**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.

Вариант 12

12. Даны стек и очеpедь целых чисел от 0 до 1000. Из общих элементов

сфоpмиpовать новый стек.

А) реализация через массивы:

#include <iostream>

#include<time.h>

#include<set>

#include <algorithm>

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, int capacity) {

if (st.top == capacity) {

cout << "Stack overflow" << endl;

}

else

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;

}

void PrintStack(Stack1& st) {

Stack1 tmp = st;

while(tmp.top!=-1)

cout << pop(tmp) << endl;

cout << endl;

}

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;

}

}

void PrintQueue(Queue1& q) {

for (int i = q.head; i != next(q, q.tail); i = next(q, i))

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

cout << endl;

}

int main() {

srand(time(NULL));

Stack1 st1, st2;

int capacity = 1 + rand() % 100;

InitStack(st1, capacity);

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

push(st1, rand() % 1000, capacity);

}

PrintStack(st1);

Queue1 Que1;

capacity = 1 + rand() % 100;

InitQueue(Que1, capacity);

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

add(Que1, rand() % 1000);

PrintQueue(Que1);

set<int> num1;

while (st1.top != -1) {

num1.insert(pop(st1));

}

nullStack(st1);

set<int> num2;

for (int i = Que1.head; i != next(Que1, Que1.tail); i = next(Que1, i))

num2.insert(Que1.data[i]);

nullQueue(Que1);

for (int n : num1)

cout << n << " ";

cout << endl;

cout <<"=============================="<< endl;

for (int n : num2)

cout << n << " ";

cout << endl;

cout << "==============================" << endl;

set<int> num3;

set\_intersection(num1.begin(), num1.end(),

num2.begin(), num2.end(),

inserter(num3, num3.begin()));

capacity = num3.size();

InitStack(st2, capacity);

for (int n : num3)

push(st2, n, capacity);

PrintStack(st2);

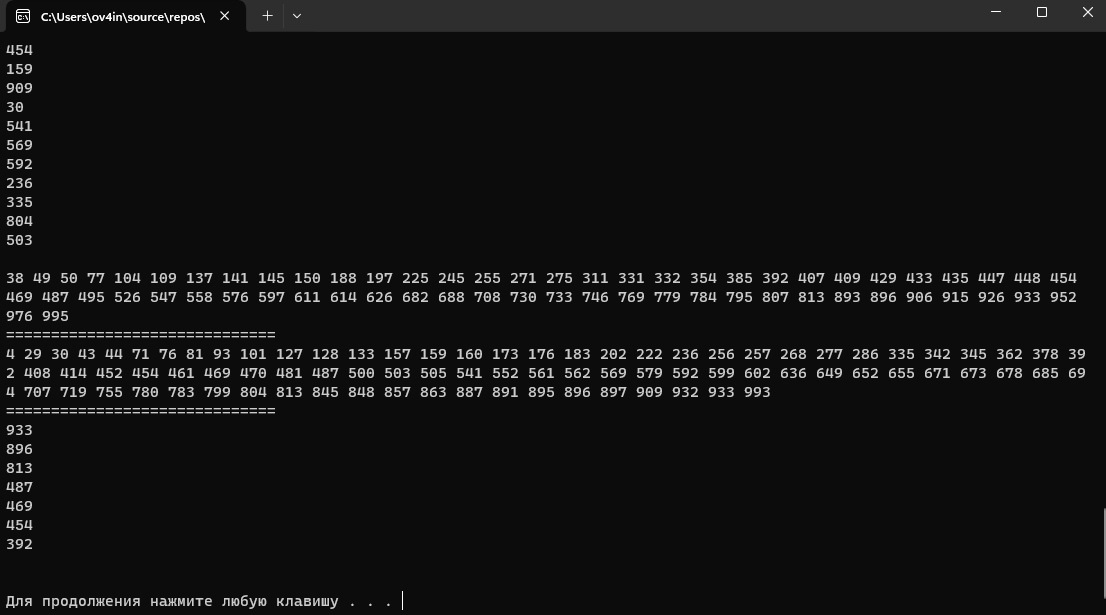
cout << endl;

nullStack(st2);

