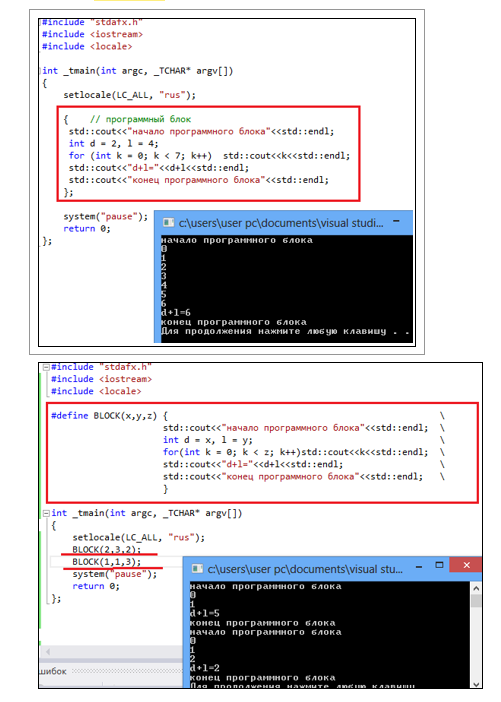
**31. Структура языка программирования: программные конструкции (блоки, функции,  
процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций  
в C++: соглашения о вызовах (\_fastcall).**

Программные блоки представлены:

****

Функции:

***Функция*** – фрагмент программного кода, к которому можно обратиться из другого места программы. Функция должна быть ***объявлена*** и ***определена***.

***Функция*** – подпрограмма, выполняющая какие-либо операции и возвращающая значение.

***Процедура*** – подпрограмма, которая выполняет операции, и не возвращает значения.

**Передача параметров в функцию**

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 вызовах \_fastcall (не стандартизированный, для внутренних вызовов), параметры передаются через регистры (первые 2 – через регистры, остальные справа налево в стек), стек освобождает вызываемый код, возврат через регистр EAX. В Borland (Delphi) (параметры слева на право.

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

В составе языка программирования, как правило, есть обязательный (стандартный) набор функций. Такие функции называют встроенными.

Встраиваться функции могут тремя способами:

* + прямо в код транслятора;
  + находиться в отдельной библиотеке;
  + сочетание первого и второго случаев.

Основные требования к набору средств стандартной библиотеки (Бьёрн Страуструп):

* + ***эффективность***;
  + ***независимость от алгоритмов*** — должна предоставлять возможность задавать алгоритмы в качестве параметров;
  + ***удобство и безопасность***;
  + ***завершённость*;**
  + ***органично сочетаться с языком*;**
  + ***типобезопасность*;**
  + ***поддержка общепринятых стилей программирования***;
  + ***расширяемость*** — способность единообразно работать со встроенными типами данных и с типами, определяемыми пользователем

Подходы к разработке стандартных библиотек языков программирования:

* + должна содержать в себе только те процедуры и функции, которые используются практически всеми и обладают максимальной универсальностью;
  + должна содержать в себе максимально возможное количество типичных алгоритмов, обеспечивать простую работу с большинством объектов (в идеале, со всеми), с которыми может взаимодействовать программа. Пример реализации этого подхода является язык Python с девизом «Batteries included» (батарейки в комплекте).

Стандартная библиотека C++ STL:

библиотека стандартных шаблонов (STL) – набор согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++.

STL (Standard Template Library) – стандартная библиотека шаблонов. Библиотека содержит универсальные шаблонные классы и функции, реализующие большое количество распространенных универсальных алгоритмов и структур данных. Т.к. библиотека STL состоит из шаблонных классов, ее алгоритмы и структуры можно применять практически к любым типам данных.

Частью *стандартной библиотеки С++* является библиотека *STL*. Библиотека STL содержит пять основных видов компонентов:

* *контейнер* (*container*): управляет набором объектов в памяти.
* *итератор* (*iterator*): обеспечивает для алгоритма средство доступа к содержимому контейнера.
* *алгоритм* (*algorithm*): определяет вычислительную процедуру.
* *функциональный объект* (*function object*): инкапсулирует функцию в объекте для использования другими компонентами.
* *адаптер* (*adaptor*): адаптирует компонент для обеспечения различного интерфейса.

Функции для работы со строками:

strlen() (от слова length – длина)

strcpy()

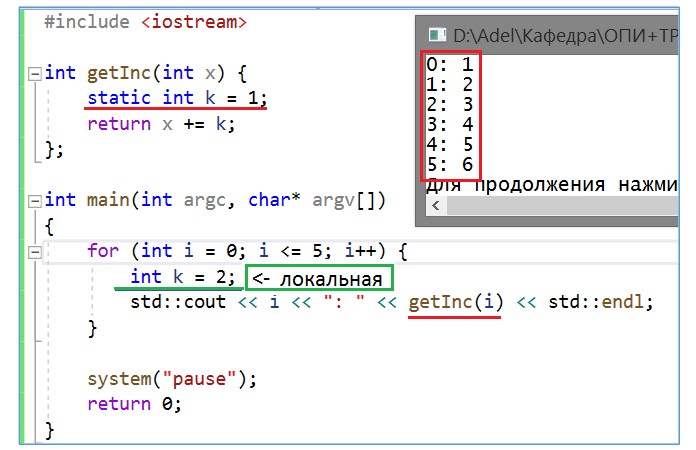
strcmp()

