Compiladors II

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. Veure també les especificacions d'instruccións de codi de tres adreces per a major informació en relació als diferents tipus d'instruccions de codi intermedi que genera el compilador a partir de la correcta compilació del codi font en MiniAda del fitxer tractat pel compilador.

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 amb generació de codi de tres adreces.

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.

Per altra banda, en cadascuna de les regles sintàctiques, s'anirà generant el codi de tres adreces corresponent a la versió en codi objecte del codi font en MiniAda del programa que es compila.

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, la generació de codi de tres adreces, i la generació dels registres

d'activació. Aquestes impressions es generen en els respectius arxius acabats amb terminació “.ts”, “.”, i “c3a” i “.ra”.

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 i generador de codi de tres adreces, 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). Hi ha a més una cua de sentències de codi de tres adreces corresponents al símbol actual, i tres cues utilitzades en el back patching: llista_cert, llista_fals i llista_seguents:

```
typedef struct Simbol {
        char nom[STRING_LENGTH];
        Cua *sentencies; // Cua d'elements tipus_sentencia (Codi 3 adreces)
        Cua *llista_cert;        // Back patching
        Cua *llista_fals;        // Back patching
        Cua *llista_seguents;    // Back patching
        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. Hi ha a més una cua de variables locals i una cua de variables temporals utilitzades per a la generació dels registres d'activació, i finalment una cua amb sentències de tres adreces de l'àmbit del procediment.

```
typedef struct Info_Procedure {
        char nom_unic[STRING_LENGTH];     // Utilitzat pel codi de 3 adreces
        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
        Cua *variables_locals;            // Cua de simbols de variables locals
        Cua *variables_temporals;         // Cua de simbols de variables temporals
        Cua *sentencies;                 // Cua d'elements tipus_sentencia (Codi 3 adreces)
} 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 {
        char nom_unic[STRING_LENGTH];         // Utilitzat pel codi de 3 adreces
        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
        Cua *variables_locals;                // Cua de simbols de variables locals
        Cua *variables_temporals;             // Cua de simbols de variables temporals
        Cua *sentencies;                      // Cua d'elements tipus_sentencia (Codi 3 adreces)
} 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, codi de tres adreces i registres d'activació en els corresponents arxius de sortida del compilador.

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_ID_VARIABLE
%token <simbol_ptr>         PR_ID_FUNCTION
%token <simbol_ptr>                 PR_ID_PROCEDURE
%token <simbol_ptr>                 PR_ID_TYPE

%token <vinteger>           PR_NUMERIC_LITERAL
%token <vfloat>             PR_DECIMAL_LITERAL
%token <character>          PR_CHARACTER_LITERAL
%token <string>                     PR_STRING_LITERAL

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

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.

En l'analitzador sintàctic, tal i com es comentarà a continuació, es genera el codi de tres adreces associat al codi font original de MiniAda que és processat pel compilador. Aquest tractament es duu a terme emmagatzemant en la taula de símbols tota la informació necessària de codi de tres adreces que pertany a un àmbit en concret o a un símbol, per tal de pujar la informació per l'arbre sintàctic.

6.Codi de Tres Adreces i Registres d'Activació

Codi de Tres Adreces

Durant el processament de la compilació, en les diferents etapes de tractament de les regles jeràrquiques de l'analitzador sintàctic, es va construint el codi de tres adreces traduint cada expressió, sentència, definició, etc. de codi en MiniAda en el seu seguit de proposicions de codi de tres adreces. Per a això, s'empren en el tipus de dades Símbol els següents camps:

```
Cua *sentencies; // Cua d'elements tipus_sentencia (Codi 3 adreces)
Cua *llista_cert;        // Back patching
Cua *llista_fals;        // Back patching
Cua *llista_seguents;    // Back patching
```

En la cua de sentències s'emmagatzemen el conjunt de sentències que engloben la traducció a codi objecte de la instrucció de MiniAda corresponent. Entenent el context del funcionament de l'analitzador sintàctic, s'utilitza el fet de pujar informació de regles filles a regles pares en el sintàctic amb informació associada de la regla tractada en la variable $$, la qual referència a la estructura union i es guarda habitualment el símbol per retornar als pares de l'arbre sintàctic. En aquesta evolució de regles de fills a pares, en alguns casos (e.g. al trobar la paraula reservada “true”) es va acumulant en aquesta cua de sentencies les diferents proposicions de tres adreces fins arribar un punt a on es guarda aquesta informació en una variable global. En altres casos, el codi generat es guarda directament a aquesta variable global. Quan s'arriba a conformar tot el llistat de proposicions de tres adreces d'un àmbit sencer, s'assigna la llista de sentències de la variable global al camp sentències de la estructura Info_Function o Info_Procedure de l'àmbit corresponent.

A més, s'empren els camps “llista_cert”, “llista_fals”, i “llista_seguents” per manegar el tractament de la tècnica del *Back Patching*, emprada en diferents estructures com ara expressions booleanes o proposicions de flux de control. Amb aquesta tècnica es possibilita tractar en una única passada pel codi tota la generació de codi objecte.

El problema principal de la generació de codi per a les expressions booleanes i les proposicions de flux de control en una sola passada radica en el fet que amb un únic recorregut és possible que no es coneguin les etiquetes (posicions de memòria) a les que ha de anar el control en el moment en el qual es generen les proposicions de salt. Es pot evitar aquest problema generant una serie de proposicions de ramificació sense especificar temporalment els destins dels salts. Cadascuna d'aquestes proposicions s'ubicarà en una llista de proposicions “goto” les etiquetes de les quals s'ompliran a mida que es pugui determinar l'etiqueta adequada. Aquest fet d'omplir les etiquetes a posteriori s'anomena *back patching*.

Una vegada ha finalitzat l'execució del compilador de MiniAda i el codi font no contenia cap error, s'ha generat aleshores el codi objecte de tres adreces equivalent al de MiniAda, i s'ha emmagatzemat en els diferents àmbits i símbols de la taula de símbols el llistat de proposicions de codi de tres adreces equivalents per a cada cas.

En aquest punt, es crida a la funció print_codi_tres_adreces del fitxer print_c3a.c, la qual mostrarà totes les proposicions de codi de tres adreces generades pel compilador, corresponents al codi MiniAda original. Per a cada àmbit es mostraran totes les proposicions contingudes en la cua de sentències, on cada element és una estructura Tipus_Sentencia.

A diferència de la impressió de la taula de símbols, en el cas de la impressió del codi de tres adreces no s'inclou la taula de símbols pare d'àmbit inicial donat que no aporta cap informació aquest àmbit en el que hi ha únicament les funcions, tipus i procediments definits pel llenguatge. Aleshores, es passa per paràmetre a la funció recursiva de impressió de codi de tres adreces.

