Представление текста в ЯП. Юникод.

Юникод состоит из двух основных частей:

- универсальный набор символов (universal character set-UCS)
- набор кодировок (UCS Transformation Format - UTF), которые определяют способ представления символов на компьютере

Кодовая точка (U+xxxxxxx) <=> ?

Самые популярные - UTF-32, UTF-16, UTF-8

Были еще UTF-1, UTF-7

UTF-32:

Кодовая точка <=> целое б\з длиной 32 бита (как этот тип представлен в разных языках?) - Go - rune - синоним uint32

Проще некуда - но МИНИМУМ 11 битов пропадают зря (на деле от 16 до 25 из 32!) - это нули - неэкономно по памяти. Только для представления текста в ОП - никогда - для передачи и т.п.

Представление текста в ЯП. Юникод.

Юникод состоит из двух основных частей:

- универсальный набор символов (universal character set-UCS)
- набор кодировок (UCS Transformation Format - UTF), которые определяют способ представления символов на компьютере

UTF-16:

Кодовая точка <=> последовательность из одного или двух 16-битных целых б/з (uint16)

Принцип простой:

кодовая точка из плоскости 0 (BMP) => 1 число (uint16)

кодовая точка из плоскостей 1-17 => 2 числа (uint16 uint16)

По памяти - не хуже (а в целом - лучше), чем UTF-32.

Как не спутать символ из BMP с двумя символами из UTF-16?

BMP - неплотный диапазон (есть промежутки)

Суррогатные пары (surrogate pairs) из BMP:

U+D800 ... U+DBFF - верхняя часть СП

U+DC00 ... U+DAFF - нижняя часть СП

Эти диапазоны не включают себя ни одного символа из BMP

Представление текста в ЯП. Юникод.

Юникод состоит из двух основных частей:

- универсальный набор символов (universal character set-UCS)
- набор кодировок (UCS Transformation Format - UTF), которые определяют способ представления символов на компьютере

UTF-16:

Кодовая точка <=> последовательность из одного или двух 16-битных целых б/з (uint16)

Суррогатные пары (surrogate pairs) из BMP:

U+D800 ... U+DBFF - длина = 0x400 = 1024 (10 битов)

U+DC00 ... U+DFFF - нижняя часть СП - длина = 1024 (10 битов)

Представление текста в ЯП. Юникод.

UTF-16: алгоритм преобразования из КТ в UTF-16

```
uint32 UCS; // кодовая точка - задана
uint16 utf16 [2]; // надо получить
int len = 0; // длина полученной последовательности
if (UCS < 0x10000) { // BMP
                utf16[0] = UCS; len = 1;
} else {
   // сдвинем на длину плоскости (0x10000)
   UCS -= 0x10000; // диапазон от 0 до 2^20 -1 - 20 битов
   // выделим старшие 10 (верхние биты) и сдвинем их в диапазон верхних суррогатных пар U+D800 ... U+DBFF
   utf16[0] = (uint16)(UCS >> 10) + 0xD800;
  // выделим младшие 10 (нижние биты) и сдвинем их в диапазон нижних суррогатных пар U+DC00 ... U+DFFF
   utf16[1] = (uint16)(UCS & 0x0x3FF) + 0xDC00;
   len = 2;
}
```

Обратный алгоритм (из UTF-16 в КТ)

Представление текста в ЯП. Юникод.

UTF-16: Обратный алгоритм (из UTF-16 в КТ) - наивный - можно и получше, но менее ясно

```
uint32 UCS; // кодовая точка - надо получить
uint16 utf16 [2]; // задана
int decodedLen = 0; // сколько utf-16 символов декодировано
uint16 high_surrogate = utf16[0];
if (high_surrogate < 0xD800 || high_surrogate  >0xDBFF ) { // BMP
                        UCS = high_surrogate ; decodedLen = 1;
} else {
   // выделим закодированные 10 битов (старшие)
   high_surrogate -= 0xD800;
   USC = ((uint32) high_surrogate) << 10;
   // выделим закодированные 10 битов (младшие)
   uint16 low_surrogate = utf16[1] ;
   if (low_surrogate < 0xDC00 ||  low_surrogate > 0xDFFF)
      error("broken surrogate pair");
   low_surrogate -= 0xDC00;
   USC = USC || low_surrogate;
   UCS += 0x10000; // сдвинем в нужную плоскость
   decodedLen = 2;
}
```

Вопрос: почему 17 плоскостей?

