**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 1**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **«Множество как объект»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1308 |  | Макаров М.В. |
| Студент гр. 1308 |  | Томилов Д. |  | Макаров М.В. |
| Преподаватель |  | Манирагена В. |  | Морозов С.М. |

Санкт-Петербург

2022

**Цель работы.**

Целью работы является изучение того, как использование классов будет влиять на выполнение операций с множествами.

**Задание (вариант 3)**

Универсум: шестнадцатеричные цифры.

Надо вычислить: Множество, содержащее цифры, имеющиеся в любом из множеств *A*, *B*, *C*, и *D*

Алгоритм должен быть реализован при помощи массива, списка, отображения на универсум при помощи массива битов и при помощи отображения на универсум с помощью машинного слова. Все реализации должны быть выполнены при помощи классов.

Нужно измерить время, за которое каждая реализация справляется с задачей. и сравнить его с реализацией без классов.

Также нужно отследить вызовы функций-членов классов и изучить, не происходят ли в программе утечки памяти.

**Постановка задачи и описание решения**

Единственная операция, используемая в задаче – это операция ИЛИ.

В реализации через массив и через массив битов программа сначала походит по каждому элементу универсума и смотрит, есть ли хоть в одном из 2-х исходных множеств этот элемент. Если он есть , то он добавляется в выходное множество и программа переходит к следующему элементу универсума.

В реализации через список программа сначала копирует одно из множеств, а затем проходит по элементам второго множества, смотря на то, присутствуют ли они уже в выходном. Если какого-то символа там не оказывается, то он добавляется к выходному множеству и программа переходит к следующему элементу.

В реализации через машинное слово для получения выходного множества используется операция побитового ИЛИ.

Для измерения времени и для отслеживания вызовов функция были написаны 2 программы, поскольку на ввод-вывод требуется очень много времени в сравнении с самими операциями. Вывод последовательности выполнения функций производится в отдельный файл.

**Формализация задания**

Формула для вычисления множества по четырем заданным:

E = A ∪ B ∪ C ∪ D

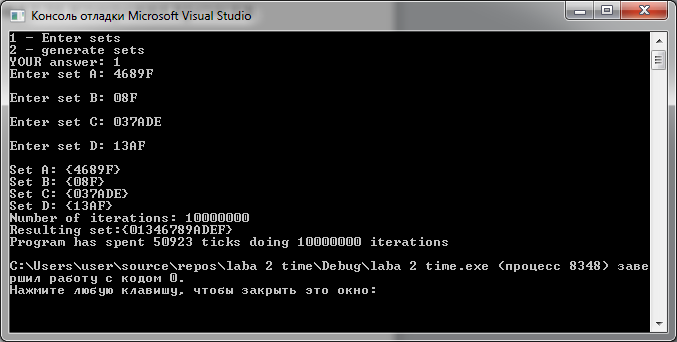
**Пример, на котором рассматривается выполнение программы**

|  |
| --- |
| **Входные данные** |
| множество A: {4689F}  множество B: {08F}  множество C: {037ADE}  множество D: {13AF} |
| **Выходные данные** |
| Результирующее множество: 01346789ADEF |

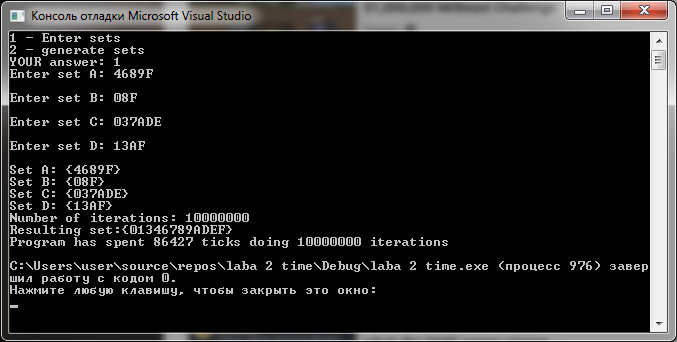
**Примеры выполнения программы для измерения времени**

