INE 5426

Trabalho 3: Analisador Semântico

SDD para EXPA:

NUMEXPRESSION -> TERM REC_PLUS_MINUS_TERM	NUMEXPRESSION.val = REC_PLUS_MINUS_TERM.val REC_PLUS_MINUS_TERM.her = TERM.val
REC_PLUS_MINUS_TERM1 -> PLUS_OR_MINUS TERM REC_PLUS_MINUS_TERM2	REC_PLUS_MINUS_TERM2.her = REC_PLUS_MINUS_TERM1.her PLUS_OR_MINUS.op TERM.val REC_PLUS_MINUS_TERM1.val = REC_PLUS_MINUS_TERM2.val
REC_PLUS_MINUS_TERM1 -> &	REC_PLUS_MINUS_TERM1.val = REC_PLUS_MINUS_TERM1.her
PLUS_OR_MINUS -> +	PLUS_OR_MINUS.op = “+”
PLUS_OR_MINUS -> -	PLUS_OR_MINUS.op = “-”

SDD para EXPA:

TERM -> UNARYEXPR REC_UNARYEXPR	REC_UNARYEXPR.her = UNARYEXPR.val TERM.val = REC_UNARYEXPR.val
REC_UNARYEXPR -> UNARYEXPR_OP TERM	REC_UNARYEXPR.val = REC_UNARYEXPR.her UNARYEXPR_OP.op TERM.val
REC_UNARYEXPR -> &	REC_UNARYEXPR.val = REC_UNARYEXPR.her
UNARYEXPR_OP -> *	UNARYEXPR_OP.op = " * "
UNARYEXPR_OP -> /	UNARYEXPR_OP.op = " / "
UNARYEXPR_OP -> %	UNARYEXPR_OP.op = " % "

SDD para EXPA:

UNARYEXPR -> PLUS_OR_MINUS FACTOR	UNARYEXPR.val = PLUS_OR_MINUS.op FACTOR.val
UNARYEXPR -> FACTOR	UNARYEXPR.val = FACTOR.val
FACTOR -> int_constant	FACTOR.val = lex
FACTOR -> float_constant	FACTOR.val = lex
FACTOR -> string_constant	FACTOR.val = lex

SDD para EXPA:

FACTOR -> return_null	FACTOR.val = lex
FACTOR -> LVALUE	FACTOR.val = LVALUE.val
FACTOR -> (NUMEXPRESSION)	FACTOR.val = NUMEXPRESSION.val
LVALUE -> ident OPT_ALLOC_NUMEXP	LVALUE.val = ident

SDD para EXPA é L-atribuida?

- Como demonstrado durante a apresentação da SDD, todos os atributos herdados vem ou do pai, ou do irmão à esquerda.

- Portanto a SDD é considerada L-atribuida.

SDT para EXPA:

NUMEXPRESSION -> TERM REC_PLUS_MINUS_TERM	NUMEXPRESSION.node = REC_PLUS_MINUS_TERM.node REC_PLUS_MINUS_TERM.her = TERM.node
REC_PLUS_MINUS_TERM1 -> PLUS_OR_MINUS TERM REC_PLUS_MINUS_TERM2	temp_node = PLUS_OR_MINUS.node temp_node.fe = REC_PLUS_MINUS_TERM1.her temp_node.node.fd = TERM.node REC_PLUS_MINUS_TERM2.her = temp_node.node REC_PLUS_MINUS_TERM1.node = REC_PLUS_MINUS_TERM2.node
REC_PLUS_MINUS_TERM1 -> &	REC_PLUS_MINUS_TERM1.node = REC_PLUS_MINUS_TERM1.her
PLUS_OR_MINUS -> +	PLUS_OR_MINUS.node = new node('+', ,)
PLUS_OR_MINUS -> -	PLUS_OR_MINUS.node = new node('-', ,)

SDT para EXPA:

TERM -> UNARYEXPR REC_UNARYEXPR	REC_UNARYEXPR.her = UNARYEXPR.node TERM.node = REC_UNARYEXPR.node
REC_UNARYEXPR -> UNARYEXPR_OP TERM	temp_node = UNARYEXPR_OP.node temp_node.node.fe = REC_UNARYEXPR.her temp_node.node.fd = TERM.node REC_UNARYEXPR.node = temp_node.node
REC_UNARYEXPR -> &	REC_UNARYEXPR.node = REC_UNARYEXPR.her
UNARYEXPR_OP -> *	UNARYEXPR_OP.node = new node(*, ,)
UNARYEXPR_OP -> /	UNARYEXPR_OP.node = new node(/, ,)
UNARYEXPR_OP -> %	UNARYEXPR_OP.node = new node(%, ,)

SDT para EXPA:

UNARYEXPR -> PLUS_OR_MINUS FACTOR	PLUS_OR_MINUS.node.fe = FACTOR.node UNARYEXPR.node = PLUS_OR_MINUS.node
UNARYEXPR -> FACTOR	UNARYEXPR.node = FACTOR.node
FACTOR -> int_constant	FACTOR.node = new node(int_constant, lex)
FACTOR -> float_constant	FACTOR.node = new node(float_constant, lex)
FACTOR -> string_constant	FACTOR.node = new node(string_constant, lex)

SDT para EXPA:

FACTOR -> return_null	FACTOR.node = new node(return_null, lex)
FACTOR -> LVALUE	FACTOR.node = LVALUE.node
FACTOR -> (NUMEXPRESSION)	FACTOR.node = NUMEXPRESSION.node
LVALUE -> ident OPT_ALLOC_NUMEXP	LVALUE.node = new node(ident, val_from_table,)

SDT para EXPA é L-atribuida?

