Лабораторная работа №4
Исследование возможностей интегрированной среды разработки Visual C# для создания приложений линейной структуры с использованием логических операций.

Цель работы – исследовать возможности интегрированной среды разработки Visual Studio 2010 и получить практические навыки по созданию приложений, использующие логические операторы.

1. Теоретические сведения

В любой программе требуется производить вычисления. Для вычисления значений используются выражения, представляющие собой формулы для вычисления значений переменных. Выражения — это операции, которые выполняют программы. Выражения широко используются в программах на языке C# и представляют собой формулы для вычисления значений переменных. Они состоят из операндов (переменные, константы и др.), соединенных знаками операций (сложение, вычитание, умножение и др.).

Выражения могут иметь простую и сложную структуру. Сложность выражений зависит от количества переменных и операций, производимых над ними. Сложные выражения вычисляются в соответствии с фиксированным приоритетом, назначенным каждому оператору. Порядок выполнения операторов при вычислении значения выражения определяется их приоритетами и может регулироваться с помощью круглых скобок. Например,

(a + 0.12)/6
x &&y || !z
(t * Math.Sin(x)-1.05e4)/((2 * k + 2) * (2 * k + 3))

В языке C# предусмотрен обширный ряд операторов, предоставляющих программирующему возможность полного контроля над построением и вычислением выражений. Большинство операторов в C# относится к следующим категориям: арифметические, поразрядные, логические и операторы отношения. Кроме того, в C# предусмотрен ряд других операторов для особых случаев, включая индексирование массивов, доступ к членам класса и обработку лямбда-выражений.

Выражения, содержащие арифметические операции, записываются с помощью арифметических операторов **(+, -, /, *, %, ++, --**). Выражения, содержащие логические операции (**И, ИЛИ, НЕ**), записываются с помощью логических операторов **(&, |, !**).

1.1. Операторы отношений

В обозначениях оператор отношения и логический оператор термин отношения означает взаимосвязь, которая может существовать между двумя значениями, а термин логический — взаимосвязь между логическими значениями "истина" и "ложь". И поскольку операторы отношения дают истинные или ложные результаты, то они нередко применяются вместе с логическими операторами.

Результатом выполнения оператора отношения или логического оператора является логическое значение типа bool.

В целом, объекты можно сравнивать на равенство или неравенство, используя операторы отношения == и ! =. А операторы сравнения <, >, <= или >= могут применяться только к тем типам данных, которые поддерживают отношение порядка.

Следовательно, операторы отношения можно применять ко всем числовым типам данных. Но значения типа bool могут сравниваться только на равенство или неравенство,

поскольку истинные (true) и ложные (false) значения не упорядочиваются. Например, сравнение true > false в C# не имеет смысла.

Операторы отношений обрабатывают свои операнды таким образом, чтобы получить либо истинный, либо ложный результат. Выражение (A < B) истинно только в случае, если A меньше B. Переменные A и B должны принадлежать к таким типам данных, которые можно сравнивать. Обычно это целые или значения с плавающей запятой. Для сравнения строк операторы отношений использовать нельзя.

В таблице 1 перечислены все операторы отношений, а в таблице 2 – логические операторы.

Таблица 1. Перечень операций отношений

Оператор	**Описание**	**Пример**
<	меньше	a < b
<=	меньше или равно	a <= b
>	больше	a > b
>=	больше или равно	a >= b
==	равно	a = = b
!=	не равно	a != b

Таблица 2. Перечень логических операторов

Оператор	**Описание**	**Пример**
&	И	a & b
|	ИЛИ	a | b
^	исключающее ИЛИ	a ^ b
&&	укороченное И	a && b
||	укороченное ИЛИ	a || b
!	НЕ	!a

1.2. Логические операторы

Логические операторы в C# выполняют наиболее распространенные логические операции. В C# предусмотрен набор логических операторов, которого достаточно для построения практически любой логической операции.

Операнды логических операторов должны относиться к типу bool, а результат выполнения логической операции также относится к типу bool. Логические операторы &, |, ^ и ! поддерживают основные логические операции И, ИЛИ, исключающее ИЛИ и НЕ в соответствии с нижеприведенной таблицей истинности (табл.3).

