Министерство образования Республики Беларусь

Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №3.
"Перегрузка операций"

Выполнил:
Студент 2-го курса
Группы АС-53
Вожейко М.В.
Проверил:
Давидюк Ю. И.

Брест, 2020

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

2. Постановка задачи (Вариант 6)

6. АТД – множество с элементами типа **char.** Дополнительно перегрузить следующие операции:

> – проверка на принадлежность(char in set Паскаля);

* – пересечение множеств;

< – проверка на подмножество.

3. Определение класса:

```
#pragma once
class mychset { // Класс множества
private:
	char* value; // Массив char
	int count; // Количество элементов
public:
	mychset() : count(0), value(nullptr) { } // Конструктор без параметров
	mychset(const mychset&); // Констурктор копирования
	~mychset(); // Деструктор
	inline bool empty() const { return count == 0; } // Пустое ли множество
	inline char getChar(int position) const { return value[position]; } // Получить позицию
элемента в множестве
	inline int size() const { return count; } // Размер множества
	void push(const char item); // Добавить элемент
	void print(); // Вывести множество в поток
	void input(int size); // Ввод множества
	bool subset(const char item); // Входит ли char в множество
	bool subset(const mychset&); // Является ли подмножеством

	mychset& operator=(const mychset&); // Оператор присваивания
	mychset& operator*=(const mychset&); // Пересечение
	bool operator<(const mychset&); // Проверка на подмножество
	bool operator>(const char item); // Принадлежность
	friend mychset operator*(const mychset&, const mychset&); // Пересечение
};
```

Описание методов и функций класса:

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

mychset::mychset(const mychset& mychset) { // Конструктор копирования
	value = new char[mychset.count]; // Выделяем память
	count = mychset.count;
	for (int i = 0; i < count; i++) // Заполняем
		value[i] = mychset.value[i];
}

void mychset::input(int size) { // Ввод элементов
	char key;
	for (int k = 0; k < size; k++) {
		std::cout << "Enter element #" << k << ": ";
		std::cin >> key;
		this->push(key);
	}
}
```

```
void mychset::print() { // Вывод элементов
	for (int i = 0; i < count; i++)
		std::cout << value[i] << "\t";
	std::cout << "\nSize: " << count << "\n" << std::endl;
}

bool mychset::subset(const char item) { // Проверка на наличие элемента в множестве
	for (int i = 0; i < count; i++) {
		if (value[i] == item)
			return 1;
	}
	return 0;
}

bool mychset::subset(const mychset& mychset) { // Проверка на подмножество
	bool find = false;
	if (count >= mychset.count) { // Сверяем размеры множеств
		for (int i = 0; i < mychset.count; i++) { // Проходим все элементы множества
			for (int k = 0; k < count; k++) {
				if (value[k] == mychset.getChar(i)) {
					find = true;
				}
			}
			if (!find)
				return 0;
			find = false;
		}
		return 1;
	}
	else
		return 0;
}

void mychset::push(const char item) // Добавление элемента
{
	char* p2; // Выделяем память
	p2 = value;
	bool isFind = false;

	try {
		if (subset(item)) // Проверяем есть ли такой элемент
			return;
		value = new char[count + 1]; // Выделяем память
		for (int i = 0; i < count; i++) // Копируем
			value[i] = p2[i];
		for (int i = 0; i < count; i++) {
			if (item < value[i])
			{
				for (int k = count; k > i; k--) // Сдвигаем элементы для вставки
нужного
				{
					value[k] = value[k - 1];
				}
				value[i] = item;
				isFind = true;
				break;
			}
		}
		if (!isFind)
			value[count] = item;
		count++;
```

```
		if (count > 0)
			delete[] p2; // Освобождаем память
	}
	catch (std::bad_alloc e) {
		std::cout << e.what() << std::endl; // Если память не была выделена выводим
причину
	}
}

mychset::~mychset() { // Деструктор
	if (count > 0)
		delete[] value;
}

mychset& mychset::operator=(const mychset& obj) { // Оператор присваивания
	char* val2;

	try {
		val2 = new char[obj.count]; // Выделяем память
		if (count > 0)
			delete[] value; // Освобождаем исхоную если была аллоцирована
		value = val2;
		count = obj.count;
		for (int i = 0; i < count; i++) // Копируем
			value[i] = obj.value[i];
	}
	catch (std::bad_alloc e)
	{
		std::cout << e.what() << std::endl;
	}
	return *this;
}

mychset& mychset::operator*=(const mychset& _mychset) { // Оператор пересечения 1
	mychset* buff = new mychset();
	for (int i = 0; i < _mychset.size(); i++) {
		if (subset(_mychset.value[i])) {
			buff->push(_mychset.value[i]);
		}
	}
	*this = *buff;
	return *this;
}

