

CMDK

Copyright © 2016-2017 by Marat Sungatullin

В этом документе даются сведения, необходимые для программирования игр на движке “CMDK”.

## Простые типы

Для хранения кодовых позиций Юникода предусмотрены два типа в пространстве имен lang: Char – 16 битное целое без знака и Char32 – 32-х битное целое без знака.

## Утилиты

### UTF-конвертер

Как известно символы, представленные в Юникоде, называют кодовыми позициями или «кодами» Юникода. Назначением UTF-конвертера является кодирование кодов Юникода и ACII-7 в последовательность байт и раскодирование последовательности байт в коды Юникода и в ACII-7. UTF-конвертер поддерживает кодировки ASCII-7, UTF-8, UTF-16, UTF-32. UTF-конвертер представлен классом UTFConverter в пространстве имен lang.

Для использования UTF-конвертера нужно создать объект конвертера, передав ему C-строку с названием кодировки. Вот допустимые строки:

"ASCII-7"

"UTF-8"

"UTF-16BE"

"UTF-16LE"

Аббревиатуры “BE” и “LE” означают порядок байт: big-endian и little-endian соответственно. В интерфейсе конструктора класса в данное время отсутствует возможность задания кодировки UTF-32, хотя код обработки имеется – видимо, разработчики из Catmother LTD его не дописали в силу каких-то причин.

Для преобразования кодовой позиции в последовательность байт имеется метод класса UTFConverter::encode. При его использовании нужно передать следующие параметры в порядке их следования:

- указатель на буфер, в который будет сохраняться последовательность байт;

- указатель на память за последним байтом буфера, в который будет сохраняться последовательность байт;

- указатель на буфер, в который будет сохраняться число записанных байт;

- кодовая позиция.

Для обратного преобразования байт в кодовую позицию нужно использовать метод decode. При его использовании нужно передать следующие параметры в порядке их следования:

- указатель на буфер, из которого будет читаться последовательность байт;

- указатель на память за последним байтом буфера, из которого будет читаться последовательность байт;

- указатель на буфер, в который будет сохраняться число прочитанных байт;

- указатель на буфер, в котором будет сохраняться кодовая позиция.

Методы encode и decode автоматически проверяют размер переданного в параметрах буфера байт, возвращают true при удачном преобразовании и false – при неудачном.

Ниже приведен пример кодирования и раскодирования с применением этого класса:

using namespace lang;  
UTFConverter uconv(“UTF-16BE”);  
unsigned char bytes[4];  
int nBytes;  
Char32 cp = 0x10001;  
uconv.encode(bytes, bytes+4, &nBytes, cp);  
uconv.decode(bytes, bytes+4, &nBytes, &cp);

## Структуры данных

### Строки

В движке вводятся типы Char (16-битное беззнаковое целое) и Char32 (32-битное беззнаковое целое) в пространстве имен lang для представления символов юникода. Класс String, определенный в пространстве имен lang, служит для описания строк с поддержкой юникода и использует вышеописанные типы для представления символов. По структуре класс String содержит указатель на буфер символов типа Char. Класс строк имеет внутренний массив, рассчитанный примерно на 20 символов юникода.

Конструктор по умолчанию распределяет пространство памяти внутреннего буфера под 29 символов типа Char. В начале внутреннего буфера располагается управляющая структура с описанием внутреннего буфера, остальная часть буфера используется для хранения типов Char. Внутренний указатель m\_this указывает на смещение внутри внутреннего буфера, с которого начинаются хранение символов – то есть сразу за управляющей структурой. В первый символ буфера записывается 0.

Класс строки распознает следующие кодировки:

Таблица . Кодировки класса String

|  |  |
| --- | --- |
| Строка с описанием | Наименование |
| ASCII-7 | Стандартная кодировка ANSI c 7 значащами битами |
| UTF-8 | Юникод (на символ от 1до 6 байт) |
| UTF-16BE | Юникод (на символ от 2 до 4 байт) Big Endian |
| UTF-16LE | Юникод (на символ от 2 до 4 байт) Little Endian |
| UTF-32BE | Юникод (на символ 4 байта) Big Endian |
| UTF-32LE | Юникод (на символ 4 байта) Little Endian |

Внутренне строка хранится в формате UTF-16.

Существует несколько конструкторов класса строки.

Как уже было описано конструктор по умолчанию просто форматирует внутренний буфер, в котором можно сохранить примерно 29 символов (зависит от кодовых позиций, имеющих как известно длину от 2 до 4 байт).

В конструктор можно передать завершающуюся нулем строку Char или char типа, а можно передать указатель на буфер типа Char или типа char с указанием его длины. Например:

String s1(“Hello, world!”);

char buf[32];

strcpy(buf, “Hello, world!”);

String s2(buf, strlen(“Hello”));

В конструктор можно передавать одиночный символ типа Char.