**(на 10 000 000 итераций)**

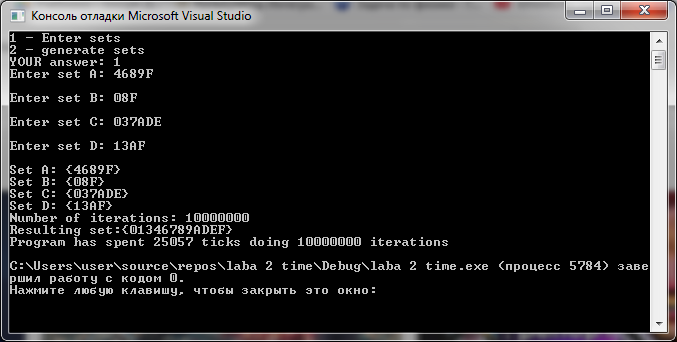
**Для массива:**



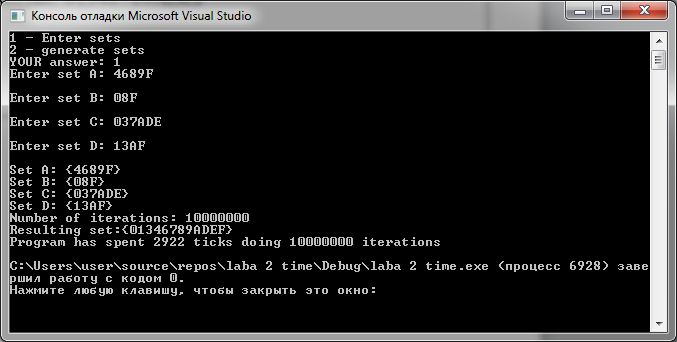
**Для списка:**



**Для массива битов:**

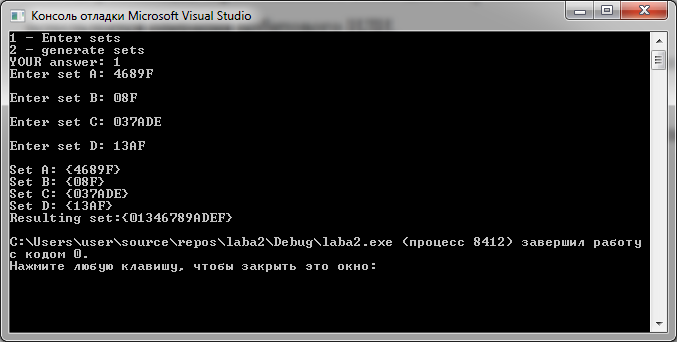


**Для битового слова:**



**Примеры выполнения программы для отслеживания вызовов функций**

**Для массива:**



OUTPUT.txt:

CREATING A SET #1 FROM A STRING tag list:{1 }

Set #1: {4689F}

CREATING A SET #2 FROM A STRING tag list:{1 2 }

Set #2: {08F}

CREATING A SET #3 FROM A STRING tag list:{1 2 3 }

Set #3: {037ADE}

CREATING A SET #4 FROM A STRING tag list:{1 2 3 4 }

Set #4: {13AF}

CREATING A SET #5tag list:{1 2 3 4 5 }

OR

CREATING A SET #6 BY COPYING FROM A SET #1 tag list:{1 2 3 4 5 6 }

Set #6: {4689F}

CREATING A SET #7 BY COPYING FROM A SET #6 tag list:{1 2 3 4 5 6 7 }

Set #7: {04689F}

DELETING A SET #6 tag list:{1 2 3 4 5 7 }

OR

CREATING A SET #8 BY COPYING FROM A SET #7 tag list:{1 2 3 4 5 7 8 }

Set #8: {04689F}

CREATING A SET #9 BY COPYING FROM A SET #8 tag list:{1 2 3 4 5 7 8 9 }

Set #9: {0346789ADEF}

DELETING A SET #8 tag list:{1 2 3 4 5 7 9 }

OR

CREATING A SET #10 BY COPYING FROM A SET #9 tag list:{1 2 3 4 5 7 9 10 }

Set #10: {0346789ADEF}

CREATING A SET #11 BY COPYING FROM A SET #10 tag list:{1 2 3 4 5 7 9 10 11 }

Set #11: {01346789ADEF}

DELETING A SET #10 tag list:{1 2 3 4 5 7 9 11 }

EQUALS

DELETING A SET #11 tag list:{1 2 3 4 5 7 9 }

DELETING A SET #9 tag list:{1 2 3 4 5 7 }

DELETING A SET #7 tag list:{1 2 3 4 5 }

DELETING A SET #5 tag list:{1 2 3 4 }

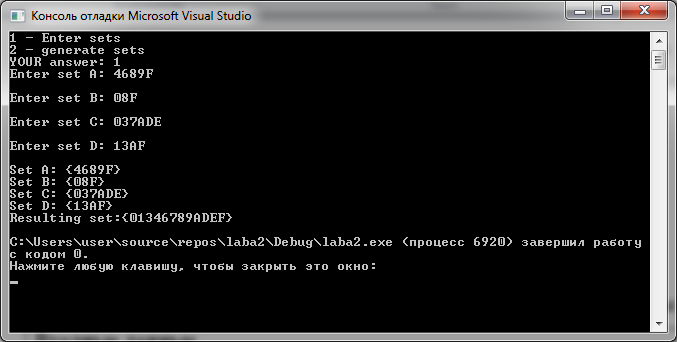
DELETING A SET #4 tag list:{1 2 3 }

DELETING A SET #3 tag list:{1 2 }

DELETING A SET #2 tag list:{1 }

DELETING A SET #1 tag list:{}

**Для списка:**



OUTPUT.txt:

