LE1 IAP **TP n°6** **2024-2025**

Traitement de chaînes de caractères

Remarque : Ce TP se déroulera en 4 parties. Les 3 premières parties sont à réaliser en séance et seront notées sur 15 pts. La 4ème partie est le compte-rendu qui est à réaliser au plus tard 7 jours après la séance. Elle sera notée sur 5pts.

Objectifs

- Synthèse des connaissances apprises en IAP
- Savoir utiliser les fonctions de manipulation des chaines de caractères définies dans la bibliothèque string.h

Compétences attendues

- Mise en œuvre d'une conception modulaire afin de favoriser la modularité
- Compilation séparée et écriture d'un makefile
- Faire des tests fonctionnels et savoir les analyser compte-tenu du cahier de charges fonctionnel
- Savoir-faire un rapport pour montrer le bon fonctionnement de l'application.

Rappels sur la compilation séparée

Les grands projets informatiques font appel à des équipes de développement. Cela amène naturellement à découper le développement en différents modules qui sont confiés à chaque membre de l'équipe. Cela amène à considérer 3 catégories de fichiers dans une application :

- le fichier de déclaration de la structure de données manipulées par le module que nous nommerons de manière générique **<modulei>.h**,
- le fichier de définition des traitements du module que nous nommerons **<modulei>.c**
- le fichier relatif au programme principal que nous désignerons **<principal>.c.**

Après l'édition de ces fichiers nous devons compiler <modulei>.c par la commande

pi@raspberrypi:~Documents/IAP1/TPX$ **gcc -c <modulei>.c**

Le résultat est le fichier <modulei>.o

Remarque : Bien faire attention à l'option de compilation **-c** qui limite l'exécution de gcc à mettre en œuvre les étapes de pré-processing et de compilation. Le fichier « .o » n'est généré que s'il n'y a pas d'erreurs

On peut ensuit construire l'application exécutable en utilisant les fichiers objets obtenus à partir de chaque module. On exécute alors la commande :

pi@raspberrypi:~Documents/IAP1/TP4 $ **gcc <modulei>.o … <modulek>.o <principal>.c -o <principal>.exe**

On notera l'utilisation de l'option **-o** qui demande à gcc d'éxécuter les 3 étapes de son

fonctionnement : pré-processing, compilation et édition de liens.

Remarque : On peut compiler le fichier <principal>.c avec la commande « gcc -c <principal>.c » avant l'option de l'ensemble des modules. Cela permet de corriger les erreurs de compilation. Mais ensuite, pour l'obtention du programme principal, il faut repartir du fichier source <principal>.c et non pas du fichier objet.

