Definição da Linguagem CMM

(Versão 1.4)

Características da linguagem:

- Imperativa;
- Fortemente tipada;
- Declaração explícita de variáveis;
- Vinculação estática de tipos;
- Sistema de escopo estático (léxico);
- Uma versão aproximada e (muito) restrita da linguagem C;
- CMM é o acrônimo de C-Mais-ou-Menos, uma linguagem mais ou menos parecida com C.

A seguir são especificadas as estruturas que compõem um programa:

Programa

Um programa consiste de uma sequência não vazia de declarações de variáveis e subprogramas. A última declaração deve ser obrigatoriamente a da rotina principal, pela qual se dará o início da execução do programa. Todas as declarações realizadas no programa (fora de qualquer subprograma) estão dentro do escopo global.

Variáveis

Vamos considerar dois tipos de variáveis: as simples e as agregadas. Variáveis simples suportam apenas um único valor de um determinado tipo em um determinado momento. Variáveis agregadas são arranjos, suportando mais de um valor de um mesmo tipo em um mesmo momento. A seguir temos como exemplo a declaração de variáveis simples e agregadas.

```
int a, b = 3, c = 1;
string str1, str2 = "String 2";
bool i, j = true;
int v[10], z[3] = {1, 5, 8};
```

Observe que múltiplas variáveis podem ser declaradas de uma vez e que elas podem ser inicializadas durante a declaração. A inicialização deve ser feita sempre com valores literais. Veja que para os arranjos, o tamanho do mesmo também deve ser especificado na declaração.

Tipos primitivos

Os tipos primitivos que a linguagem aceita são: números **inteiros**, **strings** e **booleanos**.

Subprogramas (procedimentos e funções)

A definição de procedimentos e funções possui uma sintaxe comum, exceto pela ausência do tipo de retorno para procedimentos. Diferentemente da linguagem C, não há separação entre declaração e definição de subprogramas, isto é, o subprograma é declarado em sua própria definição.

Definição de Procedimento

```
proc(int y) {
    if (y < 0) {
        return;
    }
    x = 2 * y;
}
```

Definição de Função

```
int func(int x, int y) {
    int z;
    z = x * y;
    return z + 1;
}
```

Definição de procedimento:

nome **(** listaDeParâmetros **) {** bloco **}**

Definição de função:

tipo nome **(** listaDeParâmetros **) {** bloco **}**

Comandos

- **Atribuição:**

  variável **(** **=** | **+=** | **-=** | ***=** | **/=** | **%=** **)** expressão **;**

  O comando de atribuição avalia o valor da expressão e o armazena na variável. Uma atribuição somente pode ocorrer se a variável foi previamente declarada e se o tipo do resultado da expressão é o mesmo indicado na declaração da variável.

- **Condicional If:**

  if (teste **) {** então **}** [**else {** senão **}**]

  A estrutura condicional **if** é executada testando o resultado da expressão teste. Se ela resulte no valor **true**, o bloco então será executado. Se a expressão resultar no valor **false**, caso o bloco senão tenha sido construído, apenas ele será executado. Obrigatoriamente, a expressão teste deve resultar um valor booleano.

- **Laço While:**

  while (teste **) {** corpo **}**

  O laço **while** inicia testando o resultado da expressão teste. Caso o valor seja **true**, o bloco corpo é executado e o laço volta a testar o valor da expressão teste para a próxima iteração. Caso o valor seja **false**, a execução do laço é interrompida. Obrigatoriamente, a expressão teste deve resultar um valor booleano.

- **Laço For:**

  for (inic **;** teste **;** passo **) {** corpo **}**

  O laço **for** inicia executando o comando de atribuição inic. A partir daí, antes de cada iteração, o resultado da expressão teste é testado. Se ele for **true**, o bloco corpo é executado e o comando de atribuição passo é executado em seguida, reiniciando o processo. Se o valor resultado pela expressão teste for **false**, a execução do laço é interrompida. Obrigatoriamente, teste deve ser uma expressão do tipo booleano. Obs.: você pode definir uma sintaxe mais conveniente para o laço **for**, contanto que ele contenha a variável de

controle do laço, os valores inicial e final da variável de controle e uma forma de descrever como a variável de controle irá ser alterada a cada iteração (passo da iteração).

- **Interrupção de Laço:**

  break ;

  O comando **break** interrompe o laço mais próximo que o cerca. Ele só pode aparecer dentro do corpo de comandos de repetição **while** e **for**.

- **Retorno de subprograma:**

  return [expressão] **;**

  O comando return encerra a execução do subprograma que o cerca retornando o valor resultado pela expressão. A expressão de retorno de uma função deve resultar em um valor do mesmo tipo para o qual a função foi definida. Funções devem obrigatoriamente conter pelo menos um comando return. Já procedimentos podem ou não conter comandos return. Caso o tenham, eles devem retornar nada: return; Como o programa principal é definido por meio de um subprograma (função ou procedimento), ele também pode conter comandos return.

