Universitatea Tehnică a Moldovei

Facultatea *Calculatoare, Informatică și Microelectronică*

Specialitatea *Tehnologii Informaționale*

Raport

la lucrarea de laborator nr. 5

Tema: *"Algoritmi de prelucrare a stivelor și șirurilor în așteptare"*

Disciplina: "Structuri de date și algoritmi"

Varianta 4

A efectuat:	Student grupa TI-231 FR	Apareci Aurica
A verificat:	Asistent universitar	Mantaluță Marius

Chișinău 2024

Cuprins

1. Cadrul teoretic

Scopul: Obținerea deprinderilor practice de implementare și de utilizare a tipului abstract de date „Stack” și „Queue” în limbajul C cu asigurarea operațiilor de prelucrare de bază.

Sarcina Să se scrie un program în limbajul C pentru implementarea și utilizarea tipului abstract de date „Stivă” și „Șir în așteptare” cu asigurarea operațiilor de prelucrare de bază. Funcția main() va afișa la ecran următorul meniu de opțiuni de bază:

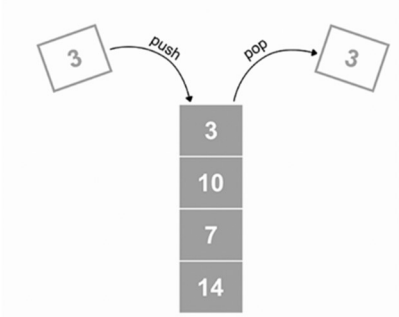
1. Crearea unei stive dinamice
2. Citirea datelor referitoare la elementele stivei de la tastatură.
3. Afișarea datelor stivei într-un fișier stiva.txt/stiva.bin.
4. Căutarea elementului maximal din stivă după un câmp numeric.
5. Copierea datelor din stivă într-un șir în așteptare (queue).
6. Afișarea datelor șirului în așteptare într-un fișier queue.txt/queue.bin.
7. Determinarea lungimii stivei (numărul de elemente).
8. Modificarea câmpurilor unui element din stivă.
9. Adăugarea datelor modificate a elementului din stivă la sfârșitul fișierului stiva.txt/.bin.
10. Eliberarea memoriei alocate pentru stivă.
11. Eliberarea memoriei alocate pentru Queue.
0. Ieșire din program.

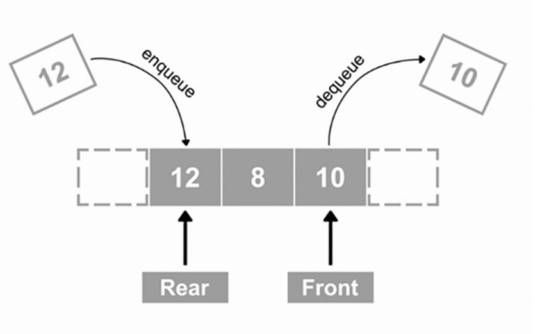
Varianta 4. Structura Imobil cu câmpurile: proprietarul, tipul, adresa, suprafața, costul.

2. Repere teoretice

Stivă (Stack)	Coadă (Queue)
O stivă este o structură de date de tip **LIFO**, ceea ce înseamnă că ultimul element adăugat în stivă este primul element extras.	O coadă este o structură de date de tip **FIFO**, ceea ce înseamnă că primul element adăugat este primul element extras.
Operațiile de bază ale unei stive sunt **push** (adaugă un element la sfârșitul stivei) și **pop** (elimină și returnează elementul cel mai recent adăugat).	Operațiile de bază ale unei cozi sunt **enqueue** (adaugă un element la sfârșitul cozii) și **dequeue** (elimină și returnează primul element adăugat).

Elementele sunt accesate într-o ordine inversă față de ordinea adăugării: ultimul element adăugat este primul element extras.	Elementele sunt accesate în ordinea în care au fost adăugate: primul element adăugat este primul element extras.





