



Compte rendu du projet Cowsay

Nom: Ourzik

Prénom: Jugurta

Groupe: INF1

Table des matières:

1-Préliminaires

- Le monde Cowsay
- Quelques options de Cowsay
- Sorties d'exécution de quelques commandes

2-Bash

- Les codes sources des exercices demandés.
- Les sorties d'exécution des scripts

3-C

- Les codes sources des exercices demandés.
- Les sorties d'exécution des scripts

4-Automates

- Le code source de l'exercice
- Sortie d'exécution du code.

1- Préliminaires:

➤Le monde Cowsay

Cowsay est une commande UNIX qui génère une image représentant une vache qui prononce une expression saisie par l'utilisateur ou même une vache qui est en train de réfléchir.

Sans arguments, cowsay accepte des entrées standards. La commande possède notamment une dizaine d'options permettant de changer l'apparence physique et émotionnelle de la vache .

➤Les options de Cowsay

- b : initie le mode Borg
- g: appelle mode gourmand
- s: fait paraître la vache complètement défoncée
- y: apparence juvénile de la vache.
- w: contraire de l'option -t
- p: provoque un état de paranoïa sur la vache
- t: fait paraître la vache fatiguée
- d: fait apparaître la vache morte

Il existe aussi d'autres options qui permettent de modifier la forme des yeux et la langue de la vache.

Les yeux sont configurables à travers “-e” et “eye_string” qui utilise que les deux premiers caractères de la chaîne passé en argument. C'est le même cas avec la langue qui est configurable par “-T” et “Tongue_string”.

Il est à noter que toute configuration effectuée par -e et -T sera perdue si l'un des modes fournis est déjà utilisé.

➤ Sorties d'exécution de quelques commandes

```
ourzikj@im2ag-turing:[~]: cowsay -b "Hello jugurta"
-----
< Hello jugurta >
-----
\   ^__^
 \  ==)\_____
  (__)\       )\/\
    ||----w |
     ||     |

ourzikj@im2ag-turing:[~]: cowsay -e "~~" "Hello jugurta"
-----
< Hello jugurta >
-----
\   ^__^
 \  ~~)\_____
  (__)\       )\/\
    ||----w |
     ||     |

ourzikj@im2ag-turing:[~]: █
ourzikj@im2ag-turing:[~]: cowsay -T "%@" "Hello jugurta"
-----
< Hello jugurta >
-----
\   ^__^
 \  (oo)\_____
  (__)\       )\/\
  %@ ||----w |
     ||     |

ourzikj@im2ag-turing:[~]: cowsay -g "Hello jugurta"
-----
< Hello jugurta >
-----
\   ^__^
 \  ($$)\_____
  (__)\       )\/\
    ||----w |
     ||     |

ourzikj@im2ag-turing:[~]: █
```

```
ourzikj@im2ag-turing:[~]: cowsay "Hello jugurta"
```

```
< Hello jugurta >
```



```
ourzikj@im2ag-turing:[~]: cowsay -t "Hello jugurta"
```

```
< Hello jugurta >
```



```
ourzikj@im2ag-turing:[~]: █
```

```
ourzikj@im2ag-turing:[~]: cowsay -y "Hello jugurta"
```

```
< Hello jugurta >
```



```
ourzikj@im2ag-turing:[~]: cowsay -p "Hello jugurta"
```

```
< Hello jugurta >
```



```
ourzikj@im2ag-turing:[~]: █
```

2-Bash :

