Compiladors I

Enginyeria en Informàtica

Pràctica de l'assignatura

Autors: Sergio Blanco Cuaresma
Francisco José Aguilar Celdrán

Curs: 2004 / 2005

Índex de Continguts

1.Descripció general

L'objectiu d'aquesta pràctica consisteix en desenvolupar la part d'anàlisi d'un compilador de MiniAda. El llenguatge MiniAda està descrit en el l'apartat d'annexos.

Informació d'entrada i sortida del compilador

- Arxius d'entrada:
  - Arxiu que conté un programa escrit en llenguatge MiniAda (d'extensió ".ma").
- Arxius de sortida:
  - Arxiu amb tota la informació de la Taula de Símbols (d'extensió ".ts").
  - Altres sortides (per la sortida estàndard, arxius de missatges, etc).

Arxiu amb la Taula de Símbols

Conté la informació de les taules de símbol corresponents a tots els subprogrames. Està estructurat de la següent manera:

- Per a cada subprograma es posa una capçalera en la que s'indiquen clarament el seu nom (si és que en té) i el seu nom únic.
- A continuació es posa el contingut de la taula de símbols corresponent a aquest subprograma: profunditat d'encaixament, variables, constants, tipus, procediments, funcions, desplaçaments, etc.
- No es pot interrompre la taula de símbols d'un subprograma per a intercalar la taula de símbols d'un altre subprograma (si cal, es pot generar un arxiu per a cada subprograma, i després es fa la concatenació de tots ells).

Altres Sortides

Si és necessari, es poden crear altres arxius, o generar missatges per la sortida estàndard:

- Sortida estàndard: s'ha d'intentar minimitzar la informació que surt per aquesta via (missatges d'arxiu no trobat, d'error de compilació de tipus lèxic, sintàctic o semàntic, etc.).
- Altres arxius: pot ser interessant generar arxius de missatges del compilador, tant per a veure el progrés de la compilació com per a facilitar la depuració del mateix (missatges de tokens reconeguts, insercions a la taula de símbols, aparicions de noms no locals, entrades i sortides en nous àmbits, etc.).

Requisits pel compilador de MiniAda

El compilador de MiniAda ha de ser capaç de fer, com a mínim, el següent:

- Anàlisi lèxica i sintàctica de qualsevol programa escrit en MiniAda.
- Utilització d'una pila de taules de símbols (o estructura equivalent) per a tenir en compte la possible presència d'àmbits encaixats, preferiblement implementades com a taules de dispersió (taules de

hash).

- Comprovació de tipus en totes les expressions que calgui: aritmètiques, lògiques, paràmetres, valors de retorn, etc.
- Conversions de tipus explícites.

Les següents característiques són opcionals:

- Comprovació dels subtipus i dels rangs de les taules.
- Possibilitat de fer sobrecàrrega de procediments i funcions.
- Tractament dels agregats.
- Tractament d'apuntadors i de la creació dinàmica d'objectes.

Per veure la descripció detallada del llenguatge MiniAda i les característiques semàntiques, sintàctiques i lèxiques a tenir en compte, consultar la documentació adjunta corresponent.

2.Disseny i Arquitectura de l'aplicació

L'aplicació desenvolupada pretén cobrir les funcionalitats necessàries per tal d'assolir la compilació de qualsevol programa escrit en llenguatge MiniAda. L'etapa de compilació inclou la generació de codi objecte corresponent al programa en codi font que es tracta durant l'execució de l'aplicatiu.

En aquest capítol es detalla breument una visió general de l'arquitectura aplicada per al disseny del compilador de MiniAda.

El desenvolupament combina l'execució sincronitzada d'un analitzador lèxic, encarregat d'obtenir els diferents tokens del fitxer de codi font tot identificant quin tipus d'element són, un analitzador sintàctic, encarregat de validar la correctesa sintàctica de totes les estructures, sentències, àmbits, etc. del codi font. L'analitzador lèxic s'ha implementat amb l'eina Flex i per a l'analitzador sintàctic s'ha utilitzat l'eina Bison.

L'analitzador sintàctic conté totes les regles corresponents a la definició del llenguatge MiniAda, i s'encarrega de cridar a l'analitzador lèxic per obtenir els diferents tokens que són identificats per lèxic. El lèxic realitzarà comprovacions de correctesa lèxica de cadascun dels tokens, i el sintàctic en validarà l'estructura de les diferents expressions i definicions que conformen l'associació dels diferents tokens, tot utilitzant gramàtiques independents del context i generant un arbre sintàctic amb les diferents regles definides.

L'analitzador sintàctic defineix a més tota la validació a efectes d'anàlisi semàntica, per tal d'aplicar totes les comprovacions de tipus que permetin disposar d'un compilador el més eficient possible.

La coordinació de l'analitzador lèxic amb el sintàtic es realitza per mitjà de la variable global yylval. Per poder emmagatzemar tots els valors tractats en la compilació, s'utilitza una Tipus Abstracte de Dades anomenat Taula de Símbols, que serà comentat en el respectiu capítol, així com els analitzadors i la generació de codi objecte.

La implementació dels analitzadors s'efectua sobre els fitxers lexic.l i sintactic.y. La taula de símbols s'incorpora en els fitxers taula.c i taula.h. Es disposa de estructures de dades addicionals utilitzades al llarg del compilador: pila i cua, i que es troben implementades en el els fitxers taula.c, taula.h, cua.c i cua.h. Existeix a més un arxiu global.h que conté definicions comunes d'àmbit global a tot l'aplicatiu.

L'inici del programa es realitza des del fitxer adac.c. En ell es rep l'arxiu amb el programa de MiniAda, s'inicialitzen els analitzadors lèxic i sintàctic, i es procedeix a executar l'analitzador sintàctic, que actua de director de l'execució. Una vegada conclou el correcte processament de la compilació, es continua l'execució en el fitxer principal adac.c, en el que s'imprimeixen, per finalitzar, totes les dades relacionades amb la taula de símbols.

S'imprimeixen durant la compilació, a més, un log de tots els tokens processats pel lèxic i un log amb totes les regles tractades pel sintàctic. Aquests logs no tenen cap valor per a l'usuari final del compilador, però sí han estat de gran ajuda per als desenvolupadors d'aquest compilador de MiniAda en

el seu procés de disseny, implantació, depuració i testeig.

La impressió dels tres arxius de sortida s'implementa en fitxers separats, donada la complexitat del seu tractament: print_ts.c, print_ts.h, print_c3a.c, print_c3a.h, print_ra.c i print_ra.h.

En cas de produir-se algun error de compilació, ja sigui lèxic, sintàctic o semàntic, s'atura la compilació i es mostra el conseqüent missatge informatiu d'advertència de l'error, tot adjuntant el número de línia i columna on s'ha produït, per tal de facilitar la ràpida detecció de l'errada per part del programador.

Finalment, esmentar que el compilador inclou un arxiu Makefile que permet aplicar una senzilla compilació del codi font del compilador de MiniAda, obtenint finalment l'arxiu executable “adac”, el qual rebrà per línia de comandes nom del fitxer de codi font que es desitja compilar.

3.Taula de Símbols

L'estructura utilitzada pel compilador per emmagatzemar la tota la informació relativa a un símbol (qualsevol identificador d'un programa que faci referència a una variable, tipus, funció, procediment, etc) s'anomena Taula de Símbols.

Per tal de gestionar la evolució de la compilació s'emmagatzema tota la informació que es va obtenint del lèxic i que és processada posteriorment pel sintàctic en una estructura de dades complexa, la taula de símbols, on es recopila tota la informació d'un símbol obtinguda a partir de l'anàlisi del codi font i que és necessària per realitzar la compilació de forma satisfactòria.

La implementació de la taula de símbols s'ha realitzat utilitzant una taula de hash de tamany prefixat que emmagatzema els diferents símbols en la posició corresponent al valor obtingut per una funció de dispersió que calcula un valor numèric natural amb mòdul el tamany de la taula de hash. Aquest valor numèric s'utilitza per indexar el símbol en la posició indicada pel càlcul. En el cas particular d'aquesta implementació, s'ha definit una taula de hash de 100 posicions amb una funció de dispersió que aplica càlcul de codis ASCII del nom del símbol que es pretén localitzar en la taula.

