INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

Projet MIPP
GM3 2022 - 2023

Interpréteur d'un langage graphique

Étudiants :
KESSLER Aymeric
LANGOLFF Clément

Enseignant-responsable du projet :
Chaignaud Nathalie

Table des matières

I Analyse descendante et clarification du problème 3

1 Comment lire et interpréter une chaine de caractères 3

2 Comment stocker les informations 3

3 Comment afficher le résultat 4

II Organisation et répartition des tâches 5

III Explications des fonctions principales 6

IV Quelques exemples 11

V Perspective d'amélioration et conclusion 13

Première partie

Analyse descendante et clarification du problème

L'objectif du projet est d'écrire un programme qui lit des instructions à partir d'une chaîne de caractères et les exécutes en affichant le résultat. Ceci nous a amené à 3 problématiques principales :

- Comment lire et interpréter une chaîne de caractères en instructions définies par le programme
- Comment stocker les informations pour pouvoir les afficher à l'écran
- Comment afficher les informations à l'écran de manière interprétable pour l'utilisateur

1 Comment lire et interpréter une chaine de caractères

Nous devons d'abord définir une instruction. Une instruction est une suite de lettre interprétable par le programme, c'est à dire qui contient toutes des directions N,S,E,O qui permettent respectivement de se déplacer en haut, en bas, à droite et à gauche. D'un mode d'écriture L ou B correspondant au stylo levé et baissé et d'un nombre de répétitions pour répéter plusieurs fois une instruction. L'utilisateur a la possibilité d'entourer un suite d'instructions par des parenthèses pour répéter plusieurs fois une instructions, nous introduisons ainsi la longueur d'une instruction qui correspond au nombre de caractères présents entre ces parenthèses. Nous pouvons donc définir un cadre de récursivité dans le sens où chaque instruction peut contenir elle même des instructions.

Exemple 4SE3(2NO) peut être diviser en 4S, E, 3(2NO) et l'instruction 3(2NO) peut être elle aussi divisée en 2N et O qui sont répétées 3 fois.

Nous définissons le type instruction qui permet de stocker la longueur, la chaine présente dans l'instruction et le nombre de répétitions de l'instruction.
instruction = structure { lg : Entier , c : chaine , rep : Entier }

Il nous faut ensuite définir une fonction LireInstruction qui permettra de stocker les informations concernant une sous instruction d'une chaine de caractères. Cette fonction prendra une chaine en paramètre et interprétera les caractères un par un.
L'utilisateur est cependant limité, il ne peut pas rentrer n'importe quel caractère à la suite. Nous avons identifier différents cas :

- l'instruction est une instruction simple : N, E ,S, O , L , B dans ce cas la longueur vaut 1 et le nombre de répétition vaut 1
- l'instruction est une instruction simple précédée d'un chiffre ou d'un nombre : 2S, 123E dans ce cas, la longueur vaut 1 et le nombre de répétition est égale à la chaine précédent l'instruction transformée en entier
- l'instruction commence par un chiffre ou un nombre suivi d'une instruction entre parenthèses : 2(SO), 45(N3(E3S)) dans ce cas la longueur est égale au nombre de caractères présents à l'intérieur des parenthèses, on stocke la chaine entre parenthèses dans la variable c de la sous instruction et le nombre de répétition est égale à la chaine précédant l'instruction transformée en entier

Nous excluons donc le cas où l'utilisateur rentre autre chose qu'un caractère qui est une instruction, un chiffre ou une parenthèse ouvrante, fermante. Nous excluons aussi le cas ou l'utilisateur ne rentre pas de chiffres avant une parenthèse. Si l'utilisateur oublie une parenthèse, le programme s'arrête, de même si le caractère n'est pas connu du programme en indiquant le caractère inconnu.

2 Comment stocker les informations

La chaine de caractères n'étant pas connue à l'avance et par soucie de stockage, l'utilisation de pointeurs pour l'allocation dynamique de la mémoire était obligatoire, il restait à savoir quel type de structure était

la plus pertinente pour stocker les informations.

