Exercices d'initiation au C

PROG
François Schwarzentruber

Exercice 1 (environnement, afficher du texte) *Écrire le programme 'Hello World !' en C.*

Exercice 2 (afficher du texte, boucle) *Écrire un programme C qui affiche un triangle :*

```
*
**
***
****
*****
```

Exercice 3 (afficher du texte, boucle) *Écrire un programme C qui affiche une pyramide :*

```
    *
   ***
  *****
 *******
*********
```

Exercice 4 (redirection, lecture) *Écrire un programme qui produit un exécutable* `nombreligne` *tel que la commande* `./nombreligne < fichier.txt` *affiche le nombre de lignes dans* `fichier.txt`*.*

Pareil pour renvoyer le nombre de mots.

Exercice 5 (redirection, lecture, tableau) *Écrire un programme qui produit un exécutable* `histogramme` *tel que la commande* `./histogramme < fichier.txt` *affiche l'histogramme qui indique le nombre de fois que chaque lettre est présente dans le texte contenu dans* `fichier.txt`*. Par exemple :*

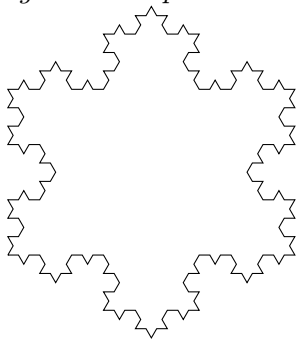
```
a ***
b ******
c **
d *
e ************
z ***
```

Exercice 6 (vecteur de bits) *Écrire un programme qui produit un exécutable* `touteslestres` *tel que la commande* `.touteslestres < fichier.txt` *écrit* `oui` *si le texte dans* `fichier.txt` *contient contient toutes les lettres de A à Z,* `non` *sinon. On demande à ce que la mémoire supplémentaire utilisée soit au plus de 4 octets.*

Exercice 7 *(*) Écrire un programme C qui écrit dans le terminal du code SVG pour :*

- *un triangle de triangle de Sierpiński*

- *un flocon de Koch*



Exercice 8 *(*) Écrire un programme C qui affiche un labyrinthe dans le terminal et/ou alors du code SVG :*

```
*********
*  *    *
* ***** *
*       *
***** ***
*       *
*********
```

Syntaxe du C et Python

PROG
François Schwarzentruber

	C	Python

Conditionnelle	`if (x > 0) {`
	`  printf("Positif\n");`
	`} else {`
	`  printf("Non positif\n");`
	`}`
		`if x > 0:`
		`  print("Positif")`
		`else:`
		`  print("Non positif")`

Boucle for	`for (int i = 0; i < 5; i++) {`
	`  printf("%d\n", i);`
	`}`
		`for i in range(5):`
		`  print(i)`

Boucle while	`int i = 0;`
	`while (i < 5) {`
	`  printf("%d\n", i);`
	`  i++;`
	`}`
		`i = 0`
		`while i < 5:`
		`  print(i)`
		`  i += 1`

Écrire dans le terminal	`printf("Bonjour %s\n", name);`	`print(f"Bonjour {name}")`

Fonction	`int add(int a, int b) {`
	`  return a + b;`
	`}`
		`def add(a, b):`
		`  return a + b`

Type structuré	`struct Point {`
	`  int x;`
	`  int y;`
	`};`
	
	`struct Point p1 = {10, 20};`
		`class Point:`
		`  def __init__(self, x, y):`
		`  self.x = x`
		`  self.y = y`
		
