1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт прикладной математики и механики

**Высшая школа кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 1-2**

1. «**Создание языка операций с полиномами**»
2. по дисциплине «Формальные грамматики и теория компиляторов»
3. Выполнил
4. студент гр. 3651003/70801 Гасанов Э.А.

<*подпись*>

1. Преподаватель
2. ассистент Семьянов П.В.

<*подпись*>

1. Санкт-Петербург
2. 2020

# Цель работы

Цель данной лабораторной работы – научиться проектировать и реализовывать собственный формальный язык.

# Задание

1. На базе калькулятора полиномов формализовать работу с полиномами, сделав работу с таким языком удобным для всех.
2. Продуманный синтаксис. Для максимального удобства синтаксис должен быть максимально похож на математический.
3. Появляются полиномы от разных переменных (x, y, z и т.п.). Если в выражении участвует одна переменная, то все должно посчитаться без ошибок. Если в выражении появляются разные переменные, например, то вашему желанию можно или выводить ошибку, или же посчитать правильный результат.
4. Появляются переменные, которым можно присваивать полиномы.
5. Программа на вашем языке должна помещаться во входной файл, откуда она считывается и исполняется.
6. Появляются развернутые сообщения об ошибках с указанием номера строки, где они произошли. Как минимум, должны появиться по 2-3 сообщения для каждого типа возможных ошибок: лексические, синтаксические, семантические.

# Принципы, заложенные в дизайн языка полиномов

При создании языка полиномов за основу было взято два главных принципа: простота и интуитивность.

Интуитивность языка связана прежде всего с его продуманным приближенным к математическому синтаксису. Например, чтобы перемножить два полинома, не нужно использовать символ \* в качестве умножения. А так же все коэффициенты, будь то коэффициенты перед переменной (2x) или коэффициент перед мономом ( 2(x-1) ) должны записываться как есть, соответствуя общепринятой математической записи.

Простота заключатся в простых математических действиях (не реализовано деление) и в возможности оперировать только одной переменной x/y/z/w, иначе идентификатор не будет распознан и это будет лексическая ошибка.

# Описание грамматики языка полиномов

program :

program quest

| quest

;

quest: expr NEWLINE

| NEWLINE

;

Expr : Tm

| - Tm

| Expr+Tm

| Expr- Tm

;

Tm : Mon

| (Expr)

| Tm(Expr)

;

Mon : Int\_opt X Pow\_opt

| NUMBER

;

Int\_opt : NUMBER

|%empty

;

Pow\_opt: POWER NUMBER

| %empty

;

Pow\_opt: POWER NUMBER | %empty;

В этом утверждении грамматики POWER - это циркумфлекс (^) означающий степень, NUMBER -любая целочисленная степень(пропускал из лекса с помощью union с уточнением int). Или степени может не быть, поэтому есть опция %empty(она же ε- эпсилон). Тогда степень установится в единицу.

Int\_opt : NUMBER| %empty;

В этом утверждении NUMBER - любое целочисленное значение, которое может находиться перед икс или у икс может не быть коэффициента (опция %empty) - устанавливается в единицу.

Mon: Int\_opt x Pow\_opt | NUMBER;

В этом утверждении описывается моном, где у икс может быть коэффициент и степень. Или внутри этого монома есть число без переменной.

Tm : Mon|(Expr)|Tm(Expr);

Expr: Tm|-Tm|Expr+Tm|Expr-Tm;

Эти два утверждения нужны,чтобы найти моном в выражении. "Поймать" знак у значения перед скобкой монома. И определить действие, находящееся внутри скобок. \*(Сноска в интересные особенности)

Expr с NEWLINE и просто NEWLINE используются для выражения и перехода на новую строку и просто перехода на новую строку соответственно.

Левая рекурсия позволяет отцеплять от групп команд отдельные команды, и обрабатывать одна по одной. Так же существует правая рекурсия. Однако этот вариант дороже обходится. Если это первое правило, тогда YACC приходится держать все команды на стеке, что занимает большой объем памяти. Поэтому для анализа больших выражений, например целых файлов, следует использовать левую рекурсию.

**Отдельные интересные правила грамматики:**

Любой моном запишется в Tm, чтобы “связаться” с остальным выражением, получить коэффициент в соответствии с общепринятыми математическими правилами записи полиномов, а также получить знак(отрицательный), если необходимо.

