

Projet Cowsay

Partie Préliminaire

Objectif

Explorer la commande `cowsay`, ses options et son fonctionnement via le manuel utilisateur (`man cowsay`), et en produire un résumé commenté avec exemples.

Présentation de `cowsay`

`cowsay` est un programme en ligne de commande amusant qui affiche un message à l'intérieur d'une bulle, avec une vache ASCII en dessous.

```
$ cowsay "Bonjour"
```



Accès au manuel

Commande pour consulter la documentation :

```
man cowsay
```

Liste des options importantes de `cowsay`

Option	Description	Exemple
-e	Change les yeux de la vache	<code>cowsay -e "oO"</code> "Salut"
-T	Change la langue de la vache	<code>cowsay -T "U "</code> "Salut"
-b	Mode “borg”, style robotisé	<code>cowsay -b "Je suis un bot"</code>
-d	Mode “mort”	<code>cowsay -d "Je suis mort"</code>
-g	Mode “greedy” (avide)	<code>cowsay -g</code> "Donne-moi tout"
-p	Mode “paranoïaque”	<code>cowsay -p "Ils m'observent"</code>
-s	Mode “stoned” (yeux dilatés)	<code>cowsay -s "Wow..."</code>
-t	Mode “tired” (fatigué)	<code>cowsay -t "Je suis épuisé"</code>
-w	Mode “wired” (surexcité)	<code>cowsay -w "Trop d'énergie"</code>
-y	Mode “youthful” (yeux naïfs)	<code>cowsay -y "Coucou"</code>
-f	Change le type de vache (utilise un autre fichier de figure)	<code>cowsay -f dragon</code> "Bonjour"

Exemples d'utilisation

Changer les yeux :

```
$ cowsay -e "XX" "Je vois tout"
```



Ajouter une

langue :

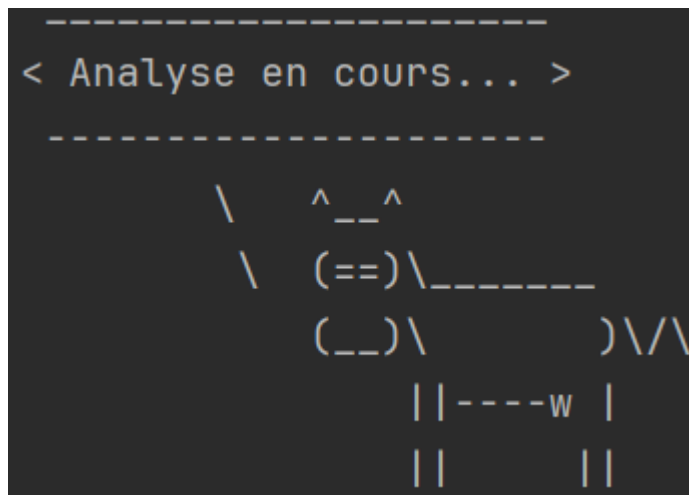
```
$ cowsay -T "~~" "Miam"
```



Mode

robot :

```
$ cowsay -b "Analyse en cours..."
```



Changer de

figure :

```
$ cowsay -f dragon "Je suis un dragon !"
```



> Pour lister toutes les figures disponibles :

`cowsay -l`

Résumé

Catégorie	Ce que j'ai découvert
Fonction de base	Affiche un message dans une bulle ASCII
Options stylées	Plusieurs modes (<code>-b</code> , <code>-d</code> , <code>-s</code> , etc.)
Personnalisation	Yeux <code>-e</code> , langue <code>-T</code> , type de vache <code>-f</code>
Documentation	Accès facile via <code>man cowsay</code>

Partie Bash

Cette section présente plusieurs scripts Bash permettant à **cowsay** de réciter diverses suites de nombres et de réaliser des calculs.

1. Script `cow_kindergarten.sh`

Objectif : la vache prononce les chiffres de 1 à 10, avec une pause d'une seconde, puis tire la langue à la fin.

