WACRENIER Matthéo
NICOLAS Théo

L2/PEI2

GIF 1301 : Dossier de Programmation

2021/2022

Sommaire :

Introduction :

Dans le domaine de l'informatique théorique, il est souvent question de calcul ou d'algorithmie. Ces notions sont étroitement liées au sujet qui nous intéresse pour ce projet, à savoir la **Machine de Turing**. Imaginé par Alan Turing en 1936, ce modèle a pour but de donner une définition précise au concept d'algorithme ou de « procédure mécanique ». La machine de Turing est en quelque sorte un modèle abstrait du fonctionnement des appareil mécaniques de calcul, comme un ordinateur par exemple. Ce concept est encore de nos jours fortement utilisés dans les domaines de la complexité algorithmique et de la calculabilité.

Le concept de la machine de Turing est aussi fortement lié à la Thèse de Church, qui est un postulat selon lequel tout problème de calcul fondé sur une procédure algorithmique peut être résolu par une machine de Turing. Ainsi, nous pouvons dire d'un système de programmation, qu'il est « **Turing-complet** », lorsque que ce dernier est capable de simuler le fonctionnement d'une machine de Turing.

Dans le cadre de notre projet, nous allons donc chercher à simuler une machine de Turing en C, qui est un langage de programmation que nous savons Turing-complet.

Portée du projet :

Objectif du projet

Comme dit précédemment, l'objectif de ce projet est de réaliser un simulateur de machine de Turing en langage C. Le but est de pouvoir entrer une bande de Turing, un code de Turing et pouvoir l'exécuter afin d'effectuer le calcul du code de Turing sur la bande de Turing et d'en observer le résultat.

Pour cela, nous pouvons séparer la tâche en deux parties. La première consistera en la saisie du programme de la machine ainsi que l'initialisation de la bande de Turing. La seconde, elle, consistera en la simulation de l'exécution du programme pas à pas, ou automatiquement.

Hypothèses retenues

Dans le cadre du modèle que nous allons programmer, nous émettons plusieurs hypothèses dont voici la première :

- L'utilisateur ne pourra pas rentrer autre chose que des 0 ou des 1 lors de l'initialisation de sa bande de Turing. Nous avons construit notre programme sur cette base, en empêchant par ailleurs l'utilisateur de rentrer n'importe quelle autre valeur ou caractère lors de l'initialisation de la bande.

La deuxième hypothèse que nous émettons pour construire notre code est la suivante :

- L'utilisateur rentrera manuellement le code de Turing nécessaire à la simulation qu'il souhaite effectuer. Il sera cependant libre de le modifier dans l'interface Homme-Machine, sans devoir tout retaper s'il a par exemple commis une erreur de frappe lors de la saisie.

Contraintes

Dans le cadre de ce projet, nous devrons programmer avec quelques contraintes imposées par le sujet du projet. Ainsi, afin de créer notre bande de Turing, nous utiliserons une structure prédéfinie et donc imposée. Il en va évidemment de même pour le code de Turing que l'utilisateur devra entrer, il sera programmé sur la base d'une structure imposée. Ces contraintes nous permettent d'avoir les fondations nécessaires pour commencer convenablement notre projet.

Analyse du projet :

Simplifications possibles

Si nous analysons plus en détail le projet, nous savons que nous pouvons apporter quelques simplifications au projet afin d'en comprendre mieux la structure. Nous pouvons déjà réfléchir en amont, avant même de commencer à coder, en analysant toutes les fonctionnalités qui nous sont demandées pour ce projet. Il nous est dit que nous devons concevoir un programme permettant de lister de code de Turing et de le modifier, ce qui signifie que nous aurons déjà 3 fonctions dans notre programme : une permettant de rentrer le code de Turing, une autre permettant de l'afficher, et une autre permettant de le modifier. Si nous continuons dans l'analyse de la consigne, il nous est demandé de concevoir un programme permettant de lister la bande de Turing et de l'initialiser. Cela fait donc 2 fonctions supplémentaires à implémenter. Enfin, nous devons être capables d'effectuer une simulation du code sur la bande, donc nous avons notre dernière fonction. Au terme de cette analyse, nous savons que nous pouvons théoriquement simplifier notre projet en l'élaboration de 6 fonctions permettant de satisfaire toutes les fonctionnalités demandées.