➤ Les codes sources des exercices demandés.

```
cow_kindergarten.sh
1  #!/bin/sh
2  i=1
3  while [ $i -lt 10 ]          #On va jusqu'à 9 et on affiche la vache tirant sa longue pour 10
4  do
5      cowsay $i
6      sleep 1
7      i=$(( expr $i + 1 ))    #incrémentation de l'indice i
8  done
9  cowsay -s 10

cow_primaryschool.sh
1  #!/bin/sh
2  i=1
3  while [ $i -lt $1 ]          #On va jusqu'à n-1 et on affiche la vache tirant sa longue pour n
4  do
5      cowsay $i
6      sleep 1      #créer une pause de 1 seconde
7      i=$(( expr $i + 1 ))
8  done
9  cowsay -s $1

cow_highschool.sh
1  #!/bin/sh
2  i=1
3  while [ $i -lt $1 ]          #On va jusqu'à n-1 et on affiche la vache tirant sa longue pour n²
4  do
5      cowsay $( expr $i \* $i )
6      sleep 1
7      i=$(( expr $i + 1 ))
8  done
9  cowsay -s $( expr $1 \* $1 )
10

cow_college.sh
1  #!/bin/sh
2  """La suite de fibonacci est définie par récurrence comme suit:
3      U0=1
4      U1=1
5      U(n+2)=U(n+1)+Un """
6
7  p=1      #p comme terme précédent
8  c=1      #c comme terme courant
9  cowsay $p
10 cowsay $c
11
12     #on a déjà affiché deux termes de la suite alors
13     #on affichera que les n-2 autres qui restent"""
14 i=1
15 while [ $i -lt $($expr $1 - 2) ]
16 do
17     s=$(( expr $p + $c ))      #s comme terme suivant
18     cowsay $s
19     p=$c
20     c=$s
21     i=$(( expr $i + 1 ))
22 done
23
24 cowsay -s $($expr $p + $c )   #La vache prononce le n-ième terme de la suite en tirant sa longue
25
```

```
cow_university.sh
1  #!/bin/sh
2
3  #Un nombre est dit premier si et seulement si il admet exactement deux diviseurs
4  #Pour cela on va diviser le nombre sur tous les entiers inférieurs à la
5  #partie entière de sa racine carrée. Si le nombre admet un diviseur alors il n'est
6  #pas premier sinon il est premier.
7
8
9  i=2
10 while [ $i -lt $1 ]
11 do
12     j=1
13     cpt=0      #compteur comme drapeau
14     while [ $j -le $($expr $i / 2 ) ]
15     do
16         if [ $($expr $i % $j ) -eq 0 ]
17         then
18             cpt=$(expr $cpt + 1 )
19         fi
20         j=$(expr $j + 1 )
21     done
22     if [ $cpt -eq 0 ]          #si le drapeau est à zéro alors i n'a pas de diviseurs , il est donc premier
23     then
24         cowsay $i      #la vache prononce le nombre premier i
25     fi
26     i=$(expr $i + 1 )
27 done
```

Une première méthode d'implémentation de smart_cow consiste à récupérer l'opérateur et ses opérandes.

```
smart_cow.sh
1  #!/bin/sh
2  if [ $1==[+]* ]           #On vérifie si l'opérateur + est dans la chaîne de caractères
3  then
4      var1=` echo $1 | cut -d "+" -f 1 `          #On récupère les opérandes avec la commande cut
5      var2=` echo $1 | cut -d "+" -f 2 `          #Avec délimiteur l'opérateur de l'expression (ici c'est +)
6      cowsay -e $( expr $var1 + $var2 ) $1
7
8
9  elif [ $1==[*]* ]          #On vérifie si l'opérateur * est dans la chaîne de caractères
10 then
11     var1=` echo $1 | cut -d "*" -f 1 `
12     var2=` echo $1 | cut -d "*" -f 2 `
13     cowsay -e $( expr $var1 \* $var2 ) $1
14 elif [ $1==[-]* ]          #On vérifie si l'opérateur - est dans la chaîne de caractères
15 then
16     var1=` echo $1 | cut -d "-" -f 1 `
17     var2=` echo $1 | cut -d "-" -f 2 `
18     cowsay -e $( expr $var1 - $var2 ) $1
19 else
20     var1=` echo $1 | cut -d "/" -f 1 `
21     var2=` echo $1 | cut -d "/" -f 2 `
22     cowsay -e $( expr $var1 / $var2 ) $1
23
24
25
26
27 fi
```

Une deuxième méthode de smart_cow consiste à faire directement le calcul de l'expression arithmétique comme suit:

```
#!/bin/sh
#METHODE 2
t=$(( $1 ))
cowsay -e $t $1
```

```
crazy_cow.sh
1  #!/bin/sh
2
3  #####La vache calcule et prononce la somme des n premiers carrés parfaits#####
4  somme=0
5  i=1
6  while [ $i -le $1 ]
7  do
8      j=1
9      cpt=0
10     while [ $j -le $i ]
11     do
12         if [ $( expr $j \* $j ) -eq $i ]      #On vérifie s'il existe un nombre j tel que j²=i
13         then
14             cpt=$(( expr $cpt + 1 ))
15         fi
16         j=$(( expr $j + 1 ))
17     done
18     if [ $cpt -ne 0 ]
19     then
20         somme=$(( expr $somme + $i ))
21     fi
22
23
24     i=$(( expr $i + 1 ))
25 done
26 cowsay -g $somme
```

>Les sorties d'exécution des scripts

cow_kindergarten.sh

La vache prononce les chiffres de 1 à 10 en tirant sa langue au dernier.

```
ourzikj@im2ag-turing:[~/INF203/COWSAY]: ./cow_kindergarten.sh
< 1 >
---
  \ ^__^
  \  (oo)\_____
    (__)\       )\/\
        ||----w |
        ||     |

< 2 >
---
  \ ^__^
  \  (oo)\_____
    (__)\       )\/\
        ||----w |
        ||     |

< 3 >
---
  \ ^__^
  \  (oo)\_____
    (__)\       )\/\
        ||----w |
        ||     |

< 4 >
---
  \ ^__^
  \  (oo)\_____
    (__)\       )\/\
        ||----w |
        ||     |

< 5 >
---
  \ ^__^
  \  (oo)\_____
    (__)\       )\/\
        ||----w |
        ||     |

< 6 >
---
  \ ^__^
  \  (oo)\_____
    (__)\       )\/\
        ||----w |
        ||     |
```

```

*****
< 7 >
-----
\ ^__^
 \  ooo\_____
    ||----w |
    ||     |
< 8 >
-----
\ ^__^
 \  ooo\_____
    ||----w |
    ||     |
< 9 >
-----
\ ^__^
 \  ooo\_____
    ||----w |
    ||     |
< 10 >
-----
\ ^__^
 \  (***)\_____
    ( )\_____)\/\
      U  ||----w |
      ||     |
ourzikj@im2ag-turing:[~/INF203/COWSAY]: 
```

