Министерство образования Республики Беларусь

Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №3.
"Перегрузка операций"

Выполнил:
Ст. 2 курса гр. АС-53
Анискин Д. В.
Проверила:
Давидюк Ю. И.

Брест, 2020

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

2. Постановка задачи (Вариант 2)

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

- – удалить элемент из множества (типа set-char);

* – пересечение множеств;

< – сравнение множеств.

3. Определение класса:

```
#pragma once
class chset {
private:
	char* value;
	int count;
public:
	chset() : count(0), value(nullptr) { }
	chset(const chset&);
	~chset();
	inline bool empty() const { return count == 0; }
	inline char getChar(int position) const { return value[position]; }
	inline int size() const { return count; }
	void push(const char item);
	void remove(const char item);
	void print();
	void input(int size);
	bool subset(const char item);
	bool subset(const chset&);

	chset& operator-=(const char);
	chset& operator=(const chset&);
	chset& operator*=(const chset&);
	bool operator<(const chset&);
	friend chset operator-(const chset&, const char);
	friend chset operator*(const chset&, const chset&);
};
```

Описание методов и функций класса:

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

chset::chset(const chset& chset) {
	value = new char[chset.count];
	count = chset.count;
	for (int i = 0; i < count; i++)
		value[i] = chset.value[i];
}

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

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

bool chset::subset(const char item) {
	for (int i = 0; i < count; i++) {
		if (value[i] == item)
			return 1;
	}
	return 0;
}

bool chset::subset(const chset& chset) {
	bool find = false;
	if (count >= chset.count) {
		for (int i = 0; i < chset.count; i++) {
			for (int k = 0; k < count; k++) {
				if (value[k] == chset.getChar(i)) {
					find = true;
				}
			}
			if (!find)
				return 0;
			find = false;
		}
		return 1;
	}
	else
		return 0;
}

void chset::push(const char item)
{
	char* p2;
	p2 = value;
	bool isFind = false;

	try {
		if (subset(item))
			return;
		value = new char[count + 1];
		for (int i = 0; i < count; i++)
			value[i] = p2[i];
		for (int i = 0; i < count; i++) {
			if (item < value[i])
			{
				for (int k = count; k > i; k--)
				{
					value[k] = value[k - 1];
				}
				value[i] = item;
				isFind = true;
				break;
			}
		}
		if (!isFind)
			value[count] = item;
		count++;
```

```
		if (count > 0)
			delete[] p2;
	}
	catch (std::bad_alloc e) {
		std::cout << e.what() << std::endl;
	}
}

chset::~chset() {
	if (count > 0)
		delete[] value;
}

void chset::remove(const char item) {
	if (count < 1)
		return;
	if (!subset(item))
		return;
	try {
		char* val2;
		val2 = new char[count - 1];
		for (int i = 0; i < count - 1; i++)
			if (value[i] != item)
				val2[i] = value[i];
			else
			{
				for (int k = i; k < count - 1; k++)
					val2[k] = value[k + 1];
				break;
			}
		count--;
		if (count > 0)
			delete[] value;
		value = val2;
	}
	catch (std::bad_alloc e)
	{
		std::cout << e.what() << std::endl;
	}
}

chset& chset::operator=(const chset& obj) {
	char* val2;

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

chset& chset::operator-=(const char item) {
```

```
	remove(item);
	return *this;
}

chset& chset::operator*=(const chset& _chset) {
	chset* buff = new chset();
	for (int i = 0; i < _chset.size(); i++) {
		if (subset(_chset.value[i])) {
			buff->push(_chset.value[i]);
		}
	}
	*this = *buff;
	return *this;
}


chset operator-(const chset& _chset, const char item) {
	chset buff(_chset);
	buff -= item;
	return buff;
}

chset operator*(const chset& _chset, const chset& _chset2) {
	chset buff(_chset);
	buff *= _chset2;
	return buff;
}
bool chset::operator<(const chset& chset) {
	return size() < chset.size();
}
```

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

- chset::chset(const chset& _chset) – конструктор копирования, требуется для корректного создания объекта, копируя уже существующий.
- chset() : count(0), value(nullptr) – конструктор без параметров.
- ~chset() – деструктор, очищает массив char*
- chset& chset::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 и удаления элементов в множество.

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

```
Данные операторы – члены класса, т.к. мы изменяем поля данного класса, и возвращаем его же.
chset& chset::operator-=(const char item) {
	remove(item);
	return *this;
}

chset& chset::operator*=(const chset& _chset) {
```

```
	chset* buff = new chset();
	for (int i = 0; i < _chset.size(); i++) {
		if (subset(_chset.value[i])) {
			buff->push(_chset.value[i]);
		}
	}
	*this = *buff;
	return *this;
}

bool chset::operator<(const chset& chset) {
	return size() < chset.size();
}


// Данные операторы – дружественные, т.к. мы должны возвращать новый объект класса и иметь доступ к полям уже существующих используемых объектов.
chset operator-(const chset& _chset, const char item) {
	chset buff(_chset);
	buff -= item;
	return buff;
}

chset operator*(const chset& _chset, const chset& _chset2) {
	chset buff(_chset);
	buff *= _chset2;
	return buff;
}
```

6. Тестовая программа:

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

int main()
{
	chset set1;
	chset set2;
	chset set3;
	set1.input(3);
	set2.input(3);
	set1.print();
	set2.print();
	set1 = set1 - 'A';
	set2 = set2 - 'A';
	set3 = set1 * set2;
	set1.print();
	set2.print();
	set3.print();
	if (set1 < set2)
		std::cout << "Set1 < set2" << std::endl;
	else
		std::cout << "Set1 >= set2" << std::endl;
	return 0;
```

```
Enter element #0: A
Enter element #1: S
Enter element #2: D
Enter element #0: A
Enter element #1: D
Enter element #2: F
A       D       S
Size: 3

A       D       F
Size: 3

D       S
Size: 2

D       F
Size: 2

D
Size: 1

Set1 >= set2
}
```

6. Вывод:

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