GeekBrains

Введение в C#

Массивы и строки. Операторы цикла

На этом уроке

1. Изучим типы, хранящие набор значений, и научимся оперировать ими.
2. Изучим операторы цикла, облегчающие работу с перечисляемыми типами.
3. Изучим подробнее строковый тип данных и его составляющие.

Оглавление

[На этом уроке](#)

[Определение массива. Виды массивов. Индексатор массива](#)

[Цикл for](#)

[Циклы с пред- и постусловием](#)

- [while](#)
- [do-while](#)
- [Сценарии использования циклов while и do-while](#)
  - [Безусловный цикл](#)
  - [Пример использования цикла с постусловием](#)
- [Оператор break](#)

[Символы и строки](#)

- [Определение символов и базовые знания](#)
- [Объявление строк](#)
- [Итерация по символам строки](#)
- [Как работает кодировка символов в байты и обратно](#)
- [Краткое знакомство с Unicode](#)

[Практическое задание](#)

[Используемые источники](#)

Определение массива. Виды массивов. Индексатор массива

Массив — это структура данных, содержащая несколько переменных, доступ к которым осуществляется по вычисляемым индексам. Содержащиеся в массиве переменные, также называемые элементами, имеют одинаковый тип. Он называется типом элементов массива.

Так выглядит объявление массива с типом `int`, содержащего в себе числа `1,2,3`:

```
int[] array = { 1, 2, 3 };
```

У массива может быть одно или несколько измерений. Одномерный массив — это подобие списка, двумерный — матрица и так далее.

Число измерений задаётся при объявлении массива с помощью запятых. Если запятых нет, то массив плоский (имеет одно измерение), если одна — двумерный, и так далее. Объявление двумерного массива выглядит так:

```
int[,] a2 = new int[10, 5];
```

А трёхмерного — так:

```
int[,,] a3 = new int[10, 5, 2];
```

Тип массива может быть любым, даже другим массивом. Такие массивы называют массивами массивов — каждый элемент массива также в свою очередь является массивом. Объявление массива массивов типа `int` выглядит так:

```
int[][] a = new int[3][];
```

Здесь `int[]` задаёт тип — массив `int`, а вторая пара квадратных скобок, как и прежде, обозначает массив.

Чтобы прочитать или записать значения элементов массива, используют его специальное свойство — индексатор. Позиция элемента в массиве называется индексом. Индексы начинаются с нуля. Операция чтения элемента с индексом 2 выглядит так:

```
int[] array = { 1, 2, 3 };
Console.WriteLine(array[2]); // 3
```

Операция присвоения значения элемента массива выглядит так:

```
int[] array = { 1, 2, 3 };
array[1] = 4; // {1, 4, 3}
```

Для массива массивов перед чтением/записью необходимо убедиться, что элементы-массивы тоже инициализированы:

```
int[][] a = new int[3][];
a[0] = new int[3];
a[1] = new int[3];
a[2] = new int[3];
int a00 = a[0][0];
a[1][1] = 1;
```

В случае с многомерными массивами, присвоение выглядит так:

```
int[,] matrix = new int[5, 5];
matrix[2, 2] = 1;
```

Внимание! *Так как при расположении в памяти элементы массива не связаны между собой ссылками, а просто расположены подряд, изменение размера массива невозможно. В таких случаях нужно создать новый массив большего размера и скопировать в него значения из старого массива.*

Цикл `for`

Для перебора элементов массива, чтения и записи его элементов, существуют специальные операторы — циклы. Циклы позволяют выполнять один и тот же набор операций со всеми элементами массива до тех пор, пока выполняется условие, указанное в объявлении цикла. Рассмотрим цикл for и принцип его работы:

```
int[] array = { 1, 2, 3, 4, 5 };
for (int i = 0; i < 5; i++)
{
    Console.WriteLine(array[i]);
}
```

Как правило, объявление цикла содержит три раздела: инициализатора, условия и итератора. Каждый из этих разделов — необязательный, но в большинстве случаев они присутствуют в описании цикла и разделены точкой с запятой:

```
int[] array = { 1, 2, 3, 4, 5 };
/*
  * 1. Инициализатор
  * 2. Условие
  * 3. Итератор
```

```
  * ┌────1────┐ ┌──2──┐┌─3─┐ */
for (int i = 0; i < 5; i++)
{
    Console.WriteLine(array[i]);
}
```

Раздел инициализатора выполняется перед первой итерацией цикла. Раздел условия проверяется перед каждой итерацией. Раздел итератора выполняется перед каждой итерацией.