Представление текста в ЯП. Юникод.

Вопрос: почему 17 плоскостей?

Ответ: а больше не закодируешь в UTF-16.

Вопрос: емкость современного Юникода?

$2^{20} + 2^{16} - 2^{11} = 1{,}112{,}064$

Представление текста в ЯП. Юникод.

Сравнение UTF-32 и UTF-16 - по памяти хуже (UTF-32), но нет прямого доступа.

Выделить i-й символ из строки?

utf32string s32; utf16string s16;

s32[i] == ?

s16[i] == ? // UTF-16 не является кодировкой прямого доступа!

Disclaimer: для UTF-32 нужна нормализация (иначе тоже не будет прямого доступа)

Проблема нормализации - для ЛЮБОЙ кодировки (ВАЖНО!!!).

Правда - для BMP UTF16 тоже обеспечивает прямой доступ (а как это выяснить? только путем ПОЛНОГО просмотра текста)

Аналогичная проблема - length(str) - линейная или нет?

Представление текста в ЯП. Юникод.

Сравнение UTF-32 и UTF-16 - по памяти хуже (UTF-32), но нет прямого доступа.

Выделить i-й символ из строки?

utf32string s32; utf16string s16;

s32[i] == ?

s16[i] == ? // UTF-16 не является кодировкой прямого доступа!

Disclaimer: для UTF-32 нужна нормализация (иначе тоже не будет прямого доступа)

Проблема нормализации - для ЛЮБОЙ кодировки (ВАЖНО!!!).

Правда - для BMP UTF16 тоже обеспечивает прямой доступ (а как это выяснить? только путем ПОЛНОГО просмотра текста)

Аналогичная проблема - length(str) - линейная или нет?

Представление текста в ЯП. Юникод.

Сравнение UTF-32 и UTF-16

UTF-16 - проблема порядка байтов.

Big-endian (прямой) и low-endian (обратный) - как по тексту распознать?

Решение BOM (и обратный BOM) - Byte-Order Mark - кодовая точка из BMP = 0xFEFF (обратный BOM = 0xFFFE) - пишем в начало текстового файла, если он в UTF16.

```
uint16 u;
n = read(fd, &u, sizeof u); // проверить n!!!
if (u == BOM)
    // big-endian
else if (u == REV_BOM)
   // low-endian
else
   // нарушитель конвенции....
```

Вопрос: почему для UTF-32 это не проблема?

Представление текста в ЯП. Юникод.

UTF-8 (К.Томпсон, Р.Пайк) - 1991 (ОС Plan-9)

Переменная длина (нет прямого доступа - а где он есть?)

Схема перевода в BMP и обратно

- Диапазон кодовых точек:
- 0….127 => 0 0 0 0 0 0 0 0 0 x x x x x x x ⇔ 0 x x x x x x x (1 байт)
- 128….2047 => 0 0 0 0 0 x x x x x y y y y y y ⇔

  1 1 0 x x x x x 1 0 y y y y y y (2 байта)
- 2048….65535 => x x x x y y y y y y z z z z z z ⇔

  1 1 1 0 x x x x 1 0 y y y y y y 1 0 z z z z z z (3 байта)

Представление текста в ЯП. Юникод.

UTF-8 (К.Томпсон, Р.Пайк) - 1991 (ОС Plan-9)

- Текст в ASCII-7 - это текст в UFT-8!
- Кодировка XML по умолчанию UTF-8.
- В среднем по памяти - эффективнее других
- Избыточна - есть возможность распознавать кодировку по тексту
- Не зависит от порядка байтов
- !!!! большинство функций из <string.h> работают корректно для UTF8(а какие не работают?)

