# Лекция 2. Линейные и ветвящиеся алгоритмы. Вычисления

**Что делать?**

1. Повторить теорию из прошлой лекции:
   1. типы данных;
   2. объявление переменных;
   3. ввод и вывод данных через консоль;
2. Ознакомиться с теорией по темам:
   1. арифметические операции;
   2. оператор присваивания;
   3. класс **Math**;
   4. условный оператор **if**;
   5. запись условий (операторы сравнения и логические операторы);
   6. условная операция **? :**
   7. оператор множественного выбора **switch**.
3. Реализовать примеры из данного файла. Выполнять коммит тогда, когда это указано в задании, или чаще. Ответить на вопросы в тексте.
4. Выполнить задания на самостоятельную работу. Если выполняется дополнительное задание, то обязательно отдельно закоммитить код без него.
5. Сдать все через GitHub. Не забудьте добавиться в последователи (кнопка «Follow» в профиле преподавателя), чтобы ваши работы можно было найти.

## +Пример 1. Сложение двух целых чисел

Написать программу, которая считывает два числа и складывает их.

### Указания к выполнению

Даже такое простое задание можно (и нужно) разбить на последовательность более простых действий:

1. Считать два числа и сохранить их в переменные.
2. Сложить значения переменных и записать их в третью переменную.
3. Вывести результат на экран.
4. Завершить работу программы.

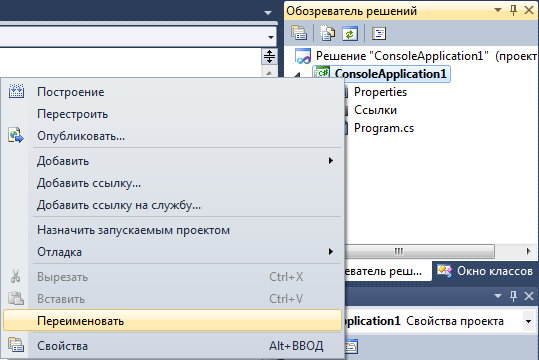
Разбиение исходной задачи на более простые части называется **декомпозицией** (композиция – соединение, де – отрицательная приставка).

Умение правильно декомпозировать проблему – один из наиболее важных навыков программиста. Зачастую декомпозицию приходится проводить многократно, разбивая программу на все более мелкие и мелкие задачи.

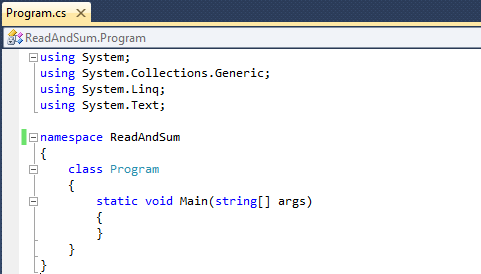
Начнем разработку программы по шагам.

Запустите среду разработки VisualStudio. Создайте новый проект – консольное приложение: «Файл» (File) → «Создать проект...» (Create Project...) (Ctrl+Shift+N) → в открывшемся окне в категории Visual C# выбрать «Консольное приложение» (Console Application). Задайте имя приложения «ReadAndSum» (без пробелов!) вместо «ConsoleApplication1».

Если вы создали приложение со стандартным именем «ConsoleApplication1», то его можно переименовать в «Обозревателе решений» (Project Explorer) (по умолчанию находится в правой верхней части окна VisualStudio).



После переименования проекта, нужно изменить название пространства имен (namespace) в исходном коде.



Сохраните проект в отдельной папке и выполните его коммит (push пока необязательно).

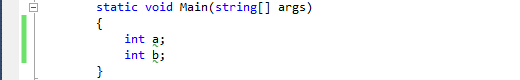
**Помните**: если вы сохранили проект прямо в локальный git-репозиторий, у вас запущен git-клиент и вы забудете выполнить коммит сделанных изменений – все ваши файлы и изменения в них **будут безвозвратно удалены!**

Либо не нужно запускать git-клиент, пока вы не собираетесь закоммитить исходный код,

либо, вы храните исходный код отдельно, по мере необходимости копируя его в локальный репозиторий.

