

## 0.1 Création d'une bibliothèque

Je vais ici simplement me contenter de reprendre l'exemple de l'introduction et réécrire le programme cette fois en utilisant une bibliothèque statique ou dynamique. Ce qui sera écrit restera vrai pour tout programme.

### 0.1.1 1. Création des fichiers composant la librairie :

Le premier fichier à créer est celui de la déclaration des fonctions et sera identique à ceci :

```
/*Dans ce fichier on ne fait que déclarer nos fonctions
   (sauf la fonction principale main évidemment) */
/*Ces trois lignes en rouges permettent à ce que le fichier
   .h ne soit utilisé qu'une fois (ce qui est suffisant)
   */
#ifndef _ENTETE_
#define _ENTETE_
int arr(double) ;
void releve (char a[128], double *z, double *e ,int *r) ;
int entree (void) ;
void calcul(double *a , double z , double e , double *r
            , int *t ) ;
void sortie (double a, int b, int flag) ;
#endif
```

Nous enregistrons ce fichier sous le nom par exemple **entete.h**. Ce qui est important ici est l'extension **.h**.

Explication sur **#ifndef \_ENTETE\_ :** si la variable **\_ENTETE\_** n'existe pas alors elle sera créée à l'aide de **#define \_ENTETE\_**, puis on continuera à exécuter la suite c'est à dire la déclaration des fonctions jusque **#endif**. Au contraire si la variable existe déjà (au delà d'une compilation) alors l'exécution s'arrêtera puisque qu'il n'y a aucune commande après **#endif**. Cela permet simplement de ne pas compiler plusieurs fois le même fichier s'il a été inclus plusieurs fois.

Nous allons maintenant créer le fichier où les fonctions seront écrites. Il ressemblera simplement à cela :

```
/*Dans ce fichier nous écrivons nos fonctions (sauf main :-))
   */
/*Il faut inclure le fichier entete.h avec : */
#include "entete.h"
/*Cette constante n'est pas très jolie dans une librairie
   mais pas le courage de tout réécrire :- ) : */
#define zero -273.15
int arr(double a)
{
return ( (int) a + 0.5 ) ;
}
```

```
int entree (void)
{int resu;
char r[20];
scanf("%s",r);
if (*r=='o') return(1);
else if (*r=='n') return(0);
else
{printf ("Vous avez saisi un mauvais caractère.\nRecommencez
s'il vous plait.\n");
return (entree ());
}
}

void releve (char l[128], double *a1, double *a2,int *b)
{char repoui='o',repon='n',rep;
int a, flag = 1;
printf ("Saisie des relevés de températures. Voulez-vous
continuez?\n");
printf ("%c/%c\n",repoui,repnon);
*b = entree();
while (*b==1 && flag == 1){printf ("Donnez un relevé de
température\nLieu :\n");
scanf ("%s",l);
printf ("Donnez la température maximale :\n");
scanf ("%lf",a1);
printf ("Donez la température minimale :\n");
scanf ("%lf",a2);
if (*a1 < *a2 )
printf ("Vous donnez une température maximale inférieure
à la température minimale.\nChercheriez-vous l'écran
bleu?\n");
else flag = 0;
if (*a1 < zero || *a2 < zero)
printf ("Changez de thermomètre : vous êtes en-dessous
du zéro absolu!\n");
else flag = 0;
}
}

void calcul(double *Max,double tempmax, double tempmin,
double *ecart, int *N)
{if ( *Max < tempmax ){
*Max = tempmax;
*N = 1;
}
```

```

}
else if ( *Max == tempmax) {++*N;}
else if ( *Max > tempmax) {/*Rien à faire*/}
if ( *ecart < (tempmax - tempmin) ) *ecart= tempmax -
    tempmin;
}
void sortie (double a, int b, int flag)
{if ( flag == 1 ){printf("Ecart de températures maximal :\n%f\n",a) ;
printf("Arrondi de l'écart maximal de températures\n%d\n",arr(a)) ;
printf("Nombre de lieu ayant observé la température maximale :\n%d\n",b) ;
}
else printf ("Pas de relevé.\n") ;
}
}

```

Cette fois nous enregistrons ce programme avec le nom **corps.c**. Comme précédemment ce qui est avant tout important est l'extension **.c** que porte le nom de ce fichier.

La l'utilisation des guillemets dans **#include "entete.h"** signifie que nous donnons le chemin relatif entre le fichier incluant et le fichier inclus. Nous pourrions placer le fichier dans le répertoire où le compilateur s'attendra à le trouver (par exemple /usr/include) et en ce cas nous donnerons le chemin relatif à ce répertoire en utilisant la syntaxe : **#include <entete.h>**.

Maintenant il ne nous reste plus qu'à écrire le programme proprement dit :

```

#include <stdio.h>
#include "entete.h"
#define zero -273.15
int main(void)
{double TMax = zero, ecartmax = -1, tempmax, tempmin;
int nblieumax = 0, drap = 1, k = 0;
char lieu[128];
printf ("Ce programme calcul l'écart entre la température
    maximale et la température minimale pour une ville donnée
    et renvoie l'écart maximal observé sur l'ensemble des
    villes observées.\n
Il renvoie aussi le nombre de lieu ayant observée la température
    maximale.\n
Les températures doivent être données en °C.\n");
while (drap == 1){
++k;
releve (lieu, &tempmax, &tempmin, &drap) ;
if ( drap == 1)
calcul (&TMax, tempmax, tempmin, &ecartmax, &nblieumax) ;
}
if (k != 1){drap = 1;
sortie(ecartmax, nblieumax, drap) ;
}
}

```

```
    }  
    else  
    {drap = 0 ;  
    sortie (ecartmax, nblieumax, drap) ;  
    }  
}
```

que nous pouvons appeler **temperature.c** par exemple.

