УДК 519.686

Студ. Е.А. Богданович,
Научн. рук. ст. преподаватель А.С.Наркевич
(кафедра информационных систем и технологий, БГТУ)

ЛОКАЛИЗАЦИЯ КОНСОЛЬНЫХ ПРИЛОЖЕНИЙ

При разработке консольных приложений возникает проблема их локализации. Проблема эта вызвана тем, что в средах разработки и выполнения программ используются разные кодовые таблицы.

Рассмотрим некоторые из них. Например, таблицу ASCII: American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Однобайтовая схема кодирования позволяет работать только с 256 символами. Для представления символьных данных в кодировке ASCII используется символьный тип char, размер которого равен 1 байту. Данная кодовая таблица состоит из трёх частей. Первая часть: US-ASCII [1]. Символы с кодом 0x00-0x7F. Данная таблица содержит различные знаки, буквы английского алфавита, а также цифры. Во всех кодовых таблицах коды данных символов одинаковы. Вторая часть: Windows1251 [2]. Символы с кодом 0x80-0xFF. Данная кодовая таблица содержит различные знаки, а также буквы русского алфавита. Третья часть: CP866 [3] — символы с кодом 0x80-0xFF. Данная кодовая таблица содержит множество различных знаков и русский алфавит.

Юникод (Unicode) [4] — двухбайтная схема кодирования позволяет работать с 65536 символами. Данная кодовая таблица содержит почти все символы всех письменных языков. Для представления символьных данных в кодировке Unicode используется символьный тип wchar_t, размер которого равен 2 байтам.

Рисунок 1- Отображение русских символов на экране

```
include<iostream>
using namespace std;
int main()
{
char x, d;
cout << "A-(cp-1251)-0xC0  A-(cp-866)-0x80"<<endl;
       x = 0xC0;
       cout << "x = 0xC0" << endl;
       cout << "x=";
       cout << x;
       cout << endl;
       d = 0x80;
       cout << "d = 0x80" << endl;
       cout << "d=";
       cout << d;
       cout << endl;
}
```

```
C:\Windows\system32\cmd.exe
A-(cp-1251)-0xC0  A-(cp-866)-0x80
x = 0xC0
x=└
d = 0x80
d=A
Для продолжения нажмите любую клав
```

В примере (рисунок 1) мы используем код большой русской буквы А из двух кодовых таблиц. Код символа А в кодировке Windows-1251 равен шестнадцатеричному числу 0xC0, а в кодировке CP-866 равен 0x80. Инициализируем созданные переменные кодами 0xC0 и 0x80. На экране отобразились два символа L и A. Просмотрев таблицы мы увидим, что код 0xC0 соответствует символу L в кодовой таблице CP-866. Такое отображение вызвано тем, что консоль имеет собственную настройку кодовой страницы CP-866. Visual Studio же, хранит содержимое cpp файлов и заголовочных файлов в кодировке Windows-1251.

Средства локализации

Для учёта особенностей, связанных со страной и языком, используются специальные среды, так называемые локальным контекстом. Настройку программы на конкретный локальный контекст выполняет функция setlocale.

char *setlocale(int category, const char *locale).

Строки определяющие локальные контексты: C — используется по умолчанию английские буквы; de_DE — немецкий язык (Германия); de_AT — немецкий язык (Австрия); Rus, rus, Russian, Russian_Russia, Russian_Russia.1251, Russian_Russia.866 — русский язык (Россия).

Аргументы Category: LC_ALL — все категории, LC_COLLATE — сравнение и преобразование строк, LC_CTYPE — обработка символов, LC_MONETARY— формирование денежных сумм, LC_NUMERIC — формирование чисел, LC_TIME — формирование времени.

```
#include<iostream>
#include<string>
using namespace std;
void main()
{       string text ="Текст";
string Text_entered_in_the_console;
        cout << "Locale is :"
<< setlocale(LC_ALL, NULL)<<endl;
        cout << "Enter the Russian world:";
        cin >> Text_entered_in_the_console;
        cout << text << "\t"
<< Text_entered_in_the_console << endl;
        cout << "Locale is :"
<< setlocale(LC_ALL, "") << endl;
        cout << text << "\t"
<< Text_entered_in_the_console << endl;
        cout << "Locale is :"
<< setlocale(LC_CTYPE, ".866") << endl;
        cout << text << "\t" << Text_entered_in_the_console << endl;
        cout << "Locale: " << setlocale(LC_CTYPE, ".1251") << endl;
        cout << text << endl;
        cout << "Locale:" << setlocale(LC_CTYPE, ".866") << endl;
        cout << Text_entered_in_the_console << endl;}
```

```
C:\Windows\system32\cmd.exe
Locale is :C
Enter the Russian world:Текст_в_консоли
╥хъёЄ   Текст_в_консоли
Locale is :Russian_Russia.1251
Текст   '?Єбв_ў_ЄR-бR<Ё
Locale is :Russian_Russia.866
╥хъёЄ   Текст_в_консоли
Locale: Russian_Russia.1251
Текст
Locale:Russian_Russia.866
Текст_в_консоли
Для продолжения нажмите любую клавишу .
```

Рисунок 2

Первый вызов функции setlocale(LC_ALL, NULL) - настройка всех функций, NULL - использование стандартного локального контекста С.

Второй вызов функции setlocale(LC_ALL,” ”) - настройка всех функций, ” ” - устанавливает локальный контекст с настройками операционной системы, где предусмотрена кодовая страница Windows-1251.

Третий вызов setlocale(LC_CTYPE,”.1251”) - настройка обработки символов, ”.1251” - установка кодовой страницы Windows-1251.

