Compiladores - Turma A - 0225 - Noturno
Prof. Thiago Fernandes Neves

ENTREGA LAB06

Integrantes:

- Gabriel Andrade Silva Pinto
- Isabela Garcia
- Joao Victor Azevedo dos Santos
- Luiz Eduardo Campos Dias
- Yago Peres dos Santos

Link para o Github

1. **Atividade 1 - nova de declarar variáveis**
   Forma proposta: **A igual int;**

```
void parse_variable_igual_syntax(struct history* history, struct token*
name_token) {
    // Verificar se o token do nome é válido
    if (name_token->type != TOKEN_TYPE_IDENTIFIER) {
        compiler_error(current_process, "Esperado nome da variável antes
de 'igual'\n");
        return;
    }

    // Segundo token deve ser "igual"
    struct token* igual_token = token_next();
    if (!token_is_keyword(igual_token, "igual")) {
        compiler_error(current_process, "Esperado 'igual' após nome da
variável\n");
        return;
    }

    // Terceiro token deve ser o tipo de dados
    struct token* type_token = token_next();
    if (!keyword_is_datatype(type_token->sval)) {
        compiler_error(current_process, "Esperado tipo de dados após
'igual'\n");
        return;
    }

    // Criar o datatype baseado no tipo especificado
    struct datatype dtype;
```

```
    memset(&dtype, 0, sizeof(struct datatype));

    if (S_EQ(type_token->sval, "int")) {
        dtype.type = DATATYPE_INTEGER;
        dtype.type_str = "int";
        dtype.size = sizeof(int);
    } else if (S_EQ(type_token->sval, "float")) {
        dtype.type = DATATYPE_FLOAT;
        dtype.type_str = "float";
        dtype.size = sizeof(float);
    } else if (S_EQ(type_token->sval, "double")) {
        dtype.type = DATATYPE_DOUBLE;
        dtype.type_str = "double";
        dtype.size = sizeof(double);
    } else if (S_EQ(type_token->sval, "char")) {
        dtype.type = DATATYPE_CHAR;
        dtype.type_str = "char";
        dtype.size = sizeof(char);
    } else if (S_EQ(type_token->sval, "bool")) {
        dtype.type = DATATYPE_INTEGER; // bool é tratado como int
        dtype.type_str = "bool";
        dtype.size = sizeof(int);
    } else if (S_EQ(type_token->sval, "void")) {
        dtype.type = DATATYPE_VOID;
        dtype.type_str = "void";
        dtype.size = 0;
    } else {
        compiler_error(current_process, "Tipo de dados não suportado:
%s\n", type_token->sval);
        return;
    }

    // Processar a variável normalmente
    parse_variable(&dtype, name_token, history);
}
```

