**Лабоpатоpная pабота №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.

**Вариант 5**

Даны два стека целых чисел. Сфоpмиpовать очеpедь из элементов пеpвого

стека, кpатных минимуму втоpого.

**Код a)**

#define \_USE\_MATH\_DEFINES

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cmath>

#include <iostream>

#include <string>

#include <ctype.h>

#include <time.h>

#include <Windows.h>

#include <cstring>

#include <ctime>

#include <cstdlib>

using namespace std;

struct Stack1 {

int top;

int\* data;

};

void initStack(Stack1 &st, int capacity) {

st.data = new int[capacity];

st.top = -1;

}

void push(Stack1 &st, int value) {

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

}

int pop(Stack1 &st) {

return st.data[st.top--];

}

void nullStack(Stack1 &st) {

st.top = -1;

}

bool empty(Stack1 &st) {

return st.top == -1;

}

struct Queue1 {

int head, tail, size;

int\* data;

};

void nullQueue(Queue1 &q){

q.head = 0; q.tail = q.size - 1;

}

void initQueue(Queue1 &q, int capacity) {

q.size = capacity + 1;

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

nullQueue(q);

}

int next(Queue1 &q, int n) {

return (n + 1) % q.size;

}

bool empty(Queue1 &q) {

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

}

void add(Queue1 &q, int value) {

if (next(q, next(q, q.tail)) == q.head)

cout << "Queue overflow" << endl;

else {

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

q.data[q.tail] = value;

}

}

int del(Queue1 &q) {

if (empty(q)) {

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

return 0;}

else {

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

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

return d;

}

}

int main() {

setlocale(LC\_ALL, "rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

int g=10; //ограничитель для rand

int r;

int min;

int size1 = rand() % g;

int size2 = rand() % g;

cout << size1 << " " << size2 << endl;

Stack1 st3; // стэки для выводов

initStack(st3, size1+1);

Stack1 st4;

initStack(st4, size1 + 1);

Stack1 st1; //инициализация стэков

initStack(st1, size1+1);

for (int i = 0; i < size1; i++) {

r = rand()%g;

push(st1, r);

}

Stack1 st2;

initStack(st2, size2 + 1);

for (int i = 0; i < size2; i++) {

r = rand() % g;

push(st2, r);

}

cout << "первый стек: " << endl; //вывод содержимого стэков на экран

while (!empty(st1)) {

int x;

x = pop(st1);

cout << x << " ";

push(st3, x);

}

cout << endl;

while (!empty(st3)) {

int x;

x = pop(st3);

push(st1, x);

}

cout << "второй стек: " << endl;

while (!empty(st2)) {

int x;

x = pop(st2);

cout << x << " ";

push(st4, x);

}

cout << endl;

while (!empty(st4)) {

int x;

x = pop(st4);

push(st2, x);

}

min = pop(st2); // находим минимальный элемент второго стэка

push(st2, min);

while (!empty(st2)) {

int x;

x = pop(st2);

if (x < min) { min = x; };

push(st4, x);

}

cout << endl;

while (!empty(st4)) {

int x;

x = pop(st4);

push(st2, x);

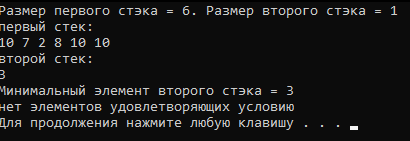
}

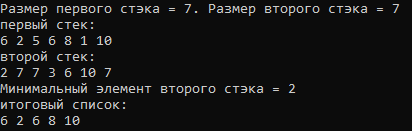
cout << min << endl;

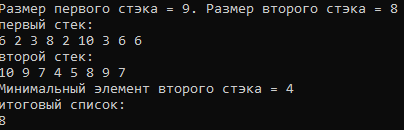
system("pause");

return 0;}

**Тесты**







**Алгоритм**

1. Определение структуры **Stack1** для стека и определение структуры **Queue1** для очереди.

2. Объявление переменных, которые понадобятся в дальнейшем: **q** для подсчета элементов, **g** ограничитель для функции **rand**, **r** для храния значений получаемых с помощью функции **rand**, **min** для поиска минимального элемента, **size1** и **size2** принимающие рандомные значения для определения размеров исходных линейных списков.

3. Объявление и инициализация стека **st1** через **InitStack**.

4. Заполнение стека **st1** случайными значениями с использованием функции **push**.

5. Объявление и инициализация стека **st2** через **InitStack**.

6. Заполнение стека **st2** случайными значениями с использованием функции **push**.

7. Создание **tmp1** копии стэка **st1** и вывод его содержимого на экран через цикл используя функию **pop**.