cow_primaryschool.sh

Pour n=3, la vache prononce les entiers inférieurs à 3 en tirant sa langue au dernier .

```

ourzikj@im2ag-turing:[~/INF203/COWSAY]: ./cow_primaryschool.sh 3
< 1 >
-----
\ ^__^
 \  ooo\_____
    ||----w |
    ||     |
< 2 >
-----
\ ^__^
 \  ooo\_____
    ||----w |
    ||     |
< 3 >
-----
\ ^__^
 \  (***)\_____
    ( )\_____)\/\
      U  ||----w |
      ||     |
ourzikj@im2ag-turing:[~/INF203/COWSAY]: 
```

cow_highschool.sh

La vache prononce les carrés des entiers inférieurs ou également n=3.

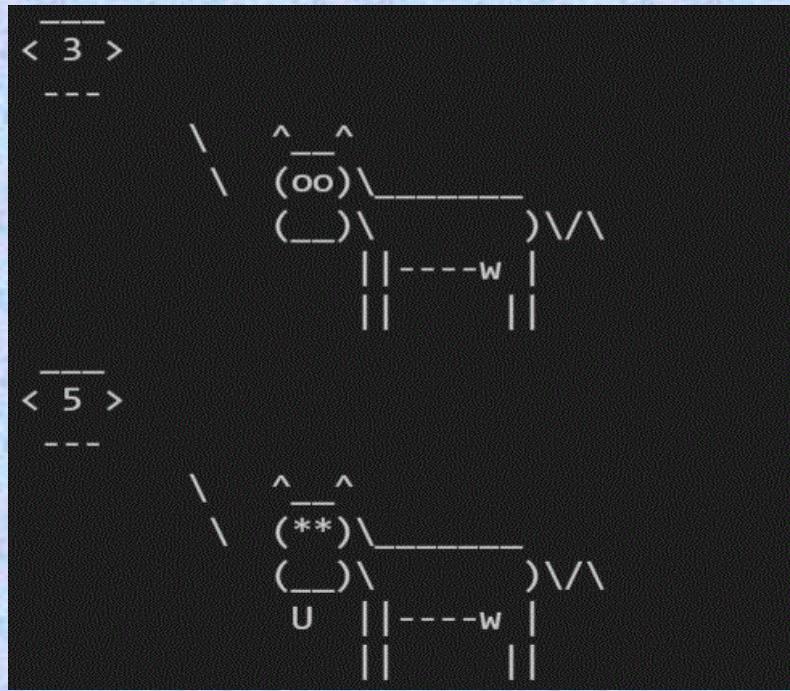
```
ourzikj@im2ag-turing:[~/INF203/COWSAY]: ./cow_highschool.sh 3  
_____  
< 1 >  
---  
 \ ^__^  
  \  (oo)\_____  
   (__)\ )\/\ |  
    ||----w |  
    ||     ||  
  
< 4 >  
---  
 \ ^__^  
  \  (oo)\_____  
   (__)\ )\/\ |  
    ||----w |  
    ||     ||  
  
< 9 >  
---  
 \ ^__^  
  \  (**)\_____  
   (__)\ )\/\ |  
    U  ||----w |  
    ||     ||  
ourzikj@im2ag-turing:[~/INF203/COWSAY]:
```

cow_college.sh

La vache maintenant prononce les n premiers termes de la suite de Fibonacci.

