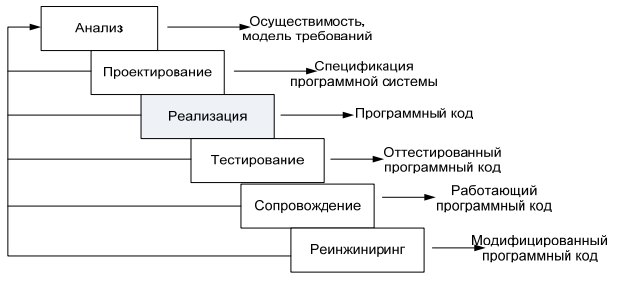
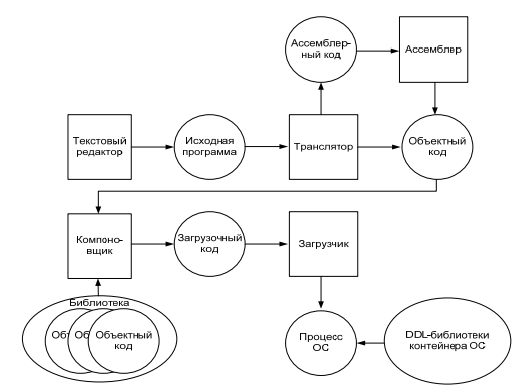
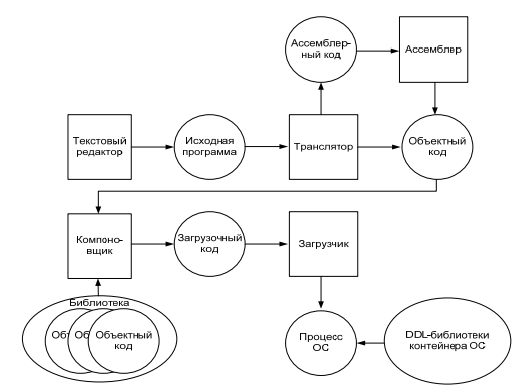
**1.Система программирования: определение, состав, назначение. Классический жизненный цикл разработки программного обеспечения**. **Текстовый редактор.** **Интегрированная среда разработки. Программный продукт**. **Общая схема преобразования исходного кода в процесс операционной системы**  
**Система программирования**: комплекс программных средств, предназначенных для автоматизации процесса разработки, отладки программного обеспечения и подготовки программного кода к выполнению. **Состав системы программирования**: трансляторы, компоновщики, отладчики, профилировщики, программные библиотеки, редакторы кода, системы поддержки версий и пр. **Система программирования используется в текстовых редакторах,** компиляторах, компановщиках, библиотеках прикладных программ, документации, загрузчиках, отладчиках и др.  
**Цикл разработки программного обеспечения:**  
  
**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 основныхфункций жизненного цикла разработки программы. **Интегрированная среда разработки:** редакторы кода, транслятор, компоновщик, отладчик, система поддержки версий. Примеры: Visual Studio, NetBeans, Eclipse, Embarcadero Delphi и пр. **Программный продукт: программа**, работающая без авторского присутствия. Программный продукт исполняется, тестируется, конфигурируется без присутствия автора и сопровождается документацией.

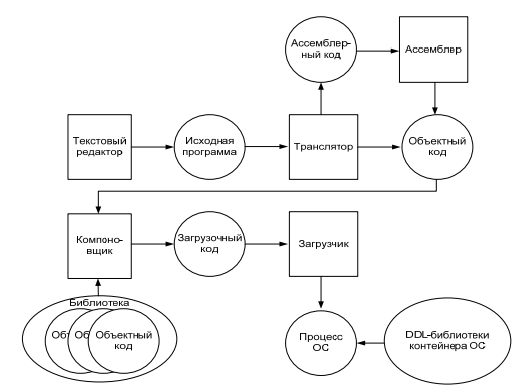


**2. Система программирования: определение, состав назначение**. **Интегрированная среда разработки: определение, состав, назначение. Транслятор: виды трансляторов**. **Транслятор: определение, назначение, примеры. Интерпретатор: определение, назначение, примеры**. **Система программирования**: комплекс программных средств, предназначенных для автоматизации процесса разработки, отладки программного обеспечения и подготовки программного кода к выполнению. **Состав системы программирования**: трансляторы, компоновщики, отладчики, профилировщики, программные библиотеки, редакторы кода, системы поддержки версий и пр**. Система программирования используется в** текстовых редакторах, компиляторах, компановщиках, библиотеках прикладных программ, документации, загрузчиках, отладчиках и др.  
**Интегрированная среда разработки** – набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы. **Интегрированная среда разработки:** редакторы кода, транслятор, компоновщик, отладчик, система поддержки версий. Примеры: Visual Studio, NetBeans, Eclipse, Embarcadero Delphi и пр**.** **Транслятор:** программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке**.** **Виды трансляторов:1.*Адресный*.** Функциональное устройство, преобразующее виртуальный адрес в реальный адрес памяти.**2.*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).**3.*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**. 4.*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)**. .5.О*днопроходной*.** Формирует объектный модуль за один последовательный просмотр исходной программы.**6.*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**.7.*Синтаксически-ориентированный*.** 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. **8.*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)** – разновидность транслятора. Переводит и выполняет программу с языка высокого уровня в машинный код строка за строкой.

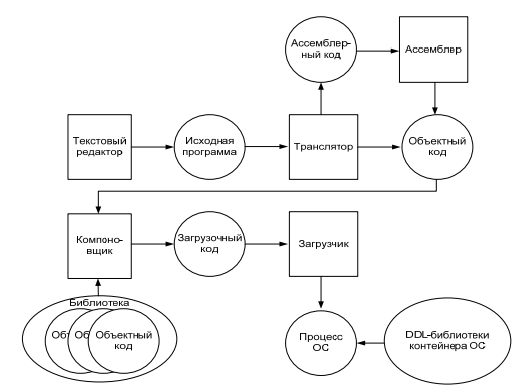
3. **Язык программирования: определение, назначение, примеры. Исходный код.** **Алфавит языка. Транслятор: определение, назначение, примеры**. **Объектный код, объектный модуль**. **Общая схема преобразования исходного кода в процесс операционной системы. Язык программирования**: формальная знаковая 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. Пример: Fortran, LISP, COBOL(1957-1959), Pascal(1970), С(1972), С++(1983),PHP, Python. **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**. Алфавит языка**: набор символов, разрешенных к использованию языком. Основывается на одной из кодировок. кодировка символов; **Транслятор:** программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке**.** **Объектный код**: результат работы транслятора, некий текст программы, написанный без специфических для какого-либо языка программирования команд, но в котором оставлены названия функций и операций, названия переменных и другое. Один файл объектного кода **– объектный модуль-**файл с промежуточным представлением отдельного модуля программы, полученный в результате обработки исходного кода компилятором**.**



**4. Язык программирования: определение, назначение, примеры. Исходный код.** **Язык ассемблера**. **Ассемблер**. **Транслятор c ассемблера: определение, назначение, примеры. Объектный код, объектный модуль.** **Общая схема преобразования исходного кода с языка ассемблера 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. Пример: Fortran, LISP, COBOL(1957-1959), Pascal(1970),С(1972), С++(1983),PHP, Python. **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**. Язык ассемблера**-машинно-ориентированный язык программирования (для конкретной архитектуры компьютера, команды которого соответствуют машинным командам). **Ассемблер**-транслятор с исходного кода на языке ассемблера в программу на машинном языке (язык, который может интерпретироваться процессором). **Транслятор -** программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке**. Объектный код**: результат работы транслятора, некий текст программы, написанный без специфических для какого-либо языка программирования команд, но в котором оставлены названия функций и операций, названия переменных и другое. Один файл объектного кода **– объектный модуль-**файл с промежуточным представлением отдельного модуля программы, полученный в результате обработки исходного кода компилятором.

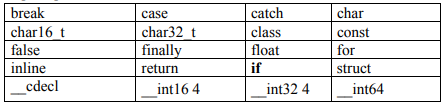
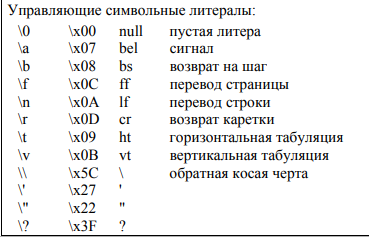


**5. Компоновщик: определение, назначение. Загрузочный модуль. Загрузчик: определение, назначение.** **Общая схема преобразования исходного кода в процесс операционной системы.** **Компоновщик (linker, редактор связей)**: программа, принимающая один или несколько объектных модулей и формирующая на их 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 или во время исполнения. **Загрузочный код-**результат работы компоновщика. Один файл загрузочного кода – **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.

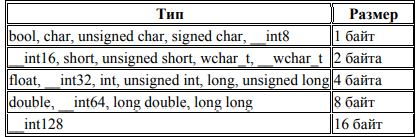
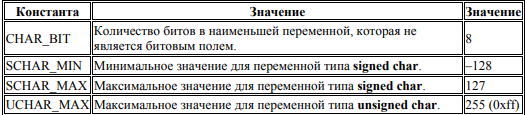


**6. Кодировка: определение, назначение, примеры. Кодировка ASCII. Структура кодировки Windows-1251. Отличие ASCII и Windows-1251. Кодировка UNICODE: назначение, структура, UCS, UTF. Кодировка** (charset) определяется как комбинация набора символов и схемы кодирования. В процессе кодирования информация преобразуется из формы, удобной для непосредственного использования, в форму, удобную для передачи, хранения или автоматической обработки. Средством кодирования служит таблица соответствия знаковых систем, которая устанавливает взаимно однозначное соответствие между знаками или группами знаков двух различных знаковых систем.  
**ASCII: American Standard Code for Information Interchange** — американский стандартный код для обмена информацией— 8-битная кодировка для представления десятичных цифр, латинского и национального алфавитов, знаков препинания и управляющих символов. Таблица кодов ASCII делится на две части. Международным стандартом является лишь первая половина таблицы, т.е. символы с номерами от 0 (00000000), до 127 (01111111) . **ASCII:** **альтернативная кодировка(CP866):** Все специфические европейские символы во второй половине таблицы заменены на кириллицу, а псевдографические символы оставлены без изменения. **ASCII**: **русская Windows-кодировка, или же Windows-1251** — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии. Visual Studio 2012 хранит содержимое cpp-файлов и h-файлов в кодировке Windows-1251. **UNICODE**: Стандарт предложен в 1991 году Unicode Consortium.**Юникод** – это стандарт кодирования символов, позволяющий представить знаки почти всех письменных языков, состоит из 2х разделов:1)**UCS** - universal character set (универсальный набор символов);2)**UTF** - Unicode transformation format (семейство кодировок).UCS расположены в 17 плоскостях (0-16), 2^16 символов в каждой плоскости, плоскость 0 – основная (основные символы), 1-14 – дополнительные, 15-16 – для частного использования. **UTF-8** — представление Юникода, обеспечивающее совместимость со старыми системами, использовавшими 8-битные символы. Алгоритм кодирования в UTF-8: 1) определить количество октетов (октет: 8 битов) – в какой диапазон значений попадает количество значащих символов (7, 11, 16, 21, 26, 31); 2) подготовить старшие биты первого октета: a. 0xxxxxxx для одного октета; b. 110xxxxx – двух; c. 1110xxxx - трех и т.д.. d. 10xxxxxx - для остальных октетов; 3) заполнить оставшиеся биты (обозначены как x) в октетах кодом символа Юникода в двоичном виде. Начать с младших битов, поставив их в младшие биты последнего октета кода. И так далее, пока все биты кода символа не будут перенесены в свободные биты октетов.В **UTF-16** символы кодируются двухбайтовыми словами с использованием всех возможных диапазонов значений (от 0 до FFFF16). ). Запись символа с помощью 4-х байт называется **суррогатной парой**. UTF-16 бывает двух видов: UTF-16BE и UTF-16LE LE (Little endian order, прямой порядок, от младшего к старшему), BE (Big endian order, обратный порядок, от старшего к младшему). . **BOM (Byte Order Mark)** Для определения формата представления Юникода в начало текстового файла записывается сигнатура — символ U+FEFF — маркер последовательности байтов.

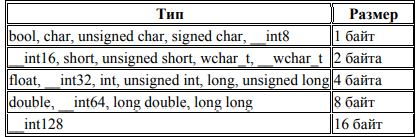
7**. Кодировка: определение, назначение, примеры.** **Кодировка UNICODE: назначение, структура, UCS, UTF. Прямой (LE) и обратный (BE) порядок байт. BOM: определение, назначение, примеры.** **Кодировка (charset)-**определяется как комбинация набора символов и схемы кодирования. Разбиение данных на кодовые слова. В процессе кодирования информация преобразуется из формы, удобной для непосредственного использования, в форму, удобную для передачи, хранения или автоматической обработки. Средством кодирования служит таблица соответствия знаковых систем, которая устанавливает взаимно однозначное соответствие между знаками или группами знаков двух различных знаковых систем.  
**UNICODE**: Стандарт предложен в 1991 году Unicode Consortium.**Юникод** – это стандарт кодирования символов, позволяющий представить знаки почти всех письменных языков, состоит из 2х разделов:1)**UCS** - universal character set (универсальный набор символов);2)**UTF** - Unicode transformation format (семейство кодировок).UCS расположены в 17 плоскостях (0-16), 2^16 символов в каждой плоскости, плоскость 0 – основная (основные символы), 1-14 – дополнительные, 15-16 – для частного использования. **UTF-8** — представление Юникода, обеспечивающее совместимость со старыми системами, использовавшими 8-битные символы. Алгоритм кодирования в UTF-8: 1) определить количество октетов (октет: 8 битов) – в какой диапазон значений попадает количество значащих символов (7, 11, 16, 21, 26, 31); 2) подготовить старшие биты первого октета: a. 0xxxxxxx для одного октета; b. 110xxxxx – двух; c. 1110xxxx - трех и т.д.. d. 10xxxxxx - для остальных октетов; 3) заполнить оставшиеся биты (обозначены как x) в октетах кодом символа Юникода в двоичном виде. Начать с младших битов, поставив их в младшие биты последнего октета кода. И так далее, пока все биты кода символа не будут перенесены в свободные биты октетов.В **UTF-16** символы кодируются двухбайтовыми словами с использованием всех возможных диапазонов значений (от 0 до FFFF16). Запись символа с помощью 4-х байт называется **суррогатной парой**. UTF-16 бывает двух видов: UTF-16BE и UTF-16LE (big-endian / little-endian). В них разный порядок байт. **LE** (Little endian order, прямой порядок, от младшего к старшему), **BE** (Big endian order, обратный порядок, от старшего к младшему). Big-endian — «естественный» порядок байт, как у арабских цифр. Little-endian — обратный порядок байт. **BOM (Byte Order Mark)** Для определения формата представления Юникода в начало текстового файла записывается сигнатура — символ U+FEFF — маркер последовательности байтов.  


