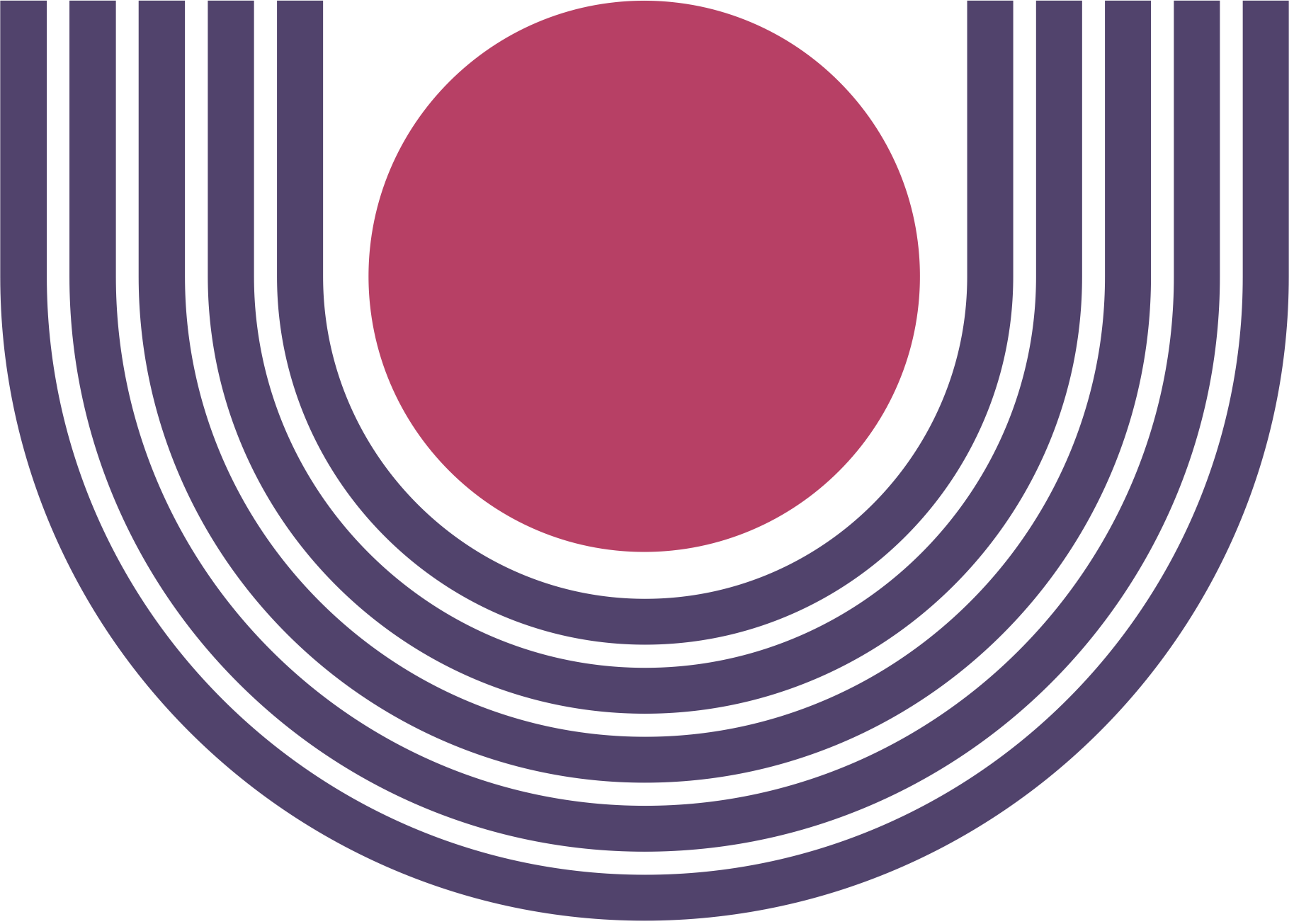
**UNIOESTE - Universidade Estadual do Oeste do Paraná**

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**

# 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.

# 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.

# 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 -2³¹ e limite superior igual a 2³¹-1, 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 1.238 até 3.438. A separação entre a parte inteira e a fracionária é efetuada utilizando ponto.

# 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.

# 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.

# 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.

# 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.

# 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.

# 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.

# Palavras reservadas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MAIN | INT | FLOAT | BOOLEAN | IF |
| ELSE | WHILE | DO | READ | WRITE |

# Construção de identificadores

As variáveis podem ser declaradas utilizando letras, números e o caractere underline. Os nomes das mesmas devem começar apenas com letras, sensíveis às maiúsculas e minúsculas. O tamanho máximo do nome da variável deve ser de 255 caracteres.

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

# 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 “ ; ”.

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

# 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> -> “{“<expr>”}”

<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> -> <,> <nome\_var> <ev> | <atribuicao> <ev> | <;>

<atribuicao> -> <recebe> <valor> | <recebe> <nome\_var>

<valor> -> int | float | boolean | char

\*<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>

<recebe> -> "="

<valor> -> int | float | boolean | char | <expr>

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

## Declaração de estrutura

<de> -> “IF(”<stmt>”)<bk> | “IF(”<stmt>”)<bk>“ELSE”<bk>

## Estrutura de repetição

<er> -> “WHILE(“<stmt>”)”<bk> | “DO<bk>WHILE(“<stmt>”)”

## Operações, relacional e lógica

<op> -> <op\_rel> | <op\_lo>

<op\_rel> -> <stmt> <sign\_rel> <stmt>

<op\_lo> -> <stmt> <sign\_lo> <stmt>

## Operadores

<sign\_rel> -> “==” | “>” | “<” | “>=” | “<=”

<sign\_lo> -> “||” | “&&” | “!”

## Operação aritmética

<op\_ar> -> <op\_ar> + <mid> | <op\_ar> - <mid> | <mid>

<inf> -> <mid> \* <inf> | <mid> / <inf> | <inf>

<inf> -> ( <op\_ar> ) | <var>