3. Listingul programului

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "user.h"

int main(){
    system("cls");
    while (go)
    {
        userChose = Menu();
        BL();
    }
    return 0;
}
```

struct.h

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

typedef struct Imobil {
    char proprietar[100];
    char tip[100];
    char adresa[15];
    float suprafata;
    int cost;
} Imobil;

Imobil * ReadData(){
    Imobil *c = (Imobil *)malloc(sizeof(Imobil));
    printf("--------------------------------\n");
    printf("Denumirea: ");
    fflush(stdin);
    scanf("%s", c->proprietar);
    printf("Adresa: ");
    fflush(stdin);
    scanf("%s", c->adresa);
    printf("Tipul: ");
```

```
    fflush(stdin);
    scanf("%s", c->tip);
    printf("Suprafata: ");
    fflush(stdin);
    scanf("%f", &c->suprafata);
    printf("Costul: ");
    fflush(stdin);
    scanf("%d", &c->cost);
    printf("--------------------------------\n");
    return c;
}

char * ToString(Imobil * c){
    char * str = (char *)malloc(1000 * sizeof(char));
    sprintf(str, "Proprietar: %s\nAdresa: %s\nTipul: %s\nSuprafata: %.2f\nCostul:
%d\n", c->proprietar, c->adresa, c->tip, c->suprafata, c->cost);
    return str;
}

void PrintData(Imobil *c){
    printf("\n--------------------------------\n");
    printf("Proprietar: %s\n", c->proprietar);
    printf("Adresa: %s\n", c->adresa);
    printf("Tipul: %s\n", c->tip);
    printf("Suprafata: %.2f\n", c->suprafata);
    printf("Costul: %d\n", c->cost);
    printf("--------------------------------\n");
}

void UpdateData(Imobil * c){
    char resp;
    printf("Modificati proprietarul?(y/n) ->");
    fflush(stdin);
    scanf("%c", &resp);
    if (tolower(resp) == 'y')
    {
        fflush(stdin);
        printf("Proprietarul noua:\t");
        scanf("%s", c->proprietar);
    }
    printf("Modificati adresa?(y/n) ->");
    fflush(stdin);
    scanf("%c", &resp);
    if (tolower(resp) == 'y')
    {
        fflush(stdin);
        printf("Adresa noua:\t");
        scanf("%s", c->adresa);
    }
    printf("Modificati tipul?(y/n) ->");
    fflush(stdin);
    scanf("%c", &resp);
    if (tolower(resp) == 'y')
    {
        fflush(stdin);
```

```
        printf("Tipul nou:\t");
        scanf("%s", c->tip);
    }
    printf("Modificati suprafata?(y/n) ->");
    fflush(stdin);
    scanf("%c", &resp);
    if (tolower(resp) == 'y')
    {
        fflush(stdin);
        printf("Suprafata noua:\t");
        scanf("%f", &c->suprafata);
    }
    printf("Modificati costul?(y/n) ->");
    fflush(stdin);
    scanf("%c", &resp);
    if (tolower(resp) == 'y')
    {
        fflush(stdin);
        printf("costul nou:\t");
        scanf("%d", &c->cost);
    }
}

void swap(Imobil *a, Imobil *b){
    Imobil aux = *a;
    *a = *b;
    *b = aux;
}
```

user.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#include "struct.h"
#include "Stack.h"
#include "Queue.h"

Stack * s; queue * q; int nr = 0;

void InitStack(){
    int n; printf("Introduceti capacitatea stivei: ");
    scanf("%d", &n);
    s = createStack(n);
}

void ReadStack(){
    int n; printf("Introduceti numarul de elemente din stiva: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        Imobil * c = ReadData();
        Element * e = createElement(c);
        push(s, e);
    }
```

```
}

void ExportStack(){
    FILE * f = fopen("stiva.txt", "w");
    for (int i = 0; i <= s->top; i++)
    {
        char * str = ToString((Imobil *)s->array[i].data);
        fwrite(str, sizeof(char), strlen(str), f);
    }
    fclose(f);
}

void StackMax(){
    Imobil * max = (Imobil *)malloc(sizeof(Imobil));
    max = (Imobil *)s->array[0].data;
    for (int i = 1; i <= s->top; i++)
    {
        if (max->cost < ((Imobil *)s->array[i].data)->cost)
        {
            max = (Imobil *)s->array[i].data;
        }
    }
    printf("Imobilul cu costul maxim este: \n");
    PrintData(max);
}

