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

ENTREGA LAB02

Integrantes: Link para o Github

- Isabela Garcia
- Joao Victor Azevedo dos Santos
- Luiz Eduardo Campos Dias
- Yago Peres dos Santos

1. **Atividade 1 - Adicionar os arquivos novos (lex_process.c e lexer.c) ao Makefile e compilar sem erros**
   lex_process.c:

```c
/* BEGIN - LAB 2 ------------------------------------------*/
#include "compiler.h"
#include "helpers/vector.h"
#include <stdlib.h>

struct lex_process* lex_process_create(struct compile_process* compiler,
struct lex_process_functions* functions, void *private) {
    struct lex_process* process = calloc(1, sizeof(struct lex_process));

    process->function = functions;
    process->token_vec = vector_create(sizeof(struct token));
    process->compiler = compiler;
    process->private = private;
    process->pos.line = 1;
    process->pos.col = 1;

    return process;
}

void lex_process_free(struct lex_process* process) {
    vector_free(process->token_vec);
    free(process);
}

void* lex_process_private(struct lex_process* process) {
    return process->private;
}
```

```
struct vector* lex_process_tokens(struct lex_process* process){
    return process->token_vec;
}
/* END - LAB 2 ------------------------------------------------*/
```

lexer.c:

```
/* BEGIN - LAB 2 ------------------------------------------------*/
#include "compiler.h"

int lex(struct lex_process* process) {

  return 0;
}



/* END - LAB 2 ------------------------------------------------*/
```

Makefile:

```
OBJECTS=./build/compiler.o ./build/cprocess.o ./build/helpers/buffer.o
./build/helpers/vector.o ./build/lex_process.o ./build/lexer.o
INCLUDES= -I./

all: ${OBJECTS}
  gcc main.c ${INCLUDES} ${OBJECTS} -g -o ./main

./build/compiler.o: ./compiler.c
  gcc ./compiler.c ${INCLUDES} -o ./build/compiler.o -g -c

./build/cprocess.o: ./cprocess.c
  gcc ./cprocess.c ${INCLUDES} -o ./build/cprocess.o -g -c

./build/helpers/buffer.o: ./helpers/buffer.c
  gcc ./helpers/buffer.c ${INCLUDES} -o ./build/helpers/buffer.o -g -c

./build/helpers/vector.o: ./helpers/vector.c
  gcc ./helpers/vector.c ${INCLUDES} -o ./build/helpers/vector.o -g -c

./build/lex_process.o: ./lex_process.c
  gcc ./lex_process.c ${INCLUDES} -o ./build/lex_process.o -g -c
```

```
./build/lexer.o: ./lexer.c
  gcc ./lexer.c ${INCLUDES} -o ./build/lexer.o -g -c

clean:
  rm ./main
  rm -rf ${OBJECTS}
```

output:

```
❯ make all
gcc ./compiler.c -I./ -o ./build/compiler.o -g -c
gcc ./cprocess.c -I./ -o ./build/cprocess.o -g -c
gcc ./helpers/buffer.c -I./ -o ./build/helpers/buffer.o -g -c
gcc ./helpers/vector.c -I./ -o ./build/helpers/vector.o -g -c
gcc ./lex_process.c -I./ -o ./build/lex_process.o -g -c
gcc ./lexer.c -I./ -o ./build/lexer.o -g -c
gcc main.c -I./ ./build/compiler.o ./build/cprocess.o
./build/helpers/buffer.o ./build/helpers/vector.o ./build/lex_process.o
./build/lexer.o -g -o ./main
❯ make clean
rm ./main
rm -rf ./build/compiler.o ./build/cprocess.o ./build/helpers/buffer.o
./build/helpers/vector.o ./build/lex_process.o ./build/lexer.o
```

2. **Atividade 2 - Atualizar a função read_next_token() para gerar 3 novos tipos de tokens: TOKEN_TYPE_KEYWORD, TOKEN_TYPE_IDENTIFIER, TOKEN_TYPE_OPERATOR, TOKEN_TYPE_SYMBOL, TOKEN_TYPE_STRING, TOKEN_TYPE_COMMENT, TOKEN_TYPE_NEWLINE**

read_next_token():

```
struct token* read_next_token() {
  struct token* token = NULL;
  char c = peekc();

  switch (c) {
    case EOF:
      // Fim do arquivo
      break;

    // TOKEN_TYPE_COMMENT_CASE ou operador '/'
    case '/':
      nextc(); // Consome o primeiro '/'
      if (peekc() == '/') {
        // Trata como comentário de linha
```

```
        nextc(); // Consome o segundo '/'
        while ((c = peekc()) != '\n' && c != EOF) {
          nextc(); // Consome o restante da linha
        }
        return read_next_token(); // Ignora o comentário e continua para
o próximo token
      } else if (peekc() == '*') {
        // Trata como comentário de múltiplas linhas
        nextc(); // Consome o '*'
        while (true) {
          c = nextc();
          if (c == EOF) {
            printf("Erro: Comentário de bloco não fechado!\n");
            break;
          }
          if (c == '*' && peekc() == '/') {
            nextc(); // Consome o '/'
            break; // Fim do comentário de bloco
          }
        }
        return read_next_token(); // Ignora o comentário e continua para
o próximo token
      } else {
        // Adiciona o caractere '/' de volta ao fluxo
        pushc('/');
        token = token_make_operator();
      }
      break;

    // TOKEN_TYPE_NUMERIC_CASE
    NUMERIC_CASE:
      token = token_make_number();
      break;

    // TOKEN_TYPE_KEYWORD_CASE
    KEYWORD_CASE:
      token = token_make_keyword_or_identifier();
      break;

    // TOKEN_TYPE_OPERATOR_CASE
    OPERATOR_CASE:
      token = token_make_operator();
```

```
      break;

    // TOKEN_TYPE_SYMBOL_CASE
    SYMBOL_CASE:
      token = token_make_symbol();
      break;

    // TOKEN_TYPE_STRING_CASE
    STRING_CASE:
      token = token_make_string();
      break;

    // TOKEN_TYPE_NEWLINE_CASE
    case '\n':
      token = token_create(&(struct token){.type = TOKEN_TYPE_NEWLINE});
      nextc();
      break;

    case ' ':
    case '\t':
      token = handle_whitespace();
      break;

    default:
      if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || c == '_')
{
        token = token_make_keyword_or_identifier();
      } else {
        printf("Caractere inválido ignorado: %c\n", c); // Depuração
        nextc();
      }
      break;
  }

  return token;
}
```