Per poder discriminar les possibles col·lisions que es provocarien per coincidència d'índex com a valor de dispersió, cada posició de la taula de símbols conté no conté un símbol sinó un Tipus Abstracte de Dades Cua, que associa tots els símbols coincidents en una posició de la taula, emmagatzemant en cada posició de la cua una estructura del tipus Simbol.

A continuació es fa un esment per concretar la forma de tractar els diferents àmbits d'un programa de MiniAda. S'utilitza una taula de símbols diferent per a cada àmbit del programa. En concret, hi haurà una taula de símbols per a cada procediment, funció o per a les dades d'un record. Aleshores, cal mantenir un lligam entre totes les taules de símbols, calent associar una taula de àmbit fill amb una taula precedent d'un àmbit pare que inclogui a l'àmbit fill. Per tant, existeix una pila de taules de símbols. Es manté una variable global que referència la taula de símbols tractada actualment. Cal emmagatzemar, a més, el nom de la taula de símbols per tenir-ne un identificador. En definitiva, l'estructura principal de la taula de símbols és la següent:

```
//* Estructura de la taula de hash. */
typedef struct Taula_Simbols {
        Cua *taula[TAM_TAULA_SIMBOLS];
        struct Taula_Simbols *anterior;
        char nom[STRING_LENGTH];
        struct Taula_Simbols *programa_principal;
} Taula_Simbols;
```

Cal indicar que la taula de símbols principal, que correspon a l'àmbit del procedure que conté el tot el cos del programa principal de l'aplicació de codi font compilada, no és la taula de símbols pare a la qual totes les demés apunten, sinó que es troba en un segon nivell de l'arbre jeràrquic, tenint per encara per damunt una altra taula de símbols, la inicial. Aquesta taula principal té la seva raó d'ésser en el fet que cal disposar de certes funcions predefinides en el llenguatge, així com diversos tipus bàsics. En concret, aquesta taula principal inclou, a més de la taula de símbols pare del programa compilat, els símbols

corresponents a:
- Funcions predefinides en el llenguatge:
  - function Get_Integer return Integer;
  - function Get_Float return Float;
  - function Get_Character return Character;
  - function Get_String return String;
  - **function** Length(Item: String) **return** Integer;
- Procediments definits en el llenguatge:
  - procedure Put(Item: Float);
  - procedure Put(Item: Integer);
  - procedure Put(Item: Character);
  - procedure Put(Item: String);
    (Es suporta sobrecàrrega de procediments i funcions per tipus i nombre de paràmetres, no per a dreça de retorn).
- Tipus de dades predefinides en el llenguatge:
  - Boolean.
  - Character.
  - Integer.
  - Float.
  - String.
- Valors predefinits (emmagatzemats com a variables) de tipus Boolean amb valor 0 i 1:
  - False.
  - True.

Tot seguit es passa a descriure el contingut de l'estructura Símbol de la que està composada cada cua d'elements de cada posició de la taula de símbols.

L'estructura Símbol està formada per una cadena de caràcters indicant el nom i una estructura de tipus Atributs que en conté tota la informació. La estructura Atributs conté diversos camps, el primer és de tipus Tipus_Simbol (indicant la classe de símbol) i el darrer és de tipus Continguts (que inclou les dades del símbol).

```
typedef struct Simbol {
        char nom[STRING_LENGTH];
        Atributs atributs;
} Simbol;

typedef struct {
        Tipus_Simbol tipus;
        Continguts continguts;
} Atributs;
```

L'estructura Tipus_Simbol és un enumerat que identifica el tipus de símbol, amb els possibles valors: variable, type, record_type, array_type, enumerated_type, procedure i function.

L'estructura Continguts és una union de C que conté informació diferent segons el tipus de símbol que s'estigui tractant, per tal de minimitzar l'espai de memòria consumit per la taula de símbols. Els possibles tipus de continguts són les següents estructures: Info_Type, Info_Variable, Info_Procedure, Info_Function, Info_Enumerated, Info_Array i Info_Record. Cadascuna d'aquestes estructures conté la

informació relativa al tipus de símbol concret.

El tipus Info_Procedure conté la següent informació per manegar totes les dades d'un procediment: una taula de símbols amb tots els símbols del seu àmbit, una cua amb els paràmetres del procediment, un booleà que indica si únicament s'ha definit o també es disposa del cos del procediment al codi font, un comptador de número de sobrecàrregues, cua de estructures Info_Procedures amb tantes entrades com procediments sobrecarregats hi hagi, un identificador de sobrecàrrega de procediment i un nom únic que identifica la funció unívocament contra qualsevol altra funció amb el mateix nom i en un àmbit no visible per a la funció actual.

```
typedef struct Info_Procedure {
        int id;                         // Identificador de les diferentes sobrecarregues
        int num_sobrecarregues;    // Numero total de sobrecarregues (només inicialitzat al info_procedure principal)
        struct Taula_Simbols *taula;
        Cua *parametres;  // Cua d'strings amb els IDs corresponents als parametres (els simbols es troben a la taula)
        int num_parametres;
        Boolean complet;                // Definicio i cos complet del procedure ja indicat?
        Cua *sobrecarregues;            // Cua d'info_procedures de sobrecarregats
} Info_Procedure;
```

La estructura Info_Function és similar a la de Info_Procedure, però inclou a més el tipus de retorn i el nom del tipus de retorn, per a tipus compostos.

```
typedef struct Info_Function {
        int id; // Identificador de les diferentes sobrecarregues
        int num_sobrecarregues;     // Numero total de sobrecarregues (només inicialitzat al info_function principal)
        struct Taula_Simbols *taula;
        Cua *parametres;  // Cua d'strings amb els IDs corresponents als parametres (els simbols es troben a la taula)
        int num_parametres;
        Tipus_Dades tipus_retorn;
        char nom_tipus_retorn[STRING_LENGTH]; // Utilitzat per variables que no son de tipus predefinits (integer,
float...) i per elements de enumerats
        Boolean complet; // Definicio i cos complet de la funcio ja indicat?
        Boolean return_indicat;              // S'ha ficat un return al cos de la funció?
        Cua *sobrecarregues;                 // Cua d'info_functions de sobrecarregats
} Info_Function;
```

Els tipus de dades simples es reconeixem amb l'enumerat Tipus_Dades, que conté: Tipus_Integer, Tipus_Integer_Derivat, Tipus_Float, Tipus_Character, Tipus_String i Tipus_Derivat.

El tipus de dades Rang, indica el valor mínim i el valor màxim d'un rang així com el tipus de dades dels valors que delimiten el rang.

L'estructura Info_Array conté un camp indicant el tipus de dades dels elements de l'array, el nom del tipus (per a tipus que no són predefinits), i el rang de l'array.

L'estructura Info_Record està formada per una taula de símbols que agrupa tots els símbols corresponents als diferents camps del registre en qüestió.

Pel que fa a l'estructura Info_Enumerated, emmagatzema una cua amb simbols de tipus variable (definits com a constants) per a cadascun dels elements de l'enumerat.

El tipus de dades Info_Type guarda la informació d'un tipus: s'indica el tipus de dades de què està

format el tipus i el rang del tipus.

Finalment, l'estructura Info_Variable registra les dades de un simbol corresponent a una variable:

```
typedef struct Info_Variable {
        Tipus_Dades tipus;
        Boolean constant;
        Boolean literal;
        Boolean inicialitzat;
        Valor valor;
        char nom_tipus[STRING_LENGTH]; // Utilitzat per variables que no son de tipus predefinits (integer, float...)
                                       // i per elements de enumerats
} Info_Variable;

typedef union {
        Boolean boolean;
        int integer;
        char string[CONTENIDOR_STRING_LENGTH];
        char character;
        double vfloat;
        struct Taula_Simbols *taula; // Si la variable es del tipus RECORD, aquí es guarden les variables del RECORD
} Valor;
```

Respecte a les funcionalitats incorporades per a operar amb la taula de símbols, entre altres, es poden efectuar les operacions de inserció d'un símbol en la taula de símbols actual, obtenir la taula de símbols precedent d'àmbit superior o cercar i recuperar un determinat símbol localitzat pel nom, ja sigui en la taula de símbols actual o bé consultant totes les taules apilades fins a la taula de símbols inicial del compilador on es troben incloses tots el tipus i totes les funcions predefinides pel llenguatge, i la taula de símbols del programa principal.

