Outils de développement

L'utilitaire ar : gestion des bibliothèques (1)

- bibliothèque : fichier construit à partir de fichiers objets (.o) indépendants
  - regroupe :
    - ✔ sous-programmes d'intérêt général
    - ✔ sous-programmes spécifiques à un projet
  - permet d'éviter d'avoir à préciser une multitude de fichiers .o lors de l'édition de liens
  - 1 bibliothèque ↔ 1 fichier entête = interface (constantes, types, prototypes des fonctions)

L'utilitaire ar : gestion des bibliothèques (2)

- l'utilitaire *ar* permet :
  - création de bibliothèques
  - l'ajout d'un fichier .o dans une bibliothèque
  - le retrait d'un fichier .o d'une bibliothèque
  - le listage des fichiers .o d'une bibliothèque

- Par convention, nom d'une bibliothèque de la forme lib---.a

L'utilitaire ar : gestion des bibliothèques (1)

- bibliothèques les plus courantes :
  - libc.a : fonctions C standard

    → automatiquement recherchée pour créer un exécutable
  - libm.a : fonctions mathématiques
  - libl.a : utilisée avec lex
  - liby.a : utilisée avec yacc

édition de liens (1)

- recherche des fichiers objets nécessaires pour créer l'exécutable
  - dans la bibliothèque standard
  - dans la liste de bibliothèques précisée sur la ligne de commande

édition de liens (2)

- Ex : utilisation de printf dans un programme
  - compilation ⇒ le .o contient une référence vers printf
  - recherche de cette référence par l'e.d.l.
    - ✔ consultation table des bibliothèques
    - ✔ accès au fichier .o contenant la définition de printf
    - ✔ ajout de ce fichier .o en fin de fichier exécutable
    - ✔ fusion des tables de symboles
  - résolution des références manquantes suivantes
- seuls les fichiers objet contenant la définition de références manquantes sont inclus dans l'exécutable.

la commande gcc (1)

- enchaîne automatiquement l'appel des différents outils réalisant la traduction d'un source en binaire
  - précompilation (source → source étendu)
  - compilation (source étendu → assembleur)
  - l'assemblage (assembleur → binaire)
  - édition de lien (binaire + fonctions des bibliothèques ⇒ fichier exécutable)
- syntaxe : gcc [-options] référence ....

la commande gcc (2)

- principales options ...
  - ... de gcc :
    - ✔ -c : suppression de l'appel à l'éditeur de lien ld ⇒ conservation du .o normalement supprimé
  - ... du préprocesseur :
    - ✔ -D*identificateur* ou -D*identificateur=valeur :*

      ⇔ #define identificateur .... dans le programme
    - ✔ -I *répertoire* : où rechercher les fichiers entête (/usr/include par défaut)

la commande gcc (3)

- ... de l'éditeur de lien (ld)
  - ✔ -o : permet de préciser le nom à donner au fichier exécutable (a.out par défaut)
  - ✔ -l*nom* : liaison avec libnom.a
  - ✔ -L *répertoire* : où rechercher les biblios (/lib ou /usr/lib par défaut)

L'utilitaire make (1)

- permet de maintenir un programme obtenu à partir d'un ensemble de modules distincts
- prend en compte :
  - dates de modifications du programme à maintenir
  - différents modules entrant dans la constitution du programme
- ⇒ création d'un *fichier des dépendances* (makefile) contenant
  - description du graphe des liaisons entre modules
  - actions à réaliser pour mettre à jour ce programme

L'utilitaire make (1)

- la commande make :

  make [-f ref_fic_make] [- args_optionnels] [ref...]

  où

  - ref_fic_make = référence du fichier des dépendances (par défaut makefile ou Makefile)
  - ref : référence du fichier à reconstruire (par défaut : premier nom de fichier apparaissant dans une ligne de dépendances du fichier ref_fic_make)

L'utilitaire make (2)

- le fichier des dépendances : fichier ASCII contenant
  - des commentaires (ignorés par make) : tout ce qui suit le caractère # sur une ligne
  - des macro-définitions
    - ✔ syntaxe : *<identificateur_macro>= <chaîne>*
    - ✔ à l'exécution de make, $*<identificateur_macro>* remplacé par *<chaîne>*

L'utilitaire make (3)

- des relations de dépendance

  syntaxe : *ref_cible* .... **:** [*ref*....]

  - ✔ références séparées par un espace
  - ✔ début de ligne en colonne 1
  - ✔ références peuvent contenir lettres, chiffres, • , / , ?, *
  - ✔ signification : *ref_cible* dépend de *ref*(s)

L'utilitaire make (4)

- des commandes shell

  syntaxe : <tab> cde_shell_sur_une_ligne

  - ✔ les commandes shell doivent être placées après une relation de dépendance
  - ✔ signification :

    si la date de dernière modification d'une réf_cible est antérieure à la date de dernière modification d'un fichier dépendant

    alors : exécuter les commandes shell qui suivent

L'utilitaire make : exemple

```
OBJ = p1.o p2.o                          # macro-définitions
INCDIR = -I rep1
LIBS = -lbib1 -lbib2
appli : $(OBJ)                           #relation de dépendance
      gcc -o appli $(LIBS) $(OBJ)    # édition de liens
          # autre notation : gcc -o $@ $+ $(LIBS)
p1.o : p1.c p.h q.h
      gcc -c p1.c  $(INCDIR) #compilation sans édition de liens
p2.o : p2.c p.h
      gcc -c p2.c  $(INCDIR) #compilation sans édition de liens
clean :                                  # cible de nettoyage
      rm *.o
```