БГТУ, ФИТ, ПОИТ, 4 курс

Дисциплина: Языки программирования

1. **Система программирования –** комплекс программных средств, предназначенный для реализации программного обеспечения (написания и отладки программного кода).

Система программирования: трансляторы, компоновщики, отладчики, профилировщики, программные библиотеки, IDE.

Компоненты системы программирования. Назначение.

1) Редактор программного кода (текстовый редактор)
2) Компилятор.
3) Отладчик (обнаружить и локализовать ошибки в программном коде)
4) Компоновщик.
5) …

2. **Интегрированная среда разработки (IDE)**

Интегрированная среда разработки программного обеспечения. Прмеры:

1) Eclipse.
2) Microsoft Visual Studio.
3) NetBeans.
4) JDeveloper
5) Sun Studio
6) ...

3. **Классический жизненный цикл разработки программного обеспечения.**

Перечень этапов разработки программного обеспечения: тестирование, анализ, реализация, сопровождение, проектирование. Порядок следования:

Анализ – определение функциональных возможностей, требования к надежности и безопасности продукта, к используемым данным, к пользовательской документации, к эксплуатации и сопровождению.

Проектирование – определение компонентов аппаратной части, программного обеспечения и действий, выполняемых персоналом.

Реализация – кодирование и документирование каждого компонента ПО.

Тестирование – подготовка тестовых процедур и соответствующих им наборов данных для тестирования каждого компонента на соответствие предъявляемым к ним требованиям.

Сопровождение – действия и задачи, которые выполняются в целях модернизации или адаптации ПО.

4. **Определения некоторых понятий:**

«Программный продукт»

- программа, которая может быть использована без присутствия автора.
- исполняется, тестируется, конфигурируется без присутствия автора; сопровождается документацией.

«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.

«Исходный код» (исходная программа)

- текст программы, написанный на языке программирования
- готовится с помощью текстовых редакторов
- поступает на вход транслятора в виде текстового файла или раздела библиотеки

«Транслятор»

- программа, преобразующий исходный код на одном языке программирования в исходный код на другом языке – целевом языке.
    - проверяет правильность синтаксических конструкций языка
    - проверяет семантические правила языка
    - обнаруживает ошибок
- разновидности транслятора:
    - трансляторы (компиляторы) обрабатывают весь исходный текст и переводят его в эквивалентный код на целевом языке
    - интерпретаторы переводят и выполняют программу с языка высокого уровня в машинный код строка за строкой.

«Язык ассемблера»

- язык низкого уровня, с помощью которого программист получает прямой доступ к аппаратным ресурсам компьютера
- машинно-ориентированный язык программирования (тесно связан с архитектурой процессора)
- машинно-ориентированный язык программирования, команды которого соответствуют машинным командам для конкретной архитектуры компьютера
- программы на языке ассемблера не переносимы по определению

«Ассемблер»

- транслятор с исходного кода на языке ассемблера в текст в машинном коде (язык, который может интерпретироваться процессором)

«Объектный код»

- результат работы транслятора.

Один файл объектного кода – **объектный модуль.**

Объектный модуль — двоичный файл, который может быть объединён с другими объектными файлами при помощи редактора связей (компоновщика) для получения готового исполняемого модуля, либо библиотеки.

«Компоновщик (linker, редактор связей)»

- программа обрабатывающая один или несколько файлов с объектным кодом и формирующая на их основе загрузочный файл
- компоновщик (linker, редактор связей) выполняет разрешение внешних адресов памяти, по которым код одного файла может обращаться к информации из другого файла

Основные задачи редактора связей:

- связывание между собой объектных модулей, порождаемых компилятором, в единый загрузочный файл;
- подготовка таблицы трансляции относительных адресов для загрузчика;
- статическое подключение библиотек с целью получения единого исполняемого модуля;
- подготовка таблицы точек вызова функций динамических библиотек.

«Загрузочный код»

- результат работы компоновщика
- один файл загрузочного кода – загрузочный модуль

Загрузочный модуль (или исполняемый файл) – файл, который может быть запущен на выполнение процессором под управлением операционной системы.

