

## Лабораторная работа №4: представление целых чисел

### Цель:

Целью данной работы является получение базовых навыков работы с двоичным представлением целых чисел

### Задание:

Разработать набор консольных программ на языке C#, реализующих следующие функции:

1. Преобразование переменной типа байт со знаком (sbyte) в массив, содержащий двоичный код числа, лежащего в переменной. Сигнатура функции может выглядеть следующим образом:  
`static int[] sbyteToBin(sbyte n)`  
где возвращаемое значение имеет тип массив, а параметр имеет тип байт со знаком.

Преобразование массива целых чисел в строку. Сигнатура функции может выглядеть следующим образом: `static string binToStr(int[] n)`  
где возвращаемое значение имеет тип строка, а параметр имеет тип массив.

#### Пример работы программы:

```
Введите число в диапазоне от -128 до 127: 2
Число в двоичном виде: 0000 0010

Введите число в диапазоне от -128 до 127: -2
Число в двоичном виде: 1111 1110
```

**Примечание:** для получения двоичного представления числа, необходимо использовать операции побитового сдвига (<< или >>) и логическое И (&).

2. Преобразование строки, длиной не более 8 символов, содержащей двоичное представление числа (string), в массив, содержащий двоичное представление числа (int[]). Сигнатура функции может выглядеть следующим образом: `static int[] strToBin(string n)`  
где возвращаемое значение имеет тип массив, а входной параметр является строкой.

Преобразование массива, содержащего двоичный код числа (int[]), в переменную типа целое байт со знаком (sbyte), содержащую это число. Сигнатура функции может выглядеть следующим образом: `static sbyte binToSByte (int[] n)`  
где возвращаемое значение имеет тип байт со знаком, а входной параметр является массивом.

#### Пример работы:

```
Введите 8 бит числа: 00000010
Число в десятичном виде: 2

Введите 8 бит числа: 11111110
Число в десятичном виде: -2
```

**Примечание:** для получения целого числа по его двоичному представлению, необходимо использовать операции побитового сдвига (<< или >>) и логическое ИЛИ (|).

3. Преобразование целого числа (int), в диапазоне от -128 до 127, в массив, содержащего двоичный код этого числа (int[]). Сигнатура функции может выглядеть следующим образом:  
`static int[] intToBin(int n)`  
где возвращаемое значение имеет тип массив, а входной параметр является целым числом.  
Для реализации представления отрицательных чисел, рекомендуется использовать две вспомогательные функции:  
`static int[] invers(int[] n) //функция инвертирования значений`  
`static int[] addOne(int[] n) //функция добавления единицы к числу`

первая из которых инвертирует число, представленное в двоичной форме, а вторая добавляет к младшему разряду числа, представленного в двоичной форме, единицу.

Для вывода результата работы функции, можно использовать функцию `binToStr`, разработанную ранее.

Пример работы:

```
Введите целое число в диапазоне от -128 до 127: 7
Число в двоичном виде: 0000 0111

Введите целое число в диапазоне от -128 до 127: -7
Число в двоичном виде: 1111 1001
```

**Примечание:** для получения двоичного представления числа, необходимо использовать последовательное вычисление целой части и остатка от деления исходного числа.

- Преобразование массива целых чисел, содержащего двоичный код числа (`int[]`), в переменную типа целое число (`int`), содержащую это число. Сигнатура функции может выглядеть следующим образом: `static int binToInt(int[] n)`

где возвращаемое значение имеет тип целое число, а входной параметр является массивом.

Для преобразования отрицательных чисел следует использовать разработанные ранее функции `invers` и `addOne`.

Для получения двоичного представления числа в виде массива, может быть использована функция `strToBin`, разработанная ранее.

Пример работы:

```
Введите 8 бит числа: 00000111
Число в десятичном виде: 7

Введите 8 бит числа: 11111001
Число в десятичном виде: -7
```

**Примечание:** для получения десятичного представления числа, необходимо использовать последовательное перемножение разрядов двоичного представления числа, на соответствующую степень двойки.

