Projet Cowsay

1 Présentation du Projet

Le projet débute au premier jour du cours INF203 et s'achève lors la dernière semaine de cours. Cette dernière fait date de rendu (dimanche soir minuit de la dernière semaine). Vous pouvez progresser sur le projet à votre rythme, mais nous vous recommandons de prendre de l'avance par rapport au cours, du moins aucun retard. Par exemple, la partie "Bash" devra être achevée au moment où les premiers cours de "C" débuteront.

1.1 Objectif du projet

```
couiller@laptop-450:~$ cowsay "Bonjour"
 ---------
< Bonjour >
 ---------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

L'objectif du projet est de découvrir le monde merveilleux de "`cowsay`". Au cours du projet, vous réaliserez les objectifs suivants:

1. **Préliminaires**. Découvrir la commande `cowsay` à travers son manuel (manpage) et l'ensemble des options qu'elle contient.
2. **Bash**. Implémenter un script Bash qui fait réciter à la vache la suite des nombres premiers, des nombres de Fibonacci, ou toute autre suite exotique de votre choix.
3. **C**. Recoder `cowsay` en C, avec de nouvelles fonctionnalités additionnelles de votre choix (comme par exemple la longueur de la queue).
4. **Automates**. En s'appuyant sur la théorie des automates, implémenter un "cow-Tamagoshi" qu'il s'agit de nourrir et faire survivre aussi longtemps que possible.

1.2 Rapport attendu

Le compte-rendu du projet doit être rédigé sous un format texte (texte simple, Markdown, Org) ou en LaTeX. Il doit comporter à la fois les codes mais également **et surtout** des commentaires et interprétations clairs et pertinents concernant (i) vos choix (justifiés, en expliquant par exemple si d'autres tentatives précédentes ont échoué et pourquoi) ainsi que (ii) vos résultats, en rapport avec les notions vues en cours. Les idées et les implémentations les plus originales et avancées seront fortement valorisées.

Précisément, le compte-rendu doit contenir a minima:

- les codes sources des exercices demandés.
- les différentes sorties d'exécutions de ces codes (ou des programmes compilés) pour plusieurs tests d'arguments pertinemment choisis.
- un maximum de commentaires à la fois dans le code (afin de le rendre lisible) ainsi que dans le compte-rendu lui-même pour justifier vos choix, vos idées, et même mentionner vos difficultés, écueils et stratégies d'adaptation quand cela a été nécessaire.

2 Préliminaires

Dans cette première section, il vous est demandé de découvrir l'exécutable

```
cowsay
```

Pour cela, lisez consciencieusement le manuel de cowsay à travers la commande

```
man cowsay
```

À l'aide du manuel, explorez les différentes options disponibles dans cowsay. Dans votre rapport, vous reporterez l'ensemble des arguments dans une liste ou un tableau en explicitant leur usage et en donnant quelques exemples d'exécutions.

3 Bash

Dans cette section, l'objectif est de proposer un script shell (bash) consistant à faire dynamiquement lister par la vache les itérés d'une suite de nombres. Pour cela, vous pourrez notamment utiliser la fonction

```
clear
```

qui rafraichit l'écran ainsi que la fonction

```
sleep
```

qui prend en argument une durée t et crée une pause de longueur t secondes (voir le manpage de sleep).

Vous produirez successivement les scripts suivants:

- un script `cow_kindergarten` dans lequel la vache prononce les chiffres de 1 à 10 avec une pause d'une seconde entre chaque chiffre. La vache tirera la langue à la fin de l'exercice.
- un script similaire, `cow_primaryschool`, mais cette fois-ci la vache prononce les chiffres de 1 à n avec n un nombre donné en argument du script.
- toujours dans cette vague, un script `cow_highschool` où cette fois-ci la vache est plus forte et prononce la suite des carrés 1, 4, 9, ..., n^2 avec n l'argument du script.
- afin de complexifier légèrement la tâche, la vache prononcera maintenant dans `cow_college` la suite des nombres de Fibonacci inférieur à n. Pour cela vous chercherez sur Internet la définition de cette suite.
- même question avec le script `cow_university` maintenant pour la suite des nombres premiers inférieurs à n.
- dans un script `smart_cow`, vous réaliserez ensuite une vache qui résout un calcul numérique simple (addition, soustraction, multiplication, division) d'au plus deux chiffres. Le script prendra pour argument une chaîne de caractère (par exemple "`3+11`") et retournera une vache qui prononce le calcul et dont les yeux se transforment en le résultat du calcul.
- enfin, laissez libre court à votre imagination pour créer un script `crazy_cow` qui implémente une "vache arithmétique" qui fait quelque chose de particulièrement fou (ou compliqué).

```
couiller@laptop-450:~$ smart_cow '3 + 11'
 --------
< 3 + 11 >
 --------
        \   ^__^
         \  (14)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

4 C

Dans cette seconde partie du projet, vous allez recoder la routine `cowsay` de sorte à lui apporter de nouvelles fonctionnalités.

