

Міністерство освіти і науки України  
Харківський національний університет радіоелектроніки

Лабораторна робота №2  
Дисципліна: Комп'ютерна дискретна математика

Виконав  
Студент групи ПЗПІ-21-1  
Попович Ярослав Васильович

Перевірів  
Асистент кафедри  
Терещенко Гліб Юрійович

Тема: Операції з булевими функціями

Мета: Навчитися створювати калькулятор з булевими виразами.

Індивідуальне завдання:

1. Реалізувати динамічне додавання та оновлювання змінних.
2. Реалізувати можливість вводу команд з консолі.
3. Вивести результат.
4. Підтвердити результати скріншотами.

Код програми:

```
/*
 * create <name>
 * add <name> {objects} ( example: add A {1,2,3} )
 * delete <name> {objects} ( example: del A {2,3} )
 * negation: !<name> -> returns set
 * union: <name_1> | <name_2> -> returns set
 * intersection: <name_1> & <name_2> -> returns set
 * compliment: <name_1> \ <name_2> -> return set
 * show: -> writes all sets in console
 * do: <command> -> writes result of expression in console ( example: do (A\B\C) |(B\A\C) |(C\A\B) |(A&B&C) )
 * ! -> & -> | -> \
 */
#include <iostream>
#include <string>
#include <vector>
#include <set>
#include <map>
#include <algorithm>
#include <stack>
#define DEBUG false
using namespace std;
class mySet {
private:
    string name = "";
    set<string> objects_set;
public:
    mySet(string name="everything") {
        this->name = name;
        if (DEBUG)
            cout << "Set " << name << " has been created.\n";
        if (DEBUG)
            print_objects();
    }
    void add(vector<string> objects) {
        if (DEBUG)
            cout << "add\n";
        for (vector<string>::iterator it = objects.begin(); it != objects.end(); ++it) {
            objects_set.insert(*it);
        }
    }
};
```

```

    }
    if (DEBUG)
        print_objects();
}

void add(set<string> objects) {
    if (DEBUG)
        cout << "add\n";
    for (set<string>::iterator it = objects.begin(); it != objects.end(); ++it) {
        objects_set.insert(*it);
    }
    if (DEBUG)
        print_objects();
}

void del(vector<string> objects) {
    for (vector<string>::iterator it = objects.begin(); it != objects.end(); ++it) {
        objects_set.erase(*it);
    }
    if (DEBUG)
        print_objects();
}

void clear() {
    this->objects_set.clear();
}

// STAFF
vector<string> get_objects() {
    vector<string> res;
    for (set<string>::iterator it = objects_set.begin(); it != objects_set.end(); ++it) {
        res.push_back(*it);
    }
    return res;
}

void print_objects() {
    bool isPrint = false;
    cout << this->name << ": {";
    for (set<string>::iterator it = objects_set.begin(); it != objects_set.end(); ++it) {
        if (isPrint)
            cout << ", " << *it;
        else
            cout << " " << *it;
        isPrint = true;
    }
    cout << " }\n";
}

};

class mySets {
public:
    mySets() {
        cout << " *   create <name>\n" <<
            " *   add <name> {objects} ( example: add A {1,2,3} )\n" <<
            " *   delete <name> {objects} ( example: del A {1,2} )\n" <<
            " *   negation: !<name> -> returns set\n" <<
            " *   union: <name_1>+<name_2> -> returns set\n" <<
            " *   intersection: <name_1>*<name_2> -> returns set\n" <<
            " *   compliment: <name_1>><name_2> -> return set\n" <<
            " *   ! -> * -> + -> >\n" <<
            " *   show: -> writes all sets in console\n" <<
            " *   do: <command> -> writes result of expression in console ( example: do !((!x+!y)*!t+!z) )\n";
    }

    void get_command(string command) {

