**КПО – Наркевич**

Оглавление

[1. Современные подходы к разработке программного обеспечения (ПО): понятие конструирования ПО, основные этапы конструирования ПО. 4](#_Toc106063743)

[2. Современные подходы к разработке программного обеспечения: понятие конструирования ПО; результат конструирования; решения, которые принимаются при конструировании. 4](#_Toc106063744)

[3. Система программирования: определение, состав, назначение. Классический жизненный цикл разработки ПО. Текстовый редактор. Интегрированная среда разработки. Программный продукт. Общая схема преобразования исходного кода в процесс операционной системы. 5](#_Toc106063745)

[4. Интегрированная среда разработки: определение, состав, назначение. Транслятор: виды трансляторов. Транслятор: определение, назначение, примеры. Интерпретатор: 6](#_Toc106063746)

[определение, назначение, примеры. 6](#_Toc106063747)

[5. Язык программирования: определение, назначение, примеры. Исходный код. Язык программирования. Алфавит языка. Транслятор: определение, назначение, примеры. Объектный код, объектный модуль. Общая схема преобразования исходного кода в процесс операционной системы. 7](#_Toc106063748)

[6. Компоновщик: определение, назначение. Загрузочный модуль. Загрузчик: определение, назначение. Общая схема преобразования исходного кода в процесс операционной системы. 7](#_Toc106063749)

[7. Иерархическая структура компонентов в Visual C++: глобальный контейнер, компоненты, входящие в его состав. Структура проекта Visual C++. 8](#_Toc106063750)

[8. Иерархическая структура компонентов в Visual C++: многофайловый проект, предварительно откомпилированные заголовки: назначение, создание, применения. Пример (C++). 8](#_Toc106063751)

[9. Кодировка: определение, назначение, примеры. Кодировка ASCII. Структура кодировки Windows-1251. Отличие ASCII и Windows-1251. Символы времени выполнения: установка кодовой страницы языкового стандарта. 9](#_Toc106063752)

[10. Кодировка: определение, назначение, примеры. Кодировка UNICODE: назначение, структура, UCS, UTF. Прямой (LE) и обратный (BE) порядок байт. BOM: определение, назначение, примеры. 9](#_Toc106063753)

[11. Структура языка программирования: алфавит языка программирования, идентификаторы, зарезервированные идентификаторы, литералы, ключевые слова, определения. Правила составления идентификаторов в C++ и других языков программирования. Примеры. 10](#_Toc106063754)

[12. Структура языка программирования: фундаментальные типы данных. Что определяет тип данных? Определения, примеры (C++ и другие языки). 10](#_Toc106063755)

[13. Структура языка программирования: литералы, типы литералов, способы задания. Строки. Массивы данных фундаментального типа. Примеры (С++). 11](#_Toc106063756)

[14. Структура языка программирования: фундаментальные типы данных. Типы данных для хранения вещественных значений. Стандарт IEEE 754. Что определяет стандарт IEEE 754. 11](#_Toc106063757)

[15. Структура языка программирования: фундаментальные типы данных. Внутреннее представление величины целого типа, спецификаторы, диапазоны значений целого типа. 12](#_Toc106063758)

[16. Структура языка программирования: система типов языка программирования, Вывод типов. Преобразование типов: автоматическое преобразование, явное преобразование. Определения и примеры (С++). 12](#_Toc106063759)

[17. Структура языка программирования: система типов языка программирования, Вывод типов. Преобразование типов: расширяющее преобразование, сужающее преобразование. Назначение оператора sizeof яыка программирования С++. Примеры. 14](#_Toc106063760)

[18. Структура языка программирования: пользовательские типы данных (структуры, перечисления, объединения). Определение, примеры (C++). Массивы данных пользовательского типа. Перегрузка операторов для пользовательских типов. 14](#_Toc106063761)

[19. Структура языка программирования: пользовательские типы данных. Определение структуры, примеры (C++). Перегрузка операторов для пользовательских типов. 15](#_Toc106063762)

[20. Структура языка программирования: понятие конфликта имен, область видимости переменных, пространства имен, псевдонимы пространства имен. 16](#_Toc106063763)

[21. Структура языка программирования: определение выражения, его состав, порядок вычисления выражения. Символ окончания последовательности. Примеры (С++). 16](#_Toc106063764)

[22. Структура языка программирования: выражения (lvalue, rvalue, побочные эффекты, точка последовательности, унарные, бинарные и тернарные выражения), константные выражения, укороченное вычисление. Примеры (С++). 16](#_Toc106063765)

[23. Структура языка программирования: инструкции языка программирования, объявление, определение, инициализация, простые и составные инструкции, инструкции выбора, инструкции циклов, инструкции переходов, примеры (С++). 17](#_Toc106063766)

[24. Структура языка программирования: инструкции языка программирования, инструкции обработки исключений, примеры (С++). 17](#_Toc106063767)

[25. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Функции: передача параметров по значению и по ссылке, передача переменного числа параметров в функции C++. Примеры. 18](#_Toc106063768)

[26. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Передача параметров по значению и по ссылке, передача значений параметров по умолчанию в функции C++. Примеры. 18](#_Toc106063769)

[27. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций: лямбда-функции в С++. Примеры. 19](#_Toc106063770)

[28. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций в C++. Перегружаемые функции в C++. Inline-функции в C++. Шаблоны функций С++. 21](#_Toc106063771)

[29. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций в C++: соглашения о вызовах (\_cdecl). 21](#_Toc106063772)

[31. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций в C++: соглашения о вызовах (\_fastcall). 23](#_Toc106063773)

[32. Структура языка программирования: стандартная библиотека. Реализация стандартной библиотеки STL в C++. Понятие контейнера, итератора и алгоритма. Стандартные функции STL для работы со строками: копирование, сравнение, вычисление длины, поиск символа и подстроки. 24](#_Toc106063774)

[33. Классы памяти: код, стек, статические данные, динамическая область памяти. Структура динамической памяти (Heap) C++. Фрагментация динамической памяти C++. 25](#_Toc106063775)

[34. Механизм обработки исключений: определение, назначение, применение. Реализация обработки исключений в C++. Пример. 25](#_Toc106063776)

[35. Препроцессор: определение, назначение, применение, директивы, выражения, макросы, директивы условной компиляции. Примеры на C++. 26](#_Toc106063777)

[36. Статическая библиотека: определение, назначение, применение. Реализация статической библиотеки в Visual C++. Утилита LIB. Создание статической библиотеки. Применение директивы препроцессора #pragma comment для компоновки со статической библиотеки. 28](#_Toc106063778)

[37. Теория формальных языков. Определения: алфавит языка, цепочка, пустая цепочка, длина цепочки, равенство (эквивалентность) цепочек, конкатенация цепочек, итерация цепочки, операторы + и \*. Примеры. 28](#_Toc106063779)

[38. Теория формальных языков. Определение формального языка, эквивалентность двух языков, способы задания формального языка. Лексика, синтаксис и семантика языка. Примеры. 29](#_Toc106063780)

[39. Теория формальных языков. Формальная грамматика (определение, назначение), выводимость цепочки символов в грамматике, сентенциальная форма грамматики, язык, порождаемый грамматикой, способы задания грамматик. Примеры. 30](#_Toc106063781)

[40. Теория формальных языков. Определение порождающей грамматики. Определение вывода, левосторонний и правосторонний выводы, дерево вывода. Примеры. 31](#_Toc106063782)

[41. Теория формальных языков: способы задания грамматик. Общая характеристика формы Бэкуса-Наура. Расширенная БНФ. Примеры. 32](#_Toc106063783)

[42. Классификация языков и грамматик по Хомскому: иерархия Хомского (неограниченные, контекстно-зависимые, контекстно-независимые, регулярные грамматики и вид правил этих грамматик), соотношения грамматик и порождаемых ими языков. 33](#_Toc106063784)

[43. Лексический анализ: определение, назначение, применение. Лексический анализатор: место в структуре транслятора, входная и выходная информация, способы взаимодействия с синтаксическим анализатором (последовательное и параллельное). 34](#_Toc106063785)

[44. Лексический анализ: определение, назначение, применение. Лексический анализатор: вид правил регулярной грамматики, нисходящий и восходящий разбор цепочек, дерево разбора. 34](#_Toc106063786)

[45. Регулярные выражения: рекурсивное определение, назначение, применение, метасимволы, примеры. Пример применения в функциях стандартной библиотеке STL C++. 34](#_Toc106063787)

[46. Конечный автомат (КА): определение, назначение, схема работы КА, примеры. 34](#_Toc106063788)

[47. Конечный автомат (КА): недетерминированные и детерминированные КА. Мгновенное описание КА. Последовательность мгновенных описаний, диаграмма мгновенных описаний. Пример. 35](#_Toc106063789)

[48. Конечный автомат: определение, схема работы, примеры. Соотношение регулярного языка, регулярной грамматики, регулярного языка и конечного автомата. 35](#_Toc106063790)

[49. Конечный автомат: определение графа переходов конечного автомата и метод его построение по регулярному выражению. Примеры построения графов переходов. 35](#_Toc106063791)

[50. Конечный автомат: алгоритм разбора цепочки символов, основанный на двух массивах. 39](#_Toc106063792)

### 1. Современные подходы к разработке программного обеспечения (ПО): понятие конструирования ПО, основные этапы конструирования ПО.

Конструирование ПО - это дисциплина программной инженерии. Это детальное создание работающего значимого программного обеспечения с помощью комбинации кодирования, верификации, модульного тестирования, интеграционного тестирования и отладки.

Существуют три стратегии конструирования ПО:

* каскадная стратегия — линейная последовательность этапов конструирования
* инкрементная стратегия — итерационное повторение проходов с целью наращивания функциональности ПО
* эволюционная стратегия — то же, что инкрементная, плюс постепенное уточнение требований

Основные этапы конструирования ПО:

− анализ;

− выработка требований;

− создание плана конструирования;

− разработка архитектуры ПО;

− детальное проектирование;

− кодирование и отладка;

− блочное тестирование;

− интеграционное тестирование;

− интеграция;

− тестирование системы;

− корректирующее сопровождение.

### 2. Современные подходы к разработке программного обеспечения: понятие конструирования ПО; результат конструирования; решения, которые принимаются при конструировании.

Существуют три стратегии конструирования ПО:

* каскадная стратегия — линейная последовательность этапов конструирования
* инкрементная стратегия — итерационное повторение проходов с целью наращивания функциональности ПО
* эволюционная стратегия — то же, что инкрементная, плюс постепенное уточнение требований

Конструирование ПО - это дисциплина программной инженерии. Это детальное создание работающего значимого программного обеспечения с помощью комбинации кодирования, верификации, модульного тестирования, интеграционного тестирования и отладки.

Результат конструирования – исходный код – часто является единственным верным описанием программы.

Основные решения, которые принимаются при конструировании:

− выбор языка программирования;

− конвенции программирования;

− выбор технологий;

− выбор основных методик конструирования.

### 3. Система программирования: определение, состав, назначение. Классический жизненный цикл разработки ПО. Текстовый редактор. Интегрированная среда разработки. Программный продукт. Общая схема преобразования исходного кода в процесс операционной системы.

Система программирования – комплекс программных средств, предназначенных для автоматизации процесса разработки, отладки программного обеспечения и подготовки программного кода к выполнению.

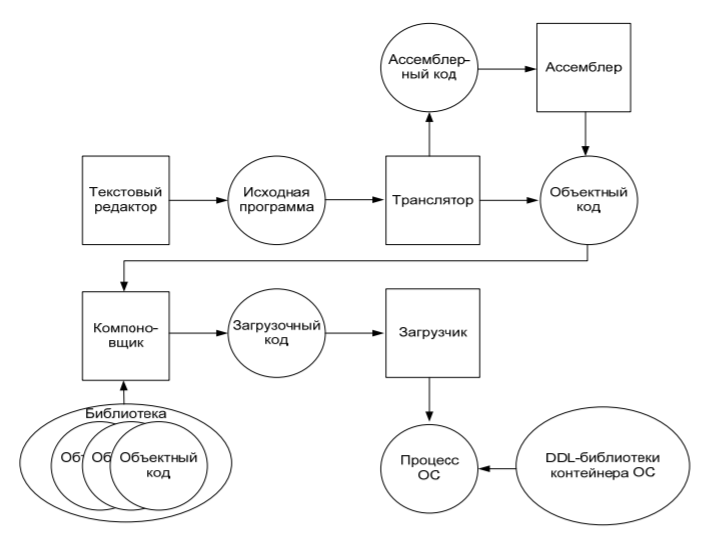
Состав системы программирования: компилятор или интерпретатор, интегрированная среда разработки, средства создания и редактирования кода, библиотеки стандартных программ и функций, отладочные программы, диалоговая среда, многооконный режим работы, мощные графические библиотеки, утилиты для работы с библиотеками, ассемблер, справочная служба.



Текстовый редактор – программа, позволяющая подготовить исходный код программы.

Интегрированная среда разработки – набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы.

Программный продукт – программа, работающая без авторского присутствия. Программный продукт исполняется, тестируется, конфигурируется без присутствия автора и сопровождается документацией.



### 4. Интегрированная среда разработки: определение, состав, назначение. Транслятор: виды трансляторов. Транслятор: определение, назначение, примеры. Интерпретатор:

### определение, назначение, примеры.

Интегрированная среда разработки – набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы.

Транслятор – программа или техническое средство, выполняющее трансляцию программы.

Трансляция программы - преобразование программы, представленной на одном из языков программирования, в программу на другом языке.

Виды трансляторов:

* Адресный. Функциональное устройство, преобразующее виртуальный адрес (Virtual address) в реальный адрес памяти (Memory address).
* Диалоговый. Обеспечивает использование языка программирования в [режиме разделения времени](https://www.tadviser.ru/index.php?title=%D0%A0%D0%B5%D0%B6%D0%B8%D0%BC_%D1%80%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8&action=edit&redlink=1).
* Многопроходной. Формирует [объектный модуль](https://www.tadviser.ru/index.php?title=%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C&action=edit&redlink=1) за несколько просмотров исходной программы.
* Обратный. То же, что [детранслятор](https://www.tadviser.ru/index.php?title=%D0%94%D0%B5%D1%82%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80&action=edit&redlink=1). См. также: [декомпилятор](https://www.tadviser.ru/index.php?title=%D0%94%D0%B5%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80&action=edit&redlink=1" \o "Декомпилятор (такой страницы не существует)), [дизассемблер](https://www.tadviser.ru/index.php?title=%D0%94%D0%B8%D0%B7%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80&action=edit&redlink=1).
* Однопроходной. Формирует объектный модуль за один последовательный просмотр исходной программы.
* Оптимизирующий. Выполняет [оптимизацию](https://www.tadviser.ru/index.php?title=%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80%D0%B0&action=edit&redlink=1) кода в создаваемом объектном модуле.
* Синтаксически-ориентированный (синтаксически-управляемый). Получает на вход описание [синтаксиса](https://www.tadviser.ru/index.php?title=%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)&action=edit&redlink=1) и [семантики](https://www.tadviser.ru/index.php?title=%D0%A1%D0%B5%D0%BC%D0%B0%D0%BD%D1%82%D0%B8%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)&action=edit&redlink=1) языка и текст на описанном языке, который и транслируется в соответствии с заданным описанием.
* Тестовый. Набор [макрокоманд](https://www.tadviser.ru/index.php?title=%D0%9C%D0%B0%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%B0&action=edit&redlink=1) [языка ассемблера](https://www.tadviser.ru/index.php/%D0%A1%D1%82%D0%B0%D1%82%D1%8C%D1%8F:%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0), позволяющих задавать различные [отладочные](https://www.tadviser.ru/index.php?title=%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B&action=edit&redlink=1) [процедуры](https://www.tadviser.ru/index.php?title=%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0&action=edit&redlink=1) в программах, составленных на языке ассемблера.

Интерпретатор – разновидность транслятора. Переводит и выполняет программу с языка высокого уровня в машинный код строка за строкой.

Пример: iGCC — тонкая обёртка над GCC, написанная на Python. Самый легковесный из интерпретаторов

### 5. Язык программирования: определение, назначение, примеры. Исходный код. Язык программирования. Алфавит языка. Транслятор: определение, назначение, примеры. Объектный код, объектный модуль. Общая схема преобразования исходного кода в процесс операционной системы.

Язык программирования – формальная знаковая система, предназначенная для записи компьютерных программ. Знаковая система определяет набор лексических, синтаксических и семантических правил написания программы (программного кода). Язык программирования представляется в виде набора спецификаций, определяющих его синтаксис и семантику.

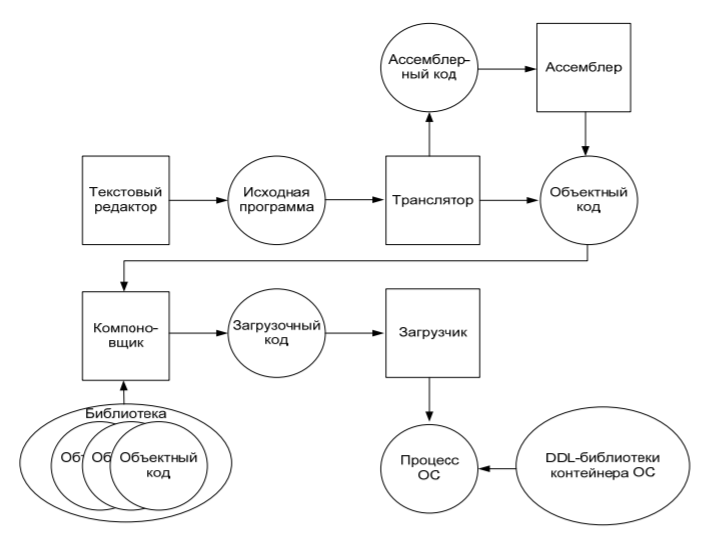
Алфавит языка программирования: набор символов, разрешенных к использованию языком программирования.

