TECNOLÓGICO DE COSTA RICA
Escuela de Ingeniería en Computación
Proyecto de Ingeniería de Software

Profesora:
Maria Estrada

Entrega 1:
Resumen técnico
de Lenguaje Triángulo

Estudiantes:
Christian Leon Guevara - 2013371982
Gabriel Ramírez Ramírez - 201020244

Fecha de entrega:
22-12-2018

Lenguaje Triángulo

Introducción

El lenguaje triángulo es un subconjunto del lenguaje de Pascal, se caracteriza por aun así ser regular y extensible. Este lenguaje es usado normalmente como método de enseñanza de cómo actúa un compilador. Este compilador contiene un alcance de declaraciones de encapsulamiento, esto se refiere que las declaraciones que son declaradas en segmento de código solo servirán para ese segmento de código o más profundo (depende del tipo de declaración).

El lenguaje triángulo tiene el siguiente tipo de Declaraciones:

- Valor: es un valor real, por ejemplo, un entero numérico, un valor de verdad, un carácter, un registro o un arreglo.
- Variable: Entidad que contiene un valor.
- Procedimiento: Entidad que tiene un bloque de código que ejecuta una serie de instrucciones.
- Función: Entidad que tiene un bloque de código que será evaluado para retornar un valor
- Tipo: Determina un conjunto de características que el valor va a tener. Cada variable, valor y función lo tienen.

El lenguaje se puede dividir en múltiples secciones, a continuación, se explicará el comportamiento de algunos segmentos de la gramática.

Comandos

Un comando es una línea de código valida, que se puede concatenar una después de otra.

```
Command        ::=  single-Command
                |   Command ; single-Command

single-Command ::=
                |   V-name := Expression
                |   Identifier ( Actual-Parameter-Sequence )
                |   begin Command end
                |   let Declaration in single-Command
                |   if Expression then single-Command
                        else single-Command
                |   while Expression do single-Command
```

- La expresión “V-name := Expression” funciona como una asignación en donde “V-name” es la variable y “Expression” es la expresión.
- La expresión Identifier(Actual-Parameter-Sequence), funciona si Identifier es un procedimiento y Actual-Parameter-Sequence es una secuencia de parámetros válida.
- Se ejecuta un comando tras otro.

- En caso de que exista “begin C end”, se ejecutará solamente C.
- El if se ejecuta de tal manera que si la primera expresión es verdadera se ejecutará la segunda expresión, de lo contrario será la tercera expresión. El lenguaje debe verificar que la primera expresión tenga valor booleano para este efecto.
- El while funciona de la siguiente manera, se evalúa la expresión y si es true entonces se sigue ejecutando el siguiente single-Command. También hay que verificar que la expresión tenga valor booleano

Expresiones

Una expresión es un bloque de código que puede devolver un valor.

```
Expression            ::=  secondary-Expression
                         |  let Declaration in Expression
                         |  if Expression then Expression else Expression

secondary-Expression  ::=  primary-Expression
                         |  secondary-Expression Operator primary-Expression


primary-Expression    ::=  Integer-Literal
                         |  Character-Literal
                         |  V-name
                         |  Identifier ( Actual-Parameter-Sequence )
                         |  Operator primary-Expression
                         |  ( Expression )
                         |  { Record-Aggregate }
                         |  [ Array-Aggregate ]

Record-Aggregate      ::=  Identifier ~ Expression
                         |  Identifier ~ Expression , Record-Aggregate

Array-Aggregate       ::=  Expression
                         |  Expression , Array-Aggregate
```

- La expresión “Integer_Literal” se refiere a un valor entero numérico.
- La expresión “Character_Literal” se refiere a un valor de tipo carácter.
- La expresión “V-name” se refiere a un nombre de una variable.
- El bloque let permite declarar una variable A y asignarle un valor B para ese segmento de código en específico. Estas variables no tienen efecto fuera del segmento de código en el que se declararon.

Nombre de variable

Un nombre de variable es un identificador al cual se le tiene asignado un tipo y se le puede asignar un valor.

```
V-name    ::=  Identifier
            |  V-name . Identifier
            |  V-name [ Expression ]
```

Un nombre de variable se puede expresar de 3 diferentes maneras:

- Simplemente escribiendo un identificador
- Haciendo la llamada a un registro (V-name.Identifier)

- Haciendo la llamada a un arreglo(V-name[Expression])

Las 2 últimas se pueden concatenar siempre y cuando siempre empiece con un identificador.

Declaraciones

Sirve para enlazar 2 cosas, usualmente a alguna expresión.

```
Declaration          ::=  single-Declaration
                       |  Declaration ; single-Declaration

single-Declaration   ::=  const Identifier ~ Expression
                       |  var Identifier : Type-denoter
                       |  proc Identifier ( Formal-Parameter-Sequence ) ~
                              single-Command
                       |  func Identifier ( Formal-Parameter-Sequence )
                              : Type-denoter ~ Expression
                       |  type Identifier ~ Type-denoter
```

Las declaraciones funcionan de la siguiente manera:

- La declaración tipo const se enlaza un valor a un identificador, este valor no se puede alterar.
- La de tipo var, declara una variable y el tipo correspondiente; si se le puede asignar un valor después.

Parámetros

Es la especificación de identificadores en el cuerpo de una función o procedimiento. Existe la especificación de los parámetros y los parámetros reales. La especificación determina ciertas características de los parámetros a usar (como declarando una variable) y los parámetros reales son cualquier expresión o llamada a función que devuelva un valor.

```
Formal-Parameter-Sequence
                     ::=
                       |  proper-Formal-Parameter-Sequence

proper-Formal-Parameter-Sequence
                     ::=  Formal-Parameter
                       |  Formal-Parameter , proper-Formal-Parameter-Sequence
```

```
Formal-Parameter  ::=  Identifier : Type-denoter
                   |   var Identifier : Type-denoter
                   |   proc Identifier ( Formal-Parameter-Sequence )
                   |   func Identifier ( Formal-Parameter-Sequence )
                           : Type-denoter

Actual-Parameter-Sequence
                  ::=
                   |   proper-Actual-Parameter-Sequence

proper-Actual-Parameter-Sequence
                  ::=  Actual-Parameter
                   |   Actual-Parameter , proper-Actual-Parameter-Sequence

Actual-Parameter  ::=  Expression
                   |   var V-name
                   |   proc Identifier
                   |   func Identifier
```

Tipos

Un tipo es una clase de etiqueta que se le da a algo para otorgarle ciertos atributos a ese algo. Se le puede enlazar un tipo a valores, constantes, variables y funciones.

```
Type-denoter          ::=  Identifier
                       |   array Integer-Literal of Type-denoter
                       |   record Record-Type-denoter end

Record-Type-denoter   ::=  Identifier : Type-denoter
                       |   Identifier : Type-denoter , Record-Type-denoter
```

- Los registros son un encapsulamiento de variables, en donde son accedidos por medio de un identificador.
- El Identifier se refiere a un Tipo ya definido por el sistema como Integer, Boolean, etc.

Bibliografía

- Watt, David A., and Deryck F. Brown. *Programming Language Processors in Java: Compilers and Interpreters*. Pearson Education (Singapore) Pty. Ltd., 2003.