Alunos:

Cesar Henrique Cícero

Pedro Augusto de Almeida Costa Tenório

Especificação da linguagem

Estrutura geral:

- Os comandos do programa são separados por ‘;’
- variáveis globais e funções são definidas logo no início do arquivo, não serão permitidos repetição de nomes de variáveis globais e funções (identificadores), mesmo que seja uma variável global e outro variável local, nem será permitido variáveis terem o mesmo identificador que nomes de funções. Variáveis globais são definidas no início do programa, antes das definições das funções, no formato:

‘tipo identificador [= const | expressão]?[‘,’ identificador [=const | expressão]?]*’;’’

Funções são definidas como:

‘’FUNC’ identificador’(‘[tipo ? identificador [= const]? [‘,’ tipo ? identificador [= const]?]*]?’)’’{‘
bloco
[‘RET’ identificador |expressão]?
‘}’’

- Variáveis locais são consideradas apenas variáveis de argumentos de funções ou a variável de iteração, e só serão usadas dentro da função ou iteração. Não é possível defini-las com expressões, e é possível repetir o nome de variáveis locais contanto que elas estejam em contextos diferentes (funções diferentes ou iterações que não estão uma dentro de outra)
- Não será permitido funções serem definidas depois do ponto inicial de execução e nem definir funções dentro de outras funções, e a ordem das funções e variáveis é importante, só podemos chamar elas após elas serem definidas de forma sequencia;

Ex: INT a = 2; b = a, c = a+b; (PERMITIDO)
Ex: INT a = 2; b = c, c = 4; (NÃO PERMITIDO)
Ex:

```
FUNC a(){}
FUNC b(){
a();
}(PERMITIDO)
FUNC c(){
d();
}(NÃO PERMITIDO)
FUNC d() {} (PERMITIDO)
```

- O ponto inicial de execução do programa começará após a linha “INIT:”, sendo que essa deve ser única, não é permitido definir variáveis que não sejam de iteração depois do INIT

Tipos:

- Inteiros: são definidos pelo tipo INT onde constante literal tem o formato ‘-?[0-9]+’, caso não seja definido é assumido o valor 0 como padrão, a variável de iteração será sempre considerado INT
- Ponto Flutuante: são definidos pelo tipo REL onde constante literal tem o formato ‘-?[0-9]+[.[0-9]*]?[E-?[0-9]+]?’, caso não seja definido é assumido o valor 0.0 como padrão

- Caracteres: são definidos pelo tipo LET onde constante literal tem o formato ‘\’.\’’, caso não seja definido é assumido o valor ‘\0’ (NULL) como padrão
- String: são definidos pelo tipo TER onde constante literal tem o formato ‘\”[[^\”] | \“\”]*\”’, caso não seja definido é assumido o valor “” (vazio) como padrão, para representar “ dentro da string usamos o parênteses duplo “”
- Booleana: são definidos pelo tipo BOL onde constante literal tem o formato ‘T | F’, caso não seja definido é assumido o valor F (falso) como padrão
- Agregado: são definidos pela chave VET[tipo] onde a constante literal tem o formato ‘\[[const[, const]? \], todas as constantes com o mesmo tipo,caso não seja definido é assumido o valor[] (agregador de tamanho 0) como padrão. a[:] retorna o tamanho do agregador a;

Ex: VET[INT] a, b = [4, 7]

Operações (INT i1,i2; REL r1, r2; LET l1, l2; TER t1, t2; LOG b1, b2; VET[tipo] v1, v2) :

Associatividade dos operadores: da esquerda para a direita

Operador ‘+’:

i1 + i2 (Soma inteira que retorna inteiro) comutativo

i1 + r1 (Soma real que retorna real e converte o inteiro para real) comutativo

i1 + l1 (concatena inteiro com char, converte inteiro e char para string e concatena)

i1 + t1 (concatena inteiro com string, converte inteiro para string e concatena)

i1 + b1 (não é suportado [erro]) comutativo

i1 + v1 (converte o inteiro para o agregador [i1] e realiza a operação de soma)

r1 + r2 (Soma real que retorna real) comutativo

r1 + l1 (concatena real com char, converte real e char para string e concatena)

r1 + t1 (concatena inteiro com string, converte real para string e concatena)

r1 + b1 (não é suportado [erro]) comutativo

r1 + v1 (converte o real para o agregador [r1] e realiza a operação de soma)

l1 + l2 (concatena char com char, converte chars para string e concatena)

l1 + t1 (concatena char com string, converte char para string e concatena)

l1 + b1 (converte o booleano e char em string e concatena)

l1 + v1 (converte o char para o agregador [l1] e realiza a operação de soma)

t1 + t2 (concatena, une as strings colocando o início de t2 no fim de t1)

t1 + b1 (converte o booleano em string e concatena)

t1 + v1 (converte o string para o agregador [t1] e realiza a operação de soma)

b1 + b2 (não é suportado [erro]) comutativo

b1 + v1 (converte o booleano para o agregador [b1] e realiza a operação de soma)

v1 + v2 (só é suportado somando dois agregadores de mesmo tipo, os tamanhos são somados e concatena os dois agregadores)

