IAP1 Travaux Pratiques n°4 Année scolaire 2025-2026

Remarque : **Ce TP comprend 5 parties qui sont toutes notées**. Vous ferez au fur et à mesure un compte-rendu de votre travail en répondant aux questions qui vous sont posées. Vous déposerez ce compte-rendu dans la zone de dépôt du serveur pédagogique moodle.centralelille.fr en réalisant une archive <TP4_NOM>.zip contenant le compte-rendu et vos fichiers sources. Vous remplacerez NOM par votre nom.

Objectifs

- **Comprendre le fonctionnement des chaines de caractères en C**,
- Manipulation des fonctions de traitement,
- Ecrire des fonctions en C avec le passage de paramètres.

Compétences attendues

- **Apprendre à développer un programme informatique de manière agile,**
- Savoir tester des traitements pour s'assurer de leur robustesse,
- **Apprendre à utiliser un outil de débogage de code compilé,**
- Faire un compte-rendu.

Etapes de développement d'une application en programmation procédurale

L'approche classique de développement repose sur un cycle en V. Elle passe par 3 phases : analyse, codage et tests.

Phase 1 : L'analyse

L'analyse consiste dans un premier temps à comprendre le besoin afin d'envisager une solution pouvant y répondre. La compréhension nécessite souvent de faire des schémas qui permettent de mieux comprendre le problème posé. La solution se traduit ensuite dans le choix de structures de données et la définition d'un algorithme par exemple sous forme de pseudo-code.

Dès que l'analyse a été faite, il faut immédiatement définir les tests à effectués pour vérifier le code qui sera obtenu. C'est la compréhension de la solution qui permet d'envisager les différents cas à tester : le cas général, mais également des cas limites. Dans tous les cas les tests devront répondre à trois objectifs : valider le besoin (ou le cahier de charges), vérifier la robustesse de programme (pas de plantage intempestif), vérifier l'ergonomie du programme.

Phase 2 : Codage

On peut ensuite passer à la phase de codage. C'est dans cette phase que l'on choisit la plateforme de développement et le langage de programmation.

2. La plateforme de développement est basée sur la distribution **linux Ubuntu**. Ce choix est justifié par des raisons pédagogiques : vous habituez à utiliser un système libre qui favorise l'insertion de l'outil informatique dans les PME. Il est également à la base des applications embarquées que l'on retrouve dans de nombreux systèmes multimedia

grand-public mais également de plus en plus dans des applications industrielles.

3. Le langage de programmation et le **langage C** pour son utilisation dans tous les secteurs de l'entreprise, notamment pour le développement des applications industrielles.

La phase de codage est d'autant plus rapide que l'analyse a bien été réalisée. Le code doit être écrit de manière à faciliter la maintenance de l'application. La relecture par une personne tierce doit être facilitée. Cet objectif est atteint par :

1. un nommage judicieux des variables correspondant à leur rôle,
2. l'indentation du programme,
3. l'insertion de commentaires dans le code.

La phase de codage comprend 3 étapes :

Etape 1 : Vous éditerez votre fichier source à l'aide d'un éditeur pleine page de type « kate » ou « gedit ».

pi@raspberrypi:~Documents/IAP1/TP4 $ kate <source>.c &

Etape 2 : Compilation et edition de liens du fichier source en code exécutable avec gcc

pi@raspberrypi:~Documents/IAP1/TP4 $ gcc <source>.c –o <executable.exe>

La compilation sert à transformer le code source en code objet → permet de vérifier la syntaxe
L'édition de lien sert à inclure le code des fonctions des bibliothèques et à transformer le code objet en code exécutable
Avec gcc, l'option « -o » permet de faire la compilation suivi de l'édition de lien

Etape 3 : Lancer l'exécution de votre programme

pi@raspberrypi:~Documents/IAP1/TP4 $./<executable.exe>

Phase 3 : Les tests
Une fois le code réalisé, il faut effectuer les tests définis dans la phase d'analyse. Ces tests doivent donner des jeux d'essais qui doivent être commentés pour indiquer en quoi il vérifie la conformité du code ou besoins ou ils prouvent sa robustesse.

