

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №3
По теме “Синтаксический анализатор”

Выполнил:
студент гр. 853504
Шевченя И.В.

Проверил:
Ст. преподаватель КИ Шиманский В. В.

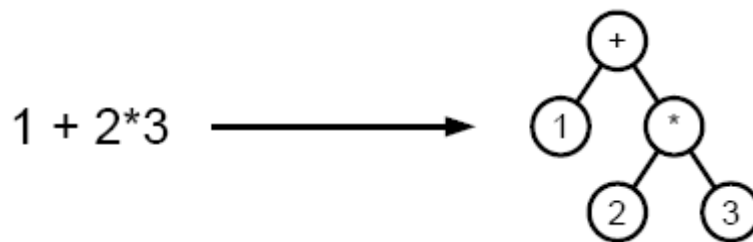
Минск 2021

1 Постановка задачи:

В данной работе ставится задача по исследованию области синтаксических анализаторов, рассмотрению аналогов и написанию своего собственного анализатора синтаксиса выбранного подмножества языка программирования. Требуется построить синтаксическое дерево. В качестве анализируемого языка программирования был использован язык программирования C++.

2 Теория:

Синтаксический анализ в лингвистике и информатике — процесс сопоставления линейной последовательности лексем (слов, токенов) естественного или формального языка с его формальной грамматикой. Результатом обычно является дерево разбора (синтаксическое дерево). Обычно применяется совместно с лексическим анализом.



Синтаксический анализатор— это программа или часть программы, выполняющая синтаксический анализ.



Пример разбора выражения в дерево

В ходе синтаксического анализа исходный текст преобразуется в структуру данных, обычно— в дерево, которое отражает синтаксическую структуру входной последовательности и хорошо подходит для дальнейшей обработки.

Как правило, результатом синтаксического анализа является синтаксическое строение предложения, представленное либо в виде дерева зависимостей, либо в виде дерева составляющих, либо в виде некоторого сочетания первого и второго способов представления.

Типы алгоритмов:

- Нисходящий парсер (англ. top-down parser) — продукции грамматики раскрываются, начиная со стартового символа, до получения требуемой последовательности токенов.
 - Метод рекурсивного спуска
 - LL-анализатор
- Восходящий парсер (англ. bottom-up parser) — продукции восстанавливаются из правых частей, начиная с токенов и кончая стартовым символом.
 - LR-анализатор
 - GLR-парсер

Программа-обработчик представляет собой многопроходный анализатор, который может обрабатывать не только отдельные слова, но и целые предложения, используя контекст предложений и абзацев, и используя его при возникновении трудностей с омонимией или в случае неполных или непонятных предложений.

Разбиение на абзацы позволяет выделить основную мысль данного абзаца (если она, конечно, есть), создавая тем самым контекст. Формально контекстом параграфа можно считать все пары подлежащих и сказуемых, встречающихся во всех предложениях данного абзаца. В случае односоставных предложений используется только один состав предложения. Если его, конечно же, удастся найти.

Вся основная работа производится на уровне предложений. Алгоритм анализа предложения достаточно прост и может быть описан в виде состояний конечного автомата.

Области применения синтаксических деревьев:

- Языки программирования — разбор исходного кода языков программирования, в процессе трансляции (компиляции или интерпретации);
- Структурированные данные — данные, языки их описания, оформления и т. д. Например, XML, HTML, CSS, JSON, ini-файлы, специализированные конфигурационные файлы и т. п.;
- Построение индекса в поисковой системе;
- SQL-запросы (DSL-язык);
- Математические выражения;
- Регулярные выражения (которые, в свою очередь, могут использоваться для автоматизации лексического анализа);
- Формальные грамматики;
- Лингвистика — естественные языки. Например, машинный перевод и другие генераторы текстов.
- Извлечение данных веб-страниц — веб-скрейпинг, является частным случаем парсинга.

Простейший способ реагирования на некорректную входную цепочку лексем — завершить синтаксический анализ и вывести сообщение об ошибке. Однако часто оказывается полезным найти за одну попытку синтаксического анализа как можно больше ошибок. Именно так ведут себя трансляторы большинства распространённых языков программирования.

Таким образом, перед обработчиком ошибок синтаксического анализатора стоят следующие задачи:

1. он должен ясно и точно сообщать о наличии ошибок;
2. он должен обеспечивать быстрое восстановление после ошибки, чтобы продолжать поиск других ошибок;
3. он не должен существенно замедлять обработку корректной входной цепочки.

