**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 2**

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

Тема: **«Множество как объект»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 | Загуменнов И.М. Пасечный Л.В. |  |
| Преподаватель | Манирагена В. |  |

Санкт-Петербург

2024

**Введение**

**Цель работы:**

Исследование эффекта от использования класса при хранении множества в памяти ЭВМ

Исследование четырёх способов хранения множеств в памяти ЭВМ

**Задание на обработку множеств:** (Вариант 28 - Строчные латинские буквы):

Множество, содержащее буквы, общие для множества A и любого из множеств B, C и D

**Формализация задания:**

E = A & B & C & D

**Контрольные тесты:**

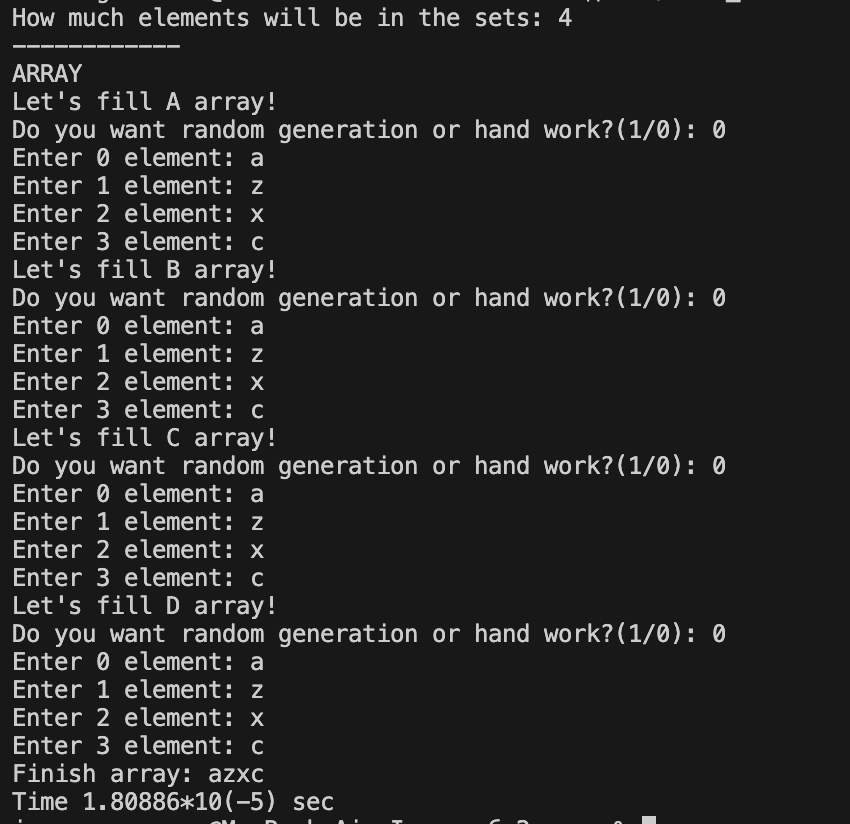


Рисунок – Массивы

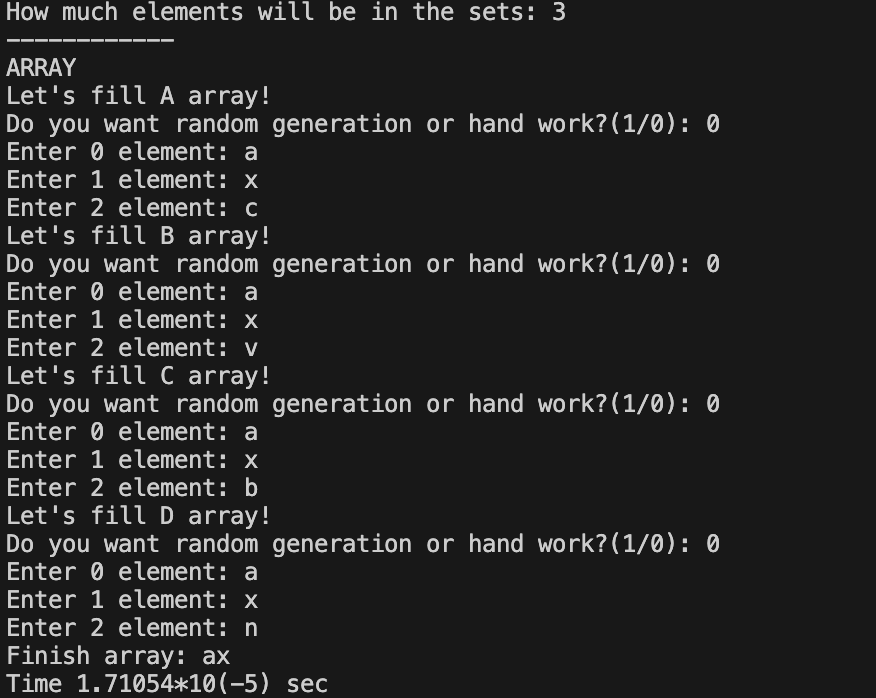


Рисунок – Списки

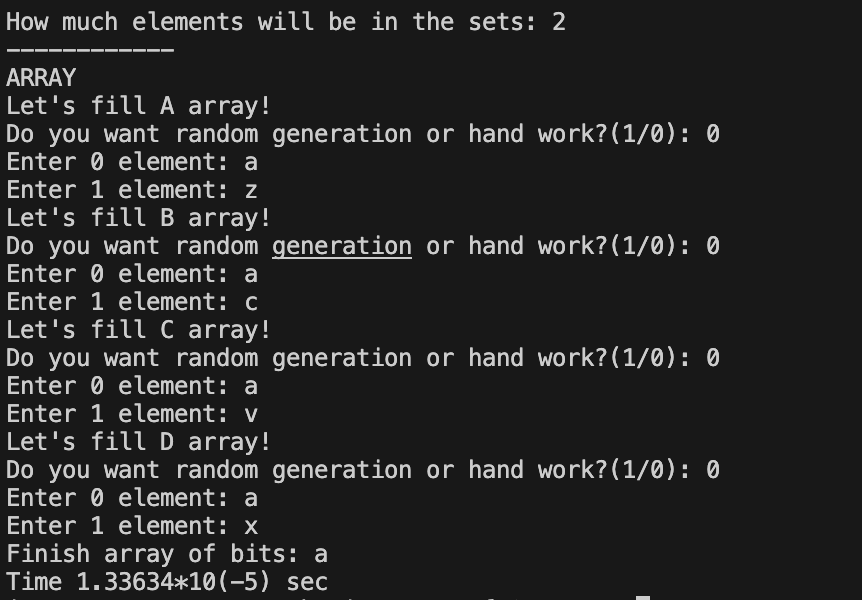


Рисунок – Массивы битов

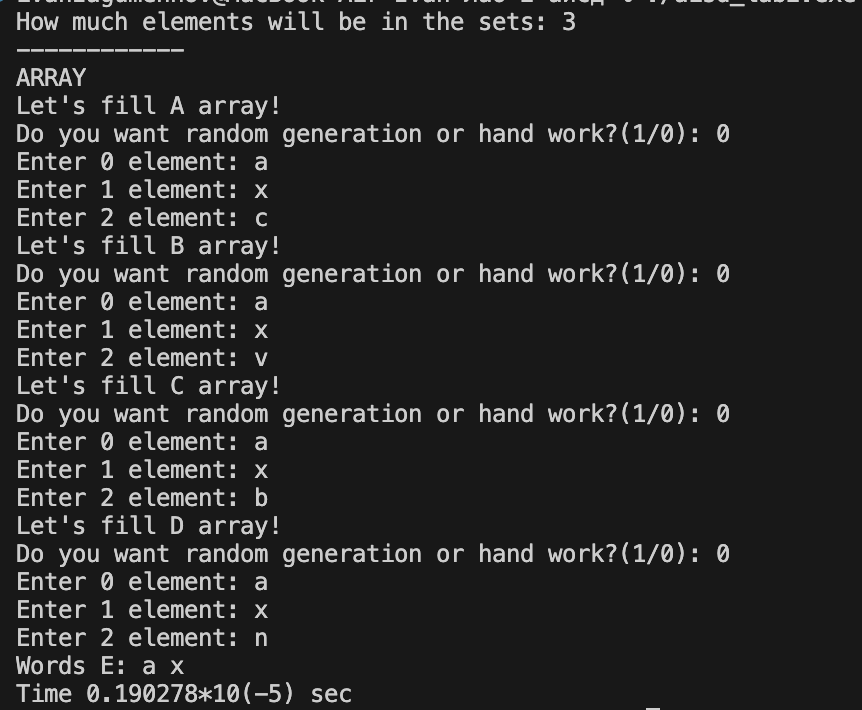


Рисунок – Машинное слово

Выше представлены контрольные примеры, использующие разные способы хранения множеств в памяти ЭВМ, реализованные как объекты класса Set. Данные о множествах были сгенерированы случайным образом. На рис. 3 и рис. 4 также представлен ответ в виде массивов битов и машинного слова (двоичного кода) соответственно

