**МИНОБРНАУКИ РОССИИ**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №2

Дисциплина: Математическая логика и теория алгоритмов

по теме «Логика предикатов»

Выполнил: ст. группы ВТ-22  
Макаров Даниил Сергеевич

Проверил: Куценко Д.А.

Белгород 2018

Задание к работе

Вариант 1. Разработать программу, способную считывать несколько формул-посылок логики высказываний и выводить на экран все формулы-следствия из этих посылок.

Заголовочный файл

#ifndef TASK\_H\_

#define TASK\_H\_

#include <stdlib.h>

#include <stdio.h>

#include "rpn.h"

#include "utilities.h"

#include <math.h>

char\* read\_permise(unsigned count\_of\_permise);

void print\_consequences(char\* str);

char\* table\_to\_pknf\_str(char \*\*table,unsigned n,unsigned m);

char\* table\_to\_pdnf\_str(char \*\*table,unsigned n,unsigned m);

char\*\* table\_to\_pknf\_arr(char \*\*table,unsigned n,unsigned m,unsigned \*count);

#endif

Исходный код

#include "task.h"

char\* read\_permise(unsigned count\_of\_permise){

char\* result\_str=(char\*)malloc(count\_of\_permise\*sizeof(char)\*50);

char temp[50];

int j,k=0;

for(int i=0;i<count\_of\_permise;i++){

scanf("%s",temp);

fflush(stdin);

j=0;

result\_str[k]='(';

k++;

while(temp[j]!='\0'){

result\_str[k]=temp[j];

k++;

j++;

}

result\_str[k]=')';

k++;

result\_str[k]='\*';

k++;

}

result\_str[k-1]='\n';

return result\_str;

}

char\*\* table\_to\_pknf\_arr(char \*\*table,unsigned n,unsigned m,unsigned \*count){

char \*\*result\_str\_arr=(char\*\*)malloc(n\*sizeof(char\*));

for(int i=0;i<n+1;i++){

result\_str\_arr[i]=(char\*)malloc(2\*m\*sizeof(char));

}

int j,p=0;

for(int i=1;i<n;i++){

j=0;

if(table[i-1][m-1]=='0'){

result\_str\_arr[p][j]='(';

j++;

for(int k=0;k<m-1;k++){

if(table[i][k]=='1'){

result\_str\_arr[p][j]='!';

j++;

}

result\_str\_arr[p][j]=table[0][k];

j++;

result\_str\_arr[p][j]='+';

j++;

}

j--;

result\_str\_arr[p][j]=')';

j++;

result\_str\_arr[p][j]='\0';

p++;

}

}

\*count=p;

return result\_str\_arr;

}

void print\_consequences(char\* str){

char varlist[50];

char \*rpn = NULL;

rpn = to\_rpn(str,varlist);

int n, m;

char \*\*t = calc\_table(rpn, varlist, &n, &m);

if (t == NULL){

printf("Invalid expression\n");

exit(1);

}

unsigned counter;

char\*\* pknf\_arr=table\_to\_pknf\_arr(t,n,m,&counter);

unsigned count\_of\_consequences = pow(2, counter);

for (int i = 0; i < count\_of\_consequences;i++){

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

if ( i & (1 << j) ) //--если j-й бит установлен

printf("%s+",pknf\_arr[j]);} //--то выводим j-й элемент множества

printf("\b \n");

}

}

Заголовочный файл

#ifndef RPN\_H

#define RPN\_H

#include "rpn.h"

#include "utilities.h"

char \*to\_rpn(char \*str, char \*varlist);

int calculator(char \*eq);

void free\_table(char \*\*t, int n, int m);

char \*\*calc\_table(char \*rpn, char \*vars, int \*n, int \*m);

#endif /\* RPN\_H \*/

Исходный код

#include <stdlib.h>

#include "rpn.h"

#include "stack.h"

#include "utilities.h"

char not(char a){

a = !(a-'0');

return a+'0';

}

char and(char a, char b){

char tmp = (a-'0') && (b-'0');

return tmp+'0';

}

char or(char a, char b){

char tmp = (a-'0') || (b-'0');

return tmp+'0';

