

# Разработка программ-конвертеров кодировок UTF16-UTF8

## Задание практикума №3 (3 семестр)

### **Введение**

Требуется разработать две программы-конвертера.

Первая программа читает из входного файла текст в кодировке UTF-16, переводит его в кодировку UTF-8 и выводит перекодированный текст в выходной файл. Вторая программа читает из входного файла текст в кодировке UTF-8, переводит его в кодировку UTF-16 и выводит перекодированный текст в выходной файл. Имена файлов задаются в командной строке. При отсутствии аргументов стандартный ввод рассматривается как входной файл, стандартный вывод как выходной.

Основные понятия и определения приведены ниже, после них более подробно описана постановка задачи.

### **Юникод и его кодировки.**

Ранние кодировки текста на компьютерах были однобайтовыми (или даже занимали меньше 8 бит на символ). В однобайтовых кодировках код каждого символа помещается в один байт. Проблема возникла тогда, когда требовалось **ОДНОВРЕМЕННО** представить текст на разных языках и в разных алфавитах. Причем эта проблема существовала не только для языков с большим набором символов алфавита (как в иероглифических языках типа китайского или японского), но и для индоевропейских языков с небольшим алфавитом (меньше сотни букв). Так, например, нет однобайтовой кодировки, в которой можно представить текст с символами английского, русского и французского языка:

Французское слово *café* переводится на русский как *кафе*

В 1991 году был разработан стандарт UNICODE (Юникод), который позже был синхронизирован с международным стандартом ISO-10646, хотя терминология у стандартов отличается. Программисты используют терминологию и понятия Юникода. Юникод состоит из двух основных частей: универсальный набор символов (universal character set-UCS) и набор кодировок (UCS Transformation Format - UTF), которые определяют способ представления символов на компьютере.

Универсальный набор символов определяет, во-первых, стандартное имя символа (например, CYRILLIC CAPITAL LETTER A – заглавная буква А кириллического алфавита или CYRILLIC SMALL LETTER ZHE – строчная буква Ж кириллического алфавита) и уникальный номер этого символа. Этот номер называется кодовой точкой Юникода и записывается в форме U+шестнадцатеричное значение кодовой точки. Кодовая точка символа CYRILLIC CAPITAL LETTER A - 0x410 (записывается так: U+0410), Кодовая точка CYRILLIC CAPITAL LETTER ZHE - 0x436 (записывается так: U+0436). Кодовые точки различных национальных алфавитов образуют (по возможности) диапазоны. Так базовый диапазон кириллицы, включающий русский алфавит, – U+0400 - ... - U+04FF. Для кириллицы есть дополнительные диапазоны для исторических вариантов алфавита, дополнительных букв в языках, использующих кириллицу и т. д. Кроме того Юникод определяет правила нормализации текста при неоднозначном представлении символа. Так упомянутый в тексте символ *é* может быть представлен кодовой точкой U+E9, так и парой символов U+65 U+301 – то есть парой из базового символа LATIN SMALL LETTER E и модифицирующим символом U+301 (верхний штрих). Есть и более сложные правила нормализации, которые не будем здесь рассматривать.

Примеры широко используемых диапазонов – U+00 ... U+1F – управляющие символы (CR, LF, BS, DEL и т.п.), U+20 ... U+7F – это символы из стандарта ASCII-7 (цифры, латинские

буквы, знаки препинания и т.д.), U+80 .... U+FF – диапазон символов ISO-Latin1 (западноевропейские языки).

