|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования  «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Расчетно-графическая работа | | |
| по дисциплине «Операционные системы, среды и оболочки» | | |
| **Разработка программы для изучения внутренних форматов хранения числовых типов данных в ОЗУ** | | |
|  | | |
|  | Бригада 10 | ХОХЛОВА НАТАЛЬЯ |
|  | Группа ПМ-04 | ШАХОВ ДАНИИЛ |
|  | Вариант 16 |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | Преподаватели | Кобылянский В. Г. |
|  |  | Сивак М.А. |
| Новосибирск, 2022 | | |

Содержание

[Задание 3](#_Toc104128296)

[Введение 4](#_Toc104128297)

[Теоретическая часть 5](#_Toc104128298)

[Описание программы 8](#_Toc104128299)

[Результаты тестирования 9](#_Toc104128300)

[Тестирование корректных входных данных для целых типов. 9](#_Toc104128301)

[Тестирование некорректных входных данных для целых типов. 12](#_Toc104128302)

[Тестирование различных входных данных для вещественных типов. 15](#_Toc104128303)

[Тестирование режима контроля 16](#_Toc104128304)

[Итог тестирования 16](#_Toc104128305)

[Заключение 17](#_Toc104128306)

[Список использованных источников 18](#_Toc104128307)

[Приложение 19](#_Toc104128308)

[Текст программы (на языке C# с использованием Windows Forms) 19](#_Toc104128309)

[Program.cs 19](#_Toc104128310)

[Functional\_Perevod.cs 19](#_Toc104128311)

[Education\_mode.cs 23](#_Toc104128312)

[Control\_mode.cs 27](#_Toc104128313)

# Задание

Разработка программы для изучения внутренних форматов хранения числовых типов данных в ОЗУ.

# Введение

Любая информация в ЭВМ представляется в виде двоичных кодов. Отдельные элементы двоичного кода, принимающие значение 0 или 1, называют **разрядами** или **битами**. Память компьютера условно делиться на отсеки или ячейки, каждая из которых имеет свой номер. Нумерация начинается с нуля.

Минимальной адресуемой ячейкой памяти называется байт – 8 двоичных разрядов. Порядковый номер байта называется его **адресом**.

Наибольшую последовательность битов, которую процессор может обрабатывать как единое целое, называют **машинным словом**.

Знание внутренних форматов хранения данных различных типов является необходимым условием для понимания основных принципов работы компьютера.

Перевод из внешнего во внутренний формат можно проводить двумя способами. Первый способ основан на использовании классических алгоритмов перевода из десятичной системы счисления в двоичную (именно этот метод будет реализованной в данной работе).

Второй способ основан на чтении всех байтов участка памяти, выделенного для хранения переменной, в которую записывается заданное число, и использует встроенные возможности алгоритмических языков.

В данной работе будет рассматриваться три группы типов данных, для которых характерны свои правила внутреннего представления, а именно:

**Целые беззнаковые типы:**

* byte
* ushort
* uint
* ulong

**Целые знаковые типы:**

* sbyte
* short
* int
* long

**Вещественные типы с плавающей запятой:**

* float
* double

# Теоретическая часть

В процессе выполнения данной расчетно-графической работы необходимо создать программу, реализующую представление внутренних форматов хранения числовых типов данных в ОЗУ. В данном пункте будут изложены алгоритмы перевода чисел из внешнего представления во внутреннее формат.

Для начала необходимо уточнить, что для перевода чисел из внешнего формата во внутренний будем использовать способ, основанный на использовании классических алгоритмов перевода из десятичной системы счисления в двоичную. Для этого введем понятия:

Прямой код. Получается путем непосредственного преобразования значения целого числа в двоичную систему счисления. Знак хранится в старшем разряде представления, остальные разряды содержат модуль числа в двоичном коде.

|  |  |
| --- | --- |
| Десятичное число | Двоичное число в прямом коде (8 бит) |
| -127 | 1111 1111 |
| -100 | 1110 0100 |
| 0 | 0000 0000 |
| 100 | 0110 0100 |
| 127 | 0111 1111 |

Данный код предельно прост, но неудобен для выполнения арифметических операций в ЭВМ. Для того, чтобы свести операцию вычитания к операции сложения используют обратный и дополнительный код.