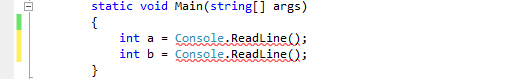
Приступим к написанию программного кода в теле функции Main.

Добавьте объявление переменных для исходных чисел. Поскольку в задании не уточняется тип данных, зададим их целыми числами типа int. В качестве имен переменных возьмем a и b, но на практике рекомендуется использовать более осмысленные имена.

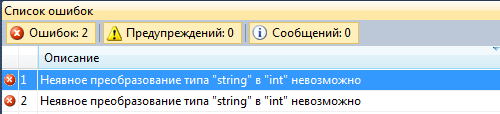


* Самостоятельно проверьте, чему равны переменные a и b, пока им не присвоены никакие значения. Для этого запустите программу в пошаговом режиме (F11). При этом программа будет делать паузу на каждой строке кода, исполняющейся в данный момент. Если в этот момент навести курсор на переменную, то во всплывающей подсказке отобразится ее значение. Значение всех переменных также отображается в окне в нижней части VisualStudio (только пока программа запущена).

Наиболее очевидное решение – сразу же ввести значения переменных с консоли:



Однако, VisualStudio сразу же укажет на ошибку, а при попытке запустить программу (F5 или кнопка  на панели инструментов) будет выведено сообщение об ошибке:



C# – строго типизированный язык. С консоли считывается значение типа string, которое не может автоматически «превратиться» в int.

Для преобразования текста в число необходимо использовать метод Parse или TryParse соответствующего типа данных[[1]](#footnote-1):

<числовое\_значение> = <тип\_данных>.Parse(<строковое\_значение>)

<тип\_данных>.TryParse(<строковое\_значение>, out <числовое\_значение>)

Например,

x = int.Parse(Console.ReadLine())

double.TryParse(str, out y)

В чем разница между Parse и TryParse? Пока текст корректно преобразуется в число – разница чисто синтаксическая.

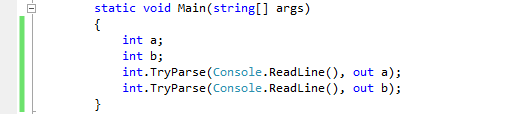
Однако, если пользователь в консоли введет текст, который нельзя корректно перевести в число (в нашем примере – целое число), то поведение этих методов различается.

Метод Parse вызовет исключение и, если его не обработать, то программа аварийно завершит свою работу (проще говоря, вылетит). Обрабатывать исключение мы научимся позже.

При использовании TryParse вылета не произойдет, даже если ввести неверное значение. Строка просто не будет преобразована в текст. TryParse возвращает значение true, если перевод был успешным, и false – в противном случае. Таким образом, TryParse можно использовать в качестве условия в операторе if.

* Самостоятельно проверьте, чему будет равна числовая переменная, если передать в TryParse неверную строку. Изменится ли ее значение?

Пока будем использовать TryParse:



Запишем сумму a и b в третью переменную с именем sum и выведем на экран ее значение. Чтобы программа не закрывалась сразу, добавим в конце ожидание нажатия клавиши.



* Можно ли вывести значение суммы без использования переменной sum? Можно ли обойтись одной переменной a? Можно ли обойтись вообще без переменных?

Закомитьте и загрузите (push) полученную программу на **GitHub**.

Протестируем работу программы на правильных и неправильных исходных данных.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | | Выходные данные |
| a | b | sum |
|  | 59 | 14 | 73 |
|  | 62 | -90 | -28 |
|  | 12 | 0 | 12 |
|  | aab | 5 | 5 |
|  | 4 | 5.5 | 4 |

При тестировании были проверены различные значения: положительные, отрицательные и нулевые целые значения, текст вместо числа и дробное число вместо целого.

Как видно из результатов теста, при неверных значениях переменных они принимаются за 0. Хотя более логичным было бы вывести сообщение об ошибке. Для этого необходимо проверять правильность перевода методом TryParse (будет в следующей работе).