8. **Структура языка программирования: идентификаторы,** **зарезервированные идентификаторы, литералы, ключевые слова. Определения, примеры (C++)**. **Идентификатор -** имя компонента программы (переменной, функции, метки, типа и пр.), составленное программистом по определенным правилам**. Правила идентификатор C++:**идентификаторы должны начинаться с буквы или подчеркивания; не может совпадать с ключевыми словами С++ или с именами библиотечных функций; ; идентификаторы чувствительны к регистру; могут состоять из любого количества символов, но компилятор гарантирует, что будет считать значащими:1)31 первых символов идентификаторов, не имеющих внешней связи;2)не более 6 значащих символов идентификаторов с внешней связью. Длина идентификатора по стандарту не ограничена. Идентификатор создается при объявлении переменной, функции, типа и т. п**. Зарезервированные идентификаторы-**идентификаторы, которые предварительно определены в системе программирования. **Зарезервированные идентификаторы С++:** все имена с двумя подчеркиваниями считаются зарезервированными. **Пример:** isxxxx, memxxxx, strxxxx, toxxxx, wcsxxxx, Eцифраxxxx, LC\_Xxxx, SIGXxxx, SIG\_Xxxxx**. Ключевые слова:** последовательности символов алфавита языка, имеющие специальное назначение. Ключевые слова зарезервированы компилятором для обозначения типов переменных, класса хранения, элементов операторов и т.д.   
  
**Литерал:** элемент программы, который непосредственно представляет значение.   В C++ существует четыре типа литералов: - целочисленный литерал, - вещественный литерал, - символьный литерал, - строковый литерал.  


**9. Структура языка программирования: фундаментальные типы данных. Определение, примеры (C++). Строки. Массивы данных фундаментального** **типа. Преобразование типов.**  **Типы данных:**1)фундаментальные (или встроенные);2) определенные программистом. **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. **C++ определены следующие ключевые слова:** - int (целый); - char (символьный); - wchar\_t (расширенный символьный); - bool (логический); - float (вещественный); - double (вещественный с двойной точностью); - тип void. Модификаторы основных типов, уточняющие внутреннее представление и диапазон значений стандартных типов: - short (короткий); - long (длинный); - signed (знаковый); - unsigned (беззнаковый). **Размеры основных типов для встроенных типов в Microsoft C++:**

  
**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 зависит от реализации. Два стандартных включаемых заголовочных файла, и , определяют числовые ограничения или минимальное и максимальное значения, которые может хранить переменная данного типа. Ограничения для некоторых целочисленных типов, заданные в стандартном файле заголовка , представлены в таблице:   
  
В языке C++ для удобной работы со строками есть класс string, для использования которого необходимо подключить заголовочный файл string. Строки в С++ представляются как массивы элементов типа char, заканчивающиеся нуль-терминатором \0 называются С строками или строками в стиле С. Строки можно объявлять и одновременно присваивать им значения: string S1, S2 = "Hello".  
**Массив** — это структура **данных**, которая содержит последовательность элементов. одинакового **типа**. Тип данных элементов массива может быть произвольным. Размер массива считается в соответствии с количеством данных и с их типом.   **Преобразование типов:** автоматическое преобразование, явное преобразование**. явные** :задаётся программистом в тексте программы с помощью: - конструкции языка; - функции, принимающей значение одного типа и возвращающей значение другого типа**. неявные** :выполняется автоматически транслятором по правилам, описанным в стандарте языка. Стандарты большинства языков запрещают неявные преобразования.

**10. Структура языка программирования: фундаментальные типы данных. Преобразование типов: автоматическое преобразование, явное преобразование. Типы данных:** - фундаментальные (или встроенные); - определенные программистом. . **C++ определены следующие ключевые слова:** - int (целый); - char (символьный); - wchar\_t (расширенный символьный); - bool (логический); - float (вещественный); - double (вещественный с двойной точностью); - тип void. **Размеры основных типов для встроенных типов в Microsoft C++:**

   
**Преобразование типов:** автоматическое преобразование, явное преобразование**. явные** :задаётся программистом в тексте программы с помощью: конструкции языка;функции, принимающей значение одного типа и возвращающей значение другого типа**. неявные** :выполняется автоматически транслятором (компилятором или интерпретатором) по правилам, описанным в стандарте языка. Стандарты большинства языков запрещают неявные преобразования. **Автоматическое (неявное) преобразование типов**: для базовых типов bool, [unsigned/signed] char, short, int, long, float, double, long double преобразование типов выполняется без потери точности**. Явное преобразование**: часто применяется для указания того, что преобразование делается осознано. Таким образом, механизм неявных преобразований может быть отключён посредством явного указания в тексте программы требуемого преобразования типов. **Ч**аще всего применяется для приведения void\* к некоторому типу**. Явное преобразование приведения типов в C++:** приведение типов const\_cast; приведения типов на этапе выполнения программы dynamic\_cast; приведения несовместимых типов reinterprent\_cast; приведения типов на этапе компиляции программы static\_cast (может отслеживать недопустимые преобразования).

**11. Структура языка программирования: пользовательские типы данных. Определение, примеры (C++). Массивы данных пользовательского типа.** **Перегрузка операторов для пользовательских типов**. Язык C++ позволяет программисту определять свои типы данных и правила работы с ними. **Переименование типов (typedef):** Для того чтобы сделать программу более ясной, можно задать типу новое имя с помощью ключевого слова typedef:typedef тип новое\_имя [ размерность ]; **Перечисления (enum):**При написании программ часто возникает потребность определить несколько именованных констант, для которых требуется, чтобы все они имели различные значения. Для этого удобно воспользоваться перечисляемым типом данных, все возможные значения которого задаются списком целочисленных констант. Формат:enum [ имя\_типа ] { список\_констант }; **Структуры (struct)** Структура может содержать элементы разных типов. В языке C++ структура является видом класса и обладает всеми его свойствами, но во многих случаях достаточно использовать структуры так, как они определены в языке С:struct [ имя\_типа ] { тип\_1 элемент\_1;тип\_2 элемент\_2;тип\_n элемент\_n;} [ список\_описателей ]; **Битовые поля** — это особый вид полей структуры. Они используются для плотной упаковки данных, например, флажков типа «да/нет». **Объединения (union).** Объединение (union) представляет собой частный случай структуры, все поля которой располагаются по одному и тому же адресу. Формат описания такой же, как у структуры, только вместо ключевого слова struct используется слово union. **Перегрузка** **операторов** (operator overloading) — это возможность применять встроенные **операторы** языка к разным **типам**, в том числе и **пользовательским**. **Перегрузка операторов в программировании —** реализация в одной области видимости нескольких различных вариантов применения оператора, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются. Синтаксис: <тип>operator<символ\_оператора> (список\_параметров) Перегруженные операторы реализуются в виде функции.  
**Массив пользовательского типа -** объект для хранения именованных переменных и массивов различных размерностей с максимальной скоростью доступа. Цель использования: максимальная скорость доступа; минимальная нагрузка на оперативную память; не нужно запоминать имена переменных (подсвечиваются, если поставить точку); возможность хранить внутри каждого из индексов массива другой массив с динамически изменяющейся размерностью; возможность быстро очистить основной массив.

**12. Структура языка программирования: область видимости переменных, пространства имен, псевдонимы пространства имен.** **Область видимости переменных в C++-**доступность переменных по их идентификатору в разных частях (блоках программы). Переменная должна быть объявлена до ее использования; переменная объявленная во внутреннем блоке (локальная переменная {…}) не доступна во внешнем; переменная объявленная во внешнем блоке доступна во внутреннем; во внутреннем блоке переменная может быть переобъявлена. Область видимости переменной (идентификатора) зависит от места ее объявления в тексте программы. Область действия идентификатора — это часть программы, в которой его можно использовать для доступа к связанной с ним области памяти. В зависимости от области действия переменная может быть локальной или глобальной. **Пространство имен**: именованная область видимости. Применяется для разрешения конфликтов имен. Пространство имен в C++: namespace, using, псевдонимы пространства имен. Для доступа к определенным внутри области объектам, используется **оператор разрешения видимости «::».Псевдоним пространства имен используется** для задания короткого синонима имени пространства.Например, длинное имя :namespace International\_Business\_Machines { /\* ... \*/ } может быть ассоциировано с более коротким синонимом: namespace IBM = International\_Business \_Machines; Объявление псевдонима начинается ключевым словом namespace, за которым следует короткий псевдоним, а за ним – знак равенства и исходное полное имя пространства. Если полное имя не соответствует никакому известному пространству, это ошибка. Псевдоним может относиться и к вложенному пространству имен.

**13. Структура языка программирования**: **инициализация переменных, присвоение значений, выражения (lvalue, rvalue, побочные эффекты, точка последовательности,** **унарные**, **бинарные и тернарные выражения), константные выражения.**   
**Инициализация переменных (памяти):** присвоение значения в момент объявления переменной; применяется литералы. Отличие от присвоения: при присвоении явно перемещаются данные**. Выражение –** объединение литералов, имен (переменных, функций и пр.), операторов и специальных символов, служащих для вычисления выражения или достижения побочных эффектов. Выражения состоят из операндов, знаков операций и скобок. Операнды выражения задают данные для вычислений. Операции задают действия, которые необходимо выполнить. Операнд может быть выражением, литералом или переменной. Порядок вычисления выражения с операторами одинакового приоритета не определен. Если при вычислении выражения значение переменной, входящей в это выражение, изменилось, то **произошел побочный эффект**. Побочный эффект является изменением состояния среды выполнения. **Побочный эффект** (side effect) возникает, когда функция или выражение изменяет состояние объекта или вызывает другие функции, которые имеют побочные эффекты. **Выражения lvalue: lvalue** – это ссылка на значение; могут использоваться в левой и правой части оператора присваивания. Имя переменной, ссылка на элемент массива по индексу, вызов функции возвращающей указатель, всегда связаны с областью памяти, адрес которой известен. Lvalue:выражения, непосредственно обозначающие объект; выражения ссылочных типов; результат операции разыменования (\*); результат префиксных операций ++, ‐‐; имя массива; строковые литералы. Выражения **rvalue** – может использоваться только в правой части оператора присваивания (не связано с адресом, связано только со значением; это могут быть литералы, вызов функции, возвращающей значение). **rvalue:** выражения, обозначающие временные объекты; результат операции взятия адреса (&); результат постфиксных операций ++, ‐‐; литералы за исключением строковых; константы перечислений**. Символ окончания последовательности** (в C++ точка с запятой) определяет точку последовательности, в которой завершились все вычисления и побочные эффекты. **Точка последовательности** – момент времени, когда побочные эффекты вычисленных выражений уже случились, а побочные эффекты следующих в последовательности выражений еще не начались. Точка последовательности в С++ обозначается символом точка с запятой (;). Выражения могут иметь двоякое толкование и могут в разных реализациях компилятора вычисляться по-разному. **Список точек следования в C++**: в конце выражения; после вычисления всех аргументов в вызове функции и до выполнения любых выражений в ее теле; после копирования возвращаемого значения функции и до выполнения любого выражения вне функции; после вычисления первого выражения в a && b, a || b, a ? b : c или a, b; после инициализации каждого базового класса и члена в списке инициализации конструктора. **Типы выражений:** **Унарные выражения –** выражения с одним операндом. Унарные операторы действуют только на один операнд в выражении, имеют ассоциативность справа налево. **Примеры**: оператор косвенного обращения (\*), оператор взятия адреса (&), оператор (+), унарного сложения оператор унарного отрицания (-), оператор логического отрицания (!), оператор дополнения (~), оператор префиксного инкремента (++), оператор префиксного декремента (--), оператор sizeof, оператор new, оператор delete. **Бинарные выражения –** выражения с двумя операндами. Мультипликативные операторы: умножение (\*), деление (/), остаток (%). Аддитивные операторы: сложение (+), вычитание (-). Операторы сдвига сдвиг вправо (>>), сдвиг влево (<), меньше или равно (<=), больше или равно (>=), равно (==), не равно (!=). Побитовые операторы. Логические операторы логическое и (& &), логическое или (||).**Выражения:** Присваивание стирает старое значение из переменной, затем записывает в нее новое значение. Выражения: Операция запятая (,) имеет самый низкий приоритет среди всех операций языка C++. У этой операции может быть 2 и более операндов. Вначале вычисляется первый операнд, затем второй и так далее, а в качестве результата возвращается последний операнд. Выражения: Тернарный оператор (?) принимает три операнда:первый операнд неявно преобразуется в bool; если первый операнд имеет значение true (1), то вычисляется второй операнд;если первый операнд имеет значение false (0), то вычисляется третий операнд. Условные выражения имеют ассоциативность справа налево.

**14. Структура языка программирования: инструкции языка программирования, объявление, простые и составные инструкции, инструкции выбора, инструкции циклов, инструкции переходов, инструкции обработки исключений, примеры.** Составная инструкция – это последовательность простых, заключенная в фигурные скобки.  
**Инструкции языка**: инструкции объявления ( if (int) условное объявление);составные инструкции ({}); инструкции выбора (if, switch); циклы (while, do-while, for); − инструкции перехода (goto, break, continue, return); − обработка исключений (try, catch, throw).  
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. **Инструкции объявления:** В С++ определение объекта, например: **int ival;** рассматривается как инструкция объявления (хотя в данном случае более правильно было бы сказать определения). Ее можно использовать в любом месте программы, где разрешено употреблять инструкции. **Инструкция if**: Инструкция if обеспечивает выполнение или пропуск инструкции или блока в зависимости от условия. Ее синтаксис таков: **if ( условие )инструкция**  
условие заключается в круглые скобки.Область видимости объекта, объявленного в условной части, ограничивается ассоциированной с if инструкцией или блоком. **Инструкция switch:** Альтернативный метод выбора одного их взаимоисключающих условий предлагает инструкция switch. **Инструкция switch состоит из следующих частей: 1)** ключевого слова switch, за которым в круглых скобках идет выражение, являющееся условием;2) набора меток case, состоящих из ключевого слова case и константного выражения, с которым сравнивается условие.3) последовательности инструкций, соотносимых с метками case.4) необязательной метки default, которая является аналогом части else инструкции if-else. **Инструкция цикла for:** В С++ предусмотрено три инструкции для организации циклов, в частности for и while, которые начинаются проверкой условия. Такая проверка означает, что цикл может закончиться без выполнения связанной с ним простой или составной инструкции. Третий тип цикла, do while, гарантирует, что тело будет выполнено как минимум один раз: условие цикла проверяется по его завершении. Синтаксис цикла for следующий: for (инструкция-инициализации; условие; выражение ) инструкция  
порядок вычислений будет следующим:1)инструкция-инициализации выполняется один раз перед началом цикла.2) Вычисляется условие. Если оно равно true, выполняется составная инструкция тела цикла.3) Вычисляется выражение. **Инструкция while:** Синтаксис инструкции while следующий: **while ( условие ) инструкция** Пока значением условия является true, инструкция выполняется в такой последовательности:**1.**Вычислить условие.**2.**Выполнить инструкцию, если условие истинно. **3.**Если самое первое вычисление условия дает false, инструкция не выполняется. **Инструкция do while:** Синтаксис цикла do while таков**: Do инструкция while ( условие );**

