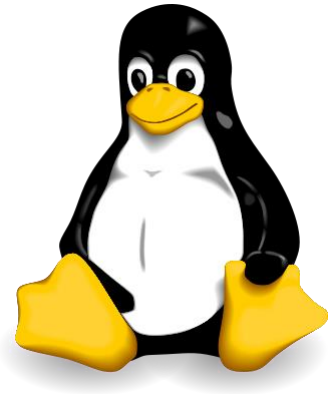
Yantra Technologies

www.yantra-technologies.com

Compilation Gcc



Facile

Normal

Difficile

Professionnel

Expert

carole.grondein@yantra-technologies.com
david.palermo@yantra-technologies.com

Yantra Technologies

Sommaire

1 - Introduction

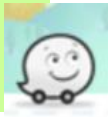
1.1 - Etapes de compilation

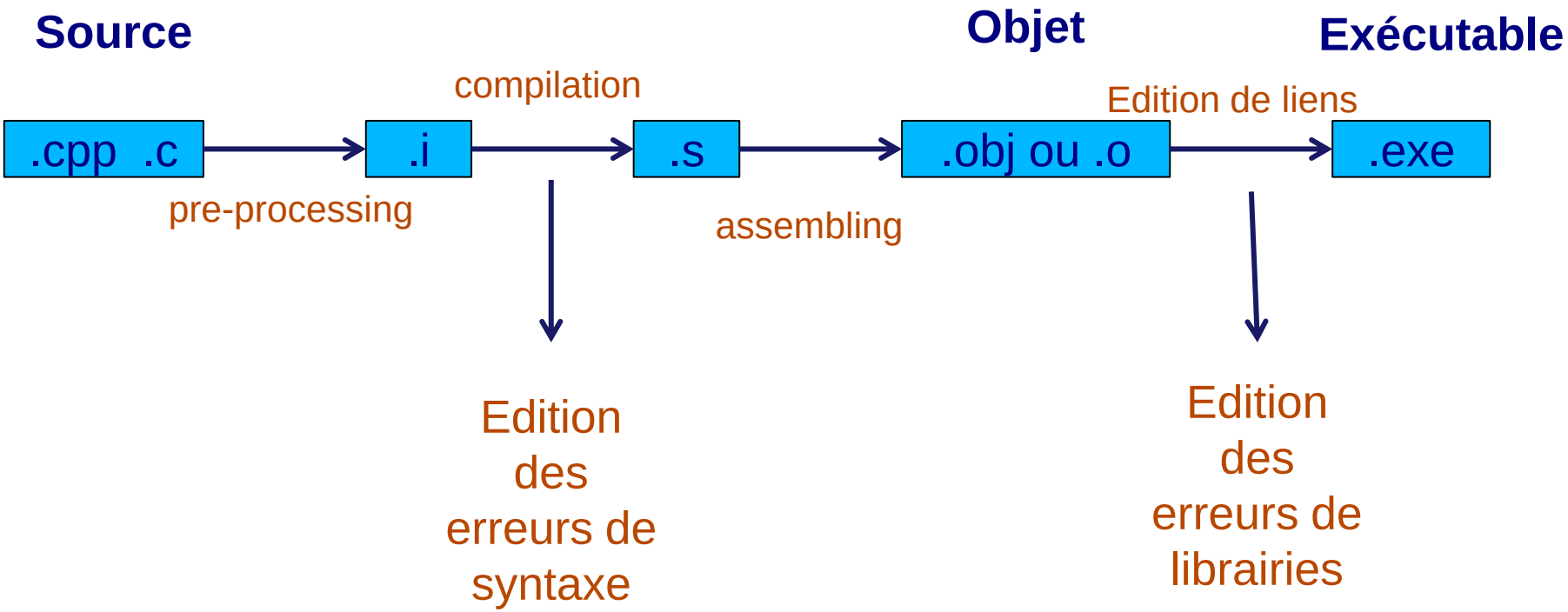
1.2 - Compilation

1.3 - Mon premier programme

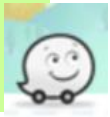
1.4 - Ma première compilation

1.1 - Etapes de compilation





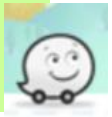
Préprocesseur



- suppression des commentaires (// et /* */)
- inclusion des fichiers .h ou .hh dans le fichier .c ou .cpp …. (#include)
- traitement des directives de compilation qui commencent par un caractère #.

