PROCESADORES DEL LENGUAJE

PRÁCTICA DE SEPTIEMBRE

Profesor: José Luis Sierra Rodríguez

Alumno: Manuel Artero Anguita

Grupo: A

Fecha: 15 – Septiembre - 2012

ÍNDICE

1. LISTA DE ERRATAS Y MODIFICACIONES REALIZADAS A LA ESPECIFICACIÓN

Corrección a la regla "Instruccion ::= call IDEN ParametrosReales"

Detección del problema: Imáginemos que tenemos en un cuerpo de un programa i1, i2, i3 valiéndonos de la promoción ListaInstrucciones ::= ListaInstrucciones, Instruccion; por ejémplo

i1 = *call f with x, y*

i2 = *out x*,

i3 = *out y*

Aplicando la regla quedaría *call f with x, y, out x, out y*. Por lo tanto no tenemos forma de determinar el final de la lista de parámetros reales y el comienzo de la siguiente instrucción. Necesitamos algún tipo de delimitador.

Solución: Existen dos posibilidades

1) Modificar la regla => Instruccion ::= call IDEN ParametrosReales **end call**
2) Modificar la regla ParametrosReales => ParametrosReales ::= with ListaParametrosReales **end with**

También se podrían combiar ambas soluciones marcando el final de *call* y el final de los parámetros sin problemas.

Correcciones a la especificación recogidas en el foro de la asignatura

(Agradecimiento en especial a *Rafael Antúnez Torrejón* por recopilarlas bajo un mismo *post*)

ListaDeclaraciones ::= ListaDeclaraciones, Declaracion

~~Declaracion.dirh = ListaDeclaraciones(0).dirh~~

Declaracion.dirh = ListaDeclaraciones(1).dir

Declaraciones ::= decs ListaDeclaraciones end decs

Declaraciones.cod = ListaDeclaraciones.cod.

Declaraciones ::= λ

Declaraciones.cod = programaVacio()

Instruccion ::= out IDEN

~~Instruccion.etq = Exp0.etq + numeroInstruccionesEscritura(IDEN.lexema, Instruccion.tsh)~~

Instruccion.etq = Instruccion.etqh + numeroInstruccionesEscritura(IDEN.lexema, Instruccion.tsh)

~~Exp0.etqh = Instruccion.etqh~~

Instruccion ::= if (exp0) then Instrucciones {ParteElse}

~~ParteElse.etqh = Instrucciones.etqh~~

ParteElse.etqh = Instrucciones.etq

ParteElse ::= else {Instrucciones}

~~Instrucciones.etqh = ParteElse.etq + 1~~

Instrucciones.etqh = ParteElse.etqh + 1

~~ParteElse.ir_f = ParteElse.etq + 1~~

ParteElse.if_f = ParteElse.etqh + 1

Correcciones a la especificación no recogidas en el foro de la asignatura

Instruccion ::= set IDEN to Exp0

~~Instruccion.etq = Exp0.etq + numeroInstruccionesAsignacion(IDEN.lex, Instruccion.tsh)~~

Instruccion.etq = Exp0.etq + numeroInstruccionesAsignacion(IDEN.lex, Instruccion.tsh, Exp0.esDesignador)

ListaParametrosReales ::= ListaParametrosReales, Exp0

~~ListaParametrosReales(0).etq = Exp0.etq + numeroInstruccionesCodigoPaso()~~

ListaParametrosReales(0).etq = Exp0.etq +
numeroInstruccionesCodigoPaso(ListaParametrosReales(0).subprogramah,
ListaParametrosReales(1).nparams + 1, Exp0.esDesignador,
ListaParametrosReales(0).tsh)

~~ListaParametrosReales(0).cod = ListaParametrosReales(1).cod || copia() || Exp0.cod ||~~
~~codigoPaso(ListaParametrosReales(0).subprogramah, ListaParametrosReales(1).nparams ,~~
~~Exp0.esDesignador, ListaParametrosReales(0).tsh)~~

ListaParametrosReales(0).cod = ListaParametrosReales(1).cod || copia() || exp0.cod ||
codigoPaso(ListaParametrosReales(0).subprogramah, ListaParametrosReales(1).nparams **+ 1,**
Exp0.esDesignador, ListaParametrosReales(0).tsh)