«Загрузчик»

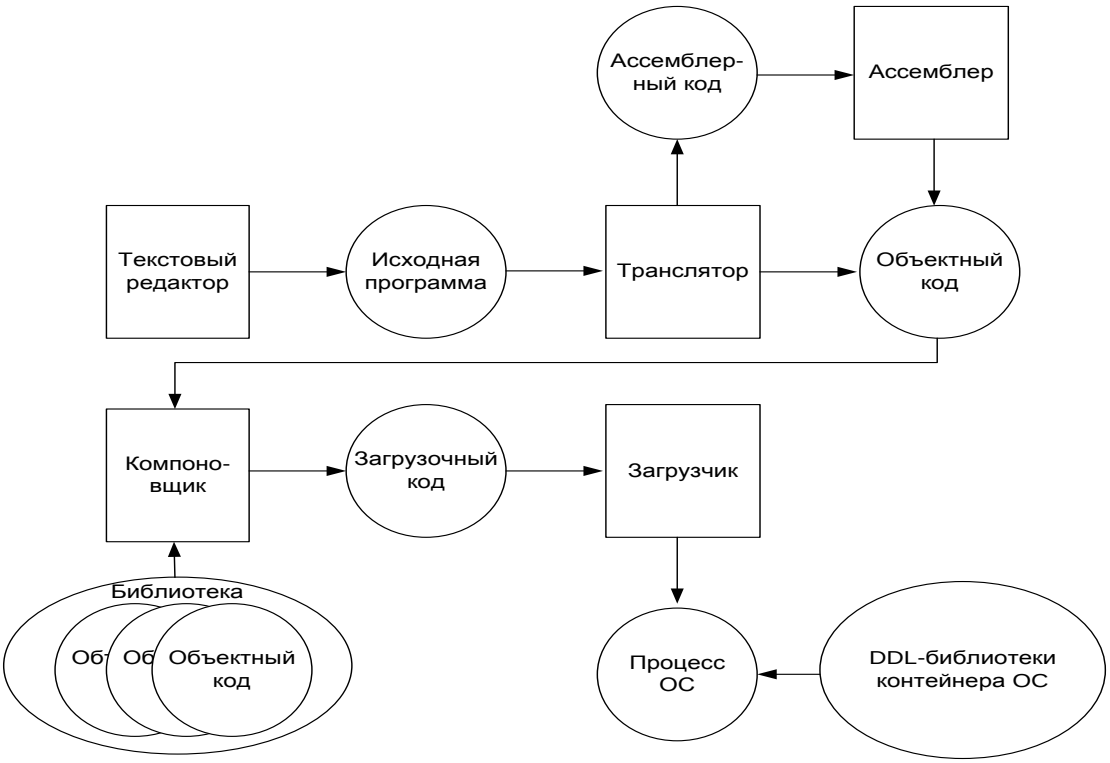
- программа, предназначенная для считывания загрузочного файла и формирования на его основе процесса операционной системы

Задача загрузчика – преобразовать относительные адреса разделов памяти в абсолютные, используя специальную таблицу, которую редактор связей вставляет в заголовок исполняемого файла.

«Отладчик»

- программа, позволяющая контролировать ход выполнения программы, просматривать и изменять области памяти

Общая схема преобразования исходного кода в процесс операционной системы:



«Алфавит языка»

- базовый набор символов, разрешенных к использованию языком, который основывается на одной из кодировок.

«Кодировка»

- кодировка (charset) определяется как комбинация набора символов и схемы кодирования.

ASCII (8-битная) - кодировка для представления десятичных цифр, символов латинского и национального алфавитов, знаков препинания и управляющих символов. Нижнюю половину кодовой таблицы (0 - 127) занимают символы US-ASCII, а верхнюю (128 - 255) — другие нужные символы (CP866, CP1251).

Windows-1251 (8-битная) - набор символов и кодировка, стандарт для русских версий Microsoft Windows.

UNICODE: – стандарт кодирования символов, состоящий из 2х разделов:

- UCS - universal character set (универсальный набор символов задаёт однозначное соответствие символов кодам);
- UTF - Unicode transformation format (семейство кодировок - определяет машинное представление последовательности кодов UCS).

Кодировки: **UTF-8**, **UTF-16 (LE/BE)**, **UTF-32**.

BOM (маркер последовательности байтов) — юникод-символ, используемый для указания порядка следования байтов текстового файла.

Примеры.

1) Кодировки. Однобайтовые кодировки:
   - ASCII
   - CP866
   - ASCII
   - CP1251
   - КОИ-8
2) Кодировки. Двухбайтовые кодировки:
   - UTF-16
   - UTF-16BE
   - UTF-16LE
3) Кодировка, которая позволяет хранить символы Юникода, используя переменное количество байт (от 1 до 6):
   - UTF-8 +

4) Дано четырех байтовое десятичное число 64.
   - представление в формате **Little Endian** (прямой порядок, от младшего к старшему):
     - 40 00 00 00
   - представление в формате **Big Endian** (обратный порядок, от старшего к младшему):
     - 00 00 00 40

5. **Термины и основные понятия языков программирования:**

«Идентификатор»

- имя компонента программы (переменной, функции, метки, типа и пр.), составленное программистом по определенным правилам.

«Зарезервированные идентификаторы языка»

- идентификатор, предопределенный правилами языка программирования или
- идентификаторы, которые предварительно определены в системе программирования.