Vous opèrerez successivement comme suit:

1. Produisez tout d'abord un code source `newcow.c` dans lequel vous créerez une fonction `affiche_vache` ne prenant pour le moment pas d'argument, et qui affiche simplement une vache sans la bulle de texte. On pourra utiliser pour cela la chaîne de caractères suivante:

```
"          \   ^__^\n\
           \  (00)\\_______\n\
              (__)\\       )\\/\\\n\
                  ||----w |\n\
                  ||     ||\n"
```

   Dans l'appel de la fonction `main`, vous effectuerez simplement l'appel à la fonction `affiche_vache`. Compilez et exécutez le programme.

2. Il s'agit désormais d'ajouter des fonctionnalités à `newcow`. Pour cela, modifiez `newcow.c` de sorte à ce que `newcow` reçoive et filtre les arguments `-e` (ou `--eyes`) et une chaîne de deux caractères qui modifie les yeux de la vache. Adaptez conformément votre appel à la routine `affiche_vache`. Compilez et testez. Faire ensuite de même pour un certain nombre d'options supplémentaires récupérées dans le manpage de cowsay.

3. Dans un second temps, en faisant preuve d'imagination (plus l'idée est techniquement élaborée, drôle et surprenante, meilleure sera votre note!), vous implémenterez une nouvelle fonctionnalité à `newcow`. Par exemple, vous pourriez ajouter une option qui allonge la taille de la queue de la vache d'une valeur `L` donnée en argument (par exemple avec un appel du type `newcow --tail 4`). On peut aussi imaginer imprimer un "troupeau" de vaches plutôt qu'une seule vache...

4. Nous allons maintenant créer une "vache animée" `wildcow`. Pour cela, testez tout d'abord et expliquez le fonctionnement des deux routines `update()` et `goto()` décrites comme suit:

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

   On pourra par exemple imprimer plusieurs lignes test (du type `printf('Ligne 1\nLigne 2\n')`) et vérifier l'action de `update()` et `goto()` sur ces lignes.

   En utilisant ces deux commandes ainsi que la routine `sleep()` disponible en C via `#include <unistd.h>`, créez une première vache animée de votre choix (elle pourra par exemple avoir des yeux ou une langue qui bougent, marcher vers l'avant ou vers l'arrière, etc.). Comme précédemment, plus le résultat est original, plus vous gagnerez de points à cette question!

5. Nous allons enfin créer une vache animée qui apprend à lire, `reading_cow`. Pour cela, rédigez un code `reading_cow.c` qui prendra comme argument un fichier dans lequel sera écrit un ou plusieurs mots (on pourra ajouter un paramètre pour que le fichier pris en compte soit l'entrée standard `stdin`). L'objectif est de faire lire le fichier à la vache, caractère par caractère: chaque caractère apparaitra alors successivement dans la gueule de la vache et sera "avalé" et alors placé à la suite des caractères déjà avalés dans la bulle de texte. Une seconde se passera entre chaque caractère affiché. On devrait typiquement obtenir ce genre d'image au milieu de l'exécution de `reading_cow file` si le fichier `file` contient la chaîne de caractères `bonjour`.

```
couiller@laptop-450:~$ reading_cow file
 _____
< bon >
 _____
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
             j  ||----w |
                ||     ||
```

5 Automates

Dans cette dernière section, vous allez créer votre premier Tamagoshi-vache. Il s'agit de mettre en place un petit jeu vidéo dans lequel vous devez nourrir votre vache afin qu'elle survive aussi longtemps que possible. Le score du joueur à la fin du jeu est précisément ce temps de survie. Le jeu se déroule ainsi:

- à chaque instant du jeu, votre vache se trouve dans l'un des trois *états* suivants: `liferocks` (en pleine forme), `lifesucks` (ne se sent pas bien, nauséeuse), `byebyelife` (décédée).
- chaque état est associé à un *niveau de santé* `fitness`: 0 (morte de faim) et 10 (morte de suralimentation) pour l'état `byebyelife`, 1 à 3 (déficit de nourriture) et 7 à 9 (excédent de nourriture) pour l'état `lifesucks` et 4 à 6 pour l'état `liferocks`. Au début du jeu, la vache se trouve dans l'état `liferocks` avec un niveau de santé `life=5`. À chaque pas de temps pendant le jeu, ce niveau de santé va évoluer.
- le joueur observe à tout moment l'*état* de la vache (grâce à trois affichages de vaches différents) mais n'est jamais au courant de son *niveau de santé*.
- le joueur a par ailleurs à sa disposition une *réserve* de nourriture qui évolue avec le temps et mesurable via la variable `stock` comprise entre 0 (réserve "à sec") et 10 (réserve "saturée"). La réserve initial est composée de `stock=5` unités de nourriture. À chaque pas de temps du jeu, le joueur décide de nourrir la vache d'une quantité `lunchfood` de nourriture extraite de la réserve (et donc évidemment `lunchfood`$\leq$`stock`).
- après que le joueur a choisi le valeur de `lunchfood`, une variable aléatoire `digestion` comprise entre 0 (digestion optimale des apports) et -3 (digestion difficile) s'ajoute à `lunchfood` pour faire évoluer le niveau de santé `fitness` de la vache. Le niveau de santé devient donc `(fitness+lunchfood)+digestion` (on s'assurera bien sûr que `fitness` reste comprise entre 0 et 10).
- de la même manière, après que le joueur a choisi le valeur de `lunchfood`, une variable aléatoire `crop` comprise entre -3 (la nourriture de la réserve devient avariée) et +3 (de nouvelles récoltes sont apportées à la réserve) fait évoluer le niveau de la réserve. La variable `stock` devient donc `(stock-lunchfood)+crop` (on s'assurera bien sûr que `stock` reste comprise entre 0 et 10).