Dins de cada àmbit, les proposicions de tres adreces es troben representades amb una numeració correlativa a partir de 1 per a cada nou àmbit, mostrant en cada línia una nova proposició alhora que s'incrementa el comptador de posició de memòria de la proposició de codi objecte.

En la taula de símbols s'emmagatzema la informació necessària per a cada proposició de tres adreces. En la següent estructura es guarden els tres arguments d'una operació de tres adreces. També hi ha un camp que indica el tipus de operació de tres adreces. Depenent del tipus de operació, només s'utilitzarà un dels arguments, dos d'ells, o bé els tres.

```
typedef struct Tipus_Sentencia {
        Tipus_Operacio operacio;
        char arg1[STRING_LENGTH];
        char arg2[STRING_LENGTH];
        char arg3[STRING_LENGTH];
        Boolean esborrat;        // Si per optimització s'esborra una sentencia, es fica a TRUE
} Tipus_Sentencia;
```

Estructura que conté els diferents tipus de proposicions de tres adreces possibles:

```
typedef enum {
       START, CHSI, CHSF, I2F, F2I, I2C, C2I, ADDI, SUBI, MULI, DIVI, MODI, ADDF, SUBF,
       MULF, DIVF, ASSIGN, ASSIGN_TO_TABLE, ASSIGN_FROM_TABLE, GOTO, IF_EQ, IF_NE, IF_LTI, IF_LEI,
       IF_GTI, IF_GEI, IF_LTF, IF_LEF, IF_GTF, IF_GEF, PARAM, CALL_FUNC, RETURN_FUNC, CALL_PROC,
       RETURN_PROC, PUTC, PUTI, PUTF, GETC, GETI, GETF, HALT
} Tipus_Operacio;
```

Aquestes codificacions de l'enumerat Tipus_Operació permeten tractar tots els tipus de proposicions de codi de tres adreces, que són les següents:

- Inici de subprograma: START name
- Assignació amb operador unari: x := op y
- Assignació amb operador binari: x := y op z
- Còpia: x := y
- Salt incondicional: GOTO L
- Salt condicional: IF x oprel y GOTO L
- Procediments:
  - Paràmetre: PARAM x
  - Crida d'un procediment: CALL p,n
  - Significat: p és el nom únic del procediment cridat, i n el seu nombre d'arguments
  - Retorn d'un procediment: RETURN

  - Procediments de sortida a pantalla predefinits (un únic argument, l'objecte que es vol fer sortir per pantalla): p = "PUTC", "PUTI", "PUTF"
  - Significats: PUT=escriure, sufix C=caràcter, sufix I=enter, sufix F=real
  - Sintaxi: totes les instruccions PARAM que corresponen a una crida han d'estar immediatament abans de la crida a CALL, és dir, no hi pot haver altres proposicions intercalades entre els PARAM o entre el darrer PARAM i el CALL.
- Funcions:
  - Paràmetre: PARAM x
  - Crida d'una funció: z := CALL f,n
  - Significat: f és els subprogrames cridats, i n el seu nombre d'arguments
  - Retorn d'una funció: RETURN x
  - Funcions de lectura de teclat predefinides (sense arguments, i retornen el valor que s'ha introduït per teclat): p = "GETC", "GETI", "GETF"
  - Funció d'adquisició dinàmica de memòria (un argument enter positiu, el nombre de bytes contigus que es volen reservar, a l'estil del calloc del llenguatge C, i retorna un apuntador al bloc reservat, és dir, l'adreça del seu primer byte): p = "ALLOC"
  - Significats: GET=llegir, ALLOC=reservar, sufix C=caràcter, sufix I=enter, sufix F=real
  - Sintaxi: totes les instruccions PARAM que corresponen a una crida han d'estar immediatament abans de la crida a CALL, és dir, no hi pot haver altres proposicions intercalades entre els PARAM o entre el darrer PARAM i el CALL.
  - Exemple: una crida

    ```
    x := 2 + 3 * Get_Integer;
    ```

    a la funció predefinida

    ```
    function Get_Integer return Integer
    ```

    genera codi del tipus

    ```
    $t1 := call GETI,0
    $t2 := 3 MULI $t1
    $t3 := 2 ADDI $t2
    x := $t3
    ```
- Consulta desplaçada: x := y[i]
- Assignació desplaçada: x[i] := y
- Consulta d'un element referenciat per un apuntador: x := *y
- Assignació a un element referenciat per un apuntador: *x := y
- Obtenció d'un apuntador a un element: x := &y
- Finalització de l'execució: HALT

Les regles d'impressió aplicades al codi de tres adreces són:

- Cada proposició de tres adreces ocupa una línia.
- Totes les línies de codi de tres adreces estan numerades consecutivament.
- Pot haver-hi línies no numerades en blanc.
- Pot haver-hi línies no numerades amb comentaris (les quals comencen pel caràcter #).
- Cada subprograma té associat una llista de proposicions de tres adreces numerades començant per 1.
- Entre subprogrames diferents hi ha d'haver, com a mínim, una línia en blanc.

Registres d'Activació

Pel que respecta als Registres d'Activació, aquests són construïts en l'etapa d'impressió posterior a la compilació a partir de tota la informació inclosa en la taula de símbols, en concret amb totes les dades de tots els símbols de tipus function o procedure que hi ha inclosos en cada àmbit.

Tal i com es realitza en la impressió del codi de tres adreces, en la impressió dels registres d'activació no es mostra l'àmbit inicial que inclou els diferents tipus i funcions predefinides en el programa, ja que no aporten cap informació rellevant a mostrar en els registres d'activació.

Tant en la funcionalitat de impressió de codi de tres adreces com en la funcionalitat de impressió dels registres d'activació, per a cada àmbit (funció, procediment) s'inclou, a més del seu nom, el seu nom únic que identifica al àmbit respecte a altres símbols de mateix nom que es trobin en àmbits no visibles. També es mostra la jerarquia de tots els àmbits precedents en els que està inclòs l'àmbit actual.

S'utilitza una funció recursiva, print_registres_activació, que obté tots els procediments i funcions a partir de la taula de símbols del procediment principal. S'ha tingut present el fet de no intercalar un registre d'activació amb informació d'un altre registre d'activació, com també s'ha fet amb el codi de tres adreces i la taula de símbols.

L'estructura d'un registre d'activació generat està formada per el valor de retorn, els paràmetres formals, les dades locals i les dades temporals.