		`p1 = Point(10, 20)`

Types primitifs en C

Type en C	**Type en Rust**	**Alias C99**	**Taille (en bits)**	**Plage de valeurs**
`signed char`	`i8`	`int8_t`	8	entiers entre -128 et 127
`unsigned char` `char`	`u8` $\neq$ `char`	`uint8_t`	8	entiers entre 0 et 255
`short` `signed short`	`i16`	`int16_t`	16	entiers entre -32768 et 32767
`unsigned short`	`u16`	`uint16_t`	16	entiers entre 0 et 65535
`int` `signed int`	`i32`	`int32_t`	32	entiers entre -2^{31} et $2^{31}-1$
`unsigned int`	`u32`	`uint32_t`	32	entiers entre 0 et $2^{32}-1$
`long` `signed long`	`i64`	`int64_t`	64	entiers entre -2^{63} et $2^{63}-1$
`unsigned long`	`u64`	`uint64_t`	64	entiers entre 0 et $2^{64}-1$
`float`	`f32`	-	32	nombres flottants norme IEEE 754 sur 32 bits $(-1)^{[\text{signe sur 1 bit}]} \times 2^{[\text{exposant sur 8 bits}]-127} \times [\text{mantisse sur 23 bits}]$
`double`	`f64`	-	64	nombres flottants norme IEEE 754 sur 64 bits $(-1)^{[\text{signe sur 1 bit}]} \times 2^{[\text{exposant sur 11 bits}]-1023} \times [\text{mantisse sur 52 bits}]$
`_Bool` avec `bool` avec `#include <stdbool.h>`	`bool`	-	8	0 et 1 `false` et `true`

Entrée/sortie en C

PROG
François Schwarzentruber

	Lire (`stdin`)	**Écrire** (`stdout`)
Caractère unique	`int c = getchar();` ou `int c = fgetc(stdin);` `miaou, répond le chat`	`putchar('m');` ou `fputc('m', stdout);` `lechatditm`

Suite de caractères	`chaine = fgets(strDest, 11, stdin);`
	`le chat dit miaou` ou
	`je vois`
	`bien que le chat est content`
		`fputs("miaou", stdout);`
		`lechatditmiaou`
		`puts("miaou")`
		`lechatditmiaou`

Données formatées	`nbVariablesLues = scanf("%d", &x);`
	`42 miaous`
	`nbVariablesLues = scanf("%d/%d/%d", &jour, &mois, &annee);`
	`scanf("%8s", str);`
		`printf("le poids du chat = %d", x);`
		`je vous dis que`
		`le poids du chat = 10`

`%d` `%i` `%u` `%o` `%x` `%X` `%c` `%s` `%f` `%.2f` `%e` `%E` `%g` `%G` `%p` `%%`

Commandes principales git

PROG
François Schwarzentruber

Initialisation

Commande	Description
`git init`	Crée un dépôt local
`git clone <url>`	Clone un dépôt distant

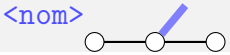
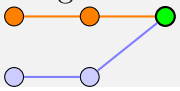
Suivi des fichiers

Commande	Description
`git status`	Affiche l'état des fichiers
`git add <fichier>`	Ajoute un fichier à l'index 
`git rm <fichier>`	Supprime un fichier suivi

Commits et historique

Commande	Description
`git commit -m "message"`	Enregistre les modifications en local
`git commit -am "message"`	Enregistre les modifications de tous les fichiers suivis
`git commit --amend`	Modifie le dernier commit
`git log --oneline`	Historique condensé

Branches

Commande	Description
`git branch`	Liste les branches
`git branch <nom>`	Crée une nouvelle branche `<nom>` 
`git switch <nom>`	Bascule sur la branche `<nom>`
`git merge <nom>`	Intègre la branche `<nom>` dans celle courante 

Dépôt distant

Commande	Description
`git remote -v`	Liste les dépôts distants
`git remote add origin <url>`	Ajoute un dépôt distant
`git push origin <branche>`	Envoie les commits
`git pull origin <branche>`	Récupère les mises à jour et tente un merge
`git pull --rebase` (on résout les conflits) `git rebase --continue`	Récupère les mises à jour Valide la résolution des conflits

Exercices sur les pointeurs

PROG
François Schwarzentruber

Exercice 9 (effet de bord) *Écrire une fonction* `void swap(int* x, int* y);` *qui échange les valeurs de deux variables.*

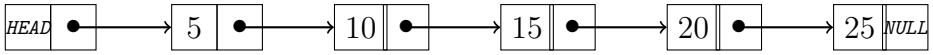
Exercice 10 (effet de bord) *Écrire un type* `struct` *pour représenter un point en 2D. Écrire des fonctions 1) pour translater un point de* (dx, dy) *en place, 2) pour tourner un point autour de l'origine d'un certain angle.*

Exercice 11 (effet de bord) *Écrire une fonction* `void calculer(int a, int b, int *somme, int *produit)` *qui calcule à la fois la somme et le produit.*

Exercice 12 (Tableau dynamique) *On implémente une structure de tableau dynamique pour les flottants.*

1. *Définir un type* `dynarray` *pour les tableaux dynamiques.*
2. *Implémenter une fonction* `dynarray dynarray_create()` *qui crée un tableau dynamique vide.*
3. *Implémenter l'ajout en place d'un élément en fin de tableau :*
   `void dynarray_push_back(dynarray a, float el);`.
4. *Implémenter une fonction qui donne la taille du tableau :*
   `int dynarray_size(dynarray a);`.
5. *Implémenter l'accès à la i-ème case :*
   ```
   float dynarray_get(dynarray a, int i);
   void dynarray_set(dynarray a, int i, float el);
   ```
6. *Implémenter la libération de la mémoire*
   `void dynarray_free(dynarray a);`.