CREATING A SET #1 FROM A STRING tag list:{1 }Alloc elem. memory: 1

adding an element to set #1{4}

Alloc elem. memory: 2

adding an element to set #1{46}

Alloc elem. memory: 3

adding an element to set #1{468}

Alloc elem. memory: 4

adding an element to set #1{4689}

Alloc elem. memory: 5

adding an element to set #1{4689F}

CREATING A SET #2 FROM A STRING tag list:{1 2 }Alloc elem. memory: 6

adding an element to set #2{0}

Alloc elem. memory: 7

adding an element to set #2{08}

Alloc elem. memory: 8

adding an element to set #2{08F}

CREATING A SET #3 FROM A STRING tag list:{1 2 3 }Alloc elem. memory: 9

adding an element to set #3{0}

Alloc elem. memory: 10

adding an element to set #3{03}

Alloc elem. memory: 11

adding an element to set #3{037}

Alloc elem. memory: 12

adding an element to set #3{037A}

Alloc elem. memory: 13

adding an element to set #3{037AD}

Alloc elem. memory: 14

adding an element to set #3{037ADE}

CREATING A SET #4 FROM A STRING tag list:{1 2 3 4 }Alloc elem. memory: 15

adding an element to set #4{1}

Alloc elem. memory: 16

adding an element to set #4{13}

Alloc elem. memory: 17

adding an element to set #4{13A}

Alloc elem. memory: 18

adding an element to set #4{13AF}

CREATING A SET #5 tag list:{1 2 3 4 5 }

OR

CREATING A SET #6 BY COPYING FROM A SET #1 tag list:{1 2 3 4 5 6 }

Alloc elem. memory: 19

adding an element to set #6{4}

Alloc elem. memory: 20

adding an element to set #6{46}

Alloc elem. memory: 21

adding an element to set #6{468}

Alloc elem. memory: 22

adding an element to set #6{4689}

Alloc elem. memory: 23

adding an element to set #6{4689F}

Alloc elem. memory: 24

adding an element to set #6{04689F}

CREATING A SET #7 BY COPYING FROM A SET #6 tag list:{1 2 3 4 5 6 7 }

Alloc elem. memory: 25

adding an element to set #7{4}

Alloc elem. memory: 26

adding an element to set #7{46}

Alloc elem. memory: 27

adding an element to set #7{468}

Alloc elem. memory: 28

adding an element to set #7{4689}

Alloc elem. memory: 29

adding an element to set #7{4689F}

Alloc elem. memory: 30

adding an element to set #7{04689F}

DELETING A SET #6 tag list:{1 2 3 4 5 7 }

OR

CREATING A SET #8 BY COPYING FROM A SET #7 tag list:{1 2 3 4 5 7 8 }

Alloc elem. memory: 31

adding an element to set #8{4}

Alloc elem. memory: 32

adding an element to set #8{46}

Alloc elem. memory: 33

adding an element to set #8{468}

Alloc elem. memory: 34

adding an element to set #8{4689}

Alloc elem. memory: 35

adding an element to set #8{4689F}

Alloc elem. memory: 36

adding an element to set #8{04689F}

Alloc elem. memory: 37

adding an element to set #8{034689F}

Alloc elem. memory: 38

adding an element to set #8{0346789F}

Alloc elem. memory: 39

adding an element to set #8{0346789AF}

Alloc elem. memory: 40

adding an element to set #8{0346789ADF}

Alloc elem. memory: 41

adding an element to set #8{0346789ADEF}

CREATING A SET #9 BY COPYING FROM A SET #8 tag list:{1 2 3 4 5 7 8 9 }

Alloc elem. memory: 42

adding an element to set #9{4}

Alloc elem. memory: 43

adding an element to set #9{46}

Alloc elem. memory: 44

adding an element to set #9{468}

Alloc elem. memory: 45

adding an element to set #9{4689}

Alloc elem. memory: 46

adding an element to set #9{4689F}

Alloc elem. memory: 47

adding an element to set #9{04689F}

Alloc elem. memory: 48

adding an element to set #9{034689F}

Alloc elem. memory: 49

adding an element to set #9{0346789F}

Alloc elem. memory: 50

adding an element to set #9{0346789AF}

Alloc elem. memory: 51

adding an element to set #9{0346789ADF}

Alloc elem. memory: 52

adding an element to set #9{0346789ADEF}

DELETING A SET #8 tag list:{1 2 3 4 5 7 9 }

OR

CREATING A SET #10 BY COPYING FROM A SET #9 tag list:{1 2 3 4 5 7 9 10 }

Alloc elem. memory: 53

adding an element to set #10{4}

Alloc elem. memory: 54

adding an element to set #10{46}

Alloc elem. memory: 55

adding an element to set #10{468}

Alloc elem. memory: 56

adding an element to set #10{4689}

Alloc elem. memory: 57

adding an element to set #10{4689F}

Alloc elem. memory: 58

adding an element to set #10{04689F}

Alloc elem. memory: 59

adding an element to set #10{034689F}

Alloc elem. memory: 60

adding an element to set #10{0346789F}

Alloc elem. memory: 61

adding an element to set #10{0346789AF}

Alloc elem. memory: 62

adding an element to set #10{0346789ADF}

Alloc elem. memory: 63

adding an element to set #10{0346789ADEF}

Alloc elem. memory: 64

adding an element to set #10{01346789ADEF}

CREATING A SET #11 BY COPYING FROM A SET #10 tag list:{1 2 3 4 5 7 9 10 11 }

Alloc elem. memory: 65

adding an element to set #11{4}

Alloc elem. memory: 66

adding an element to set #11{46}

Alloc elem. memory: 67