### Задание на самостоятельную работу

#### +Задание 1

Разработать программу для преобразования температуры в градусах Цельсия в градусы Фаренгейта.

#### +Задание 2

Напишите программу, которая будет вычислять значение функции *f*(*x*) по введенному значению *x*. Функция f(*x*) задана выражением:



где *a*, *b* и *c* – номера дня, месяца и года вашего рождения.

## +Пример 2. Максимальное из трех чисел

Вывести максимальное из трех введенных пользователем чисел.

### Указания к выполнению

Сначала напишем более простой вариант программы – для выбора максимального из двух чисел.

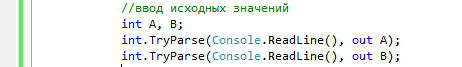
Примечание – если числа равны, то максимальным будет любое из них. Однако, если бы в задании спрашивалось, какое именно из чисел является максимальным (ответ: первое, второе или третье, а не просто знаение), то равенство было бы отдельным вариантом (оба числа максимальные).

Алгоритм работы программы:

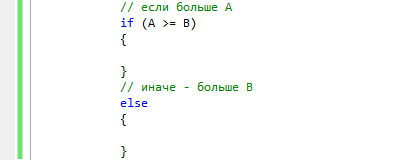
1. Пользователь вводит три числа A, B, C.
2. Если A ≥ B, то максимальное A, иначе максимальное B.
3. Вывести результат на экран.
4. Завершить работу программы.

Создайте новый проект – консольное приложение и сохраните его под именем Maximum.

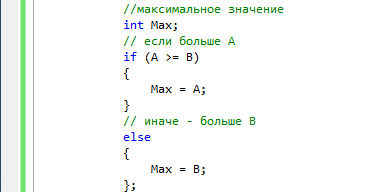
Добавьте блок ввода значений переменных по аналогии с предыдущим примером:



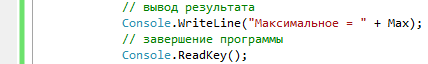
Сравним значения A и B в условии оператора **if**:



Добавим переменную Max для сохранения максимального значения и присвоим ей значение в каждом блоке.



Добавим вывод результата:

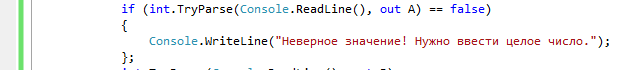


Сохраните, запустите и протестируйте программу. Если все работает корректно, выполните коммит с комментарием «Л2.П2. Максимальное из двух чисел».

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | | Выходные данные |
| A | B | Max |
|  | 121 | 12 | 121 |
|  | 14 | 50 | 50 |
|  | 20 | 20 | 20 |
|  | -55 | -2 | -2 |
|  | aab | 10 | 5 |
|  | 1 | 5.5 | 4 |
|  | -1 | b | 0 |

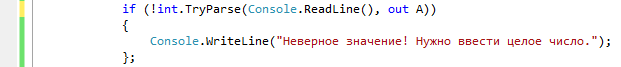
Как и в предыдущем примере, неверные значения считаются равными нулю, поэтому в тесте №6 получилось максимальное значение 1, а не 5.5, а в тесте №7 получился 0, т.к. число A < 0.

Добавим проверку на корректность введенных значений. Если TryParse возвращает true, значит, перевод был выполнен корректно, а если false – то некорректно.



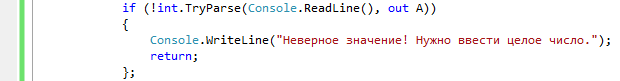
* Можно ли для проверки на корректность ввода сравнивать переменной A с нулем (A == 0)?

