Compte-rendu projet INF203 : INF-01 Rakotomalala Jonathan Mampitony Onintsoa :

Partie I :bash

Cow_kindergaten :

```
cowsay 1
sleep 1
clear
cowsay 2
sleep 1
clear
cowsay 3
sleep 1
clear
cowsay 4
sleep 1
clear
cowsay 5
sleep 1
clear
cowsay 6
sleep 1
clear
cowsay 7
sleep 1
clear
cowsay 8
sleep 1
clear
cowsay 9
sleep 1
clear
```

cowsay -T U 10

Sortie finale:

```
____
< 10 >
 ----
    \  ^__^
     \ (oo)\_______
       (__)\     )\/\
        U ||----w |
          ||    ||
```

Cow primaryschool

On initialise c à 0, et tant que c est inférieur au nombre la vache dit c+1 et attend une minute et on supprime les sorties puis à la fin la vache dit le nombre num .

```
read n
c=0
num=$(expr $n + -1)
while [ $c -lt $num ]
do
  ((c++)) #c augmente de 1 à chaque passage dans la boucle
  cowsay $c #on affiche c
  sleep 1
  clear
done
cowsay -T U $n
```

Sortie:

Sortie finale:

```
____
< 10 >
 ----
    \  ^__^
     \ (oo)\_______
```

```
        (__)\       )\/\
         U ||----w |
           ||    ||
```

cow college

Les nombres de Fibonacci sont des nombres qui sont égales à la sommes des deux nombres qui les précèdes commençant par 0.

Pour trouver le nombre d'après dans la suite de Fibonacci il suffit d'additionner tous les nombres précédents dans la liste deux par deux comme suit: F2=F1+F0.

```
read n
c1=0
c2=1
num=$(expr $n - 2)
for (( i=0; i<$num; i++))
do
  fn=$(expr $c1 + $c2) #calcul du prochain nombre dans la suite
  cowsay $fn
  sleep 1
  c1=$c2#on enregistre c2 et fn dans c1 et c2 car ils permettent de faire
  c2=$fn# le calcul du prochain nombre dans la suite
done
```

Sortie: suite de fibonacci de 1 à 5

rakotjon@im2ag-turing:[~/INF203/projet_cowsay]: ./cow_college

5

```
 ___
< 1 >
 ---
    \  ^__^
     \ (oo)\_______
       (__)\       )\/\
         ||----w |
         ||    ||
 ___
< 2 >
 ---
    \  ^__^
```

```
      \ (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||

 ___
< 3 >
 ---
     \  ^__^
      \ (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

cow highschool

On calcul la suite du carré des nombres de 1 à n^2, il suffit de multiplier un nombre par lui-même.

```
read n
c=0
num=$(expr $n + -1)
while [ $c -le $num ]
do
  ((c++))#c augmente de 1 jusqu'à ce qu'il soit égal à num
  cowsay $(expr $c \* $c) #affichage et calcul du carré du nombre
  sleep 1
  clear
done
```

Sortie:Suite des 10 premiers nombres aux carrés

rakotjon@im2ag-turing:[~/INF203/projet_cowsay]: ./cow_highschool

10

```
 ____
< 1 >
 ----
     \  ^__^
      \ (oo)\_______
```

```
        (__)\      )\/\
          ||----w |


____
< 4 >
 ----
      \  ^__^
       \ (oo)\_______
         (__)\      )\/\
           ||----w |
____
< 9 >
 ----
      \  ^__^
       \ (oo)\_______
         (__)\      )\/\
           ||----w |
            ||    ||
____
< 16 >
 ----
      \  ^__^
       \ (oo)\_______
         (__)\      )\/\
           ||----w |
            ||    ||
____
< 25 >
 ----
      \  ^__^
       \ (oo)\_______
```

