CS-210 : Bases de langage C, 2ème année

Christian Duccini, année 2008-2009

TP1 : Prise en main du langage C, compilation, make.

Principe des TP de prog C :

Les TPs se déroulent sur 1 séance. Vous avez 2 semaines calendaires max pour me faire parvenir votre code « commenté » (format « papier », un par binôme), (voir fichiers et exemples fourni). Outre les comptes-rendus, il y aura aussi parfois des (petites) évaluations en début de séance portant sur le TP précédent, et peut-être, selon les possibilités, des évaluations sur machine.

Les sujets nécessitent en général plus de temps que les 4 heures de « présence » allouées pour le TP.

Pendant les séances vous devez solliciter au maximum l'aide de l'intervenant pour prendre en main les outils en environnement de travail (messages du compilateur, de l'éditeur de lien, éventuellement du système à l'exécution, debugger) et vous faire expliquer des points de cours mal vus ou mal compris.

Je m'efforce **ne pas répondre** aux questions portant sur **du code** (du genre, « m'sieu (m'dam) ça marche pas »), si vous ne pouvez par fournir pas en même temps un algo sous forme « papier » sur lequel on peut discuter.

Objectif et évaluation de ce TP n°1 :

L'objectif de ce TP et de prendre contact avec le langage C et des outils de base (édition, compilation, make). En gros, vous devrez avoir acquis une bonne connaissance des points suivants :

- les commandes de compilation, d'édition de lien.
- la déclaration de variables et la déclaration de fonction/procédures (prototypes), l'utilisation de fichiers d'" en-têtes " (" .h ").
- la traduction en " C " des structures algorithmiques de base (*Si*, *Tantque*, *Selon*) (En gros les 2 premiers chapitres du poly " le langage C " de Cassagne)
- l'utilisation de la fonction/procédure " `printf` " (voir sujet du TD1 et/ou poly Cassagne)
- la création d'un « Makefile » simple.

Je fournis outre ce sujet et divers fichiers, un résumé de cours concernant les commandes de compilation et outils que vous allez utiliser pour les 2 premiers TP. L'intégralité de ce résumé de cours fait partie des connaissances minimales à savoir que je peux demander en évaluation.

Pour le compte-rendu, je demande uniquement les codes sources commentés de la partie 3 !!!

Partie 1 : Première compilation/exécution

Reportez-vous au résumé de cours, qui contient le code et les commandes à effectuer pour les 2 premières parties !!!

Vous allez écrire un programme C qui affiche dans la fenêtre d'entrée/sortie (dans une fenêtre de type " terminal ") le message suivant :

```
Esisar
bonjour !
```

1 édition du programme

→ 1 : Récupérez les fichiers « `bonjour.c` », « `bonjour2.c` », « `util.c` », « `util.h` ».

→ 2 : A l'aide d'un éditeur de texte (je vous conseille " `jgrasp` "), modifiez le message dans le "code source " du programme du fichier " `bonjour.c` ".

2 compilation, exécution

→ 3 : Compilez le programme à l'aide de la commande qui suit, et vérifiez le résultat :

```
$ gcc –g –Wall –pedantic bonjour.c –o bonjour
```

→ 4 : Vous pouvez maintenant exécuter votre programme, et en voir le résultat en tapant :

```
$ ./bonjour
```

→ 5 : Recommencez la manip en changeant le nom de l'exécutable (lors de la compilation, pas en renommant l'exécutable précédent !)

Partie 2 : Compilation séparée

→ 6 : Dès que des programmes complets deviennent importants (très vite !), il s'avère indispensable de séparer le code source dans plusieurs fichiers : ici on va en utiliser 3 : " `util.c` " pour la **définition** de la fonction " `dialogue` ", " `util.h` " pour la **déclaration** (de référence, ou *signature* ou *prototype*) de la fonction " `dialogue` " et " `bonjour2.c` " pour la définition de la fonction " `main` ".(voir support de cours !)