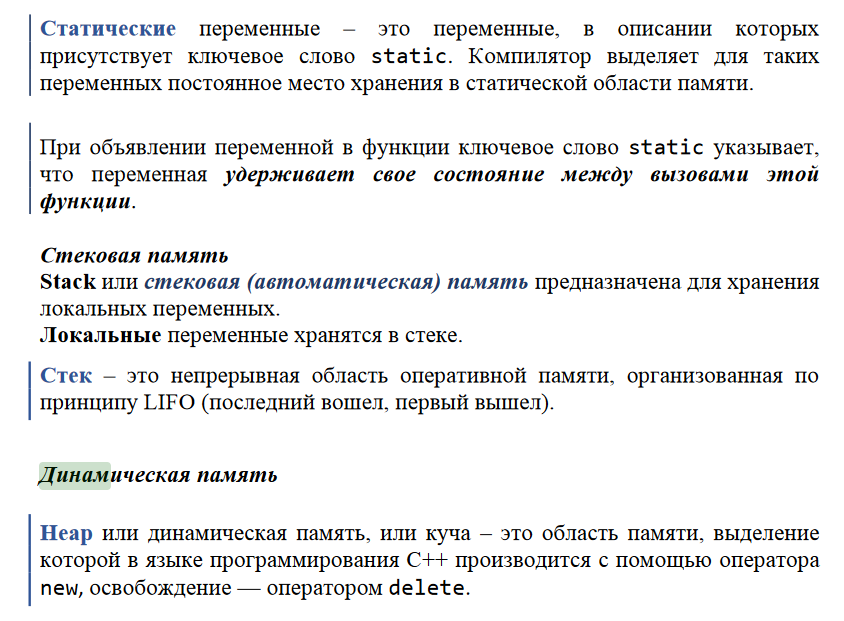
.find()

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

Область кода – память, в которой размещается код программы.

Локальная статическая память.

  
  
Переменная с ключевым словом static — это статическая переменная.  
Время ее жизни — постоянное.  
Область видимости статической переменной ограничена одним файлом,  
внутри которого она определена, ее можно использовать только после ее  
объявления.  
Ключевое слово static в языке С/С++ используется для двух различных  
целей:  
− как указание типа памяти: переменная располагается в статической  
памяти;  
− как способ ограничить область видимости переменной рамками одного  
файла (в случае описания переменной вне функции).

****

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

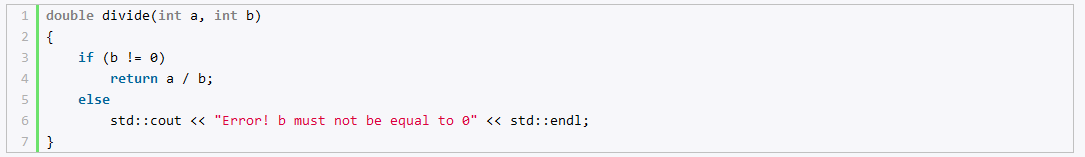
В процессе работы программы могут возникать различные ошибки. Например, при передаче файла по сети оборвется сетевое подключение или будут введены некорректные и недопустимые данные, которые вызовут падение программы. Такие ошибки еще называются исключениями. Если исключение не обработано, то при его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

Например, в следующей программе происходит деление чисел:



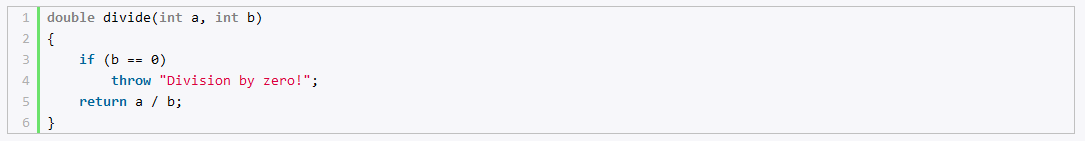
Эта программа успешно скомпилируется, но при ее выполнении возникнет ошибка, поскольку в коде производится деление на ноль, после чего программа аварийно завершится.

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



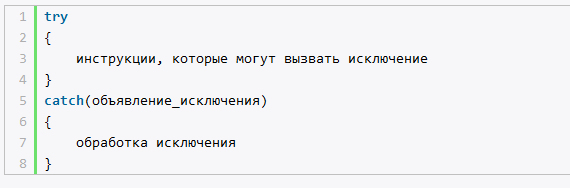
И в этом случае нам надо известить систему о возникшей ошибке. Для этого используется оператор throw.

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

****

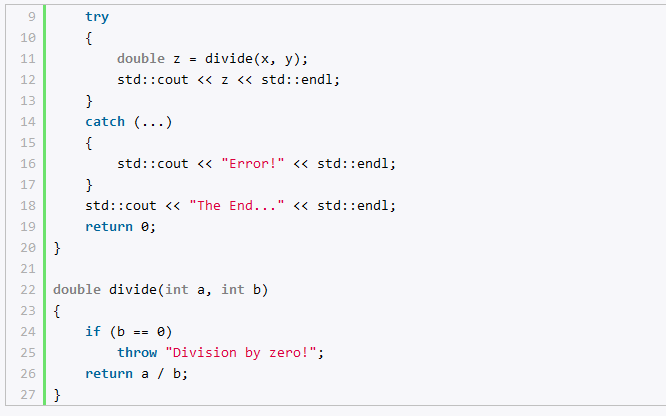
То есть если параметр b равен 0, то генерируем исключение.

