



UNIOESTE - Universidade Estadual do Oeste do Paraná
Centro de Ciências Exatas e Tecnológicas
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Linguagens de Programação

Especificação e definição da linguagem Unbending

Gilberto Antunes Monteiro Junior
Henrique Tomé Damasio
Marco Antônio Paixão Néia

1. Introdução

Essa documentação se refere a especificação da linguagem de programação Unbending, linguagem esta que obteve seu nome da qualidade de mesmo nome que de acordo com o dicionário Oxford significa "inflexível", essa definição é o que guia todo o design desta linguagem tendo a constância como seu mais importante valor.

2. Características da linguagem

Unbending é uma linguagem de programação fortemente tipada baseada na sintaxe da linguagem C, já que está é uma sintaxe largamente conhecida na comunidade de programação o uso de tal sintaxe facilitará a adoção da linguagem pela maior parte da comunidade.

Na definição de sua sintaxe várias condições serão amarradas de modo a fazer com que todo código escrito em Unbending tenha a mesma composição, o objetivo de adicionar essa característica a linguagem é evitar o surgimento de diferentes padrões de código, o objetivo é que todo o código Unbending que realize a mesma tarefa, usando as mesmas estruturas e na mesma ordem seja exatamente igual (com a exceção de nomes de variáveis), essa característica também tem a consequência secundária de diminuir a geração de código difícil de ler, já que este terá sempre a mesma estrutura.

3. Tipo de dados

↳ Inteiro

Identificador: **INT**

O inteiro é uma representação dos números naturais positivos e negativos, incluindo o zero. Sua faixa de representação terá um limite inferior igual a -2^{31} e limite superior igual a $2^{31}-1$, e representado por 4 bytes.

↳ Booleano

Identificador: **BOOLEAN**

O tipo Booleano, é uma abstração para aumentar a legibilidade do código. Seus valores possíveis são "true" quando seu retorno é igual a "1", e "false" para qualquer outro valor. Essa estrutura utiliza apenas 1 byte para o armazenamento do resultado.

↳ Float

Identificador: **FLOAT**

Float ou Ponto-flutuante é uma representação aproximada dos números racionais. A sua faixa de representação é de 1.2^{38} até 3.4^{38} . A separação entre a parte inteira e a fracionária é efetuada utilizando ponto.

4. Operadores lógicos

↳ Disjunção

Operador: ||

Representa à operação binária do OR lógico. Caso um dos operandos seja verdadeiro o resultado é verdadeiro, e falso se nenhum operando for verdadeiro.

↳ **Conjunção**

Operador: &&

Representa à operação binária do AND lógico. Caso todos os operandos sejam verdadeiros o resultado é verdadeiro, e falso caso um dos operandos não seja verdadeiro.

↳ **Negação**

Operador: !

Representa à operação unária do NOT lógico. Então a sua operação consiste em inverter o valor lógico do operador bit a bit.

5. Operadores aritméticos

↳ **Adição**

Operador: +

Se refere à operação binária que retorna a soma de dois operandos.

↳ **Subtração**

Operador: -

Se refere à operação binária que retorna a diferença entre dois operandos.

↳ **Multiplicação**

Operador: *

Diz respeito à operação binária que multiplica dois operandos.

↳ **Divisão**

Operador: /

Se refere à operação binária que divide dois operandos.

6. Operadores relacionais

↳ **Igualdade**

Operador: ==

Reflete a uma operação binária que retorna true se os dois operandos possuem valores iguais, e false caso contrário.

↳ **Maior que e Maior igual que**

Operador: >, >=

“Maior que”, reflete a operação binária que retorna true caso o primeiro operando seja maior que o segundo. Já a operação “Maior igual que”, condiz com o caso de que o primeiro valor é maior ou igual ao segundo.

↳ **Menor que e Menor igual que**

Operador: <, <=

Ambas operações são análogas às de “Maior que” e “Maior igual que”. Entretanto, o primeiro valor deve ser menor, no primeiro caso, ou menor igual no segundo, do segundo valor.

7. Estruturas de saltos

↳ **Condicional**

Estrutura:

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

Consiste em uma estrutura de decisão que testa o “<statement>” e executa o bloco definido a seguir caso o resultado do teste seja true. Caso contrário, a execução avança para a próxima estrutura de decisão. Apenas a cláusula “IF” possui expressões que podem ser lógicas e aritméticas, a estrutura deve ser acompanhada por chaves, independente do tamanho da expressão.

8. Estruturas de repetição

↳ **Laço de repetição por condição sem execução obrigatória**

Estrutura:

```
WHILE (<statement>){  
    <expr>  
}
```

Consiste em uma estrutura que executa um bloco de código enquanto a condição de controle seja true, onde o teste é feito sempre antes da execução do bloco. O laço é definido pela palavra reservada “WHILE” seguida da condição entre parênteses. O bloco a ser executado enquanto o teste for verdadeiro é definido entre chaves.

↳ **Laço de repetição por condição com uma execução obrigatória**

Estrutura:

```
DO{  
    <expr>  
}WHILE (<statement>)
```

Consiste em uma estrutura que executa um bloco de código enquanto a condição de controle seja true, onde o teste é feito sempre após a execução do bloco, obrigando que ocorra ao menos uma execução. O laço é definido pelas palavras reservadas “DO” e “WHILE”, a segunda seguida da condição entre parênteses. O bloco a ser executado enquanto o teste for verdadeiro é definido entre chaves.