Таблица 3. Таблица истинности

a	b	a & b	a | b	a ^ b	!a
false	false	false	false	false	true
true	false	false	true	true	false
false	true	false	true	true	true
true	true	true	true	false	false

Пример 1. Нижеприведенный пример программы демонстрируется применение нескольких операторов отношения и логических операторов (результат выполнения программы представлен на рис.1).

```
static void Main()
    {
      int i, j;
```

```
    bool b1, b2;
    i = 10;
    j = 11;
    Console.WriteLine("  i = " + i + "  j = " + j);
    if(i < j) Console.WriteLine("  i < j");
    if(i <= j) Console.WriteLine("  i <= j");
    if(i != j) Console.WriteLine("  i != j");
    if(i == j) Console.WriteLine("  Нельзя выполнить");
    if(i >= j) Console.WriteLine("  Нельзя выполнить");
    if(i > j) Console.WriteLine("  Нельзя выполнить");
    b1 = true;
    b2 = false;
    Console.WriteLine("  b1 = " + b1 + "  b2 = " + b2);
    if(b1 & b2) Console.WriteLine("  Нельзя выполнить");
    if(!(b1 & b2)) Console.WriteLine("  !(b1 & b2) - true");
    if(b1 | b2) Console.WriteLine("  b1 | b2 - true");
    if(b1 ^ b2) Console.WriteLine("  b1 ^ b2 - true");
  }
```

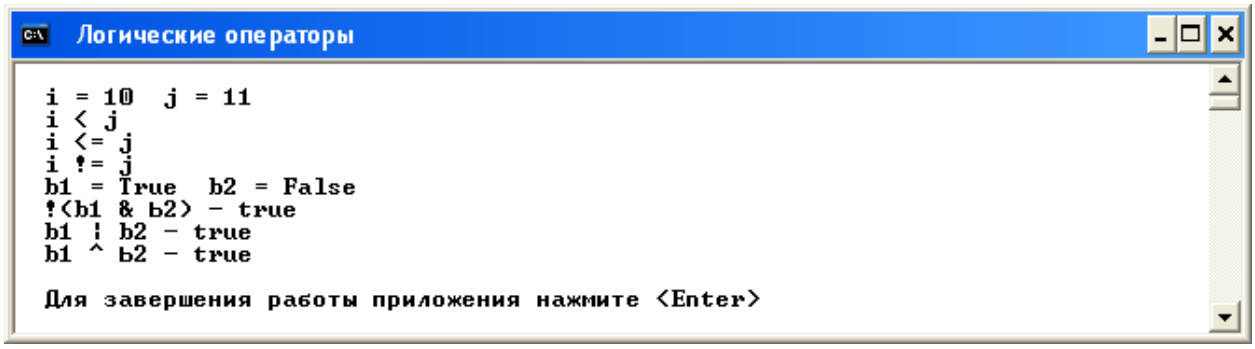

Рисунок 1 – Результат выполнения примера «Логические операторы»

Укороченные логические операторы

В C# предусмотрены также специальные укороченные варианты логических операторов И и ИЛИ, предназначенные для получения более эффективного кода, например, если первый операнд логической операции И имеет ложное значение (false), то ее результат будет иметь ложное значение независимо от значения второго операнда. Если же первый операнд логической операции ИЛИ имеет истинное значение (true), то ее результат будет иметь истинное значение независимо от значения второго операнда. Благодаря тому, что значение второго операнда в этих операциях вычислять не нужно, экономится время и повышается эффективность кода.

Укороченная логическая операция И выполняется с помощью оператора **&&**, а укороченная логическая операция ИЛИ — с помощью оператора **||**. Этим укороченным логическим операторам соответствуют обычные логические операторы & и |.

Единственное отличие укороченного логического оператора от обычного заключается в том, что второй его операнд вычисляется только по мере необходимости.

Пример 2. Нижеприведенный код консольного приложения демонстрирует вычисление логического выражения

$$z = a \cap b \cup c \cup \neg ((x \le y) \cup (x \ge -0.15)),$$

где $\cap$ – операция **И**, $\cup$ - операция **ИЛИ**, $\neg$ - операция **НЕ**.

```
static void Main()
    {
       double x, y;
       bool a, b, c, z;
       Console.Write("Введите значение a ");
       a = Convert.ToBoolean(Console.ReadLine());
       Console.Write("Введите значение b ");
       b = Convert.ToBoolean(Console.ReadLine());
       Console.Write("Введите значение c ");
       c = Convert.ToBoolean(Console.ReadLine());
       Console.Write("Введите значение x ");
       x = Convert.ToDouble(Console.ReadLine());
       Console.Write("Введите значение y ");
       y = Convert.ToDouble(Console.ReadLine());
       z = a && b || c || !((x <= y) || (x >= -0.15));
       Console.Write("Значение z равно ");
       Console.WriteLine(z);
       Console.Write("Для завершения работы нажмите клавишу <Enter>");
       Console.Read();
    }
```

1.3. Поразрядные операторы

В C# предусмотрен ряд поразрядных операторов, расширяющих круг задач, для решения которых можно применять C#. Поразрядные операторы воздействуют на отдельные двоичные разряды (биты) своих операндов. Они определены только для целочисленных операндов, поэтому их нельзя применять к данным типа bool, float или double.

Двоичные разряды, или биты, в таких числах, как 1011, полностью соответствуют переключателям внутри компьютерной памяти, работающим по принципу "включено — выключено": бит, равный 0, означает "выключено", а бит, равный 1, — "включено".

