UNIVERSIDADE FEDERAL DE ALAGOAS - UFAL
INSTITUTO DE COMPUTAÇÃO - IC/UFAL
COMPILADORES

MEER - COMPILER

UNIVERSIDADE FEDERAL DE ALAGOAS - UFAL
INSTITUTO DE COMPUTAÇÃO - IC/UFAL
COMPILADORES

MEER - COMPILER

Trabalho solicitado pelo professor Dr. **Alcino Dall'Igna Júnior**, para avaliação e composição da primeira parte da nota, feito pelos alunos Vitor Barcelos de Cerqueira e Pedro Mateus Veras, alunos do Instituto de Computação da Universidade Federal de Alagoas.

Estrutura geral de um programa

- A função inicial a ser executada será a função que tem como identificador a palavra reservada ‘init’.
- Funções não podem ser declaradas dentro de outras funções, portanto sua declaração só é válida no escopo global do programa.
- Instruções e variáveis só podem ser declaradas no escopo de uma função.

Especificação de tipos

Booleano

Para identificar uma variável do tipo booleano é usada a palavra reservada ‘boolean’, as palavras reservadas ‘true’ e ‘false’ são usadas como constantes literais.

Caractere
Para identificar uma variável do tipo caractere é usada a palavra reservada ‘char’, esse tipo suporta qualquer caractere da tabela ASCII. A constante literal desse tipo é um caractere entre aspas simples, ex. ‘@’.

Cadeia de caracteres
Para identificar uma variável do tipo cadeia de caracteres é usada a palavra reservada ‘string’, esse tipo suporta qualquer cadeia de caracteres da tabela ASCII. A constante literal desse tipo é uma cadeia de caracteres entre aspas duplas, ex. “@#$a1”.

Inteiro
Para identificar uma variável do tipo inteiro é usada a palavra reservada ‘int’, esse tipo suporta qualquer número inteiro de 32 bits. A constante literal desse tipo é uma sequência de dígitos, ex. 123.

Ponto flutuante
Para identificar uma variável do tipo inteiro é usada a palavra reservada ‘float’, esse tipo suporta qualquer número de ponto flutuante de 64 bits. A constante literal desse tipo é uma sequência de dígitos, seguido por um ‘.’ e seguido por outra sequência de dígitos, ex. 12.34.

Arranjos unidimensionais
Para definir um arranjo unidimensional usa-se os colchetes ‘[]’ após a palavra reservada do tipo, ex. int[]. Caso os colchetes estejam vazios, o tamanho do arranjo será dinâmico, caso seja definido uma constante literal do tipo inteiro entre os colchetes, então o arranjo terá tamanho fixo igual a essa constante.

Operações de cada tipo

Tipo	Operações suportadas
int	+, -, *, /, +(unário), -(unário), ==, !=, <, >, <=, >=
float	+, -, *, /, +(unário), -(unário), ==, !=, <, >, <=, >=
char	+(concatenação), ==, !=
string	+(concatenação), ==, !=
boolean	==, !=, not

Coerções Explícitas

Coerções explícitas podem ser feitas por meio das funções nativas: Int(), Float(), String() e Boolean(). No qual o parâmetro de entrada será de um tipo e o retorno será o valor convertido.

Conjunto mínimo de operadores

Operadores aritméticos:
+, -, *, /, +(unário), -(unário)

Operadores relacionais:
<, >, <=, >=, ==, !=

Operador de concatenação:
+

Instruções

Toda instrução de linha precisa de um ‘;’ para terminar a instrução.

Estrutura condicional

São definidas pela palavra reservada ‘if’, em seguida uma expressão lógica que caso retorne verdadeiro o bloco da condicional ‘if’ será executado. O bloco da condicional ‘if’ é definido pelas palavras reservadas ‘then’ e ‘endif’. Caso o valor da expressão seja falso, a próxima estrutura condicional será executada, que será um elseif ou um else.

O elseif é similar ao if, sendo usado apenas nas estruturas subsequentes do if. O else será executado caso todas as estruturas condicionais anteriores tenham sido falsas.

ex.
```
if( a  >  b ) then
        ...
endif
elseif(b  >  c) then
        ...
endif
else
        ...
endif
```

Estrutura iterativa com controle lógico

É definida usando a palavra reservada ‘while’, em seguida a expressão lógica a ser testada. Essa estrutura é pré-teste e seu bloco a ser executado é definido pelo fechamento dos parênteses da expressão e pela palavra reservada ‘endwhile’.
ex.
```
while(true)
        ...
endwhile
```

Estrutura iterativa controlada por contador

É definida usando a palavra reservada ‘for’, em seguida três parâmetros são passados entre parênteses e separados por ‘,’. O primeiro parâmetro é a variável que será iterada, o segundo a variável ou constante literal com o limite da iteração e o terceiro e único opcional o passo da iteração, caso o passo não seja dado, terá valor 1. Seu bloco a ser executado é similar ao while.

ex.

```
for(var i: int = 0, 10, 2)
        …
endfor
```

Entrada

Definido pela palavra reservada ‘input’, em seguida as variáveis que receberão os valores lidos, separados por ‘,’.

Saída

Definido pela palavra reservada ‘print’, em seguida as variáveis ou constantes literais que serão

colocadas na tela, separados por ‘,’.

Atribuição

Definido pelo comando ‘=’, a esquerda é necessário uma ou mais variáveis, e na direita uma ou mais (variáveis ou constantes literais). É necessário que o número de elementos à esquerda seja exatamente igual ao número de elementos à direita. Os elementos são separados por ‘,’.

ex.
var a: int, b: int = 1, 2;

Funções

Definida pela palavra reservada ‘def’, em seguida o identificador da função e a lista de parâmetros. O tipo de retorno da função é definido usando o símbolo ‘:’. O bloco do escopo da função é definido pelas palavras reservadas ‘begin’ e ‘end’. A palavra reservada ‘return’ é usada para definir o retorno da função. Caso o tipo da função seja void, ela não terá retorno. Os parâmetros são sempre passados por Valor-Cópia.

ex.

```
def name(var a: int, var b: int): int
        …
        return a;
end
```

Exemplos de programa

Hello world

```
def init(args: string[]): void
begin
  print("Hello World");
end
```

Shell sort

```
def shellsort(var list: int[], var size: int): void
begin

  var gap: int = Int(size / 2);
```

```
  while(gap > 0)

        for(var i = gap, size)

            var temp: int = list[i];
            var j: int = i;

            while(j >= gap and list[j-gap] > temp)
                list[j] = list[j-gap];
                j = j - gap;
            endwhile

            list[j] = temp;

        endfor

        gap = Int(gap / 2);

  endwhile

end

def init(args: string[]): void
begin

  var size: int;
  input(size);
  var list: int[];

  for(var i = 0, size, 1)
        input(list[i]);
  endfor

  for(var i = 0, size, 1)
        print(list[i]);
  endfor

  list = shellsort(list, size);

  for(var i = 0, size, 1)
        print(list[i]);
  endfor

end
```

Fibonacci

```
def fibonacci(var lim: int): void
begin
  var result: int[] = [0, 1];
  var i: int = 1;

  while(result[i-1] + result[i] < lim)
            i = i + 1;
            result[i] = result[i-2] + result[i-1];
  endwhile

  var l: int = 0;
  var step: int = 1;

  for(l, i, step)
            print("%d ,", result[l]);
  endfor

end

def init(args: string[]): void
begin
  var lim: int;

  input(lim);
  fibonacci(lim);

end
```