Solution existante

Afin de mieux comprendre ce que nous devions programmer, un exemple de simulateur de machine de Turing en ligne a été mis à notre disposition. Nous avons donc pu effectuer des tests afin de concevoir des codes de Turing fonctionnels pour effectuer nos propres tests sur notre programme. Cet exemple nous a donc été très utile afin de mieux comprendre le sujet, qui semblait pour nous de premier abord assez abstrait. En effet, il nous fallait comprendre comment la machine fonctionnait avant même d'essayer de la reproduire, et c'est en quoi cet exemple nous a aidé.

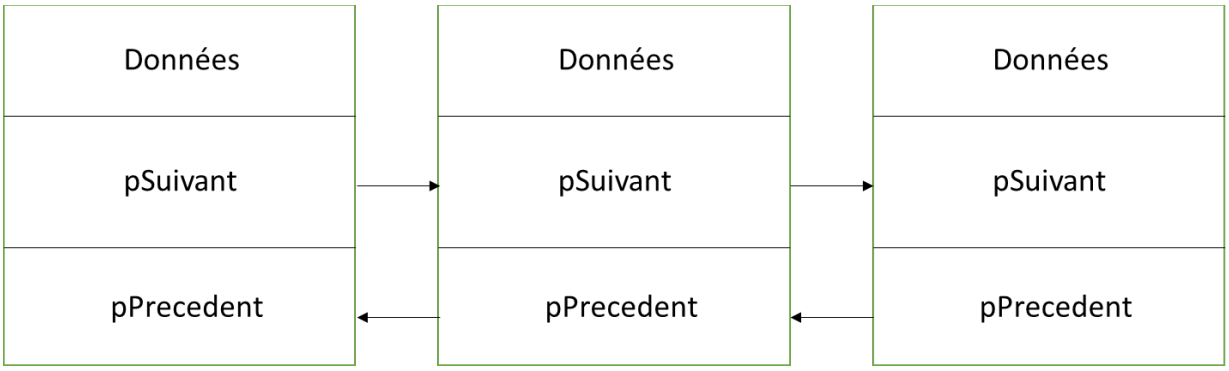
Conception :

Découpage du projet

Nous avons découpé notre projet en plusieurs étapes. Tout d'abord, nous avons commencé par le code pour créer la bande de Turing ainsi que celui qui gère son affichage. Ensuite nous nous sommes occupés du code pour créer le code de Turing et l'afficher sous la forme souhaitée. Enfin nous avons travaillé sur l'interface graphique et les différents affichages des fonctions.

Description des éléments / Principe de fonctionnement

Dans ce projet, nous devons utiliser des listes doublement chaînées afin pour réaliser la bande et le code de Turing. Les listes sont des sortes de cases reliée entre elle par des pointeurs et contenant des données. Chaque pointeur pointe sur la case suivante ou précédente.