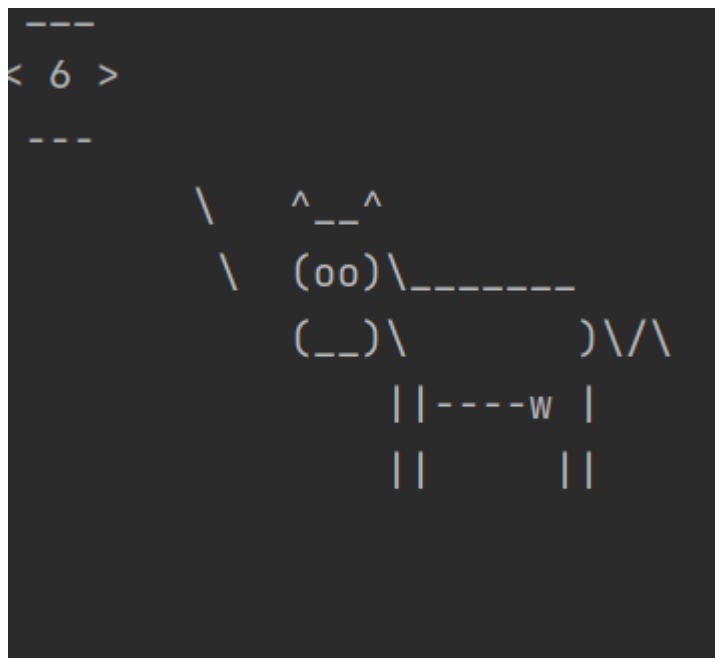
```
#!/usr/bin/env bash
# cow_kindergarten.sh
# Prononce les chiffres de 1 à 10, pause 1s, vache tire la langue.

for i in {1..10}; do
    clear
    cowsay "$i"
    sleep 1
done

# À la fin, la vache tire la langue (-T "U ")
cowsay -T "U " "Fin de l'exercice"
```

Exécution & Sortie Exemple :

```
$ chmod +x cow_kindergarten.sh
$ ./cow_kindergarten.sh
```



2. Script cow_primaryschool.sh

Objectif : la vache prononce les chiffres de 1 à n , avec n passé en argument.

```
#!/usr/bin/env bash
# cow_primaryschool.sh
# Usage: ./cow_primaryschool.sh <n>
# Prononce les chiffres de 1 à n.
```

```

if [ -z "$1" ] || ! [[ "$1" =~ ^[0-9]+$ ]]; then
    echo "Usage: $0 <nombre_entier>"
    exit 1
fi

n=$1
for ((i=1; i<=n; i++)); do
    clear
    cowsay "$i"
    sleep 1
done

```

Exécution Exemple :

```

$ chmod +x cow_primaryschool.sh
$ ./cow_primaryschool.sh 15

```



3. Script cow_highschool.sh

Objectif : la vache prononce la suite des carrés 1^2 , 2^2 , ..., n^2 .

```

#!/usr/bin/env bash
# cow_highschool.sh
# Usage: ./cow_highschool.sh <n>
# Prononce les carrés des entiers jusqu'à n.

if [ -z "$1" ] || ! [[ "$1" =~ ^[0-9]+$ ]]; then
    echo "Usage: $0 <nombre_entier>"
    exit 1

```

```

fi

n=$1
for ((i=1; i<=n; i++)); do
    square=$((i * i))
    clear
    cowsay "$square"
    sleep 1
done

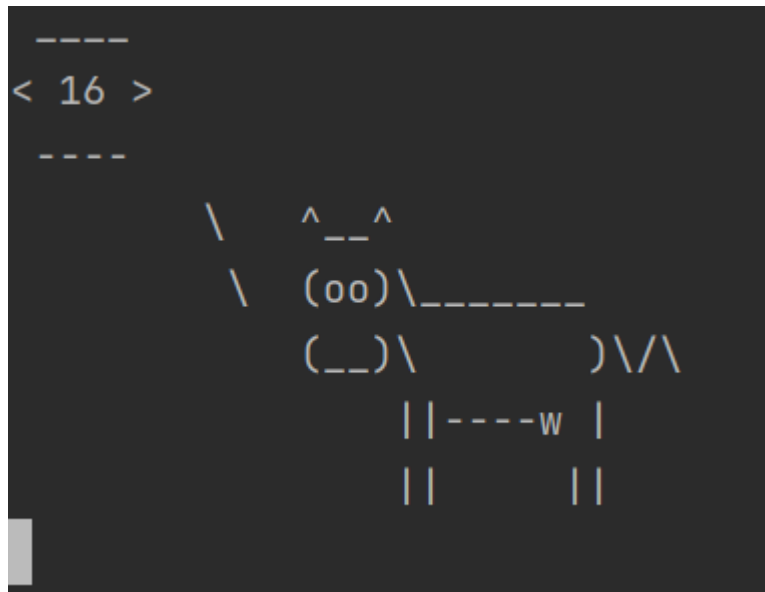
```

