**Dossier de développement logiciel**

Interpréteur de commandes

---

**Une image contenant patinage, personne, skiant, neige

Description générée automatiquement**Poursuite par équipes en patinage de vitesse

*Crédit : Damien MEYER, AFP*

Jules Doumèche & Gwénolé Martin

2019 - Groupe 111

Sommaire

[I - Présentation du projet 3](#_Toc23336142)

[1 – Introduction 3](#_Toc23336143)

[2 – Entrées/Sorties 3](#_Toc23336144)

[II – Bilan de validation 3](#_Toc23336145)

[III – Bilan de projet 3](#_Toc23336146)

[IV – Annexes 3](#_Toc23336147)

[1 – Code source du sprint 5 3](#_Toc23336148)

[2 – Trace d’exécution du sprint 5 3](#_Toc23336149)

*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 :

Une image contenant capture d’écran

Description générée automatiquement

# 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 \*/

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

\* [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

\* [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

\* [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 (en constante pour l'affichage)

\* [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

\* [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

\* [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

\* [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

\* [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

\* [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]

\* [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