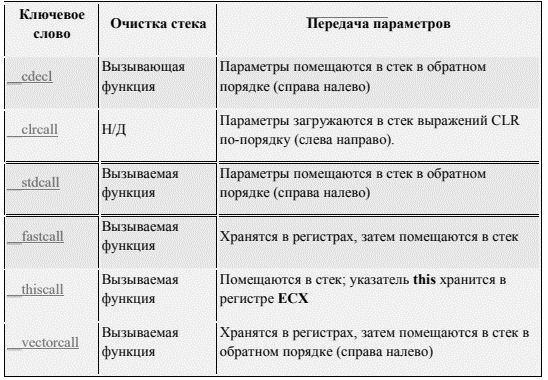
инструкция выполняется до первой проверки условия. Если вычисление условия дает false, цикл останавливается. **Инструкция break:** Инструкция break останавливает циклы for, while, do while и блока switch. Если эта инструкция используется внутри вложенных циклов или инструкций switch, она завершает выполнение того внутреннего блока, в котором находится. Цикл или switch, включающий тот цикл или switch, из которого мы вышли с помощью break, продолжает выполняться. **Инструкция continue** завершает текущую итерацию цикла и передает управление на вычисление условия, после чего цикл может продолжиться. В отличие от инструкции break, завершающей выполнение всего цикла, инструкция continue завершает выполнение только текущей итерации. **Инструкция goto** обеспечивает *безусловный*переход к другой инструкции внутри той же функции, поэтому современная практика программирования выступает против ее применения. Синтаксис goto следующий: goto метка**;**

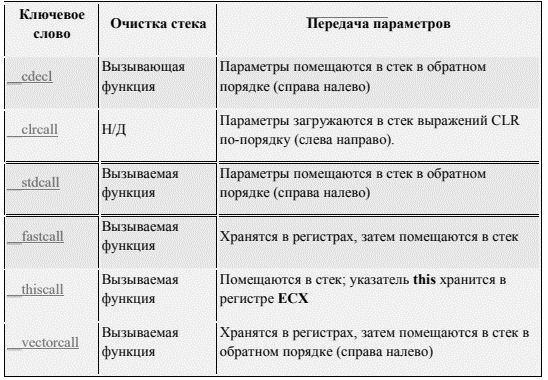
**15. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров. Передача параметров по значению и по ссылке, передача переменного числа параметров в функции C++.**  
**Программные конструкции:** функции; объявление функции; определение; параметры. **Функция** — фрагмент программного кода, к которому можно обратиться из другого места программы. С функцией связывается идентификатор (имя функции). Некоторые языки допускают безымянные функции. С именем функции неразрывно связан адрес ее первой инструкции, которой передаётся управление при обращении к функции. После выполнения функции управление возвращается в точку программы, где данная функция была вызвана (адрес возврата). Функция может принимать параметры и может возвращать некоторое значение. Функции, которые возвращают пустое значение, называют процедурами. Функция должна быть объявлена и определена. Объявление функции содержит: имя функции; список имён и типов передаваемых параметров (или аргументов); тип возвращаемого функцией значения. Определение функции содержит исполняемый код функции. В одних языках программирования определение следует непосредственно за объявлением функции. В других языках необходимо сначала объявить функцию, а потом привести её определение. Для вызова функции необходимо в требуемом месте программного кода указать имя функции и перечислить передаваемые в функцию параметры. Передача параметров в функцию: по значению; 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. Функция определяет собственную (локальную) область видимости, куда входят входные параметры, а также те переменные, которые объявляются непосредственно в теле самой функции. Функция — подпрограмма, выполняющая какие-либо операции и возвращающая значение. Процедура — подпрограмма, которая выполняет операции, и не возвращает значения. Метод — это функция или процедура, которая принадлежит классу или экземпляру класса. Программные конструкции: лямбда-выражение. **Передача параметров в функцию** происходит через стек. Код, вызывающий функцию, знает, сколько параметров ей передать и каковы значения этих параметров

16. **Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров. Передача параметров по значению и по ссылке, передача значений параметров по умолчанию в функции C++.**  
Программные конструкции: функции; объявление функции; определение; параметры. **Функция** — фрагмент программного кода, к которому можно обратиться из другого места программы. С функцией связывается идентификатор (имя функции). Некоторые языки допускают безымянные функции. С именем функции неразрывно связан адрес ее первой инструкции, которой передаётся управление при обращении к функции. После выполнения функции управление возвращается в точку программы, где данная функция была вызвана (адрес возврата). Функция может принимать параметры и может возвращать некоторое значение. Функции, которые возвращают пустое значение, называют процедурами. Функция должна быть объявлена и определена. Объявление функции содержит: имя функции; список имён и типов передаваемых параметров (или аргументов); тип возвращаемого функцией значения. Определение функции содержит исполняемый код функции. В одних языках программирования определение следует непосредственно за объявлением функции. В других языках необходимо сначала объявить функцию, а потом привести её определение. Для вызова функции необходимо в требуемом месте программного кода указать имя функции и перечислить передаваемые в функцию параметры. Передача параметров в функцию: по значению; 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. Функции, принимающие значения, заданные по умолчанию: 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 аргументам. Функция определяет собственную (локальную) область видимости, куда входят входные параметры, а также те переменные, которые объявляются непосредственно в теле самой функции. Функция — подпрограмма, выполняющая какие-либо операции и возвращающая значение. Процедура — подпрограмма, которая выполняет операции, и не возвращает значения. Метод — это функция или процедура, которая принадлежит классу или экземпляру класса. Программные конструкции: лямбда-выражение. **Передача параметров в функцию** происходит через стек. Код, вызывающий функцию, знает, сколько параметров ей передать и каковы значения этих параметров

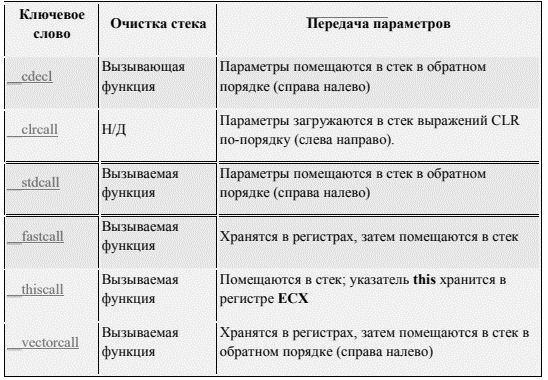
**17. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров.** **Реализация программных конструкций в C++. Лямбда-функции в С++.** Программные конструкции: функции; объявление функции; определение; параметры. **Функция** — фрагмент программного кода, к которому можно обратиться из другого места программы. С функцией связывается идентификатор. Некоторые языки допускают безымянные функции. С именем функции неразрывно связан адрес ее первой инструкции, которой передаётся управление при обращении к функции. После выполнения функции управление возвращается в точку программы, где данная функция была вызвана. Ф-ция может принимать параметры и может возвращать некоторое значение. Ф-ции, которые возвращают пустое значение, называют процедурами. Ф-ция должна быть объявлена и определена. Объявление ф-ции содержит: имя ф-ции; список имён и типов передаваемых параметров (или аргументов); тип возвращаемого ф-цией значения. Определение ф-ции содержит исполняемый код ф-ции. Для вызова ф-ции необходимо в требуемом месте программного кода указать имя функции и перечислить передаваемые в ф-цию параметры. Для переменной, переданной по значению, создаётся локальная копия и любые изменения, которые происходят в теле ф-ции, происходят с локальной копией и не изменяют значения самой переменной. Для переменной, переданной по ссылке, изменения происходят с самой переменной. Ф-ция определяет собственную область видимости, куда входят входные параметры, те переменные, которые объявляются непосредственно в теле самой ф-ции. Процедура — подпрограмма, которая выполняет операции, и не возвращает значения. Метод-это ф-ция/процедура, которая принадлежит классу или экземпляру класса. **Лямбда-выражение в программировании** — специальный синтаксис для определения функциональных объектов, заимствованный из λ-исчисления. Лямбда-выражения представляют собой анонимные функции, которые можно определить в любом месте программы. Структура лямбда-выражения: [ маска\_переменных ] ( список\_параметров ) mutable throw() -> возвращаемый\_тип { /\* тело\_лямбда-выражения \*/ } **Маска переменных (обязательно)**.Маска определяет способ получения параметров телом лямбда выражения. Лямбда может использовать переменные и объекты, которые были объявлены вне лямбда-выражения: [a,&b] a захвачена по значению (копия), b захвачена по ссылке; [this] захватывает указатель this по значению; [&] захват всех символов по ссылке; [=] захват всех символов по значению; [] ничего не захватывает. Список параметров (необязательно). Список параметров лямбда-выражения, аналогичен записи аргументов для обычных функций. Лямбда-выражение может принимать в качестве параметра другое лямбда выражение. C++14 позволяет объявлять параметры лямбда со спецификатором типа autо и использовать параметры по умолчанию. mutable (необязательно) — позволяет модифицировать параметры, переданные по значению. throw() (необязательно) — спецификация исключений. 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.

18. **Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров. Реализация программных конструкций в C++.** **Перегружаемые функции в C++.** **Inline-функции в C++. Шаблоны функций С++.** Программные конструкции: функции;объявление функции; определение; параметры. **Ф-ция** — фрагмент программного кода, к которому можно обратиться из другого места программы. С ф-цией связывается идентификатор. С именем ф-ции неразрывно связан адрес ее первой инструкции, которой передаётся управление при обращении к ф-ции. После выполнения ф-ции управление возвращается в точку программы, где данная функция была вызвана. Ф-ция может принимать параметры и может возвращать некоторое значение. Ф-ции, которые возвращают пустое значение, называют процедурами. Ф-ция должна быть объявлена и определена. Объявление ф-ции содержит: имя ф-ции; список имён и типов передаваемых параметров/аргументов; тип возвращаемого ф-цией значения. Определение ф-ции содержит исполняемый код ф-ции. Для вызова ф-кции необходимо в требуемом месте программного кода указать имя функции и перечислить передаваемые в ф-цию параметры. Для переменной, переданной по значению, создаётся локальная копия и любые изменения, которые происходят в теле ф-ции, происходят с локальной копией и не изменяют значения самой переменной. Для переменной, переданной по ссылке, изменения происходят с самой переменной. Ф-ция определяет собственную область видимости, куда входят входные параметры и переменные, которые объявляются непосредственно в теле самой ф-ции. Процедура — подпрограмма, которая выполняет операции, и не возвращает значения. Метод — это функция или процедура, которая принадлежит классу или экземпляру класса. **Передача параметров в ф-цию** происходит через стек.**Перегружаемые ф-ции:** Перегрузка процедур и ф-ций обеспечивает возможность использования одноимённых подпрограмм: процедур или ф-ций. При трансляции происходит контроль одноимённых процедур и ф-ций, чтобы они различались по сигнатуре.Чтобы исключить ошибку повтора имя подпрограммы в некоторых языках вводится требование написания дополнительного ключевого слова. Под перегрузкой ф-ции понимается, определение 2 и более ф-ций с одинаковым именем в одной области видимости, но с различными параметрами. Наборы параметров перегруженных ф-ций могут отличаться порядком следования, количеством, типом. 2 ф-ции, с разной сигнатурой, но одинаковыми именами - перегруженные ф-ции. Сигнатура – это комбинация имени функции с её параметрами. Компилятор самостоятельно выберает нужную ф-цию, анализируя сигнатуры перегруженных функций. **Программные конструкции С++:** inline (подставляемые, встраиваемые) функции; inline – указание компилятору для выполнения подстановки, компилятор может не выполнить подстановку, если не установить параметры. Встроенные ф-ции или inline-ф-ции определяются с помощью ключевого слова inline. Каждый раз, вместо вызова этой ф-ции, компилятор будет заменять его фактическим кодом ф-ции. Вызовы ф-ций занимают больше времени, чем выполнение последовательного кода. Но при такой замене увеличивается размер программы. **Шаблоны ф-ций:** обобщенное описание семейства функций. Шаблоны функций определяют семейство функций. С помощью шаблонов функций можно задавать наборы функций, основанных на одном коде, но действующих для разных типов. Шаблон ф-ции начинается с ключевого слова template, за которым в угловых скобках следует список параметров

**19. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров. Реализация программных конструкций в C++. Соглашения о вызовах (\_cdecl) в С++.** Программные конструкции: функции; − объявление функции; определение;параметры. **Функция** — фрагмент программного кода, к которому можно обратиться из другого места программы. С функцией связывается идентификатор (имя функции). Некоторые языки допускают безымянные функции. С именем функции неразрывно связан адрес ее первой инструкции, которой передаётся управление при обращении к функции. После выполнения функции управление возвращается в точку программы, где данная функция была вызвана (адрес возврата). Функция может принимать параметры и может возвращать некоторое значение. Функции, которые возвращают пустое значение, называют процедурами. Функция должна быть объявлена и определена. Объявление функции содержит: имя функции; список имён и типов передаваемых параметров (или аргументов); тип возвращаемого функцией значения. Определение функции содержит исполняемый код функции. В одних языках программирования определение следует непосредственно за объявлением функции. В других языках необходимо сначала объявить функцию, а потом привести её определение. Для вызова функции необходимо в требуемом месте программного кода указать имя функции и перечислить передаваемые в функцию параметры. Передача параметров в функцию: по значению; 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. Функция определяет собственную (локальную) область видимости, куда входят входные параметры, а также те переменные, которые объявляются непосредственно в теле самой функции. Функция — подпрограмма, выполняющая какие-либо операции и возвращающая значение. Процедура — подпрограмма, которая выполняет операции, и не возвращает значения. Метод — это функция или процедура, которая принадлежит классу или экземпляру класса. Соглашение о вызовах – это протокол для передачи аргументов функциям. **Передача параметров в функцию** происходит через стек. Код, вызывающий функцию, знает, сколько параметров ей передать и каковы значения этих параметров. 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:** 1. 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. 2. 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. 3. Возвращаемый параметр передается через регистр EAX. Соглашение о вызовах:  


**20. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров. Реализация программных конструкций в C++. Соглашения о вызовах (\_stdcall) в С++.**   
Программные конструкции: − функции; − объявление функции; − определение; − параметры. **Функция** — фрагмент программного кода, к которому можно обратиться из другого места программы. С функцией связывается идентификатор (имя функции). Некоторые языки допускают безымянные функции. С именем функции неразрывно связан адрес ее первой инструкции, которой передаётся управление при обращении к функции. После выполнения функции управление возвращается в точку программы, где данная функция была вызвана (адрес возврата). Функция может принимать параметры и может возвращать некоторое значение. Функции, которые возвращают пустое значение, называют процедурами. Функция должна быть объявлена и определена. Объявление функции содержит: имя функции; список имён и типов передаваемых параметров (или аргументов); тип возвращаемого функцией значения. Определение функции содержит исполняемый код функции. В одних языках программирования определение следует непосредственно за объявлением функции. В других языках необходимо сначала объявить функцию, а потом привести её определение. Для вызова функции необходимо в требуемом месте программного кода указать имя функции и перечислить передаваемые в функцию параметры. Передача параметров в функцию: по значению; 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. Функция определяет собственную (локальную) область видимости, куда входят входные параметры, а также те переменные, которые объявляются непосредственно в теле самой функции. Функция — подпрограмма, выполняющая какие-либо операции и возвращающая значение. Процедура — подпрограмма, которая выполняет операции, и не возвращает значения. Метод — это функция или процедура, которая принадлежит классу или экземпляру класса. **Передача параметров в функцию** происходит через стек. Код, вызывающий функцию, знает, сколько параметров ей передать и каковы значения этих параметров. 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 Соглашение о вызовах – это протокол для передачи аргументов функциям. 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 через стек, порядок параметров справа налево, стек освобождает вызываемый код, возврат через регистр EAX. Соглашение о вызовах:  


