Organisation de l'environnement de développement C

1 Existant

Le fichier *fichiers-01.tar.gz* contient les répertoires suivants :

memoire : *Makefile memoire.c memoire.h*
Source du module *memoire* d'allocation et libération de mémoire avec terminaison sur défaut de mémoire.

chaine : *Makefile chaine.c chaine.h test-chaine-1.c test-chaine-2.c*
Source du module *chaine* de fonctions utiles pour manipuler des chaînes de caractères. Ce module nécessite le module *memoire*.

vext : *Makefile vext.c vext.h test-vext-1.c test-vext-2.c test-vext-3.c*
Source du module *vext* implémentant un vecteur extensible. Ce module nécessite les modules *memoire* et *chaine*.

paire : *Makefile paire.c paire.h test-paire.c*
Source du module *paire* implémentant un chaînage au moyen d'une paire de références (modèle LISP). Ce module nécessite le module *memoire*.

file : *Makefile file.c file.h test-file.c*
Source du module *file* implémentant le type abstrait *file*. Ce module nécessite les modules *memoire*, *chaine* et *paire*.

magasin : *Makefile client.c client.h magasin.c*
Source du programme *magasin* effectuant une simulation simple de files d'attente aux caisses d'un magasin. Ce programme nécessite les modules *memoire*, *chaine*, *paire* et *file*.

2 Création et utilisation d'une bibliothèque

2.1 Principe

Sous Unix (par exemple sous GNU/Linux) une **bibliothèque** (*library*) est un fichier d'archivage UNIX contenant des fichiers objet (*.o*). Par convention, le nom d'une bibliothèque est de la forme *libnom.a*, le suffixe *.a* étant le suffixe attribué aux fichiers d'archivage Unix. L'éditeur de liens *ld* est capable d'extraire d'une telle bibliothèque les fichiers .o dont il a besoin pour effectuer l'édition de liens. La construction d'une bibliothèque se fait au moyen de la commande d'archivage *ar*.

2.2 Travail à effectuer

Référez vous au manuel pour obtenir des informations sur *gcc* ou *make*. Vous pouvez taper dans un terminal la commande *man gcc*. Ou en passant par *Emacs* avec l'aide hypertextuelle, tapez C-h puis i pour obtenir l'aide puis C-s gcc pour trouver le sujet sur *gcc*.

EXERCICE 1 – Créez dans votre répertoire ≺*environnement de developpement*≻ une copie de l'arborescence contenue dans l'archive *fichiers-01.tar.gz*.

EXERCICE 2 – Allez dans le répertoire *memoire* et reconstruisez le fichier *memoire.o* en utilisant directement la commande *gcc* (option *-c*). Faire de même dans le répertoire *chaine*.

EXERCICE 3 – Allez dans le répertoire *vext* et examinez le fichier *Makefile* qu'il contient. Puis, construisez avec la commande *make* les fichiers *vext.o*, et *test-vext-1*.

EXERCICE 4 – Remontez dans le répertoire parent et construisez, au moyen de la commande *ar*, le fichier *libtest.a* contenant les fichiers *memoire.o* et *chaine.o* (options *cr*). Vérifiez son contenu (option *t*).

EXERCICE 5 – Reconstruisez l'exécutable *test-vext-1* à l'aide de la commande *gcc*, en utilisant cette fois la bibliothèque *test* à la place du fichier *memoire.o*.

EXERCICE 6 – On peut désigner une bibliothèque au moyen de l'option *-l* de l'éditeur de liens. Lisez la documentation de cette option dans la page de manuel de la commande *gcc*. Reconstruisez *test-vext-1* en utilisant ce mécanisme. Vous pouvez utiliser l'option *-L* mentionnée dans la documentation de *gcc*.

EXERCICE 7 – Déplacez dans un répertoire *lib/* la bibliothèque. Revenez dans le répertoire *vext* et recommencez l'édition de liens avec les bonnes options.

3 Organisation de l'interface d'une bibliothèque

3.1 Principe

En C, l'interface d'une bibliothèque est l'ensemble des fichiers en-tête (`.h`) nécessaires à sa mise en oeuvre. Ils peuvent être trouvés automatiquement par le compilateur grâce à l'option `-I` qui joue un rôle similaire à `-L` pour l'édition de liens. Un fichier en-tête est recherché dans les répertoires indiqués par l'option `-I` (ou à défaut, dans le répertoire courant) s'il est placé entre guillemets (`"chaine.h"`). Les fichiers en-tête des bibliothèques du système sont entourés de chevrons (`<stdio.h>`).

3.2 Travail à effectuer

Exercice 8 – Retournez dans le répertoire `vext` et exécutez la commande `make clean`. Puis, éditez les fichiers source `.c` pour ôter les chemins relatifs (par ex. `../memoire/`) des directives `#include`. Compilez maintenant les fichiers source du répertoire `vext` en utilisant l'option `-I` pour localiser les fichiers en-tête.

Exercice 9 – Modifiez le fichier `Makefile` du répertoire `vext` de manière à utiliser la bibliothèque `test`, et utilisez la variable `CPPFLAG` pour l'option `-I`. Vous devez obtenir le résultat suivant :

```
$ make clean
rm -f *.o test-vext-1 test-vext-2 test-vext-3
$ make
cc -g -Wall -I../memoire -I../chaine -c -o vext.o vext.c
cc -g -Wall -I../memoire -I../chaine -c -o test-vext-1.o test-vext-1.c
cc test-vext-1.o vext.o -L../lib -ltest -lm -o test-vext-1
cc -g -Wall -I../memoire -I../chaine -c -o test-vext-2.o test-vext-2.c
cc test-vext-2.o vext.o -L../lib -ltest -lm -o test-vext-2
cc -g -Wall -I../memoire -I../chaine -c -o test-vext-3.o test-vext-3.c
cc test-vext-3.o vext.o -L../lib -ltest -lm -o test-vext-3
```

4 Réorganisation de l'environnement

4.1 Découpage en modules

Exercice 10 – Réinitialisez le contenu des répertoires, par exemple en exécutant `make clean` dans chaque répertoire. Vérifiez que chaque répertoire ne contient plus que les fichiers source `.c` et `Makefile`.

