IAP1 Travaux Pratiques n°5 Année scolaire 2021-2022

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 avant la séance suivante selon les directives de votre enseignant. Elle sera notée sur 5 pts.

Objectifs

- Manipulation des chaînes de caractères avec des traitements premiers niveau et des traitements 2ème niveau
- 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.

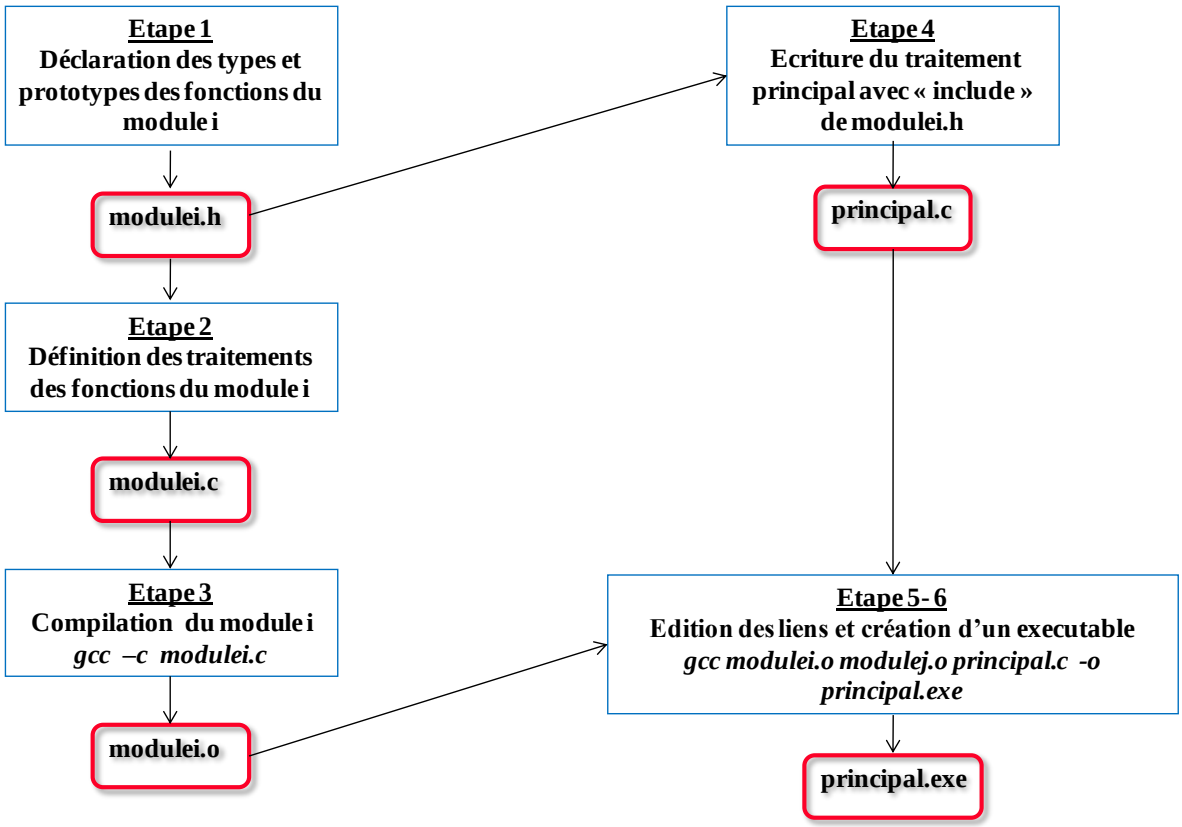

Figure 1. Principe de la compilation séparée

Énoncé du sujet du TP 5

On considère des chaînes d'au plus 80 caractères. Ces chaînes peuvent être :

- vides,
- composées d'un seul mot,
- composées de plusieurs mots, séparés chacun par un seul espace.

De plus ces chaines auront la particularité de ne jamais commencer ou finir par un espace. Les chaines peuvent être composées de plusieurs mots.