Кодовые точки разбиты также на более широкие сегменты, чем национальные диапазоны. Эти сегменты называются плоскостями. Каждая плоскость определяет диапазон символов длиной 65536 ( $2^{16}$ ). Плоскость 0 представлена диапазоном U+00....U+FFFF, плоскость 1- диапазоном U+10000 + U+1FFFF, плоскость 2 - U+20000 + U+2FFFF. Всего в Юникоде 17 плоскостей – от 0 до 16. Плоскость 16 представлена диапазоном U+100000 ....U+10FFFF. Последний код в этой плоскости – максимальная кодовая точка в современном варианте Юникода. Плоскость 0 включает в себя национальные диапазоны для основных алфавитов мира, включая все европейские и большинство азиатских и африканских алфавитов. Ее называют BMP – Basic Multilingual Plane. Поскольку кодовые точки в ней могут быть представлены 2-байтным числом, то символы из BMP называют еще UCS-2 (Universal Character Set-2).

Для представления кодовых точек в компьютере служат кодировки с общим названием UTF. Самая простая – UTF-32. В ней каждая кодовая точка представляется 4-байтным беззнаковым целым (отсюда и название). Эта кодировка используется в GNU/Linux в трансляторах C/C++ для представления типа `wchar_t` (который и добавили в эти языки специально для представления юникодовских символов). Также UTF-32 используется в строках языка Питон, начиная с третьей версии.

Нас особо будут интересовать две другие кодировки – UTF-16 и UTF-8. В кодировке UTF-16 кодовые точки из BMP представляются одним 2-байтовым целым (то есть собственно значением кодовой точки, записанным в 16-битное беззнаковое значение), а символы из остальных плоскостей представляются парой 16-битных значений, которую называют суррогатной парой. Значения из суррогатной пары содержатся в так называемом приватном диапазоне U+D800....U+DFFF из BMP, который не включает в себя ни одного символа какого-либо алфавита. Конкретный алгоритм представления кодовых точек из ненулевых плоскостей Юникода нас здесь не будет интересовать. Таким образом, если юникодовские символы находятся в BMP (напомним, что этим свойством обладают все распространенные мировые алфавиты), то кодировка UTF-16 совпадает с UCS-2, поэтому ее иногда так и называют (кодировка UCS-2), что, конечно, не совсем верно.

Кодировка UTF-16 используется в ОС Windows, а также в языке Java для представления строк. Поэтому в компиляторах языка C/C++ в ОС Windows тип `wchar_t` эквивалентен `unsigned short` (16-битное беззнаковое целое). При использовании кодировки UTF-16 возникает неприятная проблема при переносе текста с одной машинной архитектуры на другую. Представление текста, очевидно, должно быть машинно-независимым. Файл с текстом (и не только с текстом, а вообще любой файл) со времен ОС UNIX – это последовательность байтов. Когда текст представляется последовательностью байтов, каждый из которых есть код символа, то эта последовательность не зависит от архитектуры. Но если символ представляется 2-байтовым значением, то возникает вопрос – в каком порядке эти байты записываются в файл (который есть последовательность байтов). Например, в файл записывается 2-байтовое значение 0x1234. Вопрос - какой байт запишется раньше – старший (0x12) или младший (0x34). Случай, когда вначале записывается младший байт, а затем старший, называется обратным порядком следования байтов (little-endian). Порядок, обратный «обратному», очевидно, называется прямым (big-endian). Проблема кодировки UTF-16 состоит в том, что при считывании файла с текстом невозможно понять, в каком порядке стоят байты кодированного символа – прямом или обратном. Для решения этой проблемы в Юникод ввели специальную кодовую точку, которой не соответствует никакой символ какого-либо алфавита. Эта кодовая точка называется BYTE ORDER MARK (сокращенно BOM) и имеет значение 0xFEFF (1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1). По соглашению программы, которые читают юникодовский текст в кодировке UTF-16, должны прочитать первые два байта как 16-битное беззнаковое значение. Это можно сделать, например, так:

```
unsigned short utf16ch;
```

```
fread(&utf16ch, sizeof utf16ch, 1, file);
```

и проанализировать его значение. Если оно равно 0xFEFF (то есть BOM), то текст записан в прямом порядке (big-endian) (vim называет такую кодировку utf-16). Если получилось значение 0xFFFE (в Юникоде добавили и такую кодовую точку – «обратный BOM»), то текст записан в обратном порядке байтов (little-endian) (vim называет такую кодировку utf-16le). Ну а если получилось другое значение (или fread вернул значение < sizeof utf16ch), то файл не соответствует соглашению, и читающая программа должна принять порядок по умолчанию. В случае нашего задания – это обратный порядок (utf-16le).