Однако сравнивать с true и false непринято. Это не ошибка, но просто лишнее действие: сам результат проверки уже является логическим значением. Правильнее будет использовать логическое НЕ (символ !):

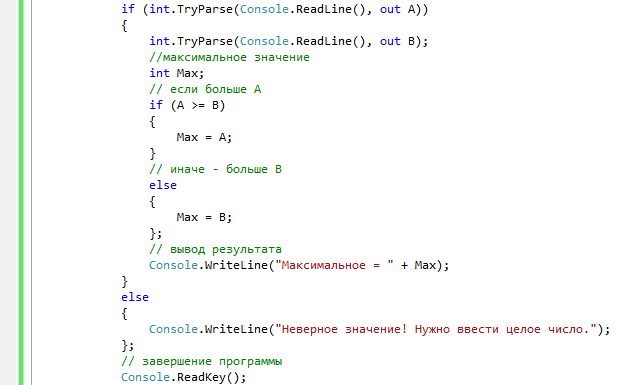


Если вы запустите программу, то увидите, что сообщение будет выведено, но программа продолжит работу. Т.е., нам необходимо выполнить все дальнейшие действия, если TryParse возвращает True, а в противном случае – только вывести сообщение и завершить программу.

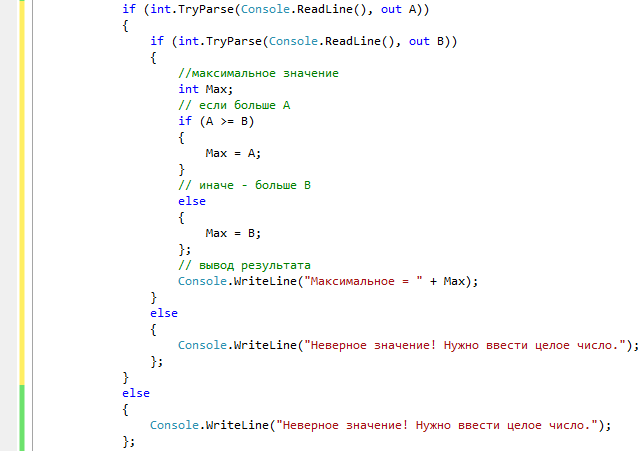
Для экстренного завершения работы можно использовать команду return, но тогда программа сразу закроется, и мы не успеем увидеть сообщение об ошибке. Кроме того, такой код считается «грязным», т.к. часто затрудняет чтение и анализ чужих программ.



Поэтому мы перенесем сообщение об ошибке в блок else, а остальную программу – внутрь блока if. В конце останется только нажатие клавиши.



Добавим аналогичную проверку для B.



* Как объединить проверки для A и B в один if? Будет ли при этом программа запрашивать ввод B, если A введено неверно?

Протестируйте работу программы.

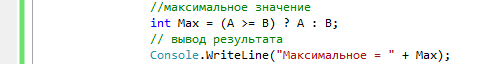
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Входные данные | | Выходные данные | |
| A | B | Max | Экран |
|  | 121 | 12 | 121 | Максимальное = 121 |
|  | 14 | 50 | 50 | Максимальное = 50 |
|  | 20 | 20 | 20 | Максимальное = 20 |
|  | -55 | -2 | -2 | Максимальное = -2 |
|  | aab |  |  | Неверное значение! Нужно ввести целое число. |
|  | 1 | 5.5 |  | Неверное значение! Нужно ввести целое число. |
|  | -1 | b |  | Неверное значение! Нужно ввести целое число. |

Выполните коммит получившейся программы с комментарием «Л2.П2. Проверка введенных значений».

Другой способ выполнения проверки – использования тернарной операции <условие> **?** <значение\_если\_true> **:** <значение\_если\_false> (тернарная – с тремя операндами).

Эту операцию можно использовать в вычислениях наравне с обычными операциями сложения, умножения и т.д. Однако, рекомендуется не злоупотребрять ей, чтобы не осложнить прочтение кода.

Тогда проверка значений A и B сократится до одной строчки:



Читается: если A больше или равно B, то присвоить A, иначе присвоить B.

Выполните коммит программы, в которой оператор if заменен на тернарный ?:, с комментарием «Л2.П2. Тернарный оператор ?:».

Расширим программу до сравнения трех переменных.

Начнем работу на основе предыдущего коммита «Л2.П2. Проверка введенных значений», где использовался if.