Otras correcciones menores

Exp1 ::= Exp1 OpAditivo Exp2

~~Exp1.esDesignador = false~~

Exp1(0).esDesignador = false

Exp0 ::= Exp1 OpComparacion Exp1

~~Exp0(0).esDesignador = false~~

Exp0.esDesignador = false

ListaParametrosReales ::= Exp0

~~ListaParametrosReales(0).nparams = 1~~

ListaParametrosReales.nParams = 1

2. DISEÑO DEL CONSTRUCTOR DE ÁRBOLES

Gramática de atributos que especifica el constructor de árboles de la gramática.
Aparecen marcadas las producciones que necesitarán acondicionamiento para permitir un analizador descendente predictivo recursivo: en azul marino las producciones con recursión a izquierdas; en gris las producciones con un problema de factorización.

Programa ::= Declaraciones Cuerpo

Programa.a = **programaR1**(Declaraciones.a, Cuerpo.a)

Declaraciones ::= decs ListaDeclaraciones end decs

Declaraciones.a = **declaracionesR1**(ListaDeclaraciones.a)

Declaraciones ::= λ

Declaraciones.a = **declaracionesR2**()

ListaDeclaraciones ::= ListaDeclaraciones , Declaracion

ListaDeclaraciones(0).a = **listaDeclaracionesR1**(ListaDeclaraciones(1).a, Declaracion.a)

ListaDeclaraciones ::= Declaracion

ListaDeclaraciones.a = **listaDeclaracionesR2**(Declaracion.a)

Declaracion ::= IDEN

Declaracion.a = **declaracionR1**(IDEN.lexema)

Declaracion ::= proc IDEN (ParametrosFormales) Declaraciones Cuerpo

Declaracion.a = **declaracionR2**(IDEN.lexema, ParametrosFormales.a, Declaraciones.a, Cuerpo.a)

ParametrosFormales ::= ListaParametrosFormales

ParametrosFormales.a = **parametrosFormalesR1**(ListaParametrosFormales.a)

ParametrosFormales ::= λ

ParametrosFormales.a = **parametrosFormalesR2**()

ListaParametrosFormales ::= ListaParametrosFormales , ParametroFormal

ListaParamFormales(0).a = **listaParametrosFormalesR1**(ListaParamFormales(1).a, ParametroFormal.a)

ListaParametrosFormales ::= ParametroFormal

ListaParametrosFormales.a = **listaParametrosFormalesR2**(ParametroFormal.a)

ParametroFormal ::= IDEN

ParametroFormal.a = **parametroFormalR1**(IDEN.lexema)

ParametroFormal ::= var IDEN

ParametroFormal.a = **parametroFormalR2**(IDEN.lexema)

Cuerpo ::= body Instrucciones end body

Cuerpo.a = **cuerpoR1**(Instrucciones.a)

Instrucciones ::= Instrucciones, Instruccion

Instrucciones(0).a = **instruccionesR1**(Instrucciones(1).a, Instruccion.a)

Instrucciones ::= Instruccion

Instrucciones.a = **instruccionesR2**(Instruccion.a)

Instruccion ::= set IDEN to Exp0

Instruccion.a = **instruccionR1**(IDEN.lexema, Exp0.a)

Instruccion ::= call IDEN ParametrosReales end call

Instruccion.a = **instruccionR2**(IDEN.lexema, ParametrosReales.a)

Instruccion ::= if Exp0 then {Instrucciones} ParteElse

Instruccion.a = **instruccionR3**(Exp0.a, Instrucciones.a, ParteElse.a)

Instruccion ::= out IDEN

Instruccion.a = **instruccionR4**(IDEN.lexema)

ParametrosReales ::= with ListaParametrosReales end with

ParametrosReales.a = **paramerosRealesR1**(ListaParametrosReales.a)

ParametrosReales ::= λ

ParametrosReales.a = **parametrosRealesR2**()

ListaParametrosReales ::= ListaParametrosReales , Exp0

ListaParametrosReales(0).a = **listaParametrosRealesR1**(ListaParametrosReales(1).a, Exp0.a)

ListaParametrosReales ::= Exp0

ListaParametrosReales.a = **listaParametrosRealesR2**(Exp0.a)

ParteElse ::= else {Instrucciones}

ParteElse.a = **parteElseR1**(Instrucciones.a)

ParteElse ::= λ

ParteElse.a = **parteElseR2**()

Exp0 ::= Exp1 OpComparacion Exp1

Exp0.a = **exp0R1**(Exp1(0).a, OpComparacion.a, Exp1(1).a)

