

## 1. Формируемые компетенции и индикаторы их достижения по дисциплине (модулю)

Код компетенции	Содержание компетенции	Код и наименование индикатора достижения компетенции
ОПК-7	Способен использовать языки программирования и технологии разработки программных средств для решения задач профессиональной деятельности;	ОПК-7.1 - Знает основные технологии разработки программных средств для решения задач в области профессиональной деятельности; ОПК-7.2 - Умеет применять языки программирования для решения профессиональных задач; ОПК-7.3 - Владеет навыками выбора и разработки алгоритмов при решении типовых задач программирования, а также навыками разработки и тестирования программ по поставленной спецификации.

## 2. Паспорт фонда оценочных средств по дисциплине (модулю)

№ п/п	Контролируемые разделы (темы) дисциплины	Код контролируемой компетенции (или ее части)	Наименование оценочного средства
<b>1 семестр</b>			
1.	Тема 1. Основные понятия языков программирования	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
2.	Тема 2. Синтаксис, семантика, формальные способы описания языков программирования	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
3.	Тема 3. Типы данных, способы и механизмы управления данными	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
4.	Тема 4. Динамическая память и указатели.	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
5.	Тема 5. Модульное программирование	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
6.	Тема 6. Создание динамических библиотек	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
7.	Тема 7. Основные понятия объектно-ориентированного программирования.	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
8.	Тема 8. Обработка исключений	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
9.	коллоквиумы	ОПК-7	контрольные вопросы, вопросы к коллоквиуму
10.	экзамен	ОПК-7	контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
<b>2 семестр</b>			
11.	Тема 1. Язык программирования C#. Пространство имен. Типы данных. Операции языка. Типы как классы.	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
12.	Тема 2. Операторы языка C#. Операторы помеченные (labeled-	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Сахалинский государственный университет»

Кафедра информатики

УТВЕРЖДЕН  
на заседании кафедры  
«19» марта 2024 г., протокол № 8  
Исполняющий обязанности  
заведующего кафедрой

\_\_\_\_\_

Осипов Г.С.

**ФОНД  
ОЦЕНОЧНЫХ СРЕДСТВ  
ПО ДИСЦИПЛИНЕ (МОДУЛЮ)**

**Б1.О.16 Языки и методы программирования**

Направление подготовки  
10.03.01 Информационная безопасность  
профиль  
Безопасность автоматизированных систем (по отрасли или в сфере профессиональной  
деятельности)

**Уровень высшего образования  
БАКАЛАВРИАТ**

Южно-Сахалинск  
2024 г.

	statement), декларирующие (declaration-statement), встроенные (embedded-statement).		коллоквиуму, вопросы к экзамену
13.	Тема 3. Массивы в C#. Массивы одномерные, многомерные. Массивы массивов. Прямоугольные массивы.	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
14.	Тема 4. Строки в C#. Строки как объекты класса string. Строка как контейнер. Применение строк в переключателях. Массивы строк. Операции над строками.	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
15.	ОПК-2, ОПК-4, ПКС-2, ПКС-4	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
16.	коллоквиумы	ОПК-7	контрольные вопросы, вопросы к коллоквиуму
17.	экзамен	ОПК-7	контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
<b>3 семестр</b>			
18.	Тема 1. Классы как типы. Объявление класса. Поля объектов. Методы объектов. Ссылка this. Конструкторы объектов класса (умолчания, общего вида, копирования, приведения). Деструкторы и финализаторы.	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
19.	Тема 2. Средства взаимодействия с объектами. Принцип инкапсуляции и методы объектов. Свойства классов. Автореализуемые свойства. Индексаторы.	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
20.	Тема 3. Включение, вложение и наследование классов. Включение объектов классов. Вложение классов. Наследование классов. Доступность членов класса при наследовании. Методы при наследовании. Абстрактные методы и абстрактные классы. Применение абстрактных классов. Опечатанные классы и методы	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
21.	Тема 4. Интерфейсы. Интерфейс как механизм наследования специфицированной функциональности. Наследование специфицированной функциональности. Реализация интерфейсов. Интерфейс как тип. Интерфейсы и наследование.	ОПК-7	Задания к лабораторным работам, контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену
22.	коллоквиумы	ОПК-7	контрольные вопросы, вопросы к коллоквиуму
23.	экзамен	ОПК-7	контрольные вопросы, вопросы к коллоквиуму, вопросы к экзамену

## 1 семестр

### Лабораторное занятие №1 (2 ч.)

#### Тема Среда визуального программирования Delphi

Вопросы для обсуждения:

1. Интерфейс, основные команды, палитра инструментов.
2. Структура программы (проекта): основные разделы, их назначение. Комментарии. Директивы компилятора.

Примерные задания:

**Задания выполняются по ходу объяснения материала**

**Простое консольное приложение.**

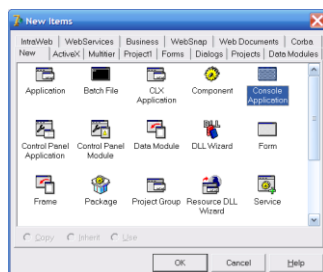
Программы, работающие в операционных системах семейства Windows, называют **приложениями**.

Delphi позволяет создавать приложения, в которых для ввода данных в оперативную память с клавиатуры используются процедуры `read` и `readln`, а для вывода результатов на экран – процедуры `write` и `writeln`. Такие приложения называют **консольными**.

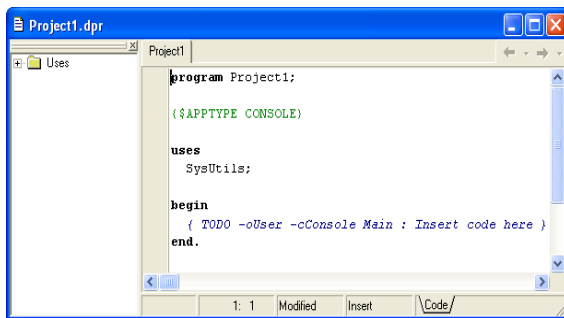
Создание консольного приложения может быть выполнено двумя способами:

#### 1 способ

1. После запуска среды Delphi в главном меню выберите **File\New\Other...** После этого откроется окно так называемого репозитория (архива) Delphi, предназначенного для накопления типовых форм и проектов.



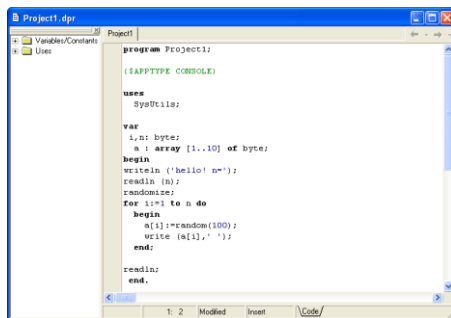
2. Выберите пиктограмму с подписью **"Console Application"**. После этого раскроется окно **фала проекта** (он имеет расширение `.dpr`) или другое название, **главного модуля проекта**:



3. Вместо комментария


*{ TODO -oUser -cConsole Main : Insert code here }*

наберите в окне файла проекта текст программы, которая формирует массив случайных натуральных чисел.



4. Перед выполнением программы необходимо ее сохранить. Каждый проект рекомендуется хранить в отдельной папке.

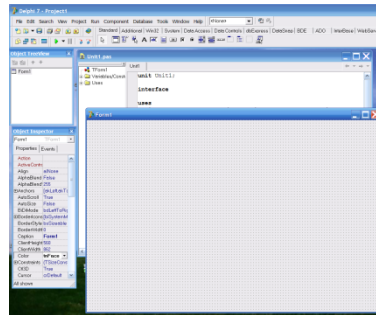
*Замечание.* По умолчанию файл проекта имеет имя ProjectN.dpr, а программа – ProjectN, где N – некоторое число. Мы можем сохранить файл проекта под любым именем, например, MyProgram.dpr. Автоматически Delphi изменить имя программы на MyProgram. Учитывая это, можно сформулировать правило: **те строки программного кода, которые формируются средой Delphi, редактировать нельзя.** Иначе работа будет осложнена различными неприятностями, например, сообщениями об ошибках.

5. После сохранения проекта, необходимо его выполнить. Это можно сделать, выполнив команду **Run\Run**, либо нажав **F9**, либо найти на панели инструментов кнопку .

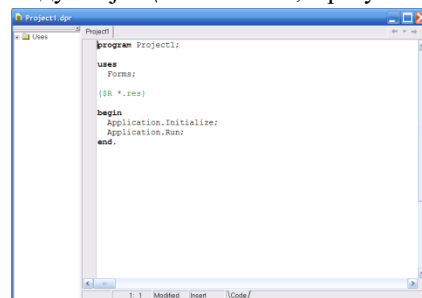
В конце текста программы консольного приложения необходимо использовать процедуру `readln` для того, чтобы увидеть результаты выполнения программы.

## 2 способ

1. После запуска Delphi необходимо закрыть окно формы – `Form1` и окно модуля приложения – окна с заголовком `Unit.pas`. При закрытии окон на запрос о сохранении изменений в `Unit.pas`, необходимо ответить "нет". В этом случае на экране останутся главное окно Delphi (с заголовком `Delphi 7 – Project1`), окно Инспектора объектов (с заголовком `Object inspector`) и окно Древа Объектов (с заголовком `Object TreeView`). Эти окна тоже можно закрыть.



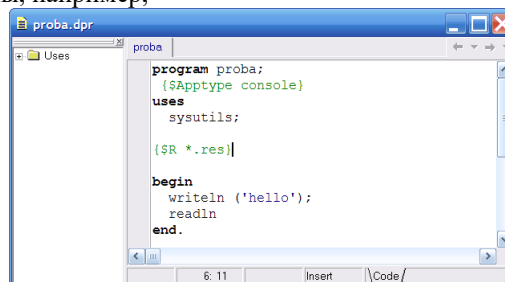
2. Выполнить команду **Project\View Source**, в результате откроется окно `Project1.dpr` файла проекта.



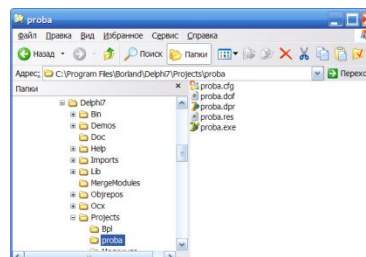
3. Сохранить файл проекта. Если имя файла будет изменено (например, на имя `proba`), это отразится в заголовке программы.

4. Убрать все строки, за исключением строк, содержащих ключевые слова `Program`, `Uses`, `Begin`, `End`.

5. Набрать текст программы, например,



Если просмотреть содержимое папки, в которой сохранялся проект, в ней будут содержаться следующие файлы:



- proba.dpr – файл проекта (главный модуль проекта);
- proba.exe – файл приложения или исполняемый файл. Он будет создан компилятором только при компиляции программы и при отсутствии синтаксических ошибок. Этот файл является автономным исполняемым файлом и для его работы не требуются никакие другие файлы;
- proba.cfg – файл конфигурации проекта. Он содержит установки проекта, например, используемые директивы компилятора;
- proba.dof – файл опций проекта. Он содержит установки опций проекта для правильной работы проекта в целом.

Файлы опций проекта и конфигурации проекта создаются Delphi автоматически, одновременно с созданием файла проекта.

Кроме перечисленных файлов, часто в папке можно обнаружить файлы с расширением `.~dpr`, например, `proba.~dpr`. Это резервная копия файла проекта. Она создается при редактировании существующего файла проекта и содержит предыдущий вариант текста программы. Резервная копия может пригодиться, если необходимо отказаться от внесенных в программу изменений. При этом достаточно убрать символ "~" в расширении имени файла.

