**Projeto II de Compiladores**

**uPortugol – Geração de Código de Máquina, v 1.0**

**Universidade Federal do Amazonas – Instituto de Computação**

**Marco Cristo**

**1. Introdução**

A linguagem *uPortugol* é uma versão do Portugol, linguagem normalmente usada em ensino de programação no Brasil. A máquina *microJVM* é uma versão simplificada de uma máquina virtual Java. Neste trabalho, você deve completar a construção de um compilador da linguagem uPortugol para o código de máquina da microJVM.

**2. Objetivo**

Fornecer uma gramática para uPortugol que, dado o fonte escrito em uPortugol, este seja traduzido para o assembler da microJVM. Além da gramática, ainda devem ser fornecidos os códigos *Compile.java*, *Parser.java* e *Scanner.java*.

A microJVM foi definida em sala de aula. No site da disciplina, são fornecidos os códigos fontes da microJVM, bem como sua documentação.

A uPortugol é descrita *informalmente* neste texto. No site da disciplina, são fornecidos códigos de teste em uPortugol (extensão .up) com descrições em código de máquina adicionadas como comentários nos fontes.

**3. Avaliação**

A avaliação será feita por meio de análise da gramática fornecida e dos códigos de máquina gerados para os fontes em uPortugol usados para teste (não necessariamente os mesmos que vocês têm, mais com sintaxe coerente com a dada).

**4. Observações**

a) Trabalho a ser realizado por equipes de, no máximo, duas pessoas.

b) Plágio não será tolerado com anulação do trabalho para as equipes envolvidas.

c) Dúvidas quanto a especificação da linguagem são esperadas e o *como resolvê-las* faz parte das decisões de projeto a serem tomadas pelas equipes.

**Anexo I – Especificação *informal* da linguagem uPortugol**

Nesta seção é apresentada uma descrição informal da linguagem uPortugol.

**I.1. Características Gerais**

Um programa em uPortugol consiste de um único arquivo com variáveis e procedimentos estáticos e um bloco principal. Quando um programa em uPortugol é chamado, este bloco é executado. Em termos gerais, a linguagem possui:

* Constantes do tipo *inteiro* (ex: 3);
* Variáveis estáticas do tipo *inteiro;*
* Tipos referência: vetores de uma dimensão de tipo *inteiro.*
* Suporte à inicialização estática de vetor (ex: var v: inteiro[]; v = novo inteiro {3, 4, 5, 6}).
* Procedimentos estáticos sem retorno ou com retorno *inteiro* (função);
* A função pré-declaradas tamanho(vetor) que fornece o tamanho de um vetor;
* Dois procedimentos de I/O: (1) *leia*(var) que lê um inteiro do teclado para var e (2) *escreva*(), que suporta listas de expressões entre vírgulas e cadeias de caracteres constantes.

A linguagem não suporta acesso a um coletor de lixo, pois irá rodar em uma VM microJava que não tem *garbage collection*.

**I.2. Estrutura léxica**

***Caracteres***

letra = 'a'..'z' | 'A'..'Z' .

digito = '0'..'9'.

semAspas = qualquer caractere exceto aspas.

Devem ser ignorados espaços, '\n', '\r' e '\t' na escrita do código fonte.

***Terminais***

ident = letra inicioletra | digitofim.

numero = ['-'] digito iniciodigitofim.

strConst = '”' iniciosemAspasfim '”'.

