**Лабораторная работа №1**

**Линейные списки**

Выполнить задания, используя для пpедставления очеpедей и стеков:

а) массивы; б) динамические списки.

Требования к программам:

1. Количество элементов исходных линейных списков заранее не определено и задается случайным образом. При дальнейшей обработке считается, что количество элементов списка не известно, т.е. обработка производится, пока не достигнут конец списка.

2. Программа должна сформировать исходные линейные списки, вывести их содержимое на экран (при этом данные из списков не должны быть потеряны), произвести обработку и вывести содержимое итогового списка на экран.

17 вариант

Даны очеpедь и стек целых чисел. Из элементов стека меньших минимального элемента очеpеди сфоpмиpовать новый стек.

**Текст программы**

а) с использованием массивов:

#include <iostream>

using namespace std;

//Стек

struct Stack1 { //определение стека

int top;

int\* data;

int capacity;

};

void InitStack(Stack1& st, int capacity) { //инициализация стека

st.data = new int[capacity];

st.top = -1;

st.capacity = capacity;

}

bool empty(Stack1& st) { //проверка стека на пустоту

return st.top == -1;

}

void push(Stack1& st, int value, int& err) { //добавление элемента в стек

if (st.top + 1 >= st.capacity) err++;

else st.data[++st.top] = value;

}

int pop(Stack1& st) { //взятие и удаление элемента из стека

if (!empty(st)) return st.data[st.top--];

else return 0;

}

void nullStack(Stack1& st) { //обнуление стека

st.top = -1;

}

void print(Stack1& st, int& err) { //вывод стека

if (!empty(st) && err == 0)

for (int i = 0; i <= st.top; i++)

cout << st.data[i] << ' ';

else if (empty(st) && err == 0) cout << "Stack is empty";

if (err != 0) cout << "Stack overflow";

cout << endl;

}

//Очередь

struct Queue1 { //определение очереди

int head, tail, size;

int\* data;

};

void nullQueue(Queue1& q) { //обнуление очереди

q.head = 0;

q.tail = q.size - 1;

}

void InitQueue(Queue1& q, int capacity) { //инициализация очереди

q.size = capacity + 1;

q.data = new int[q.size];

nullQueue(q);

}

int next(Queue1& q, int n) { //вычисление следующего индекса

return (n + 1) % q.size;

}

bool empty(Queue1& q) { //проверка очереди на пустоту

return next(q, q.tail) == q.head;

}

void add(Queue1& q, int value, int& err) { //добавление элемента в очередь

if (next(q, next(q, q.tail)) == q.head) err++; //проверка очереди на переполнение

else {

q.tail = next(q, q.tail);

q.data[q.tail] = value;

}

}

int del(Queue1& q) { //взятие и удаление элемента из очереди

if (empty(q)) return 0;

else {

int d = q.data[q.head];

q.head = next(q, q.head);

return d;

}

}

void print(Queue1& q, int& err) { //вывод очереди

if (!empty(q) && err == 0)

for (int i = q.head; i <= q.tail; i++)

cout << q.data[i] << ' ';

else if (empty(q) && err == 0) cout << "Queue is empty";

if (err != 0) cout << "Queue overflow";

cout << endl;

}

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(0));

Stack1 st;

int capacity = rand() % 20 + 1, err1 = 0;

InitStack(st, capacity);

for (int i = 0; i < capacity; i++) {

push(st, rand() % 100 - 50, err1);

}

cout << "Исходный стек: ";

print(st, err1);

Queue1 q;

int capacity2 = rand() % 20 + 1, err2 = 0;

InitQueue(q, capacity2);

for (int i = 0; i < capacity2; i++) {

add(q, rand() % 100 - 30, err2);

}

cout << "Исходная очередь: ";

print(q, err2);

int a, min = q.data[q.tail];

if (!empty(q) && err1 == 0 && err2 == 0) {

while (!empty(q)) {

a = del(q);

if (a < min) min = a;

}

cout << "Минимальный элемент очереди: " << min << endl;

Stack1 st2;

InitStack(st2, capacity);

int err3 = 0;

while (!empty(st)) {

a = pop(st);

if (a < min) push(st2, a, err3);

}

if (empty(st2)) cout << "Таких элементов нет!" << endl;

else {

cout << "Полученный стек: ";

print(st2, err3);

}

delete[] st2.data;

}

delete[] st.data;

delete[] q.data;

system("pause");

return 0;

}

б) с использованием динамических списков:

#include <iostream>

using namespace std;

//Стек

struct Node { //узел стека

int data;

Node\* next;

};

void InitStack(Node\*& top) { //инициализация стека

top = NULL;

}

void push(Node\*& top, int value) { //добавление элемента в стек

Node\* tmp = new Node;

tmp->next = top;

top = tmp;

top->data = value;