### Отладка консольных приложений

Термин **отладка** означает исправление ошибок в программе и обеспечение ее правильной работы. Возникающие в процессе создания программы ошибки можно разделить на три класса:

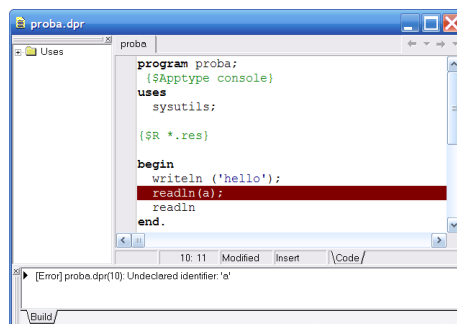
*синтаксические ошибки;*

*ошибки времени выполнения программы;*

*логические ошибки.*

Delphi позволяет легко найти и исправить ошибки, возникающие как во время компиляции (синтаксические ошибки), так и во время выполнения программы. В состав интегрированной среды разработчика Delphi входит мощный и гибкий отладчик, который позволяет построчно выполнять программу, анализируя при этом выражения и модифицируя значения переменных.

Ошибки компиляции подсвечиваются коричневым цветом, а в нижней части окна файла проекта появится сообщение об ошибке.



Если ошибок несколько, то исправлять следует, начиная с первой, поскольку часто одна ошибка является причиной появления других ошибок. Исправив первую, необходимо откомпилировать файл проекта еще раз. Во многих случаях исправление только одной ошибки существенно уменьшает общее количество сообщений об ошибках.

Кроме сообщений, начинающихся словом **[Error]** (Ошибка), компилятор может выдавать сообщения, начинающиеся словами **[Warning]** (Предупреждение) и **[Hint]** (Замечание). Предупреждения и замечания не являются ошибками и, не смотря на их наличие, компилятор **создаст** исполняемый модуль. Но, все же, следует внимательно изучить сделанные компилятором замечания и предупреждения, т.к. они направлены на улучшение программы.

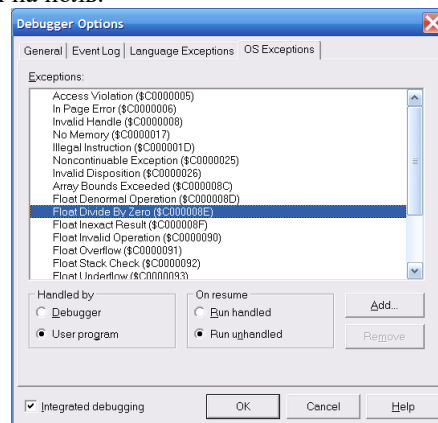
### Ошибки времени выполнения программы

Другой возможный тип ошибок – это ошибки этапа выполнения (или семантические ошибки). Такие ошибки могут возникать в случае, если во время компиляции **корректной** программы, предпринимается попытка выполнить недопустимое действие, например, открыть несуществующий файл для ввода, или выполнить недопустимое деление на ноль. В этом случае Delphi генерирует так называемое исключение (exception).

Консольное приложение, содержащее ошибку времени выполнения программы, имеет одну особенность: DOS-окно приложения, содержащее сообщение об ошибке, появляется на экране лишь на мгновение, поэтому прочитать сообщение практически невозможно.

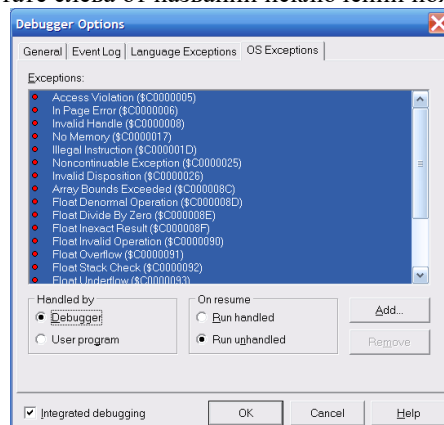
Для того чтобы увидеть сообщение о сгенерированном исключении, необходимо выполнить несложную настройку среды Delphi: выполнить команду Tools\Debugger Options... откроется окно Debugger Options, предназначенное для настройки параметров встроенного в среду Delphi отладчика.

Раскроем вкладку OS Exceptions. В окне Exceptions приведены названия различных исключений, которые может генерировать Delphi. Например, исключение Float Divide By Zero генерируется в том случае, если в программе вещественное число делится на ноль.



Ниже списка исключений находится группа переключателей **Handled by**, содержащая переключатели **Debugger** и **User program**. По умолчанию активен переключатель **User program**, обозначающий, что сгенерированные Delphi исключения должны обрабатываться программой, т.е. программист **должен предусмотреть** в программе операторы для обработки возможных исключительных ситуаций. Переключатель **Debugger** передает задачу обработки исключительных ситуаций отладчику (Debugger). В случае их возникновения на экран будет выдаваться сообщение о сгенерированном исключении.

Для того чтобы отладчик обрабатывал все исключения, необходимо их выделить мышью или с помощью клавиатуры в окне Exceptions. В результате слева от названий исключений появятся красные кружочки.



### Логические ошибки

Программа содержит логические ошибки, если реализованный в ней алгоритм не является верным.

В этом случае программа компилируется, но выдает **неверный** результат.

Для поиска и устранения логических ошибок необходимо использовать **тесты**, т.е. такие наборы исходных данных, при которых результат работы программы известен.

В больших и сложных программах логические ошибки и ошибки времени выполнения программы достаточно трудно отследить и обнаружить.

В этих случаях вполне естественно желание выполнить программу в **интерактивном** режиме, наблюдая за изменениями значений отдельных переменных или выражений.

При этом хорошо бы иметь возможность останавливаться в определенном месте программы и смотреть, что при этом происходит. Желательно также останавливаться и изменять значения некоторых переменных при выполнении программы. Это позволит повлиять на поведение программы.

В Delphi имеются возможности отладчика, являющегося составной частью интегрированной среды разработчика, которые позволяют быстро отредактировать, перекомпилировать и снова запустить программу.

Рассмотрим технологию использования отладчика Delphi на примере программы, вычисляющей  $n!$

```
program Project2;
{$APPTYPE CONSOLE}

uses
  SysUtils;

var
  n, i, fact : integer;

begin
```



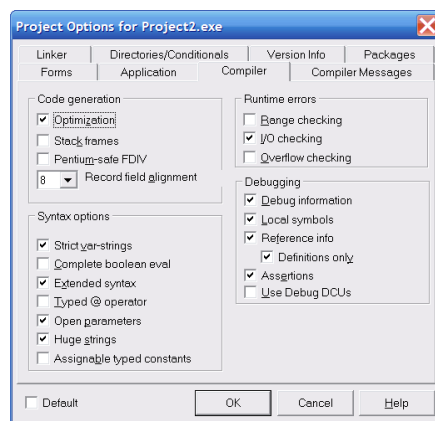
```

Write ('Enter n=');
readln (n);
fact:=1;
for i:= 2 to n do
    fact:= fact*i;
writeln ('n!=', fact);
readln;
end.

```

Прежде чем использовать отладчик, необходимо выполнить несложную настройку компилятора. Дело в том, что компилятор Delphi является оптимизирующим. Оптимизация направлена на повышение эффективности вашей программы. Например, иногда оптимизация позволяет сократить размер выполняемого файла. При этом компилятор удалит из текста программы "лишние", по его мнению, переменные или даже операторы. Побочным эффектом оптимизации является то, что она затрудняет процесс отладки. Поэтому перед отладкой оптимизирующее действие компилятора необходимо отключать. После завершения отладки окончательная версия программы должна быть откомпилирована с оптимизацией.

Для отключения оптимизации необходимо выполнить команду **Project – Options**, перейти на вкладку **Compiler** и на панели **Code generation** снять флаг выключателя **Optimization**. После этого можно приступить к оптимизации. Для возвращения в режим оптимизации необходимо добавить флаг выключателя **Optimization**.



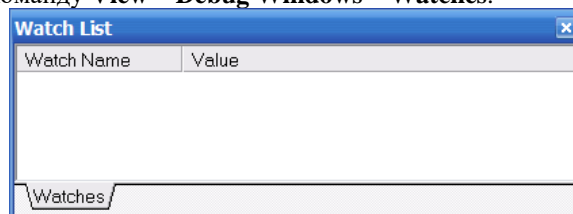
### *Отладка программы.*

**Трассировка программы** – это выполнение программы по шагам, оператор за оператором. Точнее, строка за строкой, т.к. в одной строке может располагаться несколько операторов.

В Delphi имеется два режима трассировки:

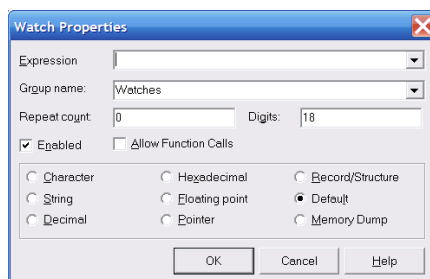
1. **трассировка без захода в процедуру (Run – Step Over** или по клавише **F8**). При этом трассировка подпрограмм не осуществляется, а подпрограмма воспринимается как один шаг алгоритма.
2. **трассировка с заходом в процедуру (Run – Trace into** или по клавише **F7**). При этом осуществляется трассировка не только основной программы, но и всех подпрограмм.

Для наблюдений за изменением значений переменных во время трассировки, необходимо открыть окно наблюдений **Watch List** и внести в него имена наблюдаемых переменных или выражений. Открыть окно наблюдений можно, выполнив команду **View – Debug Windows – Watches**.



Для добавления нового выражения или переменной необходимо щелкнуть по окну **Watch List** правой кнопкой и в открывшемся контекстном меню выбрать пункт **Add Watch...** (либо произвести двойной щелчок левой кнопкой мыши). После этого появится диалоговое окно **Watch Properties**.



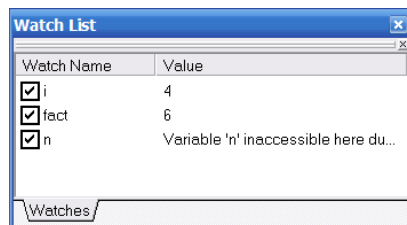


<b>Expression</b>	– ввод имен наблюдаемых переменных;
<b>Repeat Count</b>	– используется в случае наблюдения массивов и определяет количество показываемых элементов массива.
<b>Digits</b>	– поле задания количества значащих цифр при отображении вещественных значений.
<b>Enable</b>	–разрешить или запретить вывод в окно наблюдений значение соответствующей переменной или выражения.
<b>Allow Function Calls</b>	– разрешить или запретить помещать в окно наблюдения выражения, содержащие вызовы функций.

Содержащиеся в нижней части окна переключатели нужны для определения вида представления значений различных типов.

Для тестирования программы нахождения факториала числа внесем в окно наблюдения имена переменных  $n$ ,  $i$ ,  $fact$ . Для этого придется трижды открыть окно **Watch Properties** и последовательно ввести имена всех трех переменных.

Окно **Watch List** перетаскать, например, использовать его для Осуществляя наблюдаемых



является встраиваемым. Т.е. его можно на окно Инспектора Объектов и тем самым удобного размещения окон на экране. трассировку программ, видим значения переменных.

Если необходимо выполнить трассировку **части** программы, то необходимо установить в редакторе кода курсор на тот оператор, с которого следует начать трассировку и выполнить команду **Run – Run to cursor (F4)**. Затем продолжить трассировку клавишами F7 или F8.

При отладке программы можно использовать так называемые **точки останова (breakpoints)**. Точками останова программы называют некоторые, специальным образом помеченные строки программы. При достижении точки останова программа приостанавливает свою работу. В этот момент программист может просмотреть значения наблюдаемых переменных или начать трассировку переменных.

Для того чтобы добавить точку останова, необходимо выполнить команду **View – Debug Windows – Breakpoints**.

## Лабораторное занятие №2 (4 ч.)

Тема **Синтаксис, семантика, формальные способы описания языков программирования**

Вопросы для обсуждения:

1. Простые типы данных: стандартные скалярные и пользовательские.
2. Конструкции языков программирования.
3. Операторы языка.

