Projet de recherche Lambda.x GSM, l'interpréteur Scheme

Laboratoire I3S

Informatique, Signaux et Système de Nice - Sophia-Antipolis

Etude préalable de D.E.A.

- 1992-1993

Sujet :

- Implémentation d'un interpréteur SCHEME

Sous la responsabilité de :

- Professeur F. Boéri,
- Professeur J. Demartini

Etudiant :

- G. de Wailly

Laboratoire d'informatique Signaux Système

❑ 41 Bd Napoléon III - 06041 Nice cedex France. Tel.(33) 93 21 79 56
❑ Bat. 4 - 250 Av Albert Einstein - Sophia Antipolis - F 06560 Valbonne

Rédigé en mai-juin 93 sur traitement de texte Word© 2.0c pour Windows© 3.01 de MicroSoft®

Table des Matieres

SYMBOLES & CONVENTIONS

Nous présentons ici les symboles et convention utilisées dans ce rapport.

👍 Indique une proposition juste ou valide.

👎 Indique une proposition fausse ou erronée.

✋ Indique une proposition importante. En général elle est restrictive.

☞ Indique une proposition importante.

💾 Indique une proposition dont il faut se rappeler.

📖 Indique une référence bibliographie qui figure ou non dans la bibliographie.

Indique un commentaire, une précision.

Le texte du rapport est dans cette police de caractère. Les anglissimes apparaîssent en *italique*. Les exemples, le code dans un langage informatique

```
apparaîssent comme cela.
```

TETE DE CHAPITRE

Titre de niveau 3

Titre de niveau 4

Titre de niveau 5

INTRODUCTION

Ce rapport présente en annexe le code source de l'implémentation de Scheme réalisée au laboratoire, *gsm*. Nous donnons avant de présenter le fichier de code une explication sommaire des principes mis en oeuvre.

FICHIERS DU PROJET

Structure du répertoire pour le volume GUILHEM
Le numéro de série du volume est 1AAA-B030

```
 GSM                  Répertoire de gsm
|-MAKEFILE.SUC        Makefile
|-MAKEFILE.PDB
|-MAKEFILE.PDQ
|-ARCH.LST            Liste d'archivage
|-GSM.PRJ             Projet Borland c++
|
|-ARCH                Répertoire d'archive
|-----930603.ZIP
|-----930604.ZIP
|-----930605.ZIP
|-----930608.ZIP
|-----930610.ZIP
|-----930611.ZIP
|
|-LOADLIB             Répertoire de test des libraries
|-----SERVER.PDQ      Makefile server avec Quick c
|-----USR.PDQ         Makefile usr avec Quick c
|-----BMAKE.BAT       Batch makefile pour Borland c
|-----SERVER.PDB      Makefile server avec Borland c
|-----QMAKE.BAT       Batch makfile pour Quick c
|-----SERVER.C        Source server
|-----USR.PDB         Makefile usr pour Borland c
|-----SERVER.PRJ      Projet server Borland c
|-----USR.C           Source usr
|-----USR.PRJ         Projet usr pour Borland c
|
|-MAKE                Répertoire des fichiers makefile
|-----GSM.PDB         Makefile gsm pour Borland c
|-----GSM.SUC         Makefile gsm pour Sun c
|-----GSM.PDQ         Makefile gsm pour Quick c
|
|-DOC                 Répertoire de documantation
|-----GSM.D           Doc de gsm
|
|-BIN                 Répertoire binaire et batch
|-----MAKE            Batch makefile pour gsm sous sun
|-----CLEAN.BAT       Efface les résidus de compilation
|-----BMAKE.BAT       Batch makefile gsm pour Borland c
```

```
|-----QMAKE.BAT       Batch makefile gsm pour Quick c
|-----DSK.BAT         Copie des fichiers sur disquette
|-----CLEAN           Efface les résidus de compilation
|-----DSK             Copie des fichiers sur disquette
|-----GSM.EXE         Exécutable gsm
|-----DELOBJ.BAT      Effacer les fichier tmp/*.obj
|
|-GLIB                Librairie Scheme
|-----GSM.S           Fichier d'initialisation de gsm
|-----HELP.S          Programme d'aide
|-----LIB.S           Utilisation des librairies
|-----SYSTEM.S        Interface multi-système
|-----TEST.S          Fichier de test de gsm
|
|-GSM                 Répertoire des sources de gsm
|-----ATOM.C          Gestionnaire des atomes
|-----CONV.C          Procédures de conversion
|-----DISPLAY.C       Procédures d'interface
|-----DYNAMIC.C       Gestionnaire des librairies
|-----ENV.C           Gestionnaire des environnements
|-----ERROR.C         Gestionnaire d'erreur
|-----EVAL.C          Evaluateur Scheme
|-----GARBAGE.C       Gestionnaire du garbage
|-----HASH.C          Table des symboles
|-----INIT.C          Initialisation et fermeture gsm
|-----IS.C            Procédures de tests Scheme
|-----KEYWORD.C       Mots clefs de Scheme
|-----LAMBDA.C        Procédures liées aux λ-exp.
|-----MAIN.C          Noyeau de gsm
|-----MATH.C          Noyeau des opérateurs artith.
|-----MATHADD.C       Math.+
|-----MATHDIV.C       Math./
|-----MATHMUL.C       Math.*
|-----MATHSUB.C       Math.-
|-----SIGNAL.C        Gestionnaires des signaux système
|-----STRING.C        Manipulation des chaînes Scheme
|-----VECTOR.C        Vecteurs Scheme
|-----ANALYSIS.C      Analyseur syntaxique
|-----HEAP.C          Gestionnaire du tas
|-----STACK.C         Gestionnaire de la pile
|-----GSMAPI.C        Interface util. des Lib dyn.
|-----LOADLIB.C       Interface gsm des lib. dyn.
|
|-GARCH               Archiveur de programme
|-----GARCH.C
|-----MAKEFILE
|-----GARCH.D
|
|-HEAP                Répertoire de test du tas
|-----HEAP.PDB        Makefile borland c
|-----QMAKE.BAT       Batch makefile Quick c
|-----HEAP.PRJ        Projet Borland c
|-----TEST.BAT        Batch de tests complets
|-----MAIN.C          Source du test
|-----HEAP.PDQ        Makefile Quick c
```

```
|-----BMAKE.BAT       Batch makefile Borland c
|-----HEAP.EXE        Exécutable du testeur
|
|-TMP                 Répertoire temporaire de comp.
|-----HEAP.OBJ
|-----ATOM.OBJ
|-----CONIO.OBJ
|-----CONV.OBJ
|-----DISPLAY.OBJ
|-----DYNAMIC.OBJ
|-----ENV.OBJ
|-----ERROR.OBJ
|-----EVAL.OBJ
|-----GARBAGE.OBJ
|-----HASH.OBJ
|-----ANALYSIS.OBJ
|-----INIT.OBJ
|-----KEYWORD.OBJ
|-----MAIN.OBJ
|-----MATH.OBJ
|-----SIGNAL.OBJ
|-----STRING.OBJ
|-----STACK.OBJ
|-----VECTOR.OBJ
|-----LOADLIB.OBJ
|
|-LIB                 Librairie de gsm
|-----STUB.EXE        Stub pour Windows
|-----GSM.LIB         Librairie gsm
|
|-INCLUDE             Répertoire des fichiers inclus
|----- GSM.H          Fichier principal
|----- GSMAPI.H       Librairie dynamiques utilisateur
|----- GSMSERVR.H     Exportation gsm vers librairies
|----- LOADLIB.H      Librairies dynamiques serveur
|----- CONFIG.H       Options compilées de gsm
```

Avancement du projet

Le projet tel qu'il est présenté ici implémente les principales fonctionnalités de Scheme. Sont implémentés :

- □ le gestionnaire ramasse-miettes,
- □ la pile,
- □ le gestionnaire de signaux,
- □ le gestionnaires des erreurs,
- □ la récupération sur erreur (différent niveau d'erreurs),
- □ les vecteurs,
- □ les types de données (caractères, entiers, réels, chaînes, complexes),
- □ les type de procédure (formes normales, formes spéciales, formes lambda, procédures réservées, etc.),
- □ les tables de symboles,
- □ les environnements (*top-level* et temporaires de compilation),
- □ l'évaluateur,
- □ la compilation des lambda expressions,
- □ les procédures lambda, let, letrec et let*,
- □ les opérateurs mathématiques complets (opérant sur tous les types de données),
- □ le gestionnaire de bibliothèques dynamiques pour MS DOS.

Sont en version minimale :

- □ le tas sous DOS (problèmes survenus dernièrement),
- □ l'analyseur syntaxique,
- □ la gestion des ports d'entrées/sorties.

Restent à écrire :

- □ les continuations Scheme,
- □ les promesses Scheme
- □ un certain nombre de procédures non essentielles de Scheme, telles que les manipulation de chaînes de caractères.

La version actuelle met en oeuvre toutes les caractéristiques de Scheme, hormis les continuations et les promesses. L'analyseur sera réécrit comme une machine à état finis selon le formalisme proposé par J.Demartini. Le tas sous DOS se heurte au problème de la mémoire segmentée des Ibm Pc. La version en mode 386 protégé est à l'étude (espace mémoire e 4 giga Octets !).

COMPILATION

Formalisme

Nous présentons ici le formalisme syntaxique et structurel utilisé dans l'écriture de *gsm*.

Il n'y a qu'un fichier à inclure dans les fichiers sources, `gsm.h`. Dans ce fichier sont regroupées toutes les inclusions générales, les définitions de structures globales dépendantes ou non de la configuration choisie. Les définitions de type à usage local sont faites dans les fichier c.

La configuration (dépendance machine, système d'exploitation et compilateur) est définie dans un fichier unique, `config.h`, ayant une structure redondante pour permettre de créer aisément de nouvelle configuration.

Les bibliothèques dynamiques étant complètement dépendantes de la machine, les définitions les concernant sont regroupées à part pour pouvoir les utiliser dans d'autres programmes que *gsm*.

Il n'y a pas de variable globale (hormis pour les bibliothèques dynamiques). Les paramètres à usage général de l'application sont regroupés dans une structure unique nommée MAIN et passée en premier paramètre de toutes les fonctions.

Les fonctions Scheme (utilisant des paramètres sous forme de structures Scheme) ne sont pas préfixées. Les autres, à usage interne, sont préfixées par '_'. Ces dernières reçoivent des paramètres de type c (char, int, etc.). en règles générale, plus il y a de '_' en préfixe, plus la fonction est primitive.

Seules sont globales au projet les fonctions dont le prototype est présent dans le fichier d'inclusion `gsm.h`. Toutes les autres sont déclarées statiques dans les fichiers sources.

Le prototype des fonctions dépend du compilateur (c ANSI ou non). une macro-définition PROTO() définie dans `gsm.h` permet de produire des prototypes adaptés au compilateur.

Tous les messages destinés à l'utilisateur sont regroupés dans les fichiers `error.c` et `display.c`. Ainsi, il est facile de modifier le *look-and-feel* de *gsm*.

Répertoires

Les répertoires de *gsm* sont organisés "à la manière UNIX".

- ❑ ARCH/ : archivage,
- ❑ BIN/ : utilitaires et exécutable *gsm*,
- ❑ DOC/ : documentations,
- ❑ GLIB/ : librairies en SCHEME,
- ❑ GSM/ : sources C,
- ❑ INCLUDE/ : fichiers à inclure,

- LIB/ : librairies C,
- MAKE/ : makefile,
- TMP/ : fichiers temporaires de compilation,

On trouve ensuite des répertoires de tests incluant un mini-projet servant à vérifier le comportement correct d'un point particulier de *gsm*.

- LOADLIB/ : gestion des bibliothèques à liens dynamiques sous DOS. Présente une application client-serveur de base.
- HEAP/ : gestion du tas sous DOS.

Configuration

gsm est un programme multi-plateforme. Il est de ce fait hautement configurable. Le choix de la machine hôte et du système d'exploitation se fait dans le fichier `config.h` en définissant une macro-définition de la forme :

`__<machine>_<système>_compilateur.`

Les Macro actuellement reconnues sont :

- __PC_DOS_BC : pour une compilation pour Ibm Pc sous Dos avec le compilateur Borland C++ 3.1,
- __PC_DOS_QC : pour une compilation pour Ibm Pc sous Dos avec le compilateur Quick C 2.5,
- __PC_WINDOWS_BC : pour une compilation pour Ibm Pc sous Windows avec le compilateur Borland c++ 3.1,
- __SUN_UNIX_CC : pour une compilation pour Sun Sparc 2 sous Unix avec le compilateur Sun c,

Cette Macro est à définir avant toute compilation.

Un certain nombre de macro-définitions permettent de régler le comportement de *gsm*, indépendamment de la configuration matérielle.

Déboggage :

- __DEBUG_GARBAGE : permet de débogger la gestion du garbage.
- __DEBUG_HEAP : permet de débogger la gestion du tas.
- __DEBUG : active la vérification des assertions (_assert()). Cette option ralentie considérablement *gsm* et n'est utilisée qu'en mode de déboggage. Si cette Macro n'est pas définie, les deux précédentes sont automatiquement désactivées.

Nombres réels :

- __FLOAT : les nombres réels seront du type c float,
- __DOUBLE : ils seront du type double,
- __LDOUBLE : ils seront du type long double

En l'absence de l'un ce ces choix, les nombres réels ne sont pas supportés par *gsm*.

Bibliothèques dynamiques :

Pour __PC_DOS_BC, __PC_DOS_QC et __PC_WINDOWS_BC uniquement :

- __DYNAMIC : permet d'ajouter à *gsm* la gestion des bibliothèques dynamiques.

Gestion de la mémoire :

Pour DOS uniquement.

- __PC_DOS_REALMODE : la gestion de la mémoire se fait en mode réel (dans la limite des 640 Ko du Dos). La mémoire disponible est au maximum de 640 Ko.
- __PC_DOS_PROTECTEDMODE : la gestion de la mémoire dynamique se fait en mode protégé (au-delà de la limite des 640 Ko). Bien sûr, cette option n'est

disponible que pour les **processeurs 80386** et ultérieurs de Intel. Le protocole utilisé est DPMI (*Dos Protected Mode Interface*) de MicroSoft. La taille de la mémoire vive n'est plus limitée à 640 Ko.

gsm permet de configurer ses principale options (gestion de la mémoire) sur la ligne de commande. En l'absence d'indication, il prend des valeurs par défaut. Ces valeurs se trouvent dans le fichier `gsm.h`, sous la variable qui va utiliser cette valeur.
On trouve :

- PROMPT_LENGTH 50 : indique la taille du buffer destiné à recevoir le `prompt`.
- DEFAULT_PROMPT[+*] "GSM->" : le `prompt` par défaut,
- DEFAULT_HEAP_SIZE[+] 50000 : taille du tas. Le tas contient le garbage, donc il est préférable d'en tenir compte.
- DEFAULT_GARBAGE_SIZE[+] 5000 : taille du garbage en cellule. Une cellule fait comme taille 2 pointeurs + 1 mots, soit sur un pc, 32 + 32 + 16 = 80 bits = 10 octets
- DEFAULT_STACK_SIZE[+] 1000 = taille de la pile en cellules. La pile est allouée dans le tas, il faut donc en tenir compte.
- DEFAULT_HASH_SIZE[+] 100 : taille de la table des symboles principale. Il n'est pas nécessaire d'avoir une valeur trop importante.
- DEFAULT_HASH_TEMP_SIZE[+] 50 : taille des tables temporaires de compilation.
- DEFAULT_BUFFER_SIZE[+] 500 : taille du buffer d'analyse. C'est dans ce buffer que sont stockés au moment de l'analyse les noms des identificateurs.
- DEFAULT_IDENTIFIER_LENGTH 20 : nombre de caractères maximum des identificateurs. Une taille de 0 ne limite plus leur taille. il est préférable de donner une limite pour prévenir le chargement par erreur de fichiers binaires.
- DEFAULT_REDEFINE_SYMBOL[*] WARNING : erreur en cas des redéfinition des symboles. Ce champ peut prendre comme valeur `OK` (pas d'erreur), `WARNING` (avertissement), `ERR` (erreur, mais poursuite de l'analyse), `GOTOP` (abandon de l'analyse en cours), `FATAL` (quitter GSM), `EXIT` (quitter *gsm* en catastrophe - à déconseiller).
- DEFAULT_EXTENDED_SYNTAXE[*] `OK` : autorise ou non la syntaxe étendue de *gsm*. Ce champs prend les mêmes valeurs que le précédent.
- DEFAULT_VERBOSE_EVAL[*] 0 : évaluation bavarde. 0 ou 1.
- DEFAULT_DISPLAY_RESERVED[*] 0 : afficher les mots réservés en cas de consultation de la table des symboles. 0 ou 1.

(Les option suivies de + sont peuvent être définies sur la ligne de commande, celles suivies de * sont accessibles par procédure dans *gsm*.

Makefile

Les fichiers `makefile` de *gsm* sont nommés GSM.XXX où XXX indique la machine, le système et le compilateur utilisé.

- PDB : pour PC, DOS, BORLAND C
- PDQ : pour PC, DOS, QUICK C
- SUC : pour SUN, UNIX, CC

PC, DOS, BORLAND C++

```
#
# gsm.pdb : gsm makefile for Ibm pc under DOS with Borland c++ 3.1.
# command : make -f gsm.pdb
# Copyright (C) 1993 Guilhem de Wailly.
#

.AUTODEPEND

#Directories
HOMEGSM = ..
TMP     = $(HOMEGSM)\tmp
BIN     = $(HOMEGSM)\bin
INCLUDE = $(HOMEGSM)\include
MAKE    = $(HOMEGSM)\make
GSM     = $(HOMEGSM)\gsm
BCLIB   = \pgm\bc3\lib
BCINCLUDE = \pgm\bc3\include


#project
PROJECT_NAME    = gsm
PROJECT         = $(BIN)\$(PROJECT_NAME).exe

#Tools
COMPILE_FLAGS   = +$(TMP)\$(PROJECT_NAME).cfg -c
LINK_FLAGS      = /v/x/c/P-/L$(BCLIB)
CC              = bcc
LINK            = tlink

#Dependency
OBJ_dependency =
        $(TMP)\analysis.obj $(TMP)\atom.obj   $(TMP)\conio.obj   $(TMP)\display.obj \
        $(TMP)\dynamic.obj  $(TMP)\env.obj    $(TMP)\error.obj   $(TMP)\eval.obj    \
        $(TMP)\garbage.obj  $(TMP)\hash.obj   $(TMP)\heap.obj    $(TMP)\init.obj    \
        $(TMP)\keyword.obj  $(TMP)\loadlib.obj$(TMP)\main.obj    $(TMP)\math.obj    \
        $(TMP)\signal.obj   $(TMP)\stack.obj  $(TMP)\vector.obj

MAKE_dependency=$(MAKE)\gsm.pdb
INCLUDE_dependency = $(INCLUDE)\config.h $(INCLUDE)\gsm.h $(INCLUDE)\loadlib.h  \
                     $(INCLUDE)\gsmapi.h $(INCLUDE)\server.h $(MAKE_dependency)
LIB_dependency =$(MAKE)\lib.pdb
EXE_dependency =$(INCLUDE_dependency) $(OBJ_dependency) $(MAKE_dependency)
$(LIB_dependency)



$(PROJECT): $(TMP)\$(PROJECT_NAME).cfg $(EXE_dependency)
  $(LINK) $(LINK_FLAGS) @&&|
c0l.objbj+
$(TMP)\analysis.obj+
$(TMP)\atom.obj+
$(TMP)\conio.obj+
$(TMP)\display.obj+
$(TMP)\dynamic.obj+
$(TMP)\env.obj+
$(TMP)\error.obj+
$(TMP)\eval.obj+
$(TMP)\garbage.obj+
$(TMP)\hash.obj+
$(TMP)\heap.obj+
$(TMP)\init.obj+
$(TMP)\keyword.obj+
$(TMP)\loadlib.obj+
$(TMP)\main.obj+
$(TMP)\math.obj+
$(TMP)\signal.obj+
$(TMP)\stack.obj+
$(TMP)\vector.obj+
$(BCLIB)\wildargs.objbj
$(BIN)\$(PROJECT_NAME)
                # no map file
emu.lib+
mathl.lib+
ch.lib
```

```
|

$(TMP)\analysis.obj: $(GSM)\analysis.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\analysis.obj $(GSM)\analysis.c

$(TMP)\atom.obj:  $(GSM)\atom.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\atom.obj $(GSM)\atom.c

$(TMP)\conio.obj: $(GSM)\conio.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\conio.obj $(GSM)\conio.c

$(TMP)\dynamic.obj: $(GSM)\dynamic.c $(INCLUDE_dependency)
       $(CC) $(COMPILE_FLAGS) -o$(TMP)\dynamic.obj $(GSM)\dynamic.c

$(TMP)\display.obj: $(GSM)\display.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\display.obj $(GSM)\display.c

$(TMP)\env.obj:   $(GSM)\env.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\env.obj $(GSM)\env.c

$(TMP)\error.obj: $(GSM)\error.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\error.obj $(GSM)\error.c

$(TMP)\eval.obj:  $(GSM)\eval.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\eval.obj $(GSM)\eval.c

$(TMP)\garbage.obj: $(GSM)\garbage.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\garbage.obj $(GSM)\garbage.c

$(TMP)\hash.obj:  $(GSM)\hash.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\hash.obj $(GSM)\hash.c

$(TMP)\heap.obj:  $(GSM)\heap.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\heap.obj $(GSM)\heap.c

$(TMP)\init.obj:  $(GSM)\init.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\init.obj $(GSM)\init.c

$(TMP)\keyword.obj: $(GSM)\keyword.c $(GSM)\conv.c $(GSM)\string.c $(GSM)\lambda.c \
                $(GSM)\is.c $(INCLUDE_dependency)
        $(CC) -I$(GSM) $(COMPILE_FLAGS) -o$(TMP)\keyword.obj $(GSM)\keyword.c

$(TMP)\loadlib.obj: $(GSM)\loadlib.c $(INCLUDE_dependency)
        $(CC) -I$(GSM) $(COMPILE_FLAGS) -o$(TMP)\loadlib.obj $(GSM)\loadlib.c

$(TMP)\main.obj:  $(GSM)\main.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\main.obj $(GSM)\main.c

$(TMP)\math.obj:  $(GSM)\math.c $(GSM)\mathadd.c $(GSM)\mathsub.c $(GSM)\mathmul.c \
                $(GSM)\mathdiv.c $(INCLUDE_dependency)
        $(CC) -I$(GSM) $(COMPILE_FLAGS) -o$(TMP)\math.obj $(GSM)\math.c

$(TMP)\signal.obj: $(GSM)\signal.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\signal.obj $(GSM)\signal.c

$(TMP)\stack.obj: $(GSM)\stack.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\stack.obj $(GSM)\stack.c

$(TMP)\vector.obj: $(GSM)\vector.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) -o$(TMP)\vector.obj $(GSM)\vector.c

#               *Compiler Configuration File*
$(TMP)\$(PROJECT_NAME).cfg: $(MAKE)\$(PROJECT_NAME).pdb
  copy &&|
-mh
-3
-a
-A
-C
-v
-y
-G
```

```
-O
-Og
-Oe
-Om
-Ov
-Ol
-Ob
-Op
-Oi
-Z
-k-
-d
-H=$(TMP)\$(PROJECT_NAME).SYM
-Fc
-wbbf
-w-lin
-w-lvc
-w-nci
-w-inl
-w-obi
-w-ofp
-w-ovl
-wpin
-wamb
-wamp
-wasm
-wpro
-wcln
-wdef
-wsig
-wnod
-wstv
-wucp
-wuse
-w-hid
-w-ncf
-w-ibc
-w-dsz
-w-nst
-I$(INCLUDE)
-I$(BCINCLUDE)
-D__DEBUG
-D__DEMO
-D__PC_DOS_BC
| $(TMP)\$(PROJECT_NAME).cfg
```

PC, DOS, Quick C

```
#
# gsm.pdq : gsm makefile for Ibm pc under DOS with Quick c 2.
# command : nmake /F gsm.pdq
# Copyright (C) 1993 Guilhem de Wailly.
#

#Directories
HOMEGSM = ..
TMP     = $(HOMEGSM)\tmp
BIN     = $(HOMEGSM)\bin
INCLUDE = $(HOMEGSM)\include
MAKE    = $(HOMEGSM)\make
GSM     = $(HOMEGSM)\gsm

#Compiler
COMPILE_FLAGS   = /D__PC_DOS_QC /I$(INCLUDE) /I\pgm\qc\include /C /AH /W0 /Za \
        /Ox /G2 /Zl /FPi /c
LINK_FLAGS      = /CP:0xfff /NOI /SE:0x80 /ST:0x4e20 /E /F /PACKCODE
CC              = qcl
```

```
LINK            = qlink

#Dependency
PROJECT_NAME    = gsm
PROJECT         = $(BIN)\$(PROJECT_NAME).exe
INCLUDE_dependency = $(INCLUDE)\config.h $(INCLUDE)\gsm.h \
        $(INCLUDE)\loadlib.h $(INCLUDE)\gsmapi.h
OBJ_dependency = \
        $(TMP)\analysis.obj $(TMP)\atom.obj    $(TMP)\conio.obj $(TMP)\display.obj \
        $(TMP)\dynamic.obj  $(TMP)\env.obj     $(TMP)\error.obj $(TMP)\eval.obj    \
        $(TMP)\garbage.obj  $(TMP)\hash.obj    $(TMP)\heap.obj  $(TMP)\init.obj    \
        $(TMP)\keyword.obj  $(TMP)\loadlib.obj $(TMP)\main.obj  $(TMP)\math.obj    \
        $(TMP)\signal.obj   $(TMP)\stack.obj   $(TMP)\vector.obj
MAKE_dependency = $(MAKE)\gsm.pdq
MSLIB           = \pgm\qc\lib\LLIBCE.LIB
LIB_dependency  =
EXE_dependency  = $(INCLUDE_dependency) $(OBJ_dependency) $(MAKE_dependency)
$(LIB_dependency)

$(PROJECT):     $(EXE_dependency)
                echo >NUL @<<$(TMP)\$(PROJECT_NAME).crf
$(TMP)\analysis.obj +
$(TMP)\atom.obj +
$(TMP)\display.obj +
$(TMP)\dynamic.obj +
$(TMP)\env.obj +
$(TMP)\error.obj +
$(TMP)\eval.obj +
$(TMP)\garbage.obj +
$(TMP)\hash.obj +
$(TMP)\heap.obj +
$(TMP)\init.obj +
$(TMP)\keyword.obj +
$(TMP)\loadlib.obj +
$(TMP)\main.obj +
$(TMP)\math.obj +
$(TMP)\signal.obj +
$(TMP)\stack.obj +
$(TMP)\vector.obj
$(PROJECT)

$(MSLIB) ;
<<
  $(LINK) $(LINK_FLAGS) @$(TMP)\$(PROJECT_NAME).crf


$(TMP)\analysis.obj: $(GSM)\analysis.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\analysis $(GSM)\analysis.c

$(TMP)\atom.obj:  $(GSM)\atom.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\atom $(GSM)\atom.c

$(TMP)\display.obj: $(GSM)\display.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\display $(GSM)\display.c

$(TMP)\dynamic.obj: $(GSM)\dynamic.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\dynamic $(GSM)\dynamic.c

$(TMP)\env.obj:   $(GSM)\env.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\env $(GSM)\env.c

$(TMP)\error.obj: $(GSM)\error.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\error $(GSM)\error.c

$(TMP)\eval.obj:  $(GSM)\eval.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\eval $(GSM)\eval.c

$(TMP)\garbage.obj: $(GSM)\garbage.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\garbage $(GSM)\garbage.c

$(TMP)\hash.obj:  $(GSM)\hash.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\hash $(GSM)\hash.c
```

```
$(TMP)\heap.obj:  $(GSM)\heap.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\heap $(GSM)\heap.c

$(TMP)\init.obj:  $(GSM)\init.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\init $(GSM)\init.c

$(TMP)\keyword.obj: $(GSM)\keyword.c $(GSM)\conv.c $(GSM)\string.c $(GSM)\lambda.c \
        $(GSM)\is.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\keyword $(GSM)\keyword.c

$(TMP)\loadlib.obj:  $(GSM)\loadlib.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\loadlib.obj $(GSM)\loadlib.c

$(TMP)\main.obj:  $(GSM)\main.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\main.obj $(GSM)\main.c

$(TMP)\math.obj:  $(GSM)\math.c $(GSM)\mathadd.c $(GSM)\mathsub.c $(GSM)\mathmul.c \
        $(GSM)\mathdiv.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\math.obj $(GSM)\math.c

$(TMP)\signal.obj: $(GSM)\signal.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\signal.obj $(GSM)\signal.c

$(TMP)\stack.obj: $(GSM)\stack.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\stack.obj $(GSM)\stack.c

$(TMP)\vector.obj: $(GSM)\vector.c $(INCLUDE_dependency)
        $(CC) $(COMPILE_FLAGS) /Fo$(TMP)\vector.obj $(GSM)\vector.c
```

SUN, UNIX, CC

```
#
# gsm.suc  : gsm makefile for Sparc station 2 under unix with cc.
# command  : make -f gsm.suc
# Copyright (C) 1993 Guilhem de Wailly.
#

.KEEP_STATE:


#Directories
HOMEGSM = ..
TMP        = $(HOMEGSM)/tmp
BIN        = $(HOMEGSM)/bin
INCLUDE    = $(HOMEGSM)/include
LIB        = $(HOMEGSM)/lib
MAKE       = $(HOMEGSM)/make
GSM        = $(HOMEGSM)/gsm

#Compiler
COMPILE_FLAGS = -c -g -I$(INCLUDE) -D__DEBUG -D__DEMO -D__SPARC2_UNIX_CC
LINK_FLAGS    = -lm
CC            = cc
LINK          =  cc

#Dependency
PROJECT            = $(BIN)/gsm
INCLUDE_dependency = $(INCLUDE)/config.h $(INCLUDE)/gsm.h
OBJ_dependency     =$(TMP)\analysis.o $(TMP)\atom.o $(TMP)\conio.o $(TMP)\display.o \
                                      $(TMP)\env.o  $(TMP)\error.o $(TMP)\eval.o    \
                    $(TMP)\garbage.o  $(TMP)\hash.o $(TMP)\heap.o  $(TMP)\init.o    \
                    $(TMP)\keyword.o  $(TMP)\main.o $(TMP)\math.o  $(TMP)\signal.o  \
                    $(TMP)\stack.o    $(TMP)\vector.o
MAKE_dependency    = $(MAKE)/gsm.suc
LIB_dependency     =
EXE_dependency     = $(INCLUDE_dependency) $(OBJ_dependency) $(MAKE_dependency)
$(LIB_dependency)

$(PROJECT):$(EXE_dependency)
```

```
           $(LINK) -o $(PROJECT) $(OBJ_dependency) $(LIB_dependency) $(LINK_FLAGS)

$(TMP)/analysis.o: $(GSM)/analysis.c $(INCLUDE_dependency)
                   $(CC) $(COMPILE_FLAGS) -o $(TMP)/analysis.o $(GSM)/analysis.c

$(TMP)/atom.o: $(GSM)/atom.c $(INCLUDE_dependency)
               $(CC) $(COMPILE_FLAGS) -o $(TMP)/atom.o $(GSM)/atom.c

$(TMP)/display.o: $(GSM)/display.c $(INCLUDE_dependency)
                  $(CC) $(COMPILE_FLAGS) -o $(TMP)/display.o $(GSM)/display.c

$(TMP)/env.o: $(GSM)/env.c $(INCLUDE_dependency)
              $(CC) $(COMPILE_FLAGS) -o $(TMP)/env.o $(GSM)/env.c

$(TMP)/error.o: $(GSM)/error.c $(INCLUDE_dependency)
                $(CC) $(COMPILE_FLAGS) -o $(TMP)/error.o $(GSM)/error.c

$(TMP)/eval.o: $(GSM)/eval.c $(INCLUDE_dependency)
               $(CC) $(COMPILE_FLAGS) -o $(TMP)/eval.o $(GSM)/eval.c

$(TMP)/garbage.o: $(GSM)/garbage.c $(INCLUDE_dependency)
                  $(CC) $(COMPILE_FLAGS) -o $(TMP)/garbage.o $(GSM)/garbage.c

$(TMP)/hash.o: $(GSM)/hash.c $(INCLUDE_dependency)
               $(CC) $(COMPILE_FLAGS) -o $(TMP)/hash.o $(GSM)/hash.c

$(TMP)/heap.o: $(GSM)/heap.c $(INCLUDE_dependency)
               $(CC) $(COMPILE_FLAGS) -o $(TMP)/heap.o $(GSM)/heap.c

$(TMP)/init.o: $(GSM)/init.c $(INCLUDE_dependency)
               $(CC) $(COMPILE_FLAGS) -o $(TMP)/init.o $(GSM)/init.c

$(TMP)/keyword.o: $(GSM)/keyword.c $(GSM)/conv.c $(GSM)/string.c $(GSM)/lambda.c \
                  $(GSM)/is.c $(INCLUDE_dependency)
                  $(CC) $(COMPILE_FLAGS) -o $(TMP)/keyword.o $(GSM)/keyword.c

$(TMP)/main.o: $(GSM)/main.c $(INCLUDE_dependency)
               $(CC) $(COMPILE_FLAGS) -o $(TMP)/main.o $(GSM)/main.c

$(TMP)/math.o: $(GSM)/math.c $(GSM)/mathadd.c $(GSM)/mathsub.c $(GSM)/mathmul.c \
               $(GSM)/mathdiv.c $(INCLUDE_dependency)
               $(CC) $(COMPILE_FLAGS) -o $(TMP)/math.o $(GSM)/math.c

$(TMP)/signal.o: $(GSM)/signal.c $(INCLUDE_dependency)
                 $(CC) $(COMPILE_FLAGS) -o $(TMP)/signal.o $(GSM)/signal.c

$(TMP)/stack.o: $(GSM)/stack.c $(INCLUDE_dependency)
                $(CC) $(COMPILE_FLAGS) -o $(TMP)/stack.o $(GSM)/stack.c

$(TMP)/vector.o: $(GSM)/vector.c $(INCLUDE_dependency)
                 $(CC) $(COMPILE_FLAGS) -o $(TMP)/vector.o $(GSM)/vector.c
```

Documentation

```
/*
 G S C H E M E . D


 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/
```

```
+=========================================+
I                                         I
I  C O D E   I M P L E M E N T A T I O N  I
I                                         I
+=========================================+
```

```
C O N F I G U R A T I O N   D E F I N I T I O N S
-------------------------------------------------

  GSM was succesfully compiler and executed on the followed machine.
  Your are to define one of these macro to choice your configuration.

  Machine-system-compiler:
  ------------------------
    __PC_DOS_BC          : Ibm pc,      Dos 6,       Borland c 3.1
    __PC_DOS_QC          : Ibm pc,      Dos 6,       Quick c
    __PC_WINDOWS_BC      : Ibm pc,      Windows 3.1, Borland c 3.1
    __SPARC2_UNIX_CC     : Sun Sparc 2, Unix,        Sun c
    __MAC_MAC_TC         : Macintosh,   System 7,    Think c
    __VAX_VMS_CC         : Digital Vax, Vms,         Vax c
    __VAX_UNIX_CC        : Digital Vax, Unix,        Vax c


  Debug mode
  ----------
    __CHECK_C_ALLOCATOR  : checks all malloced pointer.
    __DEBUG              : activates the assert macro.
    __DEBUG_GARBAGE      : for each garnage collecting, display the heap
                           content.
    __DYNAMIC            : allows dynamic libraries loader.


  Floating point
  --------------
    None      : no floating point operation allowed
    __FLOAT   : defines the real type as float.
    __DOUBLE  : defines the real type as double.


  Sub configuration macros.
  -------------------------
    Normally, you don't have to check these macros. Use them only if you
    want to add a new configuration item.

    __DECLARE_PROTOTYPE : the function are defined with the ansi norm as
                          ret function (type name, type name, ...);
                          else they are declared as
                          ret function();
    far                 : for far pointer (under segmented machines as Ibm pc)
    near                : for near pointers (under segmented machines as Ibm pc)
    __LONG_AS_INT       : for machine in witch int have the same size than long
                          (sun sparc2, for exemple)
    __DECLARE_COMMON_DATA_TYPE : declares WORD, BYTES, ... Undefined with Windows
    IMPLEMENTATION_MACHINE : name of the implementation host machine
    IMPLEMENTATION_SYSTEM  : name of the implementation host operating system.
    MAX..., MIN...      : max and min data value (ex: MAXINT 0x7fff)
```

FUNCTION NAMES

The internal functions are prefixed by _.
The external functions are not prefixed.
All exported functions receive a MAIN pointer as first parameter and have GSM arguments.
The macro begin with an upper character.