21. **Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров**. **Реализация программных конструкций в C++. Соглашения о вызовах (\_fastcall) в С++.**Программные конструкции: − функции; − объявление функции; − определение; − параметры. **Функция** — фрагмент программного кода, к которому можно обратиться из другого места программы. С функцией связывается идентификатор (имя функции). Некоторые языки допускают безымянные функции. С именем функции неразрывно связан адрес ее первой инструкции, которой передаётся управление при обращении к функции. После выполнения функции управление возвращается в точку программы, где данная функция была вызвана (адрес возврата). Функция может принимать параметры и может возвращать некоторое значение. Функции, которые возвращают пустое значение, называют процедурами. Функция должна быть объявлена и определена. Объявление функции содержит: имя функции; список имён и типов передаваемых параметров (или аргументов); тип возвращаемого функцией значения. Определение функции содержит исполняемый код функции. В одних языках программирования определение следует непосредственно за объявлением функции. В других языках необходимо сначала объявить функцию, а потом привести её определение. Для вызова функции необходимо в требуемом месте программного кода указать имя функции и перечислить передаваемые в функцию параметры. Передача параметров в функцию: по значению; 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. Функция определяет собственную (локальную) область видимости, куда входят входные параметры, а также те переменные, которые объявляются непосредственно в теле самой функции. Функция — подпрограмма, выполняющая какие-либо операции и возвращающая значение. Процедура — подпрограмма, которая выполняет операции, и не возвращает значения. Метод — это функция или процедура, которая принадлежит классу или экземпляру класса. Соглашение о вызовах – это протокол для передачи аргументов функциям. **Передача параметров в функцию** происходит через стек. Код, вызывающий функцию, знает, сколько параметров ей передать и каковы значения этих параметров. 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 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 \_fastcall (не стандартизированный, для внутренних вызовов), параметры передаются через регистры (первых 2 через регистры, остальные справа на лево в стек), стек освобождает вызываемый код, возврат через регистр EAX. В Borland (Delphi) (параметры слева на право). Первые два параметра передаются через 2 регистра. Для передачи остальных параметров, используется стек; порядок параметров "справа налево". Соглашение о вызовах:

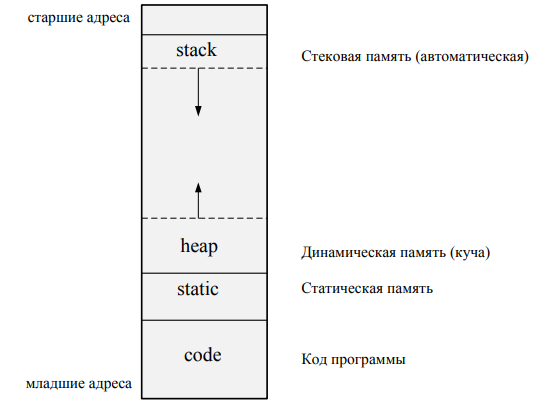
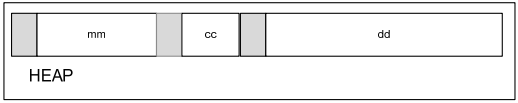


**22. Структура языка программирования: стандартная библиотека. Реализация стандартной библиотеки STL в C++. Понятие контейнера, итератора и алгоритма.** **Стандартные функции STL для работы со строками: копирования, сравнения, вычисления длины**, поиска символа и подстроки.   
**Стандартная Библиотека** означает коллекцию [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%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)) и [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%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)), написанных на [базовом языке](https://ru.wikipedia.org/w/index.php?title=%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA&action=edit&redlink=1). Стандартная Библиотека поддерживает несколько основных контейнеров, функций для работы с этими контейнерами, объектов-функции, основных типов строк и потоков (включая интерактивный и файловый ввод-вывод), поддержку некоторых языковых особенностей, и часто используемые функции для выполнения таких задач, как, например, нахождение [квадратного корня](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D1%80%D0%B5%D0%BD%D1%8C) числа.  
Большинство библиотек поддерживает как статическое связывание (для связывания библиотеки непосредственно в коде), так и динамическое связывание (для использования в коде общих библиотек DLL). При сборке финальной версии проекта одна из базовых библиотек времени выполнения C (libcmt.lib, msvcmrt.lib, msvcrt.lib) из таблицы будет скомпонована по умолчанию, в зависимости от выбранного параметра компилятора. При включении в код любого из файлов заголовков стандартной библиотеки C++, Visual C++ автоматически подключит во время компиляции стандартную библиотеку C++. STL — 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. Основные компоненты библиотеки STL: 1) контейнеры + итераторы; 2) алгоритмы; 3) потоки ввода-вывода. Контейнер – объект, содержащий другие объекты (стек, список, очередь, вектор и пр.), предназначенный для хранения однотипных объектов и обеспечения доступа к ним. Например: класс vector – определяет динамический массив; класс stack – определяет стек; класс list – позволяет работать с линейным списком; класс deque – создает двухстороннюю очередь. Контейнеры: последовательные (последовательный доступ к элементам); ассоциативные (доступ по ключу). Алгоритм определяет вычислительную процедуру (обобщённые алгоритмы) для работы с контейнерами. Алгоритмы позволяют манипулировать содержимым контейнера: инициализировать, сортировать, искать, изменять содержимое контейнера. Итератор: объект, обеспечивающий для алгоритма средство доступа к содержимому контейнера. Итератор позволяет перемещаться по содержимому контейнера подобно тому, как указатель перемещается по элементам массива. Итераторы: произвольного доступа (для ассоциативных контейнеров); двунаправленные (для последовательных).  
В языке Си предусмотрено множество разных функций, предназначенных для работы со строками. Для их использования необходимо подключить заголовочный файл **string.h**

Разберём самые простые из них: **strlen(str)** – длина строк , которая ей передана в качестве аргумента. **str**; **strcmp(str1, str2)** – сравнение строк; Данная функция сравнивает посимвольно строки, переданные ей в качестве аргументов. Функция **strcmp** вернёт нуль, если строки равны между собой, иначе какое-либо другое целое число (положительное или отрицательное). Общее правило таково: Если в функции **strcmp()** первая строка больше, чем вторая строка, то функция возвращает положительное число. Если меньше – отрицательное. **str1** и **str2**; **strcat(str1, str2)** – конкатенация (склеивание) двух строк. К строке **str1** в конце приклеивается строка **str2**. **strcpy(str1, str2)** – копирование строки **str2** в строку **str1**;  В классе string поиск символа/подстроки можно делать двумя способами, которые отличаются направлением поиска:1.путем просмотра строки от начала до конца с помощью функции find();2.путем просмотра строки от конца к началу функцией rfind().

23**. Структура языка программирования: стандартная библиотека. Реализация стандартной библиотеки STL в C++. Понятие контейнера, итератоFfра и алгоритма.** **Стандартные функции STL для работы с широкими строками: копирования, сравнения, вычисления длины, поиска символа и подстроки.** **Стандартная Библиотека** означает коллекцию [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%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)) и [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%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)), написанных на [базовом языке](https://ru.wikipedia.org/w/index.php?title=%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA&action=edit&redlink=1). Стандартная Библиотека поддерживает несколько основных контейнеров, функций для работы с этими контейнерами, объектов-функции, основных типов строк и потоков (включая интерактивный и файловый ввод-вывод), поддержку некоторых языковых особенностей, и часто используемые функции для выполнения таких задач, как, например, нахождение [квадратного корня](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D1%80%D0%B5%D0%BD%D1%8C) числа. Большинство библиотек поддерживает как статическое связывание (для связывания библиотеки непосредственно в коде), так и динамическое связывание (для использования в коде общих библиотек DLL). При сборке финальной версии проекта одна из базовых библиотек времени выполнения C (libcmt.lib, msvcmrt.lib, msvcrt.lib) из таблицы будет скомпонована по умолчанию, в зависимости от выбранного параметра компилятора. При включении в код любого из файлов заголовков стандартной библиотеки C++, Visual C++ автоматически подключит во время компиляции стандартную библиотеку C++. STL — 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. Основные компоненты библиотеки STL: 1) контейнеры + итераторы; 2) алгоритмы; 3) потоки ввода-вывода. Контейнер – объект, содержащий другие объекты (стек, список, очередь, вектор и пр.), предназначенный для хранения однотипных объектов и обеспечения доступа к ним. Например: класс vector – определяет динамический массив; класс stack – определяет стек; класс list – позволяет работать с линейным списком; класс deque – создает двухстороннюю очередь. Контейнеры: последовательные (последовательный доступ к элементам); ассоциативные (доступ по ключу). Алгоритм определяет вычислительную процедуру (обобщённые алгоритмы) для работы с контейнерами. Алгоритмы позволяют манипулировать содержимым контейнера: инициализировать, сортировать, искать, изменять содержимое контейнера. Итератор: объект, обеспечивающий для алгоритма средство доступа к содержимому контейнера. Итератор позволяет перемещаться по содержимому контейнера подобно тому, как указатель перемещается по элементам массива. Итераторы: произвольного доступа (для ассоциативных контейнеров); двунаправленные (для последовательных). В языке Си предусмотрено множество разных функций, предназначенных для работы со строками. Для их использования необходимо подключить заголовочный файл **string.h**.Разберём самые простые из них: **strlen(str)** – длина строк , которая ей передана в качестве аргумента. **str**; **strcmp(str1, str2)** – сравнение строк; Данная функция сравнивает посимвольно строки, переданные ей в качестве аргументов. Функция **strcmp** вернёт нуль, если строки равны между собой, иначе какое-либо другое целое число (положительное или отрицательное). Общее правило таково: Если в функции **strcmp()** первая строка больше, чем вторая строка, то функция возвращает положительное число. Если меньше – отрицательное. **str1** и **str2**; **strcat(str1, str2)** – конкатенация (склеивание) двух строк. К строке **str1** в конце приклеивается строка **str2**. **strcpy(str1, str2)** – копирование строки **str2** в строку **str1**;  В классе string поиск символа/подстроки можно делать двумя способами, которые отличаются направлением поиска:1.путем просмотра строки от начала до конца с помощью функции find();2.путем просмотра строки от конца к началу функцией rfind().

**24. Структура языка программирования: стандартная библиотека. Реализация стандартной библиотеки STL в C++. Понятие контейнера, итератора и алгоритма. Стандартные функции для работы с потоками данных. Стандартные функции для получения текущей даты, даты и времени.** **Стандартная Библиотека** означает коллекцию [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%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)) и [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%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)), написанных на [базовом языке](https://ru.wikipedia.org/w/index.php?title=%D0%91%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA&action=edit&redlink=1). Стандартная Библиотека поддерживает несколько основных контейнеров, функций для работы с этими контейнерами, объектов-функции, основных типов строк и потоков (включая интерактивный и файловый ввод-вывод), поддержку некоторых языковых особенностей, и часто используемые функции для выполнения таких задач, как, например, нахождение [квадратного корня](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D1%80%D0%B5%D0%BD%D1%8C) числа.  
Большинство библиотек поддерживает как статическое связывание (для связывания библиотеки непосредственно в коде), так и динамическое связывание (для использования в коде общих библиотек DLL). При сборке финальной версии проекта одна из базовых библиотек времени выполнения C (libcmt.lib, msvcmrt.lib, msvcrt.lib) из таблицы будет скомпонована по умолчанию, в зависимости от выбранного параметра компилятора. При включении в код любого из файлов заголовков стандартной библиотеки C++, Visual C++ автоматически подключит во время компиляции стандартную библиотеку C++. STL — 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. Основные компоненты библиотеки STL: 1) контейнеры + итераторы; 2) алгоритмы; 3) потоки ввода-вывода. Контейнер – объект, содержащий другие объекты (стек, список, очередь, вектор и пр.), предназначенный для хранения однотипных объектов и обеспечения доступа к ним. Например: класс vector – определяет динамический массив; класс stack – определяет стек; класс list – позволяет работать с линейным списком; класс deque – создает двухстороннюю очередь. Контейнеры: последовательные (последовательный доступ к элементам); ассоциативные (доступ по ключу). Алгоритм определяет вычислительную процедуру (обобщённые алгоритмы) для работы с контейнерами. Алгоритмы позволяют манипулировать содержимым контейнера: инициализировать, сортировать, искать, изменять содержимое контейнера. Итератор: объект, обеспечивающий для алгоритма средство доступа к содержимому контейнера. Итератор позволяет перемещаться по содержимому контейнера подобно тому, как указатель перемещается по элементам массива. Итераторы: произвольного доступа (для ассоциативных контейнеров); двунаправленные (для последовательных). **Стандартные функции для получения текущей даты, даты и времени:** Чтобы получить доступ к функциям и структурам, связанным с датой и временем, нужно будет включить заголовочный файл <ctime>. Существует четыре типа времени: clock\_t, time\_t, size\_t и tm. . time\_t time (time\_t \* time); Это возвращает текущее время календаря системы в количестве секунд, прошедших с 1 января 1970 года. char \* ctime (const time\_t \* time); Это возвращает указатель на строку формы день месяц год: минуты: секунды год \ n \ 0 .struct tm \* localtime (const time\_t \* time); Это возвращает указатель на структуру tm, представляющую локальное время. clock\_t clock (void); Это возвращает значение, которое приблизительно соответствует времени, в течение которого вызываемая программа была запущена. char \* asctime (const struct tm \* time); Это возвращает указатель на строку, содержащую информацию, хранящуюся в структуре, на которую указывает время, преобразованное в форму: day month date hours: minutes: seconds year \ n \ 0; struct tm \* gmtime (const time\_t \* time); Это возвращает указатель на время в виде структуры tm. Время представлено в скоординированном универсальном времени (UTC), которое по сути является средним временем по Гринвичу (GMT). time\_t mktime (struct tm \* time); Это возвращает эквивалент календарного времени времени, найденного в структуре, на которую указывает время.double difftime (time\_t time2, time\_t time1);Эта функция вычисляет разницу в секундах между временем 1 и временем 2. size\_t strftime (); Эта функция может использоваться для форматирования даты и времени в определенном формате. Потоки вывода (ostream)  
Для вывода данных используеся оператор <<. Этот опрератор определен для всех встроенных типов C++ и некоторых классов, входящих в стандартную библиотеку. Для вывода перевода строки можно использовать специальный объект endl.  
Некоторые методы класса ostream:1)put(char c) - записать символ *с* в поток2)write(const char\* s, streamsize n) - записать первые n элементов массива s в поток (streamsize представляет целое число со знаком, например, int)3)flush() - записать значение из буфера4)close() - закрытие потока; Для ввода используется оператор >>. Он также определен для всех встроенных типов и некоторых классов стандартной библиотеки.Оператор >> также можно определить для своих классов: Некоторые методы класса istream:1)get() - считать следующий символ2)get(char \*buf, streamsize n) - считать максимум n-1 символ и поместить в массив buf 3)get(char \*buf, streamsize n, char delim) - считывание символов до символа-разделителя delim (разделитель не считывается и остается в потоке)4)getline(char \*buf, streamsize n)5)getline(char \*buf, streamsize n, char delim)6)peek() - считывает следующий символ, но оставляет его в потоке7)ignore(streamsize n = 1, int delim = EOF) - извлекает символы из потока до тех пор, пока их число меньше n или пока не встретился символ delim;8)putback (char c) - добавляет символ с в текущую позицию потока;9)unget() - возвращает последний считанный символ в поток; Основные манипуляторы ввода/вывода:boolalpha - стороковое представление логических значений; noboolalpha - числовое представление логических значений; showbase - включает вывод 0 перед восьмеричными и 0x перед шестнадцатеричными числами;noshowbase - выключает вывод 0 и 0x;dec - вывод чисел в десятичной системе счисления; oct - в восьмеричной;hex - в шестнадцатеричной;uppercase - заглавные буквы в записи шестнадцатеричных чисел и чисел с плавающей запятой в научной записи; nouppercase - строчные буквы в записи чисел; Для реализации стандартного ввода/вывода в библиотеку C++ включен заголовочный файл iostream, содержащий следующие предопределенные объекты потоков: 1)cin - стандартный поток ввода (соответствует потоку C stdin)2)cout - стандартный поток вывода (соответствует stdout)3)cerr - стандартный поток вывода ошибок (соответствует stderr)4)clog - стандартный поток вывода журнала (соответствует stderr)

