Structuri de Date si Algoritmi
Lucrarea de laborator nr. 7

31 martie 2020

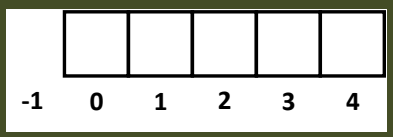
ș.l.dr.ing. Corina Cîmpanu

Cuprins

Coada - Generalități

- **Mecanism de organizare**
  - **FIFO** – **F**irst **I**n **F**irst **O**ut
  - **LILO** – **L**ast **I**n **L**ast **O**ut
- **Operații de bază**
  - InitQueue – inițializează stiva
  - IsEmpty – verifică dacă stiva este goală
  - IsFull – verifică dacă stiva este plină
  - Enqueue – adaugă un element în stivă
  - Dequeue – extrage un element din stivă
  - Front – listează elementul din vârful stivei

1. Coada ordonată circulară
1.1. Structura




```
#pragma once

#include <iostream>
using namespace std;

#define DIM 5

struct Queue{
    int head, tail;
    int* elem;
};

void InitQueue(Queue& q);
int NextPositionHead(int pos);
int NextPositionTail(int pos);
bool IsEmpty(Queue q);
bool IsFull(Queue q);
bool Put(Queue& q, int val);
bool Get(Queue& q, int& val);
bool Front(Queue q, int& val);
void DeleteQueue(Queue& q);
```

1. Coada ordonată circulară
1.2. Inițializarea

```
void InitQueue(Queue& q)
{
    cout << "\n InitQueue();";
    // se va aloca memorie pentru campul elem din structura q
    // se vor initializa campurile head si tail din q cu -1
    cout << "\n Init: head=" << q.head << ", tail=" << q.tail;
}
```

```
#include "Header.h"

int main()
{
    Queue q;
    int val;
    bool ok;

    InitQueue(q);
```

1. Coada ordonată circulară
1.3. Următoarea poziție validă

```
int NextPositionHead(int pos)
{
    // daca pos este mai mic decat ultima pozitie valida din vectorul elem
    // atunci functia va returna urmatoarea pozitie valida
    //altfel functia va returna -1 pentru resetare de la inceputul vectorului
}

int NextPositionTail(int pos)
{
    // daca pos este mai mic decat ultima pozitie valida din vectorul elem
    // atunci functia va returna urmatoarea pozitie valida
    //altfel functia va returna 0 pentru resetare de la inceputul vectorului
}
```

1. Coada ordonată circulară
1.3. Următoarea poziție validă (2)

Indexul	-1	0	1	2	3
Elementul	X	10	20	30	40
Capetele	head				tail

Vom considera că următoarea poziție validă pentru head va fi -1+1=0, respectiv pentru tail va fi 0.

Indexul	-1	0	1	2	3
Elementul	X				
Capetele					head,tail

Vom considera că următoarea poziție validă pentru head va fi -1, respectiv pentru tail va fi 0.

1. Coada ordonată circulară
1.4. Verificarea IsEmpty

Indexul	-1	0	1	2	3
Elementul	X				
Capetele					head,tail

```
bool IsEmpty(Queue q)
{
    //cout << "\n IsEmpty(q); ";
    cout << "\n IsEmpty: head=" << q.head << ", tail=" << q.tail;
    // daca pozitia capului este identica cu pozitia cozii
    // atunci coada nu contine elemente
    // coada contine elemente
}
```

```
    cout << "\n IsEmpty: " << IsEmpty(q);
    cout << "\n IsFull: " << IsFull(q);
```

1. Coada ordonată circulară
1.5. Verificarea IsFull

Indexul	-1	0	1	2	3
Elementul	X	10	20	30	40
Capetele	head				tail

Indexul	-1	0	1	2	3
Elementul	X	10	20	30	40
Capetele		tail	head		

```
bool IsFull(Queue q)
{
    //cout << "\n IsFull(q); ";
    cout << "\n ISFull: head=" << q.head << ", tail=" << q.tail;
    // daca pozitia capului este fix urmatoarea pozitie valida pentru coada sau
    //          pozitia capului este -1 si pozitia cozii este ultima pozitie valida
    // atunci coada este plina
    // coada este goala
}
```

```
    cout << "\n IsEmpty: " << IsEmpty(q);
    cout << "\n IsFull: " << IsFull(q);
```

