1/2/2023

Partes del Lenguaje Rursus

Instituto Tecnológico de Costa Rica

Escuela de Computación

Computadores e Intérpretes

Logo, company name

Description automatically generated

Profesor:

Kirstein Gatjens

Estudiantes:

Jordano Escalante López

Andrés Sánchez Rojas

Índice

1. Elementos del lenguaje con ejemplos: p3-p16
2. Tabla de Precedencia: p17
3. Operaciones de conversión entre tipos: p18-p19
4. Familias de Tokens p20
5. Errores del Scanner p21

**1) Estructura del título del programa FC**: nomen

Nos permite abrir el programa, en este caso se llama EJEMPLO.

nomen EJEMPLO .

…

…

…

**2) Apertura sección constantes:** perpetuus

Nos permite abrir la sección de constantes

perpetuus <# Abre la sección de constantes #>

…

…

…

**3) Apertura sección de tipos**: furibundus

Nos permite abrir la sección de declaración de tipos

furibundus <# Abre la sección de tipos #>

…

…

…

**4) Apertura de la sección de variables**: commutabilis

Nos permite abrir la sección de declaración de variables.

commutabilis <# Abre la sección de variables #>

…

…

…

**5) Apertura de la sección de rutinas**: corpus

Nos permite abrir la sección de rutinas

corpus <# Abre la sección de rutinas #>

…

…

…

**6) Apertura de la sección de prototipos**: exemplar

Nos permite abrir la sección de prototipos

exemplar <# Abre la sección de prototipos #>

…

…

…

**7) Manejo de Bloques de más de una instrucción**: initum, finis

Con estas dos instrucciones se van a abrir y cerrar bloques de código. Por ejemplo si tenemos un “in” y queremos que haya un bloque de código dentro de este lo abrimos con un “initum” y lo cerramos con un “finis”

initum

…

…

…

finis.

**8) Punto de entrada del programa**: firmamentum

Por aquí va a empezar a correr el código

firmamentum <# Punto de entrada del programa #>

initum

…

…

…

finis.

**9) Sistema de asignación de constantes FC**: dito

Es la operación que se usa para darle un valor a las constantes

perpetuus <# Abre la sección de constantes #>

A dito 45 .

B dito 'K' .

C dito {'A','B','C'} .

**10) Sistema de asignación de tipos FC**: sum

Con este operador se van a crear los tipos que necesitemos.

furibundus <# Abre la sección de tipos #>

Edad sum numerus .

vector sum ordo [100] autem numerus .

matriz sum ordo [100] autem vector .

tablero sum matriz .

Persona sum conjugo

catena nombre .

numerus cedula .

edad vejez .

dixi

**11) Sistema de declaración de variables FC**: est

Con este operador le damos valores a las variables.

commutabilis <# Abre la sección de variables #>

numerus X est 5 , Y est 4 , Z .

ordo [3][4] autem imago Var1 est {[ {[1,2,3,4]} , {[5,6,7,8]} , {[9,10,11,12]} ]} .

## así no es {[{[1,2,3]},{[4,5,6]},{[7,8,9]},{[10,11,12]}}.

conjugo

catena nombre .

numerus cedula .

edad vejez .

dixi Juan est << "Juan", 123456 , 88 >> .

Persona Pedro est << "Pedro", 666 , 66 >> .

**12) Tipo de dato entero**: numerus

Es el tipo entero

numerus unNumero.

**13) Tipo de dato carácter**: imago

Es el tipo caracter

imago unaLetra.

**14) Tipo de dato string**: catena

Es el tipo string

catena unaPalabra.

**15) Tipo de dato booleano**: dualis

Es el tipo booleano

dualis verdaderoOrFalso.

**16) Tipo de dato conjunto FC**: statuo

Es el tipo de datos que representa un conjunto

statuo unConjunto.

**17) Tipo de dato archivo de texto FC**: liber

Es el tipo de datos que representa un archivo de texto

liber unArchivo.

**18)** **Tipo de dato arreglos FC**: ordo, autem

Con esto vamos a declarar los arreglos

ordo [dimension1] [dimension2] autem tipo.

<#Como es un tipo de dato mas complejo, se incluir un ejemplo de su declaración también#>

ordo [3][4] autem numerus arreglo1 est {[ {[1,2,3,4]} , {[5,6,7,8]} , {[9,10,11,12]} ]}.