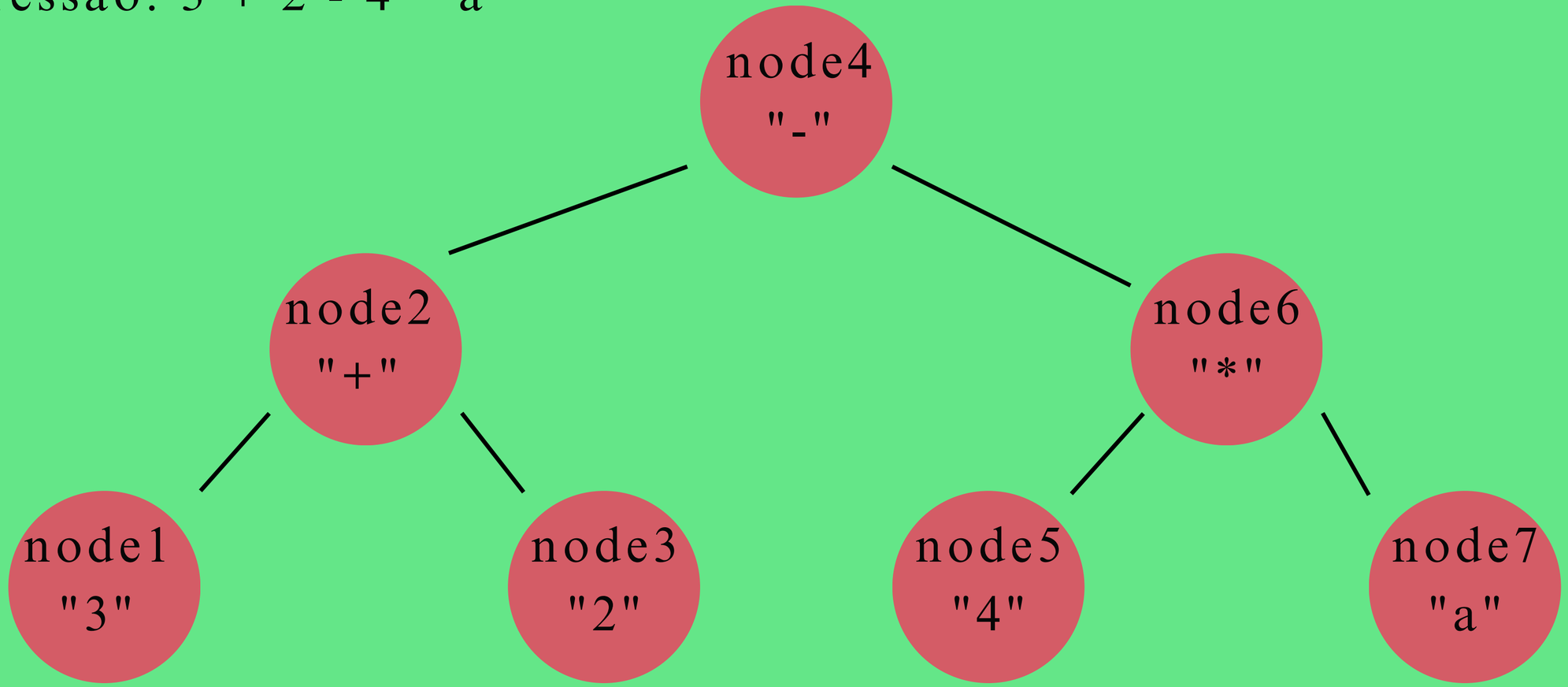
- Assim como para a SDD de EXPA, demonstramos que os atributos herdados, sempre vem do pai, ou do irmão à esquerda.

- Portanto a SDT também é considerada L-atribuida.

Exemplo de árvore gerada pela SDT:

(este exemplo foi feito à mão pelos alunos seguindo as regras da SDT)

- expressão: 3 + 2 - 4 * a



SDD e SDT para DEC

SDD para DEC:

VARDECL -> DATATYPE ident OPT_VECTOR	VARDECL.type = DATATYPE.type VARDECL.val = ident.lex VARDECL.dimension = OPT_VECTOR.dimension
DATATYPE -> int	DATATYPE.type = int.lex
DATATYPE -> float	DATATYPE.type = float.lex
DATATYPE -> string	DATATYPE.type = string.lex
OPT_VECTOR1 -> [int_constant] OPT_VECTOR2	OPT_VECTOR1.dimension = 1 + OPT_VECTOR2.dimension. OPT_VECTOR1.size = int_constant.lex
OPT_VECTOR -> &	OPT_VECTOR.dimension = 0

SDD para DEC é L-atribuida?

