Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

**RAPORT**

Lucrare de laborator Nr.5

*la Structuri de Date și Algoritmi*

Tema: Algoritmi de prelucrare a listelor liniare simplu înlănțuite,

listelor dublu înlănțuite, listelor circulare, cozilor și stivelor

A efectuat: st. gr. SI-212 Șeremet Alexandru

A verificat: lect. asist. Mititelu Vitalii

Chişinău 2022

**Scopul:**

Obținerea deprinderilor practice de implementare și de utilizare

a tipurilor abstracte de date „Listă liniară simplu înlănțuită”, „Listă liniară

dublu înlănțuită”, „Listă circulară”, „Stivă”, „Coadă” în limbajul C cu

asigurarea operațiilor de prelucrare de bază ale listei.

**Sarcina:**

15. Scrieți un program, care dintr-o listă circulară de 100 de numere aleatoare să determine numărul maximal și cel minimal. Să se determine consecutivitatea de elemente, ce se află între numărul maximal și cel minimal determinat. Să se scrie o funcție care testează inegalitatea a două mulțimi.

**Rezumat:**

**Liste circulare**

In unele aplicatii este necesar sa se prelucreze structuri de date inlantuite simple sau duble circulare. Acestea se obtin din liste liniare printr-o singura operatie daca lista este simpla sau prin doua operatii daca lista este dubla:

Fie urmatoarea lista dubla definite astfel:

struct nod

{int info;

nod \* next,\* back;};

nod\*prim,\*ultim;

Lista devine circulara prin operatiile:

ultim->next=prim; //urmatorul ultimului devine primul

prim->back=ultim; //precedentul primului devine ultimul

Prin urmare nici un nod nu va mai contine pentru campurile \*next sau \*back valoarea 0 (ultimul si primul in cazul listelor liniare) ceea ce va determina modificarea functiei de parcurgere in scopul prelucrarilor.

**Stive**

Stiva (sau „stack” in limba engleza) este o structură de date liniara ce lucreaza cu o colectie de date avand doua operatii principale:

push(int numar) – adauga numar in stiva

pop() – sterge ultimul element adaugat in lista (peek() – returneaza valoarea acestui element – deseori functia pop() contine si functia peek() integrata).

Modul in care acesta prelucreaza elementele i-a dat denumirea de LIFO ( Last In First Out= ultimul venit-primul servit ).

Avand in vedere ca este o structura de date secventiala, operatiile push si pop se intampla numai la capatul structurii (acest „capat” este numit top). Acest detaliu permite stivei sa fie implementata ca o lista simpla inlantuita. De mentionat este faptul ca stiva este o structura de date cu un numar total de elemente flexibil.

**Cozi**

Coada (sau „queue” in limba engleza) este o structura de date liniara ce lucreaza cu o colectie de date avand doua operatii principale:

push(int numar) – adauga un numar in coada

pop() – sterge ultimul element din lista (front() – returneaza valoarea acestui element – deseori functia pop() contine si functia front() integrata)

Modul in care aceasta prelucreaza elementele i-a dat denumirea de FIFO ( First In First Out = primul venit-primul servit)

Avand in vedere ca este o structura de date secventiala, operatiile push si pop se intampla numai la capetele structurii. Acest detaliu permite cozii sa fie implementata ca o lista simpla inlantuita. Nota: Coada este o structura de date cu un numar total de elemente flexibil.

Coada functioneaza pe principiul FIFO, imaginati-va o coada la intrarea intr-un club.

**Codul deplin al programului:**

**Directoriul task\_1**

**Fișierul code.c:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "includes.h"

Node \*root = NULL;

int greatest = INT\_MIN, lowest = INT\_MAX;

void insert\_end()

{

Node \*new\_node = malloc(sizeof(Node));

if (new\_node == NULL)

{

exit(1);

}

new\_node->next = NULL;

new\_node->data = rand();

if (root == NULL)

{

root = new\_node;

return;

}

Node \*curr = root;

while (curr->next != NULL)

{

curr = curr->next;

}

curr->next = new\_node;

}

void make\_cycle()

{

if (root == NULL)

{

return;

}

Node \*curr = root;

while (curr->next != NULL)

{

curr = curr->next;

}

curr->next = root;

}

void create\_list()

{

int i = 100;

for (; i > 0; i--)

insert\_end();

make\_cycle();

}

void show()

{

Node \*tmp = root;

for (; tmp->next != root; tmp = tmp->next)

{

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

}

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

}

int count()