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

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

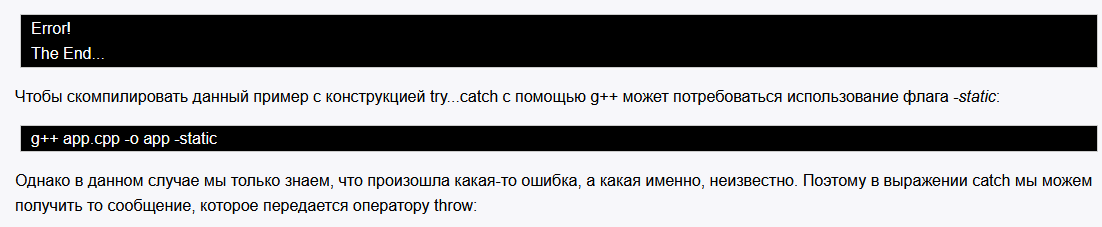
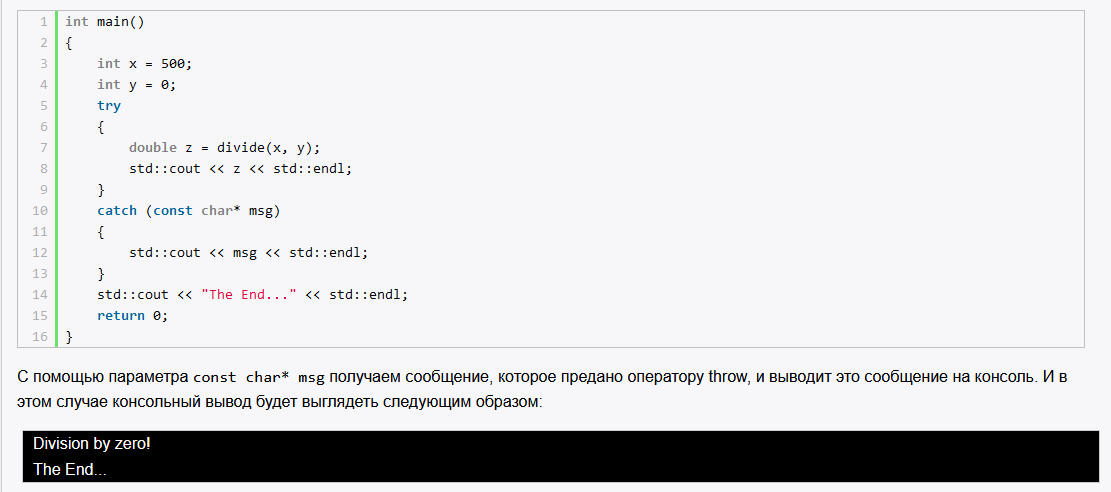
После ключевого слова catch в скобках идет параметр, который передает информацию об исключении. Затем в блоке производится собственно обработка исключения.

Так изменим весь код следующим образом:

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

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

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

**** ****

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

Препроцессор С++:  
директивы: #include, #define, #if, #else, #elif, #endif, #ifdef, #ifndef, #error, #line,  
#pragma, #undef; операторы: defined, #, ##.  
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 – программа для обработки текста.  
Препроцессор может быть отдельной программой, или же интегрирован в  
компилятор.  
Входные и выходные данные для препроцессора имеют текстовый формат.  
Препроцессор преобразует текст в соответствии с директивами препроцессора.  
В случае если текст не содержит директив препроцессора, то он остаётся без  
изменений.

Препроцессор можно вызвать отдельно для обработки текста программы без ее  
компиляции.  
Основные директивы препроцессора:  
#include – вставляет текст из указанного файла;  
#define – задаёт макроопределение (макрос) 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. директивa #include (включение файла).  
Директива #include вставляет содержимое заданного файла в место расположения  
этой директивы в исходном тексте программы.  
Синтаксис:  
#include "спецификация\_пути"  
#include <спецификация\_пути>  
где "спецификация\_пути" — это имя файла, с необязательным указанием  
его местоположения.  
Кавычки и угловые скобки определяют способ поиска включаемых файлов:  
– если спецификация файла заключена в угловые скобки, то он должен  
находиться в каталоге, указанном компилятором (обычно это каталог INCLUDE,  
в котором хранятся заголовочные файлы).  
– если имя файла заключено в кавычки, то поиск выполняется в рабочем  
каталоге.  
– если указан полный путь, то предпроцессор использует его для поиска файла.  
4. Управление процессом компиляции:  
«Прагма» – это инструкция компилятору С/С++, используется для указания опций  
компилятору для управления его работой.  
Синтаксис:  
#pragma параметры  
Так #pragma once контролирует, чтобы конкретный включаемый файл при  
компиляции подключался строго один раз.

директивы: #if, #else  
Директива условной

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

директива #error  
Синтаксис:  
#error текст  
С помощью директивы #error можно определить текстовую строку (обратите  
внимание, что строка записывается без кавычек), которая будет выведена как  
сообщение об ошибке при компиляции. предопределенные макросы  
Компилятор C++ автоматически определяет некоторые макросы, например:  
\_\_LINE\_\_  
Этот макрос заменяется номером текущей строки в форме десятичной целой  
константы.  
Несмотря на то, что он называется предопределенным макросом, значение его  
меняется динамически в зависимости от местоположения макроса.  
Этот макрос в сочетании с макросом \_\_FILE\_\_ можно использовать при генерации  
сообщения об ошибке для вывода несоответствия, обнаруженного программой. В  
этом случае сообщение будет содержать номер строки с именем исходного файла,  
в котором была обнаружена ошибка. опрераторы препроцессора # и ##.  
Стрингификация (#) – это преобразование фрагмента кода в строковую  
константу, т.е. преобразование аргумента в строку. Например, в результате  
стрингификации STR(hello) аргумент преобразуется в символьную строку "hello".  
Канкатенация (##) (оператор конкатенации) – это конкатенация двух строковых  
констант. При работе с макросами, это означает объединение двух лексических  
единиц в одну более длинную. Например, один аргумент макроса может быть  
объединен с другим аргументом или с каким-либо текстом.