adding an element to set #11{468}

Alloc elem. memory: 68

adding an element to set #11{4689}

Alloc elem. memory: 69

adding an element to set #11{4689F}

Alloc elem. memory: 70

adding an element to set #11{04689F}

Alloc elem. memory: 71

adding an element to set #11{034689F}

Alloc elem. memory: 72

adding an element to set #11{0346789F}

Alloc elem. memory: 73

adding an element to set #11{0346789AF}

Alloc elem. memory: 74

adding an element to set #11{0346789ADF}

Alloc elem. memory: 75

adding an element to set #11{0346789ADEF}

Alloc elem. memory: 76

adding an element to set #11{01346789ADEF}

DELETING A SET #10 tag list:{1 2 3 4 5 7 9 11 }

EQUALS

Alloc elem. memory: 77

adding an element to set #5{4}

Alloc elem. memory: 78

adding an element to set #5{46}

Alloc elem. memory: 79

adding an element to set #5{468}

Alloc elem. memory: 80

adding an element to set #5{4689}

Alloc elem. memory: 81

adding an element to set #5{4689F}

Alloc elem. memory: 82

adding an element to set #5{04689F}

Alloc elem. memory: 83

adding an element to set #5{034689F}

Alloc elem. memory: 84

adding an element to set #5{0346789F}

Alloc elem. memory: 85

adding an element to set #5{0346789AF}

Alloc elem. memory: 86

adding an element to set #5{0346789ADF}

Alloc elem. memory: 87

adding an element to set #5{0346789ADEF}

Alloc elem. memory: 88

adding an element to set #5{01346789ADEF}

DELETING A SET #11 tag list:{1 2 3 4 5 7 9 }

DELETING A SET #9 tag list:{1 2 3 4 5 7 }

DELETING A SET #7 tag list:{1 2 3 4 5 }

DELETING A SET #5 tag list:{1 2 3 4 }

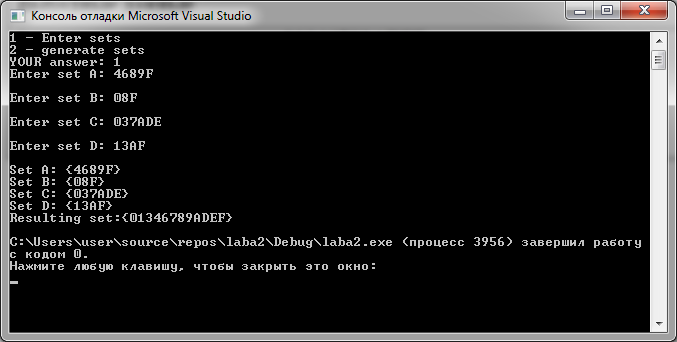
DELETING A SET #4 tag list:{1 2 3 }

DELETING A SET #3 tag list:{1 2 }

DELETING A SET #2 tag list:{1 }

DELETING A SET #1 tag list:{}

**Для массива битов:**



OUTPUT.txt:

CREATING A SET #1 FROM A STRING tag list:{1 }

Set #1: {4689F}

CREATING A SET #2 FROM A STRING tag list:{1 2 }

Set #2: {08F}

CREATING A SET #3 FROM A STRING tag list:{1 2 3 }

Set #3: {037ADE}

CREATING A SET #4 FROM A STRING tag list:{1 2 3 4 }

Set #4: {13AF}

CREATING A SET #5tag list:{1 2 3 4 5 }

OR

CREATING A SET #6 BY COPYING FROM A SET #1 tag list:{1 2 3 4 5 6 }

Set #6: {4689F}

CREATING A SET #7 BY COPYING FROM A SET #6 tag list:{1 2 3 4 5 6 7 }

Set #7: {04689F}

DELETING A SET #6 tag list:{1 2 3 4 5 7 }

OR

CREATING A SET #8 BY COPYING FROM A SET #7 tag list:{1 2 3 4 5 7 8 }

Set #8: {04689F}

CREATING A SET #9 BY COPYING FROM A SET #8 tag list:{1 2 3 4 5 7 8 9 }

Set #9: {0346789ADEF}

DELETING A SET #8 tag list:{1 2 3 4 5 7 9 }

OR

CREATING A SET #10 BY COPYING FROM A SET #9 tag list:{1 2 3 4 5 7 9 10 }

Set #10: {0346789ADEF}

CREATING A SET #11 BY COPYING FROM A SET #10 tag list:{1 2 3 4 5 7 9 10 11 }

Set #11: {01346789ADEF}

DELETING A SET #10 tag list:{1 2 3 4 5 7 9 11 }

EQUALS

DELETING A SET #11 tag list:{1 2 3 4 5 7 9 }

DELETING A SET #9 tag list:{1 2 3 4 5 7 }

DELETING A SET #7 tag list:{1 2 3 4 5 }

DELETING A SET #5 tag list:{1 2 3 4 }

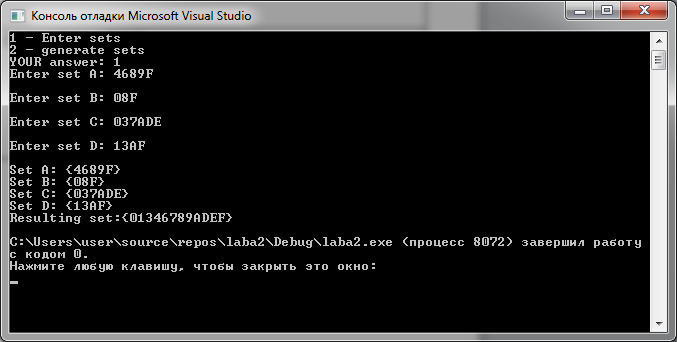
DELETING A SET #4 tag list:{1 2 3 }

DELETING A SET #3 tag list:{1 2 }

DELETING A SET #2 tag list:{1 }

DELETING A SET #1 tag list:{}

**Для битового слова:**



OUTPUT.txt:

CREATING A SET #1 FROM A STRING tag list:{1 }

Set #1: {4689F}

CREATING A SET #2 FROM A STRING tag list:{1 2 }

Set #2: {08F}

CREATING A SET #3 FROM A STRING tag list:{1 2 3 }

Set #3: {037ADE}

CREATING A SET #4 FROM A STRING tag list:{1 2 3 4 }

Set #4: {13AF}

CREATING A SET #5tag list:{1 2 3 4 5 }

OR

CREATING A SET #6 BY COPYING FROM A SET #1 tag list:{1 2 3 4 5 6 }

CREATING A SET #7 BY COPYING FROM A SET #6 tag list:{1 2 3 4 5 6 7 }

DELETING A SET #6 tag list:{1 2 3 4 5 7 }

OR

CREATING A SET #8 BY COPYING FROM A SET #7 tag list:{1 2 3 4 5 7 8 }

CREATING A SET #9 BY COPYING FROM A SET #8 tag list:{1 2 3 4 5 7 8 9 }

DELETING A SET #8 tag list:{1 2 3 4 5 7 9 }

OR

CREATING A SET #10 BY COPYING FROM A SET #9 tag list:{1 2 3 4 5 7 9 10 }

CREATING A SET #11 BY COPYING FROM A SET #10 tag list:{1 2 3 4 5 7 9 10 11 }

DELETING A SET #10 tag list:{1 2 3 4 5 7 9 11 }

EQUALS

DELETING A SET #11 tag list:{1 2 3 4 5 7 9 }

DELETING A SET #9 tag list:{1 2 3 4 5 7 }

DELETING A SET #7 tag list:{1 2 3 4 5 }

DELETING A SET #5 tag list:{1 2 3 4 }

DELETING A SET #4 tag list:{1 2 3 }

DELETING A SET #3 tag list:{1 2 }

DELETING A SET #2 tag list:{1 }

DELETING A SET #1 tag list:{}

**Время выполнения программ**

Самым долгим оказалось выполнение программы, написанной при помощи списков. Не смотря на то, что были перегружены операторы new и delete, списки все-равно выполняют задачу медленнее, чем массив и массив битов. Спискам понадобилось примерно 86000 тиков на выполнение 10 000 000 итераций.

