L03. STIVE ȘI COZI

ALOCARE STATICA

Header.h

```
#pragma once

#include <iostream>
using namespace std;

#define DIM 5

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

void initQueue(Queue& q);
bool isQEmpty(Queue q);
bool isQFull(Queue q);
bool Enqueue(Queue& q, int val);
bool Dequeue(Queue& q, int& val);
bool Front(Queue q, int& val);
void DeleteQueue(Queue& q);


struct Stack {
	int* elem;
	int sp;
};

void initStack(Stack& s);
bool isSEmpty(Stack s);
bool isSFull(Stack s);
bool Push(Stack& s, int val);
bool Pop(Stack& s, int& val);
bool Top(Stack s, int& val);
void DeleteStack(Stack& s);
```

Functii.cpp

```
#include "Header.h"

//QUEUE
void initQueue(Queue& q)
```

```
{
	q.elem = new int[DIM];
	q.head = q.tail = -1;
}

bool isQEmpty(Queue q)
{
	return q.head == q.tail;
}

bool isQFull(Queue q)
{
	return (q.head == q.tail + 1 || q.head == -1 && q.tail == DIM -
1);
}

bool Enqueue(Queue& q, int val)
{
	if (isQFull(q))
		return false;
	else
	{
		if (q.tail < DIM - 1)
			q.tail++;
		else
			q.tail = 0;
		q.elem[q.tail] = val;
		return true;
	}
}

bool Front(Queue q, int& val)
{
	int aux;
	if (isQEmpty(q))
		return false;
	else
	{
		aux = q.head;
		if (aux < DIM - 1)
			aux++;
		else
			aux = 0;
		val = q.elem[aux];
	}
}

bool Dequeue(Queue& q, int& val)
{
	if (isQEmpty(q))
		return false;
```

```
	else
	{
		if (q.head < DIM - 1)
			q.head++;
		else
			q.head = 0;
		val = q.elem[q.head];
		return true;
	}
}

void DeleteQueue(Queue& q)
{
	if (q.elem)
		delete[] q.elem;
	q.elem = NULL;
	q.head = -1;
	q.tail = -1;
}

//STACK
void initStack(Stack& s)
{
	s.elem = new int[DIM];
	s.sp = -1;
}

bool isSEmpty(Stack s)
{
	return s.sp == -1;
}

bool isSFull(Stack s)
{
	return s.sp == DIM - 1;
}

bool Push(Stack& s, int val)
{
	if (isSFull(s))
		return false;
	else
	{
		s.sp++;
		s.elem[s.sp] = val;
		return true;
	}
}

bool Pop(Stack& s, int& val)
{
```

```
	if (isSEmpty(s))
		return false;
	else
	{
		val = s.elem[s.sp];
		s.sp--;
		return true;
	}
}

bool Top(Stack s, int& val)
{
	if (isSEmpty(s))
		return false;
	else
	{
		val = s.elem[s.sp];
		return true;
	}
}

void DeleteStack(Stack& s)
{
	if (s.elem)
		delete[] s.elem;
	s.elem = NULL;
	s.sp = -1;
}
```

Main.cpp

```
#include "Header.h"

int main()
{
	Queue q;
	Stack s;
	int val, ok, i;

	cout << "\nQUEUE\n";
	initQueue(q);
	cout << "\n\n Init queue: head:" << q.head << " tail:" <<
q.tail;

	ok = isQEmpty(q);
	if (ok)
		cout << "\n\n Coada goala!";

	cout << endl;
	for (i = 0;i < DIM + 2;i++)
```

```
	{
		cout << "\n Insert queue: head:" << q.head << " tail:" <<
q.tail;
		ok = Enqueue(q, i + 1);
		if (!ok)
			cout << "\nFull queue. Insert " << i + 1;
	}

	ok = isQFull(q);
	if (ok)
		cout << "\n\n Full queue!";

	ok = Front(q, val);
	if (ok)
		cout << "\n\n Front queue: elem:" << val << " head:" <<
q.head << " tail:" << q.tail;

	cout << endl;
	for (i = 0;i < DIM + 2;i++)
	{
		cout << "\n Delete queue: head:" << q.head << " tail:" <<
q.tail;
		ok = Dequeue(q, val);
		if (ok)
			cout << "\n Pop " << val << " from queue.";
		else
			cout << "\n Empty queue. Pop!";
	}
	cout << "\n queue: head:" << q.head << " tail:" << q.tail;

	cout << "\n\n Free mem...";
	DeleteQueue(q);
	cout << "\n queue: head:" << q.head << " tail:" << q.tail;
	cout << endl;


	cout << endl << endl;
	cout << "\nSTACK:\n";

	initStack(s);
	cout << "\n\nInit stack: sp:" << s.sp;

	ok = isSEmpty(s);
	if (!ok)
		cout << "\n Stack empty sp:"<<s.sp;

	cout << endl;
	for (i = 0;i < DIM + 2;i++)
	{
		cout << "\n Push " << i + 1 << " sp:" << s.sp;
		ok = Push(s, i + 1);
```

```
            if (!ok)
                  cout << "\nFull stack " << i + 1;
      }

      ok = isSFull(s);
      if (ok)
            cout << "\n\nFull stack. sp:" << s.sp;

      ok = Top(s, val);
      if (ok)
            cout << "\n\nTop: val:" << val << " sp:" << s.sp;

      cout << endl;
      for (i = 0;i < DIM + 2;i++)
      {
            cout << "\n Pop sp:" << s.sp;
            ok = Pop(s, val);
            if (!ok)
                  cout << "\nEmpty stack sp:" << s.sp;
      }

      cout << "\n\n Free mem...";
      DeleteStack(s);
      cout << "\n stack: sp:" << s.sp;
      cout << endl;

      return 0;
}
```

