Especificação: SPL

Compiladores 2017.1

José Lucas Leite Calheiros

Agosto - 2017

Contents

1 Estrutura

Um programa SPL possui obrigatóriamente:

- Um bloco de declaração de funções iniciado pelo comando "func" e terminado pelo comando "end".
- Uma função principal que retorna um inteiro e é definida abaixo da área de declaração de funções.

```
func:
    <functions declaration>
end;
int main( ) {
    decl:
        <variables declaration>
    end;
    <instructions>
}
```

Obs: a função main respeita o modelo de estrutura de uma função comum que é definida na seção de funções.

2 Nomes

2.1 Identificadores

Em SPL os identificadores podem possuir até **16 caracteres**, obrigatoriamente começam com **letras**, sendo vetada a utilização de **caracteres especiais**, é **case sensitive**.

2.2 Palavras reservadas

- *int*
- *float*
- *char*
- *bool*
- *string*
- *array*
- *if*
- *else*
- *for*
- *while*
- *func*
- *decl*
- *end*
- *return*

3 Tipos e Declaração

SPL é **estaticamente tipada**, os tipos de suas variáveis devem ser explicitamente especificados no momento de sua declaração, e todas as suas declarações devem estar dentro do bloco de declarações da função, especificado na seção de funções, as declarações possuem a seguinte forma:

<tipo> <identificador>;

Durante a declaração não se pode realizar atribuições, as variáveis são inicializadas com valores default mostrados na Tabela 1.

- *int* : inteiro de 32 bits, seus literais são expressos em uma sequência de digitos decimais.
- *float* : ponto flutuante de 32 bits, seus literais são expressos em uma sequência de digitos decimais seguidos de um ponto e mais digitos.
- *char* : caracteres da tabela ascii, seus literais são representados por caracteres entre ' '.
- *bool* : existem dois valores deste tipo, true e false.
- *string* : cadeia de caracteres cujos literais são representados entre " ".
- *array <tipo>[tamanho] <identificador>* : arranjo unidimensional

SPL é uma linguagem fracamente tipada, possui conversão de tipos tanto de **alargamento** quanto de **estreitamento** entre float e int.

Table 1: Valores default

int	0
float	0.0
char	' '
bool	false
string	null
array	null

4 Escopo

As variáveis declaradas em SPL possuem o escopo estatico.

5 Operadores

Table 2: Tabela de precedência

Operador	Comentários	Tipos	Associatividade	Precedência
~	Menos unário	*int, float*	D - E	1
* /	Multiplicativos	*int, float*	E - D	2
%	Resto	*int*	E - D	2
+ -	Aditivos	*int, float*	E - D	3
<= >= < >	Comparativos	*int, float, char, string*	-	4
== !=	Igualdade	*int, char, string*	-	5
!	Negação	*bool*	D - E	6
&	Conjunção	*bool*	E - D	7
|	Disjunção	*bool*	E - D	8
#	Concatenação	ver item 5.4	E - D	9

5.1 Menos unário, Multiplicativos e Aditivos

Tais operações retornam o valor usual de sua utilização matemática.

Nos operadores binários Multiplicativos e Aditivos caso tenhamos um operando *int* e outro *float* teremos um float de retorno, em outro caso o valor retornado será o de seus operandos.

Em caso de utilização desses operadores se houver um operando como int e um como float a operação retornará um valor float.

5.2 Comparativos e Igualdade

Os operadores deste tipo não são associativos e retornam um valor booleano indicando a veracidade ou não da operação.

Em tais operadores pode-se apenas utilizar operandos de mesmo tipo, no caso de float e int a operação possui valor usual, no caso de char e string utiliza-se como parâmetro a ordem lexicográfica.

5.3 Negação, Conjunção e Disjunção

Tais operadores retornam um valor booleano definido pelo valor usual de sua utilização na lógica booleana.

5.4 Concatenação

No operador concatenação pelo menos um dos operandos deve ser do tipo string ou char, caso ambos operandos sejam strings ou char uma nova string é obtida concatenando-se ambas variáveis, no caso onde um dos operandos não seja string ou char é realizada uma conversão de valor para string e daí a concatenação é efetuada normalmente.