→ 7 : A l'aide de l'éditeur de texte, modifiez comme précédemment le message à afficher dans le « bon » fichier.

→ 8 : Créez les objets à l'aide des deux commandes :

```
$ cc -c -g -Wall -pedantic util.c
$ cc -c -g -Wall -pedantic bonjour2.c
```

Le compilateur va générer les fichiers objets `util.o` et `bonjour2.o`. Il faudra alors reconstituer l'exécutable (appelé cette fois " `bonjour2` ") par :

```
$ cc bonjour2.o util.o -o bonjour2
```

→ 9 : Vérifiez que l'exécution de " `bonjour2`" est correcte.

→ 10 : Modifiez de nouveau le message que vous allez afficher dans " `util.c` ", puis faites **uniquement** le nécessaire pour exécuter de nouveau votre programme (compiler uniquement le bon fichier et refaire l'édition de lien) :

```
$ cc -c -g -Wall -pedantic util.c
$ cc bonjour2.o util.o -o bonjour2
```

→ 11 : Modifiez le **nom** de la procédure « `dialogue` » (pour la nommer par exemple « `afficheMessage` ») puis faites le nécessaire pour exécuter de nouveau votre programme.

→ 12 : Modifiez le **nom** du fichier d'en-tête « `util.h` » (pour le nommer par exemple « `monEntete.h` ») puis faites le nécessaire pour exécuter de nouveau votre programme.

Partie 3 : Premier Menu, traduction algo →C

On veut créer un programme qui propose à l'opérateur une liste de 7 choix possibles : `s`, `r`, `b`, `h`, `j`, `c`, `q`, et *selon* le choix de l'opérateur va lancer :

- Sur choix " `s` " : l'affichage des NBTERMES (= 10) premiers termes de la suite définie par :
  $u_0 = -4$ et $u_{n+1} = 2u_n - 1$
- Sur choix " `r` " : l'affichage des NBTERMES (= 10) premiers termes de la suite définie par :
  $u_0 = 10$ et $u_{n+1} = -1{,}3u_n + 5$
- Sur choix " `b` " : l'affichage du caractère bissextile ou non d'une année dont le millésime est 'demandé à l'opérateur (exercice de TD1, exo 4)
- Sur choix " `h` " : l'affichage de *n* messages " hello ", le *n* étant saisi par l'opérateur dans l'intervalle [[0 ; 20]] , par appel à une procédure recevant le nombre *n* en paramètre.
- Sur choix " `j` " : le jeu de recherche du « nombre secret » : le programme choisit un nombre aléatoire entre 0 et 100, et l'opérateur doit le trouver par des essais, le programme répond par « c'est plus », « c'est moins » ou « trouvé ».
- Sur choix " `c` " : Une conversion francs/euros de 0 à 1000F par pas de 100F (0F valent 0 euros, 100F valent 15,24 euros, etc…jusqu'à 1000F inclus. Rappel : 1€ = 6.55957F)
- Sur choix " `q` " : afficher le message " AU revoir et merci ! "

Je fournis :

- " `menu.c` ", fichier " source C " contenant une ébauche de menu.
- " `menuUtil.h` ", fichier " en-tête " (" header ") contenant quelques prototypes de fonctions et procédures spécifiques appelées par la fonction principale du menu.
- " `menuUtil.c` ", fichier " source C " contenant un squelette des-dites fonctions et procédures spécifiques.
- " `testUnitaires.c` ", fichier " source C " contenant des procédures de tests pour la (seule) fonction (« `estBissextile` ») que vous allez écrire.

→ 13 : Compilez " `menu.c` ", pour créer un exécutable nommé " `menu` ", et essayez le résultat (seuls les choix '`s`' et '`q`' fonctionnent, et la saisie du choix '`b`'). Etudiez le code fourni.

