МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Алгоритмы и структуры данных»
Тема: «Списки»
12 вариант

Студент гр. 8302 _____ Кузнецов В. А.

Преподаватель _____ Тутуева А.В.

Санкт-Петербург
2020

Цель работы

Реализовать класс двусвязного списка, научиться проводить модульное тестирование методов класса с помощью CppUnitTestFramework.

Постановка задачи. Описание реализуемого класса и методов

Нужно написать класс односвязного списка с методами:

1. void push_back(int) – добавление в конец списка
2. void push_front(int) – добавление в начало списка
3. void pop_back() – удаление последнего элемента
4. void pop_front() – удаление первого элемента
5. void insert(size_t, int) – добавление элемента по индексу (вставка перед элементом, который был ранее доступен по этому индексу)
6. int at(size_t) – получение элемента по индексу
7. void remove(size_t) – удаление элемента по индексу
8. size_t get_size() – получение размера списка
9. void print_to_console() – вывод элементов списка в консоль через разделитель, не использовать at
10. void clear() – удаление всех элементов списка
11. void set(size_t, int) – замена элемента по индексу на передаваемый элемент
12. bool isEmpty() – проверка на пустоту списка
13. size_t find_first(List) – поиск первого вхождения другого списка в список

Каждая функция должна быть протестирована с помощью CppUnitTestFramework.

В результате выполнения данной лабораторной работы был создан класс MyList с полями и методами:

1. private class Node – внутренний класс класса, представляющий элемент списка с полями и методами:
    1.1. private char value – значение элемента, символ
    1.2. private Node* next – указатель на следующий элемент списка
    1.3. private Node* prev – указатель на предыдущий элемент списка

1.4. Node(), Node(char ch) – конструкторы класса для пустого и заданного значения элемента соответственно
1.5. Node* getPrev() – метод получения предыдущего элемента
1.6. Node* getNext() – метод получения следующего элемента
1.7. Node* getValue() – метод получения значения элемента
1.8. void setPrev(Node* prev) – метод установки предыдущего элемента
1.9. void setNext(Node* next) – метод установки следующего элемента
1.10. void setValue(char ch) – метод установки значения элемента

2. private Node* head – головной (начальный) элемент списка, от которого идут все остальные элементы
3. private size_t length – размер списка (количество элементов)
4. MyList(), MyList(char ch) – конструкторы списка без заданного и с заданным первым элементом соответственно
5. Void push_back(char ch) – вставляет элемент в конец списка
6. Void push_front(char ch) – вставляет элемент в начало списка
7. Void pop_back() – удаляет последний элемент списка
8. Void pop_front() – удаляет первый элемент списка
9. Void insert(char ch, size_t index) – вставляет элемент на позицию index
10. Char at(size_t index) – возвращает элемент на позиции index
11. Void remove(size_t index) – удаляет элемент на позиции index
12. Size_t getSize() – возвращает размер списка
13. Void print_to_console – печатает все элементы списка с разделителем пробел
14. Void clear() – очищает список
15. Void set(char ch, size_t index) – заменяет элемент на позиции index элементом ch
16. Bool isEmpty() – пустой список или нет
17. Int find_first(MyList* list) – возвращает индекс первое вхождение списка list в исходном списке и -1 в противном случае

Оценка временной сложности каждого метода

Функция	Сложность
push_back(char)	O(n)
push_front(char)	O(1)
pop_back()	O(n)
pop_front()	O(1)
insert(size_t, char)	O(n)
at(size_t)	O(n)
remove(size_t)	O(n)
get_size()	O(1)
print_to_console()	O(n)
clear()	O(n)
set(size_t, char)	O(n)
isEmpty()	O(1)
Find_first(MyList*)	O(n^2)

Описание реализованных unit-тестов

Перед запуском каждого теста инициализируется список [‘a’, ‘b’, ‘c’] в методе TEST_METHOD_INITIALIZE.