Обратный код. Получается из прямого путем инвертирования каждого разряда двоичного представления модуля числа. *Замечание*: знаковый разряд в инвертировании не участвует.

Дополнительный код. Получается из обратного кода путем прибавления единицы. Данный код был предложен для удобства выполнения арифметических операций. Дополнительный код отрицательного числа можно получить другими способами:

а) перевести в двоичную систему положительное число, по модулю на единицу меньшее заданного, а затем полученное представление инвертировать;

б) перевести в двоичную систему положительное число, равное (2n – N), где n – количество разрядов числа в дополнительном коде, N – модуль числа, подвергаемого переводу.

Теперь перейдем непосредственно к внутренним форматам различных типов.

1. Целые беззнаковые числа

Беззнаковые типы могут хранить только неотрицательные значения переменных. Для хранения целочисленных типов данных в ОЗУ могут выделяться 1,2,4 или 8 байтов, в зависимости от этого каждый тип характеризуется определенным диапазоном допустимых значений переменных.

|  |  |  |  |
| --- | --- | --- | --- |
| **Длина (байт)** | **Тип C#** | **Системный тип C#** | **Диапазон значений** |
| 1 | byte | System. Byte | 0…255 (0…28 – 1) |
| 2 | ushort | System. UInt16 | 0…65535 (0…216 – 1) |
| 4 | uint | System. UInt32 | 0…4294967295 (0…232 – 1) |
| 8 | ulong | System.UInt64 | 0…18446744073709551615 (0…264 – 1) |

Для перевода внешнего представления во внутреннее для беззнакового типа необходимо перевести число в двоичную систему счисления (т. е. записать прямой код, учитывая длину равную количеству бит).

Например, 10010=110 01002.

|  |  |
| --- | --- |
| Длина (байт) | Внутреннее представление |
| 1 | 0110 0100 |
| 2 | 0000 0000 0110 0100 |
| 4 | 0000 0000 0000 0000 0000 0000 0110 0100 |
| 8 | 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0110 0100 |

1. Целые знаковые числа

Знаковые типы могут хранить как положительные, так и отрицательные значения переменных. Для хранения целочисленных типов данных в ОЗУ могут выделяться 1,2,4 или 8 байтов, в зависимости от этого каждый тип характеризуется определенным диапазоном допустимых значений переменных.

|  |  |  |  |
| --- | --- | --- | --- |
| **Длина (байт)** | **Тип C#** | **Системный тип C#** | **Диапазон значений** |
| 1 | sbyte | System.SByte | –128…127 (–27…27 – 1) |
| 2 | short | System. Int16 | –32768… 32767 (–215…215 – 1) |
| 4 | int | System. Int32 | –2147483648…2147483647 (–231…231 – 1) |
| 8 | long | System. Int64 | –9223372036854775808…  9223372036854775807 (–263 …263– 1) |

Для перевода внешнего представления во внутреннее для знакового типа для начало необходимо определить знак. Отрицательному числу последнему биту (т.к. нумерация идет справа налево) соответствует 1, положительному 0.

Если число положительное записываем прямой код, учитывая длину (аналогично как в беззнаковых типах). Если же число отрицательное считаем дополнительный код модуля этого числа.

1. Вещественные числа с плавающей запятой

Внутренние форматы с плавающей точкой основаны на экспоненциальной форме представления числа в виде нормализованной мантиссы (m) и порядка (k): y = m\*2k

Мантисса двоичного числа считается нормализованной, если ее значение находится в диапазоне 1≤m<2 (в двоичной системе счисления).

Современные ЭВМ для хранения вещественных чисел в основном используют три поля: знак мантиссы (s), дробная часть мантиссы (f) и экспонента (e).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Длина (байт)** | **Тип C#** | **Системный тип C#** | **Диапазон значений** | **Точность** | **Константа** |
| 4 | float | System.Single | 3,4\*10-38 … 3,4\*1038 | 7-8 | 127 |
| 8 | double | System. Double | 5\*10-324 … 1,7\*10308 | 15-16 | 1023 |

формат Single (32 бита)

|  |  |  |
| --- | --- | --- |
| s (1 бит) | e (8 битов) | f (23 бита) |

формат Double (64 бита)

|  |  |  |
| --- | --- | --- |
| s (1 бит) | e (11 битов) | f (52 бита) |

Для представления вещественного числа необходимо для начала определить знак мантиссы (s=1 если число отрицательное, иначе s=0). Далее работаем с модулем числа. Данное десятичное число записываем в двоичной система счисления и нормализуем его. Порядок е будет представлен прямым кодом числа равному сумме константы и k. Мантиссой f будут являться символы после запятой нормализованного числа (добавление незначащих нулей или обрезки числа с учетом количества бит, выделенных на мантиссу).

