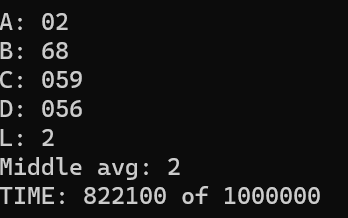
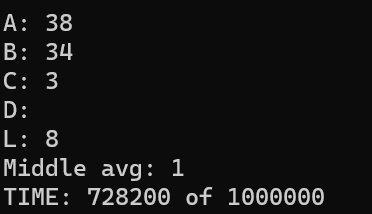
1. Цель работы: исследование эффекта от использования классов.
2. Задание на обработку множеств (можно сослаться на отчёт по теме 1).

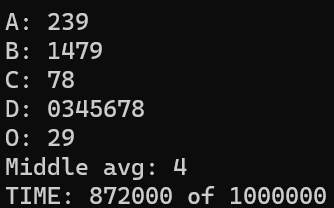
|  |  |  |
| --- | --- | --- |
| 17 | Десятичные цифры | Множество, содержащее цифры из *A*, не являющиеся общими для множеств *B* и *C*, и не встречающиеся в *D* |

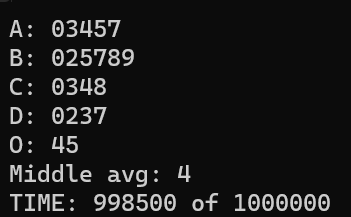
1. Результаты эксперимента с четырьмя структурами данных на основе классов (рисунки с тестами и таблица результатов измерения времени).

А) Массивы

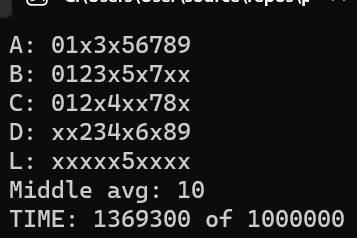
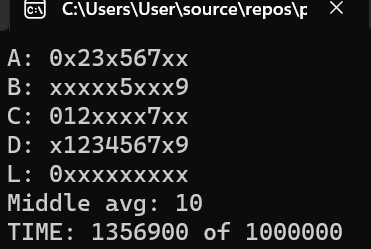