Nous avons tout d'abord pensé à utiliser des arbres pour stocker les informations. En effet, chaque instruction peut être composée d'un certain nombre indéterminé de sous instructions, qui visuellement fait penser à un arbre dans lequel chaque nœud comporterait l'instruction en question et les sous branches accrochées à ce nœud pointeraient vers des nœuds contenant les sous instructions. Cependant, nous avons trouvé la structure compliquée et nous nous sommes demandé quelles informations étaient vraiment nécessaires sachant que l'objectif est d'afficher à l'écran des lignes. Pour afficher des lignes, il nous suffit simplement de 2 points et donc de coordonnées, l'instruction, et donc la chaine de caractère, n'est absolument pas nécessaire dans ce processus. L'arbre était donc à exclure. En revanche, une liste, et plus particulièrement une pile, était amplement suffisante pour stocker des coordonnées. Nous définissons ainsi une liste de coordonnées qui sera remplie au fur est à mesure de l'interprétation des chaines de caractères en instruction. Cette liste est de plus simple à envoyer dans une fonction qui affichera à l'utilisateur le résultat. Nous devons aussi penser au instructions spéciales qui permettent de lever et baisser le stylo. L'idée de stocker un booléen dans la liste nous est naturellement venu à l'esprit. En effet, le booléen sera à 1 si le stylo est baissé et 0 si le stylo est levé, cela permettra de savoir à la fonction qui affiche le résultat de savoir s'il faut tracer une ligne ou pas.

Nous définissons un type liste et toutes les fonctions qui permettrons de manipuler cette liste.

Type nœud = structure {coord : TypeCoordonné , ModeEcriture : Booléen , succ : ^noeud } Type liste : ^nœud et TypeCoordonné = structure { x,y : Entier }

La liste sera initialisée sur un premier nœud contenant les coordonnées de l'origine et un booléen vrai pour le modeEcriture.

Nous définissons la fonction remplirListe qui remplira un nouveau nœud des coordonnées calculées à partir de l'interprétation de l'instruction. remplirListe ne peut être ainsi appelé qu'une fois que l'instruction soit suffisamment simple, c'est à dire ne contenant qu'une unique lettre précédée possiblement d'un chiffre ou d'un nombre (N, 23E, 1S). Cette fonction prendra en paramètre le caractère qui sera une direction ou un mode d'écriture en plus du nombre de répétition et de la liste. Le nouveau nœud ainsi créé sera ajouté arbitrairement (et surtout pour des raisons de facilité) en tête de la liste.

3 Comment afficher le résultat

Notre objectif était d'isoler la partie affichage du programme principale au maximum. Ainsi nous aurons plus de liberté pour choisir le type d'affichage. En effet, le code pourra être modifié facilement si l'on décide de changer d'avis sur le mode d'affichage sans avoir à modifier les programmes qui s'occupent de la lecture et de l'interprétation des chaines de caractères. A notre niveau, nous avions connaissance de 2 modes d'affichage. Soit un affichage directement sur le terminal, soit un affichage à l'aide de la bibliothèque SDL. Cette dernière s'est révélé plus pratique, bien que plus compliquée à maitriser et à mettre en place. En effet, un affichage terminal nous aurait demandé de coder nos propres fonctions d'affichage de trait pour faire des lignes par exemple. Le résultat ne serait pas propre et limité car nous ne contrôlons pas la taille du terminal (la fenêtre peut être agrandi et rétréci et le résultat peut donc complétement changer et mal interpréter). En revanche, la bibliothèque SDL dispose d'une simple fonction ligne qui permet d'afficher directement une ligne à partir de coordonnées de 2 points. Nous créons ainsi une fonction affichage qui prendra en paramètre la liste de coordonnées et qui utilisera la bibliothèque SDL.