Когда понадобится умножение мономов один из мономов будет Tm , а другой expr, то есть они вернутся из правила Expr: в Tm: , где по правилу Tm(Expr)

Чтобы рассмотреть детально моном мы используем Int\_opt : NUMBER| %empty.

# Представление полиномов и реализация операций над ними

Продемонстрируем представление полиномов в памяти и некоторые операции на примере: (16x^2+34x)(81x^3-7)

Все полиномы для подсчёта хранятся в целочисленном массиве. Сначала производится операции внутри мономов. Например, в мономе (16x^2+34х) правилом Int\_opt : NUMBER| %empty. проверяется коэффициент, он равен 16, затем правилом Pow\_opt: POWER NUMBER | %empty узнаём степень переменной x(=2). Затем всё собирается в моном правилом Mon: Int\_opt x Pow\_opt | NUMBER.

В конец массива запишется коэффициент переменной, а перед ним нули в количестве, зависящим от степени этой переменной: [0,0,16]

Аналогично происходит и с 34х, и его массив выглядит [0,34].

Моном, теперь представленный как Tm, теперь связан с Expr. Видим сумму и складываем по правилу Expr+Tm. Тогда массив принимает вид: [0,34,16]

Вторая скобка имеет такой же алгоритм работы со степенью и переменной и получает вид:[0,0,0,81]

Затем по правилу Expr-Tm записывается 7 со своим знаком минус – [-7,0,0,81]

Теперь необходимо умножить два монома, лежащих в Tm и Expr по правилу Tm(Expr).

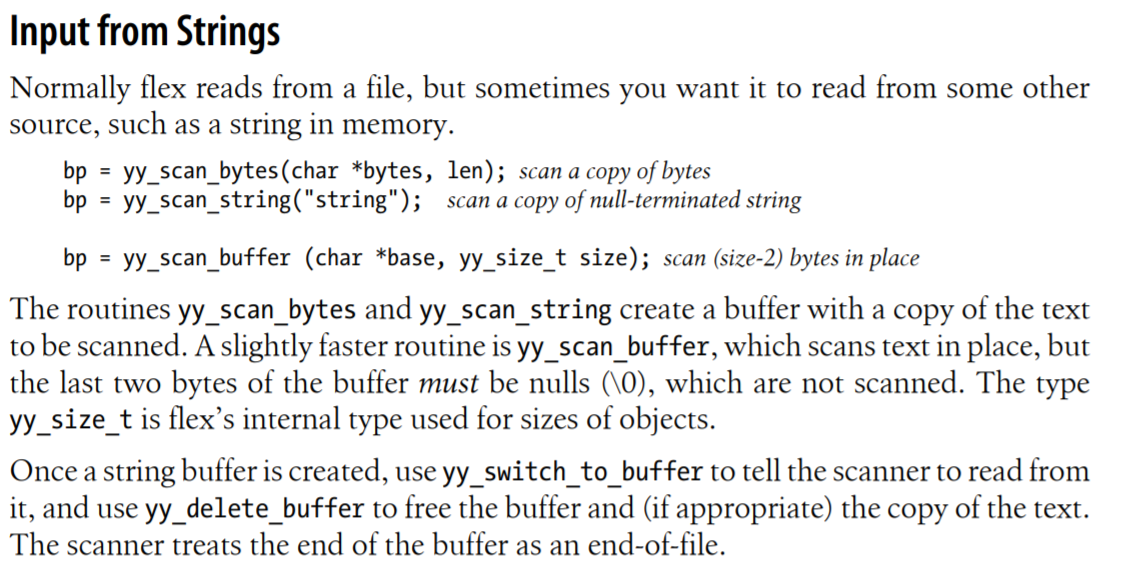
Берется первый элемент из Tm, то есть 0 и он умножается на все элементы Expr ([-7,0,0,81]). И получаем [0,0,0,0].

Затем 34(второй элемент из Tm) умножается на 7 и пишется на второе место в массиве. То есть массив принимает вид [0,-238,0,0]

И после на 81 и в результате 2754 записывается последним элементом массива, то есть [0,-238,0,0,2754].