6 Instruções

6.1 Estruturas Condicionais

Condicional de uma via

A forma da estrutura condicional de uma via em SPL é composta pela palavra **if** seguida por uma expressão de valor booleano entre parêntesis e seguida por um bloco de instruções a serem executados caso a expressão possua valor verdadeiro, tal bloco deve obrigatóriamente estar situado entre chaves '{' '}'.

```
if(<bool expression>) {
    <instructions>
}
```

Condicional de duas vias

A forma da estrutura condicional de duas vias é análoga a de uma via até o fim do fechamento da chave '}' do bloco if, após isso é posto um comando **else** seguido por um bloco de instruções entre chaves '{' '}' a ser executado se a expressão booleana do if tiver valor falso.

```
if(<bool expression>) {
    <instructions>
}
else {
    <instructions>
}
```

6.2 Estruturas de Repetição

Controle lógico

A estrutura de repetição por controle lógico é definida por um comando **while** seguido de uma expressão de valor booleano e um bloco de instruções entre chaves '{"}'.

```
while(<bool expression>) {
    <instructions>
}
```

Contador

A estrutura de repetição por contador é definida por um comando **for** seguido de uma estrutura entre parênteses definida por uma variável inteira, ':' (dois pontos), o valor inicial a ser atribuído a essa variável, ',' (vírgula), o valor final que essa variável receberá, a variável será incrementada em 1 valor, o número de iterações será dado por num = valor final - valor inicial + 1, portanto o valor final deve ser maior que o inicial, e por fim após tal estrutura entre parêntesis teremos um bloco de instruções entre chaves '{"}'.

```
for(<identificador inteiro>: <valorinicial>,<valorfinal> ) {
    <instructions>
}
```

O comando for utiliza-se do método de pre-avaliação.

6.3 Entrada e Saída

Print

A saída é realizada pela função Print(<string>), que imprime a variavel string especificada na tela.
Uso:

```
1
    string str = "oi";
    string str2 = " bom dia";
    Print(str # str2);
2
    Print("oi bom dia");
```

Ambos usos imprimem a mensagem abaixo no console

```
oi bom dia
```

Read

A entrada é realizada pela função Read("<tipo>"), que lê toda a entrada até que a tecla enter seja pressionada, e devolve o valor do tipo especificado, também há o Read(EOF) que devolve um valor booleano caso seja encontrado o final do arquivo.
Uso:

```
string str;
str = Read("string");

int num;
num = Read("int");
```

6.4 Atribuição

A atribuição é uma instrução, o valor do lado direito da instrução é atribuido a variavel do lado esquerdo, não podem ser efetuadas atribuições no bloco de declarações de um programa.

```
<id> = <id>;
<id> = <expression>;
```

As atribuições devem respeitar a compatibilidade de tipos.

No caso de: <int> = <float> Ocorre uma conversão de estreitamento, a parte inteira do float é atribuida ao int.

No caso de: <float> = <int> Ocorre uma conversão de alongamento.

7 Funções

A forma geral de declaração de uma função em SPL é:

```
<tipo> <identificador>(<tipo> <id>, <tipo> <id2>,...) {
    decl
        <declarations>
    end
    <instructions>
}
```

Onde suas variáveis são declaradas dentro do bloco de declaração definido por "decl" e "end" e após o termino da função todas as variáveis são destruídas.

As funções podem retornar qualquer tipo.

As funções devem ser especificadas na área de declaração de funções acima da função main entre os comandos "func" e "end".

Para chamada de funções basta utilizar seu nome identificador seguido de seus parâmetros.

Dentre os parametros passados na chamada de função apenas o tipo array é passado por referência, o restante é passado por cópia.

8 Comentários

SPL possui apenas comentários de uma linha definidos pelo //

9 Programas exemplo

9.1 Hello World

```
func
end
int main() {
    decl
    end
    Print("Hello World");
}
```

9.2 Fibonacci

```
func
  int fibonacci(int n) {
      decl
          int i;
          int j;
          int k;
          int t;
```

```
        end
        i = 0;
        j = 1;
        Print(""#j);
        while( j < n ) {
            t = i + j;
            i = j;
            j = t;
            Print(", "#j);
        }
    return j;
    }
end

int main() {
    decl
        int n;
    end
    n = Read("int");
    n = fibonacci(n);
    return n;
}
```

9.3 Shell Sort

```
func
    int shellSort(array int[] vet, int size) {
        decl
            int i;
            int j;
            int value;
            int gap;
            int temp;
        end
        gap = 1;
        while(gap < size) {
            gap = 3*gap+1;
        }
        while ( gap > 1) {
            gap = gap / 3;
            for(i : gap, size) {
                value = vet[i];
                j = i - gap;
                while (j >= 0 & value < vet[j]) {
                    temp = j + gap;
                    vet [temp] = vet[j];
                    j = j - gap;
                }
                temp = j + gap;
                vet [temp] = value;
            }
        }
    return 0;
    }
end

int main() {
    decl
        int size;
```

```
        int i;
        array int[1000] v;
    end
    size = Read("int");
    for( i: 0, size-1 ) {
        v[i] = Read("int");
    }
    shellSort(v,size);
    for( i: 0, size-1 ) {
        Print( v[i]#" " );
    }
    return 0;
}
```