АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Лекции – 36

Лабораторные работы – 36

РГР

Экзамен

ЦЕЛИ И ЗАДАЧИ КУРСА

Цель изучения - обучение современным приемам и методам составления алгоритмов и программ, реализующих решение вычислительных и общих задач обработки информации, получение навыков производить структурный анализ задачи.

Основные задачи дисциплины:

1) получить навыки анализа решаемых задач;

2) получить навыки программирования алгоритма;

3) получить навыки проектирования сложных программ;

4) получить навыки отладки и тестирования программ.

ЛИТЕРАТУРА

- Восходящее и нисходящее программирование [Электронный ресурс] метод. указания/ ОмГТУ ; сост. О. П. Шафеева. - Электрон. текстовые дан. (624 Кб). - Омск : Изд-во ОмГТУ, 2015. - 1 эл. опт. диск (CD-ROM)
- Объектно-ориентированное программирование. C++ [Электронный ресурс]: учеб. электрон. изд. локального распространения: метод. указания к лаб. работам / ОмГТУ; сост. О. П. Шафеева. - Электрон. текстовые дан. (738 Кб.). - Омск: Изд-во ОмГТУ, 2017. - 1 эл. опт. диск (CD-ROM).
- Андреева, Е. Г. Введение в программирование на языке BORLAND C/C ++ [Электронный ресурс]: учеб. электрон. изд. локального распространения: учеб. пособие для студентов вузов по специальности 351400 "Прикладная информатика (по обл.)" и др. экон. специальностям / Е. Г. Андреева, Л. Д. Федорова, О. В. Храповицкая; ОмГТУ. - Электрон. текстовые дан. (12,0 Мб). - Омск: Изд-во ОмГТУ, 2013. - 1 эл. опт. диск (CD-ROM).
- Титов, Дмитрий Анатольевич. Языки программирования C++/ Matlab [Электронный ресурс]: учеб. электрон. изд. локального распространения: лаб. практикум / Д. А. Титов, А. В. Косых, Е. А. Фадина; ОмГТУ. - Электрон. текстовые дан. (980 Кб). - Омск: Изд-во ОмГТУ, 2013. - 1 эл. опт. диск (CD-ROM).
- Конова, Е.А. Алгоритмы и программы. Язык C++ [Электронный ресурс]: учебное пособие / Е.А. Конова, Г.А. Поллак. – М.: Лань, 2017. – 384 с. – ЭБС Лань.

ИНФОРМАЦИОННЫЕ РЕСУРСЫ

- Научная электронная библиотека elibrary.ru.
- ЭБС «Арбуз».
- ИНТЕГРУМ.
- ЭБС Лань.

Программы и алгоритмы

Основное назначение компьютера – обработка информации, для чего необходимо выполнить определенный набор операций - **программу**.

Программа – набор инструкций, описывающих последовательность действий, приводящих к результату.

Программу можно написать на машинном языке или используя специальные языки называемые языками программирования.

Программа на языке программирования преобразуется в машинные команды, которые затем выполняются компьютером.

Программы и алгоритмы

Алгоритм – это конечная последовательность четко определенных действий, задающая обработку исходных данных с целью получения нужного результата.

Свойства алгоритмов

1. Массовость (обеспечение функций алгоритма для большой совокупности данных)
2. Дискретность (возможность представить алгоритм в виде отдельных последовательных шагов)
3. Определенность (каждый шаг алгоритма должен быть четко определен и однозначно понятен)

Свойства алгоритмов

4. Результативность (получение нужного результата)
5. Конечность (выполнение алгоритма за конечное число шагов)

Способы представления алгоритма

1. Описательная форма (на естественном языке)
2. Псевдокод (описательная форма с ограниченным числом элементов)
3. Графическая форма (схема алгоритма)
4. Табличная форма (таблицы решений)

Основные конструкции псевдокода

Псевдокод:

1. Следование

…
Действие 1
Действие 2
…

2. Ветвление

…
Если Условие
    то Действие 1
    иначе Действие 2
Все-если
…

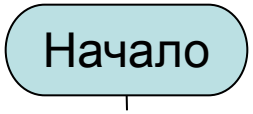
3. Цикл-пока

…
Цикл-пока Условие
    Действие
Все-цикл
…

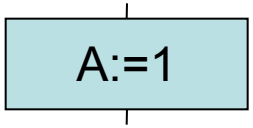
Схемы алгоритмов

Обозначения ГОСТ 19.701 – 90

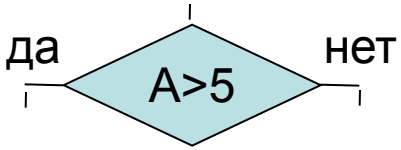
1. Терминатор (начало/конец)



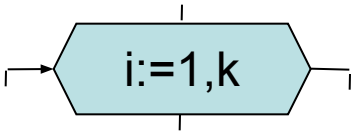
2. Процесс (вычисления)



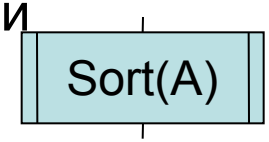
3. Анализ (проверка)



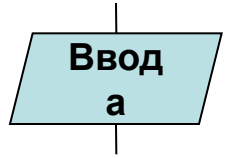
4. Модификатор (автоматическое изменение)



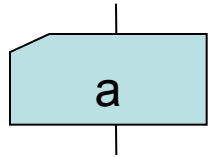
5. Предопределенный процесс (подпрограмма)



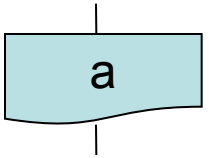
6. Ввод/вывод данных



7. Ввод с перфокарт

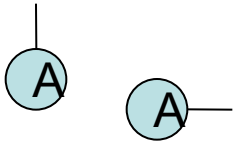


8. Вывод на принтер



9. Комментарий

10. Соединитель



Таблицы решений

Таблица составляется следующим образом.

В столбик выписываются все условия, от которых зависят дальнейшие вычисления, а по горизонтали - все случаи для вычислений.

На пересечении каждого столбца и строки ставят букву Y, если для данного решения данное условие должно выполняться, букву N, если данное условие обязательно должно не выполняться, и прочерк, если исход сравнения не важен.

Например, для алгоритма вычисления корней квадратного уравнения можно составить следующую таблицу:

Таблицы решений

	Нет корней	$x = -b / 2a$	$x=\pm(-b\sqrt{D}) / 2a$
$D < 0$	Y	N	N
$D = 0$	N	Y	N
$D > 0$	N	N	Y

Основные критерии качества программы

- надежность
- возможность точно планировать производство и сопровождение

Для достижения этих целей программа должна:

- иметь простую структуру
- быть хорошо читаемой
- быть легко модифицируемой

Парадигмы программирования

Парадигма — способ организации программы, принцип ее построения. Наиболее распространенными являются процедурная и объектно-ориентированная парадигмы.

