Universidade Federal de Alagoas
Instituto de Computação
Bacharelado em Ciência da Computação

Especificação da Linguagem - Isengard++

Márcio Henrique Vieira de Oliveira
Michael Miller Rodrigues Cardoso

Maceió - AL, 2021

Sumário

1. Introdução

A linguagem de programação **Isengard++** é estaticamente tipada, segue os paradigmas da programação funcional e imperativa, não orientada a objetos e toma como base a sintaxe da linguagem C. Seu principal objetivo é ser uma linguagem de fácil compreensão e aprendizado para programadores iniciantes.

Dessa forma, a linguagem **Isengard++** não admite coerção, possuindo maior confiabilidade, além de não existir tratamento para erros de detecção de tipo. As palavras reservadas da linguagem podem ser identificadas facilmente pelo usuário, tornando-a mais legível.

2. Estrutura geral do programa

Um programa na linguagem **Isengard++** deve ser estruturado seguindo o padrão abaixo:

- Para declarar a função principal do programa, deve ser utilizada a palavra reservada **Main**;
- A função principal deve ser declarada somente no fim do programa, após todas as outras funções declaradas ao longo do código;
- Para inicializar e finalizar uma determinada função / bloco de escopo, devem ser utilizadas as palavras reservadas **Begin** e **End** (abrir e fechar bloco, respectivamente);
- Funções são declaradas com a palavra reservada **Funct**, vindo em seguida o seu tipo e nome. Após seu nome deverá vir o bloco de parâmetros delimitado por parênteses com a declaração dos tipos e nomes das variáveis, que são separadas por vírgulas;
- O retorno padrão é apresentado de acordo com o tipo que foi referenciado na função (incluindo a **Main**), mas caso seja um retorno de função, será feita com a palavra reservada **Return** <argumento>;
- Os tipos das variáveis e/ou funções também possuem palavras reservadas, tendo como exemplo o tipo inteiro, denominado **Int**;
- As instruções de uma linha sempre encerram com “**;**”;
- As palavras reservadas sempre começam com letra maiúscula;
- Os identificadores sempre começam com letra minúscula.

Exemplo de programa (função que realiza a soma de dois inteiros):

```
Funct Int soma(Int n1, Int n2) Begin

    Int sum = n1 + n2;

    Return sum;

End
```

3. Conjuntos de tipo de dados e nomes

A linguagem apresenta o padrão case-sensitive, onde as palavras reservadas seguem o padrão Pascal Case.

3.1. Palavras Reservadas

A linguagem conta com as seguintes palavras reservadas:

Palavra Reservada	**Categoria**
Int	Tipo de Dado
Float	Tipo de Dado
Bool	Tipo de Dado
Char	Tipo de Dado
Str	Tipo de Dado
Void	Tipo de Dado
Null	Valor
Return	Instrução de Retorno
Funct	Função
Begin	Iniciar Função
End	Finalizar Função
Main	Função Principal
Input	Instrução de Entrada
Output	Instrução de Saída
And	Operador lógico
Or	Operador lógico
True	Valor do tipo Bool
False	Valor do tipo Bool
If	Estrutura Condicional
Else	Estrutura Condicional

For	Estrutura de Repetição
While	Estrutura de Repetição

3.2. Identificador

Os identificadores da linguagem **Isengard++** seguem as regras abaixo:

- O primeiro caracter de um identificador deve ser obrigatoriamente uma letra minúscula;
- Os caracteres seguintes do identificador podem ser maiúscula, minúscula, números e / ou underline.
- O tamanho máximo de um identificador é de 16 caracteres;
- É proibido o uso de palavras reservadas e de espaços em branco;
- Uma variável deve ser obrigatoriamente declarada antes de ser utilizada em uma determinada função / bloco.

3.3. Comentários

Os comentários da linguagem **Isengard++** só podem ser feitos por linha, através do símbolo '**#**'.

3.4. Inteiro

A palavra reservada **Int** identifica as variáveis com números do tipo inteiro possuindo um número de bits 32 bits. Seus literais são expressos em uma sequência de dígitos inteiros.
Ex.: **Int** value = 12;

3.5. Ponto flutuante

A palavra reservada **Float** identifica as variáveis com números do tipo ponto flutuante, possuindo um número de bits limitados a 32 bits. Seus literais são expressos em uma sequência de números inteiros seguidos por um ponto e outra sequência de números que representam a parte decimal.
Ex.: **Float** value = 23.81.