Исходный код (исходная программа) – программа, написанная на языке программирования, в текстовом формате. Программа на исходном языке (исходный код) готовится с помощью текстовых редакторов и в виде текстового файла или раздела библиотеки поступает на вход транслятора.

Объектный код – результат работы транслятора. Один файл объектного кода – объектный модуль.

Объектный модуль –  файл с промежуточным представлением отдельного модуля программы, полученный в результате обработки исходного кода компилятором.

Транслятор – программа или техническое средство, выполняющее трансляцию программы. Трансляция программы - преобразование программы, представленной на одном из языков программирования, в программу на другом языке.

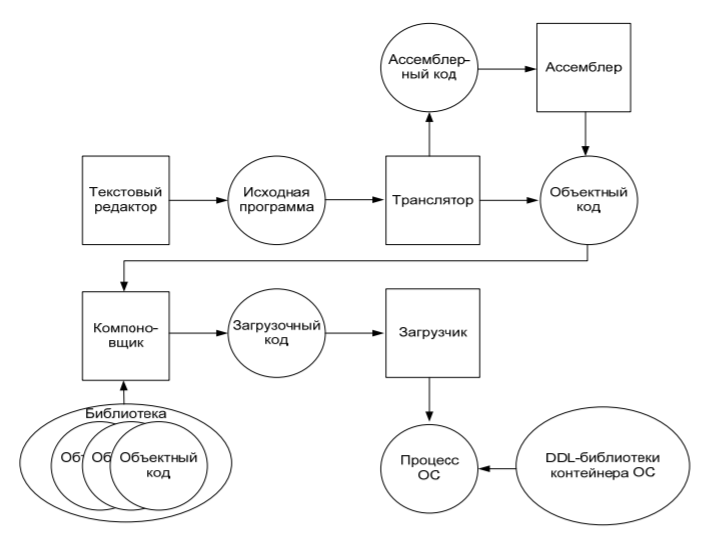


### 6. Компоновщик: определение, назначение. Загрузочный модуль. Загрузчик: определение, назначение. Общая схема преобразования исходного кода в процесс операционной системы.

Компоновщик: программа, принимающая один или несколько объектных модулей и формирующая на их основе загрузочный модуль. Если программа собирается из нескольких объектных файлов, компоновщик может собирать эти файлы в единый исполнимый модуль, вычисляя и подставляя адреса вместо символов, в течение времени компоновки (статическая компоновка) или во время исполнения (динамическая компоновка).

Загрузочный модуль – это готовая программа в машинных кодах, которая может быть записана во внутреннюю память микроконтроллера.

Загрузчик (loader): программа, обычно входящая в состав операционной системы, предназначенная для запуска процесса операционной системы на основе загрузочного модуля.



### 7. Иерархическая структура компонентов в Visual C++: глобальный контейнер, компоненты, входящие в его состав. Структура проекта Visual C++.

Иерархическая структура компонентов в Visual C++.

Глобальный контейнером (компонент, включающий в себя другие компоненты) является Решение. Решение может содержать один или несколько проектов. Проекты являются независимыми компонентами. Они имеют собственную структуру, состоящую из четырех основных каталогов:

* Внешние зависимости – содержит ссылки на все модули, которые использует программа.
* Файлы заголовков – содержит файлы кода С++ с расширением h.
* Исходные файлы – содержит файлы кода С++ с расширением срр.
* Файлы ресурсов – содержит файлы, непосредственно не относящиеся к языку С++, но необходимые для работы приложения.

Например, мультимедийные файлы. Код программного проекта может иметь сложную структуру и состоять из нескольких файлов исходного кода и конфигурационных файлов.

### 8. Иерархическая структура компонентов в Visual C++: многофайловый проект, предварительно откомпилированные заголовки: назначение, создание, применения. Пример (C++).

Иерархическая структура компонентов в Visual C++.

- Глобальный контейнером (компонент, включающий в себя другие

компоненты) является Решение. Решение может содержать один или

несколько проектов.

- Проекты являются независимыми компонентами. Они имеют

собственную структуру, состоящую из четырех основных каталогов:

- Внешние зависимости – содержит ссылки на все модули, которые

использует программа.

- Файлы заголовков – содержит файлы кода С++ с расширением h.

- Исходные файлы – содержит файлы кода С++ с расширением срр.

- Файлы ресурсов – содержит файлы, непосредственно не относящиеся к

языку С++, но необходимые для работы приложения. Например,

мультимедийные файлы.

Код программного проекта может иметь сложную структуру и состоять

из нескольких файлов исходного кода и конфигурационных файлов.

Пример – stdafx.h. Цель файла — ускорить процесс сборки.

### 9. Кодировка: определение, назначение, примеры. Кодировка ASCII. Структура кодировки Windows-1251. Отличие ASCII и Windows-1251. Символы времени выполнения: установка кодовой страницы языкового стандарта.

Кодировка - это правила перевода одного набора символов в другой.

ASCII — American Standard Code for Information Interchange. ASCII была разработана (1963 год) для кодирования символов, коды которых помещались в 7 бит (128 символов). Со временем кодировка была расширена до 8-ми бит (256 символов), коды первых 128-и символов не изменились.

Windows-1251 — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии.

в Windows 1251 можно закодировать всего 256 символов. Так как все сводится к двоичной системе исчисления, а байт – это 8 бит (0 и 1), то и максимальное число сочетаний составляет 28 = 256.

**Набор символов времени выполнения:** символы, отображаемыми в среде выполнения. Любые дополнительные символы зависят от локализации

Кодовая страница — это набор символов, который может содержать числа, знаки пунктуации и другие глифы. Различные языки и языковые стандарты могут использовать разные кодовые страницы.

### 10. Кодировка: определение, назначение, примеры. Кодировка UNICODE: назначение, структура, UCS, UTF. Прямой (LE) и обратный (BE) порядок байт. BOM: определение, назначение, примеры.

Кодировка - это правила перевода одного набора символов в другой.

UNICODE: применение этого стандарта позволяет закодировать очень большое число символов из разных письменностей: в документах Unicode могут соседствовать китайские иероглифы, математические символы, буквы греческого алфавита, латиницы и кириллицы, при этом становится ненужным переключение кодовых страниц.

− UCS – universal character set (универсальный набор символов) – представляет собой стандартный набор символов, определенный международным стандартом ISO/IEC 10646, который является основой многих символьных кодировок;

− UTF – Unicode transformation format (семейство кодировок) –  распространённый стандарт кодирования символов, позволяющий более компактно хранить и передавать символы Юникода, используя переменное количество байт

Порядок следования байтов:

− LE (Little endian order, прямой порядок, от младшего к старшему);

− BE (Big endian order, обратный порядок, от старшего к младшему).

BOM (Byte Order Mark) — маркер последовательности байтов.

BOM назначения: Для определения формата представления Юникода в начало текстового файла записывается сигнатура (обозначение) — символ U+FEFF

### 11. Структура языка программирования: алфавит языка программирования, идентификаторы, зарезервированные идентификаторы, литералы, ключевые слова, определения. Правила составления идентификаторов в C++ и других языков программирования. Примеры.

Структура языка программирования

- алфавит языка программирования: набор символов, разрешенных к использованию языком программирования. Основывается на одной из кодировок.

- Идентификатор — это последовательность символов, используемая для обозначения одного из элементов;

- литералы — это некоторое выражение, создающее объект.;

- ключевые слова - это определенные зарезервированные слова, которые имеют предопределенное значение в C++.

- фундаментальные (встроенные) типы данных: o предопределенные типы данных;

- пользовательские типы данных o типы, которые может создавать пользователь на основе фундаментальных типов (возможно описание их свойств и поведение); o массивы фундаментальных типов;

- преобразование типов: явное и неявное (автоматическое).

- инициализация памяти: присвоение значения в момент объявления переменной;

- константное выражение: выражение, которое должно быть вычислено на этапе компиляции;

- область видимости переменных: доступность переменных по их идентификатору в разных частях программы; пространства имен;

- выражения

- инструкции языка: инструкция — это некое элементарное действие, несколько идущих подряд инструкций образуют блок вычислений (последовательность инструкций); o присваивания; o инструкции объявления; o блок вычислений; o ветвление; o циклы; o инструкции перехода; o обработка исключений

программные конструкции (декомпозиция программного кода): процедуры, функции, методы, ...

Литерал — это некоторое выражение, создающее объект.

Определение предоставляет компилятору все сведения, необходимые для создания машинного кода при последующем использовании сущности в программе.

Рассмотрим правила построения идентификаторов из букв алфавита (C++ три):

1. Первым символом идентификатора C++ может быть только буква.
2. Следующими символами идентификатора могут быть буквы, буквы-цифры и буквы-подчерки.
3. Длина идентификатора неограниченна (фактически же длина зависит от реализации системы программирования).

Примеры свои: x, Max, Delta, my\_method, w23x5

Зарезервированные: auto, bool, break, case, catch, char, class…

### 12. Структура языка программирования: фундаментальные типы данных. Что определяет тип данных? Определения, примеры (C++ и другие языки).

Фундаментальные типы (или встроенные типы) задаются стандартом языка и встроены в компилятор.

Фундаментальные типы C++ определены следующие ключевые слова:

- int (целый);

- char (символьный);

- wchar\_t (расширенный символьный);

- bool (логический);

- float (вещественный);

- double (вещественный с двойной точностью);

- тип void.

Тип данных определяет:

- внутреннее представление данных в памяти компьютера;

- диапазон значений, которые могут принимать величины этого типа;

- операции и функции, которые можно применять к величинам этого типа.

Java 8 примитивные типы: boolean, byte, char, short, int, long, float, double.

### 13. Структура языка программирования: литералы, типы литералов, способы задания. Строки. Массивы данных фундаментального типа. Примеры (С++).

Литерал - это само фиксированное значение, которое присваивается переменной-константе или ещё каким-то образом используется в программе.

Типы литералов:

1. [Целочисленные литералы](https://translated.turbopages.org/proxy_u/en-ru.ru.2ba44347-628defe4-8a6bba73-74722d776562/https/www.geeksforgeeks.org/integer-literal-in-c-cpp-prefixes-suffixes/). Они используются для представления и хранения целочисленных значений.
2. Литералы с плавающей запятой. Они используются для представления и хранения действительных чисел. Действительное число имеет целочисленную часть, действительную часть, дробную часть и экспоненциальную часть.
3. Символьный литерал. Это относится к литералу, который используется для хранения одного символа в одной кавычке.
4. Строковые литералы. Строковые литералы похожи на символьные литералы, за исключением того, что они могут хранить несколько символов и используют двойные кавычки для их хранения.
5. Булевы литералы. Этот литерал предоставляется **только в C ++**, а **не в C.** Они используются для представления булевых типов.

Массивы данных фундаментальных типов: коллекция однородных данных, размещенных последовательно в памяти и допускающие доступ по индексу (вычисляется: смещение = индекс\*sizeof(базовый\_тип)).

### 14. Структура языка программирования: фундаментальные типы данных. Типы данных для хранения вещественных значений. Стандарт IEEE 754. Что определяет стандарт IEEE 754.

Фундаментальные типы C++

определены следующие ключевые слова:

- int (целый);

- char (символьный);

- wchar\_t (расширенный символьный);

- bool (логический);

- float (вещественный);

- double (вещественный с двойной точностью);

- тип void.

Стандарт языка C++ определяет три типа данных для хранения вещественных значений: float, double и long double. Стандарт IEEE 754 описывает формат представления чисел с плавающей точкой. Используется в программных и аппаратных реализациях арифметических действий (математических операций).

Стандарт описывает:

− формат чисел с плавающей точкой: мантисса, экспонента, знак числа;

− представление положительного и отрицательного нуля, положительной и отрицательной бесконечностей, а также нечисла (англ. Not-a-Number, NaN);

− методы, используемые для преобразования числа при выполнении математических операций;

− исключительные ситуации: деление на ноль, переполнение, потеря значимости, работа с денормализованными числами и другие;

− операции: арифметические и другие.

### 15. Структура языка программирования: фундаментальные типы данных. Внутреннее представление величины целого типа, спецификаторы, диапазоны значений целого типа.

Фундаментальные типы C++

определены следующие ключевые слова:

* int (целый);
* char (символьный);
* wchar\_t (расширенный символьный);
* bool (логический);
* float (вещественный);
* double (вещественный с двойной точностью);
* тип void.

Внутреннее представление величины целого типа:

* целое число в двоичном коде.
* спецификатор signed – старший разряд (бит) числа интерпретируется как знаковый (0 – положительное число, 1 – отрицательное).
* спецификатор unsigned: старший разряд (бит) рассматривается как значащий, позволяет представлять только положительные числа.

По умолчанию все целочисленные типы считаются знаковыми, то есть спецификатор signed можно опускать. Диапазон значений зависит от реализации.

### 16. Структура языка программирования: система типов языка программирования, Вывод типов. Преобразование типов: автоматическое преобразование, явное преобразование. Определения и примеры (С++).

Система типов – совокупность правил в языках программирования, назначающих свойства, называемыми типами, различным конструкциям, составляющим программу (переменным, выражениям, функциям, модулям) для возможности выполнения проверки типов во время компиляции или во время выполнения,

Основные функции системы типов данных:

− обеспечение безопасности: проверяется каждая операция на получение аргументов именно тех типов, для которых она имеет предназначена;

− оптимизация: на основе типа выбирается способ эффективного хранения значения и алгоритмов его обработки;

− документация: подчеркивается намерения программиста;

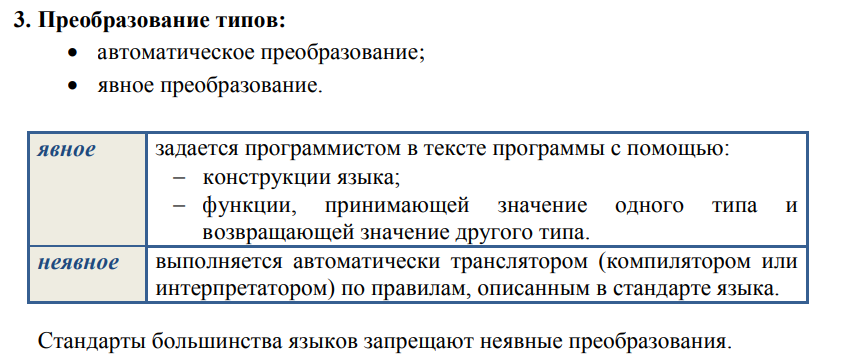
− абстракция: использование типов данных высокого уровня позволяет программисту думать о значениях как о высокоуровневых сущностях, а не как о наборе битов.

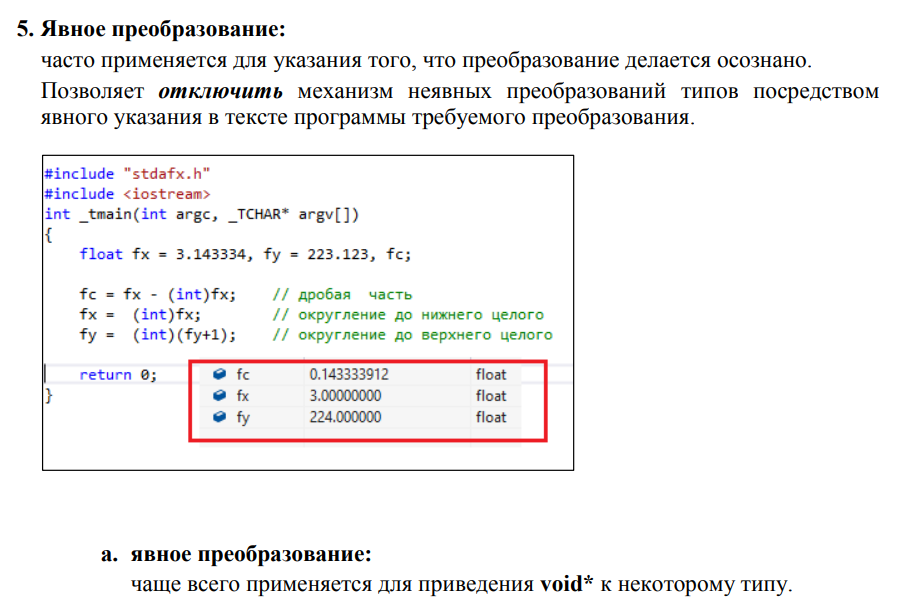
Вывод типов

В стандарте C++11 введено новое ключевое слово auto для определения явно инициализируемой переменной. Создается переменная, тип которой выводится из инициализирующего значения:

auto variable1 = 5;

auto variable2 = 2.5;





### 17. Структура языка программирования: система типов языка программирования, Вывод типов. Преобразование типов: расширяющее преобразование, сужающее преобразование. Назначение оператора sizeof яыка программирования С++. Примеры.

Система типов – совокупность правил в языках программирования, назначающих свойства, называемыми типами, различным конструкциям, составляющим программу (переменным, выражениям, функциям, модулям) для возможности выполнения проверки типов во время компиляции или во время выполнения,

Основные функции системы типов данных:

− обеспечение безопасности: проверяется каждая операция на получение аргументов именно тех типов, для которых она имеет предназначена;

− оптимизация: на основе типа выбирается способ эффективного хранения значения и алгоритмов его обработки;

− документация: подчеркивается намерения программиста;

− абстракция: использование типов данных высокого уровня позволяет программисту думать о значениях как о высокоуровневых сущностях, а не как о наборе битов.