```

```

if (get_first_word(command) == "show") {
    show();
    return;
}
if (get_first_word(command) == "create") {
    string name = del_first_word(command);
    if (DEBUG)
        cout << "name: " << name << '\n';
    this->sets[name] = new mySet(name);
}
if (get_first_word(command) == "add") {
    string name = del_first_word(command);
    if (DEBUG)
        cout << "name: " << name << '\n';
    vector<string> objects = string_to_objects(command);
    this->sets[name]->add(objects);
    update_everything();
}
if (get_first_word(command) == "del") {
    string name = del_first_word(command);
    if (DEBUG)
        cout << "name: " << name << '\n';
    vector<string> objects = string_to_objects(command);
    this->sets[name]->del(objects);
    update_everything();
}
if (get_first_word(command) == "do") {
    string com = del_only_first_word(command);
    if (DEBUG)
        cout << "com: " << com << '\n';
    operate_command(com);
    // DONE
}
}
void operate_command(string command) { // DONE
    map<char, int> importance;
    stack<mySet*> st_set;
    stack<char> st_operator;
    importance['!'] = 4;
    importance['*'] = 3;
    importance['+'] = 2;
    importance['>'] = 1;
    if (DEBUG)
        cout << "\n\n\n\noperate command\n\n";
    while (!(command.empty() && st_operator.empty() && st_set.size() == 1)) {
        if (command.empty() || importance[command[0]] > 0 || command[0] == '(' || command[0] == ')') { //
rewrite
            if (DEBUG)
                cout << "into symbol\n";
            if ((command[0] != ')') && (st_operator.empty() || importance[st_operator.top()] <
importance[command[0]] || command[0] == '(' || (importance[st_operator.top()] == importance[command[0]]
&& command[0] == '!'))) { // fixed bug with !!
                if (DEBUG)
                    cout << "adds symbol: " << command[0] << '\n';
                st_operator.push(command[0]);
                command = del_first_symbol(command);
            }
        }
        else {
            if (command[0] == ')') {

```

```

command = del_first_symbol(command);
while (st_operator.top() != '(') {
    char symb = st_operator.top();
    st_operator.pop();
    if (symb == '!') {
        mySet* set_1 = new mySet();
        *set_1 = *(st_set.top());
        st_set.pop();
        mySet* set_add = new mySet();
        *set_add = *(nigation(set_1));
        st_set.push(set_add);
    }
    else {
        mySet* set_2 = new mySet();
        *set_2 = *(st_set.top());
        st_set.pop();
        mySet* set_1 = new mySet();
        *set_1 = *(st_set.top());
        st_set.pop();
        mySet* set_add = new mySet();
        if (symb == '*')
            *set_add = *(intersection(set_1, set_2));
        if (symb == '+')
            *set_add = *(union_set(set_1, set_2));
        if (symb == '>')
            *set_add = *(compliment(set_1, set_2));
        st_set.push(set_add);
    }
}
st_operator.pop();
}
else {
    char symb = st_operator.top();
    st_operator.pop();
    if (DEBUG)
        cout << "starting making operations with symb: " << symb << '\n';
    if (symb == '!') {
        mySet* set_1 = new mySet();
        *set_1 = *(st_set.top());
        st_set.pop();
        mySet* set_add = new mySet();
        *set_add = *(nigation(set_1));
        st_set.push(set_add);
    }
    else {
        mySet* set_2 = new mySet();
        *set_2 = *(st_set.top());
        st_set.pop();
        mySet* set_1 = new mySet();
        *set_1 = *(st_set.top());
        st_set.pop();
        mySet* set_add = new mySet();
        if (symb == '*')
            *set_add = *(intersection(set_1, set_2));
        if (symb == '+')
            *set_add = *(union_set(set_1, set_2));
        if (symb == '>')
            *set_add = *(compliment(set_1, set_2));
        st_set.push(set_add);
    }
}