Кодировка UTF-8 В кодировке UTF-8 юникодовский символ (то есть кодовая точка) представляется последовательностью переменной длины (от 1 до 6). При этом символы в кодировке UTF-16 (а следовательно, и символы из BMP) представляются последовательностями от 1 до 3 байтов, в зависимости от величины символа (кодовой точки). Ниже приведена схема перевода из BMP (UTF-16) в UTF-8 и обратно. Цветом выделены разные байты в UTF-16 – оранжевый – старший байт, зеленый – младший.

Диапазон кодовых точек:

```
0...127 => 000000000xxxxxxx ⇔ 0xxxxxxx
128...2047 => 00000xxxxxyyyyyy ⇔ 110xxxxx 10yyyyyy
2048...65535 => xxxxyyyyyyzzzzzz ⇔ 1110xxxx 10yyyyyy 10zzzzzz
```

Таким образом, текст в кодировке ASCII-7 одновременно является и текстом в UTF-8! Например, программа на языке C, не использующая в строках, символьных константах и комментариях не-ASCII-7 символы, является закодированной в UTF-8 по умолчанию.

Пример перевода:

Русская буква А (U+0410):

Битовое представление:

```
0000010000010000 ⇔ 11010000 10010000
```

0x0410 ⇔ 0xD0 0x90

Другие примеры кодировки можно посмотреть в тестовых файлах для этого задания, используя утилиты xxd или od.

Одной из особенностей кодировки UTF-8 является ее «избыточность» (в отличие от UTF-16).

Не всякая последовательность байтов является валидной utf8-последовательностью. Например, она не может начинаться с байта 10xxxxxx, а также после байта 110xxxxxx не может следовать байт 0xxxxxxx. Есть и другие случаи некорректной последовательности байтов (на программистском жаргоне такой случай называют «битый юникод» или «битый utf8»). Читающая программа должна корректно обрабатывать эти случаи и возобновлять декодирование входной последовательности после «битого кода».

## ***Постановка задачи***

Требуется разработать две программы-конвертера.

Первая программа читает из входного файла текст в кодировке UTF-16, переводит его в кодировку UTF-8 и выводит перекодированный текст в выходной файл. Имя входного файла задается опцией `-i` имя\_файла, имя выходного файла задается опцией `-o` имя\_файла. При отсутствии нужного аргумента стандартный ввод рассматривается как входной файл, стандартный вывод как выходной. В случае неправильного имени файла программа должна выдавать сообщение об ошибке и завершаться.

Вторая программа читает из входного файла текст в кодировке UTF-8, переводит его в кодировку UTF-16 и выводит перекодированный текст в выходной файл. Имена файлов задаются в командной строке по тем же правилам, что и в первой программе.

Обе программы должны корректно обрабатывать маркер порядка байтов (byte order mark – BOM) – символ с кодом 0xFEFF — в начале файла. Первая программа должна читать текст в UTF-16, анализировать маркер в начале файла и обрабатывать файл в соответствии с порядком, заданным маркером. Перекодировать маркер в UTF-8 не надо, поскольку UTF-8 не зависит от порядка байтов во входном файле. В случае отсутствия маркера в начале файла порядок байтов зависит от наличия аргумента командной строки `-le` или `-be`. В первом случае принимается обратный порядок, во – втором – прямой. Оба аргумента не могут присутствовать одновременно (если не так, то программа выдает сообщение об ошибке и завершается). При отсутствии обоих аргументов `-le` и `-be` принимается обратный порядок. При наличии маркера в начале файла эти аргументы игнорируются.

