ТУБ №19:
Символьные типы данных в Delphi

Сколько бы ни называли компьютеры электронно-вычислительными машинами, на одних вычислениях (и числах) далеко не уедешь.

> Угнали Василий Иванович Чапаев и Петька самолёт. Летят. Вдруг Василий Иванович спрашивает:
> — Петька, приборы?
> — 20!
> — Что «20»?
> — А что «приборы»?

Наверное, примерно такой же диалог происходил бы каждый раз между пользователем и программой, если бы не придумали способа хранить в памяти компьютера не только числа, но и текстовую информацию. Но чтобы записать текст, для начала неплохо было бы придумать, как записывать отдельные символы. С этим ведь тоже не всё так просто: вот есть, например, буква «А» — и как её нулями и единицами обозначить?

Можно было бы взять, например, азбуку Морзе. Это ту самую, когда сидят двое, на какую-то штуку, которая ключом называется, давят быстро-быстро, а она пищит на все лады. А что? Точки (короткие сигналы) нулями обозначать, тире (длинные сигналы) — единицами. Но здесь сразу же назревает ряд проблем, и главная из них, пожалуй, заключается в том, что очень уж неудобно для разных букв разное количество бит использовать: а что если понадобится одну букву на другую заменить? Там же дальше в памяти наверняка другие данные (например, следующие буквы) будут. Куда их девать, чтобы место для более длинной последовательности бит освободить?

Попробуем использовать фиксированное количество бит, одинаковое для всех символов. Сколько? Давайте посчитаем…

Для начала у нас есть 26 букв латинского алфавита — без них пришлось бы туговато. Они могут быть заглавными и строчными — итого 52. Добавим сюда ещё 10 арабских цифр — 62. Задействуем ещё знаки препинания и разные условные обозначения — точка, запятая, вопросительный и восклицательный знаки, знаки арифметических операций и операций сравнения, скобки, пробел (да, его тоже придётся как-то обозначить) — это ещё чуть больше 30 символов. Имеем чуть меньше сотни различных символов. Соответственно, 7 бит на символ должно хватить. А чтобы удобнее было работать, просто пронумеруем символы и будем считать кодом символа назначенный ему номер.

Примерно так и рассуждали создатели таблицы кодов ASCII. Правда, у них была ещё одна задача. Дело в том, что до появления мобильных телефонов и сотовых сетей SMSки отправляли посредством телеграфной связи и называли телеграммами. И, конечно же, для этих целей тоже использовали определённые способы записи передаваемых символов (тот же код Бодо, например). А в них в свою очередь кроме собственно букв и цифр были предусмотрены специальные управляющие символы. Например, символ, обозначающий, что предыдущий переданный символ следует игнорировать — прототип современной клавиши Backspace. При разработке таблицы кодов ASCII таких символов набралось целых 32 штуки, так что все $2^7 = 128$ комбинаций по 7 бит удалось задействовать.

Конечно, одного только латинского алфавита оказалось сильно недостаточно, да и байты в какой-то момент принялись формировать по 8 бит каждый, так что вскоре появились так называемые расширенные таблицы ASCII, в которых коды с 0 по 127 включительно совпадали с изначально предложенными, а добавившиеся благодаря появлению 8-го бита кода со 128 по 255 использовались для обозначения символов национальных алфавитов (например, кириллицы) и даже элементов рамок и таблиц.

Кодовые страницы, используемые в Windows, также называют кодовыми страницами ANSI.

Unicode, или всеобщая перепись символов

Пожалуй, основной проблемой ASCII стала поддержка различных языков. Да, 256 символов вполне достаточно для, например, латиницы и кириллицы, но как быть с греческими буквами? Следствием этой проблемы стало появление большого количества так называемых кодовых страниц — способов сопоставления 256 кодам тех или иных символов. Как правило, первые 128 символов в них совпадали, а коды 128–255 различались. Поскольку в текстах обычно используется всего один или два языка, этого обычно было достаточно. Проблемы начинались тогда, когда для записи текста (условно говоря, у отправителя) использовалась одна кодовая страница, а при чтении (у получателя) — другая.

