UTF8 strings and characters/ru

From Free Pascal wiki

Jump to navigationJump to search

| English (en) | suomi (fi) | **русский (ru)** |

Красота UTF-8

Байты, начинающиеся с '0' (0xxxxxxx), зарезервированы для ASCII-совместимых однобайтовых символов. В многобайтовых кодовых точках число 1 в старшем байте определяет количество байтов, которое занимает кодовая точка. Наподобие этого:

- 1 byte : 0xxxxxxx
- 2 bytes : 110xxxxx 10xxxxxx
- 3 bytes : 1110xxxx 10xxxxxx 10xxxxxx
- 4 bytes : 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Конструкция UTF-8 имеет некоторые преимущества по сравнению с другими кодировками:

- Он обратно совместим с ASCII и создает компактные данные для западных языков. ASCII также используется в тегах языка разметки и других метаданных, что дает UTF-8 преимущество с любым языком. Однако обратная совместимость не распространяется на код, так как код должен быть пересмотрен, чтобы избежать искажения строк utf8.
- Целостность многобайтовых данных может быть проверена по количеству '1'-битов в начале каждого байта.
- Вы всегда можете найти начало многобайтовой кодовой точки, даже если вы перепрыгнули на случайную позицию байта.
- Байт в определенной позиции в многобайтовой последовательности никогда не может быть перепутан с другими байтами. Это позволяет использовать старые быстрые строковые функции, такие как `Pos()` и `Copy()` во многих ситуациях. Смотрите примеры ниже.

Примечание: аналогичные функции целостности также существуют в UTF-16. (Диапазон D800 сигнализирует первый суррогат, сигнал диапазона DC00 - второй суррогат).

- Надежный код. Код, который имеет дело с кодовыми точками, всегда должен быть выполнен правильно с UTF-8, потому что многобайтовые кодовые точки являются общими. Для UTF-16 есть много небрежного кода, который предполагает, что кодовые точки имеют фиксированную ширину.

- Большая часть текстовых данных, перемещаемых в Интернете, кодируется как UTF-8. Обработка данных непосредственно как UTF-8 исключает бесполезные преобразования. Многие (связанные с Unix) операционные системы изначально используют UTF-8.

Примеры

Простой перебор символов, как если бы строка представляла собой массив элементов одинакового размера, не работает с Юникодом. Это не что-то конкретное для UTF-8, стандарт Unicode сложен, а слово «символ» неоднозначно. Если вы хотите перебрать все кодовые точки строки UTF-8, есть два основных способа:

- перебирать все байты - полезно для поиска подстроки или при просмотре только символов ASCII в строке UTF8, например, при разборе файлов XML.
- перебирать все кодовые точки или символы - полезно для графических компонентов, таких как SynEdit, например, когда вы хотите узнать третий напечатанный символ на экране.

Поиск подстроки

Из-за особой природы UTF8 вы можете просто использовать обычные строковые функции для поиска в подстроке. Поиск правильной строки UTF-8 с помощью Pos всегда будет возвращать правильную позицию байта UTF-8:

```
uses LazUTF8;
...
procedure Where(SearchFor, aText: string);
var
  BytePos: LongInt;
  CodepointPos: LongInt;
begin
  BytePos:=Pos(SearchFor,aText);
  CodepointPos:=UTF8Pos(SearchFor,aText);
  writeln('Подстрока "',SearchFor,'" находится в тексте "',aText,'"',
    ' в позиция байта ',BytePos,' и в позиция кодовой точки ',CodepointPos);
end;
```

Поиск и копирование

Еще один пример того, как `Pos()`, `Copy()` и `Length()` работают с UTF-8. Эта функция не имеет кода для работы с кодировкой UTF-8, но она всегда работает с любым действительным текстом UTF-8.

```
function SplitInHalf(Txt, Separator: string; out Half1, Half2: string):
Boolean;
var
  i: Integer;
begin
```

```
  i := Pos(Separator, Txt);
  Result := i > 0;
  if Result then
  begin
    Half1 := Copy(Txt, 1, i-1);
    Half2 := Copy(Txt, i+Length(Separator), Length(Txt));
  end;
end;
```

Перебор строки в поисках символов ASCII

Если вы хотите найти только символы в ASCII-области, вы можете использовать тип Char и сравнивать с Txt[i], как в старые времена. Большинство парсеров делают это, и они продолжают работать.

```
procedure ParseAscii(Txt: string);
var
  i: Integer;
begin
  for i:=1 to Length(Txt) do
    case Txt[i] of
      '(': PushOpenBracketPos(i);
      ')': HandleBracketText(i);
    end;
end;
```

Итерация по строке в поисках символов Юникода или текста

Если вы хотите найти все вхождения определенного символа или подстроки в строку, вы можете повторно вызывать PosEx().