Exp0 ::= Exp1

Exp0.a = **exp0R2**(Exp1.a)

Exp1 ::= Exp1 OpAditivo Exp2

Exp1(0).a = **exp1R1**(Exp1(1).a, OpAditivo.a, Exp2.a)

Exp1 ::= Exp2

Exp1.a = **exp1R2**(Exp2.a)

Exp2 ::= Exp2 OpMultiplicativo Exp3

Exp2(0).a = **exp2R1**(Exp2(1).a, OpMultiplicativo.a, Exp3.a)

Exp2 ::= Exp3

Exp2.a = **exp2R2**(Exp3.a)

Exp3 ::= OpUnario Exp3

Exp3(0).a = **exp3R1**(OpUnario.a, Exp3(1).a)

Exp3 ::= Exp4

Exp3.a = **exp3R2**(Exp4.a)

Exp4 ::= NUM

Exp4.a = **exp4R1**(NUM.lexema)

Exp4 ::= IDEN

Exp4.a = **exp4R2**(IDEN.lexema)

Exp4 ::= (Exp0)

Exp4.a = **exp4R3**(Exp0.a)

OpComparacion ::= =

OpComparacion.a = **opComparacionR1**()

OpComparacion ::= =/=

OpComparacion.a = **opComparacionR2**()

OpComparacion ::= >

OpComparacion.a = **opComparacionR3**()

OpComparacion ::= >=

OpComparacion.a = **opComparacionR4**()

OpComparacion ::= <

OpComparacion.a = **opComparacionR5**()

50) OpComparacion ::= <=

OpComparacion.a = **opComparacionR6**()

51) OpAditivo ::= +

OpAditivo.a = **opAditivoR1**()

52) OpAditivo ::= -

opAditivo.a = **opAditivoR2**()

53) OpAditivo ::= or

opAditivo.a = **opAditivoR3**()

54) OpMultiplicativo ::= *

opMultiplicativo.a = **opMultiplicativoR1**()

55) OpMultiplicativo ::= /

opMultiplicativo.a = **opMultiplicativoR2**()

56) OpMultiplicativo ::= and

opMultiplciativo.a = **opMultiplciativoR3**()

3. TRANSFORMACIONES REALIZADAS SOBRE EL CONSTRUCTOR DE ÁRBOLES

Resultado de resolver la recursión a izquierdas sobre *ListaDeclaraciones*, *ListaParametrosFormales*, *Instrucciones*, *ListaParametrosReales*, *Exp1*, *Exp2* ; así como la factorización sobre *Exp0*

ListaDeclaraciones ::= Declaracion RlistaDeclaraciones

RListaDeclaraciones.ah = **listaDeclaracionesR2**(Declaracion.a)

ListaDeclaraciones.a = RListaDeclaraciones.a

RListaDeclaraciones ::= , Declaracion RListaDeclaraciones

RListaDeclaraciones(1).ah = **listaDeclaracionesR1**(RListaDeclaraciones(0).ah, Declaracion.a)

RListaDeclaraciones(0).a = RListaDeclaraciones(1).a

RListaDeclaraciones ::= λ

RListaDeclaraciones.a = RListaDeclaraciones.ah

ListaParametrosFormales ::= ParametroFormal RListaParametrosFormales

RListaParametrosFormales.ah = **listaParametrosFormalesR2**(ParametroFormal.a)

ListaParametrosFormales.a = RListaParametrosFormales.a

RListaParametrosFormales ::= , ParametroFormal RListaParametrosFormales

RListaParamFormales(1).ah = **listaParametrosFormalesR1**(RListaParamFormales(0).ah, ParamFormal.a)

RListaParametrosFormales(0).a = RListaParametrosFormales(1).a

RListaParametrosFormales ::= λ

RListaParametrosFormales.a = RlistaParametrosFormales.a

Instrucciones ::= Instruccion RInstrucciones

RInstrucciones.ah = **instruccionesR2**(Instruccion.a)

Instrucciones.a = RInstrucciones.a

RInstrucciones ::= , Instruccion RInstrucciones

RInstrucciones(1).ah = **instruccionesR1**(RInstrucciones(0).ah, Instruccion.a)

RInstrucciones(0).a = RInstrucciones(1).a

RInstrucciones ::= λ

RInstrucciones.a = Rinstrucciones.ah

ListaParametrosReales ::= Exp0 RListaParametrosReales

RListaParametrosReales.ah = **listaParametrosRealesR2**(Exp0.a)

