**Лабораторна робота № 7. Алгоритми обслуговування черг**

***Мета:*** набути практичного досвіду та закріпити знання про пред-ставлення стека, дека, приоритетної черги та дисциплінах їх обслуговування.

**1 Вимоги**

**1.1 Розробник**

* Макаренко Владислав Олександрович
* Студент 1-го курсу
* Групи КІТ-120а

**1.2 Загальне завдання**

Розробити функції, що забезпечують запис та читання запитів з приоритетної черги, стека або дека.

В кожному завданні для організації вказаної черги використати дві структури. Перевірити працездатність розроблених функцій. Послідовність виконання операцій запису та читання обирати випадково.

**1.2 Загальне завдання**

Стек. Стек організований на масиві та на двоспрямованому списку.

**2 Описи програм**

**2.1 Стек організований на двоспрямованому списку**

**Код програми**

#include <iostream>  
#include <ctime>  
#include <cstdlib>  
using namespace std;  
  
template <class T>  
struct node{  
 T data;  
 node\* prev;  
 node\* next;  
};  
  
template <class T>  
class myStack{  
 node<T> \*last;  
public:  
 myStack(){last=NULL;};  
 int push(T *data*);//добавить  
 T pop();//извлечь  
 T read();//прочитать последний  
 int empty(){if(last==NULL) return 1;return 0; };  
};  
  
int main(){  
 srand(time(NULL));  
 setlocale(LC\_ALL,"Russian");  
 cout<<"Проверим стек:"<<endl;  
 myStack<int> test;  
 if(!test.empty())  
 cout<<test.read()<<endl;  
 else  
 cout<<"Стек пуст"<<endl;  
 cout << endl;  
 cout<<"Заполним стек;"<<endl;  
 for(int i=0;i<7;i++)  
 test.push(rand()%10);  
 cout << endl;  
 cout<<"Прочитаем последний элемент: ";  
 if(!test.empty())  
 cout<<test.read()<<endl;  
 else  
 cout<<"Стек пуст"<<endl;  
 cout << endl;  
 cout<<"Выведем и очистим стек:"<<endl;  
 int k = 0;  
 while(!test.empty()) {  
 cout << "Елемент[" << k << "] = " << test.pop() << endl;  
 k++;  
 }  
 return 0;  
  
}  
template <class T>  
int myStack<T>::push(T *data*){  
 node<T> \*add = new node<T>;  
 add->data=*data*;  
 add->next=NULL;  
 add->prev=NULL;  
 if(last==NULL)//стек пуст  
 this->last=add;  
 else{  
 last->next=add;  
 last->next->prev=last;  
 last=add;  
 }  
 return 0;  
}  
template <class T>  
T myStack<T>::pop(){  
 T copy = this->last->data;  
 node<T> \*del = last;  
 last=last->prev;  
 delete del;  
 return copy;  
}  
template <class T>  
T myStack<T>::read(){  
 T copy = this->last->data;  
 return copy;  
}

}

**Результати виконання програми**

За алгоритмом коду демонструємо роботу програми (див. рис. 1).

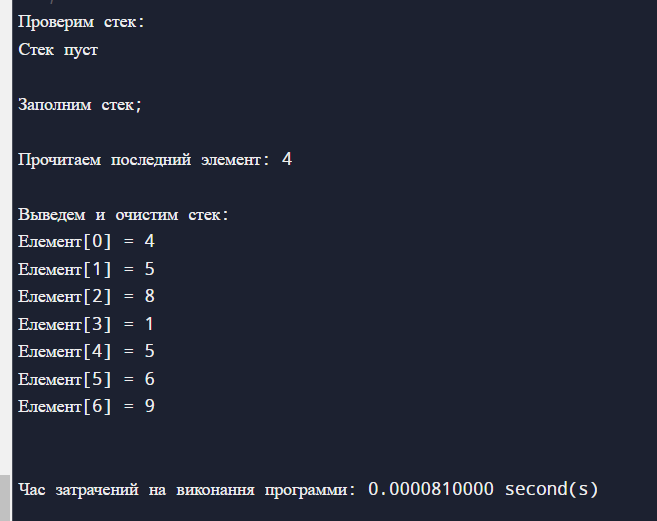


Рисунок 1 – Результати роботи першого кроку

**2.2 Стек організований на динамічному масиві**

**Код програми**

#include <iostream>  
#include <new>  
#include <ctime>  
#include <cstdlib>  
using namespace std;  
  