**Результаты эксперимента с четырьмя структурами данных на основе классов**

Временная сложность алгоритмов обработки сохраняется в соответствии с лабораторной работой №1. Аналогично достоинства, недостатки и области применения.

Приведем время выполнения алгоритма с множествами, представленными как объектами

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер попытки | Время выполнения алгоритма для представленного способа (сек) | | | |
| Массивы | Списки | Массивы битов | Машинное слово |
| 1 | 1.69906\*10(-5) | 1.71054\*10(-5) | 1.37125\*10(-5) | 0.177559\*10(-5) |
| 2 | 1.77821\*10(-5) | 1.71354\*10(-5) | 1.37312\*10(-5) | 0.187359\*10(-5) |
| 3 | 1.58106\*10(-5) | 1.70542\*10(-5) | 1.38225\*10(-5) | 0.189329\*10(-5) |
| 4 | 1.69503\*10(-5) | 1.73123\*10(-5) | 1.39012\*10(-5) | 0.172341\*10(-5) |
| 5 | 1.67891\*10(-5) | 1.69992\*10(-5) | 1.31234\*10(-5) | 0.176742\*10(-5) |
| 6 | 1.60123\*10(-5) | 1.72346\*10(-5) | 1.39056\*10(-5) | 0.171124\*10(-5) |
| 7 | 1.59912\*10(-5) | 1.71054\*10(-5) | 1.34672\*10(-5) | 0.177559\*10(-5) |
| 8 | 1.67891\*10(-5) | 1.71453\*10(-5) | 1.37125\*10(-5) | 0.191235\*10(-5) |
| 9 | 1.70322\*10(-5) | 1.71054\*10(-5) | 1.38345\*10(-5) | 0.162351\*10(-5) |
| 10 | 1.67351\*10(-5) | 1.72213\*10(-5) | 1.36931\*10(-5) | 0.173521\*10(-5) |
| Среднее значение | 1.66324\*10(-5) | 1.71321\*10(-5) | 1.37233\*10(-5) | 0.171268\*10(-5) |

**Результат эксперимента с отслеживанием вызовов функций-членов**

**Заключение:**

В ходе выполнения задания успешно были применены операции над множествами для различных типов представления данных: списков, массивов, массивов битов и машинной формы. Мы провели объединение множеств, сформировав итоговое множество, которое содержит элементы из множеств A и B и C и D.

Для каждого представления были реализованы соответствующие алгоритмы, что позволило укрепить навыки работы с различными структурами данных и их эффективной обработки. Применение списков продемонстрировало гибкость и наглядность работы с последовательностями, массивы позволили эффективно управлять структурой данных с фиксированным размером, битовые операции показали преимущества работы с данными на побитовом уровне, что особенно полезно в задачах с ограниченными ресурсами, а машинная форма обеспечила минимизацию пространства и максимальную скорость выполнения операций за счёт низкоуровневого представления данных.

Решение задачи позволило продемонстрировать эффективность использования различных подходов к работе с множествами для решения практических задач по объединению, пересечению и фильтрации данных. Сравнение времени выполнения операций для различных представлений множеств даёт наглядное понимание того, как выбор структуры данных влияет на производительность и эффективность программы в целом.

