



Task #07

C# Basic Syntax. if-else Conditional Statement. Branching Algorithms



LEARN. GROW. SUCCEED.

© 2020. STEP Computer Academy - a leader in the field of professional computer education
by Viktor Ivanchenko / ivanvikvik@gmail.com / Minsk

Task #07

Базовый синтаксис языка Java. Условная конструкция if-else. Разветвляющиеся алгоритмы

Цель работы

Изучить синтаксис условной конструкции *if-else* языка программирования C# для программирования разветвляющихся алгоритмов (ветвлений) и закрепить её на примере разработки простейшего интерактивного консольного приложения.

Требования

- 1) Для каждого задания в начале рекомендуется разработать блок-схему алгоритма решения.
- 2) При разработке программ рекомендуется придерживаться принципа единственной ответственности (***Single Responsibility Principle, SRP***), т.е. любой код (метод, класс и т.д.) должен быть настолько самостоятельным, насколько это возможно, чтобы его можно было повторно использовать в дальнейшем в других приложениях.
- 3) Все задания необходимо решать, используя только базовые операции (простые операторы), определённые над примитивными типами данных в языке программирования C#, и условные конструкции (т.е. не нужно использовать циклические конструкции, массивы, строковые данные и операции над ними и т.д.).
- 4) Если логически не подразумевается или в задании иного не указано, то входными и выходными данными являются вещественные числа (числа с плавающей запятой).
- 5) Целевые данные программы должны задаваться пользователем с консоли.

- 6) В соответствующих компонентах бизнес-логики необходимо предусмотреть «защиту от дурака», т.е. прежде чем выполнять действия с данными нужно проверить, являются ли данные адекватными (непротиворечивыми).
- 7) Для осуществления ввода-вывода данных с консоли рекомендуется использовать соответствующие статические методы класса **Console** из стандартного пространства имён **System**.
- 8) Для генерирования псевдослучайных данных рекомендуется использовать соответствующие методы объекта класса **Random** из стандартного пространства имён **System**.
- 9) Программа должна быть снабжена дружелюбным и интуитивно понятным интерфейсом для взаимодействия с пользователем. Рекомендуется отображать интерфейс программы на английском языке.
- 10) При проверке работоспособности приложения необходимо проверить все тестовые случаи.
- 11) При разработке программ придерживайтесь соглашений по написанию кода на C# (C# Code-Convention).

Основное задание

- 1) Даны три числа. Напишите программу «*The Greatest*», которая определяет, которое из данных трёх чисел наибольшее (или можно наименьшее). Предусмотреть возможность равенства всех или некоторых значений.
- 2) В молодом возрасте дракон каждый год отращивает по три головы (а может и больше), но после того, как ему исполнится 200 лет – только по две (или другое количество голов), а после 300 лет – лишь по одной. Разработайте программу, которая высчитывала бы, сколько голов и глаз у дракона, которому N лет?
- 3) Даны три стороны в виде вещественных чисел. Напишите программу, которая определяет, являются ли данные стороны сторонами треугольника.
- 4) Напишите программу, которая бы эмулировала игру «*Dice*» (игра в кости). Суть игры заключается в броске двух шестигранных кубиков (костей) и подсчёта общей суммы очков, которые выпали на первой и второй костей.

- 5) Напишите программу, которая бы определяла, является ли введённая буква гласной или согласной (постарайтесь сделать данное задание двумя способами: с использованием конструкции **if-else** и операций отношений в комбинации с логическими операциями).
- 6) Разработать интерактивную программу «*Quadric Equation*» («Квадратное уравнение») для решения квадратных уравнений вида: $ax^2 + bx + c = 0$. Программа должна запрашивать у пользователя соответствующие параметры a , b и c , в зависимости от вычисленного дискриминанта D , выдавать соответствующий результат. В случае отрицательного дискриминанта программа должна выводить сообщение о том, что действительных корней нет.