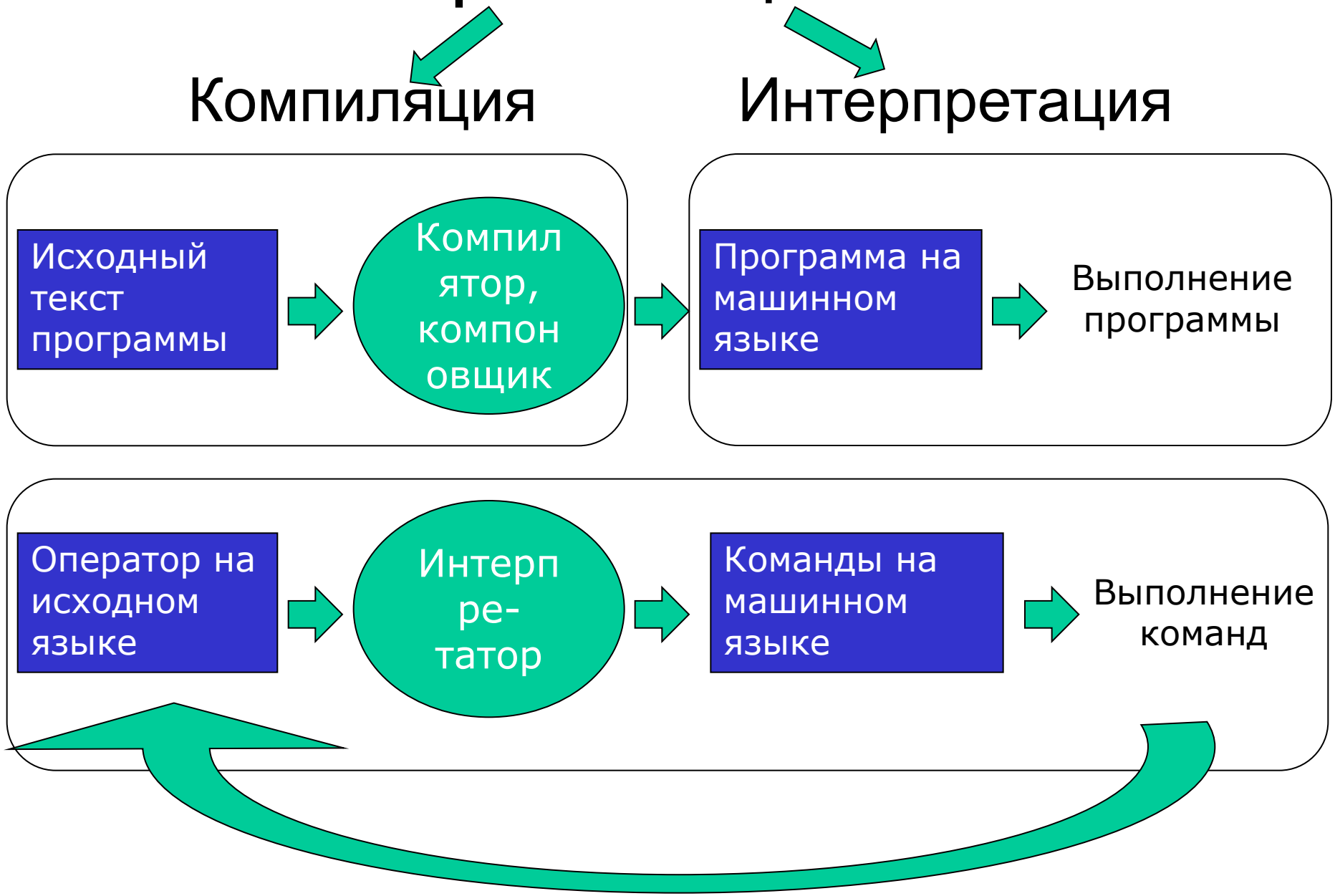
Различаются способом декомпозиции, положенным в основу при создании программы.

Процедурная декомпозиция состоит в том, что задача, реализуемая программой, делится на подзадачи, а они, в свою очередь — на более мелкие этапы, то есть выполняется пошаговая детализация алгоритма решения задачи.

