Universidade Federal de Alfenas - UNIFAL-MG
Bacharelado em Ciência da Computação
Prof. Luiz Eduardo da Silva

Trabalho de Compiladores
Projeto do Compilador SIMPLES

Objetivo

O objetivo desse trabalho é modificar o projeto do compilador (parte 3), para:

1. Alterar a tabela de símbolos para incluir informações que possibilitem a tradução de funções.
2. Incluir regras para compilação de funções com passagem de parâmetro por valor ou referência.
3. Verificar a compatibilidade de tipos nas chamadas das funções para identificar erros no número ou tipo dos parâmetros nas chamadas.

Problema

A rotina (função ou procedimento) é uma entidade dentro da programação estruturada que representa um pedaço de código (subprograma) que pode ser invocado pelo seu nome em qualquer parte do código (inclusive uma chamada recursiva do código sobre si mesmo). Nessa chamada da rotina é possível passar valores para rotina e receber argumentos modificados através do mecanismo de passagem por referência. Para rotina do tipo função ainda é possível receber o valor retornado da função, que na linguagem Simples é definido através da atribuição de um valor para a variável nome-da-função.

Descrição

1. A primeira modificação é na tabela de símbolos. As regras de declaração de **variáveis** e **rotinas** devem ser modificadas para incluir as informações das funções, parâmetros, variáveis locais, além das variáveis globais na tabela de símbolos. Uma estrutura sugerida para a tabela de símbolos é:

```
enum { INT = 1, LOG, VAL = 10, REF };

typedef struct no *lista;
struct no {
  int tipo;
  int mec;
  lista prox;
};

struct elem_tab_simbolos {
  char id[30];     // identificador (nome)
  int desloca;     // deslocamento (endereco)
  int tipo;        // tipo (inteiro, logico)
```

```
  int mec;          // mecanismo
  int rotulo;       // rotulo
  char escopo;      // G = global ou L = local
  char cat;         // P = parametro V = variavel F = funcao
  int npar;         // numero de parametros
  lista listapar; // lista de parametros
} TabSimb[TAM_TAB], elem_tab;
```

#	id	esc	dsl	rot	cat	tip	mec	npar	par

- # - identifica a posição do símbolo na tabela
- **nome(id)** - é o nome do identificador, o nome escolhido pelo programador para a entidade do programa
- **escopo(esc)** - escopo da variável. No caso da linguagem simples os símbolos só podem ter escopo global (G) ou local (L).
- **deslocamento(dsl)** - deslocamento (ou endereço) é o operando que acompanhará as variáveis na instrução CRVG, CRVL. As funções também tem valor para esse campo.
- **rótulo(rot)** - rótulo atribuído pelo compilador para a função ou procedimento. Essa informação é necessária para tradução da instrução de desvio (DSVS) na chamada do procedimento ou função.
- **categoria(cat)** - categoria do símbolo que pode ser: V = variável, F = função, P = parâmetro.
- **tipo(tip)** - tipo do símbolo (INT = inteiro ou LOG = lógico).
- **mecanismo(mec)** - mecanismo de passagem para parâmetros que pode ser REF = referência ou VAL = valor.
- **número de parâmetros (npar)** - para facilitar a verificação de consistência no número de parâmetros e argumentos de chamada da função pode-se incluir esse campo na tabela de símbolos.
- **parâmetros(par)** - lista encadeada dos parâmetros da rotina, com seus Tipos e Mecanismos de passagem de parâmetro. Essa informação é necessária para traduzir de forma correta os parâmetros na chamada da rotina. Observe que no programa principal as variáveis locais e parâmetros das rotinas são excluídos da tabela de símbolos

Exemplos:

- A seguinte declaração:

```
programa testafuncao
  inteiro x y z
  logico a b c
func inteiro fazer (inteiro x ref inteiro y)
  inteiro a b
  logico c d
  inteiro g
```

- Deve preencher as seguintes informações na tabela de símbolos:

Tabela de Simbolos

ID	DSL	TIP	ESC	MEC	ROT	CAT	LPA
x	0	1	G	-1	-1	V	
y	1	1	G	-1	-1	V	
z	2	1	G	-1	-1	V	
a	3	2	G	-1	-1	V	
b	4	2	G	-1	-1	V	
c	5	2	G	-1	-1	V	
fazer	-5	1	G	-1	1	F	2 => [(t=1,m=10) (t=1,m=11)]
x	-4	1	L	10	-1	P	
y	-3	1	L	11	-1	P	
a	0	1	L	-1	-1	V	
b	1	1	L	-1	-1	V	
c	2	2	L	-1	-1	V	
d	3	2	L	-1	-1	V	
g	4	1	L	-1	-1	V	

