# Задания

**Общие требования** для всех заданий:

а) Самостоятельно написать программу, отладить её, запустить и получить ответ на задачу.

б) При сдаче иметь в наличии: ответ, исходный текст программы (файл \*.PAS, \*.CS, \*.CPP, \*.PY и т.п., работоспособный проект (важно для Delphi, Lasarus, Visual Studio, SharpDevelop, MonoDevelop).

**Общие требования к коду**:

* значения логических выражений вычисляются по формулам (а не прописываются в виде констант)
* перебор значений логических переменных выполняется в цикле (или во вложенных циклах)

№1

Написать консольную программу, которая выведет на экран таблицы истинности для следующих логических формул:

а) A xor B (строгая дизъюнкция)

б) A🠖B (импликация)

в) (A🠖B) <=> (⎤A v B) (смысл формулы: "правда ли, что (A🠖B) эквивалента (⎤A v B) при любых A и B"

г) (A🠖B) <=> (A ≤ B) (смысл формулы: "правда ли, что (A🠖B) эквивалента (A ≤ B) при любых A и B"

**Требования к оформлению** **вывода**:

* Строки таблицы содержат: конкретный набор значений для A и B, а также значение формулы при этих значениях.
* В первом (самом левом) столбце – значения переменной A, во 2-м столбце – значения переменной B, далее идут 1 или более столбцов со значениями логических формул.
* Все столбцы должны быть выровнены (используйте форматированный вывод значений)
* Все столбцы должны быть подписаны.

Возможные варианты исполнения:

Требуется написать ровно 1 программу. Она должна вывести на экран либо 4 таблицы истинности для указанных 4-х формул, либо – одну таблицу, в которой будут столбцы для значений A и B, плюс 4 столбца для значений 4-х формул.

№2

Сколько различных решений имеет уравнение

((J🠖K) 🠖 (M & N & L)) & ((J & ⎤K) 🠖 ⎤ (M & N & L)) & (M🠖J) = 1,

где J, K, L, M, N – логические переменные?

В ответе не нужно перечислять все различные наборы значений J, K, L, M и N, при которых выполнено данное равенство. В качестве ответа нужно указать количество таких наборов.

Ответ: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

**Требования к оформлению вывода**:

* программа должна вывести на экран ровно 1 число.

№3

Сколько существует различных наборов значений логических переменных x1, x2, ... x9, y1, y2, ... y9, которые удовлетворяют всем перечисленным ниже условиям:

(⎤ (x1 ≡ y1 ) 🠖 ⎤ (x2 ≡ y2 )) & (x1 🠖 x2 ) & (y1 🠖 y2 ) = 1;

(⎤ (x2 ≡ y2 ) 🠖 ⎤ (x3 ≡ y3 )) & (x2 🠖 x3 ) & (y2 🠖 y3 ) = 1;

.. .. ..

(⎤ (x8 ≡ y8 )) 🠖 ⎤ (x9 ≡ y9 )) & (x8 🠖 x9 ) & (y8 🠖 y9 ) = 1.

В ответе не нужно перечислять все различные наборы значений переменных x1, x2, ... x9, y1, y2, ... y9, при которых выполнена данная система равенств. В качестве ответа Вам нужно указать количество таких наборов.

Ответ: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

**Требования к оформлению вывода**:

* программа должна вывести на экран ровно 1 число.

# Логические (булевы) переменные

В современных языках программирования для логических значений и переменных имеется отдельный тип данных:

|  |  |
| --- | --- |
| **Pascal** | **C#, C++** |
| Boolean | bool |

Логические (булевы) переменные могут принимать только 2 разных значения: **false** или **true** (соответственно, "ложь" и "истина"). Примеры объявления переменных:

|  |  |
| --- | --- |
| **Pascal** | **C#, C++** |
| Var  A,B,C: boolean;  BEGIN  ...  A := FALSE;  B := TRUE;  C := A;  ... | {  ...  bool A = false;  bool B = true;  bool C = A;  ...  } |

# Операции над логическими переменными

|  |  |  |  |
| --- | --- | --- | --- |
| **Логические операции** | **Pascal** | **C#** | **C++** |
| отрицание (НЕ, NOT) | not  A := not B; | !  A = !B; | !  A = !B; |
| конъюнкция (И, AND) | and  A:= B and C; | && или &  A = B & C; | &&  A = B && C; |
| дизъюнкция (ИЛИ, OR) | or  A := B or C; | || или |  A = B | C; | ||  A = B || C; |
| строгая дизъюнкция (XOR) (НЕ РАВНО) | xor или <>  A := B xor C;  A := B <> C; | ^ или !=  A = B ^ C;  A = B != C; | ^ или !=  A = B ^ C;  A = B != C; |
| эквивалентность (РАВНО) | =  A := B = C; | ==  A = B == C; | ==  A = B == C; |