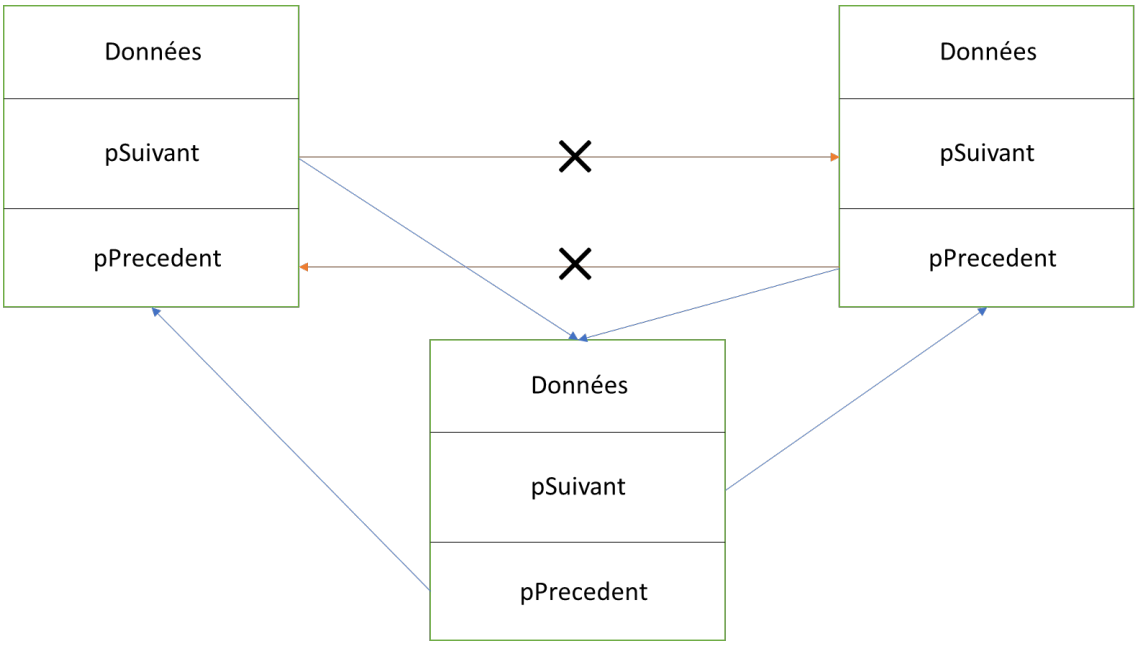


Notre programme regroupe de multiples fonctions dont nous allons ici expliquer le fonctionnement.

La fonction d'insertion :

Cette fonction est utilisée pour permettre de placer une case vide après la case souhaitée dans une liste. Cette fonction est aussi bien utilisée pour la bande de Turing que pour le code de Turing.

Son fonctionnement est simple : on utilise un pointeur temporaire pour lequel on attribue une place en mémoire et on procède à son insertion en redéfinissant les pointeurs suivants et précèdent avec les bons pointeurs :



La fonction de création de la bande :

Pour créer la bande, on utilise une liste doublement chaînée dans laquelle on entre les valeurs souhaitées, en vérifiant que ce soit bien des 0 ou des 1.

Pour cela, on définit un pointeur temporaire qui pointe sur le début de la liste et on utilise la fonction d'insertion afin de crée une nouvelle case. Ensuite, on se déplace sur cette case et on y insère la valeur rentrée par l'utilisateur, en filtrant les valeurs qui ne sont pas 0 ou 1. On boucle ensuite tant que l'utilisateur ne saisit pas « q ».

La fonction de création du code :

On utilise aussi une liste doublement chaînée pour créer le code du Turing mais on a ici plus de données à saisir que précédemment.

On procède de la même façon que pour la création de la bande mais cette fois on vérifie si la valeur d'entrée est bonne et on redemande une entrée si elle ne l'est pas, par exemple pour la valeur actuelle ou le déplacement. Aussi on utilise une fonction qui permet de ne pas prendre en compte la touche entrée comme un caractère pour la saisie suivante. Pour finir la saisie du code, il faut saisir un état suivant de 0.

La fonction d'exécution du code :

Notre fonction d'exécution est la fonction d'interprétation du code entrée par l'utilisateur. Elle exécute les actions demandées par le code de Turing sur la bande de Turing.