```
        (__)\      )\/\
          ||----w |
           ||     ||
____
< 36 >
 ----
      \  ^__^
       \ (oo)\_______
         (__)\      )\/\
           ||----w |
            ||     ||
____
< 49 >
 ----
      \  ^__^
       \ (oo)\_______
         (__)\      )\/\
           ||----w |
            ||     ||
____
< 64 >
 ----
      \  ^__^
       \ (oo)\_______
         (__)\      )\/\
           ||----w |
            ||     ||
____
< 81 >
 ----
      \  ^__^
```

```
      \ (oo)\_______
        (__)\       )\/\
          ||----w |
           ||    ||

____
< 100 >
 ----
     \  ^__^
      \ (oo)\_______
        (__)\       )\/\
         U ||----w |
           ||    ||
```

Cow university

La vache doit afficher la suite de nombre premier inférieurs à n.

On vérifie donc chaque nombre de 1 à n et on affiche seulement ceux qui sont premiers.

Pour cela on utilisé deux boucles pour pouvoir diviser les chiffres de 1 à n par des chiffres de 1 à n.

Pour savoir si un nombre est premier il suffit de vérifier si le reste de la division du nombre avec lui-même et avec 1 est seulement égale à 0.

```
#!/bin/bash
echo "entrez n"
read n
m=2
while [ $m -le $n ]
do
  i=2 flag=0
  while [ $i -le $(expr $m / 2) ] )  #si un nombre est divisible par un nombre
  do  #entre 2 et n/2 il n'est pas premier
    if [ $(expr $m % $i) -eq 0 ] #on sort de la boucle et flag=1
    then                         #si m modulo i est égal à 0
      flag=1 break
    fi
    i=$(expr $i + 1)  #sinon on passe au nombre suivant
  done
    if [ $flag -eq 0 ] )  #on affiche le chiffre quand flag =0

    then
        cowsay $m
```

```
        sleep 1
        clear
    fi
  m=$(expr $m + 1) #on passe au nombre suivant si flag n'est pas égal à 0
done
```

Sortie:

rakotjon@im2ag-turing:[~/INF203/projet_cowsay]: ./cow_university

```
___
< 2 >
 ---
    \  ^__^
     \ (oo)\_______
       (__)\     )\/\
         ||----w |
         ||    ||
 ___
< 3 >
 ---
    \  ^__^
     \ (oo)\_______
       (__)\     )\/\
         ||----w |
         ||    ||
```

smart_cow

On utilise la commande -T et -e pour pouvoir afficher le calcul à faire à la place des yeux et le résultat à la place de la langue, et on utilise cut pour extraire les chiffres dans le calcul.

```
echo '+ - / * ?' )
read operateur
echo Entrez un calcul numérique
read calcul
pre=$(echo $calcul | cut -d$operateur -f 1) #on coupe calcul en 2
deu=$(echo $calcul | cut -d$operateur -f 2) #à partir de l'opérateur
cowsay -e $(expr $pre $operateur $deu) -T P $pre $operateur $deu
```

```
rakotjon@im2ag-turing:[~/INF203/projet_cowsay]: ./smart_cow.sh


+ - / * ?

+

Entrez un calcul numérique

2+2

 _______

< 2 + 2 >

 -------

    \  ^__^

     \ (4)\_______

       (__)\     )\/\

        P ||----w |

          ||    ||
```

Crazy cow

On reprend les suite de Fibonacci et les nombres premiers et on fait réagir la vache en fonction de la nature du nombre que l'on obtient dans la suite de Fibonacci de 1 à n.

Ici on doit prendre un n supérieur ou égal à 3 pour entrer dans la boucle for et pouvoir faire les calculs et les affichages.

