

Universidade Federal de Alagoas

Instituto de Computação

Curso de Ciência da Computação

Alluph

Especificação da Linguagem

Phyllipe Matheus Bezerra Alves

Lucas Agra de Omena

Maceió

2019.1

Conteúdo

| | | |
|----------|---|-----------|
| 1 | Introdução | 3 |
| 2 | Estrutura geral do programa | 3 |
| 3 | Conjuntos de tipos de dados e nomes | 3 |
| 3.1 | Palavras reservadas | 3 |
| 3.2 | Identificador | 3 |
| 3.3 | Comentário | 4 |
| 3.4 | Inteiro | 4 |
| 3.5 | Ponto Flutuante | 4 |
| 3.6 | Caractere | 4 |
| 3.7 | Cadeia de caracteres | 5 |
| 3.8 | Booleano | 5 |
| 3.9 | Arranjos unidimensionais | 5 |
| 3.10 | Operações suportadas | 5 |
| 3.11 | Valores padrão | 6 |
| 3.12 | Coerção | 6 |
| 4 | Conjunto de operadores | 6 |
| 4.1 | Aritméticos | 6 |
| 4.2 | Relacionais | 6 |
| 4.3 | Lógicos | 7 |
| 4.4 | Concatenação de cadeias de caracteres | 8 |
| 4.5 | Precedência e Associatividade | 8 |
| 5 | Instruções | 8 |
| 5.1 | Atribuição | 8 |
| 5.2 | Estrutura condicional de uma e duas vias | 8 |
| 5.3 | Estrutura iterativa com controle lógico | 9 |
| 5.4 | Estrutura iterativa controlada por contador | 9 |
| 5.5 | Entrada e saída | 9 |
| 5.6 | Funções | 10 |
| 6 | Programas Exemplos | 10 |
| 6.1 | Hello World! | 10 |
| 6.2 | Série de Fibonacci | 10 |
| 6.3 | Shell Sort | 11 |
| 7 | Especificação da Linguagem de Programação | 13 |
| 8 | Especificação dos Tokens | 13 |
| 9 | Especificação das Expressões Regulares | 13 |
| 9.1 | Expressões Regulares Auxiliares | 13 |
| 9.2 | Lexemas | 13 |

1 Introdução

A linguagem de programação Alluph é uma linguagem criada com inspiração na linguagem de programação C, tendo como meta sua utilização na introdução e aprendizagem da programação, ou seja, Alluph foi criada e é indicada sua utilização no âmbito pedagógico e instrutivo da programação.

Em suma, Alluph não é orientada a objetos e é **case-sensitive**, (diferencia letras maiúsculas de minúsculas) e por se tratar de uma linguagem de programação de iniciação, suas funcionalidades são simplificadas. Partindo para legibilidade, Alluph não faz coerção, (não admite conversões implícitas de tipo) e possui palavras reservadas.

Sendo uma linguagem voltada para o ensino, as palavras reservadas foram escolhidas de uma maneira em que fique o mais claro possível a mensagem que está se tentando passar, todas as palavras reservadas são definidas em inglês.

Em soma, como Alluph é uma linguagem de programação estática, sendo assim não é possível o tratamento de erros para detecção de tipos, em consequência sua confiabilidade é afetada. Por último, tendo raízes em Python e C, Alluph se torna uma linguagem aparentemente familiar para quem já tem um pouco de experiência com suas raízes, e bastante intuitiva para quem está começando.

2 Estrutura geral do programa

Dentre as obrigatoriedades de Alluph, um programa escrito deve possuir:

- Um bloco de declaração de função, que por definição é sempre iniciado pela palavra reservada ***function***, seguido pelo tipo de seu retorno ***returnType*** e seu identificador, após isto, uma lista de parâmetros delimitada por abre e fecha parênteses (***paramList***), delimitada pelas palavras reservadas ***do*** e ***end***, note que um bloco é definido por ambos ***do*** e ***end***, logo, todo bloco deve conter uma abertura e um fechamento.
- Uma função principal chamada ***_main*** seguindo o modelo acima, com o tipo de retorno obrigatório sendo um número inteiro, ***int***.

3 Conjuntos de tipos de dados e nomes

Como já mencionado, Alluph é *case-sensitive* e a instrução de atribuir valor à uma variável, é feita de maneira separada da sua declaração.