**25. Классы памяти: код, стек, статические данные, динамическая область. Структура динамической памяти (Heap) C++. Фрагментация динамической памяти C++.**Структура памяти C-программ:   
Память, которую используют программы, состоит из нескольких частей — **сегментов**:1.**Сегмент кода** (или *«текстовый сегмент»*), где находится скомпилированная программа. Обычно доступен только для чтения. 2.**Сегмент bss** (или *«неинициализированный сегмент данных»*), где хранятся глобальные и [**статические переменные**](https://ravesli.com/urok-51-staticheskie-peremennye/), инициализированные нулем.3.**Сегмент данных** (или *«сегмент инициализированных данных»*), где хранятся инициализированные глобальные и статические переменные.4.**Куча**, откуда выделяются динамические переменные.5.**Стек вызовов**, где хранятся параметры функции, локальные переменные и другая информация, связанная с функциями. Классы памяти C++: объявление глобальных статических переменных с ключевым словом extern (компонуются редактором связей (linker)): Это глобальные переменные и к ним можно обращаться именами из любой функции. Поскольку внешние переменные доступны везде, их можно использовать для связи между функциями, не пренебрегая механизму формальных параметров.  
Класс памяти extern в Си используем в двух случаях:  
1.если переменная объявляется в программе ниже, чем ссылка на неё;2.если переменная объявлена в другом модуле.локальная статическая память: Спецификатор static указывает компилятору на хранение локальной переменной во время всего жизненного цикла программы вместо ее создания и разрушения при каждом входе в область действия и выходе из неё. Следовательно, возведение локальных переменных в ранг статистических позволяет поддерживать их значения между вызовами функций. Модификатор static можно также применять к глобальным переменным. В этом случае область видимости такой переменной ограничивается файлом, в котором она объявлена, это означает, что переменная будет иметь внутреннюю привязку. Внутренняя привязка говорит о том, что индикатор известен только внутри своего файла. Если функция объявлена как static, то она видна только в своём файле. Из другого файла к static функции обратиться нельзя.  
Стек (англ. stack — стопка; читается стэк) — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO.**Стек имеет свои преимущества и недостатки:**1.Выделение памяти в стеке происходит сравнительно быстро.2.Память, выделенная в стеке, остается в области видимости до тех пор, пока находится в стеке. Она уничтожается при выходе из стека.3.Вся память, выделенная в стеке, обрабатывается во время компиляции, следовательно, доступ к этой памяти осуществляется напрямую через переменные.4.Поскольку размер стека является относительно небольшим, то не рекомендуется делать что-либо, что съест много памяти стека (например, [**передача по значению**](https://ravesli.com/urok-97-peredacha-argumentov-po-znacheniyu/) или создание локальных переменных больших массивов или других *затратных* структур данных). **Сегмент кучи** (или просто ***«куча»***) отслеживает память, используемую для динамического выделения. В языке C++ при использовании оператора new динамическая память выделяется из сегмента кучи самой программы. **Куча имеет свои преимущества и недостатки:**1.Выделение памяти в куче сравнительно медленное.2.Выделенная память остается выделенной до тех пор, пока не будет освобождена (остерегайтесь утечек памяти) или пока программа не завершит свое выполнение.3.Доступ к динамически выделенной памяти осуществляется только через указатель. Разыменование указателя происходит медленнее, чем доступ к переменной напрямую.4.Поскольку куча представляет собой большой резервуар памяти, то именно она используется для выделения больших [**массивов**](https://ravesli.com/urok-74-massivy-chast-1/), [**структур**](https://ravesli.com/urok-61-struktury/) или классов.  


**26. Механизм обработки исключений: определение, назначение, применение. Реализация обработки исключений в C++.** Обработка ошибок: механизм исключений. Исключение – событие при выполнении программы, при котором ее дальнейшее выполнение становится бессмысленным. Для реализации **обработки** **исключений** в **C++** используйте выражения try, throw и catch. Блок try {…} позволяет включить один или несколько операторов, которые могут создавать **исключение**.  
В языке C++ **оператор throw** используется для сигнализирования о возникновении исключения или ошибки (аналогия тому, когда свистит арбитр). Сигнализирование о том, что произошло исключение, называется **генерацией исключения**(или ***«выбрасыванием исключения»***). Для использования оператора throw применяется ключевое слово throw, а за ним указывается значение любого типа данных, которое вы хотите задействовать, чтобы сигнализировать об ошибке. Как правило, этим значением является код ошибки, описание проблемы или настраиваемый класс-исключение. В языке C++ мы используем **ключевое слово** **try** для определения блока стейтментов (так называемого ***«блока try»***). Блок try действует как наблюдатель в поисках исключений, которые были выброшены каким-либо из операторов в этом же блоке try. Фактически, обработка исключений — это работа блока(ов) catch. **Ключевое слово catch** используется для определения блока кода (так называемого ***«блока catch»***), который обрабатывает исключения определенного типа данных. Блоки try и catch работают вместе. Блок try обнаруживает любые исключения, которые были выброшены в нем, и направляет их в соответствующий блок catch для обработки. Блок try должен иметь, по крайней мере, один блок catch, который находится сразу же за ним, но также может иметь и несколько блоков catch, размещенных последовательно (друг за другом).

Как только исключение было поймано блоком try и направлено в блок catch для обработки, оно считается обработанным (после выполнения кода блока catch), и выполнение программы возобновляется.

Параметры catch работают так же, как и параметры функции, причем параметры одного блока catch могут быть доступны и в другом блоке catch (который находится за ним). Исключения фундаментальных типов данных могут быть пойманы [**по значению**](https://ravesli.com/urok-97-peredacha-argumentov-po-znacheniyu/) (параметром блока catch является значение), но исключения нефундаментальных типов данных должны быть пойманы [**по константной ссылке**](https://ravesli.com/urok-98-peredacha-argumentov-po-ssylke/) (параметром блока catch является константная ссылка), дабы избежать ненужного копирования.

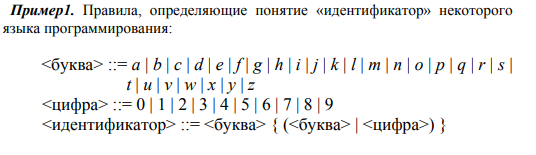
27**. Препроцессор: определение, назначение, применение,** **директивы,** выражения**, макросы. Примеры на C++.**Препроцессор: часть транслятора, которая выполняется до процесса трансляции;выполняет директивы препроцессора; результатом выполнения препроцессора является текст, сформированный из исходного под управлением директив препроцессора. Препроцессор — программа для обработки текста. Может быть отдельной программой, или интегрированной 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. Препроцессор C/С++: текстовый процессор, который обрабатывает текст исходного файла на первой фазе компиляции. Инструкции, регламентирующие работу компилятора, называются директивами препроцессора. Назначение: директивы препроцессора могут: − заменить какие-то лексемы в исходном тексте; − вставить содержимое других файлов в указанном месте; − подавить компиляцию части файла. Директивы препроцессора могут появляться в произвольном месте исходного текста, при этом они будут воздействовать только на оставшуюся часть исходного файла. Препроцессор можно вызвать отдельно для обработки текста программы без ее компиляции. Основные директивы препроцессора: #include — вставляет текст из указанного файла; #define — задаёт макроопределение или символическую константу; #undef — отменяет предыдущее определение; #if — осуществляет условную компиляцию при истинности константного выражения; #ifdef — осуществляет условную компиляцию при определённости символической константы; #ifndef — осуществляет условную компиляцию при неопределённости символической константы; #else — ветка условной компиляции при ложности выражения; #elif — ветка условной компиляции, образуемая слиянием else и if; #endif — конец ветки условной компиляции; #line — препроцессор изменяет номер текущей строки и имя компилируемого файла; #error — выдача диагностического сообщения; #pragma — действие, зависящее от конкретной реализации компилятора. директивы #define и #undef . 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. Директивы препроцессора располагаются в одной строке. Если требуется продолжить макрос на следующей строке, надо явно указать признак продолжения – символ обратный слэш (\). Функциональные макросы. 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. 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.

**28. Статическая библиотека: определение, назначение, применение. Реализация статической библиотеки в Windows C++. Утилита LIB. Компоновка статической библиотеки. Применение директивы препроцессора #pragma comment для компоновки 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. Расширения объектных файлов модулей статических библиотек: для ОС Microsoft Windows .lib; . 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:** Не забываем определить имя решения, имя 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. Страница свойств проекта: 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 Debug размещен файл статической библиотеки (.lib). 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. Для компановки на странице свойств проекта в разделе «Компоновщик» -> Командная строка в окне «Дополнительные параметры» вводим параметр, указывающий имя, местоположение статической библиотеки.Автоматическое связывание (от англ. Auto-linking) — механизм автоматического определения необходимых библиотек при сборке программ на языках С/C++. Активируется при помощи строки вида: #pragma comment(lib, )

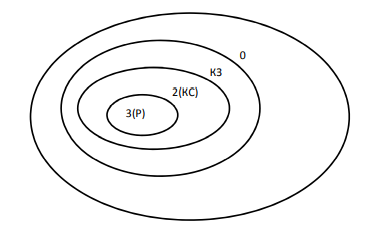
29. **Теория формальных языков: алфавит языка, цепочка, пустая цепочка, длина цепочки**, **равенство (эквивалентность) цепочек, конкатенация цепочек,** **итерация цепочки,** **операторы + и \*.**   
**Алфавит** – конечное непустое множество элементов языка. Алфавит задает язык I, состоящий из цепочек вида ab, aabb, aaabbb и т.д. Язык I представляет собой бесконечное множество цепочек, но его описание состоит всего из 8 символов, т.е. конечно. **Цепочка** – конечная последовательность символов языка. Алфавит будем обозначать заглавными буквами латинского алфавита: I, V, G, ... Символы алфавита будем обозначать строчными буквами латинского алфавита: a, b, c, ... Цепочки будем обозначать символами греческого алфавита: α, β, γ, δ, λ, ε, ω,… Пустую цепочку символов будем обозначать λ, ε. Длиной цепочки α (обозначается |α|) называется число составляющих ее символов. 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. называется итерацией цепочки α. Справедливы следующие утверждения: = ε, = α = α для n≥1. . Пусть I – алфавит, тогда– множество всех цепочек, состоящих из символов алфавита I, исключая пустую цепочку (λ). λ ∉ 10). Пусть I – алфавит, тогда I\* – множество всех цепочек, состоящих из символов алфавита I, включая пустую цепочку. λ ∈ I\*, I\* = ∪ λ

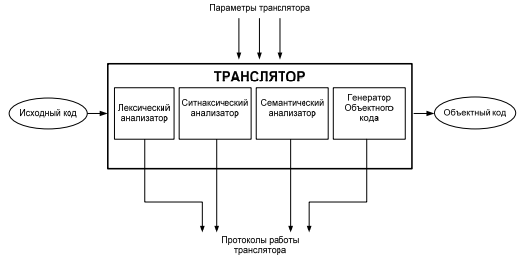
30. **Теория формальных языков: определение формального языка, эквивалентность двух языков**, **способы задания формального языка. Лексика, синтаксис и семантика языка.** **Формальный язык** - это множество конечных слов над конечным алфавитом, например, язык программирования. Языки L1(I) и L2(I) совпадают (эквивалентны), если язык L1(I) включает язык L2(I) и язык L2(I) включает язык L1(I) .  
  
**Язык L можно определить тремя способами**: a) перечислением всех цепочек языка; язык L может быть задан с помощью перечисления цепочек, если количество цепочек конечно; b) указанием способа (алгоритма) порождения цепочек; способ порождения (задания) цепочек языка L называется грамматикой языка L.; c) определением метода (алгоритма) распознавания цепочек. алгоритм (программа), распознающий цепочки языка L, называется распознавателем. Лексика языка программирования – множество цепочек языка. Синтаксис языка – набор формальных правил, определяющий конструкции (последовательности цепочек) языка. Семантика языка – набор неформальных правил, которые описываются словесно (например, в руководстве программиста). Пример: переменную надо объявить до ее применения. Чтобы создать язык программирования, следует определить: − множество допустимых символов (алфавит); − формально описать множество правильных программ; − задать семантические правила языка. Множество допустимых символов может быть проверено. Определить множество формально правильных программ можно с помощью алгоритма-распознавателя. Распознаватель строится на основе формального описания языка – его формальной грамматики. Семантические правила могут быть реализованы в виде эвристических алгоритмов (алгоритм, не имеющий строгого обоснования, но дающий приемлемое решение) или в виде словесного (неформального) описания правил языка.

31. **Теория формальных языков: грамматика (определение, назначение, примеры**), **выводимость цепочки символов в грамматике**, **сентенциальная форма грамматики**, **язык, порождаемый грамматикой,** **способы задания грамматик.**   
G = <T,N,P, S> – грамматика языка (порождающая грамматика) – это четверка, где:   
 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 вида α →β говорят, α порождает β, где α∈(N ∪T , \* β∈(N ∪T) . S – стартовый символ грамматики, S ∈ N .  
Выводом называется процесс порождения предложения языка на основе правил грамматики языка. Записывается α ⇒ β. Цепочка β = δ1γδ2 называется непосредственно выводимой из цепочки α = δ1ωδ2 в грамматике G = <T,N,P, S> , где V = N∪T, δ1, γ, δ2∈V\* , ω∈ , 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> .   
L(G) – язык, порождаемый грамматикой G . Язык L(G) содержит все терминальные цепочки, выводимые из S : L(G) = {α∈T\*| S ⇒\* α}.   
Схема грамматики содержит правила вывода, определяющие структуру цепочек порождаемого языка. Для задания правил используются различные формы описания: − символическая; − форма Бэкуса-Наура; − итерационная форма; − синтаксические диаграммы.

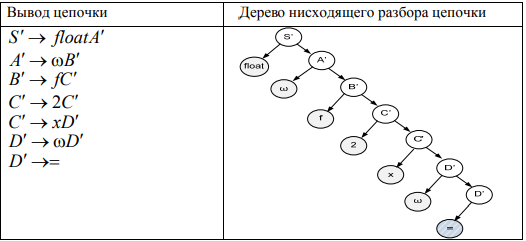
**32. Теория формальных языков: грамматика (определение, назначение, примеры),** **способы задания грамматик**. **Общая характеристика формы Бэкуса-Наура**.   
G = <T,N,P, S> – грамматика языка (порождающая грамматика) – это четверка, где:   
 T – множество терминальных символов (терминалы, алфавит языка) – 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 вида α →β говорят, α порождает β, где α∈(N ∪T , \* β∈(N ∪T) . S – стартовый символ грамматики, S ∈ N . Схема грамматики содержит правила вывода, определяющие структуру цепочек порождаемого языка. Для задания правил 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»). 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.  