gcc -E bonjour.c > bonjour.i ou g++ -E bonjour.cpp > bonjour.i

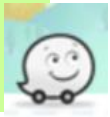
Compilation



La Compilation le code ascii en code assembleur.

gcc -S bonjour.i ou g++ -S bonjour.i création d'un bonjour.o

Assembling

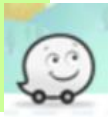


La Compilation le code ascii en code assembleur.

gcc bonjour.o ou g++ bonjour.o création d'un exécutable a.out

gcc –o bonjour bonjour.o ou g++ -o bonjour bonjour.o création d'un exécutable bonjour

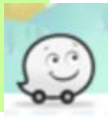
Edition de lien



La Compilation le code ascii en code assembleur.

gcc –c bonjour.s ou g++ -c bonjour.s création de bonjour.o

Compilation simplifier



gcc –c bonjour.c ou g++ –c bonjour.cpp

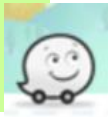
gcc –o bonjour bonjour.o

création de l'exécutable bonjour

gcc –o bonjour bonjour.c ou g++ -o bonjour bonjour.cpp

création de l'exécutable bonjour

1.2 - Compilation



Compilation d'une classe C++

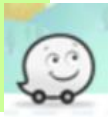
Pendant ce cours, nous utiliserons le compilateur Gnu C++, le g++, les options les plus utiles (voir manuel du g++) :

-c : Compile les fichiers sources, mais sans édition de liens. Le compilateur produit un fichier *objet* .o pour chaque fichier source.

-o *fichRes* : Donne le nom *fichRes* au fichier exécutable produit par la compilation et l'édition de liens. Quand cette option n'est pas employée, le nom d'exécutable par défaut est a.out.

-g : engendre de l'information pour le débogage, à utiliser avec un débogueur comme gdb (ou son interface utilisateur ddd).

Construction de bibliothèques



bibliothèques statique

En Unix ou Linux, pour réunir des fichiers objets, on utilisera typiquement la commande ar et ranlib, qui permet de créer des fichiers «archives» :

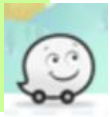
```
gcc –o F_1.o –c F_1.c
ar  -rv libmabib.a F_1.o F_2.o F_3.o
ranlib libmabib.a
```

bibliothèques dynamique

En Unix ou Linux, pour réunir des fichiers objets, on utilisera l'option spécial –fPIC pour compiler les objets et –shared a l'édition de lien:

```
gcc –fPIC –o F_1.o –c F_1.c
gcc -shared -fPIC -o libmabib.so F_1.o F_2.o F_3.o
```

Utilisation de bibliothèques



bibliothèques statique

gcc –o mon_excutable main.c /chemin/libmabib.a

ou

gcc –static –o mon_excutable main.c –L/chemin **–l**mabib

bibliothèques dynamique

gcc –o mon_excutable main.c /chemin/libmabib.so

ou

gcc –o mon_excutable main.c –L/chemin –lmabib

Ne pas oublier de mettre à jour la variable LD_LIBRARY_PATH
LD_LIBRARY_PATH=$ LD_LIBRARY_PATH:/chemin

1.3.1 - Mon premier programme

```
#ifndef Bonjour_H
#define Bonjour_H

namespace Formation
{
      void Bonjour()  ;
}

#endif
```

Bonjour.h

```
#include "Bonjour.h"
#include <iostream>

namespace Formation
{
     void Bonjour()  {
           std::cout << " Bonjour ! " << std::endl;
       }
}
```

Bonjour.cpp

1.3.1 - Mon premier programme

```
#include "Bonjour.h"

/* Mon premier programme  */

int main (int argc, char *argv[])
{
    Formation::Bonjour();
     return 0;
}
```

main.cpp

1.4 - Ma première compilation

g++ -c Bonjour.cpp

=> crée un fichier objet Bonjour.o

g++ -o Bonjour.exe main.cpp Bonjour.o

=> crée un exécutable Bonjour.exe

2 - Makefile

2.1 - Définition

Un Makefile est un fichier constitué de plusieurs règles de la forme :

Cible **:** dépendance commandes
<tab> commande

Cible ou target

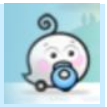
Chaque commande est précédée d'une tabulation.