## Краткая справка:

Для выполнения данной лабораторной работы, вам могут понадобиться следующие **типы данных**:

```
sbyte sb = 0;      //целое число в диапазоне от -128 до 127
int i = 0;         //целое число в диапазоне от -2 147 483 648 до 2 147 483 647
string st = "";    //строка, может содержать произвольное число текстовых символов
bool c = true;     //логическая переменная, может иметь значения только true и false
```

Операции ввода и вывода в языке C# осуществляются при помощи методов класса `Console`.

Для того что бы **вывести** сообщение без перехода на новую строку, можно использовать:

```
Console.Write("Оба сообщения ");
Console.Write("будут на одной строке");
```

**Примечание:** для того, чтобы приложение не закрывалось сразу же по завершению работы, можно использовать команду: `Console.ReadKey()`; тогда, приложение будет ожидать нажатия клавиши перед закрытием.

Для того что бы перейти на новую строку, либо вывести сообщение и потом перейти на новую строку, можно использовать:

```
Console.WriteLine("Первая строка");
Console.WriteLine();
Console.WriteLine("Третья строка");
```

для того, чтобы вывести значение переменной на экран, можно использовать:

```
int a = 1;
int b = 2;

Console.WriteLine("Содержимое переменной b {1}, содержимое переменной a {0}.", a, b);
```

Результат:

```
Содержимое переменной b 2, содержимое переменной a 1.
```

То есть, на место цифр в фигурных скобках, будет подставлено значение соответствующих переменных. Альтернативным вариантом, является сложение строк:

```
Console.WriteLine("Содержимое переменной b " + b.ToString() + ", содержимое переменной a " + a.ToString() + ".");
```

где ToString(), метод преобразования целого числа в строку. Результатом будет точно такая же строка что и в предыдущем примере. Переменные строкового типа, преобразовывать не обязательно.

**Арифметические операции** над целочисленными типами данных в языке C# выглядят следующим образом:

```
int a = 5;
int b = 2;
int c = 0;

c = a + b; //сложение, c = 7
c = a - b; //вычитание, c = 3
c = a * b; //умножение, c = 10
c = a / b; //целочисленное деление, c = 2
c = a % b; //остаток от деления, c = 1
```

**Целочисленное деление** – целая часть результата деления. Следует помнить, что если и делитель и делимое представлены целочисленными типами данных, то результатом будет целое число.

Например выражение: Console.WriteLine((1/2).ToString()); выдаст на экран 0, а не 0,5.

**Остаток от деления**, это то, что остаётся от числа, после вычитания из него делителя максимально возможное число раз ( $5 - 2 = 3 - 2 = 1$ ). В общем виде, формулу можно записать как:  $O = a - (a/b)$

где O – остаток от деления, a – делимое, b – делитель, / – целочисленное деление.

Целочисленные типы данных, могут быть организованы в **массивы**:

```
int[] m = new int[8]; //массив из 8 целых чисел
```

массив с именем m, содержит 8 ячеек которые могут содержать целочисленные значения, доступ к которым можно осуществить по их номеру, который называется индексом:

```
int a = m[3]; //в переменную a будет записано значение 4й ячейки массива m
Console.WriteLine("Содержимое 1й ячейки массива m = " + m[0].ToString());
```

Следует помнить, что нумерация ячеек массива начинается с 0. То есть, первая ячейка массива имеет индекс 0, а не 1.

