TP0 : Machine de Turing

Ce premier TP vise à vous faire un **rappel** des notions importantes du langage de programmation C. Normalement, lors de votre parcours, vous devriez avoir déjà fait du C et ce TP devrait uniquement utiliser des notions que vous avez déjà vu. Voici une liste des concepts importants:

1. Manipulations de pointeurs.
2. Allocation et gestion de la mémoire.
3. Déclaration de structures de données.
4. Gestion de fichier au travers les fonctions standards.

Les concepts suivants peuvent être utilisés dans les exercices mais ne sont pas explicitement demandés:

1. *Bitmasking* et représentation entière.
2. Utilisation d'outils comme *valgrind*.

Avant de commencer le TP, il est conseillé d'effectuer un peu de lecture sur les concepts utiles afin d'avoir les bases nécéssaires pour ce TP **ainsi que les autres** qui suivront. Si vous n'avez pas fait de programmation en C auparavant, il est impératif d'assister à la première démonstration qui vise à faire un rappel et une brève introduction au langage.

Le code que vous aller remettre, pour ce TP ainsi que les autres, doit absolument compiler. Un code qui ne compile **pas** ne mérite pas de points, sauf pour une raison en dehors de votre contrôle. Vous devez porter une attention particulière à la gestion mémoire. Une gestion incorrecte qui laisse de la mémoire non retournée au système se verra perdre des points.

En général, vous serez évalué selon le bon fonctionnement de votre code ainsi que la qualité de celui-ci (gestion d'erreur, style de code).

1 Préparation de l'environnement

Pour compléter ce TP, il est nécéssaire d'utiliser le fichier C (*template.c*) disponible sur Studium. Le fichier C contient une base minimale pour faire le TP ainsi que la fonction

```
int strcmp(char* p1, char* p2)
```

qui permet de comparer des chaînes de caractères. Le fichier squelette contient toutes les imports nécéssaires et suffisants pour faire le devoir. Ainsi, si vous ajoutez des importations, vous perdrez des points.

Si vous utiliser l'outil *Clion* (disponible gratuitement pour les étudiants de l'université de Montréal), le code donné devrait compiler directement.

Attention! Il est normal d'avoir des warning lors de la compilation quant au fait que des fonctions de la librairie standard ont été redéfinies.

2 Implémentation d'une machine de Turing

Dans cette partie, vous aller devoir implémenter une machine de Turing. Cela peut sembler une tâche complexe, mais en séparant le code en plusieurs petites fonctions, il s'agit d'un tâche assez simple.

Si vous savez déjà comment une machine de Turing fonctionne, vous pouvez sauter la description qui suivra. Sinon, il vous est recommandé de lire la description suivante.

Une machine de Turing est une construction mathématique qui sert à formaliser la notion de calcul par ordinateur. Le but de cet exercice est d'en implémenter une, ce qui ne nécéssite pas de notion préalable particulières. Une machine de Turing est un *automate* qui écrit et lit de l'information contenu sur un ruban. Afin de pouvoir exécuter des tâches complexes, elle se sert de sa description qui est stockée sous forme de transition, que vous avez pu voir dans l'exercice en Sec. 2.4. La machine utilise une tête de lecture qui se déplace dans une direction arbitraire sur le ruban. La machine peut être vue comme fonctionnant selon l'algorithme suivant:

1. Au démarrage de la machine, le mot d'entrée est placé sur le ruban et la tête est en position 0.
2. Si l'état courant est l'état acceptant ou rejettant, terminer.
3. Sinon:
   (a) Lire le symbole à l'emplacement de la tête de lecture.
   (b) Trouver la transition qui dénote l'action à prendre étant donné l'état courant et le symbole sous la tête de lecture.
   (c) Écrire le nouveau symbole à l'emplacement de la tête de lecture, déplacer la tête selon le mouvement indiqué dans la transition et changer l'état courant pour le prochain état.

Il existe plusieurs variantes de machines de Turing qui sont équivalentes. Afin de simplifier cet exercice, la machine que nous allons utiliser ici contient un unique ruban qui est infini dans la direction "droite" seulement.

Avant de commencer à coder cette machine immédiatement, il est utile de commencer à faire de petites fonctions qui vont faciliter le dévelopement.

2.1 Exercice 1

Une machine de Turing doit mesurer des mots contenus sur son ruban. Afin de le faire, il faut avoir une fonction pour aller chercher la taille d'une chaîne de charactère.