Es consulta a la taula de símbols tota la informació necessària per formar els registres d'activació. Aquestes dades es troven incloses dins la taula de símbols, en les estructures de dades Info_Procedure o Info_Function, segons l'àmbit, es consulta la llista de paràmetres, el valor de retorn (en funcions), i la llista de variables locals i temporals (els dos darrers camps són afegits per poder preparar la informació per al registre d'activació en la taula de símbols durant el processament de l'analitzador sintàctic).

S'han implementat tot un seguit de funcions recursives que permeten calcular el tamany de qualsevol tipus, per complex que sigui. Per exemple: arrays de records, records amb camps que siguin arrays de tipus derivats, etc. Amb això es pot calcular amb exactitut els bytes ocupats per cada valor de retorn, paràmetre formal, o variable local o temporal, amb la qual cosa es pot incorporar un comptador de posició de memòria en el registre d'activació que s'incrementa acumuladament segons el tamany de cada tipus.

S'inclou seguidament, a tall d'exemple, l'aspecte d'un registre d'activació generat en un dels programes de prova:

```
| ------------------------------------------------------------------- |
|                       REGISTRE  D'ACTIVACIO                         |
| ------------------------------------------------------------------- |
|    * Valor de Retorn:                                               |
|         tipus_buit                                                   |
|                         ---------------                             |
|    * Parametres Formals:                                            |
|         (0): nom: param1, tipus: integer, tamany: 2                 |
|         (2): nom: param2, tipus: float, tamany: 4                   |
```

```
|         (6): nom: param3, tipus: string, tamany: 256                 |
|         (262): nom: param4, tipus: boolean, tamany: 1                |
|         (263): nom: param5, tipus: character, tamany: 1              |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    Enllac de Control   |   Enllac d'Acces   |   Estat de la Maquina   |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    * Dades Locals:                                                   |
|         (264): nom: variable1, tipus: character, tamany: 1           |
|         (265): nom: variable2, tipus: integer, tamany: 2             |
|         (267): nom: variable3, tipus: integer_derivat, tamany: 2     |
|         (269): nom: variable4, tipus: boolean, tamany: 1             |
|         (270): nom: variable5, tipus: float, tamany: 4               |
|         (274): nom: variable6, tipus: string, tamany: 256            |
|         (530): nom: variable7, tipus: derivat, r1, tamany: 266       |
|         (796): nom: variable8, tipus: derivat, r2, tamany: 288       |
|         (1084): nom: variable9, tipus: derivat, taula1, tamany: 20   |
|         (1104): nom: variable10, tipus: derivat, taula2, tamany: 2660|
|                        ---------------                               |
|    * Dades Temporals:                                                |
|         (3764): nom: $t0, tipus: integer, tamany: 2                  |
|         (3766): nom: $t1, tipus: derivat, r1, tamany: 266            |
| -------------------------------------------------------------------- |
|  Fi del Registre d'Activacio de l'ambit subprograma1                 |
| -------------------------------------------------------------------- |
```

7.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, de la generació de codi de tres adreces, de la generació dels registres d'activació 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 Codi de Tres Adreces

Les proves d'aquest apartat validen la correcta generació del codi de tres adreces a partir del codi font en MiniAda del programa compilat:

Exemple 1

CODI FONT

```
procedure simple is
        -- ARRAY TYPES
        type vector is array (1..2) of integer;
        type vvv is array(1..2) of vector;

        type finDeSemana is (sabado, domingo);
        type semana is (lunes, martes, miercoles, jueves, viernes);

        type record01 is record
        a: integer;
        b: integer;
        c: integer range 1..2 + 2;
                d: string;
                e: boolean;
                f: vector;
        end record;

        type vector_record01 is array(1..10) of record01;
        type vector_enum is array(1..10) of finDeSemana;

        procedure test_proc(uno: integer) is
        begin
                null;
        end;

    procedure test_proc(uno: integer; dos: integer) is
                resultat01: integer;
    begin
                resultat01 := 1;
        null;
    end;

        function test_func return integer is
                resultat: integer;
        begin
                resultat := 2;
                resultat := 1;
                return 1;
        end test_func;

        function test_func(uno: float) return integer is
                bla: integer;
        begin
```

```
        bla := 2;
    return 2;
end;


entero01: integer;
entero02: integer;
entero03: integer;
entero04: integer;
float01: float;
float02: float;
caracter01: character;
caracter02: character;
vector01: vector;
vector02: vector_record01;
vector03: vvv;
vector04: vector_enum;
dia01: finDeSemana;
estructura01: record01;

cadena: string;
bool: boolean;

DIA_PROVA :semana;

begin
        dia01 := sabado;
        DIA_PROVA := jueves;
        dia01 := dia01;
        put(caracter01);
        put(entero01);
        put(float01);
        put("hola");
        entero01 := length("adios");
        entero01 := length(cadena);
        entero01 := get_integer;
        float01 := get_float;
        caracter01 := get_character;
        entero01 := abs entero01;
        float01 := abs float01;
        entero01 := entero01 ** 10;
        cadena := 'a' & cadena;
        cadena := cadena & 'a';
        cadena := cadena & cadena;
        cadena := "hola" & "adios";
        cadena := cadena & "agur";
        bool := true;
        bool := false;

        if (float01 in 1.4..100.2) then
                entero01 := 0;
        end if;

        if (entero01 in 1..100) then
    entero01 := 0;
end if;

        if (bool = false) then
                bool := false;
        end if;
        if (cadena = "hola") then
                bool := true;
        else
                bool := false;
        end if;

        cadena := "hola";
        estructura01.a := 1;
        estructura01.d := "hola";
        estructura01.e := true;
        estructura01.e := false;
        --estructura01.f(1) := 1;         -- no suportat

        vector02(2).c := 1;
        vector02(2).d := "vector";
        vector02(2).e := true;
```

```
test_proc(1);
test_proc(2, 3);
entero01 := test_func;
entero01 := test_func(2.2);

vector01(2)      := 1;
vector02(2).c := 1;
--vector03(1)(1) := 1;       -- no suportat
vector01(2) := vector01(1);
vector01(1) := entero01;

vector02(1) := vector02(2);

vector04(1) := dia01;
vector04(1) := sabado;
--vector02(1) := dia01;           -- ha de donar error

entero01 := vector02(2).a;
entero01 := vector01(1);

entero01 := 1;
entero01 := 1 * 2 + 4 / 2;        -- := 4
entero01 := entero02;
entero01 := (1 + 2) * 2; -- := 6
entero01 := 4 mod 2;              -- := 0
entero01 := entero01 + 1;
entero01 := entero01 + entero02 + entero03 * entero04;
entero01 := entero01 + 2 * 3;     -- := entero01 + 6
entero01 := - entero01;
entero01 := + entero01;
float01 := 1.2;
float01 := float02;
float01 := float02 + 1;
float01 := - float01;
caracter01 := 'a';
caracter01 := caracter02;
entero01 := entero01 mod 2;

-- Conversions
entero01 := integer(float01);
float01 := float(entero01);
caracter01 := character(entero01);
entero01 := integer(caracter01);
cadena := string(caracter01);
caracter01 := character(cadena);

vector01(1 + entero01) := 1;
vector01(1) := 1;

vector02(2).b := 1;
entero01 := vector02(2).b;
vector02(2).b := vector02(2).b;

if (entero02 not in 2..4) then
        entero02 := 2;
end if;

if (entero01 > 5) then
        entero01 := 1;
else
        entero01 := 2;
end if;

if (entero01 > entero02 and float01 < 2.20 or float01 = 2.2) then
        return;
end if;

if (false and true) then
        entero01 := 0;
end if;

if (true or true and 1 = 1 and 1 < 2 and 3 > 2 and 3 <= 1 and 2 >= 3 and 3 /= 2) then
        null;
end if;
```

```
                if (1 = 2 and 1 = 1 and 1 = 1) then
                        float01 := float02 + 1;
                end if;

                if not (1 = 2) then
                        entero01 := - entero01;
                end if;

                if (1 = 1 and 2 = 2 and 3 = 3 and 1.1 = 1.1) then
                    null;
                end if;

                if (1 < 2 and 2 > 1 and 3 <= 3 and 1.1 >= 1.1 and 1 /= 2) then
                        null;
                end if;

                if (2 = 3 and 1 < 2) then
                        entero01 := 1;
                end if;

                if (entero01 = 1 and 1 = 1) then
                        entero01 := 2;
                end if;

                if (1 < 2 or 2 > 1) then
                        entero01 := 1;
                end if;

                while (2 < 3) loop
                        entero01 := 1;
                 end loop;

                while (entero01 = 1 and float02 > 2.2) loop
                        float01 := 3.2;
                end loop;

                for entero01 in 1..100 loop
                        float02 := 4.3;
                end loop;

                null;
end;
```