}

bool empty(Node\*& top) { //проверка стека на пустоту

return top == NULL;

}

int pop(Node\*& top) { //взятие и удаление элемента и стека

if (!empty(top)) {

Node\* tmp = top;

int d = top->data;

top = top->next;

delete(tmp);

return d;

}

else return 0;

}

void nullStack(Node\*& top) { //обнуление стека

Node\* tmp;

while (!empty(top)) {

tmp = top;

top = top->next;

delete(tmp);

}

}

void print(Node\*& top) { //вывод стека

if (!empty(top)) {

Node\* tmp = top;

while (tmp != NULL) {

cout << tmp->data << " ";

tmp = tmp->next;

}

cout << endl;

delete(tmp);

}

else cout << "Stack is empty" << endl;

}

//Очередь

class Queue {

private:

struct Node { //узел очереди

int data;

Node\* next;

};

Node\* head;

Node\* tail;

public:

Queue() { //инициализация очереди

head = NULL;

tail = NULL;

}

bool empty() { //проверка очереди на пустоту

return head == NULL;

}

void add(int value) { //добавление элемента в очередь

if (empty()) { //если она пустая

head = new Node;

head->data = value;

head->next = NULL;

tail = head;

}

else { //если она не пустая

tail->next = new Node;

tail = tail->next;

tail->data = value;

tail->next = NULL;

}

}

int del() { //взятие и удаление элемента из очереди

if (!empty()) {

int d = head->data;

Node\* tmp = head;

head = head->next;

delete(tmp);

return d;

}

}

void nullQueue() { //обнуление очереди

Node\* tmp;

while (!empty()) {

tmp = head;

head = head->next;

delete(tmp);

}

}

void print() { //вывод очереди

if (!empty()) {

Node\* tmp = head;

while (tmp != NULL) {

cout << tmp->data << ' ';

tmp = tmp->next;

}

cout << endl;

delete(tmp);

}

else cout << "Queue is empty" << endl;

}

};

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(0));

Node\* st = NULL;

InitStack(st);

int capacity = rand() % 20 + 1;

for (int i = 0; i < capacity; i++) {

push(st, rand() % 100 - 50);

}

cout << "Исходный стек: ";

print(st);

Queue q;

int capacity2 = rand() % 20 + 1;

for (int i = 0; i < capacity2; i++) {

q.add(rand() % 100 - 30);

}

cout << "Исходная очередь: ";

q.print();

int min, a = 0;

if (!q.empty()) {

min = q.del();

while (!q.empty()) {

a = q.del();

if (a < min) min = a;

}

cout << "Минимальный элемент очереди: " << min << endl;

Node\* st2 = NULL;

InitStack(st2);

while (!empty(st)) {

a = pop(st);

if (a < min) push(st2, a);

}

if (empty(st2)) cout << "Таких элементов нет!";

else {

cout << "Полученный стек: ";

while (!empty(st2)) {

a = pop(st2);

cout << a << ' ';

}

}

}

cout << endl;

system("pause");

return 0;

}

**Описание программы**

а) с использованием массивов:

В начале программы задаются функции для работы со стеками и очередями:

* **struct Stack1** – определение стека
* **InitStack** – инициализация стека
* **empty** – проверка стека на пустоту
* **Push** – добавление элемента в стек
* **Pop** – взятие и удаление элемента из стека
* **nullStack** – обнуление стека
* **print** – вывод стека
* **struct Queue1** – определение очереди
* **nullQueue** – обнуление очереди
* **InitQueue** – инициализация очереди
* **Next** – вычисление следующего индекса
* **empty** – проверка очереди на пустоту
* **Add** – добавление элемента в очередь
* **Del** – взятие и удаление элемента из очереди
* **print** – вывод очереди

Содержание функции main:

1. Определяются и инициализируются переменные **capacity** (ее значение задается рандомно) и **err1** = 0
2. Определяется и инициализируется стек **st** размером **capacity**
3. Стек **st** заполняется рандомными числами
4. Стек **st** выводится на экран
5. Определяется и инициализируется очередь **q** размером **capacity**
6. Определяются и инициализируются переменные **capacity2** (ее значение задается рандомно) и **err2** = 0
7. Очередь **q** заполняется рандомными числами
8. Очередь **q** выводится на экран
9. Определяются и инициализируются переменные **a** и **min**. **min** присваивается значение **q.data[q.tail]**
10. Если переменные err1 и err2 равны нулю (т.е. стек **st** и очередь **q** не переполнены), пока очередь **q** не станет пустой, из нее берется число и присваивается переменной **а** (соответственно, из очереди оно удаляется). Далее значение переменной **a** сравнивается со значением переменной **min**. Если **a < min**, то переменной **min** присваивается значение переменной **а**
11. Значение переменной **min** (минимальный элемент очереди) выводится на экран (если очередь не пустая)
12. Определяется и инициализируется стек **st2** размером **capacity**
13. Пока стек **st** не пуст, из него берется число и присваивается переменной **a** (соответственно, из стека оно удаляется). Далее значение переменной **а** сравнивается со значением переменной **min** (минимальным значением очереди **q**). И если **a < min**, то значение переменной **a** записывается в стек **st2**
14. Стек **st2** проверяется на пустоту. Если он пуст – на экран выводится фраза «*Таких элементов нет!*». Иначе на экран выводятся все элементы стека **st2**
15. Динамические массивы **st.data**, **st2.data** и **q.data** удаляются

