Dossier de développement logiciel

Interpréteur de commandes

Poursuite par équipes en patinage de vitesse

Crédit : Damien MEYER, AFP

Jules Doumèche & Gwénolé Martin

2019 - Groupe 111

Sommaire

Contexte du projet :

Ce projet a été réalisé en période A (Semestre 1). Nous avons été encadrés par Mme Caraty, M. Alles-Bianchetti et M. Poitreneau.

I - Présentation du projet

1 – Objectif du projet

Dans le cadre du projet d'IAP de la période A (2019-2020), nous devions coder un interpréteur de commandes capable de gérer des poursuites par équipes en patinage de vitesse. Cette épreuve olympique est définie de la sorte : 2 équipes de 3 patineurs partent chacune à l'opposée d'une piste circulaire, le but étant pour chaque équipe de faire un nombre de tour donné le plus rapidement possible, l'équipe faisant le meilleur temps gagne. Le temps retenu est celui du dernier patineur de l'équipe.

2 – Entrées/Sorties

L'interpréteur de commandes doit être capable de lire 8 commandes, envoyées directement via la console ou avec une redirection d'un fichier texte (.txt). Les commandes que nous avons codées sont les suivantes : **exit, definir_parcours, definir_nombre_epreuves, inscrire_equipe, afficher_equipes, enregistrer_temps, afficher_temps et afficher_temps_equipes**.

Voici un tableau récapitulatif des entrées/sorties pour chaque commande :

Nom de la commande	Entrée(s)	Sortie(s)
definir_parcours	1 entier positif	Définie le nombre de tours par épreuve
definir_nombre_epreuves	1 entier positif	Définie le nombre d'épreuves
inscrire_equipe	4 chaines de caractères : Pays + Nom de chaque patineur	Affiche le numéro de dossard de chaque patineur inscrit : « inscription dossard + num »
afficher_equipes	Pas d'entrée	Affiche les deux équipes actuellement en compétition
enregistrer_temps	2 entiers et 1 float : numéro de dossard + numéro de tour + temps	Enregistre le temps d'un patineur pour un tour donné
afficher_temps	1 entier positif : le numéro de dossard	Affiche le temps d'un patineur (pour chaque tour)
afficher_temps_equipes	1 entier positif : le numéro du tour	Affiche les temps des équipes pour un tour donné (par ordre croissant)