*Примерные задания*

1. Вычислить периметр и площадь прямоугольного треугольника, если заданы длины его катетов, предполагая, что такой треугольник существует.
2. Треугольник задан координатами своих вершин A ( $x_1, y_1$ ), B ( $x_2, y_2$ ), C ( $x_3, y_3$ ). Найти площадь треугольника, предполагая, что такой треугольник существует.
3. Даны положительные действительные числа  $a, b, c$ . Выяснить, существует ли треугольник с длинами сторон  $a, b, c$ . Если существует, определить его тип по углам (остроугольный, прямоугольный, тупоугольный).
4. Поле шахматной доски представляется парой натуральных чисел, каждое из которых не превосходит восьми: первое число – номер вертикали (при счете снизу-вверх), второе – номер горизонтали (при счете слева направо). Даны натуральные числа  $k, l, m, n$ , каждое из которых не превосходит восьми. Требуется:
  1. выяснить, являются ли поля  $(k, l)$  и  $(m, n)$  полями одного цвета;
  2. на поле  $(k, l)$  расположен ферзь. Угрожает ли он полю  $(m, n)$ ?
  3. на поле  $(k, l)$  расположен конь. Угрожает ли он полю  $(m, n)$ ?
5. Алгоритм Евклида нахождения наибольшего общего делителя (НОД) неотрицательных целых чисел основан на следующих свойствах этой величины. Пусть  $m$  и  $n$  – одновременно не равные нулю целые неотрицательные числа и пусть  $m \geq n$ . Тогда, если  $n$  равно 0, то  $\text{НОД}(m, 0) = m$ , а если  $n \neq 0$ , то для чисел  $m, n$

и  $г$ , где  $г$  – остаток от деления  $m$  на  $n$ , выполняется равенство  $\text{НОД}(m, n) = \text{НОД}(n, г)$ . Например,  $\text{НОД}(15, 6) = \text{НОД}(6, 3) = \text{НОД}(3, 0) = 3$ .

Даны натуральные числа  $m, n$ . а) Используя алгоритм Евклида, найти наибольший общий делитель  $m$  и  $n$ .

б) Найти наименьшее общее кратное  $m, n$ , используя алгоритм Евклида.

6. *Гусеница на резине.* Гусеница ползет со скоростью 1 см/мин по куску резины, стремясь достичь противоположного конца. Кусок резины имеет длину 7 см и может растягиваться до любой длины. Каждую минуту резину растягивают на 7 см. Гусеница прочно держится на поверхности и продолжает двигаться, когда резина растягивается. Доберется ли гусеница до противоположного конца? Если да, то когда?

### Лабораторное занятие №3 (8 ч.)

Тема **Типы данных, способы и механизмы управления данными.**

Вопросы для обсуждения:

1. Структурированный тип данных: массив (статические, динамические, параметры-массивы).
2. Алгоритмы информационного поиска и сортировки)
3. Структурированный тип данных: строки (статические, динамические).
4. Структурированные тип данных: множества, записи, файлы (типизированные, нетипизированные, текстовые). Способы описания, основные процедуры и функции обработки.
5. Процедуры и функции, определяемые пользователем. Механизм передачи параметров. Рекурсивные подпрограммы.

*Примерные задания*

1. Последовательность чисел Фибоначчи  $u_0, u_1, \dots, u_n$  образуется по закону  $u_0=0, u_1=1, u_i = u_{i-2} + u_{i-1}$  ( $i=2, 3, \dots$ ). Дано натуральное число  $n>0$ . Получить  $n$  первых членов последовательности чисел Фибоначчи.
2. Дано натуральное число  $n$ , действительные числа  $x, a_0, a_1, \dots, a_n$ . вычислить, используя схему Горнера, значение многочлена  $a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0$ .
3. Вычислить с заданной точностью константу  $\pi$ , используя бесконечный

$$\text{Ряд Шарпа} \quad (1699 \text{ г.}): \quad 2\sqrt{3} \left( 1 - \frac{1}{3^1 \cdot 3} + \frac{1}{3^2 \cdot 5} - \frac{1}{3^3 \cdot 7} + \frac{1}{3^4 \cdot 9} - \dots \right).$$

$$\text{Ряд Лейбница} \quad (1673 \text{ г.}): \quad \frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots;$$

$$\text{Ряд Эйлера} \quad (1736 \text{ г.}): \quad \frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \dots;$$

Сравнить полученные значения, в качестве критерия сравнения использовать количество членов ряда, необходимых для вычисления числа  $\pi$ .

4. Вычислить цепные дроби ( $x \neq 0$ ).

а)	б)
$1 + \frac{1}{3 + \frac{1}{5 + \frac{1}{\dots}}} \\ 101 + \frac{1}{103}$	$x^2 + \frac{x}{x^2 + \frac{2}{x^2 + \frac{4}{x^2 + \frac{8}{\dots}}}} \\ x^2 + \frac{256}{x^2}$

5. Последовательность Хейеса. Рассмотрим некоторое натуральное число  $n$  ( $n>1$ ). Если оно четно, разделим его на 2, иначе умножим на 3 и прибавим 1. Если полученное число не равно 1, то повторяется то же действие и т.д., пока не получится 1. назовем вершиной наибольшее число в полученной при этом последовательности. Для заданного числа построить указанную последовательность, подсчитать число шагов и определить вершину.
6. Вывести все пары двузначных натуральных чисел таких, что значение их произведения не изменится, если в каждом сомножителе поменять местами цифры. Определить количество таких пар. Пример такой парой являются числа 27 и 72 (тривиальная пара). Все тривиальные пары не выводить.

- Найти пять наименьших натуральных чисел  $N$  таких, что  $N^2 = A^2 + B^2 + C^2$ , где  $A, B$  и  $C$  неравные друг другу натуральные числа.

## Лабораторное занятие №4 (6 ч.)

### Тема Типы данных, способы и механизмы управления данными.

Вопросы для обсуждения:

- Структурированный тип данных: массив (статические, динамические, параметры-массивы).
- Алгоритмы информационного поиска и сортировки)
- Структурированный тип данных: строки (статические, динамические).
- Структурированные тип данных: множества, записи, файлы (типизированные, нетипизированные, текстовые). Способы описания, основные процедуры и функции обработки.
- Процедуры и функции, определяемые пользователем. Механизм передачи параметров. Рекурсивные подпрограммы.

### Примерные задания

#### Тема "Строки"

- Задана строка-предложение. Необходимо подсчитать количество слов, удалив все лишние пробелы.
- Подсчитать в заданной строке количество всех символов  $B$  и удалить из нее те символы  $B$ , которым предшествуют (в исходной строке) символы  $A$ .
- Задана строка, внутри которой слова разделены одним пробелом. Исключить из нее группы символов, расположенные между круглыми скобками. Сами скобки так же исключить, а оставшиеся слова разделить только одним пробелом. Предполагается, что внутри каждой пары скобок других скобок нет.
- Определить, является ли заданная строка (фраза) палиндромом. Палиндромом называется слово, фраза или стих, одинаково читающиеся слева направо и справа налево. Строку, последовательность  $z_1, \dots, z_n$ , будем называть палиндромом, если без учета пробелов  $z_1 = z_n, z_2, \dots, z_{n-1}$  и т. д.
- В строке могут содержаться круглые, квадратные и фигурные скобки – как открывающие, так и закрывающие. Проверить баланс скобок в заданной строке. Считать, что он соблюдается, если выполнены следующие условия:
  - Для каждой открывающей скобки справа от нее есть соответствующая закрывающая скобка и наоборот;
  - Соответствующие пары скобок разных типов правильно вложены друг в друга.
- Дана некоторая строка. Известно, что она закодирована с помощью циклического сдвига:  $A \rightarrow B \rightarrow V \rightarrow \dots \rightarrow Y \rightarrow A$  на произвольное количество сдвигов. Известно, что в исходной строке присутствует ключевое слово ВРЕМЯ. Требуется расшифровать заданную строку.
- Известно, что астрологи делят год на 12 периодов и каждому из них ставят в соответствие один из знаков зодиака.

1. 20.01-18.02 – Водолей	2. 21.05-21.06 – Близнецы	3. 23.09-22.10 – Весы
4. 19.02-20.03 – Рыбы	5. 22.06-22.07 – Рак	6. 23.10-22.11 – Скорпион
7. 21.03-19.04 – Овен	8. 23.07-22.08 – Лев	9. 23.11-21.12 – Стрелец
10. 20.04-20.05 – Телец	11. 23.08-22.09 – Дева	12. 22.12-19.01 – Козерог

Используя таблицу определить по введенной дате (в формате день и месяц через пробел) знак Зодиака.

- В одну и ту же переменную  $X$  вводятся вещественные числа по модулю больше 1 и меньше 2. Количество вводимых чисел заранее не известно. Для каждого значения  $X$  вывести на экран сумму ряда, значение последнего слагаемого и его порядковый номер. Суммирование выполнять до тех пор, пока модуль разности между текущим и предыдущим членами остается больше 0.0001.

$$1 + \frac{x}{4} + \frac{x^2}{18} + \frac{x^3}{96} + \frac{x^4}{600} + \dots$$

#### Тема "Одномерные массивы"

- Дано натуральное число  $n$ , целые числа  $a_1, a_2, \dots, a_n$ . Подсчитать сколько раз встречается в этой последовательности максимальное по величине число. Поиск максимального и подсчет их количества произвести в одном цикле.
- Дано натуральное число  $n$ , целые числа  $x, a_1, a_2, \dots, a_n$ . Определить, каким по счету в последовательности  $a_1, a_2, \dots, a_n$  идет элемент, равный  $x$ . Если такого члена в последовательности нет, то предусмотреть соответствующее сообщение.
- Задача аналогична задаче №19, только поиск ведется в упорядоченной последовательности.
- В массиве  $C[m]$  каждый третий элемент заменить полусуммой двух предыдущих, а стоящий перед ним – полусуммой двух соседних с ним элементов. Дополнительный массив не использовать.

13. В массиве  $B[l]$  найти число чередований знака, т.е. число переходов с минуса на плюс или с плюса на минус. Например, в последовательности 0, -2, 0, -10, 2, -1, 0, 3, 2, -3 четыре чередования (ноль не имеет знака).
14. В массиве  $B[l]$  каждый элемент, кроме первого, заменить суммой всех предыдущих элементов.
15. Дано натуральное число  $n$ , целые числа  $a_1, a_2, \dots, a_n$ . Подсчитать наибольшее число одинаковых, подряд идущих чисел.
16. Дано натуральное число  $n$ , действительные числа  $a_1, a_2, \dots, a_n$ . Переставить члены последовательности так, чтобы сначала расположились все ее неотрицательные члены, потом – все отрицательные. Порядок как среди неотрицательных членов, так и среди отрицательных должен быть сохранен прежним.

### Тема "Процедуры, функции. Рекурсивный вызов подпрограмм"

17. В  $n$ -значном числе  $M$  посчитать количество четных цифр. Определение четности цифры описать с помощью функции.
18. Возведение любого числа в любую степень оформить в виде функции.
19. "Ханойские башни". Разместить некоторое число упорядоченных по убыванию дисков разного размера, нанизанных на вертикальный стержень 1, с помощью стержня 2 и 3. Диски перемещаются согласно правилам:
  1. *перекладывать диски можно только по одному;*
  2. *нельзя класть больший диск на меньший.*

### Тема «Тип данных множество»

20. Решето Эратосфена. Дано натуральное число  $n$  ( $n \geq 2$ ). Найти все меньшие  $n$  простые числа, используя решето Эратосфена. Решето Эратосфена называют следующим способом. Выпишем все подряд целые числа от 2 до  $n$ . Первое простое число 2. Подчеркнем его, а все большие числа, кратные 2, зачеркнем. Первое из оставшихся чисел – 3. Подчеркнем его как простое, а все большие числа, кратные 3 зачеркнем. Первое из оставшихся теперь 5, т.к. 4 уже зачеркнуто. Подчеркнем его как простое, а все большие числа, кратные 5 зачеркнем и т.д.
21. Расшифровать равенство  $\overline{ХОД} + \overline{ХОД} + \overline{ХОД} = \overline{МАТ}$ , в котором различным буквам соответствуют различные цифры.

