Universidade Federal de Alagoas - UFAL
Instituto de Computação - IC

Karla Elisabeth Cristo dos Santos
Roland dos Santos Gonçalves Sobrinho

Analisador Léxico

11 de Novembro de 2013

Universidade Federal de Alagoas - UFAL
Instituto de Computação - IC

Karla Elisabeth Cristo dos Santos
Roland dos Santos Gonçalves Sobrinho

Analisador Léxico

Trabalho Acadêmico solicitado pelo Prof. Alcino Dall Igna Júnior, com vistas à obtenção parcial de nota da disciplina Compiladores do Curso de Bacharel em Ciência da Computação — UFAL

11 de Novembro de 2013

Sumário

1 Analisador Léxico

O analisador Léxico foi escrito na linguagem C.

1.1 AnalisadorLexico.h

```
#ifndef ANALISADOR_LEXICO_H
#define ANALISADOR_LEXICO_H


typedef struct token Token;

typedef enum categoria Categoria;

Categoria achar_categoria(char *buffer);

void imprimir_token(Token *token);

char* pegar_valor_token(Token *token);

Token* delimitador();

Token* operador_aritmetico();

Token* operador_relacional();

Token* palavra();

Token* constante_literal();

Token* constante_numerica();

Token* indefinidos();

void iniciar(int argc, char *argv[]);

void encerrar();

Token* proximo_token();

void eliminar_comentario();


#endif
```

1.2 AnalisadorLexico.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include "AnalisadorLexico.h"
#define MAX 50

// Variaveis globais
FILE *arquivo;
size_t numero_bytes=100;
int bytes_lidos=0, linha1=0, coluna1=0, i=0, j=0, primeiro=0, comentario = 0, coluna2 = 0;
char *linha_do_arquivo;

// Enumeracao das Categorias.

enum categoria{

    tk_rel_op = 256,
    tk_id = 257,
    tk_const_int = 258,
    tk_const_float =259,
    tk_const_lit = 260,
    tk_kw_int = 261,
    tk_kw_float = 262,
    tk_kw_char = 263,
    tk_kw_if = 264,
    tk_kw_else = 265,
    tk_kw_for = 266,
    tk_kw_while = 267,
    tk_kw_break = 268,
    tk_kw_not = 269,
    tk_kw_and = 270,
    tk_kw_or = 271,
    tk_kw_main = 272,
    tk_kw_func = 273,
    tk_kw_return = 274,
    tk_kw_strcmp = 275,
    tk_kw_strcop = 276,
    tk_kw_input = 277,
    tk_kw_output = 278,
    tk_op_add = 279,
    tk_op_sub = 280,
    tk_op_mul = 281,
    tk_op_div = 282,
    tk_op_atrib = 283,
    tk_open_par = 284,
    tk_close_par = 285,
    tk_open_bra = 286,
    tk_close_bra = 287,
    tk_close_key = 288,
    tk_open_key = 289,
```

```
    tk_semicolon = 290,
    tk_comma = 291,
    erro  = 292
};


struct token{
    char *valor;
    Categoria categoria;
    int linha;
    int coluna;
};


// Funcao utilizada para imprimir token

void imprimir_token(Token *token){

        if(token != NULL)
    printf("[Categoria: %d Valor: %s Linha: %d Coluna: %d]\n",
(int)token->categoria, token->valor, token->linha, token->coluna);
}

