МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Лабораторна робота №4

з дисципліни

«Алгоритмізації та програмування»

Виконав:
студент групи КН-108
Левицький Богдан

Львів – 2018 р.

Варіант 15

1) Реалізувати з використанням списків двонаправлене кільце (перегляд можливий в обидва боки, від останнього елемента можна перейти до першого).
2) Роздрукувати отриманий масив, починаючи з К-ого елемента і до К-1 (по кільцю вліво).
3) Знищити з кільця перший й останній елементи.
4) Роздрукувати отриманий масив, починаючи з К-ого елемента (і до К+1 по кільцю вправо).

```
#include <stdio.h>

typedef struct Node
{
    int data;
    struct Node* next;
    struct Node* prev;
}Node;

void push(Node** head,int num);

void headPrevPtr(Node* head);

void printRight(Node* head, int k);

Node* Pos(Node* head,int pos);
```

```
void printleft(Node* head,int k);

void killFirstElem(Node** head);

void killLastElem(Node* head);

int main(int argc,char* argv[])
{
	Node* head = NULL;
	int SIZE;
	printf("Write a size of array:\t");
	scanf("%d",&SIZE);
	if(SIZE <= 0)
	{
		printf("Size should be bigger than 0");
		return -1;
	}
	int arr[SIZE];
	for(int i = 0; i < SIZE;i++)
	{
		printf("Write a %d num of array:\t",i+1);
		scanf("%d",&arr[i]);
		push(&head,arr[i]);
	}
	int k;
	do
	{
		printf("Write k,k > 0, k < %d:\t",SIZE);
```

```
        scanf("%d",&k);
    }
    while(k<=0|| k > SIZE);

    int temp = SIZE-k+1;
    printf("Array:\n\n");
    print(head);
    printf("Array from k to k - 1 leftside:\n\n");
    printleft(head,temp);
    killFirstElem(&head);
    killLastElem(head);
    printf("Array after killing first and last elements:\n\n");
    print(head);
    printf("Array after killing first and last elements from k to k + 1 rightside:\n\n");
    printRight(head,k);
    return 0;
}

void push(Node** head,int num)
{
    if(*head == NULL)
    {
        Node *tmp = (Node*) malloc(sizeof(Node));
        tmp->data = num;
        tmp->next = NULL;
        tmp->prev = NULL;
        (*head) = tmp;
    }
```