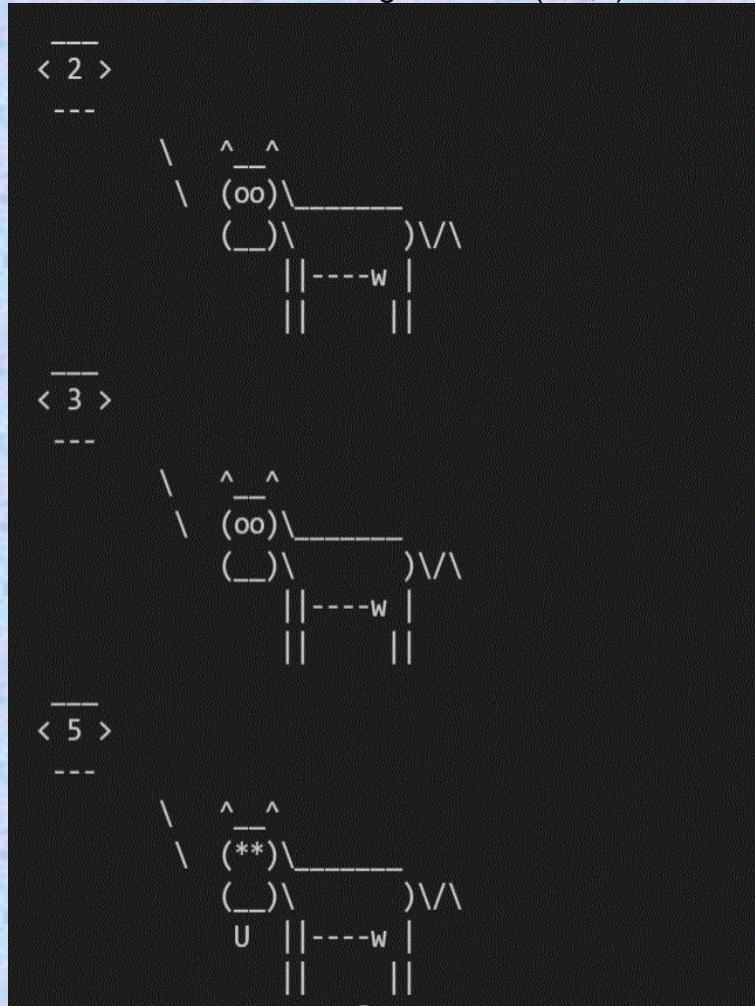
Pour n=5 on le résultat suivant:

```
ourzikj@im2ag-turing:[~/INF203/COWSAY]: ./cow_college.sh 5  
_____  
< 1 >  
---  
 \ ^__^  
  \  (oo)\_____  
   (__)\ )\/\ |  
    ||----w |  
    ||     ||  
  
< 1 >  
---  
 \ ^__^  
  \  (oo)\_____  
   (__)\ )\/\ |  
    ||----w |  
    ||     ||  
  
< 2 >  
---  
 \ ^__^  
  \  (oo)\_____  
   (__)\ )\/\ |  
    ||----w |  
    ||     ||
```



cow_university.sh

La vache prononce les nombres premiers inférieurs à la borne passée en paramètre.
Pour n=6, on aura l'affichage suivant (2,3,5)



smart_cow.sh

La vache maintenant prononce le résultat d'un calcul numérique.

```
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieBASH]: ./smart_cow.sh 72-43
```

```
< 72-43 >  
-----  
 \ ^__^  
  \ (29)\_____  
    (__)\       )\/\  
     ||----w |  
     ||     ||
```

```
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieBASH]: ./smart_cow.sh 72/2
```

```
< 72/2 >  
-----  
 \ ^__^  
  \ (36)\_____  
    (__)\       )\/\  
     ||----w |  
     ||     ||
```

```
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieBASH]:
```

```
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieBASH]: ./smart_cow.sh 3*23
```

```
< 3*23 >  
-----  
 \ ^__^  
  \ (69)\_____  
    (__)\       )\/\  
     ||----w |  
     ||     ||
```

```
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieBASH]: ./smart_cow.sh 3+76
```

```
< 3+76 >  
-----  
 \ ^__^  
  \ (79)\_____  
    (__)\       )\/\  
     ||----w |  
     ||     ||
```

```
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieBASH]:
```



crazy_cow.sh

La vache calcule et prononce la somme des carrés parfaits inférieurs à la borne passée en argument.

```
ourzikj@im2ag-turing:[~/INF203/COWSAY]: ./crazy_cow.sh 20
```

< 30 >

```
\ ^__^
 \ __$)\_\_
 (____)\ )\/\
    ||----w |
    ||       |
```

```
ourzikj@im2ag-turing:[~/INF203/COWSAY]: ./crazy_cow.sh 100
```

< 385 >

```
\ ^__^
 \ __$$_\_____
  (_)\ )\\---W |
    ||----w |
    ||     |
```

```
ourzikj@im2ag-turing:[~/INF203/COWSAY]: ./crazy_cow.sh 300
```

< 1785 >

```
\^_($$)\_\_((_)\\-----w)\\/\|
```

ourzikj@im2ag-turing:[~/INF203/COWSAY]:

3-C :

1. 2)- Pour les deux premières questions , je les ai faites ensemble du moment qu'il s'agit juste d'une amélioration du même programme .

