Manual para a linguagem DP

Escrito por: Eduardo C. Viana, Davi F. Nonnenmacher e Leomar M. Marschalk

Função principal do programa:

A função principal do compilador, é conhecida como função main, cujo valor de retorno é vazio (void) e não recebe parâmetros.

Dentro do bloco de código da função, deve-se conter, respectivamente, declarações de todas as variáveis a serem utilizadas no bloco de código da função principal, declarações de funções, e por fim o corpo do programa em si, sendo que para inicializar a implementação do corpo da função, o mesmo deve ser precedido e sucedido de **início** e **fim**. A sintaxe da função principal pode ser analisada abaixo.

```
void main()
{
      <declaração de variáveis>
      <declaração de funções>
      inicio
      <corpo>
      fim
}
```

Declaração de funções:

As declarações de funções neste compilador possuem sintaxes similares às linguagens atuais, como por exemplo o C.

Para se declarar uma função, deve-se informar primeiramente o tipo de retorno da função, seguido do nome da função, o tipo dos parâmetros que a função recebe e, por fim, o bloco de função em si, que por sinal possui a mesma sintaxe da função main. Note que, obrigatoriamente, a função deve retornar algo, caso for uma função do tipo **void**, deve-se conter apenas o **return**. Um exemplo de uma declaração de função pode ser visualizada abaixo, onde está sendo declarado uma função do tipo inteiro que retorna o valor 5 como padrão.

```
integer retorna_valor_fixo_5(integer; integer)
{
      <declaração de variáveis locais à função>
      <declaração de funções locais à função>
      inicio
      <corpo>
      fim
      return 5;
}
```

Obs: Nosso grupo acredita, contudo, que exista uma inconsistência na linguagem, pois não conseguimos nos referir aos parâmetros que a função recebe (só sabemos seus tipos).

Corpo do bloco de função:

O corpo de um bloco de função, seja ele da função main ou de outra função qualquer, consiste de instruções lógicas e aritméticas, atribuições de variáveis e/ou de laços de repetição. De forma similar à popular linguagem C, toda instrução deve ser sucedida de ponto e vírgula. Não podem haver declarações de variáveis nem de funções no corpo de um bloco.

Tipos e Variáveis:

As variáveis deste compilador são fortemente tipadas, de modo que toda declaração de variável deve-se especificar seu tipo. A linguagem suporta quatro tipos primitivos, que são: **integer**, **float**, **string**, **char**.

A sintaxe para se declarar uma variável deve-se proceder sucessivamente do seu nome e tipo, como por exemplo:

```
idade: integer;
```

Para se declarar diversas variáveis em uma mesma linha de tipos diferentes deve-se informar sucessivamente o nome da variável, seu tipo e o sinal de **;** declarando o fim da declaração da variável específica. Um exemplo pode ser visualizado abaixo:

```
idade1: integer; nome1: string; total: float;
```

Ademais, caso estejam sendo declaradas diversas variáveis do mesmo tipo, elas podem ser especificadas da seguinte forma:

```
idade1, idade2, idade3: integer; peso1, peso2, peso3: float;
```

Instruções condicionais:

Comandos podem ser colocados dentro de blocos condicionais de código para que sejam ou não executados com base em determinada condição. A sintaxe para isso é:

```
if(<condição>)
{
      <comandos>
}
```

Há também o "caso contrário" (else), de maneira similar às linguagens mainstream (e.g. C, C++, Java)

Exemplo de bloco condicional:

```
if(variavel1 == variavel2)
{
      <o que fazer caso o valor das variáveis seja igual>
}
else
{
      <o que fazer caso o valor das variáveis seja diferente>
}
```

Importante nota: Inexistem concatenadores lógicos na linguagem, de modo que é impossível, em uma condição apenas, expressar "se determinada coisa e/ou determinada outra coisa, então". Podem ser usados, para expressarem algo similar, concatenação de blocos de if/else.

Comparadores existentes na linguagem:

Este compilador possui 6 tipos primitivos de comparadores, sendo eles:

```
== (checa igualdade)
!= (checa desigualdade)
> (checa se determinado valor numérico é superior a outro)
< (checa se determinado valor numérico é inferior a outro)
>= (checa se determinado valor numérico é superior ou igual a outro)
<= (checa se determinado valor numérico é inferior ou igual a outro)
```

Loops fornecidos pela linguagem:

A linguagem fornece dois tipos de laços de repetições, sendo eles, o laço **while** (enquanto) e o laço "**for**" (para).

A sintaxe do laço **while** segue abaixo, onde necessita se informar uma condição que se prove verdadeira, e a mesma é conferida sempre que o laço termina de executar seus comandos verificando assim se a condição ainda é verdadeira.

```
while(<condição>)
{
      <comandos a serem executados enquanto a condição ser verdadeira>
}
```

Já a sintaxe do laço **for**, necessita informar uma condição inicial, sucessivamente de uma condição que se prove falsa, e também uma operação de incremento na condição para executar a cada fim de execução dos comandos do laço, a condição falsa é conferida sempre que o laço termina de executar seus comandos.

```
for(<condição inicial>; <condição> ; <incremento>)
{
       <comandos a serem executados enquanto a condição ser verdadeira>
}
```

Exemplos de Laços:

```
while(variavel1 != variavel2)
{
      <comandos a serem executados enquanto a primeira variável diferir da
segunda>
}

for(variavelDoLoop = 0 ; variavelDoLoop < 10 ; ++variavelDoLoop)
{
      <comandos a serem executados dez vezes>
}
```

Entrada e saída de dados:

Para solicitar um dado ao usuário e armazená-lo em uma variável, usa-se:

```
cin >> <nome da variavel>
```

Para exibir dados na tela, usa-se a instrução **cout**:

```
cout << literal
```

Diferentemente de **cin**, o comando **cout** precisa ser iniciado com um literal, permitindo com que sejam concatenados mais literais ou variáveis para exibição de dados, por exemplo:

```
cout << literal << <nome da variavel> <<
<nome da variavel 2>, <nome da variavel 3> << literal
```

Chamadas de função:

Para se chamar uma função, a sintaxe é:

```
callfunc <nome da função>(<parâmetros separados por vírgula>)
```

Exemplo:

```
callfunc exibe_numeros(numero1, numero2, numero3)
```

Atribuições:

A linguagem suporta atribuições de maneira similar às linguagens popularmente conhecidas. Exemplos:

```
numero1 = 5
numero1 = 2 * (1 + 3) / 5
```

Importante ressaltar a existência de açúcar sintático para o incremento unitário, de modo que as instruções abaixo são equivalentes às instruções citadas acima.

```
numero1 = numero1 + 1
++numero1
```