Вывод типов

В стандарте C++11 введено новое ключевое слово auto для определения явно инициализируемой переменной. Создается переменная, тип которой выводится из инициализирующего значения:

auto variable1 = 5;

auto variable2 = 2.5;

При классификации по диапазону значений различают:

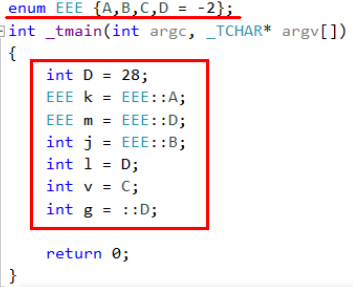
* сужающее преобразование типов. В этом случае больший тип данных сужается к меньшему (сужается диапазон значений). Здесь может возникнуть потеря данных
* расширяющее преобразования типов. При таком преобразовании меньший тип данных расширяется до большего типа данных (расширяется диапазон значений)

В С/С++ размер переменной любого типа данных зависит от компилятора и/или архитектуры компьютера. Фактический размер переменных может отличаться на разных компьютерах. Для его определения используют оператор sizeof.

### 18. Структура языка программирования: пользовательские типы данных (структуры, перечисления, объединения). Определение, примеры (C++). Массивы данных пользовательского типа. Перегрузка операторов для пользовательских типов.

пользовательские типы данных - типы, которые может создавать пользователь на основе фундаментальных типов (возможно описание их свойств и поведение);

* массивы фундаментальных типов;
* enum;
* struct;
* union;
* typedef;
* class (спецификатор типа).,



**Структура** **в** языке C++ представляет собой производный тип данных, который представляет какую-то определенную сущность, также как и класс.

**Перечисление** — это пользовательский тип, состоящий из набора целочисленных констант, называемых перечислителями.

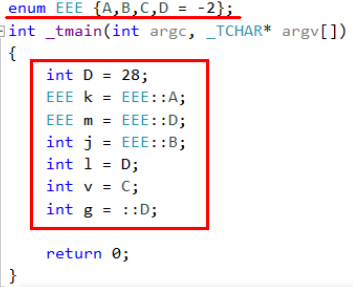
**Объединения** - это объект, позволяющий нескольким переменным различных типов занимать один участок памяти.

Массивы данных фундаментальных типов: коллекция однородных данных, размещенных последовательно в памяти и допускающие доступ по индексу (вычисляется: смещение = индекс\*sizeof(базовый\_тип)).

### 19. Структура языка программирования: пользовательские типы данных. Определение структуры, примеры (C++). Перегрузка операторов для пользовательских типов.

пользовательские типы данных - типы, которые может создавать пользователь на основе фундаментальных типов (возможно описание их свойств и поведение);

* массивы фундаментальных типов;
* enum;
* struct;
* union;
* typedef;
* class (спецификатор типа).,



Структура в языке C++ представляет собой производный тип данных, который представляет какую-то определенную сущность, также как и класс.

Перечисление — это пользовательский тип, состоящий из набора целочисленных констант, называемых перечислителями.

Объединения - это объект, позволяющий нескольким переменным различных типов занимать один участок памяти.

### 20. Структура языка программирования: понятие конфликта имен, область видимости переменных, пространства имен, псевдонимы пространства имен.

Точно так же C++ требует, чтобы все идентификаторы были однозначными. Если два идентичных идентификатора вводятся в одну и ту же программу таким образом, что компилятор или компоновщик не может их различить, компилятор или компоновщик выдаст ошибку. Эта ошибка обычно называется **конфликтом имен** (или **коллизией имен**).

зависит от места, где переменная была объявлена. Если какая-то переменная объявляется в теле метода, в цикле или условной конструкции, она исчезает после завершения работы этого метода, цикла или ветки.

Пространство имен — это совокупность определенных в настоящий момент символических имен и информации об объектах, на которые они ссылаются.

Псевдоним пространства имен используется для задания короткого синонима имени пространства. Например, длинное имя

namespace International\_Business\_Machines

{ /\* ... \*/ }

namespace IBM = International\_Business\_Machines;

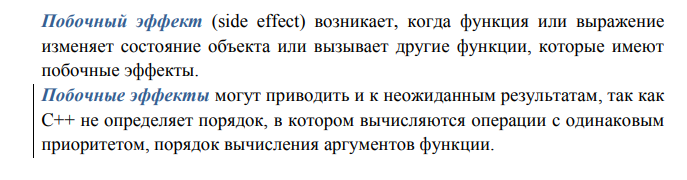
### 21. Структура языка программирования: определение выражения, его состав, порядок вычисления выражения. Символ окончания последовательности. Примеры (С++).

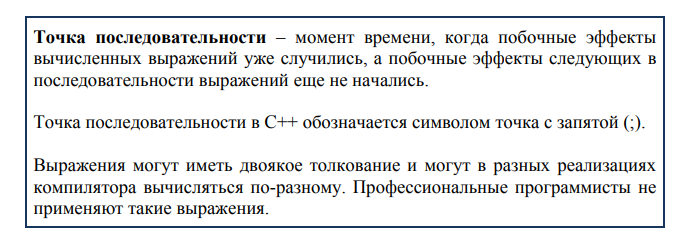
Выражения — это последовательности операторов и операндов, используемые в следующих целях.

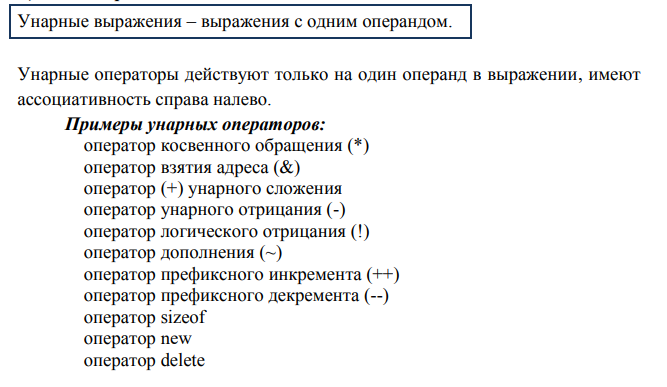
* Вычисление значения из операндов
* Назначение объектов или функций
* Создание побочных эффектов. (Побочные эффекты — это любые действия, отличные от вычисления выражения, например изменение значения объекта.)

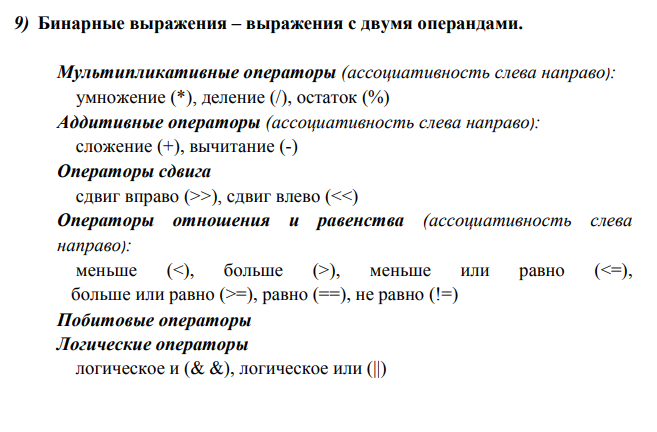
**char john[5] = "John"; // this is {'J', 'o', 'h', 'n', 0}**

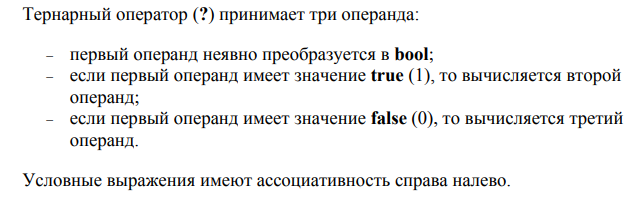
### 22. Структура языка программирования: выражения (lvalue, rvalue, побочные эффекты, точка последовательности, унарные, бинарные и тернарные выражения), константные выражения, укороченное вычисление. Примеры (С++).











Категории значений, можно встретить в сообщениях об ошибке

lvalue – левостороннее значение

rvalue – правостороннее значение

унарные: + , - , ++ , -- , \* , &

бинарные: \* , / , && , || , = , \*= , += …

тернарные: ?:

укороченное вычисление && ||

### 23. Структура языка программирования: инструкции языка программирования, объявление, определение, инициализация, простые и составные инструкции, инструкции выбора, инструкции циклов, инструкции переходов, примеры (С++).

Инстру́кция или опера́тор — наименьшая автономная часть языка программирования; команда или набор команд. Программа обычно представляет собой последовательность инструкций.

Объявление указывает уникальное имя сущности, а также сведения о его типе и других характеристиках.

*Определение* предоставляет компилятору все сведения, необходимые для создания машинного кода при последующем использовании сущности в программе.

программировании инициализация (или инициализация) - это присвоение начального значения объекту данных или переменной

Простейшей формой является пустая инструкция. Вот как она выглядит:

; // пустая инструкция

Пустая инструкция используется там, где синтаксис С++ требует употребления инструкции, а логика программы – нет.

Инструкция переходов: break, continue, goto

### 24. Структура языка программирования: инструкции языка программирования, инструкции обработки исключений, примеры (С++).

Инстру́кция или опера́тор — наименьшая автономная часть языка программирования; команда или набор команд. Программа обычно представляет собой последовательность инструкций.

В процессе работы программы могут возникать различные ошибки. Например, при передаче файла по сети оборвется сетевое подключение или будут введены некорректные и недопустимые данные, которые вызовут падение программы. Такие ошибки еще называются исключениями. Если исключение не обработано, то при его возникновении программа прекращает свою работу.

И в этом случае нам надо известить систему о возникшей ошибке. Для этого используется оператор throw.

Оператор throw генерирует исключение. Через оператор throw можно передать информацию об ошибке.

Но это исключение еще надо обработать в коде. Для обработки исключений применяется конструкция try...catch. Она имеет следующую форму:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | try  {      инструкции, которые могут вызвать исключение  }  catch(объявление\_исключения)  {      обработка исключения  } |

В блок после ключевого слова try помещается код, который потенциально может сгенерировать исключение.

После ключевого слова catch в скобках идет параметр, который передает информацию об исключении. Затем в блоке производится собственно обработка исключения.

### 25. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Функции: передача параметров по значению и по ссылке, передача переменного числа параметров в функции C++. Примеры.

Программные конструкции (декомпозиция программного кода): процедуры, функции, методы, ...

Блок — это последовательность операторов, заключенных в фигурные скобки и рассматриваются как одна программная единица. Операторы, составляющие блок, логически связаны друг с другом. Иногда блок называют составным оператором. Блок всегда начинается открывающейся фигурной скобкой { и заканчивается закрывающейся }. Блоки служат для ограничения области видимости переменных и функций, а также позволяют обращаться к блоку инструкций как к единой инструкции, могут быть пустыми или вложенными один в другой.

Процедура или функция представляет собой последовательность операторов, которая имеет имя, список параметров и может быть вызвана из различных частей программы. Функции, в отличие от процедур, в результате своего выполнения возвращают значение, которое может быть использовано в выражении.

Передача параметров по значению. Процедуре передается собственно значение параметра. При этом фактически значение параметра копируется, и процедура использует его копию, так что модификация исходного параметра оказывается невозможной. Этот механизм применяется для передачи небольших параметров, таких как байты или слова.

При передаче параметров функции по ссылке в формальный параметр функции передается не копия значения фактического параметра, а его адрес. Термин «ссылка» означает ссылку на область памяти. Функция, работая с формальным параметром, обращается по указанному в формальном параметре адресу и изменяет значение фактического параметра.

тип имя\_функции(обязательные параметры, ...)

**Передача переменного числа параметров в функции С++:**

По умолчанию параметры передаются в функцию через стек. Поэтому, технически, нет ограничения на количество передаваемых параметров. Функции с переменным числом параметров объявляются как обычные функции, но вместо недостающих аргументов ставится многоточие. Количество параметров и их типы известны только при вызове функции.

Передать функции параметры можно двумя способами:

* явно передать обязательный аргумент, задающий число параметров;
* добавить в конец списка параметр с уникальным значением, по которому будет определяться конец списка параметров.

Общий принцип работы следующий: в функции для доступа к списку параметров устанавливается указатель, значением которого будет адрес явного параметра в списке, далее изменяется значение этого указателя, чтобы переместиться на следующий параметр.

### 26. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Передача параметров по значению и по ссылке, передача значений параметров по умолчанию в функции C++. Примеры.

Блок — это последовательность операторов, заключенных в фигурные скобки и рассматриваются как одна программная единица. Операторы, составляющие блок, логически связаны друг с другом. Иногда блок называют составным оператором. Блок всегда начинается открывающейся фигурной скобкой { и заканчивается закрывающейся }. Блоки служат для ограничения области видимости переменных и функций, а также позволяют обращаться к блоку инструкций как к единой инструкции, могут быть пустыми или вложенными один в другой.

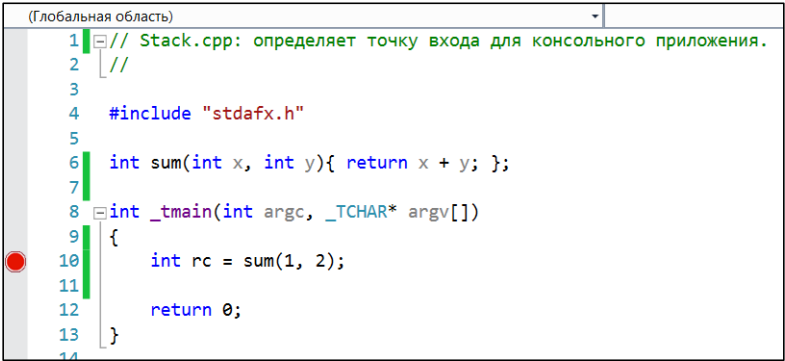
Процедура или функция представляет собой последовательность операторов, которая имеет имя, список параметров и может быть вызвана из различных частей программы. Функции, в отличие от процедур, в результате своего выполнения возвращают значение, которое может быть использовано в выражении.

Передача параметров в функцию происходит через стек. Код, вызывающий функцию, знает, сколько параметров ей передать и каковы значения этих параметров.

Стек - это область памяти, в которой поток может хранить данные, необходимые ему для выполнения. В стеке могут храниться локальные переменные, используемые кодом, временные переменные и т.д.

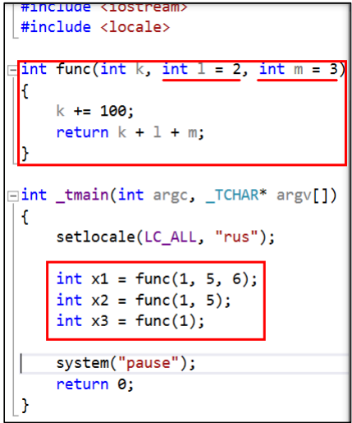
В нашем примере, если код вызывает функцию sum, которая принимает два параметра типа int, то вызывающий код: - кладет два параметра в стек с помощью двух инструкций push. В результате этого указатель стека (ESP) уменьшается на 2\*4 байта (вершина стека сдвигается на 8 байт); - выполняет инструкцию call, которая передает управление функции sum. При этом значение ESP уменьшается еще на 4 байта, потому что в стек помещается адрес точки возврата.

Вызываемая функция извлекает параметры из стека. При входе в функцию sum, в вершине стеке находится адрес точки возврата. Значение, хранящееся в ESP + 4 – это первый параметр, переданный в функцию sum. По адресу ESP + 8 хранится второй параметр функции sum.

****

Для вызова функции необходимо в требуемом месте программного кода указать имя функции и перечислить передаваемые в функцию параметры.

Передача параметров в функцию: по значению и по ссылке. Для переменной, переданной по значению, создаётся локальная копия и любые изменения, которые происходят в теле функции, происходят с локальной копией и не изменяют значения самой переменной. Для переменной, переданной по ссылке, изменения происходят с самой переменной.

При обращении к функции, можно опускать некоторые из ее аргументов. Для этого необходимо при объявлении прототипа функции проинициализировать эти параметры значениями, которые будут использоваться в функции, как значения по умолчанию. Аргументы, заданные по умолчанию, должны быть последними аргументами. Если при вызове функции не передавать ей значения формальных параметров l и m, то по умолчанию будут использоваться значения 2 и 3. В примере показаны различные способы использования функции func с аргументами по умолчанию.

### 27. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций: лямбда-функции в С++. Примеры.

Блок — это последовательность операторов, заключенных в фигурные скобки и рассматриваются как одна программная единица. Операторы, составляющие блок, логически связаны друг с другом. Иногда блок называют составным оператором. Блок всегда начинается открывающейся фигурной скобкой { и заканчивается закрывающейся }. Блоки служат для ограничения области видимости переменных и функций, а также позволяют обращаться к блоку инструкций как к единой инструкции, могут быть пустыми или вложенными один в другой.

Процедура или функция представляет собой последовательность операторов, которая имеет имя, список параметров и может быть вызвана из различных частей программы. Функции, в отличие от процедур, в результате своего выполнения возвращают значение, которое может быть использовано в выражении.

Передача параметров в функцию происходит через стек. Код, вызывающий функцию, знает, сколько параметров ей передать и каковы значения этих параметров.

Передача параметров в функцию: по значению и по ссылке. При обращении к функции, можно опускать некоторые из ее аргументов. Можно задать параметры по умолчанию.

Лямбда-выражение в программировании — специальный синтаксис для определения функциональных объектов, заимствованный из λ-исчисления.

Анонимная функция – особый вид функций, которые объявляются в месте использования и не получают уникального идентификатора для доступа к ним.

Обычно при создании анонимные функции:

* вызываются напрямую;
* ссылка на функцию присваивается переменной, с помощью которой затем можно косвенно вызывать данную функцию

В последнем случае анонимная функция получает имя и становится именованной. Если анонимная функция ссылается на переменные, не содержащиеся в её теле (захват переменных), то такая функция называется замыканием. Лямбда-выражение — типичная для многих языков синтаксическая конструкция для определения анонимной функции.