Tout d'abord, on regarde si la valeur de la bande est égale à la valeur courante du code. Si c'est le cas, on remplace la valeur courante par la valeur suivante sinon, on passe au code suivant. Ensuite, on compare la valeur de déplacement, si elle est de 1, on passe à la valeur suivante de la bande, si elle est

de -1, on passe à la valeur précédente de la bande, et à 0, on ne fait rien. Après cela, on passe au code suivant et on cherche à se replacer sur la bonne ligne de code correspondant à l'état Suivant de la ligne que nous venons d'exécuter. Pour cela, nous allons sauvegarder cet état suivant dans une variable temporaire.

Nous comparons ensuite l'état de départ et l'état suivant pour savoir de quel côté du code nous irons chercher la prochaine ligne de code à exécuter. Une fois que notre nouvelle ligne de code est trouvée, nous l'exécutons donc et recommençons la boucle jusqu'à ce que le code arrive à son terme.

Interface utilisateur

Pour notre interface, nous avons choisi de faire quelque chose de simple mais qui soit tout de même agréable pour l'utilisateur. Pour cela, nous avons créé deux fonctions principales pour l'affichage :

La fonction affichage d'initialisation :

Cette fonction permet d'entrer la bande et le code de Turing dans l'interface graphique créé. C'est aussi elle qui appelle la fonction menu.

On commence par effacer le contenu de la console et la mettre en plein écran pour un meilleur affichage de l'interface. On crée ensuite un grand cadre pour contenir toute notre interface et un petit cadre qui contiendra la bande de Turing. Après cela, on demande à l'utilisateur d'entrer la bande et le code de Turing pour ensuite les afficher. On finit par appeler la fonction menu.

La fonction menu :

On affiche avec cette fonction un menu qui nous demande si on veut exécuter la bande de Turing, la modifier, ou quitter le programme.

Pour cela, on lit la touche entrée par l'utilisateur pour pouvoir faire les actions nécessaires. S'il appuie sur F3, il va pouvoir modifier le code et on fait alors appelle à la fonction de modification du code, s'il appuie sur F4 il va pouvoir exécuter le code de Turing à l'aide de la fonction d'exécution. Tant que l'utilisateur n'appuie pas sur F10, on continue à demander l'entrée d'une touche après l'exécution des fonctions demandées.

La fonction d'affichage du code pour modification :

On utilise cette fonction pour pouvoir modifier le code de Turing entré au début.

Ici, contrairement à l'affichage basique du code de Turing, on affiche le code ligne par ligne afin de pouvoir quelle ligne on veut modifier. Pour cela, on lit le caractère entré et si l'utilisateur entre un « m », on lance la fonction de modification du code (presque identique à la fonction d'entrée du code mais ici pour une seule ligne du code), s'il appuie sur la flèche du haut, on passe au code précédent, et au code suivant avec le flèche du bas. On appui est « Echap » pour sortir.

Tests :

