Carnet de Travaux Pratiques
Algorithmique et structures de données
Licence 2 Informatique

Julien Bernard

Table des matières

Travaux Pratiques d'Algorithmique n°1

Exercice 1 : Mon premier programme en C

Le langage C est le langage que nous allons utiliser et apprendre pendant le cours d'Algorithmique. C'est un langage compilé, c'est-à-dire qu'il est nécessaire d'appeler un compilateur pour produire un exécutable qui sera fonctionnel uniquement sur la machine sur laquelle il a été compilé.

Question 1.1 Recopier le code source suivant dans un fichier appelé `hello.c`.

```
#include <stdio.h>

int main() {
  printf("Hello World\n");
  return 0;
}
```

Question 1.2 Compiler le programme avec la ligne de commande suivante :

```
gcc -Wall -std=c99 -O2 -o hello hello.c
```

Question 1.3 Exécuter le programme :

```
./hello
```

Exercice 2 : Découverte du langage C

Le but de cet exercice est de découvrir le langage C. Ce langage fait partie de la même famille que Java au niveau syntaxique, beaucoup de constructions sont similaires, de même que certains types.

Question 2.1 Recopier le code source suivant dans un fichier appelé `arg.c`.

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

int main(int argc, char *argv[]) {
  if (argc != 2) {
    printf("Un argument est nécessaire !\n");
    return 1;
  }

  int arg = atoi(argv[1]);
  printf("L'argument est : %d\n", arg);

  return 0;
}
```

Dans ce programme, on utilise la ligne de commande pour fournir un nombre au programme. S'il n'y a pas assez d'argument, un message d'erreur est renvoyé et le programme s'arrête. Sinon, l'argument est transformé en nombre grâce à la fonction `atoi(3)`. Puis, on affiche le nombre grâce à `printf(3)`. Cette fonction prend comme argument une chaîne de caractères entre guillemets, puis éventuellement des noms d'identifiants de variables. Lors de l'exécution, la chaîne de caractères est affichée ainsi que la valeur de chaque variable à l'endroit indiqué. Les variables doivent apparaître dans l'ordre de leur apparitions dans la chaîne de caractères. Un entier est marqué grâce à `%d`, un flottant grâce à `%f`, un caractère grâce à `%c`.

```
$ ./arg 32
L'argument est : 32
```

On utilisera ce code comme base pour les questions suivantes.

Question 2.2 Faire un programme qui affiche la suite de Collatz. L'argument sera l'élément initial de la suite. Pour rappel, la suite de Collatz est définie par un élément initial u_0 strictement positif et :

$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est divisible par 2} \\ 3u_n + 1 & \text{sinon} \end{cases}$$

On utilisera une boucle `while` et on s'arrêtera quand u_n vaudra 1.

```
$ ./collatz 46
46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1
```

Question 2.3 Faire un programme qui affiche les nombres de 1 à n où n est passé en argument. Dans cette suite, on remplace les multiple de 3 par «Fizz», les multiple de 5 par «Buzz» (et donc les multiples de 3 et 5 par «FizzBuzz»). On utilisera une boucle `for`.

```
$ ./fizzbuzz 16
1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz 16
```

Question 2.4 Faire un programme qui permet d'afficher un triangle d'une longueur qu'on précisera en argument. On utilisera une double boucle `for`. Par exemple, pour un argument de 7, le programme affichera :

```
$ ./triangle 7
#
# #
# # #
# # # #
# # # # #
# # # # # #
# # # # # # #
```

Travaux Pratiques d'Algorithmique n°2

Exercice 3 : Tableaux

Dans les questions suivantes, coder les fonctions demandées en prenant bien soin de réfléchir au nom et aux paramètres de la fonction. Pour chaque algorithme, on donnera l'opération considérée et sa complexité. On utilisera chaque fonction dans `main` sur un exemple quelconque.

Question 3.1 Écrire une fonction qui alloue un tableau d'entiers à l'aide de `calloc(3)`. Utiliser la fonction `rand(3)` pour remplir le tableau à l'aide de valeur entre 0 et 99. Dans `main`, on appellera cette fonction avec 10000 comme argument.

```
int *array_new(size_t size);
```

Question 3.2 Écrire une fonction qui renvoie l'indice de l'élément le plus grand du tableau.

