ЭВОЛЮЦИЯ ТИПОВ ДАННЫХ В ЯЗЫКАХ C, C++, C#

Кудрина Е.В., Огнева М.В.

Саратовский государственный университет им. Н.Г. Чернышевского,
Саратов, Россия

Развитие языков программирования шло параллельно с развитием вычислительной техники. Одним из первых «полноценных» языков программирования можно считать язык ассемблер, в котором появились символические имена для обозначения переменных и команд. Это позволило разрабатывать на ассемблере компактные и высокопроизводительные программы. Вместе с тем, для написания даже самой простой программы на ассемблере программисту требовалось знать машинные команды конкретного типа процессора и напрямую обращаться к данным, размещенным в его регистрах.

Стремительное развитие вычислительной техники во второй половине 20 века вело к быстрой смене типов и моделей процессоров, поэтому стало необходимо обеспечивать аппаратную переносимость программ. Это привело к появлению языков программирования высокого уровня. Программы стали разрабатываться на языках, «схожими» с естественными языками.

Развитие языков программирования высокого уровня позволило создавать относительно большие по объему программы, реализующие достаточно сложные вычислительные алгоритмы. Однако с ростом объема программ становилось невозможным удерживать в памяти все детали их реализации, и стало необходимым структурировать информацию, выделяя главное и отбрасывая несущественное. Это привело к появлению структурного программирования.

Первым шагом в данном направлении стало использование подпрограмм, которые позволяли после их разработки и отладки отвлечься от деталей их реализации. Вторым шагом стала возможность создания собственных типов данных, позволяющих структурировать и группировать информацию. Для работы с собственными типами данных стали разрабатываться специальные подпрограммы, которые объединялись в модули и помещались в библиотеку подпрограмм. Таким образом, во всех языках программирования появились обширные библиотеки стандартных подпрограмм.

Одним из языков, реализующим концепции структурного программирования является язык C, который был разработан Денисом Ритчи в 1972 году [2]. Среди преимуществ языка C следует отметить возможность получения программного кода, сравнимого по скорости выполнения с про-

граммами, написанными на языке ассемблера. Это связано с тем, что С имеет не только полный набор средств структурного программирования, но и обладает набором низкоуровневых средств, обеспечивающих доступ к аппаратным средствам компьютера.

Язык С является строго типизированным языком, т.е. при объявлении идентификаторов (имен переменных, констант, функций) требуется указать его тип. При этом тип определяет: внутреннее представление данных в памяти компьютера; объем оперативной памяти, необходимой для размещения значения данного типа; множество значений, которые могут принимать величины этого типа; операции и функции, которые можно применять к величинам этого типа.

Стандарт языка С включает в себя базовые типы (целые, вещественные, символьные, логический) и типы, определенные пользователем (перечислимые, массивы, структуры, файлы, ссылки и указатели). Кроме этого существует специальный тип void, который не предназначен для хранения значений и применяется обычно для объявления подпрограмм-функций, которые не возвращают значения. С одной стороны, такое многообразие типов позволяет программисту влиять на распределение ресурсов во время выполнения программы. А с другой стороны – это накладывает на программиста ответственность за допустимость совершаемых операций с данными.

Структурный подход позволил создавать достаточно крупные проекты, а также за счет готовых библиотек, уменьшать время их разработки и облегчать их модификацию. Но сложность программного обеспечения продолжала возрастать, и требовались все более сложные средства ее преодоления. Идеи структурного программирования получили свое дальнейшее развитие в объектно-ориентированном программировании (ООП). ООП – это не просто новый подход в программировании, это новая парадигма, которая определяет модель структурирования информации, организации данных и вычислений.

Парадигма ООП основывается на введении понятия «класс». В программном понимании «класс» является типом данных, определяемым пользователем. Конкретные величины типа данных «класс» называются экземплярами класса или объектами.

Основными принципами ООП являются инкапсуляция, наследование и полиморфизм. Инкапсуляция – это объединение в одном классе данных и методов их обработки в сочетании с сокрытием ненужной для использования этих данных информацией. Это позволяет изменить реализацию класса, не затрагивая саму программу, при условии, что интерфейс класса останется прежним. Наследование – это возможность создания иерархии классов, в которой потомки наследуют все свойства своих предков, могут их изменять и добавлять новые. Так как свойства предков у потомков по-

вторно не описываются, сокращается объем программ. Полиморфизм – это возможность использования в рамках одного класса или различных классов одного имени для обозначения сходных по смыслу действий, что позволяет гибко выбирать требуемое действие во время выполнения программы.