Манифест- utf8 - везде!

https://utf8everywhere.org/

Представление текста в ЯП

Программа

Представление и обработка текста:
строки и символы, операции обработки строк (сравнение, конкатенация, форматирование, поиск.....)

Культурные и региональные особенности

i18n

internationalization

Ввод-вывод:
основной ТД - каналы (потоки) для ввода-вывода текста

Окружение

Файлы с текстом в разных кодировках

Интерфейс для отображения и ввода текста

Основа - представление текста.
Для простоты - единое представление (кодировка) в программе, конвертация только при вводе-выводе. Все остальные операции - в едином представлении

Представление текста в ЯП. Строки

Основа - представление текста.
Для простоты - единое представление (кодировка) в программе, конвертация только при вводе-выводе. Все остальные операции - в едином представлении.

Если современный ЯП - "с нуля" => в целом - этот подход
(Java, C#, Go, Swift)

Но в исторически возникших языках использовались вначале только однобайтовые кодировки => char === byte; string - последовательность char.

Как добавить Юникод? Коротко:

C, C++, Python 2 - два набора ТД - "старый" и новый - для Юникода. Для обработки текста - два набора операций + перекодировщики практически для любой пары кодировок (особенно в Python 2).

Чем хорошо: совместимость

Чем плохо: усложнение работы "по-новому"

Попытки ЗАМЕНИТЬ старый стиль на новый:

Python 2 => Python 3 - очень болезненный переход

PHP 6 => epic fail (PHP - 1995 -...- PHP 4 - 2000, PHP 5 - 2004, PHP 6 - 2005 - начало проекта... 2010 - закрытие (но не завершение!), PHP 7 - 2015, сейчас - PHP 8 - 26 ноября 2020)

Представление текста в ЯП. Строки

Пример 1: C (это, конечно же и C++, но там функциональность по обработке текста больше - почему?)

Два типа данных - char и wchar_t

char - для SBCS-кодировок и MBCS-кодировок (чаще всего, UTF-8, но необязательно)

(Single-Byte Character Set, MultiByte Character Set)

строка - последовательность (массив) "символов" (char или wchar_t), заканчивающаяся "нулевым" символом (0)

char * str; wchar_t * wstr;

wchar_t - для представления юникода - точнее кодировок юникода с фикс. длиной. А конкретная кодировка (и размер) - зависит от системы программирования и даже ОС (вспомним ситуацию с LLP64 и LP64)

MS Windows - wchar_t <=> UTF16 <=> unsigned short int

Linux (GNU, Clang...) wchar_t <=> UTF32 <=> unsigned int

Стандарт - char, wchar_t - главные типы

wint_t, char_16t, char_32t - стандартные typedef-ы

НО: а что с кодировкой ИСХОДНОГО текста? -и вот тут уже проблема, которая усугубляется неоднозначностью представления текста в некоторых алфавитах.

strlen("café") == ?

strlen("Привет, Игорь!") == ?

Литералы - "café" - кодировка зависит от многих факторов, 'Ю' - char и L"café", и L'Ю' - wchar_t, u8"café" - UTF8

Кодировка литералов:

"йцукенг" - зависимо от СП, u8" йцукенг ", u"йцукенг ", U" йцукенг " + экраны:

"\x66\xa9", "\ue9", "\u65\u0301", "\U0

Представление текста в ЯП. Строки

Пример 1: C - ввод-вывод в UTF-8 (работает только если локаль консоли - UTF-8)

```
#include <stdio.h>
#include <string.h>
#define BUF_MAX 128

void outstr(const char * pStr) {
	printf("len of '%s'=%d\n", pStr, (int)strlen(pStr));
}
int main(void) {
	char * cafe = "café";
	char * hello = "Привет, Игорь!";
	char buffer[BUF_MAX];
	outstr(cafe); outstr(hello);
	scanf("%s", buffer);
	outstr(buffer);
	scanf("%s", buffer);
	outstr(buffer);
	printf("%x %x", (unsigned char)cafe[strlen(cafe)-2], (unsigned char)cafe[strlen(cafe)-1]);
	return 0;
}
```

Представление текста в ЯП. Строки

Пример 1: C (это, конечно же и C++, но там поддержка - шире - почему?)

