Задача 6. Кодировка UTF-8

Источник:	основная
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	разумное

Сегодня повсеместно используется представление символов различных языков при помощи таблицы Unicode. Самая популярная кодировка (т.е. способ сохранения в файл) для Unicode символов — это UTF-8. В данной задаче вам нужно прочитать из входного файла текст в кодировке UTF-8, преобразовать его в последовательность кодов символов, и сохранить все эти коды как 32-битовые целые числа в выходной файл (т.е. по сути записать в кодировке UTF-32LE).

Каждому символу в таблице Unicode назначен код — какое-то неотрицательное целое число. Коды символов таблицы Unicode покрывают весь диапазон от `0x0000` (нуля) до `0x10FFFF` включительно, за исключением диапазона от `0xD800` до `0xDFFF`, в котором расположены так называемые суррогаты, не соответствующие никаким символам. Таким образом, всего в таблице Unicode 1112064 (`0x10F800`) символов.

Кодировку UTF-8 можно использовать для представления произвольной последовательности беззнаковых целых чисел (не более 21 бита каждое) в виде последовательности байтов. Когда в UTF-8 кодируют текст Unicode, то в качестве этих беззнаковых чисел выступают коды символов текста. Чтобы закодировать последовательность значений в UTF-8, нужно представить каждое значение в виде последовательности от одного до четырёх байтов. Далее нужно записать все полученные последовательности байтов подряд друг за другом в порядке записи значений в исходной последовательности. Каждое отдельное значение кодируется согласно таблице:

длина	шаблон	битов
1	`0xxxxxxx`	7
2	`110xxxxx 10xxxxxx`	11
3	`1110xxxx 10xxxxxx 10xxxxxx`	16
4	`11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`	21

Здесь в первом столбце записано количество байтов в представлении, во втором столбце — шаблон, а в третьем — количество букв ‘x’ в шаблоне. В шаблоне записаны в двоичном виде байты, которые получаются при его использовании для представления значения: цифры ‘1’ и ‘0’ обозначают фиксированные значения битов, а буква ‘x’ обозначает бит, состояние которого определяется по кодируемому числу.

Чтобы закодировать целое число V в виде последовательности из k байтов, нужно:

1. Записать число V в двоичной системе исчисления.
2. Найти в таблице k-ую строку и зафиксировать её для дальнейших шагов.
3. Если требуется, дополнить V ведущими нулями так, чтобы количество битов в V совпало с числом в последнем столбце таблицы.
4. В буквы ‘x’ шаблона вставить биты числа V. Биты вставляются слева направо, в порядке от старшего бита к младшему.
5. В результате вместо шаблона получилось k байтов, по восемь бит в каждом из них — результат кодирования V.

Заметим, что если число V в двоичном виде имеет больше битов, чем букв ‘x’ в шаблоне длины k, то закодировать его в k байтов нельзя. С другой стороны, для конкретного значения V вполне можно выбрать количество байтов k несколькими способами. Стандарт UTF-8 предписывает выбирать **минимально** возможное k. Если число закодировано в неминимальное количество байтов, то это называется “overlong encoding”, и считается ошибкой.

Например, буква ‘с’ русского алфавита имеет код 1089 в таблице Unicode, что в двоичном виде выглядит как 10001000001. В UTF-8 это число можно записать тремя способами, при этом только первый способ правильный, а все остальные ошибочные:

- `110`10001 `10`000001
- `1110`0000 `10`010001 `10`000001
- `11110`000 `10`000000 `10`010001 `10`000001

Во входном файле записан текст в формате UTF-8, возможно с ошибками, общим размером не более мегабайта. Если ошибок нет, то ваша программа должна извлечь коды записанных символов и записать их в выходной файл как 4-байтовые целые числа с little-endian порядком байтов. Если в файле есть ошибки, то они должны быть обработаны строго определённым образом.

Ваша программа должна декодировать символы в том порядке, в котором они записаны в файле. Для каждого символа следует сперва обращать внимание на первый байт и определять по нему количество байтов в представлении символа. Далее нужно считывать остальные байты символа (они называются байтами продолжения) и получать код как целое число. Наконец, нужно проверять, что этот код есть в таблице Unicode и что не имеет место “overlong encoding”.

Есть четыре вида жёстких ошибок:

1. Встретился байт, у которого пять или больше старших битов единичные.
2. Нужно прочитать байт продолжения, чтобы закончить символ, а в файле нет больше байтов.
3. Нужно прочитать байт продолжения, чтобы закончить символ, а следующий байт не имеет формат `10xxxxxx`.
4. Первый байт очередного символа имеет формат `10xxxxxx`.

