Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Лабораторна робота№1

Дисципліна: Комп’ютерна дискретна математика

Виконав

Студент групи ПЗПІ-21-1

Попович Ярослав Васильович

Перевірив

Асистент кафедри

Терещенко Гліб Юрійович

2021

Тема: Операції з множинами

Мета: Навчитися створювати калькулятор з операціями над сетами.

Індивідуальне завдання:

1. Реалізувати динамічне змінювання сету.
2. Реалізувати можливість вводу команд з консолі.
3. Вивести результат.
4. Підтвердити результати скриншотами.

Код програми:

```
/*
[illegible]
```

```
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" <<
" * nigation: !<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 (A\\B\\C)|(B\\A\\C)|(C\\A\\B)|(A&B&C) )\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* nigation(mySet* to_nigate) { // DONE
set<string> tmp_evr;

vector<string> tmp_cycle = this->everything.get_objects();
for (vector<string>::iterator it = tmp_cycle.begin(); it != tmp_cycle.end(); ++it)
tmp_evr.insert(*it);

tmp_cycle.clear();
tmp_cycle = to_nigate->get_objects();
for (vector<string>::iterator it = tmp_cycle.begin(); it != tmp_cycle.end(); ++it)
tmp_evr.erase(*it);

mySet* result = new mySet();
result->add(tmp_evr);
```

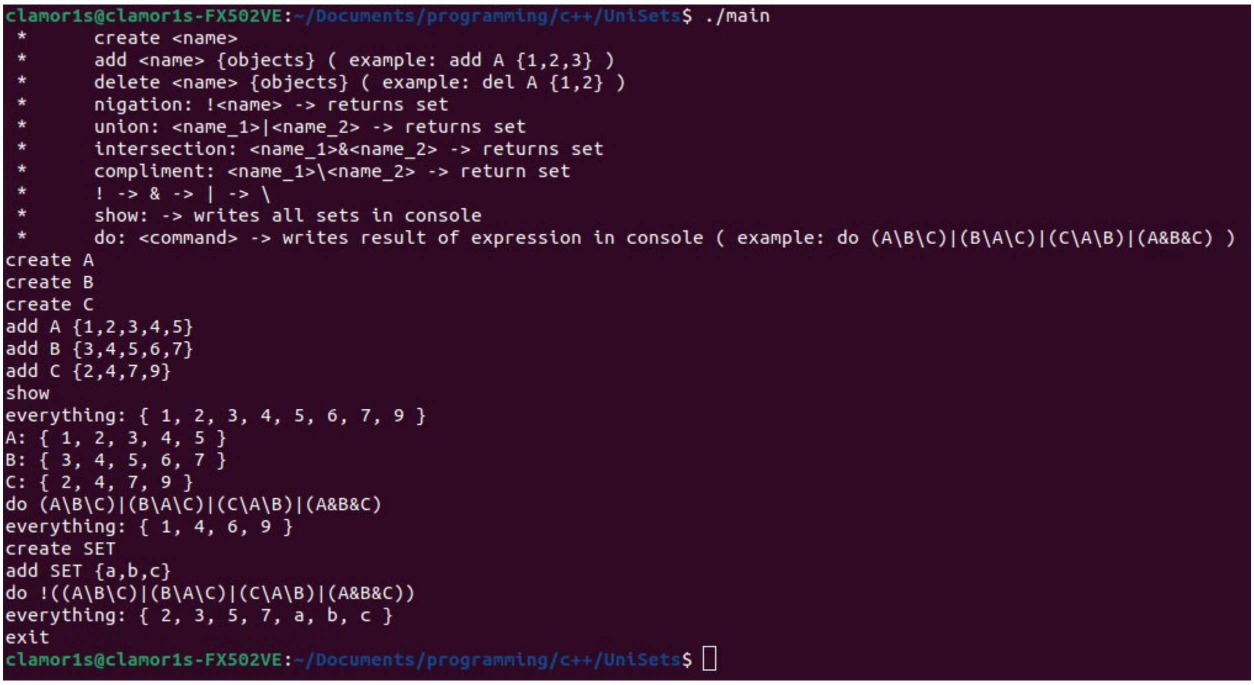
```
return result;
}

mySet* union_set(mySet* to_union1, mySet* to_union2) { // DONE
mySet* result = new mySet();
result->add(to_union1->get_objects());
result->add(to_union2->get_objects());

if (DEBUG)
cout << "union\n";

return result;
}

mySet* intersection(mySet* to_intersect1, mySet* to_intersect2) { // DONE
mySet* result = new mySet();

vector<string> res_set;
vector<string> tmp_1 = to_intersect1->get_objects();
vector<string> tmp_2 = to_intersect2->get_objects();
for (vector<string>::iterator it = tmp_2.begin(); it != tmp_2.end(); ++it) {
if (find(tmp_1.begin(), tmp_1.end(), *it) != tmp_1.end()) {
res_set.push_back(*it);
}
}
result->add(res_set);

return result;
}

mySet* compliment(mySet* to_compliment, mySet* from_compliment) { // DONE
mySet* result = new mySet();

result->add(to_compliment->get_objects());
result->del(from_compliment->get_objects());

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;

// sol.get_command("create A");
// sol.get_command("create B");
// sol.get_command("create C");
// sol.get_command("add A {1,2,3,4,5}");
// sol.get_command("add B {3,4,5,6,7}");
// sol.get_command("add C {2,4,7,9}");
// sol.get_command("show");

while (getline(cin, command)) {
if (command == "exit" || command == ".exit") {
break;
}

sol.get_command(command);
}


return 0;
}


// test: (A\B\C)|(B\A\C)|(C\A\B)|(A&B&C)
```

Результати:



```
clamor1s@clamor1s-FX502VE:~/Documents/programming/c++/UniSets$ ./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 (A\B\C)|(B\A\C)|(C\A\B)|(A&B&C) )
create A
create B
create C
add A {1,2,3,4,5}
add B {3,4,5,6,7}
add C {2,4,7,9}
show
everything: { 1, 2, 3, 4, 5, 6, 7, 9 }
A: { 1, 2, 3, 4, 5 }
B: { 3, 4, 5, 6, 7 }
C: { 2, 4, 7, 9 }
do (A\B\C)|(B\A\C)|(C\A\B)|(A&B&C)
everything: { 1, 4, 6, 9 }
create SET
add SET {a,b,c}
do !((A\B\C)|(B\A\C)|(C\A\B)|(A&B&C))
everything: { 2, 3, 5, 7, a, b, c }
exit
clamor1s@clamor1s-FX502VE:~/Documents/programming/c++/UniSets$ 
```

Висновок: навчився створювати методи для реалізації калькулятору сетів. Реалізував алгоритми за допомогою мови програмування С++. Використовував метод реалізації через класи.