3.1 Palavras reservadas

As palavras reservadas em Alluph são sempre escritas em inglês e são listadas a seguir: *function, void, int, float, char, string, bool, do, end, if, else, for, print, read, false, true, null, while, and, or, doing*.

3.2 Identificador

Os identificadores de Alluph devem seguir um modelo e um conjunto de regras básicas:

- Devem iniciar obrigatoriamente com uma letra, por exemplo, *integerResult* é um identificador válido. Porém, *1integerResult* é inválido.
- Os outros caracteres, exceto o primeiro, podem ser letras, números ou *underline*.
- A utilização de espaços em branco no identificador não é permitida.
- A utilização de palavras reservadas como um identificador não é permitida.

3.3 Comentário

Os comentários serão indicados pelo caractere '#'. Sendo assim, tudo que for escrito após o caractere de comentário, será totalmente ignorado na etapa de compilação. Alluph não contém bloco de comentários, apenas o comentário por linha.

3.4 Inteiro

A palavra reservada ***int*** identifica a variável como sendo do tipo inteiro, um número inteiro de 32 bits. Com seus literais expressos em uma sequência de números decimais.

Exemplo:

```
int inteiro;
```

3.5 Ponto Flutuante

A palavra reservada ***float*** identifica a variável como sendo do tipo ponto flutuante, um número real de 64 bits. Com seus literais expressos em uma sequência de números decimais, seguido de um ponto e os demais dígitos decimais.

Exemplo:

```
float real;
```

3.6 Caractere

A palavra reservada ***char*** identifica a variável como sendo do tipo caractere. Este tipo possui 1 byte que guarda um número de 0 a 127 referente a seu símbolo da tabela ASCII.

Exemplo:

```
char letra;
```

3.7 Cadeia de caracteres

A palavra reservada ***string*** identifica a variável como sendo do tipo cadeia de caracteres. Com seus literais expressos em um conjunto de caracteres de tamanho mínimo 0 e tamanho máximo ilimitado e delimitados por aspas duplas obrigatoriamente.

Exemplo:

```
string cadeia_de_caracteres;
```

3.8 Booleano

A palavra reservada ***bool*** identifica a variável como sendo do tipo booleana, com seus únicos dois valores possíveis, ***true*** e ***false***.

Exemplo:

```
bool booleano;
```

3.9 Arranjos unidimensionais

O vetor em Alluph, tem definição idêntica a da linguagem C, sendo dever do programador gerenciar o tamanho do vetor.

Seu formato segue a maneira:

```
<Tipo> Identificador[Tamanho]
```

Exemplos:

```
int numeros_inteiros[10];  
float numeros_reais[10];  
char caracteres[10];  
string palavras[10];
```

Ou também, o formato:

```
<Tipo>[Tamanho]
```

Como identificação do tipo de retorno de uma função somente, podendo o tamanho ser omitido.

3.10 Operações suportadas

As operações suportadas pelos tipos supracitados, são declaradas a seguir:

| Tipo | Operações Suportadas |
|--------|--|
| int | atribuição, aritméticos e relacionais |
| float | atribuição, aritméticos* e relacionais |
| char | atribuição, relacionais e concatenação |
| string | atribuição, relacionais e concatenação |
| bool | atribuição, relacionais** e lógicas |

(* menos resto da divisão de dois operandos)

(** somente os operadores de igualdade '==' e diferença '!=')

3.11 Valores padrão

Os valores padrão atribuídos a cada variável são:

| Tipo | Valor Padrão |
|--------|-----------------|
| int | 0 |
| float | 0.0 |
| char | ' ' (32) |
| string | null |
| bool | false |
| array | depende do tipo |

3.12 Coerção

A linguagem é estaticamente tipada, não aceitando coerção entre variáveis de tipos diferentes. Toda a verificação de compatibilidade de tipos será feita estaticamente, com isto, pretende-se aumentar a detecção de erros.

4 Conjunto de operadores

4.1 Aritméticos

Os operadores aritméticos são dados por:

- (unário negativo)
- * (multiplicação)
- / (divisão)
- % (resto)
- + (soma)
- (subtração)

4.2 Relacionais

Os operadores relacionais são dados por:

- == (igual)
- < (menor que)

```

> (maior que)
<= (menor ou igual que)
>= (maior ou igual que)
!= (diferente)

```

Ao serem aplicados à cadeia de caracteres ou caracteres, os operadores

```

== (igual)
!= (diferente)

```

primeiro verificam se as cadeias possuem o mesmo tamanho (1 caso seja um caractere), depois checa os caracteres um a um até achar um diferente ou o fim da cadeia.

