МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа № 8

по дисциплине: ООП

тема: «Создание шаблонов классов в C++»

Выполнил: ст. группы ПВ202
Аладиб язан
Проверил:
Буханов Д.Г.

Белгород 2022

Цель работы: Получение теоретических знаний о шаблонах классов в C++. Получение практических навыков по созданию классов-шаблонов C++.

Задание:

1. Изучить теоретические сведения о шаблонах классов в C++.
2. Разработать программу в соответствии с заданным вариантом задания.
3. Оформить отчет.

Задание Варианта :

Реализовать шаблон класса в соответствии с указанным вариантом. Предусмотреть необходимые методы для работы со структурой данных, указанной в варианте. Предусмотреть исключительные ситуации, которые могут возникнуть в процессе работы.

Дек

Выполнение работы:

```
#include <iostream>

using namespace std;

template <class Type>
class ListEl{
    Type data;
    ListEl* next_link;
    ListEl* prev_link;
public:
    ListEl();
    ListEl(Type data);
    void set_next_link(ListEl* link);
    void set_prev_link(ListEl* link);
    void set_data(Type data);
    Type get_data();
    ListEl *get_next();
    ListEl *get_prev();
    void print();
};

template <class Type>
class DoubleLinkedList{
```

```
    ListEl<Type>* list_start;
    ListEl<Type>* list_end;
    ListEl<Type>* list_current;
public:
    DoubleLinkedList();
    void add_el(Type element);
    Type pop_el();
    Type get_el();
    void ptr_to_start();
    void ptr_to_end();
    int is_end();
    void clear();
    void next_el();
    void prev_el();
    void print();
};

template <class Type>
class Deck{
    DoubleLinkedList<Type> list;
    unsigned current_size;
    unsigned max_size;
public:
    Deck(unsigned size);
    void push_back(Type data);
    void push_front(Type data);
    Type pop_back();
    Type pop_front();
    int is_empty();
    void clear();
};

template <class Type>
ListEl<Type>::ListEl(){
    this->next_link=nullptr;
    this->prev_link=nullptr;
};

template <class Type>
ListEl<Type>::ListEl(Type data):data(data){
    this->next_link=nullptr;
    this->prev_link=nullptr;
};

template <class Type>
void ListEl<Type>::set_next_link(ListEl<Type>* link){
    this->next_link=link;
};

template <class Type>
void ListEl<Type>::set_prev_link(ListEl<Type>* link){
```

```
    this->prev_link=link;
};

template <class Type>
void ListEl<Type>::set_data(Type data){
    this->data=data;
};

template <class Type>
Type ListEl<Type>::get_data(){
    return this->data;
};

template <class Type>
ListEl<Type> *ListEl<Type>::get_next(){
    return this->next_link;
};

template <class Type>
ListEl<Type> *ListEl<Type>::get_prev(){
    return this->prev_link;
};

template <class Type>
void ListEl<Type>::print(){
    cout<<"Data: "<<this->data<<endl;
};

template <class Type>
DoubleLinkedList<Type>::DoubleLinkedList(){
    this->list_start= new ListEl<Type>();
    this->list_end= new ListEl<Type>();
    this->list_current=this->list_start;

};

template <class Type>
void DoubleLinkedList<Type>::add_el(Type element){
    ListEl<Type> *temp_ptr = new ListEl<Type>(element);

    if(this->list_current->get_next()==nullptr){
        temp_ptr->set_next_link(nullptr);
        temp_ptr->set_prev_link(list_current);
    }else{
        temp_ptr->set_next_link(this->list_current->get_next());
        temp_ptr->set_prev_link(list_current);
        this->list_current->get_next()->set_prev_link(temp_ptr);
    }
    this->list_current->set_next_link(temp_ptr);
    this->next_el();
};
```

```
template <class Type>
Type DoubleLinkedList<Type>::pop_el(){
    Type temp=this->list_current->get_data();
    ListEl<Type> *pr=this->list_current->get_prev();
    ListEl<Type> *nx=this->list_current->get_next();
    pr->set_next_link(nx);
    nx->set_prev_link(pr);
    return temp;
};

template <class Type>
Type DoubleLinkedList<Type>::get_el(){
    return this->list_current->get_data();
};

template <class Type>
void DoubleLinkedList<Type>::ptr_to_start(){
    this->list_current=this->list_start;
    this->next_el();
};

template <class Type>
void DoubleLinkedList<Type>::ptr_to_end(){
    this->list_current=this->list_end;
    this->prev_el();

};

template <class Type>
void DoubleLinkedList<Type>::clear(){
    this->ptr_to_start();
    while(this->list_current!=this->list_end){
        this->pop_el();
    }
};

template <class Type>
int DoubleLinkedList<Type>::is_end(){
    return this->list_current==this->list_end;
};

template <class Type>
void DoubleLinkedList<Type>::next_el(){
    this->list_current=this->list_current->get_next();
};

template <class Type>
void DoubleLinkedList<Type>::prev_el(){
    this->list_current=this->list_current->get_prev();
};
```

```
template <class Type>
void DoubleLinkedList<Type>::print(){
    ListEl<Type> *temp_pos=this->list_current;
    this->ptr_to_start();
    int num=1;
    while(this->list_current!=this->list_end){
        cout<<"Element : "<<num<<endl;
        this->list_current->print();
        this->next_el();
        num++;
    };
    this->list_current=temp_pos;
};

template <class Type>
Deck<Type>::Deck(unsigned
size):max_size(size),size(0),list(DoubleLinkedList<Type>()){};

template <class Type>
void Deck<Type>::push_back(Type data){
    this->list.ptr_to_end();
    this->list.add_el(data);
    this->current_size++;
};

template <class Type>
void Deck<Type>::push_front(Type data){
    this->list.ptr_to_start();
    this->list.add_el(data);
    this->current_size++;
};

template <class Type>
Type Deck<Type>::pop_back(){
    this->list.ptr_to_end();
    this->current_size--;
    return this->list.pop_el();
};

template <class Type>
Type Deck<Type>::pop_front(){
    this->list.ptr_to_start();
    this->current_size--;
    return this->list.pop_el();
};

template <class Type>
int Deck<Type>::is_empty(){
    return this->current_size==0;
};
```

```
template <class Type>
void Deck<Type>::clear(){
    this->list.clear();
    this->current_size=0;
};

int main(){
    DoubleLinkedList<int> a;
    for(int i=1;i<=10;i++){
        a.add_el(i);
    }
    a.print();
    return 0;
}
```