2. **Atividade 2 - nova de declarar structs**
   Forma proposta: **struct ABC /int A\ /int B\ /int C\;**

```
void parse_struct_nova_sintaxe(struct history* history, struct token*
struct_name) {
    struct vector* members = vector_create(sizeof(struct node*));
    if (!members) {
        compiler_error(current_process, "Falha ao criar lista de membros
da struct\n");
        return;
    }

    int member_count = 0;
    while (1) {
        // Ignorar quebras de linha e comentários
        struct token* slash_open = token_next();
        while (slash_open && (slash_open->type == TOKEN_TYPE_NEWLINE ||
slash_open->type == TOKEN_TYPE_COMMENT)) {
            slash_open = token_next();
        }
        if (!slash_open) break;
        if (token_is_symbol(slash_open, ';')) {
            // Fim da struct logo após o último membro
            break;
        }
        if (!token_is_operator(slash_open, "/")) {
            compiler_error(current_process, "Esperado '/' para iniciar
membro da struct\n");
            break;
        }
        // Ler o tipo de dados
        struct token* type_token = token_next();
        if (!keyword_is_datatype(type_token->sval)) {
            compiler_error(current_process, "Esperado tipo de dados após
'/'\n");
            break;
        }

        // Criar o datatype
        struct datatype member_type;
        memset(&member_type, 0, sizeof(struct datatype));
```

```
        if (S_EQ(type_token->sval, "int")) {
            member_type.type = DATATYPE_INTEGER;
            member_type.type_str = "int";
            member_type.size = sizeof(int);
        } else if (S_EQ(type_token->sval, "float")) {
            member_type.type = DATATYPE_FLOAT;
            member_type.type_str = "float";
            member_type.size = sizeof(float);
        } else if (S_EQ(type_token->sval, "double")) {
            member_type.type = DATATYPE_DOUBLE;
            member_type.type_str = "double";
            member_type.size = sizeof(double);
        } else if (S_EQ(type_token->sval, "char")) {
            member_type.type = DATATYPE_CHAR;
            member_type.type_str = "char";
            member_type.size = sizeof(char);
        } else if (S_EQ(type_token->sval, "bool")) {
            member_type.type = DATATYPE_INTEGER;
            member_type.type_str = "bool";
            member_type.size = sizeof(int);
        } else {
            compiler_error(current_process, "Tipo de dados não
suportado: %s\n", type_token->sval);
            break;
        }

        // Ler o nome do membro
        struct token* member_name = token_next();
        if (!member_name) {
            compiler_error(current_process, "member_name é NULL\n");
            break;
        }

        // Verificar se é um ponteiro
        if (member_name->type == TOKEN_TYPE_OPERATOR &&
S_EQ(member_name->sval, "*")) {
            // É um ponteiro, ler o nome do membro
            member_type.pointer_depth = 1;
            member_name = token_next();
            if (!member_name || member_name->type !=
TOKEN_TYPE_IDENTIFIER) {
```

```
                compiler_error(current_process, "Esperado nome do membro
após '*'\n");
                break;
            }
        }

        if (member_name->type != TOKEN_TYPE_IDENTIFIER) {
            compiler_error(current_process, "Esperado nome do membro da
struct\n");
            break;
        }

        // Processar arrays se houver
        struct datatype* current_dtype = &member_type;
        while (token_next_is_operator("[")) {
            struct token* abre = token_next(); // Consome o '['