mychset operator*(const mychset& _mychset, const mychset& _mychset2) {  // Оператор пересечения
2
	mychset buff(_mychset);
	buff *= _mychset2;
	return buff;
}

bool mychset::operator<(const mychset& _mychset) { // Подмножество
	return subset(_mychset);
}

bool mychset::operator>(const char _item) { // Принадлежность
	return subset(_item);
}
```

4. Обоснование включения в класс нескольких конструкторов, деструктора и операции присваивания:

- mychset::chset(const mychset& _mychset) – конструктор копирования, требуется для корректного создания объекта, копируя уже существующий.
- mychset() : count(0), value(nullptr) – конструктор без параметров.
- ~mychset() – деструктор, очищает массив char*
- mychset& mychset::operator= - Оператор присваивания, требуется для корректного создания копии.

5. Объяснить выбранное представление памяти для объектов реализуемого класса.

Значения множества хранятся в динамическом массиве char* это требуется для корректного добавления элементов в множество.

6. Реализация перегруженных операций с обоснованием выбранного способа (функция – член класса, внешняя функция, внешняя дружественная функция).

```
// Данные операторы - члены класс, т.к. я должен иметь доступ к приватным членам класса
mychset& mychset::operator=(const mychset& obj) { // Оператор присваивания
	char* val2;

	try {
		val2 = new char[obj.count]; // Выделяем память
		if (count > 0)
			delete[] value; // Освобождаем исхоную если была аллоцирована
		value = val2;
		count = obj.count;
		for (int i = 0; i < count; i++) // Копируем
			value[i] = obj.value[i];
	}
	catch (std::bad_alloc e)
	{
		std::cout << e.what() << std::endl;
	}
	return *this;
}

mychset& mychset::operator*=(const mychset& _mychset) { // Оператор пересечения 1
	mychset* buff = new mychset();
	for (int i = 0; i < _mychset.size(); i++) {
		if (subset(_mychset.value[i])) {
			buff->push(_mychset.value[i]);
		}
	}
	*this = *buff;
	return *this;
}

bool mychset::operator<(const mychset& _mychset) { // Подмножество
	return subset(_mychset);
}

bool mychset::operator>(const char _item) { // Принадлежность
	return subset(_item);
```

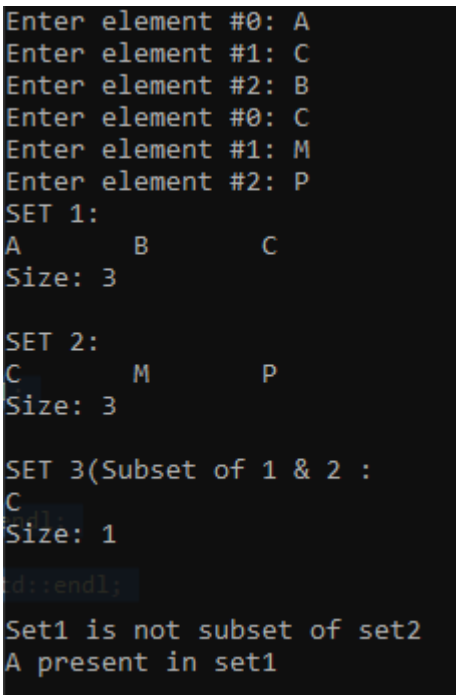
```
}
// Данный оператор – дружественный, т.к. я должнен возвращать новый объект класса и иметь
доступ к полям уже существующих используемых объектов.
mychset operator*(const mychset& _mychset, const mychset& _mychset2) {  // Оператор пересечения
2
	mychset buff(_mychset);
	buff *= _mychset2;
	return buff;
}
```

6. Тестовая программа:

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

int main()
{
	mychset set1; // Множество 1
	mychset set2; // Множество 2
	mychset set3; // Множество 3
	set1.input(3); // Ввод
	set2.input(3);
	set3 = set1 * set2; // Пересечение
	std::cout << "SET 1:\n";
	set1.print();
	std::cout << "SET 2:\n";
	set2.print();
	std::cout << "SET 3(Subset of 1 & 2 :\n";
	set3.print();
	std::cout << std::endl;
	if (set1 < set2)
		std::cout << "Set1 is subset of set2" << std::endl;
	else
		std::cout << "Set1 is not subset of set2" << std::endl;

	char isPresent = 'A'; // Символ для проверки
	if (set1 > isPresent)
		std::cout << isPresent << " present in set1" << std::endl;
	else
		std::cout << isPresent << " not present in set2" << std::endl;
	return 0;
}
```



6. Вывод:

Получил практические навыки создания абстрактных типов данных и перегрузки операций в языке C++.