# Описание программы

Программа имеет два режима работы: обучения и контроль.

В режиме обучения входные данные – внешнее представление числа и тип данных. На выходе – внутреннее представление. В режиме контроль на входе добавляется внутреннее представление, а на выходе при верном написании представления выводится двоичный код зеленым. В противном случае (в случае ошибки) - красным.

Функции:

* public static string Perevod\_byte(string value)

Переводит число типа byte в двоичный код.

* public static string Perevod\_ushort(string value)

Переводит число типа ushort в двоичный код.

* public static string Perevod\_uint(string value)

Переводит число типа uint в двоичный код.

* public static string Perevod\_ulong(string value)

Переводит число типа ulong в двоичный код.

* public static string Perevod\_sbyte(string value)

Переводит число типа sbyte в двоичный код.

* public static string Perevod\_short(string value)

Переводит число типа short в двоичный код.

* public static string Perevod\_int(string value)

Переводит число типа int в двоичный код.

* public static string Perevod\_long(string value)

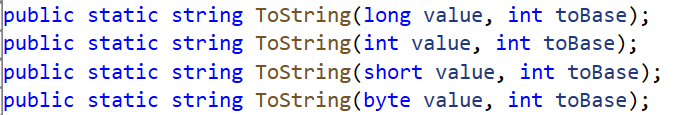
Переводит число типа long в двоичный код.

* public static string Perevod\_real(string value, int constanta, int length, int method, int accuracy)

Переводит число вещественного типа в двоичный код. На входе помимо самого числа подается константа, длина двоичного кода (в байтах), количество знаков порядка, заданная точность.

Так же используется встроенный класс для перевода целых чисел в двоичную систему счисления:

public static class Convert



Полный исходный код проекта представлен в Приложении.

# Результаты тестирования

## Тестирование корректных входных данных для целых типов.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тип | Скрин программы  (входные и выходные данные) | Назначение |
| Беззнаковые типы | | | |
| 1 | byte |  | Значение из диапазона. |
| 2 | byte |  | Граница диапазона. |
| 3 | ushort |  | Значение из диапазона. |
| 4 | ushort |  | Граница диапазона. |
| 5 | uint |  | Значение из диапазона. |
| 6 | uint |  | Граница диапазона. |
| 7 | ulong |  | Значение из диапазона. |
| 8 | ulong |  | Граница диапазона. |
| Знаковые типы | | | |
| 9 | sbyte |  | Положительное значение из диапазона. |
| 10 | sbyte |  | Отрицательное значение из диапазона. |
| 11 | sbyte |  | Граница диапазона. |
| 12 | short |  | Положительное значение из диапазона. |
| 13 | short |  | Отрицательное значение из диапазона. |
| 14 | short |  | Граница диапазона. |
| 15 | int |  | Положительное значение из диапазона. |
| 16 | int |  | Отрицательное значение из диапазона. |
| 17 | int |  | Граница диапазона. |
| 18 | long |  | Положительное значение из диапазона. |
| 19 | long |  | Отрицательное значение из диапазона. |
| 20 | long |  | Граница диапазона. |

