ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ им. Н.Э. БАУМАНА»

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА И КОМПЬЮТЕРНЫЕ
ТЕХНОЛОГИИ»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ
ПО ДИСЦИПЛИНЕ

«Конструирование компиляторов»

НА ТЕМУ:

# Интерпретатор формул TEX

Москва, 2017 г.

# Содержание

# 1 Постановка задачи

Цель работы — создать инструмент, который позволяет выполнять расчеты и одновременно их документировать.

Для этого: реализовать препроцессор для TEX, который, принимая на вход документ формата TEX с формулами вида x := выражение и x = \placeholder{}, на выходе порождает новый документ, в котором на месте заменителей (placeholder'ов) находятся вычисленные значения. Фактически получается аналог пакетов Mathcad или S-Math Studio.

Должны поддерживаться как минимум 3/4 из следующих возможностей:

- Арифметические выражения
- Присваивания
- Вектора и матрицы
- Арифметические выражения с матрицами
- Стандартные математические функции
- Объявление функций
- Циклы
- Ветвления
- Графики функций
- Проверка размерностей
- Функции высших порядков
- Оператор суммирования
- Области видимости

# 2 Возможности языка

В TEXформулы можно записывать разными способами:

- между $ и $ — формула записывается внутри текста.
- между $$ и $$ — формула записывается в отдельной строке (выключная формула).
- в окружении equation — формула записывается аналогично формуле в $$.

Препроцессор должен обрабатывать выделенные фрагменты текста с формулами из документа. Поэтому формулы для обработки препроцессором будут записываться в окружении preproc. Это значит, что они должны быть заключены между ключевыми словами \begin{preproc} и \end{preproc}. Окружение preproc для TEX по умолчанию объявлено как указано в Листинге 1.

Листинг 1: окружение preproc

```
\newenvironment{preproc}
{\begin{equation*}\begin{array}{l}}
{\end{array}\end{equation*}}
```

Оно представляет собой окружение equation* (* означает, что формула не нумеруется) с array из одного выравненного по левой стороне столбца внутри. Таким образом, в это окружение можно записать много формул, разделенных символами переноса строки: \\.

В общем случае выражения на языке препроцессора задаются так:

S ::= \begin{preproc} ( Operator \\? )* \end{preproc}

Здесь Operator — некоторая конструкция на языке препроцессора.

## 2.1 Лексический состав языка

Идентификаторами считаются последовательности из букв и цифр, начинающиеся на букву, и которые могут заканчиваться на индекс с командой \text{} (Листинг 2).

Числа могут быть целые или с плавающей запятой. При этом не допускаются ведущие нули (Листинг 2).

Листинг 2: числа и идентификаторы

```
Ident ::= Name ( '_\text{' Any '}' )?
Name ::= Char ( Char | Digit )*
Char ::= 'A' | ... | 'Z' | 'a' | ... | 'z'
Digit ::= '0' | ... | '9'
Any ::= любой набор символов
Number ::= ( '0' | ( '1' | ... | '9' ) Digit*)( '.' Digit* )?
Keyword ::= '\' Ident
```

Ключевые слова представляют собой идентификаторы с символом \в начале — в TEX-е ключевые слова обозначаются так же. Ключевые слова, которые требуются для работы препроцессора, включают как TEX-овские (точнее, их подмножество), так и дополнительно объявленные (о них рассказано подробнее в 2.2).

## 2.2 Синтаксические конструкции языка

Рассмотрим подробнее синтаксические конструкции на языке препроцессора, обозначенные как Operator:

```
Operator ::= Def | Block | If | Cases | While | Expr | Graphic
```

Конструкцией может быть объявление, блок выражений, выражение с условным оператором (с одной, двумя ветвями или с несколькими условиями), цикл, выражение и график.

Выражение задает некоторое действие над операндами. Операторами в выражении служат арифметические операторы, операторы сравнения и логические операторы. Каждый оператор имеет приоритет (Таблица 1).

Таблица 1: Приоритет операторов

| Оператор | Приоритет |
|---|---|
| Степень | 150 |
| Унарный плюс, унарный минус, логическое отрицание | 130 |
| Умножение, деление | 120 |
| Сложение, вычитание | 110 |
| Больше, больше или равно, меньше, меньше или равно | 100 |
| Равенство, неравенство | 90 |
| Логическое и | 80 |
| Логическое или | 70 |
| Присваивание | 60 |

Оператор '=' используется как оператор вывода, если после него идет ключевое слово \placeholder. Оператор вывода указывает препроцессору, что результат вычисления выражения слева от него нужно записать как аргумент \placeholder{}. Ключевое слово \placeholder объявлено как функция TEX, которая отображает свой аргумент:

```
\newcommand{\placeholder}[1]{#1}
```

Ключевое слово \graphic — функция от трех аргументов, с помощью которой задается вывод двумерного графика:

```
\newcommand{\graphic}[3]{
\begin{tikzpicture}\begin{axis}[
xmajorgrids,ymajorgrids,
axis lines=middle,
x label style={at={(axis description cs:0.5,-0.1)},
                anchor=north},
ylabel style={above left},
xlabel=\text{#1(#2)},]
\addplot[thin, mark = none] coordinates{#3};
\end{axis}\end{tikzpicture}}
```

Её первый аргумент — имя функции, второй — список аргументов, третий — координаты точек графика. Если у функции больше одного аргумента, они

перечисляются через запятую. Принимается только один переменный параметр. Переменный параметр задается ключевым словом \range. Функция \range — функция от трех аргументов: шаг, начальное и конечное значения диапазона:

```
\newcommand{\range}[3][]{%
\ifthenelse{\isempty{#1}}{[#2:#3]}{[#2:#3:#1]}}
```

Возвращаемый ею вектор заполнен значениями из указанного отрезка с соответствующим шагом. Препроцессор заполняет третий аргумент \graphic автоматически, вычисляя значения функции во всех точках диапазона.