- A SDD de DEC, como demonstramos possui apenas atributos sintetizados.
- Portanto a SDD pode ser considerada como L-atribuida.

SDT para EXPA:

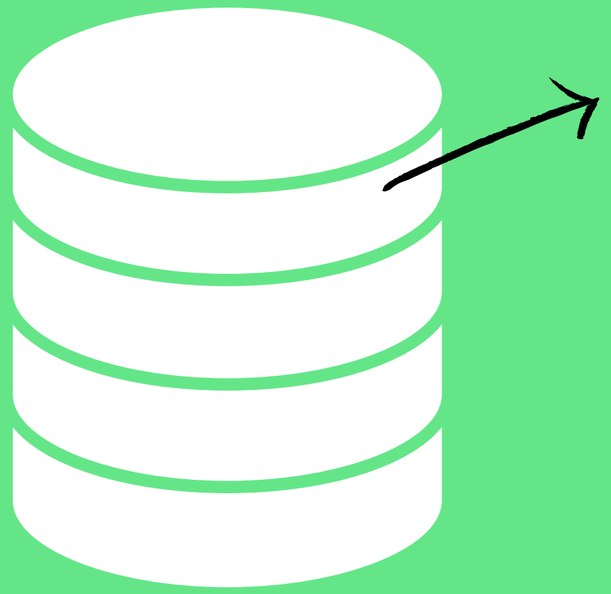
VARDECL -> DATATYPE ident OPT_VECTOR	VARDECL.entry = insert_new_sst_symbol(ident.lex, DATATYPE.type, OPT_VECTOR.dimension, OPT_VECTOR.size);
DATATYPE -> int	DATATYPE.type = int
DATATYPE -> float	DATATYPE.type = float
DATATYPE -> string	DATATYPE.type = string
OPT_VECTOR1 -> [int_constant] OPT_VECTOR2	OPT_VECTOR1.dimension = 1 + OPT_VECTOR2.dimension. OPT_VECTOR1.size = int_constant.lex
OPT_VECTOR -> &	OPT_VECTOR.dimension = 0

SDD para DEC é L-atribuida?

- A SDT de DEC, bem como sua SDD possui apenas atributos sintetizados.
- Logo a SDT também é considerada como L-atribuida.

Verificações no parser.y

Estrutura geral do parser.y

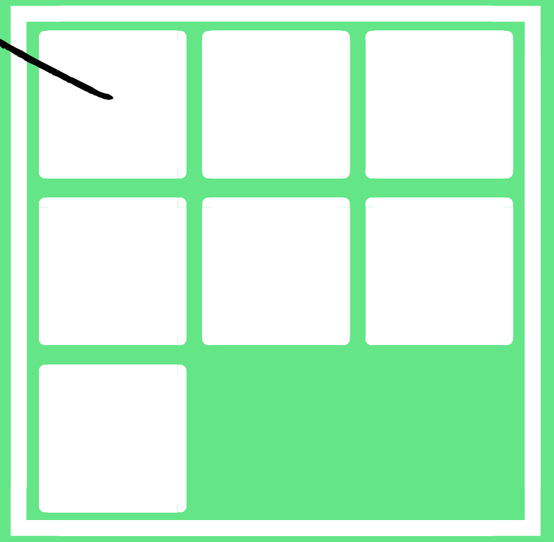


Scopes

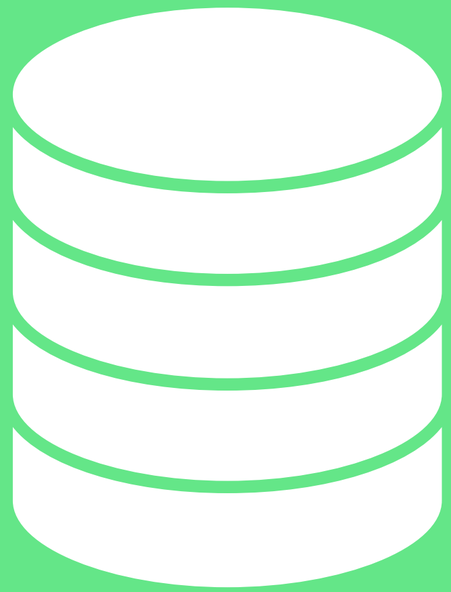
```c
typedef struct {
    sst symbol_table[MAX_SCOPES];
    int num_symbols;
    bool is_loop;
} scope;
```

```c
// Initializing the top of the
int top = 0;
int top_all_scopes = 0;

// Initializing the stack using
scope scopes[MAX_SCOPES];
scope all_scopes[MAX_SCOPES];
```