Б) Линейные списки





В) Массивы битов



Г) Машинные слова

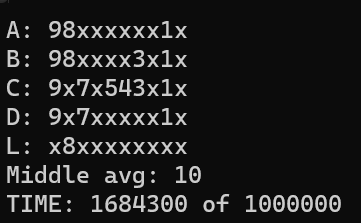
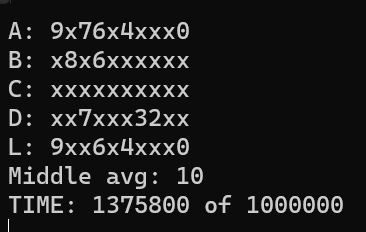
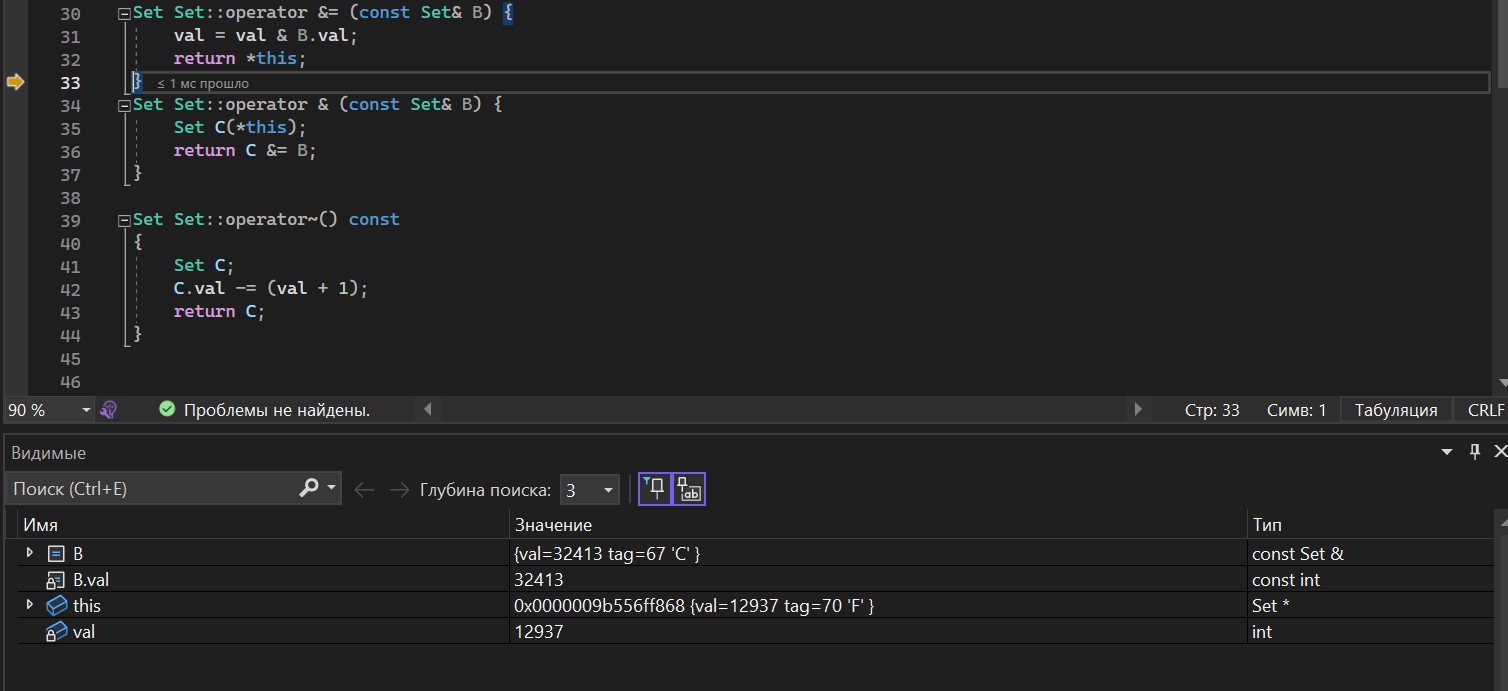
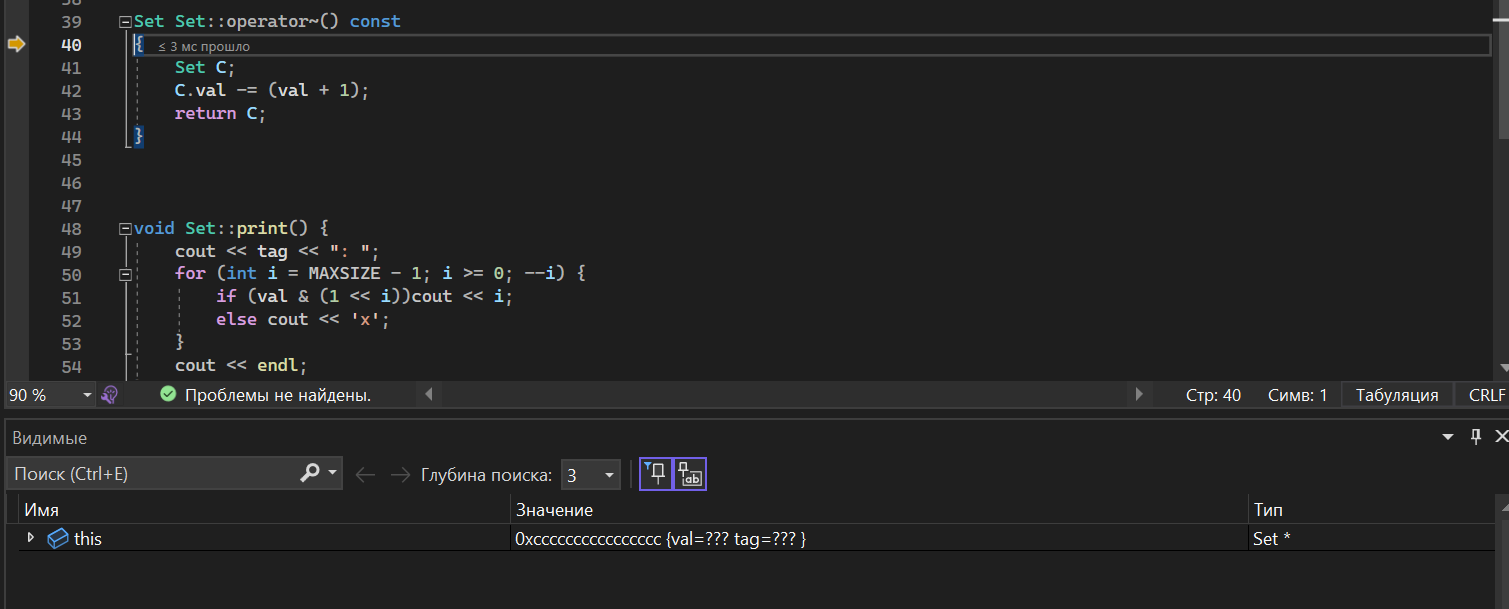
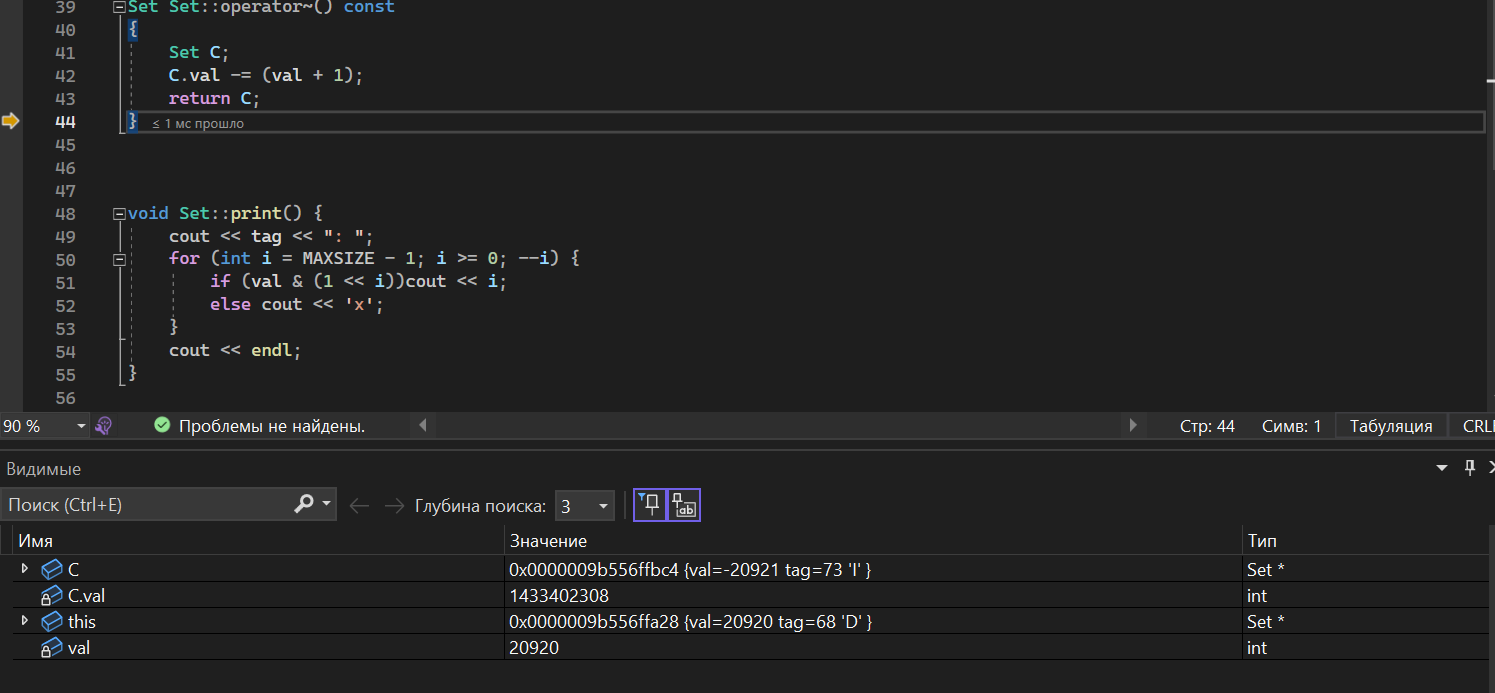
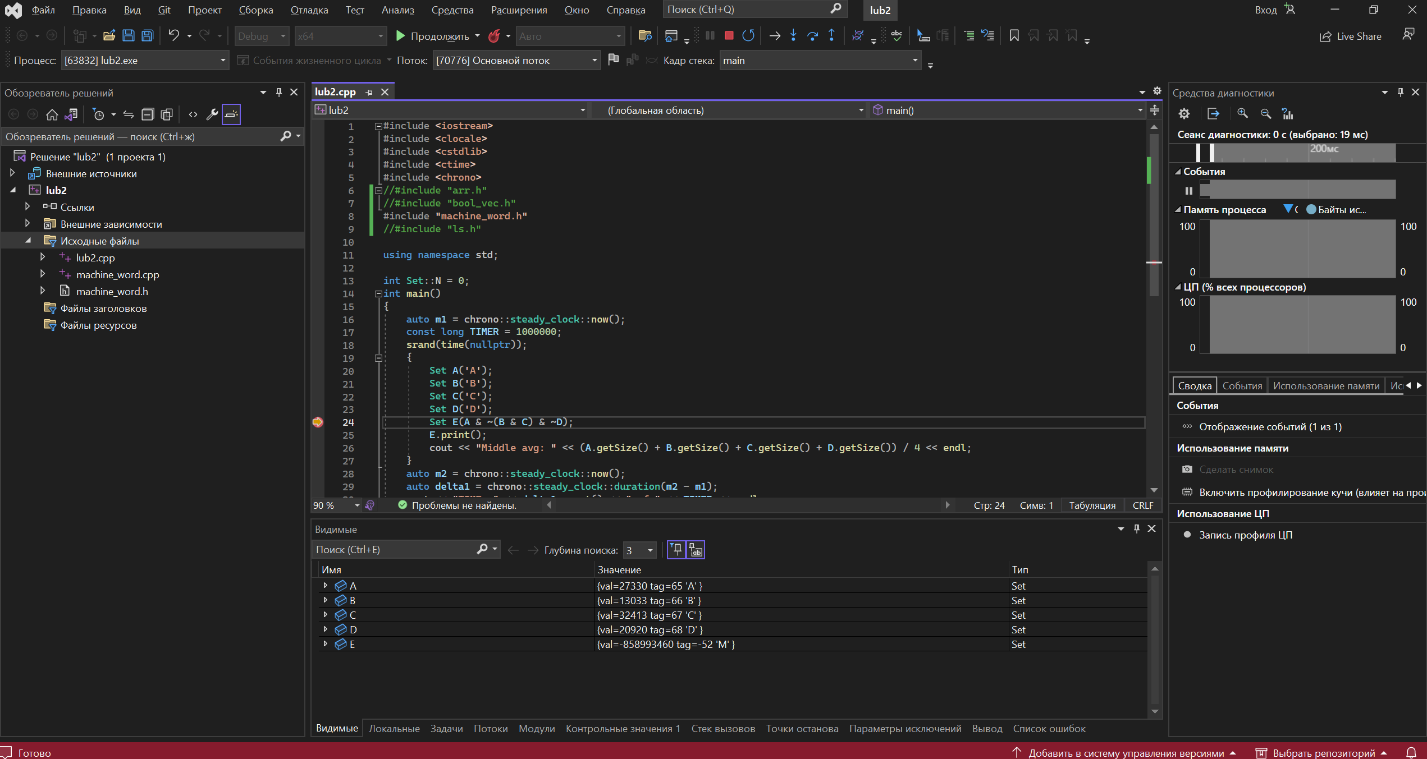