1. Coada ordonată circulară
1.6. Put

```
bool Put(Queue& q, int val)
{
    //cout << "\n Put(q,"<<val<<"); ";
    // daca coada nu este plina
    // atunci
            cout << "\n Put: Before: head=" << q.head << ", tail=" << q.tail;
    //      mutam indicele cozii pe urmatoarea pozitie valida
    //      pe aceasta poztie voi stoca valoarea val
        cout << "\n Put: After: head=" << q.head << ", tail=" << q.tail;
    // returneaza status operatie succes
    // sf.daca
    // returneaza status operatie esec
}
```

```
    for (int i = 0;i < 8;i++)
        cout << "\n Put(" << i + 1 << "); status: "<<Put(q, i+1);

    cout << "\n IsEmpty: " << IsEmpty(q);
    cout << "\n IsFull: " << IsFull(q);
```

1. Coada ordonată circulară
1.7. Get

```
bool Get(Queue& q, int& val)
{
    //cout << "\n Get(q); ";
    // daca coada nu este goala
            cout << "\n Get: Before: head=" << q.head << ", tail=" << q.tail;
    //      capul cozii va inainta pe urmatoarea pozitie valida
    //      in variabila val vom stoca valoarea din elem de pe pozitia indicata de cap
            cout << "\n Get: After: head=" << q.head << ", tail=" << q.tail;
    //      returneaza status operatie succes
    // sf.daca
    // returneaza status operatie esec
}
```

```
for (int i = 0;i < 8;i++)
{
    ok = Get(q, val);
    cout << "\n Get status: " << ok;
    if (ok)
        cout << " val: " << val;
}

cout << "\n IsEmpty: " << IsEmpty(q);
cout << "\n IsFull: " << IsFull(q);
```

1. Coada ordonată circulară
1.8. Front

```
bool Front(Queue q, int& val)
{
    int aux;
    //cout << "\n Front(q); ";
    // daca coada nu este goala
            cout << "\n Front: Before: head=" << q.head << ", tail=" << q.tail;
    //      vom prelua intr-un auxiliar urmatoarea pozitie valida pentru capul cozii
    //      in variabila val vom stoca valoarea din elem de pe pozitia indicata in auxiliar
            cout << "\n Front: After: head=" << q.head << ", tail=" << q.tail;
    //      returneaza status operatie succes
    // sf.daca
    // returneaza status operatie esec
}
```

```
ok = Front(q, val);
cout << "\n Front status: " << ok;
if (ok)
    cout << " val: " << val;
```

1. Coada ordonată circulară
1.9. Eliberare memorie

```
void DeleteQueue(Queue& q)
{
    cout << "\nDeleteQueue(q)";
    //daca elem este valid
    //atunci se elibereaza memoria pt elem
    //se reinitializeaza cu -1 capul si coada din structura
    cout << "\n DeleteQueue: head=" << q.head << ", tail=" << q.tail;
}
```

```
    DeleteQueue(q);

    system("PAUSE");
    return 0;
}
```

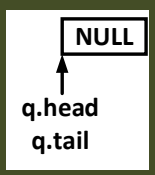
1. Coada ordonată circulară

```
#include "Header.h"

int main()
{
    //declara o coada de tip structura, un intreg valoare si un boolean ok
    //initializeaza coada
    //testeaza daca este goala sau daca este plina
    //adauga MAXDIM+3 elemente in coada si ilustreaza statusul metodei
    //verifica daca coada este goala sau plina
    //verifica ilustrarea elementului din fata cozii si a statusului operatiei
    //extrage MAXDIM+3 elemente din coada si ilustreaza statusul metodei
    //verifica daca coada este goala sau plina
    //verifica ilustrarea elementului din fata cozii si a statusului operatiei
    //adauga din nou MAXDIM+3 elemente si ilustreaza statusul metodei
    //extrage primele 2 elemente
    //adauga inca 2 elemente
    //verifica daca coada este goala sau plina
    //extragele pe toate
    //adauga 5 elemente
    //verifica daca coada este goala sau plina
    //extrage 8 elemente
    //verifica daca coada este goala sau plina
    //elibereaza memoria
```

2. Coada înlănțuită
2.1. Structura

```
struct Elem {
    int data;
    Elem* succ;
};
struct Queue {
    Elem *head, *tail;
};

void InitQueue(Queue& q);
bool IsEmpty(Queue q);
bool Put(Queue& q, int val);
bool Get(Queue& q, int& val);
bool Front(Queue q, int& val);
void DeleteQueue(Queue& q);
```