GARBAGE COLLECTOR

The garbage heap is an array of free cells linked each to other.
The first two cells are used for the main environment.
In the main structure, there is an item called free witch points on the first free cell.
When a cons claims a garbage collect the function garbage_collect marks all cells with a 0. Then it runs across the defined environments from the main-environment and marks the encounterer cells with 1.
Then it run trought the heap array and mades a new free list with the unmarked cells

```
  main->garbage----->+---+---+
                     |xxx|xxx|<-------first two for the main
                     +---+---+    |   environment
                     |xxx|xxx|<---+
                     +---+---+
  main->free-------->|   | o-+---+
                     +---+---+   |<---link of free cell
                     |xxx|xxx|   |
                     +---+---+<--+
                     |   | o-+---+
                     +---+---+<--+
                     |   | o-+---+
                     +---+---+   |
                     |xxx|xxx|   |
                     +---+---+<--+
                     |   | 0 |<-------last free cell
                     +---+---+
                     |xxx|xxx|
                     +---+---+
                     |xxx|xxx|<-------used cell
                     +---+---+
                     |xxx|xxx|
                     +---+---+
```

STACK STRATAGIE

GSM provies a stack to protect temporary created cell against a garbage collecting. Any function use a cons, so a garbage collecting can occure during these functions calls. The startegie consists in push values created in your function, call these function and pop the stack. You can't make any hypothesis before call one primitive function concerning its use of cons, so push all your temporary cells. Notes that the cons function push itself its car and cdr argument.

DATA TYPE

This is the data representation in gsm.

type	G	I	C	V	_car			_cdr	here, shown as
					type	lenght	car		
CHAR	x	1	0	0	CHAR	x	x	c	CHR
CODE	x	0	1	0	CODETYPE	x	x	f	COD
COMPLEX	x	0	0	1	COMPLEX	x	x	v	CPX
FLAG	1	1	0	0	FLAG	x	x	i	FLA
FREE	x	1	0	0	FREE	x	x	i	FRE
IDENTIF.	x	1	0	0	IDENTIF.	x	x	s	IDT
INDIRECT	x	1	0	0	INDIRECT	x	x	cdr	IND
INTEGER	x	1	0	0	INTEGER	x	x	i	INT
LAMBDA	x	0	1	0	LAMBDA	x	x	v	LBD
LONGINT	x	1	0	0	LONGINT	x	x	l	LNT
POINTER	x	1	0	0	POINTER	x	x	v	PTR
REAL	x	1	0	0	REAL	x	x	r	REA
STRING	x	1	0	0	STRING	x	x	s	STR
CELL	x	0	0	0	x	x	car	cdr	o o
USER	x	1	0	0	STRUCT	x	x	u	USR
VECTOR	x	1	0	1	x	length	x	v	VEC

CODE: A code cell has it bit cell.cod set to 1. The cell.type indicates the number of formal arguments and in addition the type of the procedure (see gsm.h and the CT_xxx macros.) Note that if the dynamic libraries are enable, a cell code may be a dynamic cell.

FLAG: There are attoms with a particular value. The allowed flags are in exemple FALSE, TRUE, UNSPECIFIED. (See gsm.h and the FLAG definition).

IDENTIFIER: Identifiers are used during the lexical, syntaxic and sementic analysises. They are identical to the STRINGs.

INDIRECT: The indirect cells are used to make indirection in the lambda expressions ans their formals arguments. The cdr of these cells is the real value of the cell. They are immediat cell.

LAMBDA: represents the lambda expressions. Has the bit code set and the type field equal to LAMBDA. The cdr of a such cell is as a vector (but the field len is set to LAMBDA). This vector has its last field sets to NULL. The last value field is the code list. All previous fields are formals arguments of the lambda-expression.

POINTER: the pointer type is used to allow particular implementation of c strutures. The structure must be allocated by the standard gsm non-garbaged allocator function. When a POINTER typed cell is not used (garbaged), gsm frees it with the standard gsm free() function. So that indictate that the pointer don't contain other allocated objects.

STRING: A string cell has its cdr witch points on a c allocated string. All string are mallocated with two complementary bytes sets to 0;

USER: The USER cells are used to make complexes c object. The first fields of the structure has to match with the USR struct defined in gsm.h. A USR structure has to have defined tree functions : free(), equal() and print(). (These header is the fields of the USR structure (see gsm.h)).

VECTOR: A vector is a joined set of cells. The car of a vector cell contains the length of the vector. The cell.vector field is set. The cdr

points on a malloced arrray of cells. In internal processes
any vectors have a particular strucure as SYMBOL, ENVironment...

P A I R

Pairs are the most basic gsm data structure.

```
      +---+---+   +---+---+
      | o | o-+-->|   |   |<----atom.
      +-+-+---+   +---+---+
        |
        v
    +---+---+
    |   |   |<--any cell structure.
    +---+---+
```

L I S T

A list is the main data structure in scheme. It consists on a set of linked cell, as follow :

```
    list                                                   NULLOBJ flag
    +---+---+  +---+---+  +---+---+  +---+---+  +---+---+  +---+---+
    | o | o-+->| o | o-+->| o | o-+->| o | o-+->| o | o-+->|NUL|   |
    +-|-+---+  +-+-+---+  +-+-+---+  +-+-+---+  +-|-+---+  +---+---+
      |          |          |          |          |
      v          v          v          v          v
  +---+---+  +---+---+  +---+---+  +---+---+  +---+---+
  | o | o |  | o | o |  | o | o |  | o | o |  | o | o |
  +---+---+  +---+---+  +---+---+  +---+---+  +---+---+
  car        cdr            ^
                            |
                            +-----contain the item value
```

V E C T O R

A vector begins by a vector typed cell. the v cell._cdr pointer points on an malloced array of CELL pointers. Each of these poointers are initialised to 0. Then they can points on a garbaged CELL.

```
  ..........................
  .                        .
  . vector      CELL[]     .  CELL
  . +---+---+   +---+      .  +---+---+
  . |VEC| o-+-->| o-+------.->|   |   |
  . +---+---+   +---+      .  +---+---+
  .             | o-+--\   .  +---+---+
  .             +---+   ---.->|   |   |
  .             | o-+--\   .  +---+---+
  .             +---+   \  .  +---+---+
  .             | o-+--\ \-.->|   |   |
  .           +---+   \  .  +---+---+
  .                    \ .  +---+---+
  .                     \.->|   |   |
  .........................  +---+---+
```

COMPLEX

The complex numbers are represented in gsm as 3 dimensioned vertors. The real part is in the first cell, the imaginary one, in the second cell and a control cell in the third cell.
Note that all system vector (witch hs to be reconize by the system as particular value) has a control cell defined staticly in the concerned .c file (CELL complex_control; is defined in math.c).

```
     complex    as a
     node       vector       cell
     +---+---+  +---------+   +---+---+
     |VCT| o-+->| real  o-+-->|   |   |
     +---+---+  +---------+   +---+---+
                | imag  o-+-->|   |   |
                +---------+   +---+---+
                | ctrl  o-+-->|   |   |
                +---------+   +---+---+
```

SYMBOL

A symbol is a name with a value. It is a vector.

```
     symbol     as a
     node       vector       cell
     +---+---+  +---------+   +---+---+   +---------------+
     |VCT| o-+->| name  o-+-->|STR| o-+-->|               |
     +---+---+  +---------+   +---+---+   +---------------+
                | value o-+-->|   |   |
                +---------+   +---+---+
                | ctrl    |
                +---------+
```

```
Note : the flag value in our implementation are staticly allocated. but the
----   symbol value may be changed by set! or define. So when a symbol is
       created with a flag value, the value is copied in a garbage allocated
       cell.
```

SYMBOL HASH TABLE

An symbol hash table is a list of symbol join to a cell vector table witch is the hash table.

The values of the vector table are the result of a hash function applied on the symbol names. The result of this function is the index (integer) in the hash vector. A list of same indexed symbol is maintained.

```
                             list of same hash
hash table  vector           value symbol.
+---+---+   +-------+        +---+---+  +---+---+
|VCT| o-+-->|   o---+------->| o | o-+->|NUL|   |
+---+---+   +-------+        +-+-+---+  +---+---+
            |  NUL  |      |
            +-------+      S<-----------S represents one symbol.
            |  NUL  |
            +-------+    +---+---+  +---+---+  +---+---+
            |   o---+--->| o | o-+->| o | o-+->|NUL|   |
            +-------+    +-+-+---+  +-+-+---+  +---+---+
            |  NUL  |      |          |          ^
            +-------+      S          S          |
            |  NUL  |                            +--- NULLOBJ flag.
            +-------+
            |  NUL  |
            +-------+
            |ctrl o-+--->points on a control cell.
            +-------+
```

ENVIRONMENT

An environment is a tree dimensionned vector where the first field is the parent environment (may be a NULLOBJ flag for the top level environment), and the third is a control cell (see env.c).

```
+---+---+   +-----------+
|VCT| o-+-->| parent  o-+-->points on the parent environment.
+---+---+   +-----------+
            | hash    o-+-->points on a hash table.
            +-----------+
            | control o-+-->points on an unic control cell.
            +-----------+
```

FUNCTION

A gsm function is a set of : function name, number of argument, type of argument, value of argument.

```
+---+---+   +--------+
|COD| o-+-->| c-code |
+---+---+   +--------+
```

Type of function argument :

We distinguishe four type of procedure : standards c calling procedures, compiled reserved keyword, reserved keyword and lambda expression. These types are exclusives.
They are dicerned by the different value of the field type of a cell :

CT_PROCEDURE : standard procedure. All arguments are evaluated before the function calling. They are evaluated during the evaluation of a tree.

CT_RESERVED : reserved keyword. They are procedures, but the system can indicates that they are reserved.

CT_NOEVAL : reserved keyword. Non arguments are evaluated. The function is not compiled.

CT_LAMBDA : defines the lambda expression. All arguments are evaluated. In addition, the list argument as the optional one are forbiden.

CT_COMPILE : these procedures are evaluated during the analysis of an expression. The result is a compressed form of this expression. The arguments are not evaluated.

CT_APPLY : these procedures don't return a value but an application of a function (generaly lambda) applyed to any arguments. In the top level environment, the evaluator evals this application and return its result. In an other environment (created by a lambda expression), the evaluator simply return the application because the application has to be compiled in the environment.

The following types describes the type of the last argument. It can be a list, or optional :

CT_LIST : the last argument is a list.

CT_OPTIONAL : the last argument is optional. This optional argument can be list typed (CT_LIST).

To forms a procedure type, you can add these types :

CT_PROCEDURE + CT_LIST + 5 : defines a procedure with 5 normals arguments and with a list for the last one.

CT_RESERVED + CT_LIST + CT_OPTIONAL : defines a pocedure with one optional argument. In addition this argument is compulsery a list.

With the CT_LAMBDA type, the CT_LIST and the CT_OPTIONAL types are not allowed.

LAMBDA EXPRESSION

The lambda expression describe a dynamic way to create gsm procedures. The syntaxe is (lambda (<formals>) (<body>)), where formal are a list of formal arguments and body, a list of gsm instructions.

```
      (lambda (a b c d) <body>) or (lambda a <body>)
```

```
lambda cell vector
+---+---+   +--------+
|LBD| o-+-->| code o-+--> <body>
+---+---+   +--------+   +---+---+
            | a1   o-+-->|UNB|   |<-- formal init value (inits to UNBOUNDED)
            +--------+   +---+---+
            | a2   o-+-->|UNB|   |
            +--------+   +---+---+
               ...          ...
            +--------+   +---+---+
            | a3   o-+-->|NUL|   |
            +--------+   +---+---+
```

A lambda expression is represented by a code cell with the CT_LAMBDA code type.
The cdr points on a vector that contain, first the <body> followed by an array of cell witch are the formal arguments value.
A lambda evaluation returns the value returned by the last procedure call

of the list <body>.

LET EXPRESSION

The let expressions perform a transformation of the analysis tree. They return an other tree witch is a lambda expression.
The way to analysis these expressions is not identical in the toplevel environment and in a lambda expression (child environment).
A let expression return an application of lambda expression on the value of the identifier :
(let ((a 3)) <body>) is identical to ((lambda a) <body>) 3).
When the let operator acts, it returns an application.
In the toplevel, the evaluator has to return the value of this application.
In opposition, when gsm analysises a lambda expression, it compiles it.
Here the let expression has not to be evaluated but it has to be compiled.
In exemple :
(let ((a 3)) a), in the top level is given to the evaluator as

```
                              eval              eval
         *=*--*=*--*=*--nul  ----->  *=*--nul  ----->  value
         |    |    |                 |
        let   |    a               lambda
              |
             *=*--nul
             |
            *=*--*=*--nul
            |    |
            a    3
```

The followed expression :

```
(lambda () (let ((a 3)) a)), will be reduced in :
           ---------------
                <body>
```

```
                       compile(2)            <body>    compile(1)
   *=*--*=*--*=*--nul -----------> *=*--*=*--*=*--nul ----------->  *=*--nul
   |    |    |(2)                  |    |    |                      |
lambda nul  *=*--*=*--*=*--nul  lambda nul  *=*--nul              lambda
 (1)        |    |    |           (1)       |                       (1)
           let   |    a                   lambda
                 |                          (2)
                *=*--nul
                |
               *=*--*=*--nul
               |    |
               a    3
```

in the two cases the analysis process is realy differents. This is possible because of the CT_APPLY code type. The evaluator tests if the current environment is the toplevel. Then it returns the evaluation a CT_APPLY procedure result. Else if the current environment is not the toplevel one, evaluator return only the result.
Of the point of view of the garbaging, we see that a let is a lambda expression with no argument : the formal argument are hiden. If the arguemt are used in the let expression, they will be garbaged. If they are not used, they will not.

DYNAMIC LIBRARIES

GSM provides a usefull dynamic libraries loader for DOS and Windows. The I deal hardly with the Unix implementation.
A dynamic library is an executable witch is dynamicly linked with GSM.

This features allowed GSM to be extended without any modification of the kernel.
When the macro __DYNAMIC is defined, the load-dynamic, register-dynamic and unload-dynamic are enabled.
The function load-dynamic awaits for the library name as first parameter.
The return value is the library handle. If the handle is less than zero, GSM can not load correctly the library. See loadlib.h to gets the error values.
unload-dynamic awaits for a library handle as first parameter.
register-dynamic allows the user to registers the library function in the GSM tables. The first parameter is the library handle, the second one is the name of the function to be registered, and the third one is the description of the function return value, and the function await parameters. This description is a string build as follow :
the parameter value types is made by a a string where each character describes a c type. The first character is the return type of the function.
The next ones are the function awaited parameters types. The function can have a maximum of seven parameters. The c types are defined as follow :

c : char
n : integer.
l : long
r : real (the function can wait only a double (no float & long double)
s : string
v : void

By exemple, a c function who await a string and an integer and who return a long may be describes as : "lsi".

```
+===================================+
I                                   I
I G S C H E M E   P R O C E D U R E S I
I                                   I
+===================================+
```

RESERVED
----
append :
begin :
boolean? :
car :
cdr :
char? :
complex? :
cond :
cons :
define :
display :
eq? :
eqv? :
equal? :
exact? :
exit :
extended-syntaxe :
file-exists? :
garbage-collect :
garbage-size :
if :
inexact? :
lambda :
length :
let :
letrec :
list :
list? :
load :

```
null? :
number? :
pair? :
procedure? :
prompt :
quote :
redefine-symbol :
restart :
reverse :
set! :
set-car! :
set-cdr! :
string? :
symbol? :
system-call :
top-level :
vector? :
verbose :
version :




CONIO
-----

clear-end-of-line :
clear-screen :
delete-line :
get-text :
get-text-info :
goto-xy :
high-video :
insert-line :
low-video :
move-text :
normal-video :
put-text :
set-cursor-type :
text-attribut :
text-background :
text-color :
text-mode :
where-x :
where-y :
window :
```

#INCLUDE

CONFIG.H

```
/*
 C O N F I G . H

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#ifndef __CONFIG_H


/* The configuration */
/*********************/

/*#define __DEBUG_GARBAGE*/
#define __DEBUG_HEAP
#define __FLOAT
/*#define __DOUBLE*/
/*#define __LDOUBLE*/
#define __DYNAMIC
/*#define __PC_DOS_REALMODE */
/*#define __PC_DOS_PROTECTEDMODE */


/****************************/
/* end of the configuration */



#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <setjmp.h>



#ifdef __PC_DOS_BC
# undef  __PC_DOS_BC
# define __CONFIG_H
# define __Msdos
# define __Borlandc
# define __Ibmpc
# define __DECLARE_PROTOTYPE
# undef  __LONG_AS_INT
# define __DECLARE_COMMON_DATA_TYPE
# define __ALLIGN_TYPE int
# undef  __C_HEAP
# define huge __huge
#endif

#ifdef __PC_DOS_QC
# undef  __PC_DOS_QC
# define __CONFIG_H
# define __Msdos
# define __Quickc
# define __Ibmpc
# define __DECLARE_PROTOTYPE
# undef  __LONG_AS_INT
# define __DECLARE_COMMON_DATA_TYPE
# define __ALLIGN_TYPE int
# undef  __C_HEAP
# define huge _huge
#endif

#ifdef __PC_WINDOWS_BC
# undef  __PC_WINDOWS_BC
# define __CONFIG_H
# include <windows.h>
# define __Windows
# define __Borlandc
# define __Ibmpc
# define __DECLARE_PROTOTYPE
# undef  __LONG_AS_INT
# undef  __DECLARE_COMMON_DATA_TYPE
# define __ALLIGN_TYPE int
# undef  __C_HEAP
# define huge __huge
#endif

#ifdef __SPARC2_UNIX_CC
# undef  __SPARC_UNIX_CC
# define __CONFIG_H
# define __Unix
# define __Krc
# define __Sparc2
# undef  __DECLARE_PROTOTYPE
# define __LONG_AS_INT
# define __DECLARE_COMMON_DATA_TYPE
# define __ALLIGN_TYPE long
# define huge
#endif

#ifdef __MAC_MAC_TC
# undef  __MAC_MAC_TC
# define __CONFIG_H
# define __System7
# define __Thinckc
# define __Mac
# undef  __DECLARE_PROTOTYPE
# undef  __LONG_AS_INT
# define __DECLARE_COMMON_DATA_TYPE
# define __ALLIGN_TYPE char
# define __C_HEAP
# define huge
#endif

#ifdef __VAX_VMS_CC
# undef  __VAX_VMS_CC
# define __CONFIG_H
# define __Vms
# define __Krc
# define __Vax
# undef  __DECLARE_PROTOTYPE
# define __LONG_AS_INT
# define __DECLARE_COMMON_DATA_TYPE
# define __ALLIGN_TYPE long
# define __C_HEAP
# define huge
#endif

#ifdef __VAX_UNIX_CC
# undef  __VAX_UNIX_CC
# define __CONFIG_H
# define __Unix
# define __Krc
# define __Vax
# undef  __DECLARE_PROTOTYPE
# define __LONG_AS_INT
# define __DECLARE_COMMON_DATA_TYPE
# define __ALLIGN_TYPE long
# define __C_HEAP
# define huge
#endif



/* configuration check */
#ifndef __CONFIG_H
# define __SPARC2_UNIX_CC
# include "config.h" /* default mode */
#endif
```

```
/* machine & system name */
#ifdef __Ibmpc
# define IMPLEMENTATION_MACHINE "Ibm-pc"
#endif
#ifdef __Sparc2
# define IMPLEMENTATION_MACHINE "Sparc-2"
#endif
#ifdef __Mac
# define IMPLEMENTATION_MACHINE "Macintosh"
#endif
#ifdef __Vax
# define IMPLEMENTATION_MACHINE "Vax"
#endif
#ifdef __Msdos
# define IMPLEMENTATION_SYSTEM  "Ms-dos"
#endif
#ifdef __Windows
# define IMPLEMENTATION_SYSTEM  "Windows"
#endif
#ifdef __Unix
# define IMPLEMENTATION_SYSTEM  "Unix"
#endif
#ifdef __System7
# define IMPLEMENTATION_SYSTEM  "System-7"
#endif
#ifdef __Vms
# define IMPLEMENTATION_SYSTEM  "Vms"
#endif

/* dynamics library */
#ifdef __Unix
# ifdef __DYNAMIC
#  include <error: dynamics libraries not allowed with Unix>
# endif
#endif
#if defined(__DYNAMIC) && !defined(__INCLUDE_API)
# define __INCLUDE_API "server.h"
#endif

/* real / protected mode for dos */
#ifndef __Msdos
# if defined (__PC_DOS_REALMODE) || defined (__PC_DOS_PROTECTEDMODE)
#   include <Error: __PC_DOS_REALMODE/__PC_DOS_PROTECTEDMODE modes are only allowed
with DOS OS>
# endif
#endif
#ifdef __PC_DOS_REALMODE
# undef __PC_DOS_PROTECTEDMODE
#endif
#if !defined(__PC_DOS_REALMODE) && !defined(__PC_DOS_PROTECTEDMODE)
# define __PC_DOS_REALMODE
#endif

/* copy jmp_buf macro (about setjmp()) */
#ifdef __Sparc2
# define jmp_cpy(d,s) memcpy((d),(s),sizeof(jmp_buf))
#else
# define jmp_cpy(d,s) *d=*s
#endif

/* machine & system name */
#ifdef __Ibmpc
# define IMPLEMENTATION_MACHINE "Ibm-pc"
#endif
#ifdef __Sparc2
# define IMPLEMENTATION_MACHINE "Sparc-2"
#endif
#ifdef __Mac
# define IMPLEMENTATION_MACHINE "Macintosh"
#endif
#ifdef __Vax
# define IMPLEMENTATION_MACHINE "Vax"
```

```
#endif
#ifdef __Msdos
# define IMPLEMENTATION_SYSTEM  "Ms-dos"
#endif
#ifdef __Windows
# define IMPLEMENTATION_SYSTEM  "Windows"
#endif
#ifdef __Unix
# define IMPLEMENTATION_SYSTEM  "Unix"
#endif
#ifdef __System7
# define IMPLEMENTATION_SYSTEM  "System-7"
#endif
#ifdef __Vms
# define IMPLEMENTATION_SYSTEM  "Vms"
#endif

/* program version */
#if defined(__Krc)
# define __VERSION "alpha 1.00-June 1993"
#else
# define __VERSION "alpha 1.00-" __DATE__
#endif

/* debug mode */
#ifndef __DEBUG
# undef __DEBUG_GARBAGE
# undef __CHECK_HEAP
#endif

/* real type */
#ifdef __FLOAT
# undef __DOUBLE
# undef __LDOUBLE
# define __REAL
#else
#  ifdef __DOUBLE
#    undef __FLOAT
#    undef __LDOUBLE
#    define __REAL
#  else
#    ifdef __LDOUBLE
#      undef __FLOAT
#      undef __DOUBLE
#      define __REAL
#    else
#      undef __REAL
#    endif
#  endif
#endif
#ifdef __REAL
# include <math.h>
#endif

/* long */
#ifdef __LONG_AS_INT
# undef __LONG
#else
# define __LONG
#endif

#endif /* __CONFIG_H */
```