Os demais,

```

< (menor que)
> (maior que)
<= (menor ou igual que)
>= (maior ou igual que)

```

comparam a ordem lexicográfica dos caracteres um a um.

Ao serem aplicados ao tipo bool, os operadores

```

== (igual)
!= (diferente)

```

retornam da seguinte forma:

| L1 | L2 | == | != |
|-------|-------|-------|-------|
| true | true | true | false |
| true | false | false | true |
| false | true | false | true |
| false | false | true | false |

4.3 Lógicos

Os operadores lógicos são dados por:

```

!    (negação unária)
and  (conjunção)
or   (disjunção)

```

| | L1 | L2 | L1 and L2 | L1 or L2 | | L1 | !L1 |
|-------|-------|-------|-----------|----------|------|-------|-------|
| onde: | true | true | true | true | , e: | true | false |
| | true | false | false | true | | false | true |
| | false | true | false | true | | | |
| | false | false | false | false | | | |

4.4 Concatenação de cadeias de caracteres

O operador de concatenação é dado por `++`. Ao ser aplicado em tipos de caractere, `bool`, ou numérico, transforma-os em string e os concatena.

4.5 Precedência e Associatividade

| Precedência | Associatividade | Operadores |
|-------------|-----------------|----------------------------|
| 1 | à direita | - (aritmético) (unário) |
| 2 | à esquerda | *, /, % (aritméticos) |
| 3 | à esquerda | +, - (aritméticos) |
| 4 | à esquerda | <, >, <=, >= (relacionais) |
| 6 | à esquerda | ==, != (relacionais) |
| 7 | à direita | ! (lógico) (unário) |
| 8 | à esquerda | and (lógico) |
| 9 | à esquerda | or (lógico) |
| 10 | à esquerda | ++ (concatenação) |

5 Instruções

As instruções da linguagem seguem o modelo de sequencia padrão imperativo, de cima para baixo, possuindo desvios condicionais, estruturas iterativas e subprogramas.

5.1 Atribuição

A atribuição é definida pelo operador `'='`, sendo o lado esquerdo o identificador e o lado direito é o valor ou expressão a ser atribuído.

Lembre-se também, que mesmo utilizando o operador de igualdade, a atribuição é na verdade uma instrução e não uma operação.

5.2 Estrutura condicional de uma e duas vias

As estruturas condicionais são definidas da seguinte forma:

```
if expressao_logica do
    < ...
    codigo
    ...>
end
else do
    < ...
    codigo
    ...>
end
```


5.3 Estrutura iterativa com controle lógico

Tem-se a estrutura *while*:

```
while expressao_logica do
  < ...
  codigo
  ...>
end
```

5.4 Estrutura iterativa controlada por contador

Tem-se a estrutura *from-to-doing*:

```
from id = expressao_aritmetica to expressao_aritmetica doing
  <incremento> do
    < ...
    codigo
    ...>
end
```

De modo opcional, tem-se a estrutura *from-to*, simplificando o incremento utilizando $id=id+1$ como padrão.

```
from id = expressao_aritmetica to expressao_aritmetica do
  < ...
  codigo
  ...>
end
```

Onde em ambos os casos tem-se *from x to y* tal que x está incluso e y não ($[x,y[$).
Ex:

```
from i = 0 to 10 do
  print(i);
end
```

O exemplo acima mostra na tela os números de 0 à 9.

5.5 Entrada e saída

A entrada é dada pela instrução

```
read(var)
```

A saída é especificada pela instrução

```
print("Print_aqui!")
```

que pode receber strings concatenadas ou variáveis.

5.6 Funções

As funções devem ser declaradas antes de usadas, todos os parâmetros devem ser especificados com seus respectivos tipos, e todos são passados por cópia. O return pode estar em qualquer parte da lista de sentenças ou até mesmo omitido em alguns casos. Todos os identificadores de funções **devem começar** com um caractere *underline*.

