**«Санкт-Петербургский государственный электротехнический университет**

**«ЛЭТИ» им. В.И.Ульянова (Ленина)»**

**(СПбГЭТУ «ЛЭТИ»)**

Кафедра вычислительной техники

Отчет по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

|  |  |
| --- | --- |
| Студенты гр. 8307 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Зуб В. |
|  | \_\_\_\_\_\_\_\_\_\_\_\_Никулин Л. |
|  |  |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_ Колинько П.Г. |

Тема: **Множества как объект**

# Цель работы

Сравнение процедурного и объектно-ориентированного подходов на примере задачи на обработки множеств.

# Задание

*8 вариант*

Множество, содержащее буквы, имеющиеся во множестве A, но не являющиеся общими для B и C, и все буквы из D

# Формула:

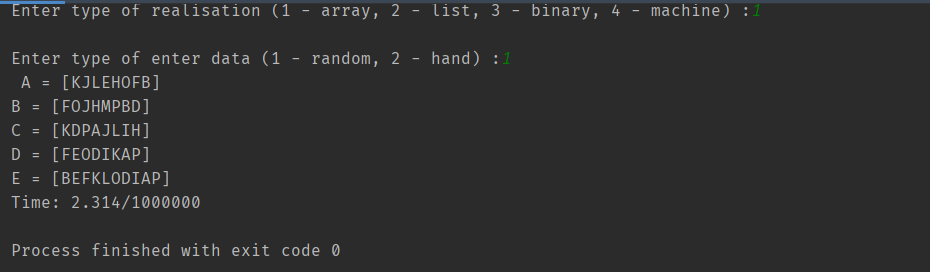
# E = A / (B∩C) ∪ D

# Контрольные тесты

**Исходные данные для работы с множеством, представленным массивом**

|  |  |
| --- | --- |
| Множество А | KJLEHOFB |
| Множество B | FOJHMPBD |
| Множество C | KDPAJLIH |
| Множество D | FEODIKAP |
| Множество E | BEFKLODIAP |

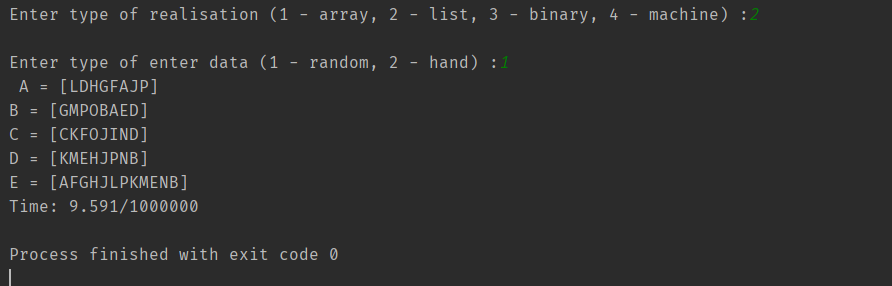
**Вывод программы**



**Исходные данные для работы с множеством, представленным списком**

|  |  |
| --- | --- |
| Множество А | LDHGFAJP |
| Множество B | GMPOBAED |
| Множество C | CKFOJIND |
| Множество D | KMEHJPNB |
| Множество E | AFGHJLPKMENB |

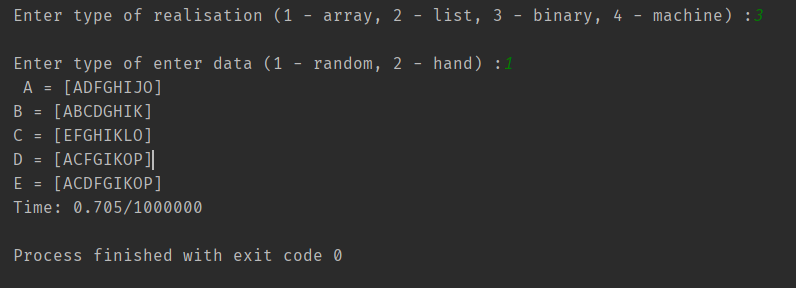
**Вывод программы**



**Исходные данные для работы с множеством, представленным массивом битов**

|  |  |
| --- | --- |
| Множество А | ADFGHIJO |
| Множество B | ABCDGHIK |
| Множество C | EFGHIKLO |
| Множество D | ACFGIKOP |
| Множество E | ACDFGIKOP |