8. Создание **tmp1** копии стэка **st2** и вывод его содержимого на экран через цикл используя функию **pop**.

9. Поиск элемента с минимальным значением в стэке **st2** (присвоение переменной **min** значения последнего элемента стэка, создание **tmp1** копии стэка **st2** и поиск элемента через цикл используя функию **pop**).

10. Определение количества элементов в стэке **st1** кратных элементу с минимальным значением стэка **st2** (объявление переменной **q**, создание **tmp1** копии стэка **st1** и подсчет элементов через цикл используя функию **pop**).

11. Объявление очереди **Que1** типа **Queue1**.

12. Объявление и инициализация очереди **Que1** через **InitQueue**.

13. Заполнение очерди **Que1** элементами удовлетворяющими условию (создание **tmp1** копии стэка **st1** поиск элементов через цикл с использованием функции **pop** с последующим добавлением их в очередь с помощью функии **add**).

14. Вывод содержимого итогового списка (очереди) **Que1** на экран (создание t**mp2** копии очереди **Que1** и вывод элементов через цикл с импользованием функции **del**).

15. Завершение программы.

**Код б)**

Header.h (создание класса Queue, объявление структуры и методов класса)

#pragma once

class Queue

{

private:

struct Node1

{

int data;

Node1\* next;

};

Node1\* head, \* tail;

public:

Queue();

bool Empty();

void Add(int value);

int Get();

int Del();

void PrintQueue();

void NullQueue();

};

Source1.cpp (определение методов класса Queue)

#include "Header.h"

#include <iostream>

Queue::Queue() {

head = NULL;

tail = NULL;

}

bool Queue::Empty() {

return head == NULL;

}

void Queue::Add(int value) {

if (Empty()) {

**head = new Node1;**

**head->data = value;**

**head->next = NULL;**

**tail = head;**

}

else {

tail->next = new Node1;

tail = tail->next;

tail->data = value;

tail->next = NULL;

}

}

int Queue::Del() {

if (Empty()) {

std::cout << "Queue is empty" << std::endl; return 0;

}

else {

Node1\* d = head;

head = head->next;

delete(d);

}

}

int Queue::Get() {

if (Empty()) {

std::cout << "Queue is empty" << std::endl;

return 0;

}

else {

int r\_data = head->data;

Node1\* tmp = head;

head = head->next;

delete tmp;

return r\_data;

}

}

void Queue::PrintQueue() {

Node1\* current = head;

while (current != NULL) {

std::cout << current->data << std::endl;

current = current->next;

}

}

void Queue::NullQueue() {

Node1\* tmp;

while (!Empty()) {

tmp = head;

head = head->next;

delete(tmp);

}

}

Source.cpp (основной код)

#define \_USE\_MATH\_DEFINES

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cmath>

#include <iostream>

#include <string>

#include <ctype.h>

#include <time.h>

#include <Windows.h>

#include <cstring>

#include <ctime>

#include <cstdlib>

#include<set>

#include <algorithm>

#include "Header.h"

using namespace std;

struct Node {

int data;

Node\* next;

};

void InitStack(Node\*& top) {

top = NULL;

}

void push(Node\*& top, int value) {

Node\* tmp = new Node;

tmp->next = top;

top = tmp;

top->data = value;

}

int pop(Node\*& top) {

Node\* tmp = top;

int d = top->data;

top = top->next;

delete(tmp);

return d;

}

bool empty(Node\*& top) {

return top == NULL;

}

void nullStack(Node\*& top) {

Node\* tmp;

while (!empty(top)) {

tmp = top;

top = top->next;

delete(tmp);

}

}

void stcpy(Node\*& sttocpy, Node\*& cpy) {

Node\* tmp;

InitStack(tmp);

while (!empty(sttocpy)) {

int x;

x = pop(sttocpy);

push(cpy, x);

push(tmp, x);

}

while (!empty(tmp)) {

int x;

x = pop(tmp);

push(sttocpy, x);

}

}

int main() {

setlocale(LC\_ALL, "rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

int q = 0;

int g = 10; //ограничитель для rand

int r;

int min;

int size1 = rand() % g + 1;

int size2 = rand() % g + 1;

cout << "Размер первого стэка = " << size1 << ". Размер второго стэка = " << size2 << endl;

Node\* st1; //инициализация стэков

InitStack(st1);

for (int i = 0; i < size1; i++) {

r = rand() % g + 1;

push(st1, r);

}

Node\* st2;

InitStack(st2);

for (int i = 0; i < size2; i++) {

r = rand() % g + 1;

push(st2, r);

}

Node\* tmp1;