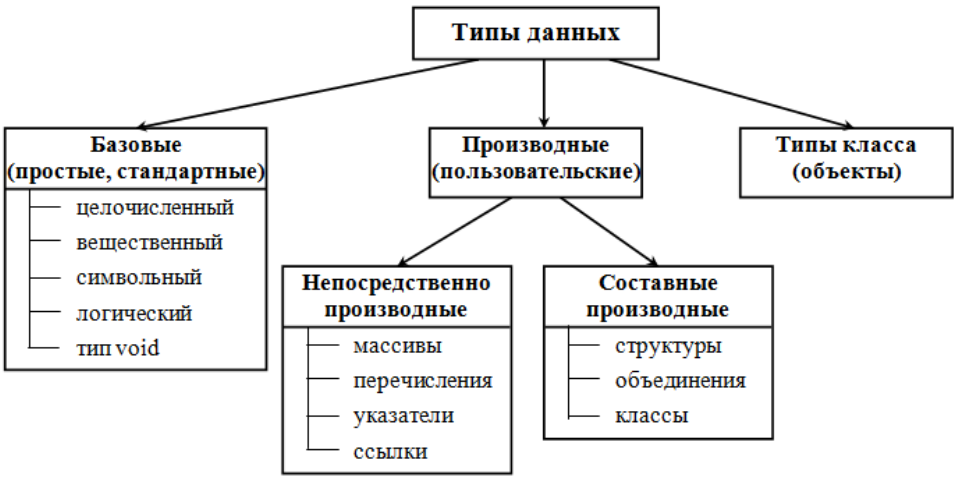
«Ключевые слова»

- последовательности символов алфавита языка, имеющие специальное назначение, зарезервированные для обозначения типов переменных, класса хранения, элементов операторов.

«Типы данных»

- фундаментальные (базовые, встроенные)
- определенные программистом (пользовательские типы данных).

Типы данных определяют их размеры в байтах и допустимые диапазоны значений, принцип размещения данных в памяти, применяемые операции.



«Литералы»

- неизменяемые величины (логические, целые, вещественные, символьные, строковые константы, дата и время)
- компилятор, выделив константу в качестве лексемы, относит ее к одному из типов данных по ее внешнему виду
- программист может задать тип константы и явным образом.

Для простых типов данных определяются границы диапазона и количество байт, занимаемых ими в памяти компьютера:

Описание типа	Размер (биты)	Диапазон	.NET	C#	MASM	C/C++	Java
Байт со знаком	8	от -128 до 127	SByte	sbyte	SBYTE	signed char	byte
Байт без знака	8	от 0 до 255	Byte	byte	BYTE	unsigned char	-
Короткое целое число со знаком	16	от -32 768 до 32 767	Int16	short	SWORD	short	short
Короткое целое число без знака	16	от 0 до 65 535	UInt16	ushort	WORD	unsigned short	-
Среднее целое число со знаком	32	от -2147483648 до 2147483647	Int32	int	SDWORD	int	int
Среднее целое число без знака	32	от 0 до 4294967295	UInt32	uint	DWORD	unsigned int	-
Длинное целое число со знаком	64	от -9223372036854775808 до 9223372036854775807	Int64	long		long	long
Длинное целое число без знака	64	от 0 до 18446744073709551615	UInt64	ulong		unsigned long	-
Вещественное число одинарной точности с плавающей запятой	32	$\pm 1{,}5*10^{-45}$ до $\pm 3{,}4*10^{33}$	Single	float		float	float
Вещественное число двойной точности с плавающей запятой	64	$\pm 5*10^{-324}$ до $\pm 1{,}7*10^{306}$	Double	double		double	double
Символ (8 бит)	8	ASCII	-	-	-	char	-
Символ (16 бит)	16	UNICODE	Char	char	-	wchar_t	char
Логический тип	8	{true, false}	Boolean	bool	-	bool	boolean

Строка	-	-	String	string	-	char*/wchar_t*	String
Пустой тип	-	-	-	void	-	void	void

«Массивы данных фундаментальных типов»

- коллекция однородных данных, размещенных последовательно в памяти, доступ к которым осуществляется по индексу.

«Пользовательские типы»

- типы, создаваемые пользователем, всегда должно быть объявление типа.

«Инициализация переменных (памяти)»

- присвоение значения в момент объявления переменной; как правило, применяются литералы.

Отличие от присвоения: при присвоении явно перемещаются данные.

Инициализация массивов, структур. Функциональный вид инициализации.

«Область видимости переменных»

- доступность переменных по их идентификатору в разных частях (блоках программы).

«Преобразование типов»

- автоматическое (неявное) преобразование
- явное преобразование.

Неявное преобразование выполняется транслятором (компилятором или интерпретатором) по правилами, определенном в стандарте языка.