Comment faire des jeux d'essais
Tous les cas doivent être testés. Les tests doivent être commentés si possibles en renvoyant à des parties de code.

Capture du test :
pi@raspberrypi:~Documents/IAP1/TP4 $ **script <fichier_resultats>**
Script demarré
=> que tout ce qui sort à l'écran est redirigé vers ce fichier. A la fin arrêter le script

pi@raspberrypi:~Documents/IAP1/TP4 $ **exit**

Enoncé du sujet du TP 4 : Traitement de niveau 1 sur les chaines de caractères

On considère construire une nouvelle bibliothèque de manipulation des chaines de caractères en C. Pour nous une chaine de caractères sera un tableau de char dont la longueur utile sera déterminée par le caractère '€'. Nous supposerons que ces chaines ont une longueur d'au plus 80 caractères. Elles peuvent être :

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

De plus ces chaines bien formées (cf. fonction format) auront la particularité de ne jamais commencer ou finir par un espace.

Exemple :

- L1= "" chaine vide
- L2= "exemple" 1 seul mot
- L3="titi et gros minet" plusieurs mots

Nous supposons que nous avons les déclarations suivantes pour définir un type chaine de caractères :

CONST :
MAXCAR=80 ;

Type Tchaine=tableau [1..MAXCAR] de caracteres ;
Tposition=0..MAXCAR ;

Remarque : Dans ce travail nous n'avons pas le droit d'utiliser la bibliothèque « string.h ». Le caractère '€' sert de sentinelle afin de délimiter la fin de toute chaine. De ce fait les fonctions de string.h ne sont pas opérationnelles ainsi que les fonctions d'E/S.

Ecrire en langage C les fonctions suivantes :

Partie I : Lecture et écriture d'une chaine de caractères

fonction lire_chaine(var ch :Tchaine, taillemax : entier) : Tchaine ;
Cette fonction lit une chaine au clavier caractère par caractères et insère après le dernier caractère lu le caractère de fin de chaine. Les caractères de la chaine seront lus à l'aide de la fonction **getchar** ou de **getc**.

Remarque : Cette fonction remplacera les fonctions C 'gets' et 'fgets'. Elle retourne l'adresse de la chaine lue.

procedure ecrire_chaine(ch : Tchaine, relour_ligne :boolean) ;
Cette procedure écrit à l'écran la chaine passée en paramètre. Si retour_ligne est vrai, elle devra placer le curseur sur la ligne suivante. Sinon, elle le laissera sur la même ligne.

Remarque : Cette procédure replacera la fonction C ’puts’.

Réaliser un programme principal basé sur un menu qui permet de tester les fonctions de la partie I. Ce programme sera complété dans les autres parties.

Partie II : Fonction de bases pour le traitement des chaines de caracteres

En vous inspirant des fonctions définies dans string.h (Utiliser man pour lire la définition des fonctions usuelles), écrire en C les fonctions suivantes :

fonction affectation_chaine (var dest : Tchaine, source : Tchaine, taillemax : 0.. MAXCARS) : Tchaine ;

fonction compare_chaine(ch1 : Tchaine, ch2 : Tchaine) : 0.. MAXCARS ;
Retourne la différence entre ch1 et ch2 ou 0 si les deux chaines sont égales.

fonction longueur_chaine(ch : Tchaine) : 0.. MAXCARS ;
Retourne la longueur d’une chaine.

fonction concatene_chaine (var dest : Tchaine, source : Tchaine) : Tchaine ;
Concatene les deux chaines passées en paramètre et retourne l’adresse de la chaine dest.

Compléter le programme principal commencé à la partie I.

Partie III : Fonctions de traitements avancés sur les chaines

fonction min_chaine (var ch :Tchaine) : Tchaine ;
Elle convertit en minuscule les caractères de la chaine ch et retourne l’adresse de cette chaine.

fonction trouve_car_chaine (ch : Tchaine, car : caracteres) : Tchaine ;
Retourne l’adresse de la sous-chaine commençant par car ou NULL si car n’existe pas dans ch.

fonction trouve_sschaine (Tchaine ch , Tchaine ss_chaine) : 0.. MAXCARS ;
Retourne la position la position de ss-chaine dans ch sinon -1.