La impressió de tots els valors de la taula de símbols es realitza una vegada acabada la correcta execució del compilador. S'utilitza la funció print_taula_simbols implementada en l'arxiu print_ts.c. Aquesta funció s'executa recursivament per tal de mostrar les dades de tots els àmbits, i mostra tota la informació de qualsevol tipus de símbol (tipus simple o compost, variable, funció, procediment, etc) en un arxiu de sortida de la compilació. La recursivitat permet tractar tots els àmbits anidats. A més, cal destacar que el disseny de la funció ha tingut en compte el fet de no intercalar símbols d'un àmbit / taula de símbols per mostrar els símbols d'un altre àmbit / taula de símbols, amb la qual cosa primer es mostren tots els símbols existents en un àmbit i seguidament es mostren els àmbits encaixats en l'actual, aconseguint d'aquesta manera una impressió d'àmbits elegant i entenedora. Aquesta funció rep inicialment la taula de símbols inicial (pare) amb la qual cosa es mostren tant les funcions i tipus predefinits com tota la informació a partir de l'àmbit principal del programa del codi font.

4.Anàlisi Lèxica

L'anàlisi lèxica permet al compilador obtenir els diferents tokens que seran tractats en el sintàctic. Segons la estructura del compilador, el lèxic, que s'ha implementat amb Flex, és cridat per l'analitzador sintàctic cada vegada que aquest requereix un nou token per a tractar.

L'eina Flex permet la implementació de l'analitzador lèxic per mitjà de l'especificació d'expressions regulars que permetin identificar tots el possibles tipus d'elements correctes a llegir en un programa de codi font. Aquest fitxer té extensió .l y posteriorment és processat pel generador Flex que obté un codi font en llenguatge C amb els corresponents arxius .c, i .h que inclouen tota la implementació de les expressions regulars definides. D'aquesta forma s'agilitza el desenvolupament de les regles lèxiques del compilador.

Cal remarcar que, pel que fa al tractament dels strings, s'ha utilitzat la tècnica de Condicions d'Arranc, amb la qual es pot validar la finalització correcta del string o bé la finalització incorrecta si no s'ha establert la cometa doble com a darrer caràcter abans del salt de línia. A continuació es mostra, a tall d'exemple, la implementació d'aquesta condició d'inici:

```
\" { BEGIN(STRING_LITERAL); }

<STRING_LITERAL>[^\n"]*
{ if (yyleng > STRING_LENGTH) {
     lexError("String massa llarg, com a maxim poden haver-hi '%i'
               caracters", STRING_LENGTH);
  }
  else {
     colNumber += yyleng;
     sprintf(yylval.string, "%c%s", strlen(yytext), yytext);
     return PR_STRING_LITERAL;
  }
}

<STRING_LITERAL>\"   { BEGIN(INITIAL); }

<STRING_LITERAL>\n
{
   lexError("Error: Salt de linia abans de finalitzacio de String. (Linia: %d, Columna: %d)\n",
             lineNumber, colNumber);
}
```

Quan l'analitzador obté tokens de tipus integer o de tipus float, en comprova el valor per validar que estigui dintre del rang permès. El rang per als enters està comprès entre -2147483648 i 2147483647. En rang per als floats es trova entre els valors -1.7E-308 i 1.7E+308. Es permet la introducció de floats amb notació científica, i, per altra banda, per a tots els tipus numèrics es permet la introducció de valors amb el caràcter “_” separant cadascun dels dígits, tal i com indiquen les especificacions lèxiques per al compilador de MiniAda.

Totes les paraules reservades són tractades directament en el lèxic, de tal forma que quan l'analitzador associa una de les paraules reservades definides en el fitxer de Flex amb un dels Tokens obtinguts de fitxer s'avisa directament a l'analitzador sintàctic retornant el tipus de paraula reservada trobada. Això permet optimitzar el rendiment del compilador, tot i que també podrien ser incorporades en la taula de símbols i consultar aquesta quan es revés un identificador, per veure si es tracta de una paraula

reservada.

De la mateixa manera, pel que fa a tots els símbols i operadors tant unaris com binaris del llenguatge, el lèxic els defineix directament en el fitxer de Flex, amb la qual cosa s'agilitza el rendiment del sistema. Cada operador té una codificació concreta de paraula reservada que és coneguda també per l'analitzador sintàctic i permet a aquest darrer establir fàcilment els tokens reservats dins les regles sintàctiques.

En el tractament de tokens identificadors, es cerca en totes les taules de símbols si existeix un símbol prèviament definit amb el nom de l'actual identificador. En cas de no existir, es crea un nou símbol que s'afegeix en la taula de símbols actual. Si existeix el símbol corresponent a l'actual identificador es retorna el tipus d'identificador, de tipus, variable, funció o procediment.

La comunicació entre l'analitzador lèxic i l'analitzador sintàctic es duu a terme per mitjà de la variable global yylval, la qual, com es veurà en el sintàctic, està definida com una estructura union de C, que pot contenir informació de diversos tipus segons el classe d'informació a comunicar. En el cas d'identificador, s'hi assigna un tipus Símbol, i en el cas de tokens literals, el respectiu valor en cada cas: float, integer, character, etc.

Es realitza un log de totes les accions dutes a terme per l'analitzador lèxic en un arxiu de sortida amb el nom del fitxer de codi font, afegint l'extensió ".lex.log". En ell es registren els diferents tipus de tokens tractats.

Els errors del lèxic són gestionats amb funció lexError i es finalitza la execució del programa en produir-se un error lèxic.

En el fitxer global.h es defineixen diverses constants d'ús comú en tot el compilador i que són utilitzades també per l'analitzador lèxic, com ara el tamany màxim d'un string.

Si el lèxic llegeix un símbol del codi font de MiniAda que no compleix cap de les regles indicades, es genera un error de tipus lèxic indicant que el caràcter introduït és desconegut, controlant, per tant, en tot moment la correctesa dels tokens tractats.

L'analitzador sintàctic manté un comptador de número de fila i un comptador de número de columna per tal de disposar de la posició exacta en la que es produeixi un error. Aquests comptadors són variables globals també accessibles per a l'analitzador sintàctic.

5.Anàlisi Sintàctica

L'anàlisi jeràrquica es denomina anàlisi sintàctica. Implica agrupar tots els components lèxics del programa font en frases gramaticals que el compilador utilitza per sintetitzar la sortida. Per regla general, les frases gramaticals del programa font es representen per mitjà d'un arbre d'anàlisi sintàctic. En aquesta pràctica s'ha emprat en generador d'analitzadors sintàctics Bison. La divisió entre l'anàlisi lèxica i l'anàlisi sintàctica és una mica arbitrària. Generalment es selecciona una separació que simplifiqui la tasca completa d'anàlisi.

L'analitzador sintàctic s'inicia per mitjà de la funció yyparse des del programa principal “adac.c”. Aleshores, l'analitzador porta el flux de control de l'execució del compilador i va tractant els diferents tokens que requereix l'analitzador lèxic (que roman a la espera de la indicació del sintàctic per proporcionar informació al sintàctic). Cadascun d'aquests tokens associarà per complir alguna de les regles sintàctiques definides en l'arbre sintàctic.

En finalitzar la correcta generació de l'analitzador sintàctic s'ha arribat a completar correctament la regla “program” que es l'arrel de l'arbre sintàctic. Aleshores, conclou satisfactòriament la compilació i es procedeix a mostrar la informació de la taula de símbols.

En el sintàctic es defineix una estructura de tipus union com a base per a la gestió de la informació manegada en una regla. A continuació se'n detallarà el funcionament.

L'analitzador lèxic retorna a l'analitzador sintàctic el tipus de token que s'ha tractat. Aquests tokens es detallaran tot seguit.

La variable global yylval permet que el lèxic retorni al sintàctic la informació associada al token llegit. En funció si és un string, integer, float, character, operador, paraula reservada, identificador, etc, retornarà un o altre tipus de valor, o no en retornarà cap (per exemple en el cas de les paraules reservades i els operadors).

La clau de aquesta interconnexió es fonamenta en què la variable global yylval es de tipus union i la seva estructura és equivalent a la definida en el union del sintàctic, per la qual cosa la creació dels diferents camps del union vindrà determinada pels diferents tipus de dades a comunicar.