All Scopes



Estrutura geral do parser.y



Scopes

```
                  1          3
PROGRAM : new_scope STATEMENT {
                pop();
                show_tables();  4
        }
        | new_scope FUNCLIST {
                pop();
                show_tables();
        }
        | { show_tables(); }
        ;
```

```
                2
new_scope : { new_scope(false); }
```

Estrutura geral do parser.y

symbol	type	usage count	dimension
printar	function	1	0
a	int	2	0
list	int	1	2
	⋮		

```c
//structing syntax symbol table
typedef struct {
    char symbol[32];
    char type[32];
    int usage_count;
    int dimension;
} sst;
```

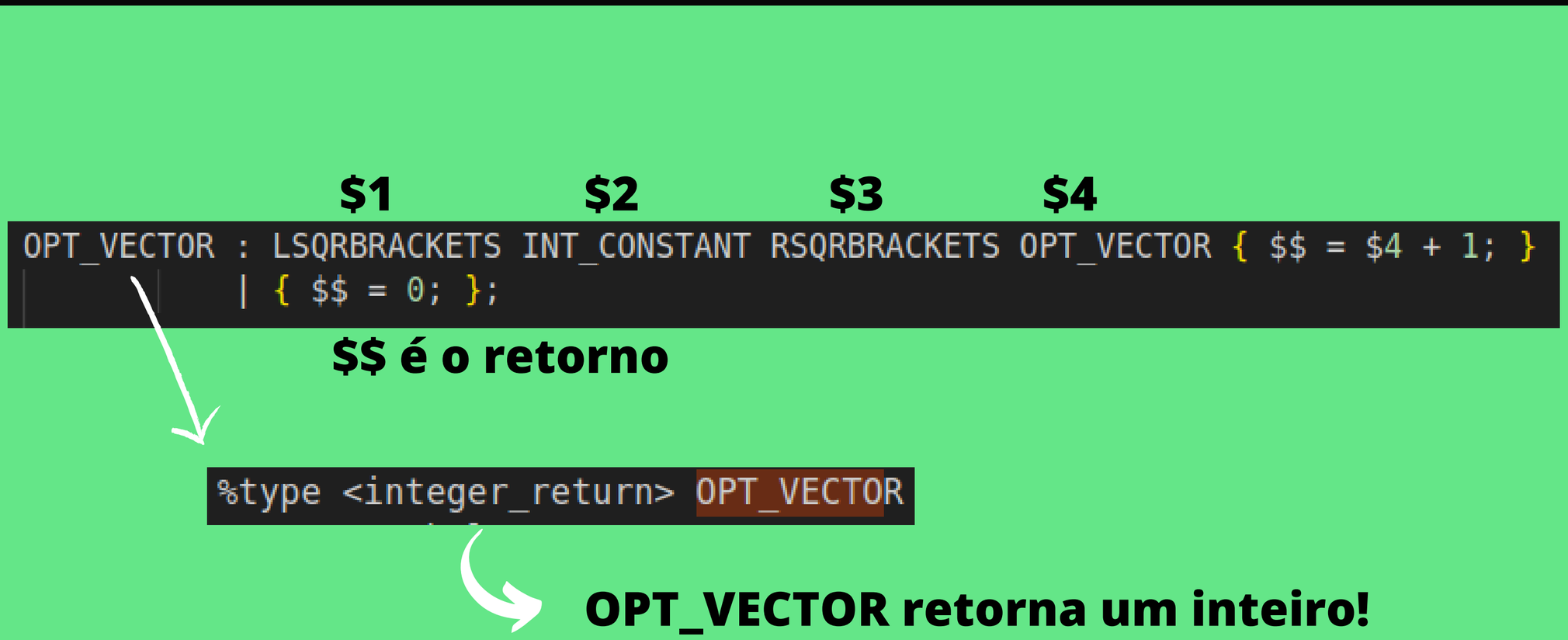
Estrutura geral do parser.y

```
110   %type <node> ALLOCEXPRESSION
111   %type <symbol> OPT_ALLOC_NUMEXP
112   %type <node> EXPRESSION
113   %type <node> OPT_REL_OP_NUM_EXPR
114   %type <symbol> REL_OP
115   %type <scope_and_expressions> NUMEXPRESSION
116   %type <scope_and_expressions> REC_PLUS_MINUS_TERM
117   %type <scope_and_expressions> PLUS_OR_MINUS
118   %type <scope_and_expressions> TERM
119   %type <scope_and_expressions> REC_UNARYEXPR
120   %type <scope_and_expressions> UNARYEXPR_OP
121   %type <scope_and_expressions> UNARYEXPR
122   %type <scope_and_expressions> FACTOR
123   %type <scope_and_expressions> LVALUE
```

```
typedef struct scope_and_expressions {
        char * operation;
        char * vector;
        node node;
} scope_and_expressions;
```

```
union Value {
    int i;
    float f;
    char str[100];
};

typedef struct {
    char * node_before;
    char * node_after;
    char operation[3];
    char * result;
    union Value value;
} node;
```