«Константное выражение»

- выражение, которое должно быть вычислено на этапе компиляции.

«Выражение»

- объединение литералов, имен (переменных, функций и пр.), операторов и специальных символов, служащих для вычисления выражения или достижения побочных эффектов (например: при применении в выражении функций).

lvalue – именующее выражение – это ссылка на значение – может использоваться в левой и правой части оператора присваивания (имя переменной, ссылка на элемент массива по индексу, вызов функции

возвращающей указатель, всегда связано с областью памяти, адрес которой известен).

rvalue – значащее выражение – может использоваться только в правой части оператора присваивания (не связано с адресом, связано только со значением, например: литералы, вызов функции, возвращающей значение).

«Пространство имен»

- именованная область видимости. Применяется для разрешения конфликтов имен.

«Классы памяти»

- код, стек, статические данные, динамическая область. Принцип разделения памяти.
- области памяти для хранения значения переменной:
    - статическая память
    - динамическая память
    - стековая память
    - регистровая

«Система обработки исключений»

- исключение – событие при выполнении программы, при котором ее дальнейшее выполнение становится бессмысленным.

«Оператор языка»

- законченное описание некоторого действия.

Операторы делят на исполняемые и неисполняемые, простые и составные. Исполняемые операторы задают действия над данными.

Неисполняемые операторы служат для описания данных.

Составной оператор или блок - это группа операторов, заключенная в операторные скобки. Блоки могут быть вложенными.

«Инструкции языка»

- инструкции объявления (if (int) условное объявление),
- составные инструкции ({}),
- инструкции выбора (if, switch), циклы (while, do while, for),
- инструкции переходов (goto, break, continue, return),
- инструкции обработки исключений (try, catch, throw).

«Препроцессор»

- часть транслятора, которая выполняется до процесса трансляции
- результатом работы препроцессора является текст, сформированный из исходного под действием директив препроцессора.
- основной задача - модификация исходного кода программы для замены одних лексем другими.
- директива препроцессора, которая автоматически указывает компоновщику, какую библиотеку нужно подключить:
    - #pragma comment(lib, "name.lib")

«Программные конструкции»

- программные блоки
- процедуры
- функции
- механизмы передачи параметров
- возврат значений
- области видимости
- перегружаемые функции

«Программные конструкции»

- соглашения о вызове: способ передачи параметров, порядок размещения параметров, очищение стека, порядок возврата значения и пр.

«Соглашения о вызовах»

- _cdecl - Параметры передаются через стек, справа налево. Стек освобождает вызывающая функция. Результат вычислений возвращается через регистр eax.
- _fastcall - Параметры передаются через регистры. Стек освобождает вызываемая функция. Результат вычислений возвращается через регистр eax.
- _stdcall - Параметры передаются через стек, справа налево. Стек освобождает вызываемая функция. Результат вычислений возвращается через регистр eax.

C++: соглашение о вызове, используемое по умолчанию (_cdecl)

C++: соглашение о вызове, заданное явно

Примеры

```
#include "stdafx.h"
#include <iostream>
int k = 10;
int sum(int k, int m){ k += m; return k; }

int _tmain(int argc, _TCHAR* argv[])
{
    int a = 2, b = 2, c;

    c = sum(a, b);
    k += c;
    c = sum(a, b);
    k += c;
    c = sum(a, b);
    k += c;

    return 0;
}
```

```
#include "stdafx.h"
#include <iostream>
#include <locale>

int k = 1;

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    {
        int d = 2, m = 4;
        for (int k = 0; k < 7; k++) std::cout << k << std::endl;
        std::cout << d + m << std::endl;
        k += m;
    }

    return 0;
}
```

Вопросы:

Вычислить значение переменной
Какое соглашение применяется при вызове функций WinAPI?
Указать явно инициализированные переменные
Указать фундаментальные типы данных
Указать пользовательские типы данных
Указать идентификаторы
Указать ключевые слова

«Статическая библиотека»

- файл, содержащий объектные модули (обычно файл с расширением lib)
- является входным файлом для компоновщика (linker).
- Утилита LIB

«Стандартная библиотека»

- как правило, в составе языка программирования есть обязательный (стандартный) набор функций.

Такие функции называют встроенными функциями. Встраиваться функции могут тремя способами:

1) встраиваться прямо в код транслятора;
2) в отдельной библиотеке;
3) сочетание первого и второго случаев.

6. Теория формальных языков. Основные понятия.