Перебирать ячейки массивов для вывода, обработки или заполнения удобно при помощи **циклов**. Например, цикл заполнения массива из 8ми элементов числами от 1 до 8 и цикл вывода содержимого массива на экран, будут выглядеть следующим образом:

```
int[] m = new int[8]; //массив из 8 целых чисел

for(int i = 0; i < 8; i++) //значение переменной i, будет изменяться от 0 до 7 по 1 за шаг цикла
{
    m[i] = i + 1;
}
```

```
for (int i = 0; i < 8; i++)
{
    Console.Write(m[i].ToString() + " ");
}
```

**Примечание**, цикл for, может содержать более одного счётчика:

```
for (int i = 0, j = 7; i < 8; i++, j--)
{
    Console.Write(m[i].ToString() + " "); //вывод массива от 1 до 8 элемента
    Console.Write(m[j].ToString() + " "); //вывод массива от 8 до 1 элемента
}
```

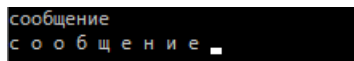
**Строки** являются подвидом массивов, но не имеют заранее определённого размера и содержат коды символов. Пример вывода на экран

```
string str = "сообщение";

Console.WriteLine(str); //вывод на экран всей строки

for (int i = 0; i < str.Length; i++)
    Console.Write(str[i] + " "); //вывод на экран символов строки
```

Результат:



В случае, если необходимо присвоить значение переменной одного типа, переменной другого типа, необходимо выполнить **приведение типов**:

```
int a = 7;
sbyte b = (sbyte)(a); //явное приведение типа
sbyte c = Convert.ToSByte(a); //использование конвертера типов
```

**Примечание:** приведение типов необходимо выполнять только в случае, если значение более вместительного типа данных присваивается в переменную менее вместительного типа данных. При приведении типов возможны потери информации.

В случае, если необходимо получить **числовое значение из строковой переменной**, можно воспользоваться одним из следующих методов:

```
string num = "69";

int n = int.Parse(num); //преобразование всей строки в целое число
Console.Write(n.ToString() + " ");

n = num[0] - '0'; //вычитание кода символа '0' из кода символа лежащего в [0]
Console.Write(n.ToString() + " ");

n = (int)char.GetNumericValue(num[1]); //получение числового значения символа лежащего в ячейке [1]
Console.Write(n.ToString() + " ");
```

В первом случае происходит преобразование всей строки в число, во втором и третьем случаях получение числа, лежащего в первой и второй ячейках строки соответственно.

Для реализации выбора, или ветвления кода, в языке C# существует **условный оператор**:

```
if (a > b)
    Console.WriteLine("Условие истинно.");
else
    Console.WriteLine("Условие ложно.");
```

Соответственно, если содержимое переменной a больше чем содержимое переменной b, то будет выведено первое сообщение, если меньше либо равно, то второе.

Ниже приведена таблица часто используемых в условиях операторов сравнения:

Операция:	Значение:
>	больше
<	меньше
==	равно
!=	Не равно
>=	Больше либо равно
<=	Меньше либо равно
!	Не (изменяет true на false и false на true)

Так же в языке C# существует **тернарный оператор**, который позволяет сокращать условия. Например, выбор и присваивание наибольшего из двух чисел можно записать как:

```
r = (a > b) ? a : b;
```

В большинстве ситуаций, имеет смысл выносить отдельные, обособленные, фрагменты кода, в **функции**. Оформление фрагментов кода в виде функций, облегчает их отладку и повторное использование. Пример функции, печатающей на экран массив:

```
static void printM(int[] m)
{
    for(int i = 0; i < m.Length; i++)
    {
        Console.Write(m[i].ToString() + " ");
    }

    Console.WriteLine();
}
```

В случае, если в процессе работы над программой появиться необходимость вывести на экран содержимое массива, можно будет это сделать простым вызовом данной функции:

```
int[] m = new int[8];
...
printM(m);
```

**Примечание:** служебное слово static означает что функция будет размещена в статической памяти и необходимо только в консольных приложениях.

