DEPENDÊNCIAS DE CONTEXTO DA LINGUAGEM U

Revisão C

<bloco> ::=
{ <lista de declarações> <lista de comandos> }

1. As variáveis globais são alocadas estaticamente; as variáveis locais são alocadas de forma automática;
2. As regras de escopo são as regras usualmente adotadas para linguagens com blocos aninhados.

<chamada de função> ::=
<nome> ((<lista de expressões>|ε))

1. A quantidade de expressões da lista de expressões deve ser igual a quantidade de parâmetros declarados;
2. Os tipos das expressões devem ser iguais aos tipos dos argumentos (correspondência posicional).

<comando> ::=
<comando simples> |
<comando condicional> |
<comando iterativo> |
{ <lista de comandos> } |
ε

<comando condicional> ::=
if <expressão> then <comando> |
if <expressão> then <comando> else <comando>

1. A expressão deve ser do tipo boolean.

<comando iterativo> ::=
while <expressão> do <comando> |
repeat <comando> until <expressão> |
for <nome de variável> := <expressão1> to <expressão2> (step <expressão3>|ε) do <comando>

1. Nos comandos while e repeat a expressão deve ser do tipo boolean;
2. No comando for, <nome de variável>, <expressão 1>, <expressão 2> e <expressão 3> devem ser do tipo inteiro.

<comando simples> ::=
<nome de variável> := <expressão> |
<chamada de função> |
return <expressão>

1. <nome de variável> deve ter o mesmo tipo de <expressão> no comando de atribuição;
2. A função chamada deve ser do tipo void;
3. O tipo de <expressão> no comando return deve ser igual ao tipo do valor retornado pela função.

<declaração> ::=
<declaração de variável> |
<declaração de função>

<declaração de função> ::=
(<tipo simples>|void) <nome> ((<lista de parâmetros>|ε)) <bloco>

1. Os parâmetros são passados sempre por valor;
2. Se o tipo da função for void, ela deve ser usada apenas como abstração de controle (procedimento); caso contrário apenas como abstração de operação (função).

<declaração de variável> ::=
<tipo> <lista de nomes>

1. Todas as variáveis possuem o mesmo tipo.

<expressão> ::=
<expressão simples> (<operador relacional> <expressão simples>|ε)

<expressão simples> ::=
<expressão simples> <operador aditivo> <termo> |
<termo>

<fator> ::=
<nome de variável> |
<chamada de função> |
<número> |
(<expressão>)

1. A função chamada deve ter um tipo diferente de void.

<lista de comandos> ::=
<lista de comandos> ; <comando> |
<comando>

<lista de declarações> ::=
<declaração> ; <lista de declarações> |
ε

<lista de expressões> ::=
<lista de expressões> , <expressão> |
<expressão>

<lista de nomes> ::=
<lista de nomes> , <nome> |
<nome>

<lista de parâmetros> ::=
<lista de parâmetros> , <parâmetro> |
<parâmetro>

<nome de variável> ::=
<nome> (ε | [<expressão>] | [<expressão>,<expressão>])

1. Se <nome> for do tipo simples, então não deverá haver indexação; se <nome> for um vetor, deverá have uma única expressão indexadora; se <nome> for uma matriz, então deverá haver duas expressões indexadoras;
2. O tipo das expressões indexadoras deve ser inteiro.

<operador aditivo> ::=
+ | - | or

1. +: int x int → int
2. +: float x flaot → float
3. +: int x float → float
4. +: float x int → float
5. -: int x int → int
6. -: float x float → float
7. -: int x float → float
8. -: float x int → float
9. or: boolean x boolean → boolean

<operador multiplicativo> ::=
<* | / | and>

1. *: int x int → int
2. *: float x flaot → float
3. *: int x float → float
4. *: float x int → float
5. /: int x int → int
6. /: float x float → float
7. /: int x float → float
8. /: float x int → float
9. and: boolean x boolean → boolean

<operador relacional> ::=
<> | < | >= | <= | <> | =

1. >, <, >=, <=: int x int → boolean
2. >, <, >=, <=: float x float → boolean
3. >, <, >=, <=: float x int → boolean
4. >, <, >=, <=: int x float → boolean
5. <>, =: int x int → boolean
6. <>, =: float x float → boolean
7. <>, =: float x int → boolean
8. <>, =: int x float → boolean
9. <>, =: boolean x boolean → boolean

<parâmetro> ::=
<tipo simples> <nome>

<programa> ::=
main <bloco>

<termo> ::=
<termo> <operador multiplicativo> <fator> |
<fator>

<tipo> ::=
<tipo simples> |
<tipo agregado>

<tipo agregado> ::=
[(<número inteiro>,< número inteiro> | <número inteiro>)] <tipo simples>

<tipo simples> ::=
int |
bool |

float