script qui modifie l'état d'affichage de la bouche ou de la longue selon l'option choisie (-e pour les yeux et t pour la longue)

```
#include <stdio.h>
#include <string.h>
//pour l'exécution on passe d'abord l'option comme premier argument puis la chaîne de caractères
void affiche_vache(char chaîne[2],char option[2]) {
    if ((strcmp(option,"-e")== 0)) //on modifie les yeux
        printf(" %s %s %s %s %s %s", " \'^__ ^\n", "\\\"(,chaîne,)\\\" _____ \\n", " \\'(____)\\\" ) \\\" \n"
    else //Sinon on modifie toujours la longue
        printf(" %s %s %s %s %s %s", " \'^__ ^\n", "\\\"(00)\\\" _____ \\n", " \\'(____)\\\" ) \\\"/\\\" \\n", " \\\" "
}

int main (int argc, char *argv[]) {
    //lors du passage des paramètres pour le fichier new_cow ,on considère que le premier
    //argument est l'option , le deuxième c'est la chaîne de caractères.
    affiche_vache(argv[2],argv[1]);
```

ci dessous, quelques sorties d'exécutions du script:

```
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieC]: ./a.out -e uu
  ^ __ ^
  \(( uu )\)\ _____
    '(____)\      ) \\\
      '||| - - - -w |
        '||           ||
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieC]: ./a.out -t uu
  ^ __ ^
  \((00)\)\ _____
    '(____)\      ) \\\
      ' uu || - - - -w |
        '||           ||
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieC]: []
```

3)-

La fonctionnalité apportée à mon script est l'allongement de la queue de la vache dans le cas où un argument a été passé en paramètre. Dans l'autre cas la vache sera triste de ne pas avoir prolongé sa queue.

```
#include <stdio.h>
void affiche_vache(char chaine[]) {
    printf(" %s %s %s %s %s %s", " \\'^__^\\n", " \\\"(oo)\\\"_____\n", " \\'(_____)\\\""
) ,chaine, "\n", " \\'|| - - -w |\n" , " \\'|| | |\\n "
);

//programme qui modifie la taille de la queue de la vache selon l'entier saisi au
//clavier
//Si on ne passe pas d'arguments la vache ne sera pas contente et on le verra dans
//ses yeux
int main(int argc, char *argv[]){
    int L;
    char Q[100];
    Q[0] = '\\';
    Q[1]= '\\';
    Q[2] = '/';
    Q[3]= '\\';
    Q[4]= '\\';

    if(argc==2) {
        sscanf(argv[1], "%d", &L);
        for (int i = 5; i <= L+5;i++){
            if (i % 2 !=0)
                Q[i] = '/';
            else
                Q[i] = '\\';
        }
        affiche_vache(Q);
    }
    else
        printf(" %s %s %s %s %s", " \\'^__^\\n", " \\\"(--)\\\"_____\n", " \\'(_____)\\\""
) ;
    return 0;
}
```

Voici quelques sorties d'exécutions de ce script:

4)-

Mon animation à ce niveau consiste à faire changer l'état des yeux et l'état de la gueule de la vache en même temps à travers les arguments qu'on devrait passer en paramètres lors de l'exécution.

Vous trouverez ci-dessous l'ensemble du code et quelques sorties

d'exécution de celui-ci:

```
#include <stdio.h>
#include <unistd.h>
void affiche_vache(char chaine[]){
    printf(" %s %s %s %s %s %s", " \'^__ ^\n","\\\"(,\"chaine,\"\")\\ \\ _____ \n", " \"'(_)\\" ) \\ \\ \\ \\ \n", " \
}
void update (){
    printf ("\033[H\033[J");
}
void gotoxy ( x , y ) {
    printf ("\033[%d;%dH",x,y);
}
//Animation
//Programme qui change l'état des yeux et l'état de la longue de la vache selon la série des chaines
//de caractères passées en arguments
//PS: On compile d'abord le script puis on fait clear et ensuite on exécute le fichier a.out

int main(int argc, char* argv[]){
    affiche_vache("uu");
    for (int i = 2; i <= argc-1; i++){
        gotoxy(4,5);
        printf("%s", argv[i]);
        gotoxy(3,5);
        printf("%s", argv[i-1]);
        sleep(1);
        printf("\n");
    }
    printf("\n");
    printf("\n");
    printf("\n");
    return 0;
}
```

Sorties d'exécution:

```
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieC]: ./a.out -- uu tt zz
      ^__^
     \( tt )\  -----
       '(zz)\      )  \\/\
         '||| - - - -w |
           '|||      ||
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieC]: 

ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieC]: ./a.out -- uu tt zz @o
      ^__^
     \( zz )\  -----
       '(@o)\      )  \\/\
         '||| - - - -w |
           '|||      ||
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieC]: 
```

5)-

Le code :

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

char lire_caractere (FILE *f) {
    char c;
    fscanf(f, "%c", &c);
    return c;
}

//Une fonction qui affiche la vache prononçant le caractère lu
void affiche_vache (char car) {
    printf(" %s %s %s %s %c %s %s\n", "      \\ '^__^\\n", "\\(00)\\ \\ _____ \\n", "\\(')\\ ")
}