```

```

        }
    }
}
} else {
    mySet* set_add = new mySet();
    string set_name = get_part_command(command);
    command = del_part_command(command);
    if (DEBUG)
        cout << "adds object: " << set_name << '\n';
    set_add->add(this->sets[set_name]->get_objects());
    st_set.push(set_add);
}
}
st_set.top()->print_objects();
}
void show() {
    this->everything.print_objects();
    for (map<string, mySet*>::iterator it = this->sets.begin(); it != this->sets.end(); ++it) {
        this->sets[it->first]->print_objects();
    }
}
mySet* negation(mySet* to_negate) { // DONE
    string val = to_negate->get_objects()[0];
    if (val == "0") {
        val = "1";
    }
    else if (val == "1") {
        val = "0";
    }
    mySet* result = new mySet();
    result->add(vector<string>{val});
    return result;
}
mySet* union_set(mySet* to_union1, mySet* to_union2) { // DONE
    string val1 = to_union1->get_objects()[0];
    string val2 = to_union2->get_objects()[0];
    mySet* result = new mySet();
    if (val1 == "1" || val2 == "1") {
        result->add(vector<string>{"1"});
    }
    else {
        result->add(vector<string>{"0"});
    }
    if (DEBUG)
        cout << "union\n";
    return result;
}
mySet* intersection(mySet* to_intersect1, mySet* to_intersect2) { // DONE
    string val1 = to_intersect1->get_objects()[0];
    string val2 = to_intersect2->get_objects()[0];
    mySet* result = new mySet();
    if (val1 == "1" && val2 == "1") {
        result->add(vector<string>{"1"});
    }
    else {
        result->add(vector<string>{"0"});
    }
    if (DEBUG)
        cout << "intersect\n";
}

```

```

        return result;
    }
    mySet* compliment(mySet* to_compliment, mySet* from_compliment) { // DONE
        string val1 = to_compliment->get_objects()[0];
        string val2 = from_compliment->get_objects()[0];
        mySet* result = new mySet();
        if (val1 == "1" && val2 == "0") {
            result->add(vector<string>{"0"});
        }
        else {
            result->add(vector<string>{"1"});
        }
        if (DEBUG)
            cout << "intersect\n";
        return result;
    }
private:
    string get_part_command(string command) {
        string result = "";
        for (int i = 0; i < (int)command.length(); ++i) {
            if (command[i] == '!' || command[i] == '*' || command[i] == '+' || command[i] == '>' || command[i] == '('
| | command[i] == ')')
                break;
            result += command[i];
        }
        return result;
    }
    string del_part_command(string command) {
        string result = "";
        bool f = false;
        for (int i = 0; i < (int)command.length(); ++i) {
            if (command[i] == '!' || command[i] == '*' || command[i] == '+' || command[i] == '>' || command[i] == '('
| | command[i] == ')')
                f = true;
            if (f)
                result += command[i];
        }
        return result;
    }
    string del_first_symbol(string str) {
        string result = "";
        for (int i = 1; i < (int)str.length(); ++i)
            result += str[i];
        return result;
    }
    string del_only_first_word(string str) {
        string result = "";
        bool f = false;
        for (int i = 0; i < (int)str.length(); ++i) {
            if (f && str[i] != ' ')
                result += str[i];
            if (str[i] == ' ')
                f = true;
        }
        return result;
    }
    void update_everything() {
        this->everything.clear();
        for (map<string, mySet*>::iterator it = this->sets.begin(); it != this->sets.end(); ++it) {

```

```

        this->everything.add(this->sets[it->first]->get_objects());
    }
}

vector<string> string_to_objects(string str) {
    bool open = false;
    vector<string> res;
    string now = "";
    for (int i = 0; i < (int)str.length(); ++i) {
        if (str[i] == '{')
            open = true;
        if (!open || str[i] == ' ' || str[i] == '{')
            continue;
        if (str[i] == ',') {
            res.push_back(now);
            now = "";
            continue;
        }
        if (str[i] == '}') {
            res.push_back(now);
            now = "";
            break;
        }
        now += str[i];
    }
    return res;
}

string get_first_word(string str) {
    string res = "";
    for (int i = 0; i < (int)str.length(); ++i) {
        if (str[i] != ' ' && str[i] != '+' && str[i] != '*' && str[i] != '>' && str[i] != '!')
            res += str[i];
        else
            break;
    }
    return res;
}

string del_first_word(string str) {
    string res = "";
    bool f = false;
    for (int i = 0; i < (int)str.length(); ++i) {
        if (f && str[i] != ' ')
            res += str[i];
        if (str[i] == ' ' || str[i] == '+' || str[i] == '*' || str[i] == '>' || str[i] == '!') {
            if (f)
                break;
            else
                f = true;
        }
    }
    return res;
}

public:
    map<string, mySet*> sets;
    mySet everything;
};

signed main(int nNumArgs, char* psArgs[]) {
    string command = "";
    mySets sol;
    while (getline(cin, command)) {