→ 14 : Le " `menuUtil.c` ", que je fournis un morceau de code permettant de réaliser la procédure " `afficheSuite()` ", mais j'y ai glissé un certain nombre d'erreurs de syntaxe. Essayez de compiler ce " `menuUtil.c` " (pour obtenir « `menuUtil.o` »), observez les messages d'erreur du compilateur, et corrigez les erreurs (l'objectif est d'apprendre à comprendre et utiliser les messages du compilateur). Une fois que cela est réalisé, modifiez « `menu.c` » pour appeler la procédure, compilez, effectuez le lien avec « `menuUtil.o` » pour constituer l'exécutable et vérifiez que le premier choix fonctionne (Il se peut qu'il reste des erreurs qui ne sont pas des erreurs « uniquement » de syntaxe !!!)

Vous avez constaté qu'on peut être amené à recompiler un certain nombre de fois un fichier de code source, sans avoir à recompiler les autres dichiers. Pour simplifier et automatiser la reconstruction de votre exécutable, vous avez intérêt à écrire un « `Makefile`». Reportez-vous au support de cours fourni.

→ 15 : Ecrire votre « `Makefile`», et vérifiez qu'il fonctionne correctement. (La modification d'un fichier provoque les recompilations et liens **nécessaires et uniquement ceux qui sont nécessaires** !!)

→ 16 : Modifiez la procédure « `afficheSuite` » pour utiliser l'instruction C " `for` " au lieu du " `while` ", recréez l'exécutable, et vérifiez les résultats.

→ 17 : Complétez la fonction " `estBissextile` ", compilez pour obtenir le " `menuUtil.o` ", puis effectuez le lien avec " `testUnitaires.o` " obtenu par compilation de " `testUnitaires.c` " pour créer un exécutable" `testUnitaires` " et vérifiez que votre fonction est bien écrite en exécutant le test. Pour les recompilations, créez une cible « `tests` » dans le « `Makefile` » !

→ 18 : Modifiez le « `menu.c` » et recréez l'exécutable " `menu` " pour que le choix '`b`' fonctionne correctement. La saisie de l'année sera effectuée dans le « `main` ». Inspirez-vous de ce qui est fourni.

→ 19 : Intégration du choix '`r`' :
Quelle est la différence fondamentale entre la suite du choix '`s`' et celle du choix '`r`'?
Rajoutez dans " `menuUtil.h` " la déclaration d'une procédure " `afficheSuite2`", dont le prototype est : " `void afficheSuite2(void) ;` ", et que vous *définirez* (= « dont vous écrirez le code source ») dans le fichier " `menuUtil.c` ".
Modifiez le menu en conséquence, compilez, faites l'édition de lien, exécutez.

→ 20 : Intégration du choix '`h`' :
Rajoutez dans " `menuUtil.h` " la déclaration d'une procédure " `afficheNHello`" que vous commenterez, dont le prototype est : " `void afficheNHello(int n) ;` ", et que vous définirez également dans le fichier " `menuUtil.c` ".
Modifiez le menu en conséquence, compilez, faites l'édition de lien, exécutez.

→ 21 : Intégration du choix '`c`' :
Procédez comme aux questions précédentes pour intégrer la procédure " `afficheConversion`" de prototype : " `void afficheConversion(void) ;` "

→ 22 : Intégration du choix '`j`' :
Procédez comme aux questions précédentes pour intégrer la procédure " `chercheNombre`" de prototype : " `void chercheNombre(void) ;` "
Pour les fonctions permettant de générer des nombres aléatoires :

```
"man rand", "man srand", "man time"
```

En gros :

- la fonction "`rand()`" génère une suite pseudo-aléatoire de nombres entiers.
- la fonction "`srand()`" permet de modifier la valeur initiale de cette suite. Elle a besoin pour cela qu'on lui fournisse la-dite valeur (seed =racine). Un moyen est de faire appel à la fonction système "time()" qui rend un nombre de millisecondes depuis une date fixe.

Un exemple d'utilisation "typique" sera alors :

```
#include <time.h>
#include <stdlib.h>
......
srand(time(NULL)) ;
nbAlea = rand() ; /* attention à l'intervalle désiré */
```