**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

тема: «Логика предикатов»

|  |  |
| --- | --- |
|  | Выполнил: ст. группы  Проверил: Бондаренко Т.В. |

Белгород 20

**Практическая часть:**

**Вариант 1**

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

#include **<stdio.h>**#include **<stdlib.h>**#include **<math.h>**#define **N** 100  
**long unsigned** d[**N**];  
**long unsigned** c[**N**];  
*//выделение памяти матрице***int** \*\*getMemoryMatr(**int** str, **int** tab) {  
 **int** \*\*matr = (**int** \*\*)malloc(str \* **sizeof**(**int** \*));  
 **for** (**int** i = 0; i < str; i++)  
 matr[i] = (**int** \*)malloc(tab \* **sizeof**(**int**));  
 **return** matr;  
}  
*//ввод КНФ пользователем в заданном программой виде***void** input\_dnf(**int**\*\* a, **int**\* b, **int** m, **int** n) {  
 **int** i, j;  
  
 printf(**"Ваши перименные: "**);  
 **for** (i = 0; i<n; i++) {  
 b[i] = **'A'** + i;  
 printf(**"%c "**, b[i]);  
 }  
 printf(**"\n"**);  
 printf(**"Обозначения:\n"**);  
 printf(**"Наличие переменной : 1\n"**);  
 printf(**"Отрицание переменной: -1\n"**);  
 printf(**"Отсутствие переменной: 0 \n"**);  
  
 **for** (i = 0; i < m; i++) {  
 printf(**"Cкобка %d:\n"**, i + 1);  
 **for** (j = 0; j < n; j++)  
 scanf(**"%i"**, &a[i][j]);  
 }  
}  
  
*//вывод ДНФ на экран***void** output\_forml(**int** \*\*a, **int**\* b, **int** m, **int** n) {  
 **int** i = 0, j = 0;  
  
 **for** (i = 0; i < m; i++) {  
 printf(**"("**);  
 j = 0;  
 **while** ((a[i][j] == 0) && (j < n))  
 j++;  
 **if** (j < n) {  
 **if** (a[i][j] == -1)  
 printf(**"!%c"**, b[j]);  
 **else** printf(**"%c"**, b[j]);  
 }  
 j++;  
 **for** (; j < n; j++) {  
 **if** (a[i][j] == -1)  
 printf(**" + !%c"**, b[j]);  
 **if** (a[i][j] == 1)  
 printf(**" + %c"**, b[j]);  
  
 }  
 printf(**")"**);  
 **if** ((i + 1) < m)  
 printf(**"\*"**);  
 }  
 printf(**"\n"**);  
}  
  
*//вывод полученной таблицы истинности, а для данной ДНФ***int**\*\* output\_tab(**int** \*\*a, **int** \*b, **int** m, **int** n) {  
 **int** i, j, mask = 1, f, x, z, k;  
 *//вывод обозначение столбца в таблице истинности* **for** (i = 0; i < n; i++)  
 printf(**"%c "**, b[i]);  
 printf(**"f\n"**);  
  
 **int** all = pow(2, n);  
 *//таблица истинности* **int** \*\*tabl = getMemoryMatr(all, n + 1);  
  
  
 **for** (i = 0; i < all; i++) {  
 *//получение двоичного вектора основываясь на предыдущем векторе* **for** (j = 0; j < n; j++) {  
 tabl[i][j] = (mask & (i >> (n - 1 - j)));  
 printf(**"%i "**, tabl[i][j]);  
 }  
 f = 1;  
 z = 0;  
 *//высчитывание формулы по полученному двоичному вектору* **while** ((z < m) && (f)) {  
 x = 0;  
 k = 0;  
 **while** (k < n) {  
 **if** (a[z][k] == 1)  
 x |= tabl[i][k];  
 **if** (a[z][k] == -1)  
 x |= !tabl[i][k];  
 k++;  
 }  
 f = f && x;  
 z++;  
 }  
 tabl[i][n] = f;  
 printf(**"%i\n"**, tabl[i][n]);  
 }  
 **return** tabl;  
}  
  