            struct token* size_token = token_next();
            if (size_token->type != TOKEN_TYPE_NUMBER) {
                compiler_error(current_process, "Esperado número como
tamanho do array\n");
                break;
            }
            current_dtype->flags |= DATATYPE_FLAG_IS_ARRAY;
            current_dtype->size = size_token->inum;
            struct token* close_bracket = token_next();
            if (!token_is_operator(close_bracket, "]")) {
                compiler_error(current_process, "Esperado ']' após
tamanho do array\n");
                break;
            }
            // Se houver mais dimensões, criar datatype_secondary
            if (token_next_is_operator("[")) {
                current_dtype->datatype_secondary = calloc(1,
sizeof(struct datatype));
                current_dtype = current_dtype->datatype_secondary;
                memset(current_dtype, 0, sizeof(struct datatype));
                current_dtype->type = member_type.type;
                current_dtype->type_str = member_type.type_str;
                current_dtype->size = 0;
            }
        }
```

```
        // Criar o node do membro
        make_variable_node_simple(&member_type, member_name, NULL);
        struct node* member_node = node_pop();
        if (!member_node) {
            compiler_error(current_process, "Falha ao criar node para
membro da struct\n");
            break;
        }
        vector_push(members, &member_node);
        member_count++;
        printf("Membro adicionado: %s %s\n", member_type.type_str,
member_name->sval);

        // Agora, garantir que o próximo token seja '\\' e consumi-lo
        struct token* slash_close = token_peek_next();
        if (!slash_close) {
            compiler_error(current_process, "token_peek_next() retornou
NULL\n");
            break;
        }

        if (slash_close->type != TOKEN_TYPE_SYMBOL || slash_close->cval
!= '\\') {
            compiler_error(current_process, "Esperado '\\' para fechar
membro da struct, encontrado: %c\n", slash_close->cval);
            break;
        }

        slash_close = token_next(); // Consome o '\\'

        // Ignorar quebras de linha e comentários após o fechamento
        struct token* next_token = token_peek_next();
        while (next_token && (next_token->type == TOKEN_TYPE_NEWLINE ||
next_token->type == TOKEN_TYPE_COMMENT)) {
            token_next();
            next_token = token_peek_next();
        }

        // Verificar o próximo token após ignorar quebras de
linha/comentários
        if (next_token && token_is_operator(next_token, "/")) {
            // Há mais membros, continuar o loop
```

```
            continue;
        }
        if (next_token && token_is_symbol(next_token, ';')) {
            // Fim da struct
            token_next(); // Consome o ';'
            break;
        }
        if (next_token) {
            compiler_error(current_process, "Esperado '/' para novo
membro ou ';' para finalizar struct\n");
            break;
        }
        // Se não há próximo token, finalizar a struct
        break;
    }

    struct node struct_node = {
        .type = NODE_TYPE_STRUCT,
        .sval = struct_name->sval,
        .pos = struct_name->pos
    };

    struct node* created_struct = node_create(&struct_node);
    if (!created_struct) {
        compiler_error(current_process, "Falha ao criar node da
struct\n");
        vector_free(members);
        return;
    }

    printf("Struct criada com %d membros\n", member_count);
    node_push(created_struct);
    return;
}
```