## Тестирование некорректных входных данных для целых типов.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тип | Скрин программы  (входные и выходные данные) | Назначение |
| Беззнаковые типы | | | |
| 1 | byte |  | Отрицательное число. |
| 2 | byte |  | Число на 1 выходящее из диапазона. |
| 3 | ushort |  | Отрицательное число. |
| 4 | ushort |  | Число на 1 выходящее из диапазона. |
| 5 | uint |  | Отрицательное число. |
| 6 | uint |  | Число на 1 выходящее из диапазона. |
| 7 | ulong |  | Отрицательное число. |
| 8 | ulong |  | Число на 1 выходящее из диапазона. |
| Знаковые типы | | | |
| 9 | sbyte |  | Число на 1 выходящее из отрицательного диапазона. |
| 10 | sbyte |  | Число на 1 выходящее из положительного диапазона. |
| 11 | short |  | Число на 1 выходящее из отрицательного диапазона. |
| 12 | short |  | Число на 1 выходящее из положительного диапазона. |
| 13 | int |  | Число на 1 выходящее из отрицательного диапазона. |
| 14 | int |  | Число на 1 выходящее из положительного диапазона. |
| 15 | long |  | Число на 1 выходящее из отрицательного диапазона. |
| 16 | long |  | Число на 1 выходящее из положительного диапазона. |
| 17 | long |  | Вещественное число. |

## Тестирование различных входных данных для вещественных типов.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тип | Скрин программы  (входные и выходные данные) | Назначение |
| Корректные входные данные | | | |
| 1 | float |  | Положительное число из диапазона. |
| 2 | float |  | Отрицательное число из диапазона. |
| 3 | double |  | Положительное число из диапазона. |
| 4 | double |  | Отрицательное число из диапазона. |
| Некорректные входные данные | | | |
| 5 | double |  | Неправильный знак разделения целой и дробной части. |

## Тестирование режима контроля

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тип | Скрин программы  (входные и выходные данные) | Назначение |
| 1 | sbyte |  | Введено верное внутреннее представление. |
| 2 | byte |  | Введено неверное внутреннее представление. |
| 3 | float |  | Не введено число. |
| 4 | int |  | Не введено внутреннее представление числа. |
| 5 | - |  | Не выбран тип данных. |

## Итог тестирования

Все тесты показали ожидаемый результат.

# Заключение

В ходе работы мы:

* + Ознакомились с перевод числовых данных в двоичный код;
  + Разобрали понятия «прямой код», «обратный код», «дополнительный код» и установили связь и различия между ними;
  + Разобрали понятия «мантиссы» и «порядка» для представления вещественных типов;
  + Ознакомились с внутренним представлением числовых типов данных;
  + Получили навыки разработки графического интерфейса с помощью Windows Forms на языке C#;
  + Получили навыки тестирования разработанной программы.

# Список использованных источников

1. <https://programforyou.ru/calculators/number-systems>
2. <https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/overview/?view=netdesktop-6.0>

# Приложение

## Текст программы (на языке C# с использованием Windows Forms)

### Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace test

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Menu());

}

}

}

### Functional\_Perevod.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace test

{

class Functional\_Perevod

{

//Итоговое число без 0

static string result0;

//Итоговое число

static string result;

//Количество незначащих нулей

static int length0;

//Знак числа (+/-)

static string znak;

public static string Perevod\_byte(string value)

{

byte value0;

result = "";

value0 = byte.Parse(value);

result0 = Convert.ToString(value0, 2);

length0 = 8 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "0";

result = result + result0;

return result;

}

public static string Perevod\_ushort(string value)

{

ushort value0;

result = "";

value0 = ushort.Parse(value);

result0 = Convert.ToString(value0, 2);

length0 = 16 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "0";

result = result + result0;

return result;

}

public static string Perevod\_uint(string value)

{

uint value0;

result = "";

value0 = uint.Parse(value);

result0 = Convert.ToString(value0, 2);

length0 = 32 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "0";

result = result + result0;

return result;

}

public static string Perevod\_ulong(string value)

{

ulong value0;

result = "";

value0 = ulong.Parse(value);

result0 = Convert.ToString(unchecked((long)value0), 2);

length0 = 64 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "0";

result = result + result0;

return result;

}

public static string Perevod\_sbyte(string value)

{

sbyte value0;

result = "";

value0 = sbyte.Parse(value);

if (value0 < 0)

{

znak = "1";

if (value0 == -128)

result = "0000000";

else

{

value0 = Math.Abs(value0);

value0--;

result0 = Convert.ToString(value0, 2);

//обратный код

result0 = result0.Replace("0", "2");

result0 = result0.Replace("1", "0");

result0 = result0.Replace("2", "1");