1. test_push_back – Тестирует метод push_back(char). В конец списка добавляется ‘d’ с помощью тестируемой функции. Далее проверяется, равен ли последний элемент списка ‘d’.
2. test_push_front – Тестирует метод push_front(char). В начало списка добавляется ‘0’ с помощью тестируемой функции. Далее проверяется, равен ли первый элемент списка ‘0’.
3. test_pop_back – Тестирует метод pop_back(). Удаляет последний элемент с помощью тестируемой функции. Далее проверяется, равна ли длина списка 2.

4. test_pop_front – Тестирует метод pop_front(). Удаляет первый элемент с помощью тестируемой функции. Далее проверяется, равен ли первый элемент ‘b’.
5. test_insert – Тестирует метод insert(size_t, char). Вставляет перед первым элементом новый узел со значением ‘0’, перед третьим элементом ‘1’ и перед пятым элементом ‘2’ с помощью тестируемой функции. Далее проверяется, равен ли первый элемент ‘0’, третий ‘1’ и пятый элемент ‘2’.
6. test_at – Тестирует метод at(size_t). Сравнивает значение второго элемента списка (полученного функцией at(1)).
7. test_remove – Тестирует метод remove(size_t). Удаляет второй элемент списка с помощью тестируемой функции. Далее проверяется, равен ли второй элемент списка ‘c’.
8. test_get_size – Тестирует метод get_size(). Сравнивает значение, полученное методом get_size() с 3.
9. test_clear – Тестирует метод clear(). Удаляет список, используя данную функцию. Если isEmpty() вернет true, то тест пройден.
10. test_set – Тестирует метод set(size_t, int). Меняет значение второго элемента на ‘f’. Далее сравнивается значение второго элемента с ‘f’.
11. test_is_empty – Тестирует метод isEmpty(). Запускает функцию до и после удаления списка. Метод должен вернуть false и true соответственно.
12. test_print_to_console – Тестирует метод print_to_console(). Если при запуске функции не произошло ошибок, то тест пройден.
13. test_find_first – Тестирует метод find_first(MyList*). Если для списков [], [‘a’], [‘b’, ‘c’], [‘a’, ‘b’, ‘c’, ‘d’] функция возвращает -1, 0, 1 и -1 соответственно, то тест пройден.

Пример работы

```
int main() {
    auto myList = new MyList();
    std::cout << "empty new list: ";
    myList->print_to_console();
    myList->push_back('b');
    myList->push_front('a');
    myList->push_back('c');
    std::cout << "pushing: ";
    myList->print_to_console();
    myList->set(1, 'k');
    myList->set(2, 'u');
    std::cout << "setting: ";
    myList->print_to_console();
    std::cout << "char at 1: " << myList->at(1) << std::endl;
    myList->pop_back();
    myList->pop_front();
    std::cout << "poping: ";
    myList->print_to_console();
    myList->clear();
    std::cout << "clear list: ";
    myList->print_to_console();
    return 0;
}
```

Рисунок 1 – Демонстрационный код

```
empty new list:
pushing: a b c
setting: a k u
char at 1: k
poping: k
clear list:
```