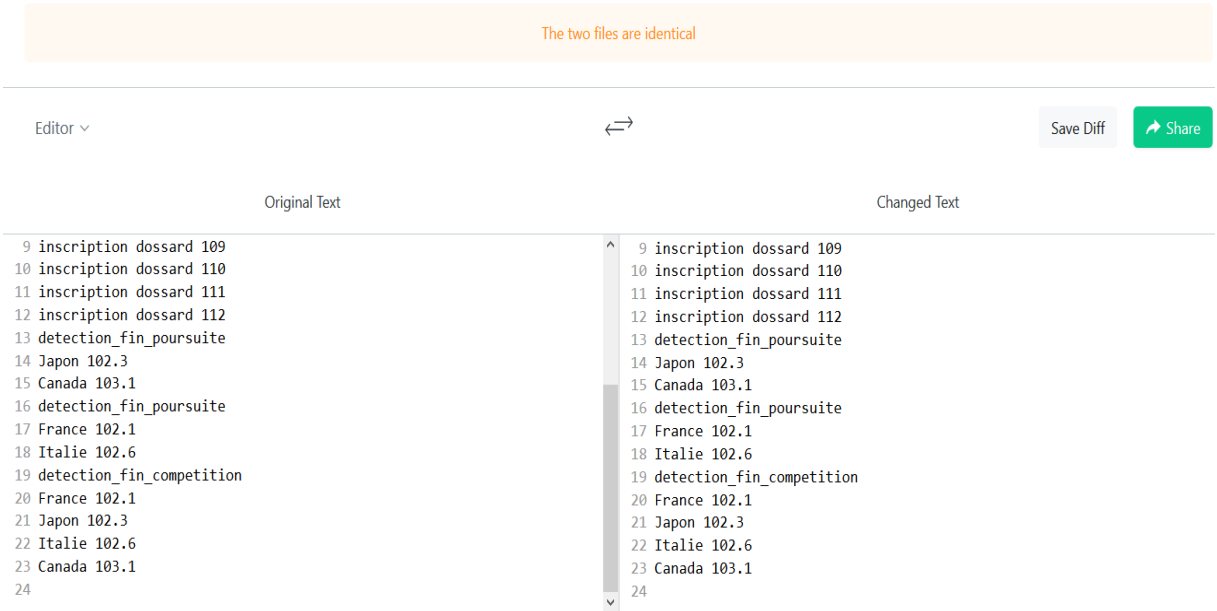
II – Bilan du développement

1 – Organisation des tests

Pour développer cet interpréteur, nous avons utilisé une méthode de développement par sprint, avec 5 niveaux de sprints différents, chaque sprint permettant d'implémenter une fonctionnalité supplémentaire dans le programme. Nous avons utilisé les jeux de données de test fournis pour tester chaque sprint, plus quelques jeux de données personnalisés, pour être sûr que les sprints ne comportaient pas de bug.

2 – Bilan de validation

Le sprint de plus haut niveau validé est le sprint 5. Le fichier *out* est identique au *out* de référence. Ci-dessous une capture d'écran sur DiffChecker pour le Jeu de Données de Test (JDT) du sprint 5 :



III – Bilan de projet

Comme nous avions commencé une semaine avant le cours de projet, nous avons refais plusieurs fois notre sprint 1. Nous avions au tout début une version sans structure, que nous avons entièrement refaite, puis optimisée pour partir sur une base « fiable » pour le reste du développement.

Pour le sprint 2, nous avons eu quelques difficultés le temps de bien comprendre comment attribuer les valeurs aux bons endroits grâce aux pointeurs, comme nous avions une idée claire de ce que nous devions faire, le reste n'a pas posé de problème.

Le sprint 3 ne nous a pas posé de problème, nous l'avons codé rapidement.

Nous avons codé les sprints 4 et 5 en même temps, il était plus simple pour nous d'implémenter les deux fonctionnalités en même temps, une fois que **detection_fin_poursuite** fonctionnait, il était simple d'implémenter **detection_fin_competition**.

Le développement n'a pas toujours été simple notamment au niveau de l'algorithme de tri pour le sprint 5, qui a nécessité de nombreux tests mais en prenant le temps de poser les idées (en écrivant sur papier le déroulement de la fonction par exemple), nous avons pu surmonter toutes les difficultés.

Le code présenté, bien qu'il ai été optimisé, peut l'être davantage, mais compte tenu des délais nous avons davantage insisté sur la correction des erreurs pour fournir un code fonctionnel à temps.