Вторым по скорости (с конца) оказалась реализация через массивы. Ей потребовалось 50 000 тиков на выполнение 10 000 000 итераций.

Ещё быстрее справилась реализация через массив битов. Ей понадобилось 25 000 тиков на выполнение 10 000 000 итераций.

Самой быстрой оказалась реализация через битовое слово. Ей понадобилось около 3 000 тиков для выполнения 10 000 000 итераций

**Анализ выполнения функций**

Как видно из отчета, генерируемого программой в файле OUTPUT.txt, все создаваемые объекты класса «множество» удаляются программой. Следовательно, утечки памяти не происходит.

**Выводы.**

Хоть тенденция скоростей выполнения программ сохранилась: реализация с битовым словом - самая быстрая, а через список – самая медленная, но время выполнения по сравнению с реализацией без классов увеличилось значительно:

\* Для списка – примерно в 8 раз.

\* Для массива – примерно в 10 раз.

\* Для массива битов – примерно в 50 раз.

\* Для битового слова – примерно в 100 раз.

Такое значительное увеличение времени можно объяснить тем, что во время выполнения операций несколько раз создаются новые множества, которые потом удаляются (чего не происходило без классов). Особенно это заметно в реализации через битовое слово, поскольку раньше не тратилось время на создание новых элементов множества, а только выполнялась операция побитового ИЛИ, которая является крайне быстрой.

Чтобы уменьшить количество создаваемых множеств, можно было бы использовать перемещающий оператор «равно» вместо копирующего. Это, в теории, должно значительно ускорить выполнение программы.

Текст программы:

(Приведен вариант только для отслеживания вызовов функций, так как вариант для отслеживания времени почти идентичный)

//main.cpp//

#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include "rand\_set.h"

//#include "array\_Set.h"

//#include "byte\_set.h"

//#include "word\_set.h"

//#include "list\_set.h"

using namespace std;

const int max\_len = 16;

int main()

{

char str[80];

char\* buffer;

int gen = 0;

ofstream fout("OUTPUT.txt");

Set::set\_fout(&fout);

srand(time(0));

cout << "1 - Enter sets\n2 - generate sets\nYOUR answer: ";

cin >> gen;

if (gen == 1) {

cout << "Enter set A: ";

cin >> str;

buffer = new char[strlen(str) + 1];

strcpy\_s(buffer, strlen(str) + 1, str);

}

else buffer = random\_set();

Set a(buffer);

cout << endl;

delete[] buffer;

if (gen == 1) {

cout << "Enter set B: ";

cin >> str;

buffer = new char[strlen(str) + 1];

strcpy\_s(buffer, strlen(str) + 1, str);

}

else buffer = random\_set();

Set b(buffer);

cout << endl;

delete[] buffer;

if (gen == 1) {

cout << "Enter set C: ";

cin >> str;

buffer = new char[strlen(str) + 1];

strcpy\_s(buffer, strlen(str) + 1, str);

}

else buffer = random\_set();

Set c(buffer);

cout << endl;

delete[] buffer;

if (gen == 1) {

cout << "Enter set D: ";

cin >> str;

buffer = new char[strlen(str) + 1];

strcpy\_s(buffer, strlen(str) + 1, str);

}

else buffer = random\_set();

Set d(buffer);

cout << endl;

delete[] buffer;

Set e;

e = a | b | c | d;

cout << "Set A: ";

a.show(&std::cout);

cout << "\n";

cout << "Set B: ";

b.show(&cout);

cout << "\n";

cout << "Set C: ";

c.show(&cout);

cout << "\n";

cout << "Set D: ";

d.show(&cout);

cout << "\n";

cout << "Resulting set:";

e.show(&cout);

cout << "\n";

return 0;

}

//array\_set.h//

#pragma once

#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include "tag\_list.h"

class Set

{

private:

char\* elements;

static const char universum[];

static const int max\_len;

static int tag;

const int own\_tag;

static std::ofstream\* fout;

public:

static tag\_list t\_list;

static void set\_fout(std::ofstream\*);

Set(const char str[]);

Set();

Set(const Set&);

~Set();

void show(std::ostream\*);

Set operator|(const Set&);

Set operator&(const Set&);

Set& operator |=(const Set&);

Set& operator &=(const Set&);

Set& operator =(const Set&);

};

//list\_set.h//

#pragma once

#include "tag\_list.h"

#include <iostream>

#include <fstream>

class Set

{

private:

class El {

private:

static const int maxmup;

static El memory[];

static int mup;

static int mup0;

public:

El\* next;

El\* prev;

char value;

El(char);

El();

~El();

static void\* operator new(size\_t);

static void operator delete(void\*, size\_t);

static void mark();

static void release();

static void clear();

void mem\_out();

};

El\* first;

El\* last;

const int own\_tag;

static int tag;

static const char universum[];

void exclude(El\*);

void clear();

static std::ofstream\* fout;

public:

static void set\_fout(std::ofstream\*);

void append(char);

void show(std::ostream\*);