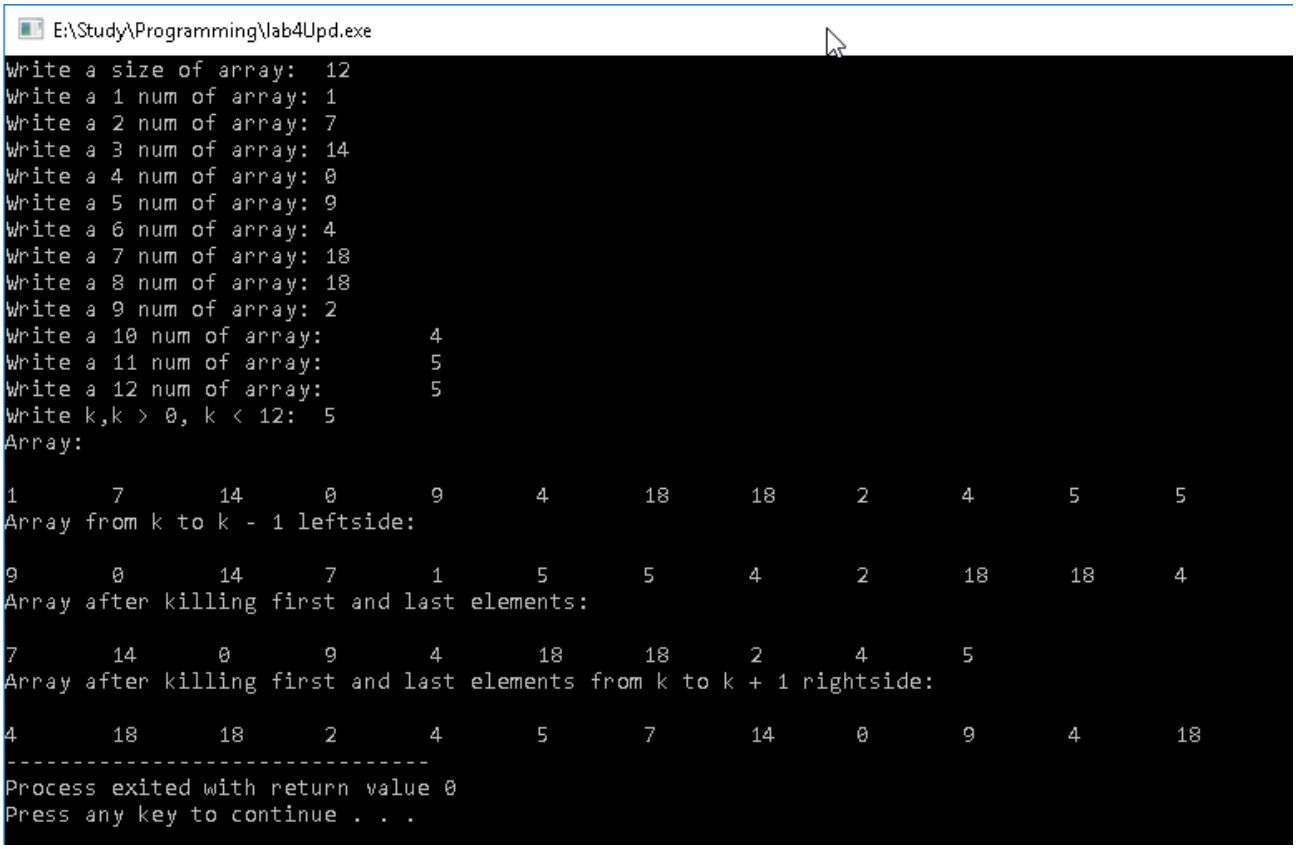
```
    else
    {
        Node* tmp = (Node*) malloc(sizeof(Node));
        tmp->data = num;
        tmp->prev = NULL;
        tmp->next = *(head);
        (*head) = tmp;
        tmp->next->prev = *(head);
    }
}

void headPrevPtr(Node* head)
{
    Node* tmp = head;
    while(tmp->next)
    {
        tmp = tmp->next;
    }
    head->prev = tmp;
}

void printRight(Node* head, int k)
{
    Node *tmp =head;
    headPrevPtr(head);
    int count = 0;
    int count1 = 0;
    int temp;
```

```
    while(tmp)
    {
        tmp = tmp->next;
        count1++;
    }
    temp = count1;
    while(count < k && head->next)
    {
        head = head->next;
        count++;
    }
    count1++;
    while(count1 !=0)
    {
        printf("%d\t",head->data);
        head = head->prev;
        count1 --;
    }
   if(temp!= 1)
    {
        printf("%d\t",head->data);
    }
}

Node* Pos(Node* head,int pos)
{
    int count = 0;
    while(count < pos && head->next)
```

```
    {
        head = head->next;
        count++;
    }
    return head;
}

void printleft(Node* head,int k)
{
    Node* temp = head;
    int count = 0;
    while(count < k-1 && head->next)
    {
        head = head->next;
        count++;
    }
    while(head)
    {
        printf("%d\t",head->data);
        head = head->next;
        if(head == NULL)
        {
            head = Pos(temp,0);
            for(int i = 0; i < count;i++)
            {
                printf("%d\t", head->data);
                head = head->next;
            }
```

```
                    break;
                }
        }
        printf("\n");
}
void print(Node* head)
{
        while(head->next)
        {
                head = head->next;
        }
        while(head)
        {
                printf("%d\t",head->data);
                head = head->prev;
        }
        printf("\n");
}
void killFirstElem(Node** head)
{
        if((*head) == NULL)
        {
                printf("List is empty");
                return -1;
        }
        if((*head)->next == NULL)
        {
                free((*head));
```

```
            (*head) = NULL;
            printf("List consists of 1 element and is empty after killing");
            exit(0);
        }
        Node* tmp = *head;
        *head = (*head)->next;
        (*head)->prev = NULL;
        free(tmp);
}
void killLastElem(Node* head)
{
        if(head == NULL)
        {
            exit(-2);
        }
        if(head->next == NULL)
        {
            printf("List consists of 2 elements and is empty after killing");
            exit(2);
        }
        while(head->next->next)
        {
            head = head->next;
        }
        Node* tmp = head;
        tmp->next = NULL;
        free(head->next);
}
```


E:\Study\Programming\lab4Upd.exe
Write a size of array: 12
Write a 1 num of array: 1
Write a 2 num of array: 7
Write a 3 num of array: 14
Write a 4 num of array: 0
Write a 5 num of array: 9
Write a 6 num of array: 4
Write a 7 num of array: 18
Write a 8 num of array: 18
Write a 9 num of array: 2
Write a 10 num of array: 4
Write a 11 num of array: 5
Write a 12 num of array: 5
Write k,k > 0, k < 12: 5
Array:
1 7 14 0 9 4 18 18 2 4 5 5
Array from k to k - 1 leftside:
9 0 14 7 1 5 5 4 2 18 18 4
Array after killing first and last elements:
7 14 0 9 4 18 18 2 4 5
Array after killing first and last elements from k to k + 1 rightside:
4 18 18 2 4 5 7 14 0 9 4 18

Process exited with return value 0
Press any key to continue . . .