Exécution Exemple :

```

$ chmod +x cow_highschool.sh
$ ./cow_highschool.sh 10

```



4. Script cow_college.sh

Objectif : la vache prononce la suite de Fibonacci inférieure à n.

```

#!/usr/bin/env bash
# cow_college.sh
# Usage: ./cow_college.sh <n>
# Prononce les termes de Fibonacci < n.

if [ -z "$1" ] || ! [[ "$1" =~ ^[0-9]+$ ]]; then
    echo "Usage: $0 <nombre_entier>"

```

```

        exit 1
    fi

    n=$1
    a=1
    b=2

    while [ "$a" -lt "$n" ]; do
        clear
        cowsay "$a"
        sleep 1
        fn=$((a + b))
        a=$b      # <-- ici sans ( )
        b=$fn     # <-- ici sans ( )
    done

```

Exécution Exemple :

```

$ chmod +x cow_college.sh
$ ./cow_college.sh 10

```

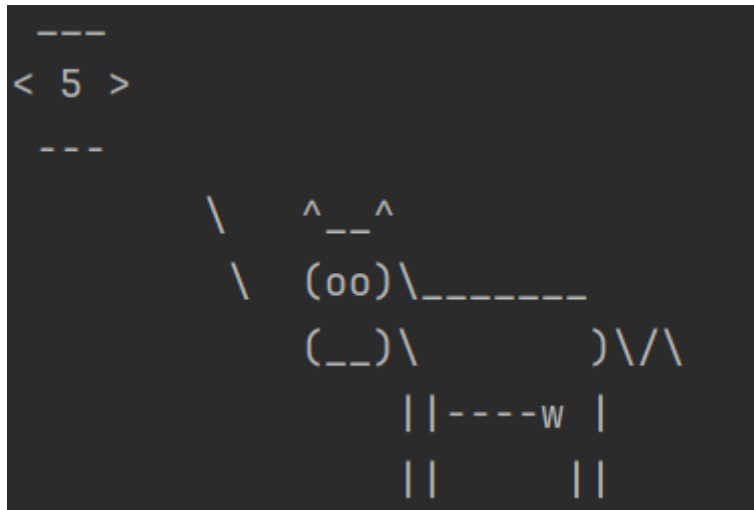


Figure 1: resultat

5. Script cow_university.sh

Objectif : la vache prononce les nombres premiers inférieurs à n.

```

#!/usr/bin/env bash
# cow_university.sh

```



```

# Usage: ./cow_university.sh <n>
# Affiche les nombres premiers < n avec cowsay

# Vérification que l'argument est bien fourni et est un entier
if [ -z "$1" ] || ! [[ "$1" =~ ^[0-9]+$ ]]; then
    echo "Usage: $0 <nombre_entier>"
    exit 1
fi

n=$1

# Fonction qui teste si un nombre donné est premier
is_prime() {
    local i=$1 # Prend directement i en paramètre
    if (( i <= 1 )); then
        return 1
    fi
    for ((j=2; j*j<=i; j++)); do
        if (( i % j == 0 )); then
            return 1
        fi
    done
    return 0
}

# Boucle sur les entiers de 2 à n-1
for ((i=2; i<n; i++)); do
    if is_prime "$i"; then
        clear
        cowsay "$i"
        sleep 1
    fi
done