**36. Статическая библиотека: определение, назначение, применение. Реализация статической  
библиотеки в Visual C++. Утилита LIB. Создание статической библиотеки. Применение**

Библиотека – файл, содержащий либо код на интерпретируемом языке, либо  
байт-код для виртуальной машины.  
Например, библиотеки для языка Python могут распространяться либо в виде  
файлов с исходным кодом (расширение «py»), либо в виде файлов с байт-  
кодом (расширение «pyc», (пайк) «py» + буква «c» от англ. compiled).

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.  
Статическая библиотека – файл с объектными модулями, который  
подключается к программе на этапе компоновки.  
Компоновщик (linker, редактор связей) – программа, принимающая один или  
несколько объектных модулей и формирующая на их основе загрузочный  
модуль:

Если программа состоит из нескольких объектных файлов, компоновщик  
собирает эти файлы в единый исполнимый модуль, вычисляя и подставляя  
адреса вместо неопределенных внешних имен, в течение времени компоновки  
(статическая компоновка) или во время исполнения (динамическая  
компоновка)

Статическая библиотека Microsoft:  
файл с расширением lib.  
Для работы с библиотекой предназначена утилита LIB.  
7.1. Создание статической библиотеки с помощью «Мастера классических  
приложений Windows». Шаг 1:

Определяем имя решения, проекта и выбираем место размещения на диске.  
7.2. Создание статической библиотеки с помощью «Мастера классических  
приложений Windows». Шаг 2.  
Выбираем тип приложения «Статическая библиотека» (снимаем флажок  
«Предварительно скомпилированные заголовки» при необходимости):

В проект добавляем один или несколько файлов, содержащих реализации  
функций библиотеки.  
7.3. Запускаем проект на выполнение

В пункте раздела «Библиотекарь»  
→ «Командная строка» отображается  
текущее значение параметра /OUT.  
Расширение выходных файлов определено как .lib.  
Директорий – папка проекта Debug.

После построения проекта в папке решения Debug размещен файл статической  
библиотеки (.lib).

В журнале проекта зафиксировано выполнение сборки проекта.  
Видим, что файл статической библиотеки создан утилитой LIB.

ежимы 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.

Параметры LIB:  
https://docs.microsoft.com/ru-ru/cpp/build/reference/overview-of-lib?view=vs-2019  
/DEF  
Создание библиотеки импорта и файла экспорта  
Дополнительно в Построение библиотеки импорта и файла экспорта.  
/ERRORREPORT  
Передача Майкрософт сведений о внутренних ошибках.  
Дополнительно в Запуск программы LIB.  
/EXPORT  
Экспорт функции из программы.  
Смори в разделе Построение библиотеки импорта и файла экспорта.  
/EXTRACT  
Создание объектного файла (OBJ-файла), содержащего копию  
элемента существующей библиотеки.  
Дополнительно в разделе Извлечение члена библиотеки.  
/INCLUDE  
Добавление символа в таблицу символов.  
Дополнительные сведения см. в разделе Построение библиотеки  
импорта и файла экспорта.  
/LIBPATH  
Переопределяет путь к библиотеке среды.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/LIST  
Отображает информацию о выходной библиотеке в стандартном виде.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/LTCG  
Инициирует построение библиотеки с помощью создания кода времени  
компоновки.  
Дополнительные сведения см. в разделе Запуск программы LIB.  
/MACHINE  
Задание целевой платформы для программы.

8  
Дополнительные сведения см. в разделе Запуск программы LIB.  
/NAME  
При построении библиотеки импорта указывает имя библиотеки DLL,  
для которой была создана библиотека импорта.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/NODEFAULTLIB  
Удаляет одну или несколько библиотек по умолчанию из списка  
искомых библиотек при разрешении внешних ссылок.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/NOLOGO  
Отключает вывод программой LIB уведомления об авторских правах и  
номере версии, а также отображение команд командного файла.  
Дополнительные сведения см. в разделе Запуск программы LIB.  
/OUT  
Переопределяет имя выходного файла, используемое по умолчанию.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/REMOVE  
Пропуск объекта из выходной библиотеки.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/SUBSYSTEM  
Сообщает операционной системе способ запуска программы,  
созданной путем привязки к выходной библиотеке.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/VERBOSE  
Отображает подробные сведения о ходе сеанса, включая имена  
добавляемых OBJ-файлов.  
Дополнительные сведения см. в разделе Запуск программы LIB.  
/WX  
Обработка предупреждений, как ошибок.  
Дополнительные сведения см. в разделе Запуск программы LIB.

Заголовочные файлы. Правила создания.  
В заголовочном файле задают:  
− прототипы всех функций, которые входят в библиотеку (интерфейс  
библиотеки), например: void Print(Inctance d);  
− типы пользователя;  
− константы, например: #define DICTNAMEMAXSIZE 20

9.2. Главная программа для использования статической библиотеки.  
Добавляем в решение новый пустой проект SE\_Lab06Ex – консольное  
приложение, которое будет использовать созданную нами библиотеку.

Назначаем проект SE\_Lab06Ex запускаемым и выполняем его.  
На странице свойств проекта в разделе «Компоновщик»  
→ Командная  
строка в окне «Дополнительные параметры» добавляем параметр,  
указывающий местоположение и имя статической библиотеки.