IV – Annexes

1 – Code source du sprint 5

```
/*          Projet 1 IAP 2019          */
/*                  Sprint 5           */
/*Jules DOUMECHE et Gwénolé MARTIN - Groupe 111*/
/*    Fin de développement le 24/10/2019     */

#pragma warning(disable:4996)
#pragma warning(disable:6031)
#pragma warning(disable:6054)
#pragma warning(disable:6262)
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

/* INITIALISATION CONSTANTES GLOBALES ET STRUCTURES*/

enum MAX_P { MAX_EQUIPES = 32, MAX_EQUIPES_PARCOURS = 2, MAX_EQUIPIERS = 3 }; //
enum MAX_MOT { MAX_MOT = 30 }; //
enum MAX_TOURS { MAX_TOURS = 10 }; //
enum MAX_EPREUVES { MAX_EPREUVES = 16 }; // Constantes pour les maximums (définis dans le CDC)
enum INIT { PREMIER_DOSSARD = 101 }; // Constante pour le premier numéro de dossard
```

```
typedef struct {
char nom[MAX_MOT + 1]; //Nom du patineur
unsigned int dossard; //Numéro de dossard du patineur
double temps[MAX_TOURS] ; //Tour du patineur
unsigned int tour ; //Temps du patineur (pour chaque tour)
}Patineur; //^^ Structure patineur ^^//

typedef struct {
char pays[MAX_MOT + 1]; //Nom du pays de l'équipe
Patineur dataPatineurs[MAX_EQUIPIERS]; //Tableau de Patineur de la taille de MAX_EQUIPIERS
unsigned int dernierPatineur; //^^ Structure équipe ^^//
}Equipe;

typedef struct {
Equipe equipe[MAX_EQUIPES_PARCOURS] ; //Tableau d'équipes
unsigned int fini; //Booléen permettant de savoir si la course est finie ou non
unsigned int gagnante;  //Permet de stocker le numéro (0 ou 1) de l'équipe gagnante pour la course
}Course; //^^ Structure course ^^//

typedef struct {
unsigned int nbInscrits;  //Variable du nombre de patineurs inscrits au total
Course course[MAX_EPREUVES]; //Tableau de course
unsigned int nbParcours; //Stocke le nombre de tours par course entré
unsigned int nbEpreuves; //Stocke le nombre d'épreuves entré
unsigned int derniere_course; //Booléen permettant de savoir si c'est la dernière course ou non
}Inscrits; //^^ Structure Inscrits ^^//
```

```
typedef struct {
double temps_tri; //Stockage du temps des équipes pour le tri
int equipe_tri; //Stockage des équipes pour le tri
}Tri; //^^ Structure Tri ^^//


/*PROTOTYPES FONCTIONS*/

void definir_parcours(Inscrits* ins);
void definir_nombre_epreuves(Inscrits* ins);
void inscrire_equipe(Inscrits* ins);
void afficher_equipes(Inscrits* ins);
void enregistrer_temps(Inscrits* ins);
void afficher_temps(Inscrits* ins);
void afficher_temps_equipes(Inscrits* ins);
void detection_fin_poursuite(Inscrits* ins);
void detection_fin_competition(Inscrits* ins);

int crs_en_cour(Inscrits* ins);
int nb_eq_ins(Inscrits* ins);
```

```
/* MAIN */
```

```c
int main() {

	/*INITIALISATION VARIABLES ET STRUCTURES*/

	char mot[MAX_MOT + 1];
	Inscrits ins = { .nbInscrits = 0,.nbParcours = 2,.nbEpreuves =
16,.derniere_course = 0 };

	/*PROGRAMME*/

	while (1) {
		scanf("%s", mot);
		if (strcmp(mot, "definir_parcours") == 0) {
			definir_parcours(&ins);
		}
		else if (strcmp(mot, "definir_nombre_epreuves") == 0) {
			definir_nombre_epreuves(&ins);
		}
		else if (strcmp(mot, "inscrire_equipe") == 0) {
			inscrire_equipe(&ins);
		}
		else if (strcmp(mot, "afficher_equipes") == 0) {
			afficher_equipes(&ins);
		}
		else if (strcmp(mot, "enregistrer_temps") == 0) {
			enregistrer_temps(&ins);
		}
		else if (strcmp(mot, "afficher_temps") == 0) {
			afficher_temps(&ins);
		}
		else if (strcmp(mot, "afficher_temps_equipes") == 0) {
			afficher_temps_equipes(&ins);
		}
		else if (strcmp(mot, "exit") == 0) {
			exit(0);
		}
	}
}

/* FONCTIONS */


/* Enregistre le nombre de "parcours"
 * [in] Inscrits ins, gère la structure de la compétition
 * [out] pas de out, changement de la valeur de "parcours" dans la structure à l'aide
d'un pointeur
 */
void definir_parcours(Inscrits* ins) {
	scanf("%d", &ins->nbParcours);
}

/* Enregistre le nombre d'"épreuves"
 * [in] Inscrits ins, gère la structure de la compétition
 * [out] pas de out, changement de la valeur de "epreuves" dans la structure à l'aide
d'un pointeur
 */
void definir_nombre_epreuves(Inscrits* ins) {
	scanf("%d", &ins->nbEpreuves);
}
```

```
/* Inscrit les équipes de Patineur
 * [in] Inscrits ins, gère la structure de la compétition
 * [out] pas de out, affiche Inscription dossard+numéro du dossard pour chaque