GSM.H

```
/*
 G S M . H

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.
```

```
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#ifndef __GSM_H
#define __GSM_H
#include <config.h>


/* max and min common type value */
#ifdef __Ibmpc
# define MAXINT           (int) 0x7fff
# define MININT           (int) 0x8000
# define MAXLONG          (long)0x7fffffff
# define MINLONG          (long)0x80000000
# define MAXFLOAT         3.37E+38
# define MINFLOAT         8.43E-37
# define MAXDOUBLE        1.797693E+308
# define MINDOUBLE        2.225074E-308
# define MAXEXP           308
# define MINEXP           (-308)
# define MAXLONGDOUBLE    0
# define MINLONGDOUBLE    0
#else
# ifdef __Sparc2
#  define MAXINT           (int) 0x7fffffff
#  define MAXLONG          (long)0x7fffffff
#  define MININT           (int) 0x80000000
#  define MINLONG          (long)0x80000000
#  define MAXFLOAT         3.37E+38
#  define MINFLOAT         8.43E-37
#  define MAXDOUBLE        1.797693E+308
#  define MINDOUBLE        2.225074E-308
# define MAXEXP           308
# define MINEXP           (-308)
#  define MAXLONGDOUBLE    0
#  define MINLONGDOUBLE    0
# else
#  define MAXINT
#  define MAXLONG
#  define MININT
#  define MINLONG
#  define MAXFLOAT
#  define MINFLOAT
#  define MAXDOUBLE
#  define MINDOUBLE
# define MAXEXP
# define MINEXP
#  define MAXLONGDOUBLE
#  define MINLONGDOUBLE
# endif
#endif

/* common data type */
#ifdef __DECLARE_COMMON_DATA_TYPE
typedef unsigned char          BYTE;
typedef unsigned               WORD;
typedef unsigned long          DWORD;
typedef char          huge*     PSTR;
# undef __DECLARE_COMMON_DATA_TYPE
#endif /* __DECLARE_COMMON_DATA_TYPE */
typedef int                     LEXEME;
typedef struct s_CELL huge*     GSM;
typedef struct s_CELL huge*huge*VECTOR;

/* real type */
#ifdef __FLOAT
# define MAXREAL MAXFLOAT
# define MINREAL MINFLOAT
  typedef float real;
#endif
#ifdef __DOUBLE
# define MAXREAL MAXDOUBLE
# define MINREAL MINDOUBLE
  typedef double real;
#endif
#ifdef __LDOUBLE
# define MAXREAL MAXLONGDOUBLE
# define MINREAL MINLONGDOUBLE
  typedef long double real;
#endif

/* c-kernigam or c-ansi function declaration style */
#ifdef __DECLARE_PROTOTYPE
# define PROTO(p) p
#else
# define PROTO(p) ()
#endif

/* assertion macro */
#ifdef __DEBUG
# ifdef __Borlandc
#  define _assert(_main,_condition,_execute)\
     if(!(_condition)) {__assert((_main), __FILE__, __LINE__, #_condition);\
                        _execute;}
# else
#  define _assert(_main,_condition,_execute)\
     if(!(_condition)) {__assert((_main), __FILE__, __LINE__, /**/_condition);\
                        _execute;}
# endif
# define _assert_false(_main, _message, _execute)\
    {__assert((_main), __FILE__, __LINE__, (_message));_execute;}
#else
# define _assert(m,c,e)       ((void)0)
# define _assert_false(m,e,x) x
#endif

/* End of string, end of file and end of line */
#define EOS        0     /* end of string     */
#define EOF        (-1)  /* end of file       */
#define EOL        '\n'  /* end of line       */



/* command line option character (in lower case) */
#define CLO_HELP                   'h'
#define CLO_GARBAGE_SIZE           'g'
#define CLO_HEAP_SIZE              'x'
#define CLO_SYMBOL_TABLE_SIZE      's'
#define CLO_PROMPT                 'p'
#define CLO_TEMP_SYMBOL_TABLE_SIZE 't'
#define CLO_DYNAMIC_FILE           'l'



/* error type */
#define OK      0  /* no error                    */
#define WARNING 1  /* warning                     */
#define ERR     2  /* error                       */
#define GOTOP   3  /* go to top level             */
```

```
#define FATAL   4  /* system corupted - end_gsm */
#define EXIT    5  /* system corupted - exit    */

/* error value */
#define ERR_STRING_TOO_LONG            500
#define ERR_UNTERMINATED_STRING        501
#define ERR_IDENTIFIER_TOO_LONG        502
#define ERR_REDEFINED_SYMBOL           503
#define ERR_NOT_ENOUGHT_MEMORY         504
#define ERR_STACK_OVERFLOW             505
#define ERR_UNABLE_TO_OPEN_FILE        506
#define ERR_BAD_VECTOR_INDEX           507
#define ERR_EXTENDED_SYNTAXE           508
#define ERR_BAD_FORMAL                 509
#define ERR_OPEN_FILE                  510
#define ERR_BAD_OPERAND                511
#define ERR_UNDEFINED_SYMBOL           512
#define ERR_DIVISION_BY_ZERO           513
#define ERR_UNEXPECTED_EOF             514
#define ERR_INVALID_OPTION             515
#define ERR_FLOATING_POINT             516
#define ERR_CONTROL_BREAK_PRESSED      517
#define ERR_HEAP_CORRUPTED             518
#define ERR_GARBAGE_CORRUPTED          519


/* MAIN STRUCTURE main structure - GUSER(GSM main) */
typedef struct s_MAIN {

  /* IO file */
  FILE  * in;                   /* file in input. May replaced by a PORT */
  FILE  * out;                  /* file in output. May replaced by a PORT */
  FILE  * err;                  /* error file output. May replaced by a PORT */
  PSTR    file;                 /* input file name */

  /* prompt */
# define PROMPT_LENGTH 50
  char    prompt[PROMPT_LENGTH];/* TOPLEVEL prompt */
# define DEFAULT_PROMPT "GSM->"

  /* curent line */
  int     line;                 /* curent line number */

  /* heap */
  DWORD   heap_size;            /* heap size - dynamicly in/decreased */
# define DEFAULT_HEAP_SIZE 50000/* default heap size value */
  DWORD   heap_free;            /* information field */
  void   *heap_base;            /* heap base pointer */
  void   *heap_last;            /* last allocazted bloc */

  /* garbage */
  DWORD   garbage_size;         /* garbage heap size */
# define DEFAULT_GARBAGE_SIZE 5000 /* initial value */
  GSM     garbage;              /* first cell of the garbage collector */
  GSM     free;                 /* first free cell of the garbage */

  /* gsm hard environemt */
  jmp_buf goto_toplevel;        /* standard setjmp() environment */
  jmp_buf goto_restart;         /* " */

  /* stack */
  WORD    stack_size;           /* stack size in GSM */
# define DEFAULT_STACK_SIZE 1000/* initial value */
  VECTOR  stack;                /* stack base */
  WORD    head;                 /* stack top */

  /* runtime environment */
  GSM     toplevel;             /* top level environmemt */
  GSM     current_environment;  /* current working environment */

  /* hash table */
  WORD    hash_size;            /* size of the TOPLEVEL symbol hash table */
# define DEFAULT_HASH_SIZE 100  /* default toplevel size */
  WORD    hash_temp_size;       /* the !TOPLEVEL hash table */
# define DEFAULT_HASH_TEMP_SIZE 50 /* default temporary hash table size */

  /* lexical & analysis */
# define DEFAULT_BUFFER_SIZE 500/* length of working buffers */
  int     identifier_length;    /* length of identifiers */
# define DEFAULT_IDENTIFIER_LENGTH 20
  GSM     value;                /* curent analysed value - See analysis() */
  LEXEME  lexeme;               /* curent lexeme - See analysis() */
  int     level;                /* count of matching paranthesises */

  /* error count */
  WORD    error;                /* error counter */
  WORD    warning;              /* warning counter */
  WORD    errno;                /* last error number - Not implemented */

  /* option */
  struct s_OPTION {
    /* W a r n i n g s   &   e r r o r s */
    WORD  redefine_symbol : 3;  /* warning : symbol xxx is already defined */
#   define DEFAULT_REDEFINE_SYMBOL  WARNING

    WORD  extended_syntaxe: 3;  /* error level if not extended syntaxe allowed */
#   define DEFAULT_EXTENDED_SYNTAXE OK

    /* O p t i o n s */
    WORD  verbose_eval    : 1;  /* verbose evaluation mode */
#   define DEFAULT_VERBOSE_EVAL 0

    WORD  display_reserved:1;   /* display reserved key-word of toplevel */
#   define DEFAULT_DISPLAY_RESERVED 0

  } option;
} MAIN;
typedef struct s_OPTION OPTION;




/* code structure */
typedef GSM (* FUNC) PROTO ((MAIN*_main, ...));

/* type of procedure */
/* is
<CT_PROCEDURE|CT_RESERVED|CT_NOEVAL|CT_LAMBDA|CT_COMPILE>+<0..0xFF>+[CT_LIST&/|CT_OPT
IONAL] */
typedef WORD CODETYPE;
#define _CT_MASKNARG 0x000F   /* logical mask for number of argument */
#define _CT_MASKTYPE 0xFFF0   /* logical mask for type of procedure */
#define CT_PROCEDURE 0x0010   /* standard procedures - all arguments evaluated */
#define CT_RESERVED  0x0020   /* reserved procedure - all arguments evaluated */
#define CT_NOEVAL    0x0040   /* reserved procedure - non evaluation */
#define CT_LAMBDA    0x0080   /* lambda definition - all arguments evaluated */
#define CT_COMPILE   0x0100   /* compiled in a lambda expression */
#define CT_APPLY     0x0200   /* return a transformed tree - arguments are not
evluated */
#define CT_LIST      0x0400   /* last argument is a list */
#define CT_OPTIONAL  0x0800   /* last argument is optionnal */
#ifdef __DYNAMIC_LIBRARIES
# define CT_LIBRARY   0x8000  /* dynamic loaded function */
#endif
#define CT_0         0x0000   /* zero argument */
#define CT_1         0x0001
#define CT_2         0x0002
#define CT_3         0x0003
#define CT_4         0x0004
#define CT_5         0x0005
#define CT_6         0x0006
#define CT_7         0x0007
#define CT_8         0x0008
#define CT_9         0x0009
```

```
#define CT_10        0x000A
#define CT_11        0x000B
#define CT_12        0x000C
#define CT_13        0x000D
#define CT_14        0x000E
#define CT_15        0x000F




/* gsm declaration structure */
typedef struct s_DECLF {
  PSTR      name; /* not temporary string pointer          */
  CODETYPE  arg;  /* number and type of waiting arguments  */
  FUNC      f;    /* pointer on c code function            */
} DECLF;




/* User structure STRUCT */
typedef struct s_USR {
# define USR struct s_USR
  void    *control;
  void   (*free)    PROTO ((MAIN*, USR*));
  int    (*equal)   PROTO ((MAIN*, USR*,  USR*));
  void   (*display) PROTO ((MAIN*, USR*));
  void    *the_usr;
# undef USR
} USR;





/* the cell structure */
typedef struct s_CELL {
  union {
    GSM  car;             /* generic car */
    WORD len;             /* vector length */
    WORD type;            /* data type */
  } _car;                 /* car union */
  union {
    char             c;   /* cdr.char */
    GSM              cdr; /* generic cdr */
    FUNC             f;   /* cdr.c_function */
    int              i;   /* cdr.int */
    long             l;   /* cdr.long */
    void           * p;   /* cdr.pointer */
#   ifdef __REAL
    real           * r;   /* cdr.real */
#   endif
    PSTR             s;   /* cdr.string */
    VECTOR           v;   /* cdr.vector */
    struct s_USR   * u;
  } _cdr;                 /* cdr union */
  WORD garbage  : 1;      /* garbage collector mark */
  WORD immediat : 1;      /* immediat flag */
  WORD code     : 1;      /* code flag */
  WORD vector   : 1;      /* vector flag */
} CELL;

#define NEWCELL(_main) cons((_main),\
                            _make_atom((_main), FLAG, F_NULLOBJ),\
                            _make_atom((_main), FLAG, F_NULLOBJ))

/* macro for an easy access of the CELL items */
#define GBG(x)        ((x)->garbage)   /* give the garbage bit of a cell */
#define IMM(x)        ((x)->immediat)  /*          immediat              */
#define COD(x)        ((x)->code)      /*          code                  */
#define VCT(x)        ((x)->vector)    /*          vector                */
#define TYP(x)        ((x)->_car.type) /*          type                  */
```

```
#define LEN(x)         ((x)->_car.len)  /*          length                 */
#define _CAR(x)        ((x)->_car)      /*          _car union             */
#define _CDR(x)        ((x)->_cdr)      /*          _cdr union             */
#define CAR(x)         (_CAR(x).car)    /*          generic car            */
#define CDR(x)         (_CDR(x).cdr)    /*          genaric cdr            */
#define CADR(x)        (CAR(CDR(x)))
#define CDDR(x)        (CDR(CDR(x)))
#define CADDR(x)       (CAR(CDDR(x)))
#define CDDDR(x)       (CDR(CDDR(x)))
#define CADDDR(x)      (CAR(CDDDR(x)))
#define CDDDDR(x)      (CDR(CDDDR(x)))
#define CADDDDR(x)     (CAR(CDDDDR(x)))
#define CDDDDDR(x)     (CDR(CDDDDR(x)))
#define CADDDDDR(x)    (CAR(CDDDDDR(x)))
#define CDDDDDDR(x)    (CDR(CDDDDDR(x)))
#define CADDDDDDR(x)   (CAR(CDDDDDDR(x)))
#define CDDDDDDDR(x)   (CDR(CDDDDDDR(x)))
#define CAAR(x)        CAR(CAR(x))
#define CAAAR(x)       CAR(CAAR(x))
#define CAAAAR(c)      CAR(CAAAR(x))



/* type (all immediat and not code) */
#define FIRSTTYPE    256              /* first type - has not to be ascii */
#define FLAG          (FIRSTTYPE +0)   /* atome flag */
#  define F_FALSE          100        /* false flag */
#  define F_OVERFLOW       200        /* overflow flag */
#  define F_NOTIMPLEMENTED 300        /* for not implemented function */
#  define F_NULLOBJ        400        /* null object flag */
#  define F_TRUE           500        /* true flag */
#  define F_UNBOUNDED      600        /* unbounded flag */
#  define F_UNDEFINED      700        /* undefined flag */
#  define F_UNEXPECTED     800        /* unexpected flag */
#  define F_UNSPECIFIED    900        /* unspecified flag */
#define CHAR          (FIRSTTYPE +1)   /* atom char */
#define INTEGER       (FIRSTTYPE +2)   /* atom int */
#ifdef __LONG
# define LONGINT      (FIRSTTYPE +3)   /* atom longint */
#endif
#ifdef __REAL
#  define REAL        (FIRSTTYPE +4)   /* atom real */
#endif
#define COMPLEX       (FIRSTTYPE +5)   /* atom complex */
#define POINTER       (FIRSTTYPE +6)   /* temporary type, will be replaced by STRUCT
or VECTOR*/
#define STRING        (FIRSTTYPE +7)   /* atom string */
#define USER          (FIRSTTYPE +8)   /* user c structure */
#define LAMBDA        (FIRSTTYPE +9)   /* lambda expression */
#define FREE          (FIRSTTYPE +10)  /* free cell */
#define INDIRECT      (FIRSTTYPE +11)  /* indirection cell */

/* lexeme value */
#define BACKQUOTE     (FIRSTTYPE +12)  /* the lexical() return value is a ...*/
#define QUOTE         (FIRSTTYPE +13)  /* " */
#define IDENTIFIER    (FIRSTTYPE +14)  /* " */

/* followed types are virtual - Used by wta in error.c */
#define T_BOOL        (FIRSTTYPE +15)  /* boolean */
#define T_CELL        (FIRSTTYPE +16)  /* cell */
#define T_CODE        (FIRSTTYPE +17)  /* code */
#define T_IMMEDIAT    (FIRSTTYPE +18)  /* immediat */
#define T_LIST        (FIRSTTYPE +19)  /* list */
#define T_PAIR        (FIRSTTYPE +20)  /* pair */
#define T_VECTOR      (FIRSTTYPE +21)  /* vector */

/* pleasant macros, isn't it ? */
#define TICV(x,t,i,c,v)     {TYP(x)=(t);IMM(x)=(i);COD(x)=(c);VCT(x)=(v);}
#define GCHAR(x)            (_CDR(x).c)
#define SCHAR(x,v)          {GCHAR(x)=(char)(v);TICV((x),CHAR,1,0,0);}
#define GCODE(x)            (_CDR(x).f)
#define SCODE(x,t,f)        {GCODE(x)=(FUNC)(f);TICV((x),(t),0,1,0);}
```

```
#define GCOMPLEX(x)          GVECTOR(x)
# define GCOMPLEXRE(x)       GCOMPLEX(x)[0]
# define GCOMPLEXIM(x)       GCOMPLEX(x)[1]
# define GCOMPLEXCONTROL(x)GCOMPLEX(x)[2]
#define SCOMPLEXE(x,v)       SVECTOR(x,v,3)
#define GENV(x)              GVECTOR(x)
# define GENVPARENT(x)       GENV(x)[0]
# define GENVHASH(x)         GENV(x)[1]
# define GENVCONTROL(x)      GENV(x)[2]
#define SENV(x,v)            SVECTOR(x,v,3)
#define GFLAG(x)             GINT(x)
#define SFLAG(x,v)           {GFLAG(x)=(WORD)(v);TICV((x),FLAG,1,0,0);}
#define GFREE(x)             CDR(x)
#define SFREE(x,v)           {GFREE(x)=(v);TICV((x),FREE,1,0,0);}
#define GINDIRECT(x)         CDR(x)
#define SINDIRECT(x,v)       {GINDIRECT(x)=(GSM)(v);TICV((x),INDIRECT,1,0,0);}
#define GINT(x)              (_CDR(x).i)
#define SINT(x,v)            {GINT(x)=(int)(v);TICV((x),INTEGER,1,0,0);}
#define GLAMBDA(x)           GVECTOR(x)
#define SLAMBDA(x,v,n)       {GLAMBDA(x)=(VECTOR)(v);TICV((x),CT_LAMBDA+(n),0,1,0);}
#ifdef __LONG
# define GLONGINT(x)         ((_CDR(x).l))
# define SLONGINT(x,v)       {GLONGINT(x)=(long)(v);TICV((x),LONGINT,1,0,0);}
#endif
#define GPOINTER(x)          (_CDR(x).p)
#define SPOINTER(x,v)        {GPOINTER(x)=(void*)(v);TICV((x),POINTER,1,0,0);}
#ifdef __REAL
#  define GREAL(x)           ((_CDR(x).r))
#  define SREAL(x,v)         {GREAL(x)=(real*)(v);TICV((x),REAL,1,0,0);}
#endif
#define GSTRING(x)           (_CDR(x).s)
#define SSTRING(x,v)         {GSTRING(x)=(v);TICV((x),STRING,1,0,0);}
#define GSYMBOL(x)           (_CDR(x).v)
# define GSYMBOLNAME(x)      (GSYMBOL(x)[0])
# define GSYMBOLVALUE(x)     (GSYMBOL(x)[1])
# define GSYMBOLCONTROL(x)   (GSYMBOL(x)[2])
#define SSYMBOL(x,v)         SVECTOR(x,v,3)
#define GUSER(x)             (_CDR(x).u)
#define SUSER(x,v)           {GUSER(x)=(USR*)(v);TICV((x),USER,1,0,0);}
#define GVECTOR(x)           (_CDR(x).v)
#define SVECTOR(x,v,l)       {GVECTOR(x)=(VECTOR)(v);TICV((x),0,1,0,1);LEN(x)=(l);}

/* question macro */
#ifdef __DEBUG
# define _IsNNULL(x) (x)
#else
# define _IsNNULL(x) (1)
#endif
#define IsICV(x,i,c,v)       (_IsNNULL(x)&&(IMM(x)==(i))&&(COD(x)==(c))&&(VCT(x)==(v)))
#define IsAFlag(x)           (IsAtom(x)&&TYP(x)==FLAG)
#define IsAtom(x)            (IsICV (x,1,0,0))
#define IsBoolean(x)         (IsAFlag(x)&&((GFLAG(x)==F_FALSE)||(GFLAG(x)==F_TRUE)))
#define IsCell(x)            (IsICV(x,0,0,0))
#define IsChar(x)            (IsAtom(x)&&TYP(x)==CHAR)
#define IsComplex(x)         _is_complex(x)
#define IsEnv(x)             _is_env(x)
#ifdef __LONG
# define IsExact(x)          (IsInteger(x)||IsLongint(x))
#else
# define IsExact(x)          IsInteger(x)
#endif
#define IsFlag(x,f)          (IsAtom(x)&&(TYP(x)==FLAG)&&(GINT(x)==(f)))
#define IsGarbaged(x)        GBG(x)
#define IsHash(x)            _is_hash(x)
#define IsIdentifier(x)      (IsAtom(x)&&TYP(x)==IDENTIFIER)
#define IsImmediat(x)        (IsAtom(x)||IsVector(x))
#define IsIndirect(x)        (IsAtom(x)&&(TYP(x)==INDIRECT))
#ifdef __REAL
# define IsInexact(x)        (IsReal(x)||IsComplex(x))
#else
# define IsInexact(x)        IsComplex(x)
#endif
```

```
#define IsInteger(x)        (IsAtom(x)&&TYP(x)==INTEGER)
#ifdef __LONG
# define IsLongint(x)       (IsAtom(x)&&TYP(x)==LONGINT)
#endif
#ifdef __REAL
#  ifdef __LONG
#   define IsNumber(x)      (IsInteger(x)||IsLongint(x)||IsReal(x)||IsComplex(x))
#  else
#   define IsNumber(x)      (IsInteger(x)||IsReal(x)||IsComplex(x))
#  endif
#else
#  ifdef __LONG
#   define IsNumber(x)      (IsInteger(x)||IsLongint(x)||IsComplex(x))
#  else
#   define IsNumber(x)      (IsInteger(x)||IsComplex(x))
#  endif
#endif
#define IsPair(x)           (IsCell(x)&&!IsCell(CDR(x))&&!IsFlag(CDR(x),F_NULLOBJ))
#ifdef __REAL
# define IsReal(x)          (IsAtom(x)&&TYP(x)==REAL)
#endif
#define IsString(x)         (IsAtom(x)&&TYP(x)==STRING)
#define IsSymbol(x)         _is_symbol(x)
#define IsTopLevel(x)       (IsEnv(x)   & IsFlag (GENVPARENT(x), F_NULLOBJ))
#define IsUser(x)           (IsAtom(x)&&TYP(x)==USER)
#define IsVector(x)         (IsICV(x,1,0,1))

/* Is... about code cell - x in followed is a cell */
#define GetCodeType(x)      (TYP(x)&_CT_MASKTYPE)
#define GetCodeArg(x)       (TYP(x)&_CT_MASKNARG)
#define GetNofArg(x)        GetCodeArg(x)
#define IsApply(x)          (IsCode(x)&&(GetCodeType(x)&CT_APPLY))
#define IsCode(x)           (IsICV(x,0,1,0))
#define IsCompile(x)        (IsCode(x)&&(GetCodeType(x)&CT_COMPILE))
#ifdef __DYNAMIC
# define IsDynamic(x)       _is_dynamic(x)
#endif
#define IsLambda(x)         (IsCode(x)&&(GetCodeType(x)&CT_LAMBDA))
#define IsLastList(x)       (IsCode(x)&&(GetCodeType(x)&CT_LIST))
#define IsLastOptional(x)   (IsCode(x)&&(GetCodeType(x)&CT_OPTIONAL))
#define IsNoEval(x)         (IsCode(x)&&(GetCodeType(x)&CT_NOEVAL))
#define IsProcedure(x)      (IsCode(x)&&(GetCodeType(x)&CT_PROCEDURE))
#define IsReserved(x)       (IsCode(x)&&(GetCodeType(x)&CT_RESERVED))



/* Function prototypes */

/* A N A L Y S I S . C */
void _analysis          PROTO ((MAIN*_main));
GSM  _aton              PROTO ((MAIN*_main,PSTR _buffer,int _len,long _radix));

/* A T O M . C */
#define _end_atom(m)
void _init_atom         PROTO ((MAIN*_main));
GSM  __make_atom        PROTO ((MAIN*_main, int _type, GSM _ptr));
#define _make_atom(m,t,p) __make_atom((m),(t),(GSM)(p))

/* D I S P L A Y . C */
void  _display          PROTO ((MAIN*_main, GSM list));
void  _display_bye      PROTO ((MAIN*_main));
void  _display_collected PROTO ((MAIN*_main));
void  _display_hello    PROTO ((MAIN*_main));
PSTR  _get_arg_name     PROTO ((MAIN*_main, GSM func));
void  _help             PROTO ((void));
void  _prompt           PROTO ((MAIN*_main));
GSM   memory            PROTO ((MAIN*_main));
GSM   newline           PROTO ((MAIN*_main));

/* E N V . C */
GSM  _define_symbol     PROTO ((MAIN*_main, PSTR _name, GSM value, GSM env));
#define _end_env(m)
```

```
GSM  _find_symbol        PROTO ((MAIN*_main, PSTR _name, GSM env));
GSM  _find_symbol_value  PROTO ((MAIN*_main, PSTR _name, GSM env));
void _init_env           PROTO ((MAIN*_main, int _size_toplevel, int _size));
int  _is_env             PROTO ((GSM env));
GSM  _make_env           PROTO ((MAIN*_main, GSM parent));

/* E R R O R . C */
void __assert            PROTO ((MAIN*_main, PSTR _file, int _line, PSTR _message));
void __error             PROTO ((MAIN*_main, PSTR _message, PSTR _value, int
_level));
void _error              PROTO ((MAIN*_main, int _type, PSTR _value, int _level));
void _wna                PROTO ((MAIN*_main, GSM func));
void _wta                PROTO ((MAIN*_main, CODETYPE _type, int _position));

/* E V A L . C */
GSM _eval                PROTO ((MAIN*_main, GSM exp));

/* G A R B A G E . C */
void  _end_garbage       PROTO ((MAIN*_main));
DWORD _garbage_size      PROTO ((MAIN*_main));
void  _init_garbage      PROTO ((MAIN*_main, DWORD _size));
GSM   cons               PROTO ((MAIN*_main, GSM car, GSM cdr));
GSM   car                PROTO ((MAIN*_main, GSM list));
GSM   cdr                PROTO ((MAIN*_main, GSM list));
GSM   garbage            PROTO ((MAIN*_main));
GSM   garbage_size       PROTO ((MAIN*_main));
#define PUSH(c)          _push(_main,(c))
#define POP()            _pop(_main)
#define POPN(n)          _pop_n(_main,(n))

/* H A S H . C */
GSM  _add_hash_symbol    PROTO ((MAIN*_main, GSM hash, PSTR _name, GSM value));
void _change_hash_value  PROTO ((MAIN*_main, GSM hash, PSTR _name, GSM value));
void _delete_hash_symbol PROTO ((MAIN*_main, GSM hash, PSTR _name));
#define _end_hash(m)
void _flush_hash         PROTO ((MAIN*_main, GSM hash));
void _init_hash          PROTO ((MAIN*_main));
int  _is_hash            PROTO ((GSM hash));
int  _is_symbol          PROTO ((GSM symbol));
GSM  _find_hash_symbol   PROTO ((MAIN*_main, GSM hash, PSTR _name));
GSM  _find_hash_value    PROTO ((MAIN*_main, GSM hash, PSTR _name));
GSM  _make_hash          PROTO ((MAIN*_main, int _size));

/* H E A P . C */
void *_calloc_heap       PROTO ((MAIN*_main, DWORD _nitems, int _size));
DWORD _coreleft_heap     PROTO ((void));
void  _end_heap          PROTO ((MAIN*_main));
void  _free_heap         PROTO ((MAIN*_main, void *_ptr));
MAIN *_init_heap         PROTO ((DWORD _size));
void *_malloc_heap       PROTO ((MAIN*_main, DWORD _size));
#ifdef __Msdos
void *_memset_heap       PROTO ((PSTR _ptr, int _char, DWORD _size));
#else
# define _memset_heap memset
#endif
void *_realloc_heap      PROTO ((MAIN*_main, void *_block, DWORD _size));

/* I N I T . C */
#define INIT_FILE "gsm.s"
void  _close_gsm         PROTO ((MAIN*_main));
void  _end_gsm           PROTO ((MAIN*_main));
MAIN* _init_gsm          PROTO ((int _argc,PSTR *_argv));

/* K E Y W O R D . C */
#define _end_keyword(m)
int  _file_exists        PROTO ((MAIN*_main, PSTR _file));
void _init_keyword       PROTO ((MAIN*_main));
int  _is_defined         PROTO ((MAIN*_main, PSTR _name));
int  _list_length        PROTO ((MAIN*_main, GSM list));
void _load               PROTO ((MAIN*_main, PSTR _file));
void _load_keyword       PROTO ((MAIN*_main, DECLF*_decl));
GSM  null_function       PROTO (());
```

```
/* L A M B D A . C  */
GSM _lambda_exec        PROTO ((MAIN*_main, GSM lambda, GSM arg ));
GSM _lambda_def         PROTO ((MAIN*_main, GSM formal, GSM body));
GSM _lambda_let         PROTO ((MAIN*_main, GSM init,   GSM body));
GSM _lambda_letrec      PROTO ((MAIN*_main, GSM init,   GSM body));

#ifdef __DYNAMIC
/* D Y N A M I C . C */
GSM  _call_dynamic      PROTO ((MAIN*_main, GSM func, GSM arg));
void _end_dynamic       PROTO ((MAIN*_main));
void _init_dynamic      PROTO ((MAIN*_main));
int  _is_dynamic        PROTO ((GSM exp));
void _load_dynamic      PROTO ((PSTR *argv, PSTR file));
#endif

/* M A T H . C */
#define _end_math(m)
int  _is_complex        PROTO ((GSM complexe));
void _init_math         PROTO ((MAIN*_main));
GSM  _make_complex      PROTO ((MAIN*_main, GSM r, GSM i));

/* S I G N A L . C */
void _register_main     PROTO ((MAIN*_main));
void _init_signal       PROTO ((MAIN*_main));
void _end_signal        PROTO ((MAIN*_main));

/* S T A C K . C */
void _end_stack         PROTO ((MAIN*_main));
void _init_stack        PROTO ((MAIN*_main, int _size));
GSM  _push              PROTO ((MAIN*_main, GSM cell));
GSM  _pop               PROTO ((MAIN*_main));
void _pop_n             PROTO ((MAIN*_main, int n));
#define PUSH(c)         _push(_main,(c))
#define POP()           _pop(_main)
#define POPN(n)         _pop_n(_main,(n))

/* V E C T O R . C */
#define _end_vector(m)
void _init_vector       PROTO ((MAIN*_main));
GSM  _make_vector       PROTO ((MAIN*_main, int _len));
GSM  _make_vector_init  PROTO ((MAIN*_main, int _len, GSM obj));
GSM  _vector            PROTO ((MAIN*_main, GSM list));
GSM  vector_to_list     PROTO ((MAIN*_main, GSM vector));
GSM  list_to_vector     PROTO ((MAIN*_main, GSM list));


#endif /* #ifndef __GSM_H */
```

GSMSERVR.H

```
/*
 S E R V E R . H

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
```

```
*/
#ifndef __SERVER_H
#define __SERVER_H


typedef struct s_GSMEXPORT {
  FARPROC test;
  FARPROC null;
} GSMEXPORT;

extern FARPROC _test;
#define test_server_export(i) _test(i)



#endif
```

GSMAPI.H

```
/*
 G S M A P I . H

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#ifndef __GSMAPI_H
#define __GSMAPI_H

#include <config.h>


/* common data type */
#ifdef __DECLARE_COMMON_DATA_TYPE
typedef unsigned char           BYTE;
typedef unsigned                WORD;
typedef unsigned long           DWORD;
typedef char          huge*     PSTR;
# undef __DECLARE_COMMON_DATA_TYPE
#endif /* __DECLARE_COMMON_DATA_TYPE */


#ifdef __Borlandc
# define far   __far
  typedef void far (* FARPROC)();
#else
# define far
  typedef void (* FARPROC)();
#endif
#define export far
#define HLIB   unsigned


typedef struct s_GSMAPI {
  PSTR    name;
  FARPROC proc;
} GSMAPI;
```

```
#ifdef __INCLUDE_API
# include __INCLUDE_API
#else
# include "api.h"
#endif
#ifndef __LOADERFILE
# define __LOADERFILE "api.gsm"
#endif

/* USER */
GSMAPI far * GetApi       (void);
PSTR         GetName      (void);
void         SetExport    (GSMEXPORT far *ge);
int          LibMain      (void);
int          Wep          (void);
int          gsm_lib_main (int, PSTR *, PSTR *);
#endif
```

LOADLIB.H

```
/*
 L O A D L I B . H

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#ifndef __LAODLIB_H
#define __LOADLIB_H

#include "gsmapi.h"

                         /**********Parameters*****************Return***/
#define  API        0x60 /* AH AL BH BL CX       SI:DI  |SI:DI      AX */
# define API_INIT   0xF1 /* G  S  M  1  API_INIT Export |export_api ds */
# define API_END    0xF2 /* G  S  M  1  API_END  .      |.          .  */
# define API_ISLOAD 0xF3 /* G  S  M  1  API_ISLD name   |.          0/1*/

#ifdef __Borlandc
# define asm __asm
#endif
#ifdef __Quickc
# define asm _asm
#endif


#ifndef __GSM_H
# define ERR_NOT_ENOUGHT_MEMORY      (-1)
#endif
#define ERR_UNABLE_TO_LOAD           (-2)
#define ERR_LIBRARY_LOADER_NOT_FOUND (-3)
#define ERR_TOO_MANY_LIBRARY         (-4)
#define ERR_UNABLE_TO_REGISTER       (-5)


/* SERVER */
```

```
int     add_proc_param    (PSTR buffer, int head, void * param, int size);
void    call_proc         (FARPROC proc, PSTR buffer, int head);
#define CALL_PROC(t)      ((t (*)(FARPROC,char*,int))call_proc)
void    end_loadlib       (void);
void    free_library      (HLIB lib_handle);
int     init_loadlib      (PSTR *argv, PSTR file, GSMEXPORT huge * ge);
PSTR    get_lib_from_proc (FARPROC proc);
FARPROC get_proc_address  (HLIB lib_handle, PSTR func_name);
HLIB    load_library      (PSTR lib_name);

#endif
```

Les outils de base systèmes

Les outils de bases sont les mécanismes indispensables pour concevoir un interpréteur. On y trouve la gestion de la mémoire (tas, pile et garbage), la gestion des tables de symboles, des environnements, la gestion des erreurs, de l'affichage de messages, etc.

Le tas

Dans l'environnement DOS, la mémoire des programmes est limitée à 640 Ko. Les bibliothèques du compilateur ne permettent pas de gérer des objets de plus de 64 Ko. De plus, le contrôle de l'allocation de mémoire est très difficile à faire, sans entrer dans le code de démarrage des programmes. C'est pour cette raison que nous avons implémenté la gestion d'un tas.

```
/*
 H E A P . C

 This file describes the garbage collector.

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/


#include <gsm.h>


#ifdef __DEBUG_HEAP
/****************************************************************************
              H E A P   A L L O C A T O R   C H E C K E R
 ****************************************************************************/


#define ADDRESS_ARRAY_SIZE 5000
static void * _debug_heap_array[ADDRESS_ARRAY_SIZE];
static int    _debug_heap_count = 0;
static void * _debug_heap_add  PROTO ((void *_address));
static void   _debug_heap_del  PROTO ((void *_address));
static void   _debug_heap_end  PROTO ((void));
static int    _debug_heap_find PROTO ((void *_address));
static void   _debug_heap_init PROTO ((void));


static void *_debug_heap_add (_address) void *_address; {
register int c = 1;

  for (c = 1; c < ADDRESS_ARRAY_SIZE; c++)
    if (! _debug_heap_array[c]) {
      _debug_heap_array[c]=_address;
      if (c > _debug_heap_count) _debug_heap_count++;
      return _address;
    }
  printf ("Address array to small.\n");
  exit (1);
  return 0; /* compiler warning */
}

static void _debug_heap_del (_address) void *_address; {
int c = _debug_heap_find (_address);

  if (!c) {
    printf ("\n--Address not found--\n");
    return;
  }
  else {
    _debug_heap_array[c] = 0;
    if (c == _debug_heap_count) _debug_heap_count--;
  }
}

static void _debug_heap_end() {
register int c;
         int head = 0;

  for (c = 1; c <= _debug_heap_count; c++)
    if (_debug_heap_array[c]) {
      if (! head) printf ("\n--Address not freed : ");
      head = 1;
      printf ("*");
    }
  if (head) printf ("--\n");
}

static int _debug_heap_find(_address) void *_address; {
register int c;

  for (c = 1; c <= _debug_heap_count; c++)
    if (_debug_heap_array[c] == _address) return c;
  return 0;
}

static void _debug_heap_init(void) {
  memset (_debug_heap_array, 0, sizeof(void*) *ADDRESS_ARRAY_SIZE);
  _debug_heap_count = 1;
}

#else
# define _debug_heap_add(a) (a)
# define _debug_heap_del(a)
# define _debug_heap_end()
# define _debug_heap_init()
#endif /* __DEBUG_HEAP */



#ifdef __C_HEAP
/***************************************************************************
           STANDARD C HEAP ALLOCATOR
```

```
 ***************************************************************************/
#include <alloc.h>

void * _calloc_heap (_main, nitems, size) MAIN*_main; DWORD nitems; int size; {
void * ptr = _malloc_heap (size*nitems);

  if (ptr) _memset_heap (ptr, 0, size*nitems);
  return ptr;
}

void _end_heap (_main) MAIN*_main; {
  _debug_heap_end(_main);
}

void _free_heap (_main, ptr) MAIN*_main; void *ptr; {
  _assert (_main, ptr, exit(1));
  _debug_heap_del (ptr);
  free (ptr);
}

MAIN* _init_heap (_size) DWORD _size; {
MAIN* _main = (MAIN*) calloc (sizeof (MAIN), 1);
  _debug_heap_init();
  return _main;
}

void * _malloc_heap (_main, size) MAIN*_main; DWORD size; {
void * ptr = malloc(size);
  if (! ptr) _error (_main, ERR_NOT_ENOUGHT_MEMORY, "malloc_heap", EXIT);
  _debug_heap_add (ptr);
  return ptr;
}

void * _realloc_heap (_main, block, size) MAIN*_main; void *block; DWORD size; {
void * ptr = realloc(block, size);
  if (! ptr) _error (_main, ERR_NOT_ENOUGHT_MEMORY, 0, EXIT);
  return ptr;
}


#else /* __C_HEAP */

# if ! defined(__Msdos) || !defined(__Ibmpc)
#  include <error: !__C_HEAP is allowed with Ibm pc under MS DOS>
# endif

/***************************************************************************
                    D O S   H E A P   A L L O C A T O R
 ***************************************************************************/

typedef union u_header {
  struct {
    DWORD                size;
    union u_header  huge* ptr;
  } s;
  __ALLIGN_TYPE x;
} HEADER;

#define NALLOC 1

#define __DEBUG_HEAP


void* _memset_heap (_ptr,_char,_size) PSTR _ptr; int _char; DWORD _size; {
PSTR p=_ptr;

  while (_size > MAXINT) {
    memset (p, _char, MAXINT);
    p    +=MAXINT;
    _size-=MAXINT;
  }
  memset (_ptr, _char, (int) _size);
  return _ptr;
}



# ifdef __PC_DOS_REALMODE
/***************************************************************************
         D O S   R E A L   M O D E   H E A P   A L L O C A T O R
 ***************************************************************************/

#ifndef __ALLIGN_TYPE
# include <error. __ALLIGN_TYPE has to defined in config.h>
#endif
#define __DOS_ALLOC  0x48
#define __DOS_FREE   0x49
#define __DOS_SIZE   0x48
#define __DOS_CHANGE 0x4A

/* DOS MEMORY BLOC
typedef union s_MCB {
  BYTE   last;
  WORD   psp;
  WORD   size;
  BYTE   reserved[11];
} MCB;
  ... is unusable because of WORD alignment compiler optimization
*/
typedef BYTE MCB[16];
#define MCB_SIZE(m) ((*(WORD huge*)((((char huge*)(m))-sizeof(MCB))+3))*16)

#ifdef __Borlandc
# define asm  __asm
#endif
#ifdef __Quickc
# define asm  _asm
#endif



static void   __free_heap     PROTO ((MAIN*_main, void *_pointer));
static void    _freebase_heap PROTO ((void*_base));
static void   *_getbase_heap  PROTO ((DWORD _size));
static HEADER *_morecore      PROTO ((MAIN*_main, DWORD _nunits));
static void   *_sbrk_heap     PROTO ((void*_base, long _size));


DWORD _coreleft_heap (void) {
WORD core;

  asm mov ah, __DOS_SIZE
  asm mov bx, 0xFFFF
  asm int 0x21
  asm mov core, bx
  return ((DWORD)core) *16;
}

static void * _sbrk_heap (_base,_size) void*_base; long _size; {
char  huge*ret;
WORD __size     = 0;
WORD __base     = (WORD)(((DWORD)_base)>>16);
long   base_size = (long)MCB_SIZE(_base);

  if (! _size) return 0;
  _size += base_size;

  if (_size %16) __size =1;
  _size /= 16;
  if (_size > MAXINT) goto sbrk_error;

  __size += (WORD) _size;

  asm mov ah, __DOS_CHANGE
  asm mov bx, __size
  asm mov es, __base
  asm int 0x21
  asm jc  sbrk_error

  ret = ((char huge*)_base) +base_size;
  return (char*)ret;

 sbrk_error:
  _error (0, ERR_NOT_ENOUGHT_MEMORY, "sbrk_heap", EXIT);
  return 0;
}

static void * _getbase_heap (_size) DWORD _size; {
WORD   base = 0;
WORD __size = 0;

  if (! _size) goto getbase_error;
  if (_size %16) __size =1;
  _size /= 16;
  if (_size > MAXINT) goto getbase_error;

  __size += (WORD) _size;

  asm mov ah, __DOS_ALLOC
  asm mov bx, __size
  asm int 0x21
  asm jc  getbase_error
  asm mov base, ax

 getbase_error:
  return (void *) (((DWORD) base) <<16);
}

static void _freebase_heap (_base) void*_base; {
WORD __base = (WORD)(((DWORD)_base) >> 16);
  asm mov ah, __DOS_FREE
  asm mov es, __base
  asm int 0x21
  asm jc  freebase_error
  return;

 freebase_error:
  _error (0, ERR_HEAP_CORRUPTED, "freebase_heap", EXIT);
}


static HEADER *_morecore (_main, _nunits) MAIN*_main; DWORD _nunits; {
char   *cp;
HEADER *up;
DWORD   rnu;

  rnu = NALLOC * ((_nunits +NALLOC -1) /NALLOC);

  cp  = _sbrk_heap (_main, rnu * sizeof (HEADER));
  if (!cp) _error (_main, ERR_NOT_ENOUGHT_MEMORY, "morecore", EXIT);
  up = (HEADER*) cp;
  up->s.size = rnu;

  __free_heap (_main, (void*)(up +1));
  _main->heap_free += rnu *sizeof(HEADER);

  return _main->heap_last;
}


void * _calloc_heap (_main, _nitems, _nbytes) MAIN*_main; DWORD _nitems; int _nbytes;
{
void * ptr = _malloc_heap (_main, _nitems*_nbytes);

  if (! ptr) _error (_main, ERR_NOT_ENOUGHT_MEMORY, 0, EXIT);
  _memset_heap (ptr, 0, _nbytes*_nitems);
  return ptr;
}


void _end_heap (_main) MAIN*_main; {
  _debug_heap_end();
  _freebase_heap(_main);
}

static void __free_heap (_main, _pointer) MAIN*_main; void *_pointer; {
HEADER huge*p;
HEADER huge*q;

  _assert (_main, _pointer, exit(1));

  p = (HEADER huge*)((void huge*)_pointer) -1;
  for (q =_main->heap_last; !(p >q && p <q->s.ptr); q =q->s.ptr)
    if (q >=q->s.ptr && (p >q || p <q->s.ptr))
      break;

  if (p+p->s.size == q->s.ptr) {
    p->s.size += q->s.ptr->s.size;
    p->s.ptr   = q->s.ptr->s.ptr;
  }
  else
    p->s.ptr = q->s.ptr;

  if (q+q->s.size == p) {
    q->s.size += p->s.size;
    q->s.ptr   = p->s.ptr;
  }
  else
    q->s.ptr   = p;
  _main->heap_last = q;
}

void _free_heap (_main, _pointer) MAIN*_main; void *_pointer; {
  __free_heap (_main, _pointer);
  _debug_heap_del (_pointer);
}


MAIN* _init_heap (_nbytes) DWORD _nbytes; {
MAIN huge* _main = (MAIN huge*) _getbase_heap (sizeof (MAIN));

  if (! _main) goto error;
  memset (_main, 0, sizeof (MAIN));
  if (_coreleft_heap() < _nbytes) {
    _error (0, ERR_NOT_ENOUGHT_MEMORY, 0, ERR);
    _nbytes = _coreleft_heap();
  }

 _nbytes = _nbytes /sizeof (HEADER);

  if (!(_main->heap_base = _sbrk_heap (_main, sizeof (HEADER) *(_nbytes +2))))
    goto error;;

  ((HEADER*)_main->heap_base)->s.size = _nbytes +1;
  ((HEADER*)_main->heap_base)->s.ptr  = _main->heap_base;

  _main->heap_last = _main->heap_base;
  _main->heap_size =
  _main->heap_free = (_nbytes+2) *sizeof(HEADER) +sizeof(MAIN);

  _debug_heap_init();
  return (MAIN*)_main;

 error:
 _error (0, ERR_NOT_ENOUGHT_MEMORY, "init_heap", EXIT);
 return 0;
}
```

```
void * _malloc_heap (_main, _nbytes) MAIN*_main; DWORD _nbytes; {
HEADER  huge*p, huge*q;
DWORD   nunits;

  nunits = 1 +(_nbytes +sizeof (HEADER) -1) /sizeof (HEADER);
  if ((q = _main->heap_last) == 0) {
    ((HEADER*)_main->heap_base)->s.ptr  = _main->heap_last = q = _main->heap_base;
    ((HEADER*)_main->heap_base)->s.size = 0;
  }
  for (p =q->s.ptr;;q =p, p =p->s.ptr) {
    if (p->s.size >=nunits) {
      if (p->s.size == nunits)
        q->s.ptr = p->s.ptr;
      else {
        p->s.size -= nunits;
        p         += p->s.size;
        p->s.size  = nunits;
      }
      _main->heap_last = q;
      return _debug_heap_add ((char*)(p+1));
    }
    if (p == _main->heap_last)
      if ((p = _morecore (_main, nunits)) == 0)
        return 0;
  }
}
# endif /* __PC_DOS_REALMODE */


# ifdef __PC_DOS_PROTECTEDMODE
/***************************************************************************
     D O S   P R O T E C T E D   M O D E   H E A P   A L L O C A T O R
 ***************************************************************************/
#  include <Error: heap allocator in protected mode is not implemented>
# endif /* __PC_DOS_PROTECTEDMODE */
#endif /* __C_HEAP */
```