Для построения дерева использовалась следующая таблица токенов:

Value (token)	Tag	Row	Column
void	VARIABLE TYPE	6	6
reverseArray	CUSTOM FUNCTION	6	19
(LEFT PAR	6	19
int	VARIABLE TYPE	6	23
arr	ID	6	27
[LEFT SQUARE	6	27
]	RIGHT SQUARE	6	28
,	COMMA	6	30
int	VARIABLE TYPE	6	34
start	ID	6	40
,	COMMA	6	41
int	VARIABLE TYPE	6	45
end	ID	6	49
)	RIGHT PAR	6	49
{	LEFT CURL	7	2
while	WHILE	8	11
(LEFT PAR	8	12
start	ID	8	18
<	LT	8	20

end	ID	8	24	
+-----+	+-----+	+-----+	+-----+	+-----+
)	RIGHT PAR	8	24	
+-----+	+-----+	+-----+	+-----+	+-----+
{	LEFT CURL	9	6	
+-----+	+-----+	+-----+	+-----+	+-----+
int	VARIABLE TYPE	10	13	
+-----+	+-----+	+-----+	+-----+	+-----+
temp	ID	10	18	
+-----+	+-----+	+-----+	+-----+	+-----+
=	EQUAL_SIGN	10	20	
+-----+	+-----+	+-----+	+-----+	+-----+
arr	ID	10	24	
+-----+	+-----+	+-----+	+-----+	+-----+
[LEFT SQUARE	10	24	
+-----+	+-----+	+-----+	+-----+	+-----+
start	ID	10	30	
+-----+	+-----+	+-----+	+-----+	+-----+
]	RIGHT SQUARE	10	30	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	11	1	
+-----+	+-----+	+-----+	+-----+	+-----+
arr	ID	11	13	
+-----+	+-----+	+-----+	+-----+	+-----+
[LEFT SQUARE	11	13	
+-----+	+-----+	+-----+	+-----+	+-----+
start	ID	11	19	
+-----+	+-----+	+-----+	+-----+	+-----+
]	RIGHT SQUARE	11	19	
+-----+	+-----+	+-----+	+-----+	+-----+
=	EQUAL_SIGN	11	22	
+-----+	+-----+	+-----+	+-----+	+-----+
arr	ID	11	26	
+-----+	+-----+	+-----+	+-----+	+-----+
[LEFT SQUARE	11	26	
+-----+	+-----+	+-----+	+-----+	+-----+
end	ID	11	30	
+-----+	+-----+	+-----+	+-----+	+-----+
]	RIGHT SQUARE	11	30	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	12	1	

;	SEMICOLON	12 1	
arr	ID	12 13	
[LEFT SQUARE	12 13	
end	ID	12 17	
]	RIGHT SQUARE	12 17	
=	EQUAL_SIGN	12 20	
temp	ID	12 25	
;	SEMICOLON	13 1	
start	ID	13 15	
++	ARITHMETIC_OPERATIONS	13 17	
;	SEMICOLON	14 1	
end	ID	14 13	
--	ARITHMETIC_OPERATIONS	14 15	
;	SEMICOLON	15 1	
}	RIGHT CURL	15 6	
}	RIGHT CURL	16 2	
void	VARIABLE TYPE	19 6	
printArray	CUSTOM FUNCTION	19 17	
(LEFT PAR	19 17	
int	VARIABLE TYPE	19 21	

arr	ID	19	25	
+-----+	+-----+	+-----+	+-----+	+-----+
[LEFT SQUARE	19	25	
+-----+	+-----+	+-----+	+-----+	+-----+
]	RIGHT SQUARE	19	26	
+-----+	+-----+	+-----+	+-----+	+-----+
,	COMMA	19	28	
+-----+	+-----+	+-----+	+-----+	+-----+
int	VARIABLE TYPE	19	32	
+-----+	+-----+	+-----+	+-----+	+-----+
size	ID	19	37	
+-----+	+-----+	+-----+	+-----+	+-----+
)	RIGHT PAR	19	37	
+-----+	+-----+	+-----+	+-----+	+-----+
{	LEFT CURL	20	2	
+-----+	+-----+	+-----+	+-----+	+-----+
for	FOR	21	8	
+-----+	+-----+	+-----+	+-----+	+-----+
(LEFT PAR	21	9	
+-----+	+-----+	+-----+	+-----+	+-----+
int	VARIABLE TYPE	21	13	
+-----+	+-----+	+-----+	+-----+	+-----+
i	ID	21	15	
+-----+	+-----+	+-----+	+-----+	+-----+
=	EQUAL_SIGN	21	17	
+-----+	+-----+	+-----+	+-----+	+-----+
0	NUMBER	21	19	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	21	20	
+-----+	+-----+	+-----+	+-----+	+-----+
i	ID	21	22	
+-----+	+-----+	+-----+	+-----+	+-----+
<	LT	21	24	
+-----+	+-----+	+-----+	+-----+	+-----+
size	ID	21	29	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	21	30	
+-----+	+-----+	+-----+	+-----+	+-----+
i	ID	21	32	
+-----+	+-----+	+-----+	+-----+	+-----+
++	ARITHMETIC_OPERATIONS	21	34	