### Тема «Тип данных запись»

22. Багаж пассажира характеризуется количеством вещей и общим весом вещей. Имеется информация о багаже нескольких пассажиров – соответствующие пары чисел. Подсчитать общее количество вещей и выяснить, имеется ли пассажир, багаж которого состоит из одной вещи весом не менее 30 кг.
23. Имеются сведения о студентах:
  - 1) фамилия, имя, отчество;
  - 2) пол (определяется символами «м», «ж»);
  - 3) результаты трех экзаменов в последнюю сессию.
 Найти средний бал группы из  $N$  студентов и процентное соотношение отличников мужского и женского пола.

### Тема «Тип данных файл»

24. Необходимо написать программу, в которой возможно переименовать или удалить файл на диске? А также просмотреть и изменить его атрибуты.
25. Сведения об автомобиле состоят из его марки, номера машины и фамилии владельца. Дан файл  $f$ , содержащий сведения о нескольких автомобилях. Найти фамилии владельцев и номера автомобилей данной марки.
26. Дан текстовый файл. Получить все его строки на экране, указав длину каждой строки, обозначив пустые строки фразой «Пустая строка» и подсчитав общее количество строк в файле.
27. Дан текстовый файл. Он состоит из предложений, оканчивающихся точкой, восклицательным или вопросительным знаком (в текст они больше не входят) и содержит не более 255 символов. Разделительными знаками в тексте служит пробел, двоеточие, запятая, точка с запятой, круглые скобки, кавычки, тире, дефис (дефис – это знак переноса или соединительный знак составного слова). Найти предложение, в котором больше всего слов, и подсчитать их количество.
28. Студентам 5 курса была предложена анкета о том, какие из прочитанных курсов могут потребоваться в будущей работе. Результаты анкетирования были занесены в файл. Напишите программу, которая: 1) создает файл анкетирования. Список прочитанных курсов: математический анализ, алгебра, геометрия, численные методы, программирование, дифференциальные уравнения; 2) определяет, сколько студентов назвали его полезным; 3) выводит название предмета, получившего наибольшую оценку (если таких несколько – выводит все); 3) определяет, сколько студентов не оценили положительно ни один из предметов и сколько оценили все предметы как полезные.

## Лабораторное занятие №4 (6 ч.)

### Тема Динамическая память и указатели. Динамические структуры данных.

Вопросы для обсуждения:

1. Ссылочные типы и указатели.
2. Представление динамических структур с помощью указателей.

#### *Примерные задания*

1. Представление динамических структур (стек, очередь) с помощью указателей. Реализовать основные операции над ними.
2. Реализация связного списка с помощью указателей.
3. Сформировать стек, в который помещаются целые положительные числа, вводимые с клавиатуры. Процесс ввода прекращается, как только среди вводимых чисел появляется отрицательное число. Вывести содержимое стека в порядке, обратном их вводу.
4. Организовать очередь из  $n$  целых чисел. Изменить ссылки так, чтобы последний элемент очереди стал первым, первый – вторым, второй – третьим, и т.д.
5. Сформировать очередь, содержащую целые числа. Упорядочить элементы очереди, не перемещая их.
6. Задача Иосифа Флавия. Вокруг считающего стоят  $n$  человек, один из которых назван первым, а остальные занумерованы стрелки числами от 2 до  $n$ . Считающий ведет счет до  $k$ , начиная с первого. Счет продолжается со следующего человека (при этом выбывшие из круга не считаются), и так до тех пор, пока не останется один человек. Требуется определить начальный номер этого человека.
7. Два упорядоченных по возрастанию списка слить в один также упорядоченный по возрастанию:
  - а) построив новый список;
  - б) поменяв соответствующим образом указатели в исходных списках.
8. Индивидуальные задания (11 вариантов)

### **Лабораторное занятие №5 (4 ч.)**

#### **Тема Модульное программирование.**

Вопросы для обсуждения:

1. Общая структура модуля.
2. Подпрограммы в модулях.
3. Компиляция и использование модулей.
4. Типы модулей.
5. Создание приложений, содержащих несколько форм.

#### *Примерные задания*

1. Создать модуль для работы с комплексными числами.
2. Создать модуль, в котором содержатся функции возведения в любую степень любого числа, нахождения НОД и НОК двух натуральных чисел, ввод-вывод массивов (одно- и двумерных).
3. Индивидуальные задания.

### **Лабораторное занятие №6 (4 ч.)**

#### **Тема Создание динамических библиотек**

Вопросы для обсуждения:

1. Особенности разработки динамических библиотек.

#### *Примерные задания*

1. Создать модуль и динамическую библиотеку для работы с комплексными числами.
2. Индивидуальные задания (20 вариантов).

### **Лабораторное занятие №7 (6 ч.)**

#### **Тема Основные понятия объектно-ориентированного программирования**

Вопросы для обсуждения:

1. Парадигмы объектно-ориентированного программирования: инкапсуляция, полиморфизм, наследование.
2. Понятие класса, объекта.
3. Поля, методы. Конструкторы, деструкторы, свойства.
4. Представление объекта в памяти.

#### *Примерные задания*

1. Создать класс для работы с комплексными числами.
2. Индивидуальные задания (20 вариантов)

### **Лабораторное занятие №8 (2 ч.)**

#### **Тема Обработка исключений**

1. Использование классов общего назначения.
2. Класс исключений. Защищаемые блоки.

### 3. Создание собственных исключений

#### Примерные задания

1. В индивидуальных заданиях по теме Классы предусмотреть исключительные ситуации, запрограммировать реакцию на них. Определить выброс собственных исключений.

## 2 семестр

### Лабораторное занятие №1 (4 ч.)

#### Тема Язык программирования C#.

Вопросы для обсуждения:

1. Интерфейс, основные команды среды разработки MS Visual Studio на платформе .NET Framework.
2. Структура (проекта): основные разделы, их назначение.
3. Пространства имен. Типы данных.
4. Операции языка. Типы как классы.
5. Парадигмы объектно-ориентированного программирования в C#. Особенности реализации.

#### Примерное занятие

#### Задания выполняются по ходу объяснения материала

<https://msdn.microsoft.com><sup>1</sup>

#### Интегрированная среда разработки Visual Studio

Microsoft Visual Studio — это **набор инструментов** для создания программного обеспечения от планирования до разработки пользовательского интерфейса, написания кода, тестирования, отладки, анализа качества кода и производительности, развертывания в средах клиентов и сбора данных телеметрии по использованию. Эти инструменты предназначены для максимально эффективной совместной работы; все они доступны в интегрированной среде разработки (IDE – Integrated Development Environment) Visual Studio.

Visual Studio можно использовать для создания различных типов приложений – от простых приложений, игр для мобильных клиентов до больших и сложных систем, обслуживающих предприятия и центры обработки данных.

В Visual Studio можно создавать:

1. приложения и игры, которые выполняются не только на платформе Windows, но и на Android и iOS;
2. веб-сайты и веб-службы на основе ASP.NET, JQuery, AngularJS и других популярных платформ;
3. приложения для самых разных платформ и устройств, включая, но не ограничиваясь: Office, Sharepoint, Hololens, Kinect и "Интернета вещей";
4. игры и графические приложения для разных устройств Windows, включая Xbox, с поддержкой DirectX.

По умолчанию Visual Studio является **открытой** языковой средой и обеспечивает поддержку C#, C и C++, JavaScript, F# и Visual Basic. Visual Studio хорошо работает и интегрируется со сторонними приложениями, например, Unity и Apache Cordova с помощью расширений. Имеется возможность самостоятельно расширить Visual Studio, создав собственные инструменты для выполнения специализированных задач.

Открытость среды **не означает** полной свободы. Все разработчики компиляторов при включении нового языка в среду разработки должны следовать определенным ограничениям. Главное ограничение, которое можно считать и главным достоинством, состоит в том, что все языки, включаемые в среду разработки Visual Studio должны использовать единую программную платформу (каркас) – Framework .Net. Благодаря этому достигаются многие желательные свойства:

- легкость использования компонентов, разработанных на различных языках;
- возможность разработки нескольких частей одного приложения на разных языках программирования;
- возможность описания класса на одном языке, а его потомков — на других языках.

Единая программная платформа приводит к сближению языков программирования, позволяя вместе с тем сохранять их индивидуальность и имеющиеся у них достоинства. Преодоление языкового барьера – одна из важнейших задач современного мира. Благодаря единой программной платформе Visual Studio.Net в определенной мере решает эту задачу в мире программистов.

.Net Framework представляет единую программную платформу среды разработки, в которой можно выделить два основных компонента:

- **статический** – FCL (Framework Class Library) – библиотеку классов платформы;
- **динамический** – CLR (Common Language Runtime) – общезыковую исполнительную среду.

#### Библиотека классов FCL – статический компонент каркаса

---

<sup>1</sup> MSDN — это веб-сайт технической документации Майкрософт. В Visual Studio можно нажать клавишу **F1**, чтобы перейти на страницу справки MSDN для активного окна. Кроме того, можно нажать клавишу **F1** в редакторе кода, чтобы перейти на страницу справки MSDN для API или ключевого слова, в котором в данный момент находится курсор.

Понятие платформы приложений – Framework Applications появилось достаточно давно, по крайней мере оно широко использовалось еще в четвертой версии Visual Studio. Несмотря на то, что каркас был представлен только статическим компонентом, уже тогда была очевидна его роль в построении приложений. Уже в то время важнейшее значение в библиотеке классов MFC<sup>2</sup> имели классы, задающие архитектуру строящихся приложений. Когда разработчик выбирал один из возможных типов приложения, например архитектуру Document-View, то в его приложение автоматически встраивались класс Document, задающий структуру документа, и класс View, задающий его визуальное представление. Класс Form и классы, задающие элементы управления, обеспечивали единый интерфейс приложений. Выбирая тип приложения, разработчик изначально получал нужную ему функциональность, поддерживаемую классами платформы. Библиотека классов поддерживала и более традиционные для программистов классы, задающие расширенную систему типов данных, в частности, динамические типы данных – списки, деревья, коллекции, шаблоны.

За счет появления динамического компонента CLR (Common Language Runtime) – общезыковой исполнительной среды роль программной платформы в построении приложений существенно возросла.

### ***Единство платформы***

Платформа стала единой для всех языков среды т.е., на каком бы языке программирования не велась разработка, она использует классы одной и той же библиотеки. Многие классы библиотеки, составляющие общее ядро, используются всеми языками. Отсюда единство интерфейса приложения, на каком бы языке оно не разрабатывалось, единство работы с коллекциями и другими контейнерами данных, единство связывания с различными хранилищами данных и другая универсальность.

### ***Встроенные примитивные типы***

Важной частью библиотеки FCL<sup>3</sup> (Framework Class Library) стали классы, задающие примитивные типы, те типы, которые считаются встроенными в язык программирования. Типы платформы покрывают все множество встроенных типов, встречающихся в языках программирования. Типы языка программирования проецируются на соответствующие типы платформы. Тип, называемый в языке Visual Basic – Integer, а в языке C# – int, проецируется на один и тот же тип платформы System.Int32. В каждом языке программирования, наряду с "родными" для языка названиями типов, разрешается пользоваться именами типов, принятыми в .Net Framework. Поэтому, по сути, все языки среды разработки могут пользоваться единой системой встроенных типов, что способствует облегчению взаимодействия компонентов, написанных на разных языках.

### ***Структурные типы***

Частью библиотеки стали не только простые встроенные типы, но и структурные типы, задающие организацию данных – строки, массивы, перечисления, структуры (записи). Это также способствует унификации и реальному сближению языков программирования.

### ***Архитектура приложений***

Существенно расширился набор возможных архитектурных типов построения приложений. Помимо традиционных Windows- и консольных приложений, появилась возможность построения Web-приложений. Большое внимание уделяется возможности создания повторно используемых компонентов – разрешается строить библиотеки классов, библиотеки элементов управления и библиотеки Web-элементов управления. Популярным архитектурным типом являются Web-службы, ставшие одним из основных видов повторно используемых компонентов. Для языков C#, J#, Visual Basic, поддерживаемых Microsoft, предлагается одинаковый набор из 12 архитектурных типов приложений. Несколько особняком стоит Visual C++, сохраняющий возможность работы не только с библиотекой FCL, но и с библиотеками MFC и ATL<sup>4</sup>, и с построением соответствующих MFC и ATL-проектов. Компиляторы языков, поставляемых другими фирмами, создают проекты, которые удовлетворяют общим требованиям среды, сохраняя свою индивидуальность. Так, например, компилятор Eiffel допускает создание проектов, использующих как библиотеку FCL, так и собственную библиотеку классов.