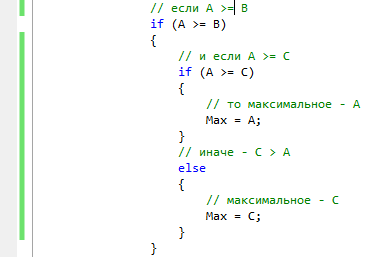
Создайте новую ветку из этого коммита: откройте &Kcy;&ncy;&ocy;&pcy;&kcy;&acy; log. Log, выделите нужный коммит и нажмите &Kcy;&ncy;&ocy;&pcy;&kcy;&acy; Add Branch. Add branch. Назовите ветку MaximumIf. Сделайте ветку активной (отслеживаемой, checkout) двойным кликом по ее имени в списке.

После этого файл с программой вернется к предыдущему протестированному варианту (если VisualStudio не выведет сообщение об обновлении файла программы автоматически, то закройте и откройте его).

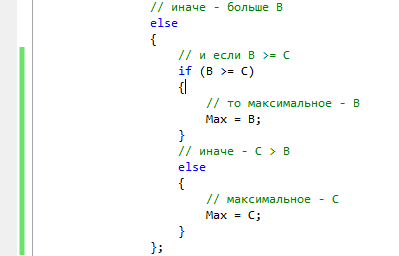
Добавьте ввод переменной C по аналогии с A и B.



Добавим проверку значений. Теперь, если A > B, его еще нужно сравнить с C:



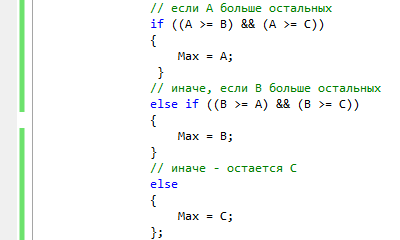
Аналогично для B:



Протестируйте правильность работы программы и выполните коммит в ветку MaximumIf с комментарием «Л2.П2 Максимальное из трех чисел через if».

Самостоятельно составьте таблицу с тестовыми значениями.

Другой вариант реализации проверок через логическое И в условиях:

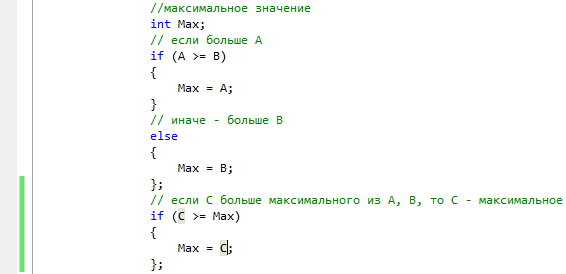


Этот код значительно короче и читабельнее, но немного затратнее по времени выполнения: если максимальное значение B или C, то необходимо выполнить четыре разных сравнения в двух if. А если максимальное значение A – то два сравнения в первом if. Причем (A >= B) в первом if и (B >= A) во втором фактически дублируют друг друга. В предыдущем варианте, какое бы число не было максимальным, всегда выполняется два сравнения.

При однократном запуске разница составит менее миллисекунды, но если этот код вызывается тысячи и более раз, разница станет заметной.

Выполните коммит в ветку MaximumIf с комментарием «Л2.П2 Максимальное из трех чисел через if с логическим И (&&)».

Наконец, третий вариант – сравнить C не напрямую с A и B, а с максимальным из них (максимальное из A и B мы уже находили):



В этом варианте будет всегда выполнено два сравнения, но и значение переменной Max будет присвоено дважды. Таким образом, этот код тоже немного медленнее первого варианта.

Выполните коммит в ветку MaximumIf с комментарием «Л2.П2 Максимальное из трех чисел через сравнение третьего с максимальным из первых двух».

После этого вернитесь на ветку master (сделайте ее активной, checkout), где мы остановились на сравнении двух чисел с помощью тернарной операции ?:.

Добавьте ввод значения C, как было показано ранее.

Реализуем поиск максимума по аналогии с последним вариантом if, когда мы сначала находим большее из A и B, а потом сравниваем его с C:



* Можно ли пропустить скобки в условии? А в if?

Выполните коммит в ветку master с комментарием «Л2.П2 Максимальное из трех чисел через операцию ?:».