Lors de l'utilisation d'un tel fichier via la commande make la première règle rencontrée, ou la règle dont le nom est spécifié, est évaluée.

L'évaluation d'une règle se fait en plusieurs étapes :

- Les dépendances sont analysées, si une dépendance est la cible d'une autre règle du Makefile, cette régle est à son tour évaluée.
- Lorsque l'ensemble des dépendances est analysé et si la cible ne correspond pas à un fichier existant ou si un fichier dépendance est plus récent que la régle, les différentes commandes sont exécutées.

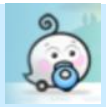
2.2 – Mon premier Makefile



```
Bonjour.exe: Bonjour.o main.o
    g++ -o Bonjour.exe Bonjour.o main.o

Bonjour.o: Bonjour.cpp Bonjour.h
    g++ -o Bonjour.o -c Bonjour.cpp -Wall –Wextra

main.o: main.cpp Bonjour.h
    g++ -o main.o -c main.c -Wall -Wextra
```

makefile

2.3 - Supprimer



```
all : Bonjour.exe
	echo "Executable crée"

Bonjour.exe: Bonjour.o main.o
	g++ -o Bonjour.exe Bonjour.o main.o

Bonjour.o: Bonjour.cpp Bonjour.h
	g++ -o Bonjour.o -c Bonjour.cpp -Wall –Wextra

main.o: main.cpp Bonjour.h
	g++ -o main.o -c main.c -Wall -Wextra

clean:
	 rm -rf *.o

mrproper: clean
	rm -rf Bonjour.exe
```

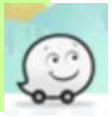
makefile

2.4 - Définition Variable

```
CXX=g++
CXXFLAGS= -Wall –Wextra
EXEC=Bonjour.exe
CXXINCLUDE=
CXXLIBS=
RM=rm

all : $(EXEC)
	echo "Executable crée"

Bonjour.exe: Bonjour.o main.o
	$(CXX) -o Bonjour.exe Bonjour.o main.o $(CXXLIBS)

Bonjour.o: Bonjour.cpp Bonjour.h
	$(CXX) $(CXXFLAGS)  $(CXXINCLUDE) -o Bonjour.o -c Bonjour.cpp

main.o: main.cpp Bonjour.h
	$(CXX) $(CXXFLAGS) $(CXXINCLUDE) -o main.o -c main.c
clean:
	$(RM) -rf *.o

mrproper: clean
	$(RM) -rf $(EXEC)
```

makefile

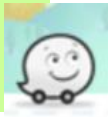
2.5a - Variables internes



Il existe plusieurs variables internes au Makefile :

- $@ Le nom de la cible
- $< Le nom de la première dépendance
- $^ La liste des dépendances
- $? La liste des dépendances plus récentes que la cible
- $* Le nom du fichier sans suffixe

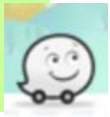
2.5b - Variables internes



```
CXX=g++
CXXFLAGS= -Wall –Wextra
EXEC=Bonjour.exe
CXXINCLUDE=
CXXLIBS=
RM=rm

all : $(EXEC)
	echo "Executable crée"

Bonjour.exe: Bonjour.o main.o
	$(CXX) -o $@ $^  $(CXXLIBS)

Bonjour.o: Bonjour.cpp Bonjour.h
	$(CXX) $(CXXFLAGS)  $(CXXINCLUDE) –o $@ -c $<
main.o: main.cpp Bonjour.h
	$(CXX) $(CXXFLAGS) $(CXXINCLUDE) -o $@ -c $<
clean:
	$(RM) -rf *.o

mrproper: clean
	$(RM) -rf $(EXEC)
```

makefile

2.6a - Les règles d'inférence



Makefile permet également de créer des règles génériques (par exemple construire un .o à partir d'un .c) qui se verront appelées par défaut.