  
**int** output\_sknf(**int** \*\*tabl, **int** n, **int**\* ABC, **int** \*b) {  
 **int** i, j, k = 1;  
 **int** n1 = pow(2, n);  
 **for** (i = 0; i < n1; i++)  
 **if** (tabl[i][n] == 0) {  
 b[k - 1] = i;  
 printf(**"%i)"**, k);  
 printf(**"("**);  
 **for** (j = 0; j<n - 1; j++) {  
 **if** (tabl[i][j] == 1)  
 printf(**"!%c+"**, ABC[j]);  
 **else** printf(**"%c+"**, ABC[j]);  
 }  
 **if** (tabl[i][n - 1] == 1)  
 printf(**"!%c"**, ABC[n - 1]);  
 **else** printf(**"%c"**, ABC[n - 1]);  
  
 printf(**")\n"**);  
 k++;  
 }  
 **return** k - 1;  
}  
  
  
**void** input(**long unsigned** c[], **int** k) {  
 **for** (**int** i = 0; i < k; i++)  
 c[i] = i + 1;  
}  
  
  
**void** output(**long unsigned** d[], **int** k, **int**\* b, **int**\*\* tabl, **int** n, **int** \*ABC) {  
 size\_t i, j;  
 **for** (i = 0; i < k; i++)  
 **if** (d[i]) {  
 printf(**"("**);  
 **for** (j = 0; j < n - 1; j++) {  
 **if** (tabl[b[i]][j] == 1)  
 printf(**"!%c+"**, ABC[j]);  
 **else** printf(**"%c+"**, ABC[j]);  
 }  
 **if** (tabl[b[i]][n - 1] == 1)  
 printf(**"!%c"**, ABC[n - 1]);  
 **else** printf(**"%c"**, ABC[n - 1]);  
 printf(**")"**);  
 }  
}  
  
  
**void** recurs(size\_t i, **int** k, **int**\* b, **int**\*\* tabl, **int** n, **int**\* ABC) {  
 **short unsigned** x;  
 **for** (x = 0; x <= 1; x++) {  
 d[i] = x;  
 **if** (i == k - 1) {  
 output(d, k, b, tabl, n, ABC);  
 printf(**"\n"**);  
 }  
 **else** recurs(i + 1, k, b, tabl, n, ABC);  
 }  
}  
*//выделение памяти массиву***int** \*getMemoryArr(**int** size) {  
 **return** (**int** \*)malloc(size \* **sizeof**(**int**));  
}  
*//очищение памяти массива***void** freeMemoryArr(**int** \*arr) {  
 free(arr);  
}  
  
  
*//очищение памяти матрицы***void** freeMemoryMatr(**int** \*\*matr, **int** str) {  
 **for** (**int** i = 0; i < str; i++)  
 free(matr[i]);  
 free(matr);  
}  
  
  
  
**int** main() {  
  
 **int** i, k;  
 **int** \*\*tabl;  
 printf(**"Количество скобок в КНФ: "**);  
 **int** m;  
 scanf(**"%i"**, &m);  
 printf(**"Количество переменных: "**);  
 **int** n;  
 scanf(**"%i"**, &n);  
 **int** \*b = getMemoryArr(pow(2, n));  
 *//матрица формулы ДНФ  
 //строка матрицы - одна скобка ДНФ  
 //каждый столбец логический связан с одной переменной  
 //значение в ячейке характеризует наличие(1), отрицание(-1) или отсутствие(0) переменной* **int** \*\*a = getMemoryMatr(m, n);  
  
 *//массив переменных в виде символов* **int** \*ABC = getMemoryArr(n);  
  
 *//ввод ДНФ* input\_dnf(a, ABC, m, n);  
 *//вывод ДНФ в привычной форме* output\_forml(a, ABC, m, n);  
 *//построение таблицы истинности и вывод её на экран* tabl = output\_tab(a, ABC, m, n);  
 k = output\_sknf(tabl, n, ABC, b);  
 input(c, k);  
 recurs(0, k, b, tabl, n, ABC);  
  
 freeMemoryArr(ABC);  
 freeMemoryMatr(a, m);  
 freeMemoryMatr(tabl, pow(2, m));  
 **return** 0;  
}