Sumário

Estrutura geral do programa **2**

- Escopo Global 2
- Ponto de início de execução 2
- Definições de procedimentos/funções 2
- Representação de um procedimento/função 2
- Definição de variáveis globais e locais 3
- Definições de instruções 3

Nomes **3**

Tipos e Estrutura de Dados **4**

Inicialização dos Tipos **5**

Constantes Literais e Suas Expressões **5**

Declaração de Variáveis **5**

Operações **6**

Atribuição e Coerções **7**

Arranjos **7**

Estruturas de Controle **7**

Comando de Iteração **8**

Desvios Incondicionais **9**

Entrada e Saída **9**

Códigos Exemplo **10**

Estrutura geral do programa

- ### Escopo Global

  A linguagem admite escopo global, ou seja, uma variável declarada globalmente poderá ser acessada em qualquer conjunto de instruções.

- ### Ponto de início de execução

  O ponto inicial de execução do programa será identificado por uma construção específica da linguagem, como mostrada abaixo:

```
fun int main() {
        .
        .
        .
        return 0;
}
```

- ### Definições de procedimentos/funções

  Estes não poderão existir se forem declaradas dentro do corpo de uma outra função ou procedimento.

- ### Representação de um procedimento/função

```
fun ‘tipoDoRetorno’ ‘id’ ‘(‘ ‘listaDeParâmetros’ ‘)’ ‘{‘
        .
        .
        .
return;
```

}’

Onde, ‘funDef’ será o token que representa o início da escrita da mesma, ‘tipoDoRetorno’ é o token que representa o tipo de variável que a função irá retornar ao final de sua execução, ‘id’ é o identificador único da função, ‘listaDeParâmetros’ será a lista que irá conter todas as variáveis que estarão contidas no escopo desse procedimento e seus tipos respectivos . O retorno terá de ser do mesmo tipo identificado em ‘tipoDoRetorno’. Para procedimentos, o ‘tipoDoRetorno’ será ‘void’ e não necessitará de um retorno de função. Softy não aceita passagem de subprogramas como parâmetros. A passagem de parâmetros se dá pelo modo de entrada, somente.

- **Definição de variáveis globais e locais**

As variáveis podem ser definidas em qualquer lugar, possuem uma forma específica de serem declaradas, conforme demonstrado a seguir: ‘tipoDaVariável’ ‘id’. Onde ‘tipoDaVariável’ poderá ser ‘int’, ‘float’, ‘void’, ‘char’, ‘bool’, ‘string’ e seguido pelo ‘id’ como identificador único da variável.

Ex:

```
int a, b;
float c, d;
char e, f;
bool g, h;
string i, j;
```

- **Definições de instruções**

As instruções podem ser declaradas somente dentro do escopo de procedimentos, não podendo ser escritas instruções dentro do corpo dos parâmetros de uma função, tal como estruturas de iteração e estruturas de seleção.

Nomes

Possuirão sensibilidade à caixa, limite de tamanho de 32 caracteres e seu formato será definido a partir da seguinte expressão regular:

*(‘letter’ | ‘_’)(‘digit’ | ‘letter’ | ‘_’)**

Exemplos de nomes aceitos pela linguagem: _var1, var2, var, _, _a;
Exemplos de nomes não aceitos pela linguagem: .abc, ab@c, 1ac.

Obs.: A linguagem admite constantes com nome.
Ex.: const int a = 5;

Tipos e Estrutura de Dados

Os tipos e estruturas de dados são denotados pelas seguintes construções:

1. Tipo inteiro: determinado pelo token ‘typeInt’ e lexema ‘int’;
2. Tipo void: determinado pelo token ‘typeVoid’ e lexema ‘void’;
3. Tipo float: determinado pelo token ‘typeFloat’ e lexema ‘float’;
4. Tipo char: determinado pelo token ‘typeChar’ e lexema ‘char’;
5. Tipo bool: determinado pelo token ‘typeBool’ e lexema ‘bool’;
6. Tipo string: determinado pelo token ‘typeString’ e lexema ‘string’;
7. Arranjos são compostos pelos tipos determinados acima e possuem as seguintes declarações: ‘tipoDeDado’ ‘id’ ‘(‘ ‘constInt’ ‘)’, o único dado que não poderá representar um arranjo será o tipo void.

