Les premiers pas en C

Objectif

Ce TP est consacré à la découverte d'un compilateur C et à l'écriture de programmes simples pour se familiariser avec la syntaxe du langage et les entrées/sorties.

1 Le compilateur C

La commande `gcc` est la commande de base enchaînant automatiquement l'appel de différents outils réalisant la traduction d'un programme source en un binaire exécutable. Les phases successives sont :

- la précompilation réalisée par le préprocesseur `cpp` qui traite un certain nombre de directives telles que l'inclusion des fichiers. Par exemple, `#include <stdio.h>` sera traité en remplaçant la ligne correspondante dans le programme par l'ensemble des lignes du fichier `/usr/include/stdio.h` (Le fichier `stdio.h` contient entre autres les entêtes des fonctions d'entrée-sortie) ;
- la compilation proprement dite qui traduit un programme source en un module objet ré-éditable ;
- l'édition des liens qui rassemble les différents fichiers binaires et extrait des bibliothèques les fonctions utilisées (en particulier la bibliothèque standard `libc` du langage C). Cette phase doit produire un fichier binaire exécutable (si c'est possible) appelé `a.out` par défaut.

La forme générale de la commande est :
`gcc [-`*options*`]` <*noms de fichier*> ...

Les noms des fichiers paramètres de la commande peuvent avoir différentes extensions (suffixes) qui déterminent à partir de quelle phase du processus décrit ci-dessus elles doivent être traitées :

- l'extension `.c` indique un fichier source à soumettre à toutes les phases ;
- l'extension `.o` caractérise un module objet auquel ne sera appliquée que la phase d'édition des liens.

Les principales options de la commande `gcc` permettent de modifier l'enchaînement ou le résultat des opérations effectuées par le compilateur :

`-c` : cette option permet de stopper la phase de traduction après la compilation (avant l'édition des liens), en générant un fichier d'extension `.o` à partir de chaque fichier paramètre d'extension `.c` avec le même nom de base. Par exemple, l'exécution de la commande `gcc -c prog.c` aboutit à la création du module objet `prog.o` ;

`-o` : cette option permet de spécifier le nom de l'exécutable qui sera généré si c'est possible (en remplacement du nom par défaut `a.out`). Par exemple : l'exécution de la commande

```
gcc prog.o -o prog
```

aboutit à la génération d'un binaire exécutable de nom `prog` à partir du module objet `prog.o` ;

`-Wall` : cette option permet d'obtenir un certain nombre de *warnings* durant la compilation. Le compilateur signale ainsi les actions les plus "suspectes", susceptibles d'engendrer une erreur

durant l'exécution. Cette option devra toujours être positionnée. **De plus votre compilation sera considérée comme correcte par votre enseignant quand vous n'aurez plus de *warnings*** ;

-pedantic : cette option permet une vérification stricte de la norme ISO C (par ex., l'interdiction de mélanger déclarations et instructions).

Pour forcer certaines options par défaut, vous pouvez par exemple définir dans votre fichier `.bashrc` un *alias*, en y ajoutant la ligne de commande suivante :

```
alias gcc='gcc -Wall -pedantic'
```

Ainsi, tout appel à `gcc` sera remplacé par un appel à `gcc -Wall -pedantic`. Vous pouvez vérifier que cet alias est pris en compte en tapant la commande `alias`.

Attention : Enfin, n'oubliez pas de lire les messages d'erreur et les avertissements du compilateur. Ces informations, associées à une réflexion personnelle encore et toujours nécessaire, constituent le seul moyen de corriger les erreurs commises, ainsi que de comprendre le bon ou mauvais fonctionnement des programmes que vous écrirez.