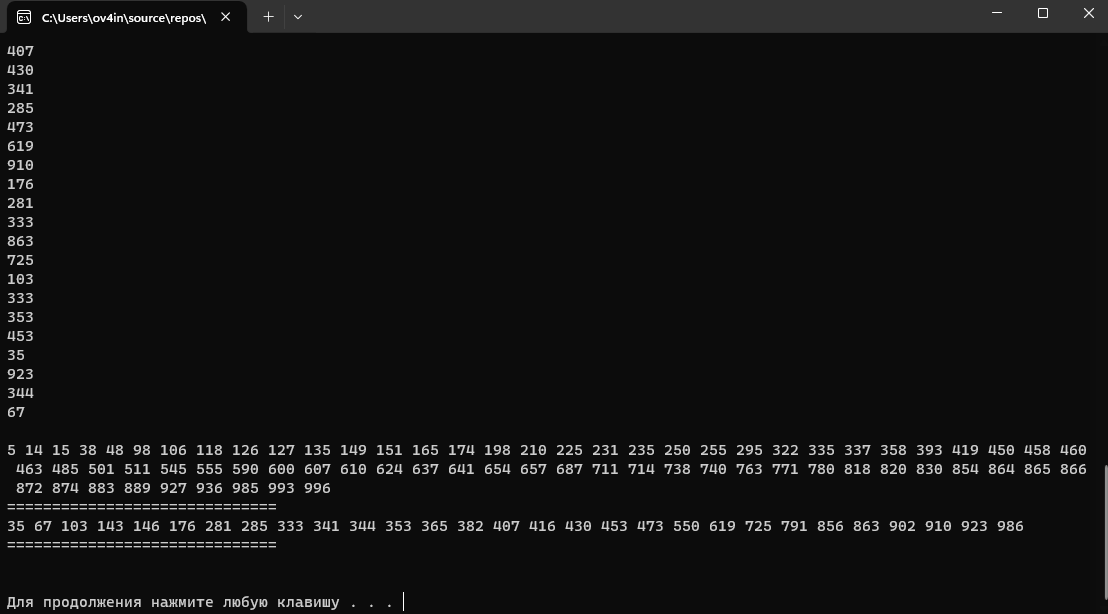
system("pause");

return 0;