La pile

```
/*
 S T A C K . C

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/


#include <gsm.h>

void _init_stack (_main, _size) MAIN*_main; int _size; {
  _assert (_main, _size, exit(1));
  _main->stack_size = _size;
  _main->stack      = (VECTOR) _calloc_heap (_main, sizeof(GSM), _size);
  _main->head       = 0;
}


void _end_stack (_main) MAIN*_main; {
  _free_heap (_main, _main->stack);
  _main->stack_size = 0;
  _main->head       = 0;
  _main->stack      = 0;
}


GSM _push (_main, cell) MAIN*_main; GSM cell; {
  _main->stack[_main->head++] = cell;
  if (! (_main->head < _main->stack_size))
    _error(_main, ERR_STACK_OVERFLOW, 0, GOTOP);
  return cell;
}


GSM _pop (_main) MAIN*_main; {
  if (! ((int)_main->head) > 0)
    _error(_main, ERR_STACK_OVERFLOW, 0, GOTOP);
  return _main->stack[--_main->head];
}


void _pop_n (_main, n) MAIN*_main; int n; {
  _assert (_main, _main->head >= n, _error(_main, ERR_STACK_OVERFLOW, 0,FATAL));
  _main->head -= n;
}
```

Le garbage

Le garbage est un grand tableau de cellule *gsm*. A chaque demande d'allocation effectuée par la fonction `cons` le gestionnaire vérifie s'il doit procéder à une collecte. La technique utilisée est celle du marquage-démarquage des cellules. Le démarquage se fait en parcourant l'environnement global, la pile, le `car` et le `cdr` passés en paramètre de la fonction `cons`.

```
/*
 G A R B A G E . C

 This file describes the garbage collector.

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/
```

```
#include <gsm.h>



static   void _unmark_garbage   PROTO ((GSM _garbage, DWORD _garbage_size));
static   GSM  _garbage_collect  PROTO ((MAIN*_main, GSM car, GSM cdr));
static   GSM  _make_free_list   PROTO ((MAIN*_main, GSM _garbage, DWORD
_garbage_size));
#ifndef __DEBUG_GARBAGE
  static void _mark_cell        PROTO ((GSM current));
  static void _mark_stack       PROTO ((VECTOR stack, WORD head));
#else
  static void _relative_address PROTO ((MAIN*_main, GSM cell));
  static void display_garbage   PROTO ((MAIN*_main));
  static void _mark_cell        PROTO ((MAIN*_main, GSM current));
  static void _mark_stack       PROTO ((MAIN*_main, VECTOR stack, WORD head));
# define CR()  if (_main->out) fprintf(_main->out, "\n")
# define SP()  if (_main->out) fprintf(_main->out, " ")
# define CH(c) if (_main->out) fprintf(_main->out, "%c", c)



 static void _relative_address (_main, cell) MAIN*_main; GSM cell; {
   if (_main->out) {
     if (IsAFlag (cell)) _display (_main, cell);
     else fprintf (_main->out, "%d ", (int)(cell - _main->garbage));
   }
 }

 static void display_garbage (_main) MAIN*_main; {
 int i,
     c = 0;
 GSM p = _main->garbage;

   if (!_main->out) return;
   for (i=0; i <_main->garbage_size; i++) {
     if (!(c++%3)) CR();
     else fprintf (_main->out, "\t");
     fprintf (_main->out, "%d=", i);
     if (IsSymbol(p)) {
       CH('<'); _relative_address(_main, GSYMBOLNAME(p));
       CH('-'); _relative_address(_main, GSYMBOLVALUE(p));
       CH('>');
     }
     else if (IsVector(p)) {
     int i;

       fprintf (_main->out, ": ");
       for (i=0; i < LEN(p); i++) _relative_address (_main, GVECTOR(p)[i]);
       CR();
       c = 0;
     }
     else if (IsCell(p)) {
       CH('('); _relative_address(_main, CAR(p));
       CH('-'); _relative_address(_main, CDR(p));
       CH(')');
     }
     else if (! TYP(p)==FREE) _display (_main, p);
     p++;
   }
 }
#endif /* __DEBUG_GARBAGE */



/* G A R B A G E   C O L L E C T I N G   A L L O C A T O R */
/* Unmarks the whole cell of the garbage - Pass one of the garbage
   collecting.  */
static void _unmark_garbage (_garbage, _garbage_size)
GSM _garbage; DWORD _garbage_size; {
register int i;

  for (i = 0; i < _garbage_size; i++, _garbage++)
    GBG(_garbage) = 0;
}



/* Marks the gsm structure (as list) recursively. Pass two of
   the garbage collecting */
#ifdef __DEBUG_GARBAGE
 static void _mark_cell (_main, current) MAIN*_main; GSM current; {
# define MARKCELL(c) _mark_cell (_main, (c))
#else
 static void _mark_cell (current) GSM current; {
# define MARKCELL(c) _mark_cell(c)
#endif
  if (!current) return;
  if (! IsGarbaged(current)) {

#   ifdef __DEBUG_GARBAGE
    _relative_address (_main, current);
#   endif

    GBG(current) = 1;
    if (IsVector (current)) {
    register i = LEN(current);

      while (i--)
        MARKCELL(*(GVECTOR(current) +i));
    }
    else if (IsLambda (current)) {
    register int    i = GetNofArg (current) +1;
    register VECTOR v = GLAMBDA(current);

      while (i--) MARKCELL(v[i]);
    }
    else if (IsCell (current)) {
      MARKCELL(CAR(current));
      MARKCELL(CDR(current));
    }
    else if (IsIndirect (current)) MARKCELL(GINDIRECT(current));
  }
# undef MARKCELL
}



/* Marks the cell pushed in the stack - Pass tree of the gc */
#ifdef __DEBUG_GARBAGE
 static void _mark_stack (_main, stack, head) MAIN*_main; VECTOR stack; WORD head; {
  while (head--)
    _mark_cell (_main, stack[head]);
}
# else
 static void _mark_stack (stack, head) VECTOR stack; WORD head; {
  while (head--)
    _mark_cell (stack[head]);
}
#endif


/* collects the unused cells of the garbage - Pass four of the gc */
static GSM _make_free_list (_main, _garbage, _garbage_size)
MAIN*_main; GSM _garbage; DWORD _garbage_size; {
GSM free_cell = 0;
GSM current   = _garbage + (_garbage_size -1);
```

```
  while (_garbage_size--) {
    if (! IsGarbaged (current)) {
#     ifdef __DEBUG_GARBAGE
      _relative_address (_main, current);
#     endif
      if (IsVector (current) || IsLambda (current))
        _free_heap (_main, (void*) GVECTOR(current));

      else if (IsAtom(current)) switch (TYP(current)) {
#       ifdef __REAL
        case REAL       :
#       endif
        case IDENTIFIER :
        case STRING     :          /* as generik pointer */
        case POINTER    :
          _free_heap (_main, CDR(current));
          CDR(current) = 0;
          break;
        case USER       :
          if (GUSER(current) && GUSER(current)->free)
            GUSER(current)->free(_main, GUSER(current));
          break;
      }
      SFREE (current, free_cell);
      free_cell = current;
    }
    current--;
  }
  return free_cell;
}



static GSM _garbage_collect (_main, car, cdr) MAIN * _main; GSM car, cdr; {
GSM ret = _main->free;

  if (! ret) {
#   ifdef __DEBUG_GARBAGE
    fprintf (_main->out, "*** garbage collect-begin ***\n");
    display_garbage (_main); CR();
#   endif

    _unmark_garbage (_main->garbage, _main->garbage_size);
#   ifdef __DEBUG_GARBAGE
    if (_main->out) fprintf (_main->out, "\ncurrent environment: ");
    _mark_cell      (_main, _main->current_environment);
    if (_main->out) fprintf (_main->out, "\nmarks stack: ");
    _mark_stack     (_main, _main->stack, _main->head);
    if (_main->out) fprintf (_main->out, "\nmarks car: ");
    _mark_cell      (_main, car);
    if (_main->out) fprintf (_main->out, "\nmarks cdr: ");
    _mark_cell      (_main, cdr);
    if (_main->out) fprintf (_main->out, "\nmarks _main->value: ");
    _mark_cell      (_main, _main->value);
    if (_main->out) fprintf (_main->out, "\nfree list: ");
    ret         =
    _main->free = _make_free_list (_main, _main->garbage, _main->garbage_size);

    display_garbage (_main);
    if (_main->out) fprintf (_main->out, "\n*** garbage collect-end ***\n");
#   else
    _mark_cell      (_main->current_environment);
    _mark_stack     (_main->stack, _main->head);
    _mark_cell      (car);
    _mark_cell      (cdr);
    _mark_cell      (_main->value);
    ret =
    _main->free = _make_free_list (_main, _main->garbage, _main->garbage_size);
#   endif
    _display_collected (_main);
    if (! ret) _error (_main, ERR_NOT_ENOUGHT_MEMORY, 0, FATAL);
  }
  _main->free = CDR(_main->free);
  return ret;
}



void _end_garbage (_main) MAIN*_main; {
  garbage (_main);
  _unmark_garbage (_main->garbage, _main->garbage_size);
  _make_free_list (_main, _main->garbage, _main->garbage_size);
  _free_heap (_main, _main->garbage);
  _main->garbage_size = 0;
  _main->free = _main->garbage = 0;
}



void _init_garbage (_main, _size) MAIN*_main; DWORD _size; {
int i;
GSM cell;

  _assert (_main, _size, exit(1));
  _main->garbage_size = _size;
  _main->free      =
  _main->garbage   = (GSM) _malloc_heap (_main, sizeof (CELL) * _size);
  for (i =0, cell = _main->garbage; i < _size; i++, cell++)
    SFREE (cell, cell+1);

  CDR(cell-1) = 0;
}


GSM garbage (_main) MAIN*_main; {
  _main->free = 0; /* forces the garbage collecting */
  _garbage_collect (_main, 0, 0);
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

DWORD _garbage_size (_main) MAIN*_main; {
register DWORD i   = 0;
register GSM   ptr = _main->free;
  while (ptr) {
    ptr = CDR(ptr);
    i++;
  }
  return i;
}


GSM garbage_size (_main) MAIN*_main; {
  return _make_atom (_main, INTEGER, _garbage_size(_main) * 100L / _main-
>garbage_size);
}



/* EXPORTED KEYWORDS */

GSM cons (_main, car, cdr) MAIN*_main; GSM car, cdr; {
GSM new = _garbage_collect(_main, car, cdr);

  TICV(new, 0, 0, 0, 0);
  CAR(new) = car;
  CDR(new) = cdr;
  return new;
}



GSM car (_main, list) MAIN*_main; GSM list; {
  if (IsCell(list)) return CAR(list);
  _wta (_main, T_LIST, 1);
  return _make_atom(_main, FLAG, F_NULLOBJ);
}



GSM cdr (_main, list) MAIN*_main; GSM list; {
  if (IsCell(list)) return CDR(list);
  _wta (_main, T_LIST, 1);
  return _make_atom(_main, FLAG, F_NULLOBJ);
}
```

Les signaux

Le gestionnaire de signaux intercepte les breaks utilisateurs pour revenir au top-level, et les erreurs sur les opérations réelles. Les autres signaux sont traités selon le systèmes d'exploitation et provoquent l'arrêt en catastrophe de *gsm*.

```
/*
 S I G N A L . C

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.

*/

#include <gsm.h>
#include <signal.h>

static MAIN*__main; /* used by register_main to handle current main */





static void (* old_sigfpe)();
static void (* old_sigint)();


static void _sigfpe() {
  signal (SIGFPE, _sigfpe);
  if (__main) _error (__main, ERR_FLOATING_POINT, 0, GOTOP);
  else exit(1);
}

static void _sigint() {
  signal (SIGINT, _sigint);
  if (__main) _error (__main, ERR_CONTROL_BREAK_PRESSED, 0, GOTOP);
  else exit(1);
}

void _register_main (_main) MAIN*_main; {
  __main=_main;
}
```

```
#ifdef __Borlandc
# pragma argsused
#endif
void _init_signal (_main) MAIN*_main; {
  old_sigfpe = signal (SIGFPE, _sigfpe);
  old_sigint = signal (SIGINT, _sigint);
}

#ifdef __Borlandc
# pragma argsused
#endif
void _end_signal (_main) MAIN*_main; {
/*  signal (SIGFPE, old_sigfpe);
  signal (SIGFPE, old_sigint);*/
}
```

Les outils de bases de l'interpréteur

Les erreurs

Le gestionnaire d'erreur propose de traiter les erreurs qui se produisent dans gsm avec une interface standardisée. La fonction de traitement est `_error()`, qui prend comme paramètre le type de l'erreur, un paramètre de cette erreurs sous forme d'une chaîne de caractères, et l'action à entreprendre (continuer, quitter, quitter en catastrophe).

La fonction `_wta()` (*`Wrong Type of Argument`*) gére le type de paramètre lors des appels de procédure Scheme. `_wna()` (*Wrong number of Argument*) gére le nombre de paramètre des procédures. La fonction `__assert()` est la primitive du gestionnaire des assertions.

```
/*
 E R R O R . C


 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/


#include <gsm.h>

#ifdef __DYNAMIC
# include <loadlib.h>
#endif

#define ERRPORT    (_main?_main->err:stderr)
#define FILENAME   (_main?_main->file:(PSTR)"stdin")
#define LINENUMBER (_main?_main->line:(-1))
#define WARNINGCNT (_main?_main->error:0)
```

```
#define ERRORCNT   (_main?_main->warning:0)


static char _buffer[DEFAULT_BUFFER_SIZE];



/* Primitive assersion function . See _assert() and _assert_false() in gsm.h */
void __assert (_main, _file, _line, _mess)
MAIN*_main; PSTR _file; int _line; PSTR _mess; {
  sprintf (_buffer, "assertion (%s) fail-file %s-line %d", _mess, _file, _line);
  __error (_main, _buffer, 0, ERR);
}



/* Error function with string as message. */
void __error (_main, _message, _value, _err_level)
MAIN * _main; PSTR _message; PSTR _value; int _err_level; {
  if (_err_level == OK) return;
  fprintf (ERRPORT, "%s(%d)-", FILENAME, LINENUMBER);

  switch (_err_level) {
    case WARNING : fprintf (ERRPORT, "Warning: "); if(_main)_main->warning++; break;
    case FATAL   : fprintf (ERRPORT, "Fatal  : "); if(_main)_main->error++;   break;
    default      : fprintf (ERRPORT, "Error  : "); if(_main)_main->error++;   break;
  }
  if (_value) fprintf (ERRPORT, "%s (%s).\n", _message, _value);
  else        fprintf (ERRPORT, "%s.\n", _message);

       if (_err_level == GOTOP) {if (_main) longjmp (_main->goto_toplevel,
GOTOP);else goto exits;}
  else if (_err_level == FATAL) {if (_main)_end_gsm (_main);
else goto exits;}
  else if (_err_level == EXIT) {
   exits:
#   ifdef __DYNAMIC
    _end_dynamic (_main);
#   endif
    exit (0);
  }
}




/* Error function with index as message. */
void _error (_main, _errno, _value, _level)
MAIN*_main; int _errno; PSTR  _value; int _level; {
char   *p;

  if (_level == OK) return;
  switch (_errno) {
#   ifdef __DYNAMIC
    case ERR_UNABLE_TO_REGISTER       : p = "unable to register";             break;
    case ERR_TOO_MANY_LIBRARY         : p = "too many libraries loaded";      break;
    case ERR_LIBRARY_LOADER_NOT_FOUND : p = "library loader file not found";  break;
    case ERR_UNABLE_TO_LOAD           : p = "unable to load library";         break;
#   endif
    case ERR_STRING_TOO_LONG          : p = "string too long";                break;
    case ERR_UNTERMINATED_STRING      : p = "unterminate string";             break;
    case ERR_IDENTIFIER_TOO_LONG      : p = "identifier too long";            break;
    case ERR_REDEFINED_SYMBOL         : p = "redefined symbol";               break;
    case ERR_NOT_ENOUGHT_MEMORY       : p = "not enought memory";             break;
    case ERR_STACK_OVERFLOW           : p = "stack over flow";                break;
    case ERR_UNABLE_TO_OPEN_FILE      : p = "unable to open file";            break;
    case ERR_BAD_VECTOR_INDEX         : p = "bad vector index";               break;
    case ERR_EXTENDED_SYNTAXE         : p = "extended syntax";                break;
    case ERR_BAD_FORMAL               : p = "bad formal argument";            break;
    case ERR_OPEN_FILE                : p = "unable to open file";            break;
    case ERR_BAD_OPERAND              : p = "bad operand";                    break;
    case ERR_UNDEFINED_SYMBOL         : p = "undefined symbol";               break;
    case ERR_DIVISION_BY_ZERO         : p = "division by zero";               break;
    case ERR_UNEXPECTED_EOF           : p = "unexpected end of file";         break;
    case ERR_INVALID_OPTION           : p = "invalid option in command line"; break;
    case ERR_FLOATING_POINT           : p = "undefined floating point error"; break;
    case ERR_CONTROL_BREAK_PRESSED    : p = "control break";                  break;
    case ERR_HEAP_CORRUPTED           : p = "heap corrupted";                 break;
    case ERR_GARBAGE_CORRUPTED        : p = "garbage corrupted";              break;
    default                           :
      _assert_false (_main, "unvalide error type", _end_gsm(_main));
  }
  if (_main)_main->errno = _errno;
  __error (_main, p, _value, _level);
}
```

```
/* wrong number of argument */
void _wna (_main, func) MAIN*_main; GSM func; {
  __error (_main,
           "Wrong number of argument - wait",
           _get_arg_name (_main, func),
           GOTOP);
}
```

```
/* Wrong type of argument */
void _wta (_main, _type, _position) MAIN*_main; CODETYPE _type; int _position; {
PSTR p;
  switch (_type) {
    case FLAG       : p = "flag";         break;
    case CHAR       : p = "character";    break;
    case INTEGER    :
#   ifdef __LONG
    case LONGINT    : p = "exact number"; break;
#   endif
#   ifdef __REAL
    case REAL       : p = "real number";  break;
#   endif
    case STRING     : p = "string";       break;
    case POINTER    : p = "pointer";      break;
    case USER       : p = "user struct";  break;
    case T_BOOL     : p = "boolean";      break;
    case T_CELL     : p = "cell";         break;
    case T_CODE     : p = "procedure";    break;
    case T_IMMEDIAT : p = "immediat";     break;
    case T_LIST     : p = "list";         break;
    case T_PAIR     : p = "pair";         break;
    case T_VECTOR   : p = "vector";       break;
    default : _assert_false (_main, "unknown data type", return);
  }
  sprintf (_buffer, "%s waited in position %d.", p, _position);
  __error (_main, _buffer, 0, GOTOP);
}
```

Les atomes

Ce fichier fournit une procédure qui fabrique des atomes Scheme. Notez que les indicateurs ne font pas partie du garbage mais sont des cellules déclarées comme statiques pour ne pas encombrer le garbage.

```
/*
```

```
 A T O M . C

 Exports the __make_atom() function. See in gsm.h the macro _make_atom witch
 casts the of the _ptr parameter.
 Notes that the flags are staticly defined.


 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>



/* curent used flags */
static CELL _false_flag;
static CELL _overflow_flag;
static CELL _not_impl_flag;
static CELL _null_obj_flag;
static CELL _true_flag;
static CELL _unbounded_flag;
static CELL _undefined_flag;
static CELL _unexpected_flag;
static CELL _unspecified_flag;


#ifdef __Borlandc
# pragma argsused
#endif
void _init_atom (_main) MAIN*_main; {
  /* make sur that performs the followed code only one time, in a
     multitasking context */
  SFLAG (&_false_flag,       F_FALSE);
  SFLAG (&_overflow_flag,    F_OVERFLOW);
  SFLAG (&_not_impl_flag,    F_NOTIMPLEMENTED);
  SFLAG (&_null_obj_flag,    F_NULLOBJ);
  SFLAG (&_true_flag,        F_TRUE);
  SFLAG (&_unbounded_flag,   F_UNBOUNDED);
  SFLAG (&_undefined_flag,   F_UNDEFINED);
  SFLAG (&_unexpected_flag,  F_UNEXPECTED);
  SFLAG (&_unspecified_flag, F_UNSPECIFIED);

  GBG (&_false_flag)      =
  GBG (&_overflow_flag)   =
  GBG (&_not_impl_flag)   =
  GBG (&_null_obj_flag)   =
  GBG (&_true_flag)       =
  GBG (&_unbounded_flag)  =
  GBG (&_undefined_flag)  =
  GBG (&_unexpected_flag) =
  GBG (&_unspecified_flag)= 1;
}
```

```
/* No pointer size hypothesis for the _ptr parameter */
GSM __make_atom (_main, _type, _ptr) MAIN *_main; int _type; GSM _ptr; {
void * tmp;
GSM    cell;

  if (_type != FLAG) {
    cell = NEWCELL(_main);
    switch (_type) {
      case FLAG       : SFLAG(cell, _ptr);     break;
      case FREE       : _assert_false (_main, "can makes FREE atom",
_end_gsm(_main)); break;
      case CHAR       : SCHAR(cell, _ptr);     break;
      case INTEGER    : SINT(cell, _ptr);      break;
#     ifdef __LONG
      case LONGINT    : SLONGINT (cell, _ptr); break;
#     endif
#ifdef __REAL
      case REAL       : tmp = _malloc_heap (_main, sizeof(real));
                        *(real*)tmp = *(real*)_ptr;
                        SREAL(cell, tmp);
                        break;
#endif
      case IDENTIFIER :
      case STRING     : tmp = _malloc_heap (_main, strlen ((char*)_ptr) +1);
                        strcpy ((char*)tmp, (char*)_ptr);
                        SSTRING(cell, tmp);
                        if (_type == IDENTIFIER) TYP(cell) = IDENTIFIER;
                        break;
      case INDIRECT   : SINDIRECT (cell, _ptr);
                        break;
      default         : _assert_false (_main, "unknown atom type", cell =
&_undefined_flag);
    }
  }
  else {
    switch ((WORD) _ptr) {
      case F_FALSE          : cell = & _false_flag;       break;
      case F_OVERFLOW       : cell = & _overflow_flag;    break;
      case F_NOTIMPLEMENTED : cell = & _not_impl_flag;    break;
      case F_NULLOBJ        : cell = & _null_obj_flag;    break;
      case F_TRUE           : cell = & _true_flag;        break;
      case F_UNBOUNDED      : cell = & _unbounded_flag;   break;
      case F_UNDEFINED      : cell = & _undefined_flag;   break;
      case F_UNEXPECTED     : cell = & _unexpected_flag;  break;
      case F_UNSPECIFIED    : cell = & _unspecified_flag; break;
      default               : _assert_false (_main, "undefined flag", cell =
&_undefined_flag);
    }
  }
  return cell;
}
```

Les vecteurs

Les vecteurs sont les tableaux de pointeurs sur des cellules allouées dans le tas. Ils sont la base des tables de symboles, des nombres complexes, et en général de toutes les structures essentielles de Scheme regroupant plus de deux cellules. Les atomes de ce type sont en fait une cellules *gsm* dont le `car` donne la taille du vecteur, et le `cdr` pointe sur le tableau.

```
/*
 V E C T O R . C


 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.
```

```
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Pulic License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>


#define IF_NOT_VECTOR(m,x,p) if(!IsVector(x))_wta((m),T_VECTOR,(p))

static GSM is_vector (_main, obj) MAIN*_main; GSM obj; {
  return _make_atom (_main, FLAG, IsVector(obj) ? F_TRUE : F_FALSE);
}


static GSM vector_fill (_main, vector, obj) MAIN*_main; GSM vector, obj; {
  IF_NOT_VECTOR (_main, vector, 1);
  else {
  WORD   l = LEN  (vector);
  VECTOR v = (VECTOR) GVECTOR (vector);

    while (l--)
      *(v+l) = obj;
  }
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}


static GSM vector_length (_main, vector) MAIN*_main; GSM vector; {
  IF_NOT_VECTOR(_main,vector, 1);
  else return _make_atom (_main, INTEGER, LEN(vector));
  return _make_atom (_main, FLAG, F_NULLOBJ);
}

static GSM vector_ref (_main, vector, index) MAIN*_main; GSM vector, index; {
GSM r = _make_atom (_main, FLAG, F_NULLOBJ);

  IF_NOT_VECTOR (_main, vector, 1);
  else {
  WORD   l = LEN  (vector);
  VECTOR v = (VECTOR) GVECTOR (vector);

    if (! IsInteger (index)) _wta (_main, INTEGER, 2);
    else {
      if (GINT(index) >= l) {
      char b[20];
        sprintf (b, "%d", GINT(index));
        _error (_main, ERR_BAD_VECTOR_INDEX, b, ERR);
      }
      else r = *(v+GINT(index));
    }
  }
  return r ? r : _make_atom (_main, FLAG, F_NULLOBJ);
}

static GSM vector_set (_main, vector, index, obj)
MAIN*_main; GSM vector, index, obj; {

  IF_NOT_VECTOR (_main, vector, 1);
  else {
  WORD   l = LEN  (vector);
  VECTOR v = (VECTOR) GVECTOR (vector);

    if (! IsInteger (index)) _wta (_main, INTEGER, 2);
    else {
      if (GINT(index) >= l) {
      char b[20];
        sprintf (b, "%d", GINT(index));
        _error (_main, ERR_BAD_VECTOR_INDEX, b, ERR);
      }
      else *(v+GINT(index)) = obj;
    }
  }
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

GSM vector_to_list (_main, vector) MAIN*_main; GSM vector; {
GSM r = _make_atom (_main, FLAG, F_NULLOBJ);

  IF_NOT_VECTOR (_main, vector, 1);
  else {
  WORD   l = LEN  (vector);
  VECTOR v = (VECTOR) GVECTOR (vector);

    while (l--)
      r = cons (_main, *(v+l), r);
  }
  return r;
}

#ifdef __Borlandc
# pragma argsused
#endif
GSM list_to_vector (_main, list) MAIN*_main; GSM list; {
  return _make_atom (_main, FLAG, F_NOTIMPLEMENTED);
}

#ifdef __Borlandc
# pragma argsused
#endif
void _init_vector (_main) MAIN*_main; {
}

GSM _make_vector (_main, _len) MAIN*_main; int _len; {
VECTOR a = (VECTOR) _calloc_heap (_main, _len, sizeof(CELL*));
GSM v = NEWCELL(_main);

  SVECTOR(v,a,_len);
  return v;
}


GSM _make_vector_init (_main, _len, obj) MAIN*_main; int _len; GSM obj; {
GSM v = _make_vector (_main, _len);

  if (! IsFlag (v, F_NULLOBJ)) {
  VECTOR c = (VECTOR) GVECTOR(v);

    while (_len--)
      c[_len] = obj;
  }
  return v;
}

#ifdef __Borlandc
# pragma argsused
#endif
GSM _vector (_main, list) MAIN*_main; GSM list; {
  return _make_atom (_main, FLAG, F_NOTIMPLEMENTED);
}
```

Les tables de symboles

Les tables de symboles sont des vecteurs de liste de symboles. Un symboles est un couple nom valeur. Notez la présence de cellule de contrôle qui sont utilisées pour vérifier que l'on a bien affaire à un symbole et à une table.

```
/*
 H A S H . C


 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/


#include <gsm.h>


static WORD _hash_string PROTO ((PSTR  _string, int _max));
static GSM  _make_symbol PROTO ((MAIN*_main, PSTR  _name, GSM value));


int         stricmp      PROTO ((const PSTR s1, const PSTR s2));


static CELL hash_control;
static CELL symbol_control;


static WORD _hash_string (_string, _max) PSTR  _string; int _max; {
WORD c = 0;
  while (*_string) {
    c += (WORD) *_string;
    _string++;
  }
  return c % _max;
}


static GSM _make_symbol (_main, _name, value)
MAIN*_main; PSTR _name; GSM value; {
GSM    symbol;
GSM    name;
VECTOR v;

  if (IsAFlag (value)) {
# define tmp name
    tmp = NEWCELL(_main);      /* Flags are staticly allocated in atom.c. */
    SFLAG(tmp, GFLAG(value));  /* So when a symbol has a flag value, the  */
    value = PUSH(tmp);         /* function creates a new cell             */
# undef tmp
  }
  else PUSH(value);
  symbol  = PUSH(_make_vector (_main, 3));
  name    = _make_atom (_main, STRING, _name);
  v       = GVECTOR(symbol);
  v[0]    = name;
  v[1]    = value;
  v[2]    = &symbol_control;
  SSYMBOL(symbol, v);
  POPN(2);
  return symbol;
}



GSM _add_hash_symbol (_main, hash, _name, value)
MAIN * _main; GSM hash; PSTR _name; GSM value; {
GSM s = _make_symbol (_main, _name, value);

  _assert (_main, IsHash(hash),        error:_end_gsm(_main));
  _assert (_main, strlen (_name), goto error);
  _assert (_main, value,          goto error);

  if (!IsFlag (s, F_NULLOBJ)) {
  WORD   c = _hash_string (_name, LEN (hash)-1);
  VECTOR v = GVECTOR(hash); /* the cell* array */

    *(v+c) = cons (_main, s, *(v+c));
  }
  return s;
}


void _change_hash_value (_main, hash, _name, value)
MAIN*_main; GSM hash; PSTR _name; GSM value; {
GSM s = _find_hash_value (_main, hash, _name);

  if (! IsAFlag (s)) s = value;
}



#ifdef __Borlandc
# pragma argsused
#endif
void _delete_hash_symbol (_main, hash, _name)
MAIN * _main; GSM hash; PSTR _name; {
GSM s;
GSM old;
WORD   c = _hash_string (_name, LEN(hash)-1);
VECTOR v = GVECTOR(hash); /* the cell* array */

  _assert (_main, IsHash(hash),        error:_end_gsm(_main));
  _assert (_main, strlen (_name), goto error);

  s   = (GSM) *(v+c);
  old = 0;

  while (   !IsFlag(s, F_NULLOBJ)
         && stricmp (_name, GSTRING(GSYMBOLNAME(CAR(s))))) {
    old = s;
    s   = CDR(s);
  }
  if (old && ! IsAFlag(s))
    CDR(old) = CDR(s);
}

void _flush_hash (_main, hash) MAIN*_main; GSM hash; {
GSM    null = _make_atom (_main, FLAG, F_NULLOBJ);
int    l    = LEN(hash) -1;
VECTOR v    = GVECTOR(hash);

  _assert (_main, IsHash(hash), _end_gsm(_main));
  while (l--)
```

```
    *(v+1) = null;
}


GSM _find_hash_symbol (_main, hash, _name) MAIN*_main; GSM hash; PSTR _name; {
GSM    s;
WORD   c = _hash_string (_name, LEN(hash)-1);
VECTOR v = GVECTOR(hash); /* the cell* array */

  _assert (_main, IsHash(hash),        error:_end_gsm(_main));
  _assert (_main, strlen (_name), goto error);

  s = (GSM) *(v+c);

  while (   !IsFlag(s, F_NULLOBJ)
         && stricmp (_name, GSTRING(GSYMBOLNAME(CAR(s)))))
    s = CDR(s);

  return IsAFlag(s) ? _make_atom (_main, FLAG, F_UNDEFINED): CAR(s);
}



GSM _find_hash_value (_main, hash, _name) MAIN*_main; GSM hash; PSTR _name; {
GSM s = _find_hash_symbol (_main, hash, _name);
  return IsAFlag(s) ? s : GSYMBOLVALUE(CAR(s));
}

#ifdef __Borlandc
# pragma argsused
#endif
void _init_hash (_main) MAIN*_main; {
  SFLAG (&hash_control,   F_NULLOBJ);
  SFLAG (&symbol_control, F_NULLOBJ);
}

int _is_hash (hash) GSM hash; {
  return IsVector(hash) && (GVECTOR(hash)[LEN(hash)-1] == (GSM)& hash_control);
}

int _is_symbol (symbol) GSM symbol; {
  return IsVector(symbol) && (GVECTOR(symbol)[LEN(symbol)-1] == (GSM)&
symbol_control);
}

GSM _make_hash (_main, _size) MAIN*_main; int _size; {
GSM hash =  _make_vector_init (_main, _size +1, _make_atom (_main, FLAG, F_NULLOBJ));
  _assert (_main, _size, _size = DEFAULT_HASH_TEMP_SIZE);
  GVECTOR(hash)[_size] = & hash_control;
  return hash;
}
```

Les environnements

Un environnement est un vecteur. Il contient un pointeur vers une table des symboles, un pointeur vers l'environnement parent, et un pointeur vers une cellule de contrôle.

```
/*
 E N V . C

 Defines the environment structures.

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
```

```
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>



/* The environment vector size */
#define ENV_SIZE 3


/* cell used for the environment control */
static CELL env_control;


/* Defines a symbol in the given environment. Notes the redefine test. */
GSM _define_symbol (_main, _name, value, env)
MAIN * _main; PSTR _name; GSM value; GSM env; {
GSM s;
GSM hash = GENVHASH(env);

  _assert (_main, IsEnv(env), error : _end_gsm(_main));
  _assert (_main, strlen (_name), goto error);
  _assert (_main, value,          goto error);

  s = _find_hash_symbol (_main, hash, _name);

  if (IsFlag(s, F_UNDEFINED))
    s = _add_hash_symbol (_main, hash, _name, value);
  else {
    if (!IsFlag (GSYMBOLVALUE(s), F_UNBOUNDED))
      _error (_main, ERR_REDEFINED_SYMBOL, _name,
              _main->option.redefine_symbol);
    GSYMBOLVALUE(s) = value;
  }
  return s;
}


/* finds a symbol from an environment. Can shearches the symbol in the
   whole parent environments.
   If the symbol is not found, the function returns the UNDEFINED flag. */
GSM _find_symbol (_main, _name, env) MAIN*_main; PSTR _name; GSM env; {
GSM s;
  _assert (_main, IsEnv(env), error: _end_gsm (_main));
  _assert (_main, strlen (_name), goto error);
  do {
    s = _find_hash_symbol (_main, GENVHASH(env), _name);
    env = GENVPARENT(env);
  } while (!IsFlag (env, F_NULLOBJ) && IsFlag (s, F_UNDEFINED));
  return s;
}


/* Shearches a symbol in only the given environment. */
GSM _find_symbol_lock (_main, _name, env) MAIN*_main; PSTR _name; GSM env; {
  _assert (_main, IsEnv(env), error : _end_gsm (_main));
  _assert (_main, strlen(_name), goto error);
  return _find_hash_symbol (_main, GENVHASH(env), _name);
}
```

```
GSM _find_symbol_value (_main, _name, env) MAIN*_main; PSTR _name; GSM env; {
GSM s = _find_symbol (_main, _name, env);
  _assert (_main, IsEnv(env), error : _end_gsm (_main));
  _assert (_main, strlen(_name), goto error);
  return IsSymbol (s) ? GSYMBOLVALUE(s) : s;
}


/* see macro IsEnv() in gsm.h */
int _is_env(env) GSM env; {
  _assert (0, env, return 0);
  return    IsVector(env) && (GENVCONTROL(env) ==(GSM) & env_control);
}

/* Initialises the toplevel environment. */
void _init_env (_main, _toplevel_size, _size)
MAIN*_main; int _toplevel_size, _size; {

  _assert (_main, _toplevel_size, exit(1));
  _assert (_main, _size,          exit(1));

  SFLAG (&env_control, F_NULLOBJ);
  _main->hash_size            = _toplevel_size;
  _main->hash_temp_size       = _size;
  _main->toplevel             = _make_vector (_main, ENV_SIZE);
  GENVPARENT (_main->toplevel) = _make_atom (_main, FLAG, F_NULLOBJ);
  GENVHASH   (_main->toplevel) = _make_hash (_main, _toplevel_size);
  GENVCONTROL(_main->toplevel) = & env_control;
  _main->current_environment   = _make_atom (_main, FLAG, F_NULLOBJ);
}

/* Makes a new empty environment. */
GSM _make_env (_main, parent) MAIN*_main; GSM parent; {
GSM v = PUSH(_make_vector (_main, ENV_SIZE));

  _assert (_main, IsEnv(parent), error: _end_gsm (_main));
  _assert (_main, ! IsAFlag (v), goto error);
  GENVPARENT(v)  = parent;
  GENVHASH(v)    = _make_hash (_main, _main->hash_temp_size);
  GENVCONTROL(v) = & env_control;
  return POP();
}
```