RESULTAT

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Practica de Compiladors II - Compilador de MiniAda.
Fitxer descriptor del Codi de Tres Adreces.
Autors: Sergio Blanco Cuaresma i Francisco Jose Aguilar Celdran.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    #------------
    #Ambit simple
    #------------
    # Nom Complet: simple
    # Jerarquia precedent: simple, $MAIN$
    # Representacio amb codi de tres adreces del codi intermig corresponent:

    1: START simple
    2: dia01 := 0
    3: dia_prova := 3
    4: dia01 := dia01
    5: PARAM caracter01
    6: call PUTC,1
    7: PARAM entero01
    8: call PUTI,1
    9: PARAM float01
    10: call PUTF,1
    11: $t0[0] := 4
    12: $t0[1] := 'h'
    13: $t0[2] := 'o'
    14: $t0[3] := 'l'
    15: $t0[4] := 'a'
    16: PARAM $t0
    17: CALL put,1
```

```
18: $t1 := 5
19: entero01 := $t1
20: $t2 := cadena[0]
21: entero01 := $t2
22: $t3 := call GETI,0
23: entero01 := $t3
24: $t4 := call GETF,0
25: float01 := $t4
26: $t5 := call GETC,0
27: caracter01 := $t5
28: IF entero01 GEI 0 GOTO 31
29: $t6 := CHSI entero01
30: GOTO 32
31: $t6 := entero01
32: entero01 := $t6
33: IF float01 GEF 0 GOTO 36
34: $t7 := CHSF float01
35: GOTO 37
36: $t7 := float01
37: float01 := $t7
38: $t8 := 10
39: $t10 := 1
40: $t9 := entero01
41: IF $t10 LEI $t8 GOTO 43
42: GOTO 46
43: $t9 := $t9 MULI 10
44: $t10 := $t10 ADDI 1
45: GOTO 41
46: entero01 := $t9
47: $t11 := 'a'
48: $t14 := cadena[0]
49: $t12[1] := $t11
50: $t13 := 2
51: IF $t13 GTI $t14 GOTO 56
52: $t15 := cadena[$t13]
53: $t12[$t13] := $t15
54: $t13 := $t13 ADDI 1
55: GOTO 51
56: $t14 := $t14 ADDI 1
57: $t12[0] := $t14
58: cadena := $t12
59: $t16 := 'a'
60: $t19 := cadena[0]
61: $t18 := 1
62: IF $t18 GTI $t19 GOTO 67
63: $t20 := cadena[$t18]
64: $t17[$t18] := $t20
65: $t18 := $t18 ADDI 1
66: GOTO 62
67: $t17[$t18] := $t16
68: $t19 := $t19 ADDI 1
69: $t17[0] := $t19
70: cadena := $t17
71: $t24 := cadena[0]
72: $t25 := cadena[0]
73: $t23 := $t24 ADDI $t25
74: $t22 := 1
75: IF $t22 GTI $t24 GOTO 80
76: $t26 := cadena[$t22]
77: $t21[$t22] := $t26
78: $t22 := $t22 ADDI 1
79: GOTO 75
80: IF $t22 GTI $t23 GOTO 85
81: $t26 := cadena[$t22]
82: $t21[$t22] := $t26
83: $t22 := $t22 ADDI 1
84: GOTO 80
85: cadena := $t21
86: cadena[0] := 9
87: cadena[1] := 'h'
88: cadena[2] := 'o'
89: cadena[3] := 'l'
90: cadena[4] := 'a'
91: cadena[5] := 'a'
92: cadena[6] := 'd'
93: cadena[7] := 'i'
```