3.6. Booleano

A palavra reservada **Bool** identifica variáveis do tipo Booleano, em que podem ser atribuídos apenas dois valores para variáveis deste tipo: True ou False.
Ex.: **Bool** value = True;

3.7. Caracter

A palavra reservada **Char** identifica as variáveis tipo caracter, com tamanho de 8 bits cada.
Ex: **Char** letter = ‘m’

3.8. Cadeira de caracteres

A palavra reservada **Str** identifica variáveis do tipo caracter com tamanho de 8 bits, combinado com o tamanho da cadeia que é dinâmica, com n caracteres respeitando o padrão ASCII. O tamanho mínimo da cadeia de caracteres é 0. É possível fazer declaração apenas da variável sem atribuição, assim como pode ser feita a atribuição direta na linha de declaração. A atribuição é feita por aspas simples (apóstrofos -> {‘’}), para delimitar o início e fim da cadeia.
Ex.:
Str word;
Str word = ‘Hello World’;
word = ‘Hello’;

3.9. Arranjo unidimensional (Vetor)

Os arranjos unidimensionais são definidos através da seguinte estrutura: <Tipo> Identificador [<Tamanho do Arranjo>]. Arranjos unidimensionais só podem armazenar variáveis do mesmo tipo.
Ex: **Int** codes[10];
Float prices[50];

3.10. Operações suportadas

A linguagem **Isengard++** suporta as seguintes operações:

Tipo	Operações
Int	Atribuição, aritméticos, relacionais, concatenação
Float	Atribuição, aritméticos, relacionais, concatenação
Bool	Atribuição, lógica, relacionais de igualdade e desigualdade, concatenação
Char	Atribuição, relacionais, concatenação
Str	Atribuição, relacionais, concatenação

3.11. Valores padrão

A linguagem possui valores por padrão para cada um dos tipos, aos quais as variáveis não atribuem valores em sua inicialização. Esses valores são:

Tipo	**Valor**
Int	0
Float	0.0
Bool	False
Char	Null
Str	Null

3.12. Coerção

A linguagem **Isengard++** não suporta coerção entre variáveis com tipos diferentes, ou seja, é estaticamente tipada. As verificações de compatibilidade por tipo são efetuadas de maneira estática.

4. Conjuntos de operações

4.1. Aritméticos

Operador	Operação
+	Soma de dois operandos
-	Subtração de dois operandos
*****	Multiplicação de dois operandos
/	Divisão de dois operandos
%	Resto da divisão entre operandos
-	Negação de variáveis do tipo **Int** e **Float**
&	Concatenação de duas cadeias **Str**

4.2. Relacionais

Operadores	Operação
==	Igualdade entre dois operandos
!=	Desigualdade entre dois operandos
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que

4.3. Lógicos

Operadores	Operação
!	Negação
And	Conjunção
Or	Disjunção

4.4. Concatenação de Cadeia de Caracteres

A concatenação na linguagem **Isengard++** é representada de maneira **explícita** pelo caractere '**&**'. Temos as seguintes possibilidades de concatenação:

- Char com Str: Resulta em Str;
- Int ou Float com Char ou Str: Resulta em Str;
- Bool com Char ou Str: Resulta em Str.

Logo, a concatenação suporta apenas o tipo de dados **Str,** possuindo associatividade da esquerda para a direita. Além de outros tipos na saída de dados.

4.5. Precedência e Associatividade

Precedência	Operadores	Associatividade
~	Menos unário	Direita → Esquerda
*** /**	Multiplicativos	Esquerda → Direita

+ -	Aditivos	Esquerda → Direita
%	Resto	Esquerda → Direita
!	Negação	Direita → Esquerda
< > <= >=	Comparativos	Sem associatividade
== !=	Igualdade	Esquerda -> Direita
And Or	Conjunção	Esquerda -> Direita

5. Instruções

As instruções de uma linha são encerradas obrigatoriamente com o símbolo “**;**”, enquanto os blocos (funções, *loops*, etc) são delimitados pelas palavras reservadas **Begin** e **End**.

5.1. Atribuição

A atribuição de uma variável em **Isengard++** é feita através do símbolo “ **=** ”, com o lado esquerdo se referindo ao identificador do tipo da variável, enquanto o lado direito se refere ao valor ou a expressão atribuída à mesma. Ambos os lados precisam ter o mesmo tipo, já que a linguagem não permite coerção. Visando confiabilidade, a linguagem trata atribuição como uma instrução e não uma operação.

