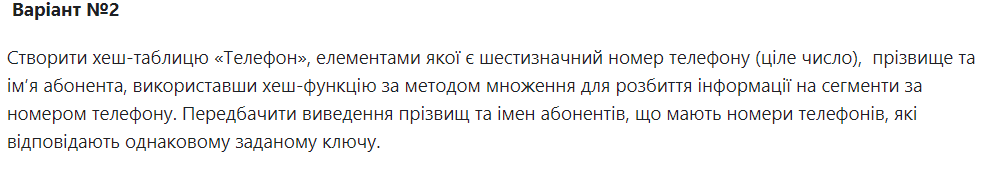
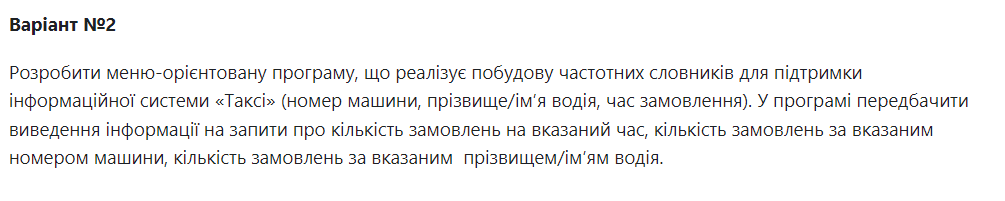
**Звіт**

**Лабораторна робота №3**

**Дисципліна: «Теорія алгоритмів»**



**Завдання 1:**

#include <iostream>

#include <cstring>

#include <cmath>

#include <limits>

using namespace std;

struct Segment

{

    string number;

    string name;

    Segment \*next;

    Segment() { next = nullptr; }

    Segment(string *Number*, string *Name*)

    {

        number = *Number*;

        name = *Name*;

        next = nullptr;

    }

    ~Segment() { delete[] next; }

};

ostream &operator<<(ostream &*os*, Segment \**names*)

{

    Segment \*temp = *names*;

*os* << "(";

    while (temp)

    {

*os* << temp->number;

        if (temp->next != nullptr)

*os* << ", ";

        else

*os* << ")";

        temp = temp->next;

    }

    return *os*;

}

class HashTable

{

private:

    int size;

    Segment \*\*Table;

public:

    HashTable(int *size*);

    ~HashTable();

    int hash(string *x*);

    void insert(string *number*, string *name*);

    bool find\_number(string *number*);

    void find\_number\_return(string *number*);

    void del(string *number*);

    void reset();

    void restruct(int *new\_size*);

    string get(string *number*);

    void output();

};

int main()

{

    int n, option, new\_size;

    string temp\_name, temp\_number;

    cout << "Введіть кількість сегментів у геш-таблиці: ";

    cin >> n;

    HashTable Results(n);

    cout << "1 - Знайти елемент" << endl;

    cout << "2 - Вставити новий елемент" << endl;

    cout << "3 - Видалити елемент" << endl;

    cout << "4 - Реструктуризувати таблицю" << endl;

    cout << "5 - Вивести таблицю на екран" << endl;

    cout << "6 - Вихід" << endl;

    do

    {

        cout << endl

             << "Що ви хочете зробити? Введіть відповідний номер: ";

        cin >> option;

        switch (option)

        {

        case 1:

            cout << "Номер телефону: ";

            cin >> temp\_number;

            if (Results.find\_number(temp\_number) == 1)

            {

                Results.find\_number\_return(temp\_number);

            }

            else

            {

                cout << "Не знайдено!" << endl;

            }

            break;

        case 2:

            cout << "Номер телефону: ";

            cin >> temp\_number;

            cin.ignore();

            cout << "Прізвище та ім'я: ";

            getline(cin, temp\_name);

            Results.insert(temp\_number, temp\_name);

            break;

        case 3:

            cout << "Прізвище та ім'я студента: ";

            cin >> temp\_number;