**19) Tipo de dato registros FC**: coniugo, dixi

Con estos vamos a declarar registros

Persona conjugo

catena nombre .

numerus cedula .

edad vejez .

dixi

<#Como es un tipo de dato mas complejo, se incluir un ejemplo de su declaración también#>

Persona Pedro est << "Pedro", 666 , 66 >> .

**20) Literales booleanas FC**: veridicus y falsidicus

Estos son los dos valores que puede tener un elemento de tipo booleano

**21) Literales de conjuntos FC**: { 'X1', 'X2',….,'Xn' }

Formato de los valores que puede tener un conjunto.

**22) Literales de archivos FC**: {@ "c:\akw\tarea1.txt" , L @}

Ejemplo de formato que puede tener un elemento de tipo archivo

**23) Literales de enteros**: -123, 0xF4EC

Ejemplos de valores que puede tener un elemento de tipo entero

**24) Literales de caracteres**: Con comilla simple: 'A', '\n', '\666'

Ejemplos de valores que puede tener un caracter

**25) Literales de strings**: "Hola \n"

Ejemplo de un posible valor de strings

**26) Literales de arreglos FC**: {[ X1, X2 , ... , Xn ]}

Ejemplo del posible valor de un arreglo

**27) Literales de registros FC**: << X1, X2, ... , Xn >>

Ejemplo del posible valor de un registro

**28) Sistema de acceso arreglos**: arreglo1[campo1][campo2]

De esta manera se accederán los arreglos. En este ejemplo se imagina arreglo1 como un arreglo de dos dimensiones y campo1 y campo2 como enteros.

**29) Sistema de acceso strings**: string $$ campo

Con esto se va a acceder a una posición de un string

**30) Sistema de acceso registros**: registro @ campo

Con este sistema se va a acceder a los registros

**31) Asignación y Familia**: := += \*= etc

Toda operación binaria se puede combinar con el igual.

**32) Operaciones aritméticas básicas de enteros**: + - \* % /

Con estas operaciones vamos a trabajar los enteros

**33) Incremento y Decremento**: incrementum, decrementum

Nos permite incrementar o decrementar el valor de un elemento por 1, debe ser usada en prefijo. Ej: incrementum x

**34) Operaciones básicas sobre caracteres**:

Con estas operaciones vamos a trabajar sobre los caracteres

Pasar a mayúscula= string[>>]

Pasar a minúscula= string[<<]

isAlpha= string[&?]

isDigit= string[#?]

**35) Operaciones lógicas solicitadas**:

Con estas operaciones vamos a trabajar con booleanos

and=et

or=aut

xor=xaut

not=non

**36) Operaciones de Strings solicitadas:**

Vamos a usar estas operaciones para trabajar con strings

Concatenar= $+

Length= $#

Cortar= x$2|>3

Recortar= x$2|<3

Find= X$?’s’

**37) Operaciones de conjuntos solicitadas**:

Estas son las operaciones que vamos a usar para trabajar con conjuntos

Agregar= addere

Borrar= necar

Unión= unionis

Intersección= intersectio

Pertenece= pertinet

isEmpty= vacua

**38) Operaciones de archivos solicitadas**:

Estas son las operaciones que vamos a usar para trabajar con archivos. (Todavía no entiendo ligo)

Open= patentibus

Close= prope

Write= scripturam

Read= lectio

Create= partum

Asociar= ligo

**39) Operaciones de comparación de enteros solicitadas**: > < = >= <= ><

Estas son operaciones de comparación de enteros. El = es para comprobar igualdad y el >< es para comprobar que sean desigualer. Los demás se explican por si solos

**40) Instrucción while**: tempus, opus

Nos permite crear búcles en el código

tempus condición opus

initum

…

…

…

finis.

**41) Instrucción if-then-else**: in-certus-mentira

Nos permite crear bifurcaciones en el código

in N<2

certus

R:= 1 .

mentiri

R:= fibonacci(N-1)+fibonacci(N-2).

.

**42) Instrucción switch**: aeger casus detrimentum

Nos permite hacer bifurcaciones en el código

aeger X

initum

casus 1 opus A := 1 . neco .

casus 2 opus A := 2 . neco .

casus 3 opus A := 3 . neco .

detrimentum A := 0 . neco .

casus 100 opus A := 100 . neco .

finis .

