבאה :

```
typedef struct ListNode
{
  void* data;
  struct ListNode *next;
} ListNode,  *PNode;
```

יש לממש את שלושת הפונקציות הכלליות לפי התאור הבא :

פונקציה **insertEntry** המכניסה צומת לסוף הרשימה,בתנאי שהנתון הזה לא נמצא כבר ברשימה. מחזירה TRUE אם ההכנסה בוצעה, אחרת – FALSE (הנתון קיים כבר ברשימה). compare_func מצביע לפונקציה ספציפית שמשווה שני נתונים שמחזירה 1 אם הם שווים ,אחרת – 0.

פונקציה **free_All** המשחררת את הזכרון של כל הצמתים ברשימה. free_func מצביע לפונקציה ספציפית לשחרור הזיכרון של המידע בצומת.

פונקציה **print_All** המציגה את המידע שנמצא בכל הצמתים ברשימה המקושרת. print_func מצביע לפונקציה ספציפית להצגת המידע בצומת כלשהי.

פונקציות ספציפיות עבור טיפוס נתונים char הן:

```
int cmp_char(void* a,void* b)
{
	if(*(char*)a==*(char*)b)
		return 1;
	return 0;
}
void prnt_char(void* a)
{
	printf(" %c ",*(char*)a);
}
void free_char(void* a)
{
	free(a);
}
```

תזכורת כללית לגבי פונקציות ספציפיות והסבר לגבי סיבה לשימוש בפונקציה ספציפית לשיחרור:

פונקציה כללית לא יכולה לטפל בנתונים עצמם כיוון שהסוג שלהם יכול להשתנות (מספרים, אותיות וכוי). בגלל זה היא נעזרת בפונקציות ספציפיות שפונות לנתונים. משתמשים בפונקציה ספציפית לשיחרור כיוון שהנתונים יכולים להיות גם מסוג מבנה וייתכן ותהיה הקצאה דינאמית פנימית במבנה שפונקציה ספציפית תצטרך לשחרר.

שאלה 2:

תזכורת – פונקציה כללית היא פונקציה שלא עוברת שום שינוי כאשר עוברים לעבוד עם נתונים מטיפוס אחר.

נתונה התכנית הבאה - מימוש תור סופי.

```
#include<stdio.h>
#include<stdlib.h>

#define EMPTY 0
#define FULL 100


   /* Node structure definition for queue*/

typedef struct elem     /* an element in the queue */
{
  int num;
  struct elem  *next;
} elem;


            /* Function declarations */
void enqueue(int, elem**, elem **,int *);
void dequeue(elem** , elem **,int *);
int empty(int);
int full(int);



                /* Function implementations */

void dequeue(elem** front, elem **rear ,int *elemNum)      /*delete the
head element from the queue and print it*/
{
 int temp;
 elem *p;

 temp = (*front) -> num;
 printf("  %d ",temp);
 p = *front;
 *front = (*front) ->next;
 (*elemNum)--;
```

```
 free(p);
}

/*-----------------------------------------------------------------*/

void enqueue(int d, elem** front, elem **rear, int *elemNum)
/*insert the element to the tail of queue*/
{
 elem *p;

 p = (elem*)malloc(sizeof(elem));
 p -> num = d;
 p -> next = NULL;
 if(empty(*elemNum) == 0)        /*Not empty*/
   (*rear)->next = p;
else
   *front = p;
 *rear = p;
(*elemNum)++;
}

/*-----------------------------------------------------------------*/

int empty(int elemNumber)
{
 if (elemNumber == 0)
    return 1;
 return 0;
}

/*-----------------------------------------------------------------*/

int full(int elemNumber)
{
 if (elemNumber == FULL)
    return 1;
 return 0;
}



int main()
{
 int pid,count=0;
```

```
elem *front,*rear;
front = rear = NULL;

printf("Enter numbers");
while(1)
{
  scanf("%d",&pid);
  if(full(count) == 0)
    enqueue(pid, &front, &rear, &count);
  else
  {
    printf("The queue is full");
    break;
  }
}

printf("\nq's shedule:\n");
while(empty(count) == 0)
  dequeue(&front, &rear, &count);

return 0;
}
```

יש להפוך את התכנית ל-ADT בהתאם לדרישות בלהלן .