### ***Модульность***

Число классов библиотеки FCL велико (несколько тысяч). Поэтому понадобился способ их структуризации. Логически классы с близкой функциональностью объединяются в группы, называемые **пространством имен** (Namespace). Для динамического компонента CLR физической единицей, объединяющей классы и другие ресурсы, является сборка (**assembly**).

Основным пространством имен библиотеки FCL является пространство System, содержащее как классы, так и другие вложенные пространства имен. Так, например, примитивный тип Int32 непосредственно вложен в пространство имен System и его полное имя, включающее имя пространства – System.Int32.

---

<sup>2</sup> Пакет Microsoft Foundation Classes (MFC) — библиотека на языке C++, разработанная Microsoft и призванная облегчить разработку GUI-приложений для Microsoft Windows путём использования богатого набора библиотечных классов.

<sup>3</sup> FCL - Framework Class Library – библиотека базовых классов, используется в языках платформы Microsoft .net, содержит более 7000 классов.

<sup>4</sup> ATL – Active Template Library. Это библиотека классов и шаблонов, предназначенная для разработки собственных компонентов.



В пространство System вложен целый ряд других пространств имен. Например, в пространстве System.Collections находятся классы и интерфейсы, поддерживающие работу с коллекциями объектов – списками, очередями, словарями. В пространство System.Collections, в свою очередь, вложено пространство имен Specialized, содержащие классы со специализацией, например, коллекции, элементами которых являются только строки. Пространство System.Windows.Forms содержит классы, используемые при создании Windows-приложений. Класс Form из этого пространства задает форму – окно, заполняемое элементами управления, графикой, обеспечивающее интерактивное взаимодействие с пользователем.

### **Общезыковая исполнительная среда CLR – динамический компонент каркаса**

Наиболее революционным изобретением .Net Framework явилось создание исполняющей среды CLR. С ее появлением процесс написания и выполнения приложений становится принципиально другим.

#### ***Двухэтапная компиляция. Управляемый модуль и управляемый код***

Компиляторы языков программирования, включенные в Visual Studio .Net, создают модули на промежуточном языке MSIL (Microsoft Intermediate Language), называемом далее просто – **IL**. Фактически компиляторы создают так называемый **управляемый модуль** – переносимый исполняемый файл (Portable Executable или PE-файл). Этот файл содержит код на IL и **метаданные** – всю необходимую информацию как для CLR, так и конечных пользователей, работающих с приложением. В зависимости от выбранного типа проекта, PE-файл может иметь расширения exe, dll, mod или mdl.

PE-файл, имеющий расширение exe, хотя и является exe-файлом, но это не совсем обычный исполняемый Windows файл. При запуске он распознается как специальный PE-файл и передается CLR для обработки. Исполняющая среда начинает работать с кодом, в котором специфика исходного языка программирования исчезла. Код на IL начинает выполняться под управлением CLR (поэтому код называется управляемым). Исполнительную среду можно рассматривать как своеобразную виртуальную IL-машину. Эта машина транслирует «на лету» требуемые для исполнения участки кода в команды реального процессора, который в действительности и выполняет код.

#### ***Виртуальная машина***

Отделение программной платформы от среды разработки явилось естественным шагом. Платформа .Net Framework перестала быть ее частью, и **является надстройкой над операционной системой**. Теперь компиляция и создание PE модулей на IL отделено от выполнения, и эти процессы могут быть реализованы на разных платформах. В состав CLR входят трансляторы JIT (Just In Time Compiler), которые и выполняют трансляцию IL в командный код той машины, где установлена и функционирует исполнительная среда CLR.

.Net Framework уже функционирует на многих платформах, отличных от Windows. Он становится **свободно распространяемой виртуальной машиной**. Это существенно расширяет сферу его применения. Производители различных компиляторов и сред разработки программных продуктов предпочитают теперь также транслировать свой код в IL, создавая модули в соответствии со спецификациями CLR. Это обеспечивает возможность выполнения их кода на разных платформах.

Microsoft использовала получивший широкое признание опыт виртуальной машины Java, улучшив процесс за счет того, что, в отличие от Java, промежуточный код не интерпретируется исполнительной средой, а компилируется с учетом всех особенностей текущей платформы. Благодаря этому создаются высокопроизводительные приложения.

Следует отметить, что CLR, работая с IL кодом, выполняет достаточно эффективную оптимизацию и, что не менее важно, защиту кода. Зачастую нецелесообразно выполнять оптимизацию на уровне создания IL кода, она иногда может не улучшить, а ухудшить ситуацию, не давая CLR провести оптимизацию на нижнем уровне, где можно учесть даже особенности процессора.

#### ***Дизассемблер и ассемблер***

Если есть готовый PE-файл, то иногда полезно анализировать его IL код и связанные с ним метаданные. В состав Framework SDK<sup>5</sup> входит **дизассемблер** – ildasm, выполняющий дизассемблирование PE-файла и показывающий метаданные, IL код с комментариями в наглядной форме.

Профессионалы, предпочитающие работать на низком уровне, могут программировать на языке ассемблера IL. В этом случае в их распоряжении будет вся мощь библиотеки FCL и все возможности CLR.

#### ***Метаданные***

Переносимый исполняемый PE-файл является самодокументируемым<sup>6</sup> файлом и содержит и код, и метаданные, описывающие код. Файл начинается с манифеста и включает в себя описание всех классов, хранимых в PE-файле, их свойств, методов, всех аргументов этих методов – всю необходимую CLR информацию. Поэтому помимо PE-файла не требуется никаких дополнительных файлов, записей в реестр, вся нужная информация извлекается из самого файла.

<sup>5</sup> SDK (от англ. software development kit) — набор средств разработки, который позволяет специалистам по программному обеспечению создавать приложения для определённого пакета программ, программного обеспечения базовых средств разработки, аппаратной платформы, компьютерной системы, игровых консолей, операционных систем и прочих платформ.

<sup>6</sup> <https://habr.com/company/geekbrains/blog/270001/>

Среди классов библиотеки FCL имеется класс Reflection, методы которого позволяют извлекать необходимую информацию. Введение метаданных не только важная техническая часть CLR, но это также часть новой идеологии разработки программных продуктов. На уровне языка C# самодокументированию уделяется большое внимание.

При проектировании класса программист может создавать собственные атрибуты, добавляемые к метаданным PE-файла. Клиенты этого класса могут, используя класс Reflection, получать эту дополнительную информацию и на ее основании принимать соответствующие решения.

### ***Сборщик мусора – Garbage Collector и управление памятью***

Еще одной важной особенностью построения CLR является то, что исполнительная среда берет на себя часть функций, традиционно входящих в ведение разработчиков трансляторов, и облегчает тем самым их работу. Один из таких наиболее значимых компонентов CLR — **сборщик мусора** (Garbage Collector).

Под **сборкой мусора** понимается освобождение памяти, занятой объектами, которые стали бесполезными и не используются в дальнейшей работе приложения. В ряде языков программирования память освобождает сам программист, в явной форме отдавая команды как на создание, так и на удаление объекта. Неизбежные ошибки программиста при работе с памятью тяжелы по последствиям, и их крайне тяжело обнаружить. Как правило, объект удаляется в одном модуле, а необходимость в нем обнаруживается в другом далеком модуле. Обоснование того, что программист не должен заниматься удалением объектов, а сборка мусора должна стать частью исполнительской среды, было дано достаточно давно. Наиболее полно оно обосновано в работах Бертрона Мейера и в его книге "Object-Oriented Construction Software", первое издание которой появилось еще в 1988 году.

В CLR эта идея реализована в полной мере. Задача сборки мусора снята не только с программистов, но и с разработчиков трансляторов; она решается в нужное время и в нужном месте – исполнительской средой, ответственной за выполнение вычислений. Здесь же решаются и многие другие вопросы, связанные с использованием памяти, в частности, проверяются возможные нарушения использования «чужой» памяти и другие нарушения, например, с использованием нетипизированных указателей. Данные, удовлетворяющие требованиям CLR и допускающие сборку мусора, называются управляемыми данными.

Но, как быть с языками, где есть нетипизированные указатели, адресная арифметика, возможности удаления объектов программистом? Такие возможности сохранены и в языке C#. Ответ следующий – CLR позволяет работать как с управляемыми, так и с неуправляемыми данными. Однако использование неуправляемых данных регламентируется и не поощряется. Так, в C# модуль, использующий неуправляемые данные (указатели, адресную арифметику), должен быть помечен как небезопасный (unsafe), и эти данные должны быть четко зафиксированы. Исполнительная среда, не ограничивая возможности языка и программистов, вводит определенную дисциплину в применении потенциально опасных средств языков программирования.

### ***Исключительные ситуации***

Что происходит, когда при вызове некоторой функции (процедуры) обнаруживается, что она не может нормальным образом выполнить свою работу? Возможны разные варианты обработки такой ситуации. Функция может возвращать код ошибки или специальное значение типа HRESULT, может выбрасывать исключение, тип которого характеризует возникшую ошибку. В CLR принято во всех таких ситуациях выбрасывать исключение. Косвенно это влияет и на язык программирования. Выбрасывание исключений наилучшим образом согласуется с исполнительской средой. В языке C# выбрасывание исключений, их дальнейший перехват и обработка – основной рекомендуемый способ обработки исключительных ситуаций.

### ***События***

У CLR есть свое видение того, что представляет собой тип. Есть формальное описание общей системы типов CTS – Common Type System. В соответствии с этим описанием, каждый тип, помимо полей, методов и свойств, может содержать и события. При возникновении событий в процессе работы с тем или иным объектом данного типа посылаются сообщения, которые могут получать другие объекты. Механизм обмена сообщениями основан на делегатах – функциональном типе. В языке C# встроен механизм событий, полностью согласованный с возможностями CLR.

Исполнительная среда CLR обладает мощными динамическими механизмами – сборки мусора, динамического связывания, обработки исключительных ситуаций и событий. Все эти механизмы и их реализация в CLR созданы на основании практики существующих языков программирования. Но уже созданная исполнительная среда в свою очередь влияет на языки, ориентированные на использование CLR. Поскольку язык C# создавался одновременно с созданием CLR, то, естественно, он стал языком, наиболее согласованным с исполнительской средой, и средства языка напрямую отображаются в средства исполнительской среды.

### ***Общие спецификации и совместимые модули***

Программная платформа .Net Framework облегчает межъязыковое взаимодействие. Для того, чтобы классы, разработанные на разных языках, мирно уживались в рамках одного приложения, для их бесшовной отладки и возможности построения разноязычных потомков, требуется соблюдение некоторых ограничений.

Эти ограничения задаются набором общезыковых спецификаций – CLS (Common Language Specification). Класс, удовлетворяющий спецификациям CLS, называется CLS-совместимым. Он доступен для использования в других языках, классы которых могут быть клиентами или наследниками совместимого класса.

Спецификации CLS точно определяют, каким набором встроенных типов можно пользоваться в совместимых модулях. Понятно, что эти типы должны быть общедоступными для всех языков, использующих .Net Framework. В совместимых модулях должны использоваться управляемые данные и выполняться некоторые другие ограничения. Ограничения касаются только интерфейсной части класса, его открытых свойств и методов. Закрытая часть класса может и не удовлетворять CLS. Классы, от которых не требуется совместимость, могут использовать специфические особенности языка программирования.