```
read n
c1=0
c2=1
num=$(expr $n - 2)
for (( i=0; i<$num; i++))
do
  fn=$(expr $c1 + $c2) #nombre dans la suite de Fibonacci
  craz=$(expr $fn % 2) #reste de la division entière de fn par 2
  if [ $craz -eq 0 ] #si le nombre de fibonacci est paire la vache dit ceci
  then
       cowsay -T ω -e ☆☆ $fn
       sleep 1
       clear
  elif [ $craz -ne 0 ]
  then
       cowsay -T $c1 -e ╥╥ $fn 'aïeeeeeeueuu'
       sleep 1
       clear
  else
       cowsay $fn
```

```
        sleep 1
        clear
    fi
    c1=$c2
    c2=$fn
done
```

Sortie:

rakotjon@im2ag-turing:[~/INF203/projet_cowsay]: ./crazy_cow

3

```
 _________________
< 1 aïeeeeeeueuu >
 -----------------
     \  ^__^
      \ (╥╥)\_______
        (__)\      )\/\
         0 ||----w |
           ||    ||
```

Partie II:C

New_cow.c

Dans cette deuxième parties on crée d'abord la fonction affiche_vache() qui affiche la vache.

Puis on crée une fonction qui lit les arguments entrées, pour cela on a choisit la méthode avec argc et *mod[] pour lire les arguments en ligne de commandes.

Quand on trouve des arguments ici mod[n] qui modifie la vache on copie la suite mod[n+1] sur la partie correspondante sur la vache, sinon la chaîne de caractères (mod[n]) et affiché dans la fonction bulle() qui affiche la bordure du haut « _ » jusqu'à ce qu'il soir égal à longueur des paroles puis passe à la ligne et affiche « < » ,les paroles et « > » pour ensuite passer à la ligne et affiché les bordure du bas « - » jusqu'à ce qu'il soit égal à la longueur des paroles.

La nouvelle fonctionnalité que j'ai implanté est la fonctionnalité -z qui transforme la vache en zébu, le zébu est un boeuf qui possède une bosse et de grandes cornes, pour cela on a ajouté deux variables à la vache char corn[50] qui correspond aux cornes et bos [10] qui correspond à la bosse.

La fonction goto(x,y) fait déplacer l'affichage d'une sortie, et la fonction update() permet de mettre à jours les sorties pour qu'ils ne s'enchaînent pas à la suite. Cependant on a remarqué que les coordonnées était inversé pour cette fonction parce que x correspond aux ordonnées et y à l'abscisse.

En utilisant ses deux fonctions j'ai pu faire l'animation d'une vache qui saute d'une longueur de 20.

Et j'ai ajouté cette fonctionnalité, ainsi l'animation s'enclenche quand l'argument -a est trouvé dans la ligne de commande.

L'animation qu'on voulait initialement faire est des chaînes de « ____ ____ » qui glissent pour donner l'illusion d'une vache qui glisse et qui saute, cependant je n'ai pas réussit et pour ne pas prendre trop de retard j'ai choisit de faire une vache qui saute.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

unsigned int m;
unsigned int longueur=0;
char eyes[6]="oo";
char tongue[1]=" ";
char top[50]="                        ";
char corn[50]="^__^";
char bos[10]="__";
int bulle=0;
int animate=0;

void update(){ printf("\033[H\033[J");}

void gotoxy(x,y){
    printf("\033[%d;%dH",x,y);
}

void affiche_vache(void){
    printf("                %s\n\
            \\    %s\n\
             \\   (%s)\\ _%s___\n\
                 (__)\\         )\\/\\\n\
                  %s  ||----w |\n\
                      ||     ||\n",top,corn,eyes,bos,tongue);
}



// bulle
void dire(int argc, char *mod[]){
    printf(" ");
    for(int n=1;n<=longueur;n++){
        printf("_");//bordure de haut de la bulle
    }
```

```
    if (bulle==0){
        printf("\n< ");//ouverture de la bulle à la ligne
    }
    else if (bulle==1){
        printf("\n( ");//sinon on affiche une parenthèse
    }
    for(int n=1;n<argc;n++){
        int i=0;
        if (!strcmp(mod[n],"-e")||!strcmp(mod[n],"--eyes")){
            i=1;
            n++;
        }
        else if (!strcmp(mod[n],"-t")||!strcmp(mod[n], "--tired")){
            i=1;
        }
        else if (!strcmp(mod[n],"-z")||!strcmp(mod[n],"--zebu")){
            i=1;
        }
        else if (!strcmp(mod[n],"-g")||!strcmp(mod[n],"--greedy")){
            i=1;
        }
        else if (!strcmp(mod[n],"-b")||!strcmp(mod[n],"--borg")){
            i=1;
        }
        else if (!strcmp(mod[n],"-y")||!strcmp(mod[n],"--young")){
            i=1;
        }
        else if (!strcmp(mod[n],"-s")||!strcmp(mod[n],"--stone")){
            i=1;
        }
        else if (!strcmp(mod[n],"-d")||!strcmp(mod[n],"--dead")){
            i=1;
        }
        else if (!strcmp(mod[n],"-p")||!strcmp(mod[n],"--paranoia")){
            i=1;
        }
        else if (!strcmp(mod[n],"-w")||!strcmp(mod[n],"--what")){
            i=1;
        }