Todos os tipos e estruturas possuirão compatibilidade por nome. Sendo os tipos primitivos descritos a seguir: valores inteiros (‘int’), valores float (‘float’), valores de caracteres (‘char’), valores booleanos (‘bool’) e cadeias de caracteres (‘string’).

Exemplos dos arranjos:

int arr(9);
float arr(129);
bool arr(1);

Inicialização dos Tipos

Tipo	Valor padrão de inicialização
int	0
float	0.0
char	‘’ (caracter vazio)
string	“” (string vazia)
bool	false
void	Não pode ser inicializada

Constantes Literais e Suas Expressões

As expressões regulares das constantes literais são denotadas da seguinte maneira:

1. Constantes de inteiros: *((‘digit’)+)*;
2. Constantes de floats: *((‘digit’)+)(‘.’)((‘digit’)+)*;
3. Constantes de char: *(‘’’)(‘letter’ | ‘symbol’ | ‘digit’)(‘’’)*;
4. Constantes de bool: *(‘true’ | ‘false’)*;
5. Constantes de string: *(‘“’)((‘letter’| ‘symbol’ | ‘digit’)*)(‘”’)*;

Declaração de Variáveis

As declarações de variáveis são declarações de tipo separadas por espaços seguidas do nome da variável (identificador). Para os arranjos, a declaração será a mesma, porém após o identificador, deverá seguir ‘(‘ ‘constInt’ ‘)’.

Operações

Operadores (Precedência)	Associatividade
‘!’ (Not)	Direita para esquerda
‘-’ (Unário Negativo)	Direita para esquerda
‘^’ (Exponenciação)	Direita para esquerda
‘*’ (Multiplicação) e ‘/’ (Divisão)	Esquerda para direita
‘+’ (Soma) e ‘-’ (Subtração)	Esquerda para direita
‘<’, ‘<=’, ‘>’ e ‘>=’	Não associativo
‘==’ e ‘!=’	Não associativo
‘&&’	Esquerda para direita
‘||’	Esquerda para direita
‘=’ (Atribuição) e ‘::’ (Concatenação)	Direita para Esquerda ‘=’ / Esquerda para Direita ‘::’

Vale ressaltar que os parênteses podem ser utilizados para alteração direta da precedência dos operadores. O operador de concatenação só poderá concatenar string com string, o tipo das operações é definido pelos operadores.

Operações Permitidas aos Tipos

Operador	Tipos que realizam a operação
‘!’	bool, int, float
‘^’	int, float
‘*’	int, float
‘/’	int, float
‘+’	int, float
‘-’	int, float
‘<’	int, float
‘<=’	int, float
‘>’	int, float
‘>=’	int, float
‘==’	int, float, bool, char, string
‘!=’	int, float, bool, char, string
‘&&’	int, float, bool
‘||’	int, float, bool
‘=’	int, float, bool, char, string
‘..”	string

Atribuição e Coerções

A atribuição é realizada através do token ‘opAttrib’ representado por um único sinal de igualdade ‘=’.

As coerções não serão admitidas para nenhum tipo primitivo da linguagem. Igualmente serve para conversão, do tipo explícita (cast).

Arranjos

Os arranjos começam na posição 0, para referenciar um elemento deve ser colocado em sequência `‘id’ ‘[’‘constInt’ | ‘id’‘]’`. *Exemplo: arr[0];.*

Há uma função que dirá o tamanho do arranjo, a função retornará um `‘constInt’`. *Exemplo: int s = arr.length();*.

Estruturas de Controle

Existem 3 estruturas de seleção, que possuem as seguintes estruturas:

‘if’’ ‘(‘ ‘ExpressãoBooleana’ ‘)’ ‘{‘
//código
‘}’

‘ceif’ ‘(‘ ‘ExpressãoBooleana’ ‘)’ ‘{‘
//código
‘}’

‘else’ ‘{‘
//código
‘}’

Onde, respectivamente, os lexemas de cada uma das construções são “if”, “ceif” (check else if) e “else” e ‘ExpressãoBooleana’ se caracteriza por uma conjunção, disjunção ou expressão booleana simples.