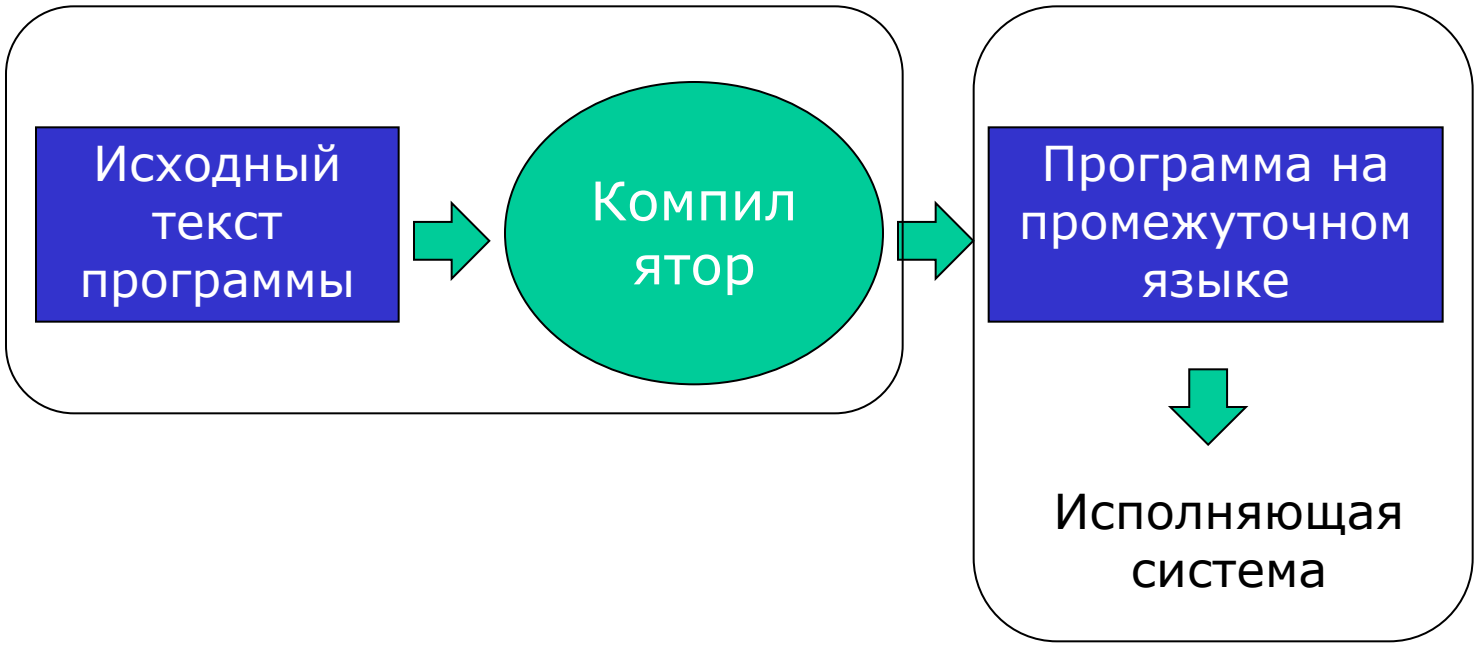
Объектно-ориентированная декомпозиция предполагает разбиение предметной области на объекты и реализацию этих объектов и их взаимосвязей в виде программы.

Язык программирования C#

Трансляция


Компиляция
Интерпретация
Исходный текст программы
Компилятор, компоновщик
Программа на машинном языке
Выполнение программы
Оператор на исходном языке
Интерпретатор
Команды на машинном языке
Выполнение команд

Гибридная схема трансляции



Платформа .NET

.NET (читается "дот нэт") или .NET Framework — это платформа программирования.

Компьютерная платформа — это аппаратный и/или программный комплекс, служащий основой для различных вычислительных систем.

Примером платформы программирования может служить операционная система компьютера.

Приложения на платформе .NET можно создавать с помощью многих языков программирования, таких как C#, F#, S#, Visual Basic и др.

Платформа содержит обширную общую для всех поддерживаемых языков библиотеку базовых классов, которые обеспечивают работу: приложений с графическими объектами, создание веб-интерфейсов, обычных (настольных) и консольных (без графики) приложений, работу с базами данных, дают возможность создавать интерфейсы для работы с удаленными объектами.

Платформа .NET

Платформа .NET Framework - это управляемая среда выполнения, предоставляющая разнообразные службы работающим в ней приложениям. Она состоит из двух основных компонентов:

- *исполняющей среды общего языка* (Common Language Runtime, CLR), являющейся механизмом, управляющим выполняющиеся приложения;
- *библиотеки классов .NET Framework*, предоставляющей библиотеку проверенного кода, предназначенного для повторного использования, который разработчики могут вызывать из своих приложений.