Структура лямбда-выражения: **[ < маска\_переменных >] (< список\_параметров > ) mutable throw() → < возвращаемый\_тип > { /\* тело\_лямбда-выражения \*/ }**

Маска переменных (обязательно). Лямбда-выражение может получать доступ к переменным вне лямбда-выражения, определенным в той же области видимости, что и лямбда, и использовать их внутри тела лямбда. Маска определяет способ получения параметров (захват переменных) телом лямбда-выражения. Захват переменных означает, что лямбда может использовать не только переменные, которые передаются в качестве параметров, но и объекты, которые были объявлены вне лямбда-выражения.

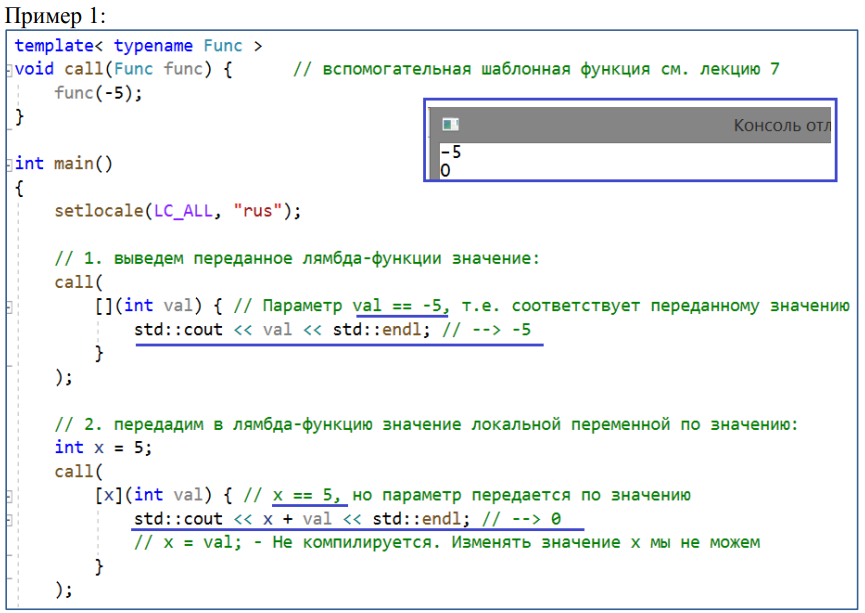
Список параметров (необязательно). Список параметров лямбда-выражения, аналогичен записи аргументов для обычных функций. Лямбда-выражение может принимать в качестве параметра другое лямбда выражение.

mutable (необязательно) — использование mutable позволяет модифицировать параметры, переданные по значению. throw() (необязательно) — спецификация исключений, то есть лямбда-выражения могут выбрасывать исключения.

Возвращаемый тип — определяет возвращаемый тип лямбда-выражения:

* если у лямбда-выражения нет возвращаемого значения, то по умолчанию устанавливается void;
* если в лямбда-выражении есть return, то компилятор вычисляет тип возвращаемого значения автоматически на основании return-выражения;
* если же в лямбда-выражении присутствует if или switch и, соответственно, несколько return, то требуется явное указание конечного возвращаемого типа в виде: → тип\_возвращаемого\_параметра, перед началом тела функции.

Тело лямбда-выражения (обязательно)



Раньше в лямбда-функциях было три вида скобок. Теперь их может быть четыре:

* квадратные скобки для переменных связывания
* угловые скобки для шаблонных параметров
* круглые скобки для списка аргументов
* фигурные скобки для тела функции.

Порядок важен: если его перепутать, будет ошибка. Если шаблонных параметров нет, то угловые скобки не пишутся, потому что пустыми они быть не могут.

### 28. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций в C++. Перегружаемые функции в C++. Inline-функции в C++. Шаблоны функций С++.

Блок — это последовательность операторов, заключенных в фигурные скобки и рассматриваются как одна программная единица. Операторы, составляющие блок, логически связаны друг с другом. Иногда блок называют составным оператором. Блок всегда начинается открывающейся фигурной скобкой { и заканчивается закрывающейся }. Блоки служат для ограничения области видимости переменных и функций, а также позволяют обращаться к блоку инструкций как к единой инструкции, могут быть пустыми или вложенными один в другой.

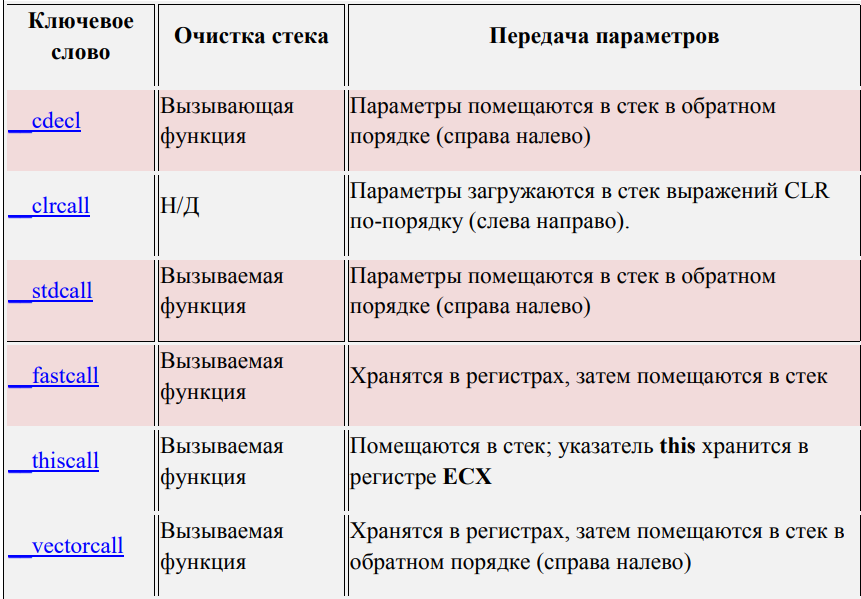
Процедура или функция представляет собой последовательность операторов, которая имеет имя, список параметров и может быть вызвана из различных частей программы. Функции, в отличие от процедур, в результате своего выполнения возвращают значение, которое может быть использовано в выражении.

### 29. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций в C++: соглашения о вызовах (\_cdecl).

Механизм вызова функций называется "Соглашение о вызовах (Calling convention)" – протокол, в соответствии с которым вызывающий код и вызываемая функция согласны общаться друг с другом.

Соглашение о вызовах – это договоренность между вызывающим и вызываемым кодом:

* о способе передачи параметров;
* о порядке их размещения в стеке;
* об очистке стека;
* о порядке возврата значения;
* о конкретной инструкции для вызова функции.

****

Соглашение вызовах \_\_cdecl (используется по умолчанию). Параметры передаются через стек, порядок следования параметров справа налево, стек освобождает вызывающий код, возврат значения через регистр EAX.

Определение. Соглашение вызовах \_\_cdecl:

* Параметры функций помещаются в стек, порядок передачи параметров «справа налево». Параметры, размер которых меньше 4-х байт, расширяются до 4-х байт. Адрес возврата кладется в стек поверх параметров.
* Стек освобождается вызывающим кодом: после инструкции CALL следует инструкция ADD, которая прибавляет к значению регистра ESP суммарный размер в байтах всех аргументов, т. о. целостность стека восстанавливается вызывающим кодом.
* Возвращаемый параметр передается через регистр EAX.

Передача аргументов функции по протоколу \_\_cdecl: параметры кладутся в стек «справа налево». Сразу за ними в стек кладется адрес возврата. Инструкция, восстанавливающая целостность стека, принадлежит вызывающему коду.

**30. Структура языка программирования: программные конструкции (блоки, функции,**

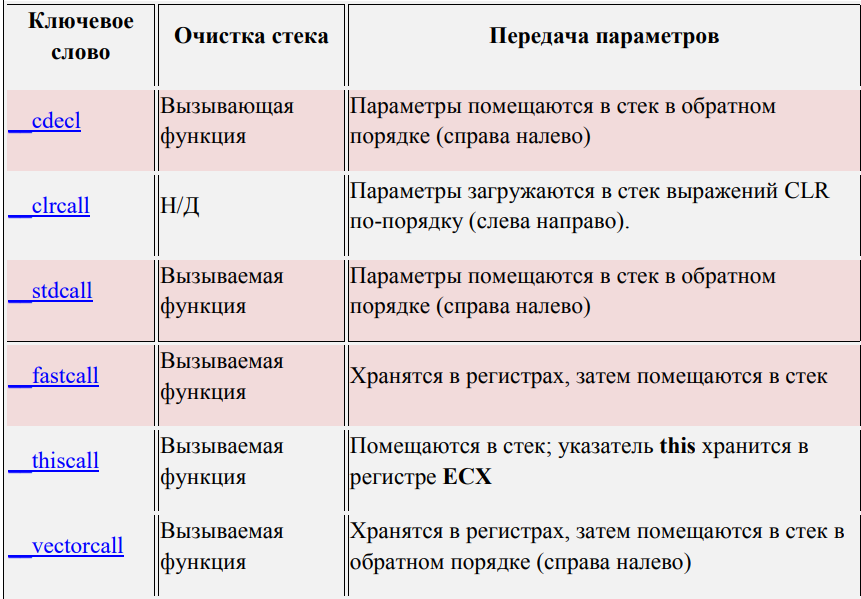
**процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций**

**в C++: соглашения о вызовах (\_stdcall).**

Механизм вызова функций называется "Соглашение о вызовах (Calling convention)" – протокол, в соответствии с которым вызывающий код и вызываемая функция согласны общаться друг с другом.

Соглашение о вызовах – это договоренность между вызывающим и вызываемым кодом:

* о способе передачи параметров;
* о порядке их размещения в стеке;
* об очистке стека;
* о порядке возврата значения;
* о конкретной инструкции для вызова функции.

****

Соглашение вызовах \_\_stdcall (Windows API): параметры передаются через стек, порядок параметров справа налево, стек освобождает вызываемый код, возврат через регистр EAX.

Определение. Соглашение вызовах \_\_stdcall:

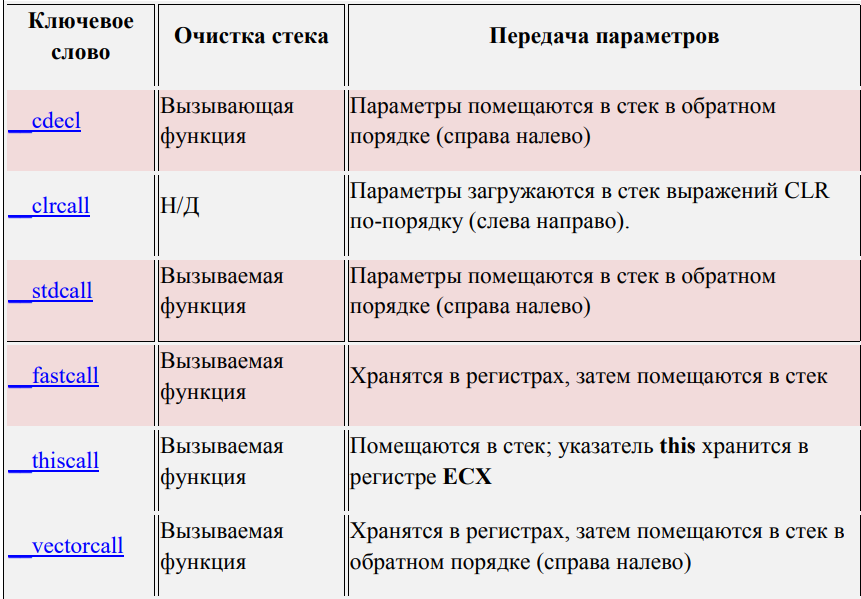
* Параметры помещаются в стек: порядок параметров «справа налево». Адрес возврата кладется в стек поверх параметров.
* вызываемый объект отвечает за очистку стека потоков, а параметры помещаются в стек справа налево. Однако к имени скомпилированной функции добавляется символ «@», за которым следует длина в байтах стекового пространства, занимаемого переданными параметрами функции. Регистры EAX, ECX и EDX указываются для использования в функции, а возвращаемое значение помещается в EAX.

### 31. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций в C++: соглашения о вызовах (\_fastcall).

Механизм вызова функций называется "Соглашение о вызовах (Calling convention)" – протокол, в соответствии с которым вызывающий код и вызываемая функция согласны общаться друг с другом.

Соглашение о вызовах – это договоренность между вызывающим и вызываемым кодом:

* о способе передачи параметров;
* о порядке их размещения в стеке;
* об очистке стека;
* о порядке возврата значения;
* о конкретной инструкции для вызова функции.

****

Определение. Соглашение вызовах \_fastcall:

* Соглашение \_\_fastcall передает первый (слева направо) параметр, который не превышает 32 бита, через регистр ECX / CX / CL, а второй параметр, который не превышает 32 бита, через регистр EDX. / DX / DL, остальные параметры помещаются в стек в порядке справа налево.
* Целостность стека восстанавливается вызываемым кодом. Возвращаемый параметр передается через регистр EAX.

### 32. Структура языка программирования: стандартная библиотека. Реализация стандартной библиотеки STL в C++. Понятие контейнера, итератора и алгоритма. Стандартные функции STL для работы со строками: копирование, сравнение, вычисление длины, поиск символа и подстроки.

Язык программирования C++ содержит два вида библиотек. В первой библиотеке хранятся стандартные универсальные функции, не принадлежащие ни одному классу, которая унаследована от языка С. Вторая библиотека содержит библиотеку классов и является объектно-ориентированной.

Стандартная библиотека C++: ANSI C89 (заголовки , пространство имен std). Стандартная Библиотека современного языка C++ включает в себя спецификации стандарта ISO C90 стандартной библиотеки языка Си и представляет собой набор файлов заголовков. В новых файлах заголовков отсутствует расширение .h. Каждый заголовочный файл из стандартной библиотеки языка Си включен в стандартную библиотеку языка C++ под именами, созданными путём отсечения расширения .h и добавлением 'c' в начале. Пример, 'time.h' стал 'ctime'. Единственное отличие между этими файлами заголовков и традиционными заголовочными файлами стандартной библиотеки языка Си заключается в том, что функции должны быть помещены в пространство имен std::

Стандартная библиотека C++ STL: библиотека стандартных шаблонов (STL) – набор согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++. STL (Standard Template Library) – стандартная библиотека шаблонов. Библиотека содержит универсальные шаблонные классы и функции, реализующие большое количество распространенных универсальных алгоритмов и структур данных. Т.к. библиотека STL состоит из шаблонных классов, ее алгоритмы и структуры можно применять практически к любым типам данных. Частью стандартной библиотеки С++ является библиотека STL. Библиотека STL содержит пять основных видов компонентов:

− контейнер (container): управляет набором объектов в памяти.

− итератор (iterator): обеспечивает для алгоритма средство доступа к содержимому контейнера.

− алгоритм (algorithm): определяет вычислительную процедуру.

− функциональный объект (function object): инкапсулирует функцию в объекте для использования другими компонентами.

− адаптер (adaptor): адаптирует компонент для обеспечения различного интерфейса.

Контейнер – объект, содержащий другие объекты (стек, список, очередь, вектор и пр.), предназначенный для хранения однотипных объектов и обеспечения доступа к ним. Контейнеры: последовательные (последовательный доступ к элементам); ассоциативные (доступ по ключу).

В библиотеке STL реализованы следующие контейнеры:

* bitset – набор битов;
* deque – двухсторонняя очередь;
* list – линейный список;
* map – ассоциативный контейнер, построенный по принципу key:value (ключ:значение), в котором каждому ключу key соответствует значение value (пары хранятся в отсортированном виде, что позволяет осуществлять быстрый поиск по ключу);
* multimap – ассоциативный контейнер, в котором одному значению (key) соответствует несколько значений (value1, value2, …, valueN);
* multiset – множество, в котором один и тот же элемент может встречаться несколько раз; − priority\_queue – очередь с приоритетами;
* queue – очередь;
* set – множество, в котором каждый элемент встречается только один раз;
* stack – стек;
* vector – динамический массив (коллекция элементов, сохраненных в массиве, размер может изменяться по мере необходимости).

Алгоритм определяет вычислительную процедуру (обобщённые алгоритмы) для работы с контейнерами. Алгоритмы позволяют манипулировать содержимым контейнера: инициализировать, сортировать, искать, изменять содержимое контейнера. Содержимое контейнеров обрабатывается с помощью алгоритмов. Алгоритмы позволяют обрабатывать контейнеры на любой вкус: инициализировать, сортировать содержимое контейнеров, преобразовывать, реализовывать различные виды поиска и тому подобное. Для доступа к алгоритмам, нужно подключить соответствующую библиотеку #include

Все реализованные функции можно поделить на три группы:

• Методы перебора всех элементов коллекции и их обработки

• Методы сортировки коллекции

• Методы выполнения определенных арифметических операций над членами коллекций

Все алгоритмы являются шаблонными функциями. Они могут быть применены к любому типу контейнера.

Функции для работы со строками:

strlen() (от слова length – длина)

strcpy() – копирование

strcmp() - Функция strcmp() сравнивает две строки: s1 и s2. Она возвращает целое число, которое меньше, больше нуля или равно ему, если s1 соответственно меньше, больше или равно s2. Функция strncmp() работает аналогичным образом, но сравнивает только первые n символов строки s1.

.find() - находит первое вхождение элемента. Он использует оператор = для сравнения.

### 33. Классы памяти: код, стек, статические данные, динамическая область памяти. Структура динамической памяти (Heap) C++. Фрагментация динамической памяти C++.

Инструкция (или оператор (англ. statement) – это одна команда языка программирования; наименьшая законченная часть.

Блок (или составной оператор) – это группа логически связанных между собой инструкций языка программирования, рассматриваемых как единое целое.

Область кода – память, в которой размещается код программы.

Статическая память Static или статическая память выделяется до начала работы программы, на стадии компиляции и служит для хранения статических переменных. Типы статических переменных: глобальные переменные и статические переменные. Глобальные переменные – это переменные, определенные вне функций. Память для глобальных переменных выделяется на этапе компиляции. Глобальные переменные доступны в любой точке программы во всех ее файлах. Статические переменные – это переменные, в описании которых присутствует ключевое слово static. Компилятор выделяет для таких переменных постоянное место хранения в статической области памяти.