ListaParametrosReales.a = RListaParametrosReales.a

RListaParametrosReales ::= , Exp0 RListaParametrosReales

RListaParametrosReales(1).ah = **listaParametrosRealesR1**(RListaParametrosReales(0).ah, Exp0.a)

RListaParametrosReales(0).a = RListaParametrosReales(1).a

RListaParametrosRealies ::= λ

RListaParametrosReales.a = RListaParametrosReales.ah

Exp0 ::= Exp1 RExp0

RExp0.ah = Exp1.a

Exp0.a = RExp0.a

RExp0 ::= OpComparacion Exp1

RExp0.a = **exp0R1**(RExp0.ah, OpComparacion.a, Exp1.a)

RExp0 ::= λ

RExp0.a = **exp0R2**(RExp0.ah)

Exp1 ::= Exp2 RExp1

RExp1.ah = **exp1R2(**Exp2.a)

Exp1.a = Rexp1.a

RExp1 ::= OpAditivo Exp2 RExp1

RExp1(1).ah = **exp1R1**(RExp1(0).ah, OpAditivo.a, Exp2.a)

RExp1(0).a = RExp1(1).a

RExp1 ::= λ

RExp1.a = RExp1.ah

Exp2 ::= Exp3 RExp2

RExp2.ah = **exp2R2**(Exp3.a)

Exp2.a = RExp2.a

RExp2 ::= OpMultiplicativo Exp3 RExp2

RExp2(1).ah = **exp2R1**(RExp2(0).ah, OpMultiplicativo.a, Exp3.a)

RExp2(0).a = RExp2(1).a

RExp2 ::= λ

RExp2.a = RExp2.ah

4. DESCRIPCIÓN DE LAS RESTRICCIONES CONTEXTUALES

Asignación correcta

Parámetros de entrada

Tabla de símbolos; `TS`

Identificador; `iden`

Tipo del identificador; `tipo`

Restriccones comprobadas

El tipo esperado (pasado como argumento) coincide con el tipo encontrado para ese identificador en la tabla de símbolos; ambos son de tipo numérico.

Condición correcta

Parámetros de entrada

Tipo que queremos comprobar; `iden`

Restricciones comprobadas

El tipo a comprobar es de tipo numérico (no es ni error, ni procedimiento)

Existe símbolo en último Nivel

Parámetros de entrada

Tabla de símbolos; `TS`

Identificador; `iden`

Restricciones comprobadas

El símbolo (iden) no se había declarado anteriormente en el mismo nivel de la TS. Devolverá error en caso de símbolo repetido.

Escritura correcta

Parámetros de entrada

Tabla de símbolos; `TS`

Identificador; `iden`

Restricciones comprobadas

El tipo de iden es numérico (no es un procedimiento)

Llamada correcta

Parámetros de entrada

Tabla de símbolos; `TS`

Identificador; `iden`

Número de parámetros esperados; `nParametros`

Restricciones comprobadas

El símbolo es de tipo procedimiento y además, el número de parámetros registrados para ese procedimiento en la TS coincide con el número de parámetros esperados

Parámetro correcto

Parámetros de entrada

Nombre del procedimiento; `nombreProc`

Número del parámetro que se quiere comprobar; `parametro_i`

Tipo de parámetro iésimo; `tipoParametro`

Booleano que indica si el parámetro es un designador; `esDesignador`

Tabla de símbolos; `TS`

Restricciones comprobadas

- El identificador hace referencia a un procedimiento.
- Existe el parámetro iésimo en la declaración del procedimiento
- Si el parámetro a chequear está declarado como parámetro por referencia, tiene que ser llamado con un designador.
- El tipo del parámetro es númerico (no es ni error, ni procedimiento)

5. DESCRIPCIÓN DE LA SÍNTESIS DE TIPOS REALIZADA

En el lenguaje considerado en la práctica sólo disponemos de tres tipos distintos: númerico, procedimiento, y error.

Tipo Op Binario

Entrada

Tipo del primer operando

Tipo del segundo operando

Tipo de devuelto

Si alguno de los tipos de entrada es error devolverá Tipo error.

Si alguno de los tipos de entrada es un procedimiento devolverá error; en otro caso devolverá tipo numérico.

Tipo Op Unario

Entrada

Tipo del operando

Tipo devuelto

Numérico si el tipo de entrada lo era, y error en cualquier otro caso.