В арифметических выражениях используются операторы сложения '+', вычитания '-', умножения '*', деления '/', возведения в степень '^'. Один и тот же оператор может обозначаться разными способами. Например, оператор умножения может быть задан не только как '*', но и как '\cdot', '\times'. Деление может обозначаться разными конструкциями. С помощью оператора '/':

```
x / y
```

И с помощью оператора '\frac {...} {...}':

```
\frac{x}{y}
```

Степень для '^' должна указываться в фигурных скобках. Операндом может быть число, переменная, вызов функции, ключевое слово, но не матрица. В грамматике это не отражено, и поэтому проверяется уже после синтаксического анализа. Под ключевым словом имеется в виду слово, начинающееся с \, например, \cos. Для транспонирования матриц определена функция \transp{}.

Выражения задаются так:

Листинг 3: выражения

```
Expr ::= Expr1 (( '\lor' | '\wedge' ) Expr)?
Expr1 ::= Expr2 (( '\land' | '\vee' ) Expr1)?
Expr2 ::= Expr3 ( '=' ( Expr2 | '\placeholder{' Any '}' ) |
                  '\neq' Expr2)?
Expr3 ::= Expr4 ( Cmp Expr3)?
Cmp := '<' | '>' | '\leq' | '\geq'
Expr4 ::= Expr5 (( '+' | '-' ) Expr4)?
Expr5 ::= Expr6 (( '*' | '\cdot' | '\times' | '/' ) Expr5)?
Expr6 ::= ( '+' | '-' | '\neg' )? Expr7
Expr7 ::= Elem ( '^''{' Expr '}' )?
```

```
Elem ::= Number | Ident Index? Arglist? | '(' Expr ')' |
         Keyword Arglist? | Matrix | Range |
         '\frac{' Expr '}''{' Expr '}' | '\transp{'Expr'}'
Arglist ::= '(' ( Expr ( ',' Expr )* )? ')'
```

Для объявления используется оператор присваивания ‘:=‘. Оператор присваивания связывает идентификатор с числом, функцией или матрицей.

Слева от оператора присваивания должен стоять идентификатор. Идентификатор считается именем функции, если после него идет список аргументов (возможно, пустой). В противном случае это идентификатор матрицы или переменной. При объявлении функции в списке её аргументов допускаются только идентификаторы. Если слева от оператора ':=' стоит ключевое слово \begin{pmatrix}, то это объявление обычной матрицы, а если \range — вектора-строки.

Так же оператор ':=' можно использовать для присваивания элементу матрицы некоторого значения. Синтаксис объявления приведен в Листинге 4.

Листинг 4: объявления

```
Def::= Ident ( Deflist? | Index ) ':=' Operator
Deflist ::= '(' ( Ident ( ',' Ident )* )? ')'
Matrix ::= '\begin{pmatrix}'
           Line ( '\\' Line )* '\\' ?
           '\end{pmatrix}'
Line ::= Expr ( '&' Expr )*
Index ::= '_'( Ident | Number | '{' Expr ( ',' Expr )? '}' )
Range ::= '\range'( '[' 'Expr' ']' )?'{'Expr'}''{'Expr'}'
```

Доступ к элементу матрицы происходит через обращение по индексу. Элементы матрицы нумеруются с (0,0). Матрица должна содержать хотя бы один элемент. Если матрица представляет собой вектор-столбец или вектор-строку, то она интерпретируется как вектор. К элементу вектора можно обратиться, указав один индекс: $x_i$, вместо $x_{\{i,0\}}$ или $x_{\{0,i\}}$.

Идентификатору присваивается синтаксическая конструкция. Идентификатору переменной присваивается значение, полученное в результате вычисления синтаксической конструкции. Если это не выражение, а, например, цикл, переменной будет присвоено значение, полученное с последней итерации цик-

ла. Идентификатору функции присваивается сама синтаксическая конструкция. Возвращаясь к примеру с циклом, функции будет присвоено выполнение этого цикла.

Блок синтаксический конструкций объявлен как окружение block:

Листинг 5: окружение block

```
\newenvironment{block}
{\begin{array}{|l}}
{\end{array}}
```

В блоке последовательно выполняются предложения на языке препроцессора. Результат последнего выполненного выражения считается результатом выполнения блока. Если в блоке нет ни одного предложения, то результатом его выполнения считается 0. Синтаксис для записи блока арифметических выражений:

```
Block::= '\begin{block}'( Operator '\\'? )*'\end{block}'
```

Ветвление задается ключевыми словами \when, \otherwise и окружением caseblock, а циклы — командой \while. Эти команды и окружение объявлены следующим образом:

Листинг 6: ключевые слова

```
\newcommand{\ifexpr}[1]{{\bf if}\enspace#1\enspace}
\newcommand{\when}{\enspace{\bf when}\enspace}
\newcommand{\otherwise}{\enspace{\bf otherwise}\enspace}
\newcommand{\while}[1]{{\bf while}\enspace#1\enspace}
\newenvironment{preproc}
{\begin{equation*}\begin{array}{l}}
{\end{array}\end{equation*}}
```

Соответственно, синтаксис для этих команд:

Листинг 7: синтаксис ветвлений и цикла

```
If ::= '\ifexpr{'Expr'}''\\'? Operator '\\'?
       ('\otherwise' Operator )?
Cases ::= '\begin{caseblock}'( Alt '\\' )+
          ('\otherwise' Operator '\\'? )?'\end{caseblock}'
Alt ::= Operator '\when' Expr
```

Окружение caseblock отличается от block тем, что внутри него идёт последовательность выражений и условий, когда функция возвращает значение того или иного элемента этой последовательности. В окружении caseblock должна буть записана хотя бы одна альтернатива. Выражение по умолчанию не обязательно и задается командой \otherwise. Если в результате выполнения caseblock-а не было выполнено ни одно из условий, он возвращает 0.