```
size_t array_index_max(const int *data, size_t size);
```

Question 3.3 Écrire une fonction qui calcule la somme des éléments du tableau.

```
int array_sum(const int *data, size_t size);
```

Question 3.4 Écrire une fonction qui renvoie le nombre d'occurrences dans le tableau d'une valeur passées en paramètre.

```
size_t array_count(const int *data, size_t size, int value);
```

Question 3.5 Écrire un algorithme qui effectue un décalage du tableau d'une case vers la gauche, la première valeur étant placée à la fin du tableau.

```
void array_shift_left(int *data, size_t size);
```

Question 3.6 Écrire une fonction qui renvoie l'indice de la plus grande suite de nombres pairs dans le tableau.

```
size_t array_longest_even_seq(const int *data, size_t size);
```

Exercice 4 : Fonctions sur les chaînes de caractères

Question 4.1 Afficher la chaîne de caractère passée en argument du programme. On mettra des guillemets autour de la chaîne sur la ligne de commande pour qu'elle soit considéré comme un argument unique pour le programme :

```
./compute_string "c'est pas faux !"
```

Question 4.2 Écrire une fonction qui calcule la longueur de la chaîne de caractère. Comparer avec ce que renvoie `strlen(3)`.

Question 4.3 Écrire une fonction qui compte le nombre d'espaces dans la chaînes. Indice : `isspace(3)`.

Question 4.4 Écrire une fonction qui affiche la chaîne en enlevant les voyelles (non-accentuées) de la chaîne de caractères.

Question 4.5 Écrire une fonction qui détermine si la chaîne est bien parenthésée.

Question 4.6 Écrire une fonction qui calcule la valeur numérique d'un entier représenté en binaire par une chaîne de caractère.

Travaux Pratiques d'Algorithmique n°3

Exercice 5 : Bibliothèque de structures de données

Cet exercice est à faire sur votre temps libre tout au long du semestre. Il consiste à implémenter une bibliothèque contenant toutes les structures de données vues en cours ainsi que les algorithmes associés. Cette bibliothèque vous sera utile dans les TP (notamment les projets). Elle contient un squelette de toutes les fonctions ainsi que des tests unitaires, c'est-à-dire des tests qui vérifient que les fonctions sont conformes aux spécifications.

Cette bibliothèque utilise des entiers comme données de base. Il se peut que vous ayez à adapter les structures de données si jamais vous utilisez ces fonctions dans un projet.

Question 5.1 Télécharger l'archive `biblialgo.tar.gz`. Décompresser l'archive.

Question 5.2 Compiler les fichiers sources à l'aide de la commande `make`. Lancer le programme `run_tests` et vérifier qu'aucun test ne passe.

Question 5.3 Implémenter les algorithmes relatifs aux chaînes de caractères.

Question 5.4 Une fois les tableaux vus en cours (partie 3), implémenter les algorithmes relatifs à `struct array` (sauf les algorithmes de tris et les algorithmes de tas).

Question 5.5 Une fois les listes vues en cours (partie 4), implémenter les algorithmes relatifs à `struct list`.

Question 5.6 Une fois les tris vus en cours (partie 5 et partie 6), implémenter algorithmes de tris relatifs à `struct array` (sauf le tri par tas) et `struct list`.

Question 5.7 Une fois les arbres binaires de recherche vus en cours (partie 8), implémenter les algorithmes relatifs à `struct tree`

Question 5.8 Une fois les tas vus en cours (partie 8), implémenter les algorithmes de tas relatifs à `struct array`.

Exercice 6 : Crible d'Ératosthène

Un nombre est dit premier s'il admet exactement 2 diviseurs distincts (1 et lui-même). 1 n'est donc pas premier. On désigne sous le nom de crible d'Ératosthène une méthode de recherche des nombres premiers plus petits qu'un entier naturel n donné. La méthode est la suivante :

1. On supprime tous les multiples de 2 inférieurs à n.
2. L'entier 3 n'ayant pas été supprimé, il ne peut être multiple des entiers qui le précèdent, il est donc premier. On supprime alors tous les multiples de 3 inférieurs à n.

3. L'entier 5 n'ayant pas été supprimé, il ne peut être multiple des entiers qui le précèdent, il est donc premier. On supprime alors tous les multiples de 5 inférieurs à n.
4. Et ainsi de suite jusque n. Les valeurs n'ayant pas été supprimées sont les nombres entiers plus petits que n.

Pour programmer cette méthode, on va utiliser un tableau `is_prime` de n entiers qui contiendra des 1 et des 0. On donne à ces 1 et 0 le sens suivant : si `is_prime[i]` vaut 0 alors `i` n'est *pas premier*, et si `is_prime[i]` vaut 1 alors `i` est *premier*. Lors de la méthode, on élimine les nombres non premiers au fur et à mesure qu'on les rencontre. Donc au départ, on suppose que tous les entiers sont premiers.

Question 6.1 Récupérer la taille n sur la ligne de commande.

Question 6.2 Allouer et initialiser un tableau `is_prime[i]` avec des 1.

Question 6.3 Implémenter la méthode du crible d'Ératosthène.

Question 6.4 Afficher les nombres premiers inférieurs à n.