Estrutura geral do parser.y



Verificações de tipo

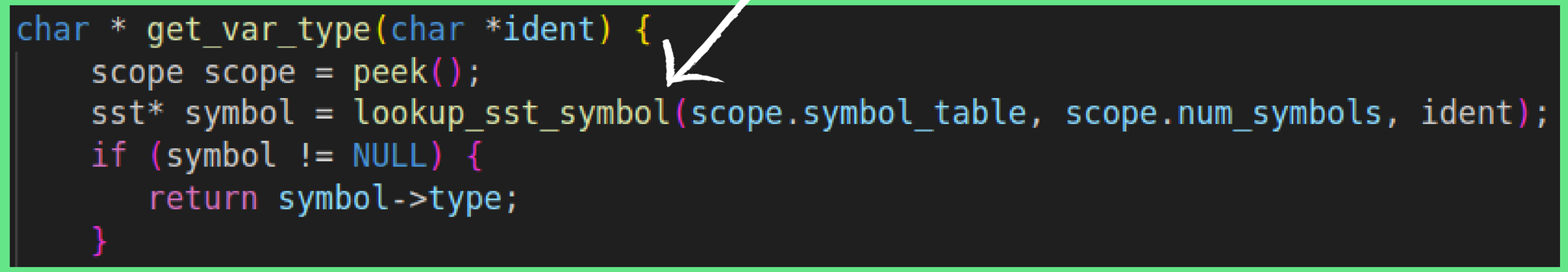
Como essas estruturas se conversam?

```
%token <symbol> IDENT
```

```
FUNCCALL_OR_EXPRESSION : IDENT FOLLOW_IDENT {
                        node new_node;
                        strcpy(new_node.node_before, $1);
                        new_node.result = get_var_type($1);

                        if ($2 != NULL && (strcpy($2->node.operation, "0") == 0)) {
                                strcpy(new_node.operation, $2->vector);

                                char * result_type = check_operation(new_node.result, $2->node.result, $2->node.operation);
                                if (strcmp(result_type, "0") == 0) {
                                        yyerror(" ");
                                        YYABORT;
                                }
                                node new_right_node_result = {new_node.result, $2->node.result, $2->node.operation, result_type};
                                new_node = new_right_node_result;

                                num_expressions[top_num_expressions] = new_node;
                                top_num_expressions += 1;
                        }
                };
```

Como essas estruturas se conversam?

Olha na tabela



```
char * get_var_type(char *ident) {
    scope scope = peek();
    sst* symbol = lookup_sst_symbol(scope.symbol_table, scope.num_symbols, ident);
    if (symbol != NULL) {
        return symbol->type;
    }
```

Como essas estruturas se conversam?

```
%token <symbol> IDENT
```

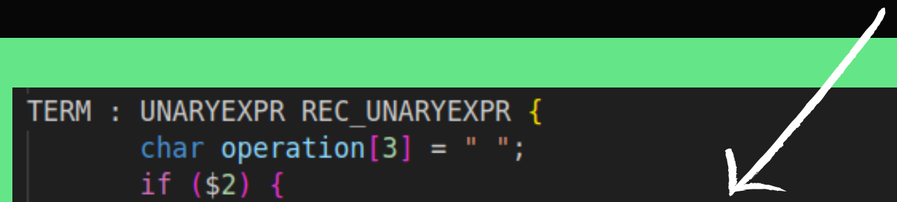
coloca o tipo que encontrou em node.result!

```
FUNCCALL_OR_EXPRESSION : IDENT FOLLOW_IDENT {
                        node new_node;
                        strcpy(new_node.node_before, $1);
                        new_node.result = get_var_type($1);

                        if ($2 != NULL && (strcpy($2->node.operation, "0") == 0)) {
                                strcpy(new_node.operation, $2->vector);

                                char * result_type = check_operation(new_node.result, $2->node.result, $2->node.operation);
                                if (strcmp(result_type, "0") == 0) {
                                        yyerror(" ");
                                        YYABORT;
                                }
                                node new_right_node_result = {new_node.result, $2->node.result, $2->node.operation, result_type};
                                new_node = new_right_node_result;

                                num_expressions[top_num_expressions] = new_node;
                                top_num_expressions += 1;
                        }
                };
```

Verificação de tipos

```
113   %type <node> OPT_REL_OP_NUM_EXPR
114   %type <symbol> REL_OP
115   %type <scope_and_expressions> NUMEXPRESSION
116   %type <scope_and_expressions> REC_PLUS_MINUS_TERM
117   %type <scope_and_expressions> PLUS_OR_MINUS
118   %type <scope_and_expressions> TERM
```