            Results.del(temp\_number);

            break;

        case 4:

            cout << "Нова кількість сегментів у таблиці: ";

            cin >> new\_size;

            Results.restruct(new\_size);

            break;

        case 5:

            Results.output();

            break;

        default:

            exit(0);

        }

    } while (option != 5);

    return 0;

}

HashTable::HashTable(int *size*)

{

    this->size = *size*;

    Table = new Segment \*[*size*];

    reset();

}

HashTable::~HashTable()

{

    delete[] Table;

}

void HashTable::reset()

{

    for (int i = 0; i < size; i++)

    {

        Table[i] = nullptr;

    }

}

bool HashTable::find\_number(string *x*)

{

    Segment \*current = Table[hash(*x*)];

    while (current != nullptr && current->number.find(*x*) == string::npos)

    {

        current = current->next;

    }

    if (current == nullptr)

        return false;

    else

        return true;

}

void HashTable::find\_number\_return(string *x*)

{

    Segment \*current = Table[hash(*x*)];

    string temp;

    while (current != nullptr)

    {

        if (current->number.find(*x*) != string::npos)

        {

            cout << current->number << " - " << current->name << endl;

        }

        current = current->next;

    }

}

void HashTable::insert(string *number*, string *name*)

{

    Segment \*temp;

    int index = hash(*number*);

    if (!find\_number(*number*))

    {

        if (Table[index] == nullptr)

        {

            temp = new Segment(*number*, *name*);

            Table[index] = temp;

        }

        else

        {

            temp = Table[index];

            while (temp->next != nullptr)

            {

                temp = temp->next;

            }

            temp->next = new Segment(*number*, *name*);

        }

    }

}

void HashTable::del(string *number*)

{

    if (find\_number(*number*))

    {

        int index = hash(*number*);

        if (Table[index]->number == *number*)

        {

            Table[index] = Table[index]->next;

        }

        else

        {

            Segment \*temp = Table[index];

            while (temp->next != nullptr && temp->next->number != *number*)

            {

                temp = temp->next;

            }

            temp->next = temp->next->next;

        }

    }

}

void HashTable::restruct(int *new\_size*)

{

    HashTable NewTable(*new\_size*);

    Segment \*temp;

    for (int i = 0; i < *new\_size*; i++)

    {

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

        {

            temp = Table[j];

            while (temp != nullptr)

            {

                NewTable.insert(temp->number, temp->name);

                temp = temp->next;

            }

        }

    }

    size = *new\_size*;

    Table = new Segment \*[*new\_size*];

    reset();

    for (int i = 0; i < *new\_size*; i++)

    {

        temp = NewTable.Table[i];

        while (temp != nullptr)

        {

            insert(temp->number, temp->name);

            temp = temp->next;

        }

    }

}

int HashTable::hash(string *number*)

{

    int sum = 0;

    for (auto letter : *number*)

    {

        sum += letter;

    }

    return sum % size;

}

string HashTable::get(string *number*)

{

    if (find\_number(*number*))

    {

        Segment \*temp = Table[hash(*number*)];

        while (temp != nullptr && temp->number != *number*)

        {

            temp = temp->next;

        }

        return temp->name;

    }

    return 0;

}

void HashTable::output()

{

    Segment \*temp;

    for (int i = 0; i < size; i++)

    {

        if (Table[i] != nullptr)

        {

            cout << i << ": ";

            temp = Table[i];

            while (temp != nullptr)

            {

                cout << "(" << temp->number << " - " << temp->name << ")";

                if (temp->next != nullptr)

                    cout << ", ";

                else

                    cout << " ";

                temp = temp->next;