9. Atribuição

Operador: =

Existe apenas um operador de atribuição para variáveis. Consiste em uma operação binária do tipo “a = b”, onde a variável mais à esquerda (a) recebe o valor da variável mais à direita (b), b pode também ter a forma de valor literal, operação matemática ou caractere. As variáveis devem ser do mesmo tipo para que a atribuição seja possível.

10. Palavras reservadas

MAIN	INT	FLOAT	BOOLEAN	IF
ELSE	WHILE	DO	READ	WRITE
TRUE	FALSE			

11. Construção de identificadores

As variáveis podem ser declaradas utilizando letras minúsculas, números e o caractere underline. Os nomes das mesmas devem começar apenas com letras minúsculas.

A declaração da variável ocorre através da definição do tipo, seguido pelo nome. É possível atribuir valores no momento da declaração.

12. Estrutura geral do programa/sintaxe

Para a execução de um código, é necessário apenas um bloco, delimitado pelas chaves que fecham o main (int main). A declaração das variáveis podem ser feitas em qualquer parte do bloco, desde que seja feita antes do uso da variável. Cada instrução deve ser finalizada com “ ; ”.

13. Comandos de entrada e saída

A palavra reservada READ representa o *stream* de entrada no Unbending. Ele realiza a leitura de um dado, sem espaços e sem tabulações, vindas do teclado.

<cmd_read> -> <read> <nome_var> <semicolon>

A palavra reservada WRITE representa o *stream* de saída no Unbending. Este *stream* é uma espécie de seqüência (fluxo) de dados a serem impressos na tela.

<cmd_write> -> <write> <nome_var> <semicolon>

14. Definições regulares

↳ Operadores

<sign_rel> -> “==” | “>” | “<” | “>=” | “<=”

<sign_lo> -> “||” | “&&” | “!”

<sign_ar> -> “+” | “-” | “*” | “/”

↳ Tipos

<int> -> “[0-9]+”

<float> -> "<int>.<int>"
<boolean> -> <true> | <false>

↳ **Id**

<nome_var> -> "[a-z]+ [_ , 0-9, a-z]* "

↳ **Palavras Reservadas**

<main> -> "MAIN"
<int> -> "INT"
<float> -> "FLOAT"
<boolean> -> "BOOLEAN"
<else> -> "ELSE"
<while> -> "WHILE"
<do> -> "DO"
<if> -> "IF"
<read> -> "READ"
<write> -> "WRITE"
<true> -> "TRUE"
<false> -> "FALSE"

↳ **Simbolos**

<par_esq> -> "("
<par_dir> -> ")"
<chave_esq> -> "{"
<chave_dir> -> "}"
<semicolon> -> ";"
<comma> -> ","

↳ **Atribuição**

<recebe> -> "="

15. Definição formal EBNF da linguagem Unbending

↳ **Abreviações utilizadas**

<declaração_var> = <dv>
<estrutura_repeticao> = <er>
<declaração_estrutura> = <de>
<comentário> = <cmt>
<operação> = <op>
<operador_relacional> = <op_rel>
<operador_relacional> = <sign_rel>
<operador_logico> = <sign_lo>
<operacao_logica> = <op_lo>
<operacao_aritmetica> = <op_ar>
<expressão> = <expr>
<especificação_var> = <ev>

<Alfabeto, Número, Underscore> = <ANU>
<operando_atribuição> = <oa>
<bloco_de_codigo> = <bk>

↳ **Programa**

<unbending> -> <main><bk>
<bk> -> <chave_esq><expr><chave_dir>
<stmt> -> <boolean> | <op> | <var>

↳ **Expressão**

<expr> -> [<dv> | <dv> <expr>] [<de> | <de> <expr>] [<er> | <er> <expre>]
[<op> | <op> <expr>] [<op_ar> | <op_ar> <expr>]

↳ **Declaração de variável**

<dv> -> <tipo> <nome_var> <ev>
<ev> -> <comma> <nome_var> <ev> | <atribuicao> <ev> | <semicolon>
<atribuicao> -> <recebe> <valor> | <recebe> <nome_var>
<valor> -> <int> | <float> | <boolean>

*<valor> se refere aos possíveis valores que se encaixam nos tipos especificados.

↳ **Atribuição de valor**

<atribuicao> -> <nome_var> <recebe> <valor> | <nome_var> <recebe>
<nome_var>
<valor> -> <int> | <float> | <boolean> | <expr>

↳ **Declaração de estrutura**

<de> -> <if> <par_esq> <stmt> <par_dir> <bk> |
<if> <par_esq> <stmt> <par_dir> <bk> <else> <bk>

↳ **Estrutura de repetição**

<er> -> <while><par_esq><stmt><par_dir><bk> |
 <do><bk><while><par_esq><stmt><par_dir>

↳ **Operações, relacional e lógica**

<op> -> <op_rel> | <op_lo>
<op_rel> -> <stmt> <sign_rel> <stmt>
<op_lo> -> <stmt> <sign_lo> <stmt>

↳ **Operação aritmética**

<op_ar> -> <op_ar> + <mid> | <op_ar> - <mid> | <mid>
<mid> -> <mid> * <inf> | <mid> / <inf> | <inf>
<inf> -> <par_esq><par_esq><op_ar><par_dir><par_dir> | <var>