3. **Atividade 3 - nova de declarar if/else**
   Forma proposta: **if A>B ? return 0 ? return 1;**

```
void parse_if_nova_sintaxe(struct history* history) {
    // A condição já foi processada, apenas pegá-la da pilha
    struct node* condition = node_pop();
    if (!condition) {
        compiler_error(current_process, "Condição do if não encontrada
na pilha\n");
        return;
    }

    // Consumir o primeiro '?'
    struct token* first_question = token_next();
    if (!first_question || !token_is_operator(first_question, "?")) {
        compiler_error(current_process, "Esperado '?' após condição do
if\n");
        return;
    }

    // Processar a ação se verdadeiro de forma simples
    struct node* true_action = NULL;

    // Ler tokens até encontrar o segundo '?'
    char action_buffer[256] = {0};
    bool first_token = true;

    while (1) {
        struct token* next_token = token_peek_next();
        if (!next_token) break;

        if (token_is_operator(next_token, "?")) {
            break;
        }

        // Concatenar tokens para formar a ação completa
        if (!first_token) {
            strcat(action_buffer, " ");
        }

        if (next_token->type == TOKEN_TYPE_KEYWORD) {
            strcat(action_buffer, next_token->sval);
```

```
        } else if (next_token->type == TOKEN_TYPE_IDENTIFIER) {
            strcat(action_buffer, next_token->sval);
        } else if (next_token->type == TOKEN_TYPE_OPERATOR) {
            strcat(action_buffer, next_token->sval);
        } else if (next_token->type == TOKEN_TYPE_NUMBER) {
            char numbuf[32];
            snprintf(numbuf, sizeof(numbuf), "%llu", next_token->llnum);
            strcat(action_buffer, numbuf);
        }

        first_token = false;
        token_next(); // Consumir o token
    }

    if (strlen(action_buffer) > 0) {
        true_action = node_create(&(struct node){
            .type = NODE_TYPE_IDENTIFIER,
            .sval = strdup(action_buffer)
        });
    }

    if (!true_action) {
        true_action = node_create(&(struct node){
            .type = NODE_TYPE_IDENTIFIER,
            .sval = "ação_verdadeira"
        });
    }

    // Consumir o segundo '?'
    struct token* second_question = token_next();
    if (!second_question || !token_is_operator(second_question, "?")) {
        compiler_error(current_process, "Esperado '?' após ação se
verdadeiro\n");
        return;
    }

    // Processar a ação se falso de forma simples
    struct node* false_action = NULL;

    // Ler tokens até encontrar o ';'
    char false_action_buffer[256] = {0};
    first_token = true;
```

```
while (1) {
    struct token* next_token = token_peek_next();
    if (!next_token) break;

    if (token_is_symbol(next_token, ';')) {
        break;
    }

    // Concatenar tokens para formar a ação completa
    if (!first_token) {
        strcat(false_action_buffer, " ");
    }

    if (next_token->type == TOKEN_TYPE_KEYWORD) {
        strcat(false_action_buffer, next_token->sval);
    } else if (next_token->type == TOKEN_TYPE_IDENTIFIER) {
        strcat(false_action_buffer, next_token->sval);
    } else if (next_token->type == TOKEN_TYPE_OPERATOR) {
        strcat(false_action_buffer, next_token->sval);
    } else if (next_token->type == TOKEN_TYPE_NUMBER) {
        char numbuf[32];
        snprintf(numbuf, sizeof(numbuf), "%llu", next_token->llnum);
        strcat(false_action_buffer, numbuf);
    }

    first_token = false;
    token_next(); // Consumir o token
}

if (strlen(false_action_buffer) > 0) {
    false_action = node_create(&(struct node){
        .type = NODE_TYPE_IDENTIFIER,
        .sval = strdup(false_action_buffer)
    });
}

if (!false_action) {
    false_action = node_create(&(struct node){
        .type = NODE_TYPE_IDENTIFIER,
        .sval = "ação_falsa"
    });
```

```
    }

    // Consumir o ';'
    struct token* semicolon = token_next();
    if (!semicolon || !token_is_symbol(semicolon, ';')) {
        compiler_error(current_process, "Esperado ';' após ação se
falso\n");
        return;
    }

    // Criar o node do if com a nova sintaxe, armazenando os filhos
    struct node if_node = {
        .type = NODE_TYPE_STATEMENT_IF,
        .pos = condition ? condition->pos : (struct pos){0, 0, NULL},
        .exp = {
            .left = condition,
            .right = true_action,
            .op = NULL
        },
        .any = false_action
    };

    struct node* created_if = node_create(&if_node);
    if (!created_if) {
        compiler_error(current_process, "Falha ao criar node do if\n");
        return;
    }

    node_push(created_if);
}
```