```
$ ./eratosthene 100
Nombres premiers inférieurs à 100 :
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Exercice 7 : Tri d'un tableau binaire

On considère un tableau dont les éléments appartiennent à l'ensemble $\{0, 1\}$. On se propose de trier ce tableau. À chaque étape du tri, le tableau est constitué de trois zones consécutives, la première ne contenant que des 0, la seconde n'étant pas triée et la dernière ne contenant que des 1.

zone de 0	zone non triée	zone de 1

On range le premier élément de la zone non triée si celle-ci n'est pas encore vide : si l'élément vaut 0, il ne bouge pas ; si l'élément vaut 1, il est échangé avec le dernier élément de la zone non triée. Dans tous les cas, la longueur de la zone non triée diminue de 1.

Question 7.1 Créer un tableau de 100000 éléments de 0 et de 1 tirés au hasard.

Question 7.2 Compter le nombre de 1 du tableau.

Question 7.3 Trier le tableau selon la méthode indiquée.

Question 7.4 Vérifier que le nombre de 1 est identique au nombre de 1 précédemment calculé.

Question 7.5 Quel est la complexité de cet algorithme ?

Travaux Pratiques d'Algorithmique n°4

Exercice 8 : Implémentation d'une pile avec une liste chaînée

Le but de cet exercice est de proposer une implémentation d'une pile grâce à une liste chaînée.

On dispose de la structure de donnée suivante :

```
struct stack_node {
  int data;
  struct stack_node *next;
};

struct stack {
  struct stack_node *first;
};
```

Question 8.1 Donner le code d'une fonction qui initialise une pile vide :

```
void stack_create(struct stack *self);
```

Question 8.2 Donner le code d'une fonction qui examine si la pile est vide :

```
bool stack_is_empty(const struct stack *self);
```

Question 8.3 Donner le code d'une fonction qui empile un élément :

```
void stack_push(struct stack *self, int data);
```

Question 8.4 Donner le code d'une fonction qui donne la valeur du premier élément d'une pile non-vide :

```
int stack_top(const struct stack *self);
```

Question 8.5 Donner le code d'une fonction qui dépile un élément :

```
void stack_pop(struct stack *self);
```

Question 8.6 Donner le code d'une fonction qui détruit une pile :

```
void stack_destroy(struct stack *self);
```

Exercice 9 : Manipulation de liste chaînée

On utilise la structure de liste suivante :

```
struct list_node {
  int data;
  struct list_node *next;
};

struct list {
  struct list_node *first;
};
```

Question 9.1 Donner le code d'une fonction qui prend en paramètre une liste et retourne une liste miroir, c'est-à-dire avec les mêmes éléments en ordre inverse. Quelle est sa complexité ?

```
void list_mirror(const struct list *self, struct list *res);
```

Question 9.2 Donner le code d'une fonction qui prend en paramètre une liste et retourne une copie de la liste. Quelle est sa complexité ?

```
void list_copy(const struct list *self, struct list *res);
```

Question 9.3 Donner le code d'une fonction qui prend en paramètre deux listes et qui renvoie la concaténation des deux listes. Quelle est sa complexité ?

```
void list_concat(const struct list *l1, const struct list *l2,
    struct list *res);
```