Службы (услуги), которые платформа .NET Framework предоставляет работающим приложениям:

- **управление памятью.** Во многих языках программирования разработчики самостоятельно назначают и выделяют ресурсы памяти и решают вопросы, связанные со временем жизни объектов. В приложениях платформы .NET Framework среда CLR предоставляет эти сервисы автоматически;

Платформа .NET

- **система общего типа.** В традиционных языках программирования базовые типы определяются компилятором, что осложняет взаимодействие между языками. В платформе .NET Framework базовые типы определяются единственной системой типа .NET Framework, называемой CTS (Common Type System). При этом используются одни и те же базовые типы для всех языков .NET Framework;
- **расширенная библиотека классов.** Вместо того чтобы писать много кода для выполнения стандартных низкоуровневых операций программирования, разработчики могут использовать легкодоступную библиотеку типов и члены из библиотеки классов .NET Framework;
- **платформы и технологии разработки.** Платформа .NET Framework включает библиотеки для конкретных областей разработки приложений, например ASP.NET для веб-приложений, ADO.NET для доступа к данным и Windows Communication Foundation для приложений, ориентированных на службы (сервисы);

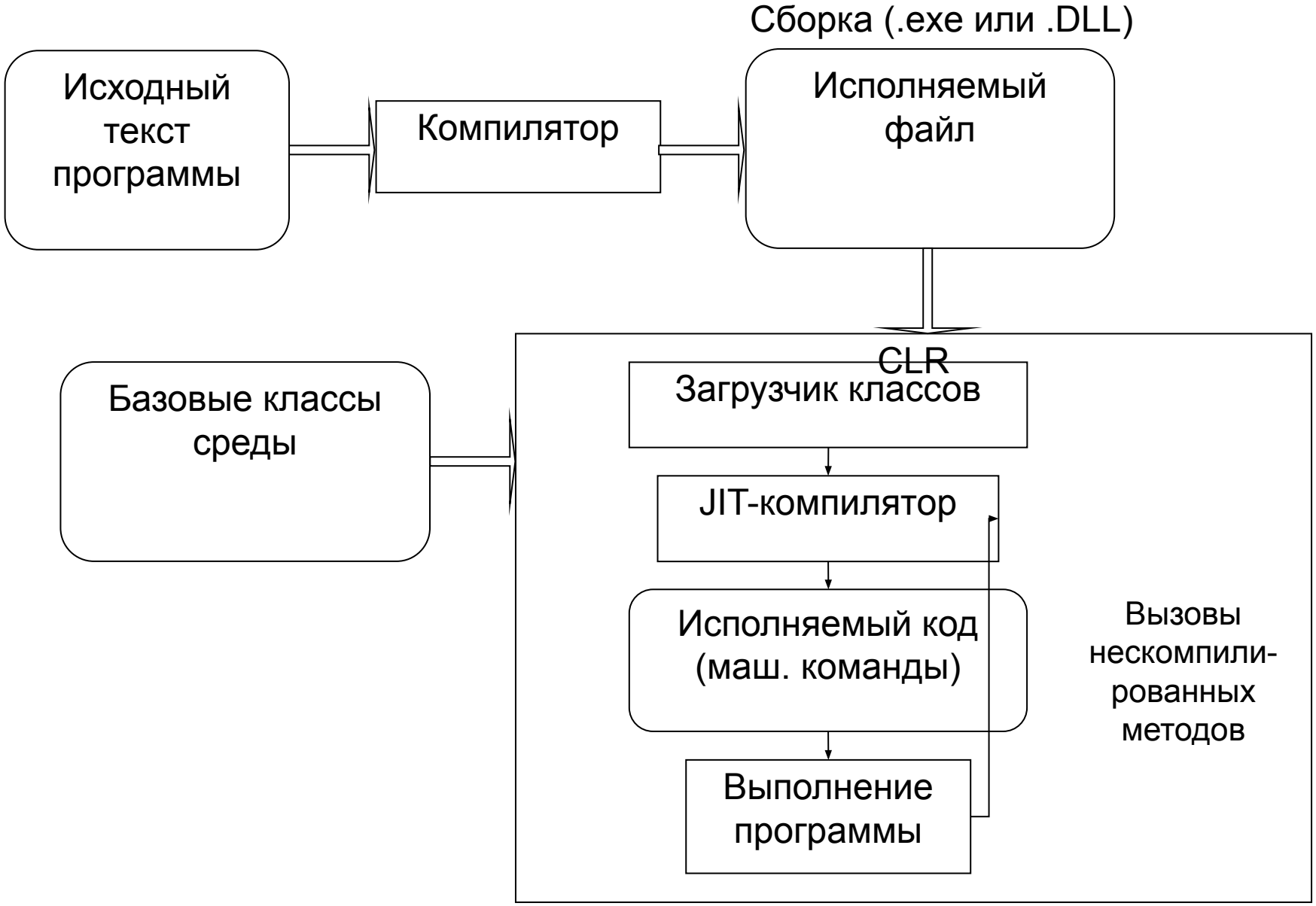
Платформа .NET

- **взаимодействие языков.** Языковые компиляторы на платформе .NET Framework компилируют приложение не в исполнительный код сразу, а в промежуточный код, называемый языком CIL (Common Intermediate Language), который впоследствии компилируется во время исполнения приложения средой CLR. Такой подход приводит к тому, что программы, написанные на одном языке, доступны в других языках, а разработчики могут сосредоточиться на создании приложений на предпочитаемом языке или языках;
- **совместимость версий.** За редкими исключениями, приложения, которые разрабатываются с помощью платформы .NET Framework определенной версии, могут выполняться без изменений на более поздней версии;
- **параллельное выполнение.** Платформа .NET Framework помогает в разрешении конфликтов версий, разрешая установку нескольких версий среды CLR на одном компьютере. Это означает, что несколько версий приложений также могут сосуществовать, и что приложение может выполняться на версии платформы .NET Framework, для которой оно было создано;

Платформа .NET

- **настройка для различных версий.** Ориентируясь на переносимую библиотеку классов платформы .NET Framework, разработчики могут создавать сборки (exe- или dll-файлы, предназначенные для исполнения), которые работают на нескольких платформах .NET Framework. Например, на .NET Framework, Silverlight, Windows Phone 7 или Xbox 360.