5.2. Estruturas condicionais

O bloco da estrutura condicional **If** sempre terá uma condição, que é avaliada por uma expressão lógica que resultará em um **Bool**, seguido do bloco de instruções a serem executadas, demarcado pelas palavras **Begin** e **End**. O programa executa as instruções dentro do bloco associado ao **If**, se e somente se, a sua expressão lógica for verdade. Se o condicional for falso as instruções a serem executadas serão as que estiverem contidas no bloco **Else**, se ele existir.

```
If (logical expression) Begin
    # Instructions
End

If (logical expression) Begin
    # Instructions
End
Else Begin
    # Instructions
End
```

5.3. Estruturas interativas

5.3.1 While

A linguagem **Isengard++** utiliza a palavra reservada **While** para implementar uma estrutura de iteração com controle lógico. O laço de repetição permanecerá sendo executado enquanto a sua condição de controle lógico permanecer verdadeira.

```
While (logical condition) Begin
    # Instructions
End
```

5.3.2 For

Na estrutura que utiliza a palavra reservada **For**, o número de interações é definido pelo programador e o controle é realizado por um contador.

A estrutura interativa **For** deve possuir um valor inicial, um passo e um valor final (start,step,stop). O valor inicial é incrementado internamente pelo passo ao final de cada ciclo. O valor final deve ser sempre maior ou igual ao inicial para que o **For** seja executado. O número de interações é definido por: (stop - start) / step.

```
For (start, step, stop) Begin
    # Instructions
End
```

5.4. Entrada e saída

A linguagem **Isengard++** implementa as funções de entrada e saída através das palavras reservadas **Input** e **Output**, respectivamente.

5.4.1 Entrada

Na função de **Input** realizamos a atribuição de uma entrada fornecida pelo usuário a uma determinada variável de tipo correspondente.
Ex: **Int** number;
Input(number);

5.4.2 Saída

A função **Output** exibe na tela o valor correspondente ao algoritmo específico no programa. Além disso, pode exibir um **Str** sem ter sido armazenado numa variável antes, sendo necessário apenas colocar entre aspas simples o que for escrito na função **Output** e será exibido na tela. Caso deseje imprimir a saída com quebra de linha, basta utilizar a palavra reservada **OutputIn**.

Para exibir na tela duas **Str** na mesma linha basta separar por vírgula, em linhas distintas basta inserir um "**&**" entre as duas **Str** e para concatenar variáveis com Str basta seguir a seguinte estrutura:

```
Output(str1&'Message');
```

Para realizar uma impressão simples de um determinado tipo, basta seguir a seguinte estrutura:

```
Int value = 10;

Output(value);
```

5.5. Funções

As funções são declaradas com a palavra reservada **Funct**, seguida por seu identificador e os parâmetros dentro de parênteses (seus tipos e nomes devem ser especificados), separados por vírgulas. Para iniciar e finalizar a função, utilizamos as palavras **Begin** e **End**, respectivamente. A linguagem não admite sobrecarga de funções, ou seja, não é possível ter funções com o mesmo identificador.

O retorno de uma função é definido pela palavra reservada **Return** e seu valor. Se o valor de retorno de uma função não for especificado, ela irá retornar o valor padrão de seu tipo de retorno. Para chamar uma função, deve ser utilizado o seu identificador, e dentro dos parênteses, os valores que serão utilizados pela mesma.

```
Funct Float dividir(Float n1, Float n2) Begin
    Float div = n1/n2;
    Return div;
End
```

6. Exemplos de algoritmos

6.1. Hello World

```
Funct Int Main() Begin
        Output('Hello World');
        Return 0;
End
```

6.2. Série de Fibonacci

```
Funct Int fibonacci(Int n) Begin
        If(n < 2) Begin
            Return n;
        End
        Else Begin
            Return fibonacci(n - 1) + fibonacci(n - 2);
        End
End

Funct Int Main() Begin
        Int n;
        Output('Digite o limite: ');
        Input(n);

        For(Int i = 0, 1, n) Begin
            total = fibonacci(i);
            Imprimir(total);
        End

        Return;
End
```

6.3. *Shell* Sort

```
Funct Void shellsort(Int array[ ], Int n) Begin
        Int h = 1, c, j;

        While (h < n) Begin
            h = h * 3 + 1;
        End

        h = h / 3;

        While(h > 0) Begin
            For (Int i = h, 1, n) Begin
                    c = array[i];
                    j = i;
                    While(j >= h E array[j - h] > c) Begin
                            array[j] = array[j - h];
                            j = j - h;
                    End
                    array[j] = c;
            End
            h = h / 2;
        End
        Return;
    End
```

```
Funct Int Main ( ) Begin
        Int n, v;
        Output('Digite o tamanho do array: ');
        Input(n);
        Int array[n];

        Output('Digite os valores para serem ordenados: ');
        For (Int i = 0, 1, n) Begin
            Input(array[i]);
        End
        Output('Valores adicionados: ');
        For (Int i = 0, 1, n) Begin
            v = array[i];
            Outputln(v);
        End
        shellsort(array[n], n);

        Output('Valores ordenados: ');
        For (Int i = 0, 1, n) Begin
            v = array[i];
            Outputln(v);
        End

        Return;
End
```