//Vetor de ponteiro para funcoes

Token* (*func[128])() = {
    &proximo_token, &proximo_token, &proximo_token,
    &proximo_token, &proximo_token, &proximo_token,//0 - 5
    &proximo_token, &proximo_token, &proximo_token, &proximo_token, &proximo_token, //6-10
    &proximo_token, &proximo_token, &proximo_token, &proximo_token, &proximo_token, //11 - 15
    &proximo_token, &proximo_token, &proximo_token, &proximo_token, &proximo_token, // 16 - 20
    &proximo_token, &proximo_token, &proximo_token, &proximo_token, &proximo_token, //21 - 25
    &proximo_token, &proximo_token, &proximo_token, &proximo_token, &proximo_token, //26- 30
    &proximo_token, &proximo_token, &operador_relacional,
    &constante_literal, &indefinidos,//31 - 35
    &indefinidos, &indefinidos, &indefinidos, &indefinidos, &delimitador, // 36 - 40
    &delimitador, &operador_aritmetico, &operador_aritmetico,
    &delimitador, &operador_aritmetico, // 41 - 45
    &indefinidos, &operador_aritmetico, &constante_numerica,
    &constante_numerica, &constante_numerica, // 46 - 50
    &constante_numerica, &constante_numerica, &constante_numerica,
    &constante_numerica, &constante_numerica, // 51 - 55
    &constante_numerica, &constante_numerica,  &indefinidos,
    &delimitador, &operador_relacional, // 56 - 60
    &operador_relacional, &operador_relacional,
    &delimitador, &indefinidos, &palavra, // 61 - 65
    &palavra, &palavra, &palavra, &palavra, &palavra,
    &palavra, &palavra, &palavra, &palavra, &palavra, // 66 - 75
    &palavra, &palavra, &palavra, &palavra, &palavra,&palavra, &palavra,
    &palavra, &palavra, &palavra, // 76 - 85
    &palavra, &palavra, &palavra, &palavra, &palavra, // 86 - 90
    &delimitador, &indefinidos, &delimitador, &indefinidos, &indefinidos, // 91 - 95
    &indefinidos, &palavra, &palavra, &palavra, &palavra,
```

```
    &palavra, &palavra, &palavra, &palavra, &palavra, // 96 - 105
    &palavra, &palavra, &palavra, &palavra, &palavra, &palavra,
    &palavra, &palavra, &palavra, &palavra, // 106 - 115
    &palavra, &palavra, &palavra, &palavra, &palavra, &palavra,
    &palavra, &delimitador, &indefinidos, &delimitador, // 116 - 125
    &indefinidos, &indefinidos // 126 e 127
};

// Funcoes que analisam os caracteres passados de acordo com sua posicao na tabela ASCII

Token* indefinidos(){
    printf("Erro Linha:%d Coluna:%d \nCaractere %c n o permitido na linguagem\n",
 linha1, coluna1, linha_do_arquivo[i]);
    return proximo_token();
}

Token* delimitador(){
    Token *token = (Token*) malloc(sizeof(Token));
    token->valor=(char*)malloc(3*sizeof(char));
    token->linha = linha1;
    token->coluna = coluna1;
    switch(linha_do_arquivo[i]){
        case '(':
            token->categoria = tk_open_par;
            strcpy(token->valor,"(");
            break;

        case ')':
            token->categoria = tk_close_par;
            strcpy(token->valor,")");
            break;

        case '{':
            token->categoria = tk_open_key;
            strcpy(token->valor,"{");
            break;

        case '}':
            token->categoria = tk_close_key;
            strcpy(token->valor,"}");
            break;

          case '[':
            token->categoria = tk_open_bra;
            strcpy(token->valor,"[");
            break;

        case ']':
            token->categoria = tk_close_bra;
            strcpy(token->valor,"]");
            break;
        case ';':
            token->categoria = tk_semicolon;
```

```
            strcpy(token->valor,";");
            break;
        case ',':
            token->categoria = tk_comma;
            strcpy(token->valor,",");
            break;
    }
    return token;
}

Token* operador_aritmetico(){
    Token *token = (Token*) malloc(sizeof(Token));
    token->valor=(char*)malloc(3*sizeof(char));
    token->linha = linha1;
    token->coluna = coluna1;
    switch(linha_do_arquivo[i]){
        case '+':
            token->categoria = tk_op_add;
            strcpy(token->valor,"+");
            break;

        case '-':
            token->categoria = tk_op_sub;
            strcpy(token->valor,"-");
            break;

        case '*':
            i++;
            if(linha_do_arquivo[i] == '/'){
                printf("Erro na Linha: %d e Colunha: %d\n", linha1, coluna1);
                puts("/* esperado");
                return proximo_token();
            }
            else{
                i--;
                token->categoria = tk_op_mul;
                strcpy(token->valor,"*");
            }
            break;

        case '/':
            i++;
            if(linha_do_arquivo[i] == '*'){
                eliminar_comentario();
                return proximo_token();
            }else{
                i--;
                token->categoria = tk_op_div;
                strcpy(token->valor,"/");
            }
            break;
    }
```

