ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2017-2018

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ

ΜΑΘΗΜΑ: ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

ΕΡΓΑΣΤΗΡΙΟ 5 – ΣΗΜΕΙΩΣΕΙΣ

Ουρές

Μια ουρά αποτελεί μια δομή δεδομένων στη λογική του First-in First-Out, FIFO, στην οποία το κάθε στοιχείο που εισάγεται πρώτο, θα εξαχθεί επίσης πρώτο. Τα νεοεισερχόμενα στοιχεία τοποθετούνται στο τέλος της ουράς ενώ απαιτούνται δύο δείκτες για τον προσδιορισμό του πρώτου και του τελευταίου στοιχείου: ο FRONT και ο REAR. Μια ουρά μπορεί εύκολα να υλοποιηθεί είτε με πίνακες ή με λίστες.

Οι ακόλουθες εικόνες παρουσιάζουν ένα παράδειγμα υλοποίησης μιας ουράς με τη βοήθεια ενός πίνακα καθώς επίσης και της μεταβολής της κατά την εισαγωγή και εξαγωγή ενός στοιχείου.

12	9	7	18	14	36				
0	1	2	3	4	5	6	7	8	9

12	9	7	18	14	36	45			
0	1	2	3	4	5	6	7	8	9

	9	7	18	14	36	45			
0	1	2	3	4	5	6	7	8	9

Κατά την εισαγωγή ενός στοιχείου μεταβάλλεται ο δείκτης REAR ενώ κατά την εξαγωγή μεταβάλλεται ο δείκτης FRONT. Σε κάθε περίπτωση, πριν την εισαγωγή ενός στοιχείου θα πρέπει να γίνεται έλεγχος ώστε να μην έχει γεμίσει η ουρά (overflow - υπερχείλιση). Ομοίως, πριν την εξαγωγή ενός στοιχείου, θα πρέπει να ελέγχουμε αν υπάρχουν στοιχεία μέσα στην ουρά (underflow – υποχείλιση). Υποχείλιση έχουμε όταν FRONT=REAR=-1. Οι αλγόριθμοι εισαγωγής και εξαγωγής στοιχείων από την ουρά έχουν ως ακολούθως:

```
Step 1: IF REAR = MAX-1
              Write OVERFLOW
              Goto step 4
        [END OF IF]
Step 2: IF FRONT = -1 and REAR = -1
              SET FRONT = REAR = 0
        ELSE
              SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = NUM
Step 4: EXIT
```

```
Step 1: IF FRONT = -1 OR FRONT > REAR
              Write UNDERFLOW
        ELSE
              SET VAL = QUEUE[FRONT]
              SET FRONT = FRONT + 1
        [END OF IF]
Step 2: EXIT
```

Code 1

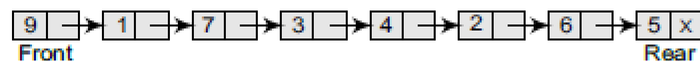
(Υλοποιεί με τη χρήση ξεχωριστών συναρτήσεων όλες τις λειτουργίες σε μια ουρά – υλοποίηση με πίνακα)

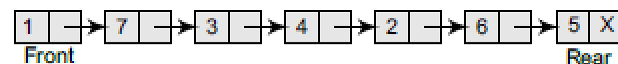
```
#include <stdio.h>
#include <conio.h>
#define MAX 10 // Changing this value will change length of array
int queue[MAX];
int front = -1, rear = -1;

void insert(void);
int delete_element(void);
int peek(void);
void display(void);

int main()
{
	int option, val;
	do
	{
		printf("\n\n ***** MAIN MENU *****");
		printf("\n 1. Insert an element");
		printf("\n 2. Delete an element");
		printf("\n 3. Peek");
		printf("\n 4. Display the queue");
		printf("\n 5. EXIT");
		printf("\n Enter your option : ");
		scanf("%d", &option);
		switch(option)
		{
			case 1:
				insert();
				break;
			case 2:
				val = delete_element();
				if (val != -1)
					printf("\n The number deleted is : %d", val);
				break;
			case 3:
				val = peek();
				if (val != -1)
					printf("\n The first value in queue is : %d", val);
				break;
			case 4:
				display();
				break;
			}
	}while(option != 5);
	getch();
	return 0;
```