В разделе инициализатора объявляются переменные, которые будут доступны в описании цикла и в его теле. Переменные объявляются через запятую:

```
for (int i = 0, j = 0; i < array.Length; i++)
```

Обратите внимание, что в инициализаторе можно объявить переменные только одного и того же типа. Если необходимо объявить несколько переменных разного типа, это нужно сделать до объявления цикла.

В разделе условия указано выражение, при истинности которого цикл продолжает свое выполнение. Если условие не указано явно, будет использоваться значение true (цикл будет выполняться бесконечно). В случае перебора элементов массива необходимо проверять, что текущий индекс не выходит за пределы массива. В противном случае выполнение цикла приведет к ошибке времени выполнения.

В разделе итератора объявляются выражения, которые будут выполняться после каждой итерации цикла. Обычно в этом разделе производят инкремент переменных цикла.

Для получения длины массива следует использовать свойство `Length`, а в случае с многомерными массивами — метод `GetLength` с указанием измерения:

```
int[,] matrix = new int[5, 5];

matrix[2, 2] = 1;

for (int i = 0; i < matrix.GetLength(0); i++)
{
    for (int j = 0; j < matrix.GetLength(1); j++)
    {
        System.Console.Write($"{matrix[i, j]} ");
    }
    System.Console.WriteLine();
}
```

Вывод программы:

```
0 0 0 0 0
0 0 0 0 0
0 0 1 0 0
0 0 0 0 0
0 0 0 0 0
```

Циклы с пред- и постусловием

C# предоставляет циклы с предусловием и циклы с постусловием. Циклы с предусловием сначала проверяют истинность условия и только после этого выполняют очередную итерацию. Циклы с предусловием сначала выполняют очередную итерации и только потом проверяют условие на истинность. Это означает, что циклы с предусловием могут не выполнить ни одной итерации, а циклы с постусловием точно выполнят хотя бы одну.

`while`

Цикл while — цикл с предусловием. Описание цикла содержит только один раздел — раздел условий:

```
int n = 0;
while (n < 5)
{
    Console.WriteLine(n);
    n++;
}
```

Чтобы цикл имел возможность корректно завершиться, в теле цикла следует позаботиться об инкременте необходимых переменных. Если из тела цикла, описанного выше, убрать инкремент переменной `n`, условие `n < 0` всегда будет верным и цикл никогда не завершится. В программировании существуют задачи, подразумевающие использование бесконечных циклов (их также называют безусловными циклами). Мы рассмотрим пример с безусловным циклом далее в этом разделе.

`do-while`

Цикл `do-while` схож с циклом `while`, но, в отличие от него, это цикл с постусловием. Этот пример кода выводит на экран числа от 0 до 5 так же, как и пример с использованием цикла `while`:

```
int n = 0;
do
{
```

```
    Console.WriteLine(n);
    n++;
} while (n < 5);
```

На примере итерации по массива вывод обоих циклов будет одинаковый. Но на практике в зависимости от поставленной задачи используется тот или иной цикл. Стоит отметить, что для итерации удобнее использовать цикл `for`, циклы `do-while` и `while` в первую очередь предназначены на повторение итераций до тех пор, пока заданное условие выполняется

Рассмотрим возможные сценарии использования обоих циклов.

Сценарии использования циклов `while` и `do-while`

Безусловный цикл

Безусловный цикл представляет собой набор операций, выполняющихся бесконечно до тех пор, пока работа программы не будет завершена принудительно. В то же время, такой цикл можно прервать операторами `return` или `break` (способы завершения циклов мы рассмотрим далее).

Для написания безусловного цикла подойдет любой из циклов `do-while`, `while`, но на практике применяют цикл `while`. Такой код будет бесконечно опрашивать пользователя и выводить на экран введенную строку:

```
while(true)
{
    string input = Console.ReadLine();
    Console.WriteLine(input);
}
```

На практике безусловный цикл применяется в тех случаях, когда условие выхода из цикла не может быть вычислено на момент объявления самого цикла. Этот код конвертирует пользовательский ввод в число и выводит результат деления на консоль. В случае, если пользователь ввел ноль, нам необходимо завершить работу программы:

```
while (true)
{
    string input = Console.ReadLine();
    double number = Convert.ToDouble(input);
    if(number == 0)
    {
        return;
```

```
    }
    Console.WriteLine(1 / number);
}
```

Чаще всего безусловный цикл используется в веб-сервисах и системных службах — они обрабатывают поступившие к ним запросы до тех пор, пока не получат запрос на завершение работы.