Полный список правил грамматики приведен в приложении А.

# 3 Реализация

## 3.1 Работа с файлами

Препроцессор работает следующим образом: ему на вход подается корректный документ формата tex, а на выход препроцессор отправляет корректный документ формата tex с заполненными placeholder-ами. Файл, который генерирует препроцессор, можно сохранить в отдельный файл или заменить им входной файл.

Программа сканирует входной файл и ищет строки, содержащие подстроки \begin{preproc} и \end{preproc}. Строки, находящиеся снаружи окружения preproc сразу пишутся во временный файл. Строки, находящиеся внутри окружения preproc (включая крайние строки — с \begin{preproc} и \end{preproc}) сначала обрабатываются препроцессором, и потом пишутся во временный файл. Когда входной файл полностью прочтен и не возникло никаких ошибок, программа удаляет его и меняет имя временного файла на имя входного или сохраняет полученный файл. Если во время обработки файла возникла ошибка, новый файл удаляется, и входной файл не изменяется.

Сканирование и запись файла выполняет экземпляр класса FileHandler, который инициализируется названием обрабатываемого файла. У FileHandler есть метод next(), который возвращает структуру ProgramString с координатами окружения preproc в файле и строкой с текстом внутри этого окружения. Если окружений preproc в файле несколько, то next() возвращает информацию о них по порядку. Если FileHandler не нашел preproc, метод next() вернет структуру с пустой строкой.

Если LaTeXвстречает в тексте символ %, он пропускает текст до конца данной строки. Поэтому, если подстрока \begin{preproc} или \end{preproc} идёт в строке после символа %, она игнорируется FileHandler-ом (то есть не распознается как начало или конец окружения preproc). Комментарии, которые записаны внутри окружения preproc, записываются в возвращаемую строку — иначе при перезаписи файла они бы удалялись.

## 3.2 Лексический анализ

Текст между \begin{preproc} и \end{preproc} отправляется на обработку лексическому анализатору. Результат работы лексического анализатора — вектор токенов. Лексический анализ осуществляет экземпляр класса Lexer с помощью функции std::vector<Token> program_to_tokens(ProgramString). Эта функция вызывает метод Lexer-a Token next(), возвращающий следующий найденный в строке токен. Если на этапе лексического анализа произошла ошибка, будет возвращен токен с её координатами в файле, и дальше строка обрабатываться не будет.

Класс Token имеет поля с координатами начала и конца токена в строке, поле с подстрокой, распознанной как токен, тэг токена и его значение (0 по умолчанию).

Лексический анализатор может присвоить токену следующие тэги: NONE, UADD, USUB, ADD, SUB, DIV, MUL, FRAC, POW, LPAREN, RPAREN, LBRACE, RBRACE, LBRACKET, RBRACKET, SET, COMMA, BREAK, INDEX, AMP, WHILE, IF, ELSE, WHEN, OTHERWISE, ALT, BEGIN, END, BEGINB, ENDB, BEGINC, ENDC, BEGINM, ENDM, LT, GT, LEQ, GEQ, NEQ, EQ, OR, AND, NOT, NUMBER, IDENT, KEYWORD, FUNC, ERROR, SPACE, PLACEHOLDER, TEXT, LIST, ROOT, GRAPHIC, RANGE.

Комментарии и пробелы не нужны для последующего анализа, поэтому им присваивается тэг SPACE, и они не включаются в вектор, возвращаемый функцией program_to_tokens.

Токен распознается как ключевое слово, если начинается с ‘\‘. При этом, если этот токен — \placeholder, \begin или \end, то он обрабатывается отдельно. В случае \placeholder, Lexer пропускает его аргумент. В случае \begin или \end, Lexer читает аргумент и в зависимости от его значения присваивает токену тэг: BEGINB или ENDB для аргумента ‘{block}‘, BEGINC или ENDC для аргумента ‘{cases}‘ и аналогично для BEGINM, ENDM и {pmatrix}. Другие окружения внутри preproc не поддерживаются. Им лексер присваивает токены BEGIN и END соответственно. Последовательность токенов, начинающаяся с BEGIN и оканчивающаяся на END, лексер игнорирует и в вектор токенов не включает.

## 3.3 Синтаксический анализ

Синтаксический анализ реализован построением синтаксического дерева по полученному вектору токенов. Листами в дереве могут быть идентификаторы (переменных, функций, векторов, ключевых слов), числа и списки. Вершинами могут быть унарные или бинарные операторы. Идентификаторы переменных и функций не различаются, потому что переменные рассматриваются как функции без аргументов, которые возвращают константное значение.