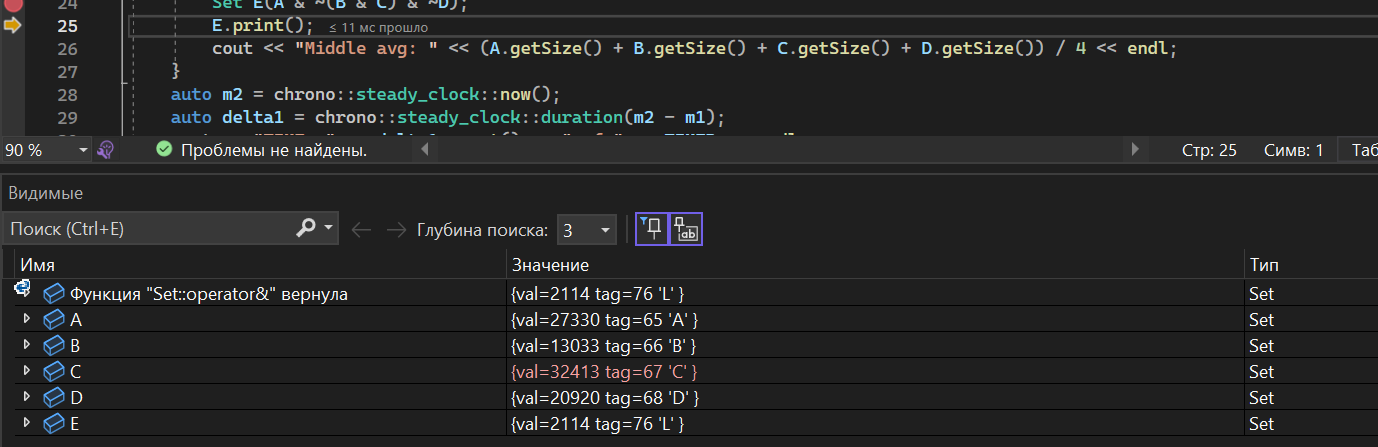
Таблица:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Средняя мощность множеств | 1 | 3 | 4 | 5 | 6 | 8 | 10 |
| Массив | 728200 | 82210 | 73620 | 746040 | 719350 | 727320 | 81140 |
| Линейный список | 872000 | 998500 | 895300 | 905400 | 9976200 | 9425600 | 9741200 |

