Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа 5

По ООПИП

Тема: «Перегрузка операций»

Выполнила:

студент 2-го курса

группы АС-53

Завадский И.В.

Проверил:

Давидюк Ю.И.

Брест 2020

**Цель.** Получить практические навыки создания абстрактных типов данных и перегрузки операций в языке C++.

Вариант 9

9. АТД – однонаправленный список с элементами типа **char.** Дополнительно перегрузить следующие операции:

+ – добавить элемент в конец (list+char);

-- – удалить элемент из конца (типа list--);

== – проверка на равенство.

Код программы:

## Lab7.cpp

```
#include <iostream>
#include "TestMyList.h"

void DataBasicTest();
void OperationTest();

int main() {
	DataBasicTest();
	OperationTest();
	system("pause");
	return 0;
}

void DataBasicTest() {
	std::cout << "Test basic data\n\n\n";
	TestMyList* pTestData = new TestMyList();
	pTestData->TestPrintData();
	pTestData->TestInputData();
	pTestData->TestPrintData();
	pTestData->TestGetCurrentSize();
	TestMyList* pTestMaxSize = new TestMyList();
	pTestMaxSize->TestGetMaxSize();
	pTestMaxSize->TestMaxSize();
	std::cout << "\n\n\n";
	delete pTestData;
	delete pTestMaxSize;
}

void OperationTest() {
	std::cout << "Test binary operation\n\n\n";
	TestMyList* pTestBinaryOperation = new TestMyList();
	pTestBinaryOperation->TestCopyList();
	pTestBinaryOperation->TestAddItemList();
	pTestBinaryOperation->TestAddItemList();
	pTestBinaryOperation->TestMoveItemList();
	TestMyList* pTestEquals = new TestMyList();
	pTestEquals->TestEqualsList();
	std::cout << "\n\n\n";
	delete pTestBinaryOperation;
	delete pTestEquals;
}
```

## MyList.h

```cpp
#pragma once
#include <iostream>
#define MAX 255

struct list {
	char letter;
	list* pNext;
};

class MyList {
private:
	list* apHead;
	list* apTail;
	int 			aSize;
	static const int	MAX_SIZE;
	void Add(char);
	MyList& Copy(const MyList&);
public:
	MyList();
	MyList(const MyList&);
	~MyList();
	int 			GetSize() const;
	int 			GetMaxSize() const;
	void		Input();
	void		Print();
	void		operator+(char);
	void		operator--(int);
	bool		operator!=(MyList&);
	MyList& operator=(MyList&);
};

const int MyList::MAX_SIZE = MAX;
MyList::MyList() {
	apHead = nullptr;
	apTail = nullptr;
	aSize = 0;
}
MyList::MyList(const MyList& rMyList) {
	Copy(rMyList);
}
MyList::~MyList() {
	while (apHead != nullptr) {
		list* pTemp = apHead->pNext;
		delete apHead;
		apHead = pTemp;
	}
}
int MyList::GetSize()		const { return aSize; }
int MyList::GetMaxSize()	const { return MAX_SIZE; }
void MyList::Add(char symbol) {
	list* pTemp = new list;
	pTemp->letter = symbol;
	pTemp->pNext = nullptr;
	if (apHead != nullptr) {
		apTail->pNext = pTemp;
		apTail = pTemp;
	}
	else {
		apHead = pTemp;
		apTail = pTemp;
	}
	aSize++;
}
MyList& MyList::Copy(const MyList& rMyList) {
	if (rMyList.apHead != nullptr) {
```

```
			list* pTemp = new list;
			pTemp->letter = rMyList.apHead->letter;
			pTemp->pNext = nullptr;
			apHead = pTemp;
			apTail = pTemp;
			list* pNextPointer = new list;
			pNextPointer = rMyList.apHead->pNext;
			while (pNextPointer) {
				apTail->pNext = new list;
				apTail = apTail->pNext;
				apTail->letter = pNextPointer->letter;
				pNextPointer = pNextPointer->pNext;
			}
			apTail->pNext = nullptr;
			delete pNextPointer;
		}
		else {
			apHead = nullptr;
			apTail = nullptr;
			aSize = 0;
		}
		aSize = rMyList.aSize;
		return *this;
}
void MyList::Input() {
		char symbol = NULL;
		while (aSize < MAX_SIZE) {
			symbol = std::cin.get();
			if ('.' == symbol) {
				break;
			}
			else {
				Add(symbol);
			}
		}
		std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
void MyList::Print() {
		if (!apHead) {
			std::cout << "List is Empty!\n";
		}
		else {
			int counter = 0;
			list* pHead = apHead;
			list* pTail = nullptr;
			do {
				std::cout << pHead->letter;
				pTail = pHead->pNext;
				pHead = pTail;
				counter++;
			} while (counter != aSize);
			std::cout << std::endl;
		}
}
inline void MyList::operator+(char symbol) {
		Add(symbol);
}
inline void MyList::operator--(int i) {
		if (nullptr == apTail->pNext) {
			int counter = 1;
			list* pTail = apHead;
			while (counter != aSize - 1) {
				pTail = pTail->pNext;
				counter++;
			}
```

```
                delete apTail;
                pTail->pNext = nullptr;
                apTail = pTail;
                aSize--;
        }
        else {
                std::cout << "List is empty!\n";
        }
}
inline bool MyList::operator!=(MyList& rMyList) {
        if ((!apHead) || !rMyList.apHead) {
                std::cout << "Some List is Empty!\n";
        }
        else {
                if (aSize != rMyList.aSize) {
                        return true;
                }
                else {
                        list* pTemp = apHead;
                        list* pTempSecond = rMyList.apHead;
                        while (pTemp->pNext != nullptr) {
                                if (pTemp->letter != pTempSecond->letter) {
                                        return true;
                                }
                                pTemp = pTemp->pNext;
                                pTempSecond = pTempSecond->pNext;
                        }
                }
                return false;
        }
        return false;
}
inline MyList& MyList::operator=(MyList& rMyList) {
        return Copy(rMyList);
}
```