Стековая память Stack или стековая (автоматическая) память предназначена для хранения локальных переменных. Локальные переменные хранятся в стеке. Стек – это непрерывная область оперативной памяти, организованная по принципу LIFO (последний вошел, первый вышел).

Динамическая память Heap или динамическая память, или куча – это область памяти, выделение которой в языке программирования С++ производится с помощью оператора new, освобождение — оператором delete. Выделение памяти в C++ производится с помощью оператора new, освобождение — оператором delete. Память, выделяемая функциями динамического распределения памяти, находится в куче (heap).

### 34. Механизм обработки исключений: определение, назначение, применение. Реализация обработки исключений в C++. Пример.

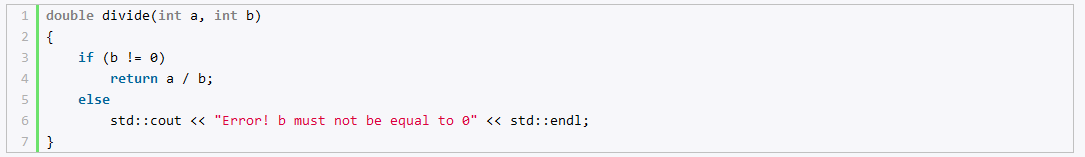
В процессе работы программы могут возникать различные ошибки. Например, при передаче файла по сети оборвется сетевое подключение или будут введены некорректные и недопустимые данные, которые вызовут падение программы. Такие ошибки еще называются исключениями. Если исключение не обработано, то при его возникновении программа прекращает свою работу.

Например, в следующей программе происходит деление чисел:



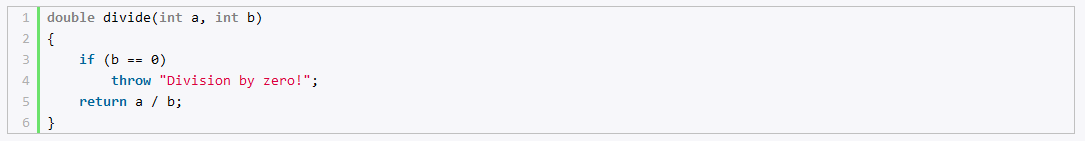
Эта программа успешно скомпилируется, но при ее выполнении возникнет ошибка, поскольку в коде производится деление на ноль, после чего программа аварийно завершится.

С одной стороны, мы можем в функции divide определить проверку и выполнять деление, если параметр b не равен 0. Однако нам в любом случае надо возвращать из функции divide некоторый результат - некоторое число. То есть мы не можем просто написать:



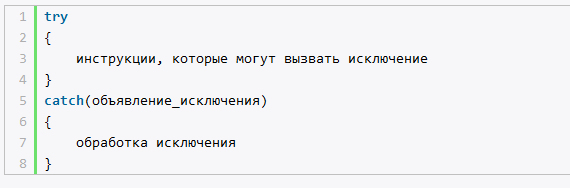
И в этом случае нам надо известить систему о возникшей ошибке. Для этого используется оператор throw.

Оператор throw генерирует исключение. Через оператор throw можно передать информацию об ошибке. Например, функция divide могла бы выглядеть следующим образом:

****

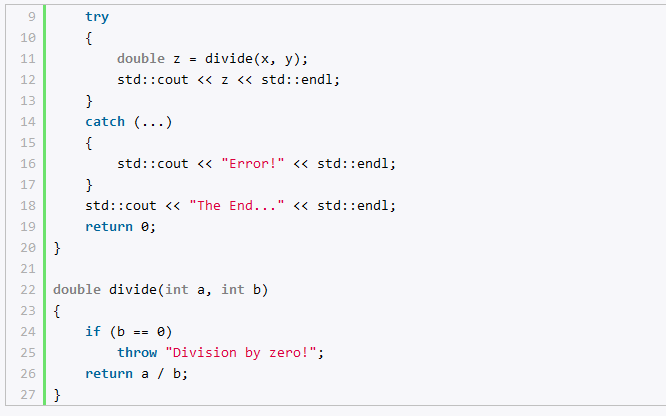
То есть если параметр b равен 0, то генерируем исключение.

Но это исключение еще надо обработать в коде, где будет вызываться функция divide. Для обработки исключений применяется конструкция try...catch. Она имеет следующую форму:

**** В блок после ключевого слова try помещается код, который потенциально может сгенерировать исключение.

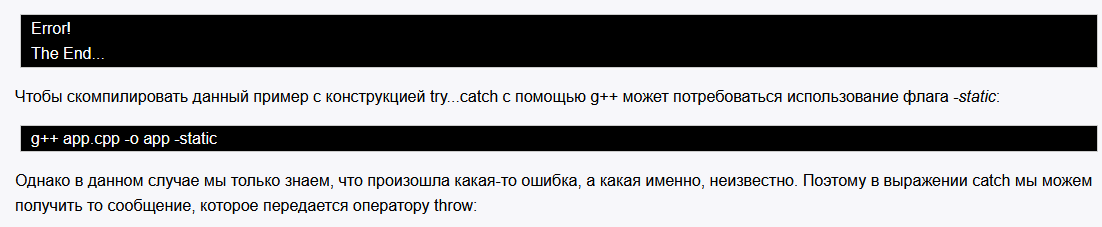
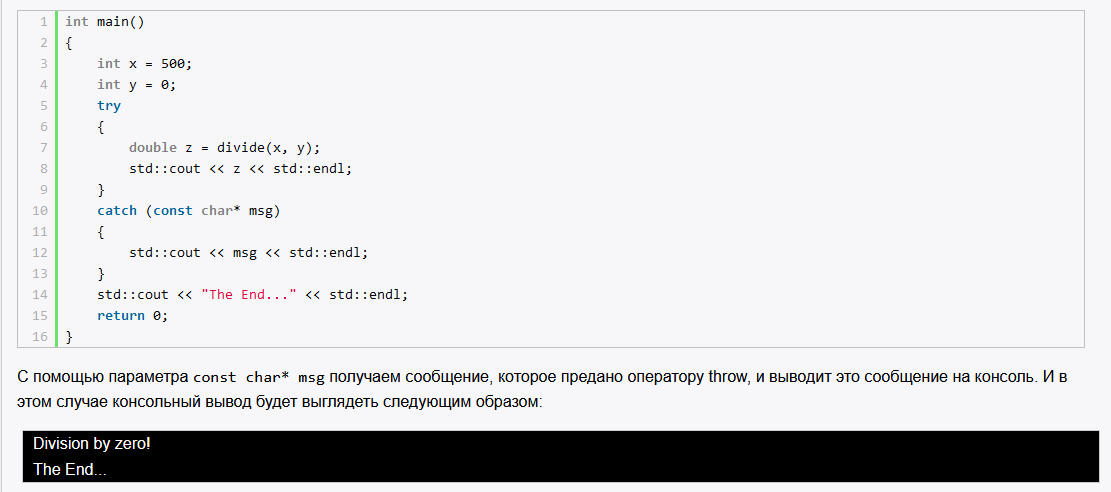
После ключевого слова catch в скобках идет параметр, который передает информацию об исключении. Затем в блоке производится собственно обработка исключения.

Так изменим весь код следующим образом:

**** од, который потенциально может сгенерировать исключение - вызов функции divide помещается в блок try.

В блоке catch идет обработка исключения. Причем многоточие в скобках после оператора catch (catch(...)) позволяет обработать любое исключение.

В итоге когда выполнение программы дойдет до строки double z = divide(x, y);, будет сгенерировано исключение, поэтому последующие инструкции из блока try выполняться не будут, а управление перейдет в блок catch, в котором на консоль просто выводится сообщение об ошибке. После выполнения блока catch программа аварийно не завершится, а продолжит свою работу, выполняя операторы после бллока catch:

**** ****

### 35. Препроцессор: определение, назначение, применение, директивы, выражения, макросы, директивы условной компиляции. Примеры на C++.

Препроцессор – программа для обработки текста. Препроцессор может быть отдельной программой, или же интегрирован в компилятор. Входные и выходные данные для препроцессора имеют текстовый формат. Препроцессор преобразует текст в соответствии с директивами препроцессора. В случае если текст не содержит директив препроцессора, то он остаётся без изменений.

Назначение. Директивы препроцессора могут:

* заменить какие-то лексемы в исходном тексте;
* вставить содержимое других файлов в указанном месте;
* подавить компиляцию части файла.

Директивa #include (включение файла). Директива #include вставляет содержимое заданного файла в место расположения этой директивы в исходном тексте программы.

Синтаксис: #include "спецификация\_пути" #include < спецификация\_пути>

Кавычки и угловые скобки определяют способ поиска включаемых файлов:

* если спецификация файла заключена в угловые скобки, то он должен находиться в каталоге, указанном компилятором (обычно это каталог INCLUDE, в котором хранятся заголовочные файлы).
* если имя файла заключено в кавычки, то поиск выполняется в рабочем каталоге. – если указан полный путь, то предпроцессор использует его для поиска файла.

«Прагма» – это инструкция компилятору С/С++, используется для указания опций компилятору для управления его работой.

Синтаксис: #pragma параметры

Так #pragma once контролирует, чтобы конкретный включаемый файл при компиляции подключался строго один раз.

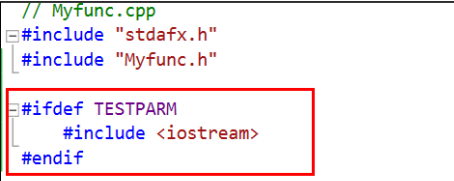
Директива препроцессора #define определяет идентификатор и последовательность символов, которая заменит этот идентификатор в тексте программы. Идентификатор — это имя макроса. Процесс замены называется макроподстановкой. Директива #define заменяет все вхождения идентификатора макроса в исходном файле на последовательность символов. Идентификатор не будет заменен, если он является частью более длинного идентификатора (подстрокой в имени более длинного идентификатора).

Пример. #define ICY 1582 . После определения директивы идентификатор ICY считается определенным. Препроцессор заменит каждое вхождение идентификатора ICY в тексте программы на целочисленный литерал 1582.

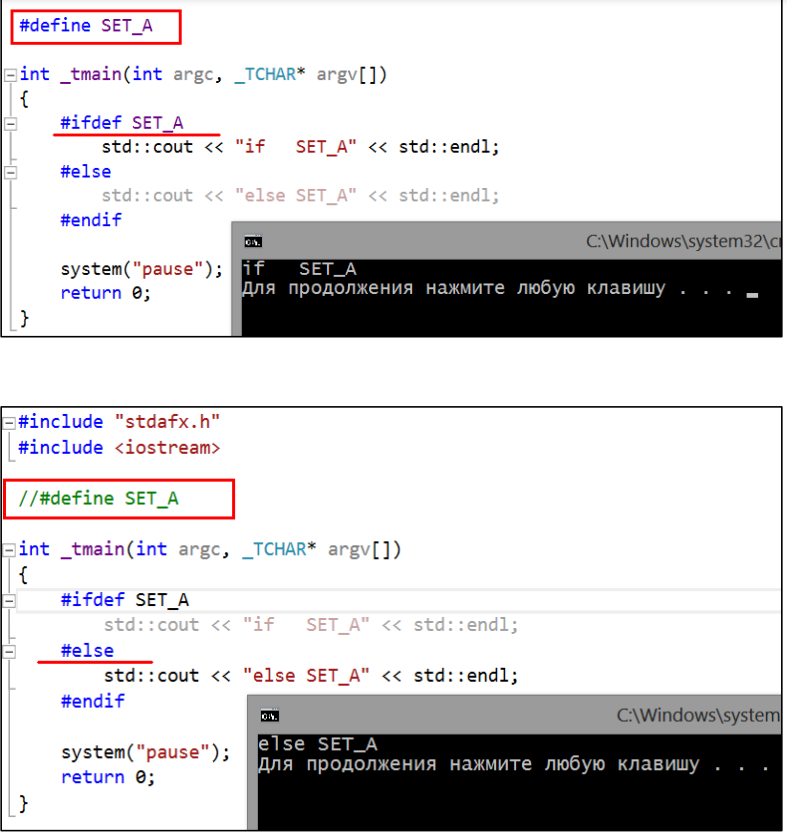
Макроопределения могут иметь параметры. Этот вид макроса называется функциональным. Формальный аргумент x в определении макроса ERRMM(x) будет заменен значением фактического параметра mm1. Директива препроцессора #undef аннулирует ранее определенный макрос. В нашем примере это: #undef ERRMM После этой директивы макрос становится неопределенным, и последующие ссылки на него будут приводить к ошибке компиляции. Использование функциональных макроопределений увеличивает скорость выполнения программы, т.к. в ней отсутствуют вызовы функций. Но при этом происходит дублирование фрагментов программы, и размер ее может значительно увеличиться, если размер функциональных макросов достаточно велик.

Директивы условной компиляции #ifdef и #ifndef управляют компиляцией части исходного файла. Каждая директива #ifdef в исходном коде должна иметь соответствующую закрывающую директиву #endif. Директивы условной компиляции позволяют определять истинность некоторых условий и в зависимости от результата проверки определяют, какие блоки исходного кода будут переданы на обработку компилятору, и какие блоки будут удалены из компиляции.

Пример применения: такая возможность позволяет выполнить автоматическую настройку компиляции программы на разных платформах (операционная система + компилятор + процессор); получить из одного исходного файла нескольких вариантов программы (отладочные версии с промежуточным выводом результатов и рабочая версия). Директива #ifdef проверяет определено ли в данный момент имя макроса TESTPARM. Результат проверки будет иметь значение "истина", если заданный макрос определен, в противном случае – "ложь":

Директива условной компиляции #if позволяет подавить компиляцию части исходного файла. Каждая директива #if в исходном файле должна иметь соответствующую закрывающую директиву #endif.

Пример с подключенным и неподключенным макросом: 

Директива #error Синтаксис: #error текст

С помощью директивы #error можно определить текстовую строку (обратите внимание, что строка записывается без кавычек), которая будет выведена как сообщение об ошибке при компиляции.

Компилятор C++ автоматически определяет некоторые макросы, например: \_\_LINE\_\_ Этот макрос заменяется номером текущей строки в форме десятичной целой константы. Этот макрос в сочетании с макросом \_\_FILE\_\_ можно использовать при генерации сообщения об ошибке для вывода несоответствия, обнаруженного программой. В этом случае сообщение будет содержать номер строки с именем исходного файла, в котором была обнаружена ошибка.

Опрераторы препроцессора # и ##. Стрингификация (#) – это преобразование фрагмента кода в строковую константу, т.е. преобразование аргумента в строку. Например, в результате стрингификации STR(hello) аргумент преобразуется в символьную строку "hello".

Канкатенация (##) (оператор конкатенации) – это конкатенация двух строковых констант. При работе с макросами, это означает объединение двух лексических единиц в одну более длинную. Например, один аргумент макроса может быть объединен с другим аргументом или с каким-либо текстом.

### 36. Статическая библиотека: определение, назначение, применение. Реализация статической библиотеки в Visual C++. Утилита LIB. Создание статической библиотеки. Применение директивы препроцессора #pragma comment для компоновки со статической библиотеки.

Статическая библиотека – файл с объектными модулями, предназначенный для подключения к программе на этапе компоновки. Библиотеки, распространяемые в виде исходного кода, преобразуются компилятором в объектные файлы. Затем компоновщик соединяет объектные файлы библиотек и объектные файлы программы в один исполняемый файл.

Статическая библиотека: файл (обычно с расширением lib), содержащий объектные модули; входной файл для компоновщика (linker).

Достоинства:

* + просто использовать;
  + не требуется наличие самой библиотеки;
  + исполняемый файл один (расширение .exe).

Недостатки:

* + платформенно зависима;
  + загружается в память с каждым экземпляром запущенного приложения;
  + при изменении кода библиотеки необходима компоновка всех приложений, которые используют библиотеку.

Создание статической библиотеки с помощью «Мастера классических приложений Windows».

* Определяем имя решения, проекта и выбираем место размещения на диске.
* Выбираем тип приложения «Статическая библиотека» (снимаем флажок «Предварительно скомпилированные заголовки» при необходимости). В проект добавляем один или несколько файлов, содержащих реализации функций библиотеки.
* В пункте раздела «Библиотекарь» → «Командная строка» отображается текущее значение параметра /OUT. Расширение выходных файлов определено как .lib. Директорий – папка проекта Debug.
* После построения проекта в папке решения Debug размещен файл статической библиотеки (.lib). В папке проекта Debug размещен файл журнала построения: В журнале проекта зафиксировано выполнение сборки проекта. Видим, что файл статической библиотеки создан утилитой LIB.

Режимы использования утилиты LIB:

* + построение или изменение библиотеки;
  + извлечение элемента-объекта библиотеки в файл;
  + создание файла экспорта и библиотеки импорта.

Эти режимы взаимоисключающие, LIB можно использовать только в одном режиме. LIB принимает те или иные входные файлы в зависимости от режима использования.

Автоматическое связывание (от англ. Auto-linking) – механизм автоматического определения необходимых библиотек при сборке программ на языках С/C++. Активируется при помощи директивы препроцессора: #pragma comment(lib, ) В этом случае для подключения статической библиотеки дополнительные параметры командной строки компоновщика не используются. Статическая библиотека подключается при помощи строки кода #pragma comment(lib, ) Директива размещается в главной функции

### 37. Теория формальных языков. Определения: алфавит языка, цепочка, пустая цепочка, длина цепочки, равенство (эквивалентность) цепочек, конкатенация цепочек, итерация цепочки, операторы + и \*. Примеры.

Формальный язык – это множество конечных слов над конечным алфавитом, например, язык программирования.