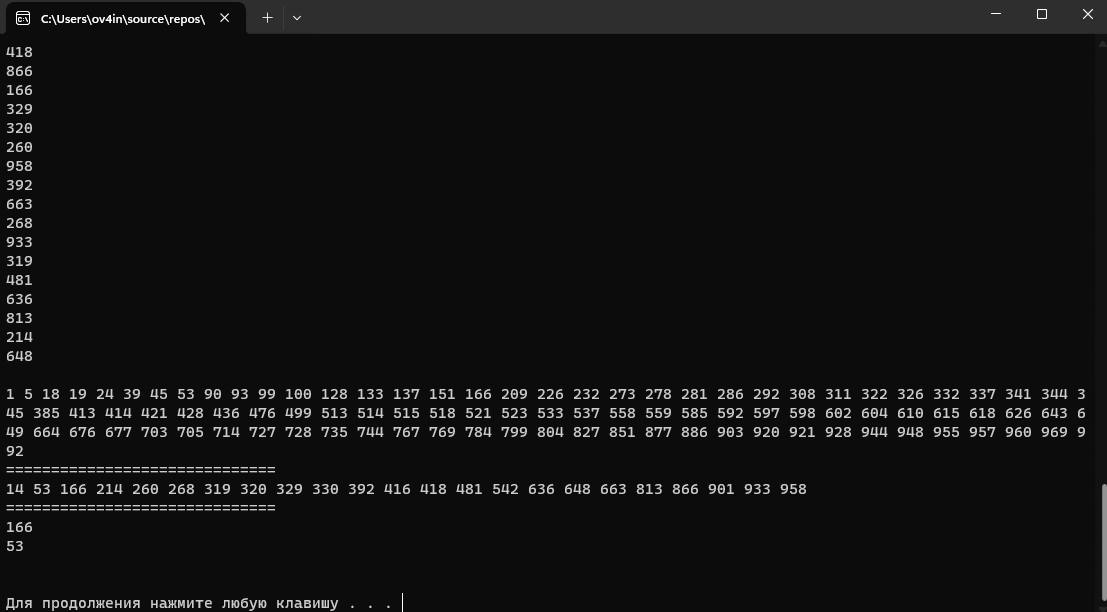
}

Тесты:  
1) 

2)



3)



Алгоритм:

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

2. Объявление стеков **st1** и **st2** типа **Stack1**.

3. Инициализация стека **st1** через **InitStack**.

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

5. Вывод содержимого стека **st1** с помощью функции **PrintStack**.

6. Объявление очереди **Que1** типа **Queue1**.

7. Инициализация очереди **Que1** через **InitQueue**.

8. Заполнение очереди **Que1** случайными значениями с использованием функции **add**.

9. Вывод содержимого очереди **Que1** с помощью функции **PrintQueue**.

10. Объявление пустого множества целочисленных значений **int** с именем **num1**.

11. Заполнение множества **num1** элементами стека **st1**.

12. Удаление стека **st1.**

13. Объявление пустого множества целочисленных значений **int** с именем **num2**.

14. Заполнение множества **num2** элементами очереди **Que1**.

15. Удаление очереди **Que1.**

16. Вывод содержимого множеств **num1** и **num2**.

17. Объявление пустого множества целочисленных значений **int** с именем **num3**.

18. Поиск общих элементов двух множеств **num1** и **num2** с помощью операции пересечения **set\_intersection** и запись общих элементов множеств в **num3**.

19. Инициализация стека **st2** через **InitStack** с вместимостью, равной кол-ву элементов множества **num3**.

18. Заполнение стека **st2** элементами множества **num3**.

19. Вывод содержимого стека **st2** с помощью функции **PrintStack**.

20. Завершение программы.

Б) реализация через динамические списки:

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();

void 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;

}

}

void Queue::Del() {

if (Empty()) {

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

}

else {

Node1\* tmp = head;

head = head->next;

delete(tmp);

}

}

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)

#include "Header.h"

#include <iostream>

#include<time.h>

#include<set>

#include <algorithm>

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 PrintNode(Node\*& top) {

Node\* current = top;

while (current != NULL) {

cout << current->data << endl;

current = current->next;

}

}

int main() {

srand(time(NULL));

Node\* st1,\* st2;

InitStack(st1);

int capacity1 = 1 + rand() % 100;

for (int i = 0; i <=capacity1; i++)

push(st1, rand() % 1000);

PrintNode(st1);

cout << endl;

Queue que;

int capacity2 = 1 + rand() % 100;

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

que.Add(rand() % 1000);

que.PrintQueue();

cout << endl;

set<int> num1;

while(!empty(st1))

num1.insert(pop(st1));

set<int> num2;

while(!que.Empty())

num2.insert(que.Get());

for (int n : num1)

cout << n << " ";

cout << endl;

cout << "==============================" << endl;

for (int n : num2)

cout << n << " ";

cout << endl;

cout << "==============================" << endl;

set<int> num3;

set\_intersection(num1.begin(), num1.end(),

num2.begin(), num2.end(),

inserter(num3, num3.begin()));

InitStack(st2);

for (int n : num3)

push(st2, n);

PrintNode(st2);