Comando de Iteração

Os comandos de iteração devem ser escritos das seguintes formas:

‘for’ ‘(‘ ‘int’ ‘id’ ‘:’ ‘(‘ (‘cte1’ | ‘id’ | ‘Ea’ ‘Op’ (‘id’ | ‘cte’), (‘cte2’ | ‘id’ | ‘Ea’ ‘Op’ (‘id’ | ‘cte’)), (‘cte3’ | ‘id’ | ‘‘Ea’ ‘Op’ (‘id’ | ‘cte’)) ‘)’ ‘)’ ‘{‘
//código
‘}’

‘while’ ‘(‘ ‘ExprBool’ ‘)’ ‘{‘

//código

}'

O laço de repetição ‘for’ haverá controle por contador. Ambos os loops farão um pré-teste para verificar se a repetição ainda ocorrerá ou não. No laço ‘for’ há uma ressalva, de que os valores de ‘cte1’, ‘cte2’ e ‘cte3’ não necessariamente precisam ser iguais, assim como os ‘id’ e os ‘Ea’. O terceiro valor é obrigatório, diz o quanto irá contar por ciclo. O valor ‘Op’ significa qualquer operador aritmético. Os respectivos significados são:

1. Para constantes: valor de inicialização, valor de comparação para parada(Ex: 5, significando que o contador irá parar quando atingir um valor > 5) e valor do incremento do contador;
2. Para id: valor de inicialização, valor de comparação para parada(Ex: ‘id’ = var. Significando que o contador irá parar quando atingir um valor > var) e valor do incremento do contador de tal forma que contador = contador + var;
3. Para expressões aritméticas: as regras permanecem as mesmas citadas acima, no entanto, elas poderão realizar as operações aritméticas: ‘+’, ‘-’, ‘/’, ‘*’, ‘%’.

Exemplos de utilização:

```
for(int i : (0, 5, 1)) {
        //código
}

while(var < 10) {
        //código
}

for(int i : (var, var + 5, 1)) {
        //código
}
```

Desvios Incondicionais

Softy não aceita nenhum tipo de desvio incondicional.

Entrada e Saída

- Entrada: É realizada através da função read().
    - O comportamento da função read dependerá do tipo o qual se está sendo lido. No caso de ser um inteiro, read atribuirá um único

inteiro ao valor designado ao ‘id’, assim como floats, char, string, bool. Caso a variável seja um arranjo, read irá ler “‘arranjo’.length()”. Read possui uma única exceção com relação a tipos, que é para variáveis do tipo void, nesse caso, a função deverá acusar erro.

- Saída: É realizada através da função print().
    - O comportamento da função print dependerá do tipo o qual se está sendo lido. No caso de ser um inteiro, print irá mostrar na tela um único inteiro: o valor designado ao ‘id’, assim como para floats, char, string, bool. Caso a variável seja um arranjo, print irá mostrar na tela “‘arranjo’.length()” valores, cada um relacionado a todas os valores nas posições do arranjo. Caso o valor seja relacionado a uma constante, print irá imprimir diretamente o valor dessa constante. Print tem uma única exceção com relação a tipos, que é para variáveis do tipo void, nesse caso, a função deverá acusar erro.

Códigos Exemplo

- Hello World:

```
fun int main() {
  print("Hello World");
}
```

- Shell Sort:

```
int n = 1000;

fun void shellSort(int arr(n)) {
        int gap = n / 2;
        while (gap > 0) {
                for (int i :(gap, n, 1)) {
                        int temp = arr[i];
                        int j = i;
                        while (j >= gap && (arr[j - gap] > temp)) {
                                arr[j] = arr[j - gap];
                                j = j - gap;
```

```
                }
                arr[j] = temp;
            }
            gap = gap / 2;
        }
    }


    fun int main() {
        int arr(n);
        read(arr);
        for (int i : (0, n, 1)) {
            print(arr[i]);
            print(" ");
        }
        shellSort(arr);
        for (int i : (0, n, 1)) {
            print(arr[i]);
            print(" ");
        }
        return 0;
    }
```

- Série de Fibonacci

```
    fun int main() {
        int n, n1 = 0, n2 = 1, n3;
        read(n);
        if (n == 1) {
            print("0\n");
        } else if (n == 2) {
            print("0 1\n");
        } else {
            print("0 1 ");
            for (int i : (0, n - 2, 1)) {
                n3 = n1 + n2;
                if (i == n - 3) {
                    print(n3);
                    print("\n");
                } else {
                    print(n3);
```

```
                print(" ");
            }
            n1 = n2;
            n2 = n3;
        }
    }
    return 0;
}
```