L'affichage

Dans ce fichier sont regroupées toutes les fonctions d'affichage (les erreurs sont elles affichées dans le fichier `error.c`). Pour modifier l'apparence de *gsm*, il faut modifier les messages de ce fichier.

```
/*
 D I S P L A Y . C

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

```
The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/


#include <gsm.h>
#ifdef __Borlandc
# include <dir.h>
#endif
#ifdef __Quickc
# include <direct.h>
#endif
#ifdef __Unix
#endif



static void _display_atom    PROTO ((MAIN*_main, GSM atom  ));
static void _display_complex PROTO ((MAIN*_main, GSM atom  ));
static void _display_env     PROTO ((MAIN*_main, GSM env   ));
static void _display_symbol  PROTO ((MAIN*_main, GSM symbol));
static void _display_vector  PROTO ((MAIN*_main, GSM vector));




static void _display_atom (_main, atom) MAIN*_main; GSM atom; {
  _assert (_main, IsAtom (atom), return);
  _assert (_main, _main->out,    return);

  switch (TYP(atom)) {
    case CHAR       : fprintf (_main->out, "%c", GCHAR(atom)); break;
    case FLAG       : switch (GINT(atom)) {
        case F_NOTIMPLEMENTED : fprintf(_main->out, "<Not implemented>"); break;
        case F_FALSE          : fprintf(_main->out, "#f"); break;
        case F_TRUE           : fprintf(_main->out, "#t"); break;
        case F_UNEXPECTED     : fprintf(_main->out, "<Unexpected>"); break;
        case F_UNDEFINED      : fprintf(_main->out, "<Undefined>"); break;
        case F_NULLOBJ        : fprintf(_main->out, "<Null object>"); break;
        case F_UNSPECIFIED    : fprintf(_main->out, "<Unspecified>"); break;
        case F_OVERFLOW       : fprintf(_main->out, "<overflow>"); break;
        case F_UNBOUNDED      : fprintf(_main->out, "<unbounded>"); break;
      }
      break;
    case FREE       : _assert_false (_main, "Unable to displays FREE atom",
_end_gsm(_main)); break;
    case IDENTIFIER : fprintf (_main->out, "<%s>", GSTRING(atom)); break;
    case INTEGER    : fprintf (_main->out, "%i",  GINT(atom)); break;
#     ifdef __LONG
    case LONGINT    : fprintf (_main->out, "%ld", GLONGINT(atom)); break;
#     endif
    case POINTER    : fprintf (_main->out, "pointer"); break;;
#     ifdef __REAL
    case REAL       : fprintf (_main->out, "%le ", *GREAL(atom)); break;;
#     endif
    case STRING     : fprintf (_main->out, "%s",   GSTRING(atom)); break;
    case USER       : if (IsUser  (atom) && GUSER(atom)->display)
                        GUSER(atom)->display (_main, GUSER(atom));
                      break;
    default         : _assert_false (_main, "Unable to display this atom type",
return);
  }
}


static void _display_complex (_main, complex) MAIN*_main; GSM complex; {
  _assert (_main, IsComplex (complex), return);
  _assert (_main, _main->out,          return);

  {
    _display_atom (_main, GCOMPLEXRE(complex));
    fprintf (_main->out, "-");
    _display_atom (_main, GCOMPLEXIM(complex));
  }
}


static void _display_env (_main,env) MAIN*_main; GSM env; {
  _assert (_main, IsEnv (env), return);
  _assert (_main, _main->out,          return);

  {
  GSM    p;
  VECTOR v = GVECTOR (GENVHASH (env));  /* hash table vector */
  int    i = LEN (GENVHASH (env));

    while (i--) {
      p = v[i];
      while (!IsFlag (p, F_NULLOBJ)) {
        if (   _main->option.display_reserved
            ||  IsAtom (GSYMBOLVALUE (CAR(p)))
            || IsLambda (GSYMBOLVALUE (CAR(p)))) {
          _display_symbol (_main, CAR(p));
          fprintf (_main->out, _main->option.verbose_eval ? "\n" : " ");
        }
        p = CDR(p);
      }
    }
  }
}


static void _display_vector (_main, vector) MAIN*_main; GSM vector; {
  _assert (_main, IsVector (vector), return);
  _assert (_main, _main->out,          return);

  {
  register        i = LEN(vector);
  register VECTOR v = GVECTOR(vector);

    fprintf (_main->out, "#(");
    while (i--) _display (_main,*(v+i));
    fprintf (_main->out, ")");
  }
}

static void _display_symbol (_main, symbol) MAIN*_main; GSM symbol; {
  _assert (_main, IsSymbol (symbol), return);
  _assert (_main, _main->out,          return);

  fprintf (_main->out, (char*)GSTRING(GSYMBOLNAME(symbol)));
  if (_main->option.verbose_eval) {
    fprintf (_main->out, ": ");
    _display (_main, GSYMBOLVALUE (symbol));
  }
}




void _display (_main, cell) MAIN *_main; GSM cell; {
  if (_main->out) {
         if (IsSymbol(cell))   _display_symbol (_main, cell);
    else if (IsEnv(cell))      _display_env    (_main, cell);
    else if (IsComplex(cell))  _display_complex(_main, cell);
    else if (IsVector(cell))   _display_vector (_main, cell);
    else if (IsIndirect(cell)) {
      fprintf (_main->out, "#@");
      _display(_main, GINDIRECT(cell));
    }
    else if (IsAtom  (cell))   _display_atom   (_main, cell);
    else if (IsCode  (cell)) {
           if (IsProcedure (cell)) fprintf (_main->out, "[procedure ");
      else if (IsReserved  (cell)) fprintf (_main->out, "[reserved ");
      else if (IsNoEval    (cell)) fprintf (_main->out, "[no_eval* ");
      else if (IsLambda    (cell)) fprintf (_main->out, "[lambda* ");
      else if (IsCompile   (cell)) fprintf (_main->out, "[compile* ");
      else if (IsApply     (cell)) fprintf (_main->out, "[apply* ");
      else _assert_false (_main, "not a known procedure type", _end_gsm(_main));
      fprintf (_main->out, "<%s>]", _get_arg_name (_main, cell));
#     ifdef __DEBUG
      if (IsLambda (cell) && _main->option.verbose_eval) {
        fprintf (_main->out, "\nas : ");
        _main->option.verbose_eval = 0; /* protect against recursive definition */
        _display (_main, GLAMBDA(cell)[0]); /* lambda body */
        _main->option.verbose_eval = 1;
      }
#     endif
    }
    else if (IsCell(cell)) {
    GSM ptr = CDR(cell);

      fprintf (_main->out, "(");
      _display (_main, CAR(cell));

      while (!IsFlag (ptr, F_NULLOBJ)) {
        if (IsAtom (ptr)) {
          fprintf (_main->out, ".");
          _display (_main, ptr);
          break;
        }
        fprintf (_main->out, " ");
        _display (_main, CAR(ptr));
        ptr = CDR(ptr);
      }
      fprintf (_main->out, ")");
    }
    else _assert_false (_main, "unknown cell type", longjmp (_main-
>goto_toplevel,0));
  }
}


void _display_bye (_main) MAIN*_main; {
time_t timer;
struct tm *tblock;

  timer = time(NULL);
  tblock = localtime(&timer);

  printf ("\nFinished at %s- %s : Warning=%d - Error=%d\n",
          asctime (tblock),
          _main->file,
          _main->warning, _main->error);
}



void _display_collected (_main) MAIN*_main; {
  if (_main->out && _main->option.verbose_eval)
    fprintf (_main->out, "\n--Collected--\n");
}



#ifdef __Borlandc
# pragma argsused
#endif
void _display_hello (_main) MAIN*_main; {
  printf ("Scheme interpreter - I3S - 92-93\n\n\n");
}
#ifdef __Borlandc
# pragma argsused
#endif
/* Returns a textual description of the argument waited by a function. */
PSTR _get_arg_name (_main, func) MAIN*_main; GSM func; {
```

```
char buffer[100];

  _assert (_main, IsCode (func), return "");
  if (IsLastList (func)) {
    if (IsLastOptional(func))
      sprintf (buffer, "%d-lo", GetNofArg (func) +1);
    else
      sprintf (buffer, "%d-l", GetNofArg (func) +1);
  }
  else if (IsLastOptional(func))
    sprintf (buffer, "%d-o", GetNofArg (func) +1);
  else sprintf(buffer, "%d", GetNofArg(func));
  return buffer;
}


void _help() {
  printf ("gms [-options] <files>*\n");
  printf ("     -%c  : this help\n",                       CLO_HELP);
  printf ("     -%c. : <garbage size>\n",                  CLO_GARBAGE_SIZE);
  printf ("     -%c. : <heap size>\n",                     CLO_HEAP_SIZE);
  printf ("     -%c. : <dynamic libraries loader file>\n", CLO_DYNAMIC_FILE);
  printf ("     -%c. : <hash table size>\n",               CLO_SYMBOL_TABLE_SIZE);
  printf ("     -%c. : <prompt>\n",                        CLO_PROMPT);
  printf ("     -%c. : <temp hash table size>\n",
CLO_TEMP_SYMBOL_TABLE_SIZE);
}

void _prompt (_main) MAIN*_main; {
  if (_main->out) {
  time_t     timer;
  struct tm *tblock;
  char       buffer[DEFAULT_BUFFER_SIZE];
  char      *p_buffer = buffer;
  char      *p_prompt = _main->prompt;
  register   dollar = 0;

    timer  = time(NULL);
    tblock = localtime(&timer);

    while (*p_prompt) {
      switch (*p_prompt) {
        case '$': dollar = 1; break;
        case 'd':
        case 'D': if (! dollar) goto DEFAULT;
                  strcpy (p_buffer, asctime (tblock));
                  p_buffer += strlen (p_buffer) -1; /* asctime() puts a '\n' at end
*/
                  break;
        case 'h':
        case 'H': goto DEFAULT;
        case 'p':
        case 'P': if (! dollar) goto DEFAULT;
                  getcwd (p_buffer, DEFAULT_BUFFER_SIZE - (int)(p_buffer - buffer));
                  p_buffer += strlen (p_buffer);
                  break;
        DEFAULT :
        default : dollar = 0;
                  *p_buffer++ = *p_prompt;
                  if ((p_buffer-buffer)>=DEFAULT_BUFFER_SIZE)
                  break;
      }
      p_prompt++;
    }
    *p_buffer = 0;
    fprintf (_main->out, "\n%s", buffer);
  }
}


GSM memory (_main) MAIN*_main; {
  garbage (_main);
  fprintf (_main->err, "GSM Memory Resources\n");
  fprintf (_main->err, "Heap:    size=%lu free=%lu system=%lu\n", _main->heap_size,
_main->heap_free, _coreleft_heap());
  fprintf (_main->err, "Garbage: size=%lu free=%lu           \n", _main-
>garbage_size, _main->garbage_size - _garbage_size(_main));
  fprintf (_main->err, "Stack:   size=%%d                    \n", _main->stack_size);
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

GSM newline (_main) MAIN*_main; {
  if (_main->out) fprintf (_main->out, "\n");
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}
```

Analyseur & évaluateur

L'analyses

L'analyse se découpe en deux parties, le lecteur de lexèmes et le constructeur d'expressions. Le lecteur de lexèmes est destiné à reconnaître dans fichier toutes les formes syntaxiques reconnues par *gsm*. La version présentée ici est une version minimale. Le constructeur d'expresions construit des listes destinées à l'évaluateur.

```
/*
 A N A L Y S I S . C

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>



#define DIGITS '0':case'1':case'2':case'3':case'5':case'7':case'8':case'9'
#define isalphanum(c) (isalpha(c)||isdigit(c))
#define issign(c) (((c)=='-')||((c)=='+'))
#define issep(c)  (((c)=='(' )||((c)==')')||\
                   ((c)==' ' )||((c)==EOL)||\
                   ((c)=='\t')||((c)==EOF))

#define gch()     fgetc  (_main->in)
#define ugch()    ungetc (c,_main->in)
#define Flag(f)  _make_atom(_main, FLAG, (f))
```

```
static GSM    _atoexact   PROTO ((MAIN*_main,PSTR _buf,int _len,long _radix));
static GSM    _atoinexact PROTO ((MAIN*_main,PSTR _buf,int _len,long _radix));
static GSM    _n_list     PROTO ((MAIN*_main));
static GSM    _list       PROTO ((MAIN*_main));
static GSM    _eval_exp   PROTO ((MAIN*_main));
static LEXEME _lexical    PROTO ((MAIN*_main));
static void   _accept     PROTO ((MAIN*_main, LEXEME _lexeme));
static GSM    _n_list     PROTO ((MAIN*_main));
static GSM    _list       PROTO ((MAIN*_main));
static GSM    _eval_exp   PROTO ((MAIN*_main));
static void   _program    PROTO ((MAIN*_main));



/* Temporary buffer */
static char buffer [DEFAULT_BUFFER_SIZE+1];





#ifdef __REAL
static GSM _atoinexact (_main, _buffer, _len, _radix)
MAIN*_main; PSTR _buffer;  int _len;  long _radix; {
int  c;
int  lead_sgn;
GSM  second;
GSM  cx;
GSM  re;
GSM  im;
int  i     = 0;
int  flg   = 0;
int  point = 0;
real res   = 0.0;
real tmp   = 0.0;
GSM  false = _make_atom (_main, FLAG, F_FALSE);

  if (i >= _len) return false;  /* zero length */

  switch (*_buffer) {           /* leading sign */
    case '-': lead_sgn = -1; i++; break;
    case '+': lead_sgn =  1; i++; break;
    default : lead_sgn =  0;
  }
  if (i==_len) return false;    /* bad if lone `+' or `-' */

  if (_buffer[i] == 'i' || _buffer[i] == 'I') { /* handle `+i' and `-i'   */
    if (lead_sgn == 0) return false; /* must have leading sign */
    if (++i < _len)      return false; /* `i' not last character */
    cx = _make_complex (_main,
                        _make_atom (_main, INTEGER, 0),
                        PUSH(_make_atom (_main, INTEGER, lead_sgn)));
    POP();
    return cx;
  }
  do {                          /* check initial digits */
    switch (c = _buffer[i]) {
      case DIGITS:
        c = c - '0';
        goto accum1;
      case 'D': case 'E': case 'F':
        if (_radix==10) goto out1; /* must be exponent */
      case 'A': case 'B': case 'C':
        c = c-'A'+10;
        goto accum1;
      case 'd': case 'e': case 'f':
        if (_radix==10) goto out1;
      case 'a': case 'b': case 'c':
        c = c-'a'+10;
      accum1:
        if (c >= _radix) return false; /* bad digit for _radix */
        res = res * _radix + c;
```

```
       flg = 1;                       /* res is valid */
       break;
     default:
       goto out1;
   }
 } while (++i < _len);
out1:

 /* if true, then we did see a digit above, and res is valid */
 if (i==_len) goto done;

 /* By here, must have seen a digit,
    or must have next char be a `.' with _radix==10 */
 if (!flg)
   if (!(_buffer[i] == '.' && _radix == 10))
     return false;

 while (_buffer[i] == '#') {  /* optional sharps */
   res *= _radix;
   if (++i==_len) goto done;
 }

 if (_buffer[i] == '/') {
   while (++i < _len) {
     switch (c = _buffer[i]) {
       case DIGITS:
         c = c - '0';
         goto accum2;
       case 'A': case 'B': case 'C': case 'D': case 'E': case 'F':
         c = c-'A'+10;
         goto accum2;
       case 'a': case 'b': case 'c': case 'd': case 'e': case 'f':
         c = c-'a'+10;
       accum2:
         if (c >= _radix) return false;
         tmp = tmp * _radix + c;
         break;
       default:
         goto out2;
     }
   }
 out2:
   if (tmp == 0.0) return false; /* `slash zero' not allowed */
   if (i < _len)
     while (_buffer[i]=='#') { /* optional sharps */
       tmp *= _radix;
       if (++i==_len) break;
     }
     res /= tmp;
     goto done;
 } /* if (_buffer[i] == '/') */

 if (_buffer[i]=='.') {               /* decimal point notation */
   if (_radix != 10) return false; /* must be _radix 10 */
   while (++i < _len) {
     switch (c = _buffer[i]) {
     case DIGITS:
       point--;
       res = res*10.0 + c-'0';
       flg = 1;
       break;
     default:
       goto out3;
     }
   }
  out3:
   if (!flg) return false;     /* no digits before or after decimal point */
   if (i==_len) goto adjust;
   while (_buffer[i]=='#') {   /* ignore remaining sharps */
     if (++i==_len) goto adjust;
   }
 }
 switch (_buffer[i]) {         /* exponent */
   case 'd': case 'D':
   case 'e': case 'E':
   case 'f': case 'F':
   case 'l': case 'L':
   case 's': case 'S': {
   int expsgn = 1,
       expon  = 0;

     if (_radix != 10) return false; /* only in _radix 10 */
     if (++i == _len)  return false; /* bad exponent */
     switch (_buffer[i]) {
       case '-': expsgn=(-1);
       case '+': if (++i==_len) return false; /* bad exponent */
     }
     if (_buffer[i] < '0' || _buffer[i] > '9') return false; /* bad exponent */
     do {
       switch (c = _buffer[i]) {
         case DIGITS:
           expon = expon*10 + c-'0';
           if (expon > MAXEXP)  return false; /* exponent too large */
           break;
         default:
           goto out4;
       }
     } while (++i < _len);
    out4:
     point += expsgn*expon;
   } /* case 's': case 'S': */
 } /* switch (_buffer[i]) */

adjust:
 if (point >= 0) while (point--) res *= 10.0;
 else            while (point++) res /= 10.0;

done:
 /* at this point, we have a legitimate floating point result */
 if (lead_sgn == -1) res = -res;
 if (i==_len) {
   cx = _make_complex (_main,
                       _make_atom (_main, REAL, &res),
                       PUSH(_make_atom (_main, INTEGER, 0)));
   POP();
   return cx;
 }

 if (_buffer[i]=='i' || _buffer[i]=='I') { /* pure imaginary number  */
   if (lead_sgn == 0) return false; /* must have leading sign */
   if (++i < _len)    return false; /* `i' not last character */
   cx = _make_complex (_main,
                       _make_atom (_main, INTEGER, 0),
                       PUSH(_make_atom (_main, REAL, & res)));
   POP();
   return cx;
 }

 switch (_buffer[i++]) {
   case '-':  lead_sgn = -1; break;
   case '+':  lead_sgn = 1;  break;
   case '@': {                 /* polar input for complex number */
   real t;

     /* get a `real' for angle */
     second = _atoinexact (_main, _buffer +i,_len-i,_radix);
     if (IsAFlag   (second)) return false;
     if (IsComplex (second)) return false;
     tmp = * GREAL(second);
     t  = res * cos(tmp);
     re = PUSH(_make_atom (_main, REAL, & t));
     t  = res * sin(tmp);
     im = _make_atom (_main, REAL, & t);
```

```
      cx = _make_complex (_main, re, im);
      POP();
      return cx;
    }
    default: return false;
  }

  /* at this point, last char must be `i' */
  if (_buffer[_len-1] != 'i' && _buffer[_len-1] != 'I') return false;

  /* handles `x+i' and `x-i' */
  if (i == (_len-1)) {
    cx = _make_complex (_main,
                        _make_atom (_main, REAL, & res),
                        PUSH(_make_atom (_main, INTEGER, lead_sgn)));
    POP();
    return cx;
  }

  /* get a `ureal' for complex part */
  second = _atoinexact (_main, _buffer +i, _len-i-1, _radix);
  if (IsAFlag  (second)) return false;
  if (IsComplex(second)) return false; /* not `ureal' */
  tmp = *GREAL(second);
  if (tmp < 0.0)        return false; /* not `ureal' */
  tmp *= (real) lead_sgn;
  re = PUSH(_make_atom (_main, REAL, & res));
  im =      _make_atom (_main, REAL, & tmp);
  cx = _make_complex (_main, re, im);
  POP();
  return cx;
}
#else
# ifdef __Borlandc
#  pragma argsused
# endif
GSM _atoinexact (_main, _buffer, _len, _radix)
MAIN*_main; PSTR _buffer;  int _len;  long _radix; {
  return _make_atom (_main, FLAG, F_NOTIMPLEMENTED);
}
#endif                          /* __REAL */


static GSM _atoexact (_main, _buffer, _len, _radix)
MAIN*_main; PSTR _buffer; int _len; long _radix; {
long ln,
     n        = 0;
int  c,
     i        = 0,
     lead_neg = 0;
GSM  false    = _make_atom (_main, FLAG, F_FALSE);

  if (0 >= _len) return false;  /* zero _length */
  switch (*_buffer) {           /* leading sign */
    case '-': lead_neg = 1;
    case '+': if (++i==_len) return false; /* bad if lone `+' or `-' */
  }
  do {
    switch (c = _buffer[i++]) {

    case 'A': case 'B': case 'C': case 'D': case 'E': case 'F':
      c -= 'A'+10;
      goto accumulate;

    case 'a': case 'b': case 'c': case 'd': case 'e': case 'f':
      c = c-'a'+10;
      goto accumulate;

    accumulate:
      if (c >= _radix) return false; /* bad digit for _radix */
      ln = n;
      n  = n * _radix + c;
      if (n < ln || (n < 0)) goto ovfl;
      break;

    default:
      if (isdigit (c)) {
        c -= '0';
        goto accumulate;
      }
      else return false;   /* not a digit */
    }
  } while (i < _len);
# ifdef __LONG
  ln = n;
  if (lead_neg) if ((n = -n) > 0) goto ovfl;
  if (ln > MAXINT) return _make_atom (_main, LONGINT, n);
  else
# endif
    return _make_atom (_main, INTEGER, (GSM) n);

 ovfl:                          /* overflow scheme integer */
  return false;
}


/* alpha to number */
GSM _aton (_main, _buffer, _len, _radix)
MAIN*_main; PSTR _buffer; int _len; long _radix;{
int  i    = 0;
char ex   = 0,
     ex_p = 0,
     rx_p = 0;  /* Only allow 1 exactness and 1 _radix prefix */
GSM  false = _make_atom (_main, FLAG, F_FALSE);

  if (_len==1)
    if (*_buffer=='+' || *_buffer=='-') /* Catches lone `+' and `-' for speed */
      return false;

  while ((_len-i) >= 2  &&  _buffer[i]=='#'  &&  ++i)
    switch (_buffer[i++]) {
      case 'b': case 'B':  if (rx_p++) return false; _radix = 2;  break;
      case 'o': case 'O':  if (rx_p++) return false; _radix = 8;  break;
      case 'd': case 'D':  if (rx_p++) return false; _radix = 10; break;
      case 'x': case 'X':  if (rx_p++) return false; _radix = 16; break;
      case 'i': case 'I':  if (ex_p++) return false; ex     = 2;  break;
      case 'e': case 'E':  if (ex_p++) return false; ex     = 1;  break;
      default:  return false;
    }

  switch (ex) {
    case 1: return _atoexact   (_main, &_buffer[i],_len-i,_radix);
    case 0: return _atoexact   (_main, &_buffer[i],_len-i,_radix);
    case 2: return _atoinexact (_main, &_buffer[i],_len-i,_radix);
  }
  return false;
}


/* Reads gsm tokens from the input file. Return a lexeme value. */
static LEXEME _lexical (_main) MAIN *_main; {
int  c;           /* char readed from the input file */
int  Comment = 0; /* Contains the line when a C comment starts, else 0 */

  _main->value = Flag(F_NULLOBJ);
  while (1) {
    c = gch();
    if (Comment) {
      if (c == '\n') {
        _main->line++;
        Comment = 0;
      }
    }
    else {
```

```
   SWITCH:
   switch (c) {
    case EOL : _main->line++; break;
    case EOF : return c;
    case ';' : Comment = 1;   break;
    case ' ' : case '\t' :    break;
    case '\'': return QUOTE;
    case '`' : return BACKQUOTE;
    case '#' :
     c = gch();
     switch (c) {
      case 't' : _main->value = _make_atom(_main, FLAG, F_TRUE);  return
T_BOOL;
      case 'f' : _main->value = _make_atom(_main, FLAG, F_FALSE); return
T_BOOL;
      case '\\':
       c = gch();
       character:
       _main->value = _make_atom(_main, CHAR, c);
       return CHAR;
      default  : goto character;
     }
    case '\"': {
    int i = 0;
     do {
      buffer[i] = c = gch();
      if (c == '\\') {
       c = gch();
       switch (c) {
        case 'n' : buffer[i] = '\n'; break;
        case 't' : buffer[i] = '\t'; break;
        case '\\': buffer[i] = '\\'; break;
        default  : ugch();
       }
      }
      if (i >= DEFAULT_BUFFER_SIZE) {
       _error (_main, ERR_STRING_TOO_LONG, 0, ERR);
       exit(1);
      }
      i++;
     } while(c != '\"' && c != '\n' && c != EOF);
     if (c == '\n' || c == EOF) {
      _error (_main, ERR_UNTERMINATED_STRING, 0, ERR);
      if (c != EOF) ugch();
     }
     buffer[i-1] = 0;
     _main->value = _make_atom (_main, STRING, buffer);
     return STRING;
    }
    default :
     if (issep (c)) return c;
     else {
     int b=0;
     int first = c;

      while (! issep (c)) {
       buffer[b++] = c;
       c = gch();
       if (b > DEFAULT_BUFFER_SIZE) {
        _error (_main, ERR_IDENTIFIER_TOO_LONG, 0, ERR);
        while (! issep(c)) c = gch();
        b = DEFAULT_BUFFER_SIZE;
        break;
       }
      }
      buffer[b] = EOS;
      if (c != EOF) ugch();

      if (isdigit (first) || issign (first)) {
       _main->value = _aton (_main, buffer, b, 10L);
       if (IsFlag (_main->value, F_FALSE)) {
```

```
                identifier:
                 if (b >= _main->identifier_length) {
                   _error (_main, ERR_IDENTIFIER_TOO_LONG, 0, ERR);
                   buffer[_main->identifier_length] = '\0';
                 }
                 _main->value = _make_atom (_main, IDENTIFIER, buffer);
                 return IDENTIFIER;
               }
             }
             else goto identifier;
             return TYP(_main->value);
           }
      }
    }
  }
}


/* Reads the next token with _lexical() if the current token is valid */
static void _accept (_main, _lexeme) MAIN *_main; LEXEME _lexeme; {

  if (_main->lexeme == _lexeme) _main->lexeme = _lexical (_main);
  else {
  int i;

    buffer[1] = 0;
    switch (_lexeme) {
      case QUOTE     : buffer[0] = '\''; break;
      case BACKQUOTE : buffer[0] = '`'; break;
      case IDENTIFIER: strcpy (buffer, "identifier"); break;
      default        : buffer[0] = _lexeme;
    }
    strcat (buffer, " expected instead of ");
    i = strlen (buffer);
    if (_main->lexeme < FIRSTTYPE && _main->lexeme >= 0) {
      buffer[i]   = _main->lexeme;
      buffer[i+1] = 0;
    }
    else switch (_main->lexeme) {
      case QUOTE     : strcat (buffer, "\'"); break;
      case BACKQUOTE : strcat (buffer, "`"); break;
      case IDENTIFIER: strcat (buffer, "identifier"); break;
      case EOF       : strcat (buffer, "<end_of_line>"); break;
    }
    strcat (buffer, ".");
    __error(_main, buffer, 0, GOTOP);
  }
}


/* syntaxic analyzer */


/* makes the next element of a list */
static GSM _n_list (_main) MAIN *_main; {
  if (_main->lexeme == EOF) _error (_main, ERR_UNEXPECTED_EOF, 0, GOTOP);
  else if (_main->lexeme != ')') {
  GSM ret = PUSH(NEWCELL(_main));

    CAR(ret) = _eval_exp(_main);
    if (! CAR(ret))_error (_main, ERR_UNEXPECTED_EOF, 0, GOTOP);
    else CDR(ret) = _n_list (_main);
    POP();
    return ret;
  }
  else return Flag(F_NULLOBJ);
  return 0; /* compiler warning */
}


/* makes the first item of a list, and then makes the followed items. */
static GSM _list (_main) MAIN *_main; {
```

```
GSM ret;
  _accept (_main, '(');
  if (_main->lexeme == EOF) _error (_main, ERR_UNEXPECTED_EOF, 0, GOTOP);
  else if (_main->lexeme != ')') {
    ret = PUSH(NEWCELL(_main));
    _main->level++;
    CAR(ret) = _eval_exp (_main);
    if (! CAR(ret)) _error (_main, ERR_UNEXPECTED_EOF, 0, GOTOP);
    else CDR(ret) = _n_list (_main);
    _main->level--;
  }
  else ret = PUSH (_make_atom (_main, FLAG, F_NULLOBJ));
  if (_main->level) _accept (_main, ')');
  else if (_main->lexeme != ')') _accept (_main, ')');
  POP();
  return ret;
}


static GSM _eval_exp (_main) MAIN *_main; {
GSM ret;

  switch (_main->lexeme) {
    case QUOTE      : {
      _accept (_main, QUOTE);
      ret = cons (_main,
                   PUSH (_make_atom (_main, IDENTIFIER, "quote")),
                   PUSH(cons (_main, PUSH(_eval_exp (_main)), Flag(F_NULLOBJ))));
      POPN(3);
      break;
    }
    case '('       : ret = _list (_main); break;
    case EOF       : return 0;
#   ifdef __DEBUG
    case BACKQUOTE :
    case IDENTIFIER:
    case CHAR      :
    case INTEGER   :
#   ifdef __LONG
    case LONGINT   :
#   endif
#   ifdef __REAL
    case REAL      :
#   endif
    case STRING    :
    case T_BOOL    :
#   else
    default        :
#   endif
      ret = PUSH (_main->value);
      if (_main->level) _accept (_main, _main->lexeme);
      POP();
      break;
  }
  return ret;
}

static void _program (_main) MAIN *_main; {
  while (_main->lexeme != EOF) {
  GSM val = _eval_exp (_main);
    if (val) _eval (_main, val);
    _assert (_main,
             _main->current_environment == _main->toplevel,
             error :
             _end_gsm (_main));
    _assert (_main, ! _main->level, goto error);
    _accept (_main, _main->lexeme);
  }
}

void _analysis (_main) MAIN *_main; {
  _main->level  = 0;
  _main->lexeme = _lexical (_main);
  _program (_main);
}
```

L'évaluation

L'évaluateur de Scheme est assez concis. il est capable de traiter toutes les expressions de gsm, compilées ou non. Si un identificateur évalué n'existe pas, une erreur de type GOTO-TOPLEVEL se produit.

```
/*
 E V A L . C

  Exports the fonction eval(). This function evaluates all valid gsm
  construct expression.

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/


#include <gsm.h>

static int  _check_arg    PROTO ((MAIN*_main, GSM func, GSM arg));
static GSM  _eval_arg     PROTO ((MAIN*_main, GSM arg));
static GSM  _call_func    PROTO ((MAIN*_main, GSM func, GSM arg));
static GSM  _apply        PROTO ((MAIN*_main, GSM func, GSM arg));




/* func is the code of the procedure */
/* arg is the list of the arguments  */
static GSM _apply (_main, func, arg) MAIN *_main; GSM func, arg; {
GSM ret;

  _assert (_main, arg, _end_gsm(_main));

  if (! IsCode (func)
# ifdef __DYNAMIC
      && ! IsDynamic (func)
# endif
  ) {
    _wta (_main, T_CODE, 1);
   error:
    ret = _make_atom (_main, FLAG, F_NULLOBJ);
  }
  else {
#   ifdef __DYNAMIC
    if (IsDynamic (func)) ret = _call_dynamic (_main, func, arg);
    else
#   endif
    if (_check_arg (_main, func, arg)) {
      if (IsReserved(func) || IsProcedure(func))
        arg = _eval_arg (_main, arg);
        ret = _call_func (_main, func, PUSH (arg));
        POP();
        if (IsApply (func) && IsTopLevel(_main->current_environment)) {
          _assert (_main, IsCell (ret),       assert:_end_gsm(_main));
          _assert (_main, IsLambda(CAR(ret)), goto assert);
          ret = _eval (_main, PUSH (ret));
          POP();
        }
    }
    else {_wna (_main, func); goto error;}
  }
  return ret;
}


/* Calls a function. Gives all arguments in the standard c-calling convention. */
static GSM _call_func (_main, func, arg) MAIN*_main; GSM func, arg; {
GSM ret = _make_atom (_main, FLAG, F_NULLOBJ);

  _assert (_main, IsCode (func),          return ret);
  _assert (_main, GetNofArg(func) <=CT_7, return ret); /* n of arg limited to seven
*/

  if (IsLambda (func)) {
    _assert (_main, ! IsLastList (func) && !IsLastOptional(func), return ret);
    ret = _lambda_exec (_main, func, arg);
  }
  else if (IsLastList (func))
    switch (GetNofArg (func)) {
      case CT_0:ret=GCODE(func)(_main,arg); break;
      case CT_1:ret=GCODE(func)(_main,CAR(arg),CDR(arg)); break;
      case CT_2:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CDDR(arg)); break;
      case CT_3:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CDDDR(arg));
break;
      case CT_4:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg),
                                CDDDDR(arg)); break;
      case CT_5:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg),
                                CADDDDR(arg),CDDDDDR(arg)); break;
      case CT_6:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg),
                                CADDDDR(arg),CADDDDDR(arg),CDDDDDDR(arg)); break;
      case CT_7:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg),
                                CADDDDR(arg),CADDDDDR(arg),CADDDDDDR(arg),
                                CDDDDDDDR(arg)); break;
    }
  else if (IsLastOptional (func))
    switch (GetNofArg (func)) {
#       define LAST(l) (IsCell(l)?CAR(l):ret)
      case CT_0:ret=GCODE(func)(_main,LAST(arg)); break;
      case CT_1:ret=GCODE(func)(_main,CAR(arg),LAST(CDR(arg))); break;
      case CT_2:ret=GCODE(func)(_main,CAR(arg),CADR(arg),LAST(CDDR(arg))); break;
      case
CT_3:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),LAST(CDDDR(arg)));
                break;
      case CT_4:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg),
                                LAST(CDDDDR(arg))); break;
      case CT_5:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg),
                                CADDDDR(arg),LAST(CDDDDDR(arg))); break;
      case CT_6:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg),
                                CADDDDR(arg),CADDDDDR(arg),LAST(CDDDDDDR(arg)));
break;
      case CT_7:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg),
                                CADDDDR(arg),CADDDDDR(arg),CADDDDDDR(arg),
                                LAST(CDDDDDDDR(arg))); break;
#     undef LAST
    }
  else
    switch (GetNofArg (func)) {
      case CT_0:ret=GCODE(func)(_main); break;
      case CT_1:ret=GCODE(func)(_main,CAR(arg)); break;
      case CT_2:ret=GCODE(func)(_main,CAR(arg),CADR(arg)); break;
      case CT_3:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg)); break;
      case CT_4:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg));
break;
      case CT_5:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg),
                                CADDDDR(arg)); break;
      case CT_6:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg),
                                CADDDDR(arg),CADDDDDR(arg)); break;
      case CT_7:ret=GCODE(func)(_main,CAR(arg),CADR(arg),CADDR(arg),CADDDR(arg),
                                CADDDDR(arg),CADDDDDR(arg),CADDDDDDR(arg)); break;
    }
  return ret;
}



/* Checks the number of argument in a function call. Returns 0 if the
   number of arguments is invalid for this function call, else a
   not nul value */
static int _check_arg (_main, func, arg) MAIN*_main; GSM func, arg; {
int len   = _list_length (_main, arg);
int n_arg = GetNofArg (func);

  _assert (_main, IsCode (func), return 0);

       if (IsLastList     (func)) return len >= n_arg;
  else if (IsLastOptional (func)) return len == n_arg || len == ++n_arg;
  else                            return len == n_arg;
}


/* Evaluates the list of argument in a function call. */
static GSM _eval_arg (_main, arg) MAIN*_main; GSM arg; {
GSM ret;

  if (IsAtom (arg)) {
    _assert (_main, IsFlag (arg, F_NULLOBJ), exit(1));
    return arg;
  }
  ret = cons (_main, PUSH(_eval (_main, CAR(arg))), PUSH(_eval_arg (_main,
CDR(arg))));
  POPN(2);
  return ret;
}

/* Evaluates an expression in an environment. The expression may be all
   valids gsm expresison construct */
GSM _eval (_main, exp) MAIN *_main; GSM exp; {
  if (_main->out && _main->option.verbose_eval) {
    fprintf (_main->out,   "evaluation of : ");
    _display (_main, exp);
    fprintf (_main->out, "\n");
  }
  if (IsIdentifier (exp)) {
  PSTR _name = GSTRING(exp);
    exp = _find_symbol_value (_main, _name, _main->current_environment);
    if (IsFlag (exp, F_UNDEFINED) || IsFlag (exp, F_UNBOUNDED))
      _error (_main, ERR_UNDEFINED_SYMBOL, _name, GOTOP);
  }
  else if (IsIndirect (exp)) exp = _eval (_main, GINDIRECT(exp));
  else if (IsCell  (exp)) {
    _main->level++;
    PUSH (exp);
    exp = _apply (_main, _eval (_main, CAR(exp)), CDR(exp));
    POP();
    _main->level--;
  }
  if (_main->out && ! _main->level) {
    if (_main->option.verbose_eval)
      fprintf (_main->out, "is           : ");
    _display  (_main, exp);
    _prompt   (_main);
  }
  return exp;
}
```

Mots clefs du langage

Sont présentées ici toutes les procédures Scheme. Elles sont appelées directement par l'évaluateur. Elles reçoivent comme paramètres des objets Scheme. De ce fait elle doivent controler le type de leurs paramètres (leur nombre est contrôlé par l'évaluateur).

Tests des objets

```
/*
  I S . C

  This file describes the is_xxx function.
  This file is #include in keyword.c.