При обнаружении жёсткой ошибки программа должна сразу завершаться, не читая остальное содержимое файла. При этом все предыдущие полностью прочитанные символы должны быть в выходном файле.

Кроме того, есть три вида мягких ошибок:

1. Записанный код символа больше `0x10FFFF`, а значит не попадает в таблицу Unicode.
2. Записанный код символа попадает в диапазон от `0xD800` до `0xDFFF`, то есть является суррогатом.
3. Записанный код символа можно было закодировать меньшим количеством байтов k, то есть он закодирован в виде “overlong encoding”.

При обнаружении ошибки код символа следует заменить на `0xFFFD` (так называемый “replacement char”), выдать его в выходной файл как обычно и продолжить работу над оставшейся частью файла.

Пример

На следующей странице приведены примеры в шестнадцатеричном виде. Скачать их в бинарном виде можно по ссылке. Рекомендуется также смотреть на примеры, записывая байты в двоичном представлении.

input.txt

```
78 E2 89 A4 28 CE B1 2B CE B2 29 C2 B2 CE B3 C2 B2
```

output.txt

```
78 00 00 00 64 22 00 00 28 00 00 00 B1 03 00 00 2B 00 00 00
B2 03 00 00 29 00 00 00 B2 00 00 00 B3 03 00 00 B2 00 00 00
```

input.txt

```
78 F7 89 A4 80 28 CE B1 2B CE B2 29 C2 B2 CE B3 C2
```

output.txt

```
78 00 00 00 FD FF 00 00 28 00 00 00 B1 03 00 00 2B 00 00 00
B2 03 00 00 29 00 00 00 B2 00 00 00 B3 03 00 00
```

input.txt

```
63 6F 6F 6C 20 6D 61 74 68 3A C0 A0 78 E2 89 A4 28 CE B1 2B CE B2 29
C2 B2 CE B3 C2 20 78 79
```

output.txt

```
63 00 00 00 6F 00 00 00 6F 00 00 00 6C 00 00 00 20 00 00 00
6D 00 00 00 61 00 00 00 74 00 00 00 68 00 00 00 3A 00 00 00
FD FF 00 00 78 00 00 00 64 22 00 00 28 00 00 00 B1 03 00 00
2B 00 00 00 B2 03 00 00 29 00 00 00 B2 00 00 00 B3 03 00 00
```

input.txt

```
F0 9F 98 82 78 ED A0 90 28 ED B4 80 CE B2 29 FE C2 B2 CE B3 C2 B2
```

output.txt

```
02 F6 01 00 78 00 00 00 FD FF 00 00 28 00 00 00 FD FF 00 00
B2 03 00 00 29 00 00 00
```

input.txt

```
D1 82 D0 B5 D1 81 D1 82 20 85 D1 0D 0A
```

output.txt

```
42 04 00 00 35 04 00 00 41 04 00 00 42 04 00 00 20 00 00 00
```

Пояснение к примеру

В первом тесте записан полностью правильный текст из десяти символов. В остальных трёх тестах есть как мягкие, так и жёсткие ошибки.

На втором тесте второй символ имеет код больше `0x10FFFF` (мягкая ошибка), так что он заменяется на replacement char. В конце файла записан байт `0xC2`, который является началом двухбайтового кода, однако байтов продолжения нет (файл закончился). Это жёсткая ошибка. При этом все символы, прочитанные до этого, сохранены в файл.

На третьем тесте сначала идёт много однобайтовых символов. В середине стоит символ пробела (код 32), записанный двумя байтами. Это overlong encoding, так как можно закодировать его одним байтом `0x20` — заменяется на replacement char (мягкая ошибка). Кроме того, почти в самом конце символ начинается байтом `0xC2`, подразумевающим двухбайтовый код, но следующий байт `0x20` не является байтом продолжения — это жёсткая ошибка.

В четвёртом тесте третий и пятый символы имеют коды `0xD810` и `0xDD00` соответственно, которые являются суррогатами (мягкие ошибки). Ближе к концу обнаруживается байт `0xFE`, который является жёсткой ошибкой, ибо в UTF-8 байт может иметь максимум четыре старших единичных бита. В пятом тесте после пяти символов внезапно стоит байт продолжения `0x85`, хотя на этом месте должен быть первый байт символа — жёсткая ошибка.