UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
COLEGIADO DE CIÊNCIA DA COMPUTAÇÃO

Especificação linguagem de programação Upper

Alunos: Gabriel Medina Marques e Vitor Luiz Caldeira Gilnek

Cascavel - Dezembro de 2021

1. Introdução

Essa documentação se refere à especificação da linguagem de programação Upper.

2. Tipos de dados

Upper é uma linguagem fortemente tipada baseada na sintaxe da linguagem C, uma vez que essa sintaxe é vastamente conhecida na comunidade de programação.

- **Inteiro**
  Identificador: INT
  Esta é uma representação do conjunto dos números inteiros. Sua faixa de representação terá um limite entre -2^{31} e $2^{31}-1$, sendo representado utilizando 4 bytes, tendo valor de iniciação 0 caso não seja atribuído nenhum valor.

- **Ponto Flutuante**
  Identificador: FLOAT
  Esta é uma representação aproximada do conjunto de números racionais, sua faixa de representação será de 1.2^{-3} até 3.4^{38}, possuindo 4 bytes para representação, tendo como valor de iniciação 0.0 caso não seja atribuído nenhum valor.

- **Booleano**
  Identificador: BOOL
  Esta é uma repŕesentação para facilitar a legibilidade do código, tendo como possíveis valores “True” e “False”, respectivamente, para 1 e 0, possuindo 1 byte para armazenamento.

3. Operadores Suportados

a. Lógicos

- **Disjunção**
  Operador: ||
  Representa a operação binária do OR lógico, ou seja, caso um dos operandos seja verdadeiro o resultado é verdadeiro, e falso se nenhum operando for verdadeiro.

- **Conjunção**
  Operador: &&
  Representa a operação binária do AND lógico, ou seja, caso todos os operando sejam verdadeiros o resultado é verdadeiro e caso algum operando seja falso o resultado é falso.

- **Negação**
  Operador: ! (ponto de exclamação)
  Representa a operação binária NOT lógico, ou seja, a operação consiste em inverter o valor lógico do operador bit a bit.

b. Aritméticos

- **Adição**
  Operador: +
  Se refere a operação binária que retorna a soma de dois operandos.

- **Subtração**
  Operador: -
  Se refere a operação binária que retorna a diferença de dois operandos.

- **Divisão**
  Operador: /
  Se refere a operação binária que retorna a divisão de dois operandos.

- **Multiplicação**
  Operador: *
  Se refere a operação binária que retorna a multiplicação de dois operandos.

c. Relacionais

- **Igualdade**
  Operador: ==
  Se refere a operação binária de igualdade, ou seja, retorna verdadeiro se os dois operandos possuírem valores iguais, e caso sejam diferentes retorna falso.

- **Maior que e Maior igual que**
  Operador: >, >=
  "Maior que" reflete a operação binária que retorna verdadeiro caso o primeiro operando seja maior que o segundo, enquanto a operação "Maior igual que" é uma operação que retorna verdadeiro caso o primeiro valor seja maior ou igual o segundo.

- **Menor que e Menor igual que**
  Operador: <, <=
  "Menor que" reflete a operação binária que retorna verdadeiro se o primeiro valor for menor que o segundo, enquanto o "Menor igual que" retorna verdadeiro se o primeiro operando for menor ou igual o segundo.

4. **Estruturas**
    a. **Salto**

    Estrutura:

    IF(<statement>){
        <expr>
    } ELSE {
        <expr>
    }

    Consiste em uma estrutura de decisão que testa se a operação "<stament>" é verdadeiro, caso seja, executa o bloco definido a seguir e caso contrário avança para a próxima estrutura de decisão. Apenas a operação "IF" possui expressões que podem ser lógicas, a estrutura sempre deve ser acompanhada de chaves, assim como a operação "ELSE".

    b. **Repetição**

    Estrutura:

    WHILE(<statement>){
        <expr>
    }

    Consiste de uma estrutura que executa um bloco de código enquanto a condição de controle seja verdadeira, o teste é feito sempre antes da execução do bloco. A operação "WHILE" deve ser seguida da condição entre parênteses e possuir chaves entre o bloco a ser executado.

5. **Atribuição**

    Operador: =

    Consiste da operação binária do tipo "a=b", onde a variável à esquerda recebe a valor da variável a direita, as variáveis precisam ser do mesmo tipo para a atribuição acontecer de forma correta, caso contrário, o programa irá possuir um erro.

6. **Entrada e saída**
    a. **Entrada**

    Operador: INPUT

    Consiste da operação de deixar o valor de uma variável ser lido pelo usuário executando o programa, o valor irá ser atribuído de acordo com o tipo da variável sendo lida.

    b. **Saída**

    Operador: PRINT

    Consiste da operação de mostrar o valor de alguma variável para o usuário na execução do programa.

7. **Formação de identificadores**

As variáveis podem ser declaradas utilizando letras minúsculas, números ou o underline, onde o nome das mesmas devem começar, obrigatoriamente, com letras minúsculas.