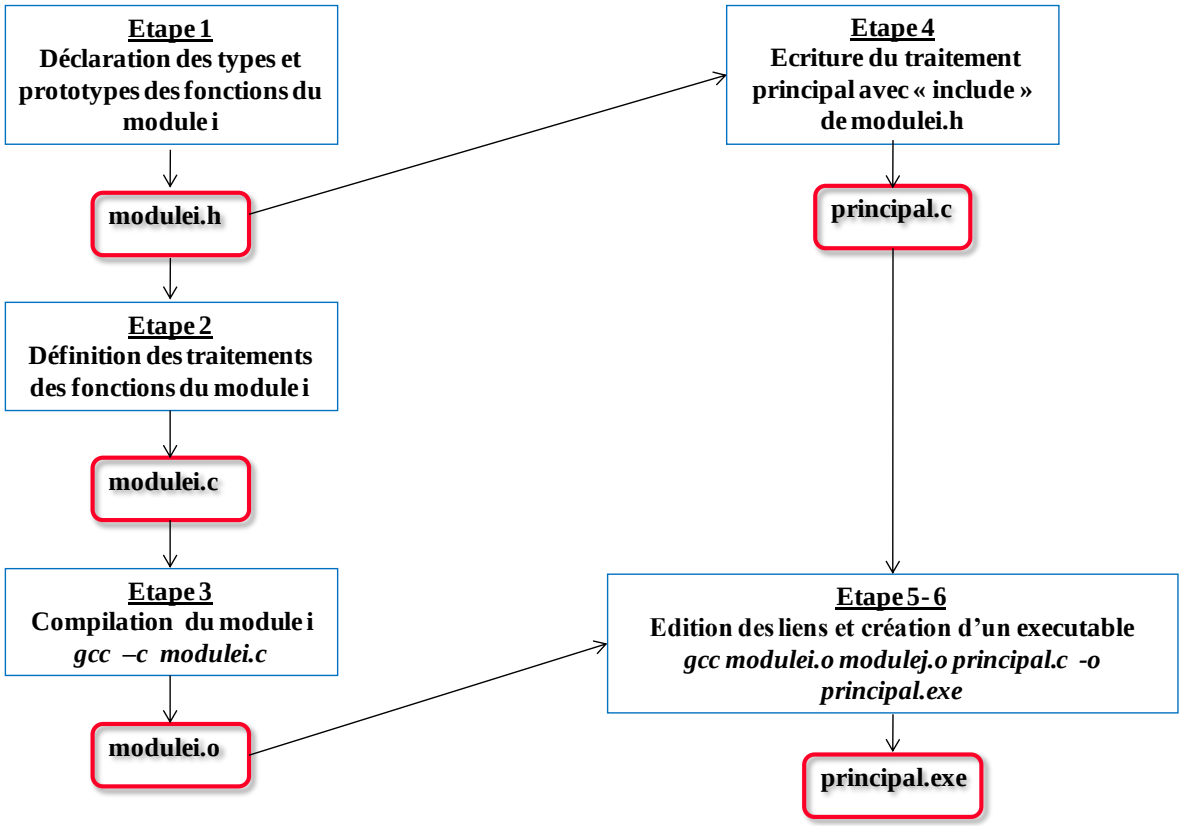

Figure 1. Principe de la compilation séparée

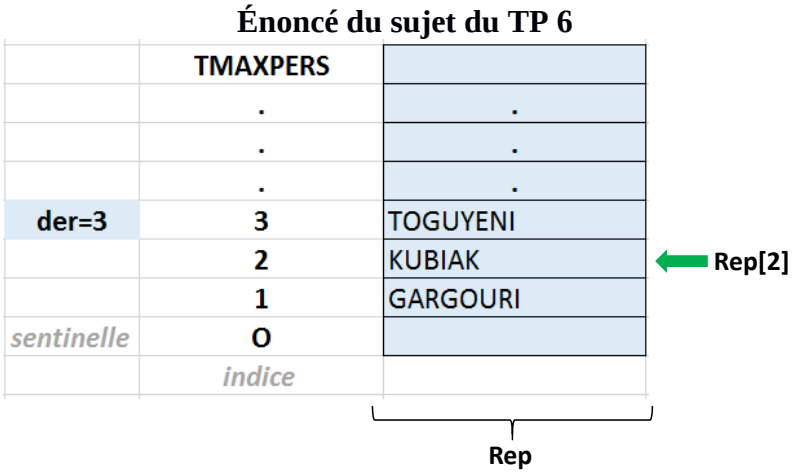

Figure 2. Structure de l'annuaire

Vous êtes chargez de développer un programme permettant de gérer un annuaire de clients (cf. Figure 2).
Cet annuaire doit permettre d'ajouter des personnes au fur et à mesure qu'elles s'inscrivent. Chaque personne est caractérisée par son nom. **Il ne doit pas y avoir deux fois la même personne dans l'annuaire (pas de doublons)**. Les recherches dans l'annuaire doivent être très rapides et faites en fonction du nom.
Le programme principal de cette application (cf. fichier binaire '**annuaire.o**') vous est donné (cf. annuaire.zip) afin de faire vos tests de validation de libannu.c.
Vous êtes chargé de développer et de tester les traitements suivants de la bibliothèque '**libannu.c'** que vous implémenterez.
Le type Tchaine est défini de la manière suivante :

```
#define MAXCAR 30
typedef char Tchaine[MAXCAR+1]; //Le +1 c'est pour le caractere '\0'
```

et le type Ttabpers permettant de déclarer des annuaires est défini par :

```
#define MAXPERS 10000
typedef Tchaine Ttabpers[MAXPERS+1]; //Le +1 c'est pour la sentinelle
```