<u>דרישות התכנית:</u>

1. כל הפונקציות הנתונות, אמורות להיות פונקציות כלליות (למעט main), ז"א לא משתנות אם משנים טיפוס נתונים.
2. ניתן להניח שכל ההקצאות הדינאמיות מצליחות ואין צורך לבדוק את זה.
3. יש להגדיר ולממש 2 פונקציות ספציפיות עבור טיפוס נתונים int. תחליטו לבד מה הן אמורות לבצע.
4. **אין** לחלק את התכנית לקבצים.
5. **השינויים שיש להכניס לתכנית ,צריכים להיות הכרחיים וקשורים רק למעבר ל-ADT**.
6. לאחר השינוי המבנה אמור להראות כך:

```
typedef struct elem     /* an element in the queue */
{
 void* num;
 struct elem  *next;
} elem;
```

פתרון שאלה 1:

```
BOOL insertEntry(PNode* head, PNode* tail, void *data, int
(*compare_func) (void*,void*), void (*free_func) (void*))
{
        PNode temp=(*head);
        int flag=0;                 /* flag will identify if "data" already exist in list */
        while(temp!=NULL)  /* check appearnces of data in list*/
        {
                        flag=compare_func(temp->data,data);
                        if (flag == 1)
                                return FALSE;    /* if "data" already exist in list */
                       temp=temp->next;
        }

        temp=(PNode)malloc(sizeof(ListNode));
        if (NULL==temp)
        {
                printf("\nCan't allocate memory");
                freeAll(head, free_func);
                 exit(1);
        }
        temp->data=data;
        temp->next=NULL;
        if((*head) == NULL)
             (*head)=temp;
        else
             (*tail)->next=temp;
         (*tail)=temp;

        return TRUE;
}

void printAll(PNode head, void (*print_func)( void*))
{
        printf("Your list is:");
        while(head!=NULL)
        {
```

```
		print_func(head->data);
		head=head->next;
	}
}



void freeAll(PNode* head, void (*free_func) (void*))
{
	PNode temp;
	while( (*head)!= NULL)
	{
		temp=(*head);
		(*head)=(*head)->next;
		free_func(temp->data);
		free(temp);
	}
}
```

פתרון שאלה 2:

```
            /* General functions declarations */
void enqueue(void*, elem**, elem **,int *);
void dequeue(elem** , elem **,int *,void(*)(void*), void(*)(void*));
int empty(int);
int full(int);

    /* Specific functions */
void free_int(void*);
void print_int(void*);


                    /* Function implementations */

void dequeue(elem** front, elem **rear, int *elemNum,
void(*print)(void*), void(*fr)(void*))      /*delete the head element from
the queue and print it*/
{
 void* temp;

 temp = (*front) -> num;
 print(temp);
 p = *front;
 *front = (*front) ->next;
 (*elemNum)--;
 fr(temp);
 free(p);
}

/*-----------------------------------------------------------------*/

void enqueue(void* d, elem** front, elem **rear, int *elemNum)
/*insert the element to the tail of queue*/
{
 elem *p;

 p = (elem*)malloc(sizeof(elem));
 p -> num = d;
 p -> next = NULL;
```

```
 if(empty(*elemNum) == 0)        /*Not empty*/
   (*rear)->next = p;
else
   *front = p;
 *rear = p;
(*elemNum)++;
}

/*-----------------------------------------------------------------*/

int empty(int elemNumber)
{
 if (elemNumber == 0)
     return 1;
 return 0;
}

/*-----------------------------------------------------------------*/

int full(int elemNumber)
{
 if (elemNumber == FULL)
     return 1;
 return 0;
}


void print_int(void* elem)
{
    printf("   %d ", *(int*)elem);
}

void free_int(void* elem)
{
    free(elem);
}

int main()
{
 int pid, count=0 ,*ppid;   /*new variable for malloc*/
 elem *front,*rear;
 front = rear = NULL;
```

```
 printf("Enter numbers");
 while(1)
 {
   scanf("%d",&pid);
   if (full(count) == 0)
   {
       ppid = (int*)malloc(sizeof(int));
       *ppid = pid;
       enqueue(ppid, &front, &rear, &count);
   }
   else
   {
     printf("The queue is full");
     break;
   }
 }

 printf("\nq's shedule:\n");
 while(empty(count) == 0)
   dequeue(&front,&rear, &count, print_int, free_int);

return 0;
}
```