        else if (!strcmp(mod[n], "-a")||!strcmp(mod[n], "--animate")){
            i=1;
        }
        else if (i==0){
            printf("%s ",mod[n]); //affichage des paroles de la vache
        }
    }

    if (bulle==0){
        printf(">\n ");//fermeture de la bulle après l'affichage des paroles
```

```
    }                   //puis passage à la ligne
    else if ( bulle==1){
        printf(")\n ");//sinon on affiche une parenthèse

    }
    for(int n=1;n<longueur;n++){
        printf("-");//bordure du bas de la bulle
    }
    printf("\n");#on passe à la ligne
}


int main(int argc, char *mod[]){
    // modifications sur la vache
    for (int n=1;n<argc;n++){
        if (!strcmp(mod[n], "-e")||!strcmp(mod[n],"--eyes")){
            m=(n+1); //si on a -e en argument on prend la longueur du prochain
            if (strlen(mod[m])>=1){ //chaine de caractère et on le copie
                    strcpy(eyes, mod[m]); //à la place des yeux de la vache

            }
            else{
                printf("erreur");
            }
        }
        else if ( !strcmp(mod[n], "-T") ||!strcmp(mod[n], "--tongue")){
            m=(n+1);
            if (strlen(mod[m])==1){
                strcpy(tongue, mod[m]);
            }
        }
        else if ( !strcmp(mod[n], "-t") ||!strcmp(mod[n], "--tired")){
            strcpy(eyes, "--");
        }
        else if ( !strcmp(mod[n], "-g") ||!strcmp(mod[n], "--greedy")){
            strcpy(eyes, "$$");
        }
        else if ( !strcmp(mod[n], "-p") ||!strcmp(mod[n], "--paranoia")){
            strcpy(eyes, "@@");
        }
        else if ( !strcmp(mod[n], "-d") ||!strcmp(mod[n], "--dead")){
            strcpy(eyes, "xx");
            strcpy(tongue, "U");
        }
        else if ( !strcmp(mod[n], "-s") ||!strcmp(mod[n], "--stone")){
            strcpy(tongue, "U");
            strcpy(eyes, "**");
        }
        else if ( !strcmp(mod[n], "-b") ||!strcmp(mod[n], "-borg")){
```

```
            strcpy(eyes, "==");
        }
        else if ( !strcmp(mod[n], "-w") ||!strcmp(mod[n], "--what")){
            strcpy(eyes, "OO");
        }
        else if ( !strcmp(mod[n], "-y") ||!strcmp(mod[n], "--young")){
            strcpy(eyes, "..");
        }
        else if ( !strcmp(mod[n], "-z") ||!strcmp(mod[n], "--zebu")){
            strcpy(top," ^   ^");
            strcpy(corn,"|\\_/|");
            strcpy(bos, "/\\");
        }
        else if (!strcmp(mod[n], "-a")||!strcmp(mod[n], "--animate")){
            animate=1;
        }
    }
    // comptage des arguments
    for(int n=1;n<argc;n++){
        int i=0;
        if (!strcmp(mod[n],"-e")||!strcmp(mod[n],"--eyes")){
            i=1;
            n++;
        }
        else if (!strcmp(mod[n],"-t")||!strcmp(mod[n],"--tired")){
            i=1;
        }
        else if (!strcmp(mod[n],"-z")||!strcmp(mod[n],"--zebu")){
            i=1;
        }
        else if (!strcmp(mod[n],"-g")||!strcmp(mod[n],"--greedy")){
            i=1;
        }
        else if (!strcmp(mod[n],"-b")||!strcmp(mod[n],"--borg")){
            i=1;
        }
        else if (!strcmp(mod[n],"-y")||!strcmp(mod[n],"--young")){
            i=1;
        }
        else if (!strcmp(mod[n],"-s")||!strcmp(mod[n],"--stone")){
            i=1;
        }
        else if (!strcmp(mod[n],"-d")||!strcmp(mod[n],"--dead")){
            i=1;
        }
        else if (!strcmp(mod[n],"-p")||!strcmp(mod[n],"--paranoia")){
            i=1;
        }
        else if (!strcmp(mod[n],"-w")||!strcmp(mod[n],"--what")){
```

```
            i=1;
        }
        else if (!strcmp(mod[n], "-a")||!strcmp(mod[n], "--animate")){
            animate=1; //animate permet de mettre une condition pour
        }              //enclencher l'animation
        else if (n<argc & i==0){
            longueur=(longueur+1+strlen(mod[n]));
        }
    }
    if (longueur == 0){
        printf("Erreur:Au moins 2 arguments");
    }
    longueur = longueur+1;
    // affichage de la vache

    printf("\n");

    if (animate==0){
    dire(argc,mod);
    affiche_vache();
    }
    else if (animate==1){ //animation de la vache
        for(int s=0;s<=10;s++){
            strcpy(eyes,"OO");
            strcpy(tongue, "U");
            dire(argc,mod);
            affiche_vache();
            sleep(1);
            update();
            strcpy(eyes,"xx");
            strcpy(tongue, " ");
            gotoxy(20,1);
            dire(argc,mod);
            affiche_vache();
            sleep(1);
            update();
        }
    }
    return 0;
}
```