)	RIGHT PAR	21	34	
+-----+	+-----+	+-----+	+-----+	+-----+
cout	FUNCTION	22	9	
+-----+	+-----+	+-----+	+-----+	+-----+
<<	OVERRIDE_OPERATION	22	12	
+-----+	+-----+	+-----+	+-----+	+-----+
arr	ID	22	16	
+-----+	+-----+	+-----+	+-----+	+-----+
[LEFT SQUARE	22	16	
+-----+	+-----+	+-----+	+-----+	+-----+
i	ID	22	18	
+-----+	+-----+	+-----+	+-----+	+-----+
]	RIGHT SQUARE	22	18	
+-----+	+-----+	+-----+	+-----+	+-----+
<<	OVERRIDE_OPERATION	22	22	
+-----+	+-----+	+-----+	+-----+	+-----+
	STRING	22	26	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	23	1	
+-----+	+-----+	+-----+	+-----+	+-----+
cout	FUNCTION	24	9	
+-----+	+-----+	+-----+	+-----+	+-----+
<<	OVERRIDE_OPERATION	24	12	
+-----+	+-----+	+-----+	+-----+	+-----+
endl	FUNCTION	24	17	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	25	1	
+-----+	+-----+	+-----+	+-----+	+-----+
}	RIGHT CURL	25	2	
+-----+	+-----+	+-----+	+-----+	+-----+
int	VARIABLE TYPE	27	5	
+-----+	+-----+	+-----+	+-----+	+-----+
main	CUSTOM FUNCTION	27	10	
+-----+	+-----+	+-----+	+-----+	+-----+
(LEFT PAR	27	10	
+-----+	+-----+	+-----+	+-----+	+-----+
)	RIGHT PAR	27	11	
+-----+	+-----+	+-----+	+-----+	+-----+
{	LEFT CURL	28	2	
+-----+	+-----+	+-----+	+-----+	+-----+
int	VARIABLE TYPE	29	9	

]	RIGHT SQUARE	29	14	
+-----+	+-----+	+-----+	+-----+	+-----+
=	EQUAL_SIGN	29	17	
+-----+	+-----+	+-----+	+-----+	+-----+
{	LEFT CURL	29	18	
+-----+	+-----+	+-----+	+-----+	+-----+
1	NUMBER	29	20	
+-----+	+-----+	+-----+	+-----+	+-----+
,	COMMA	29	21	
+-----+	+-----+	+-----+	+-----+	+-----+
2	NUMBER	29	23	
+-----+	+-----+	+-----+	+-----+	+-----+
,	COMMA	29	24	
+-----+	+-----+	+-----+	+-----+	+-----+
3	NUMBER	29	26	
+-----+	+-----+	+-----+	+-----+	+-----+
,	COMMA	29	27	
+-----+	+-----+	+-----+	+-----+	+-----+
4	NUMBER	29	29	
+-----+	+-----+	+-----+	+-----+	+-----+
,	COMMA	29	30	
+-----+	+-----+	+-----+	+-----+	+-----+
5	NUMBER	29	32	
+-----+	+-----+	+-----+	+-----+	+-----+
,	COMMA	29	33	
+-----+	+-----+	+-----+	+-----+	+-----+
6	NUMBER	29	35	
+-----+	+-----+	+-----+	+-----+	+-----+
}	RIGHT CURL	29	35	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	30	1	
+-----+	+-----+	+-----+	+-----+	+-----+
int	VARIABLE TYPE	31	9	
+-----+	+-----+	+-----+	+-----+	+-----+
n	ID	31	11	
+-----+	+-----+	+-----+	+-----+	+-----+
=	EQUAL_SIGN	31	13	
+-----+	+-----+	+-----+	+-----+	+-----+
sizeof	FUNCTION	31	20	
+-----+	+-----+	+-----+	+-----+	+-----+