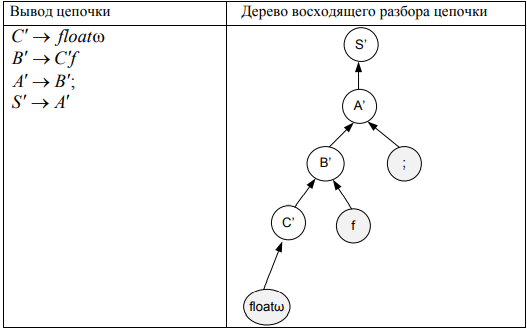
**33. Теория формальных языков: иерархия Хомского (неограниченные, контекстно-зависимые, контекстно-независимые и регулярные грамматики), соотношения грамматик и порождаемых ими языков.**

Хомский Ноам: 1928, США, лингвист, профессор Массачусетского технологического института, автор классификации формальных языков (иерархия Хомского), ввел понятие порождающей грамматики (1950).   
1) Иерархия Хомского: G0 ⊃ GI ⊃ GII ⊃ GIII , где G0, GI, GII, GIII множества грамматик типа 0, 1, 2 и 3. 2) Грамматики типа 0: G0=<T,N,P, S> – неограниченные грамматики, у которых нет никаких ограничений для правил. Правила имеют вид: α →β, где α∈ , β∈V\* . 3) Грамматики типа 1: G1=<T ,N ,P ,S> – контекстно-зависимые (КЗ) грамматики (неукорачивающие грамматики). Правила имеют вид: α →β, где α∈, β∈ и |α| ≤ |β| . 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. 4) Грамматики типа 2: G2=<T, N, P, S> – контекстно-свободные (КС) грамматики. Правила имеют вид: A → α , где A∈ N , α∈V\* , где A — нетерминал, β — цепочка нетерминалов и терминалов. 5) Грамматики типа 3: G3=<T, N, P, S> – регулярные грамматики. Регулярные грамматики бывают праволинейными и леволинейными. Правила праволинейной грамматики имеют вид: A → α или A → αB, где A,B ∈ N , α∈T\* . Правила леволинейной грамматики имеют вид: A → α или A → Bα, где A,B ∈ N ,α∈T\* . 6) Соотношения грамматик в иерархии Хомского: a) любая регулярная грамматика является контекстно-свободной грамматикой; b) любая контекстно-свободная грамматика является контекстнозависимой грамматикой; c) любая контекстно-зависимая грамматика является грамматикой типа 0.  


34. **Лексический анализ: определение, назначение, применение. Лексический анализатор: место в структуре транслятора, входная и выходная информация, способы взаимодействия с синтаксическим анализатором (последовательное и параллельное**). Лексический анализатор (или сканер) – это часть компилятора, которая читает исходную программу и выделяет в ее тексте лексемы входного языка. На вход лексического анализатора поступает текст исходной программы, а выходная информация передается для дальнейшей обработки компилятором на этапе синтаксического анализа и разбора.. Этот набор данных называют также таблицей лексем.Лексический анализ – первая (наиболее простая) фаза трансляции. Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором (сканером).   
  
На вход лексического анализатора поступает текст исходной программы, а выходная информация передаётся для дальнейшей обработки синтаксическому анализатору.

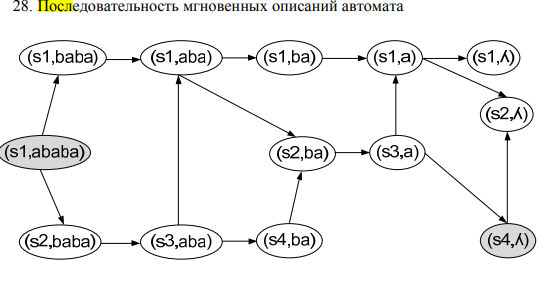
 При последовательном варианте лексический анализатор просматривает весь текст исходной программы от начала до конца и преобразует его в структурированный набор данных. Этот набор данных называют также таблицей лексем.  
При параллельном варианте лексический анализ исходного текста выполняется поэтапно так, что синтаксический анализатор, выполнив разбор очередной конструкции языка, обращается к сканеру за следующей лексемой. При этом он может сообщить информацию о том, какую лексему следует ожидать. В процессе разбора при возникновении ошибки может происходить “откат назад”, чтобы попытаться выполнить анализ текста на другой основе. И только после того, как синтаксический анализатор успешно выполнит разбор очередной конструкции языка (обычно такой конструкцией является оператор исходного языка), лексический анализатор помещает найденные лексемы в таблицу лексем и таблицу идентификаторов и продолжает разбор дальше в том же порядке.

**35. Лексический анализ: определение, назначение, применение**. **Лексический анализатор: нисходящий и восходящий разбор**, **дерево разбора.**   
Лексический анализатор (или сканер) – это часть компилятора, которая читает исходную программу и выделяет в ее тексте лексемы входного языка. На вход лексического анализатора поступает текст исходной программы, а выходная информация передается для дальнейшей обработки компилятором на этапе синтаксического анализа и разбора.. Этот набор данных называют также таблицей лексем.Лексический анализ – первая (наиболее простая) фаза трансляции. Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором (сканером).   
Вывод называется правосторонним(нисходящго), если в нем на каждом шаге вывода правило грамматики применяется всегда к крайнему правому нетерминальному символу в цепочке.  


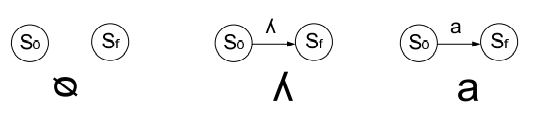
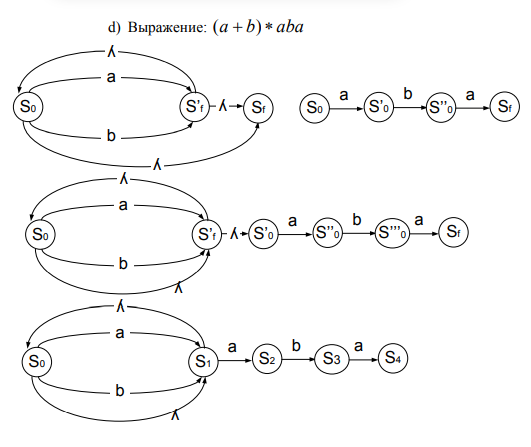
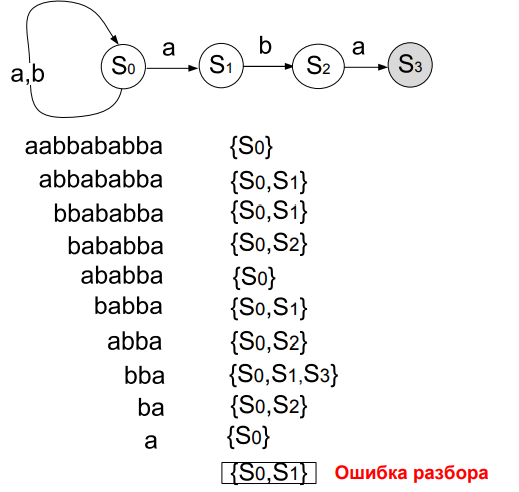
Вывод называется левосторонним, если в нем на каждом шаге вывода правило грамматики применяется всегда к крайнему левому нетерминальному символу в цепочке. Другими словами – на каждом шаге вывода происходит подстановка цепочки символов на основании правила грамматики, т.е. вместо крайнего левого нетерминального символа в исходной цепочке.  
  
Дерево разбора может рассматриваться как графическое представление порождения, из которого удалена информация о порядке замещения нетерминалов.

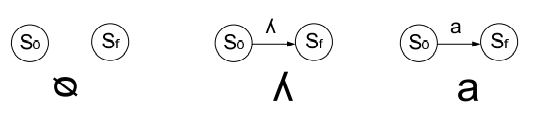
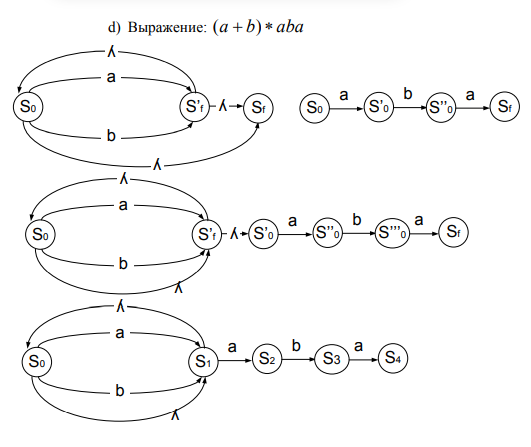
36. **Регулярные выражения: рекурсивное определение, назначение, применение, метасимволы, примеры.** Применение в функциях стандартной библиотеке STL C++.   
Для описания регулярных языков используют другую форму описания – регулярные выражения. Регулярное выражение описывает множество цепочек – формальный язык. Для записи регулярного выражения используются метасимволы. Множество цепочек описанных регулярным выражением называется регулярным множеством (или регулярным языком). Пусть I – алфавит. Регулярные выражения над алфавитом I и языки, представляемые ими, рекурсивно определяются следующим образом: 1) ∅ – регулярное выражение, представляет пустое множество; 2) λ – регулярное выражение, представляет множество {λ}; 3) для каждого a∈ I символ a является регулярным выражением и представляет множество {a}; 4) если p – регулярное выражение, представляющее множество P и q – регулярное выражение, представляющее множество Q , то p+q , pq , q\* являются регулярными выражениями и представляют множества: a) P ∪ Q (объединение), b) PQ (конкатенация множеств), c) P\* (итерация) соответственно. 5) pp\* =   
Символы, применяемые для описания регулярных выражений, называются метасимволами или символами-джокерами. Джокерами являются символы: ,∅ ,+ ,\*(,)  
**Стандартная библиотека шаблонов**— это часть Стандартной библиотеки С++, которая содержит набор шаблонов контейнерных классов, алгоритмов и итераторов. Библиотека STL содержит пять основных видов компонентов: 1.**контейнер (container)**: управляет набором объектов в памяти.2.**итератор (iterator)**: обеспечивает для алгоритма средство доступа к содержимому контейнера.3.**алгоритм (algorithm)**: определяет вычислительную процедуру. 4.**функциональный объект (function object)**: инкапсулирует функцию в объекте для использования другими компонентами.5.**адаптер (adaptor)**: адаптирует компонент для обеспечения различного интерфейса. STL — 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. Основные компоненты библиотеки STL: 1) контейнеры + итераторы; 2) алгоритмы; 3) потоки ввода-вывода. Контейнер – объект, содержащий другие объекты (стек, список, очередь, вектор и пр.), предназначенный для хранения однотипных объектов и обеспечения доступа к ним. Например: класс vector – определяет динамический массив; класс stack – определяет стек; класс list – позволяет работать с линейным списком; класс deque – создает двухстороннюю очередь. Контейнеры: последовательные (последовательный доступ к элементам); ассоциативные (доступ по ключу). Алгоритм определяет вычислительную процедуру (обобщённые алгоритмы) для работы с контейнерами. Алгоритмы позволяют манипулировать содержимым контейнера: инициализировать, сортировать, искать, изменять содержимое контейнера. Итератор: объект, обеспечивающий для алгоритма средство доступа к содержимому контейнера. Итератор позволяет перемещаться по содержимому контейнера подобно тому, как указатель перемещается по элементам массива. Итераторы: произвольного доступа (для ассоциативных контейнеров); двунаправленные (для последовательных).

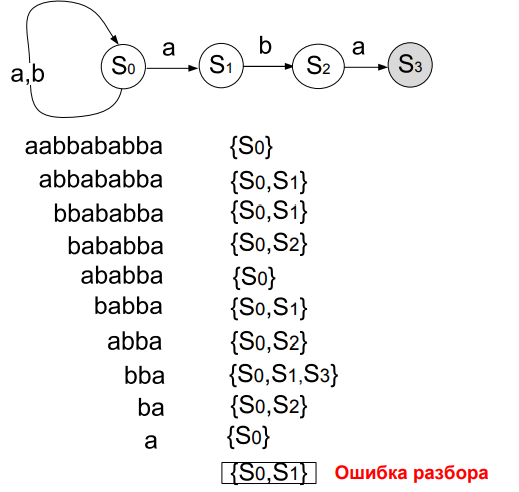
37. **Конечный автомат: определение, назначение, схема работы, примеры. Недетерминированные и детерминированные конечные автоматы.** **Мгновенное описание конечного автомата**. Последовательность мгновенных описаний, диаграмма мгновенных описаний.   
КА это пятерка M = (S,I ,δ ,s0, F), где S – конечное 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. Если δ(s,λ) = ∅ и |δ (s,a) |≤ 1, то конечный автомат – детерминированный (ДКА). Т.е. отсутствуют состояния, имеющие λ-переходы и для каждого состояния s и входного символа а существует не более одной дуги, выходящей из s и помеченной как а. ДКА – это автомат, который переходит из любого состояния по любому символу точно в одно состояние. Иначе - конечный автомат является недетерминированным (НКА) Детерминированным конечным автоматом (ДКА) называется такой автомат, в котором нет дуг с меткой ε (предложение, не содержащее ни одного символа), и из любого состояния по любому символу возможен переход не более, чем в одно состояние.  
Недетерминированный конечный автомат (НКА) является обобщением детерминированного.Недетерминированность автоматов может достигаться двумя способами: либо могут существовать переходы из состояния в состояние, вызываемые пустой цепочкой символов, то есть самопроизвольные переходы без внешних воздействий, либо из одного состояния КА может переходить в разные состояния под воздействием одного и того же символа. Мгновенное описание КА – пара (s,w), где s∈ S – состояние КА, w∈ I\* – неиспользованная часть входной цепочки.

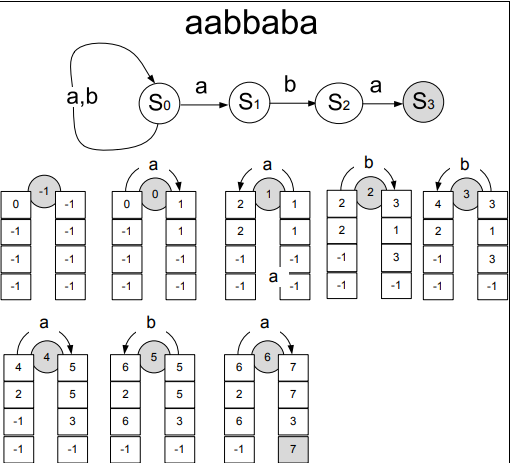


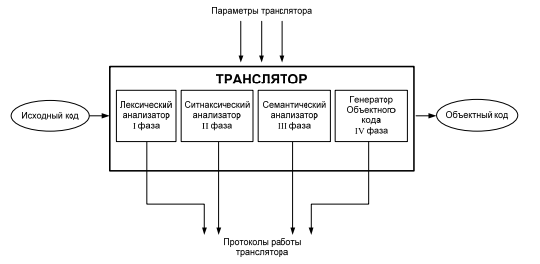
38. **Конечный автомат: определение, назначение, схема работы, примеры**. Соотношение регулярного языка, регулярной грамматики, регулярного языка и конечного автомата.   
КА это пятерка M = (S,I ,δ ,s0, F), где S – конечное множество состояний устройства управления; I – алфавит входных символов; δ – 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.  
Доказаны 4 утверждения: 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. 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.

