Rapport IPI

Gautier POURSIN

6 Janvier 2020

Introduction

Dans ce projet, nous devions implanter un interpréteur et un débogueur dans un langage de programmation appelé prog2D: cela correspond à une grille de caractères où un curseur se déplace dans la grille. Chaque caractère correspond à une instruction donnée. Le curseur peut se déplacer dans 8 directions possibles et le curseur se déplace toujours dans la même direction, tant qu'un caractère indiquant une autre direction n'est pas rencontré. En parallèle du déplacement du curseur, nous implémenterons une pile d'entier qui s'empile et se dépile en fonction des instructions données. Ensuite, le débogueur permettra à l'utilisateur d'entrer différentes commandes et d'avoir l'état du programme à chaque étape.

Dans un premier temps, nous allons nous focaliser sur ls structures de travail choisies, puis sur l'interpréteur: quels ont été les problèmes rencontrés? Comment les résoudre? Comment l'améliorer?

Sommaire

1 Les structures choisies

```
struct position{
  int x,y;
  // x:numero de lignes, y:numero de colonnes

};
typedef struct position position;
```

position.JPG

Figure 1: La structure correspondant à la position du curseur

Cette structure est composé de 2 entiers **x** et **y** correspondant respectivement au numéro de ligne et de colonnes où le curseur se situe dans la matrice. Cela permet de suivre l'avancement du curseur. Il sera nommé **position**.

```
//w:nbre de colonne, h=nbredeligne
struct array2D{
  int h,w;
  int **content;
};
typedef struct array2D array2D;
```

Figure 2: La structure correspondant à une matrice

Cette structure est composée de 2 entiers **w** et **h** corespondant au nombre de lignes et de colonnes de la matrice. De plus, elle contient un pointeur de pointeur correspondant à des entiers. En effet, la matrice dans laquelle sera placé le fichier est composé de code ASCII de chaque caractère. Il sera nommé **array2D**.

```
struct stack {
  int *t;
  int top;
  int size;
};
typedef struct stack stack;
```

Figure 3: La structure correspondant à une pile

Cette structure est composée d'un entier **top**, correspondant au haut de

la pile, d'un entier **size** correspondant à la taille de la pile (j'ai choisi de travailler avec des piles qui ont des tailles variables.) et enfin d'un pointeur **t** qui correspond à l'entier pointé par la pile. Il sera nommé **stack**.

```
struct ensemble{
  position curseur;
  array2D tableau;
  int directory;
  int caractere;
  stack s;
};
typedef struct ensemble ensemble;
```

Figure 4: La structure globale nécessaire au bon déroulement du programme

Cette structure me permet de regrouper tous les elements nécessaire pour avoir un déroulement de l'interpréteur plus simpliste, permettant d'avoir en argument qu'un élément **ensemble**, qui contient une pile, la matrice de caractère, la position du curseur, le caractère sur lequel on se situe et la direction du déplacement. La direction est représenté par un entier.

2 Interpréteur

2.1 Les problèmes rencontrés

Pour réaliser l'interpréteur, la première chose à faire est de mettre un fichier de caractère dans un matrice. Pour cela, il faut avoir la taille de la matrice nécessaire pour y mettre tous le fichier, sans allouer de mémoire en trop. Il faut donc avoir la taille exacte du fichier. Cette taille (le nombre de lignes et de colonnes) se trouve sur la première ligne du fichier.Il faut donc extraire la premiere ligne seulement à l'aide de la fonction **fgets**, qui renvoie une chaine de caractère. C'est pourquoi il est nécesssaire de repérer le premier espace dans la chaine de caractère pour avoir ensuite d'un côté, le nombre de lignes, et de l'autre, le nombre de colonnes. Cela peut être fait avec la fonction **permieres-pacedanschaine**.Ensuit, à l'aide d'une fonction d'extraction de chaîne, notée **extractionchaine** dans mon programme, je peux extraire la première partie de la chaine(ie la partie de la première ligne avant l'espace) afin avoir le nombre de colonne. La chaine extraite étant un nombre noté en caractère, il faut le convertir en entier avec la fonction **atoi**, utilisée dans la fonction **conversion**. Avec ces différentes fonctions, j'obtiens donc la taille nécessaire pour ma matrice.

Ensuite, l'objectif était de mettre le fichier dans la matrice à l'aide de la

fonction **ouverturefichier**. Cette étape à été l'une des plus difficile. Il faut d'abord allouer la mémoire nécessaire pour la matrice, ce qui est possible grâce à la fonction **create**. Il faut ensuite remplir la matrice en mettant un caractère par case. J'ai 2 possibilités qui s'offrent à moi: utiliser la fonction **fgets**, qui permet d'extraire le fichier ligne par ligne puis ensuite de repartir la ligne dans une ligne de la matrice, ou utiliser la fonction **fgetc**, qui permet d'extraire le fichier caractère par caractère. J'ai essayé dans un premier temps la fonction **fgets**. Cependant, je n'ai pas réussi à repartir la ligne du fichier dans la ligne de la matrice. J'avais utilisé la technique suivant:

```
for(int i=0;i¡=conversion(nombredeligne); i++)
for (int k=0; k¡=conversion(nombredecolonnes);k++)
fichierdansmatrice.content[i]=fgets(chaine,256,fichier)[k];
```

Cette technique s'est avérée être un échec. C'est pourquoi j'ai opté pour la fonction **fgetc**.

```
for(int i=0;i¡=conversion(nombredeligne); i++)
for (int k=0; k¡=conversion(nombredecolonnes);k++)
fichierdansmatrice.content[i][k]=fgetc(fichier);
```

Cependant, j'obtiens une matrice d'entiers et non de caractères. La matrice est remplie des valeurs ASCII de chaque caractère, il faudra donc lors de mon switch travailler sur les entiers ASCII et non les caracères. Ma fonction **printarray2D** affiche cependant bien la matrice de caractères, comme attendu dans l'énoncé.

3 Conclusion

"I always thought something was fundamentally wrong with the universe" [1]

References

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy.* San Val, 1995.