Также безусловный цикл может найти применение в разработке игр: мы постоянно выводим новый кадр игры на экран до тех пор, пока пользователь не выберет завершение игры.

Пример использования цикла с постусловием

Циклы с постусловием хорошо подходят для ситуаций, когда нам нужно повторять одни и те же операции, пока нужное нам условие не станет верным, например, до тех пор, пока пользователь не введёт пароль определённой длины и не подтвердит его:

```
string password;
do
{
    Console.WriteLine("Задайте пароль длиной не менее 5 знаков");
    password = Console.ReadLine();
} while (password.Length < 5);

string repeatedPassword;
do
{
    Console.WriteLine("Повторите пароль:");
    repeatedPassword = Console.ReadLine();
} while (repeatedPassword != password);

Console.WriteLine("Пароль успешно установлен");
```

Обратите внимание на то, как записаны условия в этих циклах: они содержат условия, выполнение которых мы хотим прекратить. До тех пор, пока они выполняются, выполняются и циклы `do-while`. Если условия перестают выполняться, значит, введенные пользователем данные удовлетворяют нашим условиям.

Оператор `break`

Ранее мы рассматривали завершение программы оператором `return`, но нам также доступна возможность прерывания цикла и возврат в родительский блок программы. Оператор `break`

прерывает выполнение любого цикла: `for`, `do-while`, `while`, а также оператора `switch`. Проще всего показать работу оператора можно на примере поиска значений в массиве:

```
string[] students =
{
    "Иванов",
    "Петров",
    "Сидоров",
    "Петрова",
    "Филиппова",
    "Егоров",
    "Козлова",
};

Console.Write("Введите фамилию студента:");
string lastName = Console.ReadLine();
bool isFound = false;

for (int i = 0; i < students.Length; i++)
{
    if (students[i] == lastName)
    {
         Console.WriteLine($"Студент с фамилией {lastName} находится под номером
{i + 1}");
        isFound = true;
        break;
    }
}

if (!isFound)
{
    Console.WriteLine("Такого студента нет в списке :(");
}
```

Мы последовательно сравниваем указанную фамилию с фамилиями студентов. Когда мы находим соответствие, дальнейшее выполнение цикла не имеет смысла. Чтобы выйти из цикла и продолжить выполнение операций, написанных вне цикла, мы прерываем цикл оператором `break`.

Символы и строки

Определение символов и базовые знания

Ранее мы уже рассматривали тип данных `string` и упоминали, что строка — это набор символов. Символы в языке C# задаются типом `char`. В переменную с таким типом можно сохранить только один символ:

```
char c = 'C';
```

Символьный тип данных имел широкое применения в языках C и C++, так как в некоторых случаях работа с символами дает преимущество по скорости и экономию по памяти в сравнении со строками. Также в будущем мы познакомимся с посимвольным чтением файлов. Такой способ чтения позволяет хранить в файлах структурированные данные и восстанавливать структуру при чтении. Популярный пример такого формата файлов — `CSV`. При помощи символов-разделителей `CSV` способен хранить таблицы с текстовыми данными. Такой файл можно в будущем открыть в табличных процессорах, например Microsoft Excel.

Так как всё в памяти компьютера представлено в виде чисел, тип `char` не будет исключением. Каждый символ имеет числовое представление и мы можем представить любое число в виде символа (важно помнить, что формат `char` имеет размерность в два байта). При обработке эти числа переводятся в символьное представление в соответствии с той или иной таблицей символов. Другими словами, когда мы записываем текст в блокнот и сохраняем его в файл, мы записываем на диск байтовое представление символов. Когда мы открываем этот файл на чтение в текстовом режиме (например, в том же блокноте), текстовый редактор подставляет для каждого байта соответствующий символ из таблицы символов, установленной по умолчанию. Сложные форматы файлов хранят не только данные о символах, но и данные о кодировке, чтобы отображение введенного текста было корректным. Для лучшего понимания рассмотрим пример работы с текстовым файлом.