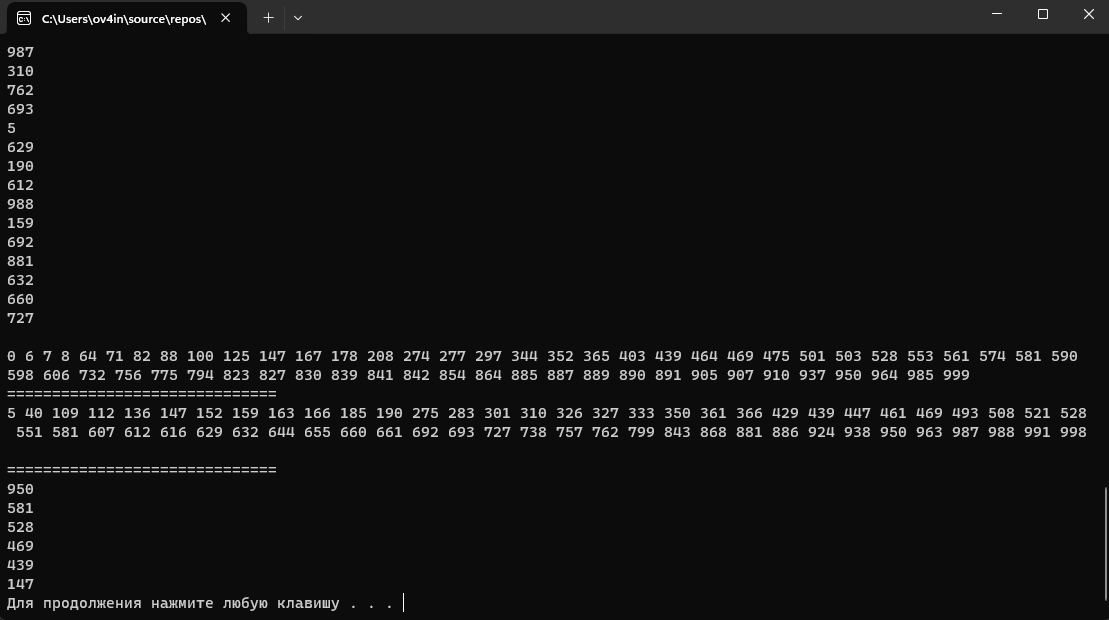
system("pause");

return 0;