Этот код можно сократить до одной строки. C# поддерживает операции присваивания «в середине» арифметического выражения. Поэтому первый раз можно присвоить значение Max прямо в проверке условия:



Выполните коммит в ветку master с комментарием «Л2.П2 Максимальное из трех чисел через операцию ?: в одну строку».

Выполните push обеих веток на GitHub. Объединять ветки не требуется.

Как видите, даже такую простую задачу можно решить множеством способов, отличающихся по длине кода, его читабельности и скорости выполнения.

* Есть ли еще варианты решения?

### Задание на самостоятельную работу

#### +Задание 3

Сумма покупки составляет Sum рублей. Если Sum больше 1000, то предоставляется скидка 15%. Вывести на экран сумму покупки с учетом скидки, либо сообщение о том, что скидка не предоставляется.

#### Задание 4

Пользователь вводит длины сторон треугольника a, b и c (не обязательно целые числа). Проверить:

1. Могут ли три введенных числа быть сторонами невырожденного треугольника? Если нет, вывести об этом сообщение и дальнейшие проверки не выполнять.
2. Является ли треугольник равнобедренным? равносторонним? Если является, то вывести об этом сообщение, если нет – ничего не выводить.
3. Является ли треугольник остроугольным, прямоугольным или тупоугольным? Вывести об этом сообщение.
4. Вычислить периметр и площадь треугольника. Если периметр больше 100 и площадь больше 200, то вывести сообщение о том, что треугольник большой. А если периметр меньше 3 или площадь меньше 1, то вывести сообщение о том, что треугольник маленький.

## Пример 3. Названия месяцев

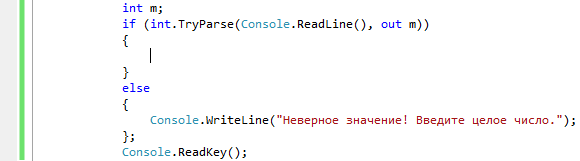
Написать программу, которая по введенному номеру месяца (от 1 до 12) выводит его название на русском языке.

### Указания к выполнению

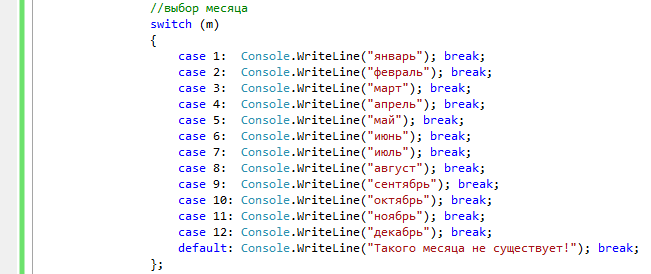
В данной программе воспользуемся оператором множественного выбора switch.

Создайте и сохраните новое консольное приложение с именем Months.

Организуйте ввод и проверку номера месяца по аналогии с предыдущим примером:



Выполните перебор номеров месяцев через Switch:



Протестируйте программу и выполните ее коммит (в ветку master) с комментарием «Л2.П3 Названия месяцев».

|  |  |  |
| --- | --- | --- |
| № | Входные данные | Выходные данные |
| m | Экран |
|  | 1 | январь |
|  | 2 | февраль |
|  | 3 | март |
|  | 4 | апрель |
|  | 5 | май |
|  | 6 | июнь |
|  | 7 | июль |
|  | 8 | август |
|  | 9 | сентябрь |
|  | 10 | октябрь |
|  | 11 | ноябрь |
|  | 12 | декабрь |
|  | 13 | Такого месяца не существует! |
|  | -1 | Такого месяца не существует! |

Данную программу также можно выполнить другими способами, например, через массивы. Но массивы мы будем рассматривать позже.

* Как будет работать программа, если убрать операторы break после каждого case?
* Как будет работать программа, если поставить ветвь default перед всеми case?

### Задание на самостоятельную работу

#### Задание 5

В программе константами заданы коэффициенты для перевода одной валюты в другую (рубли в евро, доллары, фунты, йены). Пользователь вводит сумму в рублях и выбирает, в какую валюту хочет их перевести. Программа переводит сумму в выбранную валюту с округлением до двух знаков после запятой.