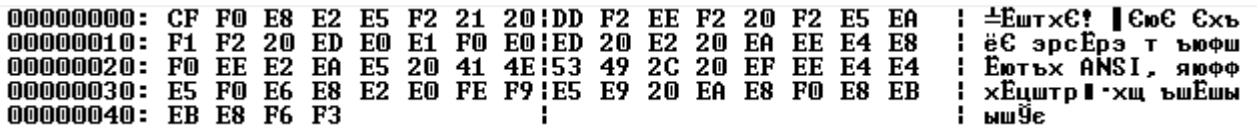
Создадим текстовый файл в кодировке ANSI (кодировку можно выбрать при сохранении файла в любом текстовом редакторе, даже в системном Блокноте) со следующим содержимым:

```
Привет! Этот текст набран в кодировке ANSI, поддерживающей кириллицу
```

Если открыть этот файл в двоичном редакторе — программе, отображающей содержимое файла как есть безо всяких преобразований, мы увидим следующее:

```
00000000: CF F0 E8 E2 E5 F2 21 20|DD F2 EE F2 20 F2 E5 EA  | Привет! Этот тек
00000010: F1 F2 20 ED E0 E1 F0 E0|ED 20 E2 20 EA EE E4 E8  | ст набран в коди
00000020: F0 EE E2 EA E5 20 41 4E|53 49 2C 20 EF EE E4 E4  | ровке ANSI, подд
00000030: E5 F0 E6 E8 E2 E0 FE F9|E5 E9 20 EA E8 F0 E8 EB  | ерживающей кирил
00000040: EB E8 F6 F3            |                        | лицу
```

Слева мы видим, что на самом деле файл хранит в себе набор чисел (в виде байтов). Справа отображается представление этих чисел в кодировке `ANSI`. Так как кодировка совпадает с той, в которой мы набирали эти символы, то текст выводится корректно. Но если мы отобразим эти же символы в другой кодировке, например `DOS Cyrillic II` (используется в командной строке Windows), мы увидим странный набор символов:



Так происходит потому, что этим же байтам в другой кодировке соответствуют другие символы. Обратите внимание, что слово `ANSI` выводится корректно, а текст, набранный в кириллице нечитаемый. Это связано с тем, что исторически большинство кодировок имеют одинаковые значения для символов латинского алфавита.

Объявление строк

Ранее мы уже объявляли строковые переменные обычным присвоением:

```
string greeting = "Hello there!";
```

Также мы можем создать строку из массива символов или массива байтов:

```
char[] chars = { 'w', 'o', 'r', 'd' };
byte[] bytes = { 0x41, 0x42, 0x43, 0x44, 0x45};

string string1 = new string(chars);
Console.WriteLine(string1); // word

string string2 = System.Text.Encoding.Default.GetString(bytes);

Console.WriteLine(string2); // ABCDE
```

Как мы упоминали ранее, для текстового представления байты необходимо сопоставить с той или иной кодировкой (таблицей символов), поэтому `string2` создается из массива байтов из кодировки по умолчанию (`Encoding.Default`) использованием функции `GetString`.

Если строка содержит escape-последовательности (переносы строк, спец.символы табуляции, обратную косую черту `\` и так далее), её значение должно начинаться с символа `@`, по аналогии с тем, как значения шаблонных строк начинается с символа `$`:

```
string text =
@"Это
многострочная
строка";
```

Итерация по символам строки

Так как строка — это набор символов, мы можем перебирать отдельные символы строки с помощью индексатора так же, как и в случае с массивами:

```
string greeting = "Hello, world!";
for (int i = 0; i < greeting.Length; i++)
{
    Console.WriteLine(greeting[i]); // построчный вывод каждого символа строки
}
```

Внимание! *Несмотря на сходства с массивами, нужно помнить, что строка — это отдельный тип данных. В отличие от массивов, строка не поддерживает изменение отдельных символов. Мы будем периодически затрагивать тему строковых переменных в дальнейших уроках и рассмотрим тип string подробнее.*

Как работает кодировка символов в байты и обратно

Ранее мы рассмотрели пример объявления строки из массива байт:

```
byte[] bytes = { 0x41, 0x42, 0x43, 0x44, 0x45};
string string2 = System.Text.Encoding.Default.GetString(bytes);
Console.WriteLine(string2); // ABCDE
```