*\*Примечание: фактическая мощность массива битов и машинного слова равна мощности универсума U=10 вне зависимости от реальной «наполненности» последних, т.е., обрабатываются как нулевые, так и единичные биты.*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест 1 | Тест 2 | Тест 3 |
| Машинное слово | 1375800 | 1684300 | 1061515 |
| Массив битов | 1356900 | 1369300 | 1369400 |

4. Результат эксперимента с отслеживанием отладчика. 



5. Выводы о результатах испытания способов представления множеств в памяти. Заключение о целесообразности и эффекте от использования классов.

Использование объектов в программе обработке множеств позволяет создать удобный функционал, собрав в единую структуру данные и набор необходимых функций для их обработки.

Сравнивая полученные данные с предыдущим отчетом, очевидно, что на работу с объектами затрачивается гораздо больше времени – вероятно, это связано с необходимостью выделения большего объема памяти под более сложный (в сравнении с «обычными» структурами данных) объект, а также создании вспомогательных объектов в ходе вызова функций-членов.

Вспомогательные множества уничтожаются сразу же по завершению работы функции, рабочие – по выходу из main, при этом, уничтожение множеств происходит в порядке, обратном их созданию.

В связи с этим, необходимо осуществление копирования/переноса данных из временного множества, созданного в теле функции, в созданные внутри main, иначе содержащиеся в них данные будут утеряны.

Впрочем, эксперимент с отслеживанием вызовов функций показал, что в действительности применялся лишь конструктор копирования.

6. Список использованных источников.

1) Поздняков С. Н., Рыбин С. В. Дискретная математика: учебник для вузов. М.: Академия, 2008. – 448 с.

2) Хагерти Р. Дискретная математика для программистов. Изд. 2‑е, испр. — М.: Техносфера, 2012. — 400 с.;

3) Новиков Ф. А. Дискретная математика для программистов. — СПб.: Питер, 2000. — 304 с., ил.

4) Страуструп Б. Язык программирования С++. 2‑е доп. изд. — М.: Бином-пресс, 2001. – 1098 с.;

7. Приложение: исходные тексты всех программ.

Lub2.cpp

#include <iostream>

#include <clocale>

#include <cstdlib>

#include <ctime>

#include <chrono>

//#include "arr.h"

//#include "bool\_vec.h"

//#include "machine\_word.h"

//#include "ls.h"

using namespace std;

int Set::N = 0;

int main()

{

auto m1 = chrono::steady\_clock::now();

const long TIMER = 1000000;

srand(time(nullptr));

{

Set A('A');

Set B('B');

Set C('C');

Set D('D');

Set E(A & ~(B & C) & ~D);

E.print();

cout << "Middle avg: " << (A.getSize() + B.getSize() + C.getSize() + D.getSize()) / 4 << endl;

}

auto m2 = chrono::steady\_clock::now();

auto delta1 = chrono::steady\_clock::duration(m2 - m1);

cout << "TIME: " << delta1.count() << " of " << TIMER << endl;

cin.get();

return 0;

}

Arr.h

#pragma once

class Set {

private:

static const int MAXSIZE = 10;

static int N;

int val[MAXSIZE] = { -1 };

int size;

char tag;

public:

Set();

Set(char ch);

Set(const Set&);

~Set();

void print();

Set operator |= (const Set& B);

Set operator | (const Set& B);

