

Universidade Estadual de Campinas  
Faculdade de Engenharia Elétrica e de  
Computação



## **Laboratório de Software Básico (EA872)**

### **Relatório 3 Analisadores Sintáticos**

Aluna: Gabriela de Castro Surita  
*gabsurita@gmail.com*

Professor: Christian Rodolfo Esteve Rothenberg

## Exercícios

### (ex4) (3 pts.) Parser para alertar sobre variáveis não iniciadas em C

(a) Exemplos de entrada e saída:

```
$ ./ex4 < teste0.c
Aviso: k nao inicializada
$ ./ex4 < teste1.c
Aviso: varE nao inicializada
$ ./ex4 < teste2.c
Aviso: ponto nao inicializada
$ ./ex4 < teste3.c
$ ./ex4 < teste4.c
Aviso: ccccc nao inicializada
$ ./ex4 < teste5.c
Aviso: tempo nao inicializada
$ ./ex4 < teste6.c
Aviso: tres nao inicializada
$ ./ex4 < teste7.c
$ ./ex4 < teste8.c
Aviso: a nao inicializada
Aviso: b nao inicializada
Aviso: d nao inicializada
$ ./ex4 < teste9.c
Aviso: tres nao inicializada
Aviso: sete nao inicializada
Aviso: sete nao inicializada
```

### (ex5) (4 pts) Geração de parser para desmembramento de comandos e armazenamento de informações

(a) Exemplo de entrada e saída

```
$ ./ex5 < ex5.in
Comando invalido na linha 2
Comando invalido na linha 2

comando_a : parametro_a1, parametro_a2, opcao_a1, 324234, x3294549,
comando_b : opcao_b1, opcao_b2, parametro_xyz,
comando_c :
comando_d : parametro_d, 22, 1, opcao_inexistente,
```

# Códigos

## (ex4) (3 pts.) Parser para alertar sobre variáveis não iniciadas em C

### (a) Analisador Léxico:

```
%{
#include "ex4.tab.h"
}%

%%

"main()_{" {return INICIO_MAIN;}
"}" {return FIM_MAIN;}
int|char|float|double {return TIPO;}
[a-zA-z][a-zA-Z0-9_]* {strcpy(yylval.text, yytext); return NOMEVAR;}
[0-9]*[.]*[0-9]* {return VALOR;}
[=] {return IGUAL;}
["+ "-" "/" "*""] {return OPERA;}
; {return PONTO_VIRGULA;}
, {return VIRGULA;}
" (" {return PARENTESSES_ESQ;}
")" {return PARENTESSES_DIR;}
. {}

%%
```

### (b) Analisador Sintático:

```
%{
#include <stdio.h>
#include <string.h>

char variaveis[64][64];
int nVar = 0;e
}%

%union {
    char text[64];
}

%token
INICIO_MAIN
FIM_MAIN
PONTO_VIRGULA
TIPO
VIRGULA
```

```

NOMEVAR
IGUAL
VALOR
OPERA
PARENTESES_ESQ
PARENTESES_DIR

%%
prog_fonte:
INICIO_MAIN conteudo_prog FIM_MAIN;

conteudo_prog:
declaracoes expressoes;

declaracoes:
linha_declara declaracoes | linha_declara;

linha_declara:
TIPO variaveis PONTO_VIRGULA;

variaveis:
identificador VIRGULA variaveis | identificador;

identificador:
NOMEVAR | atribuicao IGUAL VALOR ;

expressoes:
linha_executavel expressoes | linha_executavel;

linha_executavel:
atribuicao IGUAL operacoes PONTO_VIRGULA /* idem */;

operacoes:
operacoes OPERA operacoes
| PARENTESES_ESQ operacoes PARENTESES_DIR
| VALOR
| acesso;

atribuicao:
NOMEVAR {
    strcpy(variaveis[nVar], yylval.text);
    nVar++;
};

```

```

acesso:
NOMEVAR {
    int ui = 1;
    for(int i=0; i<nVar; i++){
        if(strcmp(yylval.text, variaveis[i]) == 0){
            ui = 0;
            break;
        }
    }
    if(ui)
        printf("Aviso:_%s_nao_inicializada\n", yylval.text);
};

%%

int main(){
    yyparse();
}

```

**(ex5) (4 pts) Geração de parser para desmembramento de comandos e armazenamento de informações**

(a) Analisador léxico (ex5.l):

```

%{
    #include <ex5.tab.h>
%}

%%
#.*          {}
,            {return VIRGULA;}
[^,:\n\t\r]+: {strcpy(yylval.text, yytext); return COMANDO;}
[^,:\n\t\r]+  {strcpy(yylval.text, yytext); return PARAMETRO;}
[\n]         {return NOVALINHA;}
.            {}
%%

```

(b) Analisador Sintático (ex5.y):

```

%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```

#include "queue.h"

/* Tipo comando */
typedef struct comando{
    char* texto;
    Queue* filaParam;
} comando;

/* Fila de comandos */
Queue* filaComandos;

/* Comando atual para insercao de parametros */
comando* comandoAtual;

/* Contador de Linha */
int nlinhas = 1;

/* Remove ocorrencias de caracter em string */
void clearString(char* input, char rem);

%}

%union {
    char text[64];
}

%token
COMANDO
PARAMETRO
VIRGULA
NOVALINHA

%%
prog:
linhas | linha;

linhas:
linha linhas | linha;

linha:
comando | parametros | fimLinha;

parametros:

```

```

parametro VIRGULA parametros | parametro NOVALINHA;

comando:
COMANDO {
    /* Limpa a string de ":" e " " */
    clearString(yylval.text, '_');
    clearString(yylval.text, ':');

    /* Cria novo comando */
    comandoAtual = (comando*)malloc(sizeof(comando));
    char* texto = (char*)malloc(sizeof(char)*strlen(yylval.text));
    strcpy(texto, yylval.text);
    comandoAtual->texto = texto;
    comandoAtual->filaParam = newQueue();

    /* Insere comando na fila */
    inQueue(&filaComandos, (char*)comandoAtual);
};

parametro:
PARAMETRO {
    /* Verifica se o comando atual e valido */
    if(comandoAtual){
        /* Cria novo parametro */
        char* texto = (char*)malloc(sizeof(char)*strlen(yylval.text));
        strcpy(texto, yylval.text);

        /* Insere parametro na fila */
        Queue* parametros = comandoAtual->filaParam;
        inQueue(&parametros, texto);
    }
    /* Caso: parametro em comando invalido */
    else{
        printf("Comando_invalido_na_linha_%d\n", nlinhas);
    }
};

fimLinha:
NOVALINHA{
    nlinhas++;
    comandoAtual = NULL;
};

%%

```

```

int main(){

    filaComandos = newQueue();
    yyparse();

    /* Imprime e esvazia conteudo da lista */
    while(notNullQueue(&filaComandos)){

        comando* iter = (comando*)outQueue(&filaComandos);

        char* texto = iter->texto;
        printf("%s: ", texto);
        free(texto);

        Queue* params = iter->filaParam;

        while(notNullQueue(&params)){
            char* texto = outQueue(&params);
            printf("%s, ", texto);
            free(texto);
        }
        destroyQueue(&params);
        free(iter);
        printf("\n");
    }
    destroyQueue(&filaComandos);
}

/* Remove ocorrencias de caracter em string
   Adaptado de http://stackoverflow.com/questions/4161822/ */
void clearString(char* input, char rem){

    char *src, *dest;
    src = dest = input;

    while(*src != '\0'){
        if(*src != rem){
            *dest = *src;
            dest++;
        }
        src++;
    }
    *dest = '\0';
}

```



(c) Cabeçalho do tipo fila (queue.h):

```
/* Define um tipo no generico */
typedef struct node {
    char* val;
    struct node* next;
} Node;

/* Define um tipo fila */
typedef struct queue {
    Node* start;
    Node* end;
} Queue;

/* Malloc com verificacao */
void* smalloc(int Size);

/* Cria uma fila */
Queue* newQueue();

/* Insere elemento na entrada da fila */
void inQueue(Queue** Qe, char* val);

/* Retira elemento da saida da fila */
char* outQueue(Queue** Queue);

/* Verifica se a lista possui elementos */
int notNullQueue(Queue** Qe);

/* Libera fila da memoria. */
void destroyQueue(Queue** Qe);

/* Cria um tipo no com valor arbitrario */
Node* newNode(char* val);
```

(d) Implementação do tipo fila (queue.c):

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

/* Malloc com verificacao */
void* smalloc(int size){
    void* mem = malloc(size);
    if(mem == NULL) {
```

```

        printf("Malloc_Error\n");
        exit(1);
    }
    return mem;
}

/* Cria um tipo no com valor arbitrario */
Node* newNode(char* val){

    Node* New = (Node*) malloc(sizeof(Node));
    New->val = val;
    return New;
}

/* Libera (no a no) uma lista da memoria. */
void destroyList(Node** Root){
    Node* Next;
    Node* Iter = *Root;

    while(Iter != NULL){
        Next = Iter->next;
        free(Iter);
        Iter = Next;
    }
}

/* Cria umq fila */
Queue* newQueue(){

    Queue* Qe = malloc(sizeof(Queue));
    Qe->start = NULL;
    Qe->end = NULL;
    return Qe;
}

/* Insere elemento na entrada da fila */
void inQueue(Queue** Qe, char* val){
    Queue* Iter = *Qe;

    Node* New = newNode(val);

    if(Iter->end){
        New->next = Iter->start;
        Iter->end->next = New;
        Iter->end = New;
    }
}

```

```

    else{
        Iter->start = New;
        Iter->end = New;
        Iter->end->next = New;
    }
}

/* Retira elemento da saida da fila */
char* outQueue(Queue** Qe){
    Node* OutNode;
    char* OutValue;
    Queue* Iter = *Qe;

    OutNode = Iter->start;

    if(Iter->end != Iter->start){
        Iter->start = Iter->start->next;
    }
    else{
        Iter->start = NULL;
        Iter->end = NULL;
    }

    OutValue = OutNode->val;
    free(OutNode);
    return OutValue;
}

/* Verifica se a lista possui elementos */
int notNullQueue(Queue** Qe){
    return (*Qe)->end != NULL;
}

/* Libera fila da memoria. */
void destroyQueue(Queue** Qe){
    while(notNullQueue(Qe)){
        outQueue(Qe);
    }
    free(*Qe);
}

```