Чтобы увеличить скорость компьютера, процессор выполняет чтение, запись и передачу информации не по одному биту, а по группам, представляющим собой 8-битовые байты или 16-битовые слова. C#-программы могут обрабатывать такие значения непосредственно в двоичном коде.

Шесть поразрядных операторов выполняют булевы логические операции, названные по имени английского математика Джорджа Буля (1815-1864). В таблице 4 сведены все поразрядные операторы языка C#.

Таблица 4. Перечень поразрядных операторов

Оператор	Описание	Пример
&	Поразрядное И	C = A & B;
|	Поразрядное ИЛИ	C = A | B;
^	Поразрядное исключающее ИЛИ	C = A ^ B;
«	Сдвиг битов влево	C = A « B;
»	Сдвиг битов вправо	C = A » B;
~	Дополнение до 1, или поразрядное отрицание	C = ~ A;

Первые три поразрядных оператора из таблицы 4 объединяют два операнда в соответствии с правилами логического И (&), логического ИЛИ (|) и исключающего ИЛИ (^). Следующие два оператора (« и ») сдвигают разряды влево и вправо. Оператор ~

инвертирует биты значения операнда, заменяя каждый 0 на 1 и каждую 1 на 0; получившееся в результате значение называется дополнением до единицы.

1.3.1. Поразрядный оператор И

В таблице 5 перечислены правила, по которым работает поразрядная операция **И**. Данная таблица отображает результат (C) применения поразрядного оператора **И** (&) к двум однобитовым операндам (A и B). В соответствии с правилами для поразрядной операции **И** результат будет равен 1 только в том случае, когда оба операнда равны 1; в противном случае результат будет равен 0.

Таблица 5. Правила работы операции И

A	&	B	==	C
0	&	0	==	0
0	&	1	==	0
1	&	0	==	0
1	&	1	==	1

Как правило, поразрядный оператор **И** используется для маскирования части значения таким образом, чтобы в результат попала только часть этого значения. При использовании в качестве операнда оператора **И** это число называется **маской**. Разряды значения, соответствующие 1 в маске, попадают в результат. А разряды, соответствующие 0, обнуляются в результате. Они сбрасываются в 0, безотносительно к их первоначальному значению.

На рис. 2 показано действие логической операции **И**. Разряды маски, равные 1, позволяют разрядам операнда попадать в результат без изменения. Разряды маски, равные 0, блокируют соответствующие разряды операнда, сбрасывая их также в 0 и в результате.

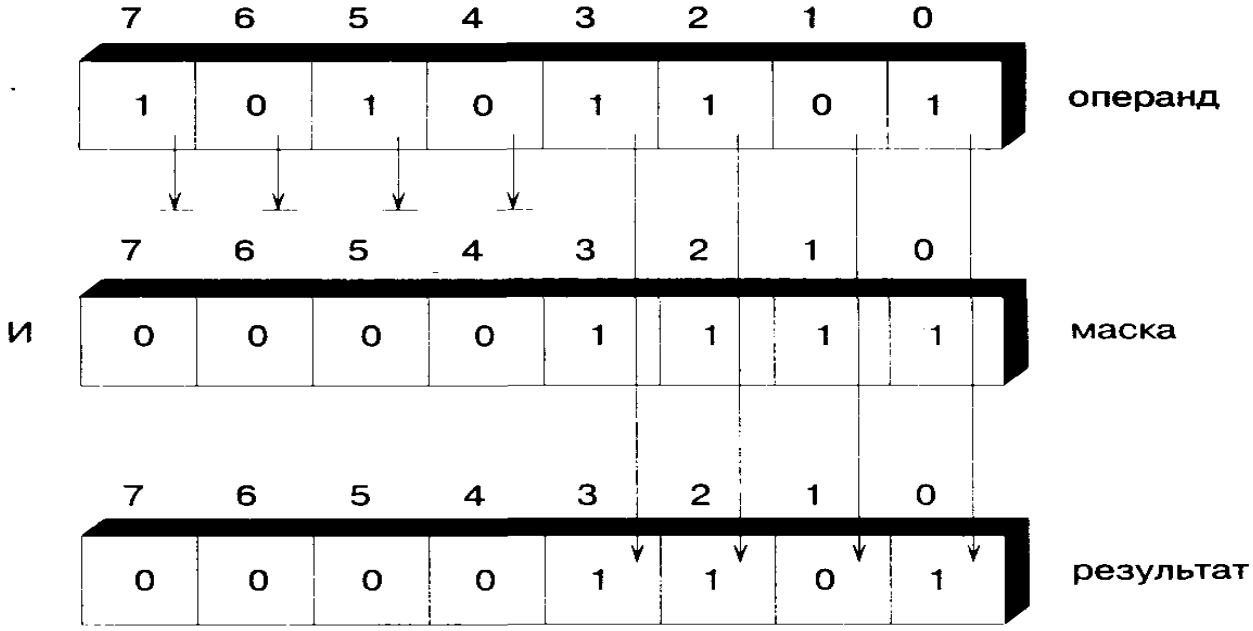

Рисунок 2 – Результат логической операции **И** над операндом и маской

1.3.2. Поразрядный оператор ИЛИ

Правила для поразрядной операции **ИЛИ** дополняют правила для поразрядной операции **И**. В таблице 6 перечислены результаты поразрядной операции **ИЛИ** для всех комбинаций двух одноразрядных операндов. Результат будет равен 0 только в том случае, если оба операнда равны 0; в противном случае, т.е. если хотя бы один из операндов (или оба сразу) равен 1, результат будет равен 1.