**Текст программ**

Array.cpp

#include "array.h"

#include <iostream>

#include <stdlib.h>

#include <vector>

#include <time.h>

#define REPEAT\_COUNT 1000000

bool Set::find\_in\_arr(char el, int num)

{ // array

for (int i = 0; i < num; i++)

{

if (set[i] == el)

{

return false;

}

}

return true;

}

void Set::create\_array()

{

char el;

int flag;

cout << "Let's fill " << name << " array!" << endl;

do

{

cout << "Do you want random generation or hand work?(1/0): ";

cin >> flag;

if (flag != 0 && flag != 1)

{

cout << "Try again" << endl;

}

} while (flag != 0 && flag != 1);

for (int i = 0; i < num; i++)

{

if (flag == 0)

{

cout << "Enter " << i << " element: ";

cin >> el;

}

else if (flag == 1)

{

el = rand() % ('z' - 'a' + 1) + 'a';

cout << el << endl;

}

if (find\_in\_arr(el, i) && isalpha(el) && isupper(el) == 0)

{

set.push\_back(el);

}

else

{

if (!find\_in\_arr(el, i))

{

cout << "This element already stay in set. Please try again" << endl;

}

else

{

cout << "Please, enter the small latin symbol, try agai." << endl;

}

i--;

}

}

}

void Set::setName(char name){

this->name = name;

}

void Set::setNum(int num){

this->num = num;

}

vector<char> Set::getKey(){

return this->set;

}

vector<char> Set::check\_key(vector<char> b, vector<char> c, vector<char> d)

{

vector<char> e;

char e1[num], e2[num];

int flag = 0;

for (int i = 0; i < num; i++)

{

for (int j = 0; j < num; j++)

{

if (this->set[i] == b[j])

{

e1[flag] = this->set[i];

flag++;

}

}

}

flag = 0;

for (int i = 0; i < num; i++)

{

for (int j = 0; j < num; j++)

{

if (c[i] == d[j])

{

e2[flag] = c[i];

flag++;

}

}

}

flag = 0;

for (int i = 0; i < num; i++)

{

for (int j = 0; j < num; j++)

{

if (e1[i] == e2[j])

{

e.push\_back(e1[i]);

flag++;

}

}

}

return e;

} // array finish

void Set::print\_key()

{

cout << "Finish array: ";

for (int i = 0; i < num; i++)

{

if (isalpha(set[i]))

{

cout << set[i];

}

}

cout << endl;

}

void Set::setHead(vector<char> a){

this->set = a;

}

void Set::timer(vector<char> b, vector<char> c, vector<char> d){

int t1 = clock();

for (int i = 0 ; i < REPEAT\_COUNT; i++){

check\_key(b, c, d);

}

int t2 = clock();

double t = double(t2 - t1) / CLOCKS\_PER\_SEC;

cout << "Time " << t << "\*10(-5) sec" << endl;

}

Array.h

#include <iostream>

#include <vector>

#include <chrono>

using namespace std;

using Clock = std::chrono::high\_resolution\_clock;

using TimePoint = std::chrono::time\_point<Clock>;

class Set{

    private:

    char name;

    int num;

    vector<char> set;

    TimePoint start\_timer, end\_timer;

    public:

    void setName(char name);

    void setNum(int num);

    vector<char> getKey();

    bool find\_in\_arr(char el, int num);

    void create\_array();

    vector<char> check\_key(vector<char> b, vector<char> c, vector<char> d);

    void print\_key();

    void setHead(vector<char> a);

    void timer(vector<char> b, vector<char> c, vector<char> d);

};

Barray.cpp

#include <iostream>

#include <vector>

#include <time.h>

#include "barray.h"

#define REPEAT\_COUNT 1000000

using namespace std;

bool Set::find\_in\_arr(char el, int num)

{ // array

    for (int i = 0; i < num; i++)

    {

        if (this->set[i] == el)

        {

            return false;

        }

    }

    return true;

}

void Set::setNum(int num){

    this->num = num;

}

void Set::setName(char name){

    this->name = name;

}

void Set::create\_array()