Lors de l'affichage, la liste sera parcouru de la dernière coordonnée calculée jusqu'au dernier nœud contenant l'origine. Nous nous sommes assuré que le sens de parcours de la liste n'avait aucune influence sur le résultat affiché.

Deuxième partie

Organisation et répartition des tâches

Notre problème étant bien divisé, nous nous sommes d'abord séparé les tâches pour écrire les pseudo codes. Nous nous sommes d'abord alors sur la fonction récursive en supposant que toutes les fonctions définies avant fonctionnaientt. Nous nous sommes ensuite séparé les taches pour faire les différentes fonctions des TAD liste et instructions.

Nous avons pu faire de même pour l'implémentation du code. Les TAD étant bien séparé nous avons pu créer nos propres bibliothèques contenant les fonctions en rapport avec ces TAD. Il a fallu ensuite ajouter le travail de la SDL où une bibliothèque a aussi été créée pour améliorer la visibilité dans le programme d'affichage.

Aymeric	Clément
TAD instruction	TAD liste
LireChaine	remplirListe
	SDL
	MaFonction

Troisième partie

Explications des fonctions principales

Algorithme 1 type instruction

```
TYPE instruction = structure :
    lg : entier
    c : chaine
    rep : entier
FIN

const instructionNulle = {
    lg = 0
    chaine = ”
    rep = 0 }

FONCTION estUneDirection( E x : caractere ) : booleen
DÉBUT
    RETOURNER x == 'N' ou x == 'S' ou x =='E' ou x == 'O'
FIN

FONCTION estUnModeDecriture( E x : caractere ) : booleen
DÉBUT
    RETOURNER x == 'L' ou x =='B'
FIN
```

Les fonctions EstUnChiffre, EstUnModeDecriture et EstUneDirection sont des booléens permettant de reconnaître si le caractère est respectivement un chiffre (0,1,2,3,4,5,6,7,8,9) , un mode d'écriture (B,L) ou une direction (N,O,S,E).

Algorithme 2 lireChaine

```
FONCTION lireChaine( E c : chaîne, E/S instructionEntreParenthese, chiffresAvant : Booléen ) : instruction
DÉCLARATION
    sous_instruction : instruction
    i, k, nbrep, n, nbparenthese ; Entier
    x : caractère
DÉBUT
    sous_instruction ← InstructionNulle
    i ← 0
    k ← 0
    n ← longueur( c )
    x ← c[i]
    nbrep ← 0
    nbparenthese ← 0
    instructionEntreParenthese ← Faux
    chiffresAvant ← Faux
    SI c =/= '' ALORS
        SI estUneDirection( x ) ou estUnModeDecriture( x ) ALORS
            sous_instruction.lg ← 1
            sous_instruction.chaine[i] ← x
            sous_instruction.rep ← 1
        SINON
            TANT QUE estUnChiffre( x ) et ( i + k ) < n FAIRE
                nbrep ← nbrep * 10 * k + charToInt( x )
                k ← k+1
                x ← c[i+k]
                chiffresAvant ← Vrai
            FIN TANT QUE
            SI ( i + k) >= n ALORS
                ECRIRE("erreur aucune instruction")
                sous_instruction ← InstructionNulle
            SINON
                sous_instruction.rep ← nbrep
                i ← i + k
                k ← 0
                SI estUneDirection( x ) ou estUnModeDecriture( x ) ALORS
                    sous_instruction.lg ← 1
                    sous_instruction.chaine[0] ← x
                SINON
                    SI x == '(' ALORS
                        nbparenthese ← 1
                        instructionEntreParenthese ← Vrai
                        TANT QUE nbparenthese =/= 0 et i < n FAIRE
                            i ← i + 1
                            x ← c[i]
                            SI x = '(' ALORS
                                nbparenthese ← nbparenthese + 1
                            FIN SI
                            SI x = ')' ALORS
                                nbparenthese ← nbparenthese - 1
                            FIN SI
                            SI nbparenthese =/= 0 ALORS
                                sous_instruction.chaine[k] ← x
                                sous_instruction.lg ← sous_instruction.lg + 1
                                k ← k + 1
                            FIN SI
                        FIN TANT QUE
```