Лексика языка программирования – правила составления слов программы из символов языка (идентификаторы, константы, служебные слова, комментарии).

Синтаксис языка – система правил, определяющих допустимые конструкции языка программирования из слов языка (построение, порядок, составление).

Семантика (смысловое значение) – смысл, который закладывается в каждую конструкцию языка. Алфавит – конечное непустое множество допустимых символов языка (букв языка). Пример: V = {a,b} – алфавит V, состоящий из двух символов a и b.

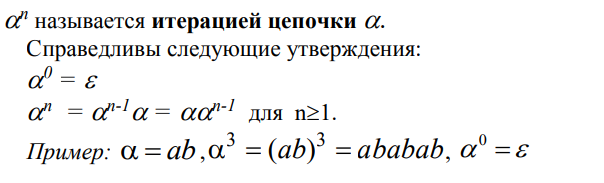
Цепочка – конечная последовательность символов языка (слово в языке). Пример: abc – цепочка из трех символов.

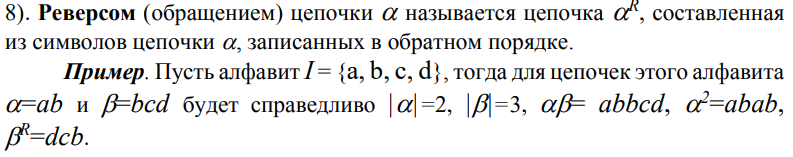
Цепочка, которая не содержит ни одного символа, называется пустой цепочкой и обычно обозначается ε или λ.

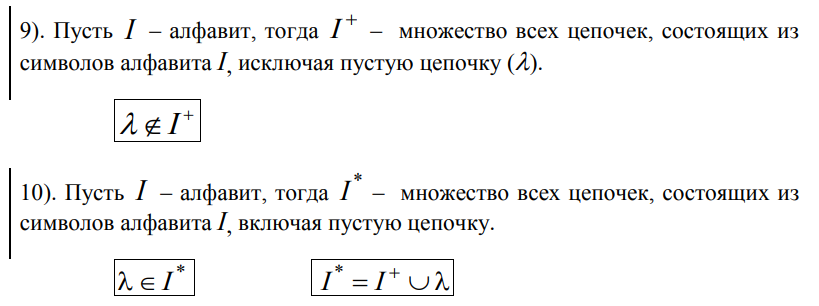
Длиной цепочки α (обозначается |α|) называется число составляющих ее символов. Пример: |α| = 3, |β| = 4, |γ| = 7. Длина пустой цепочки |ε|=0.

Цепочки α и β равны (α = β), если они имеют один и тот же состав символов, одно и тоже количество символов α = β и тот же порядок следования символов.

Конкатенацией (сцеплением) цепочек α и β называется цепочка γ=αβ, в которой символы данных цепочек записаны друг за другом. Пример: α=abc, β=aaaa, αβ= abcaaaa.







### 38. Теория формальных языков. Определение формального языка, эквивалентность двух языков, способы задания формального языка. Лексика, синтаксис и семантика языка. Примеры.

Формальный язык – это множество конечных слов над конечным алфавитом, например, язык программирования.

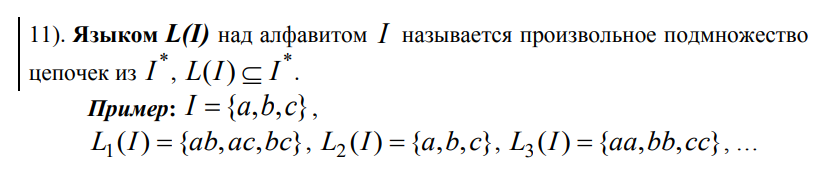
Лексика языка программирования – правила составления слов программы из символов языка

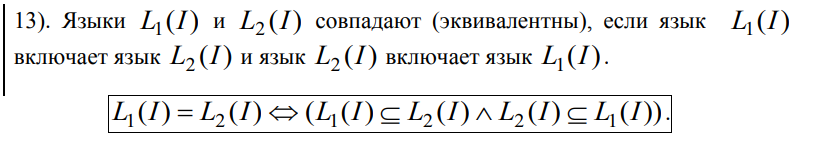
Синтаксис языка – набор формальных правил, определяющий конструкции (последовательности цепочек) языка.

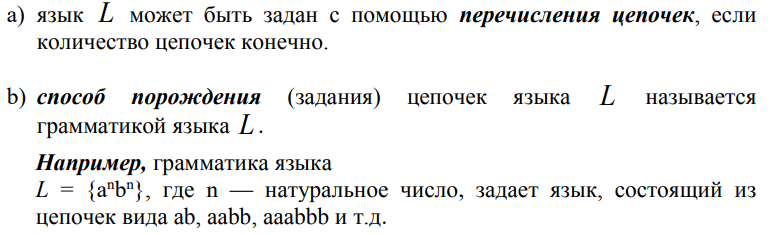
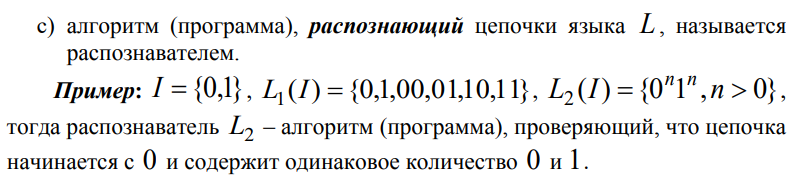
Семантика (смысловое значение) – смысл, который закладывается в каждую конструкцию языка. Пример: переменную надо объявить до ее применения.

Чтобы создать язык программирования, следует определить:

* + множество допустимых символов языка (алфавит);
  + формально описать множество правильных программ;
  + задать семантические правила языка.



****

 ****

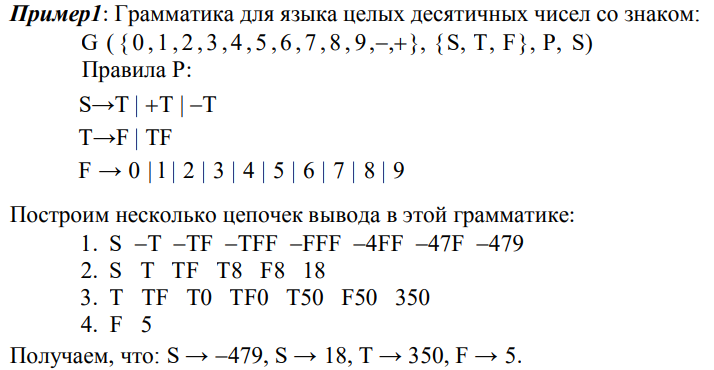
### 39. Теория формальных языков. Формальная грамматика (определение, назначение), выводимость цепочки символов в грамматике, сентенциальная форма грамматики, язык, порождаемый грамматикой, способы задания грамматик. Примеры.

Формальный язык – это множество конечных слов над конечным алфавитом, например, язык программирования.

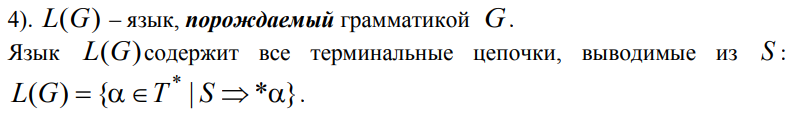
Выводом называется процесс порождения предложения языка на основе правил грамматики языка. Записывается α ⇒ β

Цепочка β = δ1γδ2 называется непосредственно выводимой из цепочки α = δ2ωδ2 в грамматике G = < T,N,P, S> , где V = N∪T, δ1 , γ, δ2 ∈ V\* , ω ∈ V+ , если в грамматике существует правило: ω → γ .

Цепочка β называется выводимой из цепочки α, если выполняется одно из двух условий: - β непосредственно выводима из α (α→β); - существует γ такая, что γ выводима из α, и β непосредственно выводима из γ (α→γ, γ→β).

****

Вывод называется законченным (или конечным), если на основе цепочки β, полученной в результате этого вывода (нельзя больше сделать ни одного шага вывода). β называется сентенциальной формой грамматики G = <T,N,P,S >, если S ⇒ β\* и β∈(T ∪ N)\* . Если S ⇒ β\* и β∈T\* , то β называется терминальной сентенциальной формой грамматики G = <T,N,P, S >. Тогда цепочки −479 и 18 являются конечными сентенциальными формами грамматики целых десятичных чисел со знаком, так как существуют выводы: S ⇒\*−479 , S ⇒\*18

Язык, порождаемый грамматикой – это множество всех выводимых из аксиомы грамматики терминальных цепочек****

Для многих языков (и для синтаксической части языков программирования в том числе) допустимо использовать формальное описание грамматики, построенное на основе системы правил (или продукций).Правило (или продукция) — это упорядоченная пара цепочек символов (a, b). В правилах очень важен порядок цепочек, поэтому их чаще записывают в виде a::=b. Такая запись читается как «a порождает b» или «b по опреде­лению есть a».

### 40. Теория формальных языков. Определение порождающей грамматики. Определение вывода, левосторонний и правосторонний выводы, дерево вывода. Примеры.

G = < T,N,P, S > – грамматика языка (порождающая грамматика) – это четверка, где:

* T – множество терминальных символов
* N – множество нетерминальных символов
* P – множество правил (или продукций) грамматики
* S – начальный символ грамматики.

T – множество терминальных символов (терминалы, алфавит языка) – это символы языка, определяемые грамматикой. Терминалы будем обозначать строчными символами.

N – множество нетерминальных символов (нетерминалы) – символы, применяемые в продукциях P (символы, определяющие слова, понятия, конструкции языка). N ∩T = ∅ Нетерминалы будем обозначать прописными буквами.

P – множество правил

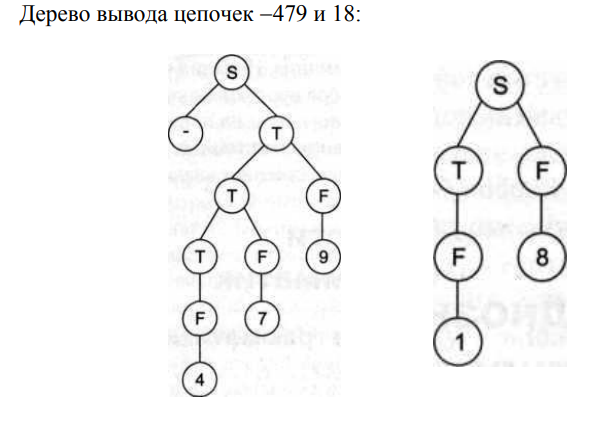
Выводом называется процесс порождения предложения языка на основе правил грамматики языка. Записывается α ⇒ β

Вывод называется левосторонним, если в нем на каждом шаге вывода правило грамматики применяется к крайнему левому нетерминальному символу в цепочке.

Вывод называется правосторонним, если в нем на каждом шаге вывода правило грамматики применяется всегда к крайнему правому нетерминальному символу в цепочке.

Деревом вывода грамматики G(T,N,P,S) называется дерево (граф), которое соответствует некоторой цепочке вывода и удовлетворяет следующим условиям:

* - каждая вершина дерева обозначается символом грамматики А ∈ (T∪N∪{ε});
* - корнем дерева является вершина, обозначенная целевым символом грамматики (или аксиома грамматики) – S;
* - листьями дерева (концевыми вершинами) являются вершины, обозначенные терминальными символами грамматики или символом пустой цепочки;
* - если некоторый узел дерева обозначен нетерминальным символом А∈N, а связанные с ним узлы — символами b1, b2 … bn; n > 0, n ≥ i > 0: bi ∈(T∪N∪{ε}), то в грамматике G(T,N,P,S) существует правило A → b1,b2 … bn.

****

### 41. Теория формальных языков: способы задания грамматик. Общая характеристика формы Бэкуса-Наура. Расширенная БНФ. Примеры.

Для задания схем грамматик используются различные формы описания:

− символическая;

− форма Бэкуса-Наура;

− итерационная форма;

− синтаксические диаграммы.

Цепочки языка могут содержать метасимволы, имеющие особое назначение. Метаязык, предложенный Бэкусом и Науром (БНФ) использует следующие обозначения:

* символ «::=» отделяет левую часть правила от правой (читается: «определяется как»);
* нетерминалы обозначаются произвольной символьной строкой, заключенной в угловые скобки «»;
* терминалы – это символы, используемые в описываемом языке;
* правило может определять порождение нескольких альтернативных цепочек, которые отделяются друг от друга символом вертикальной черты «|» (читается: «или»).

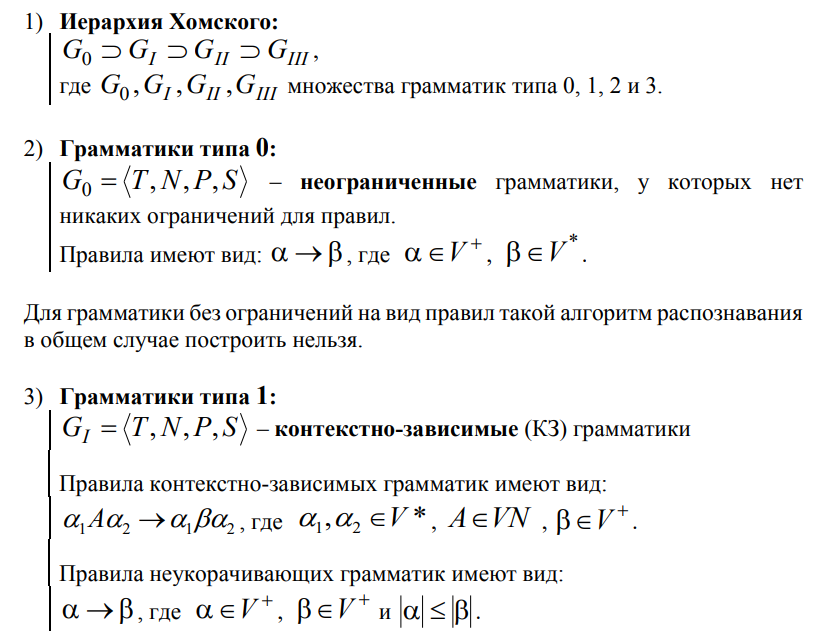
Форма Бэкуса — Наура (сокр. БНФ, Бэкуса — Наура форма) – формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории. Расширенная БНФ-нотация включает две конструкции, полезные при спецификации практических языков программирования.

Для удобства и компактности описаний, в расширенных БНФ (РБНФ) вводятся следующие дополнительные конструкции (метасимволы):

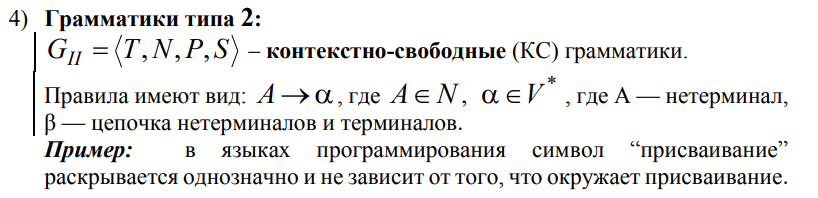
* + квадратные скобки «[» и «]» означают, что заключенная в них синтаксическая конструкция может отсутствовать;
  + фигурные скобки «{» и «}» означают повторение заключенной в них синтаксической конструкции ноль или более раз;
  + сочетание фигурных скобок и косой черты «{/» и «/}» используется для обозначения повторения один и более раз;
  + круглые скобки «(» и «)» используются для ограничения альтернативных конструкций;
  + кавычки используются в тех случаях, когда один из метасимволов нужно включить в цепочку обычным образом.

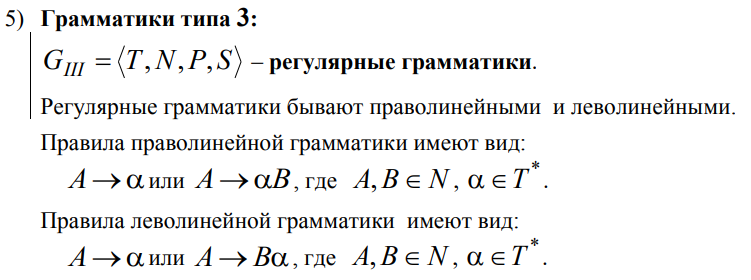
### 42. Классификация языков и грамматик по Хомскому: иерархия Хомского (неограниченные, контекстно-зависимые, контекстно-независимые, регулярные грамматики и вид правил этих грамматик), соотношения грамматик и порождаемых ими языков.

Хомский Ноам: 1928, США, лингвист, профессор Массачусетского технологического института, автор классификации формальных языков (иерархия Хомского), ввел понятие порождающей грамматики (1950).

****

Контекстно-зависимая грамматика: один и тот же нетерминальный символ может быть заменен на ту или иную цепочку символов в зависимости от контекста (цепочки) в которой они встречаются. В неукорачивающих грамматиках любая цепочка символов может быть заменена на цепочку символов не меньшей длины.

****

****

Соотношения грамматик в иерархии Хомского:

* + - любая регулярная грамматика является контекстно-свободной грамматикой;
    - любая контекстно-свободная грамматика является контекстнозависимой грамматикой;
    - любая контекстно-зависимая грамматика является грамматикой типа 0.

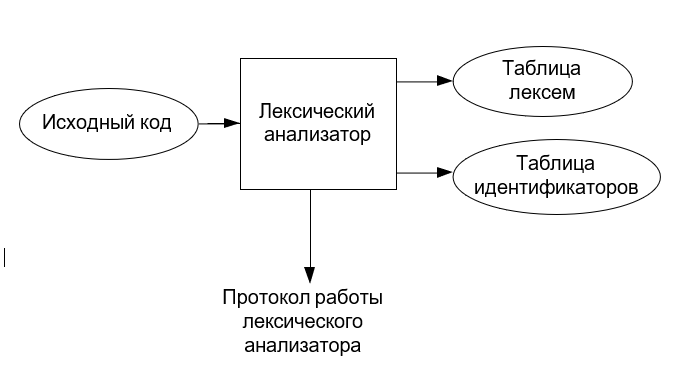
Формальные языки классифицируются по типу порождающих их грамматик.