void TransferToQueue(){
    q = createQueue(sizeof(Imobil));
    for (int i = 0; i <= s->top; i++)
    {
        data * d = (data *)malloc(sizeof(data));
        d->data = s->array[i].data;
        d->next = NULL;
        if (isEmpty(q))
        {
            q->head = q->tail = d;
        }
        else
        {
            q->tail->next = d;
            q->tail = d;
        }
        q->size++;
    }
}

void ExportQueue(){
    FILE * f = fopen("coada.txt", "w");
    data * d = q->head;
    while (d != NULL)
    {
        char * str = ToString((Imobil *)d->data);
        fwrite(str, sizeof(char), strlen(str), f);
        d = d->next;
    }
```

```
    fclose(f);
}

void Len(){
    printf("Lungimea stivei este: %d\n", s->top + 1);
}

void Update(){
    int index = 0;
    printf("Introduceti indexul Imobilului ce urmeaza a fi modificat: ");
    scanf("%d", &index);
    if (index < 0 || index > s->top)
    {
        printf("Indexul introdus nu este valid\n");
        return;
    }
    UpdateData(s->array[index].data);
}

int userChose = 0; int go = 1;

int Menu(){
    printf("-------------------------Meniul Aplicatiei-------------------------\n");
    printf("1 . \tCreaza stiva noua\n");
    printf("2 . \tCitire date stiva\n");
    printf("3 . \tExport date stiva\n");
    printf("4 . \tAfiseaza maxim din stiva\n");
    printf("5 . \tTransfer catre coada\n");
    printf("6 . \tExport date coada\n");
    printf("7 . \tLungime stiva\n");
    printf("8 . \tModificare element stiva\n");
    printf("9 . \tElibereaza memoria stivei\n");
    printf("10 . \tElibereaza memoria cozii\n");
    printf("0 . \tExit\n");
    printf("-------------------------------------------------------------------
\nOptiunea aleasa --> ");
    int op;
    scanf("%d", &op);
    system("cls");
    return op;
}

void PressAnyKey(){
    printf("\nAtingeti o tasta pentru a continua\n");
    getch(); system("cls");
}

void BL(){
    switch (userChose)
    {
        case 1:
        {
            InitStack();
            PressAnyKey();
        } break;
```

```
        case 2:
        {
            ReadStack();
            PressAnyKey();
        }break;
        case 3:
        {
            ExportStack();
            PressAnyKey();
        }break;
        case 4:
        {
            StackMax();
            PressAnyKey();
        }break;
        case 5:
        {
            TransferToQueue();
            PressAnyKey();
        }break;
        case 6:
        {
            ExportQueue();
            PressAnyKey();
        }break;
        case 7:
        {
            Len();
            PressAnyKey();
        }break;
        case 8:
        {
            Update();
            PressAnyKey();
        }break;
        case 9:
        {
            deleteStack(s);
            PressAnyKey();
        }break;
        case 10:
        {
            destroyQueue(q);
            PressAnyKey();
        }break;
        case 0:
        {
            go=0;
            system("cls");
            printf("Aplicatia s-a oprit cu succes");
            getch();
        }break;
        default:printf("Optiune necunoscuta\nIncercati din nou");break;
    }
}
```

Stack.h

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

typedef struct Element{
    void* data;
    int index;
}Element;

Element* createElement(void* data){
    Element* element = (Element*)malloc(sizeof(Element));
    element->data = data;
    return element;
}

typedef struct Stack{
    int top;
    unsigned capacity;
    Element* array;
}Stack;

Stack* createStack(unsigned capacity){
    struct Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (Element*)malloc(stack->capacity * sizeof(Element));
    return stack;
}

bool isFull(Stack* stack){
    return stack->top == stack->capacity - 1;
}

bool isStEmpty(Stack* stack){
    return stack->top == -1;
}

bool push(Stack* stack, Element* item){
    if (isFull(stack))
    {
        printf("Stack Overflow \n");
        return false;
    }
    stack->top++;
    stack->array[stack->top] = *item;
    stack->array[stack->top].index = stack->top;
    return true;
}

Element* pop(Stack* stack){
    if (isStEmpty(stack))
    {
        printf("Stack is empty\n");
        return NULL;
    }
```

```
    return &stack->array[stack->top--];
}

Element* peek(Stack* stack){
    if (isStEmpty(stack))
    {
        printf("Stack is empty\n");
        return NULL;
    }
    return &stack->array[stack->top];
}

void emptyStack(struct Stack* stack){
    stack->top = -1;
}

void deleteStack(struct Stack* stack){
    emptyStack(stack);
    free(stack->array);
    free(stack);
}

void deleteElement(Element* element){
    free(element->data);
    free(element);
}
```