```

Exécution Exemple :

```

$ chmod +x cow_university.sh
$ ./cow_university.sh 10

```



6. Script smart_cow.sh

Objectif : la vache résout un calcul simple (deux opérandes), et ses yeux affichent le résultat.

```
#!/usr/bin/env bash
# smart_cow.sh
# Usage: ./smart_cow.sh "<op1><op><op2>"
# Exemple: ./smart_cow.sh "3+11"

expr="$1"

# Si l'expression est vide, afficher l'usage
if [ -z "$expr" ]; then
    echo "Usage: $0 \"<nombre1><+|-|*|/><nombre2>\""
    exit 1
fi

# Calculer le résultat avec bc
result=$(echo "scale=0; $expr" | bc 2>/dev/null)

# Vérifier si bc a échoué (par exemple une expression invalide)
if [ $? -ne 0 ] || [ -z "$result" ]; then
    echo "Erreur : expression invalide."
    echo "Usage: $0 \"<nombre1><+|-|*|/><nombre2>\""
    exit 1
fi

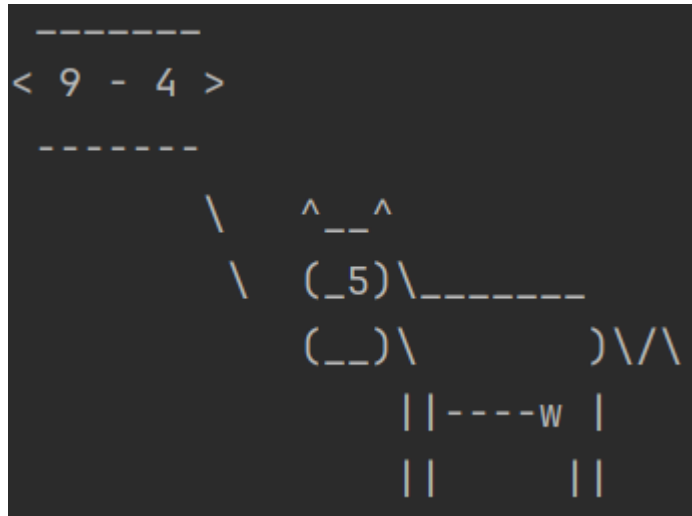
# Limiter les yeux à deux caractères (padding/troncation)
```

```
eyes=$(printf "%2s" "$result" | sed 's/ /_/g' | cut -c1-2)
```

```
clear
cowsay -e "$eyes" "$expr"
```

Exécution Exemple :

```
$ chmod +x smart_cow.sh
$ ./smart_cow.sh "9-4"
```



7. Script crazy_cow.sh

Objectif libre : ici, la vache récite une séquence aléatoire de chiffres hexadécimaux.

```
#!/usr/bin/env bash
# crazy_cow.sh
# La vache prononce 10 valeurs hexadécimales aléatoires.

for i in {1..10}; do
    hex=$(printf "%x" $((RANDOM % 256)))
    clear
    cowsay "$hex"
    sleep 1
done
```

Exécution Exemple :

```
$ chmod +x crazy_cow.sh
$ ./crazy_cow.sh
```



Commentaires généraux :

- On utilise systématiquement `clear` pour rafraîchir l’affichage.
- `sleep 1` crée une pause d’une seconde entre chaque itération.
- Chaque script vérifie la validité de l’argument.
- `smart_cow.sh` montre comment **parser** et **évaluer** une expression en Bash.
- `crazy_cow.sh` est un exemple d’utilisation de `$RANDOM` pour un comportement imprévisible.

Partie C

Dans cette section, nous recodons `cowsay` en C, étape par étape, et ajoutons des fonctionnalités.

1. `newcow.c` : Affichage de base de la vache

Code source

```
// newcow.c
#include <stdio.h>

// Affiche une vache ASCII sans bulle de texte
void affiche_vache() {
    printf(" \\  ^__^\\n");
    printf(" \\  (oo)\\_______\\n");
    printf("      (__)\\        )\\/\\n");
}
```

```

        printf("        ||----w |\n");
        printf("        ||      ||\n");
    }

    int main() {
        affiche_vache();
        return 0;
    }

```

Compilation & Exécution

```

gcc -o newcow newcow.c
./newcow

```

Sortie Exemple :

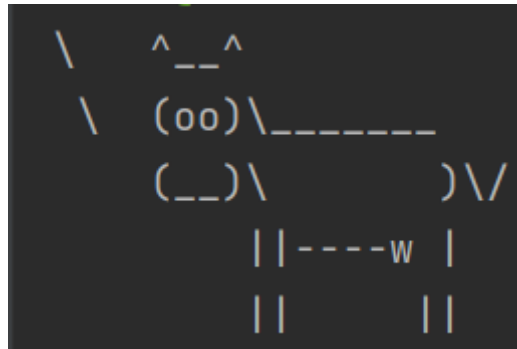


Figure 2: resultat

2. Gestion des options -e/--eyes

Objectif

- Permettre de changer les **yeux** de la vache via une option -e (ou --eyes).