Помимо прочего, функции могут иметь возвращаемое значение. Например, функция, вычисляющая сумму элементов массива, может быть записана как:

```
static int sumM(int[] m)
{
    int result = 0;

    for (int i = 0; i < m.Length; i++)
    {
        result += m[i];
    }

    return result; //возвращаемое значение
}
```

Соответственно, вызов функции будет выглядеть как:

```
int[] m = new int[8];
...
int sum = sumM(m); //переменная sum будет содержать сумму элементов массива
```

Язык C#, поддерживает следующие **побитовые операции**:

```
sbyte a = 7;           //0000 0111 содержимое переменной в двоичном виде
sbyte b = 2;           //0000 0010 содержимое переменной в двоичном виде
sbyte c = 0;           //0000 0000 содержимое переменной в двоичном виде

c = (sbyte)(a >> 1);    //0000 0011 результат побитового сдвига вправо на 1 разряд
c = (sbyte)(a << 1);    //0000 1110 результат побитового сдвига влево на 1 разряд
c = (sbyte)(a << 2);    //0001 1100 результат побитового сдвига влево на 2 разряда

c = (sbyte)(a | b);     //0000 0111 результат операции "логическое ИЛИ"
c = (sbyte)(a & b);     //0000 0010 результат операции "логическое И"
c = (sbyte)(a ^ b);     //0000 0101 результат операции "исключающее ИЛИ"
c = (sbyte)(~a);        //1111 1101 результат операции "логическое отрицание"
```

В качестве примера, выбраны переменные типа “байт со знаком”, размером в 8 бит. Размер любого типа данных в байтах можно узнать при помощи функции:

```
int s = sizeof(byte);    //в переменную будет записано значение 1
```

указав в качестве параметра требуемый тип данных.

Операция **побитового сдвига** выполняет смещение всех бит числа в соответствующую сторону, на указанное число разрядов. Пример:

0000 0111 >> 1 = 0000 0011

0000 1010 << 3 = 0101 0000

**Примечание:** с математической точки зрения, операция побитового сдвига влево эквивалентна умножению, а побитового сдвига вправо – целочисленному делению.

Операция **логическое ИЛИ** имеет следующую таблицу истинности:

a	b	ИЛИ
0	0	0
1	0	1
0	1	1
1	1	1

**Таблица истинности** показывает результат выполнения операции при заданных входных данных.

Примеры:

0000 0111 | 0010 0010 = 0010 0111

0101 0101 | 1010 1010 = 1111 1111

Операция **логическое И** имеет следующую таблицу истинности:

a	b	И
0	0	0
1	0	0
0	1	0
1	1	1

Примеры:

$$0000\ 0111 \& 0010\ 0010 = 0000\ 0010$$

$$0101\ 0101 \& 1010\ 1010 = 0000\ 0000$$

Операция **исключающее ИЛИ** имеет следующую таблицу истинности:

a	b	Исключающее ИЛИ
0	0	0
1	0	1
0	1	1
1	1	0

Примеры:

$$0000\ 0111 \& 0010\ 0010 = 0010\ 0101$$

$$0101\ 0101 \& 1010\ 1010 = 1111\ 1111$$

Операция **логическое отрицание** имеет следующую таблицу истинности:

a	НЕ
0	1
1	0

Примеры:

$$\sim 0000\ 0111 = 1111\ 1000$$

$$\sim 1010\ 1010 = 0101\ 0101$$

Алгоритмически, **преобразование десятичного числа в двоичное**, выглядит как последовательное вычисление целой части и остатка от деления на 2, исходного числа. При этом целая часть делится до тех пор, пока не станет равен 0, а остаток от деления записывается как значение соответствующего разряда двоичного представления числа, начиная с младшего (самого правого). Пример:

Шаг	Число N	Операция	Целая часть	Остаток от деления	Двоичное число
1	5	$5 / 2$	2	1	0000 0001
2	2	$2 / 2$	1	0	0000 0001
3	1	$1 / 2$	0	1	0000 0101

Результат:  $5_{10} \rightarrow 0000\ 0101_2$

Для переменных со знаком, размером в восемь бит, таким образом могут быть записаны числа от 0 до 127. **Отрицательные числа** в диапазоне от -128 до -1 могут быть записаны в виде дополнительного кода числа. **Дополнительный код** числа вычисляется путём применения к двоичному коду числа операций инвертирования и добавления единицы к младшему разряду.