Operador ‘-’ (só é suportado em operações com números):

- i1 (operador unitário negativo, inverte o sinal do número)

- r1 (operador unitário negativo, inverte o sinal do número)

i1 - i2 (subtração inteira que retorna inteiro)

r1 - r2 (subtração real que retorna real)

i1 - r1 ou r1 - i1 (subtração real que converte inteiro em real e faz a subtração retorna real)

Operador ‘*’ (só é suportado em operações com números):

i1 * i2 (multiplicação inteira que retorna inteiro)

r1 * r2 (multiplicação real que retorna real)

i1 * r1 ou r1 * i1 (multiplicação real que converte inteiro em real e faz a multiplicação retorna real)

Operador ‘/’ (só é suportado em operações com números):

i1 / i2 (divisão inteira que retorna inteiro)

r1 / r2 (divisão real que retorna real)

i1 / r1 ou r1 / i1 (divisão real que converte inteiro em real e faz a divisão retorna real)

Operador ‘>’ ou ‘<’ (maior que, menor que):

i1 > i2 ou i1 < i2 (comparação entre inteiros que retorna booleano)

r1 > r2 ou r1 > r2 (comparação entre reais que retorna booleano)

i1 > r1 ou r1 > i1 (comparação real que converte inteiro em real e faz a comparação retorna booleano)

Comparar números com caracteres ou strings não é suportado, também não é suportado comparar 2 strings

l1 > l2 ou l1 < l2 (comparação entre caracteres, compara a posição do caractere na tabela ASCII)

s1 > l1 ou l1 > s1 (converte caractere para string e faz a comparação entre strings)

s1 > s1 ou s1 > s1 (faz a comparação se a primeira string vier primeiro que na ordem alfabética ela é menor)

Operador ‘==’ ou ‘!=’ (igualdade, desigualdade):

comparar números retorna T se ambos os números forem iguais bit a bit, caso compare inteiro com real, converte o real para inteiro, a comparação também ocorre se for entre booleanos, se os dois booleanos tiverem os mesmos valores, eles são iguais, caso contrários diferentes.

Operadores ‘!’, ‘^’, ‘|’ (negação, conjunção e disjunção):

operadores comuns entre booleanos, não sendo permitidos em outros tipos

Instruções:

Estrutura do condicional:

Condicional começa com o SE, seguido de uma expressão booleana e { bloco } com o opcional de um SEN{bloco} caso contrário:

‘SE expLog {
bloco
}[SEN{
bloco
}]?‘

Interativo com controle lógico:

Começa com o ENQ, seguido de uma expressão booleana e { bloco } sendo que será sempre uma operação pré-teste:

‘ENQ expLog {
bloco
}’

Interativo controlada por controlador:

Começa com o PARA, seguido de um identificador (variável tipo inteiro local), duas constante inteira separadas por vírgula e como opcional uma terceira constante inteira, seguido de { bloco } sendo que a primeira constante inteira será o valor inicial da variável, a segunda constante será o valor final (não inclusivo) e a terceira o incremento, caso não apareça será considerado 1:

‘PARA identificador = const_int, const_int [, const_int]? {

bloco
}’

Entrada:

O comando de entrada deverá começar com a palavra reservada TEC e uma lista de identificadores seguidos por vírgula, a entrada digitada no terminal deverá ter o mesmo tipo dos identificadores que forem armazenados, caso contrário será gerado um erro durante a execução. a conversão de string para os tipos dos identificadores será realizada implicitamente durante a execução:
‘TEC identificador [‘,’ identificador]*’;’’

Saída:

O comando de saída deverá começar com MON e uma lista de identificadores e/ou constantes, todas as variáveis serão convertidas implicitamente durante a execução do programa para tipo TER e serão concatenadas para imprimir um único tipo TER no monitor.

Para quebrar linha na formatação da saída, deve-se usar o string “\n”:
‘MON identificador|const [‘,’ identificador|const]* ‘;’

Atribuição:

Utiliza-se de um comando “=” para gerar uma atribuição, entre a variável e o valor que será atribuído, respectivamente, o comando irá converter o tipo da expressão ou valor da direita para o tipo ou expressão do identificador da esquerda, caso essa conversão não seja possível (converter string em inteiro, por exemplo), será gerado um erro durante a execução do programa
INT a = 3
TER b = “compiladores”
‘[tipo]? identificador = identificador|const|expressão’;’’

Subprogramas:

Na função (já descrita neste arquivo), em algum lugar antes do ‘}’, será permitido usar ‘RET’ seguido do nome da variável a qual deseja-se retornar ou expressão lógica ou matemática, para obter o seu retorno, podendo ser de qualquer tipo.

Para passar o parâmetro em uma função, deve-se chamá-la por seu nome e em seguida digitar o valor(es) entre parênteses, caso seja mais de um, separá-los por vírgula.
funcao(1);
funcaob(3, 5, 9);