Reading_cow.c

Quant à reading_cow.c, on a décider de copier la fonction dire() en changeant cependant ses arguments en char *mod, j'ai avons choisi de concaténer le caractère lu dans une chaîne de caractère `char z[MAX]=""`, après avoir affiché la vache et copié la lettre à la place de sa langue, ainsi on obtient la prochaine lettre à la place de sa langue et les lettres lu précédemment dans la bulle.

```
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define MAX 100

char eyes[6]="oo";
char tongue[1]=" ";
char top[50]="                        ";
char corn[50]="^__^";
char bos[10]="__";
int bulle=0;

void update(){ printf("\033[H\033[J");}

void gotoxy(x,y){
    printf("\033[%d;%dH",x,y);
}

void affiche_vache(void){
    printf("                %s\n\
            \\    %s\n\
             \\   (%s)\\ _%s___\n\
                (__)\\        )\\/\\\n\
                 %s  ||----w |\n\
                    ||     ||\n",top,corn,eyes,bos,tongue);
}

void dire(char *mod){ // j'ai modifié l'argument pour qu'il en prenne 1
    int bulle=0; //elle affiche dans une bulle la chaîne de caractère
    int longueur=strlen(mod); //qu'il reçoit
    printf(" ");
    for(int n=1;n<=longueur;n++){
        printf("_");
    }
    if (bulle==0){
        printf("\n< ");
    }
    else if (bulle==1){
        printf("\n( ");
    }
    for(int n=1;n<longueur;n++){
        int i=0;
        if (strcmp(mod,"\0")){
            printf("%c",mod[n]);
        }
    }

    if (bulle==0){
```

```
        printf(">\n ");
    }
    else if ( bulle==1){
        printf(")\n ");
    }
    for(int n=1;n<longueur;n++){
        printf("-");
    }
    printf("\n");
}

void reading_cow(char *argv[]){
    char c[1];
    char z[MAX]="";
    char cop[MAX]="boop";
    int n;
    FILE *f=fopen(*argv, "r");//Hello.txt était un fichier de test qui
    if (f==NULL){f=stdin;} //contenait “Hello”
    else{
        fscanf(f,"%c",c);
        while (!feof(f)){ //on arête à la fin du fichier
            n++;
            strcpy(tongue, c); //copie de la lettre à la place de sa langue
            dire(z); //affichage de z
            affiche_vache();
            strcat(z,c); //on concatene c dans z
            fscanf(f,"%c",c); //on met à jour c
            sleep(1);
        }
    }
    fclose(f); //fermeture du fichier

}
int main(int argc, char *argv[]){
    reading_cow(&argv[1]); //on lit l’argument qui est le nom du fichier
}
```