Une telle règle se présente sous la forme suivante :

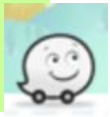
- %.o: %.c commandes

Il existe une autre notation plus ancienne de cette règle :

- .c.o: commandes

Il devient alors aisé de définir des règles par défaut pour générer nos différents fichiers.

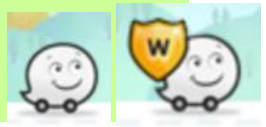
2.6b - Les règles d'inférence



```
CXX=g++
CXXFLAGS= -Wall –Wextra
EXEC=Bonjour.exe
CXXINCLUDE=
CXXLIBS=
RM=rm

all : $(EXEC)
	echo "Executable créé"

Bonjour: Bonjour.o main.o
	$(CXX) -o $@ $^ $(CXXLIBS)

%.o: %.cpp %.h
	$(CXX) $(CXXFLAGS)  $(CXXINCLUDE) –o $@ -c $<

clean:
	$(RM) -rf *.o

mrproper: clean
	$(RM) -rf $(EXEC)

main.o: Bonjour.h
Bonjour.o : Bonjour.h
```

makefile

2.7 - La cible .PHONY et .PRECIOUS



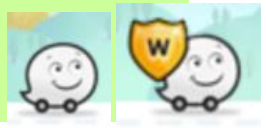
.PHONY Cible dont les dépendances seront systématiquement reconstruites.

.PRECIOUS Cible dont les dépendances ne doivent pas être effacées automatiquement

```
CXX=g++
CXXFLAGS= -Wall –Wextra
EXEC=Bonjour.exe
CXXINCLUDE=
CXXLIBS=
RM=rm

all : $(EXEC)
	echo "Executable crée"

Bonjour.exe: Bonjour.o main.o
	$(CXX) -o $@ $^ $(CXXLIBS)

%.o: %.cpp %.h
	$(CXX) $(CXXFLAGS)  $(CXXINCLUDE) –o $@ -c $<

.PHONY: clean mrproper
.PRECIOUS : *.o

clean:
	$(RM) -rf *.o

mrproper: clean
	$(RM) -rf $(EXEC)

main.o: Bonjour.h
Bonjour.o : Bonjour.h
```

makefile

2.8a - Génération de la liste des fichiers objets



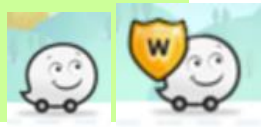
Plutôt que d'énumérer la liste des fichiers objets dans les dépendances de la règle de construction de notre exécutable, il est possible de la générer automatiquement à partir de la liste des fichiers sources. Pour cela nous rajoutons deux variables au Makefile :

- SRC qui contient la liste des fichiers sources du projet.
- OBJ pour la liste des fichiers objets.

La variable OBJ est remplie à partir de SRC de la manière suivante :

OBJ= $(SRC:.c=.o) Pour chaque fichier .c contenu dans SRC nous ajoutons à OBJ un fichier de même nom mais portant l'extension .o.

2.8b - Génération de la liste des fichiers objets



```
CXX=g++
CXXFLAGS= -Wall –Wextra
EXEC=Bonjour.exe
CXXINCLUDE=
CXXLIBS=
RM=rm
SRC= Bonjour.cpp main.cpp
OBJ= $(SRC:.cpp=.o)

all : $(EXEC)
	echo "Executable crée"

Bonjour.exe: $(OBJ)
	$(CXX) -o $@ $^  $(CXXLIBS)

%.o: %.cpp %.h
	 $(CXX) $(CXXFLAGS)  $(CXXINCLUDE) –o $@ -c $<

.PHONY: clean mrproper
.PRECIOUS : $(OBJ)
clean:
	 $(RM) -rf *.o

mrproper: clean
	$(RM) -rf $(EXEC)

main.o: Bonjour.h
Bonjour.o : Bonjour.h
```