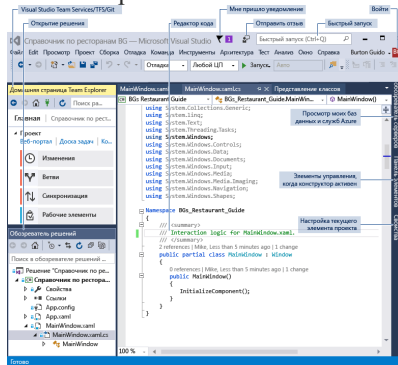
На этом я закончу обзорное рассмотрение. Одной из лучших книг, подробно освещающих Visual Studio .Net и ее платформы .Net Framework, является книга Джеффри Рихтера, переведенная на русский язык: «Программирование на платформе .Net Framework». Крайне интересно, что для Рихтера языки являются лишь надстройкой над каркасом, поэтому он говорит о программировании, использующем возможности исполнительской среды CLR и библиотеки FCL.

## Установка Visual Studio

Найти подходящий выпуск Visual Studio можно в разделе [Выпуски Visual Studio](#). Можно установить Visual Studio 201\*, загрузив эту среду со страницы [Загружаемые файлы Visual Studio](#). Дополнительные сведения о процессе установки см. в статье Установка Visual Studio.

## Основы среды IDE

На рисунке показана интегрированная среда разработки (IDE) Visual Studio с открытым проектом и окном обозревателя решений для навигации по файлам проекта, а также окном Team Explorer для перемещения в системе управления версиями и отслеживания рабочих элементов.



## Вход

При первом запуске Visual Studio можно выполнить вход с использованием учетной записи Майкрософт или рабочей учетной записи.

Вход позволяет обеспечить синхронизацию параметров, например макетов окон, на нескольких устройствах и автоматическое подключение к нужным службам, таким как подписки Azure и Visual Studio Team Services.

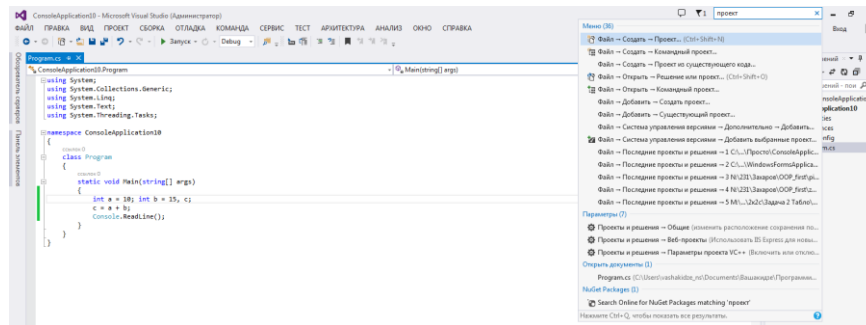
Если имеется несколько учетных записей Visual Studio Team Services, учетных записей Azure или подписок MSDN, их можно связать и осуществлять доступ к ресурсам и службам всех учетных записей, используя единый вход. Для получения дополнительной информации см. [Работа с несколькими учетными записями пользователя](#).

## Обновления

Значок уведомления в правом верхнем углу заголовка окна указывает на наличие обновлений для Visual Studio или связанных компонентов, установленных в системе. Можно закрыть уведомление или выполнить требуемое действие.

## Поиск и получение справки

Панель [Быстрого Запуска](#), показанная ниже, дает возможность быстрого поиска команд, инструментов, функций и других компонентов Visual Studio, если неизвестно сочетание клавиш или расположение меню. В поле быстрого запуска вводится название нужного компонента, и на панели быстрого запуска появится ссылка на него.



## Персонализация среды IDE

Возможно настроить макет окна среды разработки так, как удобно пользователю используя меню Сервис\Параметры\Среда.

Любое окно в любое время можно закрепить, открепить или скрыть, кроме того, редактор можно запускать в полноэкранном режиме. Можно создавать и сохранять различные настраиваемые макеты окон и работать только с теми окнами, которые требуются в определенном контексте.

Например, можно создать полноэкранный макет, чтобы на экран выводился только редактор кода. Кроме того, можно создать разные макеты для отладки и для операций команды. Для получения дополнительной информации см. [Настройка макетов окон](#).

Можно настроить Visual Studio различными способами и переносить параметры при работе на нескольких компьютерах. Для получения дополнительной информации см. [Персонализация среды IDE](#).

Сочетания клавиш предусмотрены практически для всех функций, и их также можно настроить. Чтобы создать новое сочетание клавиш, введите на панели быстрого запуска «Клавиатура», чтобы открыть диалоговое окно клавиатуры. После этого можно нажать клавишу F1, чтобы перейти на страницу справки MSDN, если требуются дополнительные сведения о параметрах. Дополнительные сведения см. в статье [Сочетания клавиш по умолчанию в Visual Studio](#).

## Создание проектов и решений

Несмотря на то, что Visual Studio можно использовать для работы с отдельными файлами кода, как правило, работа выполняется в рамках **проекта**.

**Проект** Visual Studio — это совокупность файлов и ресурсов, которые (для приложений) компилируются в единый двоичный исполняемый файл (например, EXE, DLL, APPX). Для веб-сайтов не на основе ASP.NET исполняемые файлы не создаются; проект содержит только HTML-код, файлы JavaScript и изображения.

Поскольку иногда может потребоваться создать несколько двоичных файлов или веб-сайтов, которые тесно связаны, Visual Studio использует концепцию **решения**, которое может содержать **несколько проектов** или **веб-сайтов**.

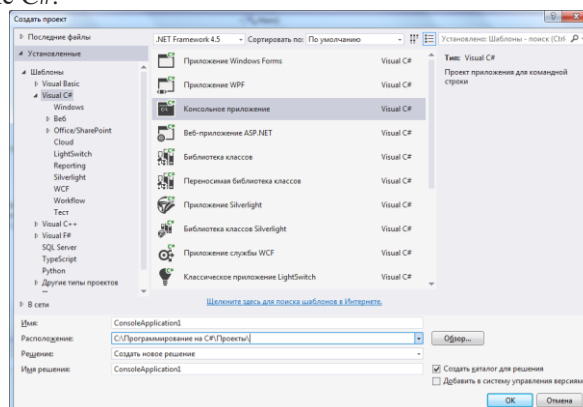
При создании **проекта** фактически создается проект в решении, что позволяет позднее добавить несколько проектов в это решение при необходимости. Например, если имеется проект библиотеки DLL, можно добавить в решение проект EXE, который будет загружать и использовать библиотеку DLL.

**Шаблон проекта** — это набор предварительно заполненных кодом файлов и параметров конфигурации, которые можно быстро настроить для создания приложения определенного типа.

В комплект Visual Studio входит большое число шаблонов проектов, кроме того, если для целей пользователя не подходит ни один из шаблонов по умолчанию, можно создать собственный.

После создания проекта с помощью шаблона можно приступить к написанию собственного кода в имеющихся или в новых добавляемых файлах. Для получения дополнительной информации см. [Проекты и решения](#).

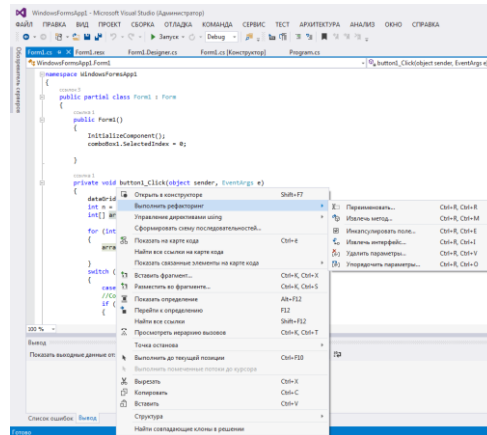
На рисунке показано диалоговое окно нового проекта с шаблонами проектов, которые доступны для создания приложений на языке C#.



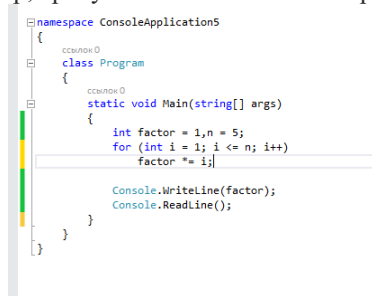
## Проектирование пользовательского интерфейса

Текстовый редактор включает много интерактивных функций (если они требуются) и функций повышения производительности, помогающих ускорить написание кода. Функции различаются в зависимости от языка, и необязательно использовать их все (введите «Редактор» на панели быстрого запуска, чтобы включить или отключить функции). Некоторые распространенные возможности повышения производительности приведены ниже.

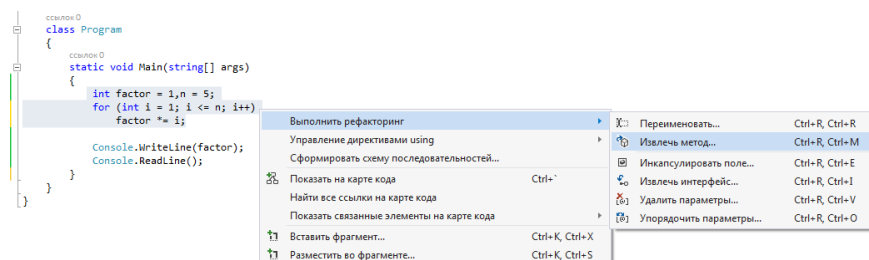
1. Рефакторинг. Рефакторинг — это процесс улучшения написанного ранее кода путем такого изменения его внутренней структуры, которое не влияет на внешнее поведение. Рефакторинг включает такие операции, как интеллектуальное переименование переменных, перемещение выделенных строк кода в отдельную функцию, перемещение кода в другие расположения, изменение порядка параметров функции и т. д.



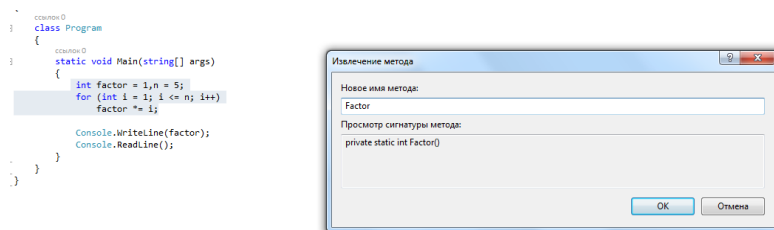
Например, требуется поместить некоторый код в функцию:



Для этого необходимо выделить требуемый фрагмент кода и из контекстного меню выбрать команду Выполнить рефакторинг\Извлечь метод:



Затем ввести имя метода:



Результат:



```

1
ссылка 0
static void Main(string[] args)
{
    int factor = Factor();

    Console.WriteLine(factor);
    Console.ReadLine();
}

ссылка 1
private static int Factor()
{
    int factor = 1, n = 5;
    for (int i = 1; i <= n; i++)
        factor *= i;
    return factor;
}

```

2. **IntelliSense** — это общий термин для набора очень популярных функций, отображающих сведения о типах в коде непосредственно в редакторе и в некоторых случаях автоматически создающих небольшие отрывки кода. По сути IntelliSense представляет собой базовую документацию, встроенную в редактор, что избавляет от необходимости поиска информации о типах в отдельном окне справки. Функции IntelliSense зависят от языка.

Например,

```

ссылка 1
public Form1()
{
    InitializeComponent();
    comboBox1.SelectedIndex = 0;
}

ссылка 1
private void b
{
    int n;
    int[] array;
    NewMethod(out n, out array);
    switch (comboBox1.SelectedIndex)
    {
        case 0:
            //Сортировать
            if (radioButton1.Checked && groupBox1.Visible == true)
            {

```

**Волнистые линии** предупреждают об ошибках или потенциальных проблемах в коде в режиме реального времени по мере ввода, что позволяет исправлять их немедленно, не дожидаясь обнаружения ошибок во время компиляции или выполнения. Если навести указатель мыши на волнистую линию, на экран будут выведены дополнительные сведения об ошибке. Кроме того, в поле слева может появляться значок лампочки с предложениями по устранению ошибок.

```

string PrintResult(int i, string s)
{
    int x = i * Math;
    var resu = s;
}

```

Имя "Math" не существует в текущем контексте

Показать возможные решения

Ссылка «Показать возможные исправления», открывает список быстрых действий, которые может предпринять лампочка.

```

string PrintResult(int i, string s)
{
    int x = i * Math;
    var resu = string.Format("{0}: {1}", s, x);
}

```

using System;

Изменить Math на System.Math

Создать свойство FilterConfig.Math

Создать поле Math в FilterConfig

Создать поле только для чтения FilterConfig.Math

Создать локальную переменную Math

CS0103 Имя Math не существует в текущем контексте

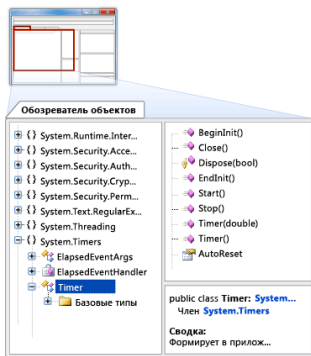
using System;

using System.web;

...