InitStack(tmp1);

cout << "первый стек: " << endl; //вывод содержимого стэков на экран

stcpy(st1, tmp1);

while (!empty(tmp1)) {

int x;

x = pop(tmp1);

cout << x << " ";

}

cout << endl;

cout << "второй стек: " << endl;

stcpy(st2, tmp1);

while (!empty(tmp1)) {

int x;

x = pop(tmp1);

cout << x << " ";

}

cout << endl;

stcpy(st2, tmp1);

min = pop(tmp1); // находим минимальный элемент второго стэка

push(tmp1, min);

while (!empty(tmp1)) {

int x;

x = pop(tmp1);

if (x < min) { min = x; };

}

cout << "Минимальный элемент второго стэка = " << min << endl;

stcpy(st1, tmp1); //находим количество элементов первого стэка кратных минимуму второго

while (!empty(tmp1)) {

int x;

x = pop(tmp1);

if (x % min == 0) { q = q + 1; };

}

if (q == 0) {

cout << "нет элементов удовлетворяющих условию" << endl;

system("pause");

return 0;

}

Queue Que1; //инициализируем очередь

stcpy(st1, tmp1); //записываем в очередь элементы первого стэка кратные минимуму второго

while (!empty(tmp1)) {

int x;

x = pop(tmp1);

if (x % min == 0) { Que1.Add(x); };

}

Queue tmp2 = Que1; //выводим содержимое итогового списка

cout << "итоговый список: " << endl;

while (!tmp2.Empty()) {

int x;

x = tmp2.Get();

cout << x << " ";

}

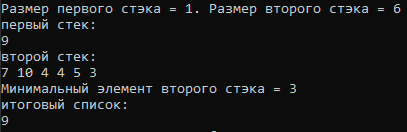
cout << endl;

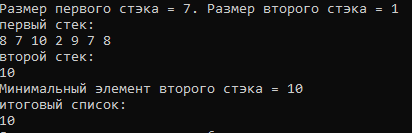
system("pause");

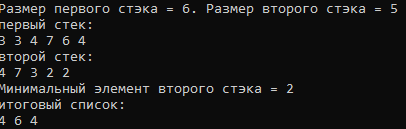
return 0;

}

**Тесты**







**Алгоритм**

1. Определение структуры **Node** для стека и определение класса **Queue** для очереди.

2. Cоздание фунции **stcpy** копирующую содержимое стэка для последующей его обработки без потери данных(фунция принимает на ввод две структуры: которую нужно скопировать **sttocpy** и в которую нужно скопировать **cpy**, объявляет и инициализирует структуру **tmp** через которую будет производиться копирование, через цикл с использованием функций **pop** и **push** перебирает элементы структуры **sttocpy** и записывает их в структуры **cpy** и **tmp**, через цикл с использованием функций **pop** и **push** перебирает элементы структуры **tmp** и записывает их обратно в структуру **sttocpy**).

3. Объявление переменных, которые понадобятся в дальнейшем: **q** для подсчета элементов, **g** ограничитель для функции **rand**, **r** для храния значений получаемых с помощью функции **rand**, **min** для поиска минимального элемента, **size1** и **size2** принимающие рандомные значения для определения размеров исходных линейных списков.

4. Объявление стеков \***st1** и \***st2** типа **Node**, их инициализация через **InitStack** и заполнение случайными значениями с использованием функции **push**.

5. Объявление стека **tmp1** типа **Node** и его инициализация через **InitStack**

6. Создание **tmp1** копии стэка **st1** и вывод его содержимого на экран через цикл используя функию **pop**.

7. Создание **tmp1** копии стэка **st2** и вывод его содержимого на экран через цикл используя функию **pop**.

8. Поиск элемента с минимальным значением в стэке **st2** (объявление переменной **min**, присвоение ей значения последнего элемента стэка, создание **tmp1** копии стэка **st2** и поиск элемента через цикл используя функию **pop**).

9. Определение количества элементов в стэке **st1** кратных элементу с минимальным значением стэка **st2** (объявление переменной **q**, создание **tmp1** копии стэка **st1** и подсчет элементов через цикл используя функию **pop**).

10. Объявление объекта **Que1** класса **Queue**.

12. Заполнение очерди **Que1** элементами удовлетворяющими условию (создание **tmp1** копии стэка **st1** поиск элементов через цикл с использованием функции **pop** с последующим добавлением их в очередь с помощью функии **add**).

13. Вывод содержимого итогового списка (очереди) **Que1** на экран (создание t**mp2** копии очереди **Que1** и вывод элементов через цикл с импользованием функции **Get**).

14. Завершение программы.