Trabalho linguagens formais

A função getNextToken() encarregada de obter o próximo token a partir do código-fonte. Ela navega pelo arquivo de entrada e devolve o próximo token, sem levar em conta espaços em branco, quebras de linha ou comentários.

**Token getNextToken() {**

**Token token = { .type = INVALID, .line = current\_line };**

**int c;**

**while ((c = fgetc(source\_code)) != EOF) {**

**if (c == '\n') {**

**current\_line++;**

**} else if (isspace(c)) {**

**continue;**

**} else if (c == '{') {**

**while ((c = fgetc(source\_code)) != EOF && c != '}') {**

**if (c == '\n') current\_line++;**

**}**

**} else {**

**break;**

**}**

**}**

**if (c == EOF) {**

**token.type = EOF\_TOKEN;**

**strcpy(token.lexeme, "EOF");**

**return token;**

**}**

**if (isalpha(c)) {**

**int i = 0;**

**token.lexeme[i++] = c;**

**while (isalnum(c = fgetc(source\_code)) && i < 99) {**

**token.lexeme[i++] = c;**

**}**

**token.lexeme[i] = '\0';**

**ungetc(c, source\_code);**

**if (strcmp(token.lexeme, "program") == 0) token.type = PROGRAM;**

**else if (strcmp(token.lexeme, "var") == 0) token.type = VAR;**

**else if (strcmp(token.lexeme, "integer") == 0) token.type = INTEGER;**

**else if (strcmp(token.lexeme, "real") == 0) token.type = REAL;**

**else if (strcmp(token.lexeme, "begin") == 0) token.type = BEGIN;**

**else if (strcmp(token.lexeme, "end") == 0) token.type = END;**

**else if (strcmp(token.lexeme, "if") == 0) token.type = IF;**

**else if (strcmp(token.lexeme, "then") == 0) token.type = THEN;**

**else if (strcmp(token.lexeme, "else") == 0) token.type = ELSE;**

**else if (strcmp(token.lexeme, "while") == 0) token.type = WHILE;**

**else if (strcmp(token.lexeme, "do") == 0) token.type = DO;**

**else token.type = IDENTIFIER;**

**return token;**

**}**

**if (isdigit(c)) {**

**int i = 0;**

**token.lexeme[i++] = c;**

**while (isdigit(c = fgetc(source\_code)) && i < 99) {**

**token.lexeme[i++] = c;**

**}**

**token.lexeme[i] = '\0';**

**ungetc(c, source\_code);**

**token.type = NUMBER;**

**return token;**

**}**

**token.lexeme[0] = c;**

**token.lexeme[1] = '\0';**

**switch (c) {**

**case ':':**

**if ((c = fgetc(source\_code)) == '=') {**

**strcpy(token.lexeme, ":=");**

**token.type = ASSIGN;**

**} else {**

**ungetc(c, source\_code);**

**token.type = COLON;**

**}**

**break;**

**case '+': token.type = PLUS; break;**

**case '-': token.type = MINUS; break;**

**case '\*': token.type = MULT; break;**

**case '/': token.type = DIV; break;**

**case '=': token.type = EQUAL; break;**

**case '<':**

**if ((c = fgetc(source\_code)) == '>') {**

**strcpy(token.lexeme, "<>");**

**token.type = NOTEQUAL;**

**} else if (c == '=') {**

**strcpy(token.lexeme, "<=");**

**token.type = LESSEQUAL;**

**} else {**

**ungetc(c, source\_code);**

**token.type = LESS;**

**}**

**break;**

**case '>':**

**if ((c = fgetc(source\_code)) == '=') {**

**strcpy(token.lexeme, ">=");**

**token.type = GREATEREQUAL;**

**} else {**

**ungetc(c, source\_code);**

**token.type = GREATER;**

**}**

**break;**

**case ';': token.type = SEMICOLON; break;**

**case ',': token.type = COMMA; break;**

**case '.': token.type = DOT; break;**

**case '(': token.type = OPENPAR; break;**

**case ')': token.type = CLOSEPAR; break;**

**default: token.type = INVALID; break;**

**}**

**return token;**

**}**

**Comportamento**

A função examina o código e desconsidera espaços em branco e observações. (que começa com {) , Reconhece operadores e delimitadores e também Retorna o token identificado ou um token de erro ( INVALID) se não for reconhecido.

A função CasaToken(TokenType expected) analisa se o próximo token está de acordo com o tipo esperado. Se não, ela provoca um erro.

**void CasaToken(TokenType expected) {**

**Token token = getNextToken();**

**if (token.type != expected) {**

**erro("token nao esperado", token);**

**}**

**}**

Chamada getNextToken()para obter o próximo token, Se o tipo de token não for o esperado, chama a função erro().

A função erro()exibe uma mensagem de erro e encerra a execução do programa.

erro(const char \*mensagem, Token token).

**void erro(const char \*mensagem, Token token) {**

**fprintf(stderr, "%d:%s '%s'.\n", token.line, mensagem, token.lexeme);**

**exit(EXIT\_FAILURE);**

**}**

Exibe a mensagem de erro, também com a linha e o lexema do token e encerra o programa.

A principal função do programa controla a leitura do código-fonte e aciona a função Programa() para dar início ao procedimento de análise.

**int main(int argc, char \*argv[]) {**

**if (argc != 2) {**

**fprintf(stderr, "Uso: %s <arquivo MicroPascal>\n", argv[0]);**

**return 1;**

**}**

**source\_code = fopen(argv[1], "r");**

**if (!source\_code) {**

**perror("Erro ao abrir o arquivo");**

**return 1;**

**}**

**Programa();**

**fclose(source\_code);**

**printf("Análise concluída com sucesso.\n");**

**return 0;**

**}**

Abre o arquivo de origem fornecido como parâmetro.

Inicia a análise ao chamar a função Programa().

Data do arquivo ao final da análise.

Testes feito certos

Programa

**program exemplo;**

**var**

**a: integer;**

**begin**

**a := 10;**

**end.**

Resultado :O prog a é declarado

**program exemplo;**

**var**

**a: integer;**

**begin**

**if a = 10 then**

**a := 20;**

**end.**

Resultado:O código if é proc

**program exemplo;**

**var**

**i: integer;**

**begin**

**i := 0;**

**while i < 10 do**

**i := i + 1;**

**end.**

Resultado **:** A es while é reconhecido

Testes feito erro

**program exemplo;**

**begin**

**a := 10;**

**end.**

Erro: Veriavel a não declarada

**program exemplo;**

**var**

**a: integer;**

**begin**

**a == 10;**

**end.**

Erro: U==e:=para atribuir

**program exemplo;**

**var**

**a: integer;**

**a := 10;**

end.

Erro : Atribuição ocorre forabegin ... end, o que é obrigatório.

Conclusão

O código analisa um documento de código fonte, categoriza-o em tokens (como palavras-chave, identificadores, números e operadores) e verifica se esses tokens estão em conformidade com a estrutura específica da linguagem. O analisador léxico identifica tokens válidos, enquanto o analisador sintático (ainda em construção) examina a estrutura do programa. Começa com o nome do programa, seguido por declarações de variáveis e instruções. Se um erro for detectado, o programa emite um alerta e interrompe a operação.