Ещё одна часто используемая логическая операция – это ИМПЛИКАЦИЯ, или операция следования (ЕСЛИ...ТО..., A🠖B). Её смысл таков: «правда ли, что из высказывания A следует высказывание В?» В формальной логике Аристотеля, помимо всего прочего, были описаны ***правила вывода*** **умозаключений** из **высказываний-посылок**. Высказывания-посылки сравнивают или устанавливают связь между свойствами объектов/понятий. Правила вывода описывают, как правильно из частей высказываний-посылок построить высказвание-умозаключение, чтобы оно было истинным, при условии, что истинны посылки. Пример правила вывода:

|  |  |  |
| --- | --- | --- |
| Посылка 1: | Все люди | смертны |
| Посылка 2: | Сократ – | человек |
| Умозаключение: | Сократ | смертен |

В формальной логике Аристотеля операция следования формулируется следующим образом: из истинного высказывания следуют только истинные высказывания, из ложного высказывания может следовать любое высказывание. Напрямую импликация A🠖B может быть закодирована так:

if (not A) *// эквивалент if (A=FALSE)*

then Result := TRUE

else Result := B; *// эквивалент if (A=TRUE) then Result := (B=TRUE)*

Для импликации (A🠖B) специального обозначения нет. Чтобы закодировать вычисление импликации, обычно приходится использовать эквивалентность логических функций:

A🠖B = Ā v B = ⎤A v B

В Паскале есть более красивый метод: дело в том, что в Паскале логический тип является перечислимым, и логические значения можно **сравнивать**: считается, что FALSE < TRUE. Поэтому для логических выражений верны, например, такие выражения: A<B, A>B, A<=B, A>=B.

В С++ этот приём тоже применим – для логических значений допустимы сравнения «больше», «меньше» и false<true. А в таких языках, как C# и Python это невозможно, потому что для логических переменных операции сравнения «больше» и «меньше» не определены. А в скриптовом языке Линукса – BASH, – значение 0 соответствует «истине», а ненулевое значение – соответствует логическому значению «ложь»; а чтобы было «по-человечески» – надо записывать выражение в специальной форме (в тройных квадратных скобках!).

Импликация A🠖B эквивалентна сравнению A <= B. Этот факт доказывается путём сравнения таблиц истинности логических выражений (функций, операций): для выражений A🠖B и A<=B таблицы истинности совпадают.

# Массивы с логическим индексом

В Паскале логический тип является перечислимым, поэтому можно объявлять массивы с индексом типа boolean:

Var

Ar: array[boolean] of integer;

BEGIN

A[FALSE] := 10;

A[TRUE] := -10;

...

# Циклы с логическими счётчиками

В Паскале логический тип является перечислимым, поэтому переменные типа boolean можно использовать в качестве счётчика в циклах FOR:

Var

A,B,C: boolean;

BEGIN *// этот фрагмент печатает таблицу истинности для функции A->B*

for A:=FALSE to TRUE do

for B:=FALSE to TRUE do

begin

C := A<=B;

Writeln(A:6, B:6, C:6);

end;

...

Увы, в C#/C++ и Питоне такого удобства нет. В программах на этих языках можно использовать несколько корявую конструкцию со счётчиком 0..1, из которого потом делают булевское значение:

…

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

{

bool A = (i=1); *// получим A=false при i==0 и A=true при i==1*

…

}

Этот приём можно расширить на несколько переменных. Например: надо вычислять логическую формулу от трёх переменных. Каждая переменная может иметь 2 значения – независимо от остальных. Значит, набор из трёх переменных может иметь 23 = 8 значений. Пусть целочисленная переменная k изменяется от 0 до 7 – тогда её младшие биты будут принимать все возможные комбинации от 0002 до 1112. Договоримся, что младший бит (бит #0) кодирует значение логической переменной C, бит #1 кодирует значение логической переменной B, бит #2 – значение логической переменной A. И получим код:

for(int k=0; k<8; i++)

{

bool A = (k & 4)>0; *// получим A=true, если бит#2 равен 1. Помним: 4 = 1<<2;*

bool B = (k & 2)>0; *// получим B=true, если бит#1 равен 1. Помним: 2 = 1<<1;*

bool C = (k & 1)>0; *// получим C=true, если бит#0 равен 1. Помним: 1 = 1<<0;*

…

}

Аналогичный код можно написать и на Паскале:

for k:=0 to 7 do

begin

A = (k and 4)>0; *// получим A=true, если бит#2 равен 1. Помним: 4 = 1 shl 2;*

B = (k and 2)>0; *// получим B=true, если бит#1 равен 1. Помним: 2 = 1 shl 1;*

C = (k and 1)>0; *// получим C=true, если бит#0 равен 1. Помним: 1 = 1 shl 0;*

…

end;

# Таблица истинности

Таблица истинности логической функции F(A,B,C…) – это таблица, в которой перечисляются все возможные комбинации аргументов функции, и для каждого набора значений аргументов указывается значение функции. Для сравнения: в арифметике чисел невозможно построить таблицу сложения любых чисел – поскольку чисел бесконечно много. А в алгебре логики переменная может иметь только 2 значения; N переменных будут иметь 2N значений – тоже конечное число. Поэтому можно перечислить все варианты значений аргументов, и для каждого – записать значение функции.