{

    char el;

    int flag;

    cout << "Let's fill " << name << " array!" << endl;

    do

    {

        cout << "Do you want random generation or hand work?(1/0): ";

        cin >> flag;

        if (flag != 0 && flag != 1)

        {

            cout << "Try again" << endl;

        }

    } while (flag != 0 && flag != 1);

    for (int i = 0; i < num; i++)

    {

        if (flag == 0)

        {

            cout << "Enter " << i << " element: ";

            cin >> el;

        }

        else if (flag == 1)

        {

            el = rand() % ('z' - 'a' + 1) + 'a';

            cout << el << endl;

        }

        if (find\_in\_arr(el, i) && isalpha(el) && isupper(el) == 0)

        {

            this->set.push\_back(el);

        }

        else

        {

            if (!find\_in\_arr(el, i))

            {

                cout << "This element already stay in set. Please try again" << endl;

            }

            else

            {

                cout << "Please, enter the small latin symbol, try agai." << endl;

            }

            i--;

        }

    }

    find\_bool();

}

void Set::find\_bool(){

    for (int i = 1; i <= 26; i++)

    {

        this->bits.push\_back(0);

        for (int j = 0; j < num; j++)

        {

            if (this->set[j] == i + 96)

            {

                this->bits[i - 1] = 1;

            }

            else if(this->bits[i-1] != 1)

            {

                this->bits[i - 1] = 0;

            }

        }

    }

}

vector<bool> Set::check\_key(vector<bool> b, vector<bool> c, vector<bool> d)

{

    vector<bool> e(26, 0);

    for (int i = 0; i < 26; i++)

    {

        if (this->bits[i] == 1 && b[i] == 1 && c[i] == 1 && d[i] == 1)

        {

            e[i] = 1;

        }

        else

        {

            e[i] = 0;

        }

    }

    return e;

}

void Set::setHead(vector<bool> e){

    this->bits = e;

}

vector<bool> Set::getKey(){

    return this->bits;

}

void Set::print\_key()

{

    char a;

    cout << "Finish array of bits: ";

    for (int i = 1; i <= 26; i++)

    {

        if (this->bits[i-1])

        {

            a = i + 96;

            cout << a << " ";

        }

    }

}

void Set::timer(vector<bool> b, vector<bool> c, vector<bool> d){

    int t1 = clock();

    for (int i = 0 ; i < REPEAT\_COUNT; i++){

        check\_key(b, c, d);

    }

    int t2 = clock();

    double t = double(t2 - t1) / CLOCKS\_PER\_SEC;

    cout << "Time " << t << "\*10(-5) sec" << endl;

}

Barray.h

#include <iostream>

#include <vector>

using namespace std;

class Set{

    private:

    char name;

    int num;

    vector<bool> bits;

    vector<char> set;

    public:

    bool find\_in\_arr(char el, int num);

    void setNum(int num);

    void setName(char name);

    void setHead(vector<bool> e);

    vector<bool> getKey();

    void create\_array();

    void find\_bool();

    void print\_key();

    vector<bool> check\_key(vector<bool> b, vector<bool> c, vector<bool> d);

    void timer(vector<bool> b, vector<bool> c, vector<bool> d);

};

List.cpp

#include <iostream>

#include <vector>

#include <typeinfo.h>

#include "list.h"

#define REPEAT\_COUNT 1000000

using namespace std;

void Set::create\_list\_from\_array()

{

    Sset \*head = nullptr;

    Sset \*tail = nullptr;

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

    {

        Sset \*newnode = new Sset{set[i], nullptr};

        if (head == nullptr)

        {

            head = newnode;

            head->next = tail;

        }

        else

        {

            tail->next = newnode;

        }

        tail = newnode;

    }

    this->head = head;

}

Sset \*Set::check\_key(Sset \*head2, Sset \*head3, Sset \*head4)

{

    Sset \*current1 = this->head;

    Sset \*current2 = head2;

    Sset \*current3 = head3;

    Sset \*current4 = head4;

    Sset \*set1\_head = nullptr;

    Sset \*set1\_tail = nullptr;

    Sset \*set2\_head = nullptr;

    Sset \*set2\_tail = nullptr;

    Sset \*set3\_head = nullptr;