Соотношения между типами формальных языков:

* + - каждый регулярный язык является контекстно-свободным языком, но существуют контекстно-свободные языки, которые не являются регулярными;
    - каждый контекстно-свободный язык является контекстно-зависимым, но существуют контекстно-зависимые, которые не являются контекстносвободными.
    - каждый контекстно-зависимый язык является языком типа 0.

### 43. Лексический анализ: определение, назначение, применение. Лексический анализатор: место в структуре транслятора, входная и выходная информация, способы взаимодействия с синтаксическим анализатором (последовательное и параллельное).

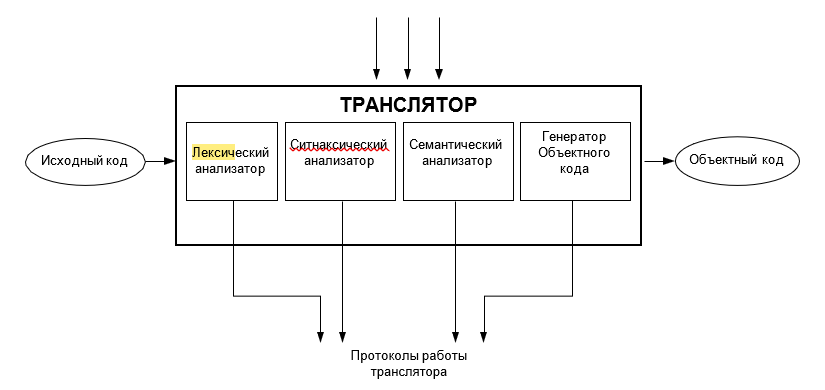
Лексический анализ – первая (наиболее простая) фаза трансляции. Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором (сканером).

****

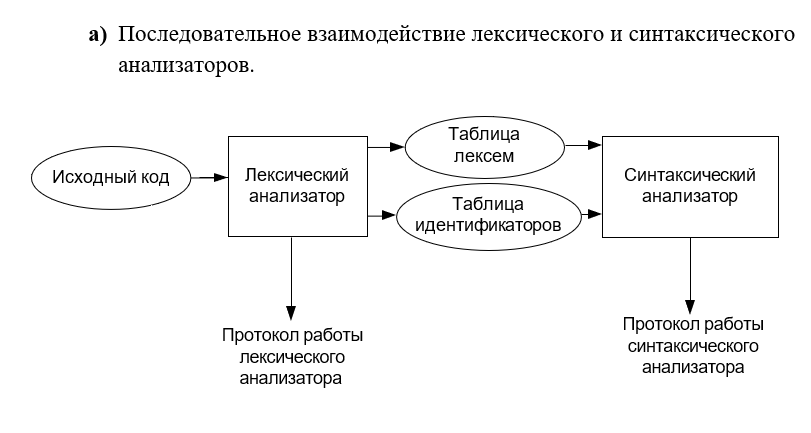
С точки зрения лексического анализатора – язык программирования набор лексем (токенов), которые распознаются (классифицируются) лексическим анализатором.

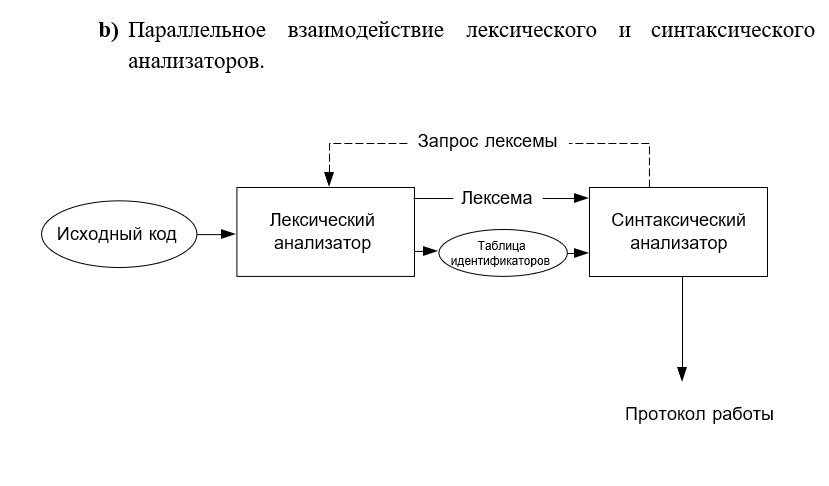
Язык программирования на уровне лексического анализа представляет собой

***регулярный язык*** (язык типа 3 иерархии Хомского).

****

На вход лексического анализатора поступает текст исходной программы, а выходная информация передается для дальнейшей обработки компилятором на этапе синтаксического анализа и разбора.

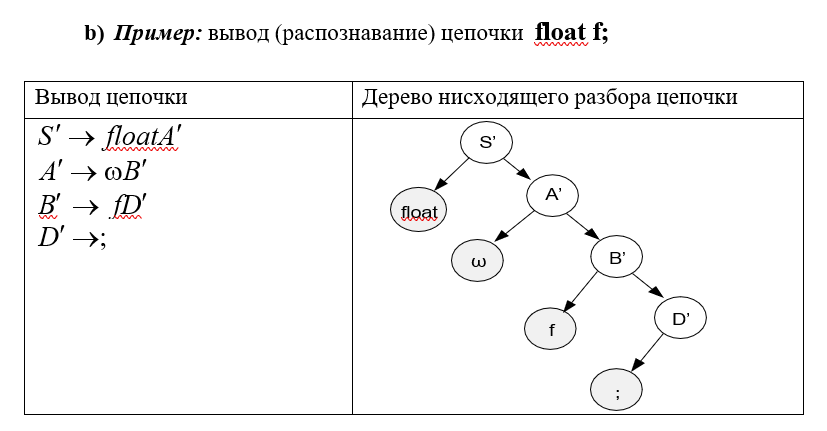
****

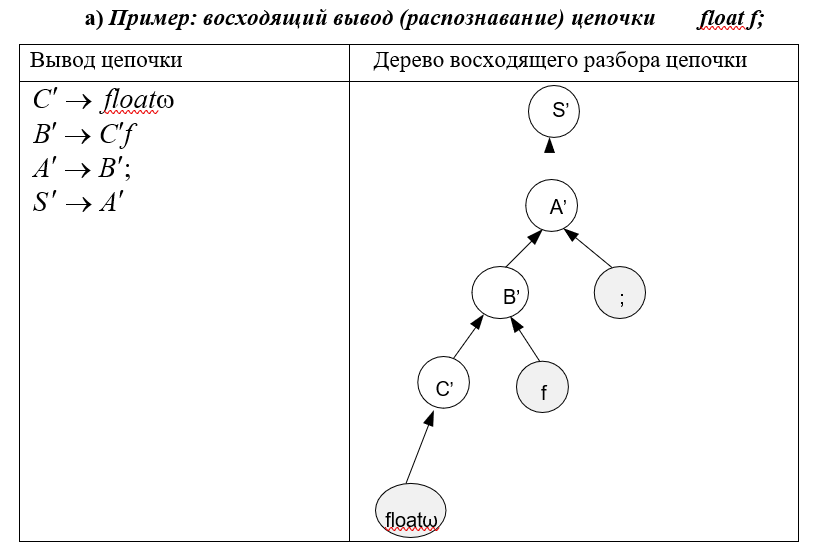
****

### 44. Лексический анализ: определение, назначение, применение. Лексический анализатор: вид правил регулярной грамматики, нисходящий и восходящий разбор цепочек, дерево разбора.

Грамматика языка описывает множество правильных цепочек символов над заданным алфавитом.

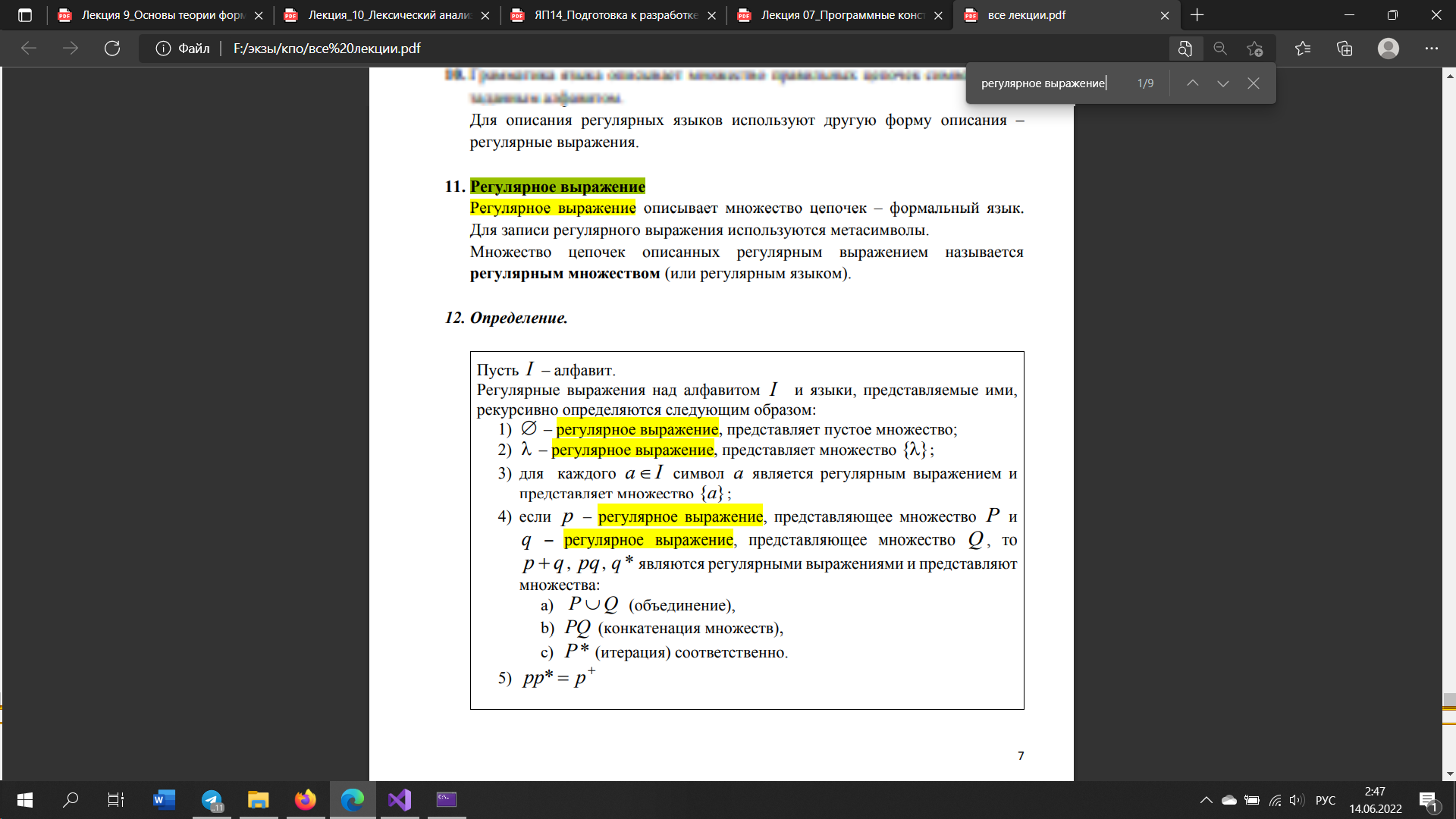
Для описания регулярных языков используют другую форму описания – регулярные выражения.

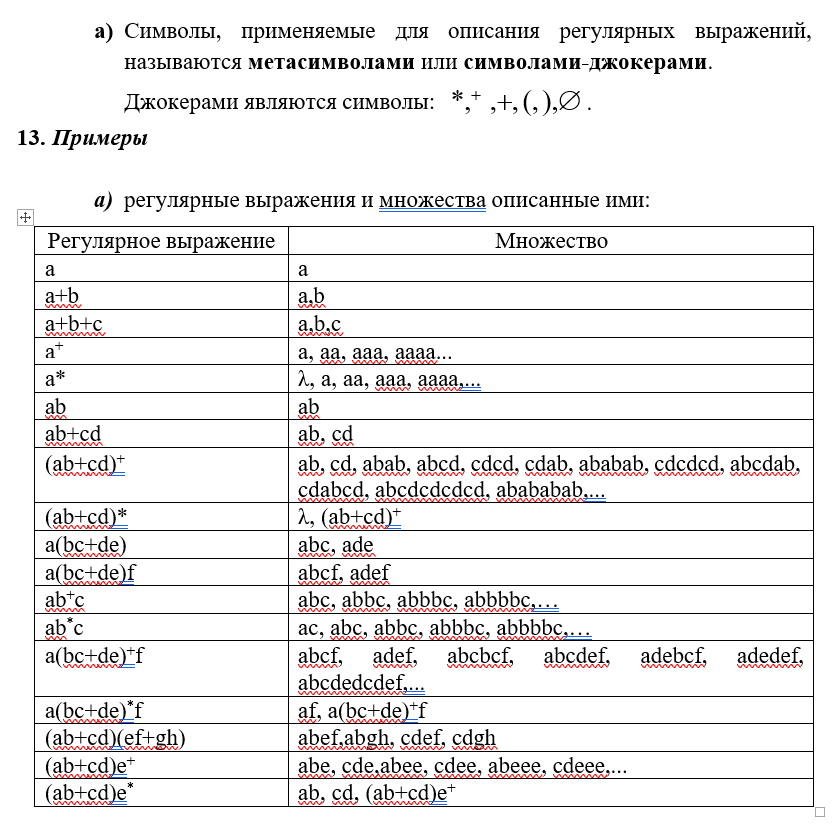
****

****

**Деревом разбора грамматики** (англ. parse tree) называется дерево, в вершинах которого записаны терминалы или нетерминалы. Все вершины, помеченные терминалами, являются листьями

### 45. Регулярные выражения: рекурсивное определение, назначение, применение, метасимволы, примеры. Пример применения в функциях стандартной библиотеке STL C++.



****

### 46. Конечный автомат (КА): определение, назначение, схема работы КА, примеры.

Определение КА: КА это пятерка M = (S, I, δ, s0 , F ), где:

* S – конечное множество состояний устройства управления;
* I – алфавит входных символов;
* δ – функция переходов, отображающая S × (I ∪{λ}) во множество подмножеств S : δ(s,i) ⊂ S,s∈S,i∈ I;
* s0 ∈ S - начальное состояние устройства управления;
* F ⊆ S - множество заключительных (допускающих) состояний устройства управления.

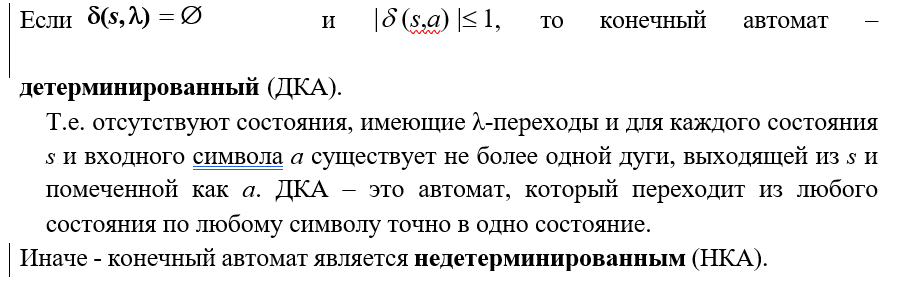
Назначение: Конечные автоматы широко используются на практике, например, в синтаксических и лексических анализаторах. Конечный автомат довольно хорошо применим в много шаговых интерфейсах, где следующий шаг зависит от того, какие параметры были выбраны на текущем (предыдущем) шаге.

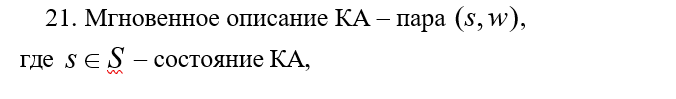
Схема работы: Принято полагать, что конечный автомат начинает работу в начальном состоянии, последовательно считывая по одному символу входного слова (цепочки входных символов). Считанный символ переводит автомат в новое состояние в соответствии с функцией переходов.

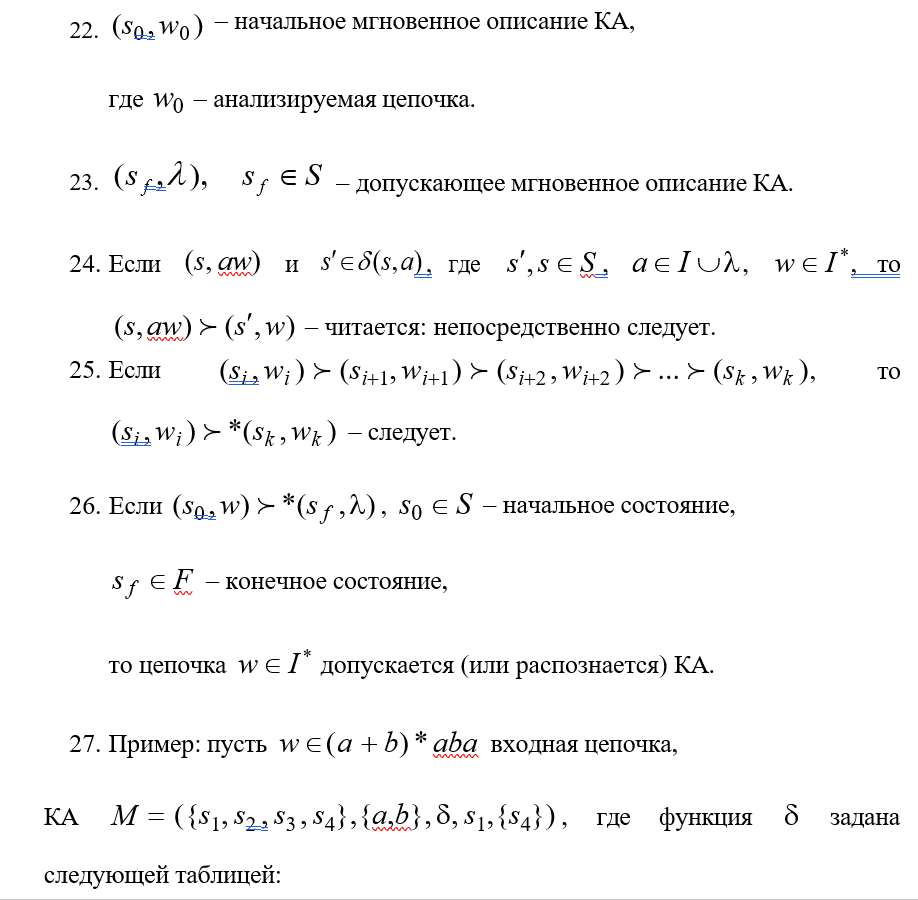
Читая входную цепочку символов {\displaystyle x} и делая переходы из состояния в состояние, автомат после прочтения последнего символа входного слова окажется в некотором конечном состоянии. {\displaystyle q'}.