Queue.h

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

typedef struct _data {
            void* data;
            struct _data* next;
}data;

typedef struct _queue {
            size_t size;
            size_t allocationSize;
            data* head;
            data* tail;
}queue;

queue* createQueue(size_t allocSize){
            queue* q = (queue*)malloc(sizeof(queue));
            if (q == NULL)
            {
            return NULL;
            }
            q->allocationSize = allocSize;
            q->size = 0;
            q->head = q->tail = NULL;
            return q;
}
```

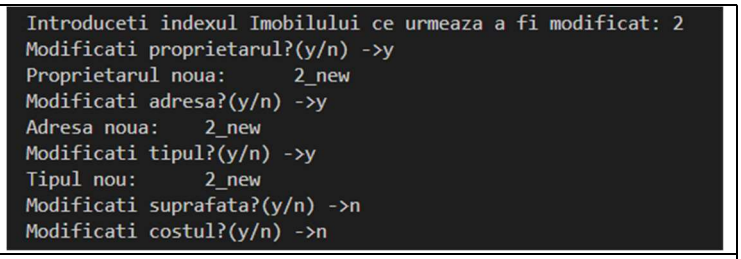
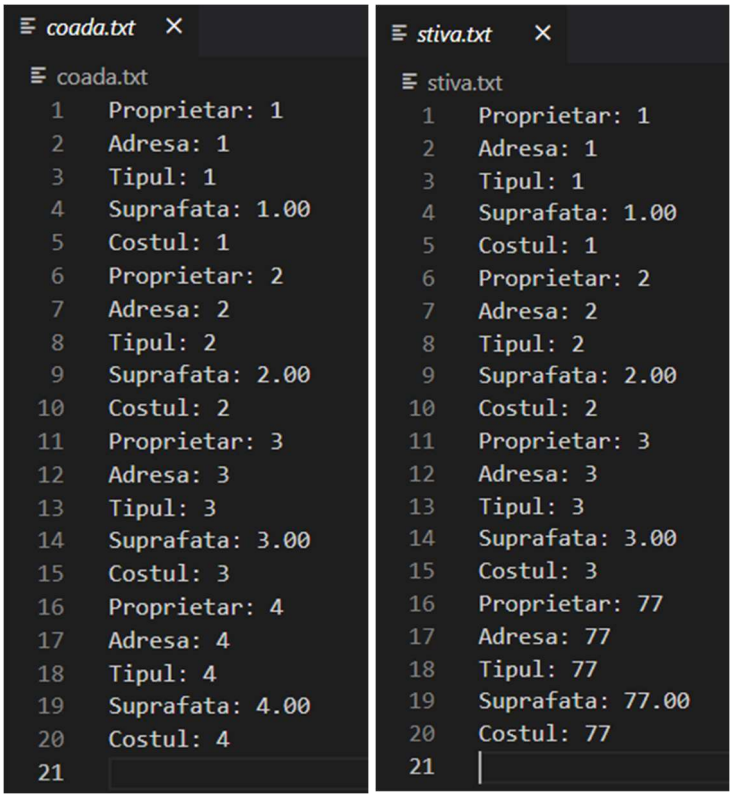
```
bool isEmpty(queue* q){
            return q->size == 0 ? true : false;
}

queue* clearQueue(queue* q){
            if (q == NULL)
            {
            return NULL;
            }

            while (!isEmpty(q))
            {
            data* temp = q->head;
            q->head = q->head->next;
            free(temp->data);
            free(temp);
            q->size--;
            }