La relació establerta entre el lèxic i el sintàctic es fonamenta, per una banda, en el tipus de token retornat (els diferents tipus es defineixen en el sintàctic tal i com s'ha descrit abans), la qual cosa permet identificar la regla on associar el token rebut; i per altra banda, la informació addicional amb un valor o un símbol inclòs en la variable yylval que és rebut en el union del sintàctic.

El tractament que realitza el sintàctic de les dades associades en el union es realitza per mitjà de la consulta dels valors associats a cadascun dels elements que integren una regla, i s'inicia en les fulles de l'arbre sintàctic i es propaga fins arribar a l'arrel de l'arbre, punt on conclou la compilació. La consulta d'aquests valors es realitza amb les variables $1, $2, ..., $N corresponents a la informació dels elements

1, 2, ..., N de la regla sintàctica. Per poder pujar informació en l'arbre sintàctic de la regla actual cap a la regla pare s'utilitza la variable $$. Tant les variables $N com la variable $$ referencien el union, de la mateixa forma que la variable yylval en el lèxic, pel que hauran d'indicar el tipus de valor a establir o consultar en cada moment.

```
%token <string>             PR_ID                           %token <simbol_ptr>       PR_IF
%token <simbol_ptr>         PR_ID_VARIABLE                  %token <simbol_ptr>       PR_IN
%token <simbol_ptr>         PR_ID_FUNCTION                  %token <simbol_ptr>       PR_LOOP
%token <simbol_ptr>                 PR_ID_PROCEDURE         %token <simbol_ptr>       PR_MOD
%token <simbol_ptr>                 PR_ID_TYPE              %token <simbol_ptr>       PR_NEW
                                                            %token <simbol_ptr>       PR_NOT
%token <vinteger>           PR_NUMERIC_LITERAL              %token <simbol_ptr>       PR_NULL
%token <vfloat>             PR_DECIMAL_LITERAL              %token <simbol_ptr>       PR_OF
%token <character>          PR_CHARACTER_LITERAL            %token <simbol_ptr>       PR_OR
%token <string>                     PR_STRING_LITERAL       %token <simbol_ptr>       PR_OTHERS
                                                            %token <simbol_ptr>       PR_PROCEDURE
// El lexic no retornara cap valor als seguents tokens      %token <simbol_ptr>       PR_RANGE
%token <simbol_ptr>       PR_ABS                            %token <simbol_ptr>       PR_RECORD
%token <simbol_ptr>       PR_ACCESS                         %token <simbol_ptr>       PR_RETURN
%token <simbol_ptr>       PR_ALL                            %token <simbol_ptr>       PR_REVERSE
%token <simbol_ptr>       PR_AND                            %token <simbol_ptr>       PR_SUBTYPE
%token <simbol_ptr>       PR_ARRAY                          %token <simbol_ptr>       PR_THEN
%token <simbol_ptr>       PR_BEGIN                          %token <simbol_ptr>       PR_TYPE
%token <simbol_ptr>       PR_CASE                           %token <simbol_ptr>       PR_WHEN
%token <simbol_ptr>       PR_CONSTANT                       %token <simbol_ptr>       PR_WHILE
%token <simbol_ptr>       PR_DECLARE                        %token <simbol_ptr>              PR_EQU
%token <simbol_ptr>       PR_IS                             %token <simbol_ptr>       PR_ARROW
%token <simbol_ptr>       PR_ELSE                           %token <simbol_ptr>       PR_MAJEQU
%token <simbol_ptr>       PR_ELSIF                          %token <simbol_ptr>       PR_MINEQU
%token <simbol_ptr>       PR_END                            %token <simbol_ptr>              PR_DIFFERENT
%token <simbol_ptr>       PR_EXIT                           %token <simbol_ptr>       PR_EXP
%token <simbol_ptr>       PR_FOR                            %token <simbol_ptr>              PR_2_POINTS
%token <simbol_ptr>       PR_FUNCTION
```

Els tipus de tokens definits en el sintàctic i enviats des del lèxic són els següents:
L'estructura union definida en el sintàctic és la següent:

```
%union {
        // Inicialitzat des de flex
        Simbol *simbol_ptr;              // Si el simbol de l'identificador ja existeix, el referenciem
         Simbol simbol;          // Valors que pasem cap adalt en l'arbre sintactic (bison).
        double  vfloat;         // Valor numeric decimal
        int     vinteger;               // Valor numeric senser
        char    string[CONTENIDOR_STRING_LENGTH]; // Strings o nom de nou identificador
        char    character;              // Valor caracters
}
```

Les regles que requereixen retornar informació a les regles pares necessiten establir-se en el sintàctic amb la notació “%type”:

```
%type <simbol> type_definition enumeration_type_definition enumeration_literal_specification record_definition
record_type_definition range direct_name name primary factor term simple_expression component_definition
range_constraint scalar_constraint subtype_mark subtype_indication defining_identifier_element constraint
indexed_component prefix relation expression selected_component type_conversion function_call array_type_definition
integer_type_definition signed_integer_type_definition constrained_array_definition discrete_subtype_definition
parameter_and_result_profile subprogram_specification procedure_call_statement designator subprogram_body_element
program_element condition
%type <string> defining_identifier defining_character_literal selector_name defining_program_unit_name
defining_designator
%type <vinteger> actual_parameter_part_element function_call_element mark
%type <simbol> sequence_of_statements if_statement n_mark loop_statement iteration_scheme loop_parameter_specification
compound_statement statement sequence_of_statements_element
```

La regla d'inici del sintàctic s'indica amb la clàusula “%start”, i fa referència a l'arrel sintàctica de llenguatge, en el cas del MiniAda, la regla “program”.

La precedència d'operadors definida és la següent:

```
%left '-' '+'
%left '*' '/'
```

```
%right PR_EXP

%left PR_IF
%left PR_THEN
%left PR_ELSE
%left PR_ELSIF
```

Segons el llenguatge, s'esperen 2 tipus de conflictes de desplaçament / reducció, els quals són inevitables en complir les regles establertes en la definició del llenguatge MiniAda, i que no suposen cap problema per a l'execució del compilador. Es determinen amb la clàusula:

```
%expect 2
```

En el sintàctic s'aplica, de la mateixa forma que l'anàlisi sintàctica, l'anàlisi semàntica, aplicant comparacions i validacions de tipus per a tots els símbols gestionats al llarg de la compilació. L'estructura de Tipus Abstracte de Dades definida en la taula de símbols facilita el tractament semàntic podent realitzant fàcilment comprovacions del tipus dels símbols.

Durant l'execució del sintàctic es completa la tota la informació possible de cada símbol de cadascuna de les taules de símbols associades als diferents àmbits del programa compilat.

En cas de no associar-se el conjunt de tokens rebut amb cap de les regles sintàctiques definides, es genera un error sintàctic. Si es detecta qualsevol error s'invoca la funció “sinerror” indicant en un missatge prou aclaridor el motiu de l'error. S'adjunta en la descripció de l'error el número de linia i columna on s'ha produit, que és actualitzat per l'analitzador lèxic.

6.Joc de Proves

En aquest capítol s'ha realitzat tot un seguit de proves amb programes en MiniAda d'exemple per tal de verificar el correcte funcionament del compilador i de la generació de la taula de símbols.

S'inclouen diversos casos d'exemple per il·lustrar el bon funcionament del compilador.