```

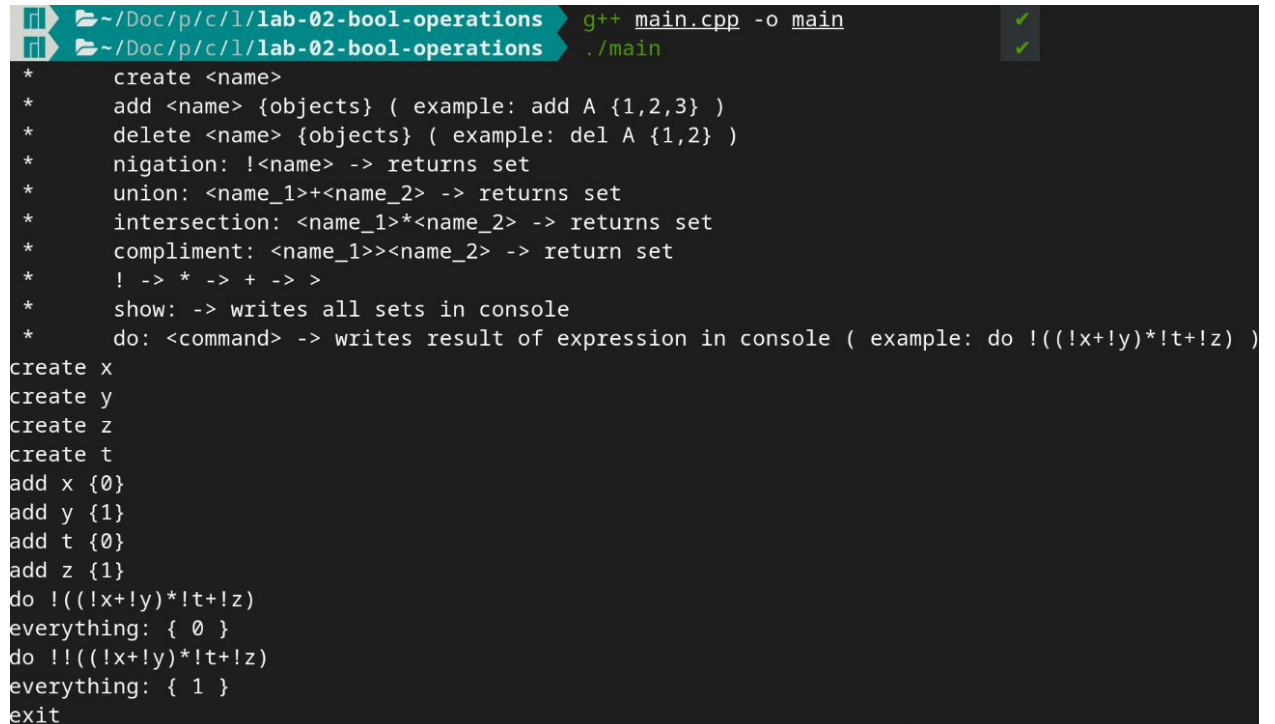


```

    if (command == "exit" || command == ".exit") {
        break;
    }
    sol.get_command(command);
}
return 0;
}

```

Результати:



```

~/Doc/p/c/1/lab-02-bool-operations g++ main.cpp -o main ✓
~/Doc/p/c/1/lab-02-bool-operations ./main ✓

*   create <name>
*   add <name> {objects} ( example: add A {1,2,3} )
*   delete <name> {objects} ( example: del A {1,2} )
*   nigation: !<name> -> returns set
*   union: <name_1>+<name_2> -> returns set
*   intersection: <name_1>*<name_2> -> returns set
*   compliment: <name_1>><name_2> -> return set
*   ! -> * -> + -> >
*   show: -> writes all sets in console
*   do: <command> -> writes result of expression in console ( example: do !((!x+!y)*!t+!z) )

create x
create y
create z
create t
add x {0}
add y {1}
add t {0}
add z {1}
do !((!x+!y)*!t+!z)
everything: { 0 }
do !((!x+!y)*!t+!z)
everything: { 1 }
exit

```

Висновок: навчився створювати методи для реалізації калькулятора булевих функцій. Реалізував алгоритми за допомогою мови програмування C++. Використовував метод реалізації через класи.

GitHub ([source code](#))