В конструктор можно передать указатель на буфер с данными, число байт в этом буфере и строку с описанием кодировки из таблицы 1 (пример):

void example\_ctors01(){  
 String s("Френк едет в Голливуд",   
 strlen("Френк едет в Голливуд"), // число символов  
 "UTF-8"); // В редакторе кода нужно задать кодировку UTF-8  
 Char buf[256];  
 s.getChars(0, s.length(), buf);  
 buf[s.length()] = 0;  
 char bytes[256];  
 // CP\_OEMCP – кодовая страница в командной строке  
 WideCharToMultiByte(CP\_OEMCP,   
 0,   
 (wchar\_t\*)&buf[0]/\*строка хранится во внутреннем буфере как UTF-16\*/,  
 -1,  
 bytes,   
 sizeof(bytes),  
 0,  
 0);  
 printf("bytes: %s\n", bytes);  
}

Существует конструктор копирования и оператор присваивания. Если память для строки выделена в куче, а не во внутреннем буфере, что бывает при хранении длинных строк, происходит только копирование внутреннего указателя.

Метод length() возвращает длину строки (завершающий нуль как и у strlen() не учитывается).

Метод charAt() возвращает символ, лежащий по заданному индексу в строке. Индекс проверяется ассертом.

С помощью метода getBytes можно закодировать символы из строки во внешний буфер, заданный указателем и размером в байтах, заданной кодировкой. Метод возвращает необходимый размер буфера в байтах.

Метод getChars() копирует (end – begin) символов типа Char во внешний буфер, начиная с позиции begin. Никаких дополнений внешнего буфера завершающим нулем не производится.

Метод endsWith() проверяет, оканчивается ли строка заданной подстрокой-суффиксом, а метод startsWith() проверяет, начинается ли строка заданной подстрокой-префиксом. Возвращаются булевы значения.

#### Поиск

Поддерживаются следующие возможности поиска для строк:

* поиск отдельных символов и подстрок;
* поиск в прямом и обратном направлениях;
* поиск, начиная с произвольной позиции (в начале или внутри строки).

Для поисковых функций используется следующая схема передачи аргументов:

* в первом аргументе всегда передается искомое значение;
* второй аргумент определяет индекс, с которого должен начинаться поиск.

Вот аргументы перегруженных поисковых методов.

Char ch

Поиск символа ch.

Char ch, int index

Поиск символа ch, начиная с позиции index.

const String& str

Поиск подстроки str.

const String& str, int index

Поиск подстроки str, начиная с позиции index.

Первая форма метода indexOf() возвращает индекс заданного символа или -1, если такой символ не будет найден. Поиск начинается с позиции, заданной в параметре index, и ведется в направлении от начала строки к ее концу. Если индекс не задан, поиск ведется, начиная с самой первой позици строки.

Вторая форма метода indexOf() возвращает индекс первого символа заданной подстроки или -1, если такая подстрока не будет найдена. Поиск начинается с позиции, заданной в параметре index, и ведется в направлении от начала строки к ее концу. Если индекс не задан, поиск ведется, начиная с самой первой позици строки.

Первая форма метода lastIndexOf () возвращает индекс заданного символа или -1, если такой символ не будет найден. Поиск начинается с позиции, заданной в параметре index, и ведется в направлении от конца строки к началу. Если индекс не задан, поиск ведется, начиная с самой последней позици строки.

Вторая форма метода lastIndexOf () возвращает индекс первого символа заданной подстроки или -1, если такая подстрока не будет найдена. Поиск начинается с позиции, заданной в параметре index, и ведется в направлении от конца строки к началу. Если индекс не задан, поиск ведется, начиная с самой последней позици строки.

Пример:

lang::String s1("Cat lies on the mat in some flat");

s1.indexOf("at"); // Возвращает 1 (первая подстрока “at”)

s1.indexOf("at", 2); // Возвращает 17 (первая подстрока “at” после s[2])

s1.lastIndexOf("at");// Возвращает 30 (последняя подстрока “at”)

s1.lastIndexOf("at", 29);// Возвращает 17 (первая подстрока “at” перед s[29])

Метод regionMatches() проверяет совпадение подстрок заданной длины этой и другой строки. Под строки задаются смещениями.

Метод replace() заменяет все вхождения символа, заданного в первом параметре, символом, заданном во втором параметре в копии строки. Возвращает модифицированную копию строки.

Метод substring() возвращает подстроку из диапазона [begin, end). Пример:

lang::String s1("Cat lies on the mat in some flat");

s1.substring(12); // Возвращает String(“the mat in some flat”)

s1.substring(12, 19); // Возвращает String(“the mat”)