Dans un premier temps, afin de mettre le problème au clair, vous dessinerez un automate des trois *états* de santé de votre vache. Vous pourrez pour cela procéder comme suit:

- pour dessiner l'automate, vous supposerez que les variables `fitness`, `digestion` et `stock` sont figées et "connues" (elles ne seront simplement pas visuellement affichées par le programme)
- l'unique action possible est celle de l'utilisateur qui choisit la valeur `lunchfood` dans son intervalle de validité (qui est fonction de `stock`). Les transitions dépendent alors de l'intervalle dans lequel `lunchfood` se trouve: cet intervalle dépend de `fitness`, `digestion` et `stock`.
- les sorties possibles correspondent (i) à l'affichage visuel de la vache dans son nouvel état, (ii) l'affichage visuel de la valeur `stock` mise à jour, et potentiellement (iii) l'affichage du score final au moment de la conclusion du programme.

Si vous le souhaitez, il vous sera possible de modéliser les actions des 2 variables aléatoires `digestion` et `crop` comme les actions menées par 2 autres acteurs du jeu (par exemple avec des flèches de couleurs distinctes de la couleur des actions du joueur). Cet automate "sur papier" vous sera fort utile pour valider votre code.

Vous mettrez alors en place un code `tamagoshi_cow.c` qui implémentera le jeu vidéo. Pour ce faire, vous pouvez (sans obligation mais nous vous le conseillons) procéder séquentiellement ainsi:

1. réexploitez la routine `affiche_vache` que vous avez précédemment mise en place pour qu'elle prenne en paramètre l'*état* de santé de la vache (par exemple `byebyelife=0` pour "décédée", `lifesucks=1` pour "ne se sent pas bien" et `liferocks=2` pour "en pleine forme"). L'affichage de la vache sera différent pour chaque état (à vous d'imaginer une représentation parlante!).

2. vous créerez les deux variables globales `stock` et `fitness`, toutes deux étant des entiers compris entre 0 et 10 et initialisés à `5`.

3. créez deux routines `stock_update()` et `fitness_update()` qui mettent à jour les variables `stock` et `fitness`, respectivement. Les deux fonctions prendront en argument la quantité de nourriture `lunchfood` allouée à la vache par le joueur et retourneront la quantité mise à jour en prenant en compte l'évolution aléatoire du niveau de forme de la vache et du stock (du fait des variables `crop` et `digestion`). Pour cela, on pourra utiliser la fonction `rand()` comme suit (pensez à l'initialiser au moins une fois):

```c
#include <stdlib.h>

int main(){
  ...
  /* Intialise le generateur de nombres aleatoires */
  time_t t;
  srand((unsigned) time(&t));
  ...
  /* Genere un nombre aleatoire entre 0 et 3 */
  int nombre;
  nombre = rand() % 3;
}
```

4. dans votre `main`, vous créerez une boucle `while` qui ne s'achèvera que lorsque la vache passe dans l'état `byebyelife`. Au sein de cette boucle, chaque itération:

   (a) affiche votre Tagmagoshi-vache dans son état actuel.
   Attention: le joueur ne doit jamais avoir accès à la valeur de la variable `fitness` (sinon le jeu serait sans intérêt), mais doit seulement voir l'état de la vache (parmi `liferocks`, `lifesucks` et `byebyelife`) grâce à l'affichage visuel.

(b) affiche l'état de la variable `stock`.

(c) demande ensuite à l'utilisateur d'entrer une quantité de nourriture (inférieure ou égale à la taille `stock` de la réserve) à allouer à la vache.

(d) met à jour les variables `stock` et `fitness` ainsi que les états de la réserve et de la vache.

(e) incrémente une variable `duree_de_vie` qui compte le nombre de passages à l'intérieur de la boucle.

Bien évidemment, le but du jeu est de faire en sorte que `duree_de_vie` soit la plus longue possible.

Bonus. Pour les plus imaginatifs (ou les plus fervents "gamers") d'entre vous, sentez-vous libres d'apporter à votre logiciel Tamagochi-vache des évolutions bien senties, pertinentes et drôles!

À vous de jouer!