Tests unitaires des divers éléments

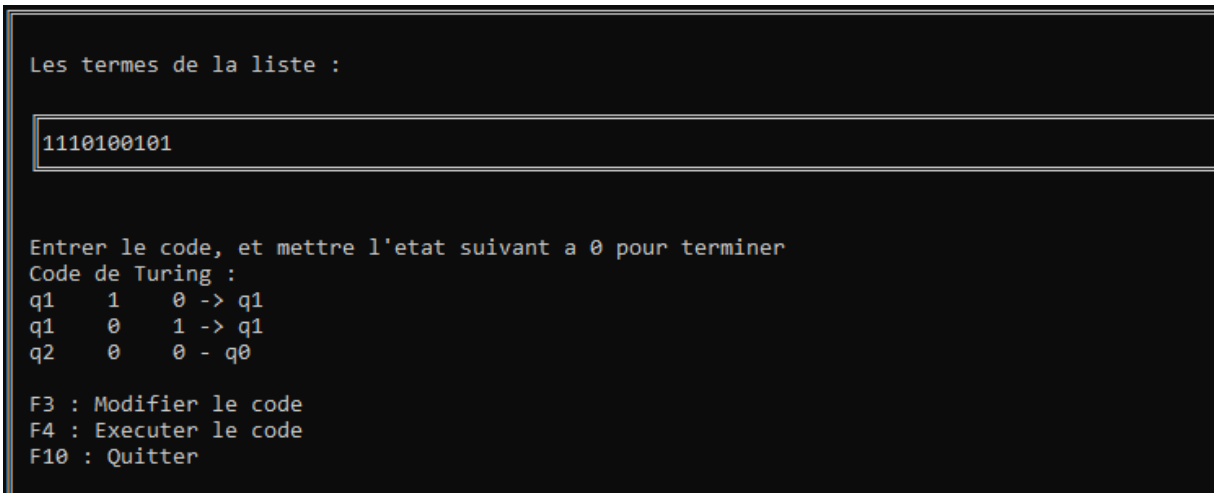
Nous avons testé chaque fonction séparément afin d'en vérifier le bon fonctionnement. Etant donné que nous avons commencé par les fonctions de création et d'affichage de la bande, elles ont été les premières que nous avons été amenés à tester. Nous n'avons pas rencontré de problème majeur sur ces fonctions. Par la suite, nous avons élaboré les fonctions d'entrée et d'affichage du code de Turing. En testant séparément ces fonctions, nous avons eu un léger problème avec l'affichage du code de Turing, qui a été facilement résolu. Ensuite, la fonction que nous avons cherchée à établir était celle de la simulation. Cette fonction n'a pas été simple à créer, et elle a demandé beaucoup d'heures de travail et beaucoup d'analyse afin de comprendre les erreurs que nous rencontrions. La principale difficulté rencontrée avec cette fonction fut par rapport à la gestion des états de Départ et des état Suivants. Nous avons testé plusieurs méthodes pour associer les états de fin et de début entre eux jusqu'à trouver une méthode fonctionnelle. De tout le projet, c'est la fonctionnalité qui nous a pris le plus de temps à faire fonctionner. Enfin, la dernière fonctionnalité mise en place fut la modification du code de Turing, qui fut implémenté sans aucun soucis au programme, étant donné qu'elle était très similaire à la fonctionnalité d'entrée du code de Turing.