Если вы хотите проверить другой текст внутри цикла, вы все равно можете использовать быстрые функции Copy() и Length(). Можно использовать специальные функции UTF-8, но они не нужны.

```
procedure ParseUnicode(Txt: string);
var
  Ch1, Ch2, Ch3: String;
  i: Integer;
begin
  Ch1 := 'Й';  // Символы для поиска. Они также могут
  Ch2 := 'ﷺ';  // быть комбинированными кодовыми точками или более длинным
текстом
  Ch3 := 'Å';
  for i:=1 to Length(Txt) do
  begin
    if Copy(Txt, i, Length(Ch1)) = Ch1 then
```

```
      DoCh1(...)
    else if Copy(Txt, i, Length(Ch2)) = Ch2 then
      DoCh2(...)
    else if Copy(Txt, i, Length(Ch3)) = Ch3 then
      DoCh3(...)
  end;
end;
```

Цикл можно оптимизировать, перескакивая через уже обработанные части.

Перебор строки с анализом отдельных кодовых точек

Этот код копирует каждую кодовую точку в переменную типа String, которая затем может быть обработана далее.

```
procedure IterateUTF8(S: String);
var
  CurP, EndP: PChar;
  Len: Integer;
  ACodePoint: String;
begin
  CurP := PChar(S);        // если S='', то PChar(S) возвращает указатель на
#0
  EndP := CurP + length(S);
  while CurP < EndP do
  begin
    Len := UTF8CodepointSize(CurP);
    SetLength(ACodePoint, Len);
    Move(CurP^, ACodePoint[1], Len);
    // Единственная кодовая точка копируется из строки. Делайте так.
    ShowMessageFmt('CodePoint=%s, Len=%d', [ACodePoint, Len]);
    // ...
    inc(CurP, Len);
  end;
end;
```

Доступ к байтам внутри одной кодовой точки UTF8

Кодовые точки в кодировке UTF-8 могут различаться по длине, поэтому лучшим решением для доступа к ним является использование итерации. Для перебора кодовых точек используйте этот код:

```
uses LazUTF8;
...
procedure DoSomethingWithString(AnUTF8String: string);
var
```

```
  p: PChar;
  CPLen: integer;
  FirstByte, SecondByte, ThirdByte, FourthByte: Char;
begin
  p:=PChar(AnUTF8String);
  repeat
    CPLen := UTF8CodepointSize(p);

    // Здесь у вас есть указатель на символ и его длину
    // Вы можете получить доступ к байтам UTF-8 Char следующим образом:
    if CPLen >= 1 then FirstByte := P[0];
    if CPLen >= 2 then SecondByte := P[1];
    if CPLen >= 3 then ThirdByte := P[2];
    if CPLen = 4 then FourthByte := P[3];

    inc(p,CPLen);
  until (CPLen=0) or (p^ = #0);
end;
```

Доступ к N-й кодовой точке UTF8

Помимо итерации можно также иметь произвольный доступ к кодовым точкам UTF-8.

```
uses LazUTF8;
...
var
  AnUTF8String, NthCodepoint: string;
begin
  NthCodepoint := UTF8Copy(AnUTF8String, N, 1);
```

Отображение кодовых точек с UTF8CharacterToUnicode

Ниже показано, как отобразить 32-битное значение кодовой точки каждой кодовой точки в строке UTF8:

```
uses LazUTF8;
...
procedure IterateUTF8Codepoints(const AnUTF8String: string);
var
  p: PChar;
  unicode: Cardinal;
  CPLen: integer;
begin
  p:=PChar(AnUTF8String);
  CPLen := UTF8CodepointSize(p);
  repeat
```

```
    unicode:=UTF8CodepointToUnicode(p,CPLen);
    writeln('Unicode=',unicode);
    inc(p,CPLen);
  until (CPLen=0) or (unicode=0);
end;
```

Составные символы

Из-за неоднозначности Unicode функции сравнения и Pos() могут показывать неожиданное поведение, например, когда одна строка содержит составные символы, а другая использует прямые коды для той же буквы. RTL автоматически это не обрабатывает. Это характерно не для какой-либо кодировки, а для Unicode в целом.

macOS

Файловые функции модуля FileUtil также заботятся о специфическом поведении macOS: macOS нормализует имена файлов. Например, имя файла 'ä.txt' может быть закодировано в Unicode двумя различными последовательностями (#$C3#$A4 и 'a'#$CC#$88). Под Linux и BSD вы можете создать имя файла с обеими кодировками. macOS автоматически преобразует умляут в трехбайтовую последовательность. Это означает:

```
if Filename1 = Filename2 then ... // недостаточно под macOS
if AnsiCompareFileName(Filename1, Filename2) = 0 then ... // недостаточно для fpc 2.2.2, даже для cwstring
if CompareFileNames(Filename1, Filename2) = 0 then ... // это работает всегда (модуль FileUtil или FileProcs)
```

См. также

- Типы символов и строк