}

char xor(char a, char b){

char t\_a = a-'0';

char t\_b = b-'0';

char tmp = (!t\_a && t\_b) || (t\_a && !t\_b);

return tmp+'0';

}

int priority(char c){

if ('a' <= c && c<='z')

return -2;

if (c == '0' || c == '1')

return -1;

switch (c){

case '(':

return 1;

case '+':

return 2;

case '\*':

return 3;

case '!':

return 4;

case ')':

return 5;

default:

return 0;

}

}

char \*to\_rpn(char \*str, char \*varlist){

stack s;

initstack(&s);

int flag;

char tmp;

int tmpprior;

int varcount = 0;

varlist[0] = '\0';

int size = 0;

int len = get\_len(str);

char \*result = (char \*)malloc(sizeof(char)\*(len+1));

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

switch (priority(str[i])){

case -2:

if (in\_str(str[i], varlist)==-1){

varlist[varcount++] = str[i];

varlist[varcount] = '\0';

}

case -1:

result[size++] = str[i];

break;

case 1:

putstack(&s, str[i]);

break;

case 2:

case 3:

tmpprior = priority(str[i]);

flag = emptystack(&s);

while (!flag){

getstack(&s, &tmp);

if (priority(tmp) >= tmpprior){

result[size++] = tmp;

}else{

putstack(&s, tmp);

flag = 1;

}

flag = flag || emptystack(&s);

}

putstack(&s, str[i]);

break;

case 4:

putstack(&s, str[i]);

break;

case 5:

getstack(&s, &tmp);

while (priority(tmp) != 1){

result[size++] = tmp;

getstack(&s, &tmp);

}

break;

}

}

while (!emptystack(&s)){

getstack(&s, &tmp);

result[size++] = tmp;

}

result[size++] = '\0';

result = (char \*)realloc(result, size\*sizeof(char));

freestack(&s);

return result;

}

int calculator(char \*eq){

int i = 0;

char a, b;

stack s;

initstack(&s);

while (eq[i]){

switch (eq[i]){

case '!':

getstack(&s, &a);

putstack(&s, not(a));

break;

case '+':

getstack(&s, &b);

getstack(&s, &a);

putstack(&s, or(a,b));

break;

case '\*':

getstack(&s, &b);

getstack(&s, &a);

putstack(&s, and(a,b));

break;

case '0':

case '1':

putstack(&s, eq[i]);

break;

default:

freestack(&s);

return -1;

}

i++;

}

if (emptystack(&s)){

return -1;

}else{

getstack(&s, &a);

freestack(&s);

return a - '0';

}

}

void free\_table(char \*\*t, int n, int m){

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

free(t[i]);

}

free(t);

}

char \*\*calc\_table(char \*rpn, char \*vars, int \*n, int \*m){

sort(vars);

int cols = get\_len(vars);

if (cols == 0 || cols >5) return NULL;

int rows = (1<<cols)+1;

cols++;

char \*\*table = (char \*\*)malloc(rows\*sizeof(char\*));

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

table[i] = (char \*)malloc(cols\*sizeof(char));

int len = get\_len(rpn);

for (int i = 0; i<cols-1; i++){

table[0][i] = vars[i];

table[1][i] = '0';

}

table[0][cols-1] = '=';

char carry;

for (int i = 2; i<rows; i++){

carry = '1';

for (int j = cols-2; j>=0; j--){

table[i][j] = xor(table[i-1][j], carry);

carry = and(table[i-1][j], carry);

}

}

char var;

int tmp\_res;

for (int i = 1; i<rows; i++){

char \*tmp = copy(rpn, len);

for (int j = 0; j<cols-1; j++){

var = vars[j];

for (int k = 0; k<len; k++){

if (tmp[k] == var){

tmp[k] = table[i][j];

}

}

}

tmp\_res = calculator(tmp);

if (tmp\_res == -1){

free\_table(table, rows, cols);

return NULL;

}

table[i][cols-1] = tmp\_res + '0';

free(tmp);

}

\*n = rows;

\*m = cols;

return table;

}