*Дополнительное задание* (на «отлично»): добавить возможность выбора валюты, в которой вводится исходная сумма. Для упрощения расчетов можно сначала переводить сумму в рубли, а из рублей – в нужную валюту.

## Пример 4. Консольный калькулятор

Разработать консольный калькулятор, поддерживающий операции сложения, вычитания, умножения, деления, взятия квадратного корня.

### Указания к выполнению

Соединим первые три примера, чтобы получить возможность выбирать действие, которое нужно выполнить с введенными числами.

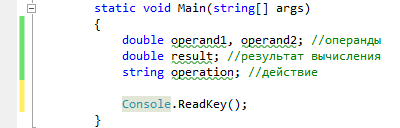
Для удобства пользователя следует организовать ввод данных в следующем порядке:

1. Считать первый операнд (слагаемое, множитель и т.п.).
2. Если первый операнд верный, то считать действие (операцию).
3. Если операция верная, то считать второй операнд.
4. Если третий операнд верный, то выполнить вычисление и вывести результат.
5. Завершить работу программы.

Однако для простоты реализации сначала мы реализуем программу, в которой сначала вводятся оба операнда, а потом операция.

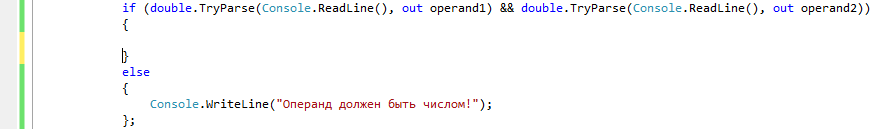
Создайте и сохраните новый проект с именем ConsoleCalc.

Сразу добавьте объявление переменных и паузу перед выходом из программы:

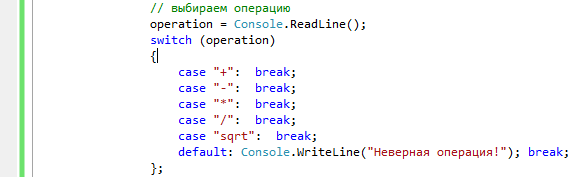


Переменные операндов объявлены как double, т.к. калькулятор должен работать не только с целыми числами (на это указывает наличие операций деления и извлечения квадратного корня).

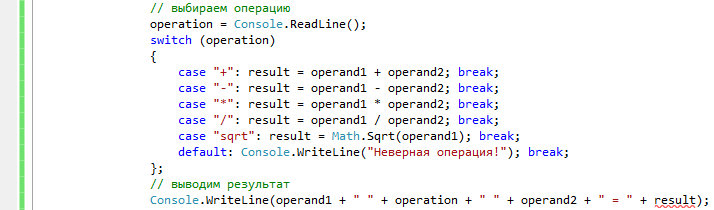
Добавьте ввод первого и второго операнда с проверкой правильности:



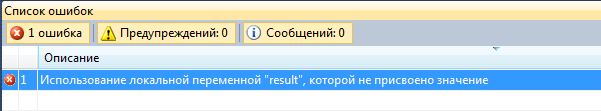
После этого считаем операцию и проверим ее через switch:



Добавьте вычисление и вывод результата по каждой операции:



Обратите внимание: в выводе результата result подчеркнуто красным. И дело не в опечатке: имя переменной написано верно. Нажмите F6 для компиляции проекта без запуска программы. Будет выведено окно с сообщением об ошибке:



Т.е. компилятор заметил, что если пройдем switch по ветке default, то в переменную result не будет записано никакое значение и ее нельзя будет вывести на экран.

Есть несколько вариантов решения этой проблемы:

1. Присвоить переменной result значение по умолчанию там, где мы ее объявили.



1. Присвоить переменной result значение в ветке default.



1. Добавить return в ветвь default, чтобы в этом случае программа прекратила работу.



В первом и втором случаях результат получится равным нулю, но программа не закроется и пользователь увидит сообщение о неверной операции. В третьем случае программа закроется прежде, чем пользователь успеет прочитать сообщение (если только он не вызвал программу из командной строки).