Можно заметить, что в правилах грамматики встречаются повторяющиеся конструкции:

```
Expr ::= Expr_1 (( '\lor' | '\wedge' ) Expr)^?
Expr_1 ::= Expr_2 (( '\land' | '\vee' ) Expr_1)^?
Expr_2 ::= Expr_3 (( '=' | '\neq' ) Expr_2)^?
Expr_3 ::= Expr_4 ( Cmp Expr_3)^?
Cmp := '<' | '>' | '\leq' | '\geq'
Expr_4 ::= Expr_5 (( '+' | '-' ) Expr_4)^?
Expr_5 ::= Expr_6 (( '*' | '\cdot' | '\times' | '/' ) Expr_5)^?
Expr_6 ::= ( '+' | '-' | '\neg' )^? Expr_7
Expr_7 ::= Elem ( ^{ Expr } )^?
```

Эти правила можно обобщить:

```
Expr ::= ( Unop )^? ( Expr ( Binop Expr )* ) | Elem
Unop ::= '-' | '+' | '\neg'\\
Binop ::= '<' | '>' | '\leq' | '\geq' | '=' | '\neq' |
          '\lor' | '\land' | '-' | '+' | '*' | '\cdot' |
          '\times' | '/' | '^' | ':='
```

Чтобы арифметические операции выполнялись в правильном порядке, соответствующим им токенам назначается приоритет — целое число. Чем больше приоритет, тем раньше должна выполниться операция. Токены, которые не являются операторами, имеют приоритет 0.

Если во время построения синтаксического дерева встречается непредвиденный символ, бросается исключение с сообщением об ошибке и координатами токена.

Дерево реализуется классом Node (Листинг 8). Поле _coord содержит координату токена в файле: строку и столбец. В зависимости от поля _tag

заполняются поля _label и _priority. Поля _coord и _tag инициализируются токеном, передающимся в конструктор Node. Поля left, right, cond, fields содержат ссылки класса Node.

Листинг 8: поля класса Node

```
class Node {
        friend class Parser;
        Coordinate _coord;
        Tag _tag = ERROR;
        std::string _label = "";
        int _priority = 0;
public:
        static name_table global;
        static replacement_map reps;
        Node *left = nullptr;
        Node *right = nullptr;
        Node *cond = nullptr;
        std::vector<Node *> fields;
        ...
}
```

Для унарных операторов используется поле right, для бинарных — right и left. На Рисунке 1 приведен пример структуры дерева с бинарными операторами.

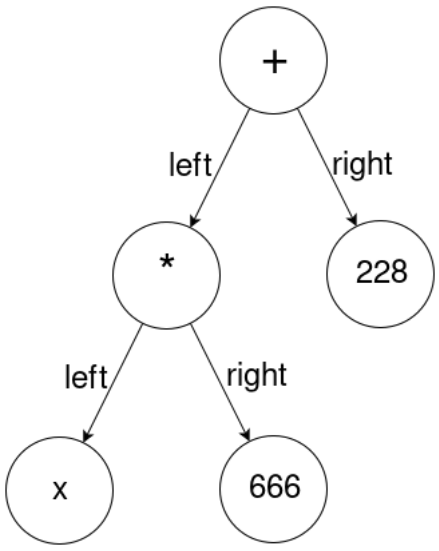



Рис. 1: Дерево с бинарными операциями

Для циклов и ветвлений добавляется cond (Рисунок 2), для блоков, функций, векторов и матриц — fields (Рисунок 3).

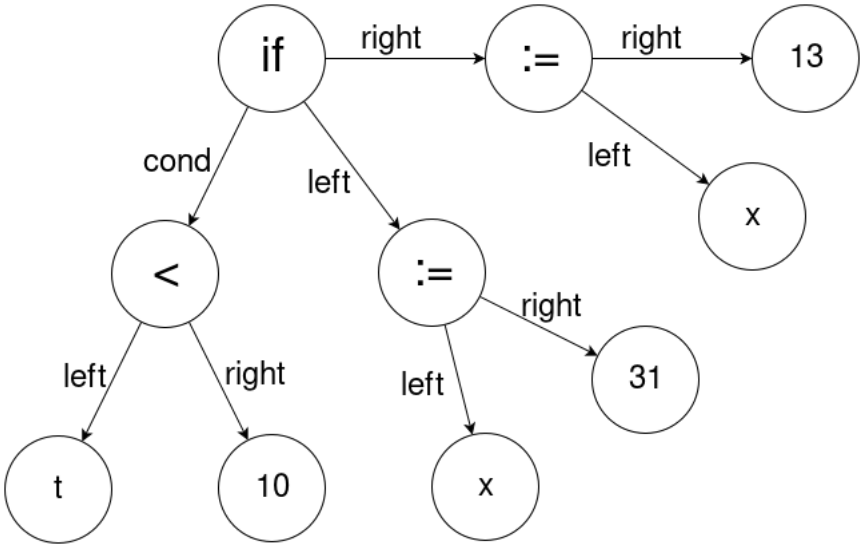



Рис. 2: Дерево с ветвлением

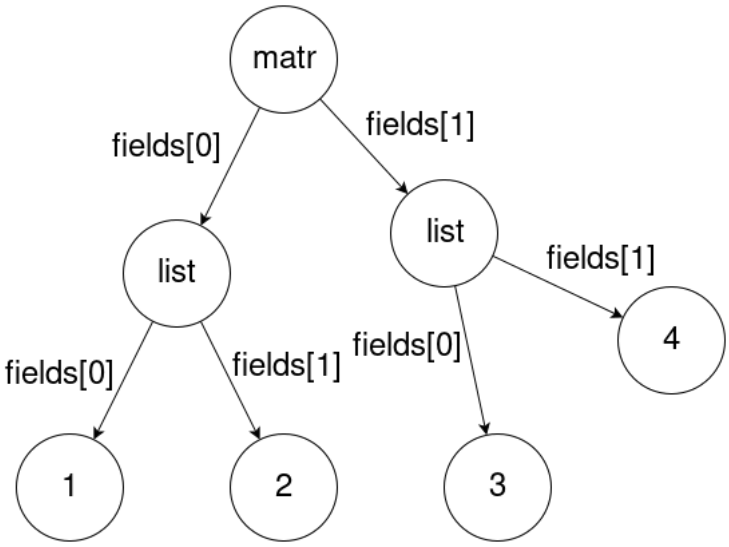



Рис. 3: Дерево матрицы

Статическое поле name_table — таблица имен, представляющая собой map с ключом — именем переменной и значением. Статическое поле replacement_map — map с ключом — координатой плейсхолдера и значением для подстановки.