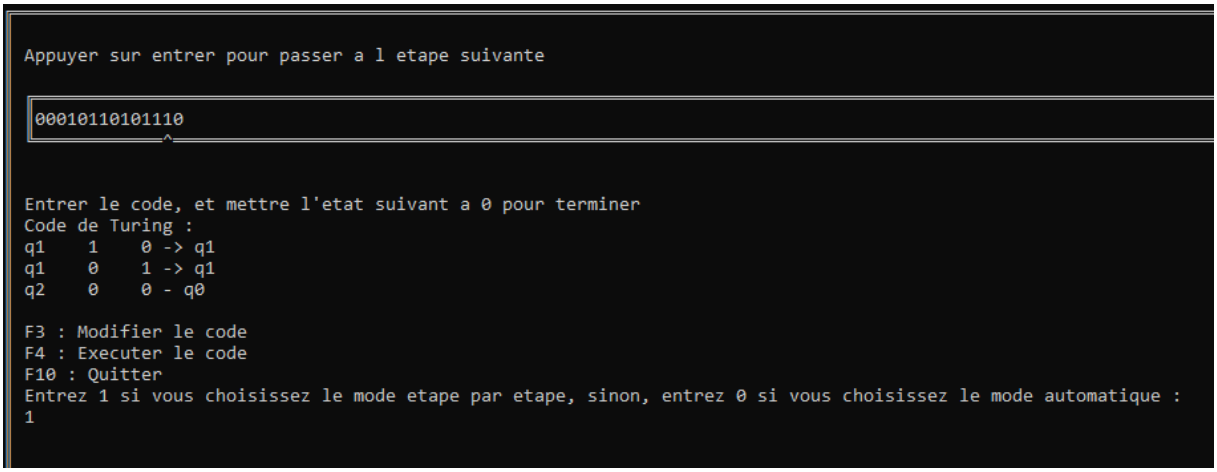
Test d'intégration

Après les tests unitaires de nos fonctions, il nous a fallu les implémenter au programme principal afin de corriger les éventuels conflits entre les fonctions. Dans la grande majorité, lorsque nos fonctions fonctionnaient indépendamment du reste, on ne rencontrait pas de problème lors de l'implémentation dans le programme principal. Encore une fois, la fonction qui nous a posé le plus de soucis fut celle traitant de la simulation du code de Turing sur la bande. Avec beaucoup de débogage, nous avons finalement réussis à implémenter correctement cette fonction, qui est le cœur du programme, étant donné que c'est cette dernière qui gère la simulation.

Résultats :

Ainsi, une fois notre programme final établi, nous avons effectué de nombreux tests avec plusieurs codes de Turing différents. Nous avons notamment testé notre programme avec des codes d'addition, ou encore de multiplication, mais aussi avec des codes beaucoup plus simples. Nous avons par exemple testé un code remplaçant les 1 par des 0 et les 0 par des 1 et ce à l'infini, en utilisant le système des bandes infinies.





Nous pouvons bien voir sur cet exemple très simple le bon fonctionnement de notre code de Turing, et par conséquent, de notre programme de simulation. Nous voyons également notre interface graphique simple mais efficace, qui permet de bien guider l'utilisateur lors de l'exécution du programme.

Perspectives :

Limites du projet et améliorations possibles :

Comme nous l'avons évoqué dans les hypothèses du projet, l'utilisateur ne peux rentrer que des 0 et des 1 dans la bande, ce qui limite les calculs possibles. Nous aurions pu améliorer cette partie en changeant nos conditions de vérification et l'interprétation du code mais il aurait fallu avoir plus de temps pour tester cela.

Quant à l'interface graphique, elle n'est pas bien optimisée pour les bandes infinies (la bande sort de son cadre) et on ne peut pas arrêter l'exécution du code sans quitter le programme. On aurait pu améliorer cela en faisant en sorte que le cadre s'agrandisse lorsque la bande est trop grande ou que la bande se décale dans le cadre et en ajoutant une condition pour sortir de l'exécution lorsque la bande est infinie. De même lorsque le code est trop grand, il dépasse du cadre.

Aussi, lorsque l'on demande à l'utilisateur de rentrer quelque chose, on peut appuyer sur entrer et écrire en dehors du cadre. Cela n'entrave pas l'entrée des données mais on se retrouve avec des chiffres à la place du cadre.

Une autre amélioration possible aurait été de permettre la modification de la bande de Turing pendant l'exécution du programme, comme nous faisons pour modifier le code. Nous aurions même pu imaginer une possibilité de créer une nouvelle bande pour simuler notre nouveau code, et ce autant de fois que nous le désirons. Cela reste une possibilité de fonctions à implémenter lors des séances post-projet.

Nous aurions pu aussi optimiser certaines fonction comme celle de modification du code qui est similaire à celle de la création du code mais qui est exécutée qu'une seule fois avec un texte afficher différent.

Conclusion :

Durant ce projet, nous avons donc utilisé nos connaissances en programmation C afin de recréer un simulateur de machine de Turing. De plus, nous avons pu réaliser une interface graphique simple mais efficace afin de guider l'utilisateur lors de la prise en main du simulateur. Tout au long de l'élaboration de ce programme, nous avons rencontré des problèmes liés à diverses erreurs que nous avons finalement su résoudre. Le programme final est cependant toujours améliorable, nous aurions pu en effet ajouter diverses fonctionnalités rendant le simulateur plus ergonomique et plus guidé, bien qu'il soit déjà assez facile à prendre en main. Nous sommes cependant très satisfaits de ce que nous avons pu produire pendant ce projet.