Выберите сами вариант, который вам больше нравится.

Выполните коммит с комментарием «Л2.П4 Консольный калькулятор: сначала вводим операнды, потом операцию».

Отметим существенный недостаток текущей последовательности ввода: для извлечения квадратного корня второй операнд не нужен, но мы запрашиваем его у пользователя.

Правильнее будет считывать второй операнд только после того, как мы убедились, что операция верная и требует второго операнда:

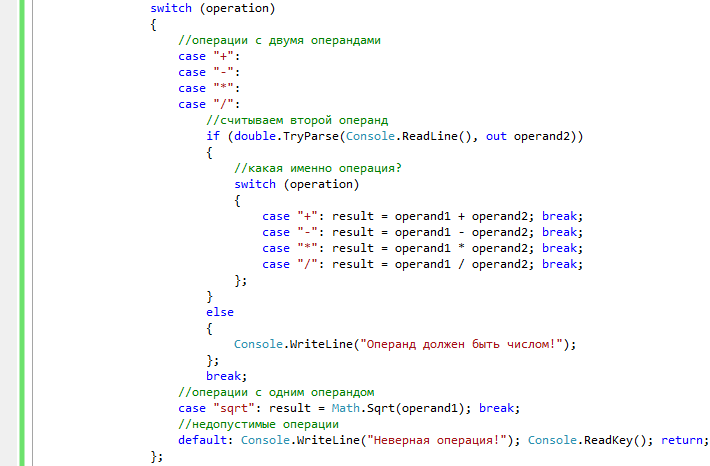


* Самостоятельно исправьте ошибку, возникающую при выводе operand2.

Код стал более громоздким и содержит много повторяющихся действий, что повышает риск опечаток (копировали, и где-то забыли поменять + на - или \*). А ведь еще надо вывести сообщение, если второй операнд неверный.

Воспользуемся вложенным switch, чтобы сократить код и убрать повторяющиеся операции.

Сначала проверим, требует ли операция ввода второго операнда. Если да, то считаем его, и уже тогда, через второй switch, вычислим результат.



Протестируйте работу программы.

* Какие еще следует добавить проверки, чтобы программа не вызывала ошибок при расчетах?

Выполните коммит с комментарием «Л2.П4 Консольный калькулятор: сначала вводим первый операнд, потом операцию, потом второй операнд».

### Задание на самостоятельную работу

#### Задание 6

Разработать консольный калькулятор для расчета процентов по кредиту. Пользователь вводит сумму и срок кредитования, годовую процентную ставку, частоту выплаты процентов (сколько раз в год) и метод начисления процентов (простые или сложные).

Программа выводит сумму начисленных процентов, итоговую сумму выплат по кредиту и величину эквивалентных выплат (равными частями в каждом периоде).

Расчеты округлять до копеек.

Пример расчета:

Сумма кредита: 100 000

Срок кредитования: 36 месяцев (3 года)

Годовая процентная ставка: 18%

Частота выплаты процентов: 12 раз в год

Метод начисления: простые проценты

По формуле простых процентов итоговая сумма выплат за три года:

100 000 \* (1 + 3 \* 0,18) = 100 000 \* 1,54 = 154 000

Из них начислено процентов:

154 000 – 100 000 = 54 000

Величина эквивалентных выплат:

154 000 / 3 / 12 = 4277,78

Аналогичный расчет для сложных процентов:

100 000 \* (1 + 0,18)3 = 100 000 \* 1,643032= 164 303,2

164 303,2 – 100 000 = 64 303,2

164 303,2 / 3 / 12 = 4563,98

*Дополнительное задание* (на «отлично»): реализовать возможность ввести два варианта кредитования. Сумма кредитования для обоих вариантов совпадает, но все условия могут различаться. Система выводит результаты расчета по обоим вариантам и рекомендует более выгодный вариант (с меньшей суммой выплат по процентам).

1. Здесь и далее в угловых скобках < > записывается то, что нужно подставить в шаблон, в квадратных скобках [ ] – часть, которую можно пропустить. Остальной текст пишется «как есть» [↑](#footnote-ref-1)