Два аналогичных набора функций
(strlen <=> wcslen, strcat <=> wcscat, strcpy <=>wcscpy.....)

Объявлены в <string.h> и <wchar.h> соответственно

+ набор функций для MBCS - но это функции для перекодировки

То же самое - для ввода-вывода -<stdio.h>:

отдельный набор функций для char и для wchar_t -

wprintf(L"%ls", pwc); wscanf(L"%d", &d); ....

Перекодировка - функции для MBCS:

mblen(), mbtowc(), mbstowcs(), wctomb(), wcstombs()

Представление текста в ЯП. Строки

Пример 1: C - ввод-вывод в wchar_t

```
#include <stdio.h>
#include <wchar.h>
#include <locale.h> // только для ввода-вывода!
#define BUF_MAX 128
void outstr(const wchar_t * pStr) {
        wprintf(L"len of '%ls'=%d\n", pStr, (int)wcslen(pStr));
}
int main(void) {
        if(!setlocale(LC_ALL, "en_US.UTF-8")) { perror(""); return 1; }
        wchar_t * cafe = L"café";
        wchar_t * hello = L"Привет, Игорь!";
        wchar_t buffer[BUF_MAX];
        outstr(cafe); outstr(hello);
        wscanf(L"%ls", buffer);
        outstr(buffer);
        wscanf(L"%ls", buffer);
        outstr(buffer);
        return 0;
}
```

А чем же wchar_t удобнее?

Представление текста в ЯП. Строки

wchar_t - независимость некоторых функций обработки текста от локали.

UTF-16 - UTF-32 - UTF-8 - что же лучше?

На первый взгляд - UTF-8 не является кодировкой с прямым доступом.

Прямой доступ => индексирование O(1) + удобство (перевод первой буквы слова, "отщипывание" флексии)

Проблема: реально нет текстов в гарантированным прямым доступом!!!

char * cafe = "café"; // это utf-8 или ANSI?

char * cafe = u8"café"; // strlen() == 5

char * cafe = "caf\xc3\xa9"; // тоже UTF-8, только "вручную"

char * cafe = "caf\xe9"; // ANSI - ISO - другая кодировка - strlen() == 4

А wchar_t выручает?

wchar_t * cafe = L"café"; // wcslen() == 4

wchar_t * cafe = L"caf\u00e9"; // wcslen() == 4

wchar_t * cafe = L"caf\u0065\u0301"; // wcslen() == 5

При выводе выглядит все АБСОЛЮТНО одинаково. Как взять последний символ?

Пример 2: C++ - как всегда - совместимость с C

Но и много чего еще...

std::string + std::wstring - динамические строки (read-write) - очень много полезных операций (+, поиск, преобразование, копирование)

+ классы для конвертации (codecvt) - "обертка" вокруг C-функций

Пример из https://en.cppreference.com/w/cpp/locale/codecvt

U0001f34c

Представление текста в ЯП. Строки

Пример 2: C++ - как всегда - совместимость с C

Но и много чего еще...

std::string + std::wstring - динамические строки (read-write) - очень много полезных операций (+, поиск, преобразование, копирование)

+ классы для конвертации (codecvt) - "обертка" вокруг C-функций

Пример из https://en.cppreference.com/w/cpp/locale/codecvt

Представление текста в ЯП. Строки

Подход MS - тип _TCHAR и функции обработки _tcsxxx (_tcscpy, _tcscat....), _T("литерал")

#define _UNICODE => _TCHAR === wchar_t и все ф-ции - из <cwchar>

#define _MBSC => _TCHAR === char и все ф-ции - из <cstring>

Удобно при переносе программ из char в Юникод и обратно

Но не панацея...

Представление текста в ЯП. Строки

C# и Java

ТД - char - UTF16 - явно несовместим с int.

string - динамические строки (в UTF16)

При вводе/выводе для потока указывается его кодировка (есть богатая библиотека классов для этого. Внутренняя кодировка - едина (UTF16)!

Перекодировка - только для I/O - библиотечными средствами.