Определения

«Алфавит формального языка»

- множество символов, из которых состоят цепочки формального языка

Примеры

1) множество, являющееся алфавитом азбуки Морзе:
   {·, –}
2) множество символов, являющееся бинарным алфавитом:
   {0, 1}
3) множество символов, являющееся алфавитом десятичной системы счисления:
   {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
4) множество символов, являющееся алфавитом шестнадцатеричной системы счисления:
   {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
5) множество символов латинского алфавита в нижнем регистре:
   {a, b,…,z}

«Символ алфавита формального языка»

- символ, входящий во множество, определяющее алфавит формального языка

«Цепочка символов над алфавитом»

- любая конечная последовательность символов этого алфавита

«Пустая цепочка»

- цепочка, которая не содержит ни одного символа

«Длина цепочки символов над алфавитом»

- число символов в цепочке

«Реверс цепочки символов»

- цепочка, символы которой записаны в обратном порядке

«Конкатенация двух цепочек формального языка»

- цепочка, полученная путем сцепления двух цепочек

Обозначения

- a – цепочка, состоящая из символа a алфавита V
- |ab| – обозначение длины цепочки ab
- λ – цепочка, не содержащая ни одного символа алфавита V
- пусть abc – цепочка символов в алфавите V, то
    - cba – реверс этой цепочки
    - |cba| – обозначение длины этой цепочки
    - 3 – длина этой цепочки

«Язык над алфавитом V»

- подмножество цепочек конечной длины над алфавитом V

Определения

1) V – подмножество цепочек конечной длины над алфавитом V
2) V* – обозначение «Множество всех цепочек над алфавитом V» - множество, содержащее все цепочки над алфавитом V, включая пустую
3) V+ – обозначение «Множество всех непустых цепочек над алфавитом V»
4) ∅ – обозначение пустого множества

«Формальный язык над алфавитом *I*»

- подмножество всех цепочек символов, которые можно составить из символов множества *I*

Способы задания формальных языков: язык L можно определить тремя способами:

- перечислением всех цепочек языка;
- указанием способа (алгоритма) порождения цепочек;
- определить метод (алгоритм) распознавания цепочек.

Лексика языка программирования – множество цепочек языка.
Синтаксис языка – набор формальных правил, определяющий конструкции (последовательности цепочек).
Семантика языка - набор неформальных правил (невозможно записать правила в виде формальных выражений), которые описываются словесно (например, в руководстве программиста).

Чтобы **создать** язык программирования, следует определить:

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

7. **Порождающая грамматика** – это упорядоченная четверка $G = \langle T, N, P, S \rangle$,

где

- T - множество терминальных символов или алфавит терминальных символов (символы языка, определяемые грамматикой)
- N - множество нетерминальных символов или алфавит нетерминальных символов (символы, определяющие слова, понятия, конструкции языка)
- P - множество правил вывода (продукций) грамматики
- S - целевой (начальный или стартовый) символ грамматики

Определения
«Формальная грамматика G»

- грамматика описывает множество правильных цепочек символов над заданным алфавитом.

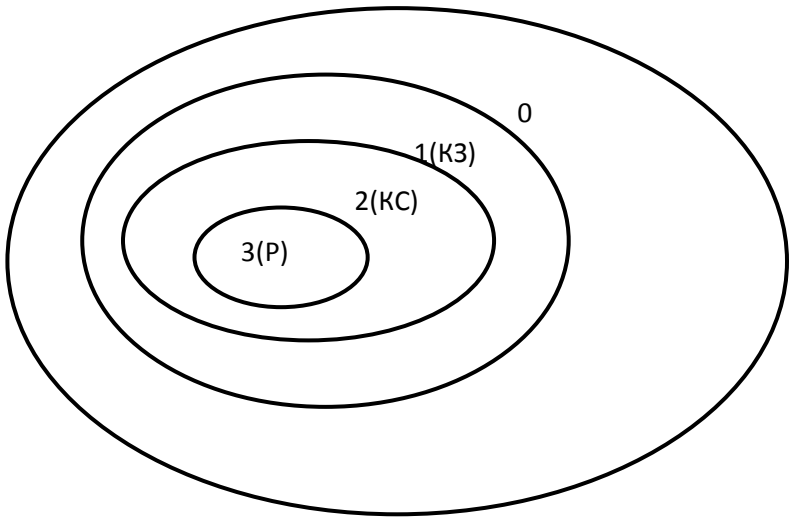
Способы описания формального языка

- простым перечислением цепочек формального языка
- порождающей формальной грамматикой языка (формальной грамматикой)

Способы задания грамматик: аналитическая форма, форма Бэкуса-Наура, синтаксическая диаграмма.

8. **Иерархия Хомского**

Соотношения грамматик в иерархии Хомского



$$G_0 \supset G_I \supset G_{II} \supset G_{III},$$

где $G_0, G_I, G_{II}, G_{III}$ - множества грамматик типа 0, 1, 2 и 3.