**43) Instrucción Repeat-until**: itero usque

Nos permite crear búcles en el código

itero

…

…

…

usque condición.

**44) Instrucción For**: panis i := 0 auctum 5 gradus 1 opus

Nos permite crear búcles en el código

panis I := 1 auctum 100 gradus 2 opus

escribonumerus(I). .

**45) Instrucción With**: sigla id opus instrucciones dixi

Se utiliza para acceder a propiedades de registros

sigla variable opus

initum

…

…

finis.

dixi.

**46)Instrucción break**: neco

Saca el flujo de un bucle e ignora el resto que pueda quedar

itero

…

in condición

certus

neco.

mentiri

…

…

usque condición.

**47) Instrucción continue**: pergo

Saca el flujo de un bucle pero termina la iteración actual

itero

…

in condición

certus

pergo.

mentiri

…

…

usque condición.

**48) Instrucción Halt**: claudeo

Detiene la ejecución del programa

in condición

certus

pergo.

mentiri

…

**49) Encabezado de funciones**: efficio id ( ... ) : tipo

Así es como se va a escribir el encabezado de las funciones

efficio fibonacci(numerus N) : numerus .

conmutabilis

Numerus R.

initum

in N<2

certus

R:= 1 .

mentiri

R:= fibonacci(N-1)+fibonacci(N-2).

.

reditus R.

finis .

**50) Encabezado de procedimientos**: directus id ( ... )

directus ejemplo1(numerus A,B . ign fractio F,G,H . catena S ).

commutabilis

numerus X .

coniugo

numerus horas .

numerus minutos .

numerus segundos .

dixi Tiempo est << 3 , 15 , 45 >> .

catena Hola .

ordo [5] autem imago vocales est {[ 'A' , 'E' , 'I' , 'O', 'U' ]} .

initum

finis .

**51) Manejo de parámetros formales**: ( numerus X,Y . livre A . ign numerus Z)

ign significa que se pasan por referencia

**52) Manejo de parámetros reales**: x(a, b, c)

Así es como se pasan los parámetros a una función

**53) Instrucción return**: reditus

**54) Operación de size of**: quantus expresion o tipo

**55) Sistema de coerción de tipos**: expresión impono tipo

**56) Manejo de la entrada estándar**: logo***tipo***

Donde dice ***tipo*** escribir el tipo deseado, string, archivo, etc… Solo pueden ser los seis tipos atómicos

**57) Manejo de la salida estándar**: scribo***tipo***

Donde dice ***tipo*** escribir el tipo deseado, string, archivo, etc… Solo pueden ser los seis tipos atómicos

**58) Terminador**: .

Tabla de Precedencia

|  |  |  |
| --- | --- | --- |
| Símbolo | Tipo de Operación | asociatividad |
| “[”, “]”, “(”, “)”, “.” | Expresión | Izquierda a Derecha |
| “incrementum”, “decrementum” | Unaria | Derecha a Izquierda |
| “\*”, “/”, “%” | Multiplicativa | Izquierda a Derecha |
| “+”, “-” | Aditiva | Izquierda a Derecha |
| “>”, “<”, “>=”, “<=”,“><” | Comparación | Izquierda a Derecha |
| “:=”, “+=”, “-=”, “\*=”, “/=”, “%=” | Asignación | Izquierda a Derecha |
| “[>>]”, “[<<]”, “[&?]”, “[#?]” | Operaciones de Caracteres | Izquierda a Derecha |
| “et”, “aut”, “xaut”, “non” | Operaciones Lógicas | Izquierda a Derecha |
| “$+”, “$#”, “$num|>num”, “$num|<num”, “$?” | Operaciones de Strings | Izquierda a Derecha |
| “addere”, “necar”, “unionis”, “intersectio”, “pertinet”, “vacua” | Operaciones de Conjuntos | Izquierda a Derecha |
| “patentibus”, “prope”, “scripturam”, “lectio”, “partum”, “ligo” | Operaciones de Archivos | Izquierda a Derecha |

Se hizo basada en: <https://learn.microsoft.com/es-es/cpp/c-language/precedence-and-order-of-evaluation?view=msvc-170>