Везде в программе - единое представление текста (UTF_16)

Представление текста в ЯП. Строки

Пример 3: C#

```
using System;
public class Test
{
	const string cafe = "café";
	const string hello = "Привет, Игорь!";
	public static void Main()
	{
		Console.WriteLine("Length of '{0}' = {1}", cafe, cafe.Length);
		Console.WriteLine("Length of '{0}' = {1}", hello, hello.Length);
		Console.WriteLine((cafe + " " + hello).ToUpper());
	}
}
```

Представление текста в ЯП. Строки

Пример 3: C#

```
using System;
public class Test
{
        const string cafe = "café";
        const string hello = "Привет, Игорь!";
        const string cafe1 = "caf\u00e9";
        const string cafe2 = "caf\u0065\u0301";
        public static void Main()
        {
                Console.WriteLine("Length of '{0}' = {1}", cafe, cafe.Length);
                Console.WriteLine("Length of '{0}' = {1}", cafe1, cafe1.Length);
                Console.WriteLine("Length of '{0}' = {1}", cafe2, cafe2.Length);
                Console.WriteLine("Length of '{0}' = {1}", hello, hello.Length);
                Console.WriteLine((cafe + " " + hello).ToUpper());
                Console.WriteLine("\U0001f342");
        }
}
```

Представление текста в ЯП. Строки

Python - типа данных char нет - только динамические строки

Python2 - первый подход (2 вида строк,), куча перекодировщиков туда-обратно, единой внутренней кодировки нет.

Python3 - второй подход - юникодовские строки, перекодировка при вводе-выводе.

Были проблемы с совместимостью....

Кодировка в последних версиях единая (UTF32)

Представление текста в ЯП. Строки

Go

типа данных char нет

rune - uint32

string - последовательность БАЙТОВ!

UTF-8 везде!

```
 const s string = "caf\u00e9"
                l := len(s)
                fmt.Println(s)
       const ss string = "cafe\u0301"
                l = len(ss)
                fmt.Println(l)
```

Посимвольно (или побайтно?)

```
 for i := 0; i < len(s); i++ {
       fmt.Printf("%x ", s [i])
  }
```

Представление текста в ЯП. Строки

Go

```
for index, runeValue := range ss {
        fmt.Printf("%#U starts at byte position %d\n", runeValue, index)
    }
 for i, w := 0, 0; i < len(ss); i += w {
        runeValue, width := utf8.DecodeRuneInString(nihongo[i:])
        fmt.Printf("%#U starts at byte position %d\n", runeValue, i)
        w = width
    }
```

Пакеты - strings, bytes, utf8 - очень много полезных средств для манипуляции строками

Вывод - непривычно, но единообразно!

Представление текста в ЯП. Строки

Swift - самое радикальное решение.

char - это что? То, что отображается на экране (бумаге и т.п.) - extended grapheme cluster

string - последовательность char

Нет прямого доступа! Нет вообще индексации по целым!

Есть специальный тип - индекс символа в строке (с итерацией)

Кодировки (8,16,32)- доступны через View - свойства строк.

Вывод - очень непривычно, но самый "честный" с точки зрения отображения путь

Представление текста в ЯП. Строки

```
let eAcuteQuestion = "Voulez-vous un caf\u{E9}?"
// "Voulez-vous un café?" используем LATIN SMALL LETTER E и COMBINING ACUTE ACCENT
let combinedEAcuteQuestion = "Voulez-vous un caf\u{65}\u{301}?"
if eAcuteQuestion == combinedEAcuteQuestion {
  print("Эти строки считаются равными")
}
for codeUnit in eAcuteQuestion.utf16 {
    print("\(codeUnit) ", terminator: " ")
}
print("")
for codeUnit in combinedEAcuteQuestion .utf16 {
    print("\(codeUnit) ", terminator: " ")
}
print("")
for scalar in combinedEAcuteQuestion .unicodeScalars {
   print("\(scalar.value) ", terminator: " ")
}
print("")
let leaf = "\u{1f342}";
print( leaf)
for scalar in leaf.unicodeScalars {
   print("\(scalar.value) ", terminator: " ")
}
```