А что делать китайцам и японцам с их иероглифами, у которых изначально счёт идёт на многие тысячи? Они-то вообще выкручивались, как могли. Упрощённо говоря, идея заключалась в том, что одному или нескольким кодам из 256 имеющихся придавался особый смысл: если в тексте встречался байт с таким значением, это означало, что для записи символа использовано больше одного байта.

Например, будем считать специальными кодами значения 254 и 255. Тогда можно записать как минимум 766 символов:

- 254 символа, обозначаемые одним байтом (значения от 0 до 253);
- 256 символов, обозначаемых двумя байтами, причём первый байт равен 254;
- 256 символов, обозначаемых двумя байтами, причём первый байт равен 255.

На практике использовались ещё более сложные схемы.

Конечно же, такое безобразие не могло продолжаться вечно. В 1991 году вышла первая версия стандарта Unicode, который используется и активно развивается до настоящего времени. Этот стандарт стал попыткой наконец-то навести порядок в нумерации символов и способах записи этих номеров.

Казалось бы, вот оно: сейчас будет наведён порядок. Но и здесь не обошлось без казусов. Первоначально предполагалось, что поддерживать нужно только используемые в повседневной жизни символы и что 65536 кодов (т.е. 2 байт на символ) должно быть достаточно. Разработчики операционной системы Windows стали использовать Unicode раньше большинства других ОС, вскоре после выхода самой первой версии стандарта.

Однако в более поздних версиях Unicode было принято решение добавлять абсолютно все возможные символы. Программы, написанные для Windows, уже полным ходом адаптировались к миру с 2 байтами на символ, поэтому изменения в стандарт пришлось вносить осторожно. В итоге пошли по пути, которые ранее уже применяли китайцы и японцы: диапазон кодов с $D800_{(16)}$ по $DFFF_{(16)}$ стали использовать как указание на то, что символ представлен так называемой «суррогатной парой» — т.е. представлен двумя 2-байтовыми элементами, один из которых находится в указанном диапазоне, а второй выбирает конкретный символ.

Результатом недальновидности разработчиков Unicode и поспешности разработчиков Windows стало действующее на данный момент ограничение Unicode в 1 112 064 символа. Впрочем, по состоянию на 2019 год придумать удалось всего лишь 137 994 символа, так что острой нехватки в ближайшее время не ожидается.

Помимо единой нумерации символов стандарт Unicode предлагает несколько способов их записи в памяти компьютера. Самый простой для понимания — UTF-32. В нём под каждый символ отводится 4 байта (32 бита), в которых записывается код символа как целое число.

Разумеется, для большинства языков (и особенно для латиницы) такая запись избыточна, поэтому ещё один популярный способ записи — UTF-8. Именно он чаще всего используется для передачи текстовых данных. В зависимости от кода символа в Unicode для его записи используется от 1 до 4 байт. Символы латиницы занимают 1 байт, кириллица, греческий и ряд других алфавитов — 2 байта, более редкие — соответственно 3 и 4 байта. Вместе с очевидным преимуществом — компактностью — UTF-8 имеет и существенный недостаток: обработка текстов в такой форме записи выполняется намного сложнее и ощутимо медленнее, чем в UTF-32 или ASCII.

Существует также способ записи UTF-16, который на сегодняшний день применяется в Windows. Как и UTF-8, этот способ использует разное количество памяти для записи разных символов, однако для символов большинства популярных языков оказывается достаточно одного 2-байтового элемента, что зачастую позволяет при более компактной записи получить производительность на уровне UTF-32.

Символьные типы данных

Предшественник Delphi — Borland Pascal — поддерживал только один символьный тип данных, который назывался Char (от англ. character — символ), имел размер в 1 байт и использовался для хранения символов ASCII.