Таблица 6. Правила работы операции ИЛИ

A	|	**B**	==	**C**
0	|	0	==	0
0	|	1	==	1
1	|	0	==	1
1	|	1	==	1

Поразрядный оператор ИЛИ часто используется для вставки (или включения) нужных разрядов в значения. На рис. 3 показано действие оператора ИЛИ. Разряды маски, равные 1, устанавливают равными 1 соответствующие разряды результата невзирая на значения соответствующих разрядов операнда. Разряды маски, равные 0, позволяют разрядам операнда попадать в результат без изменения.

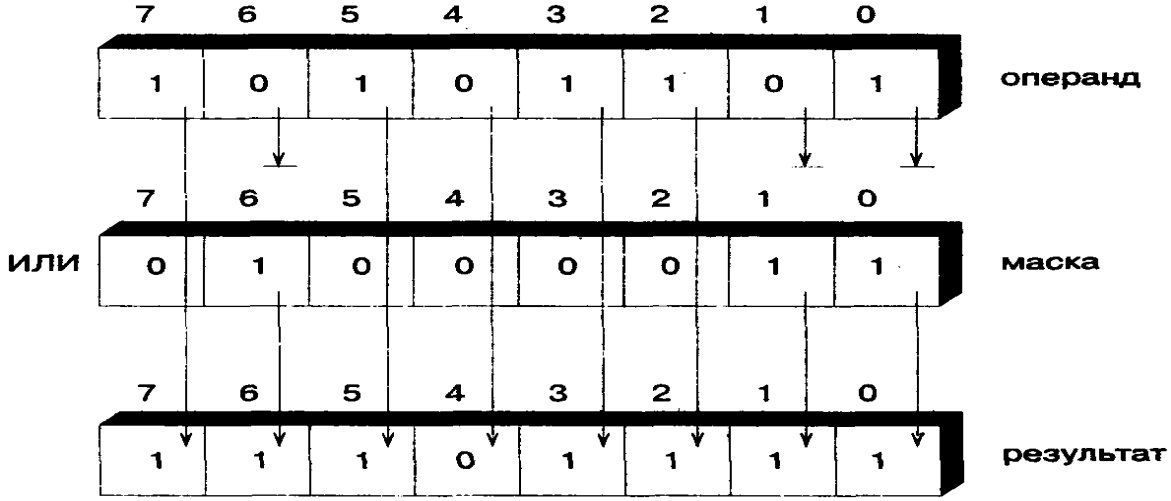

Рисунок 3 – Результат логической операции **ИЛИ** над операндом и маской

1.3.3. Поразрядный оператор исключающего ИЛИ

В таблице 7 отображены результаты применения оператора поразрядного исключающего **ИЛИ** к каждой возможной комбинации двух однобитовых операндов. Если два операнда равны, результат будет равен нулю; если отличаются друг от друга — единице. Если посмотреть на исключающее ИЛИ под другим углом зрения, то можно сказать, что наличие единицы в одном операнде переключает соответствующий разряд другого операнда с 1 на 0 или с 0 на 1.

Таблица 7. Правила работы операции исключающее ИЛИ

A	^	**B**	==	**C**
0	^	0	==	0
0	^	1	==	1
1	^	0	==	1
1	^	1	==	0

Рис. 4 иллюстрирует результат применения маски к значению операнда при использовании оператора исключающего ИЛИ. Разряды маски, равные 1, переключают значения битов операнда на противоположные. Нулевые биты маски позволяют соответствующим битам операнда перейти в результат без изменений.

.