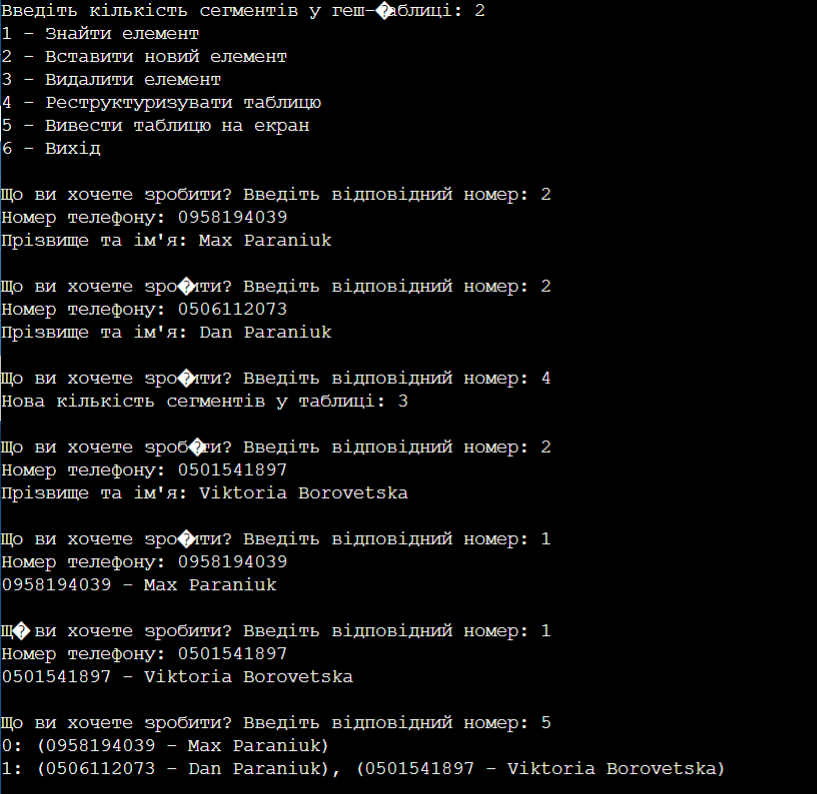
            }

            cout << endl;

        }

    }

}



**Завдання 2:**

#include <iostream>

#include <vector>

using namespace std;

struct Info

{

    string number, name, time;

    Info(){};

    Info(string *Name*, string *Number*, string *Time*)

    {

        number = *Number*;

        name = *Name*;

        time = *Time*;

    }

};

struct Dict

{

    vector<Info> key;

    int count\_number;

    int count\_name;

    int count\_time;

    Dict \*left = nullptr;

    Dict \*right = nullptr;

    Dict(Info *x*) : count\_number(1) { key.push\_back(*x*); }

};

void print(Dict \**dict*, string *prefix* = "", bool *isTail* = true, string *sb* = "");

void search(Info *x*, Dict \*&*dict*);

Dict \*locate(Info *x*, Dict \**dict*);

void get\_count(Dict \**dict*, string *word*, int *key*);

int main(int *argc*, char const \**argv*[])

{

    Dict \*root = nullptr, \*temp = nullptr;

    string x, word, t\_name, t\_number, t\_time;

    Info input;

    int n, index = 1, option, flag;

    cout << "1 - Вставити новий елемент" << endl;

    cout << "2 - Пошук за критерієм" << endl;

    cout << "3 - Вивести словник" << endl;

    cout << "4 - Вийти" << endl;

    do

    {

        cout << endl

             << "Що ви хочете зробити? Введіть відповідний номер: ";

        cin >> option;

        switch (option)

        {

        case 1:

            cout << endl;

            cout << "Введіть номер авто " << index << "-ї особи: ";

            cin >> input.number;

            cin.ignore();

            cout << "Введіть ім'я " << index << "-ї особи: ";

            getline(cin, t\_name);

            input.name = t\_name;

            cout << "Введіть час замовлення " << index << "-ї особи: ";

            cin >> input.time;

            search(input, root);

            index++;

            break;

        case 2:

            cout << "Введіть за чим здійснити пошук(1 - номер авто, 2 - ПІ, 3 - час): ";

            cin >> flag;

            cout << "Введіть дані: ";

            cin >> word;

            get\_count(root, word, flag);

            break;

        case 3:

            cout << endl;

            print(root);

            break;

        case 4:

            exit(0);

        default:

            exit(0);

            break;