 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>



static GSM is_boolean    PROTO ((MAIN*_main, GSM exp));
static GSM is_char       PROTO ((MAIN*_main, GSM exp));
static GSM is_char_alpha PROTO ((MAIN*_main, GSM c));
static GSM is_char_num   PROTO ((MAIN*_main, GSM c));
static GSM is_char_white PROTO ((MAIN*_main, GSM c));
static GSM is_complex    PROTO ((MAIN*_main, GSM exp));
#ifdef __DYNAMIC
static GSM is_dynamic    PROTO ((MAIN*_main, GSM exp));
#endif
static GSM is_eq         PROTO ((MAIN*_main, GSM x, GSM y));
static GSM is_eqv        PROTO ((MAIN*_main, GSM x, GSM y));
static GSM is_equal      PROTO ((MAIN*_main, GSM x, GSM y));
static GSM is_exact      PROTO ((MAIN*_main, GSM exp));
static GSM is_inexact    PROTO ((MAIN*_main, GSM exp));
static GSM is_list       PROTO ((MAIN*_main, GSM exp));
static GSM is_null       PROTO ((MAIN*_main, GSM exp));
static GSM is_number     PROTO ((MAIN*_main, GSM exp));
static GSM is_pair       PROTO ((MAIN*_main, GSM exp));
static GSM is_procedure  PROTO ((MAIN*_main, GSM exp));
static GSM is_string     PROTO ((MAIN*_main, GSM exp));
static GSM is_symbol     PROTO ((MAIN*_main, GSM exp));
static GSM is_vector     PROTO ((MAIN*_main, GSM exp));



static GSM is_boolean (_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsBoolean (exp) ? F_TRUE : F_FALSE);
}

static GSM is_char (_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsChar(exp) ? F_TRUE : F_FALSE);
}

static GSM is_char_alpha (_main, c) MAIN*_main; GSM c; {
  if (! IsChar (c)) return _make_atom (_main, FLAG, F_FALSE);
  return  _make_atom (_main, FLAG, isalpha (GCHAR(c)) ? F_TRUE : F_FALSE);
}

static GSM is_char_num (_main, c) MAIN*_main; GSM c; {
  if (! IsChar (c)) return _make_atom (_main, FLAG, F_FALSE);
  return  _make_atom (_main, FLAG, isdigit (GCHAR(c)) ? F_TRUE : F_FALSE);
}

static GSM is_char_white (_main, c) MAIN*_main; GSM c; {
char cc = GCHAR(c);

  if (! IsChar (c)) return _make_atom (_main, FLAG, F_FALSE);
  return  _make_atom (_main, FLAG, cc==' '||cc=='\n'||cc=='\t' ? F_TRUE : F_FALSE);
}

static GSM is_complex (_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsComplex(exp) ? F_TRUE : F_FALSE);
}

#ifdef __DYNAMIC
static GSM is_dynamic (_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsDynamic(exp) ? F_TRUE : F_FALSE);
}
#endif

static GSM is_eq (_main, x, y) MAIN*_main; GSM x, y; {
  if (IsIdentifier (x)) {
    if (IsIdentifier (y))
      return _make_atom (_main,
                         FLAG,
                         strcmp
((char*)GSTRING(x),(char*)GSTRING(y))?F_FALSE:F_TRUE);
    else return _make_atom (_main, FLAG, F_FALSE);
  }
  return _make_atom (_main, FLAG, x == y ? F_TRUE : F_FALSE);
}

static GSM is_eqv (_main, x, y) MAIN*_main; GSM x, y; {
  if (x==y) goto true;
  if (CAR(x) != CAR(y)) goto false;
  if (IsAtom(x)) {
    switch (TYP(x)) {
      case FLAG       :
      case CHAR       :
      case INTEGER    :
#     ifdef __LONG
      case LONGINT    :
#     endif
                        if (CDR(x) != CDR(y)) goto false;
                        else                  goto true;
      case REAL       : if (*GREAL(x) != *GREAL(y)) goto false;
                        else                        goto true;
      case IDENTIFIER :
      case STRING     : if (strcmp ((char*)GSTRING (x), (char*)GSTRING (y))) goto
false;
                        else                                                 goto
true;
```

```
    }
  }
  false : return _make_atom (_main, FLAG, F_FALSE);
  true  : return _make_atom (_main, FLAG, F_TRUE );
}

static GSM is_equal (_main, x, y) MAIN*_main; GSM x, y; {
GSM ret = is_eqv (_main, x, y);

  if (IsFlag (ret, F_TRUE));
  else if (IsCell (x) && IsCell (y)) {
    ret = is_equal (_main, CAR(x), CAR(y));
    if (IsFlag (ret, F_TRUE)) {
      ret = is_equal (_main, CDR(x), CDR(y));
    }
  }
  else if (IsVector (x) && IsVector (y)) {
    if (LEN(x) == LEN(y)) {
    register i = LEN (x);
      while (i--) {
        ret = is_equal (_main, GVECTOR(x)[i],GVECTOR(y)[i]);
        if (IsFlag (ret, F_FALSE)) break;
      }
    }
  }
  return ret;
}

static GSM is_exact (_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsExact (exp) ? F_TRUE : F_FALSE);
}

static GSM is_inexact (_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsInexact (exp) ? F_TRUE : F_FALSE);
}

static GSM is_list (_main, exp) MAIN*_main; GSM exp; {
GSM ptr = exp;
  while (IsCell(ptr)) ptr = CDR(ptr);
  return _make_atom (_main, FLAG, IsFlag(ptr, F_NULLOBJ)?F_TRUE:F_FALSE);
}

static GSM is_null (_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsFlag(exp, F_NULLOBJ) ? F_TRUE : F_FALSE);
}

static GSM is_number (_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsNumber(exp) ? F_TRUE : F_FALSE);
}

static GSM is_pair (_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsPair(exp) ? F_TRUE : F_FALSE);
}

static GSM is_procedure(_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsCode(exp) ? F_TRUE : F_FALSE);
}

static GSM is_string(_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsString(exp) ? F_TRUE : F_FALSE);
}

static GSM is_symbol (_main, exp) MAIN*_main; GSM exp; {
GSM symb;
  if (! IsIdentifier (exp)) _wta (_main, IDENTIFIER, 1);
  else {
   symb = _find_symbol (_main, GSTRING(exp), _main->current_environment);
   return _make_atom (_main,
                      FLAG,
                         IsFlag (symb, F_UNDEFINED)
                      || IsFlag (GSYMBOLVALUE(symb), F_UNBOUNDED)
                      ? F_FALSE
                      : F_TRUE);
  }
  return _make_atom (_main, FLAG, IsSymbol(exp) ? F_TRUE : F_FALSE);
}

static GSM is_vector (_main, exp) MAIN*_main; GSM exp; {
  return _make_atom (_main, FLAG, IsVector(exp) ? F_TRUE : F_FALSE);
}
```

Lambda expresions

```
/*
 L A M B D A . C

  This file describes the lambda definition. See gsm.d to have the lambda
  definition structure.
  This file is #include in keyword.c.


 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>

#define LET     1
#define LETSTAR 2
#define LETREC  3


static GSM _compile_lambda PROTO ((MAIN*_main, GSM cell, int _eval_sub));
static GSM _make_a_lambda  PROTO ((MAIN*_main, GSM formal, GSM body));
static GSM _make_a_let     PROTO ((MAIN*_main, GSM init,   GSM body, int recursive));





/* compiles a lambda expression in the env environment */
static GSM _compile_lambda (_main, body, _eval_sub) MAIN*_main; GSM body; int
_eval_sub; {

  /* if the expression is an apply of a compiled code, the list is
     replaced by the compiled code */
  if (IsCell(body)) {
    CAR(body) = _compile_lambda (_main, CAR(body), _eval_sub);
    if (_eval_sub && (IsCompile(CAR(body)) || IsApply (CAR(body))))
      body = _eval (_main, body);
    else CDR(body) = _compile_lambda (_main, CDR(body), ! IsNoEval(CAR(body)));
  }

  /* if the identifier don't exist, it is created in the top level
     environment */
  else if (IsIdentifier(body)) {
  GSM val = _find_symbol_value (_main, GSTRING(body), _main->current_environment);
    if (! IsFlag(val, F_UNDEFINED)) body =val;
    else body = GSYMBOLVALUE(_define_symbol (_main,
                             GSTRING(body),
                             _make_atom(_main, FLAG, F_UNBOUNDED),
                             _main->toplevel));
  }
  return body;
}




/* The formal arguments are put in the lambda vector structure, and also in a
temporary
   hash table. The value of these formals arguments are initialised to a NULL flag.
   The formal argument are intialized to UNDEFINED to allow the redefine
   symbol option to process (see _find_symbol()). Next they are flaged
   UNBOUNDED to allow _compile_lambda() to _find the formal value. */
static GSM _make_a_lambda (_main, formal, body) MAIN*_main; GSM formal, body; {
int  n_formal     = 0;
BYTE old_redefine = _main->option.redefine_symbol;
GSM  old_env      = _main->current_environment;
GSM  lambda       = 0;
GSM  ptr          = formal;
GSM  env          = _make_env (_main, _main->current_environment);
GSM  null         = _make_atom (_main, FLAG, F_NULLOBJ);

  _assert (_main, IsEnv (env), _end_gsm(_main));

  _main->current_environment    = env; /* so don't push it */
  _main->option.redefine_symbol = ERR; /* make sure that redifine formal */
  _main->errno                  = OK;  /* sets the errno value */

  /* formal number and check symbol */
  if (IsIdentifier (formal)) {
    n_formal = 1;      /* (lambda a ...) form */
    _define_symbol (_main, GSTRING(formal), _make_atom (_main, INDIRECT, null), env);
  }
  else while (!IsFlag (ptr, F_NULLOBJ)) {  /* (lambda (a b c) ...) form */
    n_formal++;
    if (!IsIdentifier(CAR(ptr))) {
      _error (_main, ERR_BAD_FORMAL, 0, GOTOP);
      goto end;
    }
    _define_symbol (_main, GSTRING(CAR(ptr)), _make_atom (_main,INDIRECT, null),
env);
    ptr = CDR(ptr); /* next formal */
  }
  if (_main->errno != OK) goto end;

  /* makes the lambda structure - vector_size = 1_code_list + n_formal */
  lambda = _make_vector (_main, n_formal +1);
  SLAMBDA (lambda, GLAMBDA (lambda), n_formal);

  /* link the formals arguments between the temporaty main hash table and
     the lambda vector */
  ptr      = formal;
  n_formal = 1;                          /* used as the vector index */
  if (IsIdentifier (formal)) {           /* (lambda a ...) form */
    GLAMBDA(lambda)[n_formal] = _find_symbol_value (_main, GSTRING(formal), env);
  }
  else while (!IsFlag (ptr, F_NULLOBJ)) {  /* (lambda (a b c) ...) form */
    GLAMBDA(lambda)[n_formal] = _find_symbol_value (_main, GSTRING(CAR(ptr)), env);
    n_formal++;
    ptr = CDR(ptr);                      /* next formal */
  }                                      /* link the code to the lambda structure */
  _main->option.redefine_symbol = old_redefine;
  PUSH(lambda);
  GLAMBDA(lambda)[0] = _compile_lambda (_main, body, 1);
  POP();
```

```
 end:
  _main->current_environment    = old_env;
  _main->option.redefine_symbol = old_redefine;
  if (! lambda) longjmp (_main->goto_toplevel, 0);
  return lambda;
}



static GSM _make_a_let (_main, init, body, _let_mode)
MAIN*_main; GSM init, body; int _let_mode; {
/* When _let_mode == 0, the function makes e (let ...) else
   it makes a (letrec ...) */
GSM  value       = _make_atom (_main, FLAG, F_UNBOUNDED);
int  n_init      = 0;
BYTE old_redefine = _main->option.redefine_symbol;
GSM  old_env     = _main->current_environment;
GSM  let         = 0;
GSM  ptr         = init;
GSM  env         = _make_env (_main, _main->current_environment);

  _assert (_main, IsEnv (env), _end_gsm(_main));

  if (_let_mode != LET)
    _main->current_environment  = env; /* so, don't need to PUSH() it */
  else PUSH (env);

  _main->option.redefine_symbol = ERR; /* make sure that argument are uniq */
  _main->errno                  = OK;  /* sets the errno value */

  /* create the let temporary symbol.
     if it is LET,    evals arguments in parent environment
     if it is LETSTAR evals arguments in new environment
     if it is LETREC  creates first the argument without evals them */
  while (!IsAtom (ptr)) {
  GSM the_init = CAR(ptr);

    _assert (_main, IsCell (ptr), goto end);
    n_init++;
    if (   !IsCell (the_init) || !IsIdentifier (CAR(the_init))
        || !IsCell (CDR(the_init)) || !IsFlag (CDR(CDR(the_init)), F_NULLOBJ)) {
      _error (_main, ERR_BAD_FORMAL, 0, GOTOP);
      goto end;
    }
    value = PUSH(NEWCELL(_main));
    if (_let_mode == LETREC) value = CADR (the_init);
    else                     value = _eval (_main, CADR (the_init));

    if (_main->errno != OK) goto end;
    _define_symbol (_main, GSTRING(CAAR(ptr)), _make_atom(_main, INDIRECT, value),
env);
    POP(); /* value */
    ptr = CDR(ptr); /* next init */
  }
  _assert (_main, IsFlag (ptr, F_NULLOBJ), _end_gsm(_main));
  if (_main->errno != OK) goto end;

  /* if LETREC, now evals the argument (and then allows recursive definitions */
  if (_let_mode == LETREC) {
    ptr = init;
    while (! IsAtom (ptr)) {
      value = _find_symbol_value (_main, GSTRING (CAAR(ptr)), env);
      _assert (_main, IsIndirect (value), goto end);
      CDR(value) = _compile_lambda (_main, CDR(value), 1);
      ptr = CDR(ptr);
    }
  }
  /* makes the let structure - vector_size = 1_code_list + n_init */
  let = _make_vector (_main, n_init +1);
  SLAMBDA (let, GLAMBDA (let), 0);
  /* restaure the old option */
  _main->option.redefine_symbol = old_redefine;

  /* joins symbol values to the corresponding vertor positions */
  ptr    = init;
  n_init = 1;   /* used as the vector index */
  while (!IsAtom (ptr)) {
    GLAMBDA(let)[n_init] = _find_symbol_value (_main, GSTRING(CAAR(ptr)), env);
    n_init++;
    ptr = CDR(ptr);                       /* next init */
  }                                       /* link the code to the let structure */
  _main->option.redefine_symbol = old_redefine;

  /* if let, now sets the current environment */
  if (_let_mode == LET) {
    _main->current_environment = env;
    POP();
  }
  PUSH(let);
  GLAMBDA(let)[0] = _compile_lambda (_main, body, 1);
  POP();

 end:
  _main->current_environment    = old_env;
  _main->option.redefine_symbol = old_redefine;
  return cons (_main, let, _make_atom (_main, FLAG, F_NULLOBJ));
}



GSM _lambda_def (_main, formal, body) MAIN*_main; GSM formal, body; {
GSM lambda;

  /* formal argument */
  if (!IsCell(formal)&&!IsIdentifier(formal)&&!IsFlag(formal,F_NULLOBJ)) {
    _error (_main, ERR_BAD_FORMAL, 0, GOTOP);
    lambda = _make_atom (_main, FLAG, F_NULLOBJ);
  }
  else lambda = _make_a_lambda (_main, formal, body);
  return lambda;
}



GSM _lambda_exec (_main, lambda, arg) MAIN*_main; GSM lambda, arg; {
GSM    ret;
int    i;
GSM    ptr_arg;
int    n_arg    = GetNofArg (lambda);
GSM    l_code   = GLAMBDA(lambda)[0];
VECTOR v_formal = GLAMBDA (lambda);

  _assert (_main, IsLambda(lambda), goto error);

  for (i = 1,        ptr_arg = arg;
       i <= n_arg && ! IsFlag (ptr_arg, F_NULLOBJ);
       i++,          ptr_arg = CDR(ptr_arg)) {
    _assert (_main, IsIndirect (v_formal[i]), goto error);
    CDR(v_formal[i]) = _eval (_main, CAR (ptr_arg));
  }
  /* Checks the number of formal arguments */
  if (i <= n_arg) goto error;  /* i has to passes the n_arg value */
  if (! IsFlag (ptr_arg, F_NULLOBJ)) goto error;

  while (! IsAtom (l_code)) {
    _assert (_main, IsCell(l_code), _display (_main, l_code); goto error);
    ret    = _eval (_main, CAR(l_code));
    l_code = CDR(l_code);
  }
  _assert (_main, IsFlag (l_code, F_NULLOBJ), goto error);
  return ret;
 error:
  _wna (_main, lambda);
  return _make_atom (_main, FLAG, F_NULLOBJ);
}


GSM _lambda_let (_main, init, body) MAIN*_main; GSM init, body; {
GSM let;

  if (!IsCell (init)) {
    _error (_main, ERR_BAD_FORMAL, 0, GOTOP);
    let = _make_atom (_main, FLAG, F_NULLOBJ);
  }
  else let = _make_a_let (_main, init, body, LET);
  return let;
}


GSM _lambda_letstar (_main, init, body) MAIN*_main; GSM init, body; {
GSM let;

  if (!IsCell (init)) {
    _error (_main, ERR_BAD_FORMAL, 0, GOTOP);
    let = _make_atom (_main, FLAG, F_NULLOBJ);
  }
  else let = _make_a_let (_main, init, body, LETSTAR);
  return let;
}


GSM _lambda_letrec  (_main, init, body) MAIN*_main; GSM init, body; {
GSM letrec;

  if (!IsCell (init)) {
    _error (_main, ERR_BAD_FORMAL, 0, GOTOP);
    letrec = _make_atom (_main, FLAG, F_NULLOBJ);
  }
  else letrec = _make_a_let (_main, init, body, LETREC);
  return letrec;
}

.
```

Les mots clefs

```
/*
 K E Y W O R D . C

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>
#ifdef __Borlandc
# pragma warn -par
```

```
#endif
/*#include "conv.c"*/
#include "is.c"
#include "lambda.c"
/*#include "string.c"*/


int stricmp PROTO ((const char *s1, const char *s2));


/* The gsm reserved keyword funCTIONS */
static GSM append            PROTO ((MAIN*_main, GSM l1, GSM l2));
static GSM begin             PROTO ((MAIN*_main, GSM first, GSM next));
static GSM cond              PROTO ((MAIN*_main, GSM exp));
static GSM define            PROTO ((MAIN*_main, GSM name, GSM exp));
static GSM display           PROTO ((MAIN*_main, GSM exp));
static GSM ext_syntax        PROTO ((MAIN*_main, GSM exp));
static GSM file_exists       PROTO ((MAIN*_main, GSM file));
static GSM gsm_if            PROTO ((MAIN*_main, GSM cons, GSM Then, GSM Else));
static GSM list              PROTO ((MAIN*_main, GSM list));
static GSM list_length       PROTO ((MAIN*_main, GSM list));
static GSM load              PROTO ((MAIN*_main, GSM file));
static GSM prompt            PROTO ((MAIN*_main, GSM prompt));
static GSM quote             PROTO ((MAIN*_main, GSM exp));
static GSM redef_symb        PROTO ((MAIN*_main, GSM exp));
static GSM reverse           PROTO ((MAIN*_main, GSM list));
static GSM restart           PROTO ((MAIN*_main));
static GSM set               PROTO ((MAIN*_main, GSM exp, GSM val));
static GSM set_car           PROTO ((MAIN*_main, GSM list, GSM car));
static GSM set_cdr           PROTO ((MAIN*_main, GSM list, GSM cdr));
static GSM system_call       PROTO ((MAIN*_main, GSM call));
static GSM top_level         PROTO ((MAIN*_main, GSM exp));
static GSM verbose           PROTO ((MAIN*_main, GSM exp));
static GSM version           PROTO ((MAIN*_main));




static DECLF _keyword[] = {
  {"append",           CT_RESERVED +2,             append        },/*keyword.c*/
  {"begin",            CT_NOEVAL   +1+CT_LIST,     begin         },/*keyword.c*/
  {"boolean?",         CT_RESERVED +1,             is_boolean    },/*is.c     */
  {"car",              CT_RESERVED +1,             car           },/*garbage.c*/
  {"cdr",              CT_RESERVED +1,             cdr           },/*garbage.c*/
  {"char?",            CT_RESERVED +1,             is_char       },/*is.c     */
  {"char-alphabetic?", CT_RESERVED +1,             is_char_alpha },/*is.c     */
  {"char-numeric?",    CT_RESERVED +1,             is_char_num   },/*is.c     */
  {"char-white?",      CT_RESERVED +1,             is_char_white },/*is.c     */
  {"complex?",         CT_RESERVED +1,             is_complex    },/*is.c     */
  {"cond",             CT_NOEVAL   +  CT_LIST,     cond          },/*keyword.c*/
  {"cons",             CT_RESERVED +2,             cons          },/*garbage.c*/
  {"define",           CT_COMPILE  +1+CT_LIST,     define        },/*keyword.c*/
# ifdef __DYNAMIC
  {"dynamic?",         CT_RESERVED +1,             is_dynamic    },/*is.c     */
# endif
  {"display",          CT_RESERVED +1,             display       },/*keyword.c*/
  {"eq?",              CT_RESERVED +2,             is_eq         },/*is.c     */
  {"eqv?",             CT_RESERVED +2,             is_eqv        },/*is.c     */
  {"equal?",           CT_RESERVED +2,             is_equal      },/*is.c     */
# ifdef __REAL
  {"exact?",           CT_RESERVED +1,             is_exact      },/*is.c     */
# endif
  {"exit",             CT_RESERVED +0,             end_gsm       },/*keyword.c*/
  {"extended-syntax",  CT_RESERVED +0+CT_OPTIONAL,ext_syntax    },/*keyword.c*/
  {"file-exists?",     CT_RESERVED +1,             file_exists   },/*keyword.c*/
  {"garbage-collect",  CT_RESERVED +0,             garbage       },/*garbage.c*/
  {"garbage-size",     CT_RESERVED +0,             garbage_size  },/*garbage.c*/
  {"if",               CT_NOEVAL   +2+CT_OPTIONAL,gsm_if        },/*keyword.c*/
# ifdef __REAL
  {"inexact?",         CT_RESERVED +1,             is_inexact    },/*is.c     */
# endif
  {"lambda",           CT_COMPILE  +1+CT_LIST,     lambda_def    },/*keyword.c*/
  {"let",             CT_APPLY    +1+CT_LIST,    lambda_let    },/*lambda.c */
  {"let*",            CT_APPLY    +1+CT_LIST,    lambda_letstar},/*lambda.c */
  {"letrec",          CT_APPLY    +1+CT_LIST,    lambda_letrec },/*lambda.c */
  {"list",            CT_RESERVED +0+CT_LIST,    list          },/*keyword.c*/
  {"list?",           CT_RESERVED +1,            is_list       },/*is.c     */
  {"length",          CT_RESERVED +1,            list_length   },/*keyword.c*/
  {"load",            CT_RESERVED +1,            load          },/*keyword.c*/
  {"newline",         CT_RESERVED,               newline       },/*display.c*/
  {"null?",           CT_RESERVED +1,            is_null       },/*is.c     */
  {"number?",         CT_RESERVED +1,            is_number     },/*is.c     */
  {"pair?",           CT_RESERVED +1,            is_pair       },/*is.c     */
  {"procedure?",      CT_RESERVED +1,            is_procedure  },/*is.c     */
  {"prompt",          CT_RESERVED +0+CT_OPTIONAL,prompt        },/*keyword.c*/
  {"quote",           CT_NOEVAL   +1,            quote         },/*keyword.c*/
  {"redefine-symbol", CT_RESERVED +  CT_OPTIONAL,redef_symb    },/*keyword.c*/
  {"restart",         CT_RESERVED +0,            restart       },/*keyword.c*/
  {"reverse",         CT_RESERVED +1,            reverse       },/*keyword.c*/
  {"set!",            CT_COMPILE  +2,            set           },/*keyword.c*/
  {"set-car!",        CT_RESERVED +2,            set_car       },/*keyword.c*/
  {"set-cdr!",        CT_RESERVED +2,            set_cdr       },/*keyword.c*/
  {"string?",         CT_RESERVED +1,            is_string     },/*is.c     */
  {"symbol?",         CT_COMPILE  +1,            is_symbol     },/*is.c     */
  {"system-call",     CT_RESERVED +1,            system_call   },/*keyword.c*/
  {"top-level",       CT_RESERVED +  CT_OPTIONAL,top_level     },/*keyword.c*/
  {"vector?",         CT_RESERVED +1,            is_vector     },/*is.c     */
  {"verbose",         CT_RESERVED +0+CT_OPTIONAL,verbose       },/*keyword.c*/
  {"version",         CT_RESERVED +0            ,version       },/*keyword.c*/
  {0,0,0 }
};


/* T H E   K E Y W O R D   F U N C T I O N S */

static GSM append (_main, l1,l2) MAIN*_main; GSM l1, l2; {
GSM ptr = l1;

  if (! IsCell (l1)) {error_l1: _wta (_main, T_CELL, 1); goto error;}
  if (! IsCell (l2)) {_wta (_main, T_CELL, 2); goto error;}

  if (IsFlag (l1, F_NULLOBJ)) return l2;
  if (IsFlag (l2, F_NULLOBJ)) return l1;

  while (! IsAtom (CDR(ptr))) ptr = CDR(ptr);
  if (! IsFlag (ptr, F_NULLOBJ)) goto error_l1;
  ptr = l2;

  return l1;

 error: return _make_atom (_main, FLAG, F_NULLOBJ);
}

static GSM begin (_main, first, next) MAIN*_main; GSM first, next; {
GSM ret = _eval (_main, first);

  while (IsCell(next)) {
    ret = _eval (_main, CAR(next));
    next = CDR(next);
  }
  if (! IsFlag (next, F_NULLOBJ)) {
    _wta (_main, CT_LIST, 1);
    return _make_atom (_main, FLAG, F_NULLOBJ);
  }
  else return ret;
}

static GSM cond (_main, exp) MAIN*_main; GSM exp; {
GSM eval;
GSM ret;
GSM clause = exp;

  if (! IsCell (exp)) goto error;
  while (! IsAtom (clause)) {
    if (! IsCell (CAR(clause))) _wta (_main, T_LIST, 1);

    eval = CAR(clause);
    if (   IsIdentifier (CAR (eval))
        && ! stricmp ((char*)GSTRING (CAR(eval)), "else"))
      goto eval_result;

    ret = _eval (_main, CAR(eval));
    if (! IsFlag (ret, F_FALSE)) goto eval_result;

    clause = CDR(clause);
  }
  _assert (_main, IsFlag (clause, F_NULLOBJ), _end_gsm (_main));
  /* result if none true clause found */
  return _make_atom (_main, FLAG, F_UNSPECIFIED);

 eval_result:
   eval = CDR(eval);
   /* exentended syntaxe : (<test> => <recipient>) */
   if (   _main->option.extended_syntaxe
       && IsIdentifier (CAR (eval))
       && !stricmp ((char*)GSTRING(CAR(eval)), "=>")) {
     eval = CDR(eval);
     if (! IsCell (eval)) goto error;
     eval = _eval (_main, CAR (eval));
     if (! IsCode (eval)) goto error;
     if (   ! (GetNofArg (eval) == 1
         || (GetNofArg (eval) == 0 && IsLastOptional (eval)))) goto error;
     eval = cons (_main,
                  PUSH (eval),
                  PUSH (cons (_main,
                              ret,
                              _make_atom (_main, FLAG, F_NULLOBJ))));
     POPN(2);
     return _eval (_main, eval);
   }

   while (! IsAtom (eval)) {
     ret = _eval (_main, CAR(eval));
     eval = CDR(eval);
   }
   _assert (_main, IsFlag (eval, F_NULLOBJ), _end_gsm (_main));
   return ret;

 error :
  _error (_main, ERR_BAD_OPERAND, 0, GOTOP);
  return _make_atom (_main, FLAG, F_NULLOBJ); /* compiler warning */
}

static GSM define (_main, name, exp) MAIN*_main; GSM name, exp; {
  if (IsCell (name)) {
    /* (define (function a b c) (...)) */

    if (! _main->option.extended_syntaxe)
      _error (_main, ERR_EXTENDED_SYNTAXE, 0, GOTOP);
    else if (! IsIdentifier (CAR(name))) _wta (_main, IDENTIFIER, 1);
    else {
      _define_symbol (_main,
                      GSTRING(CAR(name)),
                      PUSH(_lambda_def (_main, CDR(name), exp)),
                      _main->current_environment);
      POP();
    }
  }
  else {
    /* (define identifier value) */
    if (! IsIdentifier (name)) _wta (_main, IDENTIFIER, 1);
    else if (! IsFlag (CDR(exp), F_NULLOBJ))
      _error (_main, ERR_EXTENDED_SYNTAXE, 0, GOTOP);
    else {
      _define_symbol (_main,
                      GSTRING (name),
                      PUSH(_eval (_main, CAR(exp))),
                      _main->current_environment);
      POP();
    }
  }
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

static GSM display (_main, exp) MAIN*_main; GSM exp; {
FILE*old_out = _main->out;
  _main->out = stdout;
  _display (_main, exp);
  _main->out = old_out;
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

static GSM ext_syntax (_main, exp) MAIN*_main; GSM exp; {
  if (IsFlag (exp, F_NULLOBJ))
    return _make_atom (_main, FLAG,_main->option.extended_syntaxe?F_TRUE:F_FALSE);
  else if (IsInteger (exp))        _main->option.extended_syntaxe = GINT(exp);
  else if (IsFlag (exp, F_TRUE))   _main->option.extended_syntaxe = 1;
  else if (IsFlag (exp, F_FALSE))  _main->option.extended_syntaxe = 0;
  else _wta (_main, T_BOOL, 1);
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

static GSM file_exists (_main, file) MAIN*_main; GSM file; {
int res = 0;

  if (! IsString (file)) _wta (_main, STRING, 1);
  else res = _file_exists (_main, GSTRING(file));
  return _make_atom(_main, FLAG, res ? F_TRUE : F_FALSE);
}

static GSM gsm_if (_main, cond, Then, Else)MAIN*_main; GSM cond, Then, Else; {
GSM res = _eval (_main, cond);

  if (IsFlag (res, F_FALSE)) {
    if (IsFlag (Else, F_NULLOBJ)) return _make_atom (_main, FLAG, F_UNSPECIFIED);
    else return _eval (_main, Else);
  }
  else   return _eval (_main, Then);
}

#ifdef __Borlandc
# pragma argsused
#endif
static GSM list (_main, list) MAIN*_main; GSM list; {
  return list;
}

static GSM list_length (_main, list) MAIN*_main; GSM list; {
  return _make_atom (_main, INTEGER, _list_length (_main, list));
}

static GSM load (_main, file) MAIN*_main; GSM file; {
  if(!IsString(file)) _wta (_main, STRING, 1);
  else _load (_main, GSTRING(file));
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

static GSM prompt (_main, string) MAIN*_main; GSM string; {
  if (IsFlag(string, F_NULLOBJ)) return _make_atom (_main, STRING, _main->prompt);
  else if (! IsString (string)) _wta (_main, STRING, 2);
  else {
    strncpy (_main->prompt, (char*)GSTRING(string), PROMPT_LENGTH);
    _main->prompt[PROMPT_LENGTH -1] = 0;
  }
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}
```