Рисунок 2 – Результат работы

Листинг

Lab1.cpp

```
#include <iostream>

class MyList {
private:
	class Node {  //private internal class of node of list
	private:
		char value;  //value is symbol
		Node* next;  //pointer to the next node
		Node* prev;  //pointer to the previous node
	public:
		Node() {  //constructor without value of node
			this->value = NULL;
			this->next = nullptr;
			this->prev = nullptr;
		};
		Node(char ch) {  //constructor with value of node
			this->value = ch;
			this->next = nullptr;
			this->prev = nullptr;
		};
		~Node() = default;  //default destructor
		Node* getPrev() {  //get previous node
			return prev;
		}
		Node* getNext() {  //get next node
			return next;
		}
		char getValue() {  //get value of node
			return value;
		}
		void setPrev(Node* prev) { //set previous node
			this->prev = prev;
			if (prev != nullptr)
				prev->next = this;
		}
		void setNext(Node* next) {  //set next node
			this->next = next;
			if (next != nullptr)
				next->prev = this;
		}
		void setValue(char ch) {  //set value of node
			this->value = ch;
		}
	};
	Node* head;  //head of list
	size_t length;  //size
public:
	MyList() {  //constructor without first node
		this->head = new Node();
		this->length = 0;
	}
	MyList(char ch) {  //constructor with first node
		this->head = new Node(ch);
		this->length = 1;
	}
	~MyList() {  //destructor of list
		if (length == 0 || length == 1) {
			delete head;
```

```
        }
        else {
                auto nextNode = head->getNext();
                for (size_t i = 1; i < length; i++) {
                        delete nextNode->getPrev();
                        nextNode = nextNode->getNext();
                }
                delete nextNode;
        }
        length = NULL;
}
void push_back(char ch) {  //pushing symbol in back of list
        if (length == 0) {
                head->setValue(ch);
        }
        else {
                auto pushNode = new Node(ch);
                auto node = head;
                for (size_t i = 0; i < length - 1; i++) {
                        node = node->getNext();
                }
                node->setNext(pushNode);
        }
        ++length;
}
void push_front(char ch) {  //pushing symbol in front of list
        if (length == 0) {
                head->setValue(ch);
        }
        else {
                auto pushNode = new Node(ch);
                head->setPrev(pushNode);
                head = head->getPrev();
        }
        ++length;
}
void pop_back() {  //removing last node of list
        if (length == 0) {
                return;
        }
        else if (length == 1) {
                delete head;
                head = new Node();
        }
        else {
                auto popNode = head;
                for (size_t i = 0; i < length - 1; i++) {
                        popNode = popNode->getNext();
                }
                auto prevNode = popNode->getPrev();
                delete popNode;
                prevNode->setNext(nullptr);
        }
        --length;
}
void pop_front() {  //removing first node of list
        if (length == 0) {
                return;
        }
        else {
                auto newHead = head->getNext();
                delete head;
                head = newHead;
```

```
                --length;
        }
    }
    void insert(char ch, size_t index) {  //insert symbol 'ch' in list in the position of
'index'
        if (index == 0) {
            push_front(ch);
        }
        else if (index == length) {
            push_back(ch);
        }
        else if (index > length) {
            return;
        }
        else {
            auto node = new Node(ch);
            auto prevNode = head;
            for (size_t i = 1; i < index; i++) {
                prevNode = prevNode->getNext();
            }
            auto nextNode = prevNode->getNext();
            prevNode->setNext(node);
            nextNode->setPrev(node);
            ++length;
        }
    }
    char at(size_t index) {  //get symbol on position 'index' of list
        if (length == 0 || index >= length) {
            return NULL;
        }
        else {
            auto node = head;
            for (size_t i = 0; i < index; i++) {
                node = node->getNext();
            }
            return node->getValue();
        }
    }
    void remove(size_t index) {  //remove symbol on position 'index' of list
        if (index == 0) {
            pop_front();
        }
        else if (index == length - 1) {
            pop_back();
        }
        else if (index >= length) {
            return;
        }
        else {
            auto prevNode = head;
            for (size_t i = 1; i < index; i++) {
                prevNode = prevNode->getNext();
            }
            auto nextNode = prevNode->getNext()->getNext();
            delete prevNode->getNext();
            prevNode->setNext(nextNode);
            nextNode->setPrev(prevNode);
            --length;
        }
    }
    size_t getSize() {  //getting size of list
        return length;
    }
```

```
	void print_to_console() {  //printing all symbols of list in console with delimiter '
'
		if (length == 0) {
			std::cout << " " << std::endl;
			return;
		}
		auto node = head;
		std::cout << node->getValue() << " ";
		for (size_t i = 1; i < length; i++) {
			node = node->getNext();
			std::cout << node->getValue() << " ";
		}
		std::cout << std::endl;
	}
	void clear() {  //removing all symbols of list
		if (length == 0) {
			return;
		}
		else if (length == 1) {
			delete head;
		}
		else {
			auto nextNode = head->getNext();
			for (size_t i = 1; i < length; i++) {
				delete nextNode->getPrev();
				nextNode = nextNode->getNext();
			}
			delete nextNode;
		}
		head = new Node();
		length = 0;
	}
	void set(size_t index, char ch) {  //setting symbol 'ch' instead of symbol in
position of 'index'
		if (index >= length) {
			return;
		}
		else {
			auto node = head;
			for (size_t i = 0; i < index; i++) {
				node = node->getNext();
			}
			node->setValue(ch);
		}
	}
	bool isEmpty() {  //returns whether the list is empty
		return length == 0;
	}
	int find_first(MyList* list) {  //returns the first entry (index) of the list 'list'
in original list
		auto listNode = head;  //original list
		for (size_t i = 0; i < length; i++) {  //iteration of nodes of original list
			if (length - i < list->length)  //if remaining length of list shorter
than the length of 'list' so there are no entries
				return -1;
			auto otherListNode = list->head;  //'list'
			if (listNode->getValue() == otherListNode->getValue()) {
				auto listNode2 = listNode;
				bool flag = true;  //whether or not an entry is found
				for (size_t j = 1; j < list->length; j++) {  //iteration of nodes
of 'list'
					listNode2 = listNode2->getNext();
					otherListNode = otherListNode->getNext();
```

```
                                    if (listNode2->getValue() != otherListNode->getValue()) {
//elements are not equal, so there is no entry
                                          flag = false;
                                          break;
                                    }
                              }
                              if (flag)  //entry found
                                    return i;  //index of first entry
                        }
                        listNode = listNode->getNext();
                  }
                  return -1;  //entry not found
            }
};

int main() {
      return 0;
}
```