6. INTRUCCIONES GENERADAS POR LAS FUNCIONES

Activación del programa principal

Parámetros de entrada

Tamaño de los datos; `tamDatos`

Dirección de inicio del programa; `dirInicio`

Nivel de anidamiento; `anidamiento`

Lista de instrucciones generada:

Fijar el valor de CP

```
apila(anidamiento +1 + tamDatos + 2)
desapila_dir(0)
```

Actualizar display

```
apila(1 + anidamiento + 2 + 1)
desapila_dir(1)
```

Comenzar la ejecución del programa

```
ir_a(dirInicio)
```

Código de acceso a una variable

Parámetros de entrada

Tabla de Símbolos; `TS`

Identificador; `iden`

Consultas Auxiliares a la Tabla de símbolos

Nivel del identificador; `nivel`

Dirección del identificador; `direccion`

Booleano que determina si la clase del símblo es PVAR; `es_PVAR`

Lista de instrucciones generada

```
apila_dir(nivel + 1)
apila(dir);
suma();
si (es_PVAR) => apila_ind()
```

Codigo de asignación de una variable (escritura en memoria)

Parámetros de entrada

Tabla de símbolos; `TS`

Identificador; `iden`

Listado de instrucciones máquina pre-generado; `cod`

Booleano que determina si el símbolo es un designador; `es_Designador`

Consultas auxiliares a la tabla de símbolos

Nivel del identifcador; `nivel`

Dirección del identificador; `dir`

Booleano que determina si la clase del símbolo es PVAR; `es_PVAR`

Lista de instrucciones generada

Cálculo de la direccion de la Variable

```
apila_dir(nivel + 1)
apila(dir)
suma()
si(es_PVAR) => apila_ind()
```

Obtener el valor de la variable

```
concatenarInstrucciones(cod)
si(esDesinador) => apila_ind()
```

Escritura en memoria

```
desapila_ind()
```

Codigo de Escritura de una variable (por consola)

Parámetros de entrada

Tabla de símbolos; `TS`

Identificador; `iden`

Consultas auxiliares a la tabla de símbolos

Nivel del identifcador; `nivel`

Dirección del identificador; `dir`

Booleano que determina si la clase del símbolo es PVAR; `es_PVAR`

Lista de instrucciones generada

Acceso a la dirección

```
apila_dir(nivel + 1)
apila(dir)
suma()
si(es_PVAR) => apila_ind()
```

Acceso al valor y llamada a la función de escritura

```
apila_ind()
escribe()
```

Operación de comparación, operación aditiva, operación multiplicativa

***Nota: las tres funciones generan la misma lista de instrucciones**

Parámetros de entrada

Lista de instrucciones correspondiente al primer operando; `cod_e1`

Lista de instrucciones correspondiente al segundo operando; `cod_e2`

Lista de instrucciones correspondientes al operador; `cod_op`

Booleano que indica si el primer operando es un designador; `es_Designador1`

Booleano que indica si el segundo operando es un designador; `es_Designador2`

Lista de Instrucciones generada

```
concatenarInstrucciones(cod_e1)
si(esDesignador_1) => apila_ind()
concatenarInstrucciones(cod_e2)
si(esDesignador_2) => apila_ind()
concatenarInstrucciones(cod_op)
```

Operación con operador unario

Parámetros de entrada

Lista de instrucciones correspondiente al operando; `cod_e1`

Lista de instrucciones correspondientes al operador; `cod_op`

Booleano que indica si el operando es un designador; `es_Designador1`

Lista de instrucciones generada

```
concatenarInstrucciones(lista, cod_e);
si(esDesignador_e) => concatenarInstruccion(lista, apila_ind());
concatenarInstrucciones(lista, cod_op);
```

Código para generar el prólogo de una función

Parámetros de entrada

Tamaño de los datos, `tamDatos`

Nivel; `nivel`

Lista de instrucciones generada

Almacenar el antiguo display

```
apila_dir(0)
apila(2)
suma()
apila_dir(nivel + 1)
desapila_ind()
```

Fijar el nuevo display

```
apila_dir(0)
apila(3)
suma()
desapila_dir(nivel + 1)
```

Actualizar el CP

```
apila_dir(0)
apila(tamDatos + 2)
suma()
desapila_dir(0)
```