Verificação de tipos



```
TERM : UNARYEXPR REC_UNARYEXPR {
        char operation[3] = " ";
        if ($2) {
                char* result_type = check_operation($1->node.result, $2->node.result, $2->node.operation);
                if (strcmp(result_type, "0") == 0) {
                        yyerror(" ");
                        YYABORT;
                }
                node new_node;
                strcpy(new_node.node_before, $1->node.result);
                strcpy(new_node.node_after, $2->node.result);
                strcpy(new_node.operation, $2->node.operation);
                new_node.result = result_type;

                scope_and_expressions * this_scope = malloc(sizeof(scope_and_expressions));
                this_scope->node = new_node;
                strcpy(this_scope->operation, $2->node.operation);
                $$ = this_scope;
        } else {
                $$ = $1;
        }
};
```

Verificação de tipos

```
char * check_operation(char* node_before, char* node_after, char* operation) {
    if (strcmp(node_before, node_after) == 0) {
        if (strcmp(node_before, "string") == 0) {
            if (strcmp(operation, "+") != 0) {
                printf("Operator between strings is not +");
            }
            return node_before;
        }

    }

    printf("Error: Invalid Type Operation between %s %s %s!\n", node_before, op
    return "0";
}
```

Verificação de tipos

```
{elemento esquerda, elemento direita, operador, resultado}
{"string", "string", "+", "string"},
{"int", "int", "+", "int"},
{"int", "int", "-", "int"},
{"int", "int", "*", "int"},
{"int", "int", "%", "int"},
{"int", "int", "/", "int"},
{"float", "float", "+", "float"},
{"float", "float", "-", "float"},
{"float", "float", "*", "float"},
{"float", "float", "%", "float"},
{"float", "float", "/", "float"},
```

Verificação de identificadores por escopo

Verificação de identificadores por escopo

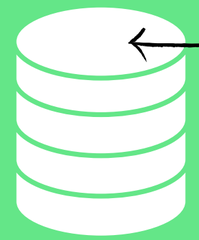
```
%token <symbol> IDENT
```

```
FUNCCALL_OR_EXPRESSION : IDENT FOLLOW_IDENT {
                        node new_node;
                        strcpy(new_node.node_before, $1);
                        new_node.result = get_var_type($1);

                        if ($2 != NULL && (strcpy($2->node.operation, "0") == 0)) {
                                strcpy(new_node.operation, $2->vector);

                                char * result_type = check_operation(new_node.result, $2->node.result, $2->node.operation);
                                if (strcmp(result_type, "0") == 0) {
                                        yyerror(" ");
                                        YYABORT;
                                }
                                node new_right_node_result = {new_node.result, $2->node.result, $2->node.operation, result_type};
                                new_node = new_right_node_result;

                                num_expressions[top_num_expressions] = new_node;
                                top_num_expressions += 1;
                        }
                };
```

Verificação de identificadores por escopo

Pega o escopo atual (topo da pilha de escopos)

```
scope peek(){
    scope x = scopes[top];
    return x;
}
```

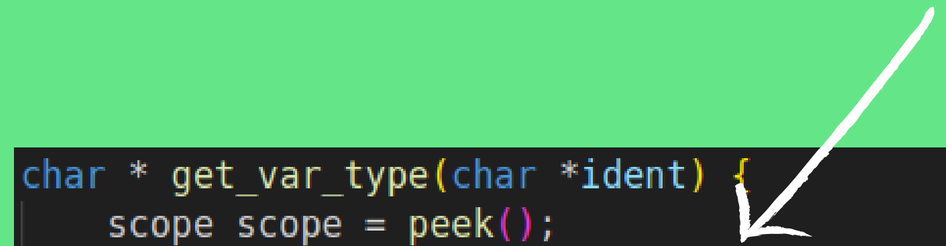


peek

```
char * get_var_type(char *ident) {
    scope scope = peek();
    sst* symbol = lookup_sst_symbol(scope.symbol_table, scope.num_symbols, ident);
    if (symbol != NULL) {
       return symbol->type;
    }

    printf("Error: Variable %s was not declared!\n", ident);
    return NULL;
}
```

Verificação de identificadores por escopo

Procura dentro do escopo, se o identificador existe em sua symbol table



```c
char * get_var_type(char *ident) {
    scope scope = peek();
    sst* symbol = lookup_sst_symbol(scope.symbol_table, scope.num_symbols, ident);
    if (symbol != NULL) {
        return symbol->type;
    }
```