//перевод незначащих 0 в 1

length0 = 7 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "1";

}

}

else

{

znak = "0";

result0 = Convert.ToString(value0, 2);

length0 = 7 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "0";

}

result = znak + result + result0;

return result;

}

public static string Perevod\_short(string value)

{

short value0;

result = "";

value0 = short.Parse(value);

if (value0 < 0)

{

znak = "1";

if (value0 == -32768)

result = "000000000000000";

else

{

value0 = Math.Abs(value0);

value0--;

result0 = Convert.ToString(value0, 2);

//обратный код

result0 = result0.Replace("0", "2");

result0 = result0.Replace("1", "0");

result0 = result0.Replace("2", "1");

//перевод незначащих 0 в 1

length0 = 15 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "1";

}

}

else

{

znak = "0";

result0 = Convert.ToString(value0, 2);

length0 = 15 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "0";

}

result = znak + result + result0;

return result;

}

public static string Perevod\_int(string value)

{

int value0;

result = "";

value0 = int.Parse(value);

if (value0 < 0)

{

znak = "1";

if (value0 == -2147483648)

result = "0000000000000000000000000000000";

else

{

value0 = Math.Abs(value0);

value0--;

result0 = Convert.ToString(value0, 2);

//обратный код

result0 = result0.Replace("0", "2");

result0 = result0.Replace("1", "0");

result0 = result0.Replace("2", "1");

//перевод незначащих 0 в 1

length0 = 31 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "1";

}

}

else

{

znak = "0";

result0 = Convert.ToString(value0, 2);

length0 = 31 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "0";

}

result = znak + result + result0;

return result;

}

public static string Perevod\_long(string value)

{

long value0;

result = "";

value0 = long.Parse(value);

if (value0 < 0)

{

znak = "1";

if (value0 == -9223372036854775808)

result = "000000000000000000000000000000000000000000000000000000000000000";

else

{

value0 = Math.Abs(value0);

value0--;

result0 = Convert.ToString(value0, 2);

//обратный код

result0 = result0.Replace("0", "2");

result0 = result0.Replace("1", "0");

result0 = result0.Replace("2", "1");

//

length0 = 63 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "1";

}

}

else

{

znak = "0";

result0 = Convert.ToString(value0, 2);

length0 = 63 - result0.Length;

for (int i = 0; i < length0; i++)

result = result + "0";

}

result = znak + result + result0;

return result;

}

// constanta - константа для вещественного числа; length - количество бит; method - количество бит порядка; accuracy - точность для вещественного числа

public static string Perevod\_real(string value, int constanta, int length, int method, int accuracy)

{

double value0;

//целая часть вещественного числа

int whole;

//порядок

string order;

//дробная часть вещественного числа

double fractional;

//мантисса вещественного числа

string mantissa = "";

value0 = double.Parse(value);

//целая часть числа

whole = (int)Math.Truncate(value0);

//дробная часть числа

fractional = Math.Abs(value0 - whole);

//программа перевода вещественного числа в двоичную

int flag = 1;

for (int i = 0; (i < length + accuracy) && flag == 1; i++)

{

if (fractional \* 2 >= 1)

{

mantissa = mantissa + "1";

fractional = fractional \* 2 - 1;

}

else

{

mantissa = mantissa + "0";

fractional \*= 2;

}

if (fractional == 1)

{

flag = 0;

}

}

//определение знака мантиссы

if (value0 < 0)

{

znak = "1";

whole = Math.Abs(whole);

}

else

znak = "0";

if (whole == 0)

{

//первое вхождение 1 для определения значения порядка

int indexOf1 = mantissa.IndexOf("1");

constanta = constanta - indexOf1 - 1;

mantissa = mantissa.Substring(indexOf1 + 1);

}

else

{

order = Convert.ToString(whole, 2);

//подсчет значения порядка

constanta = constanta + order.Length - 1;

//добавление мантиссы

mantissa = order.Substring(1, order.Length - 1) + mantissa;

}

//порядок в двоичной системе

order = Convert.ToString(constanta, 2);

//количество незначащих нулей порядка

length0 = method - order.Length;

for (int i = 0; i < length0; i++)

order = "0" + order;

result = znak + order + mantissa;

result = result.Substring(0, length);

return result;