Proves de generació de la Taula de Símbols

Amb aquestes proves es mostra el contingut de la taula de símbols corresponent a diversos programes de prova:

Exemple 1

CODI FONT

```
procedure principal is

	enter10: integer;

	procedure subprograma1(param1: integer; param2: float; param3: string; param4: boolean; param5: character) is
		type subrang1 is range 1..10;

		type r1 is record
			a: float;
			b: integer;
			c: string;
			d: character;
			e: boolean;
			f: subrang1;
		end record;

		type taula1 is array (1..10) of integer;

		type r2 is record
			a1: r1;
			a2: taula1;
			b: integer;
		end record;

		type taula2 is array (1..10) of r1;

		variable1: character;
		variable2: integer;
		variable3: subrang1;
		variable4: boolean;
		variable5: float;
		variable6: string;
		variable7: r1;
		variable8: r2;
		variable9: taula1;
		variable10: taula2;
	begin
		variable10(variable2 + 2) := variable10(1);
		null;
	end subprograma1;

	function funcio1(p1: float; p2: boolean; p3: string) return integer is
		type registre is record
			r1: float;
```

```
                        r2: integer;
                        r3: character;
                end record;

                type taula is array (1..100) of integer;

                v1: string;
                v2: character;
                v3: registre;
                v4: taula;
                v5: integer;
        begin
                v5 := 1;
                return v5;
        end;

        function funcio2 return integer is

                type cap_de_setmana is (dissabte,diumenge);
                procedure encaixat(primer:string; segon: float; tercer: cap_de_setmana) is
                        a,b,c: integer;
                        d,e: float;
                        f: boolean;
                begin
                        null;
                end encaixat;
        begin
           return 1;
    end;

begin
         for enter10 in 1..100 loop
            null;
         end loop;
end;
```

RESULTAT

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Practica de Compiladors II - Compilador de MiniAda.
Fitxer descriptor de la Taula de Simbols.
Autors: Sergio Blanco Cuaresma i Francisco Jose Aguilar Celdran.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    ------------
    Ambit $MAIN$
    ------------
    Variable BOOLEAN :: nom = false, valor = FALSE (es una constant)
    Tipus FLOAT :: nom = float
    Tipus BOOLEAN :: nom = boolean
    Tipus CHARACTER :: nom = character
    Rutina FUNCTION :: nom = length, num_parametres = 1, definicio de funcio = completa
    Valor de retorn de la funcio length: tipus = integer
    Parametres de la funcio length:
        nom: item, tipus: string
    Fi dels parametres de la funcio length
    Rutina PROCEDURE :: nom = put, num_parametres = 1, definicio de procediment = completa
    Parametres del procediment put:
        nom: item, tipus: integer
    Fi dels parametres del procediment put
    El procediment put te 3 sobrecarregues:
        Rutina PROCEDURE :: nom = put, num_parametres = 1, definicio de procediment = completa
        Parametres del procediment put:
            nom: item, tipus: float
        Fi dels parametres del procediment put
        Rutina PROCEDURE :: nom = put, num_parametres = 1, definicio de procediment = completa
        Parametres del procediment put:
            nom: item, tipus: character
        Fi dels parametres del procediment put
        Rutina PROCEDURE :: nom = put, num_parametres = 1, definicio de procediment = completa
        Parametres del procediment put:
            nom: item, tipus: string
        Fi dels parametres del procediment put
    Fi de sobrecarregues del procediment put
    Variable BOOLEAN :: nom = true, valor = TRUE (es una constant)
    Rutina FUNCTION :: nom = get_float, num_parametres = 0, definicio de funcio = completa
```

```
Valor de retorn de la funcio get_float: tipus = float
La funcio get_float no te parametres
Tipus INTEGER :: nom = integer
Rutina FUNCTION :: nom = get_character, num_parametres = 0, definicio de funcio = completa
Valor de retorn de la funcio get_character: tipus = character
La funcio get_character no te parametres
Rutina PROCEDURE :: nom = principal, num_parametres = 0, definicio de procediment = completa
El procediment principal no te parametres
Tipus STRING :: nom = string
Rutina FUNCTION :: nom = get_integer, num_parametres = 0, definicio de funcio = completa
Valor de retorn de la funcio get_integer: tipus = integer
La funcio get_integer no te parametres
Rutina FUNCTION :: nom = get_string, num_parametres = 0, definicio de funcio = completa
Valor de retorn de la funcio get_string: tipus = string
La funcio get_string no te parametres
    ------------
    Ambit length
    ------------
    Variable STRING :: nom = item, valor = (no inicialitzat)

    ---------
    Ambit put
    ---------
    Variable INTEGER :: nom = item, valor = (no inicialitzat)

    -------------------------
    Ambit put (sobrecarrega 1)
    -------------------------
    Variable FLOAT :: nom = item, valor = (no inicialitzat)

    -------------------------
    Ambit put (sobrecarrega 2)
    -------------------------
    Variable CHARACTER :: nom = item, valor = (no inicialitzat)

    -------------------------
    Ambit put (sobrecarrega 3)
    -------------------------
    Variable STRING :: nom = item, valor = (no inicialitzat)

    ---------------
    Ambit get_float
    ---------------

    -------------------
    Ambit get_character
    -------------------

    ---------------
    Ambit principal
    ---------------
    # Nom Complet: principal_0
    # Jerarquia precedent: principal, $MAIN$

    Rutina PROCEDURE :: nom = subprograma1, num_parametres = 5, definicio de procediment = completa
    Parametres del procediment subprograma1:
        nom: param1, tipus: integer
        nom: param2, tipus: float
        nom: param3, tipus: string
        nom: param4, tipus: boolean
        nom: param5, tipus: character
    Fi dels parametres del procediment subprograma1
    Variable INTEGER :: nom = enter10, valor = (no inicialitzat)
    Rutina FUNCTION :: nom = funcio1, num_parametres = 3, definicio de funcio = completa
    Valor de retorn de la funcio funcio1: tipus = integer
    Parametres de la funcio funcio1:
        nom: p1, tipus: float
        nom: p2, tipus: boolean
        nom: p3, tipus: string
    Fi dels parametres de la funcio funcio1
    Rutina FUNCTION :: nom = funcio2, num_parametres = 0, definicio de funcio = completa
    Valor de retorn de la funcio funcio2: tipus = integer
    La funcio funcio2 no te parametres
        ------------------
        Ambit subprograma1
        ------------------
```

```
# Nom Complet: subprograma1_1
# Jerarquia precedent: subprograma1, principal, $MAIN$

Tipus INTEGER_DERIVAT :: nom = subrang1
Rang del tipus integer derivat subrang1 : Minim = 1, Maxim = 0, Tipus = integer
Variable TIPUS DERIVAT :: nom = variable10, tipus = taula2, categoria de tipus: array
Tipus RECORD :: nom = r1
Elements del record r1:
    Variable INTEGER_DERIVAT :: nom = f, tipus = subrang1, valor = (no inicialitzat)
    Variable FLOAT :: nom = a, valor = (no inicialitzat)
    Variable INTEGER :: nom = b, valor = (no inicialitzat)
    Variable STRING :: nom = c, valor = (no inicialitzat)
    Variable CHARACTER :: nom = d, valor = (no inicialitzat)
    Variable BOOLEAN :: nom = e, valor = (no inicialitzat)
Fi dels elements del record r1

Tipus RECORD :: nom = r2
Elements del record r2:
    Variable TIPUS DERIVAT :: nom = a1, tipus = r1, categoria de tipus: record
    Variable TIPUS DERIVAT :: nom = a2, tipus = taula1, categoria de tipus: array
    Variable INTEGER :: nom = b, valor = (no inicialitzat)
Fi dels elements del record r2

Variable INTEGER :: nom = param1, valor = (no inicialitzat)
Variable FLOAT :: nom = param2, valor = (no inicialitzat)
Variable STRING :: nom = param3, valor = (no inicialitzat)
Variable BOOLEAN :: nom = param4, valor = (no inicialitzat)
Variable CHARACTER :: nom = param5, valor = (no inicialitzat)
Tipus ARRAY :: nom = taula1
Rang de l'array taula1 : Minim = 1.000000, Maxim = 10.000000, Tipus = integer
Tipus ARRAY :: nom = taula2
Rang de l'array taula2 : Minim = 1.000000, Maxim = 10.000000, Tipus = integer
Variable CHARACTER :: nom = variable1, valor = (no inicialitzat)
Variable INTEGER :: nom = variable2, valor = (no inicialitzat)
Variable INTEGER_DERIVAT :: nom = variable3, tipus = subrang1, valor = (no inicialitzat)
Variable BOOLEAN :: nom = variable4, valor = (no inicialitzat)
Variable FLOAT :: nom = variable5, valor = (no inicialitzat)
Variable STRING :: nom = variable6, valor = (no inicialitzat)
Variable TIPUS DERIVAT :: nom = variable7, tipus = r1, categoria de tipus: record
Variable TIPUS DERIVAT :: nom = variable8, tipus = r2, categoria de tipus: record
Variable TIPUS DERIVAT :: nom = variable9, tipus = taula1, categoria de tipus: array
Variable INTEGER :: nom = $t0, valor = (no inicialitzat)
Variable TIPUS DERIVAT :: nom = $t1, tipus = r1, categoria de tipus: record