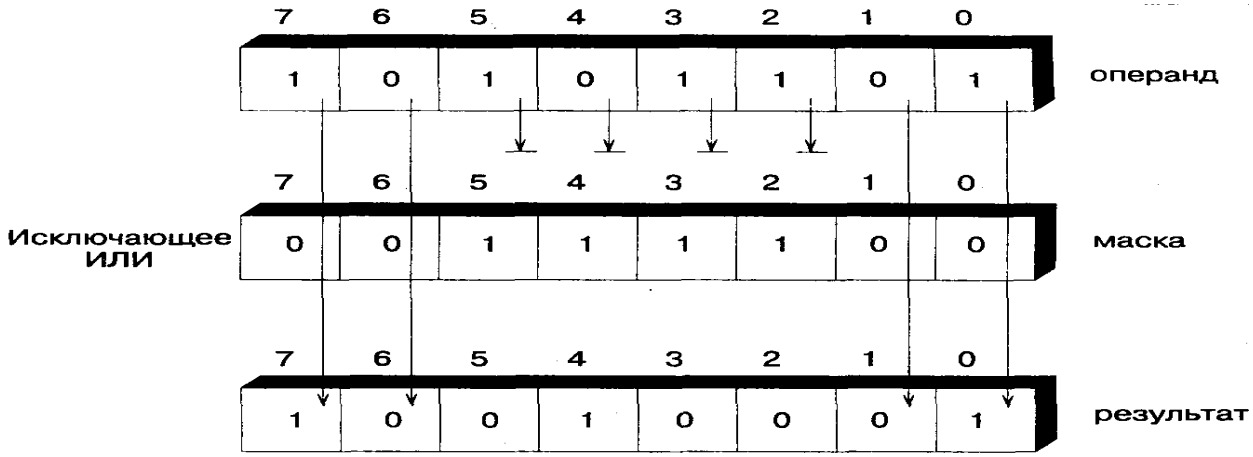

Рисунок 4 – Результат операции исключающего **ИЛИ** над операндом и маской

1.3.4. Сдвиг битов влево

Оператор сдвига влево («) используется для выполнения сдвига битов влево на нуль или другое число позиций. Оператор

C = A « 3;

присваивает переменной **C** значение переменной **A**, сдвинутое на три бита влево.

В двоичном представлении сдвиг битов на одну позицию влево эквивалентен умножению исходного значения на 2, т.е. на основание двоичной системы счисления. То же самое справедливо и для значений, записанных в других системах счисления. Например, сдвиг цифр в десятичном числе 1234 влево на одну позицию и внесение нуля справа даст в результате число 12340, которое в 10 раз больше исходного числа по основанию 10. Но в языке C# все сдвиги выполняются в двоичном формате.

Поскольку двоичные компьютеры выполняют сдвиг влево очень быстро, то операции поразрядного сдвига влево являются быстрым способом умножения чисел на степень числа два.

Рис. 5 иллюстрирует действие оператора **C = A « B;** , где A равно двоичному значению 10101101, а B равно 3. Как видно по результату (C), освобождающиеся справа позиции заполняются нулями, в то время как сдвигаемые влево старшие биты теряются (что иллюстрируется на рисунке символом электрического заземления).

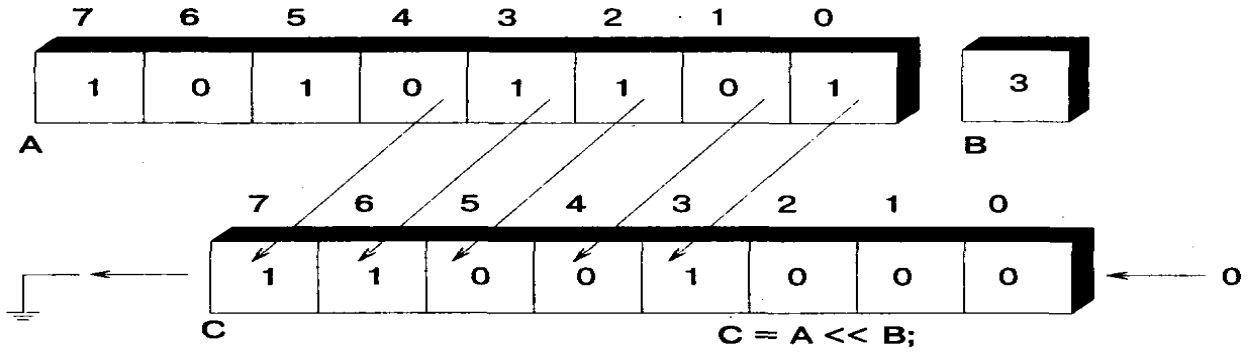

Рисунок 5 – Действия оператора поразрядного сдвига влево

1.3.5. Сдвиг битов вправо

Оператор поразрядного сдвига вправо работает аналогично оператору сдвига влево, но сдвигает биты в другом направлении. Оператор

C = A » 3;

присваивает переменной C значение переменной A, сдвинутое на три бита вправо.

Если операции сдвига влево умножают числа на 2, то операции сдвига вправо выполняют деление чисел. Аналогичное утверждение справедливо для основания любой системы счисления. Сдвиг числа 12340 вправо на одну цифру и отбрасывание нуля даст число 1234, которое мы получили бы при делении исходного числа на основание 10. Но в языке C# все сдвиги выполняются в двоичном формате.