)	RIGHT PAR	31	24	
+-----+	+-----+	+-----+	+-----+	+-----+
/	ARITHMETIC_OPERATIONS	31	27	
+-----+	+-----+	+-----+	+-----+	+-----+
sizeof	FUNCTION	31	34	
+-----+	+-----+	+-----+	+-----+	+-----+
(LEFT PAR	31	34	
+-----+	+-----+	+-----+	+-----+	+-----+
arr	ID	31	38	
+-----+	+-----+	+-----+	+-----+	+-----+
[LEFT SQUARE	31	38	
+-----+	+-----+	+-----+	+-----+	+-----+
0	NUMBER	31	40	
+-----+	+-----+	+-----+	+-----+	+-----+
]	RIGHT SQUARE	31	40	
+-----+	+-----+	+-----+	+-----+	+-----+
)	RIGHT PAR	31	41	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	31	43	
+-----+	+-----+	+-----+	+-----+	+-----+
printArray	ID	33	16	
+-----+	+-----+	+-----+	+-----+	+-----+
(LEFT PAR	33	16	
+-----+	+-----+	+-----+	+-----+	+-----+
arr	ID	33	20	
+-----+	+-----+	+-----+	+-----+	+-----+
,	COMMA	33	21	
+-----+	+-----+	+-----+	+-----+	+-----+
n	ID	33	23	
+-----+	+-----+	+-----+	+-----+	+-----+
)	RIGHT PAR	33	23	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	34	1	
+-----+	+-----+	+-----+	+-----+	+-----+
reverseArray	ID	35	18	
+-----+	+-----+	+-----+	+-----+	+-----+
(LEFT PAR	35	18	
+-----+	+-----+	+-----+	+-----+	+-----+
arr	ID	35	22	
+-----+	+-----+	+-----+	+-----+	+-----+
,	COMMA	35	23	

0	NUMBER	35	25	
+-----+	+-----+	+-----+	+-----+	+-----+
,	COMMA	35	26	
+-----+	+-----+	+-----+	+-----+	+-----+
n	ID	35	28	
+-----+	+-----+	+-----+	+-----+	+-----+
1	NUMBER	35	32	
+-----+	+-----+	+-----+	+-----+	+-----+
)	RIGHT PAR	35	32	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	36	1	
+-----+	+-----+	+-----+	+-----+	+-----+
cout	FUNCTION	37	10	
+-----+	+-----+	+-----+	+-----+	+-----+
<<	OVERRIDE_OPERATION	37	13	
+-----+	+-----+	+-----+	+-----+	+-----+
Reversed array is	STRING	37	33	
+-----+	+-----+	+-----+	+-----+	+-----+
<<	OVERRIDE_OPERATION	37	36	
+-----+	+-----+	+-----+	+-----+	+-----+
endl	FUNCTION	37	41	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	38	1	
+-----+	+-----+	+-----+	+-----+	+-----+
printArray	ID	39	16	
+-----+	+-----+	+-----+	+-----+	+-----+
(LEFT PAR	39	16	
+-----+	+-----+	+-----+	+-----+	+-----+
arr	ID	39	20	
+-----+	+-----+	+-----+	+-----+	+-----+
,	COMMA	39	21	
+-----+	+-----+	+-----+	+-----+	+-----+
n	ID	39	23	
+-----+	+-----+	+-----+	+-----+	+-----+
)	RIGHT PAR	39	23	
+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	40	1	
+-----+	+-----+	+-----+	+-----+	+-----+
return	RETURN	41	12	
+-----+	+-----+	+-----+	+-----+	+-----+
0	NUMBER	41	14	

+-----+	+-----+	+-----+	+-----+	+-----+
;	SEMICOLON	42	1	
+-----+	+-----+	+-----+	+-----+	+-----+
}	RIGHT CURL	42	2	
+-----+	+-----+	+-----+	+-----+	+-----+

3. Программа и комментарии:

Код парсера:

```
from lexer_constants import *

class Parser(object):
    """
    class Parser
    """

    def __init__(self):
        self.tokens = None

    def _node(self, pos):
        """
        return new node and pos
        """
        possible_values = {
            '{': '}',
            '(': ')',
        }

        node = list()
        while self.tokens[pos].value not in possible_values.values():
            delimiter = self.tokens[pos]
            if delimiter.value in possible_values.keys():
                new_node, pos = self._node(pos + 1)
                node.append([delimiter.value, new_node, possible_values[delimiter.value]])
            else:
                if delimiter.value in [*ARITHMETIC_OPERATIONS, *OVERRIDE_OPERATION, *COMPARE_SIGNS, ',', '']:
                    node.append([self.tokens[pos]])
                elif delimiter.tag in [WHILE, FOR, IF]:
                    node.extend([self.tokens[pos], ["condition:"]])
                else:
                    node.append(self.tokens[pos])
                pos += 1

        return node, pos

    def build(self, tokens):
        """
        return ast
        """
        ast = list()
```

```

        if tokens:
            pos = 0
            self.tokens = tokens
            ast.append("program: ")
            while pos < len(tokens):
                if tokens[pos].value == '{':
                    node, pos = self._node(pos + 1)
                    pos += 1
                    ast.append([node])
                else:
                    if tokens[pos].tag == TYPE:
                        pos_copy = pos
                        if tokens[pos + 1].tag ==
FUNC_DECLARATION:
                            node, pos = self._node(pos + 3)
                            ast.append(["function declaration:",
tokens[pos_copy].value, tokens[pos_copy + 1].value,
                                ["args:", "(", node,
                                ")", "body:"])
                            pos += 1
                            continue

                        msg = 'Parser error! Expected "{" but given
"%s"' % tokens[pos].value
                        msg += ' in line {}, column
{}'.format(tokens[pos].col - 1, tokens[pos].row)
                        raise Exception(msg)
                        ast.append("end program")

            return ast

def syntax_analyzer(ast, tabs):
    for i in ast:
        if isinstance(i, list):
            syntax_analyzer(i, tabs + 1)
        else:
            result, value = tabs * ' |', i.value if not
isinstance(i, str) else i
            print('{}{}'.format(result, value))

if __name__ == "__main__":
    path = "main_1.cpp"
    lexer = Lexer(path)
    parser = Parser()
    tokens = lexer.tokens()
    if len(check_if_main_exist(tokens)) == 0:
        lexer.error("Program should have function 'main'")

    if lexer.errors_list:
        lexer.errors()

```

```

draw_result_table(tokens)
draw_tags_groups(tokens)

ast = parser.build(tokens)
syntax_analyzer(ast, 2)

```

Расширение кода для более детального анализа:

```

class Node(object):
    def parts_str(self):
        return '\n'.join(map(str, [x for x in self.parts]))

    def __init__(self, type, parts):
        self.type = type
        self.parts = parts

    def __str__(self):
        return self.type + ":\n____" +
self.parts_str().replace("\n", "\n____")

    def add_parts(self, parts):
        self.parts += parts
        return self

def p_program(p):
    '''program :
        | function
        | program function
    '''
    if len(p) <= 2:
        p[0] = Node('program', p[1:])
    else:
        p[0] = p[1].add_parts([p[2]])

def p_function(p):
    '''function : func_header func_body
    '''

    p[0] = Node('function', p[1:])

def p_func_header(p):
    '''func_header : VARIABLE_TYPE FUNCDECL LPAR args RPAR'''
    p[0] = Node('func_declaration', [p[1], p[2], p[4]])

def p_args(p):
    '''args :
        | expr
        | args COMMA expr'''

```

```

    if len(p) <= 2:
        p[0] = Node('args', p[1:] if p[1:] else ['EMPTY'])
    else:
        p[0] = p[1].add_parts([p[3]])

def p_func_body(p):
    '''func_body : block'''
    p[0] = p[1]

def p_block(p):
    '''block : LCURL body RCURL'''
    p[0] = Node('block', [p[2]])

def p_body(p):
    '''body :
        | body line semicolons
        | body multiline'''
    if len(p) > 1:
        if p[1] is None:
            p[1] = Node('body', [])
            p[0] = p[1].add_parts([p[2]])
        else:
            p[0] = Node('body', [])

def p_semicolons(p):
    '''semicolons : SEMICOLON
        | semicolons SEMICOLON'''

def p_multiline(p):
    '''multiline : if_statement
        | while_statement
        | for_statement'''
    p[0] = p[1]

def p_line(p):
    '''line : modal_function
        | init
        | func
        | assign'''
    p[0] = p[1]

def p_modal_function(p):
    '''modal_function : RETURN arg
        | BREAK
        | CONTINUE'''
    if len(p) == 3:

```



```

        p[0] = Node("modal_function", p[1:])
    else:
        p[0] = Node("modal_function", [p[1]])

def p_var_call(p):
    '''var_cal : ID LCUADR expr RCUADR'''
    p[0] = Node('var_call', [p[1], p[3]])

def p_if_statement(p):
    '''if_statement : IF LPAR condition RPAR block
                    | if_statement ELSE block'''
    if len(p) == 4:
        p[0] = p[1].add_parts(['else', p[3]])
    else:
        p[0] = Node('if', [p[3], p[5]])

def p_while_statement(p):
    '''while_statement : WHILE LPAR condition RPAR block'''
    p[0] = Node('while', [p[3], p[5]])

def p_for_statement(p):
    '''for_statement : FOR LPAR init SEMICOLON condition
                    SEMICOLON change_val RPAR block'''
    p[0] = Node('for', [p[3], p[5], p[7], p[9]])

def p_change_value(p):
    '''change_val : ID expr'''
    p[0] = Node("change_val", p[1:])

def p_condition(p):
    '''condition : expr cond_sign expr'''
    p[0] = Node('condition', [p[1], p[2], p[3]])

def p_cond_sign(p):
    '''cond_sign : DEQUAL
                | GT
                | LT
                | GE
                | LE
                | NOTEQUAL'''
    p[0] = p[1]

def p_init(p):
    '''init :
            | VARIABLE_TYPE ID'''