Чтобы понять, почему байты `{ 0x41, 0x42, 0x43, 0x44, 0x45}` выводят именно `ABCDE`, нам нужно ознакомиться с системной кодировкой Windows. По умолчанию Windows использует кодировку ANSI. ANSI имеет множество вариаций для разных языков и регионов (в частности, для русскоязычной версии ОС используется кодовая страница 1251). Это означает, что при переводе массива байт в строку, в нашем приложении будет использоваться таблица символов для кодировки Windows-1251. Чтобы узнать кодировку, установленную в вашей системе, можно выполнить следующий код:

```
Console.WriteLine(System.Text.Encoding.Default.HeaderName);
```

Также можно запустить следующий код в терминале PowerShell (Пуск -> Выполнить -> powershell):

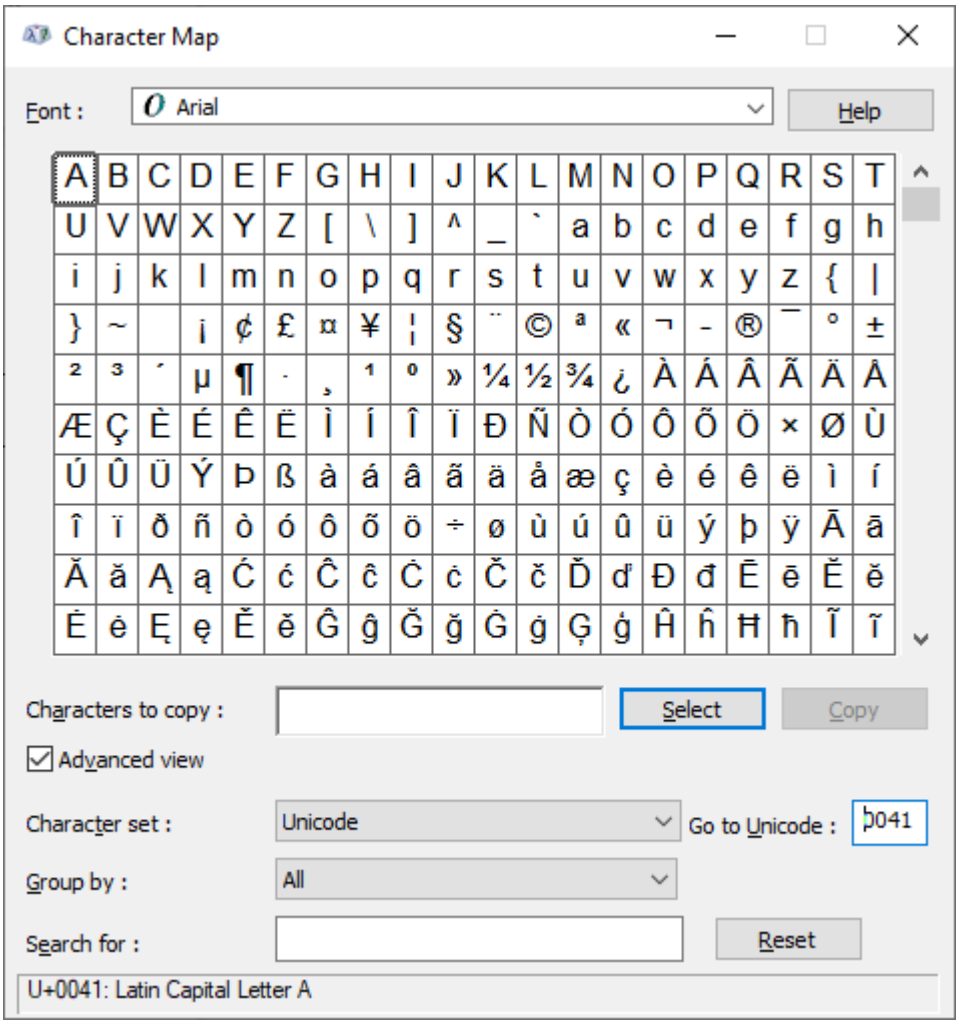
```
[System.Text.Encoding]::Default
```

```
PS C:\Users\Vaagn> [System.Text.Encoding]::Default


IsSingleByte      : True
BodyName          : koi8-r
EncodingName      : Cyrillic (Windows)
HeaderName        : windows-1251
WebName           : windows-1251
WindowsCodePage   : 1251
IsBrowserDisplay  : True
IsBrowserSave     : True
IsMailNewsDisplay : True
IsMailNewsSave    : True
EncoderFallback   : System.Text.InternalEncoderBestFitFallback
DecoderFallback   : System.Text.InternalDecoderBestFitFallback
IsReadOnly        : True
CodePage          : 1251
```

Если в вашей ОС установлена другая кодировка, не переживайте, это никак не повлияет на воспроизведение дальнейшего примера.