**Palavras-chave**

|  |  |  |
| --- | --- | --- |
| **Palavra-chave** | **Descrição** | **Exemplo** |
| *inicio ... fim* | Sequencia de comandos em procedimento ou programa | inicio  c = 0;  b = 1;  fim |
| *variavel* | Declaração de variáveis | variavel a, b: inteiro;  variavel v: inteiro[]; |
| *se ... entao ... fimse* | Salto condicional | se a == 0 entao  c = 0;  fimse; |
| *se ... entao ... senao ... fimse* | Alternativa ao salto feito pelo *se* | se a == 0  entao c = 0;  senao c = 1;  fimse; |
| *caso* | Seleção com múltiplas alternativas. Apenas um *caso* pode ser executado por seleção. Opção *outrocaso* é escolhida quando nenhum *seja* pode ser satisfeito. | caso dia  seja 1 faca  escreva(“Domingo: ”);  escreva (“fim de semana\n”);  seja 2 faca  escreva (“Segunda-feira: ”);  escreva (“dia util\n”);  seja 3 faca  escreva(“Terca-feira: ”);  escreva (“dia util \n”);  seja 4 faca  escreva (“Quarta-feira: ”);  escreva (“dia util \n”);  seja 5 faca  escreva (“Quinteiroa-feira: ”);  escreva (“dia util \n”);  seja 6 faca  escreva (“Sexta-feira: ”);  escreva (“dia util \n”);  seja 7 faca  escreva (“Sabado: ”);  escreva (“fim de semana\n”);  outrocaso:  escreva (“Erro!\n”);  fimcaso |
| *enquanto* | Iteração condicional | enquanto a < TAM faca  a = a + 1;  fimenquanto |
| *para* | Iteração condicional com inicialização e atribuição de variável de incremento | para i = 0 ate TAM - 1 passo 2 faca  a = a + i;  fimpara |
| *repita* | Iteração condicional com teste no fim | repita  a = a + i;  ate a == TAM; |
| *leia* | Leitura de variável inteira do teclado. Se entrada for inválida, *n* recebe valor 0 | leia(n); |
| *escreva* | Exibição de expressões e strings | escreva(“v[”, i, “] = ”, v[i], “\n”); |
| *retorne* | Saída de função com indicação de valor de retorno | retorne a + b; |
| *constante* | Declaração de constante inteira | constante MAX = 10; |
| *novo* | Requisição de alocação de vetor no *heap* | v = novo inteiro[MAX];  v = novo inteiro {1, 2, 3, 4, 5}; |
| *algoritmo* | Título do algoritmo | algoritmo Quicksort |

**Operadores**

*Aritméticos*: + (adição), - (subtração), \* (multiplicação), / (divisão), % (resto)

*Relacionais*: == (igualdade), != (dseerença), > (maior que), >= (maior ou igual a), < (menor que), <= (menor ou igual a)

*Agrupamento de expressões*: ( )

*Identificação de vetor e posição em vetor*: [ ]

*Lista de valores de inicialização de vetor*: {}

*Atribuição*: =

*Fim de instrução*: ; (comandos de fim de instrução fim\* não precisam ser seguidos de ;)

*Separador de itens em listas (expressões, parâmetros e variáveis)*: ,

**Comentários**

Comentários devem ser envolvidos pelas sequências “/\*” e “\*/” e devem ser aninhados.

**I.3. Semântica**

*Tipos de referência*: vetores;

*Tipo de uma constante*: constante inteira é do tipo inteiro;

*Mesmo tipo*: dois tipos são os mesmos se (a) eles são denotados pelo mesmo nome de tipo ou (b) se ambos são vetores e os tipos de seus elementos são os mesmos.

*Tipos compatíveis*: dois tipos são compatíveis se eles são os mesmos.

*Compatibilidade de atribuição*: o tipo *origem* é compatível com o *destino* se (a) *origem* e *destino* são do mesmo tipo ou (b) se o *destino* é um tipo referência e a *origem* é tipo nulo (chamadas à função tamanho()).

Nomes pré-declarados

*inteiro* = tipo dos valores inteiros;

*tamanho*(*vLEN*) = função que retorna o número de elementos de um vetor vLEN;

*Escopo*

Um escopo é uma região de texto de um programa ou procedimento. Ela se estende por todo corpo do arquivo (escopo de programa ou global) ou dentro de um bloco *inicio*...*fim* (escopo local, de *algoritmo* ou *procedimento*). Um escopo exclui outros escopos que se localizam em seu interior. Nós assumimos que há um escopo mais externo que chamamos *universo.* Este escopo possui todos os nomes pré-declarados (ex: função *tamanho*()). A declaração de um nome em um escopo interno esconde o mesmo nome declarado em um escopo mais externo. O bloco principal inicio...fim define o escopo local *algoritmo*.

*Notas*

a) Recursão indireta não é suportada, uma vez que todo o nome deve ser declarado antes de ser usado.

b) Nomes pré-declarados podem ser re-declarados (ex: inteiro), mas isso não é recomendado. *Isto não deve ocorrer em nenhum exemplo dado ou usado para avaliação.*

**I.4. Condições contextuais**

Condições gerais

* Todo nome deve ser declarado antes de usado;
* Um nome não deve ser declarado duas vezes no mesmo escopo;
* Toda a expressão aritmética deve ser do tipo inteiro;
* Na função *tamanho*(a), a é um vetor;