```

```

        | VARIABLE_TYPE ID EQUAL ID DIVMUL NUMBER
        | VARIABLE_TYPE ID EQUAL expr
        | VARIABLE_TYPE ID EQUAL var_cal
        | VARIABLE_TYPE ID LCUADR RCUADR EQUAL array_init'''
if len(p) > 5:
    p[0] = Node('init', [p[1], "[", p[2], p[5], p[6]])
else:
    p[0] = Node('init', p[1:])

def p_array_init(p):
    '''array_init : LCURL init_block RCURL'''
    p[0] = Node("array_init", [p[2]])

def p_init_block(p):
    '''init_block : arg
                  | arg COMMA
                  | init_block arg
                  | init_block arg COMMA'''
    if len(p) == 2:
        p[0] = Node("init_block", p[1:])
    else:
        if p[2] != ",":
            p[0] = p[1].add_parts(p[2:])
        else:
            p[0] = Node("init_block", p[1:])

def p_assign(p):
    '''assign : ID EQUAL expr
              | ID EQUAL var_cal
              | var_cal EQUAL expr
              | var_cal EQUAL var_cal
              | ID expr'''
    if len(p) == 5:
        p[0] = Node('assign', [p[2], p[4]])
    elif len(p) == 4 or len(p) == 3:
        p[0] = Node('assign', p[1:])
    else:
        p[0] = Node('assign', [p[1], p[3]])

def p_func(p):
    '''func : CUSTOM_FUNC LPAR args RPAR
            | ID LPAR args RPAR
            | BUILD_IN LPAR args RPAR
            | BUILD_IN output_operator'''
    if len(p) == 3:
        p[0] = Node('func_call', [p[1], p[2]])
    else:
        p[0] = Node('func_call', [p[1], p[3]])

```

```
def p_output_operator(p):
    '''output_operator : LT LT arg
                        | LT LT BUILD_IN
                        | LT LT ID'''

    p[0] = Node('output_operator', ["<<", p[3]])
```

```
def p_expr(p):
    '''expr : fact
            | PLUSMINUS PLUSMINUS
            | expr PLUSMINUS fact
            | expr MOD fact
            | ID'''
    if len(p) == 2:
        p[0] = p[1]
    elif len(p) == 3:
        if p[2] == "+":
            p[0] = Node('assign', ["++"])
        elif p[2] == "-":
            p[0] = Node('assign', ["--"])
    else:
        p[0] = Node(p[2], [p[1], p[3]])
```

```
def p_fact(p):
    '''fact : term
            | fact DIVMUL term'''
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[0] = Node(p[2], [p[1], p[3]])
```

```
def p_term(p):
    '''term : arg
            | LPAR expr RPAR'''
    if len(p) == 2:
        p[0] = p[1]
    else:
        p[0] = p[2]
```

```
def p_arg(p):
    '''arg : NUMBER
           | STRING
           | VARIABLE_TYPE ID
           | VARIABLE_TYPE ID LCUADR RCUADR
           | ID LCUADR RCUADR
           | var_cal
           | NUMBER ID
           | func'''
```

```

    if len(p) == 2:
        p[0] = Node('arg', [p[1]])
    else:
        p[0] = Node('arg', p[1:])

def p_error(p):
    print('Unexpected token in line %d: %s' % (p.lineno, p))

def build_tree(code):
    parser = yacc.yacc()
    return parser.parse(code, debug=True)

if __name__ == '__main__':
    tree = build_tree(data)
    print(tree)

```

Код программ для разбора:

```

1)

void reverseArray(int arr[], int start, int end)
{
    while (start < end)
    {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}

void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";

    cout << endl;
}

int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6};

    int n = sizeof(arr) / sizeof(arr[0]);

    printArray(arr, n);

    reverseArray(arr, 0, n - 1);
}

```

```

    cout << "Reversed array is";
    cout << endl;

    printArray(arr, n);

    if (a < b)
    {
        int t = 10;
    }
    else {
        int t = 15;
    }

    cout << t;

    return 0;
}