Структура Parser (Листинг 9) строит синтаксическое дерево. Функцией init(std::vector<Token>&) инициализируется переменная program. Она содержит последовательность токенов, на которые лексический анализатор разбил текст внутри директивы preproc. Функции next, cur и get нужны для навигации

по program. Функция skip(Tag) проверяет, что текущий символ равен ожидаемому, и переводит Parser на следующий токен.

Функция unexpr(Token*) возвращает синтаксическое дерево, построенное для выражения с унарным оператором, блока выражений, цикла, ветвлений. Функция binexpr(Token*,Node*) возвращает синтаксическое дерево для выражения с бинарным оператором. Функции block(Tag), list(Tag), cases(), matrix(), line() — вспомогательные функции для инициализации поля fields в Node с соответствующими тегами. Функция expression(int) служит связующим звеном для unexpr и binexpr. Чтобы дерево строилось в соответствии с приоритетами операторов, в неё передается аргумент с приоритетом.

Листинг 9: структура Parser

```
typedef struct Parser {
        std::vector<Token> program;
        int i = 0;

        Token *next() { return &program[++i]; }
        Token *cur() { return &program[i]; }
        Token *get() { return &program[i++]; }
        bool skip(Tag);

        void init(std::vector<Token> &);

        Node *unexpr(Token *);
        Node *binexpr(Token *, Node *);
        Node *expression(int pr);
        std::vector<Node *> block(Tag stop = NONE);
        std::vector<Node *> line();
        std::vector<Node *> cases();
        std::vector<Node *> list(Tag close);
        std::vector<Node *> matrix();
} Parser;
```

## 3.4 Области видимости

Имена переменных и функций записываются в таблицу символов. Если имя было объявлено в окружение preproc, то оно будет записано в глобальную таблицу. Исключение составляют функции. Для каждой функции создается новая таблица символов — локальная таблица. Переменные, объявленные внутри функции, видны только внутри неё. После возврата из функции они принимают старые значения (или снова становятся неопределенными). Когда препроцессор обрабатывает идентификатор, он сначала ищет его в локальной таблице символов (если она существует), а потом в глобальной. Так же при вызове функции в локальную таблицу добавляются значения её аргументов.

## 3.5 Структуры данных времени выполнения

Чтобы обрабатывать функции, матриц и переменные единообразно, для их представления используется класс Value. Value представляет собой размеченное объединение — структуру с enum (для обозначения типа) и union (для хранения данных) (Листинг 10). Функции get_double, get_matrix, get_function в зависимости от типа возвращают значение или бросают исключение, если тип не соотвествует ожидаемому. Тип Matrix определяется как std::vector<std::vector<Value>>. Func — это структура для хранения представления функции (Листинг 11). Она содержит имена аргументов, локальную таблицу символов и тело функции типа Node*. Во время построения синтаксического дерева проверяется, что нет повторяющхся имен аргументов.

Листинг 10: класс Value

```
class Value {
public:
        typedef enum Type {
                DOUBLE, MATRIX, FUNCTION
        } Type;
private:
        Type _type;
        union {
                double _double_data;
                std::vector<std::vector<Value>> *_matrix_data;
```

```
                Func *_function_data;
        };
public:
        ...
        double get_double() const;
        Matrix &get_matrix() const;
        Func *get_function() const;
        ...
};
```

Листинг 11: структура Func

```
typedef struct Func {
        std::vector<std::string> argv;
        name_table local;
        Node *body;
} Func;
```

Такой подход позволяет хранить имена переменных, функций и матриц в одной таблице. Поэтому можно передавать именая функций в качестве аргументов функции (Листинг 12).

Листинг 12: пример функции высшего порядка

```
\begin{preproc}
f(x) := x+1\\
q(y,z) := z(y)\\
t := 10\\
q(t,f) = \placeholder{}\\
\end{preproc}
```

При этом нельзя назначить одно и то же имя переменной и функции. Ключевые слова переопределять нельзя, оператор ':=' для них не выполняется.

Таблица имен имеет тип std::map<std::string, Value>. В качестве ключа используется имя переменной, функции или матрицы. В качестве значения — экземпляр Value.

Когда синтаксическое дерево построено, у его корня вызывается метод Value exec(name_table). Этот метод обходит дерево и вычисляет данные вре-

мени выполнения. При объявлении exec записывает в указанную таблицу имен новыую переменную. Если при обходе exec находит вершину с тегом EQ, то он проверяет, если поле right этой вершины имеет тег PLACEHOLDER. Если это так, exec пишет в Node::replacement_map вычисленное Value поля left. Аналогичное действие совершается при нахождении вершины с тегом GRAPHIC.

Если во время построения дерева или его обхода произошла ошибка, то будет брошено исключение с координатами этой ошибки в файле и строкой с сообщением, в чем состоит ошибка.

После обхода дерева идет запись в файл вычисленных значений для плейсхолдеров и графиков. В плейсхолдер может быть записано число или матрица. Если в плейсхолдер писать значение функции, то это будет строка "function".

# 4 Руководство пользователя

Перед применением препроцессора к документу нужно:

1. Подготовить корректный документ в формате tex.
2. Включить в его преамбулу файл с определениями для препроцессора: \input{preproc.tex} (Приложение B).
3. Области текста для обработки препроцессором добавить в окружение preproc.