Четвертый вызов setlocale(LC_CTYPE,”.866”) - настройка обработки символов, ”.866” - установка кодовой страницы CP-866.

Локаль ”.1251” корректно отображает текст инициализированный в среде vs.

```
#include<iostream>
#include<string>
#include<locale>
using namespace std;
void main()
{       locale locale_1;
        cout << "Locale is"
                <<locale_1.name()<<endl;
        string text = "Текст",
                Text_entered_in_the_console;
        cout << "Enter the Russian word:";
        cin >> Text_entered_in_the_console;
        cout << text << " "
                <<Text_entered_in_the_console<<endl;
        locale os("");
        locale::global(os);
        cout << "Locale is " << os.name()<<endl;
        cout << text << " "
                << Text_entered_in_the_console<<endl;
        locale locale_3("Russian_Russia.866");
        locale::global(locale_3);
        cout << "locale is " << locale_3.name()<<endl;
        cout << text << " "
                << Text_entered_in_the_console << endl;
        locale locale_4("Russian_Russia.1251");
        locale::global(locale_4);
        cout << "Locale is " << locale_4.name() << endl;
        cout << text << endl;
        locale::global(locale_3);
        cout << "Locale is " << locale_3.name() << endl;
        cout << Text_entered_in_the_console << endl;}
```

```
C:\Windows\system32\cmd.exe
Locale is C
Enter the Russian word:текст_в_консоли
╥хъёЄ текст_в_консоли
Locale is
Текст в?Єбв_ў_ЄR-бR<Ё
locale is Russian_Russia.866
╥хъёЄ текст_в_консоли
Locale is Russian_Russia.1251
Текст
Locale is Russian_Russia.866
текст_в_консоли
Для продолжения нажмите любую клавишу
```

Рисунок 3 - Отображение текста в консоли

В C++ локальный контекст реализуется с помощью библиотечного класса Locale (Рисунок 3). Первый шаг подключаем библиотеку locale. Далее с помощью конструкторов locale создаем объекты локального контекста. Locale locale_1- создаем стандартный локальный контекст С. С помощью метода name получаем название локального контекста. Locale os(“”)-создаем локальный контекст с настройками операционной системы, где предусмотрена кодовая таблица Windows-1251. С помощью функции global(os) устанавливаем локальный контекст глобальным. Locale locale_3(“Russian_Russia.866”) и Locale locale_4(“Russian_Russia.1251”). Устанавливаем кодовые страницы CP-866 и Windows-1251.

```
#include<iostream>
#include<Windows.h>
#include<string>
using namespace std;
void main()
{      string text,text1="Текс";
       cout << "GetConsoleCP()-"
              << GetConsoleCP() << endl;
       cout << "GetConsoleOutputCP()-"
              << GetConsoleOutputCP() << endl;
       SetConsoleCP(1251);
       SetConsoleOutputCP(1251);
       cout << "Вывод инициализированного текста:"
 << endl;
       cout << text1<<endl;
       cout << "Введите текст:"<<endl;
       cin >> text;
       cout << "Вывод введённого текста:" << endl;
       cout << text;
       cout << endl;}
```

Рисунок 4

Рисунок 4 - Функции настройки консоли

Кроме локализации, возможность которой заложена в стандартах языков С и С++, существуют нестандартные средства. Для использования функций настройки консоли требуется подключить библиотеку Windows.h. В данной программе мы используем 4 функции. С помощью функций GetConsoleCP() и GetConsoleOutputCP()-получаем установленные кодовые страницы для ввода и вывода в консоли. А с помощью функций SetConsoleCP(), SetConsoleOutputCP() сами устанавливаем необходимую кодовую страницу.

В данном случае это Windows-1251. В результате программа отображает корректно все символы, введённые в консоли, либо инициализированные в программе.

В итоге мы рассмотрели 3 способа локализации консольного приложения. Самым простым и удобным, на мой взгляд, является последний, ведь там используется всего две функции, и текст, введённый в консоли, и текст из среды разработки будет отображаться корректно.

ЛИТЕРАТУРА

1. http://vcampus.co/blogs/5190/ascii-table-with-html-entity .
2.https://otvet.imgsmail.ru/download/b02dc38922ae1ad730945228f8b8349b_i-104.jpg
3.https://allyslide.com/thumbs/d34770da0f90a71b4fa24a4bc57c3fbd/img
4. https://unicode-table.com/ru/#arabic-supplement .
5. http://www.c-cpp.ru/content/setlocale

Студ. А. А. Худницкая
Науч. рук. ассист. Л. С. Мороз
(кафедра информационных технологий, БГТУ)

ИНТЕРНЕТ-ПРИЛОЖЕНИЕ ДЛЯ УЧЕТА ЛИЧНЫХ ФИНАНСОВЫХ РАСХОДОВ

Личное финансовое планирование позволяет любому человеку правильно оценить свое финансовое состояние и распределить ресурсы для получения большего дохода. Личное финансовое планирование – это полноценный бизнес-процесс, который включает несколько этапов:

– анализ текущей ситуации, составление личного финансового плана на базе первичной информации;

– составление личного финансового плана на базе уточненных данных;

– мониторинг выполнения личного финансового плана.

Как и любой бизнес-процесс, личное финансовое планирование имеет входы и выходы (результат). В качестве входа выступают имеющиеся активы и пассивы человека, текущие доходы и расходы и предполагаемый темп их изменения в будущем, финансовые цели, их предполагаемая стоимость и срок реализации.

Выходом, или результатом процесса «Личное финансовое планирование» является личный финансовый план. [1]