ECOLE NATIONALE SUPÉRIEUR D'INFORMATIQUE POUR L'INDUSTRIE ET L'ENTREPRISE

Rapport : IPI 2019-2020

Akarioh Samir

28 décembre 2019

TABLE DES MATIÈRES

1 Démarche générale

1.1 Les structures de données utilisées ainsi que leurs fonctions associées

— Pile

La pile est une structure importante pour le lancement des programmes,etant donne que le fonctionnement des programmes repose sur un système de déplier empiler. J'avais le choix entre une structure de tableau de taille fixe ou une liste chaînée. La meilleure solution était de mettre en place une liste chaine étant donne qu'on ne peut pas savoir le nombre d'appels a la fonction empiler et déplier en avance.

```
typedef struct Element Element ;
struct Element
{
    int nombre ;
    Element * suivant ;
} ;

typedef struct Pile Pile ;
struct Pile
{
    Element * premier ;
} ;
```

Les différentes opérations possible sur la pile sont :

— Créer une pile vide.

— Empiler un nouveau élément au début de la pile.

— Supprimer le sommet de la pile retourne 0 si la pile est vide.

— Afficher la pile.

— Supprimer tout le contenu de la pile.

— Matrice :

J'ai choisi par souci de simplicité de copier les données du fichier .p2d dans une matrice de char** que j'ai alloué grâce à cette fonction si et seulement si le fichier .p2d existe et qu'on a au moins un argument donné au programme pour éviter les fuites mémoires .

```
Matrice allocation (int colonne , int ligne ){i
   nt i ;
   Matrice grille =( Matrice ) malloc ( ( ligne +1) * sizeof ( char * ) ) ;
   for ( i =0; i < ligne ; i ++)
      grille [i]= calloc ( colonne , sizeof ( Matrice )
   ) ; return grille ;
}
```

Afin de remplir ma grille j'ouvre mon fichier et récupère le nombre de lignes et de colonne via fscanf

```
1 fscanf(fichier,"%d %d\n",colonne,ligne)
```

J'appelle ensuite la fonction remplissage afin de remplir mon tableau via fgetc pour lire caractère par caractère et le copier dans la grille au bon endroit.

```
1  Matrice remplisage(int *colonne,int *ligne,char *argv[]){
2
3    FILE *fichier;
4    int j,i;
5    fichier=fopen(argv[1],"r");
6    fscanf(fichier,"%d %d\n",colonne,ligne);
7    Matrice grille=allocation(*colonne,*ligne);
8
9    for(i=0;i<*ligne;i++)
10    {
11     for(j=0;j<*colonne;j++)
12          grille[i][j]=fgetc(fichier);
13     fgetc(fichier);
14     }
15   fclose(fichier);
16   return grille;
17 }
```

Les fonctions utiles a l'interpréteur et au débogueur repose sur cette grille :

— Exécute le fichier.p2d via la fonction lancement(l'interpréteur lance directement cette fonction et le débogueur la lance si l'utilisateur entre run) .

— Step n avance de n étapes, c'est-à-dire exécute l'instruction sous le curseur et déplace ce dernier (le débogueur la lance si l'utilisateur entre step n.

— Prec n recule de n étape, c'est-à-dire revient dans l'état précédent la n eme instruction..

— Breakpoint x y ajoute un point d'arrêt à la position (x,y) ; lors d'une commande ultérieure run, step n ou prec n, force l'arrêt si la position du curseur est dans la liste des points d'arrêts..

— Remove x y retire le point d'arrêt à la position (x,y) ; sans effet si le point d'arrêt à cette position n'existait pas.

— Restart : retourne à l'état initial le débogueur la lance si l'utilisa- teur entre restart.

— Quit : quitte le débogueur si l'utilisateur entre quit.

— Si l'utilisateur entre une ligne vide, on répète la dernière commande pour le débogueur.