Vous devez implémenter la fonction suivante:

```
error_code strlen(char* s);
```

Qui mesure la chaîne de caractère à l'adresse du pointeur *s*. Attention! La représentation des chaînes de caractères en C est différente de d'autres langages. Votre fonction doit avoir le même comportement que la fonction standard[1].

[1] https://linux.die.net/man/3/strlen

2.2 Exercice 2

La machine de Turing que nous allons implémenter doit aller chercher sa description dans un fichier texte. Pour nous simplifier la tâche, nous allons lire le ficher ligne par ligne. Il serait par contre utile d'avoir une façon d'aller cherche ce nombre de lignes.

Vous devez implémenter la fonction

```
error_code no_of_lines(FILE *fp);
```

Qui va extraire le nombre de lignes d'un fichier. Attention! Votre fonction ne doit **pas** changer la position courante du curseur dans le fichier. Je vous invite à aller lire les pages suivantes:

1. `https://linux.die.net/man/3/fopen`
2. `https://linux.die.net/man/3/ftell`
3. `https://linux.die.net/man/3/fseek`
4. `https://linux.die.net/man/3/getc`

2.3 Exercice 3

La lecture de fichier en C peut être assez pénible. Afin de se faciliter la vie, on va utiliser une fonction de haut niveau qui va lire une ligne de l'entrée d'un seul coup.

Vous devez implémenter la fonction suivante:

```
error_code readline(FILE *fp, char **out, size_t max_length);
```

Les entrées sont, en ordre, un pointeur vers un descripteur de fichier, un pointeur *vers un pointeur* de caractère ainsi que la longeur maximale de la ligne à lire.

Cette fonction doit avoir le comportement suivant:

1. Allouer un bloc mémoire de la taille de $max\_length + 1$.
2. Lire en entrée la ligne au curseur de fp dans le bloc mémoire précédemment alloué.
3. Affecter la bonne valeur à *out*.
4. Retourner un code d'erreur si nécessaire.

Un code d'erreur doit être retourné dès que la mémoire n'a pas pu être allouée. On utilisera -1 comme code d'erreur. S'il n'y a pas d'erreur, la fonction doit retourner le nombre de caractères non-nuls écrits dans le bloc mémoire.

Vous devez aussi expliquer pourquoi on alloue un bloc mémoire de taille $max\_length + 1$ et non pas seulement max_length. Ajouter en commentaire une explication avant la ligne de l'allocation.

Je vous conseil d'aller voir la description de la fonction *malloc* pour allouer de la mémoire:

`https://linux.die.net/man/3/malloc`

2.4 Exercice 4

Afin d'exécuter la machine de Turing, il faudra aller chercher une description de celle-ci sur le disque. Cette description aura le format suivant:

```
q0
qA
qR
(q0,0)->(qR,0,R)
(q0,1)->(qA,1,R)
...
```

Vous pouvez assumer que chaque description est correcte. Pour simplifier la tâche, un état possède **au plus** 5 caractères et évidemment au moins 1. Les trois premières lignes contiendront toujours l'état initial, l'état acceptant et l'état rejetant dans cet ordre.

Cet exercice portera sur les transitions de la machine de Turing, données dans les lignes 4 et 5 de cet exemple. Les transitions sont du format:

```
(état_courant, symbole_a_lire)->(nouvel_état,symbole_a_écrire,mouvement)
```

Le mouvement est décrit par un seul caractère, soit G, R, D, respectivement gauche, reste et droite.

Votre tâche dans cet exercice est de créer une structure (struct) qui contiendra les champs suivants:

1. Un pointeur vers une chaîne de caractères pour l'état courant.
2. Un pointeur vers une chaîne de caractères pour le prochain état.
3. Un entier qui représente le mouvement relatif décrit par le mouvement.
4. Deux charactères qui stockeront le caractère à lire et celui à écrire respectivement.

2.5 Exercice 5

Avant d'aller chercher sur le disque les transitions et de les charger en mémoire, nous aurons besoin d'une fonction pour copier de la mémoire d'un pointeur vers un autre. Vous devez implémenter la fonction suivante:

```
error_code memcpy(void* dest, void* src, size_t len);
```

Cette fonction copie *len* **bytes** de la source vers la destination.

2.6 Exercice 6

Maintenant que nous avons une façon de représenter les états et une façon de lire les lignes, il faut une façon de décomposer une ligne en une transition.