Перед разработчиками Delphi стояла двойная задача: с одной стороны нужно было сделать так, чтобы программы, написанные для Borland Pascal, могли быть перекомпилированы без внесения изменений, с другой — поддержать работу с Unicode, в частности, с использующимся в Windows UTF-16 (на момент создания Delphi — UCS-2). Результатом стал следующий набор символьных типов:

Название типа	Размер	Описание
Доступные в ранних версиях Delphi		
AnsiChar	1 байт	256 символов ASCII/ANSI
WideChar	2 байта	Символы Unicode (UCS-2, UTF-16)
Доступные в современных версиях Delphi		
AnsiChar	1 байт	256 символов ASCII/ANSI
WideChar	2 байта	Символы Unicode (UCS-2, UTF-16)
UCS2Char	2 байта	Эквивалентен WideChar
UCS4Char	4 байта	Символы Unicode (UTF-32)
Тип Char		
Char	Эквивалентен AnsiChar в ранних версиях Delphi Эквивалентен WideChar в современных версиях Delphi	

Программы, которые не полагаются на конкретные размеры символьного типа данных, а лишь используют его для работы с символами, могут использовать тип Char. Характеристики этого типа могут изменяться по мере развития способов представления символьных данных. В тех же случаях, когда необходимо работать с символами строго определённым образом, можно использовать AnsiChar или WideChar, а также, в современных версиях Delphi, UCS2Char и UCS4Char.

Операции, встроенные функции над символьными типами данных

Над символьными типами данных определены только операции сравнения. При сравнении большим считается тот символ, код которого имеет большее числовое значение. Для сравнения символов удобно иметь в виду, что как в ASCII/ANSI, так и в Unicode первые 128 символов определены одинаково, причём:

- символам десятичных цифр от «0» до «9» назначены последовательные значения кодов соответственно от 48 до 57, что в двоичной системе счисления соответствует значениям вида 0011 xxxx, где xxxx — двоичное представление соответствующей цифры;
- символам букв латинского алфавита назначены последовательные значения кодов от 65 до 90 для заглавных и от 97 до 122 для строчных, что в двоичной системе счисления соответствует значениям вида 010xxxxx для заглавных и 011xxxxx для строчных, где xxxxx — двоичное представление порядкового номера буквы в латинском алфавите.

Имя	Вид	Описание	Тип результата
UpCase(x)	Функция	Если x — строчная буква латинского алфавита, возвращает соответствующую ей прописную, иначе возвращает x	Символьный
Определённые для всех перенумерованных типов			
Ord(x)	Функция	Возвращает порядковый номер значения аргумента во множестве значений типа	Integer
Pred(x)	Функция	Возвращает значение, следующее во множестве значений типа перед значением аргумента	Тип x
Succ(x)	Функция	Возвращает значение, следующее во множестве значений типа после значения аргумента	Тип x
Определённые для всех типов			
SizeOf(x)	Функция	Возвращает размер типа в байтах	Integer

Функция UpCase используется для преобразования символов в верхний регистр. Работа этой функции только для букв латинского алфавита обусловлена тем, что для других алфавитов это преобразование может выполняться сложнее или и вовсе неоднозначно.

Например, алфавит русского языка как в Unicode, так и в кириллической кодировке Windows-1251 представлен последовательными кодами, как и латиница, однако с пропуском буквы «Ё» (как среди заглавных, так и среди строчных), причём соотношение между кодами символов «Ё» и «ё» отличается от аналогичных соотношений для остальных букв русского алфавита. Для белорусского языка аналогичную проблему составляет ещё и буква «Ў».

Ещё больше проблем связано с особенностями конкретных языков. Например, в греческом языке заглавной букве «Σ» в начале и середине слова соответствует строчная «σ», а в конце слова — «ς». Это может показаться сравнительно безобидным, но есть и более сложные случаи [1, 2]. Следует также понимать, что иногда правила языка подвергаются изменениям на государственном уровне, поэтому встраивать готовое универсальное решение в язык программирования было бы просто неоправданным. Для решения этой задачи, как правило, стараются применять специализированные библиотеки.

Как следует из перечня встроенных функций, символьные типы являются перенумерованными. Порядковым номером значения символьного типа (его возвращает функция Ord) считается код этого символа, т.е. результат интерпретации его двоичной записи как целого числа.

[1] https://habr.com/ru/post/147387/

[2] https://devblogs.microsoft.com/oldnewthing/20030905-00/?p=42643

Дополнительные вопросы

1. Как можно записать преобразование строчной латинской буквы в заглавную без использования функции UpCase (используя только операции и встроенные функции Delphi)?