Подключаем в main заголовочный файл библиотеки:

Получаем исполнимый файл SE\_Lab06Ex:

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

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

Алфавитом V называется конечное множество символов, допустимых в  
языке.  
Пример:},,{ cbaV = , I = {anbn},  
где n — натуральное число.  
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.Цепочкой  
α (альфа) в алфавите I называется любая конечная  
последовательность символов этого алфавита.  
Цепочка, которая не содержит ни одного символа, называется пустой  
цепочкой и обычно обозначается  
ε или  
λ.

Длиной цепочки  
α (обозначается |  
α|) называется число составляющих ее  
символов.  
Пример: |  
α| = 3, |  
β| = 4, |  
γ| = 7. Длина пустой цепочки |  
ε|=0.  
4). 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.  
5). Для любой цепочки  
α справедливо утверждение  
αε=  
εα.  
6).  
αn называется итерацией цепочки  
α.  
Справедливы следующие утверждения:  
α0 =  
ε  
αn =  
αn-1  
α =  
ααn-1 для n≥1.  
Пример:ab=α ,abababab ==α 33 )( ,ε  
α  
=0  
7). Цепочки  
α и  
β равны (  
α =  
β), если они имеют один и тот же состав  
символов, одно и тоже количество символовβ=α и тот же порядок  
следования символов.  
8). Реверсом (обращением) цепочки  
α называется цепочка  
αR, составленная  
из символов цепочки  
α, записанных в обратном порядке.  
Пример. Пусть алфавит I = {a, b, c, d}, тогда для цепочек этого алфавита  
α=ab и  
β=bcd будет справедливо |  
α|=2, |  
β|=3,  
αβ= abbcd,  
α2=abab,  
βR=dcb.  
9). ПустьI – алфавит, тогда+  
I – множество всех цепочек, состоящих из  
символов алфавита I, исключая пустую цепочку (  
λ).+  
∉ I  
λ  
10). ПустьI – алфавит, тогда\*  
I – множество всех цепочек, состоящих из  
символов алфавита I, включая пустую цепочку.\*  
I∈λλ∪= +  
II \*

4  
11). Языком L(I) над алфавитомI называется произвольное подмножество  
цепочек из\*  
I ,\*  
)( IIL ⊆ .  
Пример:},,{ cbaI = ,},,{)(1 bcacabIL =  
,},,{)(2 cbaIL = ,},,{)(3 ccbbaaIL = , ...  
12). Язык)(1 IL является подмножеством языка)(2 IL , если каждая цепочка,  
входящая в язык1L , входит в язык2L ; язык2L включает язык1L .))()(()()( 2121 ILILILIL ∈α⇒∈α∀⇔⊆  
13). Языки)(1 IL и)(2 IL совпадают (эквивалентны), если язык)(1 IL  
включает язык)(2 IL и язык)(2 IL включает язык)(1 IL .))()()()(()()( 122121 ILILILILILIL ⊆∧⊆⇔=.

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,...  
Пример: α = a 1... a n – цепочка из n символов.  
Пустую цепочку символов будем обозначать  
λ,  
ε.

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

Теория формальных языков (формальных грамматик) занимается описанием, распознаванием и переработкой языков. Описание любого языка должно быть конечным, хотя сам язык мо­жет содержать бесконечное множество цепочек. Полезно иметь возможность описания отдельных типов языков, имеющих те или иные свойства, т. е. иметь различные типы конечных описаний. Синтаксис языка можно задать с использованием синтаксических диаграмм или другим альтернативным способом, воспользовавшись нотацией Бэкуса-Наура (НБН). При ис­пользовании нотации Бэкуса-Наура нетерминальные символы (подлежащие дальнейшему определению) языка заклю­чаются в угловые скобки вида < ... > .

Последовательность символов : : = означает «определя­ется как», а символ | означает "или". Пример описания в нотации Бэкуса-Наура арифметических выражений, содержащих переменные *a, b,*приведен ниже.

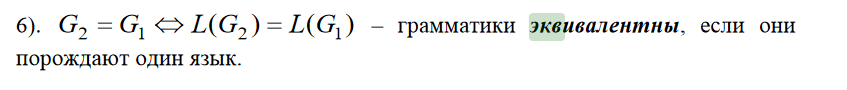
< выражение > : : = < терм > | < терм > + < выражение > | < терм > - < выражение >

< терм > : : = < множитель > | < множитель > \* < терм > | < терм > / < множитель >

< множитель > : : = a|b| ( < выражение > )

+ Существуют два универсальных способа описания отдельных классов языков. Первый способ – *грамматики*как механизмы, порождающие цепочки символов. Второй способ определяет язык в терминах множества цепочек, допускаемых некоторым распознающим устройством, называемым*автоматом*.

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

****

Основными объектами грамматики являются базовые элементы языка или терминальные символы, а также цепочки, построенные из этих элементов.

При определении грамматики будем придерживаться следующих соглашений:

* прописными буквами обозначаются нетерминальные символы языка;
* строчными буквами латинского алфавита обозначаются терминальные символы языка;
* цепочки обозначаются либо прописными буквами латинского алфавита, либо греческими буквами.
* символ «→»используется для обозначения отношения "определяется как".

Введем в рассмотрение пустую цепочку ε, не содержащую ни одного символа.