Code source

```

// newcowe.c (version avec -e/--eyes)
#include <stdio.h>
#include <string.h>

void affiche_vache(const char *eyes) {
    // Eyes doit être exactement 2 caractères
    char e1 = eyes[0];
    char e2 = eyes[1];
}

```

```

printf(" \\    ^__^\\n");
printf(" \\    (%c%c)\\_____\\n", e1, e2);
printf("      (__)\\                )\\/\\n");
printf("          ||----w |\\n");
printf("          ||        ||\\n");
}

int main(int argc, char *argv[]) {
    const char *eyes = "oo"; // Valeur par défaut
    // Parcours des arguments
    for (int i = 1; i < argc; i++) {
        if ((strcmp(argv[i], "-e") == 0 || strcmp(argv[i], "--eyes") == 0) && i + 1 < argc)
            if (strlen(argv[i+1]) == 2) {
                eyes = argv[i+1];
            } else {
                fprintf(stderr, "Erreur: les yeux doivent être 2 caractères.\\n");
                return 1;
            }
        i++;
    }
    affiche_vache(eyes);
    return 0;
}

```

Compilation & Exécution

```

gcc -o newcowE newcowE.c
./newcowE -e "AX"

```

Sortie Exemple :

```

\    ^__^
\    (AX)\_____
    (__) \                )\/
        ||----w |
        ||        ||

```

Figure 3: resultat

3. Option --tail : Longueur de la queue

Objectif

- Ajouter l'option --tail <L> pour allonger la queue de la vache de L traits supplémentaires.

Code source

```
// newcowT.c (version avec --tail)
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void affiche_vache(const char *eyes, int tail) {
    char e1 = eyes[0], e2 = eyes[1];

    printf(" \\   ^__^\\n");
    printf(" \\   (%c%c)\\_______\\n", e1, e2);
    printf("      (__)\\        )\\/\\n");
    printf("         ||----w |");
    // Allonger la queue
    for (int i = 0; i < tail; i++) {
        printf("-");
    }
    printf("\\n");
    printf("         ||        ||\\n");
}

int main(int argc, char *argv[]) {
    const char *eyes = "oo";
    int tail = 0;

    for (int i = 1; i < argc; i++) {
        if ((strcmp(argv[i], "-e") == 0 || strcmp(argv[i], "--eyes") == 0) && i + 1 < argc)
            if (strlen(argv[i+1]) == 2) eyes = argv[i+1];
            else { fprintf(stderr, "Yeux : 2 caractères\\n"); return 1; }
            i++;
        } else if ((strcmp(argv[i], "--tail") == 0) && i + 1 < argc) {
            tail = atoi(argv[i+1]);
            if (tail < 0) tail = 0;
            i++;
        }
    }
    affiche_vache(eyes, tail);
}
```

```

    return 0;
}

```

Compilation & Exécution

```

gcc -o newcowT newcowT.c
./newcowT -e AZ --tail 4

```

Sortie Exemple :

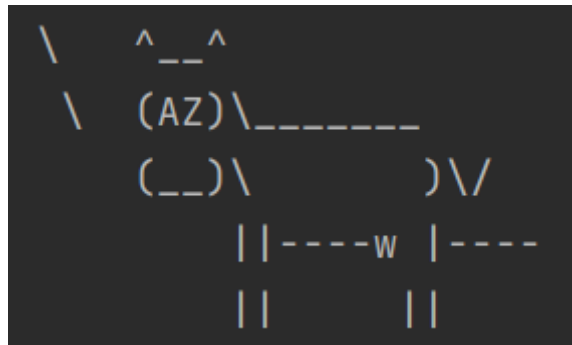


Figure 4: resultat

4. wildcow.c – Animation de la vache

Objectif

Créer une “vache animée” qui se déplace à l’écran et dont la langue ou les yeux bougent, en utilisant : - `update()` : efface l’écran (`printf("\033[H\033[J")`) - `gotoxy(x,y)` : positionne le curseur (`printf("\033[%d;%dH", y, x)`) - `usleep()` pour la pause entre les frames