Pour tester les traitements implémentés il faut compiler votre bibliothèque (**libannu.c**) avec le binaire du programme principal (**annuaire.o**).

Sachant que '***nom***' désigne le nom de la personne, '***Rep***' désigne l'annuaire et '***der***' la position de la dernière personne ajoutée dans le répertoire, le travail demandé est de :

Exercice 1
Réaliser un traitement permettant d'ajouter une nouvelle personne dans l'annuaire en conservant un annuaire trié. On utilisera pour cela l'ajout par insertion basée sur la technique de la sentinelle. La fonction retourne la position du nom ajouté dans le tableau ou 0 si l'ajout est impossible. Son prototype sera :

fonction insererpers(var Rep : Ttabpers, var der : 1..MAXPERS ; nom : Tchaine) : 0.. MAXPERS ;

Remarque : Chaque composante de Rep est une chaine de caractères !!! Ainsi Rep[1] est en fait une chaine de caractères.

Exercice 2
Réaliser un traitement permettant d'afficher l'ensemble des clients enregistrés à un moment donné dans l'annuaire.

procedure afficherrep(Rep : Ttabpers, der : 1..MAXPERS) ;

Exercice 3
Réaliser un traitement permettant de rechercher la position d'un client dans l'annuaire. S'il ne s'y trouve pas, la fonction renverra la valeur 0.

fonction chercherpers(Rep : Ttabpers, der : 1 .. MAXPERS, nom : Tchaine) :0.. MAXPERS ;

Exercice 4
Réaliser une fonction permettant de supprimer de l'annuaire l'un des clients. Le paramètre position correspond à la position dans Rep de la personne a supprimée. Si position ne permet pas la suppression, la fonction doit retourner 0 sinon elle retourne la position de l'élément supprimé.

fonction supprimerpers(var Rep : Ttabpers, var der : 1.. MAXPERS, position : 1.. MAXPERS) : 0 .. MAXPERS ;

Le pseudocode de cette fonction est le suivant :

```
fonction supprimerpers(var Repertoire :Tchaine ; var der :entier ; position : entier) ;
(* la paramètre formel 'position' désigne la position dans l'annuaire du client à supprimer *)
var i : entier ; (* compteur *)
    pos :entier ; (*mémoire*)
debut
si (position>der ou position <1) alors supprimerpers <- 0 ;
sinon
        debut
        pos <-position ;
        pour i allant de position à dernier-1 faire Repertoire[i]=Repertoire[i+1] ;
        dernier <- dernier -1 ;
        supprimerpers <- pos ;
        fin ; (* fin sinon)
fin_si ;
fin ; (* fin fonction *)
```

Exercice 5
Réaliser un makefile afin de produire l'exécutable en utilisant la programmation séparée.

Exercice 6
Déboguer les erreurs du fichier source du programme principal (cf. **annuaire.c**) afin qu'il fonctionne comme celui de '**annuaire.o**' qui vous a permis de tester les fonctions des exercices précédents. **Donner un jeu d'essais avec les tests qui vous ont permis de déboguer le programme.**

Exercice 7
Faire des tests de toutes les fonctions précédentes. ***Vous devez tester tous les cas d'utilisation de vos traitements afin de prouver qu'ils sont fonctionnels et robustes***. Vous récupérerez les copies d'écran dans le fichier 'testannu.txt' que vous joindrez dans l'archive à déposer sur moodle.

Exercice 8
Faire des tests de performances de votre application.