makefile

2.9 - Commandes silencieuses

```
CXX=g++
CXXFLAGS= -Wall –Wextra
EXEC=Bonjour.exe
CXXINCLUDE=
CXXLIBS=
RM=rm
SRC= Bonjour.cpp main.cpp
OBJ= $(SRC:.cpp=.o)

all : $(EXEC)
	@echo "Executable crée"

Bonjour.exe: $(OBJ)
	@$(CXX) -o $@ $^  $(CXXLIBS)

%.o: %.cpp %.h
	 @$(CXX) $(CXXFLAGS)  $(CXXINCLUDE) –o $@ -c $<

.PHONY: clean mrproper
.PRECIOUS : $(OBJ)
clean:
	 @$(RM) -rf *.o

mrproper: clean
	@$(RM) -rf $(EXEC)

main.o: Bonjour.h
Bonjour.o : Bonjour.h
```

makefile

2.10 - Les Makefiles conditionnels

```
CXX=g++
DEBUG=yes
ifeq ($(DEBUG),yes)
    CXXFLAGS= -Wall –Wextra -g
else
    CXXFLAGS= -Wall –Wextra
endif

EXEC=Bonjour.exe
CXXINCLUDE=
CXXLIBS=
RM=rm
SRC= Bonjour.cpp main.cpp
OBJ= $(SRC:.cpp=.o)

all : $(EXEC)
ifeq ($(DEBUG),yes)
    @echo " Executable crée en mode debug"
else
    @echo " Executable crée en mode release"
endif
```

```
Bonjour.exe: $(OBJ)
	@$(CXX) -o $@ $^  $(CXXLIBS)
%.o: %.cpp %.h
	@$(CXX) $(CXXFLAGS) $(CXXINCLUDE) –o $@ -c $<
.PHONY: clean mrproper
.PRECIOUS : $(OBJ)
clean:
	@$(RM) -rf *.o
mrproper: clean
	@$(RM) -rf $(EXEC)
main.o: Bonjour.h
Bonjour.o : Bonjour.h
```

makefile

2.11 - Création dépendance avec GCC

```
CXX=g++
DEBUG=yes
ifeq ($(DEBUG),yes)
        CXXFLAGS= -Wall –Wextra -g
else
        CXXFLAGS= -Wall –Wextra
endif
EXEC=Bonjour.exe
CXXINCLUDE=
CXXLIBS=
RM=rm
SRC= Bonjour.cpp main.cpp
OBJ= $(SRC:.cpp=.o)

all : $(EXEC)
ifeq ($(DEBUG),yes)
        @echo " Executable crée en mode debug"
else
        @echo " Executable crée en mode release"
endif

Bonjour.exe: $(OBJ)
        @$(CXX) -o $@ $^ $(CXXLIBS)
%.o: %.cpp %.h
         @$(CXX) $(CXXFLAGS)  $(CXXINCLUDE) –o $@ -c $<
```

%.d: %.cpp
$(CXX) $(CXXFLAGS) $(CXXINCLUDE) -MM -MP -MF"$@" -MT"$(@:%.d=%.o)" -c $<

```
.PHONY: clean mrproper
.PRECIOUS : $(OBJ)

clean:
         @$(RM) -rf *.o
mrproper: clean
        @$(RM) -rf $(EXEC)
```

-include $(OBJ:.o=.d)

makefile

2.12 - Création Librairie static

```
CXX=g++
DEBUG=yes
ifeq ($(DEBUG),yes)
	CXXFLAGS= -Wall –Wextra -g
else
	CXXFLAGS= -Wall –Wextra
endif

EXEC=Bonjour.exe
CXXINCLUDE=
CXXLIBS= -L.-lBonjour
RM=rm
SRC= Bonjour.cpp
OBJ= $(SRC:.cpp=.o)
MAIN=main.cpp
LIB=libBonjour.a

all : $(EXEC)
ifeq ($(DEBUG),yes)
	@echo " Executable crée en mode debug"
else
	@echo " Executable crée en mode release"
endif

Bonjour.exe: main.o $(LIB)
	@$(CXX) -o $@  $<  $(CXXLIBS)

libBonjour.a : $(OBJ)
	ar -cr $@ $^
	ranlib  $@

%.o: %.cpp %.h
	 @$(CXX) $(CXXFLAGS)  $(CXXINCLUDE) –o $@ -c
	$<

%.d: %.cpp
	$(CXX) $(CXXFLAGS)  $(CXXINCLUDE) -MM -MP -
	MF"$@" -MT"$(@:%.d=%.o)" -c $<

.PHONY: clean mrproper
.PRECIOUS : $(OBJ)

clean:
	 @$(RM) -rf *.o
mrproper: clean
	@$(RM) -rf $(EXEC)

-include $(OBJ:.o=.d)
-include $(MAIN:.o=.d)
```