```
#ifdef __Borlandc
# pragma argsused
#endif
static GSM quote (_main, exp) MAIN*_main; GSM exp; {
  return exp;
}

static GSM redef_symb (_main, exp) MAIN*_main; GSM exp; {
  if (IsFlag (exp, F_NULLOBJ))
    return _make_atom (_main, FLAG, _main->option.redefine_symbol ? F_TRUE :
F_FALSE);
  else if (IsInteger (exp))         _main->option.redefine_symbol = GINT(exp) %
(FATAL +1);
  else if (IsFlag (exp, F_TRUE))    _main->option.redefine_symbol = WARNING;
  else if (IsFlag (exp, F_FALSE))   _main->option.redefine_symbol = OK;
  else _wta (_main, T_BOOL, 1);
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

static GSM reverse (_main, list) MAIN*_main; GSM list; {
GSM res = _make_atom (_main, FLAG, F_NULLOBJ);
GSM p   = list;

  for(;IsCell(p);p = CDR(p)) {
    res = cons (_main, CAR(p), PUSH(res));
    POP();
  }
  return res;
}

static GSM restart (_main) MAIN*_main; {
  longjmp (_main->goto_restart, 0);
  return 0; /* compiler warning */
}

static GSM set (_main, ident, val) MAIN*_main; GSM ident, val; {
  if (! IsIdentifier (ident)) _wta (_main, IDENTIFIER, 1);
  else {
  GSM symb = _find_symbol (_main, GSTRING(ident), _main->current_environment);
    if (IsFlag (symb, F_UNDEFINED))
      _error (_main, ERR_UNDEFINED_SYMBOL, GSTRING(ident), GOTOP);
    else GSYMBOLVALUE(symb) = val;
  }
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

static GSM set_car (_main, list, car) MAIN*_main; GSM list, car; {
  if (IsCell (list)) CAR(list) = car;
  else _wta (_main, T_LIST, 1);
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

static GSM set_cdr (_main, list, cdr) MAIN*_main; GSM list, cdr; {
  if (IsCell (list)) CDR(list) = cdr;
  else _wta (_main, T_LIST, 1);
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

static GSM system_call (_main, call) MAIN*_main; GSM call; {
  if (! IsString (call)) {
    _wta (_main, STRING, 1);
    return _make_atom (_main, FLAG, F_NULLOBJ);
  }
# ifdef __Windows
  return WinExec (GSTRING(call), SW_NORMAL);
# else
  return _make_atom (_main, INTEGER, (int) system ((char*)GSTRING(call)));
#endif
}

static GSM top_level (_main, exp) MAIN*_main; GSM exp; {
  if (IsFlag (exp, F_NULLOBJ)) return _main->toplevel;
  else if (IsInteger (exp))       _main->option.display_reserved = GINT(exp);
  else if (IsFlag (exp, F_TRUE))  _main->option.display_reserved = 1;
  else if (IsFlag (exp, F_FALSE)) _main->option.display_reserved = 0;
  else _wta (_main, T_BOOL, 1);
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

static GSM verbose (_main, exp) MAIN*_main; GSM exp; {
  if (IsFlag (exp, F_NULLOBJ))
    return _make_atom (_main, FLAG, _main->option.verbose_eval ? F_TRUE : F_FALSE);
  else if (IsInteger (exp))         _main->option.verbose_eval = GINT(exp);
  else if (IsFlag (exp, F_TRUE))    _main->option.verbose_eval = 1;
  else if (IsFlag (exp, F_FALSE))   _main->option.verbose_eval = 0;
  else _wta (_main, T_BOOL, 1);
  return _make_atom (_main, FLAG, F_UNSPECIFIED);
}

static GSM version (_main) MAIN*_main; {
  return _make_atom (_main, STRING, __VERSION);
}
```

```
#ifdef __Borlandc
# pragma argsused
#endif
int _file_exists (_main, _file) MAIN*_main; PSTR _file; {
FILE * f;

  f = fopen ((char*)_file, "r");
  if (f) {
    fclose (f);
    return 1;
  }
  return 0;
}


void _init_keyword (_main) MAIN*_main; {
  _load_keyword  (_main, _keyword);
  _define_symbol (_main, "machine", _make_atom (_main, STRING,
IMPLEMENTATION_MACHINE),_main->toplevel);
  _define_symbol (_main, "system",  _make_atom (_main, STRING, IMPLEMENTATION_SYSTEM
) ,_main->toplevel);
}


#ifdef __Borlandc
# pragma argsused
#endif
int _is_defined (_main, _name) MAIN*_main; PSTR _name; {
register int i = 0;
  while (_keyword[i].name) {
    if (! stricmp ((char*)_name, (char*)_keyword[i].name)) return i;
    i++;
  }
  return F_NULLOBJ;
}


#ifdef __Borlandc
# pragma argsused
#endif
int _list_length (_main, list) MAIN*_main; GSM list; {
int i = 0;

  while (IsCell(list)) {
    list = CDR(list);
    i++;
  }
  if (!IsFlag (list, F_NULLOBJ)) return -1;
  return i;
}


void _load (_main, _file) MAIN*_main; PSTR _file; {
OPTION  option;
WORD    stack_hd;
jmp_buf old_toplevel;
int     level    = _main->level;
LEXEME  lex      = _main->lexeme;
int     old_line = _main->line;
FILE   *old_in   = _main->in;
FILE   *old_out  = _main->out;
PSTR    old_file = _main->file;

  PUSH(_main->value);
  if (_file) {
    if (! _main->option.verbose_eval) _main->out= 0;
    _main->in = fopen ((char*)_file, "r");
  }
  else {
    _main->err  =  stderr;
    _main->out  =  stdout;
    _main->in   =  stdin;
    _file       = "stdin";
  }
  if (!_main->in) {_error (_main, ERR_OPEN_FILE, _file, ERR); goto end;}
  _main->line   = 1;
  _main->file   = _malloc_heap (_main, strlen ((char*)_file)+1);
  strcpy ((char*)_main->file, (char*)_file);
  stack_hd = _main->head;
  jmp_cpy (old_toplevel, _main->goto_toplevel);
  memcpy (& option, & _main->option, sizeof (OPTION));

  setjmp (_main->goto_toplevel);
  memcpy (& _main->option, & option, sizeof (OPTION));
  _main->current_environment = _main->toplevel;
  _main->head   = stack_hd;
  _main->level  = level;
  _prompt (_main);
  _analysis (_main);

  _free_heap (_main, _main->file);
  _main->file   = 0;
  if (_main->in != stdin) fclose (_main->in);

 end:
  _main->head   = stack_hd;
  _main->value  = POP();
  _main->level  = level;
  _main->lexeme = lex;
  _main->line   = old_line;
  _main->in     = old_in;
  _main->out    = old_out;
  _main->file   = old_file;
  jmp_cpy (_main->goto_toplevel, old_toplevel);
}

GSM null_function(){
  /* _make_atom don't need main to return a FLAG */
  return _make_atom (0, FLAG, F_UNDEFINED);
}

void _load_keyword (_main, _decl) MAIN*_main; DECLF *_decl; {
  while (_decl->name) {
  GSM new = NEWCELL(_main);

    _assert (_main, _decl, return);
    SCODE (new, _decl->arg, _decl->f);
    _define_symbol (_main, _decl->name, new, _main->toplevel);
    _decl++;
  }
}
```

Opérateurs arithmétiques

Noyeau central

```
/*
 M A T H . C


 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>

#ifdef __Borlandc
# pragma warn -cln
#endif


#define ADD     0
#define SUB     1
#define MUL     2
#define DIV     3
#define MOD     4
#define REM     5

#define OP_NUM  6
#define TYP_NUM 7

static CELL _complex_control;

static int _is_zero    PROTO ((GSM));
static GSM _cast       PROTO ((MAIN*_main, GSM who, GSM in));
static GSM __cast_down PROTO ((MAIN*_main, WORD _type, GSM val));
#define _cast_down(m,t,v) __cast_down((m),(t),(GSM)(v))
static GSM _execute    PROTO ((MAIN*_main, GSM x, GSM y, int _op));

#include "mathadd.c"
#include "mathsub.c"
#include "mathmul.c"
#include "mathdiv.c"
```

```
static FUNC _operator[OP_NUM][TYP_NUM] = {
  {  (FUNC) add_int,
     (FUNC) add_long,
     (FUNC) add_real,
     (FUNC) add_complex,
     (FUNC) add_string},

  {  (FUNC) sub_int,
     (FUNC) sub_long,
     (FUNC) sub_real,
     (FUNC) sub_complex,
     (FUNC) sub_string},

  {  (FUNC) mul_int,
     (FUNC) mul_long,
     (FUNC) mul_real,
     (FUNC) mul_complex,
     (FUNC) mul_string},

  {  (FUNC) div_int,
     (FUNC) div_long,
     (FUNC) div_real,
     (FUNC) div_complex,
     (FUNC) div_string}
};


static int _is_zero (cell) GSM cell; {
  if (IsComplex (cell)) {
    return _is_zero (GCOMPLEXRE(cell)) && _is_zero (GCOMPLEXIM (cell));
  }
  else if (IsAtom (cell))
    switch (TYP(cell)) {
      case CHAR    : return !(int) GCHAR    (cell);
      case INTEGER : return !(int) GINT     (cell);
#     ifdef __LONG
      case LONGINT : return !(int) GLONGINT (cell);
#     endif
#     ifdef __REAL
      case REAL    : return !(int)*GREAL    (cell);
#     endif
      default      : return 0;
    }
  else return 0;
}

static GSM _cast (_main, who, in)MAIN*_main; GSM who, in; {
  _assert (_main, IsAtom(who) || IsComplex (who), goto undefined);
  _assert (_main, IsAtom(in)  || IsComplex (in),  goto undefined);

  if (TYP(who) != TYP(in)) {
  GSM new = NEWCELL(_main);

    switch (TYP(in)) {
      case CHAR:
        switch (TYP(who)) {
          case INTEGER: SCHAR(new, (char) GINT(who)); break;
#         ifdef __LONG
          case LONGINT: SCHAR(new, (char) GLONGINT(who)); break;
#         endif
#         ifdef __REAL
          case REAL   : SCHAR(new, (char) *GREAL(who)); break;
#         endif
          case STRING : SCHAR(new, * GSTRING(who)); break;
          default     : goto undefined;
        }
        who = new;
        break;
      case INTEGER:
        switch (TYP(who)) {
          case CHAR   : SINT(new, (int) GCHAR(who)); break;
#         ifdef __LONG
```

```
          case LONGINT: SINT(new, (int) GLONGINT(who)); break;
#         endif
#         ifdef __REAL
          case REAL   : SINT(new, (int) GREAL(who)); break;
#         endif
          case STRING :
          default     : goto undefined;
        }
        who = new;
        break;
#     ifdef __LONG
      case LONGINT:
        switch (TYP(who)) {
          case CHAR   : SLONGINT(new, (long) GCHAR(who)); break;
          case INTEGER: SLONGINT(new, (long) GINT(who));  break;
#         ifdef __REAL
          case REAL   : SLONGINT(new, (long) GREAL(who)); break;
#         endif
          case STRING :
          default     : goto undefined;
        }
        who = new;
        break;
#       endif /* __LONG */
#     ifdef __REAL
      case REAL: {
      real r;

        switch (TYP(who)) {
          case CHAR   : r = (real) GCHAR(who); break;
          case INTEGER: r = (real) GINT (who); break;
#         ifdef __LONG
          case LONGINT: r = (real) GLONGINT(who); break;
#         endif
          case STRING :
          default     : goto undefined;
        }
        who = _make_atom (_main, REAL, &r);
        break;
      }
#     endif  /* __REAL */
      case STRING: {
      char buffer[50];

        switch (TYP(who)) {
          case CHAR   : buffer[0] = GCHAR(who); buffer[1] = '\0'; break;
          case INTEGER: sprintf (buffer, "%d", GINT(who)); break;
#         ifdef __LONG
          case LONGINT: sprintf (buffer, "%ld", GLONGINT(who)); break;
#         endif
#         ifdef __REAL
          case REAL   : sprintf (buffer, "%le", GREAL(who)); break;
#         endif
          default     :
            if (IsVector(who))
              goto notimplemented;
        }
        who = new;
        break;
      }
    }
  }
  return who;
 undefined:
  return _make_atom(_main, FLAG, F_UNDEFINED);
 notimplemented:
  return _make_atom (_main, FLAG, F_NOTIMPLEMENTED);
}

static GSM __cast_down (_main, _type, val) MAIN*_main; WORD _type; GSM val; {
  switch (_type) {
    case INTEGER : break;
```

```
#   ifdef __LONG
    case LONGINT :
      if ((long)val <= MAXINT && (long)val >= MININT)_type = INTEGER;
      break;
#   endif
#   ifdef __REAL
    case REAL    :
#   define VAL (*((real*)val))
      if (floor (VAL) == VAL) {
#       ifdef __LONG
        if ((((long)VAL) <= MAXLONG) && (((long) VAL) >= MINLONG))
          return _cast_down (_main, LONGINT, ((long) VAL));
#       else
        if (((int) VAL <= MAXINT) && ((int) VAL >= MININT))
          return _make_atom (_main, INTEGER, ((int) VAL));
#       endif
      }
      break;
#   undef VAL
#   endif /* __REAL */
    case STRING  : break;
    default      : _assert_false (_main, "Not allowed type",
                                  _type = FLAG; val = (GSM) F_NULLOBJ);
  }
  return _make_atom (_main, _type, val);
}



static GSM _execute (_main, x, y, _op) MAIN*_main; GSM x, y; int _op; {
       if (TYP(x) > TYP(y)) y = _cast(_main, y, x);
  else if (TYP(x) < TYP(y)) x = _cast(_main, x, y);
  if (IsAFlag (y)) return y;
  return _operator[_op][TYP(x) -INTEGER](_main, x, y);
}



static DECLF _math[] = {
  {"+",      CT_PROCEDURE+CT_LIST,  (FUNC) gadd        },
  {"-",      CT_PROCEDURE+CT_LIST,  (FUNC) gsub        },
  {"*",      CT_PROCEDURE+CT_LIST,  (FUNC) gmul        },
  {"/",      CT_PROCEDURE+CT_LIST,  (FUNC) gdiv        },

  {"+e",     CT_PROCEDURE+CT_LIST,  (FUNC) add_exact   },
  {"-e",     CT_PROCEDURE+CT_LIST,  (FUNC) sub_exact   },
  {"*e",     CT_PROCEDURE+CT_LIST,  (FUNC) mul_exact   },
  {"/e",     CT_PROCEDURE+CT_LIST,  (FUNC) div_exact   },
# ifdef __REAL
  {"+i",     CT_PROCEDURE+CT_LIST,  (FUNC) add_inexact },
  {"-i",     CT_PROCEDURE+CT_LIST,  (FUNC) sub_inexact },
  {"*i",     CT_PROCEDURE+CT_LIST,  (FUNC) mul_inexact },
  {"/i",     CT_PROCEDURE+CT_LIST,  (FUNC) div_inexact },
# endif
  {0,0,0}
};


int _is_complex (complex) GSM complex; {
  return IsVector(complex) && GCOMPLEXCONTROL (complex) == (GSM)&_complex_control;
}

void _init_math (_main) MAIN*_main; {
  _load_keyword (_main, _math);
  SFLAG (& _complex_control, F_NULLOBJ);
}

GSM _make_complex (_main, r, i) MAIN*_main; GSM r, i; {
GSM v = _make_vector (_main, 3);

  GCOMPLEXRE(v)      = r;
  GCOMPLEXIM(v)      = i;
  GCOMPLEXCONTROL(v) = & _complex_control;
  return v;
}
```

Addition

```
/*
 M A T H A D D . C

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>



static GSM add_int     PROTO ((MAIN*, GSM, GSM));
static GSM add_long    PROTO ((MAIN*, GSM, GSM));
static GSM add_real    PROTO ((MAIN*, GSM, GSM));
static GSM add_complex PROTO ((MAIN*, GSM, GSM));
static GSM add_string  PROTO ((MAIN*, GSM, GSM));
static GSM gadd        PROTO ((MAIN*, GSM));
static GSM add_exact   PROTO ((MAIN*, GSM));
static GSM add_inexact PROTO ((MAIN*, GSM));



/*********/
/* A D D */
/*********/
static GSM add_int (_main, x, y) MAIN *_main; GSM x, y; {
register int res = GINT(x) + GINT(y);

  if ((GINT(x) <0 && GINT(y) <0)) {
    if (res >= 0) goto upper_type;
    else goto normal_add;
  }
  else if (GINT(x) >0 && GINT(y) >0) {
    if (res <= 0) goto upper_type;
    else goto normal_add;
  }

  normal_add:
    return _make_atom (_main, INTEGER, res);

# ifdef __LONG
  upper_type:
    return _make_atom (_main, LONGINT, (long) GINT(x) + (long)GINT(y));
# else
#   ifdef __REAL
    upper_type: {
      real r = (real) GINT(x) + (real) GINT(y);
        return _make_atom (_main, REAL, &r);
      }
#   else
      return _make_atom(_main, FLAG, F_OVERFLOW);
```

```
#   endif

#endif
}


static GSM add_long (_main, x, y) MAIN *_main; GSM x, y; {
#ifdef __LONG
register long res = GLONGINT(x) + GLONGINT (y);

  if ((GLONGINT(x) <0 && GLONGINT(y) <0)) {
    if (res >= 0) goto upper_type;
    else goto normal_add;
  }
  else if (GLONGINT(x) >0 && GLONGINT(y) >0) {
    if (res <= 0) goto upper_type;
    else goto normal_add;
  }

  normal_add:
    return _cast_down (_main, LONGINT, res);

# ifdef __REAL
  upper_type: {
    real r = (real) GLONGINT(x) + (real) GLONGINT(y);
    return _make_atom (_main, REAL, &r);
  }
# else
  upper_type:
    return _make_atom (_main, FLAG, F_OVERFLOW);
# endif
#else
# ifdef __DEBUG
   _assert_false (_main, "add long not allowed", _end_gsm(_main));
# else
   return _make_atom(_main, FLAG, F_NULLOBJ);
# endif
#endif /* __LONG */
}


static GSM add_real (_main, x, y) MAIN *_main; GSM x, y; {
#ifdef __REAL
real r;
  r = *GREAL(x) + *GREAL(y);
  return _cast_down (_main, REAL, & r);
#else
# ifdef __DEBUG
   _assert_false (_main, "add real not allowed", _end_gsm(_main));
# else
   return _make_atom(_main, FLAG, F_NULLOBJ);
# endif
#endif /* __REAL */
}

static GSM add_complex (_main, x, y) MAIN *_main; GSM x, y; {
GSM re = PUSH(_execute (_main, GCOMPLEXRE(x), GCOMPLEXIM(x), ADD));
GSM im = PUSH(_execute (_main, GCOMPLEXRE(y), GCOMPLEXIM(y), ADD));
GSM cx = _make_complex (_main, re, im);

  POPN(2);
  return cx;
}

static GSM add_string (_main, x, y) MAIN *_main; GSM x, y; {
  if (_main->option.extended_syntaxe) {
  GSM   new = NEWCELL(_main);
  char *buf = (char*) _malloc_heap (_main, strlen ((char*)GSTRING(x)) + strlen
((char*)GSTRING(y)) +1);
    strcpy (buf, (char*)GSTRING(x));
    strcat (buf, (char*)GSTRING(y));
    SSTRING(new, buf);
    return new;
  }
  else return _make_atom (_main, FLAG, F_NULLOBJ);
}

static GSM gadd (_main, list) MAIN*_main; GSM list; {
GSM ret = _make_atom (_main, INTEGER, 0);

  while (! IsAtom (list)) {

    if (! IsAtom (CAR(list))) goto bad_operande;
    ret = _execute (_main, ret, CAR(list), ADD);
    if (IsAFlag(ret)) goto bad_operande;
    list = CDR(list);
    _assert (_main, list, ;);
  }
  if (! IsFlag (list, F_NULLOBJ)) goto bad_operande;
  return ret;

 bad_operande:
  _error (_main, ERR_BAD_OPERAND, 0, GOTOP);
  return _make_atom (_main, FLAG, F_NULLOBJ);
}

static GSM add_exact (_main, list) MAIN*_main; GSM list; {
long res = 0;

  while (! IsAtom (list)) {
#   ifdef __LONG
    res = res + (IsInteger (CAR(list)) ? GINT (CAR(list)) : GLONGINT (CAR(list)));
#   else
    res = res + GINT (CAR(list));
#   endif
    list = CDR(list);
    _assert (_main, list, ;);
  }
# ifdef __LONG
  return _cast_down (_main, LONGINT, res);
# else
  return _make_atom (_main, INTEGER, res);
# endif
}
#ifdef __REAL
static GSM add_inexact (_main, list) MAIN*_main; GSM list; {
real res = 0;

  while (! IsAtom (list)) {
    res = res + *GREAL (CAR(list));
    list = CDR(list);
    _assert (_main, list, ;);
  }
  return _cast_down (_main, REAL, &res);
}
#endif
```

Soustraction

```
/*
 M A T H S U B . C

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/


#include <gsm.h>

static GSM sub_int     PROTO ((MAIN*, GSM, GSM));
static GSM sub_long    PROTO ((MAIN*, GSM, GSM));
static GSM sub_real    PROTO ((MAIN*, GSM, GSM));
static GSM sub_complex PROTO ((MAIN*, GSM, GSM));
static GSM sub_string  PROTO ((MAIN*, GSM, GSM));
static GSM gsub        PROTO ((MAIN*, GSM));
static GSM sub_exact   PROTO ((MAIN*, GSM));
static GSM sub_inexact PROTO ((MAIN*, GSM));



static GSM sub_int (_main, x, y) MAIN *_main; GSM x, y; {
register int res = GINT(x) - GINT(y);

  if ((GINT(x) <0 && GINT(y) >0)) {
    if (res >= 0) goto upper_type;
    else goto normal_sub;
  }
  else if (GINT(x) >0 && GINT(y) <0) {
    if (res <= 0) goto upper_type;
    else goto normal_sub;
  }

  normal_sub:
    return _make_atom (_main, INTEGER, res);

# ifdef __LONG
  upper_type:
    return _make_atom (_main, LONGINT, (long) GINT(x) - (long)GINT(y));
# else
#   ifdef __REAL
    upper_type: {
      real r = (real) GINT(x) - (real) GINT(y);
        return _make_atom (_main, REAL, &r);
      }
#   else
      return _make_atom(_main, FLAG, F_OVERFLOW);
#   endif
#endif
}

static GSM sub_long (_main, x, y) MAIN *_main; GSM x, y; {
#ifdef __LONG
register long res = GLONGINT(x) - GLONGINT(y);

  if ((GLONGINT(x) <0 && GLONGINT(y) >0)) {
    if (res >= 0) goto upper_type;
    else goto normal_sub;
  }
  else if (GLONGINT(x) >0 && GLONGINT(y) <0) {
    if (res <= 0) goto upper_type;
    else goto normal_sub;
  }

  normal_sub:
    return _cast_down (_main, LONGINT, res);

# ifdef __REAL
  upper_type: {
    real r = (real) GLONGINT(x) - (real) GLONGINT(y);
    return _make_atom (_main, REAL, &r);
  }
# else
  upper_type:
    return _make_atom (_main, FLAG, F_OVERFLOW);
# endif
#else
# ifdef __DEBUG
   _assert_false (_main, "sub long not allowed", _end_gsm(_main));
# else
   return _make_atom(_main, FLAG, F_NULLOBJ);
# endif
#endif /* __LONG */
}


static GSM sub_real (_main, x, y) MAIN *_main; GSM x, y; {
#ifdef __REAL
real r = *GREAL(x) - *GREAL(y);
  return _cast_down (_main, REAL, & r);
#else
# ifdef __DEBUG
   _assert_false (_main, "sub real not allowed", _end_gsm(_main));
# else
   return _make_atom(_main, FLAG, F_NULLOBJ);
# endif
#endif /* __REAL */
}

static GSM sub_complex (_main,x, y) MAIN *_main; GSM x, y; {
GSM re = PUSH(_execute (_main, GCOMPLEXRE(x), GCOMPLEXIM(x), SUB));
GSM im = PUSH(_execute (_main, GCOMPLEXRE(y), GCOMPLEXIM(y), SUB));
GSM cx = _make_complex (_main, re, im);

  POPN(2);
  return cx;
}

#ifdef __Borlandc
# pragma argsused
#endif
static GSM sub_string (_main, x, y) MAIN *_main; GSM x, y; {
  return _make_atom (_main, FLAG, F_NULLOBJ);
}


static GSM gsub (_main, list) MAIN*_main; GSM list; {
GSM ret;

  if (IsAtom(list)) {
    if (IsFlag (list, F_NULLOBJ))
      return _make_atom (_main, INTEGER, 0);
    else goto bad_operande;
  }
  ret  = CAR (list);
  list = CDR (list);
  if (IsAtom(list)) {
    if (IsFlag (list, F_NULLOBJ))
      return _execute (_main, _make_atom (_main, INTEGER, 0), ret, SUB);
    else goto bad_operande;
  }

  while (! IsAtom (list)) {

    if (! IsAtom (CAR(list))) goto bad_operande;
    ret = _execute (_main, ret, CAR(list), SUB);
    if (IsAFlag(ret)) goto bad_operande;
    list = CDR(list);
    _assert (_main, list, ;);
  }
  if (! IsFlag (list, F_NULLOBJ)) goto bad_operande;
  return ret;
```

```
 bad_operande:
  _error (_main, ERR_BAD_OPERAND, 0, GOTOP);
  return _make_atom (_main, FLAG, F_NULLOBJ);
}


static GSM sub_exact (_main, list) MAIN*_main; GSM list; {
long res;

  if (IsAtom(list)) return _make_atom (_main, INTEGER, 0);
# ifdef __LONG
  res  = IsInteger (CAR (list)) ? GINT (CAR(list)) : GLONGINT (CAR(list));
# else
  res  = GINT (CAR (list));
# endif

  list = CDR (list);
  if (IsAtom(list)) return _make_atom (_main, TYP(list), -res);

  while (! IsAtom (list)) {
#   ifdef __LONG
    res = res - (IsInteger (CAR(list)) ? GINT (CAR(list)) : GLONGINT (CAR(list)));
#   else
    res = res - GINT (CAR(list));
#   endif
    list = CDR(list);
    _assert (_main, list, ;);
  }
# ifdef __LONG
  return _cast_down (_main, LONGINT, res);
# else
  return _make_atom (_main, INTEGER, res);
# endif
}
#ifdef __REAL
static GSM sub_inexact (_main, list) MAIN*_main; GSM list; {
real tmp, res;

  if (IsAtom(list)) return _make_atom (_main, INTEGER, 0);
  res  = * GREAL(CAR(list));
  list = CDR (list);
  if (IsAtom(list)) {
    if (! res) _error (_main, ERR_DIVISION_BY_ZERO, 0, GOTOP);
    res = 1 / res;
    return _make_atom (_main, REAL, &res);
  }
  while (! IsAtom (list)) {
    tmp = * GREAL (CAR (list));
    if (! tmp) _error (_main, ERR_DIVISION_BY_ZERO, 0, GOTOP);
    res = res / tmp;
    list = CDR(list);
    _assert (_main, list, ;);
  }
  return _make_atom (_main, REAL, &res);
}
#endif
```

.

Multiplication

```
/*
 M A T H M U L . C

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>

static GSM mul_int     PROTO ((MAIN*, GSM, GSM));
static GSM mul_long    PROTO ((MAIN*, GSM, GSM));
static GSM mul_real    PROTO ((MAIN*, GSM, GSM));
static GSM mul_complex PROTO ((MAIN*, GSM, GSM));
static GSM mul_string  PROTO ((MAIN*, GSM, GSM));
static GSM gmul        PROTO ((MAIN*, GSM));
static GSM mul_exact   PROTO ((MAIN*, GSM));
static GSM mul_inexact PROTO ((MAIN*, GSM));



static GSM mul_int (_main, x, y) MAIN *_main; GSM x, y; {
register int res = GINT(x) * GINT(y);

  if ((GINT(x) <0 && GINT(y) <0)) {
    if (res >= GINT(x) || res >= GINT(x)) goto upper_type;
    else goto normal_mul;
  }
  else if (GINT(x) >0 && GINT(y) >0) {
    if (res <= GINT(x) || res <= GINT(x)) goto upper_type;
    else goto normal_mul;
  }

  normal_mul:
    return _make_atom (_main, INTEGER, res);

# ifdef __LONG
  upper_type:
    return _make_atom (_main, LONGINT, (long) GINT(x) * (long)GINT(y));
# else
#   ifdef __REAL
    upper_type: {
      real r = (real) GINT(x) *(real) GINT(y);
      return _make_atom (_main, REAL, &r);
    }
#   else
      return _make_atom(_main, FLAG, F_OVERFLOW);
#   endif
#endif
}


static GSM mul_long (_main, x, y) MAIN *_main; GSM x, y; {
#ifdef __LONG
register long res = GLONGINT(x) * GLONGINT(y);

  if ((GLONGINT(x) <0 && GLONGINT(y) <0)) {
    if (res >= GLONGINT(x) || res >= GLONGINT(x)) goto upper_type;
    else goto normal_mul;
  }
  else if (GLONGINT(x) >0 && GLONGINT(y) >0) {
    if (res <= GLONGINT(x) || res <= GLONGINT(x)) goto upper_type;
    else goto normal_mul;
  }

  normal_mul:
    return _cast_down (_main, LONGINT, res);

# ifdef __REAL
  upper_type: {
    real r = (real) GLONGINT(x) *(real) GLONGINT(y);
    return _make_atom (_main, REAL, &r);
  }
# else
  upper_type:
    return _make_atom (_main, FLAG, F_OVERFLOW);
# endif
#else
# ifdef __DEBUG
   _assert_false (_main, "mul long not allowed", _end_gsm(_main));
# else
   return _make_atom(_main, FLAG, F_NULLOBJ);
# endif
#endif /* __LONG */
}


static GSM mul_real (_main, x, y) MAIN *_main; GSM x, y; {
#ifdef __REAL
real r = *GREAL(x) * *GREAL(y);
  return _cast_down (_main, REAL, & r);
#else
# ifdef __DEBUG
   _assert_false (_main, "mul real not allowed", _end_gsm(_main));
# else
   return _make_atom(_main, FLAG, F_NULLOBJ);
# endif
#endif /* __REAL */
}

/* (a+jb)(x+jy)=ax-by+j(ay+bx) */
static GSM mul_complex (_main,x, y) MAIN *_main; GSM x, y; {
GSM ax = PUSH(_execute (_main, GCOMPLEXRE(x), GCOMPLEXRE(y), MUL));
GSM by = PUSH(_execute (_main, GCOMPLEXIM(x), GCOMPLEXIM(y), MUL));
GSM ay = PUSH(_execute (_main, GCOMPLEXRE(x), GCOMPLEXIM(x), MUL));
GSM bx = PUSH(_execute (_main, GCOMPLEXIM(x), GCOMPLEXRE(x), MUL));
GSM re = PUSH(_execute (_main, ax, by, SUB));
GSM im = PUSH(_execute (_main, ay, bx, ADD));
GSM cx = _make_complex (_main, re, im);

  POPN(6);
  return cx;
}

#ifdef __Borlandc
#  pragma argsused
#endif
static GSM mul_string (_main, x, y) MAIN *_main; GSM x, y; {
  return _make_atom (_main, FLAG, F_NULLOBJ);
}

static GSM gmul (_main, list) MAIN*_main; GSM list; {
GSM ret = _make_atom (_main, INTEGER, 1);

  while (! IsAtom (list)) {

    if (! IsAtom (CAR(list))) goto bad_operande;
    ret = _execute (_main, ret, CAR(list), MUL);
    if (IsAFlag(ret)) goto bad_operande;
    list = CDR(list);
    _assert (_main, list, ;);
  }
  if (! IsFlag (list, F_NULLOBJ)) goto bad_operande;
  return ret;

 bad_operande:
  _error (_main, ERR_BAD_OPERAND, 0, GOTOP);
  return _make_atom (_main, FLAG, F_NULLOBJ);
```

```
}


static GSM mul_exact (_main, list) MAIN*_main; GSM list; {
long res = 1;

  while (! IsAtom (list)) {
#   ifdef __LONG
    res = res * (IsInteger (CAR(list)) ? GINT (CAR(list)) : GLONGINT (CAR(list)));
#   else
    res = res * GINT (CAR(list));
#   endif
    list = CDR(list);
    _assert (_main, list, ;);
  }
# ifdef __LONG
  return _cast_down (_main, LONGINT, res);
# else
  return _make_atom (_main, INTEGER, res);
# endif
}
#ifdef __REAL
static GSM mul_inexact (_main, list) MAIN*_main; GSM list; {
real res = 0;

  while (! IsAtom (list)) {
    res = res * *GREAL (CAR(list));
    list = CDR(list);
    _assert (_main, list, ;);
  }
  return _cast_down (_main, REAL, &res);
}
#endif
```

.

Division

```
/*
 M A T H D I V . C

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>

static GSM div_int     PROTO ((MAIN*, GSM, GSM));
static GSM div_long    PROTO ((MAIN*, GSM, GSM));
static GSM div_real    PROTO ((MAIN*, GSM, GSM));
static GSM div_complex PROTO ((MAIN*, GSM, GSM));
static GSM div_string  PROTO ((MAIN*, GSM, GSM));
static GSM gdiv        PROTO ((MAIN*, GSM));
static GSM div_exact   PROTO ((MAIN*, GSM));
static GSM div_inexact PROTO ((MAIN*, GSM));
static GSM div_int (_main, x, y) MAIN *_main; GSM x, y; {
  if (! GINT (y))_error (_main, ERR_DIVISION_BY_ZERO, 0, GOTOP);

# ifdef __REAL
  if (! (GINT(x) % GINT(y)))
# endif
    return _make_atom (_main, INTEGER, GINT(x) /GINT(y));
# ifdef __REAL
  else {
  real r = (real) GINT(x) / (real) GINT(y);
    return _make_atom (_main, REAL, &r);
  }
#endif
}


static GSM div_long (_main, x, y) MAIN *_main; GSM x, y; {
# ifdef __LONG
    if (! GLONGINT (y))_error (_main, ERR_DIVISION_BY_ZERO, 0, GOTOP);

#   ifdef __REAL
    if (! (GLONGINT(x) % GLONGINT(y)))
#   endif
      return _cast_down (_main, LONGINT, GLONGINT(x) /GLONGINT(y));
#   ifdef __REAL
    else {
    real r = (real) GLONGINT(x) / (real) GLONGINT(y);
      return _make_atom (_main, REAL, &r);
    }
#  endif
# else
# ifdef __DEBUG
   _assert_false (_main, "div long not allowed", _end_gsm(_main));
# else
   return _make_atom(_main, FLAG, F_NULLOBJ);
# endif
#endif /* __LONG */
}

static GSM div_real (_main, x, y) MAIN *_main; GSM x, y; {
#ifdef __REAL
real r;
  if (! *GREAL(y)) _error (_main, ERR_DIVISION_BY_ZERO, 0, GOTOP);
  r = *GREAL(x) / *GREAL(y);
  return _cast_down (_main, REAL, & r);
#else
# ifdef __DEBUG
   _assert_false (_main, "div real not allowed", _end_gsm(_main));
# else
   return _make_atom(_main, FLAG, F_NULLOBJ);
# endif
#endif /* __REAL */
}

/* (a+jb)/(x+jy) = (ax+by+j(bx-ay)/(xx-yy) */
static GSM div_complex (_main,x, y) MAIN *_main; GSM x, y; {
GSM ax = PUSH(_execute (_main, GCOMPLEXRE(x), GCOMPLEXRE(y), MUL));
GSM by = PUSH(_execute (_main, GCOMPLEXIM(x), GCOMPLEXIM(y), MUL));
GSM ay = PUSH(_execute (_main, GCOMPLEXRE(x), GCOMPLEXIM(x), MUL));
GSM bx = PUSH(_execute (_main, GCOMPLEXIM(x), GCOMPLEXRE(x), MUL));
GSM xx = PUSH(_execute (_main, GCOMPLEXRE(y), GCOMPLEXRE(y), MUL));
GSM yy = PUSH(_execute (_main, GCOMPLEXIM(y), GCOMPLEXIM(y), MUL));
GSM qu = PUSH(_execute (_main, xx, yy, ADD));
GSM re = PUSH(_execute (_main, ax, by, ADD));
GSM im = PUSH(_execute (_main, ay, bx, SUB));
GSM cx;

  re = PUSH(_execute (_main, re, qu, DIV));
  im = PUSH(_execute (_main, im, qu, DIV));
  cx = _make_complex (_main, re, im);