A declaração da variável occore através da definição do tipo, seguido pelo nome, por exemplo “INT a = 10”, sendo possível atribuir valores no momento da declaração ou após a declaração das variáveis

8. **Palavras reservadas**
    - **MAIN**
    - **INT**
    - **FLOAT**
    - **BOOL**
    - **PRINT**
    - **INPUT**
    - **IF**
    - **ELSE**
    - **WHILE**
    - **True**
    - **False**

9. **ExempĺO de sintaxe**

```
MAIN {
        INT a = 10;
        INT aux = 2;
        INT var;
        FLOAT b = 2.5;
        BOOL c = False;
        BOOL d = True;

        var = 10 - 2;
        var = 10 / 2;
        var = 10 * 2;

        PRINT(a);

        IF (a <= 9 OR b >= 3.0) {
                PRINT(a):
        }

        IF (!c AND d) {
                PRINT(a):
        }
```

```
        IF (a >= 10) {
                PRINT(a);
        } ELSE {
                PRINT(b);
        }

        FOR(INT a = 0; a > 10; a++) {
                INT b = 10;
                PRINT(A);
        }

        WHILE(d) {
                a = a + 1:
                IF (a > 10) {
                        d = False;
                }
        }

        INPUT(var);
        PRINT(var);
}
```

10. Definições Formais

- **Operadores**

  <EQ> :== “==”
  <LE> :== “<=”
  <GE> :== “>=”
  <LT> :== “<”
  <GT> :== “>”
  <OR> :== “||”
  <AND> :== “&&”
  <NOT> :== “!”
  <PLUS> :== “+”
  <MINUS> :== “-”
  <MULT> :== “*”
  <DIV> :== “/”

- **Atribuição**

  <ATTR> :== “=”

- **Entrada e Saída**

<CMD_PRINT> :== <PRINT> <LBRACKET> <ID> <RBRACKET> <PCOMMA>
<CMD_INPUT> :== <INPUT> <LBRACKET> <ID> <RBRACKET> <PCOMMA>

- **Palavras Reservadas**
  <MAIN> :== “MAIN”
  <FLOAT> :== “FLOAT”
  <INT> :== “INT”
  <BOOL> :== “BOOL”
  <IF> :== “IF”
  <ELSE> :== “ELSE”
  <WHILE> :== “WHILE”
  <PRINT> :== “PRINT”
  <INPUT> :== “INPUT”
  <TRUE> :== “True”
  <FALSE> :== “False”

- **Tipos**
  <INTEGER> :== “[0-9]+”
  <FLOATN> :== “<INTEGER>.<INTEGER>”
  <BOOLEAN> :== <TRUE>|<FALSE>
  <TIPO> :== <INTEGER> | <FLOATN> | <BOOLEAN>

- **ID**
  <ID> :== “[a-z]+ [_ , 0-9, a-z]*”

- **Caracteres**
  <LBRACKET> :== “(”
  <RBRACKET> :== “)”
  <LBRACE> :== “{”
  <RBRACE> :== “}”
  <PCOMMA> :== “;”

11. Especificação EBNF

- **Legenda**
  <declaração_var> = <dv>
  <declaração_estrutura> = >de>
  <estrutura_repetição> = <er>
  <operação> = <op>
  <operação_aritmetica> = <op_ar>
  <operador_relacional> = <sign_rel>
  <operação_relacional> = <op_rel>
  <operador_logico> = <sign_log>

<operação_logica> = <op_log>
<expressão> = <expr>
<especificação_var> = <ev>
<bloco_codigo> = <bk>

- **Programa**
  <program> :== <MAIN><bk>
  <bk> :== <LBRACE><expr><RBRACE>
  <stmt> :== <BOOLEAN> | <op> | <ID>

- **Expressão**
  <expr> :== <dv>|<dv> <expr> | <de> | <de> <expr> | <er> | <er>
  <expr> | <op> | <op> <expr> | <op_ar> | <op_ar> <expr> |
  <CMD_PRINT> | <CMD_PRINT> <expr> | <CMD_READ> |
  <CMD_READ> <expr>

- **Declaração de variável**
  <dv> :== <TIPO> <ID> <ev>
  <ev> :== <attr_dv> | <PCOMMA>
  <attr_dv> :== <ATTR> <valor> | <ATTR> <ID>
  <valor> :== <INTEGER> | <BOOLEAN> | <FLOATN>

- **Atribuição de valor**
  <atribuicao> :== <ID> <ATTR> <valor> | <ID> <ATTR> <ID>

- **Declaração de estrutura**
  <de> :== <IF> <LBRACKET> <stmt> <RBRACKET> <bk> | <IF>
  <LBRACKET> <stmt> <RBRACKET> <bk> <ELSE> <bk>

- **Estrutura de repetição**
  <er> :== <WHILE> <LBRACKET> <stmt> <RBRACKET>

- **Operações relacionais e lógicas**
  <op> :== <op_rel> | <op_log>
  <op_rel> :== <stmt><sign_rel><stmt>
  <op_lo> :== <stmt><sign_lo><stmt>

- **Operação aritmética**
  <op_ar> :== <op_ar> + <aux> | <op_ar> - <aux> | <aux>

<aux> :== <final> * <aux> | <final> / <aux> | <final>
<final> :== <LBRACKET> <op_ar> <RBRACKET> | <ID> | <valor>

12. Autômato