Pour ce qui est de la rédaction des fichiers nous avons terminé. Il ne reste plus qu'à compiler, et comme dit initialement : c'est la compilation qui fait que nous aurons une librairie statique ou bien une librairie dynamique.

### 0.1.2 2. Compilation pour une librairie statique :

Allons droit dans le vif du sujet : ouvrez un terminal et placez-vous dans le répertoire courant où sont rangés vos trois fichiers. Maintenant il ne vous reste plus qu'à saisir cela :

```
gcc -c temperature.c  
gcc -c corps.c  
ar -q corps.a corps.o  
gcc -o temperature temperature.c corps.a  
./temperature
```

Allons y pour les explications !

La commande **gcc -c** permet de compiler sans créer de lien ; autrement dit lorsque nous compilons **temperature.c**, nous ne touchons pas à **corps.c** bien que celui-ci soit inclus (**#include**). La ligne de commande **gcc -c temperature.c** va générer un fichier **temperature.o**, de même nous allons créer un fichier **corps.o** avec la seconde ligne de commande.

**ar** permet de créer, modifier ou extraire des archives. Ainsi la commande **ar -q corps.a corps.o** permet de créer l'archive **corps.a** (option **-q** : concaténation rapide). Et évidemment cette archive n'est en réalité rien d'autre que notre librairie... Petite précision : si je voulais ajouter un fichier **corps2.o** à ma librairie, il me faudrait utiliser la syntaxe suivante de **ar** : **ar -qa corps.o corps.a corps2.o** Comme vous l'avez compris il faut spécifier lors d'ajout à **ar** après quel fichier dans l'archive il doit écrire.

Et enfin la dernière ligne **gcc -o temperature temperature.c corps.a** permet de réaliser les liens (entre le programme et la bibliothèque statique). En réalité, c'est le programme **ld** qui crée les liens mais **gcc** y fait appel. Dans ce cas, la librairie est incluse dans le programme lors de la compilation d'où le nom de **librairie statique**. A noter que l'option **-o** de **gcc** permet seulement de spécifier le nom du programme, ici **temperature**.

Une petite remarque au passage : nous avons utilisé le nom de **corps.a** pour l'archive. Cela n'est pas une obligation mais par convention le **.a** est utilisé pour désigner les librairies statiques.

### 0.1.3 3. Compilation pour une librairie dynamique :

Ici il va y avoir quelques petits changements, je vous laisse les découvrir :

```
gcc -c temperature.c  
gcc -c corps.c
```

```
gcc -o corps.so -shared corps.o
gcc -o temperature temperature.o corps.so
export LD_LIBRARY_PATH=. :$LD_LIBRARY_PATH
./temperature
```

Les deux premières lignes sont identiques au cas de la librairie statiques donc sans commentaire.

L'option `-shared` de la troisième ligne permet de produire **un objet partagé qui peut être lié avec d'autres objets pour former un exécutable**, comme indiqué dans le man. Autrement dit, c'est par cette ligne que nous créons notre **bibliothèque dynamique**.

La ligne **`gcc -o temperature temperature.o corps.so`** permet de créer l'exécutable en lui disant de faire appel à notre **librairie dynamique corps.so**. Celle-ci sera incluse à l'exécutable qu'au moment de l'exécution.

**`export LD_LIBRARY_PATH=. :$LD_LIBRARY_PATH`** : et oui car autrement lors de l'exécution, le programme ne trouvera pas la bibliothèque :-). En fait les librairies dynamiques sont rangées dans des répertoires spécifiques. Lors de l'exécution d'un programme faisant appel à une librairie dynamique, ce sont ces répertoires qui sont regardés. Cette ligne permet de rajouter le répertoire courant où se trouve notre librairie dynamique au **PATH** des librairies dynamiques. En tapant : **`more /etc/ld.so.conf`** vous pourrez connaître les répertoires où sont rangées vos librairies dynamiques de votre système. Ainsi une autre solution serait de rajouter votre librairie dans l'un de ces répertoires avec pour commande : **`cp corps.so /un_répertoire_de_/etc/ld.so.conf`**.

Et finalement nous pouvons exécuter le programme.

Une nouvelle fois **`.so`** n'est pas obligatoire mais est la convention pour désigner les librairies dynamiques. Une page de Léa qui m'a aidé et que je vous recommande : [ICI](#)<sup>1</sup>.

«« Précédent<sup>2</sup> Suivant »»<sup>3</sup>

---

<sup>1</sup> <http://www.lea-linux.org/documentations/index.php/Dev-libc>

<sup>2</sup> <http://www.trustonme.net/didactels/153.html>

<sup>3</sup> <http://www.trustonme.net/didactels/155.html>