Algorithme 3 lireChaine(suite)

```
                    SI i >= n ALORS
                        ECRIRE("erreur il manque une parenthese")
                        sous_instruction ← InstructionNulle
                    FIN SI
                SINON
                    ECRIRE("erreur caractère i + k + 1 inconnu")
                    sous_instruction ← InstructionNulle
                FIN SI
            FINSI
        FIN SI
    FIN SI
SINON
    ECRIRE( "erreur chaine vide")
    sous_instruction ← InstructionNulle
FIN SI
RETOURNER sous_instruction
FIN
```

Nous avons précédemment expliqué le principe de la structure instruction. Son principe va être utilisé de manière récursive dans la fonction MaFonction. Mais pour se faire il faut déjà établir comment lire une à une ces instructions dans la liste chaînée grâce à la fonction lireChaine. Cette fonction prend en paramètre une chaîne et deux booléens en entrée/sortie qui permettront à la procédure MaFonction de savoir si l'instruction lu était entre parenthèses et si elle possédait un chiffre ou un nombre avant. Premièrement, on va lire le premier caractère dans la liste chaînée, si c'est une direction (Est = E, Ouest = O, Nord = N ou Sud= S) ou un mode d'écriture (Levé = L ou Baissé = B) cela constitue dès lors une instruction complète qu'on peut retourner, celle ci est de longueur 1 et est répétée 1 fois. Autrement, cela signifie qu'on a soit un nombre soit une parenthèse pour le premier caractère de l'instruction. Dans le cas où le caractère est un chiffre, il se peut que les prochains caractères soient aussi des chiffres, il faut donc lire la totalité des caractères formant ce nombre afin de le stocker dans le nombre de répétition de l'instruction (appelée rep dans la structure de instruction). Il suffit donc de stocker les chiffres un par un dans une variable en faisant la conversion de caractère à nombre, à chaque itération on multiplie cette variable par dix jusqu'à ce qu'on tombe sur un autre caractère qui n'est pas un chiffre. Si le caractère qui suit ce nombre est une instruction, on la stocke directement dans la chaîne de notre instruction et on peut retourner l'instruction. Dans le cas contraire le caractère est une parenthèse, on veut alors stocker l'entièreté de l'instruction contenu dans cette parenthèse. Pour se faire on va compter le nombre de parenthèses ouvrantes et fermantes , si la somme des parenthèses ouvrantes moins la somme des parenthèses fermantes vaut zéro alors on a atteint la fin de l'instruction de la première parenthèse, qu'on va donc stocker dans la chaîne. Il faut aussi prendre en compte les différentes erreurs pouvant se produire dans le programme si la liste chaînée n'est pas conforme auquel cas il faudra renvoyer qu'une erreur s'est produite lors de la saisie, notamment si un caractère non autorisé est entré , si il manque une parenthèse ou si la chaîne est vide.

Algorithme 4 remplirListe

```
PROCÉDURE remplir_liste( E/S l : liste , E c : caractere , E n : entier )
DÉCLARATION
    l' : liste
DÉBUT
    l' ← l
    CAS c // switch case en C
        E : ajouterEnTete( l , l'^.x + n , l'^.y , l'^.modeEcriture )
        O : ajouterEnTete( l , l'^.c - n , l'^.y , l'^.modeEcriture )
        N : ajouterEnTete( l , l'^.x , l'^.y + n , l'^.modeEcriture )
        S : ajouterEnTete( l , l'^.x , l'^.y - n , l'^.modeEcriture)
        L : ajouterEnTete( l , l'^.x , l'^.y , 0 )
        B : ajouterEnTete( l , l'^.x , l'^.y , 1 )
    FIN CAS
FIN
```