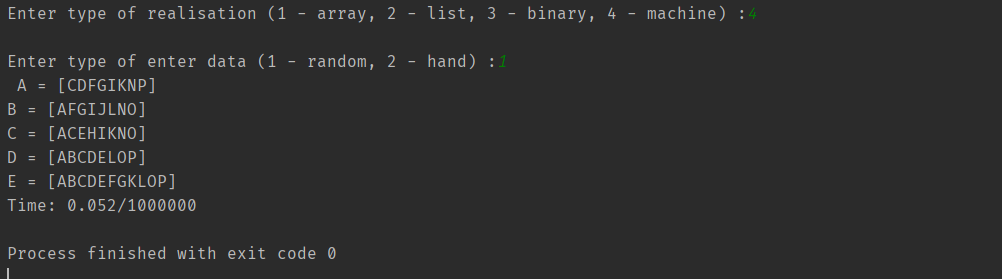
**Вывод программы**



**Исходные данные для работы с множеством, представленным машинным словом**

|  |  |
| --- | --- |
| Множество А | CDFGIKNP |
| Множество B | AFGIJLNO |
| Множество C | ACEHIKNO |
| Множество D | ABCDELOP |
| Множество E | ABCDEFGKLOP |

**Вывод программы**



# Результаты измерений времени обработки

Сравнение результатов измерений времени обработки проводилось на множествах из контрольных тестов выше. Для сравнения использовалась программа из отчета №1 (без использования классов) для каждого способа представления множеств в памяти соответственно.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Время обработки | | | |
| Массив | Список | Массив битов | Машинное слово |
| С использованием классов | 2.314 | 9.591 | 0.705 | 0.052 |
| Без использования классов | 0.595 | 1.504 | 0.17 | 0,007 |

Можно заметить, что время обработки множеств увеличилось с использованием классов.

# Выводы:

Использование классов дает нам удобный интерфейс для работы с программой, возможность создавать сложные типы данных. Организация кода как объектов, значительно облегчает восприятие содержания программы, ее поддерживаемость и тестируемость, время обработки при этом увеличивается.

# Список использованных источников

1. Колинько П.Г. Алгоритмы и структуры данных. Часть 1: Методические указания к самостоятельной работе на ПЭВМ и курсовому проектированию. Вып. 1909. – СПб.: СПбГЭТУ «ЛЭТИ», 2019. – 69 с.
2. Отчет к 1 лабораторной работе по теме «Множества», Никулин Л. и Зуб В.

# Текст программы

**main.cpp**

#include <iostream>

#include <time.h>

#include "SetArray.h"

#include "SetList.h"

#include "SetBinary.h"

#include "SetMachine.h"

#define RANDOM\_SIZE 8

#define TESTS 1000000

using namespace std;

void realisationArray(const string &enter) {

SetArray A('A'), B('B'), C('C'), D('D'), E('E');

if (enter == "1") {

A.randomEnter(RANDOM\_SIZE);

B.randomEnter(RANDOM\_SIZE);

C.randomEnter(RANDOM\_SIZE);

D.randomEnter(RANDOM\_SIZE);

} else {

A.handEnter();

B.handEnter();

C.handEnter();

D.handEnter();

}

auto start = clock();

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

E |= A & (B & C).invert();

E |= D;

}

auto stop = clock();

double time = (stop - start) / (double) CLOCKS\_PER\_SEC;

A.print();

B.print();

C.print();

D.print();

E.print();

cout << "Time: " << time << "/" << TESTS << endl;

}

void realisationList(const string &enter) {

SetList A('A'), B('B'), C('C'), D('D'), E('E');

if (enter == "1") {

A.randomEnter(RANDOM\_SIZE);

B.randomEnter(RANDOM\_SIZE);

C.randomEnter(RANDOM\_SIZE);

D.randomEnter(RANDOM\_SIZE);

} else {

A.handEnter();

B.handEnter();

C.handEnter();

D.handEnter();

}

auto start = clock();

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

E |= A & (B & C).invert();

E |= D;

}

auto stop = clock();

double time = (stop - start) / (double) CLOCKS\_PER\_SEC;

A.print();

B.print();

C.print();

D.print();

E.print();

cout << "Time: " << time << "/" << TESTS << endl;

}

void realisationBinary(const string &enter) {

SetBinary A('A'), B('B'), C('C'), D('D'), E('E');

if (enter == "1") {

A.randomEnter(RANDOM\_SIZE);

B.randomEnter(RANDOM\_SIZE);

C.randomEnter(RANDOM\_SIZE);

D.randomEnter(RANDOM\_SIZE);

} else {

A.handEnter();

B.handEnter();

C.handEnter();

D.handEnter();

}

auto start = clock();

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

E |= A & (B & C).invert();

E |= D;

}

auto stop = clock();

double time = (stop - start) / (double) CLOCKS\_PER\_SEC;

A.print();

B.print();

C.print();

D.print();

E.print();

cout << "Time: " << time << "/" << TESTS << endl;

}

void realisationMachine(const string &enter) {

SetMachine A('A'), B('B'), C('C'), D('D'), E('E');

if (enter == "1") {

A.randomEnter(RANDOM\_SIZE);

B.randomEnter(RANDOM\_SIZE);

C.randomEnter(RANDOM\_SIZE);