Затем 16 умножается на -7 и запишется на 3 место и получится [0,-238,-112,0,2754]. Затем 81 умножился на 16. И получилось [0,-238,-112,0,2754,1296]

Чтобы придать вид полиному, нужно дописать в соответствующие места нужные переменные со степенями, получилось 1296x^5+2754x^4-112x^2-238x

Для того, чтобы подавать на вход yyparse строку из памяти, воспользуемся функцией yy\_scan\_string,yy\_scan\_buffer из книги Джона Левина Flex&Bison 

Врезка 1, функции для подачи в yyparse

# Выводимые ошибки и примеры входных цепочек

**Синтаксические ошибки:**

**Неправильная запись переменной**

$ABC=(x^7+x^4-334x^2-51)(x^4-33x^2-5)

В переменной должно быть только $ и одна заглавная буква

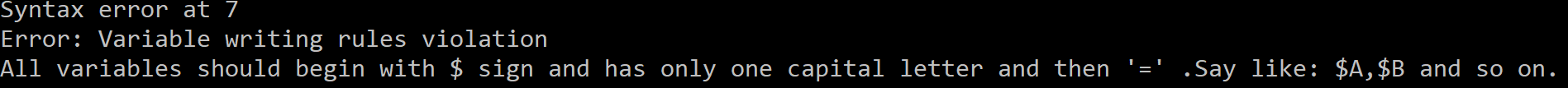


Рис.1, синтаксическая ошибка в записи переменной

$c=(x^7+x^4-334x^2-51)(x^4-33x^2-5)

В переменной должно быть только $ и одна заглавная буква

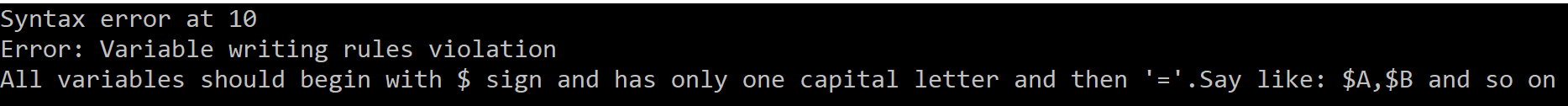


Рис.2, синтаксическая ошибка в записи переменной

$ A=(x^7+x^4-334x^2-51)(x^4-33x^2-5)

Никаких пробелов в именах переменных

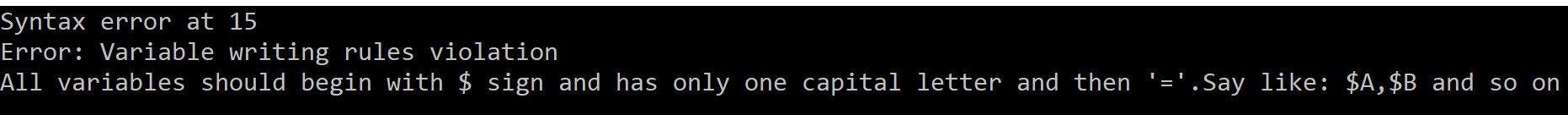


Рис.3, синтаксическая ошибка

~A=(x^7+x^4-334x^2-51)(x^4-33x^2-5)

Начинаться должно только с $

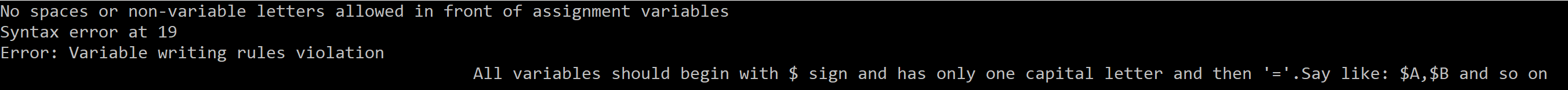


Рис.4, синтаксическая ошибка

**Синтаксические ошибки в записи полиномов**

Любые пробелы в полиноме - синтаксическая ошибка

$A=(x^7+ x^4-334x^2-51)(x^4-33x^2-5)

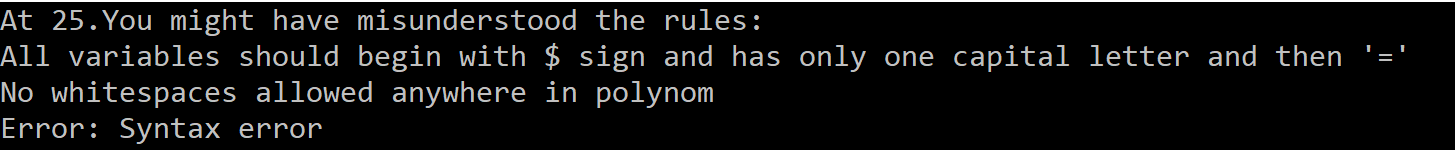


Рис.5, синтаксическая ошибка

$A=(x)\*(x)