Código para generar el epílogo de una función

Parámetros de entrada

Tamaño de los datos, `tamDatos`

Nivel; `nivel`

Lista de instrucciones generada

Restaurar CP

```
apila_dir(nivel + 1)
apila(2)
resta()
apila_ind()
```

Restaurar Display

```
apila_dir(nivel + 1)
apila(3)
resta()
copia()
desapila_dir(0)
```

Recuperar Direccion de retorno

```
apila(2)
suma()
apila_ind()
desapila_dir(nivel + 1)
salto_ind()
```

Código para llamar a un procedimiento

Lista de Instrucciones generada

```
apila_dir(0)
apila(3)
suma()
```

Código para ejecutar el fin de una llamada a procedimiento

Parámetros de entrada

Nombre del subprograma; `nombreProcedure`

Tabla de símbolos; `TS`

Dirección de retorno; `dirRetorno`

Consultas auxiliares a la tabla de símbolos

Dirección de comienzo del procedure; `dirComienzo`

Lista de instrucciones generada

```
apila_dir(0)
apila(1)
suma()
apila(direccionRetorno)
desapila_ind()
salto(dirComienzo)
```

Código para efectuar el paso de una variable

Parámetros de entrada

Nombre del subprograma; `nombreProcedure`

Posición del parámetro en la lista de parametros; `numParametro`

Booleano que indica si el parámetro es un designador; `es_Designador`

Tabla de símbolos; `TS`

Consultas auxiliares a la tabla de símbolos

Lista de Parámetros del subprograma

Booleano que indica si el parámetro es pasado por valor; `por_valor`

Lista de Instruccions generada

```
si(esDesignador && por_valor) => apila_ind()
desapila_ind()
apila(1)
suma()
```

7. AÑADIDO: EXPLICACIÓN DE LOS PROGRAMAS PROBADOS

Programas básicos (declaración, asignación y escritura)

Primeras pruebas realizadas para comprobar el correcto funcionamiento de la tabla de símbolos al declarar variabes; instrucciones de acceso a estas variables, asignación de valores, operaciones numéricas básicas y escritura por consola.

Programa A-1

Comprubea

Registro correcto de variable, instrucción de asignación, escritura de variable

Código

```
decs
	entero
end decs
body
	set entero to 103,
	out entero
end body
```

Programa A-2

Comprueba

Declaración de varias variables, operaciones básicas, asignación a otro designador

Código

```
decs
	x,y,z
end decs
body
	set x to 3,
	set y to x + 1,
	set z to x * y,
	out x,
	out y,
	out z
end body
```

Programa A-no-1

Comprueba

Error contextual esperado al acceder a una variable no esperada

Código

```
body
	set x to c,
	out x
end body
```

Programa A-no-2

Comprueba

Error contextual esperado al hacer una asignación incorrecta

Código

```
decs
	x
end decs
body
	set x to +
end body
```

Instrucciones condicionales

Comrpobar el correcto funcionamiento de las instrucciones condicionales así como de los operadores con caráter “booleano”

Programa B-1

Comprueba

Ejecución de parte if de las instrucciones condicionales, ejecución de la parte else, operadores and not y or

Código

```
decs
	or_ejecutado, and_ejecutada,
	or_no_ejecutado, and_no_ejecutada
end decs
body
	set or_ejecutado to 1,
	set or_no_ejecutado to 2,
	set and_ejecutada to 3,
	set and_no_ejecutada to 4,
	if(1 or 0) then {
		out or_ejecutado
	} else {
		out or_no_ejecutado
	},
	if(1 and 0) then {
		out and_ejecutada
	} else {
		out and_no_ejecutada
	}
end body
```

Programa B-2

Comprueba

Ejecución de instrucciones condicionales anidadas y operadores “<” “>” introducidos

Código

```
decs
	x
end decs
body
	set x to 7,
	if (x = 6) then
	{
		set x to x + 1,
		out x
	}
	else
	{
		set x to 8 + 2,
		out x,
		if (x > 9) then
		{
			if (x < 100) then
			{
				set x to x + x,
				out x
			}
		}
	}
end body
```

Programa B-no-1

Comprueba

Error sintáctico esperado en la construcción del árbol de una instrucción “if”

Código

```
decs
	x,y
end decs
body
	set x to 1,
	set y to 2,
	if(x = y) then
	else
	{
		out x
	}
end body
```

Llamadas a procedimientos (con parámetros por valor y referencia)

Comprobar el funcionamiento de llamadas a subprogramas, contrucción correcta de talba de símbolos de varios niveles, y comportamiento esperado de parámetros por valor o por referencia.

