Universidade Federal de Alagoas - UFAL
Instituto de Computação - IC
Curso de Ciência da Computação

Professor Alcino Dall'Igna Júnior

Especificação da Linguagem

Ana Carolina Nesso Guedes
Eduardo Antônio de Lucena Lisboa
Guilherme Melo de Medeiros

Maceió, 16 de dezembro de 2021

Conteúdo

1 Introdução

A linguagem de programação Haine tem como base as linguagens C++ e Python. Não admite escopo global e não possui conversão implícita de tipos (coerção), é estaticamente tipada e a compatibilidade de tipos é realizada por nome. Não é orientada à objetos e é sensível ao caso (*case sensitive*).

Os arquivos escritos em Haine devem possuir a extensão ".hne".

2 Estrutura Geral

Um programa escrito em Haine deve possuir:

- A palavra reservada de início do programa (`begin`) seguido de chaves ({ }) e o restante do código deve ficar entre essas chaves.
  Exemplo:

```
begin {
    ...
    code
    ...
}
```

- Uma função deve apresentar seu tipo de retorno e um nome, este iniciado por letra maiúscula, e seus parâmetros dentro de parênteses e separados por vírgula, com cada parâmetro apresentando primeiro seu tipo e depois seu identificador. Por fim, colocam-se também chaves ({ }) e o código da função entre estas chaves, juntamente com a palavra reservada de retorno (`return`).
  Exemplo:

```
returnType FunctionName(typeA paramA, typeB paramB,...) {
    ...
    code
    ...
}
```

- Exemplo de código completo:

```
begin {
    int MyFunction(int var1, int var2) {
        return var1 + var2;
    }
}
```

3 Tipos de dados e Nomes

3.1 Variáveis globais/locais

As variáveis devem ser declaradas dentro das funções em que serão utilizadas.

3.2 Palavras reservadas

Lista de palavras reservadas: `int`, `float`, `char`, `string`, `bool`, `begin`, `end`, `if`, `else`, `while`, `from`, `to`, `increase`, `get`, `show`, `return`.

3.3 Identificadores

Não há limite de tamanho para os identificadores e todos devem seguir as seguintes regras:

- Todas as variáveis devem iniciar com letra minúscula e funções devem ser iniciadas com letra maiúscula;
- A partir do segundo caractere podem ser usadas letras maiúsculas ou minúsculas;
- Não é permitido o uso de espaços;
- Não é permitido o uso de palavras reservadas.

3.4 Declaração de variáveis

A declaração de variáveis deve apresentar o tipo da variável e seu nome (vide o item 3.3). Uma atribuição de valor imediata é opcional.

Exemplo:

```
<type> identifierA;
<type> identifierB = value;
```

Os tipos disponíveis para utilização são do item 3.5 ao item 3.10.

3.5 Inteiro

Representa um número inteiro de 64 bits, deve ser identificado pela palavra reservada `int` e seus literais são da forma `[0-9]+`

Exemplo:

```
int integer;
int myInt = 16;
int myOtherInt = 5500;
```

3.6 Ponto Flutuante

Representa um número real de 64 bits, deve ser identificado pela palavra reservada `float` e seus literais são da forma `[0-9]+\.[0-9]*`

Exemplo:

```
float floatingPoint;
float myFloat = 7.64;
```

3.7 Caracteres

Cada caractere representa um símbolo da tabela ASCII, estes sendo denotados por um valor entre 0 e 127, armazenados em 1 byte.

Deve ser identificado pela palavra reservada `char` e ao ser atribuído à uma variável, deve vir entre aspas simples (’). Para denotar as próprias aspas, deve-se usar o caractere de escape contrabarra (\), o mesmo vale para o próprio caractere de escape.

Exemplo:

```
char a = 'a';
char b = '\'';
```

3.8 Cadeia de Caracteres

Representam uma sequência de caracteres e deve ser identificado pela palavra reservada `string`.

Ao ser atribuída uma string à uma variável, deve-se utilizar aspas duplas (”). Para denotar as próprias aspas, a mesma regra do item 3.7 se aplica.

Exemplo:

```
string str;
string myString = "spam";
string otherString = "e \" ggs";
```

3.9 Arranjo Unidimensional

Define-se utilizando o tipo dos valores que estarão contidos no vetor juntamente com o identificador e logo após o tamanho, ou seja, a quantidade de elementos dentro do vetor, entre colchetes ([ ]).

Exemplo:

```
<type> indetifier[size];
<type> indetifier[size] = {<values>};
```

Para o segundo caso, onde os valores já são passados no momento da inicialização do vetor, o tamanho do mesmo é opcional.

Cada elemento do vetor pode ser acessado utilizando

```
identifier[index];
```

onde o `index` varia de `0` até `size - 1`.

Existem duas possibilidade para o caso de a quantidade de elementos em um vetor ser diferente da quantidade declarada:

1. Quantidade menor que a declarada:
   O vetor é preenchido com os valores padrão do tipo declarado.

2. Quantidade maior que a declarada:
   Erro.

3.10 Booleano

Representam dois valores apenas: verdadeiro e falso, estes denotados por `true` e `false`, e devem ser identificados utilizando a palavra reservada `bool`.