1) любая регулярная грамматика является контекстно-свободной грамматикой;
2) любая контекстно-свободная грамматика является контекстно-зависимой грамматикой;
3) любая контекстно-зависимая грамматика является грамматикой типа 0.

Иерархия Хомского определяет 4 типа грамматик

- Грамматика типа 0 – неограниченные грамматики
- Грамматика типа 1 – контекстно-зависимые грамматики
- Грамматика типа 2 – контекстно-свободные грамматики (правила вида N–>a)
- Грамматика типа 3 – регулярные грамматики

Формальные языки классифицируются по типу порождающих их грамматик.

Определения

G – формальная грамматика и L(G) – формальный язык порождаемый грамматикой G. Тогда язык L(G) это:

- множество всех терминальных цепочек, формально выводимых грамматикой G

Определение порождающей грамматики

- Четверка G=(T, N, P, S)

Контекстно-свободная грамматика в форме Грейбах:

Задана КС-грамматика **G=(T, N, P, S),**

где a – цепочка, состоящая из терминалов и нетерминалов,
t – цепочка терминалов,
n – цепочка нетерминалов.

Тогда вид правил грамматики G в форме Грейбах:

- N–>ta

Пример.

Дана грамматика G = ({yes,no},{S,A,B},P,S),

где P состоит из правил:
S –> A | B; A –> yes; B –> no.

Тогда цепочки, порождаемые грамматикой:

1) yes, no

Регулярное выражение

- описывает множество цепочек – формальный язык.

Для записи регулярного выражения используются метасимволы. Множество цепочек описанных регулярным выражением называется регулярным множеством (или регулярным языком).

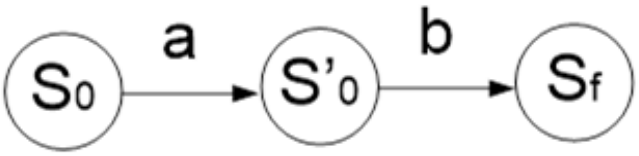
Символы, применяемые для описания регулярных выражений, называются **метасимволами** или **символами-джокерами**, например: джокерами являются символы: $*, ^{+}, +, (,), \varnothing$.

Примеры.

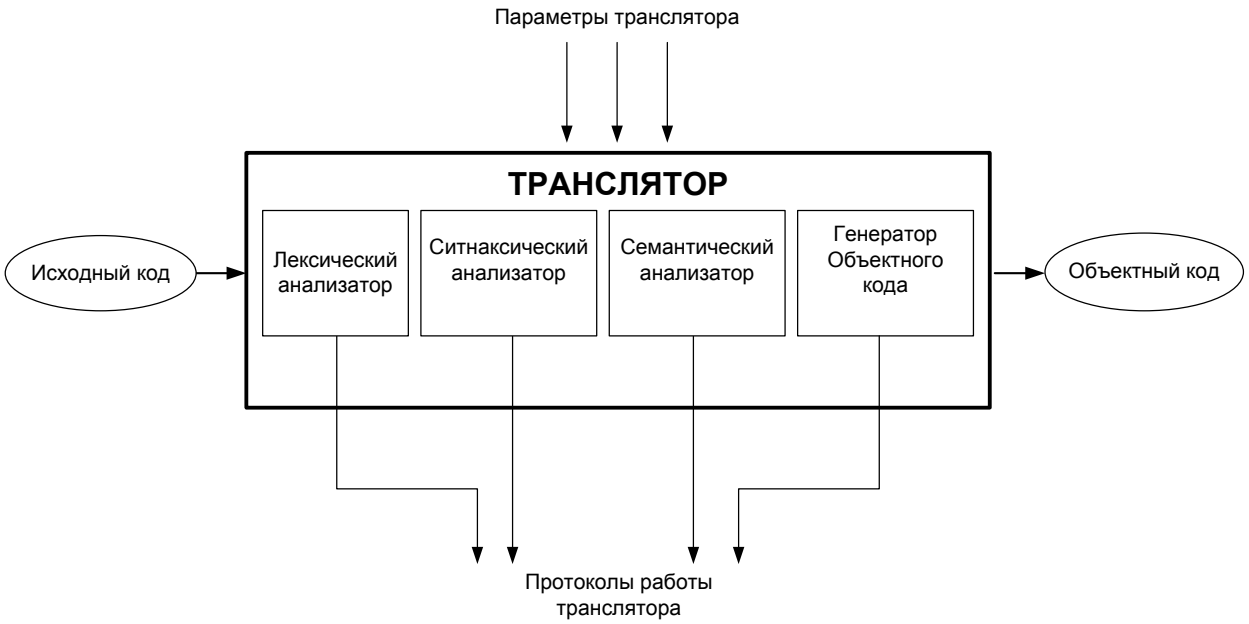
1) регулярные выражения и множества описанные ими:

	Регулярное выражение	Множество	Описание
1	a	a	множество из одного символа
2	a+b	a,b	множество из двух символов
3	a+b+c	a,b,c	множество из трех символов
4	a^+	a, aa, aaa, aaaa...	повторяется 1 и более раз (исключая λ)
5	a*	λ, a, aa, aaa, aaaa,...	повторяется 0 и более раз (включая λ)
6	ab	ab	Конкатенация символов
7	ab+cd	ab, cd	…
8	ab^+c	?	