*Длиной*цепочки будем называть число символов, входящих в эту цепочку, например:

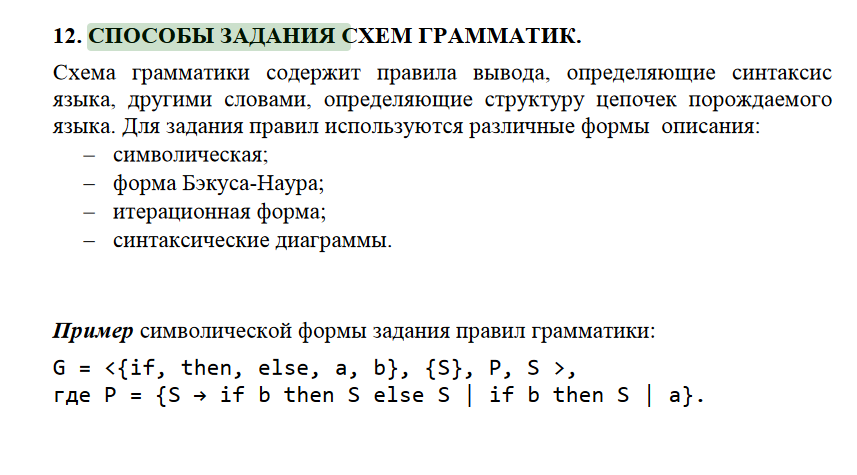
*B = abab, |B| = |abab*| = 4,

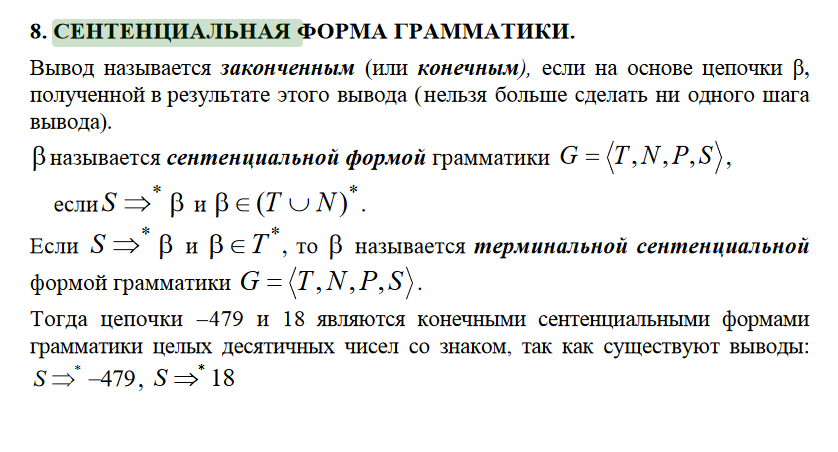
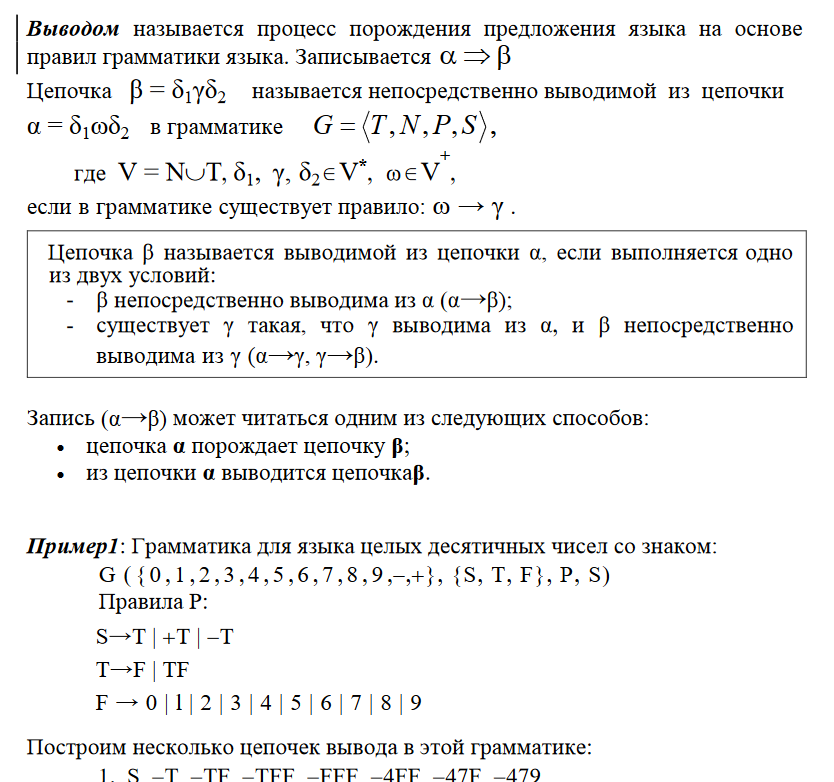
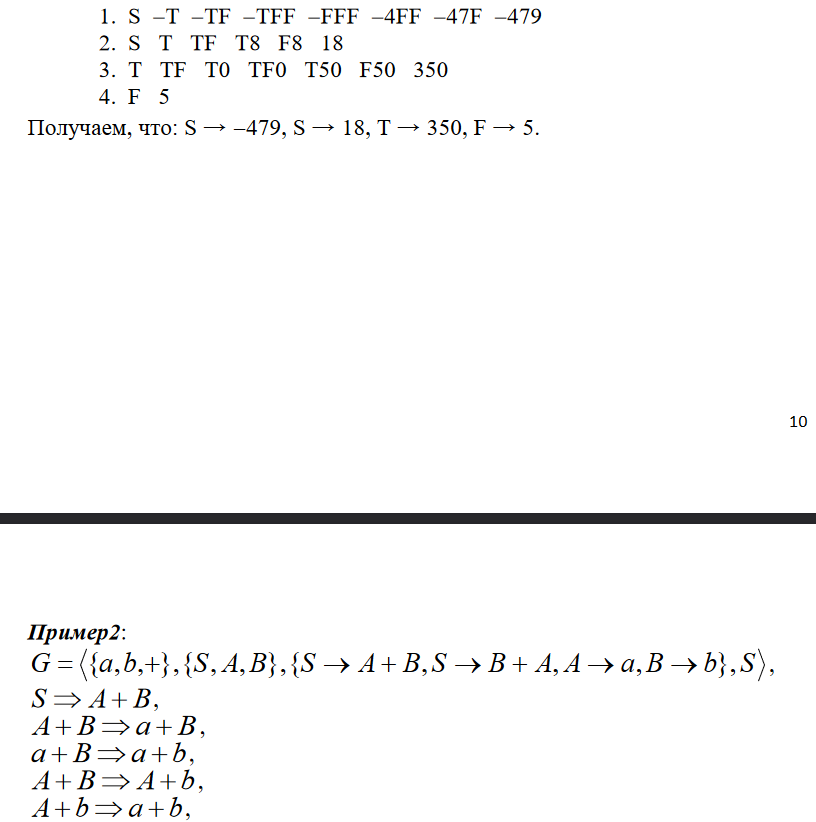
| ε| = 0.