Condições específicas

|  |  |
| --- | --- |
| **Produção** | **Condição** |
| DeclaracaoProcedimento = "procedimento" ident "(" [Parametros] ")" [":" Tipo] "inicio" { Instrucao } "fim". | Procedimento não pode retornar *vetor*;  Se retorna Tipo, então deve possuir instrução *retorne* (isso deve ser checado em tempo de execução); |
| Instrucao = Designator "=" Expr ";". | Designador deve denotar uma variável ou um elemento de um vetor;  O tipo de Expr deve ser assinalável para o tipo do Designator; |
| Instrucao = Designator ParamsDeclarados ";". | Designador deve denotar uma função; |
| Instrucao = "retorne" [Expr] . | O tipo de Expr deve ser compatível com o tipo da função atual;  Se Expr não é usado, a função deve ter tipo *nada*; |
| ParamsPassados = "(" [ Expr {"," Expr}] ")". | O número de parâmetros passados deve ser igual ao declarado;  O tipo de todo parâmetro passado deve ser compatível com o tipo declarado na mesma posição; |
| Condicao = Expr OpRelacional Expr. | Vetores (não os seus elementos!) só podem ser comparados para igualdade e diferença; |
| Expr = … Termo … Fator … | Termo e Fator devem ser do tipo inteiro. |
| Designator = ident "[" Expr "]". | O tipo do ident deve ser vetor; |
| Fator = Designator ParamsPassados. | Designador deve denotar um nome de função |
| Fator = "novo" ident ("[" Expr "]" | Inicializ). | Ident deve denotar um tipo;  Lista de inicialização deve ter ao menos um elemento; |

**I.5. Restrições de implementação**

Dadas as instruções de endereçamento local e global, a VM a ser usada só suporta 127 variáveis locais e 32767 globais.

**I.6. Exemplo de código uPortugol correto**

/\*

comparacao de metodos de ordenacao, Marco, 2014

\*/

algoritmo ComparacaoDeAlgoritmosOrdenacao

constante TAM = 10;

procedimento imprime(v: inteiro[])

inicio

variavel i: inteiro;

para i = 0 ate tamanho(v) - 1 faca

escreva(v[i], " ");

fimpara;

fim

procedimento copia(v1: inteiro[], v2: inteiro[])

inicio

variavel i: inteiro;

para i = 0 ate (tamanho(v1) - 1) faca

v1[i] = v2[i];

fimpara;

fim

procedimento troque(v: inteiro[], i: inteiro, j: inteiro)

inicio

variavel tmp: inteiro;

tmp = v[i];

v[i] = v[j];

v[j] = tmp;

fim

/\*\*\*\*\* bolha \*\*\*\*\*\*\*/

procedimento bolha(v: inteiro[]): inteiro

inicio

variavel custo, i, troca: inteiro;

custo = 0;

troca = 1;

enquanto troca == 1 faca

troca = 0;

para i = 0 ate tamanho(v) - 2 faca

custo = custo + 1;

se v[i] > v[i+1] entao

troque(v, i, i+1);

troca = 1;

fimse;

fimpara;

fimenquanto;

retorne custo;

fim

/\*\*\*\*\* selecao \*\*\*\*\*\*\*/

procedimento minpos(v: inteiro[], i: inteiro, f: inteiro): inteiro

inicio

variavel j, min, minj: inteiro;

min = v[i];

minj = i;

para j = i + 1 ate f - 1 faca

se v[j] < min entao

min = v[j];

minj = j;

fimse;

fimpara;

retorne minj;

fim

procedimento selecao(v: inteiro[]): inteiro

inicio

variavel i, minp, custo: inteiro;

custo = 0;

para i = 0 ate tamanho(v) - 2 faca

custo = custo + tamanho(v) - i - 1;

minp = minpos(v, i + 1, tamanho(v));

se v[i] > v[minp] entao

troque(v, i, minp);

fimse;

fimpara;

retorne custo;

fim

/\*\*\*\*\* insercao \*\*\*\*\*\*\*/

procedimento jMaior0EauxMenorVjmenos1(j: inteiro, aux: inteiro, v: inteiro[]): inteiro

inicio

se j > 0 entao se aux < v[j - 1] entao retorne 1; fimse; fimse;

retorne 0;

fim

procedimento insercao(v: inteiro[]): inteiro

inicio

variavel i, j, custo, aux: inteiro;

custo = 0;

para i = 1 ate tamanho(v) - 1 faca

aux = v[i];

j = i;

enquanto jMaior0EauxMenorVjmenos1(j, aux, v) == 1 faca

custo = custo + 1;

v[j] = v[j - 1];

j = j - 1;

fimenquanto;

v[j] = aux;

fimpara;

retorne custo;

fim

/\*\*\*\* mergesort \*\*\*\*/

procedimento esqMEfimEsqEmeioMEdir(esq: inteiro, fimEsq: inteiro,

meio: inteiro, dir: inteiro) : inteiro

inicio

se esq <= fimEsq entao se meio <= dir entao retorne 1; fimse; fimse;