remplirListe est une fonction qui prend en entrée une liste, un caractère (qui peut être une direction ou un mode d'écriture) et un entier donnant le nombre de répétition. Cette fonction permet de stocker les directions données ainsi que leur mode d'écriture dans une liste dont la structure est la suivante : 2 entiers pour la position en abscisse et en ordonnée ainsi qu'un booléen donnant le mode d'écriture (0 quand le « crayon » et levé et 1 quand le « crayon » est baissé). Pour connaître la nouvelle position de notre « crayon » on regarde le dernier élément ajouté en tête à la liste (qu'on va lire grâce à un pointeur) et on incrémente n fois la direction entrée dans le cas d'une direction. Si le caractère est un mode d'écriture, on conserve la position et on change le booléen en fonction du caractère B ou L. Puis on ajoute en tête cette structure dans la liste.

Algorithme 5 MaFonction

```
PROCÉDURE MaFonction( E/S l : liste , E c : chaine )
DÉCLARATION
    sous_instruction : instruction
    i, indiceChaine : entier
    ch_suiv : chaine
    p : liste
    instructionEntreParenthese, chiffreAvant : Booléen
DÉBUT
    p ← l
    indiceChaine = 0 ;
    SI c =/= " " ALORS
        sous_instruction ← lireChaine( c , instructionEntreParenthese, chiffreAvant)
        SI sous_instruction.lg > 1 ALORS
            POUR i ← 1 à sous_instruction.rep FAIRE
                MaFonction( p , sous_instruction.c )
                l←p
            FIN POUR
        SINON
            remplir_liste( l , sous_instruction.chaine[0] , sous_instruction.rep )
        FIN SI
        indiceChaine ← sous_instruction.lg
        SI chiffreAvant ALORS
            indiceChaine ← indiceChaine + longueur(intToString(sous_instruction.rep))
        FIN SI
        SI instructionEntreParenthese ALORS
            indiceChaine ← indiceChaine + 2
        FIN SI
        SI c[indiceChaine] =/= "" ALORS
            ch_suiv ← &c[indiceChaine]
            MaFonction( p , ch_suiv )
            l←p
        FIN SI
    FIN SI
FIN
```

La procédure MaFonction contient le principe récursif du programme. Tout d'abord nous initialisons une variable indiceChaine à 0, cette variable nous permettra de connaitre la position dans la chaine pour déterminer si la chaine est fini ou pas. La condition d'arrêt de l'algorithme est quand la chaine est vide. Si la chaîne n'est pas vide alors on stocke les informations d'une instruction grâce à la fonction lireChaine à laquelle nous envoyons la chaine et les deux booléen instructionEntreParentheses et chiffreAvant destiné à être modifiés. Dans le cas ou la longueur de l'instruction est plus grande que 1 (l'instruction était entre parenthèse) alors nous envoyons la chaîne contenu dans la sous instruction dans MaFonction et nous répétons jusqu'à atteindre le nombre de répétition de cette sous instruction. Sinon, cela signifie que l'instruction lu était simple et donc nous pouvons envoyer cette instruction à remplirListe pour stocker les informations de l'instruction. Dans les 2 cas, il faut vérifier ensuite si la chaine n'est pas terminée, c'est à dire si elle contient d'autres instructions. C'est pour cela que la variable indiceChaine nous est utile. On incrémente cette variable à la longueur de la chaine de la sous instruction. Si cette sous instruction était précédée d'un chiffre ou d'un nombre alors nous convertissons ce nombre en chaine et ajoutons la longueur de cette chaine à indiceChaine. Si la chaine était une instruction entre parenthèses alors on ajoute la place prise par ces parenthèses, c'est à dire 2. Nous attribuons alors à la variable ch_suiv la partie restante de la chaîne. Si cette chaîne n'est pas vide alors il faut continuer le procédé en envoyant ch_suiv dans MaFonction.

Quatrième partie

Quelques exemples

Dans cette partie, nous exploiterons des jeux de données bien choisie pour determiner la validité de notre programme.