Compléter le programme principal de la partie II.

Partie IV – Déboguer le traitement consistant à formater une chaine de caractères (cf. code fourni)

fonction formater_chaine(var ch :Tchaine) : Tchaine ;
Cette fonction formate une chaine de manière à ce qu'elle respecte les spécifications imposées au début de cet énoncé. Lors de la saisie (effectuée impérativement à l’extérieur de format), l’utilisateur peut avoir entré au début et à la fin des espaces ou des tabulations. Les mots de la

chaine saisie peuvent être séparés également par plusieurs espaces ou tabulations. « formater_chaine » doit renvoyer une chaine parfaitement formatée.

Vous devez corriger les erreurs de syntaxe et les erreurs d'exécution de la fonction **formater_chaine** dont le code C est donné dans le fichier '*format.c*' joint à ce sujet. Vous pourrez vous appuyez sur le schéma de la Figure 1 à partir duquel a été généré ce code. L'état en vert est l'état initial et l'état rouge l'état final. Afin de corriger les erreurs d'exécution, je vous suggère d'utiliser gdb l'outil débogage de GNU. Placer un point d'arrêt sur la ligne 24 du code fourni. Lancer l'exécution du programme et dès le point d'arrêt passé en exécution pas à pas.

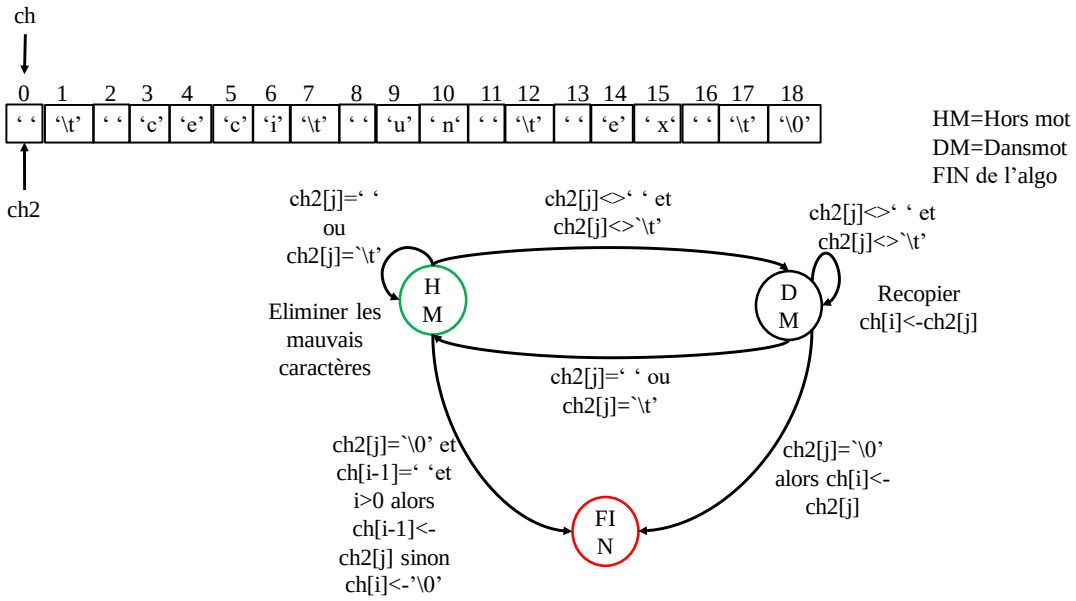

Figure 1. Automate à états finis permettant l'analyse syntaxique pour le formatage d'une chaine de caractère.

Exemple :
int main()
{
char ch1[]=‘’ Ceci est un exemple ‘’;
ecrire_chaine(‘’\n La chaine formatee est : ‘’, faux),
ecrire_chaine(***formater_chaine(ch1), vrai);***
return 1;
}

Compléter le programme principal de la partie III afin d'intégrer le test de cette fonction.

Partie V – Faire une compte rendu

Faire un compte-rendu des quatre parties précédentes et montrant que les traitements implémentés sont opérationnels et robustes.
Indiquer dans votre compte-rendu le nom de chaque fonction auxquelles correspondent les fonctions des parties II et III.