Set operator &= (const Set& B);

Set operator & (const Set& B);

Set operator ~();

int getSize() const { return size; }

};

Arr.cpp

#include <iostream>

#include <clocale>

#include <cstdlib>

#include <list>

#include <ctime>

#include <chrono>

#include "arr.h"

Set::~Set() = default;

Set::Set() : tag('E' + N++), size(0) {}

Set::Set(char ch) : tag('A' + N++), size(0) {

for (int i = 0; i < MAXSIZE; i++) if (rand() % 7 == 0) val[size++] = i;

print();

}

Set::Set(const Set& B) : tag('A' + N++), size(B.size) {

for (int i = 0; i < MAXSIZE; i++) val[i] = B.val[i];

}

Set Set::operator |= (const Set& B) {

bool alreadyExist = 0;

for (int i = 0; i < B.size; i++) {

for (int j = 0; j < size; j++) {

if (B.val[i] == val[j]) alreadyExist = 1;

}

if (!alreadyExist) val[size++] = B.val[i];

}

return \*this;

}

Set Set::operator | (const Set& B) {

Set C(\*this);

return C |= B;

}

Set Set::operator &= (const Set& B) {

for (int i = 0; i < size; i++) {

bool isCorrect = 0;

for (int j = 0; j < B.size; j++) {

if (val[i] == B.val[j]) isCorrect = 1;

}

if (!isCorrect) {

for (int k = i; k < size - 1; k++) val[k] = val[k + 1];

size--;

}

}

return \*this;

}

Set Set::operator & (const Set& B) {

Set C(\*this);

return C &= B;

}

void Set::print() {

std::cout << tag << ": ";

for (int i = 0; i < size; i++) std::cout << val[i];

std::cout << std::endl;

}

Set Set::operator~()

{

Set C;

for (int c = 0; c <= 9; ++c) {

bool f = true;

for (int j = 0; j < size; ++j) {

if (c == val[j]) { f = false; break; }

}

if (f) C.val[C.size++] = c;

}

return C;

}

Ls.h

#pragma once

#include <iostream>

#include <stdlib.h>

class Elem {

private:

int i;

Elem\* next;

public:

Elem(int number);

~Elem() { delete next; };

friend class Set;

};

class Set {

private:

static const int MAXSIZE = 10;

static int N;

int size;

Elem\* first;

char tag;

public:

Set();

Set(char ch);

Set(const Set&);

Set(Set&& B);

~Set();

void print();

void add(int x);

Set operator |= (const Set& B);

Set operator | (const Set& B) const;

Set operator &= (const Set& B);

Set operator & (const Set& B)const;

Set operator~ () const;

void swap(Set& other) { std::swap(first, other.first); std::swap(size, other.size); }

int getSize() { return size; }

};

Ls.cpp

#include "ls.h"

Elem::Elem(int number) :i(number), next(nullptr) {}

Set::~Set() {

delete first;

}

void Set::add(int x) {

if (first == nullptr) first = new Elem(x);

else {

Elem\* cur = first;

while (cur->next) cur = cur->next;

cur->next = new Elem(x);

}

}

Set::Set(Set&& B) : tag('A' + N++), size(B.size), first(B.first) {

B.first = nullptr;

}

Set::Set() : tag('A' + N++), size(0), first(nullptr) {}

Set::Set(char ch) : tag('A' + N++), size(0), first(nullptr) {

for (int i = 0; i < MAXSIZE; i++) if (rand() % 2 == 0) { add(i); size++; }

print();

}

Set::Set(const Set& B) : tag('A' + N++), first(nullptr), size(0) {

Elem\* cur = B.first;

while (cur) {

add(cur->i); size++;

cur = cur->next;

}

}

Set Set::operator |= (const Set& B) {

Set C(\*this);

for (Elem\* cur = B.first; cur; cur = cur->next) {

bool alreadyExist = 0;

for (Elem\* check = C.first; check; check = check->next) {

if (cur->i == check->i) alreadyExist = 1;

}

if (!alreadyExist) { C.add(cur->i); C.size++; };

}

swap(C);

return \*this;

}

Set Set::operator | (const Set& B) const {

Set C(\*this);

return C |= B;

}

Set Set::operator &= (const Set& B) {

Set X;

for (Elem\* e = first; e; e = e->next) {

for (Elem\* j = B.first; j; j = j->next) {

if (e->i == j->i) {

X.add(e->i);