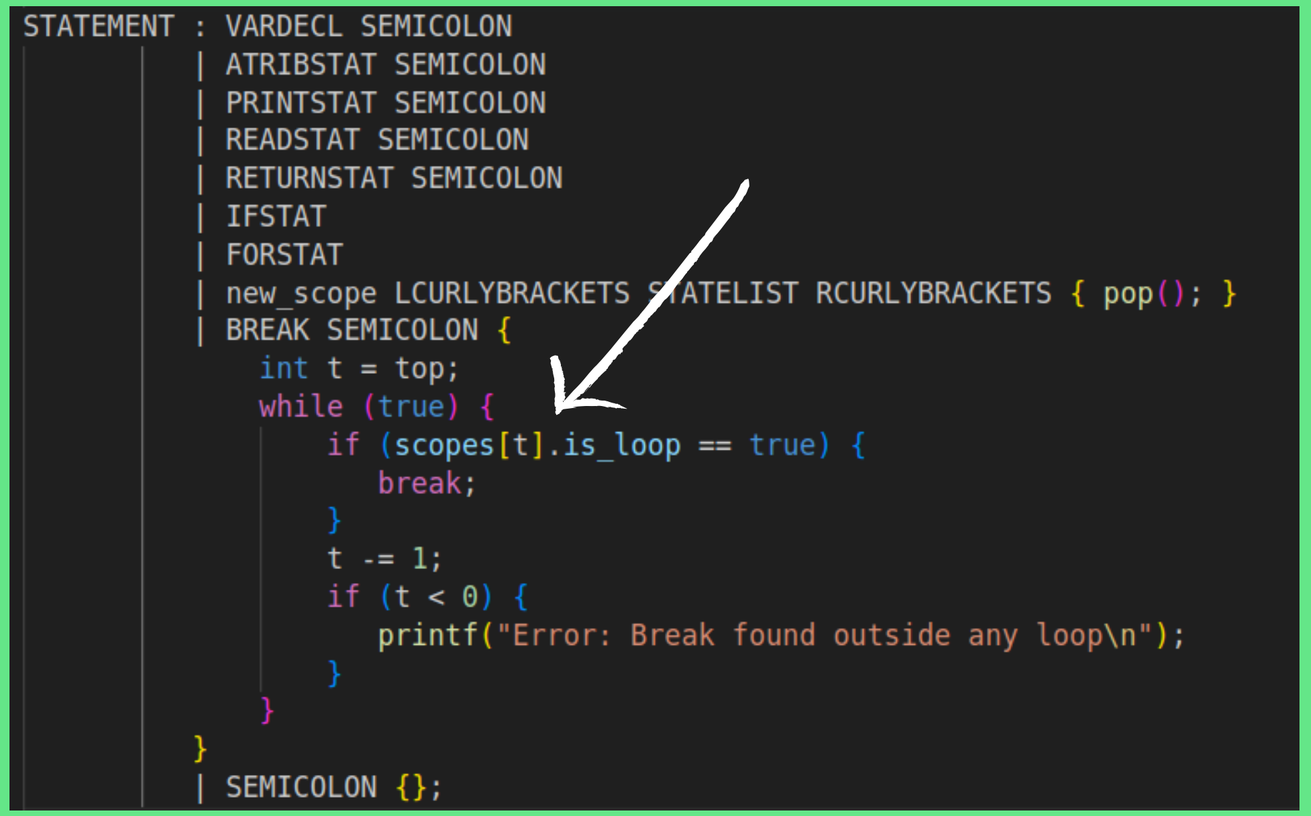
```c
sst* lookup_sst_symbol(sst *symbol_table, int num_symbols, char* symbol) {
    int i;
    sst * teste = symbol_table;
    for (i = 0; i < num_symbols; i++) {
        if (strcmp(symbol_table[i].symbol, symbol) == 0) {
            return &symbol_table[i];
        }
    }
    return NULL;
}
```

Verificação de identificadores por escopo

```
LVALUE : IDENT OPT_ALLOC_NUMEXP {
        scope_and_expressions* this_scope = malloc(sizeof(scope_and_expressions));
        this_scope->node.operation = malloc(strlen($1) + strlen($2) + 1); // Allocate memory for concatenated string
        strcpy(this_scope->node.operation, $1); // Copy $1 into the operation string
        strcat(this_scope->node.operation, $2); // Concatenate $2 to the operation string
        this_scope->node.result = get_var_type($1);
        $$ = this_scope;
};
```

Verificando o break

Verificando o break



```
STATEMENT : VARDECL SEMICOLON
          | ATRIBSTAT SEMICOLON
          | PRINTSTAT SEMICOLON
          | READSTAT SEMICOLON
          | RETURNSTAT SEMICOLON
          | IFSTAT
          | FORSTAT
          | new_scope LCURLYBRACKETS STATELIST RCURLYBRACKETS { pop(); }
          | BREAK SEMICOLON {
              int t = top;
              while (true) {
                  if (scopes[t].is_loop == true) {
                     break;
                  }
                  t -= 1;
                  if (t < 0) {
                     printf("Error: Break found outside any loop\n");
                  }
              }
            }
          | SEMICOLON {};
```

Dificuldades na análise semântica

Dificuldades na análise semântica

Gastamos muito tempo consertando erros e entendendo como trabalhar com strings (char) e ponteiros na linguagem C

```
scope_and_expressions * this_scope = malloc(sizeof(scope_and_expressions));
this_scope->node.value.i = $1;
strcpy(this_scope->node.result,"int");
```

```
typedef struct {
    char * node_before;
    char * node_after;
    char operation[3];
    char * result;
    union Value value;
} node;
```

```
%union{
  int address;
  char symbol[50];
  int usage_count;
  int integer_return;
  float float_return;
  struct recursive_list *recursive_list;
  struct node *node;
  struct scope_and_expressions *scope_and_expressions;
}
```