Чтобы узнать, какие символы соответствуют байтам с 41 по 45, воспользуемся системной утилитой Charmap. Для ее запуска в окне Пуск -> Выполнить введите `charmap`.

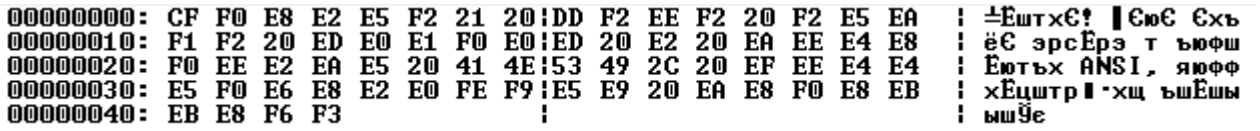


Окно утилиты содержит панель настройки отображения (внизу) и панель вывода символов выбранной кодировки. По умолчанию выводятся символы для кодировки `Unicode` (о ней мы поговорим немного позже). Выберем в выпадающем списке `Character set` кодировку `Windows: Cyrillic`. Выберем в панели вывода символов символ латинской буквы A. В строке состояния отобразится ее код — 41. Это значение нашего первого байта. Выберем следующий символ — B. Его код равен 42 и так далее. Получается, когда мы конвертируем массив байт в строку, класс Encoding сопоставляет каждый байт массива с кодировкой Default (кодировка по умолчанию) и в результате мы получаем строку ABCDE.

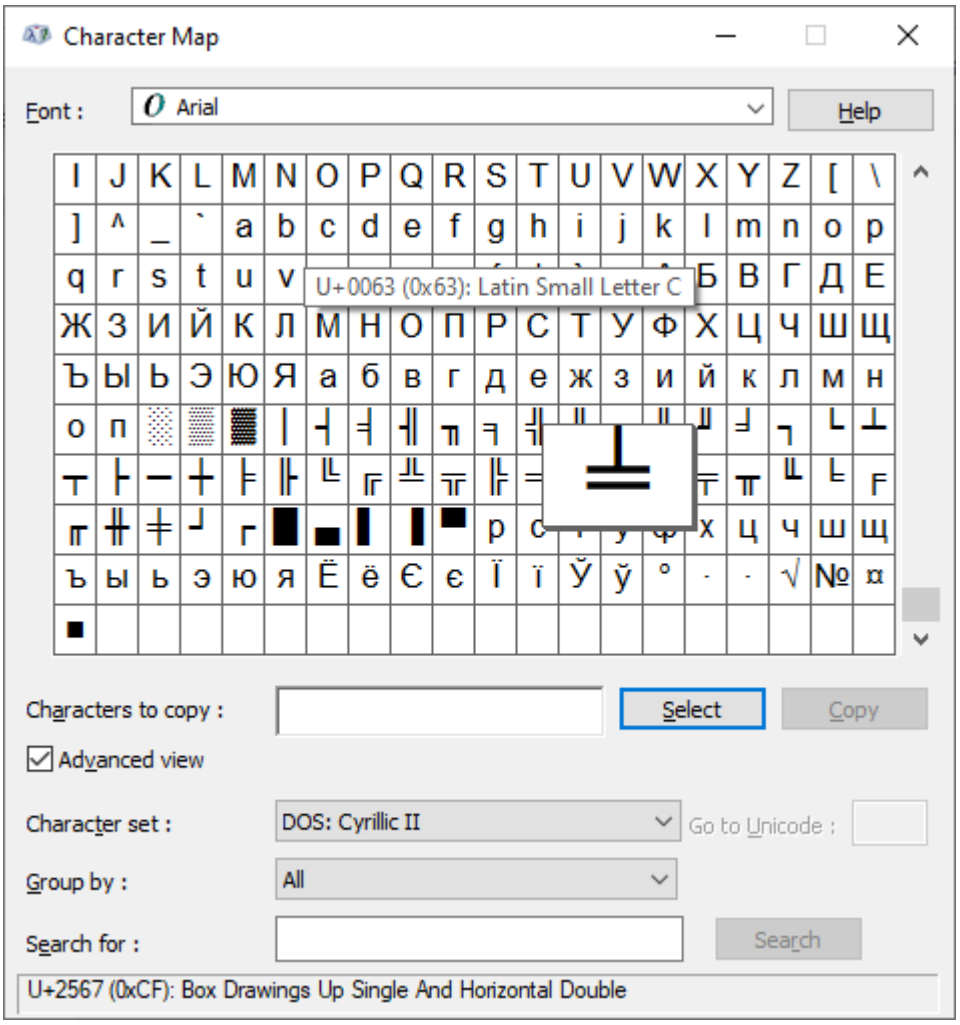
Чтобы закрепить навыки работы с утилитой charmap, давайте разберемся, почему отображение ранее используемого текста

```
Привет! Этот текст набран в кодировке ANSI, поддерживающей кириллицу
```