Одним из языков, реализующим парадигму ООП является язык C++. Язык C++ начал разрабатывать Бьерн Страуструп в 1979 году [3]. Первоначально он был назван "C с классами", но в 1983 году это имя было изменено на C++. C++ полностью включает элементы языка C, а большинство внесенных дополнений предназначены для поддержки ООП. Поскольку язык C++ является прямым наследником языка C, то он унаследовал все его типы данных, к которым добавились классы. Позже в язык C++ добавилась библиотека стандартных шаблонов (Standard Template Library, STL), которая включает в себя стандартные классы и функции, реализующие наиболее популярные и широко используемые алгоритмы и структуры данных. Например, в STL поддерживаются такие структуры данных как вектора, списки, очереди и стеки. В ней также определены различные процедуры доступа к этим структурам данных. STL строится на основе классов-шаблонов, поэтому входящие в нее алгоритмы и структуры применимы почти ко всем типам данных, определенным в C++.

В 2000 году компания Microsoft объявила о создании нового языка программирования - языка C#, основоположниками которого стали Андерс Хейлсберг, Скотт Вилтамут и Питер Гольде [1]. Эта акция стала частью более значительного события - появления платформы .NET (.NET Framework). Платформа .Net по сути представляла собой новую модель создания приложений, которая включает в себя следующие возможности: использование библиотеки базовых классов (Framework Class Library, FCL), предлагающих целостную ООП-модель программирования для всех языков программирования, поддерживающих .NET; полное и абсолютное межъязыковое взаимодействие, позволяющее разрабатывать фрагменты одного и того же проекта на различных языках программирования; единую среду выполнения .Net приложений (Common Language Runtime, CLR), которая выполняет приложения независимо от того, на каких языках программирования платформы .Net они были созданы, и независимо от того, на какой аппаратной платформе выполняется приложение. При этом CLR берет на себя контроль за безопасностью выполнения приложений и автоматическое управление ресурсами.

В языке C# сглажено различие между типом и классом. Все типы – базовые и определенные пользователем - являются классами, связанными отношением наследования. Родительским классом является класс Object. Все остальные классы являются его потомками. Поэтому в C# используется другая классификация типов данных, согласно которой все типы можно

разделить на три категории: тип void, значимые типы (value) и ссылочные типы (reference). Кроме этого существует специальный тип указатель (pointer), который используется для работы с неуправляемым кодом, т.е. кодом, не контролируемым средой CLR.

Принципиальное различие между значимыми и ссылочными типами состоит в способе хранения их значений в памяти. В первом случае значение хранится в стеке данных (или как часть объекта ссылочного типа). Во втором случае адрес переменной ссылочного типа хранится в стеке, а сам объект – в куче. Сборщик мусора среды CLR автоматически уничтожает программные элементы: в стеке данных после того, как закончит существование раздел стека, в котором они объявлены; а в куче объект уничтожается через некоторое время после того, как уничтожена последняя ссылка на него.

К значимым типам в C# относятся: логический тип, все арифметические типы, структуры, перечисление. Массивы, строки и классы относятся к ссылочным типам.

Для того, чтобы наиболее полно раскрыть эволюцию типов данных в языках C, C++ и C#, проанализируем особенности представления строк в этих языках.

Язык C не содержит стандартного типа данных «строка», например, как в языке Pascal. Строка в стиле C представляет собой массив символов, заканчивающийся нуль-символом '\0'. По положению нуль-символа определяется фактическая длина строки. Объявить строку и инициализировать ее строковым литералом можно следующим образом: *char str[20]= "программирование"*. В этом случае под строку выделится 20 байт, 16 из которых будет занято символами строкового литерала, а семнадцатый – нуль-символом.

Обратиться к символу строки можно так же как и к элементу обычного массива, указав имя строковой переменной и, в квадратных скобках, индекс требуемого символа. Например, чтобы обратиться к первому символу строки, нужно записать *a[0]*, т.к. нумерация элементов массива начинается с нуля. Обрабатывать данную строку можно посимвольно, например, с помощью цикла for. При этом допустимо модифицировать строку.

Однако при работе с такой строкой надо следить, чтобы нуль-символ не был заменен каким-нибудь другим значением, т.к. в этом случае невозможно будет определить конец строки. И хотя программа сможет продолжить работу, результат этой работы будет непредсказуемым, т.к. за счет выхода за пределы массива будет обрабатываться фрагмент памяти, не имеющий отношения к строке.

Строки в стиле C отличаются от других типов данных тем, что многие операции языка C к ним неприменимы. Так, для строк недопустимо использовать операцию присваивания. А при использовании операции

сравнения будут сравниваться не значения строк, а их адреса. Поэтому для работы со строками разработана стандартная библиотека cstring, которая содержит функции поиска, модификации и копирования массива символов. Например, скопировать стоку str в строку s можно с помощью функции strcpy следующим образом: *strcpy(s,str).* Следует отметить, что данная функция не проверяет, поместится ли значение str в s, следовательно, может возникнуть ошибка – потеря нуль-символа. Таким образом, при работе со строками в стиле С на программиста ложится огромная ответственность за корректность выполняемых действий.

В С возможно создать неизменяемую строку, например, следующим образом: *char *str="программирование"*. В данном случае str является указателем на строковую константу и никакое изменение ее значения невозможно. Например, оператор *str[0]='П'* недопустим. Однако можно использовать стандартные функции поиска и копирования из библиотеки для работы со строками в стиле С.