D.randomEnter(RANDOM\_SIZE);

} else {

A.handEnter();

B.handEnter();

C.handEnter();

D.handEnter();

}

auto start = clock();

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

E |= A & (B & C).invert();

E |= D;

}

auto stop = clock();

double time = (stop - start) / (double) CLOCKS\_PER\_SEC;

A.print();

B.print();

C.print();

D.print();

E.print();

cout << "Time: " << time << "/" << TESTS << endl;

}

int main() {

srand(time(NULL));

string realisation, enter;

do {

cout << "\nEnter type of realisation (1 - array, 2 - list, 3 - binary, 4 - machine) : ";

cin >> realisation;

}

while(realisation != "1" && realisation != "2" && realisation != "3" && realisation != "4");

do {

cout << "\nEnter type of enter data (1 - random, 2 - hand) : ";

cin >> enter;

}

while(enter != "1" && enter != "2");

if(realisation == "1") {

realisationArray(enter);

} else if(realisation == "2") {

realisationList(enter);

} else if(realisation == "3") {

realisationBinary(enter);

} else {

realisationMachine(enter);

}

return 0;

}

**SetArray.cpp**

#include "SetArray.h"

#include <iostream>

#include <limits>

using namespace std;

char universumArray[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',

'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};

SetArray::SetArray(char C) {

n = 0;

TAG = C;

container = new char[N];

}

SetArray SetArray::copy() const {

SetArray set = SetArray(this->TAG);

set.n = this->n;

set.container = new char[N];

for(int i = 0; i < set.n; i++) {

set.container[i] = this->container[i];

}

return set;

}

bool SetArray::check(char C) const {

for (int i = 0; i < this->n; i++) {

if(this->container[i] == C)

return false;

}

for (int i = 0; i < SetArray::N; i++) {

if(universumArray[i] == C)

return true;

}

return false;

}

void SetArray::add(char C) {

if(!check(C))

return;

this->container[n] = C;

this->n++;

}

int SetArray::getSize() {

return this->n;

}

void SetArray::print() {

cout << this->TAG << " = [";

for (int i = 0; i < this->n; ++i) {

cout << this->container[i];

}

cout << "]" << endl;

}

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

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

if(this->check(B.container[i])) {

this->container[n] = B.container[i];

this->n++;

}

}

return \*this;

}

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

SetArray copy = this->copy();

return copy |= B;

}

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

SetArray temp = B.copy();

for (int i = 0; i < temp.n; ++i) {

if(this->check(temp.container[i])) {

temp.container[i] = '\0';

}

}

this->container = new char[N];

this->n = 0;

for (int j = 0; j < temp.n; ++j) {

if(temp.container[j] != '\0') {

this->container[this->n] = temp.container[j];

this->n++;

}

}

return \*this;

}

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

SetArray copy = this->copy();

return copy &= B;

}

SetArray SetArray::invert() {

SetArray temp = this->copy();

this->n = 0;

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

if(temp.check(universumArray[i])) {

this->container[n] = universumArray[i];

this->n++;

}

}

return \*this;

}

void SetArray::handEnter() {

int t;

char c;

do {

cout << "\nEnter size of SetArray: ";

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

} while(!(cin >> t));

while(getSize() < t) {

do {

cout << this->TAG << "[" << getSize() + 1 << "]: ";

cin.clear();

} while (!(cin >> c));

add(c);

}

}

void SetArray::randomEnter(int n) {

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

while (getSize() < n) {

char c = universumArray[rand() % SetArray::N];

add(c);

}

}

}

**SetList.cpp**

#include <iostream>

#include <limits>

#include "SetList.h"

using namespace std;

char universumList[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',

'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};

SetList::SetList(char C) {

n = 0;

TAG = C;

list = new List();

}

SetList SetList::copy() const {

SetList set = SetList(this->TAG);

set.n = this->n;

set.list = new List;

for(List \*p = list; p; p = p->next) {

set.add(p->c);

}

return set;

}

bool SetList::check(char C) const {

for (List \*p = list; p; p = p->next) {

if (p->c == C)

return false;

}

for (int i = 0; i < SetList::N; i++) {

if(universumList[i] == C)

return true;

}

return false;

}

void SetList::add(char C) {

if (!check(C))

return;

this->n++;

if (list->c == '\0') {

list->c = C;

return;

}

for (List \*p = list; p; p = p->next) {

if (p->next == nullptr) {

List \*current = new List;

current->c = C;

p->next = current;

break;

}

}

}

int SetList::getSize() {

return this->n;

}

void SetList::print() {

cout << this->TAG << " = [";

for (List \*p = list; p; p = p->next) {

cout << p->c;

}

cout << "]" << endl;

}

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

for (List \*p = B.list; p; p = p->next) {

if(this->check(p->c)) {

add(p->c);

this->n++;

}

}

return \*this;