Si vous avez de l'expérience en C, vous remarquerez peut-être que la structure fixe du fichier permetterait d'analyser celui-ci sans allouer dynamiquement de la mémoire. Cependant, ce TP est construit pour vous forcer à le faire, ce que vous devez respecter.

Vous devez implémenter la fonction suivante:

```
transition* parse_line(char* line, size_t len);
```

Les paramètres sont respectivement la ligne à lire et la longueur de cette ligne.

Cette fonction doit avoir le comportement suivant:

1. Allouer une transition en mémoire (retourner NULL si impossible de le faire)
2. Extraire le contenu de la ligne de transition dans la structure. Chaque état doit être une nouvelle chaîne allouée par malloc. Attention aux erreurs d'allocation des chaînes! En cas d'erreur, votre fonction doit retourner la mémoire déjà allouée au système et renvoyer NULL.

   Bien sur, cette fonction n'analyse que les lignes qui représentent des transitions dans la description de la machine, c'est à dire les lignes $l \geq 3$.

3 Exercice 7

Nous avons presque fini. La dernière tâche requiert de mettre ensemble toutes les fonctions que nous avons utilisé pour exécuter la machine de Turing. Vous devez maintenant écrire la fonction suivante:

```
error_code execute(char* machine_file, char* input);
```

Qui exécute la machine de Turing sur le mot décrit par *input*. La description sera lu du fichier $machine\_file$. La fonction renvoie 1 si et seulement si la machine à acceptée (à moins qu'elle boucle). Votre fonction doit faire les choses suivantes:

1. Allouer un tableau qui contiendra toutes les descriptions (utiliser le nombre de lignes)
2. Lire le fichier de description

3. Allouer un ruban de travail (au moins deux fois la taille du mot reçu en entrée)
4. Copier le mot sur le ruban de travail
5. Placer la tête de lecture au début du ruban
6. Exécuter la machine
7. Si on atteint éventuellement un état acceptant ou rejetant, terminer l'exécution de la machine

Attention! Le ruban étant infini dans une direction, il est nécessaire de changer sa taille au fur et à mesure que le besoin se présente.

Remise

Ce travail est à faire **individuellement**. Le code est à remettre par remise électronique avant la date indiquée.

Chaque jour de retard est -15%, mais après le deuxième jour la remise ne sera pas acceptée.

Indiquez clairement votre nom et votre matricule dans un commentaire en bloc au début de chaque fichier, dans ce format:

```
Prenom1 NomDeFamille1 Matricule1
```

Le programme doit être exécutable sur les ordinateurs du DIRO.

Pour la remise, vous devez remettre un fichier (main.c) par la page Studium du cours dans un fichier zip.

Assurez-vous que tout fonctionne correctement sur les ordinateurs du DIRO.

Barème

- Votre note sera divisé équitablement entre chaque question, sauf la question 7 qui vaut l'équivalent de 2 questions.
- Tout usage de matériel (code ou texte) emprunté à quelqu'un d'autre (trouvé sur le web, etc.) doit être dûment mentionné, sans quoi cela sera considéré comme du plagiat. Si pour une question votre solution est directement copiée, même si il y a attribution de la source, cette question se verra attribuée la note de zéro. Vous pourrez cependant l'utiliser dans les sections suivantes sans pénalité.
- Vous serez pénalisés pour chaque warning lors de la compilation à raison de 1% par warning, sauf pour les warning reliés à l'assignation à NULL, à la comparaison avec NULL, et aux override des fonctions de librairie. Les warning qui sont les mêmes que ceux retrouvées
- Si une fuite mémoire mémoire est identifiée, vous perdrez 15%. Vous ne perdrez pas plus de points pour les fuites si vous en avez une ou trente.

- Les accès mémoire illégaux identifiés par Valgrind entraîneront jusqu'à 5% de perte, à raison de 1% par accès. La répétition d'un même accès sera comptée comme 1% de plus quand même.
- Votre devoir sera corrigé automatiquement en très grande partie. Si vous déviez de ce qui est demandé en output, les points que vous perdrez seront perdus pour de bon. Si vous n'êtes pas certains d'un caractère demandé, demandez sur le forum studium et nous répondrons de façon à ce que chaque étudiant puisse voir la réponse.
- La méthode de développement recommandée est d'utiliser CLion et son intégration avec Valgrind. Si vous voulez utiliser d'autres techniques, vous pouvez le faire, mais nous ne vous aiderons si vous rencontrez des problèmes avec ces techniques.