UnitTestLab1.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../lab1/lab1.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTestLab1
{
      TEST_CLASS(UnitTestLab1)
      {
      private:
            MyList* myList = new MyList();
      public:
            TEST_METHOD_INITIALIZE(init) {
                  myList->push_back('a');
                  myList->push_back('b');
                  myList->push_back('c');
            }
            TEST_METHOD(test_push_back) {
                  myList->push_back('d');
                  Assert::AreEqual(myList->at(3), 'd');
            }
            TEST_METHOD(test_push_front) {
                  myList->push_front('0');
                  Assert::AreEqual(myList->at(0), '0');
            }
            TEST_METHOD(test_pop_back) {
                  myList->pop_back();
                  Assert::AreEqual(myList->getSize(), (size_t)2);
            }
            TEST_METHOD(test_pop_front) {
                  myList->pop_front();
                  Assert::AreEqual(myList->at(0), 'b');
            }
            TEST_METHOD(test_insert) {
                  myList->insert('0', 0);
                  myList->insert('1', 2);
                  myList->insert('2', 4);
                  Assert::AreEqual(myList->at(0), '0');
                  Assert::AreEqual(myList->at(2), '1');
                  Assert::AreEqual(myList->at(4), '2');
```

```
		}
		TEST_METHOD(test_at) {
			Assert::AreEqual(myList->at(1), 'b');
		}
		TEST_METHOD(test_remove) {
			myList->remove(1);
			Assert::AreEqual(myList->at(1), 'c');
		}
		TEST_METHOD(test_get_size) {
			Assert::AreEqual(myList->getSize(), (size_t)3);
		}
		TEST_METHOD(test_print_to_console) {
			myList->print_to_console();
		}
		TEST_METHOD(test_clear) {
			myList->clear();
			Assert::IsTrue(myList->isEmpty());
		}
		TEST_METHOD(test_set) {
			myList->set(1, 'f');
			Assert::AreEqual(myList->at(1), 'f');
		}
		TEST_METHOD(test_is_empty) {
			Assert::IsFalse(myList->isEmpty());
			myList->clear();
			Assert::IsTrue(myList->isEmpty());
		}
		TEST_METHOD(test_find_first) {
			MyList* list2 = new MyList();
			MyList* list3 = new MyList('a');
			MyList* list4 = new MyList('b');
			MyList* list5 = new MyList('a');
			list4->push_back('c');
			list5->push_back('b');
			list5->push_back('c');
			list5->push_back('d');
			Assert::AreEqual(myList->find_first(list2), -1);
			Assert::AreEqual(myList->find_first(list3), 0);
			Assert::AreEqual(myList->find_first(list4), 1);
			Assert::AreEqual(myList->find_first(list5), -1);
		}
	};
}
```

Вывод

В ходе выполнения работы был реализован класс линейного двусвязного списка, элементами которого являются символы типа данных char. Также было реализовано юнит-тестирование всех методов класса и успешное их выполнение.