{

int c = 1;

for (Node \*curr = root; curr->next != root; curr = curr->next)

{

c++;

}

return c;

}

Node \*get\_node(int index)

{

int i = 0;

for (Node \*tmp = root; tmp->next != root; tmp = tmp->next, i++)

{

if (i == index)

return tmp;

}

}

void find\_extremes()

{

Node \*tmp = root;

for (; tmp->next != root; tmp = tmp->next)

{

if (tmp->data > greatest)

greatest = tmp->data;

if (tmp->data < lowest)

lowest = tmp->data;

}

printf("\nmaxim: %d \nminim: %d", greatest, lowest);

}

void get\_extremes\_interval()

{

Node \*tmp = root;

for (; tmp->data != greatest; tmp = tmp->next)

;

tmp = tmp->next;

printf("\n");

for (; tmp->data != lowest; tmp = tmp->next)

{

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

}

}

void free\_list()

{

Node \*curr = root;

while (curr != NULL)

{

Node \*aux = curr;

curr = curr->next;

free(aux);

}

root = NULL;

}

**Fișierul includes.h:**

#pragma once

#define MAX\_STRLEN 20

typedef struct Node

{

int data;

struct Node \*next;

} Node;

void create\_list();

void show();

int count();

Node \*get\_node(int index);

void find\_extremes();

void get\_extremes\_interval();

void free\_list();

**Fișierul main.c:**

#include <stdio.h>

#include <stdlib.h>

#include "includes.h"

int main()

{

create\_list();

printf("\nLista:\n");

show();

printf("\n%d de elemente\n", count());

find\_extremes();

printf("\nIntervalul dintre numerele maxim si minim:");

get\_extremes\_interval();

return (0);

}

**Execuția programului:**



**Directoriul task\_2**

**Fișierul code.c:**

#include <stdio.h>

int arr\_ncmp(int \*arr1, int \*arr2, int len)

{

for (int i = 0; i < len; i++)

{

if (arr1[i] != arr2[i])

return (arr1[i] - arr2[i]);

}

return 0;

}

int main()

{

int arr1[] = {/\*Set de date afisat la executie\*/};

int arr2[] = {/\*Set de date afisat la executie\*/};

printf("%d", arr\_ncmp(arr1, arr2, 5));

return (0);

}

**Execuția programului:**

Set de date nr. 1:

int arr1[] = {1, 2, 3, 4, 5};

int arr2[] = {1, 2, 3, 4, 5};



Set de date nr. 2:

int arr1[] = {3, 2, 3, 4, 5};

int arr2[] = {1, 2, 3, 4, 5};



Set de date nr. 3:

int arr1[] = {1, 2, 3, 4, 5};

int arr2[] = {1, 2, 6, 4, 5};



**Concluzii:**

* O listă este o colecție de elemente de informație (noduri), legate între ele prin referințe, realizându-se, astfel, o stocare necontiguă a datelor în memorie. Lungimea unei liste este dată de numărul de noduri din listă.
* Structura corespunzătoare de date trebuie să permită determinarea eficientă a primului sau ultimului nod din structură și care este predecesorul sau succesorul unui nod dat, dacă există.
* O stivă (engleză stack) este o structură de date ale cărei elemente sunt considerate a fi puse unul peste altul, astfel încât orice element adăugat se pune în vârful stivei, iar extragerea unui element se poate face numai din vârful acesteia, în ordinea inversă celei în care elementele au fost introduse, asemănător cu modul în care ai aranja mai multe farfurii una peste alta.
* O coadă este lucrează după principiul FIFO. Variațiile sunt Deque și coadă prioritate

**Surse:**

* infoscience.3x.ro, Cursul de programare in C, Tema: Liste Circulare: <http://infoscience.3x.ro/c++/liste_circulare.htm>
* tutoriale-pe.net, Cursul de programare in C, Tema: Stive:

<https://tutoriale-pe.net/stiva-stack-ce-este-c/>

* tutoriale-pe.net, Cursul de programare in C, Tema: Cozi:

<https://tutoriale-pe.net/coada-queue-ce-este-in-c/>

* wikipedia, Pagina: Tipuri de date:

<https://ro.wikipedia.org/wiki/Tipuri_de_date>

* wikipedia, Pagina: Stivă (structură de date):

<https://ro.wikipedia.org/wiki/Stiv%C4%83_(structur%C4%83_de_date)>

* wikipedia, Pagina: Listă (structură de date):

<https://ro.wikipedia.org/wiki/List%C4%83_(structur%C4%83_de_date)>