```
    return token;
}

Token* operador_relacional(){
    Token *token = (Token*) malloc(sizeof(Token));
    token->valor=(char*)malloc(3*sizeof(char));
    token->categoria = tk_rel_op;
    token->linha = linha1;
    token->coluna = coluna1;

    switch(linha_do_arquivo[i]){
        case '<':
            i++;
            if(linha_do_arquivo[i] == '='){
                strcpy(token->valor,"<=");
            }else{
              i--;
              strcpy(token->valor,"<");
            }
            break;

         case '>':
            i++;
            if(linha_do_arquivo[i] == '='){
                strcpy(token->valor,">=");
            }else{
              i--;
              strcpy(token->valor,">");
            }
            break;

        case '!':
            i++;
            if(linha_do_arquivo[i] == '='){
                strcpy(token->valor,"!=");
            }else{
              i--;
              puts("Palavra n o permitida pela linguagem");
            }
            break;

        case '=':
            i++;
            if(linha_do_arquivo[i] == '='){
                strcpy(token->valor,"==");
            }else{
              i--;
              token->categoria = tk_op_atrib;
              strcpy(token->valor,"=");
            }
            break;
    }
```

```
    return token;
}

Categoria achar_categoria(char *buffer){
    if(!strcmp(buffer, "int"))
        return tk_kw_int;
    if((!strcmp(buffer, "float")))
        return tk_kw_float;
    if((!strcmp(buffer, "char")))
        return tk_kw_char;
    if((!strcmp(buffer, "if")))
        return tk_kw_if;
    if((!strcmp(buffer, "else")))
        return tk_kw_else;
    if((!strcmp(buffer, "for")))
        return tk_kw_for;
    if((!strcmp(buffer, "while")))
        return tk_kw_while;
    if((!strcmp(buffer, "break")))
        return tk_kw_break;
    if((!strcmp(buffer, "not")))
        return tk_kw_not;
    if((!strcmp(buffer, "and")))
        return tk_kw_and;
    if((!strcmp(buffer, "or")))
        return tk_kw_or;
    if((!strcmp(buffer, "main")))
        return tk_kw_main;
    if((!strcmp(buffer, "function")))
        return tk_kw_func;
    if((!strcmp(buffer, "return")))
        return tk_kw_return;
    if((!strcmp(buffer, "strcmp")))
        return tk_kw_strcmp;
    if((!strcmp(buffer, "strcop")))
        return tk_kw_strcop;
    if((!strcmp(buffer, "input")))
        return tk_kw_input;
    if((!strcmp(buffer, "output")))
        return tk_kw_output;
    return erro;
}

Token* palavra(){
    Token *token = (Token*) malloc(sizeof(Token));
    token->valor=(char*)malloc(3*sizeof(char));

    char *palavra = (char *)malloc(MAX*sizeof(char));
    token->linha = linha1;
    token->coluna = coluna1;

    while(isalpha(linha_do_arquivo[i]) || isdigit(linha_do_arquivo[i]) ||
            linha_do_arquivo[i]== '_'){
```

```
        if(j > MAX-10){
            palavra = (char*)realloc(palavra, (MAX+50)*sizeof(char));
        }
        palavra[j] = tolower(linha_do_arquivo[i]);
        j++, i++, coluna2++;
    }
    palavra[j] = '\0';
    i--;
    coluna2--;


    if(achar_categoria(palavra) != erro)
    {
        strcpy(token->valor, palavra);
        token->categoria = achar_categoria(palavra);
    }
    else{
        token->categoria = tk_id;
        strcpy(token->valor, palavra);
    }
    coluna1 += coluna2;
    coluna2 = 0, j = 0;
        return token;
}