Sortie:

__

< >

-

\ ^__^

\ (oo)_____

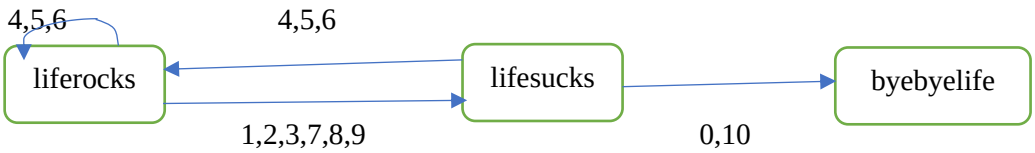
(__)\ )\/\

```
          H  ||----w |
             ||    ||
 ___
< H>
 --

        \    ^__^
         \   (oo)\_______
             (__)\       )\/\
              e  ||----w |
             ||    ||
 ____
< He>
 ---

        \    ^__^
         \   (oo)\_______
             (__)\       )\/\
              l  ||----w |
             ||    ||
 _____
< Hel>
 ----

        \    ^__^
         \   (oo)\_______
             (__)\       )\/\
              l  ||----w |
             ||    ||
 ______
< Hell>
 -----
```

```
\   ^__^
 \  (oo)\_______
    (__)\       )\/\
     o  ||----w |
        ||    ||
```

Partie III: Automates

Dans cette partie j'ai d'abord commencer par copier toutes les fonctions précédentes qui pouvait être utile, puis j'ai fait le diagramme ci-dessous



Quand on passe à byebyelife on ne peut plus revenir aux états précédents, lifesucks est l'état de transition de l'état initial liferocks à l'état final byebyelife.

Ainsi il suffit que le tableau de transition soit remplit correctement en ces points pour que l'automate puisse bien fonctionner.

Voici le tableau de transitions que l'on propose pour cet automate :

0 :byebyelife

1 :lifesucks

2 :byebyelife

	0	1	2	3	4	5	6	7	8	9	10
Byebyelife	0	0	0	0	0	0	0	0	0	0	0
Liferocks	0	1	1	1	2	2	2	1	1	1	0
Lifesucks	0	1	1	1	2	2	2	1	1	1	0

La fonction lire_entree_lunch a été crée pour filtrer la valeur de lunchfood entrée par l'utilisateur, pour ne pas avoir d'erreur dans les calculs du stock et de fitness.

La boucle while a été mise dans la fonction simule automate() pour faciliter des tests qui était nécessaire comme celui de la fonction rand() et pour vérifier le tableau de transitions.

Pour obtenir les nombres aléatoires entre -3 et 3 de crop et digestion j'ai utilisé la fonction (rand() % -3) + 3.

J'ai décider de rajouter une fonctionnalité qui fait que si on a plus de 10 de stocks le surplus est vendu. Cependant on pourra récupérer ce surplus quand on aura un stock nulle, en tapant en entrée « racheter ».

Et quand on prend plus d'une seconde à taper une entrée la vache perd 2 de fitness, cela a été possible en mesurant le temps entre le début du passage dans la boucle et la saisie d'une entrée.