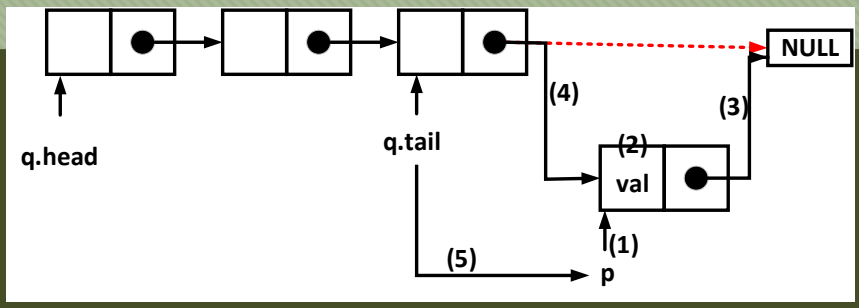
2. Coada înlănțuită
2.2. Inițializarea
2.3. Verificarea IsEmpty

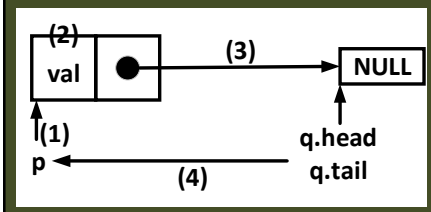



```
void InitQueue(Queue& q)
{
    cout << "\n Initializare coada...";
    //atat primul cat si ultimul element din coada vor indica la nullptr
}


bool IsEmpty(Queue q)
{
    //coada este goala daca primul si ultimul element sunt identice cu nullptr
}
```

2. Coada înlănțuită
2.4. Put

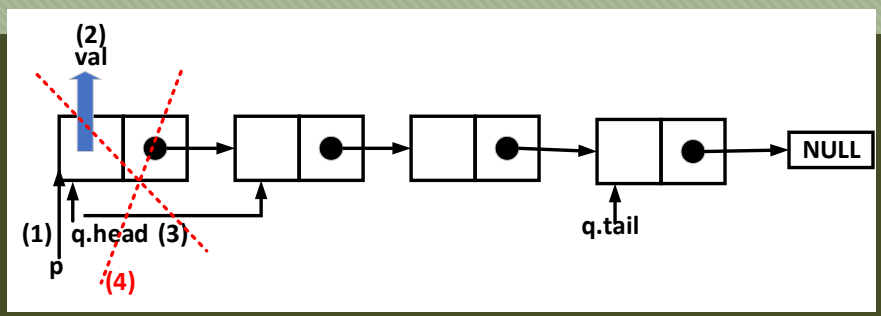
Cazul 1. Coada vida
(1)
(2)
(3)
(4)

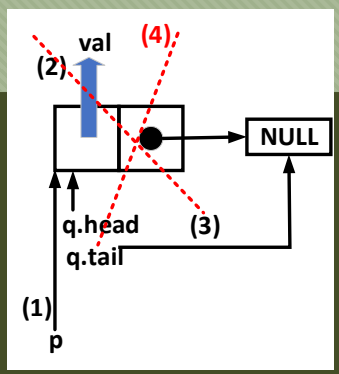




Cazul 2. Coada conține o serie de elemente
(1)
(2)
(3)
(4)
(5)

2. Coada înlănțuită
2.5. Get

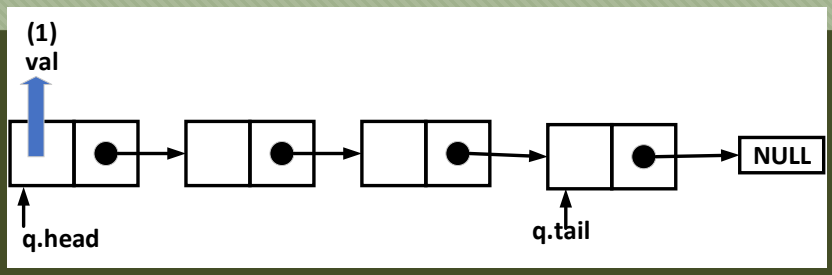


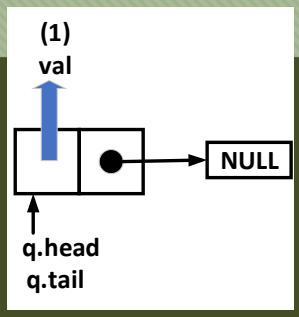


Cazul 1. Coada conține o serie de elemente
(1)
(2)
(3)
(4)

Cazul 2. Coada are 1 element
(1)
(2)
(3')
(3'')
(4)