Просмотр изменений

3. **Закладки** позволяют быстро переходить к определенным строкам в файлах, с которыми работает пользователь.
4. В контекстном меню текстового редактора можно вызвать окно **Иерархия вызовов** для отображения методов, которые вызываются методом или вызывают метод, в котором установлен курсор.
5. **CodeLens** позволяет находить ссылки на код, изменения кода, связанные ошибки, рабочие элементы, проверки кода и модульные тесты — все это, не выходя из редактора. Дополнительные сведения см. в разделе [Поиск изменений кода и других журналов](#).
6. Окно **Показать определения** позволяет просмотреть определение метода или типа в окне редактора, не покидая текущий контекст. Это окно теперь поддерживает и XAML-код.
7. Пункт контекстного меню **Перейти к определению** позволяет перейти непосредственно к тому месту, где определена функция или объект. Другие команды навигации также доступны по щелчку правой кнопкой мыши в редакторе.
8. Связанный инструмент, **обозреватель объектов**, позволяет исследовать сборки .NET или среды выполнения Windows в системе, просматривая содержащиеся в них типы (а также методы и свойства в этих типах).



Большинство пунктов меню «Правка» и «Вид» тем или иным образом связаны с редактором кода. Дополнительные сведения о редакторе кода см. в разделах [Создание кода](#) и [Правка кода](#).

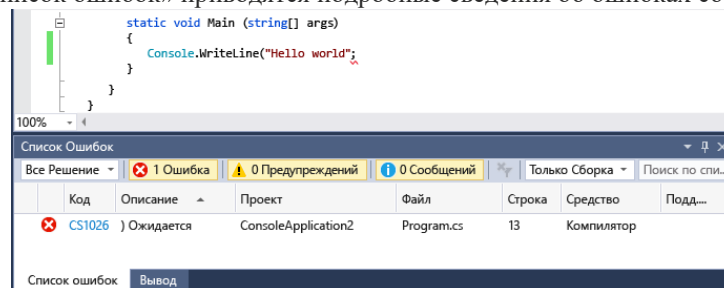
## Компилирование и сборка кода

**Сборка** проекта означает компиляцию исходного кода и выполнение действий, необходимых для создания исполняемого файла. В разных языках предусмотрены разные операции сборки, а для обычных веб-сайтов сборка вообще не выполняется. Но независимо от типа проекта меню «Сборка» — это стандартное расположение этих команд. Чтобы скомпилировать и запустить код одним нажатием клавиши, используется клавиша **F5**.

Все компиляторы можно настроить через среду IDE. Панель инструментов сборки позволяет указать, следует ли создавать отладочную версию программы с включенными символами и дополнительной проверкой на наличие ошибок для поддержки точек останова и пошагового режима отладчика или сборку выпуска, которая в итоге предоставляется клиентам.

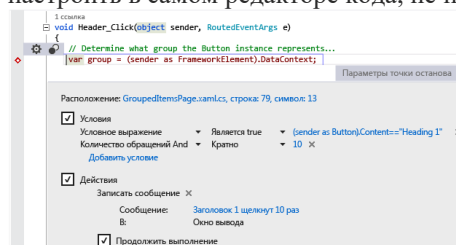
На странице свойств проекта можно настроить дополнительные параметры сборки и многие другие параметры. Щелкните имя проекта в обозревателе решений и выберите пункт «Свойства» или выберите пункт меню Проект\Свойства). Кроме того, сборку можно выполнить из командной строки.

Выходные данные сборки, включая сообщения об ошибках или успешном выполнении, отображаются в окне вывода. В окне «Список ошибок» приводятся подробные сведения об ошибках сборки.



## Отладка кода

Современный отладчик Visual Studio позволяет выполнять отладку кода в локальном проекте, на удаленном устройстве или в эмуляторе, например для устройств Android или Windows Phone. Можно просматривать код с шагом в один оператор, проверяя значения переменных; пошагово выполнять многопоточные приложения, а также задать точки останова, которые срабатывают только при выполнении указанного условия. Все это можно настроить в самом редакторе кода, не покидая контекст кода.



Сам отладчик имеет ряд окон, в которых можно просматривать локальные переменные, стек вызовов и другие аспекты среды выполнения и выполнять операции с ними. Эти окна доступны из меню **Отладка**.

[Окно интерпретации](#) позволяет ввести выражение и сразу увидеть его результат.

[IntelliTrace](#) регистрирует все вызовы методов и другие события в работающей программе .NET и может помочь быстро найти источник проблемы.

Дополнительные сведения см. в разделе [Отладка в Visual Studio](#).

## Лабораторное занятие №2 (4 ч.)

Тема **Операторы языка C#.**



Вопросы для обсуждения:

1. Операторы помеченные (labeled-statement), декларирующие (declaration-statement), встроенные (embedded-statement).
2. Особенности реализации операторов в языке C#.

Примерные задания

1. **Числа.** Найти двузначные числа  $\overline{ab}$  и  $\overline{cd}$  такие, что  $\overline{abcd} = \overline{ab} * \overline{cd}$ .
2. **Фальшивая монета.** Имеется  $3^n$  монет, среди которых есть фальшивая (тяжелее всех остальных). Требуется с помощью чашечных весов без гирь ровно за  $n$  взвешиваний определить номер фальшивой монеты. Пользователь вводит 0, если весы уравновешены. 1 – если перевесила левая чаша. 2 – если перевесила правая чаша.
3. **Последовательность** 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, ..., состоящая из нулей и единиц строится так: первый ее элемент равен 1, а остальные получаются из предшествующих с помощью логической операции отрицания:  $\text{not}(1) = 0$ ,  $\text{not}(0) = 1$ . Второй элемент равен отрицанию первого, третий и четвертый – отрицанию первого и второго соответственно и т.д. По заданному  $n$  вычислить  $n$ -ый член указанной последовательности.
4. **Квадраты.** От заданного прямоугольника каждый раз отрезается квадрат максимальной площади (длины сторон выражаются натуральными числами). Найти количество таких квадратов
5. **Лестница.** Поднимаясь по лестнице, заяц прыгает либо на следующую ступеньку, либо через одну, либо через две ступеньки. Сколькими способами он может подняться на ступеньку с номером  $n$ ?
6. Вычислить  $n$ -ый член последовательности натуральных чисел 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, ... по заданному  $n$ .
7. **Остатки.** Задумано целое число  $X$ . Известны числа  $k, m, n$  – остатки от деления этого числа на 3, 5, 7 соответственно. Найдите число  $X$ .
8. **Колония роботов.** Колония роботов живет и развивается по следующим законам: один раз в начале года они объединяются в группы по 3 или 5 роботов. За один год группа из 3 роботов собирает 5 новых роботов, а группа из 5 роботов собирает 9 новых роботов. Роботы объединяются так, чтобы собрать за год наибольшее количество новых роботов. Каждый робот живет три года после сборки. Известно, что начальное количество роботов равно  $k$  и все они только что собраны. Сколько роботов будет содержать колония через  $n$  лет?

### Лабораторное занятие №3 (12 ч.)

Тема **Массивы в C#.**

Вопросы для обсуждения:

1. Массивы одномерные, многомерные.
2. Массивы массивов.
3. Непрямоугольные массивы.
4. Особенности реализации массивов в языке C#.

Примерные задания

1. Объяснение работы на примере решения задачи создания одномерного и двумерного массивов, поиска наибольшего и наименьшего значений элементов.
2. Индивидуальные задания по теме Одномерные массивы (20 вариантов).
3. Индивидуальные задания по теме Двумерные массивы (20 вариантов).

### Лабораторное занятие №4 (6 ч.)

Тема **Строки в C#.**

Вопросы для обсуждения:

1. Строки как объекты класса string.
2. Строка как контейнер.
3. Применение строк в переключателях.
4. Массивы строк. Операции над строками.
5. Особенности реализации строк в языке C#.

Примерные задания

1. **Перенос.** Примем следующие правила переноса русских слов:

- В каждой из разделяемых частей должно быть более одной буквы, из которых хотя бы одна – гласная;
- Нельзя разделять согласную и следующую за ней гласную;
- Буквы Й, Ъ, Ы считать согласными, но перенос после них допустим.

**Требуется.** В каждом из вводимых слов поставить все возможные знаки переноса, например: ОБЯ-ЗА-ТЕ-ЛЬНЫЙ. Строчные и прописные буквы считать неразличимыми.

*Интересные тесты для проверки (Па-пка, Дом, Она, Йо-гурт, Эль-ни-ньо, Ап-прок-си-ма-ция, Ария, Прей-ску-рант, Вью-га, Длин-но-шеее, Уха, Шея, Съесть, Ари-эль, Змее-ед, Арий-цы).*

2. **Табло.** В городе N собираются праздновать день рождения одного знаменитого жителя. В честь этого решено установить в центре города табло, на котором должно отображаться количество дней, оставшихся до знаменательной даты. Требуется написать программу для табло.
3. **Длина.** В строке, состоящей из слов, разделенных пробелами, определить длину самого короткого и длинного слов.
4. **Шифрованное сообщение.** Студенту №1 от Студента №2 поступило зашифрованное сообщение. Студенту №3 удалось установить, что каждое слово в сообщении перевернуто. Требуется восстановить исходное сообщение.
5. **Новобранцы.** На первом построении вновь призванные в армию солдаты построились в шеренгу. После небольшого объяснения им правил выполнения в строю различных команд последовала команда "налево". В результате исполнения этой команды некоторые солдаты повернулись налево, а некоторые – направо. Солдаты, которые оказались лицом к лицу со своим соседом, сразу поняли, что совершили ошибку. Чтобы ее исправить, каждый из них опять быстро повернулся на 180 градусов. Если названная ситуация затем опять повторялась, то есть, в каких-то парах солдаты оказались лицом друг к другу, то такие солдаты снова поворачивались на 180 градусов. Эта процедура продолжалась до тех пор, пока в шеренге была хотя бы одна пара солдат, стоящих лицом друг к другу.

**Требуется** написать программу, которая по расположению солдат сразу после исполнения команды "налево" вычисляет число пар солдат, совершивших впоследствии разворота на 180 градусов в соответствии с вышеописанной процедурой.

Пример:

Пусть 6 солдат стоят следующим образом:

>><<<<

Ответом будет число 7.

В таблице 1 для этого примера приведены расположения солдат в шеренге после очередного завершения разворотов на 180 градусов соответствующих пар солдат.

Таблица 1

Расположение солдат	Количество пар, которые должны развернуться	Комментарий
>><<<<	2	Расположение солдат сразу после исполнения команды "налево"
><<<<	2	Расположение солдат после первого завершения разворотов соответствующих
<<<<<<	2	Расположение солдат после второго завершения разворотов соответствующих

<<<>>>	1	Расположение солдат после третьего завершения разворотов соответствующих
<<<>>>	Общее количество	Конечное расположение солдат

### Лабораторное занятие №5 (10 ч.)

#### Тема Методы в С#.

Вопросы для обсуждения:

1. Методы–процедуры и методы-функции.
2. Соответствие фиксированных параметров и аргументов.
3. Параметры с типами ссылок.
4. Методы с переменным числом аргументов.
5. Перегрузка методов.
6. Рекурсивные методы. Особенности реализации методов в языке С#.