Граф конечного автомата для регулярного выражения ab ???


a
b
S0
S'0
Sf

10. **Структура транслятора**:



Структура транслятора: фазы трансляции

- лексический анализ
- синтаксический анализ
- сематический анализ.
- генерация кода.

Последовательность обработки исходного текста транслятором

- лексический анализ
- синтаксический анализ
- семантический анализ
- генерация кода

Лексический анализ

- первая фаза трансляции.
- входной поток – поток символов исходного кода программы.
- выходной поток – таблица лексем и таблица идентификаторов
- основная задача лексического анализатора – сканирование исходного кода программы для получения на выходе лексем
- выполняется программой (входящей в состав транслятора), называемой лексическим анализатором (сканером).
- взаимодействие лексического и синтаксического анализаторов: последовательное или параллельное.

Вопросы

Структура транслятора. Назовите необязательную фазу трансляции

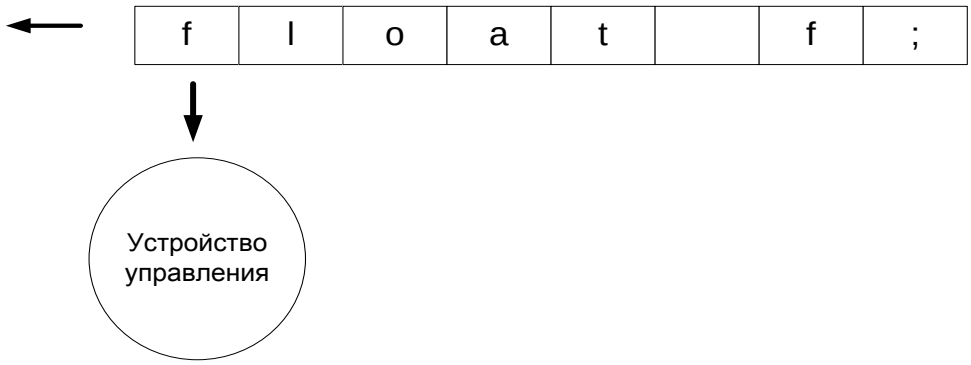
- Оптимизация кода

Назовите вторую фазу трансляции ???

Результатом успешного лексического разбора является

- таблицы лексем и идентификаторов

Схема работы лексического анализатора



Определение КА.

КА это пятерка $M = (S, I, \delta, s_0, F)$,
где
S – **конечное множество состояний устройства управления**;
I – **алфавит входных символов**;
δ – **функция переходов**, отображающая $S \times (I \cup \{\lambda\})$ в множество подмножеств S: $\delta(s,i) \subset S, s \in S, i \in I$;
$s_0 \in S$ – **начальное состояние устройства управления**;
$F \subseteq S$.– **множество заключительных (допускающих) состояний устройства управления**.

Определение конечного автомата:

- Пятерка $M = (S, I, \delta, s_0, F)$
- S – множество состояний
- I – алфавит входных символов
- δ – функция переходов автомата
- s_0– начальное состояние автомата
- ***F*** – множество конечных состояний автомата

Определение детерминированного конечного автомата:

- автомат, не содержащий пустых дуг, который на каждом такте работы может перейти точно в одно состояние
  или
- автомат, который переходит из любого состояния по любому символу точно в одно состояние

Определение недетерминированного конечного автомата:

- автомат может перейти в одно из нескольких возможных состояний, есть переходы по пустой цепочке

Синтаксический анализ:

- фаза трансляции, выполняемая после фазы лексического анализа
- вторая фаза трансляции
- основная фаза трансляции

Назначение синтаксического анализа

- распознавание синтаксических конструкций и формирования промежуточного кода.

Синтаксический анализатор

- часть компилятора, выполняющая синтаксический анализ.
- входом для синтаксического анализа является таблица лексем (токенов) и таблица идентификаторов.
- выходом – дерево разбора.

Синтаксический анализатор

- входной поток – таблица лексем и таблица идентификаторов
- выходной поток – дерево разбора

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) если нет ошибок, формирование дерева разбора.