Code source

```

// wildcow.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

// Efface l'écran
void update() {
    printf("\033[H\033[J");
}

// Positionne le curseur en (x,y)

```



```

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

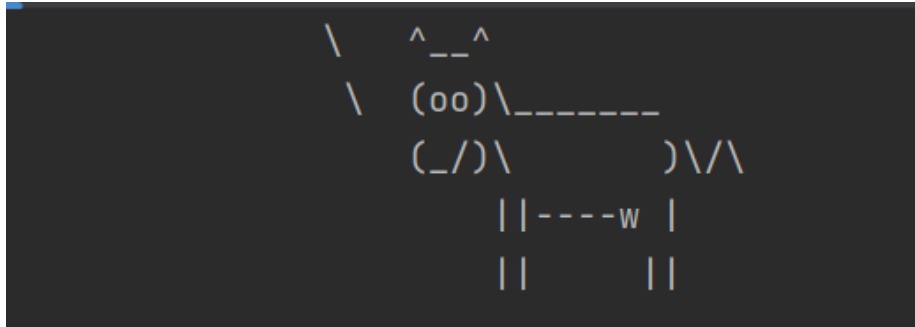
// Affiche la vache à la position pos, avec animation de la langue
void affiche_vache(int pos, int frame) {
    // Dessin statique de la vache
    const char *lines[] = {
        "  \ \      ^__^",
        "  \ \    (oo)\ \____",
        "      (__)\ \)  /\ \/",
        "          ||----w |",
        "          ||     ||"
    };
    // Affichage ligne par ligne
    for(int i = 0; i < 5; i++) {
        gotoxy(pos, i + 1);
        printf("%s\n", lines[i]);
    }
    // Animation de la langue ou œil
    if(frame % 2 == 0) {
        gotoxy(pos + 6, 3);
        printf("\ \ \n"); // langue vers la gauche
    } else {
        gotoxy(pos + 7, 3);
        printf("/ \n"); // langue vers la droite
    }
}

int main() {
    int pos = 1, dir = 1;
    int frame = 0;
    // Boucle d'animation (50 frames)
    while(frame < 50) {
        update();
        affiche_vache(pos, frame);
        usleep(200000); // 0.2 seconde
        pos += dir;
        if(pos > 40 || pos < 1) dir = -dir; // rebond
        frame++;
    }
    return 0;
}

```

Compilation & Exécution

```
gcc -o wilddcow wilddcow.c
./wilddcow
```



5. reading_cow.c – Lecture animée caractère par caractère

Objectif

Créer un programme qui : 1. Lit un fichier (ou `stdin`) char par char. 2. Affiche chaque caractère successivement dans une bulle de `cowsay`. 3. “Avaler” le caractère (il reste dans la bulle), avec pause de 1 seconde.

Code source

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

// Efface l'écran
void update() {
    printf("\033[H\033[J");
}

// Affiche une bulle de texte autour de msg
void affiche_bulle(const char *msg, const char *next) {
    int len = strlen(msg);
    // Dessus de la bulle
    printf(" ");
    for(int i = 0; i < len + 2; i++) printf("_");
    printf("\n");
    // Ligne du message
    printf("< %s >\n", msg);
    // Bas de la bulle
```

```

printf(" ");
for(int i = 0; i < len + 2; i++) printf("-");
printf("\n");

printf(" \\   ^__^\\n");
printf("  \\  (oo)\\____\\n");
printf("     (__)\\        )\\/\\n");
printf("       %s ||----w |\\n", next);
printf("       ||      ||\\n");

}

int main(int argc, char *argv[]) {
FILE *fp = stdin;
if(argc > 1) {
    fp = fopen(argv[1], "r");
    if(!fp) {
        perror("fopen");
        return 1;
    }
}
char buffer[1024] = {0};
char buffer1[1024] = {0};
char ch;
char next[2] = {0};