**TestMyList.h**

```
#pragma once
#include <iostream>
#include "MyList.h"

class TestMyList {
private:
        MyList aTestList;
public:
        TestMyList();
        TestMyList(const TestMyList&);
        ~TestMyList();
        void TestInputData();
        void TestPrintData();
        void TestGetCurrentSize();
        void TestGetMaxSize();
        void TestMaxSize();
        void TestCopyList();
        void TestAddItemList();
        void TestMoveItemList();
        void TestEqualsList();
};

TestMyList::TestMyList() { TestMyList::aTestList; }
TestMyList::TestMyList(const TestMyList& rTestList) { }
TestMyList::~TestMyList() { }
void TestMyList::TestInputData() {
```

```
	std::cout << "Enter the String(read to the first dot character ('.') or 255
characters):\n";
	aTestList.Input();
}
void TestMyList::TestPrintData() {
	std::cout << "Text in variable  ->  ";
	aTestList.Print();
}
void TestMyList::TestGetCurrentSize() {
	std::cout << "Current list size  =  "
		<< aTestList.GetSize() << std::endl;
}
void TestMyList::TestGetMaxSize() {
	std::cout << "Max list size  =  " << aTestList.GetMaxSize() << std::endl;
}
void TestMyList::TestMaxSize() {
	std::cout << "Max size test! Enter more than 255 characters:\n";
	aTestList.Input();
	std::cout << "Read data is  ->  ";
	aTestList.Print();
	std::cout << "Current list size  =  " << aTestList.GetSize()
		<< "\nMax list size  =  " << aTestList.GetMaxSize() << std::endl;
}
void TestMyList::TestCopyList() {
	TestInputData();
	MyList pTestList;
	pTestList = aTestList;
	std::cout << "One list has been assigned"
		<< " to another by the operator '=' (list1=list2)  ->  ";
	aTestList.Print();
	std::cout << "Copied list  ->  ";
	pTestList.Print();
}
void TestMyList::TestAddItemList() {
	char item = NULL;
	std::cout << "Enter one item  ->  ";
	std::cin >> item;
	aTestList + item;
	std::cout << "An item was added to the"
		<< "list using the operation '+' (list+char)  ->  ";
	aTestList.Print();
}
void TestMyList::TestMoveItemList() {
	aTestList--;
	std::cout << "Item in the list was "
		<< "deleted using the operation '--' (list--)  ->  ";
	aTestList.Print();
}
void TestMyList::TestEqualsList() {
	char item = '!';
	TestInputData();
	MyList pTestList;
	pTestList = aTestList;
	pTestList--;
	aTestList + item;
	std::cout << "Text entered with last character removed  ->  ";
	pTestList.Print();
	std::cout << "Copied text with a symbol added at the end '!' ->  ";
	aTestList.Print();
	std::cout << "Comparison of two texts with an operation '!=' (compare "
		<< "if the list characters do not match the characters"
		<< "in another list, then 'true')  ->  ";
	if (aTestList != pTestList) {
		std::cout << "true";
	}
```

```
        else {
                std::cout << "false";
        }
}
```