Язык С++ также не содержит типа данных «строка» в классическом понимании. Он наследует от С представление строк в виде массива символов, но при этом появляется второй способ работы со строковыми данными с помощью класса string. Объявить такую строку и инициализировать ее строковым литералом можно следующим образом: *string str ("программирование").*

В классе string реализованы операции и функции, которые автоматически контролируют длину строки, гарантируя невозможность выхода за ее границу, динамически изменяют длину строки при необходимости, позволяют обрабатывать строку посредством стандартных операций, например, сравнения и присваивания. С++, что позволяет использовать строки в составе стандартных выражений. Например, к строке str добавить строку *s* можно с помощью операции *str+=s* или с помощью функции *str.append(s).*

Количество функций класса string примерно соответствует количеству функций в библиотеке С cstring, но механизм перегрузки существенно расширяет возможности их применения. Например, различные варианты метода append могут добавить в строку *str* только часть другой строки, или некоторый символ *n* раз. При необходимости к символам строки можно обращаться с помощью индекса, например, следующим образом *str[0]*, и обрабатывать строку посимвольно с помощью, например, цикла for.

Для языка С# обработка текстовой информации является одной из самых важных задач, поэтому в С# расширяется набор средств для работы со строками, к которым относятся массивы символов Char[], и классы String, StringBuilder и Regex..

В С# Char[] – это массив символов, своеобразный аналог строк в стиле С. Однако в отличие от С и С++, массив символов С# не задает стро-

ку, заканчивающуюся нулем, он создает динамический массив символов. Объявить объект класса Char[] и инициализировать его можно одним из следующих способов: *Char[] mas ={'Э','В','М'}* или *Char [] sl="ЭВМ".ToCharArray()*. Метод ToCharArray преобразует строку в массив символов. Инициализировать массив символов строковой константой в C# нельзя.

Массив символов, как и всякий класс-массив в C#, является наследником не только класса Object, но и класса Array, и, следовательно, обладает всеми методами родительских классов, которые позволяют копировать элементы массива, осуществлять поиск, сортировать элементы массива и много другое. Например, упорядочить массив символов mas по алфавиту можно следующим образом: *Array.Sort(mas)*. Специфических методов, которые позволяли бы выполнять операции над строками, для массива символов не существует.

Класс String в C# предназначен для создания неизменяемых строк. Создать объект класса и инициализировать его можно несколькими способами. Например, допустимо инициализировать объект класса *string* строковой константой : *String str="программирование"*.

String обладает богатым набором методов для сравнения строк, поиска и замены данных в строке, удаления подстроки и много другое. К объектам класса String допустимы также многие стандартные операции, такие как присваивание, сравнение. Но так как класс String является ссылочным типом данных, то при выполнении этих операций будут рассматриваться не значения строк, а ссылки на них. «Неизменяемость» объекта класса *String* выражается в том, что в результате выполнения операций и методов изменяются копии объекта, а не сам объект. Например, если выполнить следующее действие *str+=" и ЭВМ"* для строки *String str="программирование"*, то создастся новый объект, инициализированный строковой константой *"программирование и ЭВМ"*, на которую будет установлена ссылка *str*, а ссылка на исходный объект будет потеряна. «Потерянный» объект будет занимать память, но ссылка на него не будет существовать. Такие объекты через некоторое время удаляются сборщиком мусора среды CRL, но их поиск снижает быстродействие программы.

При необходимости к объекту класса String можно обращаться с помощью индекса, но в этом случае можно только просмотреть требуемый символ строки. При попытке изменить его будет создан новый объект.

Класс StringBuilder языка C# предназначен для работы с изменяемыми строками. Объявить и инициализировать объект данного класса можно только с помощью явного конструктора (для Char[] и String явным образом конструктор не вызвался), например, следующим образом:
StringBuilder str= new StringBuilder("программирование").

Методы класса StringBuilder менее развиты, чем методы класса String, но зато они позволяют изменять сам объект, а не его копию.

Класс Regex – это класс регулярных выражений, который позволяет осуществить эффективный поиск данных в тексте по заданному шаблону, редактирование текста, а также формирование итоговых отчетов по результатам работы с текстом. Регулярные выражения - мощный и сложный инструмент C#, базирующийся на таких структурах данных как списки и коллекции, его описание – это тема для отдельной книги.

Список литературы

1. Джесс Либерти. Программирование на C#.: Пер. с англ. – 2-е изд, С.Пб.: Издательство «Символ-плюс», 2002. – 684 с.

2. *Керниган Б., Ритчи Д.* Язык программирования Си. \ Пер. с англ., 3-е изд., испр. – СПб.: «Невский диалект», 2001.

3. *Липпман С.Б., Лажойе Ж.* Язык программирования C++. Вводный курс. 4-е изд-е. Изд. дом "Вильямс", 2007 г..