**Инвертирование**, это операция эквивалентная логическому отрицанию. Пример:

$$0100\ 1101 \rightarrow 1011\ 0010$$

**Добавление единицы** к младшему разряду числа осуществляется при помощи двоичного сложения. Таблица результатов **двоичного сложения** выглядит следующим образом:

a	b	+
0	0	0
1	0	1
0	1	1
1	1	10

Пример добавления единицы к младшему разряду числа:

Шаг	Двоичное число	Результат операции	Добавляемое значение
1	0000 1011	$1 + 1 = 10$	1
2	0000 1010	$1 + 1 = 10$	10
3	0000 1000	$1 + 0 = 1$	100
4	0000 1100	-	-

Таким образом, получение дополнительного кода для числа -5 будет выглядеть как:

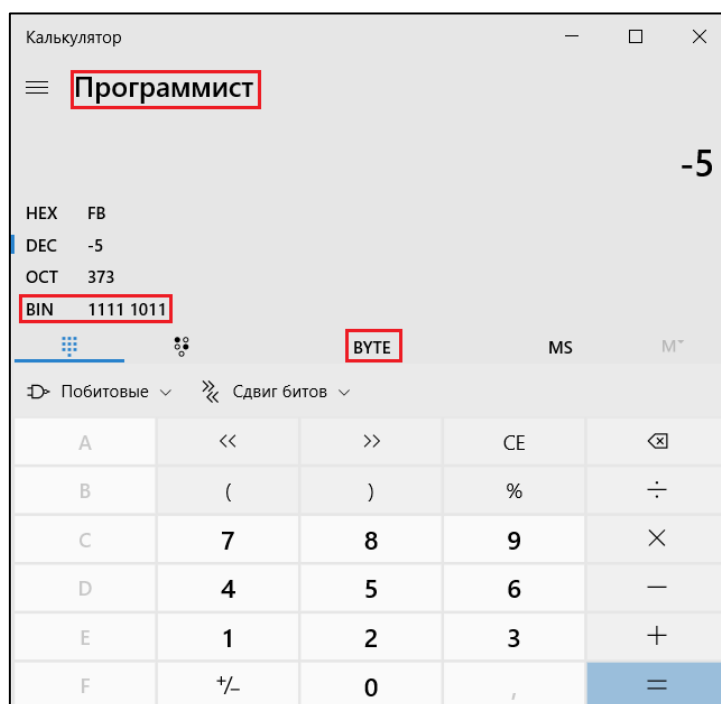
$$5_{10} \rightarrow 0000\ 0101_2$$

$$\sim 0000\ 0101 \rightarrow 1111\ 1010$$

$$1111\ 1010 + 0000\ 0001 = 1111\ 1011$$

Результат:  $-5_{10} = 1111\ 1011_2$

Проверить результат можно используя стандартный калькулятор Windows, в режиме “Программист”:



**Примечание:** желательно указать размер переменной – BYTE.

Преобразование числа **из двоичной в десятичную**, происходит путём вычисления суммы последовательных перемножений разрядов двоичного числа на соответствующую степень двойки, начиная с младшего разряда:



Шаг	Двоичное число	Степень двойки	Операция	Результат
1	0000 0101	0	$0 + 1 * 2^0$	1
2	0000 0101	1	$1 + 0 * 2^1$	1
3	0000 0101	2	$1 + 1 * 2^2$	5

Результат:  $0000\ 0101_2 \rightarrow 5_{10}$

**Примечание:** преобразование отрицательных чисел происходит после применения к ним инвертирования и добавлении единицы. То есть, тем же способом что и при преобразовании отрицательных чисел в двоичную форму представления.

### Список литературы:

Документация по C#: <https://docs.microsoft.com/ru-ru/dotnet/csharp/>

Условные операторы:

<https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/conditional-operator>

Оператор for: <https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/for>

Побитовые операторы и операторы сдвига: <https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators>

Двоичные числа и двоичная арифметика:

<https://www.intuit.ru/studies/curriculum/16009/courses/541/lecture/12186>