2. Coada înlănțuită
2.6. Front

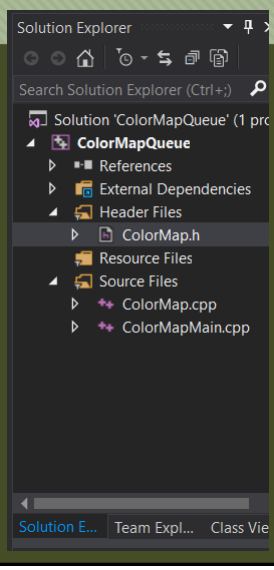


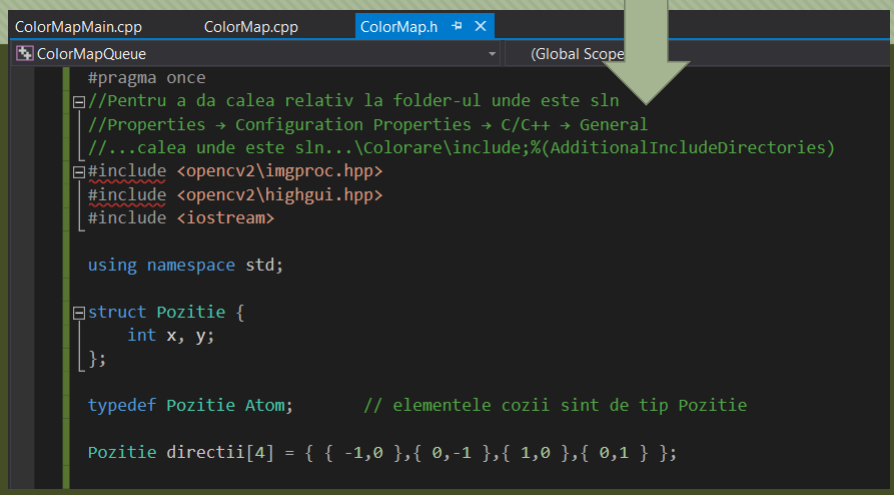


2. Coada înlănțuită
2.7.Main test

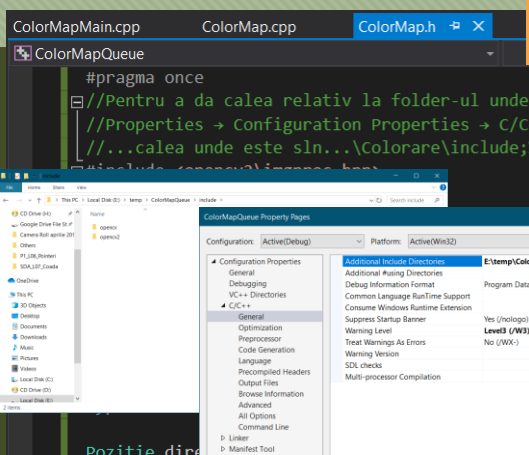
```
//declara o coada de tip structura, un intreg valoare si un boolean ok
//initializeaza coada
//testeaza daca este goala
//adauga MAXDIM+3 elemente in coada si ilustreaza statusul metodei
//verifica daca coada este goala
//verifica ilustrarea elementului din fata cozii si a statusului operatiei
//extrage MAXDIM+3 elemente din coada si ilustreaza statusul metodei
//verifica daca coada este goala
//verifica ilustrarea elementului din fata cozii si a statusului operatiei
//adauga din nou MAXDIM+3 elemente si ilustreaza statusul metodei
//extrage primele 2 elemente
//adauga inca 2 elemente
//verifica daca coada este goala
//extragele pe toate
//adauga 5 elemente
//verifica daca coada este goala
//extrage 8 elemente
//verifica daca coada este goala
//elibereaza memoria
```

3. Colorarea hărții folosind cozi

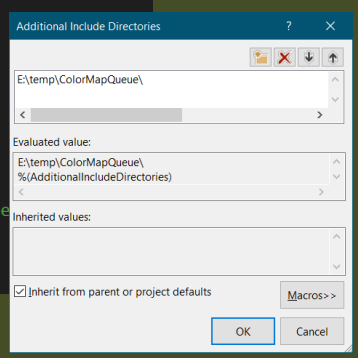




3. Colorarea hărții folosind cozi



Click dreapta pe proiect ColorMapQueue
Properties -> C/C++ -> General



3. Colorarea hărții folosind cozi

```
Pseudocod:
        coloreaza_domeniu(pozitie_initiala)
        {
         memoreaza culoarea domeniului (culoarea pozitiei initiale)
         put(C, pozitie_initiala);
         coloreaza pozitia_initiala;
         WHILE not isEmpty(C) DO
                p := get(C);    // scoate în p urmatoarea pozitie din coada
                FOR pi:=toate pozitiile invecinate lui p DO
                   IF pi nu iese în afara ecranului si
                        pi are culoarea domeniului THEN
                      put(C, pi);
                      coloreaza pi;
        }
```

4. Probleme propuse

- Coada ordonată – implementare
- Coada înlănțuită – implementare
- Colorarea hărtii – indicații in referatul de laborator