Exemple :

- L1= "" chaine vide

- L2= "exemple" 1 seul mot
- L3="titi et gros minet" plusieurs mots

Le type chaîne et des fonctions de 1er niveau (cf. TP4) sont définies par la bibliothèque chaine4. Le fichier header chaine4.h contient toutes les déclarations de constantes, de types et des prototypes de fonctions. Le code de ces fonctions vous est fourni par le fichier chaine4.o.

Partie I – Ecriture de macro-fonctions
En utilisant la bibliothèque « string.h », ecrire un fichier header nommé **chaine5.h** permettant de définir les macro-fonctions suivantes :

VIDE(ch)
Description : Elle permet de tester si une chaine ch est vide.

EGAL(ch1, ch2)
Description : Elle permet de comparer si les chaines ch1 et ch2 sont identiques.

Partie II – Bibliothèque de traitement de deuxième niveau
L'objectif de cette partie est d'apprendre à utiliser des fonctions appartenant à des bibliothèques **en faisant la différence entre fonctions de bibliothèques prédéfinies (« standards ») et fonctions de bibliothèques fournies par un autre utilisateur**.

En utilisant désormais uniquement les fonctions définies dans « chaine4.h » et les fonctions strlen, strcpy, strcmp, définir les fonctions suivantes que vous implémenterez sous la forme d'une bibliothèque. Vous compléterez le fichier « chaine5.h » de la partie 1 à l'aide des prototypes des fonctions définies dans cette partie.

Remarque **: Les exemples donnés ci-dessous sont du pseudo-code. Ils ne correspondent pas à la syntaxe en langage C**.

1. **dernier**

Syntaxe : ***fonction dernier (var mot : Tchaine ; ch : Tchaine) : Tchaine ;***
DESCRIPTION : Elle retourne un pointeur sur une chaîne m contenant le dernier mot de la chaîne pointée par ch. Si la chaine est limitée à un mot, dernier est équivalent à ce mot. Si la chaine est vide, dernier est une chaine vide.
E/S : « ch » correspond à la chaine dont on désire le dernier mot et « mot » est le paramètre recevant le dernier mot de cette chaine
Valeur retournée : Retourne un pointeur sur le dernier mot de la chaine ch ou NULL si mot correspond à une chaine vide.

Exemple1 :
Var ch2 :Tchaine ;
ecrire (dernier(ch2, ‘’ceci est un exemple’’) ; => affiche « *exemple* »
Exemple2 :
var ch1,ch2 : Tchaine ;
ch1= dernier(ch2, ‘’ceci est un exemple’’) ;
erire(‘’ch1 = ‘’, ch1) ; => affiche « *ch1=exemple* »

erire(‘’ch2 = ‘’, ch2) ; => affiche « *ch2=exemple* »
Exemple3 :
Var ch2 :Tchaine ;
ecrire (dernier(ch2, ‘’ceci’’) ; => affiche « ceci »
Exemple4 :
Var ch2 :Tchaine ;
ecrire (dernier(ch2, ‘’’’) ; => affiche une chaine vide

2. saufdernier

Syntaxe : ***fonction saufdernier (var debch : Tchaine ; ch : Tchaine) : Tchaine ;***
DESCRIPTION : Donne une chaine constituée de tous les mots de la chaine initiale sauf le dernier. Si la chaîne est composée d'un seul mot saufdernier est équivalent à la chaine vide.
E/S : « ch » correspond à la chaine dont on désire tout sauf le dernier mot et « debch » est le paramètre recevant le début de cette chaine.
Valeur retournée : Un pointeur sur la chaine saufdernier où NULL si la saufdernier est équivalent à une chaine vide.