Patineur inscrit
 */
void inscrire_equipe(Inscrits* ins) {
	int no_crs = ins->derniere_course;
	int no_eq = (nb_eq_ins(ins) - (ins->derniere_course * 2));
	static int dernier_dossard = PREMIER_DOSSARD - 1;
	scanf("%s", ins->course[no_crs].equipe[no_eq].pays);

	for (int i = 0; i < MAX_EQUIPIERS; ++i) {
		scanf("%s", ins->course[no_crs].equipe[no_eq].dataPatineurs[i].nom);
		dernier_dossard++;
		ins->course[no_crs].equipe[no_eq].dataPatineurs[i].dossard =
dernier_dossard;
		printf("inscription dossard %d\n",
ins->course[no_crs].equipe[no_eq].dataPatineurs[i].dossard);
		ins->nbInscrits++;
	}

	if ((no_eq % 2) == 1 && (ins->derniere_course + 1) != ins->nbEpreuves) {
		ins->derniere_course++;
	}
}

/* Affiche les équipes de Patineur
 * [in] Inscrits ins, gère la structure de la compétition
 * [out] pas de out, affiche le pays de chaque équipe puis le nom et le numéro de
dossard de chaque patineur
 */
void afficher_equipes(Inscrits* ins) {
	int no_crs = crs_en_cour(ins);

	for (int i = 0; i < MAX_EQUIPES_PARCOURS; ++i) {
		printf("%s ", ins->course[no_crs].equipe[i].pays);
		for (int j = 0; j < MAX_EQUIPIERS; ++j) {
			printf("%s ", ins->course[no_crs].equipe[i].dataPatineurs[j].nom);
			printf("%d", ins-
>course[no_crs].equipe[i].dataPatineurs[j].dossard);
			if (j == (MAX_EQUIPIERS - 1)) {
				continue;
			}
			printf(" ");
		}
		printf("\n");
	}
}

/* Enregistre un temps pour un dossard et un tour donné
 * [in] Inscrits ins, gère la structure de la compétition
 * [out] pas de out, enregistre le temps en fonction du dossard et du tour
 */
void enregistrer_temps(Inscrits* ins) {

	int dossard;
	scanf("%d", &dossard);

	int numero_joueur = ((dossard - 101) % 3);
	int numero_equipe = ((dossard - 101) / 3) % 2;
	int no_crs = ((dossard - 101) / 3) / 2;
```

```
	//enregistrement du numéro du tour
	scanf("%d",
&ins->course[no_crs].equipe[numero_equipe].dataPatineurs[numero_joueur].tour);

	//enregistrement du temps pour le tour
	scanf("%lf",
&ins->course[no_crs].equipe[numero_equipe].dataPatineurs[numero_joueur].temps
[(ins->course[no_crs].equipe[numero_equipe].dataPatineurs[numero_joueur].tour) - 1]);

	detection_fin_poursuite(ins);
	detection_fin_competition(ins);
}

/* Affiche le temps d'un Patineur
 * [in] Inscrits ins, gère la structure de la compétition
 * [out] pas de out, affiche le pays et le temps ou les temps (pour chaque tour) d'un
patineur en fonction de son numéro de dossard
 */
void afficher_temps(Inscrits* ins) {
	int dossard;
	scanf("%d", &dossard);

	int numero_joueur = ((dossard - 101) % 3);
	int no_crs = ((dossard - 101) / 3) / 2;
	int numero_equipe = ((dossard - 101) / 3) % 2;

	for (unsigned int i = 0; i < (ins-
>course[no_crs].equipe[numero_equipe].dataPatineurs[numero_joueur].tour); ++i) {
		printf("%s ", ins->course[no_crs].equipe[numero_equipe].pays);
		printf("%d ", (i + 1));
		printf("%s ", ins-
>course[no_crs].equipe[numero_equipe].dataPatineurs[numero_joueur].nom);
		printf("%.1f\n", ins-
>course[no_crs].equipe[numero_equipe].dataPatineurs[numero_joueur].temps[i]);
	}
}

/* Affiche le temps d'une équipe de Patineurs
 * [in] Inscrits ins, gère la structure de la compétition
 * [out] pas de out, affiche le pays et le temps du dernier patineur de chaque équipe
(temps le plus grand)
 */
void afficher_temps_equipes(Inscrits* ins) {
	int no_crs = crs_en_cour(ins);
	int tour;
	scanf("%d", &tour);
	--tour; // décrémente tour pour la position dans le tableau temps

	for (int i = 0; i < MAX_EQUIPES_PARCOURS; ++i) {
		int enregistrement = 1;
		for (int j = 0; j < MAX_EQUIPIERS; ++j) {
			if (ins->course->equipe[i].dataPatineurs[j].temps[tour] <= 0.001)
{
				enregistrement = 0;
				break;
			}
		}
		if (enregistrement) {
			printf("%s ", ins->course[no_crs].equipe[i].pays);
			double dernier_temps = ins-
>course[no_crs].equipe[i].dataPatineurs[0].temps[tour];
			for (int j = 1; j < MAX_EQUIPIERS; ++j) {
```

```
                        if
(ins->course[no_crs].equipe[i].dataPatineurs[j].temps[tour] > dernier_temps) {
                                dernier_temps =
ins->course[no_crs].equipe[i].dataPatineurs[j].temps[tour];
                        }
                }
                printf("%.1f", dernier_temps);
        }
        else {
                printf("indisponible\n");