    Sset \*set3\_tail = nullptr;

    while (current1 != nullptr)

    {

        current2 = head2;

        while (current2 != nullptr)

        {

            if (current1->el == current2->el)

            {

                Sset \*newNode = new Sset{current1->el, nullptr};

                if (set1\_head == nullptr)

                {

                    set1\_head = newNode;

                    set1\_head->next = set1\_tail;

                }

                else

                {

                    set1\_tail->next = newNode;

                }

                set1\_tail = newNode;

            }

            current2 = current2->next;

        }

        current1 = current1->next;

    }

    while (current3 != nullptr)

    {

        current4 = head4;

        while (current4 != nullptr)

        {

            if (current3->el == current4->el)

            {

                Sset \*newNode = new Sset{current3->el, nullptr};

                if (set2\_head == nullptr)

                {

                    set2\_head = newNode;

                    set2\_head->next = set2\_tail;

                }

                else

                {

                    set2\_tail->next = newNode;

                }

                set2\_tail = newNode;

            }

            current4 = current4->next;

        }

        current3 = current3->next;

    }

    Sset \*current5 = set1\_head;

    Sset \*current6 = set2\_head;

    while (current5 != nullptr)

    {

        current6 = set2\_head;

        while (current6 != nullptr)

        {

            if (current5->el == current6->el)

            {

                Sset \*newNode = new Sset{current5->el, nullptr};

                if (set3\_head == nullptr)

                {

                    set3\_head = newNode;

                    set3\_head->next = set3\_tail;

                }

                else

                {

                    set3\_tail->next = newNode;

                }

                set3\_tail = newNode;

            }

            current6 = current6->next;

        }

        current5 = current5->next;

    }

    return set3\_head;

}

void Set::print\_key()

{

    Sset \*current = this->head;

    if (current == nullptr)

    {

        cout << "Error, set is empty" << endl;

    }

    else

    {

        cout << "Finish list: ";

        while (current != nullptr)

        {

            cout << current->el << " ";

            current = current->next;

        }

    }

    cout << endl;

} // finish list

void Set::setNum(int num){

    this->num = num;

}

void Set::setName(char name){

    this->name = name;

}

Sset \*Set::getKey(){

    return this->head;

}

void Set::create\_array()

{

    char el;

    int flag;

    cout << "Let's fill " << name << " array!" << endl;

    do

    {

        cout << "Do you want random generation or hand work?(1/0): ";

        cin >> flag;

        if (flag != 0 && flag != 1)

        {

            cout << "Try again" << endl;

        }

    } while (flag != 0 && flag != 1);

    for (int i = 0; i < num; i++)

    {

        if (flag == 0)

        {

            cout << "Enter " << i << " element: ";

            cin >> el;

        }

        else if (flag == 1)

        {

            el = rand() % ('z' - 'a' + 1) + 'a';

            cout << el << endl;

        }

        if (find\_in\_arr(el, i) && isalpha(el) && isupper(el) == 0)

        {

            this->set.push\_back(el);

        }

        else

        {

            if (!find\_in\_arr(el, i))

            {

                cout << "This element already stay in set. Please try again" << endl;

            }

            else

            {

                cout << "Please, enter the small latin symbol, try agai." << endl;

            }

            i--;

        }

    }

    create\_list\_from\_array();

}

bool Set::find\_in\_arr(char el, int num)

{ // array

    for (int i = 0; i < num; i++)

    {

        if (this->set[i] == el)

        {

            return false;

        }

    }

    return true;

}

void Set::setHead(Sset \*head){

    this->head = head;

}

void Set::timer(Sset \*head2, Sset \*head3, Sset \*head4){

    int t1 = clock();

    for (int i = 0 ; i < REPEAT\_COUNT; i++){

        check\_key(head2, head3, head4);

    }

    int t2 = clock();

    double t = double(t2 - t1) / CLOCKS\_PER\_SEC;

    cout << "Time " << t << "\*10(-5) sec" << endl;

}

List.h

#include <iostream>

#include <vector>

using namespace std;

struct Sset // list

    {

    char el;

    Sset \*next;

    };

class Set{

    private:

    char name;

    int num;