Token* constante_numerica(){
    int ponto_flutuante = 0, k = 0, erro = 0;
    char *constante = (char *)malloc(MAX*sizeof(char));
    Token *token = (Token*) malloc(sizeof(Token));
    token->valor=(char*)malloc(3*sizeof(char));
    token->linha = linha1;
    token->coluna = coluna1;

    while(isdigit(linha_do_arquivo[i]) || linha_do_arquivo[i] == '.'){
        if(isdigit(linha_do_arquivo[i])){
            constante[k] = linha_do_arquivo[i];
            i++, k++, erro = 0;
        }
        else if(linha_do_arquivo[i] == '.' && ponto_flutuante == 0){
            constante[k] = linha_do_arquivo[i];
            ponto_flutuante = 1, i++, k++, erro = 0;
        }else{
            printf("Erro na Linha: %d e Coluna: %d\n", linha1, coluna1);
            puts("Palavra n o permitida pela linguagem");
            erro = 1;
            return proximo_token();
        }
    }
    if(erro == 0)
    {
        constante[k] = '\0';
        i--, erro = 1;
        if(ponto_flutuante){
```

```
            token->categoria = tk_const_float;
            strcpy(token->valor, constante);
        }else{
            token->categoria = tk_const_int;
            strcpy(token->valor, constante);
        }
    }
    return token;
}

Token* constante_literal(){

    char *buffer = (char *)malloc(MAX*sizeof(char));
    Token *token = (Token *)malloc(sizeof(Token));
    token->linha = linha1;
    token->coluna = ++coluna1;
    token->categoria = tk_const_lit;
    i++;

    while(1){
        buffer[j] = linha_do_arquivo[i];
        if(j > MAX-10){
            buffer= (char*)realloc(buffer, (MAX+50)*sizeof(char));
        }
        if(i == bytes_lidos-1){
            printf("Faltando caracter terminador %c\n",34);
            return proximo_token();
        }
        i++, j++, coluna2++;
        if(linha_do_arquivo[i] == '"'){
            break;
        }
   }
    buffer[j] = '\0';
    token->valor=(char*)malloc((strlen(buffer)+1)*sizeof(char));
    strcpy(token->valor, buffer);
    coluna1 += coluna2, j = 0;
    free(buffer);
    return token;
}

//Funcao para elimnar comentarios,
//tudo que esta entre os delimitadores de comentarios sera desconsiderado

void eliminar_comentario(){
    comentario = 1;
        while( comentario == 1 && bytes_lidos != -1)
        {
        if(i == (bytes_lidos-1) ){
            i=0, coluna1 = 0;
            bytes_lidos = getline(&linha_do_arquivo, &numero_bytes, arquivo);
            linha1++, coluna1++;
        }else{
```

```
            i++, coluna1++;
        }
        if(linha_do_arquivo[i] == '*' && (i != bytes_lidos-1))  {
            if(linha_do_arquivo[i + 1] == '/'){
                i++, comentario = 0;
            }
        }
    }
    if(comentario == 1){
         printf("Erro na Linha: %d e Colunha: %d\n", linha1, coluna1);
        puts(" */ esperado");
    }
}


/*Funcao principal a partir dela que sao chamadas outras funcoes para ser feita a analise.
* A analise e' feita no modo on the fly
* Onde cada lexema do codigo e' lido, analisado e retornado.
*
*/
Token* proximo_token(){
    if(i == (bytes_lidos-1) || primeiro == 0 ){
        i=0, coluna1 = 0;
        bytes_lidos = getline(&linha_do_arquivo, &numero_bytes, arquivo);
        if(bytes_lidos == -1){
            return NULL;
        }
        linha1++, coluna1++, primeiro = 1;

    }else if(feof(arquivo)){
        puts("Fim do arquivo");
        return NULL;

    }else{
        i++;
        coluna1++;
    }
    return func[(int)linha_do_arquivo[i]]();
}

//Funcoes de inicializacao e de encerramento

void iniciar(int argc, char *argv[]){
    linha_do_arquivo = (char*) malloc (numero_bytes);
    arquivo = fopen(argv[1], "r");
    if(arquivo == NULL){
        puts("Erro ao abrir o arquivo");
    }
}

void encerrar(){
    fclose(arquivo);
}
```

1.3 main.c

```
#include "AnalisadorLexico.h"
#include<stdlib.h>


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

    iniciar(argc, argv);
    Token *token = proximo_token();
    while(token != NULL){
        imprimir_token(token);
        token = proximo_token();
    }
    encerrar();
    return 0;
}
```

2 Algoritmos

Segue alguns exemplos de algoritmos na linguagem após serem feitas as análises.

Alo Mundo!