Set(const char str[]);

Set(const Set&);

Set();

~Set();

Set operator |(const Set&);

Set operator &(const Set&);

Set& operator |=(const Set&);

Set& operator &=(const Set&);

Set& operator = (const Set&);

static tag\_list t\_list;

};

//word\_set.h//

#pragma once

#include "tag\_list.h"

#include <fstream>

class Set

{

private:

static const char universum[];

static const int max\_len;

static int tag;

const int own\_tag;

int i, j, counter;

unsigned int word;

static std::ofstream\* fout;

public:

static tag\_list t\_list;

static void set\_fout(std::ofstream\*);

Set(const char str[]);

Set();

Set(const Set&);

~Set();

void show(std::ostream\*);

Set operator|(const Set&);

Set operator&(const Set&);

Set& operator |=(const Set&);

Set& operator &=(const Set&);

Set& operator =(const Set&);

};

//byte\_set.h//

#pragma once

#include "tag\_list.h"

class Set

{

private:

static const char universum[];

static const int max\_len;

static int tag;

const int own\_tag;

char\* array;

static std::ofstream\* fout;

public:

static tag\_list t\_list;

static void set\_fout(std::ofstream\*);

Set(const char str[]);

Set();

Set(const Set&);

~Set();

void show(std::ostream\*);

Set operator|(const Set&);

Set operator&(const Set&);

Set& operator |=(const Set&);

Set& operator &=(const Set&);

Set& operator =(const Set&);

};

//tag\_list.h//

#pragma once

#include <fstream>

#include <iostream>

class tag\_list

{

private:

struct El

{

El\* next;

El\* prev;

int value;

El(int);

};

El\* first;

El\* last;

public:

void append(int);

void exclude(int);

void show(std::ostream\*);

tag\_list();

~tag\_list();

};

//rand\_set.h//

#pragma once

char\* random\_set();

//array\_set.cpp//

#include "array\_Set.h"

const char Set::universum[] = "0123456789ABCDEF";

const int Set::max\_len = 16;

int Set::tag = 1;

tag\_list Set::t\_list;

std::ofstream\* Set::fout = nullptr;

Set Set::operator|(const Set& other)

{

\*fout << "OR\n";

Set ret(\*this);

return(ret |= other);

}

Set Set::operator&(const Set& other)

{

\*fout << "AND\n";

Set ret(\*this);

return(ret &= other);

}

Set& Set::operator|=(const Set& other)

{

int go,i,j;

int counter = 0;

char buffer[max\_len + 1];

for (i = 0; universum[i] != 0; i++) {

go = 1;

for (j = 0; go && elements[j] != 0; j++) {

if (elements[j] == universum[i]) {

buffer[counter++] = universum[i];

go = 0;

}

}

for (j = 0; go && other.elements[j] != 0; j++) {

if (other.elements[j] == universum[i]) {

buffer[counter++] = universum[i];

go = 0;

}

}

}

buffer[counter] = 0;

delete[] elements;

elements = new char[strlen(buffer) + 1];

strcpy\_s(elements, strlen(buffer)+1, buffer);

return \*this;

}

Set& Set::operator=(const Set& other)

{

\*fout << "EQUALS\n";

if (this != &other) {

delete[] elements;

elements = new char[strlen(other.elements) + 1];

strcpy\_s(elements, strlen(other.elements) + 1, other.elements);

}

return \*this;

}

Set& Set::operator &=(const Set& other)

{

char buffer[max\_len + 1];

int counter = 0;

int i, j;

for (i = 0; other.elements[i] != 0; i++) {

for (j = 0; elements[j] != 0; j++) {

if (other.elements[i] == elements[j]) {

buffer[counter++] = other.elements[i];

break;

}

}

}

buffer[counter] = 0;

delete[] elements;

elements = new char[strlen(buffer)+1];

strcpy\_s(elements, strlen(buffer)+1, buffer);

return \*this;

}

void Set::show(std::ostream\* file)

{

\*file << "{" << elements << "}";

}

void Set::set\_fout(std::ofstream\* file)

{

fout = file;

}

Set::Set(const char str[]) :own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag << " FROM A STRING" << " ";

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

\*fout << std::endl;

int counter;

int i, j;

counter = 0;

for (i = 0; universum[i] != 0; i++) {

for (j = 0; str[j] != 0; j++) {

if (str[j] == universum[i]) {

counter++;

break;

}

}

}

elements = new char[counter + 2];

counter = 0;

for (i = 0; universum[i] != 0; i++) {

for (j = 0; str[j] != 0; j++) {

if (str[j] == universum[i]) {

elements[counter++] = universum[i];

break;

}

}

}

elements[counter] = 0;

\*fout << "Set #" << own\_tag << ": ";

show(fout);

\*fout << std::endl;

}

Set::Set() :own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag;

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

\*fout << std::endl;

elements = new char[1];

elements[0] = 0;

}

Set::Set(const Set& other) :own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag << " BY COPYING FROM A SET #" << other.own\_tag << " ";

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

\*fout << std::endl;

elements = new char[strlen(other.elements)+1];

strcpy\_s(elements, strlen(other.elements) + 1,other.elements);

\*fout << "Set #" << own\_tag << ": ";

show(fout);

\*fout << std::endl;

}

Set::~Set()

{

\*fout << "DELETING A SET #" << own\_tag << " ";

t\_list.exclude(own\_tag);

t\_list.show(fout);//deleting a tag

\*fout << std::endl;

delete[] elements;

}

//list\_set.cpp//

#include "list\_set.h"

const char Set::universum[] = "0123456789ABCDEF";

int Set::tag = 1;

tag\_list Set::t\_list;

const int Set::El::maxmup = 10000;

Set::El Set::El::memory[maxmup];

int Set::El::mup = 0;

int Set::El::mup0 = 0;

std::ofstream\* Set::fout = nullptr;

Set::El::El(char val) {

prev = nullptr;

next = nullptr;

value = val;

}

Set::El::El() :next(nullptr), prev(nullptr), value('!')