// класс, реализующий стек в виде динамического массива  
template <typename T>  
class STACK  
{  
private:  
 T\* stack; // Динамический масив-указатель на стек  
 int count; // Вершина стека - количество элементов типа T в стеке  
public:  
 STACK() // конструктор по умолчанию  
 {  
 count = 0; // количество элементов в стеке определяется по значению count  
 }  
 void push(T *item*) // помістити елемент в стек  
 {  
 T\* tmp; // временный указатель  
 tmp = stack; // указатель указывает на stack  
 stack = new T[count + 1]; // выделить память на 1 элемент больше, чем было выделено до этого  
 count++; // увеличить количество элементов в стеке на 1  
 for (int i = 0; i < count - 1; i++) // скопировать данные из памяти, на которую указывает tmp в память, на которую указывает stack  
 stack[i] = tmp[i];  
 stack[count - 1] = *item*; // добавить последний элемент  
 // освободить память, выделенную перед этим для stack, на эту память указывает tmp  
 if (count > 1)  
 delete[] tmp;  
 }  
 // Вытягнуть элемент из стека  
 // При вытягивании элемента из стека память не переопределяется  
 T pop()  
 {  
 if (count == 0)  
 return 0; // стек пуст  
 count--;  
 return stack[count];  
 }  
 // Просмотр элемента в вершине стека  
 T Head()  
 {  
 if (count == 0)  
 return 0;  
 return stack[count - 1];  
 }  
 // Количество элементов в стеке  
 int Count()  
 {  
 return count;  
 }  
 // Функция, которая определяет пуст ли стек  
 bool IsEmpty()  
 {  
 return count == 0;  
 }  
 // Функция, выводящая стек  
 void Print()  
 {  
 T\* p; // временный указатель, двигается по элементах стека  
 // 1. Установить указатель p на вершину стека  
 p = stack;  
 // 2. Вивід  
 cout << "Стек: " << endl;  
 if (count == 0)  
 cout << "Пусто" << endl;  
 for (int i = 0; i < count; i++)  
 {  
 cout << "Елемент[" << i << "] = " << \*p << endl;  
 p++; // прокрутить указатель на следующий элемент  
 }  
 cout << endl;  
 }  
};  
  
int main()  
{  
 clock\_t start = clock();  
 srand(time(NULL));  
 setlocale(LC\_ALL,"Russian");  
 STACK <int> st1; // объявить стек из целых чисел  
 st1.Print(); // st1 = { }  
 // заполняем стек  
 cout << "Заполним стек рандомными елементами: " << endl;  
 cout << endl;  
 st1.push(rand()%10);  
 st1.push(rand()%10);  
 st1.push(rand()%10);  
 st1.push(rand()%10);  
 st1.Print();  
 cout << "Количество елементов: " << st1.Count() << endl;  
 cout << endl;  
 int t;  
 t = st1.pop(); // t = 7  
 cout << "Удалим елемент: " << t << endl;  
 cout << endl;  
 st1.Print(); // 5, 9, 13  
 cout << "Прочитаем последний элемент: " << st1.Head() << endl;  
 int k;  
 t = st1.pop(); // st1 = { 5, 9 }  
 k = st1.pop(); // st1 = { 5 }  
 cout << "Удалим елементы: " << t << " и " << k << endl;  
 cout << endl;  
 st1.Print();  
 cout << "Очистим стек: " << endl;  
 st1.pop();  
 if (st1.IsEmpty())  
 cout << "Стек пустой." << endl;  
 else  
 cout << "Стек не пустой" << endl;  
 clock\_t end = clock();  
 double time = (double )(end - start) / CLOCKS\_PER\_SEC;  
 printf("%.10f", time);  
}

**Результати виконання програми**

За алгоритмом коду демонструємо роботу програми (див. рис. 2).

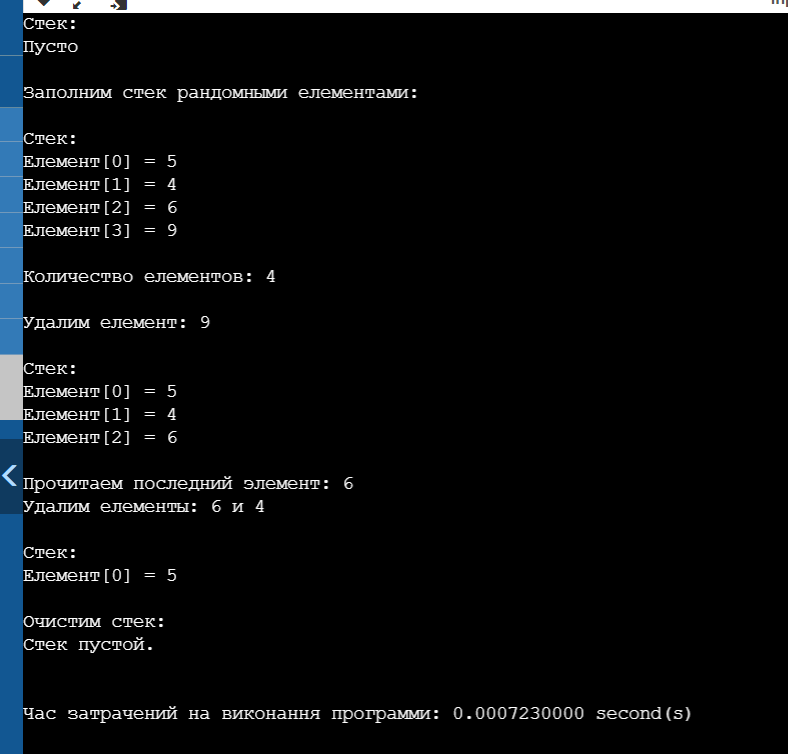


Рисунок 2 – Результати роботи другого кроку

**3 Порівняння часу**

Порівняємо часу виконання програми на основі списку(зліва) та динамічного масиву (справа) за допомогою функції *clock()* (див. рис.3).

|  |  |  |
| --- | --- | --- |
|  | **<** |  |

Рисунок 3 – Порівняння часу

Як бачимо з (рис.3) задана програма працює бистріше для списку, а не для масиву.

**Висновок:** на цій лабораторній роботі ми набули практичного досвіду та закріпили знання про представлення стека, дека, приоритетної черги та дисциплінах їх обслуговування..