1.2 AFFICHAGE

Dans le débogueur on doit afficher la grille entourer de repère,l'état de la pile et le code ASCII associe à chaque élément de la pile et cela à chaque étape du programme et la position des curseurs. Cela est fait par la fonction affichage qui nous renvoie ça :

1.3 COMMANDE A EXÉCUTÉ EN FONCTION DU CARACTÈRE RENCONTRÉ DANS LA GRILLE

+	Addition : On dépile deux entiers a et b, et on empile a+b
-	Soustraction : On dépile deux entiers a puis b, et on empile b-a
*	Multiplication : On dépile deux entiers a et b, et on empile a×b
:	Division entière : On dépile deux entiers a puis b, et on empile la division euclidienne de b par a. Si a vaut 0, on empile 42.
%	Reste : On dépile deux entiers a puis b, et on empile le reste de la division euclidienne de b par a. Si a vaut 0, on empile 0xbadc0de.
!	Non logique : On dépile un entier. Si c'est 0 on empile 1, sinon on empile 0.
`	Plus grand que : On dépile deux entiers a et b, et on empile 1 si b>a, 0 sinon.
>	Change la direction pour aller à droite.
<	Change la direction pour aller à gauche.
^	Change la direction pour aller en haut.
v	Change la direction pour aller en bas.
?	Choisit une direction au hasard (diagonales comprises)
'	Dépile un entier. Change la direction en fonction du reste de la division euclidienne de cet entier par 8 : 7 0 1 6 2 5 4 3
]	Tourne à gauche de 45°.
[	Tourne à droite de 45°.
_	Conditionnelle horizontale : dépile un entier, change la direction vers la droite si c'est 0, vers la gauche sinon.
|	Conditionnelle verticale : dépile un entier, change la direction vers le bas si c'est 0, vers le haut sinon.
/	Conditionnelle diagonale : dépile un entier, change la direction vers le haut et la droite si c'est 0, vers le bas et la gauche sinon.
\	Conditionnelle diagonale : dépile un entier, change la direction vers le bas et la droite si c'est 0, vers le haut et la gauche sinon.
"	Passe en mode *chaîne de caractère* : empile la valeur ASCII de tous les caractères suivants dans la direction courante jusqu'à atteindre un autre "
=	Duplique le sommet de la pile. Si la pile est vide, empile deux fois l'entier 0.
$	Échange les deux valeurs au sommet de la pile. Si la pile contient un seul élément, empile l'entier 0 au-dessus. Si la pile est vide, empile deux fois l'entier 0.
;	Retire la valeur au sommet de la pile. Sans effet si la pile est déjà vide.
.	Dépile un entier et l'affiche comme un entier.
,	Dépile un entier et affiche le caractère correspond au code ASCII de sa valeur tronquée à l'intervalle [0;255].
#	Pont : dépile un entier n et saute au-dessus des n prochains caractères (c'est-à-dire qu'on avance sans exécuter les instructions auxquelles ils correspondent éventuellement).
g	*Get* : dépile deux entiers x et y, et empile la valeur (code ASCII) de la grille à la position (x,y). Si la position est en dehors de la grille, empile 0.
p	*Put* : dépile trois entiers x, y et z, et écrit z (code ASCII de la valeur tronquée entre 0 et 255) dans la grille à la position (x,y). Si la position est en dehors de la grille, ne fait rien.
&	Demande à l'utilisateur d'entrer un entier, et empile cet entier.
~	Demande à l'utilisateur d'entrer un caractère, et empile la valeur ASCII de ce caractère.
@	Fin du programme.
0 à 9	Empile l'entier correspondant.
' '	Espace : Ne fait rien.

2 OPTIMISATION DU CODE

Avec le recul, j'ai eu quelques idées pour avoir un code plus propre.

Pour alléger visuellement le code, il faudrait fusionner la fonction step et lancement en une seule il suffit de mettre en place une variable compteur que l'on incrémente à chaque tour et une variable fin de type int qui vaut zéro si on veut faire step et 1 si on a l'instruction run.

De plus,strtol me permet de savoir si l'utilisateur entre bien un entier cependant si on rentre 4i4 strtol nous renvoie 4 je ne l'ai pas modifié par manque de connaissance .

Dans la fonction lancement et step pour les chiffres de 0 a 9 a la place de faire plusieurs if et else if pour chaque numero on aurai pu faire empiler(maPile,(int)grille[*i][*j]-48); car 48 est le code ASCII du 0.J'ai choisi de ne pas le faire pour eviter des bugs.