// Lecture caractère par caractère
while((ch = fgetc(fp)) != EOF) {
    size_t len = strlen(buffer);
    if(len < sizeof(buffer) - 1) {
        buffer[len] = ch;
        buffer[len + 1] = '\\0';
    }

    if (len > 0) {
        buffer1[0] = ' ';
        for (int i = 0; i < len; i++) {
            buffer1[i + 1] = buffer[i];
        }
        buffer1[len + 2] = '\\0'; // Finir proprement
        next[0] = buffer[len];
    } else {
        buffer1[0] = '\\0';
        next[0] = ' ';
    }
}

```

```

    }
    next[1] = '\0'; // Finir next aussi

    update();
    affiche_bulle(buffer1, next);
    sleep(1); // pause 1 seconde
}

if(fp != stdin) fclose(fp);
return 0;
}

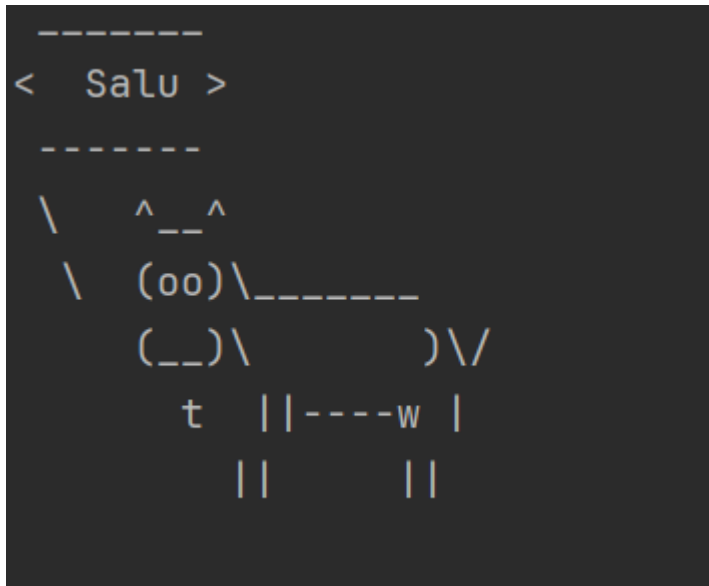
```

Compilation & Exécution

```

gcc -o reading_cow reading_cow.c
./reading_cow file.txt

```



Commentaires : - `affiche_vache()` gère l’affichage, paramétré par `eyes` et `tail`. - `main()` parse les options et valide les arguments. - `update()` et `gotoxy()` permettent un affichage animé. - `reading_cow.c` gère dynamiquement la taille de la bulle en fonction du message. - Cette partie met en pratique la manipulation du terminal et la gestion de flux d’entrée/sortie en C.

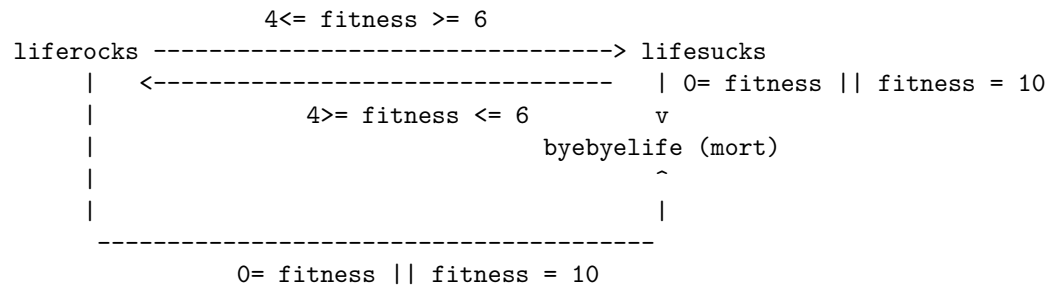
Partie Automates

Dans cette section, nous modélisons et implémentons un **Tamagochi-vache** à l'aide d'un automate à trois états, puis codons le jeu en C.

1. Automate des États de Santé

Les états de la vache : - **liferocks** (en pleine forme) - **lifesucks** (état intermédiaire) - **byebyelife** (décédée)

Les transitions dépendent de la **quantité de nourriture (lunchfood)** choisie par le joueur, ainsi que de variables aléatoires **digestion** et **crop**.



Sorties : - Affichage visuel de la vache selon son état - Affichage de la valeur de la réserve (**stock**) - Affichage du score final (durée de vie) à la fin du jeu

2. Code tamagoshi_cow.c

2.1. Déclarations et includes

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