retorne 0;

fim

procedimento merge(v: inteiro[], tmp: inteiro[], esq: inteiro,

meio: inteiro, dir: inteiro): inteiro

inicio

variavel i, fimEsq, tam, k, custo: inteiro;

custo = 0;

fimEsq = meio - 1;

k = esq;

tam = dir - esq + 1;

enquanto esqMEfimEsqEmeioMEdir(esq, fimEsq, meio, dir) == 1 faca

custo = custo + 1;

se v[esq] <= v[meio] entao

tmp[k] = v[esq];

esq = esq + 1;

senao

tmp[k] = v[meio];

meio = meio + 1;

fimse;

k = k + 1;

fimenquanto;

enquanto esq <= fimEsq faca

tmp[k] = v[esq];

esq = esq + 1;

k = k + 1;

custo = custo + 1;

fimenquanto;

enquanto meio <= dir faca

tmp[k] = v[meio];

meio = meio + 1;

k = k + 1;

custo = custo + 1;

fimenquanto;

para i = 0 ate tam - 1 faca

v[dir] = tmp[dir];

dir = dir - 1;

fimpara;

retorne custo + 3;

fim

procedimento msort(v: inteiro[], tmp: inteiro[], esq: inteiro, dir: inteiro): inteiro

inicio

variavel meio, custo: inteiro;

custo = 0;

se dir > esq entao

meio = (dir + esq) / 2;

msort(v, tmp, esq, meio);

msort(v, tmp, meio + 1, dir);

custo = custo + merge(v, tmp, esq, meio + 1, dir);

fimse;

retorne custo;

fim

procedimento mergesort(v: inteiro[]): inteiro

inicio

variavel tmp: inteiro[];

tmp = novo inteiro [tamanho(v)];

retorne msort(v, tmp, 0, tamanho(v) - 1);

fim

/\*\*\*\* quicksort \*\*\*\*/

variavel custoq: inteiro;

procedimento iMenorFimEviMenorPivo(i: inteiro, vfim: inteiro, v: inteiro[], pivo: inteiro): inteiro

inicio

se i < vfim entao se v[i] < pivo entao retorne 1; fimse; fimse;

retorne 0;

fim

/\* particione \*/

procedimento particione(v: inteiro[], ini: inteiro, vfim: inteiro): inteiro

inicio

variavel i, j, pivo: inteiro;

i = ini + 1;

j = vfim;

pivo = v[ini];

enquanto i <= j faca

enquanto iMenorFimEviMenorPivo(i, vfim, v, pivo) == 1 faca

i = i + 1;

custoq = custoq + 1;

fimenquanto

enquanto v[j] > pivo faca

j = j - 1;

custoq = custoq + 1;

fimenquanto

custoq = custoq + 2;

se i < j entao

troque(v, i, j);

i = i + 1;

j = j - 1;

senao

i = i + 1;

fimse;

fimenquanto;

troque(v, j, ini);

retorne j;

fim

/\* quicksort \*/

procedimento quicksort(v: inteiro[], ini: inteiro, vfim: inteiro)

inicio

se (vfim - ini) < 1 entao

retorne;

fimse

variavel p: inteiro;

p = particione(v, ini, vfim);

se ini < (p - 1) entao

quicksort(v, ini, p - 1);

fimse

se vfim > (p + 1) entao

quicksort(v, p + 1, vfim);

fimse

fim

inicio

variavel custo, i: inteiro;

variavel v, v2: inteiro[];

v = novo inteiro { 61, 57, 72, 18, 8, 30, 21, 93, 67, 15, 78, 12, 81, 32, 27, 92 };

v2 = novo inteiro [tamanho(v)];

escreva(" ORIGINAL = "); imprime(v); escreva("\n");

copia(v2, v);

custo = bolha(v2);

escreva(" BOLHA = "); imprime(v2); escreva("apos ", custo, " comparacoes\n");

copia(v2, v);

custo = selecao(v2);

escreva(" SELECAO = "); imprime(v2); escreva("apos ", custo, " comparacoes\n");

copia(v2, v);

custo = insercao(v2);

escreva(" INSERCAO = "); imprime(v2); escreva("apos ", custo, " comparacoes\n");

copia(v2, v);

custo = mergesort(v2);

escreva("MERGESORT = "); imprime(v2); escreva("apos ", custo, " comparacoes\n");

copia(v2, v);

custoq = 0;

quicksort(v2, 0, tamanho(v2) - 1);

escreva("QUICKSORT = "); imprime(v2); escreva("apos ", custoq, " comparacoes\n");

fim