Exemple1 :
Var ch2 :Tchaine ;
ecrire (dernier(ch2, ‘’ceci est un exemple’’) ; => affiche « *ceci est* un»
Exemple2 :
var ch1,ch2 : Tchaine ;
ch1= saufdernier(ch2, ‘’ceci est un exemple’’) ;
erire(‘’ch1 = ‘’, ch1) ; => affiche « *ch1= ceci est* un»
erire(‘’ch2 = ‘’, ch2) ; => affiche « *ch2= ceci est* un »
Exemple3 :
var ch2 :Tchaine ;
ecrire (saufdernier(ch2, ‘’ceci’’) ; => affiche la chaine vide
Exemple4 :
var ch2 :Tchaine ;
ecrire (saufdernier(ch2, ‘’’’) ; => affiche une chaine vide

3. miroir

Syntaxe : ***fonction miroir (var mirch : Tchaine ; ch : Tchaine) : Tchaine ;***
DESCRIPTION : Donne une chaine constituée de tous les mots de la chaine initiale mais dans l'ordre inverse en allant du dernier élément au premier.
E/S : « ch » correspond à la chaine à la chaine initiale et « mirch » est le paramètre recevant la chaine avec les mots dans l'ordre inverse des mots de départ.
Valeur retournée : Un pointeur sur la chaine définie par « miroir » où NULL si elle est équivalente à une chaine vide.

Exemple1 :
var ch1, ch2 : Tchaine ;
ch1= miroir(ch2, ‘’ceci est un exemple’’) ;
ecrire(‘’ch1 = ‘’, ch1) ; => affiche « *ch1= exemple un est ceci*»
ecrire(‘’ch2 = ‘’, ch2) ; => affiche « *ch2= exemple un est ceci* »

4. member

Syntaxe : ***fonction member (ssch : Tchaine ; ch : Tchaine) : boolen ;***
DESCRIPTION : Indique si une chaine est sous-chaine d'une autre chaine.
E/S : « ch » correspond à la chaine et ssch à la sous-chaine.
Valeur retournée : 1 si « ssch » est une sous-chaine de « ch » sinon 0.

Remarque : On n'a pas le droit d'utiliser « strstr » pour écrire cette fonction.
Exemple1 :

```
Si member (''est un'', ''ceci est un exemple'') alors ecrire (''est une sous-chaine'') ;
        sinon ecrire (''N'est pas une sous-chaine'') ;
ecrire (saufdernier(ch2, ''') ; => affiche une chaine vide
```

5. efface

Syntaxe : ***fonction efface (var ch : Tchaine ; mot : Tchaine) : Tchaine ;***
DESCRIPTION : Supprime dans une chaine, toutes les occurrences d'un mot donné.
E/S : « ch » correspond à la chaine dans laquelle on désire supprimer toutes les occurrences de « mot ». A la fin, « ch » contient la nouvelle chaine.
Valeur retournée : Un pointeur sur la nouvelle chaine « ch » ou NULL si la chaine résultante est vide.

Exemple1 :

```
var ch1,ch2 : Tchaine ;
ch2= ''ceci un est un exemple'' ;
ch1= efface(ch2, ''un'') ;
ecrire(''ch1 = '', ch1) ; => affiche « ch1= ceci est exemple»
ecrire(''ch2 = '', ch2) ; => affiche « ch2= ceci est exemple »
```

Remarque : Cette fonction ne pourra passer « ch » en valeur immédiate !!!

Construire un programme principal vous permettant de tester au fur et à mesure les traitements précédents afin de vous assurer qu'ils fonctionnent.

Partie III : Implémentation d'un « interpréteur » de commande

Afin de tester les différences fonctions demandées dans ce TP, vous réaliserez un mini-interpréteur de commandes de la manière suivante :
Soit la liste de commandes : (PRE, SFP, PHR, DER, MIR, MIR, MMB, EFF, HLP, FIN).
La commande PRE affichera le 1er mot d'une chaîne, etc..., HLP permettra d'afficher la liste des commandes disponibles, et la commande FIN permettra de quitter le programme.
L'exécutable « **tp5** » récupérée dans l'archive vous donne le type d'interpréteur que l'on désire mettre en œuvre.

1. ***Compléter le programme principal donné par le fichier tp5_squelette.c afin de mettre en œuvre les différentes commandes du mini-interpréteur.***
2. ***Réaliser un makefile afin de produire l'exécutable en utilisant la programmation séparée.***

Partie IV – Compte-rendu

Réaliser un compte-rendu de la séance montrant le bon fonctionnement de cette application.