Exercice 13 (File avec une liste chaînée) *Une liste chaînée est modélisée avec des* maillons. *Un maillon contient deux champs : une valeur et un pointeur* `NULL` *ou vers un maillon.*



L'objectif est de créer un type `queue` *proposer une structure de données basée sur la notion de liste simplement chaînée qui implémente les opérations d'une file :*

1. *Implémenter un* `struct` *qui représente un maillon d'une liste chaînée et un* `struct`.
2. *Définir le type* `queue`.
3. *Implémenter une fonction* `queue queue_create()` *qui crée une file vide.*
4. *Implémenter l'enfilage et le retrait d'un élément dans la file en* $O(1)$ *:*
   ```
   void queue_enqueue(queue q, float el);
   float queue_dequeue(queue q);
   ```
5. *Implémenter la libération de la mémoire avec*
   `void queue_free(queue q);`.

Conteneurisation avec Docker

PROG
François Schwarzentruber

Définition d'une classe dans plusieurs langages

PROG
François Schwarzentruber

Python

```python
class Personne:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age

    def parler(self):
        print(f"Je suis {self.nom}, {self.age} ans.")
```

Java

```java
public class Personne {
    private String nom;
    private int age;

    public Personne(String nom, int age) {
        this.nom = nom;
        this.age = age;
    }

    public void parler() {
        System.out.println("Je suis " + nom +
                           ",  " + age + " ans.");
    }
}
```

JavaScript (ES6)

```javascript
class Personne {
  constructor(nom, age) {
      this.nom = nom;
      this.age = age;
  }

  parler() {
    console.log(`Je suis ${this.nom}, ${this.age} ans.`);
  }
}
```

C++

```cpp
class Personne {
private:
    std::string nom;
    int age;

public:
    Personne(std::string n, int a) : nom(n), age(a) {}
    void parler() {
        std::cout << "Je suis " << nom
                  << ",  " << age << " ans."
                  << std::endl;
    }
};
```

Rust

```rust
struct Personne {
    nom: String,
    age: u32,
}

impl Personne {
    fn new(nom: String, age: u32) -> Self {
        Personne { nom, age }
    }

    fn parler(&self) {
        println!("Je suis {}, {} ans.",
                 self.nom, self.age);
    }
}
```

Définition de classes dans plusieurs langages

PROG
François Schwarzentruber

Python

```python
from typing import override

class Animal:
    def __init__(self, nom: str) -> None:
        self.nom: str = nom

    def parler(self) -> None:
        print("L'animal fait un bruit.")

class Chat(Animal):
    def __init__(self, nom: str, longueurMoustache: int) -> None:
        super().__init__(nom)
        self.longueurMoustache: int = longueurMoustache

    @override
    def parler(self) -> None:
        print(f"{self.nom} miaule avec ses {self.longueurMoustache}mm de moustaches.")
```

```python
felix: Chat = Chat("Felix", 120)
felix.parler()
```

Java

```java
class Animal {
    protected String nom;

    public Animal(String nom) {
        this.nom = nom;
    }

    public void parler() {
        System.out.println("L'animal fait un bruit.");
    }
}

class Chat extends Animal {
    private int longueurMoustache;

    public Chat(String nom, int longueurMoustache) {
        super(nom);
        this.longueurMoustache = longueurMoustache;
    }

    @Override
    public void parler() {
        System.out.println(nom + " miaule avec ses " + longueurMoustache + "mm de moustaches.");
    }
}
```

```java
Chat felix = new Chat("Felix", 120);
felix.parler();
```