Commençons par les erreurs provoquées lors de la saisie de la chaine de l'utilisateur.

```
[clement@laptop CODE_FINI_PAS_TOUCHER]$ ./main
entrez les instructions
2NE12
erreur aucune instruction
[clement@laptop CODE_FINI_PAS_TOUCHER]$ ./main
entrez les instructions
NS4(OS8(NO)
erreur il manque une parenthese
[clement@laptop CODE_FINI_PAS_TOUCHER]$ ./main
entrez les instructions
2NELBSOVNS
erreur caractère V inconnu
[clement@laptop CODE_FINI_PAS_TOUCHER]$
```

Puis quelques exemples de dessins

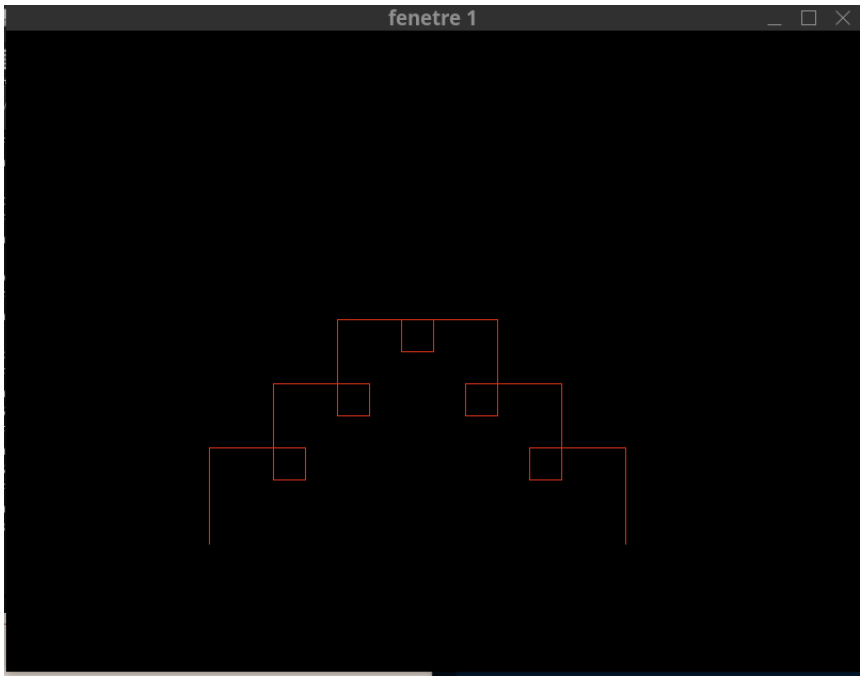

FIGURE 1 – 3(ON3E3S)L7O6NB3(EN3O3S)