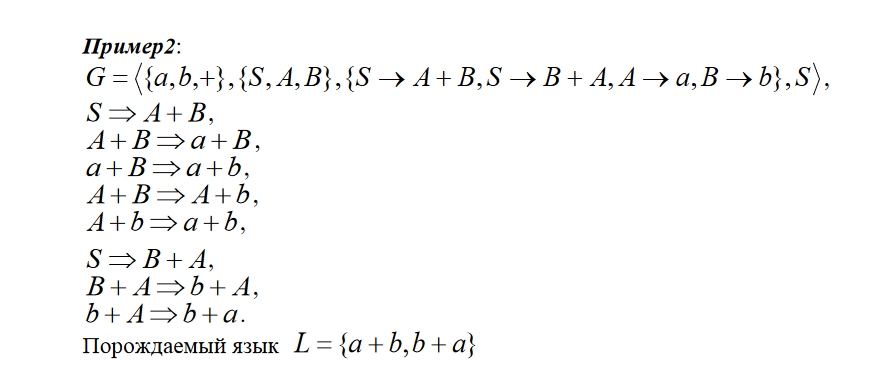
*Конкатенацией*двух цепочек*X*и*Y*называется такая цепочка*Z*, которая получается непосредственным слиянием цепочки*X*, стоящей слева, и цепочки*Y*, стоящей справа. Например, если*X = ffg*,*Y = ghh*, то конкатенация*X*и*Y*– это цепочка*Z*=*ffgghh*.

+ Будем обозначать через А\*множество всех возможных цепочек конечного множества А базовых элементов (символов), включая пустую цепочкуε; Множество А называют также*алфавитом*. Любое множество цепочек L ⊆A\*называется*формальным языком*, определенным на алфавите А.

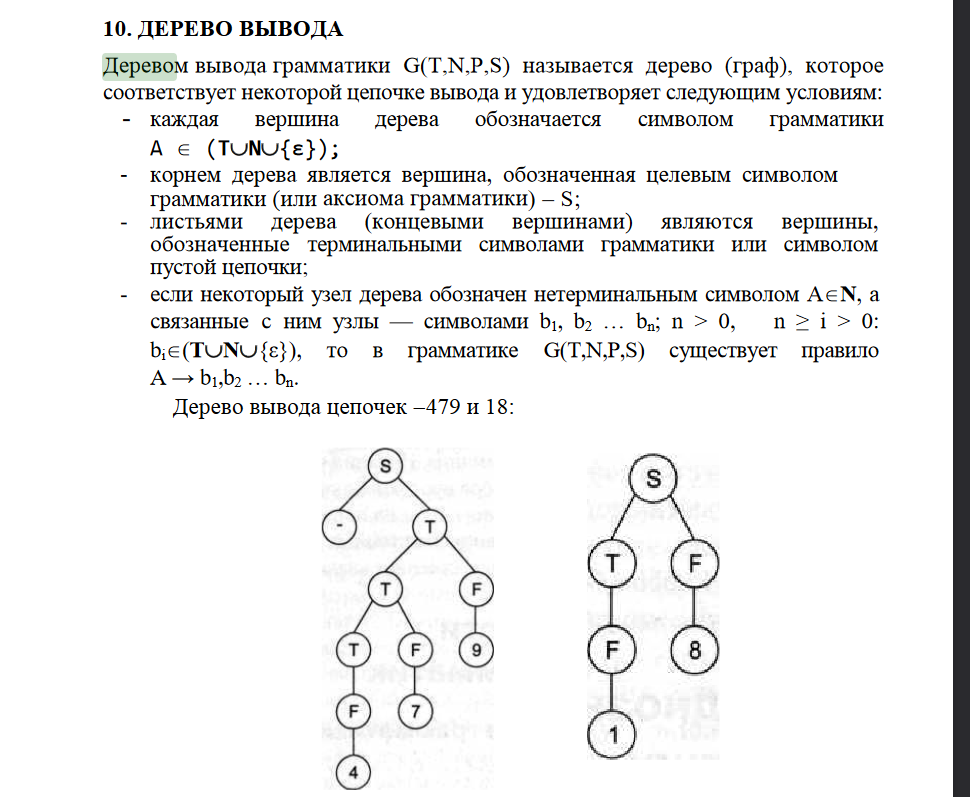
**39. Теория формальных языков. Формальная грамматика (определение, назначение),  
выводимость цепочки символов в грамматике, сентенциальная форма грамматики, язык,  
порождаемый грамматикой, способы задания грамматик. Примеры.**