Grâce à la soustraction de time_t begin=time(NULL) et time_t end=time(NULL), de la librairie time.h

```c
#include <stdlib.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>



#define liferocks 0
#define lifesucks 1
#define byebyelife 2
#define CLOCKS_PER_SEG 100
typedef int etat ;

unsigned int m;
unsigned int longueur=0;
char eyes[6]="oo";
char tongue[1]=" ";
char top[50]="                        ";
char corn[50]="^__^";
char bos[10]="__";
int bulle=0;

int stocks = 5;
int lunchfood;
int fitness = 5;
int life = 5;
int crop = 0;
int digestion = 0;

int duree_de_vie = 0;


char sorties[3][10];

void dire(char *mod){ //copie de dire de reading_cow.c
    int bulle=0;
    int longueur=strlen(mod);
    printf(" ");
    for(int n=1;n<=longueur;n++){
        printf("_");
    }
    if (bulle==0){
        printf("\n< ");
    }
```

```
    else if (bulle==1){
        printf("\n( ");
    }
    for(int n=0;n<longueur;n++){
        int i=0;
        if (strcmp(mod,"\0")){
            printf("%c",mod[n]);
        }
    }

    if (bulle==0){
        printf(">\n ");
    }
    else if ( bulle==1){
        printf(")\n ");
    }
    for(int n=1;n<=longueur;n++){
        printf("-");
    }
    printf("\n");
}

int lire_entree_lunch(int stocks){ //lit l'entrée dans lunchfood s'il est bien
    int c; //possible de nourrir la vache avec la quantité entrée
    scanf("%d", &c) ;
    while ((c < 0) | (c>(stocks))) {
        printf("entree invalide !\n") ;
        scanf("%d", &c) ;
    }
    return c; //retourne la valeur entrée
}


int lire_entree() {//fonction qui lit une entrée
    char c;
    scanf(" %s", &c);
    return c;
}

void update(){ printf("\033[H\033[J");}

void affiche_vache(int fitness){
    if (fitness==1|fitness==2|fitness==3|fitness==7|fitness==8|fitness==9){
        strcpy(eyes, "oo");//si l'état de la vache est lifesucks elle
    }                      //est normale
    if (fitness==0|fitness==10){ //si son état est byebyelife
        strcpy(eyes,"xx");//elle prend l'apparence d'une vache morte
        strcpy(tongue,"u");
    }
```

```
    if (fitness==4|fitness==5|fitness==6){
        strcpy(eyes,"##-"); //si son état est liferocks il a des lunettes
    }
    printf("                %s\n\
            \\    %s\n\
             \\   (%s)\\ _%s___\n\
                 (__)\\        )\\/\\\n\
                  %s  ||----w |\n\
                      ||     ||\n",top,corn,eyes,bos,tongue);
}

etat etat_initial(){
    return liferocks; //initialisation de l'état initial
}



int transition(etat life, int fitness){ //établit un tableau de transitions
    char transitions[3][10];
    for (int i=0;i<=2;i++){
        for (int j=0;j<=10;j++){
            transitions[i][j]=0; //toutes les transitions en 0 mène à 0
            if (j==1||j==2||j==3||j==7||j==8||j==9){
                transitions[i][j] = lifesucks;
            }
            if (j==0||j==10){
                transitions[i][j] = byebyelife;
            }
            if (j==4||j==5||j==6){
                transitions[i][j] = liferocks;
            }
        }
    }
    return transitions[fitness][fitness]; //retourne la transition choisit
}




int stock_update(int lunchfood){ //mise à jour du stock
    int a;
    time_t t ;
    srand((unsigned) time(&t)) ; //initialisation
    crop = (rand() % -3) + 3;
    a=(stocks-lunchfood)+crop;
    return a;
}
```

```c
int fitness_update(int lunchfood){
    int a;
    time_t t ;
    srand((unsigned) time(&t)) ;
    digestion = (rand() % -3) + 3;
    a = (fitness+lunchfood)+digestion;
    return a;
}



int simule_automate(){
    etat etat_courant, etat_suivant;
    char entree;
    char smth[18] = "Hello";
    etat_courant = etat_initial();//initialisation de l'etat_courant
    int reste=0;
    while (etat_courant!=byebyelife){
        time_t begin = time(NULL); //on prend le temps avant le début
        dire(smth);
        affiche_vache(fitness);
        time_t t ;
        printf("Stock:%d\n",stocks);
        printf("Tapez \"lunchfood\" pour le nourrir quand il le faut\n");
        scanf("%s",&entree);
        time_t end = time(NULL);//on prend le temps après l'entrée
        if (!strcmp(&entree,"lunchfood")){
            printf("combien ?\n");
            lunchfood = lire_entree_lunch(stocks);//lit la valeur de lunchfood
            printf("\ndigestion\n");
            fitness = fitness_update(lunchfood); //mise à jour de fitness
            stocks = stock_update(lunchfood); //mise à jour de stocks
            if ((stocks==0)&(!strcmp(&entree,"racheter"))){//quand on a 0
            affiche_vache(7); //on peut aller racheter le surplus de stock
            printf("Oh!\n");// qu'on a vendu
            lunchfood=0;
            fitness = fitness_update(lunchfood);
            stocks = stock_update(lunchfood)+reste;
            }
            else if (stocks>10){ //si stocks>10 le surplus est vendu
            printf("\nle reste a été vendu :)"); //est le stock revient à 10
            reste=stocks-10;
            stocks=10;
            }
        }
        else if (!strcmp(&entree,"non")||!strcmp(&entree,"Non")){
            dire("fais gaffe!");
            strcpy(eyes,"--");
            affiche_vache(2);
```

```
            sleep(3);
            lunchfood=0;
            fitness = fitness_update(lunchfood); //mise à jour de fitness
            stocks = stock_update(lunchfood); //mise à jour de stocks
        }
        //Si on prend plus d'une seconde entre le début d'entrée dans la
        // boucle et la commande à faire on perd 2 de fitness
        else if ((now - begin)> 1){
            fitness=fitness-2;
        }
        //etat_suivant est la transition correspondant à fitness
        //etat_courant
        etat_suivant = transition(etat_courant,fitness);
        update();
        etat_courant = etat_suivant; //mise à jour de l'etat_courant
        duree_de_vie=duree_de_vie+1;
        if (fitness==0|fitness==10){
            affiche_vache(fitness);
            etat_courant=byebyelife;
        }
        //duree_de_vie augmente à chaque passage dans la boucle
    }
    return duree_de_vie;
}


int main(){
/* Intialise le generateur de nombres aleatoires */
    time_t t ;
    srand((unsigned) time(&t)) ;
    int nombre ;
    nombre=simule_automate();
    printf("Voici ton score : %d",nombre);
}
```