```

2)

```

void merge(int *a, int n)
{
    int mid = n / 2;
    if (n % 2 == 1)
    {
        mid++;
    }
    int h = 1;
    int *c = (int*)malloc(n * sizeof(int));
    int step;
    while (h < n)
    {
        step = h;
        int i = 0;
        int j = mid;
        int k = 0;
        while (step <= mid)
        {
            while ((i < step) && (j < n) && (j < (mid + step)))
            {
                if (a[i] < a[j])
                {
                    c[k] = a[i];
                    i++; k++;
                }
                else {
                    c[k] = a[j];
                    j++; k++;
                }
            }
            while (i < step)
            {

```

```

        c[k] = a[i];
        i++; k++;
    }
    while ((j < (mid + step)) && (j < n))
    {
        c[k] = a[j];
        j++; k++;
    }
    step = step + h;
}
h = h * 2;

for (i = 0; i < n; i++)
{
    a[i] = c[i];
}
}
}

```

```

int main()
{
    int a[8];
    for (int i = 0; i < 8; i++)
    {
        a[i] = rand() % 20 - 10;
    }
    for (int i = 0; i < 8; i++)
    {
        printf("%d ", a[i]);
    }
    printf("\n");
    merge(a, 8);
    for (int i = 0; i < 8; i++)
    {
        printf("%d ", a[i]);
    }
    printf("\n");

    getchar();
    return 0;
}

```

Результат работы парсера в виде синтаксического дерева:

```

program:
__function:
____func_declaration:
_____void
_____reverseArray
_____args:
_____arg:

```

```

_____int
_____arr
_____ [
_____ ]
_____arg:
_____int
_____start
_____arg:
_____int
_____end
_____block:
_____body:
_____while:
_____condition:
_____start
_____<
_____end
_____block:
_____body:
_____init:
_____int
_____temp
_____ =
_____var_call:
_____arr
_____start
_____assign:
_____var_call:
_____arr
_____start
_____ =
_____var_call:
_____arr
_____end
_____assign:
_____var_call:
_____arr
_____end
_____ =
_____temp
_____assign:
_____start
_____assign:
_____++
_____assign:
_____end
_____assign:
_____--
_____function:
_____func_declaration:
_____void
_____printArray
_____args:

```

```

_____arg:
_____int
_____arr
_____
_____]
_____arg:
_____int
_____size
_____block:
_____body:
_____for:
_____init:
_____int
_____i
_____
_____
_____arg:
_____0
_____condition:
_____i
_____<
_____size
_____change_val:
_____i
_____assign:
_____++
_____block:
_____body:
_____func_call:
_____cout
_____output_operator:
_____<<
_____arg:
_____var_call:
_____arr
_____i
_____func_call:
_____cout
_____output_operator:
_____<<
_____arg:
_____ " "
_____func_call:
_____cout
_____output_operator:
_____<<
_____endl
_____function:
_____func_declaration:
_____int
_____main
_____args:
_____EMPTY
_____block:

```



```

_____body:
_____init:
_____int
_____[]
_____arr
_____ =
_____array_init:
_____init_block:
_____arg:
_____1
_____ /
_____arg:
_____2
_____ /
_____arg:
_____3
_____ /
_____arg:
_____4
_____ /
_____arg:
_____5
_____ /
_____arg:
_____6
_____init:
_____int
_____n
_____ =
_____/:
_____arg:
_____func_call:
_____sizeof
_____args:
_____arr
_____arg:
_____func_call:
_____sizeof
_____args:
_____arg:
_____var_call:
_____arr
_____arg:
_____0
_____func_call:
_____printArray
_____args:
_____arr
_____n
_____func_call:
_____reverseArray
_____args:
_____arr

```

```

_____arg:
_____0
_____ -:
_____n
_____arg:
_____1
_____func_call:
_____cout
_____output_operator:
_____<<
_____arg:
_____ "Reversed array is"
_____func_call:
_____cout
_____output_operator:
_____<<
_____endl
_____func_call:
_____printArray
_____args:
_____arr
_____n
_____if:
_____condition:
_____a
_____<
_____b
_____block:
_____body:
_____init:
_____int
_____t
_____ =
_____arg:
_____10
_____else
_____block:
_____body:
_____init:
_____int
_____t
_____ =
_____arg:
_____15
_____func_call:
_____cout
_____output_operator:
_____<<
_____t
_____modal_function:
_____return
_____arg:
_____0

```

Составные части дерева:

Блоки: program, function, func_header, func_body, args, block, body, line, multiline, semicolons, if_statement, while_statement, for_statement, modal_function, init, func, assign, arg, var_cal, expr, change_val, condition, cond_sign, array_init, init_block, func, output_operator, fact, term.