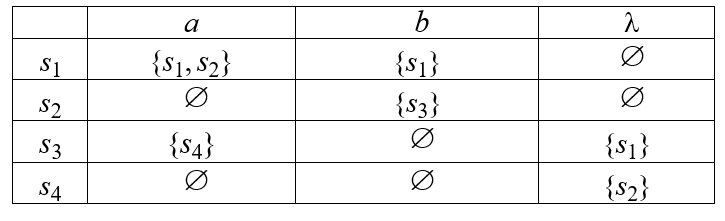
Если это состояние является заключительным, то говорят, что автомат допустил слово

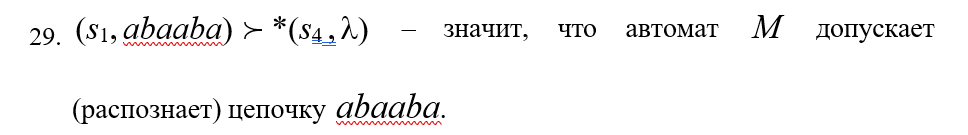
### 47. Конечный автомат (КА): недетерминированные и детерминированные КА. Мгновенное описание КА. Последовательность мгновенных описаний, диаграмма мгновенных описаний. Пример.











### 48. Конечный автомат: определение, схема работы, примеры. Соотношение регулярного языка, регулярной грамматики, регулярного языка и конечного автомата.

Определение КА: КА это пятерка M = (S, I, δ, s0 , F ), где:

* S – конечное множество состояний устройства управления;
* I – алфавит входных символов;
  + – функция переходов, отображающая S × (I ∪{λ}) во множество подмножеств S : δ(s,i) ⊂ S,s∈S,i∈ I;
* s0 ∈ S - начальное состояние устройства управления;
* F ⊆ S - множество заключительных (допускающих) состояний устройства управления.

**// (википедия** [математическая абстракция](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%B0%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D1%8F), [модель](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) [дискретного устройства](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE), имеющего один вход, один выход и в каждый момент времени находящегося в одном [состоянии](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5) из множества возможных)

*Другими словами:*

любой регулярный язык может быть задан регулярной грамматикой, регулярным выражением или конечным автоматом.

*Или:*

любой конечный автомат задает регулярный язык, а значит грамматику или регулярное выражение.

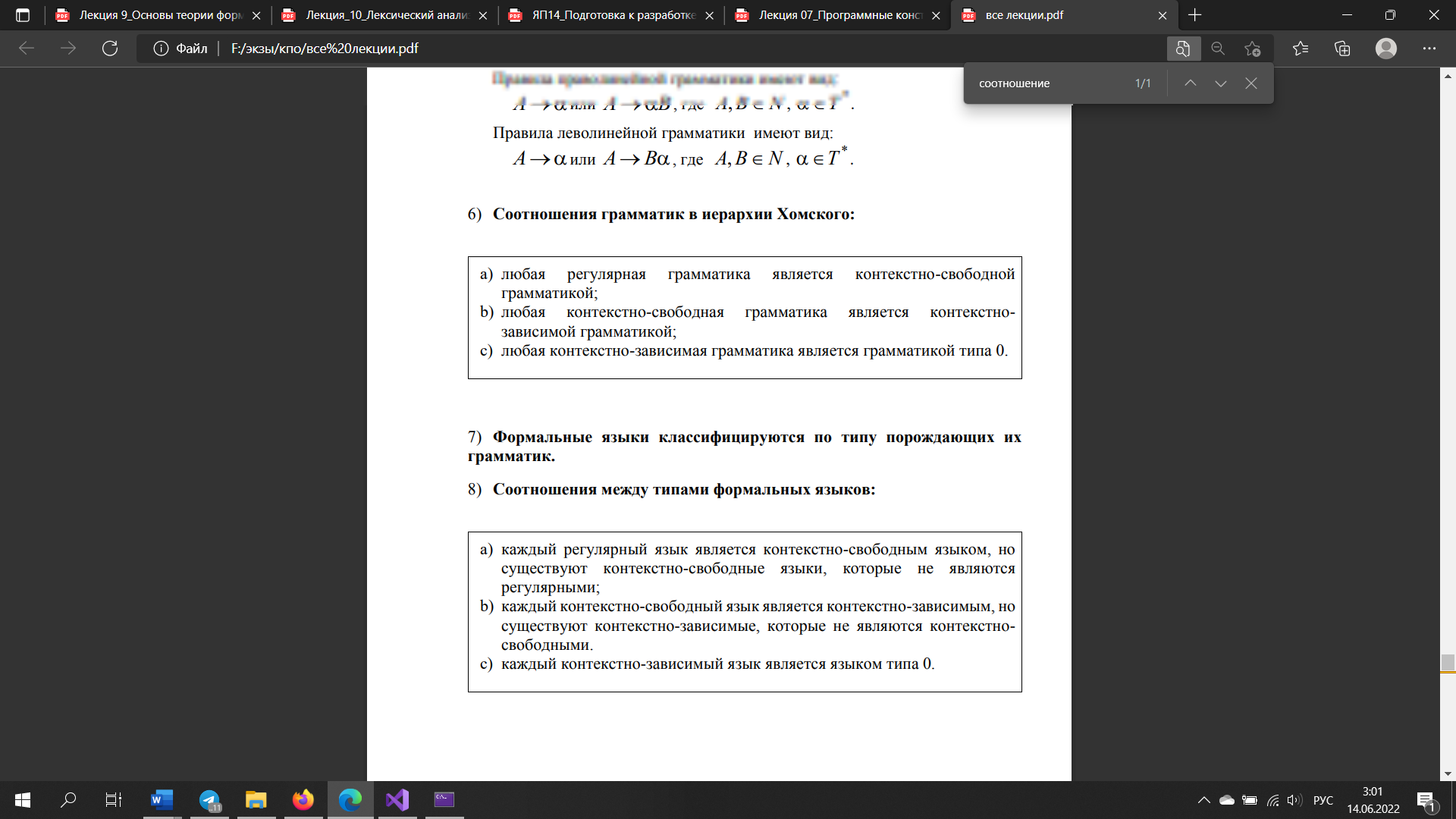
**Назначение**: Конечные автоматы широко используются на практике, например, в синтаксических и лексических анализаторах. Конечный автомат довольно хорошо применим в много шаговых интерфейсах, где следующий шаг зависит от того, какие параметры были выбраны на текущем (предыдущем) шаге.

**Схема работы**: Принято полагать, что конечный автомат начинает работу в начальном состоянии, последовательно считывая по одному символу входного слова (цепочки входных символов). Считанный символ переводит автомат в новое состояние в соответствии с функцией переходов.

Читая входную цепочку символов {\displaystyle x} и делая переходы из состояния в состояние, автомат после прочтения последнего символа входного слова окажется в некотором конечном состоянии. {\displaystyle q'}.

Если это состояние является заключительным, то говорят, что автомат допустил слово

1. язык является регулярным множеством тогда и только тогда, когда он задан регулярной грамматикой;
2. язык может быть задан регулярной грамматикой (левосторонней или правосторонней) тогда и только тогда, когда язык является регулярным множеством;
3. язык является регулярным множеством тогда и только тогда, когда он задан конечным автоматом;
4. язык распознается с помощью конечного автомата тогда и только тогда, когда он является регулярным множеством.
5. язык является регулярным множеством тогда и только тогда, когда он задан регулярной грамматикой;
6. язык может быть задан регулярной грамматикой (левосторонней или правосторонней) тогда и только тогда, когда язык является регулярным множеством;
7. язык является регулярным множеством тогда и только тогда, когда он задан конечным автоматом;
8. язык распознается с помощью конечного автомата тогда и только тогда, когда он является регулярным множеством.



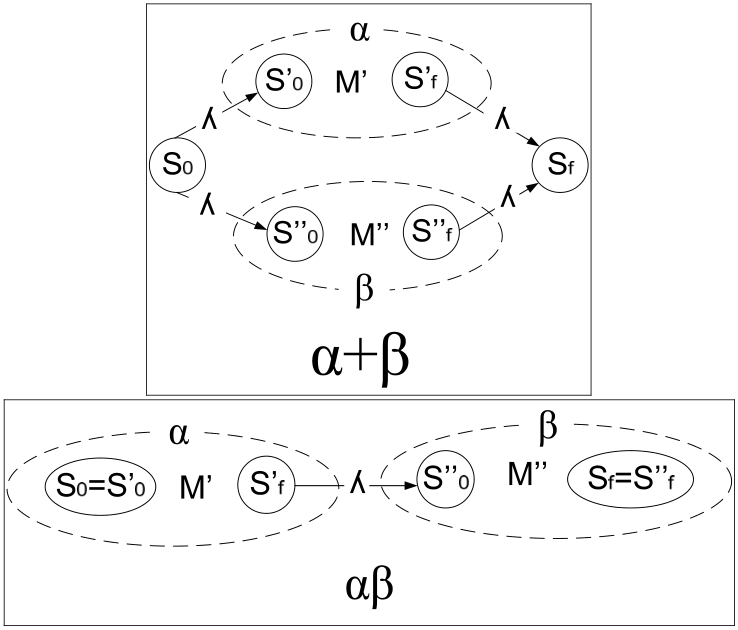
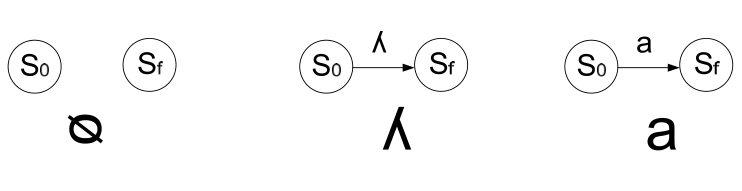
### 49. Конечный автомат: определение графа переходов конечного автомата и метод его построение по регулярному выражению. Примеры построения графов переходов.

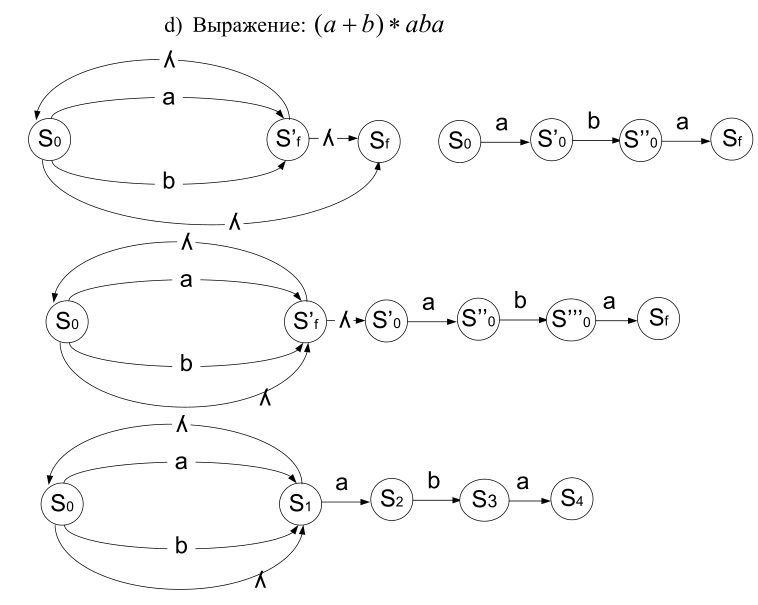
**Граф переходов** — графическое представление множества состояний и функции переходов.

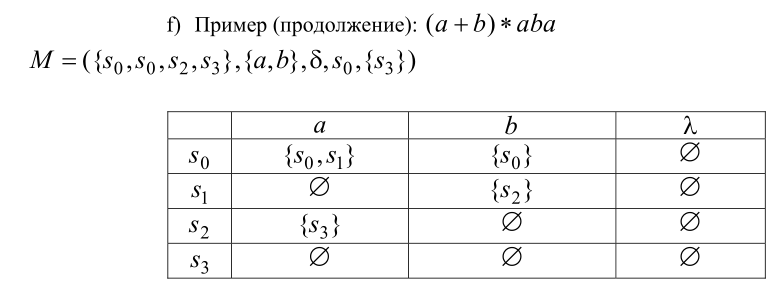
**Таблица переходов** — табличное представление функции *δ*. Обычно в такой таблице каждой строке соответствует одно состояние, а столбцу — один допустимый входной символ. В ячейке на пересечении строки и столбца записывается состояние, в которое должен перейти автомат, если в данном состоянии он считал данный входной символ.

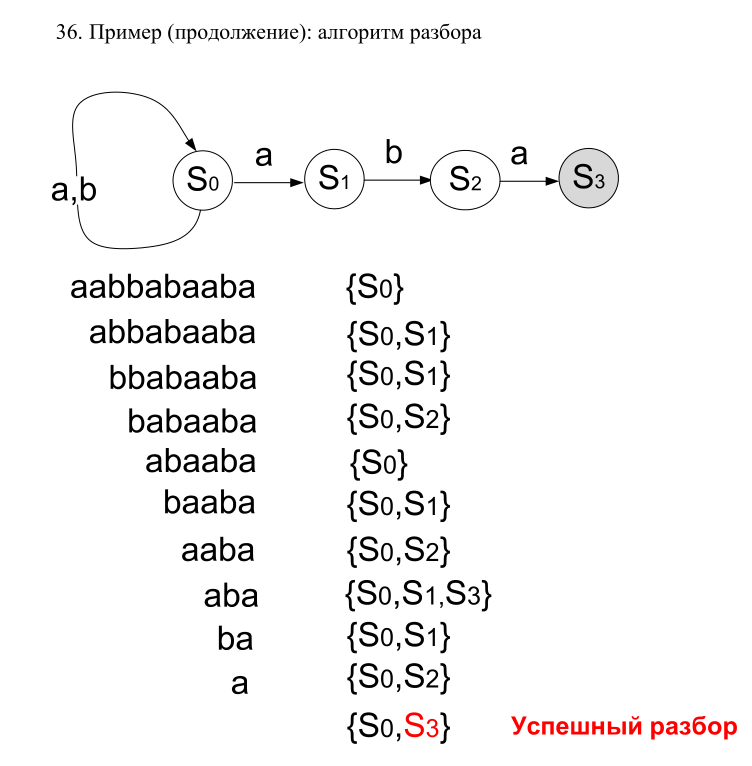
Построение графа конечного автомата по регулярному выражению:

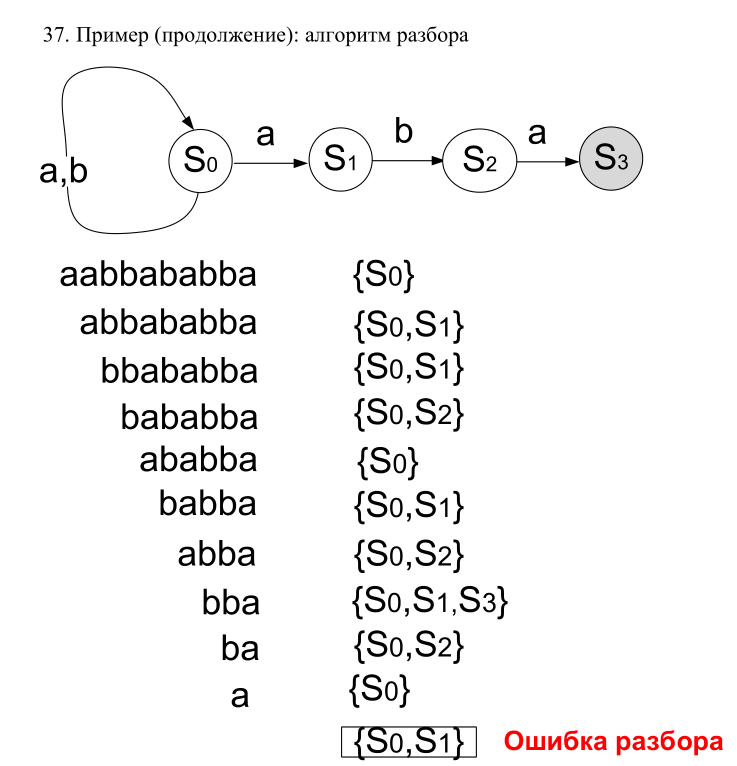
**Метод построения.** Автомат для выражения строится композицией из автоматов, соответствующих подвыражениям. На каждом шаге построения строящийся автомат имеет в точности одно заключительное состояние, в начальное состояние нет переходов из других состояний и нет переходов из заключительного состояния в другие.











### 50. Конечный автомат: алгоритм разбора цепочки символов, основанный на двух массивах.