б) с использованием динамических списков:

В начале программы задаются функции для работы со стеками и очередями:

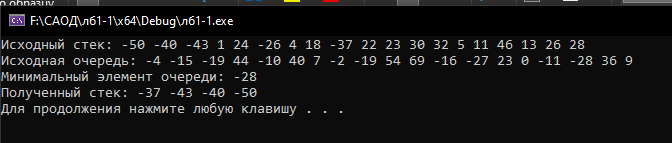
* **struct Node** – узел стека
* **InitStack** – инициализация стека
* **Push** – добавление элемента в стек
* **empty** – проверка стека на пустоту
* **Pop** – взятие и удаление элемента из стека
* **nullStack** – обнуление стека
* **print** – вывод стека
* **struct Node** – узел очереди
* **Queue** – инициализация очереди
* **empty** – проверка очереди на пустоту
* **Add** – добавление элемента в очередь
* **Del** – взятие и удаление элемента из очереди
* **nullQueue** – обнуление очереди
* **print** – вывод очереди

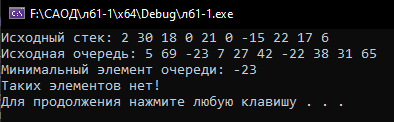
Содержание функции main:

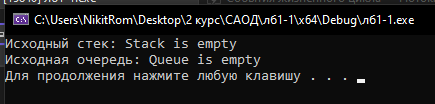
1. Определяется и инициализируется стек **st**
2. Определяется и инициализируется переменная **capacity**, ее значение задается рандомно
3. Стек **st** заполняется рандомными числами в количестве **capacity**
4. Стек **st** выводится на экран
5. Определяется и инициализируется очередь **q**
6. Очередь **q** заполняется рандомными числами в количестве **capacity**
7. Очередь **q** выводится на экран
8. Определяются и инициализируются переменные **min**, **a**, им присваиваются начальные значения: **min = q.del()**, **a = 0**
9. Если очередь **q** не пуста, пока она не станет пустой, из нее берется число и присваивается переменной **a** (соответственно, из очереди оно удаляется). После этого оно сравнивается со значением **min**. Если **a < min**, то переменной **min** присваивается значение переменной **а**
10. Значение переменной **min** (минимальный элемент очереди) выводится на экран (если очередь не пустая)
11. Определяется и инициализируется стек **st2**
12. Пока стек **st** не пуст, из него берется число и присваивается переменной **a** (соответственно, из стека оно удаляется). Далее значение переменной **а** сравнивается со значением переменной **min** (минимальным значением очереди **q**). И если **a < min**, то значение переменной **a** записывается в стек **st2**
13. Стек **st2** проверяется на пустоту. Если он пуст – на экран выводится фраза «*Таких элементов нет!*». Иначе на экран выводятся все элементы стека **st2**

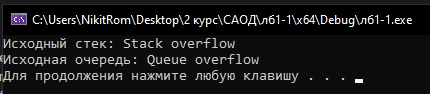
**Результаты тестов**

а) с использованием массивов

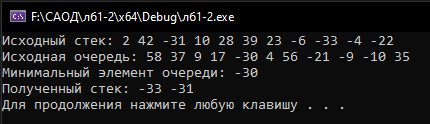


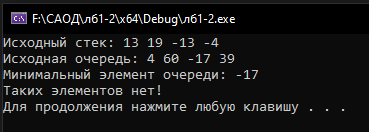


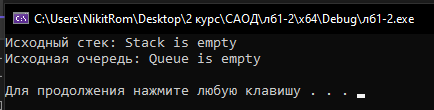




б) с использованием динамических списков:







**Вывод**

Мною была написана программа с использованием линеных списков – стеков и очередей.

Они были представлены 2 способами: с помощью массива и с помощью динамического списка.

Программа формирует исходные линейные списки (стек **st** и очередь **q**), выводит их содержимое на экран и затем производит обработку: находит минимальный элемент очереди **q** и записывает в стек **st2** все элементы стека **st**, которые меньше этого минимального элемента. Далее программа выводит содержимое итогового стека на экран.