}

}

}

### Education\_mode.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static test.Functional\_Perevod;

namespace test

{

public partial class Education\_mode : Form

{

//Введенное число

static string value;

//Итоговое число

static string result;

public Education\_mode()

{

InitializeComponent();

}

//Вернуться в меню

private void button1\_Click(object sender, EventArgs e)

{

this.Hide();

Menu menu\_e = new Menu();

menu\_e.Show();

}

//Подсчет результата

private void translation\_e\_Click(object sender, EventArgs e)

{

value = textBox1.Text;

//число не введено

if (value == null || value == "")

{

textBox2.Text = "Введите число";

return;

}

//не выбран тип данных

if (comboBox1.SelectedItem == null)

{

textBox2.Text = "Выберите тип данных";

return;

}

string selectedType = comboBox1.SelectedItem.ToString();

switch (selectedType)

{

case "sbyte":

try

{

result = Functional\_Perevod.Perevod\_sbyte(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "short":

try

{

result = Functional\_Perevod.Perevod\_short(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "int":

try

{

result = Functional\_Perevod.Perevod\_int(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "long":

try

{

result = Functional\_Perevod.Perevod\_long(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "byte":

try

{

result = Functional\_Perevod.Perevod\_byte(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "ushort":

try

{

result = Functional\_Perevod.Perevod\_ushort(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "uint":

try

{

result = Functional\_Perevod.Perevod\_uint(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "ulong":

try

{

result = Functional\_Perevod.Perevod\_ulong(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "float":

try

{

result = Functional\_Perevod.Perevod\_real(value, 127, 32, 8, 7);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "double":

try

{

result = Functional\_Perevod.Perevod\_real(value, 1023, 64, 11, 15);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

}

textBox2.Text = result;

}

}

}

### Control\_mode.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static test.Functional\_Perevod;

namespace test

{

public partial class Control\_mode : Form

{

//Введенное число

static string value;

//Итоговое число

static string result;

public Control\_mode()

{

InitializeComponent();

}

//Вернуться в меню

private void button1\_Click(object sender, EventArgs e)

{

this.Hide();

Menu menu\_с = new Menu();

menu\_с.Show();

}

//Подсчет результата

private void translation\_Click(object sender, EventArgs e)

{

this.textBox2.ForeColor = System.Drawing.Color.Black;

value = textBox1.Text;

///число не введено

if (value == null || value == "")

{

textBox2.Text = "Введите число";

return;

}

///не введено внутреннее представление

if (textBox3.Text == null || textBox3.Text == "")

{

textBox2.Text = "Введите внутреннее представление числа";

return;

}

//не выбран тип данных

if (comboBox1.SelectedItem == null)

{

textBox2.Text = "Выберите тип данных";

return;

}

string selectedType = comboBox1.SelectedItem.ToString();

switch (selectedType)

{

case "sbyte":

try

{

result = Functional\_Perevod.Perevod\_sbyte(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "short":

try

{

result = Functional\_Perevod.Perevod\_short(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "int":

try

{

result = Functional\_Perevod.Perevod\_int(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "long":

try

{

result = Functional\_Perevod.Perevod\_long(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "byte":

try

{

result = Functional\_Perevod.Perevod\_byte(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "ushort":

try

{

result = Functional\_Perevod.Perevod\_ushort(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "uint":

try

{

result = Functional\_Perevod.Perevod\_uint(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "ulong":

try

{

result = Functional\_Perevod.Perevod\_ulong(value);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "float":

try

{

result = Functional\_Perevod.Perevod\_real(value, 127, 32, 8, 7);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

case "double":

try

{

result = Functional\_Perevod.Perevod\_real(value, 1023, 64, 11, 15);

}

//выход за пределы

catch (OverflowException)

{

result = "Некорректные данные";

}

//неверный формат

catch (System.FormatException)

{

result = "Некорректные данные";

}

break;

}

//внутреннее представление совпадает

if (textBox3.Text == result)

this.textBox2.ForeColor = System.Drawing.Color.Green;

//внутреннее представление НЕ совпадает

else

this.textBox2.ForeColor = System.Drawing.Color.Red;

textBox2.Text = result;

result = "";

}

}

}