Рис. 6 иллюстрирует действие оператора C = A » B;

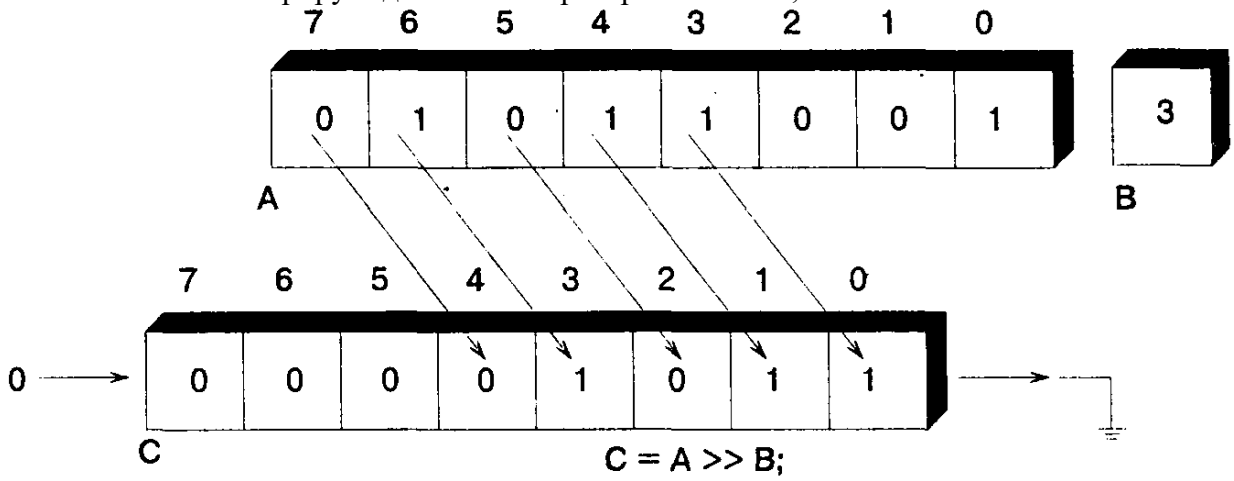

Рисунок 6 – Действия оператора поразрядного сдвига вправо

1.3.6. Дополнение до единицы

Оператор поразрядного дополнения до единицы, представленный символом ~ (тильда), переключает все биты значения с 1 на 0 и с 0 на 1. Дополнение до единицы является унарным оператором, который предшествует своему операнду подобно унарному плюсу или минусу (как в выражениях -34 или **+i**).

Унарную операцию дополнения до единицы используется для выполнения поразрядного отрицания двоичных значений. Например, выражение

A = ˜B;

присваивает переменной A значение дополнения или отрицания переменной B.

Дополнение до единицы — это обычная операция низкого уровня, которую большинство компьютерных процессоров выполняет в мгновение ока. Являясь двоичной логической операцией, дополнение до единицы имеет два назначения: инвертирование значений "истина" и "ложь", а также преобразование отрицательных значений в положительные и наоборот.

Если 0 представляет "ложь", то -0 является "истиной", так как отрицание нуля дает -1 (беззнаковое число 65535), а любое ненулевое число означает "истину".

Кроме этого, операция дополнения до единицы играет роль в преобразовании отрицательных и положительных целых. Прибавление единицы к результату выражения дополнения до единицы формирует другое значение, называемое дополнением до двух.

Рис. 7 иллюстрирует действие оператора дополнения до единицы, примененного к двоичному значению 00101100.

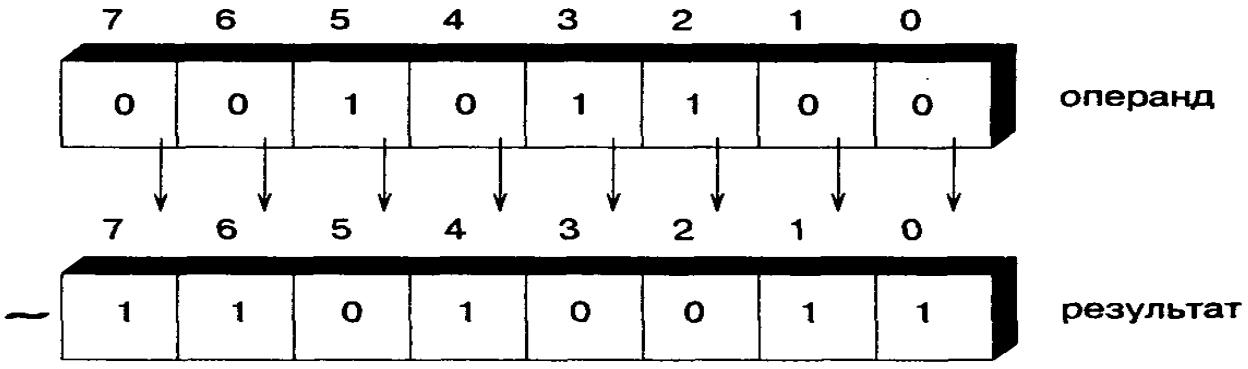

Рисунок 7 – Действия оператора дополнения до единицы

1.4. Системы счисления

Системой счисления называется совокупность правил записи чисел. Системы счисления подразделяются на позиционные и непозиционные. Как позиционные, так и непозиционные системы счисления используют определенный набор символов — цифр, последовательное сочетание которых образует число.