```
main()
{
output("Hello World");
}
```

```
[Categoria: 272 Valor: main Linha: 1 Coluna: 1]
[Categoria: 284 Valor: ( Linha: 1 Coluna: 5]
[Categoria: 285 Valor: ) Linha: 1 Coluna: 6]
[Categoria: 289 Valor: { Linha: 2 Coluna: 1]
[Categoria: 278 Valor: output Linha: 5 Coluna: 1]
[Categoria: 284 Valor: ( Linha: 5 Coluna: 7]
[Categoria: 260 Valor: Hello World Linha: 5 Coluna: 9]
[Categoria: 285 Valor: ) Linha: 5 Coluna: 21]
[Categoria: 290 Valor: ; Linha: 5 Coluna: 22]
[Categoria: 288 Valor: } Linha: 7 Coluna: 1]
```

Serie de Fibonacci iterativa

```
function fibonacci(int n)
{
     int i = 0;
     int j = 1;
     int k;
     for(k = 1; k < n; k = k + 1){
             output(i);
             int t = i + j;
             i = j;
             j = t;
     }
     return j;
}
```

```
[Categoria: 273 Valor: function Linha: 1 Coluna: 1]
[Categoria: 257 Valor: fibonacci Linha: 1 Coluna: 10]
[Categoria: 284 Valor: ( Linha: 1 Coluna: 19]
[Categoria: 261 Valor: int Linha: 1 Coluna: 20]
[Categoria: 257 Valor: n Linha: 1 Coluna: 24]
[Categoria: 285 Valor: ) Linha: 1 Coluna: 25]
[Categoria: 289 Valor: { Linha: 2 Coluna: 1]
[Categoria: 261 Valor: int Linha: 3 Coluna: 6]
[Categoria: 257 Valor: i Linha: 3 Coluna: 10]
[Categoria: 283 Valor: = Linha: 3 Coluna: 12]
[Categoria: 258 Valor: 0 Linha: 3 Coluna: 14]
[Categoria: 290 Valor: ; Linha: 3 Coluna: 15]
[Categoria: 261 Valor: int Linha: 4 Coluna: 6]
[Categoria: 257 Valor: j Linha: 4 Coluna: 10]
[Categoria: 283 Valor: = Linha: 4 Coluna: 12]
[Categoria: 258 Valor: 1 Linha: 4 Coluna: 14]
[Categoria: 290 Valor: ; Linha: 4 Coluna: 15]
[Categoria: 261 Valor: int Linha: 5 Coluna: 6]
[Categoria: 257 Valor: k Linha: 5 Coluna: 10]
[Categoria: 290 Valor: ; Linha: 5 Coluna: 11]
[Categoria: 266 Valor: for Linha: 6 Coluna: 6]
[Categoria: 284 Valor: ( Linha: 6 Coluna: 9]
[Categoria: 257 Valor: k Linha: 6 Coluna: 10]
[Categoria: 283 Valor: = Linha: 6 Coluna: 12]
[Categoria: 258 Valor: 1 Linha: 6 Coluna: 14]
[Categoria: 290 Valor: ; Linha: 6 Coluna: 15]
[Categoria: 257 Valor: k Linha: 6 Coluna: 17]
[Categoria: 256 Valor: < Linha: 6 Coluna: 19]
[Categoria: 257 Valor: n Linha: 6 Coluna: 21]
[Categoria: 290 Valor: ; Linha: 6 Coluna: 22]
[Categoria: 257 Valor: k Linha: 6 Coluna: 24]
[Categoria: 283 Valor: = Linha: 6 Coluna: 26]
[Categoria: 257 Valor: k Linha: 6 Coluna: 28]
[Categoria: 279 Valor: + Linha: 6 Coluna: 30]
```