// États de la vache
typedef enum { BYEBYLIFE = 0, LIFESUCKS = 1, LIFEROCKS = 2 } State;

// Variables globales
int stock = 5; // Réserve initiale
int fitness = 5; // Santé initiale
int duration = 0; // Durée de vie
```



```

State state = LIFEROCKS;

do {
    update();
    // Déterminer l'état selon 'fitness'
    if (fitness >= 4 && fitness <= 6) {
        state = LIFEROCKS;
    } else {
        state = LIFESUCKS;
    }
    // Si extrême, la vie s'achève
    if (fitness == 0 || fitness == 10) {
        state = BYEBYLIFE;
        break;
    }

    // Affichage
    affiche_vache(state);
    printf("Stock: %d \n", stock );
    printf("Durée: %d\n", duration);
    printf("Quantité de nourriture à donner (0 à %d) : ", stock);

    int lunchfood;
    scanf("%d", &lunchfood);
    if (lunchfood < 0) lunchfood = 0;
    if (lunchfood > stock) lunchfood = stock;

    // Mise à jour des variables
    update_fitness(lunchfood);
    update_stock(lunchfood);

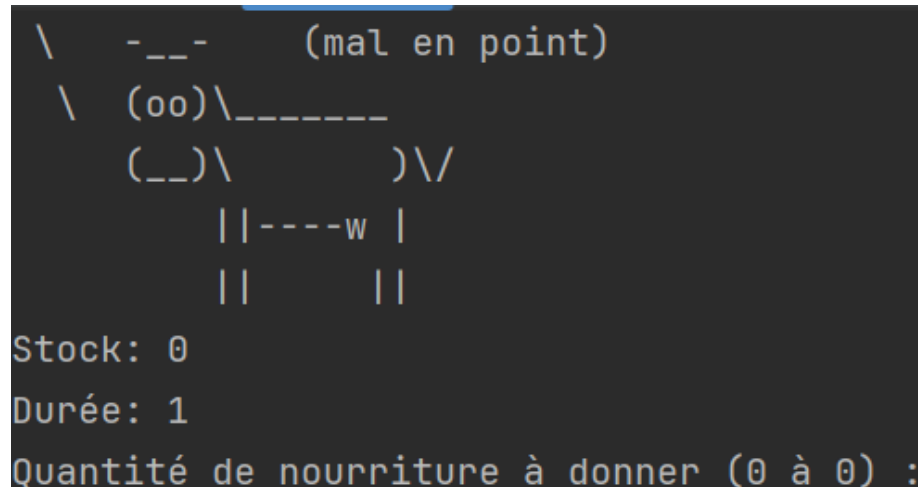
    duration++;
} while (state != BYEBYLIFE);

// Fin du jeu
update();
affiche_vache(BYEBYLIFE);
printf("Game Over! Durée de vie: %d étapes\n", duration);
return 0;
}

```

3. Compilation & Exécution

```
gcc -o tamagoshi_cow tamagoshi_cow.c  
./tamagoshi_cow
```



```
\  _--_      (mal en point)
 \ (oo)\_____
    (__)\       )\/\
        ||----w |
        ||     ||

Stock: 0
Durée: 1
Quantité de nourriture à donner (0 à 0) :
```

Lors de l'exécution, la vache change d'état en fonction de ton choix de nourriture et de l'aléa, jusqu'à sa mort. Le score final est le nombre d'itérations survivies.

4. Commentaires et Observations

- **Automate** : modélisation fidèle des états et transitions basées sur fitness.
- **Aléatoire** : digestion et crop ajoutent de la variabilité, rendant le jeu plus imprévisible.
- **Interface** : simple CLI, mais extensible (ex : choix de plusieurs vaches, animations).
- **Extensions possibles** : ajout de nouveaux états, gérer un inventaire, sauvegarder le score.