Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №8
«Разработка шаблона класса»
по курсу: «Языки и методы программирования»

Выполнил:
Студент группы ИУ9-21Б
Гречко Г.В.

Проверил:
Посевин Д.П.

Москва, 2022

Цели

Целью данной работы является изучение шаблонов классов языка C++.

Задачи

List – последовательность значений типа T, реализованная через двунаправленный список с операциями поиска значения, вставки значения в указанное место, удаления значения. В List дополнительно должна присутствовать операция вычисления суммы значений.

Решение

Исходный код

List.hpp

```
#ifndef DOUBLE_LINKED_LIST
#define DOUBLE_LINKED_LIST

#include <cstdlib>
#include <string_view>
#include <iostream>

template <typename T>
class List
{
    struct Elem
    {
        T data;
        Elem* next;
        Elem* prev;
    };

private:
    Elem* head;
    Elem* cur;
    unsigned long length;
public:
    List();
    List(T v);
    List(const List<T>& obj);
    virtual ~List();
    void push(T data);
    T pop();
    void Delete(T data);
    void nextElem();
    void prevElem();
    bool isEmpty();
    bool ListSearch(T data);
    void print();
    T countSum();
    T peek();
};

template <typename T>
List<T>::List(){
    head = NULL;
    cur = NULL;
    length = 0;
}
```

```
template <typename T>
List<T>::List(T data){
    head = NULL;
    cur = NULL;
    length = 0;
    push(data);
}

template <typename T>
List<T>::List(const List<T>& obj){
    this->head = NULL;
    this->cur = NULL;
    this->length = NULL;
    Elem* ptr = obj.cur;

    for (size_t i = 0; i < obj.length; i++)
    {
        push(ptr->data);
        ptr = ptr->next;
    }
}

template <typename T>
List<T>::~List(){
    size_t n = length;
    for (size_t i = 0; i < n; i++)
    {
        pop();
    }
}

template <typename T>
void List<T>::Delete(T data){
    if(!isEmpty()){
        Elem* ptr = cur;
        size_t n = length;
        for (size_t i = 0; i < n; i++)
        {
            if (cur->data == data){
                if (cur == ptr)
                    ptr = ptr->next;
                pop();
            }
            nextElem();
        }
        cur = ptr;
    }
}

template <typename T>
void List<T>::nextElem(){
    if (!isEmpty())
    {
        cur = cur->next;
    }
}

template <typename T>
void List<T>::prevElem(){
    if (!isEmpty())
    {
```

```
        cur = cur->prev;
    }
}

template <typename T>
T List<T>::peek(){
    if (!isEmpty()){
        return cur->data;
    }
    return 0;
}

template <typename T>
void List<T>::push(T data){
    Elem* tmp;
    tmp = new Elem;
    tmp->data = data;
    if (isEmpty()) {
        head = tmp;
        cur = tmp;
        cur->next = tmp;
        cur->prev = tmp;
    } else {
        tmp->next = cur->next;
        tmp->next->prev = tmp;
        cur->next = tmp;
        tmp->prev = cur;
    }
    length++;
    cur = tmp;
}

template <typename T>
T List<T>::pop(){
    T res;
    if (isEmpty())
        return 0;
    Elem* temp = cur;
    res = cur->data;
    if (length == 1){
        head = NULL;
        cur = NULL;
    } else {
        cur->next->prev = cur->prev;
        cur->prev->next = cur->next;
        cur = cur->prev;
    }
    if (temp == head)
        head = head->next;

    length--;
    delete temp;
    return res;
}

template <typename T>
bool List<T>::ListSearch(T data){
    if(!isEmpty()){
        Elem* ptr = cur->next;
        while (ptr != cur && ptr->data != data)
        {
            ptr = ptr->next;
```

```
        }
        bool ans = ptr->data == data;
        cur = ptr;
        return ans;
    }
    return false;
}

template <typename T>
void List<T>::print(){
    Elem* temp = head;
    for (size_t i = 0; i < length; i++)
    {
        std::cout << temp->data << " ";
        temp = temp->next;
    }
    std::cout << "\n";

}


template <typename T>
constexpr auto type_name() {
  std::string_view name, prefix, suffix;
#ifdef __clang__
  name = __PRETTY_FUNCTION__;
  prefix = "auto type_name() [T = ";
  suffix = "]";
#elif defined(__GNUC__)
  name = __PRETTY_FUNCTION__;
  prefix = "constexpr auto type_name() [with T = ";
  suffix = "]";
#elif defined(_MSC_VER)
  name = __FUNCSIG__;
  prefix = "auto __cdecl type_name<";
  suffix = ">(void)";
#endif
    name.remove_prefix(prefix.size());
    name.remove_suffix(suffix.size());
    return name;
}

template <typename T>
T List<T>::countSum(){
    if (type_name<T>() != "int"){
        std::cout<<"Not implemented for " << type_name<T>() << "\n";
        return NULL;
    } else {
        Elem* temp = head;
        int sum = 0;
        for (size_t i = 0; i < length; i++)
        {
            sum += temp->data;
            temp = temp->next;
        }
        return sum;
    }
}

template <typename T>
bool List<T>::isEmpty(){
    if (cur == NULL){
```

```
        return true;
    }
    return false;
}

#endif
```

main.cpp

```
#include "List.hpp"
#include <iostream>
#include <cstdlib>
#include <string>

int main() {
    List<int> a;

    a.push(10); // Вставка значения
    a.push(20);
    a.push(40);
    a.push(80);

    a.print();
    a.Delete(40); // Удаление значения
    a.print();
    std::cout<< a.countSum() << std::endl; // Сумма списка для Int
    auto c = a.ListSearch(10); // Поиск значения в списке
    std::cout<< a.pop() << std::endl << c << std::endl;

    List<std::string> b; // Работает с разными типами
    b.push("abc");
    b.push("1246");
    b.print();
    return 0;
}
```

Пример вывода