Operaciones de Conversion entre tipos

1. Int a Bool: Si es 0 o 1 lo va a dejar quieto, si es mayor a 1 se le asignará un 1, si es menor a 0 será 0
2. Char a Bool: Si el caracter no tiene por valor 0 va a ser 1, de lo contrario va a ser 0
3. File a Bool: Se va a tomar la primera letra del string se le aplicará el mismo proceso que a los Char
4. String a Bool: va a tomar el la primera letra del nombre y se le aplicará el mismo proceso que a los Char
5. Set a Bool: va a tomar el primer elemento del conjunto y le va a aplicar la misma operación de los Char
6. Int a Char: Va a tomar el valor en ascii del caracter, si el número se excede, se le asignará un 127
7. Bool a Char: '1' (49) si es verdadero '0' (48) si es falso
8. File a Char: va a tomar la primera letra del nombre
9. String a Char: va a tomar la primera letra del string
10. Set a Char: va a tomar el primer elemento del conjunto y le va a aplicar la misma operación de los Int
11. Char a File: Va a tomar el caracter y crear un archivo con esa letra como nombre
12. Bool a File: Va a crear un archivo con nombre 1 o 0
13. Int a File: va a tomar los caracteres del número y crear un archivo con eso como nombre
14. String a File: Va a crear un archivo con el string como nombre
15. Set a File: Va a tomar cada elemento del conjunto y los va a usar como
16. Char a Int: Va a tomar el valor en ascii del carácter
17. Bool a Int: 1 si es verdadero 0 si es falso
18. File a Int: va a tomar el largo del nombre del archivo como el resultado
19. String a Int: va a tomar el largo del string como el resultado
20. Set a Int: va a tomar el tamaño del conjunto como resultado
21. Char a Set: Va a crear un conjunto con el caracter como único elemento
22. Bool a Set: Va a crear un conjunto con 1 o 0 como único elemento
23. File a Set: Va a crear un conjunto con las letras del nombre del archivo
24. String a Set: Va a crear un conjunto con las letras del string
25. Int a Set: Va a crear un conjunto con el enteo como único elemento
26. Char a String: Se va a crear un dollarString con el caracter como su único elemento
27. Bool a String: Se va a crear un dollarString con el caracter '1' o '0' como su único elemento
28. File a String: Se va a crear un dollarString con el nombre del archivo
29. Set a String: Se va a crear un dollarString con los valores del set, se manejarán como caracteres
30. Int a String: Se ca a crear un dollarString con el valor del int, se manejará como un caracter

Familias de Tokens

Se va a usar strings así como expresiones regulares y una explicación de los algoritmos que reconocen las diferentes familias o categorías cuando sea necesaria.

Operaciones = [ "<", ">", "=", ">=", "<=", "><", "[>>]", "[<<]", "[&?]", "[#?]", "$\+", "\$\#", r'[\*]$[\*]\|\>[\*]', r'[\*]$[\*]|<[\*]', r'[\*]$\?[\*]', "\+", "\-", "\\*", "%", "/", "\:=", "\+=", "\\*=", "%=", "\/=", "\:", "addere", "necar", "unionis", "intersectio", "pertinet", "vacua","patentibus", "prope", "scripturam", "lectio", "partum", "ligo"]

Palabras reservadas = ["incrementum", "decrementum", "numerus", "imago", "catena", "dualis", "statuo", "liber", "ordo", "autem", "conjugo", "dixi", "firmamentum", "detrimentum", "casus", "neco", "finis", "aeger", "initum", "opus", "dixi", "itero", "usque", "sigla", "panis", "auctum", "gradus", "tempus", "certus", "mentiri", "pergo", claudeo", "directus", "est", "sum", "dito", "nomen", "perpetuus", "furibundus", "commutabilis", "exemplar", "corpus", "in","\.", "\,", "\(","\)","\[","\]","\{","\}"]

Enteros = r'[\+\-]?\b[0-9]+\b'

Identificadores = "[\w\-]+"

Para strings se buscan las comillas de apertura, y se considera todo lo que se encuentre parte de este string hasta encontrar las comillas que lo terminen.

Para caracteres se busca la comilla de apertura y si el siguiente carácter o el que le sigue no son la comilla de cerradura marca un error.

Si el token no matchea con ninguna de estas familias se le considera un error.

Errores del Scanner

Error 100: No reconoció un identificador

Error 101: No reconoció un caracter