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, 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 de uma função é dado por **Return** <identificador>, onde o valor retornado é sempre o definido pelo tipo da função. Caso o valor de retorno não seja especificado, o mesmo será dado pelo valor padrão do tipo da função;
- 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:

```
Funct Int Main() Begin

    Outputln(“Programa na Linguagem Isengard++”);

    Return;
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 bloco de código
End	Finalizar bloco de código
Main	Função Principal
Input	Instrução de Entrada
Output	Instrução de Saída
Outputln	Instrução de Saída com quebra de linha no fim
And	Operador lógico
Or	Operador lógico
Not	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 deve ser obrigatoriamente uma letra minúscula;
- Os caracteres seguintes do identificador podem ser quaisquer alfanuméricos;
- O tamanho máximo de um identificador é de 16 caracteres;
- 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 '**#**'.

Exemplo:

```
#Comentando na linguagem Isengard++
```

3.4. Inteiro

A palavra reservada **Int** identifica as variáveis com números do tipo inteiro tendo como tamanho 32 bits. Seus literais são expressos em uma sequência de dígitos inteiros.

Exemplo:

```
Int value = 12;
```

3.5. Ponto flutuante

A palavra reservada **Float** identifica as variáveis com números do tipo ponto flutuante, tendo como tamanho 64 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.

Exemplo:

```
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.

Exemplo:

```
Bool value = True;
```

3.7. Caracter

A palavra reservada **Char** identifica as variáveis tipo caracter, com tamanho de 8 bits cada. A atribuição é feita por apóstrofos -> {‘’}.

O tipo char possui como valor padrão o ‘\0’, que representa um caractere da tabela ASCII e do conjunto de caracteres Unicode cujo valor é 0 (a lista de valores padrões dos tipos está disponível na seção 3.11. Valores padrão).

Exemplo:

```
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. A atribuição é feita por aspas -> {“”}, para delimitar o início e fim da cadeia.

Exemplo:

```
Str word = "Hello World";
```

3.9. Arranjo unidimensional (Vetor)

Os arranjos unidimensionais são definidos através da seguinte estrutura: **Tipo** Identificador [**Tamanho do Arranjo**]. O tamanho mínimo do Arranjo é de 1 elemento.

Os elementos dos Arranjos unidimensionais são apenas do tipo que foi definido anteriormente. O valor padrão dos elementos é definido pelo valor padrão do tipo (a lista de valores padrão de cada tipo está presente na seção 3.11. Valores padrão).

Exemplos:

```
Int codes[10];
Float prices[50];
```

A indexação de elementos do Arranjo é feita de 0 a n - 1 (primeiro ao último elemento), sendo n o tamanho do Arranjo.

A atribuição de valores aos elementos pode ser feita de 2 formas:

- Atribuição de valores a todos os elementos, no momento da definição do arranjo. Para a atribuição dos múltiplos valores, os mesmos devem ser escritos dentro de colchetes, separados por vírgulas. Cada valor é atribuído a um elemento do array, seguindo a ordem crescente, do elemento 0 ao n - 1.

  Exemplo de atribuição de múltiplos elementos no momento da declaração:

```
Int array[2] = [1, 2];
```

- Atribuição de valor a um elemento específico no Arranjo, podendo ocorrer apenas após a definição do mesmo. Para a atribuição de valor nesse caso, é necessário que o identificador do Arranjo seja sucedido pelo índice do elemento específico, dentro de colchetes.

  Exemplo de atribuição de valor ao último elemento do Arranjo:

```
Int inventory[4];

inventory[3] = 5;
```

Na chamada de uma função, a passagem de um parâmetro Arranjo ocorre sempre por referência. A sintaxe da utilização de Arranjos como parâmetros de funções pode ser encontrada na seção 5.5 Funções.

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	‘\0’
Str	Null

3.12. Coerção

A linguagem **Isengard++** suporta apenas coerção explícita, através da concatenação, entre o tipo **Str** e os demais tipos, resultando no tipo **Str** (para mais

detalhes, consultar a seção 4.4 Concatenação de Cadeia de Caracteres). Os demais tipos de coerção não são permitidos.

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 variáveis

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**
Not	Negação

And	Conjunção
Or	Disjunção

4.4. Concatenação de Cadeia de Caracteres

A concatenação na linguagem **Isengard++** é representada de maneira **explícita,** através do caractere '**&**'. A concatenação utiliza a seguinte sintaxe:

<identificador de variável 1> & <identificador de variável 2>.

Uma das variáveis deve obrigatoriamente ser do tipo Str ou do tipo Char.

Temos as seguintes possibilidades de concatenação:

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

Exemplo:

```
Str string = "Hello";
Float pi = 3.14168;

string = string & pi;
```

4.5. Tamanho do arranjo unidimensional

O tamanho de um Arranjo Unidimensional pode ser atribuído a uma variável **Int** através do operador **@**. O lado esquerdo da atribuição possui a variável na qual o valor será atribuído, enquanto o lado direito possui o operador @ sucedido pelo identificador do Arranjo.

Exemplo:

```
Int arranjo[4];
Int size = @ arranjo;
```

4.6. Precedência e Associatividade

A seguir a listagem de precedência e associatividade dos operadores, com ordem de precedência decrescente.

Precedência	Operador(es)	Associatividade
@	Tamanho de Arranjo	Não associativo
~	Menos unário	Direita → Esquerda
Not	Negação	Direita → Esquerda
*** / %**	Multiplicativos	Esquerda → Direita
+ -	Aditivos	Esquerda → Direita
< > <= >=	Comparativos	Não associativo
== !=	Igualdade	Não associativo
And Or	Conjunção	Esquerda → Direita
&	Concatenação	Esquerda → Direita
=	Atribuição	Não associativo

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.

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.

Exemplo:

```
Int i;

If (i < 0) Begin
    i = ~i;
End
Else Begin
    i = i * 2;
End
```

5.3. Estruturas iterativas

5.3.1 Com controle lógico

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.

Exemplo:

```
Int i = 1;

While (i < 10) Begin
    Outputln(i);
    i = i + 1;
End
```

5.3.2 Controlada por contador

A linguagem também possui outra estrutura de iteração, que utiliza a palavra reservada **For**, a qual define um número fixo de iterações, através de um contador do tipo **Int**.

A estrutura é definida através de 3 elementos do tipo **Int** entre parênteses, logo após a palavra reservada **For.** Os elementos são, respectivamente, o contador (declarado antes ou dentro dos parênteses do **For)**, o valor limite e o valor do incremento do contador a cada iteração. Enquanto o contador for menor que o valor limite, a estrutura é executada.

A estrutura permite ainda a omissão do valor do incremento, recebendo dessa forma apenas o contador e o valor limite, respectivamente. Nesse caso, o incremento assume implicitamente o valor 1.

Exemplos:

```
For (Int i = 1, 10, 1) Begin
    Outputln(i);
End

For (Int i = 1, 10) Begin
    Outputln(i);
End
```

Obs: Os exemplos acima são equivalentes ao exemplo da estrutura de repetição While.

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

A função **Input** atribui entradas fornecidas pelo usuário aos parâmetros que foram passados na chamada da função. A função de entrada suporta a utilização de 1 ou mais parâmetros.

Exemplo:

```
Int number1, number2;
Input(number1, number2);
```

5.4.2 Saída

A função **Output** exibe na tela o valor correspondente aos parâmetros passados na sua chamada, sem quebra de linha ao final. Além disso, pode exibir um **Str** sem ter sido armazenado numa variável antes, sendo necessário apenas colocar a cadeia de caracteres entre aspas.

Para a saída com quebra de linha ao fim da impressão, basta utilizar a palavra reservada **Outputln,** a qual é similar à função .Output, apenas com o acréscimo da quebra de linha.

Exemplo:

```
Str str1 = "text ";
Output(str1, "message");
```

5.5. Funções

As funções são declaradas com a palavra reservada **Funct**, seguida por seu tipo, identificador e seus parâmetros formais definidos dentro de parênteses, separados entre si por vírgulas.

Caso a função não possua parâmetros, deve ser mantido o abre e fecha parênteses. Cada parâmetro deve ser definido com seu tipo e identificador. Parâmetros do tipo Arranjo devem ser definidos com tipo, identificador e abre e fecha colchetes.

O método de passagem de parâmetros é por Valor-Resultado nos tipos **Int**, **Float**, **Char**, **Str** e **Bool.** Em Arranjos Unidimensionais de qualquer tipo, a passagem de parâmetros é feita por referência. A tabela abaixo detalha melhor os métodos utilizados:

Tipo(s)	Nome do Método	Semântica	Transferência de Dados
Int, Float, Char, Str, Bool	Por Valor-Resultado	Entrada/Saída	Cópia
Arranjo Unidimensional	Por Referência	Entrada/Saída	Caminho de Acesso

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**, seguida de seu valor. O retorno de Arranjos Unidimensionais não é permitido.

Se o valor de retorno de uma função não for especificado, ela irá retornar o valor padrão do tipo da função. Caso o tipo da função seja **Void**, não haverá valor de retorno ao utilizar **Return,** porém a execução da função é interrompida.

Para chamar uma função, deve ser utilizado o seu identificador, e dentro dos parênteses, os parâmetros reais que serão utilizados pela mesma.

Exemplos:

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


Funct Void lerArray (Int array[]) Begin
    Int n = @ array;
    For (Int i = 0, n) Begin
        Input(array[i]);
    End
End

Funct Int Main () Begin
    Int array[5];

    lerArray(array);

    Return;
End
```

5.5.1 Função Principal

A função principal de um programa é declarada através da palavra reservada **Main,** sempre após a declaração das outras funções do programa**.** A função Main é do tipo Int e não suporta o recebimento de parâmetros.

Exemplo:

```
Funct Int Main () Begin
    Output("Main");

    Return;
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 Void fibonacci(Int n) Begin
    Int v1 = 0, v2 = 1, v3;

    If(n == 0) Begin
        Output(n);
        Return;
    End
    If(n == 1) Begin
        Output("0, ", n);
        Return;
    End
    Else Begin
        Output("0, 1, ");
        While(True) Begin
            v3 = v1 + v2;

            If(n < v3) Begin
                Return;
            End

            Output(", ", v3);
            v1 = v2;
            v2 = v3;
        End
    End
End

Funct Int Main() Begin
    Int n;
    Output("Digite o limite: ");
    Input(n);
    fibonacci(n);
    Return;
End
```

6.3. *Shell* Sort

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

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

     h = h / 3;

     While(h > 0) Begin
          For (Int i = h, n) Begin
            c = array[i];
            j = i;
            While(j >= h And 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, i;
     Output("Digite o tamanho do array: ");
     Input(n);
     Int array[n];

     Output("Digite os valores para serem ordenados: ");
    For (i = 0, n) Begin
        Input(array[i]);
    End
```

```
    Output("Valores adicionados: ");
    For (i = 0, n) Begin
        v = array[i];
     Output(v, " ");
    End

     shellsort(array);

     Output("Valores ordenados: ");
    For (i = 0, n) Begin
        v = array[i];
     Output(v, " ");
    End

     Return;
End
```