в кодировке `DOS Cyrillic II` было именно таким, каким мы его видели в двоичном редакторе:



Выберем в выпадающем списке `Character set` кодировку `DOS Cyrillic II`. Теперь мы видим символы, содержащиеся в этой кодировке. Найдем символ, имеющий байтовое представление равное CF (в кодировке 1251 означает букву П — первую букву нашей текстовой строки):



Следующий байт (F0) соответствует букве Ё и так далее.

Краткое знакомство с Unicode

Предыдущий раздел мог натолкнуть на мысль, что кодировки — это избыточное явление, ведь можно же было использовать одну кодировку и избежать множества проблем! На самом деле существует довольно большое число кодировок и это было обусловлено возможностями компьютеров и историческими моментами. В прошлом компьютеры были не такими производительными, как сейчас, и идея хранить в кодировке полный набор существующих в мире символов казался очень расточительным. Представьте времена, когда компьютеры ещё не были объединены в глобальную сеть. В таких условиях вероятность того, что компьютеру с англоязычным интерфейсом нужно будет отображать китайские иероглифы — достаточно мала, как и вероятность того, что компьютеру с китайским интерфейсом понадобится отображать кириллический текст. Ограниченный набор символов позволял создавать компактные таблицы символов. Так появилась таблица ASCII, которая предоставляла набор десятичных цифр, латинских букв и имела свободное пространство для национального алфавита.

В настоящее время компьютеры объединены в глобальную сеть Интернет. Люди со всех концов мира могут общаться между собой на разных языках, просматривать иностранные статьи или смотреть

иностранный фильм с субтитрами. Также неотъемлемой частью интернет-общения стали специальные красочные текстовые символы — Emoji. Все эти возможности привнес с собой стандарт кодирования символов Unicode (Юникод). Он содержит практически все существующие в мире национальные символы, а также символы нот, математические символы и многое другое. По аналогии с ранее рассматриваемыми кодировками, каждый символ в кодировке Юникод имеет свое байтовое представление. Каждый символ в Юникоде может занимать до 4 байт (в зависимости от версии Юникода). Это позволяет хранить огромные объемы символов. В настоящий момент стандарт насчитывает около 144 тысяч символов, а сам стандарт позволяет описать вплоть до 1.1 млн символов.

Благодаря Юникод работа с текстовыми файлами стала проще, так как чаще всего текстовые файлы кодируются в Юникод, однако на более ранних версиях Windows встроенный в систему текстовый редактор «Блокнот» не поддерживал Юникод. В актуальной версии Windows 10 этот редактор поддерживает работу со стандартом Юникод. Чаще всего за реализацию стандарта в текстовых редакторах отвечает формат UTF-8.

Практическое задание

1. Написать программу, выводящую элементы двухмерного массива по диагонали.
2. Написать программу — телефонный справочник — создать двумерный массив 5*2, хранящий список телефонных контактов: первый элемент хранит имя контакта, второй — номер телефона/e-mail.
3. Написать программу, выводящую введенную пользователем строку в обратном порядке (olleH вместо Hello).
4. * «Морской бой» — вывести на экран массив 10x10, состоящий из символов X и O, где X — элементы кораблей, а O — свободные клетки.

Используемые источники

1. Массивы | METANIT.
2. Тип char. Справочник по C.
3. Escape-символы в регулярных выражениях .NET.
4. Оператор for. Справочник по C.
5. Как работают кодировки текста. Откуда появляются «кракозябры». Принципы кодирования. Обобщение и детальный разбор.
6. Предпосылки создания и развитие Юникода.
7. Наборы - Таблица символов Юникода.