```
[Categoria: 258 Valor: 1 Linha: 6 Coluna: 32]
[Categoria: 285 Valor: ) Linha: 6 Coluna: 33]
[Categoria: 289 Valor: { Linha: 6 Coluna: 34]
[Categoria: 278 Valor: output Linha: 7 Coluna: 14]
[Categoria: 284 Valor: ( Linha: 7 Coluna: 20]
[Categoria: 257 Valor: i Linha: 7 Coluna: 21]
[Categoria: 285 Valor: ) Linha: 7 Coluna: 22]
[Categoria: 290 Valor: ; Linha: 7 Coluna: 23]
[Categoria: 261 Valor: int Linha: 8 Coluna: 14]
[Categoria: 257 Valor: t Linha: 8 Coluna: 18]
[Categoria: 283 Valor: = Linha: 8 Coluna: 20]
[Categoria: 257 Valor: i Linha: 8 Coluna: 22]
[Categoria: 279 Valor: + Linha: 8 Coluna: 24]
[Categoria: 257 Valor: j Linha: 8 Coluna: 26]
[Categoria: 290 Valor: ; Linha: 8 Coluna: 27]
[Categoria: 257 Valor: i Linha: 9 Coluna: 14]
[Categoria: 283 Valor: = Linha: 9 Coluna: 16]
[Categoria: 257 Valor: j Linha: 9 Coluna: 18]
[Categoria: 290 Valor: ; Linha: 9 Coluna: 19]
[Categoria: 257 Valor: j Linha: 10 Coluna: 14]
[Categoria: 283 Valor: = Linha: 10 Coluna: 16]
[Categoria: 257 Valor: t Linha: 10 Coluna: 18]
[Categoria: 290 Valor: ; Linha: 10 Coluna: 19]
[Categoria: 288 Valor: } Linha: 11 Coluna: 6]
[Categoria: 274 Valor: return Linha: 12 Coluna: 6]
[Categoria: 257 Valor: j Linha: 12 Coluna: 13]
[Categoria: 290 Valor: ; Linha: 12 Coluna: 14]
[Categoria: 288 Valor: } Linha: 13 Coluna: 1]
```

ShellSort

```
function shellsort(int vet[], int size) {
    int i , j , value;
    int gap = 1;
    while(gap < size) {
        gap = 3*gap+1;
    }
    while ( gap > 1) {
        gap = gap/3;
        for(i = gap; i < size; i = i + 1) {
            value = vet[i];
            j = i - gap;
            while (j >= 0 and value < vet[j]) {
                vet [j + gap] = vet[j];
                j = j - gap;
            }
            vet [j + gap] = value;
        }
    }
}
```

```
[Categoria: 273 Valor: function Linha: 1 Coluna: 1]
[Categoria: 257 Valor: shellsort Linha: 1 Coluna: 10]
[Categoria: 284 Valor: ( Linha: 1 Coluna: 19]
[Categoria: 261 Valor: int Linha: 1 Coluna: 20]
[Categoria: 257 Valor: vet Linha: 1 Coluna: 24]
[Categoria: 286 Valor: [ Linha: 1 Coluna: 27]
[Categoria: 287 Valor: ] Linha: 1 Coluna: 28]
[Categoria: 291 Valor: , Linha: 1 Coluna: 29]
[Categoria: 261 Valor: int Linha: 1 Coluna: 31]
[Categoria: 257 Valor: size Linha: 1 Coluna: 35]
[Categoria: 285 Valor: ) Linha: 1 Coluna: 39]
[Categoria: 289 Valor: { Linha: 1 Coluna: 41]
[Categoria: 261 Valor: int Linha: 2 Coluna: 5]
[Categoria: 257 Valor: i Linha: 2 Coluna: 9]
[Categoria: 291 Valor: , Linha: 2 Coluna: 11]
[Categoria: 257 Valor: j Linha: 2 Coluna: 13]
[Categoria: 291 Valor: , Linha: 2 Coluna: 15]
[Categoria: 257 Valor: value Linha: 2 Coluna: 17]
[Categoria: 290 Valor: ; Linha: 2 Coluna: 22]
[Categoria: 261 Valor: int Linha: 3 Coluna: 5]
[Categoria: 257 Valor: gap Linha: 3 Coluna: 9]
[Categoria: 283 Valor: = Linha: 3 Coluna: 13]
[Categoria: 258 Valor: 1 Linha: 3 Coluna: 15]
[Categoria: 290 Valor: ; Linha: 3 Coluna: 16]
[Categoria: 267 Valor: while Linha: 4 Coluna: 5]
[Categoria: 284 Valor: ( Linha: 4 Coluna: 10]
[Categoria: 257 Valor: gap Linha: 4 Coluna: 11]
[Categoria: 256 Valor: < Linha: 4 Coluna: 15]
[Categoria: 257 Valor: size Linha: 4 Coluna: 17]
```

