UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO

UNIVERSIDADE FEDERAL
DE ALAGOAS

Linguagem BFS - Especificação

Bruna Leal Torres Silva
Eirene de Oliveira Fireman
Samuel Lucas Vieira Lins Barbosa

Maceió - AL
20 de Dezembro de 2021

Sumário

1. Introdução

A linguagem de programação BFS é uma linguagem de simples compreensão voltada para o aprendizado de desenvolvedores iniciantes. BFS traz à memória o famoso algoritmo de busca em largura, utilizado em Teoria dos Grafos. Entretanto, BFS vai além. Seu nome é um acrônimo para Bruna, Fireman e Samuel, três graduandos em Ciência da Computação que se reuniram para criar uma linguagem que pudesse ser utilizada para o ensino de lógica de programação. BFS é uma passagem de conhecimento e uma porta de entrada para a área de desenvolvimento de software, ela parte do simples para aguçar a vontade de seus usuários a aprenderem mais e mais até superarem algoritmos como “Breadth-First Search”.

BFS é uma linguagem estaticamente tipada de paradigma imperativo. Ela não permite coerção, nem a declaração de funções dentro de outras funções. É uma linguagem de boa escritabilidade e possui palavras reservadas comuns às linguagens de programação mais populares para facilitar o aprendizado. A criação de um bloco de código depende de sua função principal: *function void main() {}.*

2. Estrutura Geral do Programa

A estrutura básica de um programa em BFS segue os seguintes princípios:

- **DEPENDÊNCIA DA FUNÇÃO PRINCIPAL:** Todo programa em BFS deve ter como ponto de partida sua função principal (*function void main() {}*). Todo bloco de código, exceto o corpo de funções, deve ser escrito dentro das chaves de main. Ela deve obrigatoriamente ser precedida da palavra function e do tipo *void* e seguida de parênteses, sem parâmetro.
- **DECLARAÇÃO DE VARIÁVEIS:** As variáveis ou constantes em BFS devem ser tipadas (int, float, char, str, bool) e sempre precedidas de *underscore*(_).
- **DECLARAÇÃO DE FUNÇÕES:** As declarações de funções devem seguir o seguinte padrão:
    - FUNCTION: toda declaração de função conter primeiramente a palavra reservada *function*.
    - <TIPO DE RETORNO>: após function, deverá ser escrito o tipo de retorno que a função irá devolver (ex: int, float, void, str…)
    - <NOME DA FUNÇÃO>: em seguida, o usuário pode definir o nome desejado para a função, respeitando os critérios de incluir *underscore*(_) obrigatoriamente como o primeiro caractere.
    - (<PARÂMETROS>): depois do nome, deverá ser aberto parênteses, e, dentro deles poderá conter, ou não, parâmetros para a execução da função. Caso existam, estes parâmetros devem ser tipados e separados por vírgula, respeitando a indicação de variáveis em BFS.

    - {<BLOCO DE CÓDIGO>}: por último, a execução da função deverá ser escrita dentro do bloco de código entre as chaves da função.
- **CHAMADA DE VARIÁVEIS E FUNÇÕES:** Sempre devem ser precedidas por *underscore*(_).
- **PONTO E VÍRGULA:** Toda instrução deve acabar com um ponto e vírgula, a não ser que ela seja um escopo condicional, funcional ou de repetição (if, for, while, function, etc).

Exemplo:

```
function int _fibonacci(int _n) {
    if( _n < 2 ) {
        return _n;
    }

    return _fibonacci(_n - 1) + _fibonacci(_n - 2);
}

function void main() {

    int _n, _total;
    SysOut("Digite o tamanho da sequencia:");
    SysIn(_n);

    int _i;
    for(_i = 0, _i < _n) {
        _total = _fibonacci(_i);
        SysOut(_total);
    }
}
```

3. Conjuntos de Tipo de Dados e Nomes

3.1. Palavras Reservadas

function
return
void
int
float
char
string
bool
array
true
false
SysIn
SysOut
if
elif
else
while
for
and
or

3.2. Identificador

Os identificadores são utilizados para dar nomes a variáveis, constantes, tipos e funções. Na linguagem em questão, os identificadores são iniciados sempre por um *underscore*(_) seguido de um caractere maiúsculo ou minúsculo. Após o caractere, pode-se então, utilizar números ou letras maiúsculas e minúsculas.