Непозиционные системы счисления появились раньше позиционных. Они характеризуются тем, что в них символы, обозначающие то или иное число, не меняют своего значения в зависимости от местоположения в записи этого числа. Классическим примером такой системы счисления является римская. В ней для записи чисел используются буквы латинского алфавита. При этом буква I означает единицу, V — пять, X — десять, L — пятьдесят, C — сто, D — пятьсот, M — тысячу. Для получения количественного эквивалента числа в римской системе счисления необходимо просто просуммировать количественные эквиваленты входящих в него цифр. Исключение из этого правила составляет случай, когда младшая цифра идет перед старшей, — в этом случае нужно не складывать, а вычитать число вхождений этой младшей цифры.

Например, число **577** в римской системе счисления представляется как **DLXXII**

DLXXII = 500 + 50 + 10 + 10 + 5 + 1 + 1 = 577.

Другой пример: **CDXXIX** = 500 - 100 + 10 + 10 - 1 + 10 = **429**.

В позиционной системе счисления количество символов в наборе равно **основанию** системы счисления. Место каждой цифры в числе называется **позицией**. Номер позиции символа в числе называется **разрядом**.

Число **A**, имеющее **n** разрядов целой части и **m** разрядов дробной части, изображается последовательностью цифр $\mathbf{a_k}$, каждая из которых имеет свой вес **W**, определяемый её позицией, т.е. номером разряда в условной записи числа:

$$\mathbf{A = a_{n-1}\, a_{n-2} \ldots a_k \ldots a_1\, a_0\, ,\, a_{-1}\, a_{-2} \ldots a_{-m}}$$
$$\mathbf{W = p^{n-1}\, p^{n-2} \ldots p^k \ldots p^1\, p^0\, ,\, p^{-1}\, p^{-2} \ldots p^{-m}}$$

В общем случае, количественный эквивалент некоторого положительного числа **A** в позиционной системе счисления можно представить выражением:

$$\mathbf{A = a_{n-1}*p^{n-1} + a_{n-2}*p^{n-2} + \ldots + a_1*p^1 + a_0*p^0\, ,\, a_{-1}*p^{-1} + a_{-2}*p^{-2} + \ldots + a_{-m}*p^{-m},}$$

где **p** — основание системы счисления (некоторое целое положительное число);

a — цифра данной системы счисления.

Для получения количественного эквивалента числа в некоторой позиционной системе счисления необходимо сложить произведения количественных значений цифр на степени основания, показатели которых равны номерам разрядов.

В вычислительных машинах на различных этапах составления программы и решения задачи применяются следующие системы счисления:

- двоичная p = 2, a_k = {0, 1};

- восьмеричная p = 8, a_k = {0, 1, . . . 7};
- шестнадцатеричная p = 16, a_k = {0, 1, . . . 9, A, B, C, D, E, F};
- десятичная p = 10, a_k = {0, 1, . . . 9}.

Двоичная система счисления применяется для представления чисел в вычислительных машинах. Запись числа представляет собой последовательность нулей и единиц. Например, запись 1010,011 в двоичной системе соответствует десятичному числу 10,375:

$$\mathbf{A = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 + 0*2^{-1} + 1*2^{-2} + 1*2^{-3}\ .}$$

Сложение и вычитание двоичных чисел выполняется так же, как и для других позиционных систем счисления, например десятичной. Точно так же выполняются заем и перенос единицы из (в) старший разряд. Например:

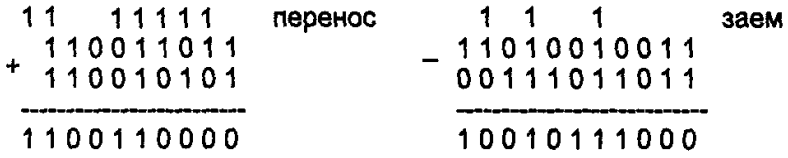

Рисунок 8 – Сложение и вычитание двоичных чисел

В таблице 8 приведены степени двойки, а в таблице 9 соответствие чисел в различных системах счисления.

Таблица 8. Степени двойки

k	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10
2^k	0,0625	0,125	0,25	0,5	1	2	4	8	16	32	64	128	256	512	1024

Таблица 9. Представление чисел в системах счисления

Десятичное число	Двоичная тетрада	Шестнадцатеричное число
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

Восьмеричная и шестнадцатеричная системы применяются для записи команд программы на машинном языке.

Десятичная система используется для записи чисел в программах.