-------------
Ambit funcio1
-------------
# Nom Complet: funcio1_2
# Jerarquia precedent: funcio1, principal, $MAIN$

Tipus ARRAY :: nom = taula
Rang de l'array taula : Minim = 1.000000, Maxim = 100.000000, Tipus = integer
Variable FLOAT :: nom = p1, valor = (no inicialitzat)
Variable BOOLEAN :: nom = p2, valor = (no inicialitzat)
Variable STRING :: nom = p3, valor = (no inicialitzat)
Variable STRING :: nom = v1, valor = (no inicialitzat)
Variable CHARACTER :: nom = v2, valor = (no inicialitzat)
Tipus RECORD :: nom = registre
Elements del record registre:
    Variable FLOAT :: nom = r1, valor = (no inicialitzat)
    Variable INTEGER :: nom = r2, valor = (no inicialitzat)
    Variable CHARACTER :: nom = r3, valor = (no inicialitzat)
Fi dels elements del record registre

Variable TIPUS DERIVAT :: nom = v3, tipus = registre, categoria de tipus: record
Variable TIPUS DERIVAT :: nom = v4, tipus = taula, categoria de tipus: array
Variable INTEGER :: nom = v5, valor = (no inicialitzat)

-------------
Ambit funcio2
-------------
# Nom Complet: funcio2_3
# Jerarquia precedent: funcio2, principal, $MAIN$

Tipus ENUMERATED :: nom = cap_de_setmana
Elements de l'enumerat cap_de_setmana:
```

```
            nom: dissabte, valor: 0
            nom: diumenge, valor: 1
        Fi dels elements de l'enumerat cap_de_setmana
        Rutina PROCEDURE :: nom = encaixat, num_parametres = 3, definicio de procediment = completa
        Parametres del procediment encaixat:
            nom: primer, tipus: string
            nom: segon, tipus: float
            nom: tercer, tipus: derivat nom tipus derivat: cap_de_setmana
        Fi dels parametres del procediment encaixat
        Variable INTEGER_DERIVAT :: nom = diumenge, tipus = cap_de_setmana, valor = '1' (es una constant)
        Variable INTEGER_DERIVAT :: nom = dissabte, tipus = cap_de_setmana, valor = '0' (es una constant)
            --------------
            Ambit encaixat
            --------------
            # Nom Complet: encaixat_4
            # Jerarquia precedent: encaixat, funcio2, principal, $MAIN$

            Variable BOOLEAN :: nom = f, valor = (no inicialitzat)
            Variable FLOAT :: nom = segon, valor = (no inicialitzat)
            Variable TIPUS DERIVAT :: nom = tercer, tipus = cap_de_setmana, categoria de tipus: enumerated
            Variable STRING :: nom = primer, valor = (no inicialitzat)
            Variable INTEGER :: nom = a, valor = (no inicialitzat)
            Variable INTEGER :: nom = b, valor = (no inicialitzat)
            Variable INTEGER :: nom = c, valor = (no inicialitzat)
            Variable FLOAT :: nom = e, valor = (no inicialitzat)
            Variable FLOAT :: nom = d, valor = (no inicialitzat)

    -----------------
    Ambit get_integer
    -----------------

    ----------------
    Ambit get_string
    ----------------
```

Exemple 2

CODI FONT

```
procedure principal is

        type reg_simple is record
                camp1: string;
                camp2: character;
                camp3: boolean;
        end record;

        type llista_simple is array (1..50) of integer;

        type enum_simple is (aaa, bbb, ccc, ddd, eee);

        type llista1 is array (1..10) of float;
        type llista2 is array (1..10) of integer;
        type llista3 is array (1..10) of reg_simple;
        type llista4 is array (1..10) of llista_simple;

        type reg_compost is record
                r1: reg_simple;
                r2: llista1;
                r3: llista2;
                r4: llista3;
                r5: llista4;
                r6: enum_simple;
        end record;

        function funcio_a(val1: float; val2: float; e: enum_simple) return integer is
                int_value: integer;
                dec_value: float;
                char: character;
                bool: boolean;
        begin
                return 0;
        end funcio_a;

        function funcio_b(b: boolean; l1: llista1; l2: llista2; l3: llista3) return string is
```

```
                int_value: integer;
                dec_value: float;
                char: character;
                bool: boolean;

                var2: llista2;
                var3: llista3;

        begin
                --var2 := l2;
                return "Valor de retorn";
        end funcio_b;

        function funcio_c(reg: reg_compost) return integer is
                int_value: integer;
                dec_value: float;
                char: character;
                bool: boolean;
                reg_local: reg_compost;
        begin
                --reg := reg_local;
                return 1000;
        end funcio_c;


        r_simple: reg_simple;
        e_simple: enum_simple;
        ll_simple: llista_simple;

        f1,f2: float;
        b1: boolean;
        i1: integer;
        s1: string;
        enumerat: enum_simple;

        lli1: llista1;
        lli2: llista2;
        lli3: llista3;

begin

        b1 := true;
        f1 := 1.0;
        f2 := f1 * 2 + 5;
        i1 := funcio_a(f1, f2, enumerat);
        s1 := funcio_b(b1, lli1, lli2, lli3);