Программа препроцессора принимает один обязательный аргумент — имя входного файла, и один необязательный — имя выходного файла. Если указать только входной файл, он будет перезаписан препроцессором. Если еще указать имя выходного файла, то запись будет идти в него. Выходной файл будет отличаться от исходного только значениями в плейсхолдерах.

Так как предполагается, что исходный файл – корректный, то обработанный препроцессором документ будет корректным тоже. Поэтому получить из него pdf или dvi можно стандартным образом.

# 5 Тестирование

В Листинге 13 приведен пример объявления переменной, переменной с индексом, функции, матрицы и присваивания элементу матрицы значения. Так же показано, что можно переобъявлять идентификаторы, задавать диапазоны и строить графики.

Листинг 13: фрагмент документа Tex до обработки

```
\begin{preproc}
x := 1\\
x := 100\\
x0_\text{текст} := 228\\
f(x,y) := x\cdot (x+y)\\
v := \begin{pmatrix}
1 \\ 2 \\ x0_\text{текст}
\end{pmatrix}
v_0 := 666\\
v = \placeholder{}\\
\graphic{f}{\range{-1}{1}, 0.5}{}\\
\end{preproc}
```

Далее рассмотрим, как препроцессор должен изменять файл (Листинг 13, 14).

После обработки препроцессором такой фрагмент текста должен иметь следующий вид:

Листинг 14: фрагмент документа Tex после обработки

```
\begin{preproc}
x := 1\\
x := 100\\
x0_\text{текст} := 228\\
f(x,y) := x\cdot (x+y)\\
v := \begin{pmatrix}
1 \\ 2 \\ x0_\text{текст}
\end{pmatrix}
v_0 := 666\\
v = \placeholder{\begin{pmatrix}
```

```
666.000000\\
2.000000\\
228.000000\end{pmatrix}}\\
\graphic{f}{\range{-1}{1}, 0.5}{(-1.000000,0.500000)
(-0.900000,0.360000)
(-0.800000,0.240000)
(-0.700000,0.140000)
(-0.600000,0.060000)
(-0.500000,0.000000)
(-0.400000,-0.040000)
(-0.300000,-0.060000)
(-0.200000,-0.060000)
(-0.100000,-0.040000)
(-0.000000,-0.000000)
(0.100000,0.060000)
(0.200000,0.140000)
(0.300000,0.240000)
(0.400000,0.360000)
(0.500000,0.500000)
(0.600000,0.660000)
(0.700000,0.840000)
(0.800000,1.040000)
(0.900000,1.260000)
(1.000000,1.500000)
}\\
\end{preproc}
```

Фрагмент текста с Листинга 14 в печатном виде:

$x := 1$

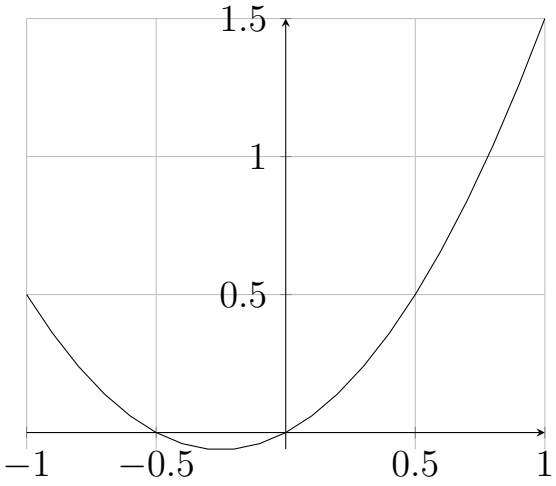
$x := 100$

$x0_{\text{текст}} := 228$

$f(x, y) := x \cdot (x + y)$

$$v := \begin{pmatrix} 1 \\ 2 \\ x0_{\text{текст}} \end{pmatrix} \quad v_0 := 666$$

$$v = \begin{pmatrix} 666.000000 \\ 2.000000 \\ 228.000000 \end{pmatrix}$$




f([-1:1], 0.5)

Далее проверим, что препроцессор правильно обрабатывает текст. В первую очередь следует рассмотреть базовые вещи: то, что соблюдается порядок операций в зависимости от приоритета оператора, работают условные операторы, оператор цикла, окружения, предопределенные функции и константы.

Приоритет арифметических операций соблюдается:

$1 - 2 * 3/4^5 = 0.994141$

$-2 * 3/4^5 + 1 = 0.994141$

$-2/4^5 * 3 + 1 = 0.994141$

Скобки меняют порядок выполнения арифметических операций:

$$(1-2)*3/4^5 = -0.002930$$
$$-2*(3/4)^5+1 = 0.525391$$
$$-(2/4^5*3+1) = -1.005859$$

Разные символы, обозначающие один и тот же оператор, действуют одинаково:

$$10*10\cdot 10\times 10 = 10000.000000$$
$$1/10 = 0.100000$$
$$\tfrac{1}{10} = 0.100000$$

Операторы сравнения корректно работают:

$$1 < 2 = 1.000000$$
$$1 \leq 2 = 1.000000$$
$$1 > 2 = 0.000000$$
$$1 \geq 2 = 0.000000$$
$$1 \neq 2 = 1.000000$$
$$1+1 = 2 = 1.000000$$

Приоритет логических операций соблюдается:

$$true = 1.000000$$
$$false = 0.000000$$
$$(\neg true) = 0.000000$$
$$(true \vee false) = 1.000000$$
$$(true \wedge false) = 0.000000$$
$$(true \wedge false \vee true) = 1.000000$$
$$(10+10 > 0 \wedge 4 = 5 \vee 7 = 7) = 1.000000$$

Замечание: приоритет логических операций ниже приоритета оператора '=', поэтому нужно ставить скобки.