//Une fonction qui affiche le mot complet
void affiche_mot (char *chaine){
    printf("-----\\n");
    printf("< %s >\\n", chaine);
    printf("-----\\n");
}
```

```
int main (int argc, char* argv[]){
    FILE *f;
    char c;
    char mot[50];
    //tests sur le nombre d'arguments passés en paramètre
    //Possibilité de lecture depuis l'entrée standard
    if (argc==1)
        f = stdin;
    else
        f = fopen(argv[1], "r");

    if (f==NULL){    // test sur la possibilité d'ouverture du fichier
        perror(argv[1]);
        return 1;
    }
    int i = 0;
    c = lire_caractere(f);
    while(!feof(f)){
        if(c==' '){
            mot[i] = c;
            affiche_mot(mot);
            affiche_vache(c);
        }
        else{
            /*quand il s'agit d'un espace j'écrase le contenu de ma variable auxiliaire mot avec des espaces,
            la chaîne d'espaces est supposée plus grande pour éviter de remplacer seulement quelques lettres
            du mot si le mot est de taille inférieure à celle du précédent*/

            strcpy(mot, " ");
            i = 0; //je remets mon indexeur à 0 pour pouvoir lire un nouveau mot |
        }
        i++;
        c = lire_caractere(f);
    }

    return 0;
}
```

Sorties d'exécutions:

Pour un fichier « file_readtest.txt » contenant le texte suivant:

**“Bonjour Jugurta
J’ai 20 ans.”**

On aura la sortie suivante :

```
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieC]: clang reading_cow.c
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieC]: ./a.out file_readtest.txt
```