```
94: cadena[8] := 'o'
95: cadena[9] := 's'
96: $t27[0] := 4
97: $t27[1] := 'a'
98: $t27[2] := 'g'
99: $t27[3] := 'u'
100: $t27[4] := 'r'
101: $t31 := cadena[0]
102: $t32 := $t27[0]
103: $t30 := $t31 ADDI $t32
104: $t29 := 1
105: IF $t29 GTI $t31 GOTO 110
106: $t33 := cadena[$t29]
107: $t28[$t29] := $t33
108: $t29 := $t29 ADDI 1
109: GOTO 105
110: IF $t29 GTI $t30 GOTO 115
111: $t33 := $t27[$t29]
112: $t28[$t29] := $t33
113: $t29 := $t29 ADDI 1
114: GOTO 110
115: cadena := $t28
116: bool := 1
117: bool := 0
118: IF float01 GTF 1.4 GOTO 120
119: GOTO 123
120: IF float01 LTF 100.2 GOTO 122
121: GOTO 123
122: entero01 := 0
123: IF entero01 GTI 1 GOTO 125
124: GOTO 128
125: IF entero01 LTI 100 GOTO 127
126: GOTO 128
127: entero01 := 0
128: IF bool EQ 0 GOTO 130
129: GOTO 131
130: bool := 0
131: $t34[0] := 4
132: $t34[1] := 'h'
133: $t34[2] := 'o'
134: $t34[3] := 'l'
135: $t34[4] := 'a'
136: $t35 := cadena[0]
137: $t36 := $t34[0]
138: IF $t35 NE $t36 GOTO 147
139: IF $t35 EQ 0 GOTO 145
140: $t37 := cadena[$t35]
141: $t38 := $t34[$t35]
142: IF $t37 NE $t38 GOTO 147
143: $t35 := $t35 SUBI 1
144: GOTO 139
145: bool := 1
146: GOTO 148
147: bool := 0
148: cadena[0] := 4
149: cadena[1] := 'h'
150: cadena[2] := 'o'
151: cadena[3] := 'l'
152: cadena[4] := 'a'
153: estructura01[0] := 1
154: estructura01[6] := 4
155: estructura01[7] := 'h'
156: estructura01[8] := 'o'
157: estructura01[9] := 'l'
158: estructura01[10] := 'a'
159: estructura01[262] := 1
160: estructura01[262] := 0
161: vector02[271] := 1
162: vector02[273] := 6
163: vector02[274] := 'v'
164: vector02[275] := 'e'
165: vector02[276] := 'c'
166: vector02[277] := 't'
167: vector02[278] := 'o'
168: vector02[279] := 'r'
169: vector02[529] := 1
```

```
170: PARAM 1
171: CALL test_proc_1,1
172: PARAM 2
173: PARAM 3
174: CALL test_proc_2,2
175: $t39 := CALL test_func_3,0
176: entero01 := $t39
177: PARAM 2.2
178: $t40 := CALL test_func_4,1
179: entero01 := $t40
180: vector01[2] := 1
181: vector02[271] := 1
182: $t41 := vector01[0]
183: vector01[2] := $t41
184: vector01[0] := entero01
185: $t42 := vector02[267]
186: vector02[0] := $t42
187: vector04[0] := dia01
188: vector04[0] := 0
189: entero01 := vector02[267]
190: entero01 := vector01[0]
191: entero01 := 1
192: entero01 := 4
193: entero01 := entero02
194: entero01 := 6
195: entero01 := 0
196: $t43 := entero01 ADDI 1
197: entero01 := $t43
198: $t44 := entero01 ADDI entero02
199: $t45 := entero03 MULI entero04
200: $t46 := $t44 ADDI $t45
201: entero01 := $t46
202: $t47 := entero01 ADDI 6
203: entero01 := $t47
204: entero01 := CHSI entero01
205: float01 := 1.2
206: float01 := float02
207: $t48 := float02 ADDF 1
208: float01 := $t48
209: float01 := CHSF float01
210: caracter01 := 'a'
211: caracter01 := caracter02
212: $t49 := entero01 MODI 2
213: entero01 := $t49
214: $t50 := F2I float01
215: entero01 := $t50
216: $t51 := I2F entero01
217: float01 := $t51
218: $t52 := I2C entero01
219: caracter01 := $t52
220: $t53 := C2I caracter01
221: entero01 := $t53
222: $t54[0] := 1
223: $t54[1] := caracter01
224: cadena := $t54
225: $t55 := cadena[1]
226: caracter01 := $t55
227: $t56 := 1 ADDI entero01
228: vector01[$t56] := 1
229: vector01[0] := 1
230: vector02[269] := 1
231: entero01 := vector02[269]
232: $t57 := vector02[269]
233: vector02[269] := $t57
234: IF entero02 LTI 2 GOTO 236
235: GOTO 239
236: IF entero02 GTI 4 GOTO 238
237: GOTO 239
238: entero02 := 2
239: IF entero01 GTI 5 GOTO 241
240: GOTO 243
241: entero01 := 1
242: GOTO 244
243: entero01 := 2
244: IF entero01 GTI entero02 GOTO 246
245: GOTO 248
```

```
246: IF float01 LTF 2.2 GOTO 250
247: GOTO 248
248: IF float01 EQ 2.2 GOTO 250
249: GOTO 251
250: RETURN
251: GOTO 254
252: GOTO 253
253: entero01 := 0
254: GOTO 256
255: GOTO 256
256: GOTO 257
257: GOTO 258
258: GOTO 259
259: GOTO 262
260: GOTO 262
261: GOTO 262
262: GOTO 267
263: GOTO 264
264: GOTO 265
265: $t58 := float02 ADDF 1
266: float01 := $t58
267: GOTO 268
268: entero01 := CHSI entero01
269: GOTO 270
270: GOTO 271
271: GOTO 272
272: GOTO 273
273: GOTO 274
274: GOTO 275
275: GOTO 276
276: GOTO 277
277: GOTO 278
278: GOTO 281
279: GOTO 280
280: entero01 := 1
281: IF entero01 EQ 1 GOTO 283
282: GOTO 285
283: GOTO 284
284: entero01 := 2
285: GOTO 287
286: GOTO 287
287: entero01 := 1
288: GOTO 289
289: entero01 := 1
290: GOTO 288
291: IF entero01 EQ 1 GOTO 293
292: GOTO 297
293: IF float02 GTF 2.2 GOTO 295
294: GOTO 297
295: float01 := 3.2
296: GOTO 291
297: IF entero01 GTI 1 GOTO 299
298: GOTO 304
299: IF entero01 LTI 100 GOTO 301
300: GOTO 304
301: entero01 := entero01 ADDI 1
302: float02 := 4.3
303: GOTO 297
304: HALT

    #---------------
    #Ambit test_func
    #---------------
    # Nom Complet: test_func_3
    # Jerarquia precedent: test_func, simple, $MAIN$
    # Representacio amb codi de tres adreces del codi intermig corresponent:

    1: START test_func_3
    2: resultat := 2
    3: resultat := 1
    4: RETURN 1
    5: HALT

    #--------------------------------
    #Ambit test_func (sobrecarrega 1)
    #--------------------------------
```

```
        # Nom Complet: test_func_4
        # Jerarquia precedent: test_func, simple, $MAIN$
        # Representacio amb codi de tres adreces del codi intermig corresponent:

        1: START test_func_4
        2: bla := 2
        3: RETURN 2
        4: HALT



        #---------------
        #Ambit test_proc
        #---------------
        # Nom Complet: test_proc_1
        # Jerarquia precedent: test_proc, simple, $MAIN$
        # Representacio amb codi de tres adreces del codi intermig corresponent:

        1: START test_proc_1
        2: HALT

        #-------------------------------
        #Ambit test_proc (sobrecarrega 1)
        #-------------------------------
        # Nom Complet: test_proc_2
        # Jerarquia precedent: test_proc, simple, $MAIN$
        # Representacio amb codi de tres adreces del codi intermig corresponent:

        1: START test_proc_2
        2: resultat01 := 1
        3: HALT
```

Exemple2

CODI FONT

```
procedure prova is

       function func_sobrecarregada(param1: integer; param2: integer) return boolean is
       begin
               if(param1 > param2) then
                       return false;
               else
                       return true;
               end if;
       end func_sobrecarregada;

       function func_sobrecarregada(param1: float; param2: float) return boolean is
          cont: integer := 0;
          index: integer;
          val: float;
       begin
           while(cont < 10) loop
               if(val <= param2) then
                       for index in -5 .. 5 loop
                           val := val + param1 * float(index);
                       end loop;
               else
                       if(func_sobrecarregada(integer(val),integer(param1)) and cont < 5) then
                           val := val * 2;
                       end if;
               end if;
           end loop;
           if(val > param1 and param1 <= param2) then
              return true;
           else
              return false;
           end if;
       end func_sobrecarregada;
```

```
       procedure proc_test(a: integer; b: float; c: string) is
          i: integer;
          numero: integer;
          bool_value: boolean;

          function acumula (base: integer; valor: integer; vegades: integer) return integer is
          begin
             if(vegades > 0) then
                return acumula(base + valor, valor, vegades - 1);
             else
                 return base;
             end if;
          end acumula;
       begin
          if(integer(b) < a) then
             bool_value := func_sobrecarregada(a, a * a);
          else
                numero := acumula(a, integer(b), a + integer(b));
          end if;
       end proc_test;

       bool_value: boolean;
       a1, b1: float;
       a2, b2: integer;

begin

       a1 := 6.0;
       b1 := 12.5 E 100;
       a2 := 6;
       b2 := 8;
       bool_value := func_sobrecarregada(a2, b2);
       bool_value := func_sobrecarregada(a1, b1);
       proc_test(b2, b1, "cadena");

end prova;
```

RESULTAT

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Practica de Compiladors II - Compilador de MiniAda.
Fitxer descriptor del Codi de Tres Adreces.
Autors: Sergio Blanco Cuaresma i Francisco Jose Aguilar Celdran.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    #-----------
    #Ambit prova
    #-----------
    # Nom Complet: prova
    # Jerarquia precedent: prova, $MAIN$
    # Representacio amb codi de tres adreces del codi intermig corresponent:

    1: START prova
    2: a1 := 6
    3: b1 := 12.5
    4: a2 := 6
    5: b2 := 8
    6: PARAM a2
    7: PARAM b2
    8: $t0 := CALL func_sobrecarregada_1,2
    9: bool_value := $t0
    10: PARAM a1
    11: PARAM b1
    12: $t1 := CALL func_sobrecarregada_2,2
    13: bool_value := $t1
    14: PARAM b2
    15: PARAM b1
    16: $t2[0] := 6
    17: $t2[1] := 'c'
    18: $t2[2] := 'a'
    19: $t2[3] := 'd'
    20: $t2[4] := 'e'
    21: $t2[5] := 'n'
    22: $t2[6] := 'a'
```

```
23: PARAM $t2
24: CALL proc_test_3,3
25: HALT

    #--------------
    #Ambit proc_test
    #--------------
    # Nom Complet: proc_test_3
    # Jerarquia precedent: proc_test, prova, $MAIN$
    # Representacio amb codi de tres adreces del codi intermig corresponent:

    1: START proc_test_3
    2: $t0 := F2I b
    3: IF $t0 LTI a GOTO 5
    4: GOTO 11
    5: $t1 := a MULI a
    6: PARAM a
    7: PARAM $t1
    8: $t2 := CALL func_sobrecarregada_1,2
    9: bool_value := $t2
    10: GOTO 19
    11: $t3 := F2I b
    12: $t4 := F2I b
    13: $t5 := a ADDI $t4
    14: PARAM a
    15: PARAM $t3
    16: PARAM $t5
    17: $t6 := CALL acumula_4,3
    18: numero := $t6
    19: HALT

        #-------------
        #Ambit acumula
        #-------------
        # Nom Complet: acumula_4
        # Jerarquia precedent: acumula, proc_test, prova, $MAIN$
        # Representacio amb codi de tres adreces del codi intermig corresponent:

        1: START acumula_4
        2: IF vegades GTI 0 GOTO 4
        3: GOTO 12
        4: $t0 := base ADDI valor
        5: $t1 := vegades SUBI 1
        6: PARAM $t0
        7: PARAM valor
        8: PARAM $t1
        9: $t2 := CALL acumula_4,3
        10: RETURN $t2
        11: GOTO 13
        12: RETURN base
        13: HALT

    #-------------------------
    #Ambit func_sobrecarregada
    #-------------------------
    # Nom Complet: func_sobrecarregada_1
    # Jerarquia precedent: func_sobrecarregada, prova, $MAIN$
    # Representacio amb codi de tres adreces del codi intermig corresponent:

    1: START func_sobrecarregada_1
    2: IF param1 GTI param2 GOTO 4
    3: GOTO 6
    4: RETURN param1
    5: GOTO 7
    6: RETURN param1
    7: HALT

    #----------------------------------------
    #Ambit func_sobrecarregada (sobrecarrega 1)
    #----------------------------------------
    # Nom Complet: func_sobrecarregada_2
    # Jerarquia precedent: func_sobrecarregada, prova, $MAIN$
    # Representacio amb codi de tres adreces del codi intermig corresponent:

    1: START func_sobrecarregada_2
    2: IF cont LTI 10 GOTO 4
```

```
3: GOTO 27
4: IF val LEF param2 GOTO 6
5: GOTO 17
6: IF index GTI -5 GOTO 8
7: GOTO 2
8: IF index LTI 5 GOTO 10
9: GOTO 2
10: index := index ADDI 1
11: $t0 := I2F index
12: $t1 := param1 MULF $t0
13: $t2 := val ADDF $t1
14: val := $t2
15: GOTO 6
16: GOTO 2
17: $t3 := F2I val
18: $t4 := F2I param1
19: PARAM $t3
20: PARAM $t4
21: $t5 := CALL func_sobrecarregada_1,2
22: IF cont LTI 5 GOTO 24
23: GOTO 2
24: $t6 := val MULF 2
25: val := $t6
26: GOTO 2
27: IF val GTF param1 GOTO 29
28: GOTO 33
29: IF param1 LEF param2 GOTO 31
30: GOTO 33
31: RETURN param1
32: GOTO 34
33: RETURN param1
34: HALT
```