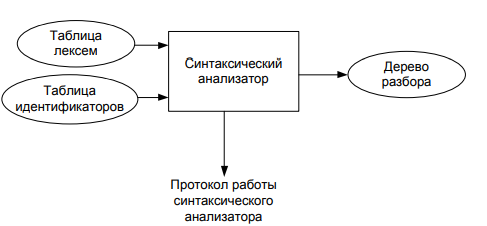
39. **Конечный автомат: построение графа переходов конечного автомата по регулярному выражению.** **Примеры построения графов переходов.** **Алгоритм разбора цепочки символов.** Построение графа конечного автомата по регулярному выражению: Метод построения. Автомат для выражения строится композицией 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.  
Примеры:  
 **Алгоритм разбора цепочки символов.** 

40. Конечный автомат: **построение графа переходов конечного автомата по регулярному выражению. Примеры построения графов переходов. Алгоритм разбора цепочки символов.** **Реализация алгоритма разбора с помощью двух массивов.** 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.  
  
Пример: 

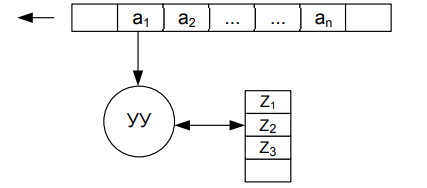
**Алгоритм разбора цепочки символов:**  
**Реализация алгоритма разбора с помощью двух массивов.**

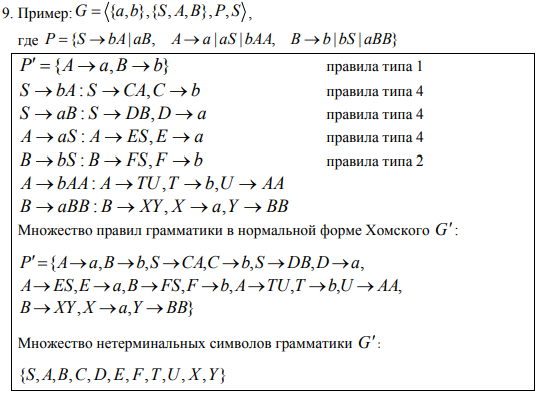


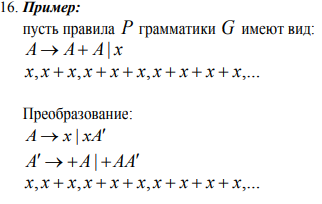
**41. Синтаксический анализ: определение, назначение, применение. Синтаксический анализатор: определение, входные данные, выполняемые задачи, результат работы. Соотношение между лексическим и синтаксическим анализом.**   
Синтаксический анализ: вторая фаза трансляции:  


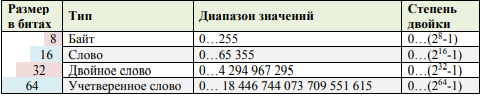
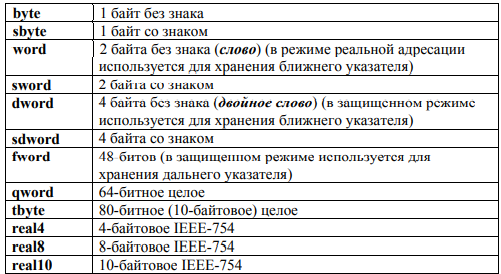
Синтаксический анализ выполняется после фазы лексического анализа и предназначен для распознавания синтаксических конструкций и формирования промежуточного кода. Синтаксический анализ: основная фаза трансляции. Без нее процесс трансляции не имеет смысла. Все задачи лексического анализа могут быть решены в рамках синтаксического анализа. Т.е. можно создать транслятор без лексического анализатора. Лексический анализ необходим для освобождения алгоритма синтаксического разбора от рутинных алгоритмов. Программа, выполняющая синтаксический анализ называется синтаксическим анализатором. На вход синтаксического анализатора поступают таблица лексем (ТЛ) и таблица идентификаторов (ТИ). Выходом является дерево разбора. Входная и выходная информация синтаксического анализатора:    
Нет четкой границы между лексическим и синтаксическим анализатором. Алгоритм разбора распределяется между лексическим и синтаксическим анализатором. Конструкции языка, которые разбираются лексическим или синтаксическим анализатором, определяет разработчик транслятора. **Задачи, выполняемые синтаксическим анализатором**: 1) поиск и выделение синтаксических конструкций в исходном тексте (разбор); 2) распознавание (проверка правильности) синтаксических конструкций; 3) выявление ошибок и продолжение процесса распознавания после обработки ошибок; 4) формирование дерева разбора в случае, если нет ошибок. Исходный текст программы для синтаксического анализатора представляется в виде таблицы лексем

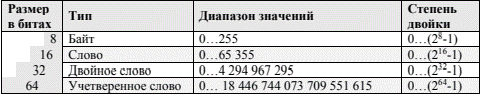
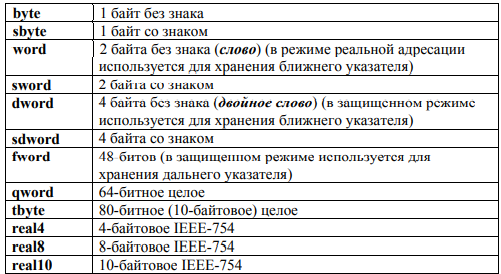
42. **Приведение контекстно-свободной грамматики: удаление бесплодных символов, недостижимых символов**, λ-правил, цепных правил.   
**Алгоритм удаления бесплодных символов** Рекурсивно строим множества N0, N1, N2,… 1) N0 = ∅ 2) N1 ={A| (A → α)∈ P ∧ α∈(N0 ∪T)\* }∪N0 3) если N1 ≠ N0, то переход на шаг 4 иначе G′ = (T,N1, P′, S) , где P′ ⎯ правила из P , содержащие только символы V ′ = N1 ∪T 4) N2 ={A| (A → α)∈ P ∧ α∈(N1 ∪T)\* }∪N1 5) если N2 ≠ N1, то переход на шаг 6 иначе G′ = (T, N2, P′, S) , где P′ ⎯ правила из P , содержащие только символы V ′ = N2 ∪T 6) N3 ={A| (A → α)∈ P ∧ α∈(N2 ∪T)\*}∪N2 7) если N3 ≠ N2 , то переход на шаг 8 иначе G′ = (T, N3, P′, S) , где P′ ⎯ правила из P , содержащие только символы V ′ = N3 ∪T 8) ...  
В общем виде алгоритм удаления бесплодных символов можно записать следующим образом: 1) N0 = ∅ 2) i=1,Ni ={ A| (A → α) ∈ P ∧ α∈(Ni-1∪T)\* ∪ Ni−1 3) если Ni ≠ Ni−1, то i= i+1 b переход на шаг 2 иначе G’=(T,Ni ,P’ ,S) , где P′ ⎯ правила из P , содержащие только символы V ′ = Ni ∪T.  
Определение недостижимого символа. Символ X ∈(N ∪T) в грамматике G = <T,N,P,S> называется недостижимым, если он не встречается ни в одной сентенциальной форме грамматики. Другими словами: недостижимым символом называется символ, который не может быть выведен из стартового символа грамматики S . Очевидно, что недостижимый символ грамматике не нужен. если S⇒ \*β и β ∈(T ∪ N)\* , то β называется сентенциальной формой грамматикиG = <T,N,P,S >. А**лгоритм удаления недостижимых символов:**Перед удалением недостижимых символов, должны быть удалены бесплодные символы Строим множества V0, V1, V2,… : Описание алгоритма. Строим множество достижимых символов. Первоначально в это множество входит только стартовый символ S грамматики, затем множество пополняем на основе грамматики. Все символы, которые не войдут в это множество, являются недостижимыми и могут быть исключены в новой грамматике из словаря и из правил. 1)V0 = {S0} , i =1 2) Vi= { x|x (N ∪T) u(A→αxβ) ∈P,A ∈ Vi-1} ∪Vi-1, где α,β∈(N ∪T)\* 3) если Vi ≠ Vi−1, то i=i+1 и переход на шаг 2, иначе G′ = (T′ ,N′ ,P′ ,S), где N = N ∩Vi ,T’ = T ∩Vi P′ ⎯ правила из P , содержащие только символы Vi . Алгоритмы удаления бесплодных и недостижимых символов упрощают грамматики, сокращают количество символов алфавита и правил грамматики. Правило вида A → λ называется λ-правилом или аннулирующим правилом. Теорема: для грамматики G, содержащей λ-правила, всегда можно найти эквивалентную грамматику G′ не содержащую λ-правил. **Алгоритм исключения λ-правил.** Выполнить все возможные подстановки пустой цепочки вместо аннулирующего нетерминала во всех правилах грамматики. Правило A → B, где A,B∈ N называется цепным. Теорема: для грамматики G, содержащей цепные правила, всегда можно найти эквивалентную грамматику G′, не содержащую цепных правил. Правила вида A → B, B →C , C → aX могут быть заменены одним правилом A → aX , т.к. вывод A⇒ B ⇒ C ⇒ aX цепочки aX в грамматике G может быть заменен выводом A⇒ aX с помощью правила A → aX. **Алгоритм исключения цепных правил:** P ⎯ множество правил грамматики G. Описание алгоритма. Разобьем множество P на два подмножества P1 и P2 по следующему принципу: P1 содержит правила вида Ai → Bi , и P2 ⎯ все остальные. На множестве P1 построим множество правил P(Ai) , для которых Bi →α и α∈(N ∪T)\* правила из P2. Замещаем цепные правила Ai → Bi в P(Ai) на правила Ai →α. Тогда правила грамматики P′ = P(A1) ∪ P(A2) ∪...∪ P2 являются правилами грамматики G′ , не содержащие цепных правил.

**43. Автомат с магазинной памятью: определение, схема работы, пример, конфигурация (текущее состояние автомата).**Автоматы с магазинной памятью (МП-автоматы) – распознаватели контекстно-свободных языков, которые можно представить в виде следующей схемы:  
   
Формальное описание МП-автомата: M = <Q,V,Z,δ,q0 ,z0 ,F>: Q ⎯ множество состояний управляющего устройства; V ⎯ алфавит входных символов; Z ⎯ специальный алфавит магазинных символов; δ ⎯ функция переходов автомата Q× (V ∪ {λ}) × Z → P(Q× Z\*) , где P(Q× Z)\* - множество подмножеств Q× Z\*; q0 ∈Q ⎯ начальное состояние автомата; z0 ∈Z ⎯ начальное состояние магазина (маркер дна); F ⊆ Q ⎯ множество конечных состояний. Функция переходов δ отображает тройки (q, a,z) в пары (q′,γ ) для детерминированного автомата или во множество таких пар для недерминорованного автомата, где , q′∈Q\*, γ - символ в вершине магазина. Эта функция описывает состояние магазинного автомата, при чтении символа с входной ленты и перемещении головки. **Конфигурация МП-автомата:** Конфигурация автомата (текущее состояние) описывается тройкой: (q,α,ω), где q- текущее состояние автомата; α - остаток цепочки. Первый символ этой цепочки просматривается входной головкой автомата. Если α= {λ}, то входной символ прочитан; ω - цепочка-содержимое магазина (стека). Если ω= {λ}, то магазин пустой**. Один такт работы автомата:** (q,aα,zω) ϒ(перевернутое) (q ′ ,α, γω) (переходит в конфигурацию»), если (q ′ , γ)∈δ(q,a,z). При выполнении такта из магазина (стека) удаляется верхний символ, соответствующий условию перехода, и добавляется цепочка, соответствующая правилу перехода. Первый символ цепочки становится вершиной стека. Допускаются переходы, при которых входной символ игнорируется. Эти переходы называются λ-переходами. Начальное состояние МП-автомата (q0,α ,z0) , где q0 ⎯ начальное состояние автомата, α ⎯ входная цепочка, z0 ⎯ маркер дна магазина. Цепочка α является допустимой автоматом, если из начальной конфигурации за конечное число тактов работы автомат перейдет в заключительное состояние: M =<Q,V,Z,δ,q0 ,z0 ,F> , если (q0, α, z0) ϒ(перевернутое) \*(q′, λ, λ) и q ′∈ F . Работа автомата M = <Q,V,Z,δ,q0 ,z0 ,F> 1) состояние автомата (q,aα,zβ) 2) читает символ a находящийся под головкой (сдвигает ленту); 3) не читает ничего (читает λ , не сдвигает ленту); 4) из δ определяет новое состояние q′ , если (q ′ , γ)∈δ(q,a,z) или (q ′ , γ)∈δ(q,λ,z). 5) читает верхний (в стеке) символ z и записывает цепочку γ т.к. (q ′ , γ)∈δ(q,a,z), при этом, если γ = λ, то верхний символ магазина просто удаляется. 6) работа автомата заканчивается (q,λ,λ). На каждом шаге автомата возможны три случая: 1) функция δ(q,a,z) определена ⎯ осуществляется переход в новое состояние; 2) функция δ(q,a,z) не определена, но определена δ(q,λ,z) ⎯ осуществляется переход в новое состояние (лента не продвигается); 3) функции δ(q,a,z) и δ(q,λ,z) не определены ⎯ дальнейшая работа автомата не возможна (цепочка не разобрана). Язык L(M) ={α|q0, α, z0) ϒ(перевернутое) \*(q′, λ, λ) и q ′∈ F ⎯ язык, допускаемый автоматом M .

44**. Нормальная форма Хомского контекстно-свободной грамматики. Алгоритм приведения контекстно-свободной грамматики к нормальной форме Хомского.** **Примеры.**   
Для построения МП-автомата необходимо привести контекстно-свободную грамматику к одной из нормальных форм: нормальной форме Хомского нормальной форме Грейбах. Нормальная форма Хомского: КС-грамматика G = T,N,P,S имеет нормальную форму Хомского, если правила P имеют вид: 1) A → BC, где A,B,C ∈ N ; 2) A → a , где A∈ N ,a∈T ; 3) S → λ, где S ∈ N — начальный символ, и если такое правило существует, то нетерминал S не должен встречаться в правой части правил. Грамматика в нормальной форме Хомского называется бинарной, т.к. один нетерминальный символ может быть заменен на два нетерминальных символа. В дереве вывода грамматики в нормальной форме Хомского каждая вершина распадается: − на две другие вершины (в соответствии с первым видом правил A → BC), − либо содержит один последующий лист с терминальным символом (в соответствии со вторым видом правилом вывода A → a ). Третий вид правил введен для того, чтобы к нормальной форме Хомского можно было преобразовывать грамматики КС-языков, содержащих пустые цепочки символов. Алгоритм преобразования контекстно-свободной грамматики G = <T,N,P,S> к грамматике G′ = <T,N′ ,P′ ,S’> в нормальной форме Хомского: I. преобразовать исходную грамматику к приведенному виду (исключить бесплодные и недостижимые символы, цепные и λ -правила); II. установить N′ = N III. построение P′. Правила вида: 1) A → a , где A∈ N ,a∈T переносятся P′ без изменений; 2) A → BC, где A,B,C ∈ N переносятся в P′ без изменений; 3) S → λ, где S ∈ N переносятся в P′ без изменений; 4) A → aB, где A,B∈ N , a∈T преобразуются в правила вида A → DB и D → a добавляются во множество правил P′, нетерминальный символ D добавляется во множество нетерминалов N′ грамматики G′; 5) правила вида A → Ba , где A,B∈ N , a∈T преобразуются в правила вида A → BD и D → a грамматики P′, нетерминальный символ D добавляется в N′ ; 6) правила вида A → ab, где A∈ N , a,b∈T преобразуются в правила вида A → BD,B → a , D → b грамматики P′ , нетерминальные символы B, D добавляется в N′ ; 7) правила вида A → X1,. X2,…,Xk где k > 2 и Xi ∈ N ∪T преобразуются в правила вида A → Y1Y2 , Y2 →Y3Y4 , Y4 →Y5Y6 , …, Y1 → X1, Y3 → X2 , Y5 → X3… Правила вида Yi → X j могут потребовать дальнейшего преобразования. Если достигнут вид правил, который указан в определении, то они добавляются в P′ , а новые нетерминальные символы в N′. Стартовым символом результирующей грамматики G′ является стартовый символ исходной грамматики G.  