}

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

SetList copy = this->copy();

return copy |= B;

}

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

SetList temp = B.copy();

for (List \*p = temp.list; p; p = p->next) {

if(this->check(p->c)) {

p->c = '\0';

}

}

this->list = new List;

this->n = 0;

for (List \*p = temp.list; p; p = p->next) {

if(p->c != '\0') {

this->add(p->c);

this->n++;

}

}

return \*this;

}

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

SetList copy = this->copy();

return copy &= B;

}

SetList SetList::invert() {

SetList temp = this->copy();

this->list = new List();

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

if(temp.check(universumList[i])) {

this->add(universumList[i]);

}

}

return \*this;

}

void SetList::handEnter() {

int t;

char c;

do {

cout << "\nEnter size of SetList: ";

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

} while(!(cin >> t));

while(getSize() < t) {

do {

cout << this->TAG << "[" << getSize() + 1 << "]: ";

cin.clear();

} while (!(cin >> c));

add(c);

}

}

void SetList::randomEnter(int n) {

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

while (getSize() < n) {

char c = universumList[rand() % SetList::N];

add(c);

}

}

}

**SetBinary.cpp**

#include "SetBinary.h"

#include <iostream>

#include <limits>

using namespace std;

char universumBinary[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',

'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};

SetBinary::SetBinary(char C) {

n = 0;

TAG = C;

container = new bool[N];

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

container[i] = false;

}

}

SetBinary SetBinary::copy() const {

SetBinary set = SetBinary(this->TAG);

set.n = this->n;

set.container = new bool[N];

for (int i = 0; i < SetBinary::N; i++) {

set.container[i] = this->container[i];

}

return set;

}

bool SetBinary::add(char C) {

int k;

for (k = 0; universumBinary[k] != C; k++) {};

if(container[k]) {

return false;

}

container[k] = true;

n++;

return true;

}

int SetBinary::getSize() {

return this->n;

}

void SetBinary::print() {

cout << this->TAG << " = [";

for (int i = 0; i < SetBinary::N; ++i) {

if (container[i])

cout << universumBinary[i];

}

cout << "]" << endl;

}

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

for (int i = 0; i < SetBinary::N; ++i) {

this->container[i] |= B.container[i];

}

return \*this;

}

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

SetBinary copy = this->copy();

return copy |= B;

}

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

for (int i = 0; i < SetBinary::N; ++i) {

this->container[i] &= B.container[i];

}

return \*this;

}

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

SetBinary copy = this->copy();

return copy &= B;

}

SetBinary SetBinary::invert() {

for (int i = 0; i < this->n; ++i) {

container[i] = !container[i];

}

return \*this;

}

void SetBinary::handEnter() {

int t;

char c;

do {

cout << "\nEnter size of SetBinary: ";

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

} while (!(cin >> t));

while (getSize() < t) {

do {

cout << this->TAG << "[" << getSize() + 1 << "]: ";

cin.clear();

} while (!(cin >> c));

add(c);

}

}

void SetBinary::randomEnter(int n) {

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

while (getSize() < n) {

char c = universumBinary[rand() % SetBinary::N];

add(c);

}

}

}

**SetMachine.cpp**

#include <iostream>

#include <limits>

#include "SetMachine.h"

using namespace std;

char universumMachine[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',

'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};

SetMachine::SetMachine(char C) {

n = 0;

TAG = C;

}

SetMachine SetMachine::copy() const {

SetMachine set = SetMachine(this->TAG);

set.n = this->n;

set.container = this->container;

return set;

}

bool SetMachine::add(char C) {

int k;

for (k = 0; C != universumMachine[k]; ++k) {}

if(bool((1 << k) & container)) {

return false;

}

container ^= (1 << k);

n++;

return true;

}

int SetMachine::getSize() {

return this->n;

}

void SetMachine::print() {

cout << this->TAG << " = [";

for (int i = 0; i < SetMachine::N; i++) {

if ((container >> i & 1)) {

cout << universumMachine[i];

}

}

cout << "]" << endl;

}

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

this->container |= B.container;

return \*this;

}

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

SetMachine copy = this->copy();

return copy |= B;

}

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

this->container &= B.container;

return \*this;

}

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

SetMachine copy = this->copy();

return copy &= B;

}

SetMachine SetMachine::invert() {

this->container = ~this->container;

return \*this;

}

void SetMachine::handEnter() {

int t;

char c;

do {

cout << "\nEnter size of SetMachine: ";

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

} while (!(cin >> t));

while (getSize() < t) {

do {

cout << this->TAG << "[" << getSize() + 1 << "]: ";

cin.clear();

} while (!(cin >> c));

add(c);

}

}

void SetMachine::randomEnter(int n) {

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

while (getSize() < n) {

char c = universumMachine[rand() % SetMachine::N];

add(c);

}

}

}