TD2 Le compilateur gcc La compilation séparée

Durée : 1 heure.

Exercice 1 – La bibliothèque mathématique /lib/libm.so

Il s'agit ici d'obtenir une table des cosinus des angles de 0 à π par pas de π/10.

Contraintes :

- Le fichier source sera nommé *table.c*
- Le fichier exécutable sera nommé *table_cosinus*

Un exemple d'exécution peut être le suivant :

```
$ ./table_cosinus
 cos(0.000000)=1.000000
 cos(0.314159)=0.951057
 cos(0.628319)=0.809017
 cos(0.942478)=0.587785
 cos(1.256637)=0.309017
 cos(1.570796)=0.000000
 cos(1.884956)=-0.309017
 cos(2.199115)=-0.587785
 cos(2.513274)=-0.809017
 cos(2.827433)=-0.951057
 cos(3.141593)=-1.000000
$
```

1. En étudiant le manuel (commande Unix *man*) à propos de la fonction du langage C *cos*, indiquer ci dessous le fichier d'entête à inclure pour pouvoir utiliser cette fonction *cos* et compléter le prototype de la fonction.

```
#include...
...      cos(...            );
```

2. Étudier le fichier d'en-tête *math.h* pour déterminer la valeur de la constante représentant π.
   Astuce : utiliser le filtre grep et la valeur 3.14

   Constante égale à π :

3. Rédiger le fichier source *table.c*.
4. Donner la commande *gcc* permettant la construction de l'exécutable *table_cosinus*

   Commande :

Exercice 2 – Compilation séparée

→ La théorie

Il est courant que le code source d'un exécutable soit réparti dans plusieurs fichiers :

- soit parce que le projet implique plusieurs développeurs, chacun travaillant sur son propre fichier source
- soit parce qu'on souhaite utiliser un code source écrit par un autre programmeur.

Un exemple. Un fichier source référencé *max.c* contient le code d'une fonction prenant en argument deux entiers et retournant le plus grand des deux.

```
// Fonction max
// Paramètres d'entrée : deux entiers a et b
// Valeur de retour : le plus grand de ces deux entiers
int max(int a, int b)
{
  if(a > b) return(a);
  else return(b);
}
```

Fichier source « max.c »

→ On peut remarquer que ce fichier ne peut pas produire seul un exécutable puisqu'il ne contient pas de fonction *main*.

Un programmeur veut utiliser la fonction *max* précédente dans son programme appelé *prog.c.* Comment faire ? Deux étapes sont à respecter :

- **Etape 1 : faire figurer le prototype des fonctions externes dans le code source du programme appelant.**

```
#include<stdio.h>
#include<stdlib.h>

int max(int, int);     // Ceci est l'étape 1

int main()
{
  int x, y;
  scanf("%d", &x);
  scanf("%d", &y);
  printf("Le max est %d\n", max(x,y));      utilisation de la fonction max
  exit(0);
}
```

Fichier source « prog.c »

- **Etape 2 : indiquer à *gcc* les fichiers source nécessaires à l'obtention de l'exécutable final.**

On peut utiliser directement les fichiers source « point c » ...

gcc max.c prog.c -o prog

… ou alors les modules objet « point o » s'ils existent :

gcc max.o prog.o -o prog

→ Exercice

Julie et Pierre souhaitent réaliser un logiciel qui indique si une année est bissextile.

Julie rédige dans un fichier source nommé *julie.c* une fonction *isBissextile* :

- admettant en paramètre d'entrée : un entier *a*
- retournant en sortie : 1 si *a* est bissextile, 0 sinon.

Parallèlement, Pierre rédige la fonction principale *main* dans un fichier source référencé *pierre.c*. Cette fonction principale demande à l'utilisateur de saisir une année et indique si cette année est bissextile en utilisant la fonction mise au point par Julie dans *julie.c*.

1. Comme Julie, rédiger le fichier source *julie.c*
2. Comme Pierre, rédiger le fichier source *pierre.c*
3. Bâtir l'exécutable final nommé *bissextile*.

Commande :

Rappels.
Depuis l'ajustement du calendrier grégorien, sont bissextiles les années :

- *soit divisibles par 4 mais non divisibles par 100*
- *soit divisibles par 400*

Ainsi :

- *2012 est bissextile car divisible par 4 et non divisible par 100*
- *1900 n'est pas bissextile car divisible par 4 mais également divisible par 100 ; et non divisible par 400*
- *2000 est bissextile car certes divisible par 100, mais aussi divisible par 400*

TD3 L'interface shell

Durée : 1 heure.

Exercice 1 – Passage d'argument et accès à l'environnement

L'en-tête de commande *int main(int argn, char *argv[], char *arge[])* sera utilisé pour rédiger un programme appelé *interface* qui affiche sur la sortie standard le nom de la commande, les arguments passés sur la ligne de commande et les variables d'environnement (noms et contenus).
La valeur du code de retour sera le nombre d'arguments passés sur la ligne de commande, sans compter le nom de la commande elle-même. On obtiendra un affichage du type :

```
$ ./interface oui non
 Nombre d'arguments : 2
 argv[0] = interface
 argv[1] = oui
 argv[2] = non
 arge[0] = LANG=Fr
 …
$ echo $?      ← Affichage du code de retour de la commande.
 2
$
```

Question 1. Rédiger ce programme.
Question 2. Que penser de ce code de retour ?

Exercice 2 – Travail personnel : la fonction getenv

Ce programme, qui est à réaliser en dehors des séances TD, est facultatif et constitue un approfondissement de ce qui a été vu en cours. N'hésitez pas à le réaliser. La correction sera disponible sur l'ENT.

Rédiger un programme appelé *ckoidonc* prenant en paramètre le nom d'une variable d'environnement et affichant sur *stdout* sa valeur si elle existe. Utiliser pour cela la fonction *getenv* de la bibliothèque standard du langage C, qui constitue une alternative au tableau *arge* vu en cours. On obtiendra un affichage du type :

```
$ ./ckoidonc HOME
 La variable d'environnement HOME vaut /home/gaston
$ ./ckoidonc toto
 toto : variable inconnue
$ ./ckoidonc
 ckoidonc : un argument requis
$
```