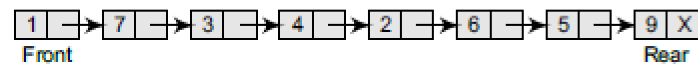
```
}

void insert()
{
	int num;
	printf("\n Enter the number to be inserted in the queue : ");
	scanf("%d", &num);
	if(rear == MAX-1)
		printf("\n OVERFLOW");
	else if(front == -1 && rear == -1)
		front = rear = 0;
	else
		rear++;
	queue[rear] = num;
}

int delete_element()
{
	int val;
	if(front == -1 || front>rear)
	{
		printf("\n UNDERFLOW");
		return -1;
	}
	else
	{
		val = queue[front];
		front++;
		if(front > rear)
		front = rear = -1;
		return val;
	}
}

int peek()
{
	if(front==-1 || front>rear)
	{
		printf("\n QUEUE IS EMPTY");
		return -1;
	}
	else
	{
		return queue[front];
	}
}

void display()
```

```
{
        int i;
        printf("\n");
        if(front == -1 || front > rear)
                printf("\n QUEUE IS EMPTY");
        else
        {
                for(i = front;i <= rear;i++)
                        printf("\t %d", queue[i]);
        }
}
```

Το μειονέκτημα της χρήσης των πινάκων είναι γνωστό πως αφορά στο σταθερό τους μέγεθος. Αν όμως δεν μπορούμε να γνωρίζουμε εκ των προτέρων το μέγεθος μιας ουράς, τότε η καλύτερη λύση είναι να χρησιμοποιήσουμε υλοποίηση με λίστες. Οι λίστες βολεύουν ιδιαίτερα όταν η ουρά πρόκειται να έχει μεγάλο μέγεθος. Στην υλοποίηση με λίστες, κάθε κόμβος έχει τα δεδομένα καθώς και το δείκτη προς τον επόμενο κόμβο. Ο πρώτος κόμβος είναι ο FRONT ενώ σε σχέση με την υλοποίηση μιας απλής συνδεδεμένης λίστας, υιοθετούμε ακόμα ένα δείκτη προς τον τελευταίο κόμβο τον REAR. Όλες οι εισαγωγές γίνονται στο τέλος ενώ οι εξαγωγές στην αρχή (μπορούμε να υιοθετήσουμε τις κατάλληλες συναρτήσεις που μελετήσαμε στο προηγούμενο εργαστήριο). Η ακόλουθη εικόνα παρουσιάζει ένα παράδειγμα διαχείρισης μιας ουράς με τη βοήθεια μιας λίστας:







Ακολουθούν οι αλγόριθμοι εισαγωγής και εξαγωγής ενός στοιχείου από μια ουρά.

```
Step 1: Allocate memory for the new node and name
        it as PTR
Step 2: SET PTR -> DATA = VAL
Step 3: IF FRONT = NULL
            SET FRONT = REAR = PTR
            SET FRONT -> NEXT = REAR -> NEXT = NULL
        ELSE
            SET REAR -> NEXT = PTR
            SET REAR = PTR
            SET REAR -> NEXT = NULL
        [END OF IF]
Step 4: END
```

```
Step 1: IF FRONT = NULL
            Write "Underflow"
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = FRONT
Step 3: SET FRONT = FRONT -> NEXT
Step 4: FREE PTR
Step 5: END
```

Code 2

(Υλοποιεί με τη χρήση ξεχωριστών συναρτήσεων όλες τις λειτουργίες σε μια ουρά – υλοποίηση με λίστα)

```
/*Queue - Linked List implementation*/
#include<stdio.h>
#include<stdlib.h>
struct Node {
	int data;
	struct Node* next;
};
// Two glboal variables to store address of front and rear nodes.
struct Node* front = NULL;
struct Node* rear = NULL;

// To Enqueue an integer
void Enqueue(int x) {
	struct Node* temp =
		(struct Node*)malloc(sizeof(struct Node));
	temp->data =x;
	temp->next = NULL;
	if(front == NULL && rear == NULL){
		front = rear = temp;
		return;
	}
	rear->next = temp;
	rear = temp;
}

// To Dequeue an integer.
void Dequeue() {
	struct Node* temp = front;
	if(front == NULL) {
		printf("Queue is Empty\n");
		return;
	}
	if(front == rear) {
		front = rear = NULL;
	}
	else {
		front = front->next;
	}
	free(temp);
}

int Front() {
	if(front == NULL) {
		printf("Queue is empty\n");
		return 0;
	}
	return front->data;
}

void Print() {
	struct Node* temp = front;
```

```
        while(temp != NULL) {
                printf("%d ",temp->data);
                temp = temp->next;
        }
        printf("\n");
}

int main(){
        /* Drive code to test the implementation. */
        // Printing elements in Queue after each Enqueue or Dequeue
        Enqueue(2); Print();
        Enqueue(4); Print();
        Enqueue(6); Print();
        Dequeue();  Print();
        Enqueue(8); Print();
}
```