test.c:

```
#include <stdio.h>

/* Assim */
```

```c
int main() {
  int a = 10;
  float b = 20.5;
  char c = 'x';

  // Operadores
  a = a + b - c * 2 / 1;

  // Controle de fluxo
  if (a > b) {
    printf("a é maior que b\n");
  } else {
    printf("b é maior ou igual a a\n");
  }

  // Laços
  for (int i = 0; i < 10; i++) {
    printf("Valor de i: %d\n", i);
  }


  // Operadores e símbolos adicionais
  a = (b + c) * 2;
  int arr[5] = {1, 2, 3, 4, 5};

  return 0;
}
```

output:

```
❯ ./main test.c
Compiladores - TURMA A - GRUPO 7
Token: # (SYMBOL)
Token: include (KEYWORD)
Token: < (OPERATOR)
Token: stdio.h (IDENTIFIER)
Token: > (OPERATOR)
Token: int (KEYWORD)
Token: main (IDENTIFIER)
Token: ( (OPERATOR)
Token: ) (SYMBOL)
Token: { (SYMBOL)
Token: int (KEYWORD)
```

```
Token: a (IDENTIFIER)
Token: = (OPERATOR)
Token: 10 (NUMBER)
Token: ; (SYMBOL)
Token: float (KEYWORD)
Token: b (IDENTIFIER)
Token: = (OPERATOR)
Token: 20.5 (NUMBER)
Token: ; (SYMBOL)
Token: char (KEYWORD)
Token: c (IDENTIFIER)
Token: = (OPERATOR)
Token: x (STRING)
Token: ; (SYMBOL)
Token: a (IDENTIFIER)
Token: = (OPERATOR)
Token: a (IDENTIFIER)
Token: + (OPERATOR)
Token: b (IDENTIFIER)
Token: - (OPERATOR)
Token: c (IDENTIFIER)
Token: * (OPERATOR)
Token: 2 (NUMBER)
Token: / (OPERATOR)
Token: 1 (NUMBER)
Token: ; (SYMBOL)
Token: if (KEYWORD)
Token: ( (OPERATOR)
Token: a (IDENTIFIER)
Token: > (OPERATOR)
Token: b (IDENTIFIER)
Token: ) (SYMBOL)
Token: { (SYMBOL)
Token: printf (IDENTIFIER)
Token: ( (OPERATOR)
Token: a é maior que b\n (STRING)
Token: ) (SYMBOL)
Token: ; (SYMBOL)
Token: } (SYMBOL)
Token: else (KEYWORD)
Token: { (SYMBOL)
Token: printf (IDENTIFIER)
```

```
Token: ( (OPERATOR)
Token: b é maior ou igual a a\n (STRING)
Token: ) (SYMBOL)
Token: ; (SYMBOL)
Token: } (SYMBOL)
Token: for (KEYWORD)
Token: ( (OPERATOR)
Token: int (KEYWORD)
Token: i (IDENTIFIER)
Token: = (OPERATOR)
Token: 0 (NUMBER)
Token: ; (SYMBOL)
Token: i (IDENTIFIER)
Token: < (OPERATOR)
Token: 10 (NUMBER)
Token: ; (SYMBOL)
Token: i (IDENTIFIER)
Token: + (OPERATOR)
Token: + (OPERATOR)
Token: ) (SYMBOL)
Token: { (SYMBOL)
Token: printf (IDENTIFIER)
Token: ( (OPERATOR)
Token: Valor de i: %d\n (STRING)
Token: , (SYMBOL)
Token: i (IDENTIFIER)
Token: ) (SYMBOL)
Token: ; (SYMBOL)
Token: } (SYMBOL)
Token: a (IDENTIFIER)
Token: = (OPERATOR)
Token: ( (OPERATOR)
Token: b (IDENTIFIER)
Token: + (OPERATOR)
Token: c (IDENTIFIER)
Token: ) (SYMBOL)
Token: * (OPERATOR)
Token: 2 (NUMBER)
Token: ; (SYMBOL)
Token: int (KEYWORD)
Token: arr (IDENTIFIER)
Token: [ (OPERATOR)
```

```
Token: 5 (NUMBER)
Token: ] (SYMBOL)
Token: = (OPERATOR)
Token: { (SYMBOL)
Token: 1 (NUMBER)
Token: , (SYMBOL)
Token: 2 (NUMBER)
Token: , (SYMBOL)
Token: 3 (NUMBER)
Token: , (SYMBOL)
Token: 4 (NUMBER)
Token: , (SYMBOL)
Token: 5 (NUMBER)
Token: } (SYMBOL)
Token: ; (SYMBOL)
Token: return (KEYWORD)
Token: 0 (NUMBER)
Token: ; (SYMBOL)
Token: } (SYMBOL)
Todos os arquivos foram compilados com sucesso!
```