С помощью таблиц истинности можно доказать эквивалентность логических формул: таблицы истинности двух формул совпадают тогда и только тогда, когда формулы эквивалентны.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **A🠖B** | **A≤B** | **(A🠖B) = (A≤B)** |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Таблицы истинности заполняют по следующему правилу: пусть логическая формула зависит от N переменных. N столбцов слева отводятся для записи значений переменных (1-я переменная в 1-м столбце, и т.д.). Столбцы справа отводятся для записи значений функции (или формулы). Таблица должна содержать 2N строк, плюс строку заголовков.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***№*** | **A** | **B** | **A🠖B** | **A≤B** | **(A🠖B) = (A≤B)** |
| *0=002* | 0 | 0 | 1 | 1 | **1** |
| *0=****012*** | **0** | **1** | 1 | 1 | **1** |
| *0=102* | 1 | 0 | 0 | 0 | **1** |
| *0=112* | 1 | 1 | 1 | 1 | **1** |

Пронумеруйте строки таблицы от 0 до 2N‑1 – в двоичной системе счисления эти числа можно записать N-битными последовательностями от 00..02 до 11..12, причём будут перечислены все N‑значные комбинации нулей и единиц. Теперь в столбцах, отведённых для заиси значений переменных, надо вписать значения соответствующих битов номера строки (использовать двоичные записи номеров строк). После этого следует заполнить столбцы для значений функций/формул.

Пример правильного оформления доказательства эквивалентности формул (см. 2-ю таблицу):

а) заполняем таблицу истинности для для первой функции;

б) рядом с ней (справа) дописываем столбец соответствующих значений для первой функции (форма табл.ист.);

в) заполняем ещё одинн столбец значений: F = ( f1(A,B,C…) <=> f2(A,B,C…) ): если в данной строке значения функций совпадают – ставим 1, а иначе – 0.

г) Если во всех строках функция F имеет значение 1 (т.е. она является тождественной истиной), то формулы эквивалентны, иначе – не эквивалентны.

Заполнение последнего столбца кажется излишним – ведь можно просто сравнить два последних столбца. Это верно, НО! При быстром сравнении значений из двух столбцов человек может ошибиться, не заметить ошибки. А если просматривать только один столбец, где 1 означает совпадение, а 0 – различие значений, то вероятность ошибки практически исключена. Вот пусть компутер и поработает для нашего удобства и удовольствия.

# Система логических уравнений

В задании №3 система логических уравнений содержит очень много переменных (всего 18 штук: x1, x2, ... x9, y1, y2, ... y9). В разделе IV [Циклы с логическими счётчиками](#_Циклы_с_логическими) приводится пример, как можно одним циклом с помощью целочисленной переменной задавать сразу много логических значений. Однако кодировать проверку каждого из 18 бит числа, с помощью 18 констант – это всё равно долго и утомительно. Для решения подобных задач следует использовать массивы логических переменных.

В данном задании можно создать два массива логических переменных:

|  |  |
| --- | --- |
| **Pascal** | **C#** |
| Var  X, Y: array [1..9] of boolean; | bool[] X = new bool[9];  bool[] Y = new bool[9]; |

Также надо выбрать, какие биты будут соответствовать переменным X[i], и какие – Y[i]. Варианты могут быть такие:

* Младшие 9 бит (биты #0 … #8) – это X[i], следующие 9 бит (биты #9 … #17) – это Y[i].
* Чётные биты (биты #0, #2, #4 … #16) – это X[i], нечётные биты (биты #1, #3, #5 … #17) – это Y[i].
* Возможны и другие варианты, но они будут запутанными и более сложными для кодирования.

Теперь получение значений битов можно легко запрограммировать:

|  |  |
| --- | --- |
| **Pascal** | **C#, C++** |
| for k:=0 to (1 shl 18)-1 do  begin  for i:=0 to 9 do  begin  X[i]:= k and ( 1 shl i) >0;  Y[i]:= k and (512 shl i) >0;  end;  ... | for(int k = 0; k<(1<<18); k++)  {  for(int i=0; i<18; i += 2)  {  X[i] = k & (1<< i ) >0;  Y[i] = k & (1<<(i+1)) >0;  }  ... |

В примере на Паскале X[i] и Y[i] выбираются по первому варианту, в варианте на C# – по второму варианту.