```
- - - - - < B >
```

```
\ ' ^ _ ^  
\(00)\ _____  
'(_)\ ) \\\\  
' B || - - - -w | ||
```

```
- - - - - < Bo >
```

```
\ ' ^ _ ^  
\(00)\ _____  
'(_)\ ) \\\\  
' o || - - - -w | ||
```

```
- - - - - < Bon >
```

```
\ ' ^ _ ^  
\(00)\ _____  
'(_)\ ) \\\\  
' n || - - - -w | ||
```

```
- - - - - < Bonj >
```

```
\ ' ^ _ ^  
\(00)\ _____  
'(_)\ ) \\\\  
' j || - - - -w | ||
```

```
- - - - - < Bonjo >
```

```
\ ' ^ _ ^  
\(00)\ _____  
'(_)\ ) \\\\  
' o || - - - -w | ||
```

```
- - - - - < Bonjou >
```

```
\ ' ^ _ ^  
\(00)\ _____  
'(_)\ ) \\\\  
' u || - - - -w | ||
```

< Bonjour >

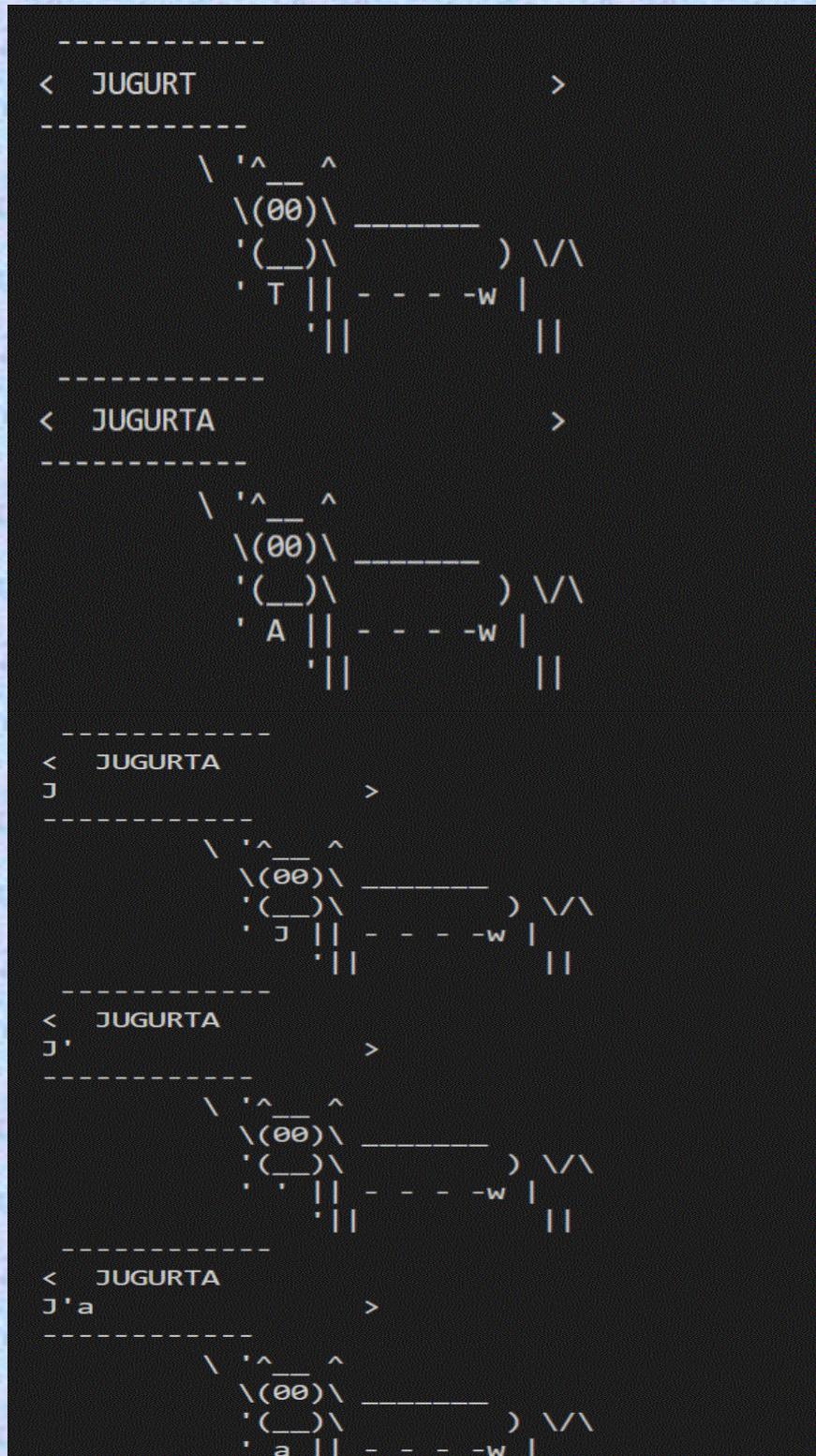
< J >

< JU >

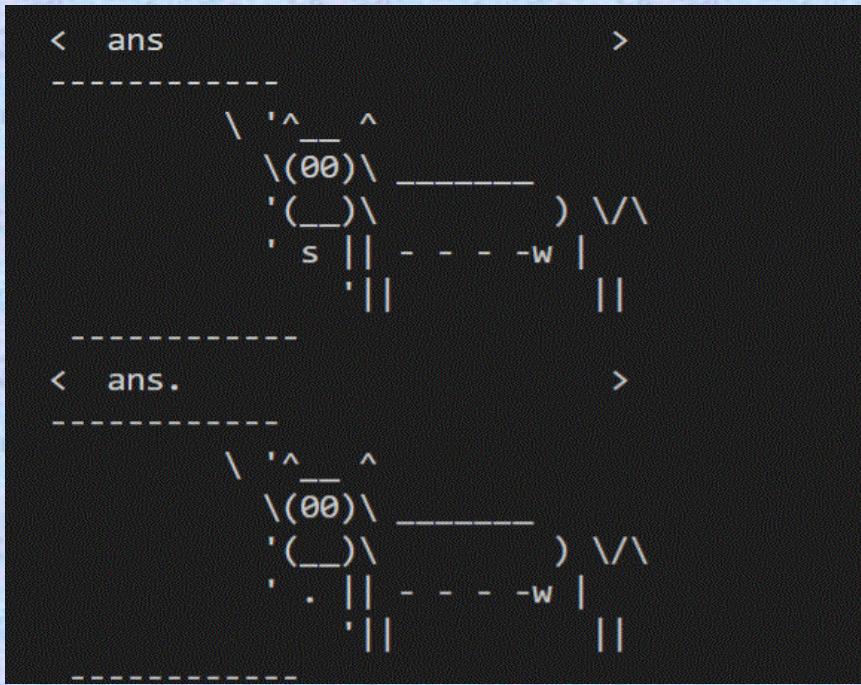
< JUG >

< JUGU >

< JUGUR >