{

}

Set::El::~El() {

}

void\* Set::El::operator new(size\_t)

{

\*fout << "Alloc elem. ";

void\* ret = nullptr;

if (mup < maxmup) {

ret = &memory[mup++];

}

\*fout << "memory: " << mup << std::endl;

return ret;

}

void Set::El::operator delete(void\*, size\_t)

{

}

void Set::El::mark()

{

mup0 = mup;

}

void Set::El::release()

{

mup = mup0;

}

void Set::El::clear()

{

mup = 0;

}

void Set::El::mem\_out()

{

std::cout << "Current memory: " << mup << std::endl;

}

void Set::set\_fout(std::ofstream\* file)

{

fout = file;

}

void Set::append(char value)

{

El\* cur;

cur = new El(value);

if (first == nullptr) {

first = cur;

last = cur;

}

else {

last->next = cur;

cur->prev = last;

last = cur;

}

\*fout << "adding an element to set #" << own\_tag;

show(fout);

\*fout << "\n";

}

void Set::show(std::ostream\* file)

{

int i;

El\* cur;

\*file << "{";

for (i = 0; universum[i] != 0; i++) {

for (cur = first; cur != nullptr; cur = cur->next) {

if (universum[i] == cur->value) {

\*file << universum[i];

break;

}

}

}

\*file << "}";

}

void Set::exclude(El\* elem)

{

if (elem->next == nullptr) {

last = elem->prev;

}

else {

elem->next->prev = elem->prev;

}

if (elem->prev == nullptr) {

first = elem->next;

}

else {

elem->prev->next = elem->next;

}

\*fout << "Excluding element " << elem->value << "from set #" << own\_tag;

show(fout);

\*fout << "\n";

delete elem;

}

void Set::clear()

{

El\* oth;

El\* cur;

for (cur = first; cur != nullptr;) {

oth = cur->next;

delete cur;

cur = oth;

}

first = nullptr;

last = nullptr;

}

Set::Set(const char str[]) :own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag << " FROM A STRING" << " ";

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

El\* cur;

first = nullptr;

last = nullptr;

cur = nullptr;

int i;

int j;

for (i = 0; universum[i] != 0; i++) {

for (j = 0; str[j] != 0; j++) {

if (universum[i] == str[j]) {

append(universum[i]);

break;

}

}

}

}

Set::Set(const Set& other) :own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag << " BY COPYING FROM A SET #" << other.own\_tag << " ";

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

\*fout << std::endl;

El\* cur;

first = nullptr;

last = nullptr;

for (cur = other.first; cur != nullptr; cur = cur->next) {

append(cur->value);

}

}

Set::Set() :own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag << " ";

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

\*fout << std::endl;

first = nullptr;

last = nullptr;

}

Set::~Set()

{

\*fout << "DELETING A SET #" << own\_tag <<" ";

t\_list.exclude(own\_tag);

t\_list.show(fout);//deleting a tag

\*fout << std::endl;

El\* cur;

for (; first != nullptr;) {

cur = first->next;

delete first;

first = cur;

}

first = nullptr;

last = nullptr;

}

Set Set::operator|(const Set& other)

{

\*fout << "OR\n";

Set ret(\*this);

ret |= other;

return ret;

}

Set Set::operator&(const Set& other)

{

\*fout << "AND\n";

Set ret(\*this);

ret &= other;

return ret;

}

Set& Set::operator|=(const Set& other)

{

El\* cur;

El\* oth;

int go = 1;

if (this != &other) {

for (oth = other.first; oth != nullptr; oth = oth->next) {

go = 1;

for (cur = first; cur != nullptr; cur = cur->next) {

if (oth->value == cur->value) {

go = 0;

break;

}

}

if (go) append(oth->value);

}

}

return \*this;

}

Set& Set::operator&=(const Set& other)

{

int go;

El\* cur;

El\* oth;

if (this != &other) {

for (cur = first; cur != nullptr;) {

go = 1;

for (oth = other.first; oth != nullptr; oth = oth->next) {

if (oth->value == cur->value) {

go = 0;

break;

}

}

oth = cur->next;

if (go) exclude(cur);

cur = oth;

}

}

return \*this;

}

Set& Set::operator=(const Set& other)

{

\*fout << "EQUALS\n";

El\* oth;

if (this != &other) {

clear();

for (oth = other.first; oth != nullptr; oth = oth->next) {

append(oth->value);

}

}

return \*this;

}

//byte\_set.cpp//

#include "byte\_set.h"

#include <iostream>

int Set::tag = 1;

tag\_list Set::t\_list;

const int Set::max\_len = 16;

const char Set::universum[] = "0123456789ABCDEF";

std::ofstream\* Set::fout = nullptr;

void Set::set\_fout(std::ofstream\* file)

{

fout = file;

}

Set::Set(const char str[]) :own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag << " FROM A STRING" << " ";

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

\*fout << std::endl;

array = new char[max\_len + 1];

int i, j;

for (i = 0; universum[i] != 0; i++) {

int go = 1;

for (j = 0; str[j] != 0; j++) {

if (str[j] == universum[i]) {

array[i] = 1;

go = 0;

break;

}

}

if (go) array[i] = 0;

}

\*fout << "Set #" << own\_tag << ": ";

show(fout);

\*fout << std::endl;

}

Set::Set() : own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag;