Réorganisez cette nouvelle arborescence afin d'obtenir la structure suivante :

```
bin/                src/bcb/              src/sdd/vext/
include/            src/bcb/chaine/       src/sdd/paire/
lib/                src/bcb/memoire/      src/sdd/file/
src/                src/sdd/              src/magasin/
```

Exercice 11 – Déplacez l'ensemble des fichiers en-tête (`*.h`) dans le répertoire `include/`. Dans la suite, il vous faudra mettre à jour au fur et à mesure les fichiers `Makefile` (avec l'option `-I`) ainsi que les fichiers source `.c`, afin d'en retirer systématiquement les chemins relatifs dans les directives `#include`.

4.2 Constructions des bibliothèques *bcb* et *sdd*

Effacez votre bibliothèque `lib/libtest.a`. Nous appelerons désormais cette bibliothèque `bcb` pour *Bibliothèque C de Base*. Elle sera contruite automatiquement via l'utilitaire `make`.

Exercice 12 – Placez vous dans le répertoire `src/bcb/memoire/`. Modifiez le fichier `Makefile` permettant de générer le fichier objet `memoire.o`. Utilisez une variable `INCDIR` pour factoriser le chemin du répertoire `include/`. Faites de même dans le répertoire `src/bcb/chaine/`.

Exercice 13 – Placez vous dans le répertoire `src/bcb/`. Éditez le fichier `Makefile`, qui permettra de créer la bibliothèque `libbcb.a` contenant les fichiers objets `memoire.o` et `chaine.o`. Ce fichier `Makefile` fera appel à la commande `make` depuis les sous-répertoires `memoire/` et `chaine/` (option `-C`).

Exercice 14 – Ajoutez au fichier `src/bcb/Makefile` une cible `clean` qui supprime l'archive et provoque le nettoyage par descente récursive des sous-répertoires `memoire/` et `chaine/` (en invoquant la commande `make clean` de ces sous-répertoires). Pour ce faire, votre `Makefile` utilisera une boucle `for` afin d'invoquer la commande `make clean` des sous-répertoires `memoire/` et `chaine/`.

Exercice 15 – Rajoutez une action associée à une cible de nom `install` permettant d'installer automatiquement la bibliothèque. On pourra éventuellement tester l'existence du répertoire `lib`, et le créer automatiquement lorsqu'il est absent.

Exercice 16 – Rajoutez les actions permettant de construire les programmes de test.

Exercice 17 – Créez sur le modèle précédent dans le répertoire *src/sdd* un fichier *Makefile* qui génère et installe une seconde bibliothèque *libsdd.a*. Cette bibliothèque *structures de données* regroupera les objets *vext*, *paire* et *file*.

5 Exercice complémentaire – Génération de Makefile

Allez dans le répertoire *src/magasin*. Renommez le fichier *Makefile* en *Makefile.old*. Écrivez un programme en langage shell, de nom *makemake*, qui, lorsqu'il est invoqué de la manière suivante : *makemake magasin* produise sur sa sortie standard le résultat :

```
ROOT=../..
CFLAGS= -g -std=c99 -Wall
CPPFLAGS= -I$(ROOT)/include
LDFLAGS= -L$(ROOT)/lib -lbcb -lsdd -lm

OUTFILE=magasin

OBJ= client.o magasin.o

$(OUTFILE) : $(OBJ)
        $(CC) $(OBJ) $(LDFLAGS) -o $(OUTFILE)

client.o: client.c client.h
magasin.o: magasin.c client.h

.PHONY: clean
clean :
        rm -f $(OBJ) $(OUTFILE)
```

Utiliser ce programme pour générer le Makefile du programme magasin. Tester le Makefile obtenu.

Remarque Certaines parties sont recopiées littéralement et d'autres sont générées par des mécanismes de substitution, ou en utilisant la commande *gcc* avec l'option *-MM*.

On peut soit utiliser les mécanismes de redirection

```
cat <<EOF
⋮
EOF
```

et

```
cat <<'EOF'
⋮
EOF
```

soit inclure des fichiers de dépendances (`.d`) générés par `gcc -MM`.

Exercice 18 – Modifier le fichier *Makefile* de manière à rajouter une cible *install* qui installe le programme exécutable *magasin* dans un répertoire *bin* situé dans le répertoire *src*.

Exercice 19 – Ajouter un fichier *Makefile* dans les répertoires *src* et *hierarchie* avec les cibles *install* et *clean*

Exercice 20 – Améliorer la commande *makemake* de manière à ce que le résultat aille dans un fichier de nom *Makefile*.

Exercice 21 – Améliorer la commande *makemake* de manière à ce que, lorsqu'un fichier *Makefile* existe déjà, elle le sauvegarde sous un autre nom avant de créer le nouveau fichier.

Exercice 22 – Améliorer la commande *makemake* de manière à ce qu'elle accepte en option :
- les bibliothèques à utiliser lors de l'édition de liens (sous la forme *-lnom*)
- les fichiers *.c* à prendre en compte