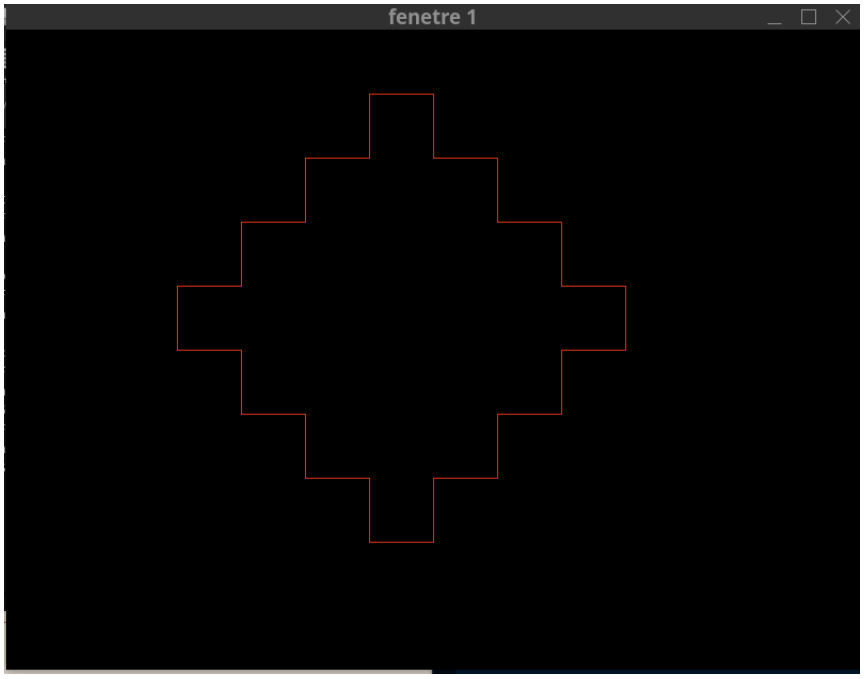

FIGURE 2 – L8OB4(2N2E)3(2S2E)4(2S2O)3(2N2O)

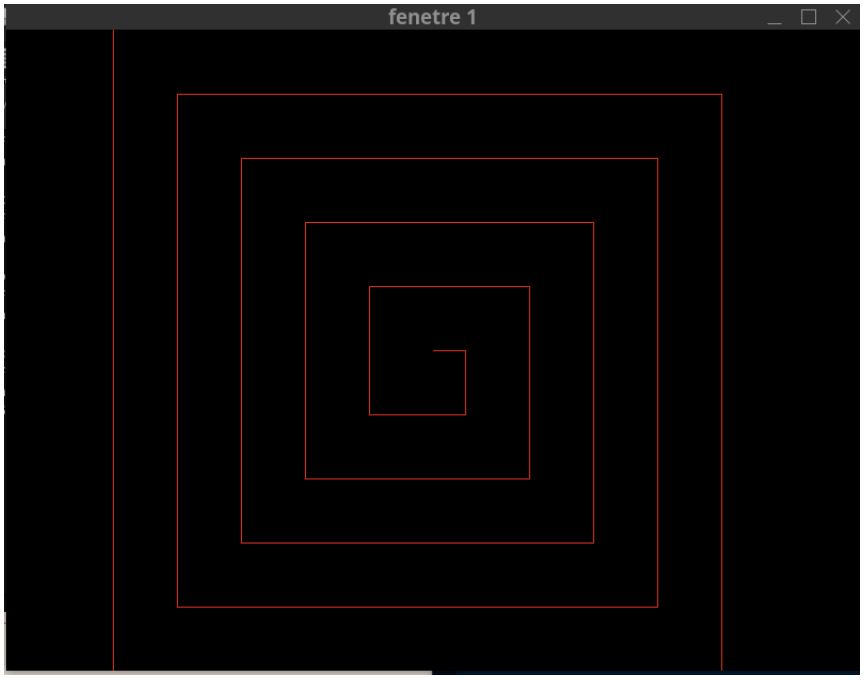

FIGURE 3 – L10N10OB20S19E18N17O16S15E14N13O12S11E10N9O8S7E6N5O4S3E2N1O

Cinquième partie

Perspective d'amélioration et conclusion

Les améliorations pour ce programme peuvent se faire. Nous aurions pu par exemple permettre à l'utilisateur de rentrer des caractères en minuscule pour les transformer en majuscules et donc en caractères compréhensible par le programme. Au lieu de s'arrêter directement après avoir détecté une erreur, le programme pourrait continuer, se mettre à la position de l'erreur et redemander à l'utilisateur la suite de la chaine. Plusieurs améliorations esthétiques peuvent aussi être faites sur l'affichage, comme par exemple changer la couleur, la taille du trait, la taille de la fenêtre...

Ce projet fût très intéressant, le caractère visuel du projet nous a particulièrement touché. Le programme, utilisant beaucoup de pointeurs et de bibliothèques que nous n'avions pas l'habitude d'utiliser, a permis de consolider nos connaissances sur les pointeurs et les listes chaînées.