Дерево синтаксического разбора это:

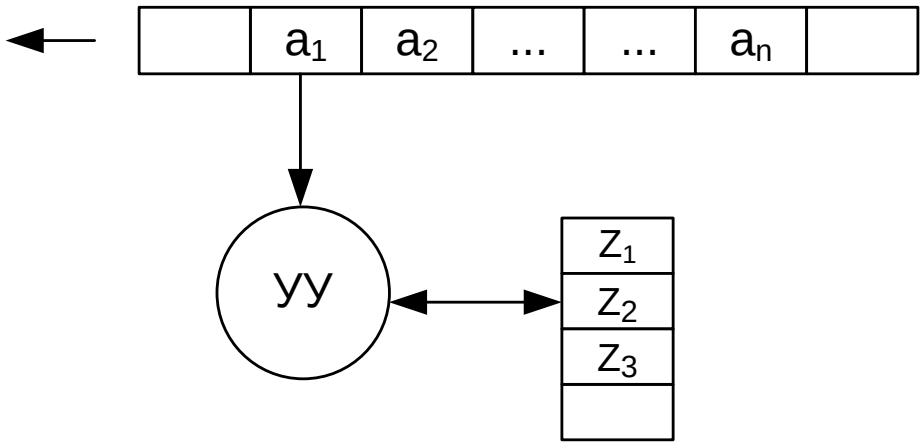
- последовательность правил формальной грамматики, приводящая к успешному разбору текста

В рамках синтаксического анализатора, применяют грамматики типа 2 – контекстно-свободные грамматики, правила которой имеют вид $A \to \alpha$

Правила контекстно-свободных грамматик.

Грамматики типа 2: $G_{II} = \langle T, N, P, S \rangle$ – контекстно-свободные грамматики, правила которых имеют вид: $A \to \alpha$, где $A \in N$, $\alpha \in V^*$, $V = N \cup T$ - словарь грамматики G_{II}.

11. Распознаватели контекстно-свободных языков – автоматы с магазинной памятью (МП-автоматы). Схема:



Определение конечного автомата с магазинной памятью:

- Семерка M=<Q,V,Z,δ,q_0,z_0,F>
- Q – множество состояний
- V – алфавит входных символов
- Z – специальный алфавит магазинных символов
- δ – функция переходов автомата
- q_0– начальное состояние автомата
- z_0– начальное состояние магазина (маркер дна)
- F – множество конечных состояний автомата

Для построения МП-автомата необходимо привести контекстно-свободную грамматику к одной из нормальных форм: к **нормальной форме Хомского** или **нормальной форме Грейбах**.

12. **Нормальная форма Грейбах:**

Контекстно-свободная грамматика $G = \langle T, N, P, S \rangle$ имеет нормальную форму Грейбах, если она не является леворекурсивной (не содержит леворекурсивных правил) и правила P имеют вид:

1) $A \to a\alpha$, где $a \in T, \alpha \in N^*$;
2) $S \to \lambda$, где $S \in N$ - начальный символ, если есть такое правило, то S не должен встречаться в правой части правил.

13. **Основная задача оптимизации кода.**

- преобразование промежуточного кода исходной программы с целью улучшения его характеристик по одному или нескольким параметрам.

14. **Генератор кода**

- Четвертая фаза трансляции
- входной поток - промежуточное представление исходной программы и таблица идентификаторов
- выход - целевая программа на машинном языке

Основная задача генератора кода.

- преобразование кода из промежуточного представления в эквивалентную целевую программу.

15. **Код программы на MASM**

```
   .586
   .model flat, stdcall
   includelib kernel32.lib
   ExitProcess        PROTO   :DWORD
   .stack 4096
   .const
   .data
    aa  dword  12345678h
    bb  dword   1111h

   .code
 main PROC

    mov eax, aa
    mov ebx, bb
    cmp eax, ebx

    je  result1
    ja  result2
    jb  result3
    mov ecx, 0
    jmp result
result1:
    mov ecx, 1
    jmp result
result2:
    mov ecx, 2
    jmp result
result3:
    mov ecx, 3
result:
    push  0
    call ExitProcess
    main ENDP
end main
```

Найдите в программе на MASM:

директива ассемблера, определяющая систему команд процессора
начало сегмента стека
имя точки входа программы
начало сегмента кода
начало сегмента констант
начало сегмента данных
метка
команду, определяющая начало процедуры
команду, определяющая конец процедуры
используемое соглашение о вызовах
команды, выделяющие память для хранения 8-разрядных целых значений
команды, выделяющие память для хранения 16-разрядных целых значений
команды, выделяющие память для хранения 32-разрядных целых значений
инструкция пересылки данных
инструкция, которая помещает данные в стек

инструкция, извлекающая данные из стека
инструкция, вызывающая процедуру
команда вычитания
команда сложения
команда безусловного перехода
регистры общего назначения

Вопросы

На каком этапе создания exe-модуля используются файлы статической библиотеки?

Дайте определение детерминированного конечного автомата