Операторы \if, \otherwise, \while и окружения block, caseblock:

$$
\begin{array}{l}
\textbf{if}\ \ 1 < 2 \wedge 2 < 3 \\
\left| \begin{array}{l} x := 228 \end{array} \right. \\
\ \textbf{otherwise} \\
\left| \begin{array}{l} x := -666 \end{array} \right. \\
x = 228.000000 \\
i := 0 \\
\textbf{while}\ \ i < 5\ \ i := i + 1 \\
i = 5.000000 \\
x := \left| \begin{array}{l} 1\ \ \textbf{when}\ \ x > 0 \\ 0\ \ \textbf{when}\ \ x = 0 \\ -1\ \ \textbf{when}\ \ x < 0 \end{array} \right. \\
x = 1.000000
\end{array}
$$

Операции над матрицами и векторами:

$$
\begin{array}{l}
A := \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}\ A^{\mathrm{T}} = \begin{pmatrix} 1.000000 & 3.000000 & 5.000000 \\ 2.000000 & 4.000000 & 6.000000 \end{pmatrix} \\
A \cdot A^{\mathrm{T}} = \begin{pmatrix} 5.000000 & 11.000000 & 17.000000 \\ 11.000000 & 25.000000 & 39.000000 \\ 17.000000 & 39.000000 & 61.000000 \end{pmatrix} \\
x := [1 : 3 : 1] \\
y := [1 : 6 : 2] \\
x = \begin{pmatrix} 1.000000 & 2.000000 & 3.000000 \end{pmatrix} \\
y = \begin{pmatrix} 1.000000 & 3.000000 & 5.000000 \end{pmatrix} \\
x + y = \begin{pmatrix} 2.000000 & 5.000000 & 8.000000 \end{pmatrix} \\
x - y = \begin{pmatrix} 0.000000 & -1.000000 & -2.000000 \end{pmatrix} \\
x * y = 22.000000
\end{array}
$$

Функции высшего порядка:

$$fact_{\text{raw}}(f, n) := \left| \begin{array}{l} 1 \ \textbf{when} \ n \leq 0 \\ n \cdot f(n-1) \ \textbf{otherwise} \end{array} \right.$$

$$Y(f) := \left| \begin{array}{l} r(x) := f(Y(f), x) \\ r \end{array} \right.$$

$$fact := Y(fact_{\text{raw}})$$

$$fact(5) = 120.000000$$

Предопределенные математические функции:

$$\cos(\pi/3) = 0.500000$$

$$\sin(\pi/3) = 0.866025$$

$$\tan(\pi/3) = 1.732051$$

$$\cot(\pi/3) = 0.577350$$

$$\arcsin(0.866025) = 1.047197$$

$$\arccos(1/2) = 1.047198$$

$$\arctan(1.732051) = 1.047198$$

В приложении C приведен листинг с перечисленными примерами в исходном виде.

# 6 Заключение

В рамках данного курсового проекта был реализован препроцессор для TEX, в котором поддерживаются следующие возможности:

- Арифметические выражения
- Присваивания
- Вектора и матрицы
- Арифметические выражения с матрицами
- Стандартные математические функции
- Объявление функций
- Циклы
- Ветвления
- Функции высших порядков
- Области видимости
- Графики

# A Синтаксис

```
S ::= '\begin{preproc}' Operator* '\end{preproc}'
Operator ::= Def | Block | If | Cases | While | Expr | Graphic
Def::= Ident ( Deflist? | Index ):=' Operator
Block::= '\begin{block}' ( Operator '\\'? )* '\end{block}'
If ::= '\ifexpr{' Expr '}' '\\'? Operator '\\'?
    ( '\otherwise' Operator )?
Cases ::= '\begin{caseblock}' ( Alt '\\' )+
    ( '\otherwise' Operator '\\'? )?'\end{caseblock}'
Alt ::= Operator '\when' Expr
While ::= '\while{' Expr '}' Operator
Graphic ::= '\graphic{'Ident'}''{'(Expr(','Expr)*'}''{'Any'}'
Expr ::= Expr1 (( '\lor' | '\wedge' ) Expr)?
Expr1 ::= Expr2 (( '\land' | '\vee' ) Expr1)?
Expr2 ::= Expr3 ( '=' (Expr2 | '\placeholder{' Any '}') |
    '\neq' Expr2)?
Expr3 ::= Expr4 ( Cmp Expr3)?
Cmp := '<' | '>' | '\leq' | '\geq'
Expr4 ::= Expr5 (( '+' | '-' ) Expr4)?
Expr5 ::= Expr6 (( '*' | '\cdot' | '\times' | '/' ) Expr5)?
Expr6 ::= ( '+' | '-' | '\neg' )? Expr7
Expr7 ::= Elem ( '^''{' Expr '}' )?
Elem ::= Number | Ident Index? Arglist? | '(' Expr ')' |
      Keyword Arglist? | Matrix | Range |
      '\frac{' Expr '}''{' Expr '}' | '\transp{'Expr'}'
Range ::= '\range'('['Expr']')?'{'Expr'}''{'Expr'}'
Matrix ::= '\begin{pmatrix}'
        Line ( '\\' Line )* '\\' ?
        '\end{pmatrix}'
Line ::= Expr ( '&' Expr )*
Keyword ::= '\' Ident
Number ::= ( '0' | ( '1' | ... | '9' ) Digit*)( '.' Digit* )?
Digit ::= '0' | ... | '9'
```

```
Ident ::= Name ( '_\text{' Any '}' )?
Name ::= Char ( Char | Digit )?
Char ::= 'A' | ... | 'Z' | 'a' | ... | 'z'
Deflist ::= '(' ( Ident ( ',' Ident )* )? ')'
Arglist ::= '(' ( Elem ( ',' Elem )* )? ')'
Index ::= '_' ( Ident | Number | '{' Expr ( ',' Expr )? '}' )
Place ::= Operator '=' '\placeholder{' Any '}'
Any ::= любой набор символов
```

