Program Prog;

var a,b:integer;

Procedure proc1;

var v1, v1: integer;

Procedure proc2;

Var x,y:boolean;

begin

...

end;

begin

...

end;

begin

...

end.

begin

a:=5;

b:=a+2;

write(b);

end.

Analisador Lexico e Sintatico para um Pascal simplificado.

Simplificacoes:

1) Tipos so integer e boolean

2) Loop so while

3) Seletor so if

class AnalisadorLexico

{

public:

typedef enum {

INICIO\_DE\_PROGRAMA,

INICIO\_DE\_DECLARACAO\_DE\_VARIAVEL\_OU\_PARAMETRO\_POR\_REFERENCIA,

INICIO\_DE\_DECLARACAO\_DE\_PROCEDIMENTO,

INICIO\_DE\_DECLARACAO\_DE\_FUNCAO,

INICIO\_DE\_COMANDO\_COMPOSTO,

FIM\_DE\_COMANDO\_COMPOSTO,

SE,

ENTAO,

SENAO,

ENQUANTO,

FACA,

VIRGULA,

PONTO\_E\_VIRGULA,

DOIS\_PONTOS,

PONTO,

ABRE\_PARENTESES,

FECHA\_PARENTESES,

LEIA,

ESCREVA,

...

TRUE,

FALSE,

NUMERO,

IDENTIFICADOR,

NULO,

Pedaco\_DESCONHECIDO }

TipoDePedaco;

AnalisadorLexico (char\* nomeArq);

~AnalisadorLexico (); // fecha arquivo

// 1o grupo de gets

TipoDePedaco getPedacoSemConsumir ();

TipoDePedaco getPedacoConsumindo ();

// 2o grupo de gets

char\* getNome (); // deve ser usado quando o ultimo retorno de um dos gets do 1o grupo tiver sido identificador; dar excecao caso contrario

int getValor (); // deve ser usado quando o ultimo retorno de um dos gets do 1o grupo tiver sido numero; dar excecao caso contrario

private:

char\* conhecido [] = {"program","var","procedure",...

FILE\* arquivo;

TipoDePedaco buffer, ultimoPedacoRetornado;

char nomeDoIdenficador [256];

int valorDoNumero;

};

-----------------------------------------------------

class Simbolo

{

public:

typedef enum {

VARIAVEL

PROCEDIMENTO,

FUNCAO,

PARAMETRO }

TipoDeSimbolo;

typedef enum {

INTEIRO,

BOOLEANO }

TipoDeTipo;

// construtor, getters

private:

TipoDeSimbolo tipo;

char nome [255];

unsigned int escopo;

};

class Variavel: public Simbolo

{

public:

// construtor, getters

private:

TipoDeTipo tipo;

};

class Parametro: public Simbolo

{

public:

typedef enum {

POR\_VALOR,

POR\_REFERENCIA }

TipoDePassagem;

// construtor, getters

private:

TipoDeTipo tipo;

TipoDePassagem passagem;

};

class Procedimento: public Simbolo

{

public:

// construtor, getters (fazer TipoDeTipo getParametro (unsigned int i))

private:

TipoDeTipo\* tipoDoParametro; // vetor de TipoDeTipo

};

class Funcao: public Simbolo

{

public:

// construtor, getters (fazer TipoDeTipo getParametro (unsigned int i))

private:

TipoDeTipo\* tipoDoParametro; // vetor de TipoDeTipo

TipoDeTipo tipoDoRetorno;

};

class TabelaDeSimbolos

{

private:

Simbolo\*\* declaracao; // vetor de Simbolo\*

unsigned int escopoAtual; // comeca com 0, aumenta quando comeco a compilacao de um procedimento ou funcao e diminui quando termino a compilacao de um procedimento ou funcao

public:

void guarde (Simbolo\* s); // se s for procedimento ou funcao, armazena-o e aumenta o escopoAtual

Simbolo\* getSimbolo (char\* nome);

void morteAoEscopoAtual (); // remover tudo que tem escopo igual ao escopo atual e diminui o escopo

};