Схема выполнения программы в .NET



Классы

- Понятие *класс* аналогично обыденному смыслу этого слова в контексте «класс членистоногих», «класс задач».
- Класс является обобщенным понятием, определяющим характеристики и поведение некоторого множества конкретных объектов этого класса, называемых *экземплярами класса (объектами)*.
- Класс содержит данные, задающие свойства объектов класса, и функции (методы), определяющие их поведение.
- Все классы .NET имеют одного общего предка — класс object, и организованы в единую иерархическую структуру.
- Классы логически сгруппированы в так называемые пространства имен, которые служат для упорядочивания имен классов и предотвращения их конфликтов: в разных пространствах имена могут совпадать. Пространства имен могут быть вложенными

Основные понятия языка

Состав языка

Состав языка

- **Символы**:
  - буквы: A-Z, a-z, _
  - цифры: 0-9, A-F
  - спец. символы: +, *, {, …
  - пробельные символы
- **Лексемы:**
  - константы 2 0.11 “Текст”
  - имена T11 a _11
  - ключевые слова double do if
  - знаки операций + - =
  - разделители ; [ ] ,
- **Выражения**
  - выражение - правило вычисления значения: a + b
- **Операторы**
  - исполняемые: c = a + b;
  - описания: double a, b;

Ключевые слова, знаки операций, разделители

- Ключевые слова - идентификаторы, имеющие специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены.
- Например, для оператора перехода определено слово goto.
- Знак операции - один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются.
- Например, сложение +, деление /, сложное присваивание %=.
- Операции делятся на унарные (с одним операндом), бинарные (с двумя) и тернарную (с тремя).
- Разделители используются для разделения или, наоборот, группирования элементов. Примеры разделителей: скобки, точка, запятая.

Ключевые слова C#

abstract as base bool break byte case catch char
checked class const continue decimal default
delegate do double else enum event explicit extern false
finally fixed float for foreach goto if implicit in int
interface internal is lock long namespace new null
object operator out override params private protected
public readonly ref return sbyte sealed short sizeof
stackalloc static string struct switch this throw true try
typeof uint ulong unchecked unsafe ushort using
virtual void volatile while

Имена (идентификаторы)

- имя должно начинаться с буквы или _;
- имя должно содержать только буквы, знак подчеркивания и цифры;
- прописные и строчные буквы различаются;
- длина имени практически не ограничена.
- имена не должны совпадать с ключевыми словами, однако допускается: @if, @float…
- в именах можно использовать управляющие последовательности Unicode

Примеры правильных имен:
V_a, _13, \u00F2\u01DD, @while.