Proves de generació de Registres d'Activació

A continuació es valida la correcta generació dels registres d'activació a partir del codi font en MiniAda del programa compilat:

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 dels Registres d'Activacio.
Autors: Sergio Blanco Cuaresma i Francisco Jose Aguilar Celdran.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

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

| ------------------------------------------------------------------- |
|                       REGISTRE  D'ACTIVACIO                         |
| ------------------------------------------------------------------- |
|    * Valor de Retorn:                                               |
|         tipus_buit                                                   |
|                          ---------------                            |
|    * Parametres Formals:                                            |
|         No hi ha parametres.                                         |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    Enllac de Control   |   Enllac d'Acces   |   Estat de la Maquina   |
```

```
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    * Dades Locals:                                                            |
|         (0): nom: enter10, tipus: integer, tamany: 2                          |
|                            ---------------                                    |
|    * Dades Temporals:                                                         |
|         No hi ha variables temporals!                                         |
| ----------------------------------------------------------------------------- |
|  Fi del Registre d'Activacio de l'ambit principal                             |
| ----------------------------------------------------------------------------- |

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

| ----------------------------------------------------------------------------- |
|                        REGISTRE  D'ACTIVACIO                                  |
| ----------------------------------------------------------------------------- |
|    * Valor de Retorn:                                                         |
|         tipus_buit                                                            |
|                            ---------------                                    |
|    * Parametres Formals:                                                      |
|         (0): nom: param1, tipus: integer, tamany: 2                           |
|         (2): nom: param2, tipus: float, tamany: 4                             |
|         (6): nom: param3, tipus: string, tamany: 256                          |
|         (262): nom: param4, tipus: boolean, tamany: 1                         |
|         (263): nom: param5, tipus: character, tamany: 1                       |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    Enllac de Control   |   Enllac d'Acces   |   Estat de la Maquina          |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    * Dades Locals:                                                            |
|         (264): nom: variable1, tipus: character, tamany: 1                    |
|         (265): nom: variable2, tipus: integer, tamany: 2                      |
|         (267): nom: variable3, tipus: integer_derivat, tamany: 2              |
|         (269): nom: variable4, tipus: boolean, tamany: 1                      |
|         (270): nom: variable5, tipus: float, tamany: 4                        |
|         (274): nom: variable6, tipus: string, tamany: 256                     |
|         (530): nom: variable7, tipus: derivat, r1, tamany: 266                |
|         (796): nom: variable8, tipus: derivat, r2, tamany: 288                |
|         (1084): nom: variable9, tipus: derivat, taula1, tamany: 20            |
|         (1104): nom: variable10, tipus: derivat, taula2, tamany: 2660         |
|                            ---------------                                    |
|    * Dades Temporals:                                                         |
|         (3764): nom: $t0, tipus: integer, tamany: 2                           |
|         (3766): nom: $t1, tipus: derivat, r1, tamany: 266                     |
| ----------------------------------------------------------------------------- |
|  Fi del Registre d'Activacio de l'ambit subprograma1                          |
| ----------------------------------------------------------------------------- |

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

| ----------------------------------------------------------------------------- |
|                        REGISTRE  D'ACTIVACIO                                  |
| ----------------------------------------------------------------------------- |
|    * Valor de Retorn:                                                         |
|         (0): tipus: integer, tamany: 2                                        |
|                            ---------------                                    |
|    * Parametres Formals:                                                      |
|         (2): nom: p1, tipus: float, tamany: 4                                 |
|         (6): nom: p2, tipus: boolean, tamany: 1                               |
|         (7): nom: p3, tipus: string, tamany: 256                              |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    Enllac de Control   |   Enllac d'Acces   |   Estat de la Maquina          |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    * Dades Locals:                                                            |
|         (263): nom: v1, tipus: string, tamany: 256                            |
|         (519): nom: v2, tipus: character, tamany: 1                           |
|         (520): nom: v3, tipus: derivat, registre, tamany: 7                   |
|         (527): nom: v4, tipus: derivat, taula, tamany: 200                    |
|         (727): nom: v5, tipus: integer, tamany: 2                             |
|                            ---------------                                    |
```

```
|    * Dades Temporals:                                                  |
|         No hi ha variables temporals!                                  |
| ---------------------------------------------------------------------- |
|  Fi del Registre d'Activacio de l'ambit funcio1                        |
| ---------------------------------------------------------------------- |

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

| ---------------------------------------------------------------------- |
|                       REGISTRE  D'ACTIVACIO                            |
| ---------------------------------------------------------------------- |
|    * Valor de Retorn:                                                  |
|         (0): tipus: integer, tamany: 2                                 |
|                           ---------------                              |
|    * Parametres Formals:                                               |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    Enllac de Control   |   Enllac d'Acces   |   Estat de la Maquina    |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    * Dades Locals:                                                     |
|         No hi ha variables locals.                                     |
|                           ---------------                              |
|    * Dades Temporals:                                                  |
|         No hi ha variables temporals!                                  |
| ---------------------------------------------------------------------- |
|  Fi del Registre d'Activacio de l'ambit funcio2                        |
| ---------------------------------------------------------------------- |

          #--------------
          #Ambit encaixat
          #--------------
          Nom Complet: encaixat_4
          Jerarquia precedent: encaixat, funcio2, principal, $MAIN$