3.3. Comentário

BFS só permite comentários de uma linha, que são definidos com '**#**' no início da linha.

3.4. Inteiro

A palavra reservada **int** identifica as variáveis que vão conter um inteiro ou um ponto flutuante (tamanho limitado a 4 bytes). Em casos

em que se faz referência a números inteiros, será representada por uma sequência de dígitos.

3.5. Ponto Flutuante

A palavra reservada **float** identifica as variáveis que vão conter um inteiro ou um ponto flutuante (tamanho limitado a 4 bytes). Já em casos em que se faz referência a pontos flutuantes, teremos um número inteiro, seguido por um ponto e uma sequência de dígitos.

3.6. Caracteres

A palavra reservada **char** identifica as variáveis que vão conter um caractere (tamanho limitado a 1 byte). A constante literal de um char é representada por um caractere ASCII e deve estar contido entre ‘’ (apóstrofos) .

3.7. Cadeia de caracteres

A palavra reservada **string** identifica as variáveis que vão conter uma cadeia de caracteres (tamanho dinâmico). A constante literal de uma string é representada por uma cadeia de caracteres ASCII e deve estar contido entre “” (aspas).

3.8. Booleanos

A palavra reservada **bool** identifica as variáveis que vão conter um booleano (tamanho limitado a 1 byte). A constante literal de um bool pode assumir dois valores: **true** ou **false**.

3.9. Arranjo Unidimensional

Um arranjo unidimensional é definido ao adicionar colchetes ao tipo da variável (tamanho é dinâmico). Importante ressaltar que arranjos unidimensionais só podem armazenar variáveis do mesmo tipo.

3.10. Operações suportadas

As operações suportadas por cada tipo em BFS são detalhadas abaixo.

Tipo	Operador(es)
int	atribuição, aritméticos, relacionais e concatenação com string ou

	char
float	atribuição, aritméticos, relacionais e concatenação com string ou char
string	atribuição, relacionais e concatenação
char	atribuição, relacionais e concatenação
bool	atribuição, igualdade, desigualdade, lógicos e concatenação com string ou char

3.11. Valores default

Tipo	Valores Padrão
int	0
float	0.0
string	null
char	null
bool	false

3.12. Coerção

A linguagem BFS não suporta coerção de tipos, ou seja, não será possível alterar uma entidade de um tipo de dado em outro.

4. Conjuntos de Operações

A linguagem de programação BFS possui os seguintes tipos de operadores:

- **Operadores Aritméticos**
- **Operadores de Atribuição**
- **Operadores Lógicos**
- **Operadores relacionais**

Os operadores na linguagem de programação BFS podem ser binários ou unários.

Um operador binário necessita de no mínimo dois operandos e um operador entre eles, seguindo a seguinte sintaxe:

$< operando > \quad < operador > \quad < operando >$

Exemplos: a + b ou 4 - 2

Já os operadores unários necessitam apenas de um operando e podem vir antes:

$< operador > \quad < operando >$

Exemplo: !d;

4.1. Operadores Aritméticos

Operadores	Operação
+	Adição
-	Subtração
*****	Multiplicação
/	Divisão
%	Resto da divisão
&	Concatenação

4.2. Operador de atribuição

Operador	Operação
=	Atribuição

4.3. Operadores Lógicos

Operadores	Operação
and	E lógico

or	Ou lógico
!	Negação lógica

4.4. Operadores Relacionais

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

4.5. Concatenação de Cadeias de Caracteres

A concatenação de cadeias de caracteres na linguagem BFS é feita utilizando o símbolo **“&”**. Que suporta variáveis do tipo **string** e **char** ou variáveis do tipo **int**, **float**, ou **bool** acompanhadas de uma **string** ou **char**.

4.6. Ordem de Precedência e Associatividade

Precedência	Operadores	Associatividade
*** /**	Multiplicativos	Esquerda para Direita
%	Resto	Esquerda para Direita
+ -	Aditivos	Esquerda para Direita
!	Negação	Direita para Esquerda
&	Concatenação	Esquerda para

		Direita
< > <= >=	Comparativos	Sem associatividade
== !=	Igualdade	Esquerda para Direita
and or	Conjunção	Esquerda para Direita

5. Instruções

Na linguagem BFS as instruções de uma linha são encerradas exclusivamente pelo símbolo ‘;’. Já os blocos são iniciados após a abertura de chaves ‘{‘ e encerradas quando fechadas as mesmas ‘}’.