Никаких звёздочек в умножении не используется

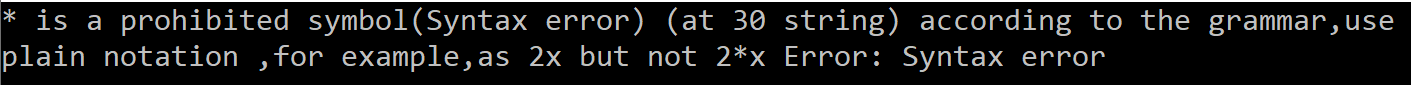


Рис.6, синтаксическая ошибка в полиноме

**Синтаксическая ошибка в комментариях**

Любой комментарий должен начинается строго с начала строки , без каких-либо символов вначале

lol// этот комментарий записан неверно



Рис.7, синтаксическая ошибка в комментарии

**Семантические ошибки:**

$A=$L+$K

Если мы попытаемся воспользоваться переменной, которая не была инициализирована полиномом – получим семантическую ошибку

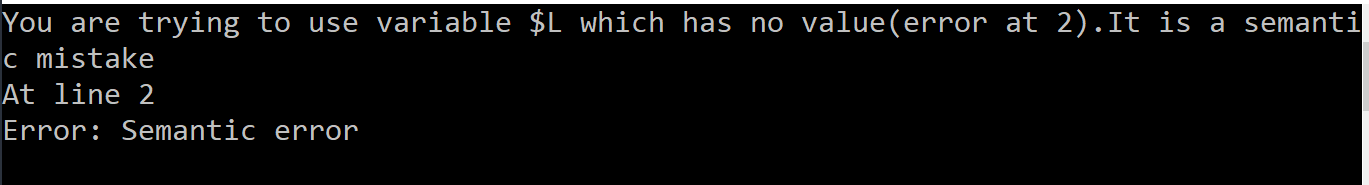


Рис.8, семантическая ошибка

$B=($A)(33.5x)

print=$B

Если мы попытаемся использовать переменную с плавающей точкой, тогда мы получим предупреждение о том, что результат может быть неточным.

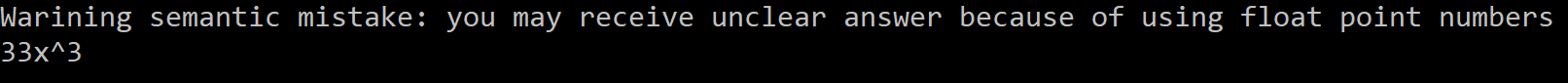


Рис.9, семантическая ошибка

**Лексические ошибки:**

Если используются две разные переменные в одном полиноме, то это ошибка:

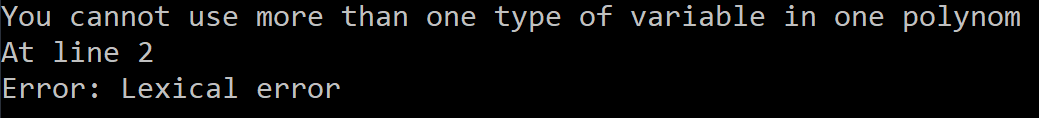


Рис.10, лексическая ошибка с разными переменными в одном полиноме

Неверное использование print

printf=B

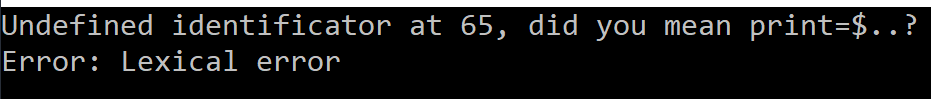


Рис.11, лексическая ошибка с print

Деление недоступно



Рис.12, недоступное деление

# Ошибки, которые раньше выводил YACC

%option noyywrap решил проблему с EOF: он возвращает единицу, чтобы остановить процесс лексинга, когда достигнут EOF.

yy\_switch\_to\_buffer();

yy\_delete\_buffer();

Эти функции вызывали implicit declaration у gcc и чтобы от этих предупреждений избавиться пришлось их явно объявлять как прототипах этих функций:

void yy\_switch\_to\_buffer(void\*);

void \*yy\_scan\_string(const char\*);

void \*yylex\_destroy();

# Результаты

Мы создали свой собственный язык полиномов, который имеет систему обработок ошибок, простой и понятный интерфейс.

# Выводы

Мы научились формализовывать грамматику для поставленной задачи и научились её реализовывать, а так же поняли механизм работы компиляторов.