Рассмотрим текст программы с ошибками. При обнаружении их происходит вывод уведомления об ошибке (красным выделено место ошибки или отсутствующий символ/фрагмент):

1 Отсутствие открывающейся фигурной скобки:

Фрагмент кода:

```
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6};
    t n = sizeof(arr) / sizeof(arr[0]);

    printArray(arr, n);
    verseArray(arr, 0, n - 1);
    ut << "Reversed array is" << endl;
    intArray(arr, n);

    return 0;
}
```

Код ошибки:

```
Traceback (most recent call last):
  File "/home/shevchenya/PycharmProjects/mtran/lexer.py", line 562, in <module>
    ast = parser.build(tokens)
  File "/home/shevchenya/PycharmProjects/mtran/parser.py", line 72, in build
    raise Exception(msg)
Exception: Parser error! Expected "{" but given "int" in line 29
```

2 Лишняя закрывающая скобка:

Фрагмент кода:

```
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6};

    int n = sizeof(arr) / sizeof(arr[0]);

    printArray(arr, n);
    ...
}
```

Код ошибки:

```
Unexpected token in line 24: LexToken(RPAR,')',24,366)
program:
```

3 Пропущен оператор «=»

Фрагмент кода:

```
if (a < b)
{
    int t 10;
}
```

Код ошибки:

```
>>> runfile('/home/shevchenya/PycharmProjects/mtran/syntax_analysis.py')
Unexpected token in line 41: LexToken(NUMBER,10,41,621)
program:
----
```

4 Отсутствие символа «;» между аргументами в определении for:

Фрагмент кода:

```
void printArray(int arr[], int size)
{
    for (int i = 0 i < size; i++)
    {
        cout << arr[i];
        cout << " ";
    }
    cout << endl;
}
```

Код ошибки:

```
>>> runfile('/home/shevchenya/PycharmProjects/mtran/syntax_analysis.py',
Unexpected token in line 16: LexToken(LT,'<',16,269)
program:
```

5 Использование фраз на другом языке

Фрагмент кода:

```
void reverseArray(int arr[], int start, int end)
{
    привет
    while (start < end)
    {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}
```

Код ошибки:

```
runfile('/home/shevchenya/PycharmProjects/mtran/syntax_analysis.py',
Illegal character 'н' at line 4
Illegal character 'р' at line 4
Illegal character 'и' at line 4
Illegal character 'в' at line 4
Illegal character 'е' at line 4
Illegal character 'т' at line 4
program:
```

6 Появление неизвестного символа:

Код ошибки:

```
Unknown character: "#" in line 1, column 2
Unknown character: "." in line 1, column 17
Missing end of line: in line 1, column 19
Unknown character: "#" in line 2, column 2
Missing end of line: in line 2, column 20
```

7 Отсутствие символа окончания строки:

Код ошибки:

```
Missing end of line: in line 22, column 29
```

8 Отсутствие необходимой закрывающей фигурной скобки

Код ошибки:

```
Traceback (most recent call last):
  File "/home/shevchenya/PycharmProjects/mtran/lexer.py", line 562, in <module>
    ast = parser.build(tokens)
  File "/home/shevchenya/PycharmProjects/mtran/parser.py", line 57, in build
    node, pos = self._node(pos + 1)
  File "/home/shevchenya/PycharmProjects/mtran/parser.py", line 41, in _node
    raise Exception(msg)
Exception: Parser error! Missing symbol "}" in line 43, column 0
```

9 Отсутствие в определении параметров функции закрывающей круглой скобки:

Код ошибки:

```
Traceback (most recent call last):
  File "/home/shevchenya/PycharmProjects/mtran/lexer.py", line 562, in <module>
    ast = parser.build(tokens)
  File "/home/shevchenya/PycharmProjects/mtran/parser.py", line 57, in build
    node, pos = self._node(pos + 1)
  File "/home/shevchenya/PycharmProjects/mtran/parser.py", line 41, in _node
    raise Exception(msg)
Exception: Parser error! Missing symbol ")" in line 19, column 28
```

Вывод:

В результате была создана программа, строящая синтаксическое дерево по коду программы на языке программирования C++. Программа позволяет

отслеживать синтаксические ошибки и выводит сообщения о них. Программа пытается выделить синтаксические конструкции из групп токенов согласно определённой грамматике. Если это не получается — значит в коде присутствует синтаксическая ошибка. Об этом сигнализируется пользователю.