            return q;
}

size_t getSize(queue* q){
            return q->size;
}

void destroyQueue(queue* q){
            clearQueue(q);
            free(q);
}

queue* enqueue(queue* q, void* _data){
            if (q == NULL)
            {
            return NULL;
            }
            data* toInsert = (data*)malloc(sizeof(data));
            if (toInsert == NULL)
            {
            return NULL;
            }
            toInsert->data = malloc(q->allocationSize);
            if (toInsert->data == NULL)
            {
            return NULL;;
            }
            toInsert->next = NULL;
            memcpy(toInsert->data, _data, q->allocationSize);
            if (q->size == 0)
            {
            q->head = q->tail = toInsert;
            }
            else
            {
            q->tail->next = toInsert;
```

```
            q->tail = toInsert;
            }

            q->size++;

            return q;
}

queue* dequeue(queue* q, void* toRet){
            if (q == NULL)
            {
            return NULL;
            }

            data* toDel = q->head;
            if (q->size == 1)
            {
            memcpy(toRet, toDel->data, q->allocationSize);
            free(toDel->data);
            free(toDel);
            q->head = q->tail = NULL;
            q->size--;
            return q;
            }
            q->head = q->head->next;
            memcpy(toRet, toDel->data, q->allocationSize);
            free(toDel->data);
            free(toDel);
            q->size--;

            return q;
}

queue* front(queue* q, void* toRet){
            if (q == NULL)
            {
            return NULL;
            }

            memcpy(toRet, q->head->data, q->allocationSize);

            return q;
}

queue* reverse(queue* q){
            if (q == NULL) return NULL;
            if (getSize(q) == 0) return q;
            else {
            data temp;
            dequeue(q, &temp);
            reverse(q);
            enqueue(q, &temp);
            return q;
            }
}
```

4. Testarea aplicației

```
------------------------Meniul Aplicatiei------------------------
1 .     Creaza stiva noua
2 .     Citire date stiva
3 .     Export date stiva
4 .     Afiseaza maxim din stiva
5 .     Transfer catre coada
6 .     Export date coada
7 .     Lungime stiva
8 .     Modificare element stiva
9 .     Elibereaza memoria stivei
10 .    Elibereaza memoria cozii
0 .     Exit
-----------------------------------------------------------------
```

Nr.	Input	Output
1.	Citire date stivă	Introduceti numarul de elemente din stiva: 4 --- Denumirea: 1 Adresa: 1 Tipul: 1 Suprafata: 1 Costul: 1 --- --- Denumirea: 2 Adresa: 2 Tipul: 2 Suprafata: 2 Costul: 2 --- --- Denumirea: 3 Adresa: 3 Tipul: 3 Suprafata: 3 Costul: 3 ---
2.	Afișare maxim din stivă	Imobilul cu costul maxim este: --- Proprietar: 4 Adresa: 4 Tipul: 4 Suprafata: 4.00 Costul: 4 ---
3.	Transfer către stivă și determinare lungime stivă	Lungimea stivei este: 4 Atingeti o tasta pentru a continua

4.	Modificare element stivă	
6.	Export date coadă/stivă	

5. Concluzii

În concluzie, lucrarea de laborator realizată a reprezentat o oportunitate de aplicare a cunoștințelor teoretice în practică. Sarcina evidențiază succesul în implementarea și utilizarea tipului abstract de date "Stivă" și "Șir în așteptare" în limbajul C, satisfăcând cerințele de bază și oferind operații de prelucrare esențiale.

Programul dezvoltat permite utilizatorului să interacționeze eficient cu stiva și coada, oferind funcționalități precum crearea și gestionarea dinamică a stivei, citirea și afișarea datelor, căutarea elementului maximal, copierea datelor într-un șir în așteptare, determinarea lungimii stivei, modificarea câmpurilor elementului și eliberarea memoriei alocate. Structura Imobil, cu câmpurile sale specifice (proprietarul, tipul, adresa, suprafața, costul), a fost integrată cu succes în acest sistem, demonstrând capacitatea programului de a gestiona diverse tipuri de date. În concluzie, implementarea și utilizarea eficientă a acestor tipuri abstracte de date în limbajul C reprezintă un pas semnificativ în dobândirea deprinderilor practice în programare și prelucrarea datelor.