Вторая программа должна генерировать текст в UTF-16 в соответствии со значением аргумента командной строки `-le` для обратного порядка или `-be` для прямого порядка. Оба аргумента не могут присутствовать одновременно (в противном случае должна быть зафиксирована ошибка). В случае отсутствия аргумента принимается LE-порядок (то есть по умолчанию BOM представлен байтами 0xFF 0xFE в начале файла). Маркер BOM всегда записывается программой-перекодировщиком в UTF-16. Если в начале UTF8-файла находится закодированный маркер BOM (некоторые программы так делают), то он игнорируется, а поведение программы определяется исключительно приведенными выше правилами.

Также программы должны обрабатывать и случаи некорректного представления входного текста — нечетное количество байтов в UTF-16, некорректные последовательности в UTF-8. В этих случаях программы должны выдавать в стандартный канал вывода сообщений об ошибках (`stderr`) диагностику, включающую в себя значение некорректного символа (последовательности), а также его смещение относительно начала файла. После этого программы должны «восстанавливаться» и продолжать чтение и перекодировку входной последовательности.

К заданию приложены тестовые файлы в обеих кодировках для контроля правильности заданий, а также для отладки программ.

## **Методические указания и требования к оформлению исходного кода программ**

Задание нужно реализовать на языке Си в операционной системе UNIX (LINUX).

Поскольку в постановке задачи требуется реализовать ДВЕ программы-перекодировщика, то необходимо разбить текст программ на модули, чтобы минимизировать зависимости внутри проектов. В частности, надо выделить функции, реализующие перекодировку на уровне массивов из `char` и `unsigned short`, в отдельный модуль (или даже два модуля), содержащий заголовочный файл (`.h`) с прототипами функций и, возможно, необходимых объявлений типов, и файл реализации (`.c`), содержащий определения нужных функций. Главные модули (содержащие функцию `main()`) должны использовать модуль с функциями перекодировки через включение соответствующего заголовочного файла. Для компиляции программ настоятельно рекомендуется разработать `make`-файл.

Для хранения UTF16-символов следует использовать тип данных `unsigned short`, а для UTF-8 символов - `char`.

Для ввода-вывода текста можно использовать библиотечные функции `getchar()` и `putchar()` для UTF-8 и `fread()` / `fwrite()` для UTF-16 (хотя последние функции можно использовать, при желании, везде). При тестировании и отладке программ должны использоваться данные из заранее подготовленных файлов. Эти файлы связываются со стандартными каналами ввода-вывода путем механизма перенаправления ввода-вывода, предоставляемого любой оболочкой в ОС UNIX или же задаются в командной строке:

```
utf2usc <le.ucs >le.utf
utf2usc -i le.ucs -o le.utf
```

Как читать файлы в разной кодировке?

Двоичный образ файла можно посмотреть программой `od` (octal dump). Команда `od le.utf`

покажет содержимое файла `le.utf` как бинарного файла из 16-битных чисел в восьмеричном формате.

Для побайтного вывода файла в 16-чном формате можно использовать команду (без объяснения опций – для этого смотрите `man`):

```
od -A x -t x1z -v le.utf
```

Полезной может быть и такое использование утилиты `od`, когда она показывает файл посимвольно через пробел, но если символ не входит в набор отображаемых символов ASCII-7 (коды 31-127), то выводится восьмеричный код символа с префиксом `\` (обратная косая):

```
od -c le.utf
```

Возможно, что более удобной для просмотра двоичного файла окажется утилита `xxd`. Например, команда

```
xxd -b file
```

показывает содержимое файла `file` в двоичном виде, а по умолчанию:

```
xxd file
```

в шестнадцатеричном виде.

Если в системе установлена кодировка UTF-8 (что верно для многих инсталляций Linux или MacOS), то файл в UTF-8 кодировке можно просто выдать на экран командой `cat имя_файла` или же просмотреть в любом текстовом редакторе.