1.1 Premier programme

Écrire le programme suivant dans le fichier bonjour.c :

```
#include <stdio.h>
/*                                                    */
/* programme simple affichant "Bonjour tout le monde !" */
/*                                                    */
int main() {

  printf("Bonjour tout le monde !\n");

}
```

1. compiler le programme en tapant les commandes :
   ```
   gcc -c bonjour.c
   gcc bonjour.o -o bonjour
   ```
2. corriger le programme pour ne plus avoir de *warnings*

2 Exercices de familiarisation avec C

2.1 Puissance

- écrire une fonction qui calcule x^n (version impérative), $x \in \mathbb{R}$, $n \in \mathbb{N}$;
- écrire une fonction qui calcule x^n (version récursive),
- tester ces deux fonctions avec x et n lus au clavier (voir annexe pour les formats de lecture et d'écriture).

2.2 Sous programme de recherche du maximum d'un tableau

Compléter le code fourni `manip_tableau.c` par l'implantation et l'utilisation de la fonction de recherche du maximum d'un tableau d'entier.

```
#include <stdio.h>

/* Prototype des sous-programmes */

/* fonction max :
   renvoie l'indice de la valeur maximum d'un tableau d'entiers
   paramètres :
     v : le tableau (D)
     n : le nombre d'élément du tableau (D)
   retour :
     l'indice de la valeur maximum
   pré : le tableau n'est pas vide (n>0)
   post : quelquesoit i dans {0..n-1}, v(max(v,n)) >= v(i)
  ---------------------------------------------------------- */

int max(int v[], int n);

/*                                                            */
/* programme qui affiche le maximum d'un tableau              */
/*                                                            */
int main(){

  /* dimension du tableau */
  int t[] = {12, 1, 0, 2, -6, 14, 9, 11, 2, 5, 1, 0, 1, 2, 1, 9, 67};
  const long n = sizeof(t)/sizeof(int);

  /* à compléter par l'utilisation de la fonction max */

  return 0;
}

/* à compléter par l'implantation de la fonction max */
```

2.3 Recherche d'une valeur dans un tableau

Écrire un sous-programme qui recherche l'indice de la première occurrence d'une valeur x dans un tableau d'entiers de taille n (vous pouvez utiliser le même fichier que dans l'exercice précédent).

2.4 Recherche de la *keme* occurrence d'une valeur dans un tableau

Écrire un sous-programme qui recherche l'indice de la *keme* occurrence d'une valeur x dans un tableau d'entiers de taille n.

Remarque

Pour ces deux derniers exercices, n'oubliez pas de compléter la spécification en explicitant, par exemple, ce qui se passe si la valeur cherchée (ou la *keme* occurrence) n'existe pas.

3 Annexe : Formats pour les Entrées/Sorties

3.1 printf

```
int printf(const char * format, ...);
```

L'argument format est une chaîne de caractères qui détermine ce qui sera affiché par `printf` et sous quelle forme. Cette chaîne est composée de texte « normal » et de séquences de contrôle permettant d'inclure des variables dans la sortie. Les séquences de contrôle commencent par le caractère « % » suivi d'un caractère parmi :

- `d` ou `i` pour afficher un entier signé au format décimal (int) ;
- `f` pour afficher un réel (float/double) avec une précision fixe ;
- `e` pour afficher un réel (float/double) en notation scientifique ;
- `c` pour afficher en tant que caractère ;
- `s` pour afficher une chaine de caractère C standard ;
- `p` pour afficher la valeur d'un pointeur, généralement sous forme hexadécimale.

3.2 scanf

```
int scanf(const char * format, ...);
```

- `d` extrait un nombre décimal signé de type int, ignorant les espaces se trouvant éventuellement avant le nombre ;
- `f` extrait un nombre réel, en sautant les blancs, de type float ;
- `lf` extrait un nombre réel, en sautant les blancs, de type double ;
- `c` lit un caractère (de type char), y compris un blanc ;
- `s` : extrait la chaîne de caractères, en ignorant les blancs initiaux et ce jusqu'au prochain blanc. L'argument correspondant doit être de type char * et pointer vers un bloc mémoire suffisamment grand pour contenir la chaîne et son caractère terminal.

`scanf` renvoie le nombre d'items effectivement lus.