X.size++;

}

}

}

swap(X);

return \*this;

}

Set Set::operator & (const Set& B) const {

Set C(\*this);

return C &= B;

}

Set Set::operator~() const

{

Set C;

Elem\* a = C.first;

for (int c = 0; c <= 9; ++c) {

bool f = true;

for (Elem\* e = first; e; e = e->next) {

if (c == e->i) { f = false; break; }

}

if (f) {

C.add(c);

++C.size;

}

}

return C;

}

void Set::print() {

std::cout << tag << ": ";

if (first == nullptr) std::cout << "[EMPTY]" << std::endl;

else {

Elem\* cur = first;

while (cur) {

std::cout << cur->i;

cur = cur->next;

}

}

std::cout << std::endl;

}

Bool\_vec.h

#pragma once

class Set {

private:

static const int MAXSIZE = 10;

static int N;

bool val[MAXSIZE] = { 0 };

char tag;

int size;

public:

Set();

Set(char ch);

Set(const Set&);

~Set();

void print();

Set operator |= (const Set& B);

Set operator | (const Set& B);

Set operator &= (const Set& B);

Set operator & (const Set& B);

Set operator~ () const;

int getSize() { return size; }

};

Bool\_vec.cpp

#include <iostream>

#include <clocale>

#include <cstdlib>

#include "bool\_vec.h"

using namespace std;

Set::Set() : tag('E' + N++), size(0) {

}

Set::Set(char ch) : tag('A' + N++), size(0) {

for (int i = 0; i < MAXSIZE; i++) { val[i] = rand() % 2; size++; }

print();

}

Set::Set(const Set& B) : tag('A' + N++), size(0) {

for (int i = 0; i < MAXSIZE; i++) { val[i] = B.val[i]; size++; }

//cout << "Set " << tag << " has been created!" << endl;

//print();

}

Set Set::operator |= (const Set& B) {

for (int i = 0; i < MAXSIZE; i++) {

if (!val[i] && B.val[i]) size++;

val[i] = val[i] | B.val[i];

}

return \*this;

}

Set Set::operator | (const Set& B) {

Set C(\*this);

return C |= B;

}

Set Set::operator &= (const Set& B) {

for (int i = 0; i < MAXSIZE; i++) {

if (val[i] && B.val[i]) size++;

val[i] = val[i] & B.val[i];

}

return \*this;

}

Set Set::operator & (const Set& B) {

Set C(\*this);

return C &= B;

}

Set::~Set() {}

void Set::print() {

cout << tag << ": ";

for (int i = 0; i < MAXSIZE; i++) {

if (val[i]) cout << i;

else cout << 'x';

}

cout << endl;

}

Set Set::operator~() const

{

Set C;

for (int i = 0; i < size; ++i) {

if (val[i] != true) {

C.val[i] = true;

}

}

return C;

}

Machine\_word.cpp

#include <iostream>

#include <clocale>

#include <cstdlib>

#include "machine\_word.h"

using namespace std;

Set::Set() : val(0), tag('E' + N++) {

//cout << "Empty set " << tag << " has been created!" << endl;

}

Set::Set(char ch) : val(rand()), tag('A' + N++) {

print();

}

Set::Set(const Set& B) : val(B.val), tag('A' + N++) {

//cout << "Set " << tag << " has been created!" << endl;

//print();

}

Set Set::operator |= (const Set& B) {

val = val | B.val;

return \*this;

}

Set Set::operator | (const Set& B) {

Set C(\*this);

return C |= B;

}

Set Set::operator &= (const Set& B) {

val = val & B.val;

return \*this;

}

Set Set::operator & (const Set& B) {

Set C(\*this);

return C &= B;

}

Set Set::operator~() const

{

Set C;

C.val -= (val + 1);

return C;

}

void Set::print() {

cout << tag << ": ";

for (int i = MAXSIZE - 1; i >= 0; --i) {

if (val & (1 << i))cout << i;

else cout << 'x';

}

cout << endl;

}

Machine\_word. H

#pragma once

class Set {

private:

static const int MAXSIZE = 10;

static int N;

int val;

char tag;

public:

Set();

Set(char ch);

Set(const Set&);

void print();

Set operator |= (const Set& B);

Set operator | (const Set& B);

Set operator &= (const Set& B);

Set operator & (const Set& B);

Set operator~ () const ;

int getSize() { return 10; }

};