Для того, чтобы просмотреть содержимое файла в любой кодировке, можно использовать редактор `vim`. Открываете файл командой `vim имя_файла` и указываете кодировку файла командой:

```
:set fileencoding=ucs-2
```

Заметьте, что кодировка `ucs-2` использует прямой порядок байтов (`big-endian`). При

чтении файла с обратным порядком байтов (little-endian) нужно использовать имя `ucs-2le`. Вместо `ucs-2` и `ucs-2le` можно использовать имена `utf-16` и `utf-16le`. Различия в этих кодировках не проявляются в тестовых файлах.

При тестировании и отладке программ рекомендуется проверить ваши программы на прямом-обратном перекодировании. Например, файл `text.utf` можно перекодировать в файл `t1.ucs`, а потом `t1.ucs` перекодировать обратно в `utf-8`, например, в файл `t1.utf`. Файлы `t1.ucs` и `t1.utf` должны полностью совпасть. Проверить совпадение можно с помощью утилит `diff` или `vimdiff`. Заметим, что утилита `vimdiff`, в отличие от `diff`, читает текст в кодировке UTF-16 с учетом ВОМ-маркера, поэтому она считает (вполне резонно), что файлы `letext.ucs` и `betext.ucs` — одинаковы.

## Тестовые файлы

Для тестирования и отладки предлагается использовать следующий (минимальный) набор файлов, который находится в каталоге tests и является частью архива, содержащего текст задания.

### UTF-16 файлы

- letext.ucs – текст в UTF-16 в перевернутом представлении (LE-порядок) с меткой BOM
- betext.ucs – текст в UTF-16 в прямом представлении (BE-порядок) с меткой BOM
- letextbad1.ucs – текст в UTF-16 в перевернутом представлении (LE-порядок) без метки BOM
- betextbad1.ucs – текст в UTF-16 в прямом представлении (BE-порядок) без метки BOM
- letextbad2.ucs – текст в UTF-16 в перевернутом представлении (LE-порядок) с меткой BOM, но с неверным символом (однобайтовым)
- betextbad2.ucs – текст в UTF-16 в прямом представлении (BE-порядок) с меткой BOM, но с неверным символом (однобайтовым)
- leempty.ucs – пустой текст в UTF-16 в перевернутом представлении (LE-порядок) с меткой BOM
- beempty.ucs – пустой текст в UTF-16 в прямом представлении (BE-порядок) с меткой BOM
- le30.ucs – односимвольный (код=0x30 – символ 0) текст в UTF-16 в перевернутом представлении (LE-порядок) с меткой BOM
- be30.ucs – односимвольный (код=0x30 – символ 0) текст в UTF-16 в прямом представлении (BE-порядок) с меткой BOM
- le42f.ucs – односимвольный (код=0x042F – символ Я) текст в UTF-16 в перевернутом представлении (LE-порядок) с меткой BOM
- be42f.ucs – односимвольный (код=0x042F – символ Я) текст в UTF-16 в прямом представлении (BE-порядок) с меткой BOM
- le263A.ucs – односимвольный (код=0x263A – символ ☺) текст в UTF-16 в перевернутом представлении (LE-порядок) с меткой BOM
- be262A.ucs – односимвольный (код=0x263A – символ ☺) текст в UTF-16 в прямом представлении (BE-порядок) с меткой BOM

### UTF-8 файлы

- text.utf – текст в UTF-8 с меткой BOM (кодированной)
- text2.utf – текст в UTF-8 без метки BOM
- textbad1.utf – текст в UTF-8 с неверной последовательностью (начинается с байта продолжения) без метки BOM
- textbad2.utf – текст в UTF-8 с неверной последовательностью (отсутствует байт продолжения) без метки BOM
- empty.utf – пустой текст в UTF-8 с меткой BOM
- 30.utf – односимвольный (код=0x30 – символ 0) текст в UTF-8
- 42f.utf – односимвольный (код=0x042F – символ Я) текст в UTF-8
- 263A.utf – односимвольный (код=0x263A – символ ☺) текст в UTF-8