test.c:

```
// Variáveis
A igual int;
matriz igual float[5][5];

// Structs
struct ABC / int A\ / int B\ / int C\;
struct Pessoa / int idade\ / float altura\ / char nome[50]\;
struct Complexo / int *ponteiro\ / float matriz[3][3]\;

// If/Else
if c ? return 0 ? return 1;
if x == y ? x = x + 1 ? x = x - 1;
if a > b ? a = a + 1 ? a = a - 1;
```

output:

```
./main test.c
Compiladores - TURMA A - GRUPO 7

#Input file: test.c
#Output file: (null)

TOKEN   CO:
TOKEN   NL
TOKEN   ID: A
TOKEN   KE: igual
TOKEN   KE: int
TOKEN   SY: ;
TOKEN   NL
TOKEN   ID: matriz
TOKEN   KE: igual
TOKEN   KE: float
TOKEN   OP: [
TOKEN   NU: 5    PARENTESES: (null)
TOKEN   SY: ]
TOKEN   OP: [
TOKEN   NU: 5    PARENTESES: (null)
TOKEN   SY: ]
TOKEN   SY: ;
TOKEN   NL
TOKEN   NL
```

```
TOKEN   CO:
TOKEN   NL
TOKEN   KE: struct
TOKEN   ID: ABC
TOKEN   OP: /
TOKEN   KE: int
TOKEN   ID: A
TOKEN   SY: \
TOKEN   OP: /
TOKEN   KE: int
TOKEN   ID: B
TOKEN   SY: \
TOKEN   OP: /
TOKEN   KE: int
TOKEN   ID: C
TOKEN   SY: \
TOKEN   SY: ;
TOKEN   NL
TOKEN   KE: struct
TOKEN   ID: Pessoa
TOKEN   OP: /
TOKEN   KE: int
TOKEN   ID: idade
TOKEN   SY: \
TOKEN   OP: /
TOKEN   KE: float
TOKEN   ID: altura
TOKEN   SY: \
TOKEN   OP: /
TOKEN   KE: char
TOKEN   ID: nome
TOKEN   OP: [
TOKEN   NU: 50   PARENTESES: (null)
TOKEN   SY: ]
TOKEN   SY: \
TOKEN   SY: ;
TOKEN   NL
TOKEN   KE: struct
TOKEN   ID: Complexo
TOKEN   OP: /
TOKEN   KE: int
TOKEN   OP: *
```

```
TOKEN   ID: ponteiro
TOKEN   SY: \
TOKEN   OP: /
TOKEN   KE: float
TOKEN   ID: matriz
TOKEN   OP: [
TOKEN   NU: 3    PARENTESES: (null)
TOKEN   SY: ]
TOKEN   OP: [
TOKEN   NU: 3    PARENTESES: (null)
TOKEN   SY: ]
TOKEN   SY: \
TOKEN   SY: ;
TOKEN   NL
TOKEN   NL
TOKEN   CO:
TOKEN   NL
TOKEN   KE: if
TOKEN   ID: c
TOKEN   NL
TOKEN   OP: ?
TOKEN   KE: return
TOKEN   NU: 0    PARENTESES: (null)
TOKEN   OP: ?
TOKEN   KE: return
TOKEN   NU: 1    PARENTESES: (null)
TOKEN   SY: ;
TOKEN   NL
TOKEN   KE: if
TOKEN   ID: x
TOKEN   NL
TOKEN   OP: ==
TOKEN   ID: y
TOKEN   OP: ?
TOKEN   ID: x
TOKEN   OP: =
TOKEN   ID: x
TOKEN   OP: +
TOKEN   NU: 1    PARENTESES: (null)
TOKEN   OP: ?
TOKEN   ID: x
TOKEN   OP: =
```

```
TOKEN   ID: x
TOKEN   OP: -
TOKEN   NU: 1    PARENTESES: (null)
TOKEN   SY: ;
TOKEN   NL
TOKEN   KE: if
TOKEN   ID: a
TOKEN   NL
TOKEN   OP: >
TOKEN   ID: b
TOKEN   OP: ?
TOKEN   ID: a
TOKEN   OP: =
TOKEN   ID: a
TOKEN   OP: +
TOKEN   NU: 1    PARENTESES: (null)
TOKEN   OP: ?
TOKEN   ID: a
TOKEN   OP: =
TOKEN   ID: a
TOKEN   OP: -
TOKEN   NU: 1    PARENTESES: (null)
TOKEN   SY: ;
TOKEN   NL


Arvore de nodes:
└── VARIABLE_LIST (count: 1)
|   └── VARIABLE (name: A, type: int)


└── VARIABLE_LIST (count: 1)
|   └── VARIABLE (name: matriz, type: float, ARRAY[5][5])


Processando struct:
Nome da struct: ABC
Membro adicionado: int A
Membro adicionado: int B
Membro adicionado: int C
Struct criada com 3 membros
└── STRUCT (name: ABC)
```

```
Processando struct:
Nome da struct: Pessoa
Membro adicionado: int idade
Membro adicionado: float altura
Membro adicionado: char nome
Struct criada com 3 membros
└── STRUCT (name: Pessoa)


Processando struct:
Nome da struct: Complexo
Membro adicionado: int ponteiro
Membro adicionado: float matriz
Struct criada com 2 membros
└── STRUCT (name: Complexo)


└── IF_STATEMENT (nova sintaxe)
|   ├── CONDITION
|   |   └── IDENTIFIER (c)
|   ├── TRUE_ACTION
|   |   └── IDENTIFIER (return 0)
|   └── FALSE_ACTION
|   |   └── IDENTIFIER (return 1)


└── IF_STATEMENT (nova sintaxe)
|   ├── CONDITION
|   |   └── IDENTIFIER (y)
|   ├── TRUE_ACTION
|   |   └── IDENTIFIER (x = x + 1)
|   └── FALSE_ACTION
|   |   └── IDENTIFIER (x = x - 1)


└── IF_STATEMENT (nova sintaxe)
|   ├── CONDITION
|   |   └── IDENTIFIER (b)
|   ├── TRUE_ACTION
|   |   └── IDENTIFIER (a = a + 1)
```

```
|    └── FALSE_ACTION
|    |    └── IDENTIFIER (a = a - 1)


Todos os arquivos foram compilados com sucesso!
```