    Sset \*head;

    vector<char> set;

    public:

    void create\_list\_from\_array();

    Sset \*check\_key(Sset \*head2, Sset \*head3, Sset \*head4);

    void print\_key();

    void setNum(int num);

    void setName(char name);

    void setHead(Sset \*head);

    Sset\* getKey();

    void create\_array();

    bool find\_in\_arr(char el, int num);

    void timer(Sset \*head2, Sset \*head3, Sset \*head4);

};

Mword.cpp

#include <iostream>

#include "mword.h"

#include <vector>

#define REPEAT\_COUNT 1000000

using namespace std;

void Set::setName(char name){

    this->name = name;

}

void Set::setNum(int num){

    this->num = num;

}

void Set::create\_array()

{

    char el;

    int flag;

    cout << "Let's fill " << name << " array!" << endl;

    do

    {

        cout << "Do you want random generation or hand work?(1/0): ";

        cin >> flag;

        if (flag != 0 && flag != 1)

        {

            cout << "Try again" << endl;

        }

    } while (flag != 0 && flag != 1);

    for (int i = 0; i < num; i++)

    {

        if (flag == 0)

        {

            cout << "Enter " << i << " element: ";

            cin >> el;

        }

        else if (flag == 1)

        {

            el = rand() % ('z' - 'a' + 1) + 'a';

            cout << el << endl;

        }

        if (find\_in\_arr(el, i) && isalpha(el) && isupper(el) == 0)

        {

            set.push\_back(el);

        }

        else

        {

            if (!find\_in\_arr(el, i))

            {

                cout << "This element already stay in set. Please try again" << endl;

            }

            else

            {

                cout << "Please, enter the small latin symbol, try agai." << endl;

            }

            i--;

        }

    }

    fill\_machine\_word();

}

bool Set::find\_in\_arr(char el, int num)

{ // array

    for (int i = 0; i < num; i++)

    {

        if (set[i] == el)

        {

            return false;

        }

    }

    return true;

}

void Set::setHead(int arr){

    this->word = arr;

}

void Set::fill\_machine\_word()

{

    bool flag;

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

    {

        for (int j = 0; j < sizeof(int) \* 8; j++)

        {

            int result = int(this->set[i] - 96) & (1 << j);

            if (result)

            {

                this->word |= 1 << int(this->set[i] - 97);

            }

        }

    }

}

void Set::print\_binar\_long(int word)

{

    for (int i = (sizeof(int) \* 8 - 1); i >= 0; i--)

    {

        int u1 = 1 << i;

        cout << ((word & u1) ? '1' : '0');

    }

    cout << endl;

}

int Set::getKey(){

    return this->word;

}

int Set::check\_key(int b, int c, int d)

{

    int e = 0;

    for (int i = 0; i < num; i++)

    {

        for (int j = 0; j < sizeof(int)\*8; j++){

        int result = (1 << j);

        if ((word & result) && (result & c) && (result & b) && (result & d))

        {

            e |= word & result;

        }

        }

    }

    return e;

}

void Set::print\_key()

{

    cout << "Words " << this->name << ": ";

    for (int i = 0; i < 26; i++)

    {

        if (this->word >> i & 1)

        {

            char a = i + 97;

            cout << a << " ";

        }

    }

    cout << endl;

}

void Set::timer(int b, int c , int d){

    int t1 = clock();

    for (int i = 0 ; i < REPEAT\_COUNT; i++){

        check\_key(b, c, d);

    }

    int t2 = clock();

    double t = double(t2 - t1) / CLOCKS\_PER\_SEC;

    cout << "Time " << t << "\*10(-5) sec" << endl;

}

Mword.h

#include <iostream>

#include <vector>

using namespace std;

class Set{

    private:

    char name;

    int num;

    vector<char> set;

    int word = 0;

    public:

    void setNum(int num);

    void setName(char name);

    void setHead(int arr);

    void create\_array();

    int getKey();

    bool find\_in\_arr(char el, int num);

    void fill\_machine\_word();

    int check\_key(int b, int c, int d);

    void print\_key();

    void print\_binar\_long(int word);

    void timer(int b, int c , int d);

};