JavaScript (ES6)

```javascript
class Animal {
    constructor(nom) {
        this.nom = nom;
    }
    parler() {
        console.log("L'animal fait un bruit.");
    }
}

class Chat extends Animal {
    constructor(nom, longueurMoustache) {
        super(nom);
        this.longueurMoustache = longueurMoustache;
    }

    parler() {
        console.log(`${this.nom} miaule avec ses ${this.longueurMoustache}mm de moustaches.`);
    }
}
```

```javascript
const felix = new Chat("Felix", 120);
felix.parler();
```

C++

```cpp
class Animal {
    protected:
        std::string nom;

    public:
        Animal(std::string n) : nom(n) {}
        virtual void parler() {
            std::cout << "L'animal fait un bruit." << std::endl;
        }
};

class Chat : public Animal {
    private:
        int longueurMoustache;

    public:
        Chat(std::string n, int l) : Animal(n), longueurMoustache(l) {}

        void parler() override {
            std::cout << nom << " miaule avec ses "
                      << longueurMoustache << " mm de moustaches."
                      << std::endl;
        }
};
```

```cpp
Chat felix("Felix", 120);
felix.parler();
```

Rust

```rust
trait Animal {
    fn parler(&self);
}

struct Chat {
    nom: String,
    longueur_moustache: u32,
}

impl Animal for Chat {
    fn parler(&self) {
        println!("{} miaule avec ses {} mm de moustaches.",
                 self.nom, self.longueur_moustache);
    }
}

impl Chat {
    fn new(nom: &str, longueur_moustache: u32) -> Self {
        Chat {
            nom: nom.into(),
            longueur_moustache
        }
    }
}
```

```rust
let felix = Chat::new("Felix", 120);
felix.parler();
```

Bonnes pratiques de programmation

PROG
François Schwarzentruber

Code

☐ Mes noms de variables et fonctions sont informatifs. Ni trop longs, ni trop courts.
☐ J'utilise le même style pour les noms (pas de `camelCase`, puis un peu de `snake_case`)
☐ Pas de changement de langues en tout cas (pas de franglais).
☐ Je programme en anglais de préférence (noms des variables, et commentaires).
☐ Mes noms de classes, fonctions sont conformes à des concepts classiques (itérateurs, patrons de conception, etc.)
☐ Je privilégie des fonctions sans effets de bord (quand c'est pertinent).
☐ Je suis cohérent dans la mise en forme du code. J'utilise des outils comme `pycodestyle`, `pydocstyle`, `pylint`, ou `black` pour formater le code.
☐ Je respecte un guide de style (PEP8 pour Python par exemple).
☐ Je spécifie chaque fonction (entrées/sorties/effets).
☐ Le code de chaque fonction est court. J'ai découpé mon code.
☐ J'ai évité trop de niveaux d'imbrication de `if`, `for`, `while`.
☐ Mon code est généralement sans surprise. Si surprise il y a, commentaire il y a.
☐ J'écris des tests unitaires. J'utilise des outils comme `pytest`, `coverage`, etc.
☐ Si je code dans un langage bas niveau, il n'y a pas d'erreur de segmentation et pas de fuite de mémoire. J'ai utilisé `valgrind` pour m'en assurer.

Organisation du projet

☐ Une classe a (généralement) qu'une seule responsabilité. Pas de classes à rallonge qui font la cuisine, le ménage et de la musique.
☐ Le couplage est réduit : on dépend d'interfaces (ou classes abstraites selon le contexte) et non pas d'implémentation (classes concrètes).
☐ J'applique des patrons de conception quand cela est pertinent.
☐ Je comprends l'architecture de mon projet : j'ai conscience de fonctionnalités faciles à ajouter au projet et de celles qui le sont moins.

Dépôt

☐ Le dépôt contient un `README.md` qui explique le but du logiciel, comment l'installer et le lancer.
☐ Le `README.md` contient, si nécessaire, un schéma qui explique l'utilisation ou le logiciel.
☐ J'ai facilité l'installation des bibliothèques annexes pour la compilation et/ou l'exécution. Par exemple `manifest.scm`, `requirements.txt`, etc.
☐ Je n'ai pas de dépendances à des bibliothèques inutiles.
☐ La compilation est facile. Il y a un `Makefile` ou équivalent pour construire les binaires facilement.
☐ J'ai mis en place des mécanismes d'intégration continue.
☐ Je respecte des conventions de nommage pour les branches, e.g. `main` (production), `develop`, `feature/newinstrument`, `bugfix/savingmusic`