- **Chamada de procedimento:**

  nome **(** listaDeExpressões **) ;**

  Como a chamada de procedimentos não resulta em um valor, é necessário um comando para sua execução. A chamada de funções possui sintaxe semelhante, exceto por não ser um comando, e sim uma expressão.

- **Entrada Read:**

  read variável **;**

- **Saída Write:**

  write listaDeExpressões **;**

Bloco

Um bloco é uma sequência de (nenhuma ou várias) **declarações de variáveis** seguida de uma sequência de (nenhum ou vários) **comandos**. Um bloco define um novo escopo local.

Expressão

Uma expressão pode conter valores dos três tipos definidos (inteiros, string e booleanos), variáveis, chamadas de função e outras expressões. Uma expressão pode estar cercada por parênteses e se relacionada a outras expressões por meio dos seguintes operadores:

Precedência	Operador	Descrição	Associatividade
1	-	Negativo Unário	À direita
	!	Não lógico	
2	* / %	Multiplicação, divisão e resto	À esquerda

3	`+ -`	Adição e subtração	
4	`< <=`	Operadores relacionais < e ≤ respectivamente	
	`> >=`	Operadores relacionais > e ≥ respectivamente	
5	`== !=`	Operadores relacionais = e ≠ respectivamente	
6	`&&`	E lógico	
7	`||`	OU lógico	
8	`? :`	Condicional ternário	À direita

- O operador condicional ternário é formado da seguinte maneira:

  `teste` **`?`** `então` **`:`** `senão`

  A expressão `teste` é avaliada. Se o resultado for **`true`** a expressão `então` é resultada, caso contrário, a expressão `senão` é resultada. Dessa forma, o resultado desse operador é sempre uma expressão. O operador pode ser utilizado assim: `x = a > 0 ? a * 2 : a + 1;`

  Obrigatoriamente, a expressão `teste` deve resultar em um valor booleano.

- Uma lista de expressões consiste de nenhuma ou várias expressões separadas por vírgula:

  Exemplo:

  `2+3, 7 + f(a, 2), "Olá, mundo!"`

Convenções Léxicas

Identificadores

Os identificadores seguem a mesma regra de formação da linguagem C:

- Devem iniciar com uma letra (minúscula ou maiúscula) ou um subtraço seguido de letras, subtraços ou dígitos entre 0 e 9.

Números

Apenas números inteiros serão aceitos. Os números negativos não serão processados na fase léxica, mas sim na sintática e semântica. Dessa forma, o número -4, por exemplo, consiste de dois *tokens*: `<'-', neg>` e `<4, num>` e serão tratados como uma operação aritmética nas fases sintática e semântica.

Strings

As strings são como na linguagem C, exceto que os únicos caracteres de controle são o \t e \n, os quais devem ser substituídos lexicamente pelos caracteres ASCII correspondentes.

Comentários:

A linguagem possui dois tipos de comentários, semelhantes à linguagem C:

- **Comentários monolinha:** começam com `//` e seguem até o final da linha.

- **Comentários multilinha:** começam com /* e avançam até a primeira ocorrência de */

De forma geral, os comentários podem conter qualquer tipo de símbolo, inclusive os não permitidos pela linguagem. Os comentários devem ser processados corretamente pelo analisador léxico e em seguida descartados.

Palavras reservadas e símbolos:

```
bool break for false if int return string true while
( ) [ ] { } , ; + - * / == != > >= < <= || && ! = += -= *= /= %= ? :
```

Exemplo de programa:

Bubble-Sort

```
int v[10];

/*
   Procedimento de ordenação por troca
   Observe como um parâmetro de arranjo é declarado
*/
bubblesort(int v[], int n) {
    int i=0, j;
    bool trocou = true;
    while (i < n-1 && trocou) {
        trocou = false;
        for (j=0; j < n-i-1; j+=1) {
            if (v[j] > v[j+1]) {
                int aux;
                aux = v[j];
                v[j] = v[j+1];
                v[j+1] = aux;
                trocou = true;
            }
        }
        i += 1;
    }
}

main() {
    int i;
    for (i=0; i < 10; i+=1) {
        read v[i];
    }
    bubblesort(v, 10);
    for (i=0; i < 10; i+=1) {
        write v[i], " ";
    }
}
```

Especificação inicial da gramática

```
<programa> ::= <seqDeclaração>
<seqDeclaração> ::= <declaração> {<declaração>}

<declaração> ::= tipo id ';'
               | ...

<comando> ::= <comandoIf>
            | <comandoFor>
            | ...

<comandoIf> ::= 'if' '(' <expressão> ')' '{' <bloco> '}'
              | 'if' '(' <expressão> ')' '{' <bloco> '}' 'else' '{' <bloco> '}'

<expressão> ::= num
              | str
              | lógico
              | <variável>
              | <chamadaFunção>
              | <expressão> <op> <expressão>
              | ...

<bloco> ::= ...

<op> ::= ...
```