# B preproc.tex

```
\newcommand{\placeholder}[1]{#1}
\newcommand{\ifexpr}[1]{{\bf if}\enspace#1\enspace}
\newcommand{\when}{\enspace{\bf when}\enspace}
\newcommand{\otherwise}{\enspace{\bf otherwise}\enspace}
\newcommand{\while}[1]{{\bf while}\enspace#1\enspace}
\newcommand{\true}{true}
\newcommand{\false}{false}
\newcommand{\transp}[1]{{#1}^{\rm T}}
\newcommand{\range}[3][]{%
        \ifthenelse{\isempty{#1}}{[#2:#3]}{[#2:#3:#1]}}
\newcommand{\graphic}[3]{
        \begin{tikzpicture}\begin{axis}[
        xmajorgrids,ymajorgrids,axis lines=middle,
        x label style={at={(axis description cs:0.5,-0.1)},
                anchor=north},
        ylabel style={above left},
        xlabel=\text{#1(#2)},]
        \addplot[thin, mark = none] coordinates{#3};
        \end{axis}\end{tikzpicture}}

\newenvironment{preproc}
{\begin{equation*}\begin{array}{l}}
{\end{array}\end{equation*}}

\newenvironment{block}
{\begin{array}{|l}}{\end{array}}

\newenvironment{caseblock}
{\begin{array}{|l}}{\end{array}}
```

# C Тестовые примеры

```
\begin{preproc}
1-2*3/4^{5} = \placeholder{0.994141}\\
-2*3/4^{5}+1 = \placeholder{0.994141}\\
-2/4^{5}*3+1 = \placeholder{0.994141}\\
(1-2)*3/4^{5} = \placeholder{-0.002930}\\
-2*(3/4)^{5}+1 = \placeholder{0.525391}\\
-(2/4^{5}*3+1) = \placeholder{-1.005859}\\
10*10\cdot10\times10 = \placeholder{10000.000000}\\
1/10 = \placeholder{0.100000}\\
\frac{1}{10} = \placeholder{0.100000}\\
1 < 2 = \placeholder{1.000000}\\
1 \leq 2 = \placeholder{1.000000}\\
1 > 2 = \placeholder{0.000000}\\
1 \geq 2 = \placeholder{0.000000}\\
1 \neq 2 = \placeholder{1.000000}\\
1+1 = 2 = \placeholder{1.000000}\\
\true = \placeholder{1.000000}\\
\false = \placeholder{0.000000}\\
(\neg \true) = \placeholder{0.000000}\\
(\true \lor \false) = \placeholder{1.000000}\\
(\true \land \false) = \placeholder{0.000000}\\
(\true \land \false \lor \true) = \placeholder{1.000000}\\
(10+10 > 0 \land 4 = 5 \lor 7 = 7) = \placeholder{1.000000}\\
\ifexpr{1 < 2 \land 2 < 3}\\
        \begin{block}
        x := 228
        \end{block}\\
\otherwise\\
        \begin{block}
        x := -666
        \end{block}\\
x = \placeholder{228.000000}\\
```

```
i := 0\\
\while{i < 5} i := i+1\\
i = \placeholder{5.000000}\\
x := \begin{caseblock}
          1 \when x > 0 \\
          0 \when x = 0 \\
          -1 \when x < 0
          \end{caseblock}\\
x = \placeholder{1.000000}\\
A := \begin{pmatrix}
        1 & 2\\3 & 4\\5 & 6
\end{pmatrix}
\transp{A} = \placeholder{\begin{pmatrix}
                  1.000000 & 3.000000 & 5.000000\\
                  2.000000 & 4.000000 & 6.000000\end{pmatrix}}\\
A\cdot \transp{A} = \placeholder{\begin{pmatrix}
        5.000000 & 11.000000 & 17.000000\\
        11.000000 & 25.000000 & 39.000000\\
        17.000000 & 39.000000 & 61.000000\end{pmatrix}}\\
x := \range[1]{1}{3}\\
y := \range[2]{1}{6}\\
x = \placeholder{\begin{pmatrix}
        1.000000 & 2.000000 & 3.000000\end{pmatrix}}\\
y = \placeholder{\begin{pmatrix}
        1.000000 & 3.000000 & 5.000000\end{pmatrix}}\\
x+y = \placeholder{\begin{pmatrix}
        2.000000 & 5.000000 & 8.000000\end{pmatrix}}\\
x-y = \placeholder{\begin{pmatrix}
        0.000000 & -1.000000 & -2.000000\end{pmatrix}}\\
x*y = \placeholder{22.000000}\\
fact_\text{raw}(f, n) :=
\begin{caseblock}
        1 \when n \leq 0 \\
        n \cdot f(n - 1) \otherwise
```

```
\end{caseblock}\\
Y(f) := \begin{block}
        r(x) := f(Y(f), x) \\
        r
\end{block} \\
fact := Y(fact_\text{raw}) \\
fact(5) = \placeholder{120.0000000} \\
\cos(\pi/3) = \placeholder{0.500000}\\
\sin(\pi/3) = \placeholder{0.866025}\\
\tan(\pi/3) = \placeholder{1.732051}\\
\cot(\pi/3) = \placeholder{0.577350}\\
\arcsin(0.866025) = \placeholder{1.047197}\\
\arccos(1/2) = \placeholder{1.047198}\\
\arctan(1.732051) = \placeholder{1.047198}\\
\end{preproc}
```