Дополнительное задание

Заданы три целых числа, которые задают некоторую дату по Григорианскому календарю (https://ru.wikipedia.org/wiki/Григорианский_календарь). Определить дату следующего дня. Запрещается использовать типы стандартной библиотеки языка для работы с датой и временем. Также необходимо учесть то, что по Григорианскому календарю (используется в настоящем момент) високосный год определяется следующим образом:

- годы, кратные 4 – високосные (например, 2008, 2012, 2016);
- годы, кратные 4 и 100 – невисокосные (например, 1700, 1800, 1900);
- годы, кратные 4, 100 и 400 – високосные (например, 1600, 2000, 2400).





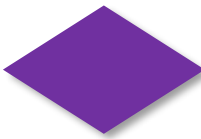

Best of LUCK with it, and remember to HAVE FUN while you're learning :)
Victor Ivanchenko

Графическое представление алгоритмов



Для общего представления решения задачи без привязки к конкретному языку программирования на практике используют блок-схемы. Они позволяют в графическом виде представить алгоритм решения задачи, который понятен не только разработчику, но даже домохозяйке.

Для графического представления алгоритмов решения задачи использую специальные унифицированные блоки, каждый из которых несёт в себе определённую смысловую нагрузку. Кратко каждый из наиболее востребованных блоков описывается в нижеприведённой таблице.

Таблица 1 – Наиболее часто используемые блоки

#	Shape (блок)	Description (описание)
1.		Блок начала/окончания выполнения программы
2.		Блок данных – используется для ввода, объявления и инициализации переменных программы
3.		Блок действия – используется для вычисления любых выражений программы
4.		Блок вызова процедур или функций – используется для обозначения вызова пользовательской функции или процедуры, код или реализация которой находится в другом файле
5.		Блок условия – задаёт соответствующие условия дальнейшего выбора хода выполнения кода программы
6.		Блок вывода данных – используется для обозначения выводимых данных или результата работы программы

Продолжение таблицы 1

7.		Блок соединитель на странице – используется в случае, если блок-схема алгоритма не может идти всё время сверху вниз и требуется её перенести на другую часть свободного места на том же листе
8.		Блок соединитель между страницами– используется в случае, если блок-схема алгоритма не помещается на одной странице и одну из её частей нужно перенести на другую страницу

7 смертных грехов программирования



Джон Парселл (John Purcell), создатель онлайн курсов по Java и др. языкам программирования и технологиям (www.caveofprogramming.com)

1. Использовать «Пробел» вместо «Tab». Всегда, всегда используйте **«Tab»**, а не «Пробел».
2. Использовать «Tab» вместо «Пробела». Всегда, всегда используйте **«Пробел»**, а не «Tab».
3. Не использовать автоформатирование. Забудьте про весь мусор вроде табов и пробелов, **используйте автоформатирование** в своем коде и людям не придется видеть ваши странные скобки и отступы.
4. Использовать интегрированную среду разработки (IDE) с ее автоформатированием и цветными клавишами. Все коды должны быть написаны в vi или Emacs, что подтверждает безупречность ваших навыков программирования.
5. Не использовать IDE. Никто не хочет платить за время, которое вы тратите на набор текста, если это можно сделать в один клик, или за прокручивание вверх-вниз с помощью заумной комбинации клавиш из LISP.
6. Не учить C и C++. Два этих языка жизненно необходимы любому программисту. Думаете, Java так же хорош? Отлично, создайте мне систему управления гоночными автомобилями в режиме реального времени на Java, и я вам поверю.

7. Учить С и С++ в то время, которое вы могли бы использовать на что-то более современное, например, на Java. Признайте – все таблицы, написанные на С или С++, изживают себя в течение 5 лет. И в таком случае в программном обеспечении есть серьезные ошибки, которые Java просто не позволил бы вам совершить.

Source: <https://www.kv.by/post/1053298-7-smertnyh-grehov-programmirovaniya>