Dificuldades na análise semântica

Função yylex(), se descomentada, fazia o analisador sintático seguir para as produções erradas!

```c
int main() {
    char str[20];
    int isExit = 1;

    while (isExit != 0) {
        printf("Please, type the file path or exit to leave: ");
        scanf("%[^\n]%*c", str);

        int isExit = strcmp(str, "exit");
        if (isExit != 0) {
            yyin = fopen("./tests/test1.txt","r");
            if (yyin) {
                //yylex();
                yyparse();
                if(valid){
                    printf("\nSatisfies the grammar\n");
                    print_table();
                    print_tokens();
                } else {
                    break;
                }
            } else {
                printf("Can not find this file!\n");
            }
        } else {
            break;
        }
    }

    return 0;
```

test1.txt

tests > test1.txt

```
1  def perimetro(int a, int b, int c) {
2      int perimetro;
3      perimetro = a + b + c;
```

```
PROGRAM : new_scope STATEMENT {
                pop();
                show_tables();
        }
        | new_scope FUNCLIST {
                pop();
                show_tables();
        }
        | { show_tables(); }
        ;
```

Dificuldades na análise semântica

Colocamos para mostrar as tabelas de símbolos por escopo, mas não conseguimos corrigir nosso código a tempo, para um caso de teste dar certo e mostrá-las!

```
void show_tables() {
    int i;
    printf("Symbol Table\n");
    for (i = 0; i < top; i++) {
        print_sst_table(all_scopes[i].symbol_table, all_scopes[i].num_symbols);
    }
}
```

INE 5426

Trabalho 3: Gerador de C.I.

SDT para Geração de CI

FUNCLIST -> FUNCDEF FUNCLISTAUX	FUNCLIST.code = FUNCDEF.code + FUNCLISTAUX.code
FUNCLISTAUX -> FUNCLIST	FUNCLISTAUX.code = FUNCLIST.code
FUNCLISTAUX -> &	FUNCLISTAUX.code = “ “
FUNCDEF -> DEF ident (PARAMLIST) { STATELIST }	jump_lable = new_lable(); FUNCDEF.code = “goto “ + jump_lable + “\n” + ident + “:\n” + PARAMLIST.code + STATELIST.code + “\n” + jump_label + “ :\n”

SDT para Geração de CI

PARAMLIST -> DATATYPE ident PARAMLISTAUX	PARAMLIST.code = “param “ + ident + PARAMLISTAUX.code
PARAMLIST -> &	PARAMLIST.code = “ “;
PARAMLISTAUX -> comma PARAMLIST	PARAMLISTAUX.code = “, “ + PARAMLIST.code
PARAMLISTAUX -> &	PARAMLIST AUX.code = “\n“
STATEMENT -> VARDECL ;	STATEMENT.code = VARDECL.code
STATEMENT -> ATRIBSTAT ;	STATEMENT.code = ATRIBSTAT.code

SDT para Geração de CI

STATEMENT -> PRINTSTAT ;	STATEMENT.code = PRINTSTAT.code
STATEMENT -> READSTAT ;	STATEMENT.code = READSTAT.code
STATEMENT -> RETURNSTAT ;	STATEMENT.code = RETURNSTAT.code
STATEMENT -> IFSTAT	STATEMENT.code = IFSTAT.code
STATEMENT -> FORSTAT	STATEMENT.code = FOR STATE.code
STATEMENT -> { STATELIST }	STATEMENT.code = STATELIST.code

SDT para Geração de CI

STATEMENT -> BREAK ;	last_loop_end_label = get_last_loop_end_label() STATEMENT.code = “goto ” + last_loop_end_label + “\n”;
STATEMENT -> ;	STATEMENT.code = “ “

SDT para Geração de CI

- Expmplo de entrada e saida (arivo tests/test5.txt):

```
def print_oi() {
    string a;
    a = "OI";
    print a;
    return;
}

def main() {
    int retorno;

    retorno = print_oi();
}
```

```
goto LABEL0
print_oi :
 string a
t1 = "OI"
 a = t1
 t2 = a
 t3 = t2
t4 = t3
 t5 = t4
print t5
return

LABEL0 :
goto LABEL1
main :
 int retorno
 t7 = call print_oi
 retorno = t7

LABEL1 :
```

SDT para Geração de CI

- A implementação completa da SDT esta no código fonte no arquivo src/code.y
- Para rodar o gerador de códe deve-se utilizar o comando "make code". Então basta passar no terminal caminho para o código fonte.
- O código intermediário gerado aprecerá no terminal.

SDT para Geração de CI dificuldades

- Dureante a geração de código intermediário ainda exitem alguns bugs na criação de código para expressões.
- Acreditamos que estes bud estejam ocorrendo devido a erros com a utilização des strings em C no código da SDT.
- O grupo identificou e corrigiu alguns destes erros, porém para as expressões, não conseguimos identificar todos os erros a tempo.