end;
```

RESULTAT

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Practica de Compiladors II - Compilador de MiniAda.
Fitxer descriptor de la Taula de Simbols.
Autors: Sergio Blanco Cuaresma i Francisco Jose Aguilar Celdran.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    ------------
    Ambit $MAIN$
    ------------
    Variable BOOLEAN :: nom = false, valor = FALSE (es una constant)
    Tipus FLOAT :: nom = float
    Tipus BOOLEAN :: nom = boolean
    Tipus CHARACTER :: nom = character
    Rutina FUNCTION :: nom = length, num_parametres = 1, definicio de funcio = completa
    Valor de retorn de la funcio length: tipus = integer
    Parametres de la funcio length:
        nom: item, tipus: string
    Fi dels parametres de la funcio length
    Rutina PROCEDURE :: nom = put, num_parametres = 1, definicio de procediment = completa
    Parametres del procediment put:
        nom: item, tipus: integer
    Fi dels parametres del procediment put
    El procediment put te 3 sobrecarregues:
        Rutina PROCEDURE :: nom = put, num_parametres = 1, definicio de procediment = completa
        Parametres del procediment put:
```

```
        nom: item, tipus: float
    Fi dels parametres del procediment put
    Rutina PROCEDURE :: nom = put, num_parametres = 1, definicio de procediment = completa
    Parametres del procediment put:
        nom: item, tipus: character
    Fi dels parametres del procediment put
    Rutina PROCEDURE :: nom = put, num_parametres = 1, definicio de procediment = completa
    Parametres del procediment put:
        nom: item, tipus: string
    Fi dels parametres del procediment put
Fi de sobrecarregues del procediment put
Variable BOOLEAN :: nom = true, valor = TRUE (es una constant)
Rutina FUNCTION :: nom = get_float, num_parametres = 0, definicio de funcio = completa
Valor de retorn de la funcio get_float: tipus = float
La funcio get_float no te parametres
Tipus INTEGER :: nom = integer
Rutina FUNCTION :: nom = get_character, num_parametres = 0, definicio de funcio = completa
Valor de retorn de la funcio get_character: tipus = character
La funcio get_character no te parametres
Rutina PROCEDURE :: nom = principal, num_parametres = 0, definicio de procediment = completa
El procediment principal no te parametres
Tipus STRING :: nom = string
Rutina FUNCTION :: nom = get_integer, num_parametres = 0, definicio de funcio = completa
Valor de retorn de la funcio get_integer: tipus = integer
La funcio get_integer no te parametres
Rutina FUNCTION :: nom = get_string, num_parametres = 0, definicio de funcio = completa
Valor de retorn de la funcio get_string: tipus = string
La funcio get_string no te parametres
    ------------
    Ambit length
    ------------
    Variable STRING :: nom = item, valor = (no inicialitzat)

    ---------
    Ambit put
    ---------
    Variable INTEGER :: nom = item, valor = (no inicialitzat)

    --------------------------
    Ambit put (sobrecarrega 1)
    --------------------------
    Variable FLOAT :: nom = item, valor = (no inicialitzat)

    --------------------------
    Ambit put (sobrecarrega 2)
    --------------------------
    Variable CHARACTER :: nom = item, valor = (no inicialitzat)

    --------------------------
    Ambit put (sobrecarrega 3)
    --------------------------
    Variable STRING :: nom = item, valor = (no inicialitzat)

    ---------------
    Ambit get_float
    ---------------

    -------------------
    Ambit get_character
    -------------------

    ---------------
    Ambit principal
    ---------------
    # Nom Complet: principal_0
    # Jerarquia precedent: principal, $MAIN$

    Variable INTEGER :: nom = $t2, valor = (no inicialitzat)
    Variable INTEGER_DERIVAT :: nom = eee, tipus = enum_simple, valor = '4' (es una constant)
    Variable STRING :: nom = $t3, valor = (no inicialitzat)
    Rutina FUNCTION :: nom = funcio_a, num_parametres = 3, definicio de funcio = completa
    Valor de retorn de la funcio funcio_a: tipus = integer
    Parametres de la funcio funcio_a:
        nom: val1, tipus: float
        nom: val2, tipus: float
        nom: e, tipus: derivat nom tipus derivat: enum_simple
```

```
Fi dels parametres de la funcio funcio_a
Rutina FUNCTION :: nom = funcio_b, num_parametres = 4, definicio de funcio = completa
Valor de retorn de la funcio funcio_b: tipus = string
Parametres de la funcio funcio_b:
    nom: b, tipus: boolean
    nom: l1, tipus: derivat nom tipus derivat: llista1
    nom: l2, tipus: derivat nom tipus derivat: llista2
    nom: l3, tipus: derivat nom tipus derivat: llista3
Fi dels parametres de la funcio funcio_b
Rutina FUNCTION :: nom = funcio_c, num_parametres = 1, definicio de funcio = completa
Valor de retorn de la funcio funcio_c: tipus = integer
Parametres de la funcio funcio_c:
    nom: reg, tipus: derivat nom tipus derivat: reg_compost
Fi dels parametres de la funcio funcio_c
Variable TIPUS DERIVAT :: nom = e_simple, tipus = enum_simple, categoria de tipus: enumerated
Variable BOOLEAN :: nom = b1, valor = (no inicialitzat)
Variable FLOAT :: nom = f1, valor = (no inicialitzat)
Variable FLOAT :: nom = f2, valor = (no inicialitzat)
Variable INTEGER :: nom = i1, valor = (no inicialitzat)
Variable TIPUS DERIVAT :: nom = r_simple, tipus = reg_simple, categoria de tipus: record
Variable TIPUS DERIVAT :: nom = ll_simple, tipus = llista_simple, categoria de tipus: array
Tipus RECORD :: nom = reg_simple
Elements del record reg_simple:
    Variable STRING :: nom = camp1, valor = (no inicialitzat)
    Variable CHARACTER :: nom = camp2, valor = (no inicialitzat)
    Variable BOOLEAN :: nom = camp3, valor = (no inicialitzat)
Fi dels elements del record reg_simple

Variable STRING :: nom = s1, valor = (no inicialitzat)
Variable TIPUS DERIVAT :: nom = enumerat, tipus = enum_simple, categoria de tipus: enumerated
Variable TIPUS DERIVAT :: nom = lli1, tipus = llista1, categoria de tipus: array
Variable TIPUS DERIVAT :: nom = lli2, tipus = llista2, categoria de tipus: array
Variable TIPUS DERIVAT :: nom = lli3, tipus = llista3, categoria de tipus: array
Tipus ENUMERATED :: nom = enum_simple
Elements de l'enumerat enum_simple:
    nom: aaa, valor: 0
    nom: bbb, valor: 1
    nom: ccc, valor: 2
    nom: ddd, valor: 3
    nom: eee, valor: 4
Fi dels elements de l'enumerat enum_simple
Tipus RECORD :: nom = reg_compost
Elements del record reg_compost:
    Variable TIPUS DERIVAT :: nom = r1, tipus = reg_simple, categoria de tipus: record
    Variable TIPUS DERIVAT :: nom = r2, tipus = llista1, categoria de tipus: array
    Variable TIPUS DERIVAT :: nom = r3, tipus = llista2, categoria de tipus: array
    Variable TIPUS DERIVAT :: nom = r4, tipus = llista3, categoria de tipus: array
    Variable TIPUS DERIVAT :: nom = r5, tipus = llista4, categoria de tipus: array
    Variable TIPUS DERIVAT :: nom = r6, tipus = enum_simple, categoria de tipus: enumerated
Fi dels elements del record reg_compost

Variable INTEGER_DERIVAT :: nom = aaa, tipus = enum_simple, valor = '0' (es una constant)
Tipus ARRAY :: nom = llista_simple
Rang de l'array llista_simple : Minim = 1.000000, Maxim = 50.000000, Tipus = integer
Variable INTEGER_DERIVAT :: nom = bbb, tipus = enum_simple, valor = '1' (es una constant)
Variable INTEGER_DERIVAT :: nom = ccc, tipus = enum_simple, valor = '2' (es una constant)
Tipus ARRAY :: nom = llista1
Rang de l'array llista1 : Minim = 1.000000, Maxim = 10.000000, Tipus = integer
Tipus ARRAY :: nom = llista2
Rang de l'array llista2 : Minim = 1.000000, Maxim = 10.000000, Tipus = integer
Variable INTEGER_DERIVAT :: nom = ddd, tipus = enum_simple, valor = '3' (es una constant)
Tipus ARRAY :: nom = llista3
Rang de l'array llista3 : Minim = 1.000000, Maxim = 10.000000, Tipus = integer
Tipus ARRAY :: nom = llista4
Rang de l'array llista4 : Minim = 1.000000, Maxim = 10.000000, Tipus = integer
Variable FLOAT :: nom = $t0, valor = (no inicialitzat)
Variable FLOAT :: nom = $t1, valor = (no inicialitzat)
    --------------
    Ambit funcio_a
    --------------
    # Nom Complet: funcio_a_1
    # Jerarquia precedent: funcio_a, principal, $MAIN$

    Variable CHARACTER :: nom = char, valor = (no inicialitzat)
    Variable BOOLEAN :: nom = bool, valor = (no inicialitzat)
    Variable FLOAT :: nom = dec_value, valor = (no inicialitzat)
```

```
        Variable INTEGER :: nom = int_value, valor = (no inicialitzat)
        Variable FLOAT :: nom = val1, valor = (no inicialitzat)
        Variable FLOAT :: nom = val2, valor = (no inicialitzat)
        Variable TIPUS DERIVAT :: nom = e, tipus = enum_simple, categoria de tipus: enumerated

        --------------
        Ambit funcio_b
        --------------
        # Nom Complet: funcio_b_2
        # Jerarquia precedent: funcio_b, principal, $MAIN$

        Variable CHARACTER :: nom = char, valor = (no inicialitzat)
        Variable BOOLEAN :: nom = bool, valor = (no inicialitzat)
        Variable FLOAT :: nom = dec_value, valor = (no inicialitzat)
        Variable TIPUS DERIVAT :: nom = l1, tipus = llista1, categoria de tipus: array
        Variable TIPUS DERIVAT :: nom = l2, tipus = llista2, categoria de tipus: array
        Variable TIPUS DERIVAT :: nom = l3, tipus = llista3, categoria de tipus: array
        Variable INTEGER :: nom = int_value, valor = (no inicialitzat)
        Variable TIPUS DERIVAT :: nom = var2, tipus = llista2, categoria de tipus: array
        Variable TIPUS DERIVAT :: nom = var3, tipus = llista3, categoria de tipus: array
        Variable BOOLEAN :: nom = b, valor = (no inicialitzat)
        Variable STRING :: nom = $t0, valor = (no inicialitzat)

        --------------
        Ambit funcio_c
        --------------
        # Nom Complet: funcio_c_3
        # Jerarquia precedent: funcio_c, principal, $MAIN$

        Variable CHARACTER :: nom = char, valor = (no inicialitzat)
        Variable TIPUS DERIVAT :: nom = reg, tipus = reg_compost, categoria de tipus: record
        Variable BOOLEAN :: nom = bool, valor = (no inicialitzat)
        Variable FLOAT :: nom = dec_value, valor = (no inicialitzat)
        Variable TIPUS DERIVAT :: nom = reg_local, tipus = reg_compost, categoria de tipus: record
        Variable INTEGER :: nom = int_value, valor = (no inicialitzat)


    -----------------
    Ambit get_integer
    -----------------

    ----------------
    Ambit get_string
    ----------------
```

Proves de detecció d'errors de compilació

En aquest darrer apartat es realitzen diverses proves per tal de verificar el correcte funcionament del compilador. D'aquesta forma es pretén comprovar la detecció de totes les possibles incorreccions el codi font de MiniAda rebut com a entrada del nostre compilador de MiniAda.