****

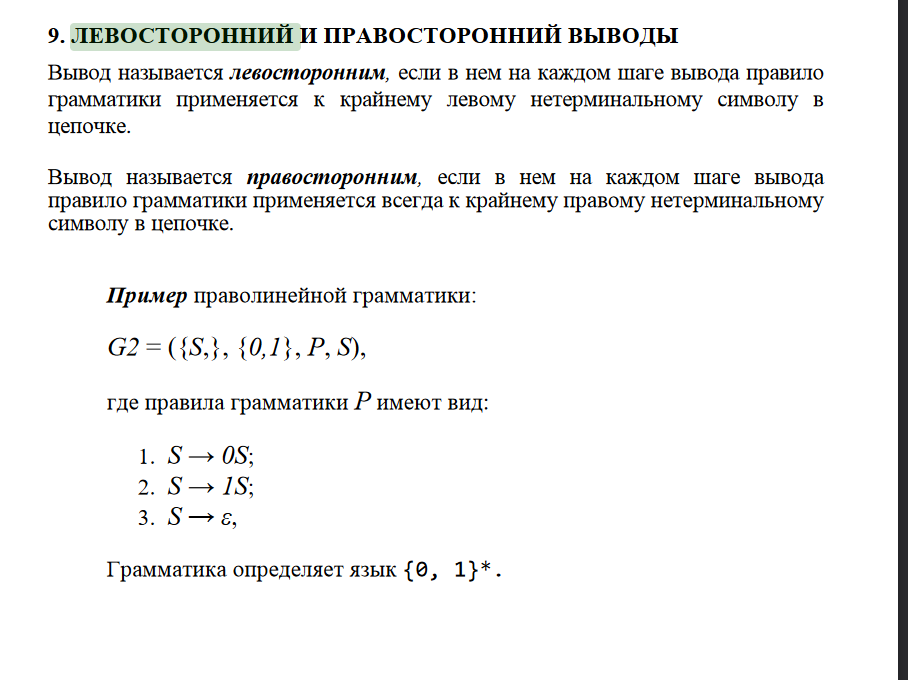
**** **** 

****

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

****

13  
10. ДЕРЕВО ВЫВОДА  
Деревом вывода грамматики G(T,N,P,S) называется дерево (граф), которое  
соответствует некоторой цепочке вывода и удовлетворяет следующим условиям:  
- каждая вершина дерева обозначается символом грамматики  
А ∈ (T∪N∪{ε});  
- корнем дерева является вершина, обозначенная целевым символом  
грамматики (или аксиома грамматики) – S;  
- листьями дерева (концевыми вершинами) являются вершины,  
обозначенные терминальными символами грамматики или символом  
пустой цепочки;  
- если некоторый узел дерева обозначен нетерминальным символом А∈N, а  
связанные с ним узлы — символами b1, b2 ... bn; n > 0, n ≥ i > 0:  
bi∈(T∪N∪{ε}), то в грамматике G(T,N,P,S) существует правило  
A → b1,b2 ... bn.  
Дерево вывода цепочек −479 и 18:  
1). Цепочка\*  
)( TN ∪∈β выводима из цепочки+  
∪∈α )( TN в  
грамматикеSPNTG ,,,= , если существуют цепочки, такие чтоβ⇒γ⇒⇒γ⇒γ⇒γ⇒α n...210  
. Записывается какβ⇒α \* .  
Тогда последовательностьβγγγγα ,,,,, 210 n называется выводом длиныn .  
Пример:SAAAASASG },,1000,10{},,{},1,0{ λ→→→=  
Вывод100010010 AAAS ⇒⇒⇒  
Длина вывода цепочки1000A из стартового символаS равна 2.  
(Запись второго правила рекурсивна).

14  
2). Записьβ⇒α \* предполагает0≥n шагов выводаβ изα .  
В том случае, еслиβ⇒α , то число шагов вывода0=n .  
Записьβ⇒α + предполагает0>n шагов выводаβ изα .  
3). Для записи правил вывода с одинаковыми левыми частями  
α → β1 α → β2 ... α → βn  
будем пользоваться сокращенной записью  
α → β1 | β2 |...| βn.  
Каждое βi , i = 1, 2, ... ,n – альтернатива правила вывода из цепочки α.  
4).)(GL – язык, порождаемый грамматикойG .  
Язык)(GL содержит все терминальные цепочки, выводимые изS :}\*|{)( \* α⇒∈α= STGL  
.  
Язык, порождаемый грамматикой – это множество всех выводимых из  
аксиомы грамматики терминальных цепочек  
5).)(GL – множество терминальных сентенциальных форм грамматикиG .  
Пример:SbBaAABSBASBASbaG },,,,{},,,{},,,{ →→+→+→+=  
,},{)( abbaGL ++=  
6).)()( 1212 GLGLGG =⇔= – грамматики эквивалентны, если они  
порождают один язык.  
Пример:  
Язык L в алфавите}0,1{=V , состоящий из пустой строки и строк, каждая из  
которых содержит строку, состоящую из нулей и такого же количества  
единиц, можно также описать с помощью формальной системы определения  
множеств как L={0n1n | n ≥0}. ****