Sortie :

rakotjon@im2ag-turing:[~/INF203/projet_cowsay]: ./a.out

```
 _____
< Hello>
 -----
        \   ^__^
         \  (##-)\_______
            (__)\       )\/\
```

```
                ||----w |
                ||     ||
```

Stock:5

Tapez "lunchfood" pour le nourrir quand il le faut

0

```
_____
< Hello>
 -----


        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Stock:5

Tapez "lunchfood" pour le nourrir quand il le faut

1

```
_____
< Hello>
 -----


        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Stock:4

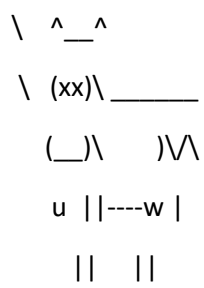
Tapez "lunchfood" pour le nourrir quand il le faut

Lunchfood

0

```
_____
```

< Hello>



Voici ton score : 3

Conclusion :

Ce projet a permit de mieux comprendre les notions que nous avons vu en cours magistrale et en TD, et d'autres que nous n'avons pas vu cette année.

Toutes les parties sont biens faisables, même si l'augmentation de la difficulté se sent beaucoup entre la première partie et la deuxième partie du projet, j'ai fait le choix de ne pas faire de programme trop complexe pour les parties d'imaginations pour ne pas perdre beaucoup de temps, et pouvoir mieux avancés sur le projet.