makefile

2.12 - Création Librairie dynamic

```
CXX=g++
ifeq ($(DEBUG),yes)
CXXFLAGS= -Wall -Wextra -g
else
CXXFLAGS= -fpic -Wall -Wextra -pedantic -
      msse2 -mfpmath=sse
endif
EXEC=Bonjour-static Bonjour-dyn
CXXINCLUDE=-I .
CXXLIBS= -L. -lBonjour
RM=rm
SRC= Bonjour.cpp
OBJ= $(SRC:.cpp=.o)
MAIN=main.o
LIB=libBonjour.a libBonjour.so

all : $(EXEC)
ifeq ($(DEBUG),yes)
      echo " Executable crée en mode debug"
else
      echo " Executable crée en mode release"
endif
```

```
Bonjour-static: $(MAIN) $(LIB)
      $(CXX) -static -o $@  $<  $(CXXLIBS)
Bonjour-dyn: $(MAIN) $(LIB)
      $(CXX) -o $@  $< $(CXXLIBS)
libBonjour.a : $(OBJ)
      ar -cr $@ $^
      ranlib  $@
libBonjour.so : $(OBJ)
      $(CXX) -o $@ -shared $^
%.o: %.cpp %.h
       $(CXX) $(CXXFLAGS)  $(CXXINCLUDE) -o $@ -c $<
%.d: %.cpp
      $(CXX) $(CXXFLAGS)  $(CXXINCLUDE) -MM -MP -MF"$@" -
      MT"$(@:%.d=%.o)" -c $<
.PHONY: clean mrproper
.PRECIOUS : $(OBJ)
clean:
       @$(RM) -rf *.o *.d
mrproper: clean
      @$(RM) -rf $(EXEC) libBonjour.a
-include $(OBJ:.o=.d)
-include $(MAIN:.o=.d)
```

makefile

2.13a - Option compilateur Gcc

Option	Descriptif
-msse2	Active les extensions des processeurs Intel et AMD
-mfpmath=sse	Utilise l'extension SSE pour les calculs en nombre flottant
-mno-align-double **-malign-double**	Contrôle si GCC aligne les variables « double », « long double » et « long long » à des limites de deux ou de un mot. Aligner les variables « double » à des limites de deux mots produira du code tournant un peu plus rapidement sur un Pentium aux dépens d'une plus grosse consommation mémoire.
-m32	Force la compilation en 32 bits sur les machines 64 bits
-ansi :	Limite le langage à la norme ANSI C
-Wall :	affiche un maximum d'avertissements. Permet souvent d'éviter des bugs.
-Wextra	Active plus d'avertissements que -Wall (pour gcc < 4.0, l'option s'appelait "-W" tout court)
-fmessage-length=n	Essaye de formater les messages d'erreur de sorte qu'ils conviennent à des lignes d'environ *n* caractères. La valeur par défaut est de 72 caractères pour **g++**, et 0 pour les autres frontaux supportés par GCC. Si *n* vaut zéro, aucun découpage en lignes ne sera effectué ; chaque message d'erreur apparaîtra sur une seule ligne.
-static	Crée un exécutable statique, ce qui accroit la portabilité.

2.13b - Option compilateur Gcc

Option	Descriptif
-On	Indique le niveau d'optimisation (n peut prendre les valeurs allant de 0 à 3, ou « s » pour optimiser la taille des binaires)
-fabi-version=0	indique au compilateur de choisir une implémentation ABI particulière : • 0 : l'implémentation ABI la plus récente • 1 : l'implémentation ABI compatible g++ 3.2 • 2 : l'implémentation ABI la plus conforme
-fpic	Génère du code indépendant de son adresse de chargement (nécessaire pour les bibliothèques dynamiques, dont le chargement peut être réalisé par le système n'importe où en mémoire à priori
-march=cpu	Indique le type de processeur pour lequel le code doit être généré. Le code généré sera spécifique à ce processeur, et ne fonctionnera peut-être pas sur un autre modèle de la même famille. Cette option active automatiquement l'option -mcpu avec le même processeur.
-mtune=cpu	Fait en sorte que le code soit plus rapide sur architecture cpu tout en continuant à tourner sur i386.
-fbacktrace	Affiche lors d'une sortie en erreur du programme où l'erreur s'est produite
-fbounds-check	Permet la génération de contrôles durant l'exécution sur des indices de tableau et contre les valeurs minimales et maximales déclarées.