Segue a seguinte sintaxe:

```
function return_type name(type1 param1, type2 param2, ...) do
    <...
    codigo
    ...>

    return something;
end
```

6 Programas Exemplos

6.1 Hello World!

```
function int main() do

    print("Hello World");
    return 0;

end
```

6.2 Série de Fibonacci

```
function int fibonacci (int target) do

    int a;
```

```

    int b;
    int next;

    a = 1;
    b = 1;

    while next <= target do

        next = a + b;
        a = b;

        print(a ++ ", ");

        b = next;

    end

    print(a);

    return 1;
end

function int _main() do

    int number;
    int result;

    read(number);

    result = _fibonacci(number);

    return 0;

end

```

6.3 Shell Sort

```

function int[] _shellSort(int nums[], int size) do
    int i;
    int j;
    int value;
    int gap;

    gap = 1;
    while gap < size do
        gap = 3*gap+1;
    end

```

```

while gap > 0 do

    from i = gap to size do

        value = nums[i];
        j = i;

        while j > gap-1 and value <= nums[j-gap] do

            nums[j] = nums[j-gap];
            j = j-gap;

        end

        nums[j] = value;

    end

    gap = gap/3;

end

return nums;
end

function int _main() do

    int size;
    int i;

    print("Insira a quantidade de números");
    read(size);

    int numbers[size];

    print("Insira os numeros");

    from i = 0 to size do
        read(numbers[i]);
    end

    from i = 0 to size do
        print(numbers[i]);
    end

    numbers = _shellSort(numbers, size);

    from i = 0 to size do
        print(numbers[i]);
    end
end

```

```
        return 0;
    end
```

7 Especificação da Linguagem de Programação

Os analisadores léxico e sintático da linguagem Alluph serão implementados na linguagem C, utilizando o analisador preditivo tabular.

8 Especificação dos Tokens

A lista de tokens é definida por:

```
typedef enum category {

    catFunction, catDo, catEnd, catMain, catInt, catFloat,
    catString, catBool, catChar, catVoid, catOpPar, catClsPar,
    catComma, catIf, catWhile, catFrom, catTo, catElse,
    catDoing, catSemiCol, catOpBrac, catClsBrac, catPrint,
    catRead, catReturn, catOpeConc, catOpeOr, catOpeAnd,
    catOpeNeg, catOpeAtr, catOpeEq, catOpeDif, catCteBool,
    catOpeGt, catOpeGte, catOpeLt, catOpeLte, catOpeSum,
    catOpeSub, catOpeMult, catOpeDiv, catOpeMod, catFunId,
    catId, catCteInt, catCteFloat, catCteStr, catCteChar,
    catEOF

} Category;
```

9 Especificação das Expressões Regulares

As expressões regulares seguirão o padrão Flex.

9.1 Expressões Regulares Auxiliares

```
letter = '[a-zA-Z] '
digit = '[0-9] '
symbol = '[ .,:;?!+-*\/_%&#$<>=() [] {} | " ' ] '
```

9.2 Lexemas

Main:

```
main = '_main'
```

Identificador:

```
id = '(letter)(letter|digit|_)*'
funId = '(_letter)(letter|digit|_)*'
```

Tipos primitivos:

```
tInt = 'int'
tFloat = 'float'
tChar = 'char'
tString = 'string'
tBool = 'bool'
```

Delimitadores:

```
Escopo:
    escBegin = 'do'
    escEnd = 'end'
```

```
Parâmetros:
    paramBegin = '\('
    paramEnd = '\).'
```

```
Array:
    arrBegin = '\['
    arrEnd = '\]'
```

```
Finalizador:
    endSent = ';'.
```

```
Separador:
    sep = ','.
```

Definições de tipos:

```
constNumInt = '-?digit+'
constNumFloat = '-?digit+(\.digit+)?'
constBool = '(true|false)'
constChar = ''(letter|digit|symbolAspasSimples)''
constString = '"(letter|digit|symbolAspasDuplas)"*'
```

Palavras reservadas de fluxo:

```
prIf = 'if'
prElse = 'else'
prFrom = 'from'
prTo = 'to'
prDoing = 'doing'
prWhile = 'while'
prFunction = 'function'
```

```

prReturn = 'return'

Operadores lógicos:

opLogicAnd = 'and'
opLogicOr = 'or'
opLogicNot = '!'

Operadores aritméticos:

opAritAdd = '(\+|-)'
opAritMult = '(\*|\/|%)'
opAritNeg = '-'

Operadores relacionais:

opRel1 = '(==|!=)'
opRel2 = '(<|>)=?'

Operador de concatenação:

opConc = '\+\+'

Fim de arquivo

catEOF = 'eof'

```