ALOCARE DINAMICA

Header.h

```
#pragma once

#include <iostream>
using namespace std;

struct Elem {
	int data;
	Elem* succ;
};

void Init(Elem*& e);
bool isEmpty(Elem* e);
void List(Elem* e);
void List1(Elem* e);

bool Push(Elem*& stack, int val);
bool Pop(Elem*& stack, int& val);
bool Top(Elem* stack, int& val);


bool Enqueue(Elem*& queue, int val);
bool Dequeue(Elem*& queue, int& val);
bool Front(Elem* queue, int& val);
void ListQueue(Elem* queue);
```

Functii.cpp

```
#include "Header.h"

void Init(Elem*& e)
{
	e = NULL;
}

bool isEmpty(Elem* e)
{
	return e == NULL;
}

void List(Elem* e)
{
	if (e)
	{
		List(e->succ);
		cout << e->data << " ";
```

```
	}
	else
		cout << endl;
}

void List1(Elem* e)
{
	if (e)
	{
		cout << e->data << " ";
		List(e->succ);
	}
	else
		cout << endl;
}

bool Push(Elem*& stack, int val)
{
	cout << "\n Insert " << val << " to stack.";
	Elem* p = new Elem;
	p->data = val;
	p->succ = stack;
	stack = p;
	return true;
}

bool Pop(Elem*& stack, int& val)
{
	if (isEmpty(stack))
		return false;
	else
	{
		Elem* p = stack;
		val = p->data;
		cout << "\n Pop " << val << " from stack.";
		stack = stack->succ;
		delete p;
		p = NULL;
		return true;
	}
}

bool Top(Elem* stack, int& val)
{
	if (isEmpty(stack))
		return false;
	else
	{
		val = stack->data;
		cout << "\n Top " << val << " in stack.";
		return true;
```

```
	}
}

bool Enqueue(Elem*& queue, int val)
{
	Elem* p = new Elem;
	p->data = val;
	cout << "\n Push " << val << " to queue.";
	p->succ = queue;
	queue = p;
	return true;
}

bool Dequeue(Elem*& queue, int& val)
{
	if (isEmpty(queue))
		return false;
	else
	{
		Elem* p = queue;
		if (p->succ)
		{
			Elem* q;
			while (p->succ)
			{
				q = p;
				p = p->succ;
			}
			q->succ = NULL;
		}
		else
			queue = NULL;
		val = p->data;
		delete p;
		return true;
	}
}

bool Front(Elem* queue, int& val)
{
	if (isEmpty(queue))
		return false;
	else
	{
		Elem* p = queue;
		while (p->succ)
			p = p->succ;
		val = p->data;
		return true;
	}
}
```

Main.cpp

```
#include "Header.h"

int main()
{
	int i, val, ok;
	Elem* stack;
	Elem* queue;

	cout << "\nSTACK:\n";
	cout << "\n InitStack:";
	Init(stack);

	ok = isEmpty(stack);
	if (ok)
		cout << "\n\nEmpty stack.";

	cout << "\n\nPush to stack:";
	for (i = 0;i < 5;i++)
		ok = Push(stack, i + 1);

	cout << "\n\nTop:";
	ok = Top(stack, val);
	if (ok)
		cout << "\n Top value " << val << " from stack.";


	cout << "\n\nStack (bottom->top): ";
	List(stack);

	cout << "\n\nStack (top->bottom): \n";
	List1(stack);
	cout << endl;

	cout << endl << endl << "Pop:";
	for (i = 0;i < 10;i++)
	{
		ok = Pop(stack, val);
		if (!ok)
			cout << "\n Empty stack.";
	}

	cout << "\n\nStack (bottom->top): ";
	List(stack);
	cout << endl << endl;

	stack = NULL;
```

```
	cout << "\n\nQUEUE:\n";
	Init(queue);

	ok = isEmpty(queue);
	if (ok)
		cout << "\n\nEmpty queue.";

	cout << "\n\n Push to queue";
	for (i = 0;i < 7;i++)
		ok = Enqueue(queue, i+1);

	cout << "\n\n Queue (head->tail): ";
	List(queue);

	cout << "\n\n Front:";
	ok = Front(queue, val);
	if (ok)
		cout << "\n Front " << val << " from queue.";


	cout << "\n\nPop from queue";
	for (i = 0;i < 9;i++)
	{
		ok = Dequeue(queue, val);
		if (!ok)
			cout << "\n Empty queue.";
		else
			cout << "\n Pop " << val << " from queue.";
	}

	queue = NULL;

	return 0;
}
```