2.13b - Option compilateur Gcc

Option	Descriptif
-fpermissive	autorise le code non conforme
-traditional	Essaie de supporter certains aspects des compilateurs C traditionnels. Pour les détails, voyez le manuel GNU C; la liste dupliquée ici a été supprimée afin de ne pas recevoir de réclamations quand elle n'est pas à jour. Mais notons une chose sur les programmes C++ uniquement (pas C). `-traditional' a un effet supplémentaire en C++ : l'affectation vers this est permise. Ceci a le même effet que `-fthis-is-variable
-Wunderflow	Produit un avertissement quand des expressions constantes numériques sont rencontrées, qui rapportent un dépassement de borne pendant la compilation
-shared	Produit un objet partagé qui peut être lié avec d'autres objets pour former un exécutable. Seul un petit nombre de systèmes supportent cette option.

2.13c - Option compilateur Gcc

Option	Descriptif
-finit-integer	Sert à initialiser les variables de type integer
-finit-character	Sert à initialiser les variables de type character
-finit-logical	Sert à initialiser les variables de type logical
-finit-real	Sert à initialiser les variables de type real
-falign-commons	Sert à forcer l'alignement approprié de toutes les variables dans un bloc COMMUN en les alignant comme nécessaire. Sur certaines plates-formes cela est obligatoire, sur d'autres il augmente la performance à l'exécution
-fno-automatic	Rend toutes les variables locales static (SAVE)
-finit-local-zero	Sert à initialiser les variables locales à zéro

2.13d - Option compilateur Gcc

Option	Descriptif
-DSYMBOL[=valeur]	Cette option permet de définir des constantes ayant une valeur pour le préprocesseur. L'exemple précédent montre que le compilateur ajoute lui-même un grand nombre de ces constantes, permettant de caractériser le système sur lequel la compilation s'exécute (compilation conditionnelle).
-Ldirectory -llibrary	Ces options sont utilisées pour l'édition de liens. Elles permettent d'inclure des fichiers objets supplémentaires à ceux que l'on a compilé. Ces fichiers objets sont regroupés dans un fichier unique, nommé *librairie*. Une librairie est donc un ensemble de fichiers .o. Elle peut être liée statiquement ou dynamiquement à l'exécutable. L'option -Ldirectory indique le directory où le linker doit rechercher la librairie (au moment de l'édition des liens), et l'option -llibrary précise le nom de cette librairie. Selon les architectures, le fichier aura comme nom liblibrary.so pour une librairie dynamique ou library.a pour une librairie statique.
-IDirectory	Ajoute un répertoire pour la recherche des fichiers inclus.

2.13e - Option compilateur Gcc

Option	Descriptif
-finline-functions	Active la fonction d'« inline »
-ffast-math et **-funsafe-math-optimizations**	Se permet de réorganiser les opérations mathématiques sur les nombres flottants. Par défaut, gcc se l'interdit car cette opération peut produire des erreurs de calcul (comprendre, sur les chiffres après les virgules, on peut perdre en précision).
-funroll-loops	Déroule les boucles
-pedantic	Encore plus restrictif que -ansi. En gros, si votre programme se compile avec -ansi, il se compile partout
-Werror	Les avertissements sont traités comme des erreurs

3 - Bibliographie

- Tutorial :
  - http://www.gnu.org/software/make/
  - http://gl.developpez.com/tutoriel/outil/makefile/
  - http://iihm.imag.fr/blanch/howtos/GNUMake.html
  - http://www.siteduzero.com/tutoriel-3-31992-compilez-sous-gnu-linux.html
  - http://www.linux-kheops.com/doc/man/manfr/man-ascii-0.9/man1/gcc.1.txt.html