        }
        printf("\n");
    }
}

/* Détecte la fin d'une poursuite (épreuve)
 * [in] Inscrits ins, gère la structure de la compétition
 * [out] pas de out, imprime la commande de détection de fin de poursuite et les temps
des équipes
 */
void detection_fin_poursuite(Inscrits* ins) {
    int num_crs = crs_en_cour(ins);
    int fini = 1;
    for (int i = 0; ((i < MAX_EQUIPES_PARCOURS) && fini); ++i) {
        for (int j = 0; j < MAX_EQUIPIERS; ++j) {
            if ((ins->course[num_crs].equipe[i].dataPatineurs[j].temps[(
ins->nbParcours) - 1]) < 0.001) {
                        fini = 0;
                        break;
                }
                else {
                        continue;
                }
        }
    }
    if (fini) {
        printf("detection_fin_poursuite\n");
        ins->course[num_crs].fini = 1;
        int tour = ins->nbParcours - 1;

        double dernier_temps[MAX_EQUIPES_PARCOURS];
        for (int i = 0; i < MAX_EQUIPES_PARCOURS; ++i) {
            dernier_temps[i] = ins-
>course[num_crs].equipe[i].dataPatineurs[0].temps[tour];
            ins->course[num_crs].equipe[i].dernierPatineur = 0;
            for (int j = 1; j < MAX_EQUIPIERS; ++j) {
                        if
(ins->course[num_crs].equipe[i].dataPatineurs[j].temps[tour] > dernier_temps[i]) {
                                dernier_temps[i] =
ins->course[num_crs].equipe[i].dataPatineurs[j].temps[tour];
                                ins->course[num_crs].equipe[i].dernierPatineur = j;
                        }
                }
        }
        if (dernier_temps[1] > dernier_temps[0]) {
            ins->course[num_crs].gagnante = 0;
            printf("%s %.1f\n", ins->course[num_crs].equipe[0].pays,
ins->course[num_crs].equipe[0].dataPatineurs
[ins->course[num_crs].equipe[0].dernierPatineur].temps[tour]);
            printf("%s %.1f\n", ins->course[num_crs].equipe[1].pays,
ins->course[num_crs].equipe[1].dataPatineurs
[ins->course[num_crs].equipe[1].dernierPatineur].temps[tour]);
```

```
			}
			else {
				ins->course[num_crs].gagnante = 1;
				printf("%s %.1f\n", ins->course[num_crs].equipe[1].pays,
ins->course[num_crs].equipe[1].dataPatineurs
[ins->course[num_crs].equipe[1].dernierPatineur].temps[tour]);
				printf("%s %.1f\n", ins->course[num_crs].equipe[0].pays,
ins->course[num_crs].equipe[0].dataPatineurs
[ins->course[num_crs].equipe[0].dernierPatineur].temps[tour]);
			}
		}
}

/* Détecte la fin d'une compétition
 * [in] Inscrits ins, gère la structure de la compétition
 * [out] pas de out, imprime la commande de détection de fin de compétition et les
temps des équipes
 */
void detection_fin_competition(Inscrits* ins) {
		if (ins->course[ins->nbEpreuves - 1].fini == 1) {
			printf("detection_fin_competition\n");
			Tri tri[MAX_EQUIPES];
			int c = 0;
			for (int i = 0; i < ((nb_eq_ins(ins) / 2)); i++) {
				for (int j = 0; j < MAX_EQUIPES_PARCOURS; j++) {
					tri[c].temps_tri =
ins->course[i].equipe[j].dataPatineurs[ins->course[i].equipe[j].dernierPatineur].temps
[ins->nbParcours - 1];
					tri[c].equipe_tri = c;
					c++;
				}
			}

			for (int i = 0; i < (nb_eq_ins(ins)); i++) {
				for (int j = 0; j < (nb_eq_ins(ins)); j++) {
					if (tri[i].temps_tri < tri[j].temps_tri) {
						double temp_1 = tri[i].temps_tri;
						int temp_2 = tri[i].equipe_tri;
						tri[i].temps_tri = tri[j].temps_tri;
						tri[i].equipe_tri = tri[j].equipe_tri;
						tri[j].temps_tri = temp_1;
						tri[j].equipe_tri = temp_2;
					}
				}
			}

			for (int i = 0; i < (nb_eq_ins(ins)); i++) {
				printf("%s %.1f\n", ins->course[(tri[i].equipe_tri /
2)].equipe[(tri[i].equipe_tri) % 2].pays, tri[i].temps_tri);
			}
			exit(0);
		}
}

/*
 * [in] Inscrits ins, gère la structure de la compétition
 * [out]
 */
int crs_en_cour(Inscrits* ins) {
		for (unsigned int i = 0; i <= ins->nbEpreuves; ++i) {
			if (ins->course[i].fini == 1) {
				continue;
```

```
			}
			else {
				return i;
				break;
			}
		}
		return (ins->nbEpreuves - 1);
}

/* Retourne le nombre d'équipe
 * [in] Inscrits ins (nombre de patineurs inscrits)
 * [out] le nombre d'équipes (nombre de patineurs / nombre de patineurs par équipe)
 */
int nb_eq_ins(Inscrits* ins) {
		return ((ins->nbInscrits) / MAX_EQUIPIERS);
}
```