Для перевода целых чисел из одной системы счисления в другую применяют метод деления переводимого числа на основание новой системы счисления.

Для перевода дробных чисел из одной системы счисления в другую применяется метод умножения переводимого числа на основание новой системы счисления.

Примеры перевода чисел представлены в таблице 10.

Таблица 10. Примеры перевода чисел

Деление	A^i	a_i
37:2	18	1 a_0
18:2	9	0 a_1
9:2	4	1 a_2
4:2	2	0 a_3
2:2	1	0 a_4
1:2	0	1 a_5
$37_{10} = 100101_2$		

Умножение	b_i	
0,37*2	0 b_{-1}	0,74
0,74*2	1 b_{-2}	0,48
0,48*2	0 b_{-3}	0,96
0,96*2	1 b_{-4}	0,92
0,92*2	1 b_{-5}	0,84
0,84*2	1 b_{-6}	0,68
$0,37_{10} = 0,010111_2$		

2. Рабочее задание

Задание 1. Руководствуясь теоретическим материалом раздела 1 изучить возможности языка C# по созданию приложений, использующие логические операции, и выполнить практически все примеры, описанные в этом разделе.

Задание 2. Разработать приложение с заголовком «Логические операторы», которое демонстрирует действия логических операторов. Результат выполнения должен отображаться в десятичном и шестнадцатеричном виде (один из вариантов решения представлен на рис. 8).

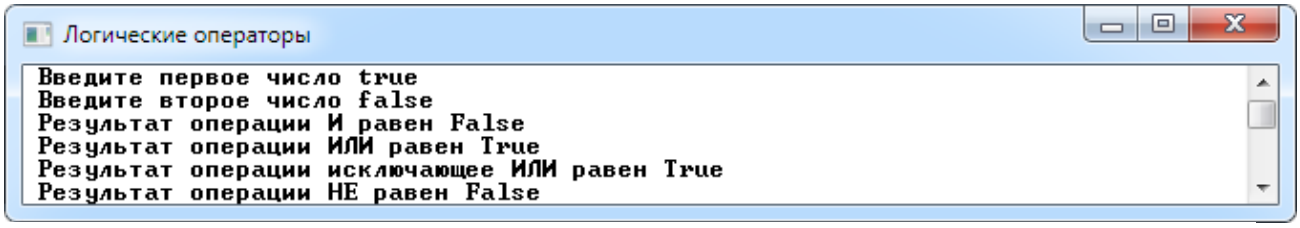

Рисунок 8 – Вариант консольного приложения «Логические операторы»

Задание 3. Разработать приложение с заголовком «Поразрядные операторы», которое демонстрирует действия поразрядных логических операторов. Результат выполнения должен отображаться в десятичном и шестнадцатеричном виде (один из вариантов решения представлен на рис. 9). В отчете представить блок-схему, исходный код и результаты выполнения приложения.

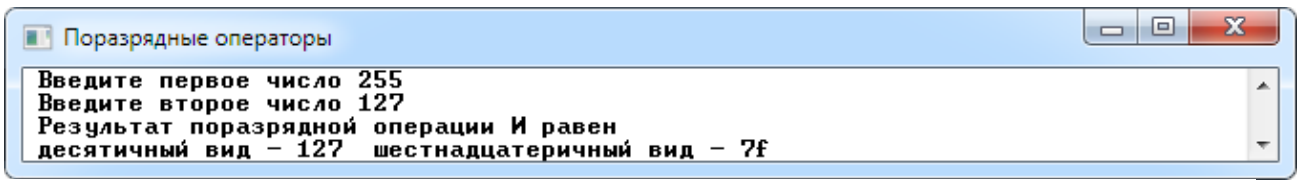

Рисунок 9 – Вариант консольного приложения «Поразрядные операторы»

3. Контрольные вопросы

1. Перечислите и дайте характеристику операторам отношения.
2. Перечислите и дайте характеристику логическим операторам.
3. Перечислите и дайте характеристику поразрядным операторам.

4. Что такое системы счисления и их разновидности?
5. Как осуществляется перевод целых чисел из одной системы счисления в другую?
6. Как осуществляется перевод дробных чисел из одной системы счисления в другую?

Литература

1. Голощапов А.Л. Microsoft Visual Studio 2010. – СПб.:БХВ-Петербург, 2011. – 544 с.: ил.
2. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1. Пер. с англ. - М.: «Русская Редакция», 2002.- 576 с.: ил.
3. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 2. Пер. с англ. - М.: «Русская Редакция», 2002.- 624 с.: ил.
4. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4.0. Пер. с англ. - М.: Издательский дом "Вильямс", 2011. — 1392 с.: ил.
5. Фленов М.Е. Библия C#. - СПб.: БХВ-Петербург, 2011. – 560с.: ил.
6. Шилдт Г. C# Учебный курс. – СПб.: Питер, Издательская группа BHV, 2003. – 512 с.: ил.