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

\*fout << std::endl;

array = new char[max\_len + 1];

int i;

for (i = 0; i <= max\_len; i++) {

array[i] = 0;

}

}

Set::Set(const Set& other) : own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag << " BY COPYING FROM A SET #" << other.own\_tag << " ";

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

\*fout << std::endl;

array = new char[max\_len + 1];

int i;

for (i = 0; i <= max\_len; i++) {

array[i] = other.array[i];

}

\*fout << "Set #" << own\_tag << ": ";

show(fout);

\*fout << std::endl;

}

Set::~Set()

{

\*fout << "DELETING A SET #" << own\_tag << " ";

t\_list.exclude(own\_tag);

t\_list.show(fout);//deleting a tag

\*fout << std::endl;

delete[] array;

}

void Set::show(std::ostream\* file)

{

int i;

\*file << "{";

for (i = 0; universum[i] != 0; i++) {

if (array[i]) \*file << universum[i];

}

\*file << "}";

}

Set& Set::operator |=(const Set& other)

{

int i;

for (i = 0; i <= max\_len; i++) {

if (array[i] | other.array[i]) array[i] = 1;

}

return \*this;

}

Set& Set::operator &=(const Set& other)

{

int i;

for (i = 0; i <= max\_len; i++) {

if (array[i] & other.array[i]) array[i] = 1;

}

return \*this;

}

Set& Set::operator =(const Set& other)

{

\*fout << "EQUALS\n";

int i;

if (this != &other) {

for (i = 0; i <= max\_len; i++) {

array[i] = other.array[i];

}

}

return \*this;

}

Set Set::operator|(const Set& other)

{

\*fout << "OR\n";

Set ret(\*this);

return(ret |= other);

}

Set Set::operator&(const Set& other)

{

\*fout << "AND\n";

Set ret(\*this);

return(ret &= other);

}

//word\_set.cpp//

#include "word\_set.h"

#include <iostream>

const char Set::universum[] = "0123456789ABCDEF";

int Set::tag = 1;

tag\_list Set::t\_list;

std::ofstream\* Set::fout = nullptr;

void Set::set\_fout(std::ofstream\* file)

{

fout = file;

}

Set::Set(const char str[]) :own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag << " FROM A STRING" << " ";

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

\*fout << std::endl;

word = 0;

for (i = 0; universum[i] != 0; i++) {

for (j = 0; str[j] != 0; j++) {

if (str[j] == universum[i]) {

word = word | (1 << i);

break;

}

}

}

\*fout << "Set #" << own\_tag << ": ";

show(fout);

\*fout << std::endl;

}

Set::Set() : own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag;

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

\*fout << std::endl;

word = 0;

}

Set::Set(const Set& other) : own\_tag(tag++)

{

\*fout << "CREATING A SET #" << own\_tag << " BY COPYING FROM A SET #" << other.own\_tag << " ";

t\_list.append(tag - 1);

t\_list.show(fout);//adding a new tag

\*fout << std::endl;

word = other.word;

}

Set::~Set()

{

\*fout << "DELETING A SET #" << own\_tag << " ";

t\_list.exclude(own\_tag);

t\_list.show(fout);//deleting a tag

\*fout << std::endl;

}

void Set::show(std::ostream\* file)

{

\*file << "{";

for (i = 0; universum[i] != 0; i++) {

if ((word & (1 << i)) != 0) \* file << universum[i];

}

\*file << "}";

}

Set Set::operator|(const Set& other)

{

\*fout << "OR\n";

Set ret(\*this);

return (ret |= other);

}

Set Set::operator&(const Set& other)

{

\*fout << "AND\n";

Set ret(\*this);

return (ret&= other);

}

Set& Set::operator|=(const Set& other)

{

word = word | other.word;

return \*this;

}

Set& Set::operator&=(const Set& other)

{

word = word & other.word;

return \*this;

}

Set& Set::operator=(const Set& other)

{

\*fout << "EQUALS\n";

word = other.word;

return \*this;

}

//tag\_list.cpp//

#include "tag\_list.h"

#include <iostream>

#include "list\_set.h"

void tag\_list::append(int i)

{

El\* cur = nullptr;

cur = new El(i);

if (first == nullptr) {

first = cur;

last = cur;

}

else {

last->next = cur;

cur->prev = last;

last = cur;

}

}

void tag\_list::exclude(int val)

{

El\* cur = nullptr;

for (cur = first; cur != nullptr; cur = cur->next) {

if (cur->value == val) {

if (cur->next == nullptr) {

last = cur->prev;

}

else {

cur->next->prev = cur->prev;

}

if (cur->prev == nullptr) {

first = cur->next;

}

else {

cur->prev->next = cur->next;

}

delete cur;

break;

}

}

}

void tag\_list::show(std::ostream\* file)

{

El\* cur = nullptr;

\*file << "tag list:{";

for (cur = first; cur != nullptr; cur = cur->next) \*file << cur->value << " ";

\*file << "}";

}

tag\_list::tag\_list()

{

first = nullptr;

last = nullptr;

}

tag\_list::~tag\_list()

{

El\* a;

El\* cur = nullptr;

for (cur = first; cur != nullptr;) {

a = cur->next;

delete cur;

cur = a;

}

}

tag\_list::El::El(int i)

{

next = nullptr;

prev = nullptr;

value = i;

}

//rand\_set.cpp//

#include "rand\_set.h"

#include <random>

char universum[] = "0123456789ABCDEF";

const int max\_len = 16;

char\* random\_set()

{

int i = 0;

int k = 0;

char\* buffer;

buffer = new char[max\_len + 1];

for (i = 0; universum[i] != 0; i++) if (rand() % 5 == 0) buffer[k++] = universum[i];

buffer[k] = 0;

return buffer;

}