```
[Categoria: 285 Valor: ) Linha: 4 Coluna: 21]
[Categoria: 289 Valor: { Linha: 4 Coluna: 23]
[Categoria: 257 Valor: gap Linha: 5 Coluna: 9]
[Categoria: 283 Valor: = Linha: 5 Coluna: 13]
[Categoria: 258 Valor: 3 Linha: 5 Coluna: 15]
[Categoria: 281 Valor: * Linha: 5 Coluna: 16]
[Categoria: 257 Valor: gap Linha: 5 Coluna: 17]
[Categoria: 279 Valor: + Linha: 5 Coluna: 20]
[Categoria: 258 Valor: 1 Linha: 5 Coluna: 21]
[Categoria: 290 Valor: ; Linha: 5 Coluna: 22]
[Categoria: 288 Valor: } Linha: 6 Coluna: 5]
[Categoria: 267 Valor: while Linha: 7 Coluna: 5]
[Categoria: 284 Valor: ( Linha: 7 Coluna: 11]
[Categoria: 257 Valor: gap Linha: 7 Coluna: 13]
[Categoria: 256 Valor: > Linha: 7 Coluna: 17]
[Categoria: 258 Valor: 1 Linha: 7 Coluna: 19]
[Categoria: 285 Valor: ) Linha: 7 Coluna: 20]
[Categoria: 289 Valor: { Linha: 7 Coluna: 22]
[Categoria: 257 Valor: gap Linha: 8 Coluna: 9]
[Categoria: 283 Valor: = Linha: 8 Coluna: 13]
[Categoria: 257 Valor: gap Linha: 8 Coluna: 15]
[Categoria: 282 Valor: / Linha: 8 Coluna: 18]
[Categoria: 258 Valor: 3 Linha: 8 Coluna: 19]
[Categoria: 290 Valor: ; Linha: 8 Coluna: 20]
[Categoria: 266 Valor: for Linha: 9 Coluna: 9]
[Categoria: 284 Valor: ( Linha: 9 Coluna: 12]
[Categoria: 257 Valor: i Linha: 9 Coluna: 13]
[Categoria: 283 Valor: = Linha: 9 Coluna: 15]
[Categoria: 257 Valor: gap Linha: 9 Coluna: 17]
[Categoria: 290 Valor: ; Linha: 9 Coluna: 20]
[Categoria: 257 Valor: i Linha: 9 Coluna: 22]
[Categoria: 256 Valor: < Linha: 9 Coluna: 24]
[Categoria: 257 Valor: size Linha: 9 Coluna: 26]
[Categoria: 290 Valor: ; Linha: 9 Coluna: 30]
[Categoria: 257 Valor: i Linha: 9 Coluna: 32]
[Categoria: 283 Valor: = Linha: 9 Coluna: 34]
[Categoria: 257 Valor: i Linha: 9 Coluna: 36]
[Categoria: 279 Valor: + Linha: 9 Coluna: 38]
[Categoria: 258 Valor: 1 Linha: 9 Coluna: 40]
[Categoria: 285 Valor: ) Linha: 9 Coluna: 41]
[Categoria: 289 Valor: { Linha: 9 Coluna: 43]
[Categoria: 257 Valor: value Linha: 10 Coluna: 13]
[Categoria: 283 Valor: = Linha: 10 Coluna: 19]
[Categoria: 257 Valor: vet Linha: 10 Coluna: 21]
[Categoria: 286 Valor: [ Linha: 10 Coluna: 24]
[Categoria: 257 Valor: i Linha: 10 Coluna: 25]
[Categoria: 287 Valor: ] Linha: 10 Coluna: 26]
[Categoria: 290 Valor: ; Linha: 10 Coluna: 27]
[Categoria: 257 Valor: j Linha: 11 Coluna: 13]
[Categoria: 283 Valor: = Linha: 11 Coluna: 15]
[Categoria: 257 Valor: i Linha: 11 Coluna: 17]
[Categoria: 280 Valor: - Linha: 11 Coluna: 19]
[Categoria: 257 Valor: gap Linha: 11 Coluna: 21]
```