Exemplo:

```
bool boolean;
bool myBool = false;
```

3.11 Valores Padrão

Tipo	Valor
`int`	`0`
`float`	`0.0`
`char`	`''`
`string`	`''`
`bool`	`false`

3.12 Operações Aceitas por Tipo

Tipo	Operações
`int`	Atribuição, Aritmética, Relacional
`float`	Atribuição, Aritmética, Relacional
`char`	Atribuição, Relacional
`string`	Atribuição, Relacional
`bool`	Atribuição, Relacional*, Lógica

* somente operadores "==" e "! =" são aceitos.

4 Operadores

4.1 Aritméticos

Operação	Símbolo
Soma	$+$
Subtração	$-$
Multiplicação	$*$
Divisão	$/$

4.2 Relacionais

Operação	Símbolo
Igual	$==$
Diferente	$!=$
Maior que	$>$
Menor que	$<$
Maior ou igual	$>=$
Menor ou igual	$<=$

4.3 Lógicos

Operação	Símbolo
Conjunção	`and`
Disjunção	`or`

4.4 Precedência

Quanto menor o valor, maior sua precedência.

Operador	Precedência
$*$, $/$	1
$+$, $-$	2
$==$, $!=$	3
$>$, $<$, $>=$, $<=$	4
`and`	5
`or`	6

5 Instruções

5.1 Atribuição

Define-se pelo operador '=', onde do lado esquerdo fica o identificador que receberá o valor e do lado direito o valor a ser atribuído.

Exemplo:

```
int myInt = 519;
string myString = "spam and eggs";
```

5.2 Estrutura Condicional

Para uma estrutura simples, utiliza-se a palavra reservada `if` para identificar o inicio do bloco condicional, seguida pela expressão lógica que, ao ser verdadeira, entrará na condição, e então chaves ({ }), onde ficará o código a ser executado.

Já em uma estrutura composta, utiliza-se o mesmo da estrutura simples com a adição que, após o fecha chaves (}) que encerra o bloco do `if`, vem a palavra reservada `else` identificando o bloco que deverá ser executado caso a condição da expressão lógica inserida no primeiro bloco seja falsa. O bloco do `else` não possui expressão lógica associada diretamente à ele.

Exemplo:

- Simples:

```
if expressao_logica {
    ...
    code
    ...
}
```

- Composta:

```
if expressao_logica {
    ...
    code
    ...
} else {
    ...
    code
    ...
}
```

5.3 Estrutura Iterativa com Controle por Contador

Utiliza-se a palavra reservada `from` seguida de uma variável que será utilizada como contador já com seu valor inicial atribuído, seguido da palavra reservada `to` e o valor de destino do contador e por fim a palavra reservada `increase` e o valor que será incrementado no contador. Um bloco de chaves ({ }) identifica o código que deverá ser executado durante as iterações.

Exemplo:

```
int varInicio = valorInicial;
from varInicio to valorDestino increase valorIncremento {
    ...
    code
    ...
}
```

5.4 Estrutura Iterativa com Controle Lógico

Utiliza-se a palavra reservada `while` seguida de uma expressão lógica. Um bloco de chaves ({ }) identifica o código que deverá ser executado durante as iterações.

Exemplo:

```
while expressao_logica {
    ...
    code
    ...
}
```

5.5 Entrada e Saída

- Entrada:

  É feita através da instrução `get()` que recebe como argumento as variáveis que irão receber a(s) entrada(s). Os tipos das variáveis que receberão as entradas deve obrigatoriamente ser iguais aos tipos dos valores inseridos.

  Exemplo:

```
<type> identifier;
get(identifier);
```

- Saída:
  É feita através da instrução `show()` que recebe como argumentos:

  1. Uma cadeia de caracteres, que pode conter ou não instruções de formatação;
  2. As variáveis que deverão ser impressas, separadas por vírgula, caso haja instruções de formatação.

A formatação deve ser feita como segue:

Tipo	Formatação
`int`	%d
`float`	%*f
`char`	%c
`string`	%s
`bool`	%b

* Valor numérico opcional que define a quantidade de casa decimais impressas.

Exemplos:

```
show("Hello, World!");
show("%d", intVar);
show("Spam and %s", "eggs")
show("%3f", floatVar);
```

6 Programas Exemplo

6.1 Hello, World!

```
begin {
    show("Hello, World!");
}
```

6.2 Fibonacci

```
begin {
    int Fibonacci(int value) {
        int last = 1;
        int penultimate = 1;
        int next;

        if value == 1 {
            return 1;
        }
        if value == 2 {
            return 1;
```

```
        }

        int count = 3;
        while count <= value{
            next = last + penultimate;
            penultimate = last;
            last = next;
            count = count + 1;
        }

        return next;
    }

    int nthTerm;
    int result;

    get(nthTerm);

    result = Fibonacci(value);
    show("%d", result);
}
```

6.3 Shell Sort

```
begin {
    int ShellSort(int array[], int n) {
        int gap = n / 2;
        while gap > 0 {
            int i = gap;
            from i to (n - 1) increase 1 {
                int temp = array[i];
                int j = i;

                while j >= gap and array[j - gap] > temp {
                    array[j] = array[j - gap];
                    j = j - gap;
                }
                array[j] = temp;
            }
        }
        return 0;
    }

    int array[10];
    int i = 0;

    from i to 9 increase 1 {
        int input;
```

```
        get(input);
        array[i] = input;
    }

    ShellSort(array, 10);

    from i to 9 increase 1 {
        show("%d ", array[i]);
    }

}
```