2 – Trace d'exécution du sprint 5

IN (avec le JDT fourni pour le sprint 5) :

definir_parcours 2

definir_nombre_epreuves 2

inscrire_equipe Canada Blondin Weidemann Morrison

inscrire_equipe Japon Takagi Sato Takagu

inscrire_equipe France Pierron Huot Monvoisin

inscrire_equipe Italie Lollobrigida Mascitto Valcepina

enregistrer_temps 101 1 53.1

enregistrer_temps 102 1 53.2

enregistrer_temps 104 1 53.3

enregistrer_temps 105 1 53.7

enregistrer_temps 106 1 53.9

enregistrer_temps 103 1 54.1

enregistrer_temps 105 2 100.6

enregistrer_temps 106 2 101.7

enregistrer_temps 104 2 102.3

enregistrer_temps 101 2 102.5

enregistrer_temps 103 2 102.8

enregistrer_temps 102 2 103.1

enregistrer_temps 111 1 50.9

enregistrer_temps 108 1 52.1

enregistrer_temps 112 1 53.2

enregistrer_temps 107 1 53.5

enregistrer_temps 109 1 53.8

enregistrer_temps 110 1 54.1

enregistrer_temps 110 2 99.1

enregistrer_temps 109 2 100.3

enregistrer_temps 107 2 101.5

enregistrer_temps 112 2 101.8

enregistrer_temps 108 2 102.1

enregistrer_temps 111 2 102.6

OUT :

inscription dossard 101

inscription dossard 102

inscription dossard 103

inscription dossard 104

inscription dossard 105

inscription dossard 106

inscription dossard 107

inscription dossard 108

inscription dossard 109

inscription dossard 110

inscription dossard 111

inscription dossard 112

detection_fin_poursuite

Japon 102.3

Canada 103.1

detection_fin_poursuite

France 102.1

Italie 102.6

detection_fin_competition

France 102.1

Japon 102.3

Italie 102.6

Canada 103.1