#### Примерные задания

1. Индивидуальные задания по теме Методы (20 вариантов).

### 3 семестр

#### Лабораторное занятие №1 (6 ч.)

#### Тема Классы как типы.

Вопросы для обсуждения:

1. Объявление класса.
2. Поля объектов.
3. Методы объектов.
4. Ссылка this.
5. Конструкторы объектов класса (умолчания, общего вида, копирования, приведения).
6. Деструкторы и финализаторы.

#### Примерные задания

1. Жизнь. ([http://www.arbuz.uz/gazeta\\_23.html](http://www.arbuz.uz/gazeta_23.html))

На бесконечном плоском клеточном поле, состоящем из пустых клеток, поселяется колония организмов. Каждая клетка имеет ровно 8 соседних клеток. Колония живет и развивается, подчиняясь следующим законам:

- Организм выживает, если имеет 2 или 3 соседей;
- организм умирает от одиночества, если у него не более одного соседа;
- организм умирает от перенаселения, если у него более 3 соседей;
- новый организм рождается, если число соседей равно 3.

#### Лабораторное занятие №2 (8 ч.)

#### Тема Средства взаимодействия с объектами.

Вопросы для обсуждения:

1. Принцип инкапсуляции и методы объектов.
2. Свойства классов.
3. Автореализуемые свойства.
4. Индексаторы.

#### Примерные задания

1. Индивидуальная задача по теме «Классы»

Каждый разрабатываемый класс должен, как правило, содержать следующие элементы: скрытые поля, конструкторы с параметрами и без параметров, методы, свойства, индексаторы, перегруженные операции. Функциональные элементы класса должны обеспечивать непротиворечивый, полный, минимальный и удобный интерфейс класса. В программе должна выполняться проверка всех разработанных элементов класса.

2. **Задача об инфекции стригущего лишая** (Ван Тассел Д. Стил, разработка, эффективность, отладка и испытание программ. – М.: Мир, 1981)

Промоделировать процесс распространения инфекции – стригущего лишая по участку кожи размером  $n \times n$  ( $n$  – нечетное) клеток. Предполагается, что исходной зараженной клеткой кожи является центральная. В каждый интервал времени пораженная инфекцией клетка может с вероятностью 0,5 заразить любую из соседних здоровых клеток. По прошествии 6 единиц времени зараженная клетка становится невосприимчивой к инфекции. Возникший иммунитет действует в течение последующих 4 единиц времени, а затем клетка оказывается здоровой.

В ходе моделирования описанного процесса необходимо выдавать текущее состояние исследуемого участка кожи в каждый интервал времени, отмечая зараженные, невосприимчивые к инфекции и здоровые клетки.

### Лабораторное занятие №3 (12 ч.)

Тема **Включение, вложение и наследование классов.**

Вопросы для обсуждения:

1. Включение объектов классов.
2. Вложение классов.
3. Наследование классов.
4. Доступность членов класса при наследовании.
5. Методы при наследовании.
6. Абстрактные методы и абстрактные классы.
7. Применение абстрактных классов.
8. Опечатанные классы и методы.

*Примерные задания*

1. **Построить три класса (базовый и 2 потомка), описывающих некоторых работников с почасовой оплатой (один из потомков) и фиксированной оплатой (второй потомок).**

Описать в базовом классе абстрактный метод для расчета среднемесячной заработной платы. Для «повременщиков» формула для расчета такова: «среднемесячная заработная плата = 20.8 \* 8 \* почасовую ставку», для работников с фиксированной оплатой «среднемесячная заработная плата = фиксированной месячной оплате».

а) Упорядочить всю последовательность работников по убыванию среднемесячного заработка. При совпадении зарплат – упорядочивать данные по алфавиту по имени. Вывести идентификатор работника, имя и среднемесячный заработок для всех элементов списка.

б) Вывести первые 5 имен работников из полученного в пункте а) списка.

с) Вывести последние 3 идентификатора работников из полученного в пункте а) списка.

д) Организовать запись и чтение коллекции в/из файл.

е) Организовать обработку некорректного формата входного файла

2. Индивидуальная задача по теме «Наследование классов»

### Лабораторное занятие №4 (10 ч.)

Тема **Интерфейсы.**

Вопросы для обсуждения:

1. Интерфейс как механизм наследования специфицированной функциональности.
2. Наследование специфицированной функциональности.
3. Реализация интерфейсов.
4. Интерфейс как тип.
5. Интерфейсы и наследование.

*Примерные задания*

1. Индивидуальная задача по теме «Интерфейсы»

Форма контроля	За одну работу	Всего
----------------	----------------	-------

	Мин. баллов	Макс. баллов	Мин. баллов	Макс. баллов
Текущий контроль:				
Активная работа на занятии	0,25	0,5	9	18
Выполнение домашнего задания	0,75	0,75	27	27
Выполнение заданий самостоятельной работы	1	3	1	3
коллоквиум	1	3	3	9
Промежуточная аттестация (экзамен)			20	43
<b>Итого за семестр</b>			60	100

## Вопросы к экзамену

### 1 семестр

1. Интуитивное понятие алгоритма. Свойства алгоритма. Способы описания. Формальное описание алгоритма. Нормальные алгоритмы.
2. Структуры алгоритмов.
3. Введение. Особенности языка Delphi. Возможности языка Delphi. Система типов данных языка Delphi. Тожественность и совместимость типов.
4. Выражения, операнды, операции.
5. Структура программы, ее основные разделы и их назначение. Комментарии.
6. Некоторые приемы оптимизации программ.
7. Операторы: простые, структурированные. Форматы записи.
8. Структурированные типы данных: строки. Описание типа. Динамические и статические строки. Строковые процедуры и функции.
9. Структурированные типы данных: массивы (статические, динамические). Описание типа. Действия над массивами. Действия над элементами массива.
10. Простейшие алгоритмы обработки массивов: ввод-вывод массивов, нахождение следа матрицы, суммирование элементов строк матрицы, умножение матрицы на вектор и матрицы на матрицу, циклический сдвиг массива, инвертирование массива, формирование массива из элементов другого массива, удовлетворяющих условию.
11. Простейшие алгоритмы обработки массивов: удаление элемента из одномерного массива, удаление заданной строки из матрицы, включение элемента в заданную позицию массива, включение элемента в упорядоченный массив с сохранением упорядоченности, включение строки в массив, перестановка элементов массива, перестановка строк матрицы, преобразование двумерного массива в одномерный.
12. Алгоритмы информационного поиска и сортировки: задача поиска и ее разновидности: нахождение минимального значения элементов последовательности (все элементы разные); нахождение номера минимального элемента последовательности (все элементы разные); нахождение минимального элемента и его номера в последовательности с совпадающими элементами; нахождение номера элемента с заданным значением (все элементы разные), поиск элемента в упорядоченной последовательности.
13. Алгоритмы информационного поиска и сортировки: задача сортировки, основные понятия. Простые методы сортировки массивов (выбором, обменом, вставками). Алгоритм Шелла. Алгоритм Хоара.
14. Структурированные типы данных: множества. Описание типа. Операции над множествами.
15. Структурированные типы данных: записи. Описание типа. Записи с вариантами.
16. Процедуры и функции, определяемые пользователем. Механизм передачи параметров. Область действия идентификаторов. Рекурсивные подпрограммы. Предварительное описание подпрограмм. Процедурные типы. Перегрузка функций.
17. Операции ввода и вывода. Структурированные типы данных: файлы. Описание типа. Стандартные процедуры и функции для типизированных файлов.
18. Структурированные типы данных: файлы. Текстовые файлы. Стандартные процедуры и функции для текстовых файлов. Обмен информацией между программой и внешним текстовым файлом на диске.
19. Структурированные типы данных: файлы. Нетипизированные файлы. Стандартные процедуры и функции для нетипизированных файлов.
20. Ссылочные типы и указатели. Описание типа. Действия над ссылками.

21. Проблема потерянных ссылок. Совместимость ссылочных типов.
22. Организация динамических структур с помощью указателей.
23. Организация кольцевого двусвязного списка с помощью указателей.
24. Основные понятия модульного программирования. Общая структура модуля. Подпрограммы в модулях. Компиляция и использование модулей. Типы модулей в Delphi. Создание приложения, содержащего несколько форм.

## **2 семестр**

1. Основные понятия объектно-ориентированного программирования. Объект, класс, составляющие класса. Наследование, полиморфизм, инкапсуляция.
2. Поля, свойства и методы (статические, динамические, виртуальные, абстрактные, методы класса, перегружаемые методы. Конструкторы, деструкторы.
3. Представление объекта, его внутренняя структура. Операторы `is` и `as`.
4. Области видимости элементов класса.
5. Обработка исключений.
6. Структура программы на C#. Понятие пространства имен.
7. Типы данных. Классификация типов. Типы данных. Простые типы. Константы (литералы). Объявление переменных и констант базовых типов.
8. Операции языка C#. Операции присваивания и оператор присваивания. Операции инкремента и декремента. Поразрядные операции. Переполнения при операциях с целыми значениями. Автоматическое и явное приведение арифметических типов. Логический тип и логические выражения. Выражения с символьными операндами.
9. Простые (базовые) типы C# как классы платформы .NET Framework. Специфические методы и поля простых типов.
10. Общие сведения об операторах языка C#. Метки и оператор безусловного перехода. Операторы выбора (операторы ветвлений). Операторы передачи управления. Операторы цикла.
11. Одномерные массивы в C#. Массивы как наследники класса `Array`. Нестатические методы (методы объектов) класса `Array`. Виды массивов и массивы многомерные.
12. Массивы массивов и непрямоугольные массивы. Поверхностное копирование.
13. Строки – объекты класса `string`. Строковые литералы. Строковые объекты и ссылки типа `string`. Операции над строками. Аргументы метода `Main()`. Неизменяемость объектов класса `String`.

## **3 семестр**

1. Методы в C#. Методы–процедуры и методы-функции. Соотношение фиксированных параметров и аргументов. Методы с переменным числом аргументов. Перегрузка методов. Рекурсивные методы.
2. Класс как совокупность статических членов. Статические члены класса. Статические поля классов. Статические константы.
3. Класс как совокупность статических членов. Статические члены класса. Статические методы. Статический конструктор. Статические классы.
4. Классы как типы. Объявление класса. Поля объектов. Методы объектов. Ссылка `this`. Конструкторы объектов класса (умолчания, общего вида, копирования, приведения). Деструкторы и финализаторы.
5. Средства взаимодействия с объектами. Принцип инкапсуляции и методы объектов. Свойства классов. Автореализуемые свойства. Индексаторы.
6. Включение, вложение и наследование классов. Включение объектов классов. Вложение классов.

### **Критерии оценки:**

– **оценка «отлично»** выставляется студенту, если студент свободно ориентируется в теоретическом материале; умеет изложить и корректно оценить различные подходы к излагаемому материалу, способен сформулировать и доказать собственную точку зрения; обнаруживает свободное владение понятийным аппаратом; демонстрирует готовность применять теоретические знания в практической деятельности и полное освоение показателей формируемых компетенций;

– **оценка «хорошо»** выставляется студенту, если студент хорошо ориентируется в теоретическом материале; имеет представление об основных подходах к излагаемому материалу; знает определения основных теоретических понятий излагаемой темы, в

основном демонстрирует готовность применять теоретические знания в практической деятельности и освоение большинства показателей формируемых компетенций;

– **оценка «удовлетворительно»** выставляется студенту, если студент может ориентироваться в теоретическом материале; в целом имеет представление об основных понятиях излагаемой темы, частично демонстрирует готовность применять теоретические знания в практической деятельности и освоение некоторых показателей формируемых компетенций;

– **оценка «неудовлетворительно»** выставляется студенту, если студент не ориентируется в теоретическом материале; не сформировано представление об основных понятиях излагаемой темы, не демонстрирует готовность применять теоретические знания в практической деятельности и освоение показателей формируемых компетенций.

Составитель \_\_\_\_\_  
(подпись)

Вашакидзе Н.С

«12» марта 2024 г.