| ---------------------------------------------------------------------- |
|                       REGISTRE  D'ACTIVACIO                            |
| ---------------------------------------------------------------------- |
|    * Valor de Retorn:                                                  |
|         tipus_buit                                                     |
|                           ---------------                              |
|    * Parametres Formals:                                               |
|         (0): nom: primer, tipus: string, tamany: 256                   |
|         (256): nom: segon, tipus: float, tamany: 4                     |
|         (260): nom: tercer, tipus: derivat, cap_de_setmana, tamany: 1  |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    Enllac de Control   |   Enllac d'Acces   |   Estat de la Maquina    |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    * Dades Locals:                                                     |
|         (261): nom: c, tipus: integer, tamany: 2                       |
|         (263): nom: b, tipus: integer, tamany: 2                       |
|         (265): nom: a, tipus: integer, tamany: 2                       |
|         (267): nom: e, tipus: float, tamany: 4                         |
|         (271): nom: d, tipus: float, tamany: 4                         |
|         (275): nom: f, tipus: boolean, tamany: 1                       |
|                           ---------------                              |
|    * Dades Temporals:                                                  |
|         No hi ha variables temporals!                                  |
| ---------------------------------------------------------------------- |
|  Fi del Registre d'Activacio de l'ambit encaixat                       |
| ---------------------------------------------------------------------- |
```

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 dels Registres d'Activacio.
Autors: Sergio Blanco Cuaresma i Francisco Jose Aguilar Celdran.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

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

| ------------------------------------------------------------------- |
|                       REGISTRE  D'ACTIVACIO                         |
| ------------------------------------------------------------------- |
|    * Valor de Retorn:                                               |
|         tipus_buit                                                   |
|                           ---------------                           |
|    * Parametres Formals:                                            |
|         No hi ha parametres.                                         |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    Enllac de Control   |   Enllac d'Acces   |   Estat de la Maquina   |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    * Dades Locals:                                                  |
|         (0): nom: r_simple, tipus: derivat, reg_simple, tamany: 258   |
|         (258): nom: e_simple, tipus: derivat, enum_simple, tamany: 1  |
|         (259): nom: ll_simple, tipus: derivat, llista_simple, tamany: 100|
|         (359): nom: f2, tipus: float, tamany: 4                      |
|         (363): nom: f1, tipus: float, tamany: 4                      |
|         (367): nom: b1, tipus: boolean, tamany: 1                    |
|         (368): nom: i1, tipus: integer, tamany: 2                    |
|         (370): nom: s1, tipus: string, tamany: 256                   |
|         (626): nom: enumerat, tipus: derivat, enum_simple, tamany: 1  |
|         (627): nom: lli1, tipus: derivat, llista1, tamany: 40        |
|         (667): nom: lli2, tipus: derivat, llista2, tamany: 20        |
|         (687): nom: lli3, tipus: derivat, llista3, tamany: 2580      |
|                           ---------------                           |
|    * Dades Temporals:                                               |
|         (3267): nom: $t0, tipus: float, tamany: 4                    |
|         (3271): nom: $t1, tipus: float, tamany: 4                    |
|         (3275): nom: $t2, tipus: integer, tamany: 2                  |
|         (3277): nom: $t3, tipus: string, tamany: 256                 |
| ------------------------------------------------------------------- |
|  Fi del Registre d'Activacio de l'ambit principal                   |
| ------------------------------------------------------------------- |

       #--------------
       #Ambit funcio_a
       #--------------
       Nom Complet: funcio_a_1
       Jerarquia precedent: funcio_a, principal, $MAIN$

| ------------------------------------------------------------------- |
|                       REGISTRE  D'ACTIVACIO                         |
| ------------------------------------------------------------------- |
|    * Valor de Retorn:                                               |
|         (0): tipus: integer, tamany: 2                               |
|                           ---------------                           |
|    * Parametres Formals:                                            |
|         (2): nom: val1, tipus: float, tamany: 4                      |
|         (6): nom: val2, tipus: float, tamany: 4                      |
|         (10): nom: e, tipus: derivat, enum_simple, tamany: 1         |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    Enllac de Control   |   Enllac d'Acces   |   Estat de la Maquina   |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    * Dades Locals:                                                  |
|         (11): nom: int_value, tipus: integer, tamany: 2              |
|         (13): nom: dec_value, tipus: float, tamany: 4                |
|         (17): nom: char, tipus: character, tamany: 1                 |
```

```
|         (18): nom: bool, tipus: boolean, tamany: 1                    |
|                          ---------------                              |
|    * Dades Temporals:                                                 |
|         No hi ha variables temporals!                                 |
| --------------------------------------------------------------------- |
|  Fi del Registre d'Activacio de l'ambit funcio_a                      |
| --------------------------------------------------------------------- |

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

| --------------------------------------------------------------------- |
|                       REGISTRE  D'ACTIVACIO                           |
| --------------------------------------------------------------------- |
|    * Valor de Retorn:                                                 |
|         (0): tipus: string, tamany: 256                               |
|                          ---------------                              |
|    * Parametres Formals:                                              |
|         (256): nom: b, tipus: boolean, tamany: 1                      |
|         (257): nom: l1, tipus: derivat, llista1, tamany: 40           |
|         (297): nom: l2, tipus: derivat, llista2, tamany: 20           |
|         (317): nom: l3, tipus: derivat, llista3, tamany: 2580         |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    Enllac de Control   |   Enllac d'Acces   |   Estat de la Maquina    |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    * Dades Locals:                                                    |
|         (2897): nom: int_value, tipus: integer, tamany: 2             |
|         (2899): nom: dec_value, tipus: float, tamany: 4               |
|         (2903): nom: char, tipus: character, tamany: 1                |
|         (2904): nom: bool, tipus: boolean, tamany: 1                  |
|         (2905): nom: var2, tipus: derivat, llista2, tamany: 20        |
|         (2925): nom: var3, tipus: derivat, llista3, tamany: 2580      |
|                          ---------------                              |
|    * Dades Temporals:                                                 |
|         (5505): nom: $t0, tipus: string, tamany: 256                  |
| --------------------------------------------------------------------- |
|  Fi del Registre d'Activacio de l'ambit funcio_b                      |
| --------------------------------------------------------------------- |

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

| --------------------------------------------------------------------- |
|                       REGISTRE  D'ACTIVACIO                           |
| --------------------------------------------------------------------- |
|    * Valor de Retorn:                                                 |
|         (0): tipus: integer, tamany: 2                                |
|                          ---------------                              |
|    * Parametres Formals:                                              |
|         (2): nom: reg, tipus: derivat, reg_compost, tamany: 3899      |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    Enllac de Control   |   Enllac d'Acces   |   Estat de la Maquina    |
| - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - |
|    * Dades Locals:                                                    |
|         (3901): nom: int_value, tipus: integer, tamany: 2             |
|         (3903): nom: dec_value, tipus: float, tamany: 4               |
|         (3907): nom: char, tipus: character, tamany: 1                |
|         (3908): nom: bool, tipus: boolean, tamany: 1                  |
|         (3909): nom: reg_local, tipus: derivat, reg_compost, tamany: 3899|
|                          ---------------                              |
|    * Dades Temporals:                                                 |
|         No hi ha variables temporals!                                 |
| --------------------------------------------------------------------- |
|  Fi del Registre d'Activacio de l'ambit funcio_c                      |
| --------------------------------------------------------------------- |
```

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

(mateix cas que exemple1 de prova de registre d'activació)

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

(mateix cas que exemple2 de prova de registre d'activació)

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.