A continuació es detallen diverses proves de detecció d'errors realitzades, tot tenint en compte la verificació de la correctesa lèxica, sintàctica i semàntica:

Partim del següent fitxer d'exemple de codi de MiniAda que compila correctament:

```
procedure simple is

-- ENUMERATED TYPES
type capDeSetmana is (dissabte, diumenge);
type lletres is ('a', 'b', 'c', 'd', 'e', 'f');

-- INTEGER TYPES
type enter1 is range 1 + 1/1..5*2;
--type enter2 is range 'a' .. 'z'; -- Nomes permetem rangs d'enters
```

```
--type enter3 is range 'a' (2>=1 and 2 not in 4..6) .. "z";

-- ARRAY TYPES
type vector is array (1..2) of integer;

-- RECORD TYPES
type date is record
    mes: integer;
    dia: integer;
        test: integer range 1..2 + 2;
        test2: capDeSetmana;
end record;

-- VARIABLES
enter10: integer;
enter20: constant integer := 1 + 2 * 2;
testtest: constant capDeSetmana := dissabte;
--enter30: vector := (1, 2);  -- agregats/aggregates no han de ser suportats
enter30: vector;
enter40: integer range 1..2;
adsfasdf: array (1..6) of integer := 1;
enter50: constant integer range 1..2;
--a,b: float;                -- Es verifica que a i b formen part de un enumerat i no es permeten definir com a
variables
data: date;

procedure test(hola: boolean);
procedure test(uno: integer; dos: integer);
procedure test(uno: integer; dos: integer) is  -- Mirar si s'ha de comprovar duplicats entre parametres del procedure
i variables locals
begin
        null;
end;

procedure test(hola: boolean) is
begin
        null;
end;

function abc return integer;     -- Definicio de funcio incompleta
function abc return integer is
begin
        return 0;
end abc;


function testFunc1(param1: integer; param2: float) return integer is
begin
        null;
        return 0;
end;
function testFunc1(param1: boolean; param2: boolean) return integer is
begin
        null;
        return 0;
end;

--function testFunc2(param1: integer) return integer; -- Es controla que existeixi una implementacio de la definicio
incompleta de funcio

--inicializa: integer := testFunc1(1, 2.1); --Detecta que no existeix la implementacio de la funcio definida;
--inicializa: integer := abc;

procedure prova is

        function func_1(param1: float; param2: float) return boolean is
        begin
                if(param1 <= param2) then
                        return true;
                else
                        return false;
                end if;

        end func_1;

        bool_value: boolean;
```

```
        a1, b1: float;

begin

        a1 := 6.0;
        b1 := 12.5 E 100;
        bool_value := func_1(a1, b1);

end prova;


cadena: string := "123456789";

begin
        --enter20 := 2;  -- Detecta que es una constant i que no es pot assignar un valor
        data.mes := dissabte;
        if (true and then true) then
                null;
        end if;

        if (true or else true) then
        null;
    end if;

        null;

        if (true) then
                null;
        else    -- "else if" no s'ha de suportar
                 if (1 > 2) then
                        null;
                end if;
        end if;

        for enter10 in 1..100 loop
      null;
    end loop;

        --for enter40 in 1..100 loop -- Detecta que enter40 no pot contenir valors d'aquest rang
      --null;
    --end loop;

        --for enter20 in 1..100 loop      -- Detecta que la variable es constant i no pot ser utilitzada
      --null;
    --end loop;

        --while (1) loop -- Detecta que no es una condicio valida
        --      null;
        --end loop;

        while (2 < 3) loop
                null;
        end loop;


        return;
end;
```

L'anterior fitxer compila correctament. Tot seguit hi fem petites modificacions errònies per verificar detalls lèxics, sintàctics i semàntics.

Errors Lèxics

Linia 95:
modifiquem: `cadena: string := "123456789";`
per: `cadena: string :=`

```
  "123456789.....................................................................
.....................................................................................
.....................................................................................
.....................................................................................
```

```
.....................................................................................";
```
obtenim: *Error a la linia 95 columna 18:String massa llarg, com a maxim poden haver-hi '255' caracters*

Linia 95:
modifiquem: `cadena: string := "123456789";`
per: `cadena: string := "123456789;`
obtenim: *Error a la linia 95 columna 28:Error: Salt de linia abans de finalitzacio de String. (Linia: 95, Columna: 28)*

Linia 2:
modifiquem: `procedure simple is`
per: `procedure simple is ?`
obtenim: Error a la linia 2 columna 20:Trobada la sequencia desconeguda '?' (Linia: 2, Columna: 20)

Linia 44:
modifiquem: `enter20: constant integer := 1 + 2 * 2;`
per: `enter20: constant integer := 999999999999999 + 2 * 2;`
obtenim: Error a la linia 26 columna 44:El numero decimal surt del rang permes -2147483648 i 2147483647.

Linia 89:
modifiquem: `b1 := 12.5 E 100;`
per: `b1 := 12.5 E 400;`
obtenim: Error a la linia 89 columna 15:El numero decimal surt del rang permes -1.7E-308 i 1.7E+308.

Errors Sintàctics

Linia 104:
modifiquem: `if (true or else true) then`
per: `if (true or else true) aleshores`
obtenim: parse error a la linia 104 columna 33

Linia 140:
modifiquem: `end loop;`
per: `--end loop;`
obtenim: parse error a la linia 140 columna 4

Linia 140:
modifiquem: `function abc return integer;`
per: `function sabc return integer;`
obtenim: Error a la linia 140 columna 4: La funcio 'sabc' no te un cos definit.

Els errors sintàctics es produeixen en no complir cap de les regles de l'analitzdor sintàctic, amb el que es produeix un error genèric del tipus dels dos exemples anteriors en totes les situacions desconegudes pel compilador.

Errors Semàntics

Es realitzen comprovacions de tipus per tal de validar la correctesa semàntica de totes les expressions.

S'adjunten a continuació alguns exemples per mostrar la captura d'errors semàntics:

Linia 104:
modifiquem: `if (true) then`
per: `if (100.0) then`
obtenim: parse error a la linia 104 columna 33

Linia 113:
modifiquem: `if (1 > 2) then`
per: `if (1 > "cadena") then`
obtenim: parse error a la linia 113 columna 13

Linia 118:
modifiquem: `for enter10 in 1..10 loop`
per: `for enter10 in 1..-100.5 loop`
obtenim: Error a la linia 118 columna 30: Els tipus de limit de rang tenen tipus diferents

Linia 90:
modifiquem: `bool_value := func_1(a1, b1);`
per: `bool_value := func_1("cadena", 'c');`
obtenim: Error a la linia 90 columna 30: La funcio 'func_1' requereix '2' parametre(s) i no s'han trobat cap tipus coincident.

Linia 51:
modifiquem: `return 0;`
per: `return true;`
obtenim: Error a la linia 51 columna 14: Error al valor de retorn, el tipus no coincideix.

Totes les comprovacions de tipus per a tipus de dades simples són extensibles a qualsevol tipus de dades per complex que sigui, donat que la taula de símbols està preparada per permetre les comprovacions de tipus de forma senzilla per a qualsevol possible símbol.

Llistat de tests

Els següents tests es poden trobar junt al codi font de la pràctica:

test01: Tipus, variables, funcions i procedures (sobrecarregats o no).
test02: Statements (operacions).
test03: Sobrecàrrega.
test04: Overflows.
test05: Tipus complexos.
test06: Procediments encaixats.