```
[Categoria: 290 Valor: ; Linha: 11 Coluna: 24]
[Categoria: 267 Valor: while Linha: 12 Coluna: 13]
[Categoria: 284 Valor: ( Linha: 12 Coluna: 19]
[Categoria: 257 Valor: j Linha: 12 Coluna: 20]
[Categoria: 256 Valor: >= Linha: 12 Coluna: 22]
[Categoria: 258 Valor: 0 Linha: 12 Coluna: 24]
[Categoria: 270 Valor: and Linha: 12 Coluna: 26]
[Categoria: 257 Valor: value Linha: 12 Coluna: 30]
[Categoria: 256 Valor: < Linha: 12 Coluna: 36]
[Categoria: 257 Valor: vet Linha: 12 Coluna: 38]
[Categoria: 286 Valor: [ Linha: 12 Coluna: 41]
[Categoria: 257 Valor: j Linha: 12 Coluna: 42]
[Categoria: 287 Valor: ] Linha: 12 Coluna: 43]
[Categoria: 285 Valor: ) Linha: 12 Coluna: 44]
[Categoria: 289 Valor: { Linha: 12 Coluna: 46]
[Categoria: 257 Valor: vet Linha: 13 Coluna: 17]
[Categoria: 286 Valor: [ Linha: 13 Coluna: 21]
[Categoria: 257 Valor: j Linha: 13 Coluna: 22]
[Categoria: 279 Valor: + Linha: 13 Coluna: 24]
[Categoria: 257 Valor: gap Linha: 13 Coluna: 26]
[Categoria: 287 Valor: ] Linha: 13 Coluna: 29]
[Categoria: 283 Valor: = Linha: 13 Coluna: 31]
[Categoria: 257 Valor: vet Linha: 13 Coluna: 33]
[Categoria: 286 Valor: [ Linha: 13 Coluna: 36]
[Categoria: 257 Valor: j Linha: 13 Coluna: 37]
[Categoria: 287 Valor: ] Linha: 13 Coluna: 38]
[Categoria: 290 Valor: ; Linha: 13 Coluna: 39]
[Categoria: 257 Valor: j Linha: 14 Coluna: 17]
[Categoria: 283 Valor: = Linha: 14 Coluna: 19]
[Categoria: 257 Valor: j Linha: 14 Coluna: 21]
[Categoria: 280 Valor: - Linha: 14 Coluna: 23]
[Categoria: 257 Valor: gap Linha: 14 Coluna: 25]
[Categoria: 290 Valor: ; Linha: 14 Coluna: 28]
[Categoria: 288 Valor: } Linha: 15 Coluna: 13]
[Categoria: 257 Valor: vet Linha: 16 Coluna: 13]
[Categoria: 286 Valor: [ Linha: 16 Coluna: 17]
[Categoria: 257 Valor: j Linha: 16 Coluna: 18]
[Categoria: 279 Valor: + Linha: 16 Coluna: 20]
[Categoria: 257 Valor: gap Linha: 16 Coluna: 22]
[Categoria: 287 Valor: ] Linha: 16 Coluna: 25]
[Categoria: 283 Valor: = Linha: 16 Coluna: 27]
[Categoria: 257 Valor: value Linha: 16 Coluna: 29]
[Categoria: 290 Valor: ; Linha: 16 Coluna: 34]
[Categoria: 288 Valor: } Linha: 17 Coluna: 9]
[Categoria: 288 Valor: } Linha: 18 Coluna: 5]
[Categoria: 288 Valor: } Linha: 19 Coluna: 1]
```