Sugestão de modificação das regras para atualizar a tabela de símbolos:

```
(...)

int ehVariavel;
int ehReferencia;

%%
programa : cabecalho
  variaveis
  rotinas
  T_INICIO lista_comandos T_FIM
;

cabecalho : T_PROGRAMA T_IDENTIF ;

rotinas : |
  // gerar DSVS L0
  lista_funcoes
  // gerar L0 NADA
;

lista_funcoes : funcao lista_funcoes | funcao ;

funcao : T_FUNC tipo T_IDENTIF
  // inserir nome da funcao na tabela.
  // gerar ENSP
  // mudar o escopo para local
  T_ABRE
  lista_parametros T_FECHA
  // ajustar deslocamentos e incluir lista de parametros
  variaveis
  T_INICIO lista_comandos T_FIMFUNC
  // remover variaveis locais
  // mudar o escopo para global
  // gerar RTSP
;

lista_parametros : | lista_parametros parametro;

parametro : mecanismo tipo T_IDENTIF
```

```
  // incluir parametro na tabela de simbolos
;

mecanismo : { mecanismo = VAL; } | T_REF { mecanismo = REF; };

(...)
```

2. A segunda modificação é nas regras para **comandos** e **expressões**. Modificar essas regras para gerar os códigos que traduzem as chamadas das funções, os parâmetros e as variáveis locais nas expressões. Além de verificar a compatibilidade de número e tipo dos argumentos e parametros na chamada da função.

   Sugestão de modificações nas regras de comandos e expressões:

```
(...)

identificador : T_IDENTIF
   {
     int i = busca_simbolo (atomo);
     if (i == -1)
        msg ("Identificador desconhecido!");
     empilha (i, 'i'); // empilhar a posicao do id na tabsimb
   }
 ;

leitura : T_LEIA identificador
  // gerar ARMI se a leitura eh para um parametro por referencia
  // gerar ARZL para parametro por valor ou variavel local
  // gerar ARZG para variavel global
;

(...)

atribuicao : identificador
  {
    int pos = desempilha();
    empilha (pos, 'i');
    empilha (TabSimb[pos].tipo, 't');
  }
  T_ATRIB expressao
  // gerar ARMI se a atribuicao eh para um parametro por referencia
  // gerar ARZL para parametro por valor ou variavel local ou nome de funcao
  // gerar ARZG para variavel global

expressao : expressao T_VEZES expressao
  | expressao T_DIV expressao
  | expressao T_MAIS expressao
  (...)
  | termo ;

argumentos :
  |
  // contar argumentos
  lista_argumentos
  // comparar numero de argumentos com o numero de parametros
;

lista_argumentos : lista_argumentos argumento | argumento ;

argumento :
```

```
  expressao
  // comparar tipo e categoria do argumento com o parametro
  //   (somente variaveis podem ser passadas por referencia)
;

termo : identificador
  // gerar CRVG - se a variavel eh global
  // gerar CREG - se a variavel eh global e a passagem por referencia
  // gerar CREL - se a variavel eh local e a passagem por referencia
  // gerar CRVL - se a variavel eh local e a passagem por valor ou
  //              se a variavel eh por referencia e a passagem por referencia
  // gerar CRVI - se a variavel eh por referencia e a passagem por valor
  | identificador T_ABRE
  // gerar AMEM 1
    argumentos
    T_FECHA
  // gerar SVCP e DSVS
;

| T_NUMERO
(...)
```

Entrega

1. Incluir um comentário no cabeçalho de cada programa fonte com o seguinte formato:

```
/*+------------------------------------------------------------------
|           UNIFAL - Universidade Federal de Alfenas.
|             BACHARELADO EM CIENCIA DA COMPUTACAO.
| Trabalho..: Compilador Simples - Funcao
| Disciplina: Teoria de Linguagens e Compiladores
| Professor.: Luiz Eduardo da Silva
| Aluno.....: Fulano da Silva
| Data......: 99/99/9999
+------------------------------------------------------------------*/
```

2. A pasta com o projeto deverá incluir o arquivo makefile.

3. O código do compilador deve incluir, **obrigatoriamente**, a seguinte função main:

```
int main (int argc, char *argv[])
{
  char *p, nameIn[100], nameOut[100];
  argc--;
  argv++;

  if (argc < 1) {
    puts("\nCompilador SIMPLES");
    puts("\tUSO: ./simples <nomefonte>[.simples]\n\n");
    exit(10);
  }
  if (p = strstr (argv[0], ".simples"))
    *p = 0;
  strcpy (nameIn, argv[0]);
  strcat (nameIn, ".simples");
  strcpy (nameOut, argv[0]);
  strcat (nameOut, ".mvs");

  yyin = fopen (nameIn, "r");
  if (!yyin) {
    puts ("Programa fonte nao encontrado!");
    exit(20);
  }
  yyout = fopen (nameOut, "w");

  if (!yyparse ())
     printf ("\nPrograma Ok!\n\n");
}
```

4. O compilador deverá ter o nome "simples"(linux) ou "simples.exe"(windows) e ser chamado através da seguinte linha de comando (conforme discutido em sala):

```
./simples nomeprograma  //no linux ou
simples nomeprograma    //no windows
```

5. Enviar num arquivo único (.ZIP), a pasta do projeto com somente os arquivos fontes (lexico.l, sintatico.y, tabela.c, pilha.c, makefile, etc.) necessários para compilação projeto, através do Envio de Arquivo do MOODLE.