Programa C-1

Comprueba

Llamada correcta a un procedimiento sin parámetros

Código

```
decs
	proc imprimir_tres()
	decs
		x
	end decs
	body
		set x to 3,
		out x
	end body
end decs
body
	call imprimir_tres end call
end body
```

Programa C-2

Comprueba

Llamada correcta a un procedimiento con un parámetro por valor, llamada con un designador y con un valor numérico

Código

```
decs

	x,
	proc imprimir_variable(entero)
	body
		out entero
	end body
end decs
body
	set x to 5,
	call imprimir_variable with x end call,
	call imprimir_variable with 5 end call
end body
```

Programa C-3

Comprueba

Llamada a subprograma con variable por referencia modificando su valor al salir de la rutina

Código

```
decs
        x,
        proc aumentar_variable(var x)
        body
                set x to x+1
        end body
end decs
body
        set x to 1,
        out x,
        call aumentar_variable with x end call,
        out x
end body
```

Programa C-no-1

Comprueba

Restricción contextual al intentar llamar a un procedimiento con un número de parámetros incorrecto

Código

```
decs
        x, y,
        proc f(a)
        body
                out a
        end body
end decs
body
        call f with x,y end call
end body
```

Programa C-no-2

Comprueba

Restricción contextual al intentar pasar un valor numérico como parámetro real a un parámetro declarado por referencia

(*)Programa C-no-2

Código

```
decs
        x,
        proc f(x, var y)
        body
                set y to x,
                out y
        end  body
end decs
body
        call f with x,3 end call
end body
```

Programa C-no-3

Comprueba

Error contextual al intentar llamar a un procedimiento con un parámtero de tipo proc.

Código

```
decs
        proc g(a)
        body
                out a
        end body,
        proc f()
        decs
                x
        end decs
        body
                set x to 3
        end body
end decs
body
        call g with f end call
end body
```

Llamadas a procedimientos (pruebas más elavoradas)

Programa D-1

Comprueba

Llamadas a procedimientos anidadas, creando varios niveles de TS y llamando a subprogramas dentro de subprogramas, (función 1 => función 2 => función 3)

(*) Programa D-1

Código

```
decs
        x,
        proc funcion_uno(a)
        decs
                proc funcion_dos(a)
                decs
                        proc funcion_tres(a)
                        body
                                out a
                        end body
                end decs
                body
                        call funcion_tres with a end call
                end body
        end decs
        body
                call funcion_dos with a end call
        end body
end decs
body
        set x to 9,
        call funcion_uno with x end call
end body
```

Programa D-2

Comprueba

Llamada a procedimientos con varios parámetros, declaración de mismo símbolo a diferntes niveles de la TS

Código

```
decs
        x,y,z,
        proc f(a,b,c)
        decs x end decs
        body
                set x to a * b * c,
                out x
        end body,
        proc g(a, var b)
        decs x end decs
        body
                set b to x / b
        end body
end decs
body
        set x to 1, set y to 2, set z to x - y,
        call g with 1, x end call,
        if( x < 0 ) then {
                out x,
                call f with x,y,z end call
        }
end body
```

Pruebas finales (Fibonacci y factorial de un número)

Objetivo:

Comprobar el correcto funcionamiento de las llamadas recursivas a subprogramas en dos algoritmos clásicos no inventados: el factorial de un número y la sucesión de Fibonacci

Factorial

```
decs
	proc factorial(x, acumulado)
	decs
		uno
	end decs
	body
		set uno to 1,
		if (x=0) then
		{
			out uno
		}
		else
		{
			if(x = 1) then
			{
				out acumulado
			}
			else
			{
				set acumulado to x * acumulado,
				call factorial with x-1, acumulado end call
			}
		}
	end body,
	x,
	uno
end decs

body
	set uno to 1,
	set x to 30,
	call factorial with x, uno end call
end body
```

Fibonacci

```
decs
	x,
	resultado,
	proc fibonacci(num, var r)
	decs
		aux1, aux2
	end decs
	body
		if (num < 2) then
		{
			set r to 1
		}
		else
		{
			call fibonacci with num-1, aux1 end call,
			call fibonacci with num-2, aux2 end call,
			set r to aux1+aux2
		}
	end body
end decs

body
	set x to 6,
	call fibonacci with x,resultado end call,
	out resultado
end body
```