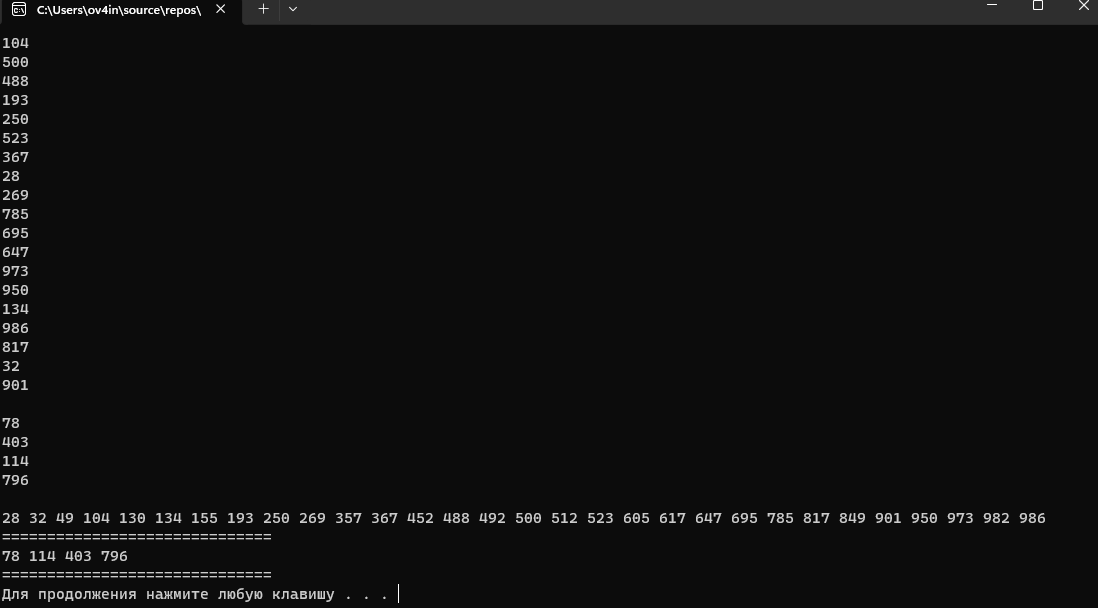
}

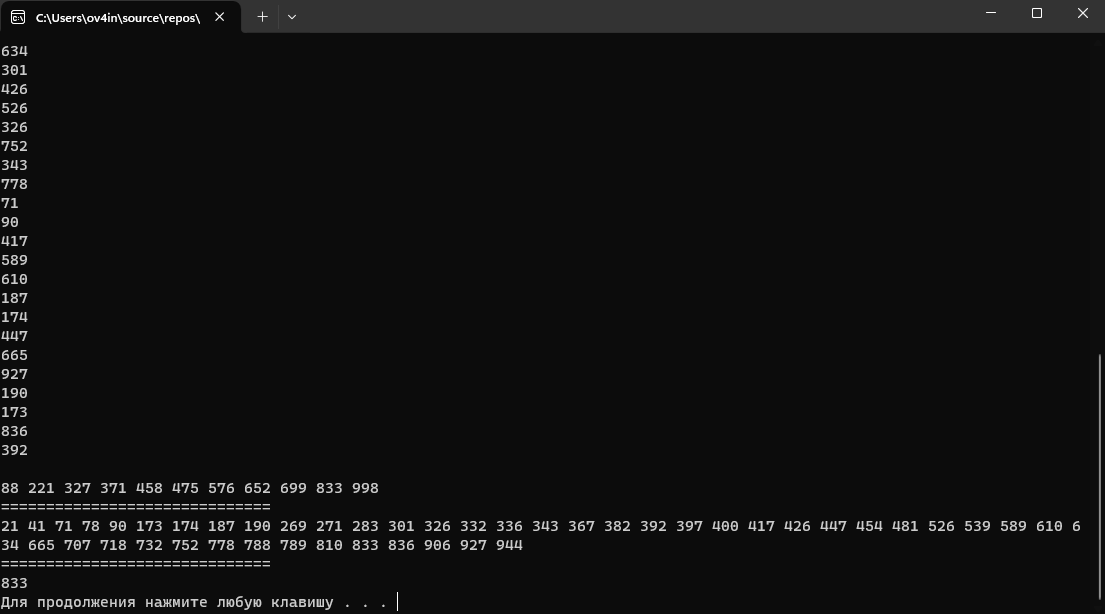
Тесты:

1)



2)



3) 

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

2) Объявление стеков \***st1** и \***st2** типа **Node**.

3) Инициализация стека **st1** через **InitStack**.

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

5) Вывод содержимого стека **st1** с помощью функции **PrintNode**.

6) Объявление объекта **que** класса **Queue.**7) Заполнение очереди **que** случайными значениями с использованием публичного метода класса **Add**.

8)Вывод содержимого очереди **que** с помощью публичного метода **QueuePrint.**

9) Объявление пустого множества целочисленных значений **int** с именем **num1**.

10) Заполнение множества **num1** элементами стека **st1**.

11) Объявление пустого множества целочисленных значений **int** с именем **num2**.

12) Заполнение множества **num2** элементами очереди **que**.

13) Вывод содержимого множеств **num1** и **num2**.

14) Объявление пустого множества целочисленных значений **int** с именем **num3**.

15) Поиск общих элементов двух множеств **num1** и **num2** с помощью операции пересечения **set\_intersection** и запись общих элементов множеств в **num3**.

16) Инициализация стека **st2** через **InitStack.**

17) Заполнение стека **st2** элементами множества **num3**.

18) Вывод содержимого стека **st2** с помощью функции **PrintNode**.

19) Завершение программы