        }

    } while (option != 4);

    return 0;

}

void print(Dict \**dict*, string *prefix*, bool *isTail*, string *sb*)

{

    if (*dict*->right != nullptr)

    {

        print(*dict*->right, *prefix* + (*isTail* ? " │   " : "     "), false, *sb*);

    }

    string temp = *prefix* + (*isTail* ? " └── " : " ┌── ") + *dict*->key[0].number + " " + *dict*->key[0].name + " " + *dict*->key[0].time + "\n";

*sb* += temp;

    cout << temp;

    if (*dict*->left != nullptr)

    {

        print(*dict*->left, *prefix* + (*isTail* ? "     " : " │   "), true, *sb*);

    }

}

void search(Info *x*, Dict \*&*dict*)

{

    if (*dict* == nullptr)

    {

*dict* = new Dict(*x*);

    }

    else

    {

        if (*x*.number < *dict*->key[0].number)

            search(*x*, *dict*->left);

        else if (*x*.number > *dict*->key[0].number)

            search(*x*, *dict*->right);

        else

        {

*dict*->count\_number++;

*dict*->key.push\_back(*x*);

        }

        if (*x*.name < *dict*->key[0].name)

            search(*x*, *dict*->left);

        else if (*x*.name > *dict*->key[0].name)

            search(*x*, *dict*->right);

        else

        {

*dict*->count\_name++;

*dict*->key.push\_back(*x*);

        }

        if (*x*.time < *dict*->key[0].time)

            search(*x*, *dict*->left);

        else if (*x*.time > *dict*->key[0].time)

            search(*x*, *dict*->right);

        else

        {

*dict*->count\_time++;

*dict*->key.push\_back(*x*);

        }

    }

}

void get\_count(Dict \**dict*, string *word*, int *key*)

{

    Dict \*temp = *dict*;

    int flag = 0;

    if (*key* == 1)

    {

        while (temp != nullptr)

        {

            if (flag == 1)

            {

                break;

            }

            if (temp->key[0].number < *word*)

                temp = temp->right;

            else if (temp->key[0].number > *word*)

                temp = temp->left;

            else

            {

                for (int i = 0; i < temp->key.size(); i++)

                {

                    if (temp->key[i].number == *word*)

                        cout << "Кількість замовлень за вказаним номером: " << temp->count\_number << endl;

                    flag = 1;

                    break;

                }

            }

        }

    }

    else if (*key* == 2)

    {

        while (temp != nullptr)

        {

            if (flag == 1)

            {

                break;

            }

            if (temp->key[0].name < *word*)

                temp = temp->right;

            else if (temp->key[0].name > *word*)

                temp = temp->left;

            else

            {

                for (int i = 0; i < temp->key.size(); i++)

                {

                    if (temp->key[i].name == *word*)

                        cout << "Кількість замовлень за вказаним іменем: " << temp->count\_name + 1 << endl;

                    flag = 1;

                    break;

                }

            }

        }

    }

    else if (*key* == 3)

    {

        while (temp != nullptr)

        {

            if (flag == 1)

            {

                break;

            }

            if (temp->key[0].time < *word*)

                temp = temp->right;

            else if (temp->key[0].time > *word*)

                temp = temp->left;

            else

            {

                for (int i = 0; i < temp->key.size(); i++)

                {

                    if (temp->key[i].time == *word*)

                        cout << "Кількість замовлень на вказаний час: " << temp->count\_time + 1 << endl;

                    flag = 1;

                    break;

                }

            }

        }

    }

    if (flag == 0)

    {

        cout << "Не знайдено" << endl;

    }

}