5.1 Atribuição

Atribuições em BFS são feitas através do símbolo “ = ”, sendo o lado esquerdo se refere ao identificador do tipo da variável, enquanto o lado direito se refere ao valor ou a expressão atribuída a mesma. Vale ressaltar que os dois lados da igualdade tem que ser do mesmo tipo, porque a linguagem não permite coerção.

Ex.:

```
int _a;
_a = 5;

ou

int _a = 5;
```

5.2 Estrutura condicional de uma ou duas vias.

5.2.1 If, elif, else

O bloco da estrutura condicional **if** sempre terá uma condição, que é avaliada por uma expressão lógica ou tão somente uma variável do tipo **Booleano**, seguido do bloco de instruções a serem executadas.
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 **elif.** Caso a condicional do **elif** não seja verdadeira, temos o último bloco **else** que não possui condicional e é executado em último caso.

Ex.:

```
if(<expressão lógica>){
        <instruções>
}
elif(<expressão lógica>){
        <instruções>
}
else {
        <instruções>
}
```

5.3 Estruturas Interativas.

5.3.1 Controle Lógico - Enquanto.

A estrutura de interação com controle lógico da linguagem BFS implementa a estrutura **while**. A repetição na estrutura ocorre enquanto a condição de controle for verdadeira. O laço de repetição é executado enquanto a condição de controle lógica da estrutura for verdade. O laço encerra a partir do momento que a condição se torna falsa. Condições neste bloco são em sua maioria do tipo lógicas ou com variáveis booleanas.

Ex.:

```
while(<condição lógica>) {
        < instruções do bloco while>
}
```

5.3.2 Controle por Contador - for.

Já na estrutura **for**, o número de interações é definido pelo programador e seu controle é feito por um contador, fazendo contraponto ao while descrito no 5.3.2. Em uma estrutura interativa controlada por um contador, tem que possuir um valor inicial, e um valor final (start, stop). O valor inicial é incrementado sempre em um 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:

Nº de interações = (valor final - valor inicial)/step

5.4 Entrada e Saída.

BFS implementa entrada e saída mediante o uso de palavras reservadas: “SysIn” e “SysOut”.

5.4.1 Entrada.

Na função de input temos a atribuição de uma entrada fornecida pelo usuário a uma determinada variável de tipo correspondente, através do método “SysIn”.
Ex.:
SysOut("Digite o tamanho da sequencia:");

5.4.2 Saída.

Irá mostrar na tela o(s) valor(es) correspondente(s) aos parâmetros especificados pelo usuário. Além disso, “SysOut”, pode mostrar na tela uma String sem ter sido necessariamente declarada antes, sendo preciso
apenas colocar entre aspas duplas o que for escrito na função “SysOut” e será impresso na tela. Ou seja, podemos mostrar na tela textos escritos que não foram armazenados em variáveis. Todas as declarações dentro de um só “SysOut” serão mostradas na mesma linha e em “SysOut”s separados terá a quebra de linha.
Ex.:
SysOut("Alô Mundo.");

5.5 Funções.

O início de funções na linguagem BFS ocorre a declaração da palavra reservada **function**, para dizer ao compilador que ali começa uma função, em seguida o tipo de retorno da função, podendo ser int, void, float, char, string ou bool. Em seguida é necessário definir um identificador para a função, onde obrigatoriamente tem que se iniciar com um **‘_’**. Em seguida, abertura de parênteses, parâmetros (ou não) e fechamento de parênteses. Por fim, abertura de chaves para iniciar a escrita do bloco de instruções a serem executadas e ao fim fechamento de chaves para
informar o encerramento do bloco.
A linguagem BFS não aceita sobrecarga de funções, ou seja, não é definida outra função com o mesmo identificador. Antes do fechamento de chaves é obrigatório ter o return palavra reservada essa que efetua o retorno da função, caso não seja atribuído valor a ele, o valor default devolvido por ele é o valor padrão em cada tipo.