  POPN(11);
  return cx;
}

#ifdef __Borlandc
#  pragma argsused
#endif
static GSM div_string (_main, x, y) MAIN *_main; GSM x, y; {
  return _make_atom (_main, FLAG, F_NULLOBJ);
}

static GSM gdiv (_main, list) MAIN*_main; GSM list; {
GSM ret;

  if (IsAtom(list)) {
    if (IsFlag (list, F_NULLOBJ))
      return _make_atom (_main, INTEGER, 0);
    else goto bad_operande;
  }
  ret  = CAR (list);
  list = CDR (list);
  if (IsAtom(list)) {
    if (IsFlag (list, F_NULLOBJ)) {
      if (_is_zero (ret)) _error (_main, ERR_DIVISION_BY_ZERO, 0, GOTOP);
      return _execute (_main, _make_atom (_main, INTEGER, 1), ret, DIV);
    }
    else goto bad_operande;
  }
  while (! IsAtom (list)) {

    if (! IsAtom (CAR(list))) goto bad_operande;
    ret = _execute (_main, ret, CAR(list), DIV);
    if (IsAFlag(ret)) goto bad_operande;
    list = CDR(list);
    _assert (_main, list, ;);
  }
  if (! IsFlag (list, F_NULLOBJ)) goto bad_operande;
  return ret;

 bad_operande:
  _error (_main, ERR_BAD_OPERAND, 0, GOTOP);
  return _make_atom (_main, FLAG, F_NULLOBJ);
}


static GSM div_exact (_main, list) MAIN*_main; GSM list; {
#ifdef __REAL
real res = 1;
#else
long res = 1;
#endif
long tmp;

  if (IsAtom(list)) return _make_atom (_main, INTEGER, 0);
# ifdef __LONG
  res  = IsInteger (CAR (list)) ? GINT (CAR(list)) : GLONGINT (CAR(list));
# else
  res  = GINT (CAR (list));
# endif

  list = CDR (list);
  if (IsAtom(list)) {
    if (! res) goto divide_by_zero;
    res = 1 / res;
    goto end;
  }
  while (! IsAtom (list)) {
#   ifdef __LONG
    tmp = IsInteger (CAR(list)) ? GINT (CAR(list)) : GLONGINT (CAR(list));
    if (! tmp) goto divide_by_zero;
#    ifdef __REAL
    res = res / (real) tmp;
#    else
```

```
    res = res / (long) tmp;
#    endif
#   else
    tmp = GINT(CAR(list));
    if (! tmp) goto divide_by_zero;
#    ifdef __REAL
    res = res / (real) tmp;
#    else
    res = res / (long) tmp;
#    endif
#   endif
    list = CDR(list);
    _assert (_main, list, ;);
  }
 end:
#ifdef __REAL
  return _cast_down (_main, REAL, &res);
# else
#  ifdef __LONG
  return _cast_down (_main, LONGINT, res);
#  else
  return _make_atom (_main, INTEGER, res);
#  endif
# endif
 divide_by_zero:
  _error (_main, ERR_DIVISION_BY_ZERO, 0, GOTOP);
  return 0; /* compiler warning */
}
#ifdef __REAL
static GSM div_inexact (_main, list) MAIN*_main; GSM list; {
real res = 0;

  while (! IsAtom (list)) {
    res = res + *GREAL (CAR(list));
    list = CDR(list);
    _assert (_main, list, ;);
  }
  return _cast_down (_main, REAL, &res);
}
#endif
```

.

Noyeau

Initialisation et fin

```
/*
 I N I T . C


 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/


#include <gsm.h>
#ifdef __Borlandc
# pragma warn -cln
#endif


#define IsStd(f) ((f)==stdin||(f)==stdout||(f)==stderr)

static DWORD _get_heap_size   PROTO ((PSTR *_argv));
static void  _preload_dynamic PROTO ((PSTR *_argv));
static void  _set_option      PROTO ((MAIN * _main, int _argc, PSTR *_argv));

static DWORD _get_heap_size (_argv) PSTR *_argv; {
PSTR hs = 0;
int  i  = 0;

  while (_argv[i]) {
    if (_argv[i][0] == '-' && (tolower(_argv[i][1]) == CLO_HEAP_SIZE)) {
      hs = _argv[i] +2;
      break;
    }
    i++;
  }
  if (*hs) return atol ((char*)hs);
  else     return 0;
}


#ifdef __DYNAMIC
static void _preload_dynamic (_argv) PSTR*_argv; {
PSTR file = 0;
int  i    = 0;

  while (_argv[i]) {
    if (_argv[i][0] == '-' && (tolower(_argv[i][1]) == CLO_DYNAMIC_FILE)) {
      file = _argv[i] +2;
      break;
    }
    i++;
  }
 _load_dynamic (_argv, file);
}
#endif

static void _set_option (_main, _argc, _argv)
MAIN * _main; int _argc; PSTR *_argv; {
#define ARG _argv[arg_count]
int   arg_count;
char  buffer[100];
char  stdin_name[] = "stdin";

  _main->in   = stdin;
  _main->out  = stdout;
  _main->err  = stderr;
  _main->file = _malloc_heap (_main, sizeof (stdin_name) +1);
  strcpy ((char*)_main->file, stdin_name);

  for (arg_count =1; arg_count <_argc; arg_count++)
    if (ARG[0] == '-' || ARG[0] == '\n') {
      switch (ARG[1]) {
```

```
#       define CASE(c) case c: case (c-'a'+'A')
        help     :
        CASE(CLO_HELP):
          _help();
          newline (_main);
          _display_bye (_main);
          exit (0);
          break;
        CASE(CLO_GARBAGE_SIZE):
          _main->garbage_size = atoi ((char*)ARG +2);
          break;
        CASE(CLO_SYMBOL_TABLE_SIZE):
          _main->hash_size = atoi ((char*)ARG +2);
          break;
        CASE(CLO_PROMPT):
          strcpy (buffer, (char*)ARG + 2);
          buffer[PROMPT_LENGTH] = 0;
          strcpy (_main->prompt, buffer);
          break;
        CASE(CLO_TEMP_SYMBOL_TABLE_SIZE):
          _main->hash_temp_size = atoi ((char*)ARG +2);
          break;
        CASE(CLO_DYNAMIC_FILE):
        CASE(CLO_HEAP_SIZE)   :break;
        default  : _error (_main, ERR_INVALID_OPTION, ARG, ERR);
                   goto help;
      }
      ARG[0] = '*'; /* for keep option during (restart) */
#     undef CASE
    }
#undef ARG
}

void _close_gsm (_main) MAIN*_main; {
  if (! IsStd (_main->in )) fclose (_main->in );
  if (! IsStd (_main->out)) fclose (_main->out);
  if (! IsStd (_main->err)) fclose (_main->err);

  _free_heap  (_main, _main->file);
  _end_stack  (_main);
  _end_atom   (_main);
  _end_env    (_main);
  _end_hash   (_main);
  _end_keyword(_main);
  _end_math   (_main);
  _end_vector (_main);
  _end_garbage(_main);
  _end_signal (_main);
# ifdef __DYNAMIC
  _end_dynamic(_main);
# endif
  _end_heap   (_main);
}


void _end_gsm (_main) MAIN*_main; {
int errno = _main->errno;
  _display_bye (_main);
  _close_gsm (_main);
  exit (errno);
}


MAIN *_init_gsm (_argc, _argv)
int    _argc; PSTR *_argv; {
MAIN * _main;
DWORD heap_size;

# ifdef __DYNAMIC
 _preload_dynamic(_argv);
# endif
 heap_size = _get_heap_size (_argv);
 _main                          = _init_heap (heap_size ? heap_size :
DEFAULT_HEAP_SIZE);
 _main->line                    = 1;
 _main->error                   =
 _main->warning                 = 0;
 _main->identifier_length       = DEFAULT_IDENTIFIER_LENGTH;
 _main->option.redefine_symbol  = DEFAULT_REDEFINE_SYMBOL;
 _main->option.extended_syntaxe = DEFAULT_EXTENDED_SYNTAXE;
 _main->option.verbose_eval     = DEFAULT_VERBOSE_EVAL;
 _main->option.display_reserved = DEFAULT_DISPLAY_RESERVED;
  strncpy (_main->prompt, DEFAULT_PROMPT, PROMPT_LENGTH);

 _set_option (_main, _argc, _argv);

 _init_signal (_main);
 _init_garbage(_main, _main->garbage_size ? _main->garbage_size :
DEFAULT_GARBAGE_SIZE);
 _init_stack  (_main, _main->stack_size? _main->stack_size: DEFAULT_STACK_SIZE);
 _init_atom   (_main);
 _init_hash   (_main);
 _init_env    (_main, _main->hash_size     ?_main->hash_size     :DEFAULT_HASH_SIZE,
                      _main->hash_temp_size?_main-
>hash_temp_size:DEFAULT_HASH_TEMP_SIZE);
 _init_keyword(_main);
 _init_math   (_main);
 _init_vector (_main);
# ifdef __DYNAMIC
 _init_dynamic(_main);
# endif

# if defined(__Borlandc) && defined(__Msdos)
  {
  void _init_conio PROTO ((MAIN*_main));
    _init_conio (_main);
  }
# endif
  return _main;
}
```

Lancement

```
/*
 M A I N . C

 Scheme implementation.
   Copyright (C) 1993 Guilhem de Wailly.

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <gsm.h>



int main(_argc, _argv) int _argc; char **_argv; {
int     errno;
int     arg_count;
jmp_buf restart;
MAIN   *_main = 0;

  setjmp (restart);
  if (_main) _close_gsm(_main);

  _main = _init_gsm(_argc, (PSTR *)_argv);
  jmp_cpy (_main->goto_restart, restart);

  _display_hello (_main);
  _main->level = 1; /* no display the evaluation result for loaded file */
  _register_main (_main); /* signal interrupt allowed */
  if (_file_exists (_main, INIT_FILE)) _load (_main, INIT_FILE);
  for (arg_count =1; arg_count <_argc; arg_count++)
    if (*_argv[arg_count]!='*')_load (_main, _argv[arg_count]);
  _main->level = 0;
  _load (_main, (char*)0);  /* 0 assumed as stdin by _load */
  errno = _main->errno;
  _end_gsm(_main);
  return errno;
}
```

Bibliothèque SCHEME

Fichier d'initialisation

```
;
; GSM.S
;
; Scheme implementation.
;   Copyright (C) 1993 Guilhem de Wailly.
;
;This program is free software; you can redistribute it and/or modify
;it under the terms of the GNU General Public License as published by
;the Free Software Foundation; either version 1, or (at your option)
;any later version.
;
;This program is distributed in the hope that it will be useful,
;but WITHOUT ANY WARRANTY; without even the implied warranty of
;MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;GNU General Public License for more details.
;
;You should have received a copy of the GNU General Public License
;along with this program; if not, write to the Free Software
;Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
;
;The author can be reached at gdw@cob.unice.fr or
;Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.

(verbose 0)
(system-call "cls")
(extended-syntax 1)
(define no-error 0)
(define warning  1)
(define error    2)
(define gotop    3)
(define fatal    4)
(redefine-symbol no-error)
(define (try-load file)(if (file-exists? file) (load file)))
(define (garbage-size)(let ((gs garbage-size))(display (gs))(display "%\n")))
(define (garbage-collect)(let ((gc garbage-collect))(gc)(garbage-size)))
(define (char=? c1 c2) (if (char? c1) (if (char? c2) (eq? c1 c2))))

(display    "Version  : ")(display (version))
(display ".\nEthernet : gdw@cob.unice.fr.\n\n")
```

```
(display "(exit) to quit gsm.\n\n\n")
(redefine-symbol warning)
(prompt "$d-GSM>")
;(load "test.s")
(verbose 0)
```

Fichier de test

```
;
; T E S T . S
;
; Scheme implementation.
;   Copyright (C) 1993 Guilhem de Wailly.
;
;This program is free software; you can redistribute it and/or modify
;it under the terms of the GNU General Public License as published by
;the Free Software Foundation; either version 1, or (at your option)
;any later version.
;
;This program is distributed in the hope that it will be useful,
;but WITHOUT ANY WARRANTY; without even the implied warranty of
;MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;GNU General Public License for more details.
;
;You should have received a copy of the GNU General Public License
;along with this program; if not, write to the Free Software
;Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
;
;The author can be reached at gdw@cob.unice.fr or
;Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.

(verbose 1)
(extended-syntax 1)
(redefine-symbol no-error)

; quote car cdr
'(1 2 3)
(car '(1 2 3))
(cdr '(1 2 3))
(cons (car '(1 2 3)) (cdr '(1 2 3)))
(display "Step 1\n")

; lambda
(define foo 21)
(define (toto) 21)
((lambda (a b)(+ a b)) 3 4)
((lambda a a) 3)
((lambda a (a)) verbose)
(display "Step 2\n")

; let
(let ((a 3)(b 4)) (+ a b))
(let ((a 3)(b (+ a 5))) (+ a b))
(display "Step 3\n")

; letrec
(letrec ((a 2)(b 3))(+ a b))
(letrec ((a (lambda (a b) (+ a b))) (b (a 3 4))) (+ b 100))
(display "Step 4\n")

; begin
(begin (display 1)(display "string")(display 3))

; if
(if #f 1 2)
(if #f 1)
(if #t 1 2)
(if #t 1)
(display "Step 5\n")
```

```
; cond
(cond (#f 1)
      (#f 2)
      (#t 3))
(cond (#f 1)
      (#f 2)
      (#f 3)
      (else 4))
(cond (1 => verbose) (2 => top-level))
(verbose 0)
(display "Step 6\n")

; arithmetic
(display "arithmetic\n")
(+ 1 2 3 4)
(+)
(+ 1)
(- 1 2 3 4)
(-)
(- 1)
(display "Step 7\n")
```

Bibliothèques dynamiques sous DOS

Notre implémentation des bibliothèques n'est pour l'instant disponible que sous Dos et Windows. Un prochain travail consistera à les implémenter sous Unix. Le principe de fonctionnement et simple, l'implémentation sous Dos ne l'est pas du tout! Il repose sur les programmes résidents. Le serveur demande à ce qu'une bibliothèque soit chargée, puis il enrégistre les fonctions dont il a besoin. Pour cela, le client exporte une table des ses fonctions assorties de leur nom symbolique. Il faut noter que le serveur export des fonction vers le client. Celles-ci sont en fait des pointeurs sur fonction (voir `server.h`).
Toute la difficulté est reportée dans les fichiers `loadlib.c` (pour le serveur) et `gsmapi.c` (pour le client).

Le chargeur

```
/*
 L O A D L I B . C

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#define __INCLUDE_API <server.h>
#define __DECLARE_COMMON_DATA_TYPE
#include <loadlib.h>
```

```
#include <process.h>
#include <stdio.h>
#include <string.h>
#include <dos.h>

#undef  FP_OFF
#undef  FP_SEG
#undef  MK_FP
#define FP_OFF(p)  ((unsigned)(p))
#define FP_SEG(p)  ((unsigned)(((unsigned long)(p))>>16))
#define MK_FP(s,o) ((void huge*)((((unsigned long)(s))<<16)|(o)))

#ifdef __Borlandc
# pragma warn -pro /* Call to function with no prototype     */
# pragma warn -nod /* No declaration for function 'function' */
#endif

typedef union REGS REG;
typedef struct s_LIBINST {
  char        loaded;
  char        name[50];
  GSMAPI huge*api;
  unsigned    ds;
} LIBINST;


#define MAX_LIB 10
LIBINST *lib;
static   int         max_lib = 0;
static   FARPROC far *gsm_exp = 0;


/*
 T H U N C K

 This file implements a dos version of the Windows Thunck. A thunck is
 a dynamic created peace of code witch prepares the registers for a call
 to a function dynamicly loaded. The main problem is the sets the ds and
 es registers to the value they have when the library was loaded.
 Because of the libraries functions can be called with parameters, it is
 not allowed to uses the c stack.
 In addition, the thunck has to keeps the values of these registers.
 After the call, the thunck has to retaures the registers of the module
 witch call the function.
 We use the possibility to create dynamicly some code, and to reserves
 some places for data storage in the code.
 In the thunck, the call to the function is do with a jmp (the return
 address was previously pushed on the old return location address in
 the stack).
 Makes a thunck is to malloc memory and to "compiles" it. The compilation
 replaces the library ds and es thunck values by the library ones.
 Then, the compilation places the true values of the return address, and
 the jmp seg:off values.
 Then, the compilation replaces in the thucnk all references of the
 thunck data by the reals addresse (cs = seg(thunck)).

*/


char _thunck[] = {
/*  jmp label_1            0*/ 0xEB, 0x0A,
/*  old_ds DW 0x9090       2*/ 0x90, 0x90,
/*  old_cs DW 0x9090       4*/ 0x90, 0x90,
/*  old_ip DW 0x9090       6*/ 0x90, 0x90,
/*  sav_ax DW 0x9090       8*/ 0x90, 0x90,
/*  sav_bp DW 0x9090      10*/ 0x90, 0x90,
/* label_1:                 */
/*  mov  sav_bp, bp       12*/ 0x2E, 0x89, 0x2E, 0x08, 0x00,
/*  mov  bp, sp           17*/ 0x8B, 0xEC,
/*  mov  ax, [bp]         19*/ 0x8B, 0x46, 0x00,
/*  mov  old_ip, ax       22*/ 0x2E, 0xA3, 0x0C, 0x00,
/*  mov  ax:[bp+2]        26*/ 0x8B, 0x46, 0x02,
/*  mov  old_cs, ax         29*/ 0x2E, 0xA3, 0x0A, 0x00,
/*  mov  [bp], 0xBBCC       33*/ 0xC7, 0x46, 0x00, 0xCC, 0xBB,
/*  mov  [bp+2], 0xDDEE     38*/ 0xC7, 0x46, 0x02, 0xEE, 0xDD,
/*  push ds                 43*/ 0x1E,
/*  mov  ds, old_ds         44*/ 0x2E, 0x8E, 0x1E, 0x04, 0x00,
/*  pop  old_ds             49*/ 0x2E, 0x8F, 0x06, 0x04, 0x00,
/*  jmp  0x2233:0xFF11      54*/ 0xEA, 0x11, 0xFF, 0x33, 0x22,
/* label_end:                 */
/*  sub  sp, 4              59*/ 0x83, 0xEC, 0x04,
/*  push ds                 62*/ 0x1E,
/*  mov  ds, old_ds         63*/ 0x2E, 0x8E, 0x1E, 0x04, 0x00,
/*  pop  old_ds             68*/ 0x2E, 0x8F, 0x06, 0x04, 0x00,
/*  mov  sav_ax, ax         73*/ 0x2E, 0xA3, 0x0E, 0x00,
/*  mov  ax, old_ip         77*/ 0x2E, 0xA1, 0x0C, 0x00,
/*  mov  [bp], ax           81*/ 0x89, 0x46, 0x00,
/*  mov  ax, old_cs         84*/ 0x2E, 0xA1, 0x0A, 0x00,
/*  mov  [bp+2], ax         88*/ 0x89, 0x46, 0x02,
/*  mov  ax, sav_ax         91*/ 0x2E, 0xA1, 0x0E, 0x00,
/*  mov  bp, sav_bp         95*/ 0x2E, 0x8B, 0x2E, 0x08, 0x00,
/*  retf                   100*/ 0xCB
};

#define THUNCK_SIGN 0x0AEB
#define ODS_LOC  2
#define OCS_LOC  4
#define OIP_LOC  6
#define SAX_LOC  8
#define SBP_LOC 10
#define  IP_LOC 36
#define  CS_LOC 41
#define LJO_LOC 55
#define LJS_LOC 57
#define IPO_RET 59

static unsigned ods_off[] = {47, 52, 66, 71, 0};
static unsigned ocs_off[] = {31, 86, 0};
static unsigned oip_off[] = {24, 79, 0};
static unsigned sax_off[] = {75, 93, 0};
static unsigned sbp_off[] = {15, 98, 0};

static void set_word (char * thunck, int index, unsigned word) {
unsigned * p;

  thunck += index;
  p       = (unsigned *) thunck;
  *p      = word;
}

void compile_thunck (char * thunck, FARPROC proc, unsigned lib_ds) {
int i;
unsigned offset = FP_OFF(thunck);

  memcpy (thunck, _thunck, sizeof (_thunck));
  /* old_ds */
  set_word (thunck, ODS_LOC, lib_ds);
  /* ip & cs */
  set_word (thunck, IP_LOC, offset + IPO_RET);
  set_word (thunck, CS_LOC, FP_SEG(thunck));
  /* long jmp val */
  set_word (thunck, LJS_LOC, FP_SEG(proc));
  set_word (thunck, LJO_LOC, FP_OFF(proc));

  for (i=0; ods_off[i]; i++) set_word (thunck, ods_off[i], offset + ODS_LOC);
  for (i=0; ocs_off[i]; i++) set_word (thunck, ocs_off[i], offset + OCS_LOC);
  for (i=0; oip_off[i]; i++) set_word (thunck, oip_off[i], offset + OIP_LOC);
  for (i=0; sax_off[i]; i++) set_word (thunck, sax_off[i], offset + SAX_LOC);
  for (i=0; sbp_off[i]; i++) set_word (thunck, sbp_off[i], offset + SBP_LOC);
}



FARPROC _make_proc_instance (FARPROC proc, unsigned lib_ds) {
```

```
char *thunck = malloc(sizeof(_thunck));

  compile_thunck (thunck, proc, lib_ds);
  return (FARPROC) thunck;
}

FARPROC make_proc_instance (HLIB lib_handle, FARPROC proc) {
  return _make_proc_instance (proc, lib[lib_handle].ds);
}

void free_proc_instance (FARPROC proc) {
  if (proc && (*((unsigned*) proc) == THUNCK_SIGN)) {
    free (proc);
    *((unsigned*)proc) = 0xCB; /* ret far */
  }
}







/*
 L O A D L I B
*/

int add_proc_param (PSTR buffer, int head, void *param, int size) {
unsigned *_param  = (unsigned*)param;
unsigned *_buffer = ((unsigned*)buffer) + head;

  size = (size + 1) / 2;
  while (size--) {
    *_buffer++ = *_param++;
    head++;
  }
  return head;
}

void call_proc (FARPROC proc, PSTR buffer, int head) {
unsigned *_buffer = ((unsigned*) buffer) +head;
unsigned _ax, _dx, pushed = 0;

  if (! proc) return;
  while (head--) {
  int tmp = *--buffer;
    asm  mov ax, tmp
    asm  push ax
    pushed++;
  }
  #undef far
  proc();
  asm  mov _ax, ax
  asm  mov _dx, dx
  while (pushed--)
    asm  pop ax

  asm mov ax, _ax
  asm mov dx, _dx
}
void end_loadlib(void) {
REG  reg;
int      i;
FARPROC *p;

  for (i=0; i <max_lib; i++) if (lib[i].loaded) free_library (i);
  for (p = gsm_exp; gsm_exp && *p; p++) free_proc_instance (*p);
  if (_dos_getvect (API)) {
    reg.h.ah = 'G';
    reg.h.al = 'S';
    reg.h.bh = 'M';
    reg.h.bl = '1';
    reg.x.cx = API_END;
    int86 (API, & reg, & reg);
  }
}
void free_library (HLIB lib_handle) {
GSMAPI huge*p;

  if (lib_handle >= max_lib || !lib[lib_handle].loaded) return;
  --lib[lib_handle].loaded;
  if (lib[lib_handle].loaded) return;
  p = lib[lib_handle].api;
  while (p && p->name && *p->name) {
    free_proc_instance (p->proc);
    p++;
  }
}
int init_loadlib (PSTR *argv, PSTR file, GSMEXPORT huge*ge) {
int  i;
char buffer[250];
FILE    *load_lib = fopen ((char*)file, "rt");
FARPROC *p_export = (FARPROC *) ge;

  if (! load_lib) return ERR_LIBRARY_LOADER_NOT_FOUND;
  if (! _dos_getvect (API)) {
    while (! feof (load_lib)) {
      fgets (buffer, sizeof (buffer), load_lib);
printf ("sur la ligne de commande %s\n", buffer);
      if (*buffer) {
      char *p_export;
        buffer[strlen (buffer) -1] = 0;
        p_export = strchr (buffer, ' ');
        if (p_export) *p_export++ = 0;
        if (spawnlp (P_WAIT, buffer, buffer, (char huge*) "GSM1", p_export, 0) == -1)
          return ERR_UNABLE_TO_LOAD;
        buffer[0] = 0;
      }
    }
    fclose (load_lib);
/*  execve (argv[0], argv, 0); */
    i       = 0;
    *buffer = 0;
    while (argv[i]) {
      strcat ((char*)buffer, (char*)argv[i]);
      strcat ((char*)buffer, " ");
      i++;
    }
printf ("je vais executer %s\n", buffer);
    system (buffer);
    exit(0);
    return 1;
  }
  else {
    while (! feof (load_lib)) {
      fgets (buffer, sizeof (buffer), load_lib);
      if (*buffer) max_lib++;
      *buffer = 0;
    }
    fclose (load_lib);
    gsm_exp = (FARPROC *) ge;
    lib     = calloc (max_lib, sizeof (LIBINST));
    if (! lib) return ERR_NOT_ENOUGHT_MEMORY;
    while (*p_export) {
    int tmp;
      asm mov tmp, ds
      *p_export = _make_proc_instance (*p_export, tmp);
      p_export++;
    }
  }
  return 1;
}
HLIB load_library (PSTR lib_name) {
HLIB lib_handle;
```

```
REG  reg;

  for (lib_handle = 0; lib_handle < max_lib; lib_handle++) {
    if (   lib[lib_handle].loaded
        && ! stricmp (lib[lib_handle].name, (char*)lib_name)) {
      lib[lib_handle].loaded++;
      return lib_handle;
    }
  }
  for (lib_handle = 0; lib_handle < max_lib; lib_handle++) {
    if (! lib[lib_handle].loaded) break;
  }
  if (lib_handle == max_lib)
    return ERR_TOO_MANY_LIBRARY;

  reg.h.ah = 'G';
  reg.h.al = 'S';
  reg.h.bh = 'M';
  reg.h.bl = '1';
  reg.x.cx = API_ISLOAD;
  reg.x.si = FP_SEG(lib_name);
  reg.x.di = FP_OFF(lib_name);

  int86 (API, & reg, &reg);
  if (reg.x.ax) {

    strcpy (lib[lib_handle].name, (char*)lib_name);
    lib[lib_handle].loaded = 1;

    reg.h.ah = 'G';
    reg.h.al = 'S';
    reg.h.bh = 'M';
    reg.h.bl = '1';
    reg.x.cx = API_INIT;
    reg.x.dx = lib_handle;
    reg.x.si = FP_SEG (gsm_exp);
    reg.x.di = FP_OFF (gsm_exp);

    int86 (API, & reg, & reg);
    lib[lib_handle].api = (GSMAPI huge*)MK_FP (reg.x.si, reg.x.di);
    lib[lib_handle].ds = reg.x.ax;
  }
  else return ERR_UNABLE_TO_LOAD;
  return lib_handle;
}
PSTR get_lib_from_proc (FARPROC proc) {
int i;
  for (i = 0; i <max_lib; i++) {
  GSMAPI huge* p = lib[i].api;
    while (p && p->name && *p->name) {
      if (proc == p->proc) return lib[i].name;
      p++;
    }
  }
  return 0;
}
FARPROC get_proc_address (HLIB lib_handle, PSTR func_name) {
GSMAPI huge*p;

  if (! lib[lib_handle].loaded) return 0;
  p = lib[lib_handle].api;
  while (p && p->name && *p->name) {
    if (! stricmp ((char*)p->name, (char*)func_name)) {
      /* 0xEB0C if the code for jmp 0x12 witch is the first instruction
         of a thunck */
      if (*((unsigned*)p->proc) != THUNCK_SIGN) {
        p->proc = _make_proc_instance (p->proc, lib[lib_handle].ds);
      }
      return p->proc;
    }
    p++;
  }
  return 0;
}
```

L'interface

```
/*
 G S M A P I . C

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <dos.h>
#include <stdio.h>
#include <string.h>
#define __INCLUDE_API <server.h>
#define __DECLARE_COMMON_DATA_TYPE
#include <loadlib.h>



#ifdef __Quickc
# define asm       _asm
# define interrupt _interrupt
# define enable    _enable
# define disable   _disable
# define peek      _peek
void far * make_far_pointer (unsigned seg, unsigned off) {
  return (void far *) ((((unsigned long)(seg))<<16)+off);
}
# define MK_FP(s,o) make_far_pointer(s,o)
#endif
extern unsigned int  _psp;
static void (interrupt far * old_api)();
static int  lib_psp;
static int  IsInit = 0;


void Sound(unsigned frequency) {
  asm {
    mov     bx,  frequency
    mov     ax,  34DDh
    mov     dx,  0012h
    cmp     dx,  bx
    jnb     stop
    div     bx
    mov     bx,  ax
    in      al,  61h
    test    al,  3
    jne     j1
    or      al,  3
    out     61h, al
    mov     al,  0B6h
    out     43h, al
  }
 j1:
  asm {
    mov     al,  bl
    out     42h, al
    mov     al,  bh
    out     42h, al
  }
 stop: ;
}
void NoSound (void) {
  asm {
    in      al,  61H
    and     al,  0fcH
    out     61H, al
  }
}
void beep (int level) {
int i;
  for (i = 1; i; i++);
  for (i = 1; i; i++);
  for (i = 1; i; i++);
  Sound (level);
  for (i = 1; i; i++);
  for (i = 1; i; i++);
  for (i = 1; i; i++);
  NoSound();
}



#ifdef __Quickc
void interrupt far _process( unsigned res, unsigned rds, unsigned rdi,
                             unsigned rsi, unsigned rbp, unsigned rsp,
                             unsigned rbx, unsigned rdx, unsigned rcx,
                             unsigned rax, unsigned rip, unsigned rcs,
                             unsigned rflags ) {
#else
#pragma argsused
void interrupt _process (unsigned rbp, unsigned rdi, unsigned rsi, unsigned rds,
                         unsigned res, unsigned rdx, unsigned rcx, unsigned rbx,
                         unsigned rax, unsigned rip, unsigned rcs, unsigned rflag) {
#endif
#define ah (rax>>8)
#define al (rax&0x00ff)
#define bh (rbx>>8)
#define bl (rbx&0x00ff)

  if (   ah == 'G' && al == 'S'
      && bh == 'M' && bl == '1') {

    switch (rcx) {
      case API_INIT : {
      GSMAPI far * p = GetApi();
        IsInit = 1;
        SetExport (MK_FP (rsi, rdi));
        while (p && p->name && *p->name) {
          if (! strcmpi ((char*)p->name, "LibMain")) {
            p->proc();
            break;
          }
          p++;
        }
        p   = GetApi();
        rsi = FP_SEG(p);
        rdi = FP_OFF(p);
        asm mov rax, ds
        return;
      }
      case API_ISLOAD :
        if (! strcmpi (MK_FP (rsi, rdi), GetName())) rax = 1;
        else if (old_api) {
          old_api();
          asm mov rax, ax
        }
        else rax = 0;
        return;
      case API_END : {
      GSMAPI far * p = GetApi();

        if (IsInit) while (p && p->name && *p->name) {
          if (! strcmpi (p->name, "Wep")) {
            p->proc();
            break;
          }
          p++;
        }
        _dos_setvect (API, old_api);
        if (old_api)
          asm int API

{
unsigned s;
  asm mov s, es
  printf ("USR env=%x\n", *((unsigned*)MK_FP(s,0x2c)));
  printf ("USR psp=%x\n", lib_psp);
}
        asm mov es, lib_psp
        asm mov ax, es:[0x2c]
        asm mov es, ax
        asm mov ax, 0x4900
        asm int 0x21
        asm mov es, lib_psp
        asm mov ax, 0x4900
        asm int 0x21
        break;
      }
    }
  }
  else if (old_api) old_api();
}

int gsm_lib_main (int argc, PSTR *argv, PSTR *env) {
unsigned _rss, _rsp;

  if (argc!=2 && strcmp (argv[1], "GSM1")) {
    printf ("This program require GSM for dos.\n");
    return 1;
  }
  lib_psp = _psp;
  disable();
  old_api = _dos_getvect (API);
  _dos_setvect (API, _process);
  enable();
  asm mov _rss, ss
  asm mov _rsp, sp
  _dos_keep (0, _rss + (_rsp/16) - _psp);
  return 0;
}
```

Le point commun

```
/*
 S E R V E R . H

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
```

```
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/
#ifndef __SERVER_H
#define __SERVER_H


typedef struct s_GSMEXPORT {
  FARPROC test;
  FARPROC null;
} GSMEXPORT;

extern FARPROC _test;



#endif
```

Le serveur

```
/*
 S E R V E R . C

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/

#include <stdio.h>
#define __INCLUDE_API <server.h>
#define __DECLARE_COMMON_DATA_TYPE
#include <loadlib.h>

static GSMEXPORT gsm_export;


void error (type) {

  switch (type) {
    case ERR_UNABLE_TO_LOAD           :
      printf ("SERVER: Error Unable to load.\n");
      break;
    case ERR_LIBRARY_LOADER_NOT_FOUND :
      printf ("SERVER: Error Library loader not found.\n");
      break;
    case ERR_TOO_MANY_LIBRARY         :
      printf ("SERVER: Error Too many libraries.\n");
      break;
    case ERR_NOT_ENOUGHT_MEMORY       :
      printf ("SERVER: Error UNot enought memory.\n");
      break;
  }
  exit (1);
}


void export test_gsm_export (int i) {
  printf ("LOADLIB.TEST_GSM_FUNC: %d\n", i);
}



int main (int argc, char**argv) {
HLIB    lib_handle;
FARPROC proc;
char    buffer[50];
int     i;
int     head = 0;


  gsm_export.null = 0;
  gsm_export.test = test_gsm_export;

  i = init_loadlib (argv, argv[1], & gsm_export);
  if (i <0) error (i);

  lib_handle = load_library ("usr.exe");
  proc = get_proc_address (lib_handle, "beep");
  if (proc) proc(111);
  proc = get_proc_address (lib_handle, "beep");
  if (proc) proc(222);

  i = 612;
  head = add_proc_param (buffer, head, & i, sizeof (i));
  call_proc (proc, buffer, head);

  end_loadlib();
  return 0;
}
```

Le client

```
/*
 U S R . C

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 1, or (at your option)
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The author can be reached at gdw@cob.unice.fr or
Guilhem de Wailly, 81 bis bd Cessole, 06200 Nice France.
*/
#include <stdio.h>
#define __INCLUDE_API <server.h>
#define __DECLARE_COMMON_DATA_TYPE
#include <gsmapi.h>
```

```
void beep (int);

#ifdef __Quickc
  int _setargv__; /* external symbol defined by Borlanc c */
#endif

FARPROC _test;

void export f_beep(int a) {
  printf ("LIB.F_BEEP:%d\n", a);
  beep(1500);
  _test(123);
}

GSMAPI _export_api[] = {
  {"LibMain",  (FARPROC) LibMain },
  {"Wep",      (FARPROC) Wep     },
  {"beep",     (FARPROC) f_beep},
  {0,0}
};

PSTR GetName(void) {
  return "USR.EXE";
}

GSMAPI far * GetApi(void) {
  return _export_api;
}

void SetExport (GSMEXPORT far *ge) {
  _test = ge->test;
}

int LibMain() {
  printf ("LIB.LIBMAIN\n");
  return 1;
}

int Wep() {
  printf ("LIB.WEP\n");
  return 1;
}

int main (int argc, char**argv, char**env) {
  return gsm_lib_main (argc, argv, env);
}
```

CARACTERISTIQUES TECHNIQUES

- ❑ **Machine** : Ibm PC (80X86), Sun Sparc2
- ❑ **Sytème d'exploitation** : Dos,Windows, Unix.
- ❑ **Compilateur** : Sun c, Borland C++ 3.1, Quick C 2.5
- ❑ **Capacité mémoire** : 200 Ko minimum. Sous Dos, *gsm* gère des données de plus de 64 Ko (Inter segment). En mode protégé, il accède aux blocs de mémoire étendue selon le protocole DMPI.
- ❑ **Mémoire dynamique** : 50 Ko minimum. *gsm* offre la possibilité de régler au moment du lancement la taille du tas et du garbage.
- ❑ **Bibliothèques dynamiques** : sous Dos et Windows, *gsm* offre la possibilité de précharger dynamiquement des bibliothèques à liens dynamiques qui deviennent une extention du langage.