Примеры неправильных имен:
2late, Big gig

Структура программы на C#

```
using System;
namespace A
{
    class Class1
    {
      static void Main()
       {

          // описания и операторы
       }

      // описания
    }
}
```

Структура программы C#

В C# приложение оформляется в виде специальной структуры - класса, а сама программа, задается как член этой структуры со своим заголовком (и своим блоком строк, помещенных в фигурные скобки.

```
using System;
namespace app1
{
 class Program
  {
   public static void Main(string[] args)
    {
     Console.WriteLine("Hello World!");
     Console.Write("Press any key to continue...");
     Console.ReadKey(true);
    }
  }
}
```

Структура программы C#

using System;. Это говорит о том, что среда программирования обязательно подключает к создаваемому приложению набор системных программных средств, часть из которых может понадобиться программисту при создании приложения.

Например, могут потребоваться средства ввода-вывода данных, работы с базами данных, графические средства и т. п.

Пространство System содержит полный набор таких средств, программист в ходе создания программы выделяет из System необходимые подпространства и средства.

Например, подпространство Console, которое содержит средства консольного ввода-вывода, т. е. ввода-вывода без графического интерфейса, а только используя консольное окно.

Структура программы C#

Namespace app1. Является заголовком создаваемого приложения (программы). Фактически это заголовок блока строк, помещенных между открывающей и закрывающей фигурными скобками (первой и последней).

Создаваемое приложение всегда помещается в пространство имен, которому автоматически присваивается имя самого приложения. Можно самостоятельно изменить на другое.

В блок namespace app1 входит блок (ограниченный своей парой фигурных скобок), названный **class Program**.

В C# приложение оформляется в виде специальной структуры - класса, а сама программа задается как член этой структуры со своим заголовком (в данном случае — это public static void Main(string[] args)) и своим блоком строк, помещенных в фигурные скобки.

Структура программы C#

Main значит "главный". Это - точка входа в программу, место, с которого начнет выполняться программа. Компилятор по этому ключевому слову задает адрес в памяти, начиная с которого программа запустится на выполнение.

Конструкция Main оформлена в виде элемента языка, называемого **функцией**.

Признаком функции, по которому компилятор ее отличает от других элементов программы, являются открывающая и закрывающая простые скобки, внутри которых задаются (а могут и не задаваться) параметры функции.

Параметры служат для взаимодействия данной программы с другими (которые тоже начинаются с Main()) путем обмена данными через эти параметры. public static void Main()

Структура программы C#

Консольные средствами ввода-вывода.

Все эти средства - это функции и находятся в специальной структуре под названием Console, которая входит в общую систему System.

Чтобы отметить (для компилятора), куда входят средства ввода-вывода, их имена пишутся через точку от имени их родителя Console.

Весь ввод-вывод для консоли представляется в виде потока символов (т. е. символы передаются на устройство ввода-вывода по одному друг за другом).

Структура программы C#

С учетом этого:

- **WriteLine** (параметр - строка текста) вставляет в поток вывода строку текста вместе с символом перевода строки и возврата каретки: это специальный управляющий символ, который, когда его прочитает средство вывода на экран, заставляет перевести вывод на следующую строку экрана, начиная с самой левой позиции.
- **Write**(параметр - строка текста). Делает то же, что и WriteLine(),но не формирует символа перевода строки и возврата каретки. Если применять несколько ряд подряд это средство вывода, то выводимые на экран строки будут помещаться одна за другой без перехода на новую строку.
- **ReadLine().** Обеспечивает ввод строк с клавиатуры (говорят: выдает данные из входного потока, пока не будет нажата клавиша <Enter>). Введенную строку помещает в определенную пользователем переменную.
- **Read().** Вводит один символ с клавиатуры. Введенный символ помещает в определенную пользователем