**45. Нормальная форма Грейбах контекстно-свободной грамматики. Алгоритм приведения контекстно-свободной грамматики к нормальной форме Грейбах. Примеры.** Для построения МП-автомата необходимо привести контекстно-свободную грамматику к одной из нормальных форм: нормальной форме Хомского нормальной форме Грейбах. **Нормальная форма Грейбах:** контекстно-свободная грамматика G = <T,N,P,S> имеет нормальную форму Грейбах, если она не леворекурсивная (не содержит леворекурсивных правил), а правила P имеют вид: 1) A → aα, где a∈T,α∈(N ∪T)\* ; 2) S → λ, где S ∈ N — начальный символ, и если такое правило существует, то нетерминал S не должен встречаться в правой части правил. Эта нормальная форма называется по имени Шейлы Грейбах (Sheila Greibach), которая первой описала способ построения таких грамматик. Алгоритм устранения левой рекурсии. Пусть правила грамматики G = T,N,P,S имеют вид: A → Aα1 |Aα2| …|Aαm|β1| |β2|…|βn, где αi, βi ∈ (T∪ N)\* и цепочки βi не начинаются с нетерминала A. Введем нетерминал B. Тогда эквивалентные правила без левой рекурсии: A →|β1| β2|… |βn|β1B| β2B|…|βnB|; B →|α1|α2|…|αm|α1B|α2B|…| αmB   


**46. Язык ассемблера: структура программы, типы данных, директивы для определения данных, регистры общего назначения центрального процессора: назначение, названия. Язык ассемблера** — это машинно-ориентированный язык программирования. **Программа состоит из одного или нескольких сегментов:** сегмент кода; область памяти, в которой размещаются выполняемые команды программы; сегмент данных; область памяти с данными; сегмент стека – область памяти, отведенная под стек. **Регистры** – участки высокоскоростной памяти центрального процессора, предназначенные для оперативного хранения данных и быстрого доступа к ним. **Программа на ассемблере** – это модуль, включающий одну главную, или основную, процедуру, с которой начинается выполнение. .586 – эта директива выбирает поддерживаемый набор команд ассемблера, указывая модель процессора. Модуль может содержать сегмент кода, сегменты данных и стека. Нужно указать модель памяти при помощи директивы .MODEL. Плоская модель памяти flat (flat memory model). Эта модель памяти используется в операционной системе Windows. Адресация любой ячейки памяти будет определяться содержимым одного 32-битного регистра. Определяет приложение, выполняющееся, в защищенном режиме с использованием линейной модели памяти. stdcall – используемое соглашение о вызовах процедур. Подключение необходимых библиотек. Объявление прототипа функции с использованием директивы PROTO (после символа «:» указывается тип параметра, параметры разделяются символом «,»). Параметры WinAPI-функций 32-битные ( целые числа). Все WinAPI-функции созданы по соглашению stdcall. **Типы данных.** В семействе процессоров IA-32 аппаратно поддерживаются процессором размеры для хранения типов данных:  
  
Внутренние типы данных В MASM определены несколько внутренних типов данных, значения которых могут быть присвоены переменным, либо они могут являться результатом выполнения выражения:  
  
В языке ассемблера существует 5 (пять) директив для определения данных: − DB (define byte) – определяет переменную размером в 1 байт; − DW (define word) – определяет переменную размеров в 2 байта (слово); − DD (define double word) – определяет переменную размером в 4 байта (двойное слово); − DQ (define quad word) – определяет переменную размером в 8 байт (учетверённое слово); − DT (define ten bytes) – определяет переменную размером в 10 байт. Регистры общего назначения. Названия регистров происходят от их назначения: EAX/AX/AH/AL (аккумулятор) – применяется для хранения промежуточных данных, автоматически применяется при операциях умножения, деления для хранения первого операнда; • EBX/BX/BH/BL (база) – регистр базы, применяется для хранения базового адреса некоторого объекта в памяти (например, массива); ECX/CX/CH/CL (регистр-счетчик) – автоматически применяется в качестве счетчика цикла, его использование может быть неявно и скрыто в алгоритме работы соответствующей команды; EDX/DX/DH/DL – регистр данных, применяется в операциях умножения и деления, используется как расширение регистра аккумулятора EAX при работе с 32- разрядными числами; ESI/SI – индекс источника; EDI/DI – индекс приёмника (получателя); ESP/SP – регистр указателя стека; EBP/BP – регистр указателя базы. Подрегистры AX, BX, CX, DX позволяют независимо обращаться к их старшей (H) и младшей (L) половине. Подрегистры старшей (H) и младшей (L) половины имеют размерность 8 бит и названия AH, AL, BH, BL, CH, CL, DH, DL сответственно.

47. **Язык ассемблера: типы данных, инициализация данных, синтаксис оператора определения данных**, **массивы и их инициализация. Примеры.**   
**Типы данных.** В семействе процессоров IA-32 аппаратно поддерживаются процессором размеры для хранения типов данных:  
  
Внутренние типы данных В MASM определены несколько внутренних типов данных, значения которых могут быть присвоены переменным, либо они могут являться результатом выполнения выражения:  
  
Синтаксис оператора определения данных: [имя] директива инициализатор [инициализатор]; Имя переменной (идентификатор) – метка, значение которой соответствует смещению данной переменной относительно начала сегмента, в котором она размещена. С помощью директив в программе выделяется память под одну или несколько знаковых или беззнаковых переменных соответствующей длинны: BYTE определяет беззнаковый байт. SBYTE определяет знаковый байт. WORD определяет слово без знака для хранения 16-разрядных целых значений. SWORD определяет слово со знаком для хранения 16-разрядных целых значений. DWORD определяет слово без знака для хранения 32-разрядных целых значений. SDWORD определяет слово со знаком для хранения 32-разрядных целых значений. TBYTE определяет 10 байтов для хранения 80-разрядных целых значений. Этот тип данных в основном используется для хранения десятичных упакованных целых чисел. Для работы с этими числами используется специальный набор команд математического сопроцессора. Если инициализатор равен ?, то выделяется память для хранения не инициализированных переменных, заданной длины. Для создания массива можно либо явно перечислить значения каждого элемента массива через запятую, либо воспользоваться оператором DUP. a. Множественна инициализация. Если в операторе определения данных используется несколько инициализаторов, то присвоенная оператору метка относится только к первому элементу данных и этот элемент располагается со смещением 0 относительно этой метки. Следующий элемент располагается со смещением равным размеру элемента в байтах относительно метки оператора. Метки нужны не для всех операторов определения данных. В одном операторе определения данных могут использоваться инициализаторы, заданные в разных системах счисления. Кроме того, могут использоваться вперемешку как символы, так и строковые константы. b.**Оператор DUP** используется для выделения памяти под массив, содержащий повторяющиеся значения байтов, которые могут быть инициализированы или нет. В качестве счетчика байтов используется константное выражение.

**48. Язык ассемблера: формат команды ассемблера,** **команды пересылки данных, команды целочисленного сложения и вычитания,** **операторы OFFSET (понятие**), **PTR,** **TYPE,** **SIZEOF и LENGTHOF. Примеры.** **Команда** – оператор программы, который непосредственно выполняется процессором. **Команды языка ассемблера** – это символьная форма записи машинных команд. Команды имеют следующий синтаксис: **[метка] (необязательный) мнемоника [операнд(ы)] [;комментарий]**; **Метка** – идентификатор, с помощью которого, можно пометить участок кода или данных. Метка кода должна отделяться двоеточием. **Мнемоника команды** – короткое имя, определяющее тип выполняемой процессором операции. Операнд определяет данные (регистр, ссылка на участок памяти, константное выражение), над которыми выполняется действие по команде, если операндов несколько, то они отделяются друг от друга запятыми. Основные типы операндов:1)непосредственно значение; 2) регистр;3)память**. Команды пересылки данных**:1) Команда MOV копирует данные из операнда-источника в операнд-получатель: **MOV получатель источник**; Оба операнда должны быть одного типа и иметь одинаковую длину. Оба операнда не могут одновременно быть памятью. В качестве получателя нельзя использовать регистры CS, EIP и IP MOV - мнемоника команды Операнды: «регистр–регистр»; «регистр–память»; «память–регистр»; «регистр–число»; «память– число».2) Команда расширения целых беззнаковых чисел MOVZX (move with zero-extend) копирует содержимое исходного операнда в больший по размеру регистр получателя данных (используется для беззнаковых чисел). Оставшиеся неопределенными биты регистра-получателя сбрасываются в ноль: слово –> 32-разрядный регистр (старшие 16 бита сбрасываются в ноль); байт –> 32-разрядный регистр (старшие 24 бита сбрасываются в ноль);байт –> 16-разрядный регистр (старшие 8 бита сбрасываются в ноль). 3)Команда расширения целых беззнаковых чисел MOVSX (move with sing-extend – переместить и дополнить знаком) копирует содержимое исходного операнда в больший по размеру регистр получателя данных. При этом оставшиеся неопределенными биты регистра-получателя дополняет значением знакового бита исходного операнда. 4) Команда XCHG (exchange data – обмен данными) обменивает содержимое двух операндов, длины операндов должны совпадать. Допустимые операнды: «регистр-регистр»; «регистр-память»; «память-регистр». **Целочисленное сложение и вычитание** :1. Команды INC и DEC прибавляют и вычитают единицу из указанного операнда.2. Команда ADD прибавляет значение операнда-источника к значению операнда получателя. Команда ADD изменяет значения флагов переноса, переполнения, знака и др.3. Команда SUB вычитает операнд-источник из операнда получателя данных. Процессор заменяет ее на команду сложения с отрицательным числом (отрицательные числа представляются в доп. коде). Длины операндов должны быть равны. .4. Команда NEG изменяет знак числа на противоположный. Опреатор OFFSET – возвращает смещение метки данных относительно начала сегмента. Под смещением понимается то количество байтов, которое отделяет метку данных относительно начала сегмента. . Оператор TYPE возвращает размер указанной переменной в байтах. Оператор PTR: Если размеры операндов не совпадают, то можно переопределить размер операнда.Для такого уточнения можно использовать оператор переопределения типа PTR. Типы могут быть byte, word, dword. Операторы SIZEOF и LENGTHOF Опреатор LENGTHOF определяет количество элементов в массиве, перечисленного в одной строке. Опреатор SIZEOF возвращает значение равное произведению значений, возвращаемых операторами LENGTHOF и TYPE.

**49. Язык ассемблера:** а**дресация – прямая и косвенная. Косвенная адресация: с косвенным операндом. Косвенная адресация: операнды с индексом. Примеры косвенной адресации.** Чаще всего используются регистры: ESI (индекс источника) EDI (индекс получателя). **Прямая адресация**: В ассемблере прямая адресация возможна в том случае, если переменной присвоена метка. Пример прямой адресации: MAS DB ‘HELLO’ MOV AL, MAS ;содержимое байта с именем MAS загружается в AL -> AL=’H’. Имя переменной (метка MAS) – значение, соответствующее смещению данной переменной относительно начала сегмента, в котором она размещена. Прямую адресацию неудобно применять при обработке массивов, т.к. каждому элементу массива невозможно присвоить собственную метку. **Косвенная адресация:** Адресуемая память: необходимо заранее загрузить относительный адрес с помощью оператора offset (смещение) обрабатываемой области памяти в РОН. При косвенной адресации в качестве указателя на текущий элемент массива используется один из 32-разрядных регистров общего назначения (РОН): EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP Синаксис: [] Для перехода с следующему элементу массива достаточно увеличить значение указателя на длину элемента массива. Такая адресация называется ковенной, а регистр, в котором хранится адрес эдемента массива, называется косвенным операндом (indirect operand). **Косвенная адресация**. Операнды с индексом. Синтаксис первой формы представления: имя\_переменной[индексный\_регистр]4. Косвенная адресация. Операнды с индексом. Синтаксис второй формы представления: [имя\_переменной +индексный\_регистр]

**50. Язык ассемблера: операции работы со стеком, команды переходов (безусловные, условные), команда LOOP. Назначение, использование. Примеры. Команды переходов** После загрузки программы в память процессор начинает автоматически выполнять последовательность ее команд. При этом счетчик команд (EIP) автоматически изменяется на длину выполненной команды и всегда указывает на адрес следующей команды. Изменить порядок следования команд можно с помощью команд передачи управления. Команда **JMP** – команда безусловной передачи управления на другой участок кода программы по метке. Синтаксис: JMP метка\_перехода; Kоманда LOOP выполняет блок команд заданное число раз. В качестве счетчика используется регистр ECX. Предварительно в регистр ECX загружается количество повторений цикла. Выполнение: На каждом шаге выполнения цикла значение ECX автоматически уменьшается на 1 и сравнивается с 0.Если результат не ноль – переход по метке. В противном случае выпоняется следующая по порядку команда. Синтаксис: LOOP метка\_перехода <метка\_перехода>: … ; loop <метка\_перехода >;Операции со стеком: PUSH, POP, PUSHAD, POPAD, CALL, RET, регистр ESP В регистре ESP хранится 32-разрядное смещение вершины стека. Содержимое ESP изменяется автоматически следующими командами: CALL, RET, PUSH и POP. Команды работы со стеком: PUSH – помещает 32-разрядное число в стек и вычитает 4 байта из значения, хранящегося в ESP. POP – извлекает 32-разрядное число из стека и прибавляет 4 байта к значению, хранящемуся в ESP. Сохранить несколько используемых в процедуре регистров можно опреатором USES. Это необходимо, чтобы процедура не «испортила» их значение. По команде USES сохраняются перечисленные регистры при входе в процедуру и они восстанавливаются непосредственно перед выходом из процедуры. Команды PUSHAD и POPAD – сохраняют 32-разрядные значения всех регистров и восстанавливают их соответственно. Команды **call** и **ret** используются в паре. Команда **call** помещает регистр EIP (указатель на следующюю команду, которая должна быть выполнена) в стек, а команда **ret** извлекает его и передаёт управление этому адресу. 