```
*****  
★  
★  
★  
★  
★  
★-----  
< JUGURTA  
J'ai >  
-----  
 \ ' ^ _ ^  
 \(00)\ _____ ) \|\\  
 '(__)\ ) | - - - -w | ||  
 i || | |  
 -----  
< 2 >  
-----  
 \ ' ^ _ ^  
 \(00)\ _____ ) \|\\  
 '(__)\ ) | - - - -w | ||  
 2 || | |  
 -----  
< 2θ >  
-----  
 \ ' ^ _ ^  
 \(00)\ _____ ) \|\\  
 '(__)\ ) | - - - -w | ||  
 θ || | |  
 -----  
-----  
< a >  
-----  
 \ ' ^ _ ^  
 \(00)\ _____ ) \|\\  
 '(__)\ ) | - - - -w | ||  
 a || | |  
 -----  
< an >  
-----  
 \ ' ^ _ ^  
 \(00)\ _____ ) \|\\  
 '(__)\ ) | - - - -w | ||  
 n || | |  
 -----  
< ans >  
-----  
 \ ' ^ _ ^  
 \(00)\ _____ ) \|\\  
 '(__)\ ) | - - - -w | ||
```



4- Automates :

Le code source :

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4
5
6  #define byebyelife 1
7  #define liferocks  4
8  #define lifesucks 7
9  // Une fonction qui affiche la vache en modifiant l'état de ses yeux pour exprimer les trois états
10 // '--> vache morte
11 // 'oo' -> vache en pleine forme
12 // '@' -> vache avec excédent de nourriture
13 void affiche_vache(char chaîne[2]) {
14     printf(" %s %s %s %s %s %s %s", " \'^__ ^\n", "\\\\"(chaîne,")\\_ _____ \n", " \\'(_)_\\ ) \\\\ /\\ \\\n", " \
15 }                                |
16
17 // Génère un entier entre 0 et borne-1
18 int generer_entier(int borne) {
19     static int seme = 0;
20     if (!seme) {
21         srand(getpid());
22         seme = 1;
23     }
24     return random() % borne;
25 }
```

```

28     int fitness_update(int lunchfood,int fitness){
29
30         int digestion = generer_entier(4) - 3;
31
32         return (fitness + lunchfood )+ digestion;
33     }
34
35     int stock_update(int lunchfood,int stock){
36         int crop, n;
37         n = generer_entier(2);
38         if (n==0)
39             crop = generer_entier(4) - 3;
40         else
41             crop = generer_entier(4);
42
43
44
45         return (stock - lunchfood) + crop;;
46     }
47 ///////////////////////////////////////////////////////////////////
48
49     int main(){
50         int fitness = 5;
51         int stock = 5;
52         int etat_courant = liferocks;
53         int etat_suivant;
54         int duree_vie = 0;
55         int lunchfood;
56
57
58
59         printf("Stock INITIAL est:%d\n", stock); //stock initial à afficher dès le début de la partie
60         while (fitness!=0 && stock>=0 && stock<=10){
61
62
63             //je calcule les transitions possibles en affichant l'état de la vache et le stock courant
64
65             if (etat_courant==byebyelife){
66                 if (fitness>=4 && fitness<=6){
67                     etat_suivant = liferocks;
68                     affiche_vache("oo");
69                     printf("Stock courant est:%d\n", stock);
70                 }
71                 if (fitness>=7 && fitness<=9){
72                     etat_suivant = lifesucks;
73                     affiche_vache( "@@");
74                     printf("Stock courant est:%d\n", stock);
75
76
77
78             }
79         }
80         else if (etat_courant==liferocks){
81             if (fitness>=1 && fitness<=3){
82                 etat_suivant = byebyelife;
83                 printf("Stock courant est:%d\n", stock);
84                 affiche_vache( "xx");
85

```

```
86
87     }
88     if (fitness>=7 && fitness<=9){
89         etat_suivant = lifesucks;
90         affiche_vache( "@@");
91
92         printf("Stock courant est:%d\n", stock);
93
94     }
95
96 }
97 else {
98     if (fitness>=1 && fitness<=3){
99         etat_suivant = byebyleife;
100        affiche_vache( "xx");
101        printf("Stock courant est:%d\n", stock);
102
103    }
104    if (fitness>=4 && fitness<=6){
105        etat_suivant = liferocks;
106        affiche_vache( "oo");
107        printf("Stock courant est:%d\n", stock);
108
109    }
110
111 }
112
113
114 }
115 etat_courant = etat_suivant;

116
117     printf("Veuillez saisir une quantité de nourriture inférieure au stock courant:\n");
118     scanf("%d", &lunchfood);
119     stock = stock_update(lunchfood, stock); //mise à jour du stock
120     fitness = fitness_update(lunchfood, fitness); // mise à jour du niveau de santé
121
122     duree_vie++;
123 }
124     affiche_vache("--");
125     printf("C'est mooooort\n");
126
127     printf("Votre score est:%d\n", duree_vie);
128
129 return 0;
130 }
```

Sortie d'exécution :

Pour un stock et un niveau de santé initiaux qui valent 5 on a la sortie suivante :

```
ourzikj@im2ag-turing:[~/INF203/COWSAY/PartieAutomates]: ./a.out
Stock INITIAL est:5
Veuillez saisir une quantité de nourriture inférieure au stock courant:
2
      ^__^
     \(oo )\ -----
       '(--)\
         || - - - -w |
           ||       ||
Stock courant est:3
Veuillez saisir une quantité de nourriture inférieure au stock courant:
3
      ^__^
     \(@@ )\ -----
       '(--)\
         || - - - -w |
           ||       ||
Stock courant est:2
```

```
Veuillez saisir une quantité de nourriture inférieure au stock courant:
1
      ^__^
     \(oo )\ -----
       '(--)\
         || - - - -w |
           ||       ||
Stock courant est:0
Veuillez saisir une quantité de nourriture inférieure au stock courant:
0
      ^__^
     \(-- )\ -----
       '(--)\
         || - - - -w |
           ||       ||
C'est mooooort
Votre score est:4
```