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

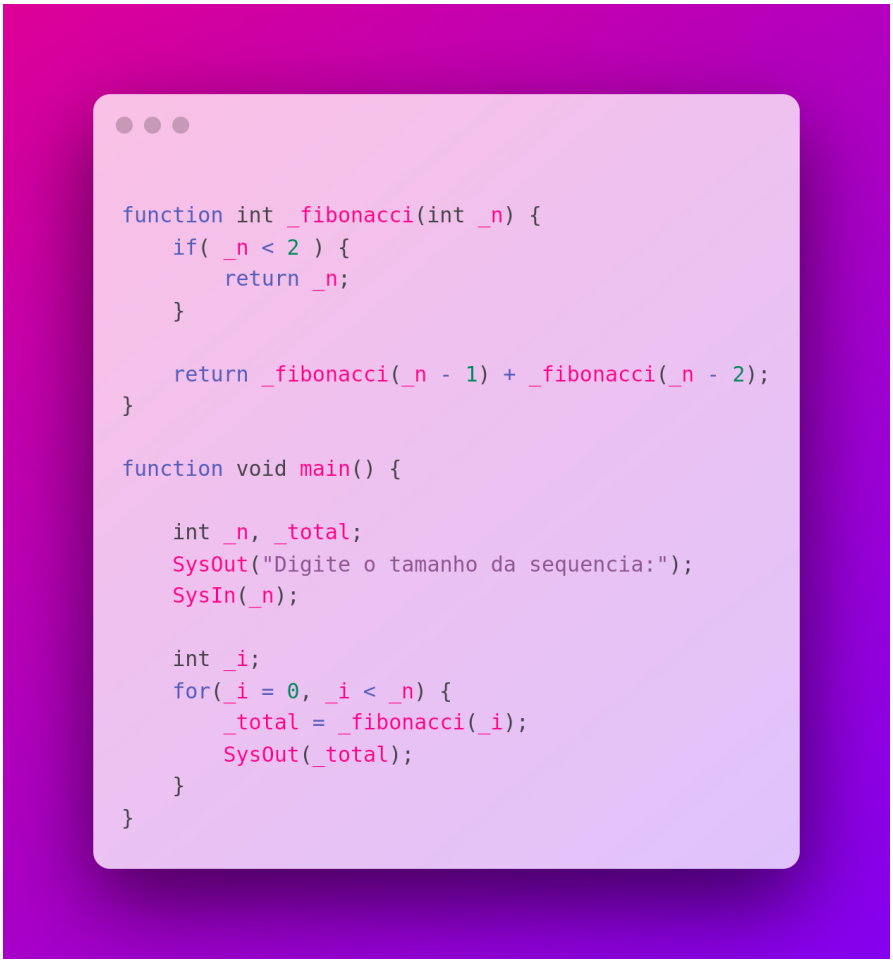
Ex.:

```
function <Tipo (int,float...etc> _<nome> (<Tipo> <identificador>) {
        <declarações de Tipos e variáveis>;
        <instruções>;
        return <valor a ser retornado>;
}
```

6. Exemplos de Algoritmos

6.1 Alô Mundo

6.2 Fibonacci



```
function int _fibonacci(int _n) {
    if( _n < 2 ) {
        return _n;
    }

    return _fibonacci(_n - 1) + _fibonacci(_n - 2);
}

function void main() {

    int _n, _total;
    SysOut("Digite o tamanho da sequencia:");
    SysIn(_n);

    int _i;
    for(_i = 0, _i < _n) {
        _total = _fibonacci(_i);
        SysOut(_total);
    }
}
```

6.3 Shell Sort

```
function void _shellSort(array int _nums, int _numsSize) {
    int _h = 1;

    while (_h < _numsSize) {
        _h = _h * 3 + 1;
    }

    _h = _h / 3;
    int _c, _j, _i;

    while (_h > 0) {
        for(_i = _h, _i < _numsSize) {
            _c = _nums[_i];
            _j = _i;
            while (_j >= _h and _nums[_j - _h] > _c) {
                _nums[_j] = _nums[_j - _h];
                _j = _j - _h;
            }
            _nums[_j] = _c;
        }
        _h = _h / 2;
    }
}

function void main() {
    int _n;

    SysOut("Insira o tamanho do array:");
    SysIn(_n);

    array int _valores[_n];
    SysOut("Insira os valores do array:");

    int _i;
    for(_i = 0, _i < _n) {
        SysIn(_valores[_i]);
    }

    SysOut("Array antes de organizar:");
    for (_i = 0, _i < _n) {
        SysOut(_valores[_i]);
    }

    _shellSort(_valores, _n);

    SysOut("Array depois de organizar:");
    for (_i = 0, _i < _n) {
        SysOut(_valores[_i]);
    }
}
```