Titre du TIPE sur Jeux et Sports

Code du TIPE

Ceci est un modèle pour générer un PDF qui présente le code d'un TIPE.
Le fichier principal `rendu_code.tex` est à compiler avec l'option `-shell-escape`.

Conseils :

- Commencez par faire du ménage dans votre code, supprimer les fichiers obsolètes, les brouillons... Si ce n'est pas fait, c'est peut-être aussi l'occasion de séparer son code en plusieurs fichiers...

- Placez le dossier avec ce code latex à proximité de votre code dossier de code (pas dedans) et indiquez les fichiers de code à inclure grâce à un chemin relatif. Typiquement si votre dossier `TIPE` contient les dossiers `Code` et `rendu_code`, vous écrirez `../code/sous_dossier/fichier.c`. Il est déconseillé de copier tout son code dans un dossier spécial pour générer le rendu de code, cela n'est pas très robuste aux modifications et ajouts de code ultérieurs.

- Si vous importez le code du fichier en entier, pensez à supprimer les lignes vides en fin de fichier, les portions de code commenté devenu inutile...

- Si au contraire vous utilisez l'import par blocs, il est conseillé de sauter des lignes entre les différentes fonctions de sorte que chaque bloc commence à une ligne multiple de 10 (par exemple). Cela permet, dans le cas où vous modifiez a posteriori une fonction, en ajoutant une ligne de commentaire par exemple, d'avoir à modifier seulement les numéros de lignes du bloc correspondant, et pas ceux de tous les blocs suivants.

- Pensez à ajuster le modèle pour une version finale :
  · renseignez nom, prénom et numéro de candidat dans le fichier `mise_en_forme_MPI.tex`, pour la première page l.6 ET pour le pied de page l.32 ;
  · adaptez le titre et la date ;
  · supprimez cette section de conseils en commentant la l.20 ;
  · à la fin il ne doit plus y avoir de rouge dans le PDF.

- Compilez plusieurs fois si nécessaire pour ajuster le nombre total de pages en pied de page.

- Si vous ne réduisez pas la police, vous pouvez imprimer ce PDF livret (A4-> A5, en 2 pages par feuille...), ça reste lisible.

1 Code sur la première partie

1.1 Code du fichier

Ici, on insère le fichier entier, tel quel, pas forcément adapté.

```c
#include <stdio.h>
#include <stdbool.h>
#include <assert.h>

void type_simple_triangle (int a, int b, int c){
//hyp : a>=0 && b>=0 && c>=0
  if (a==b){
    if(b==c)
      printf("équilatéral");
    else
      printf("isocèle");
  }
  else{
    if((b==c) || (a==c))
      printf("isocèle");
    else
      printf("scalène");
  }
  printf("\n");
}




bool est_triangle(int a, int b, int c){
  return (a<=b+c) && (b<=a+c) && (c<=a+b) ;
}









bool est_plat(int a, int b, int c){
  return (a==b+c) || (b==a+c) || (c==a+b) ;
}









bool est_rectangle(int a, int b, int c){
  return (a*a + b*b == c*c) || (b*b + c*c == a*a) || (a*a + c*c == b*b) ;
}
```

```c
void affiche_type_triangle(int a,int b, int c){
  if (!est_triangle(a,b,c))
    printf("Ces longueurs ne sont les 3 côtés d'un même triangle\n");
  else{
    if (a==b){
      if(b==c){
         if (a==0)// alors a=b=c=0
           printf("équilatéral plat = réduit à un point");
         else
          printf("équilatéral");
        }
      else if(2*b*b == c*c)  // dans ce cas a²+b²=c²
        printf("isocèle rectangle");
      else if (a+b == c)
        printf("isocèle plat");
      else
        printf("isocèle");
    }
    else{ // ici on sait que a!=b
      if (b == c) {
        if(2*b*b == a*a)  // dans ce cas c²+b²=a²
          printf("isocèle rectangle");
        else if (b+c == a)
          printf("isocèle plat");
        else
          printf("isocèle");
      }
      else if (a == c){
        // ici on sait que a!=b et b!=c et a=c
        if(2*a*a == b*b)  // dans ce cas a²+c²=b²
          printf("isocèle rectangle");
        else if (a+c == b)
          printf("isocèle plat");
        else
          printf("isocèle");
      }
      else{ // ici on sait que a!=b et b!c et a!=c
        if (est_plat(a,b,c))
          printf("plat");
        else if (est_rectangle(a,b,c))
          printf("rectangle");
        else
          printf("scalène");
      }
    }
```

```
100     printf("\n");
101   }
102 }
103
104
105 int main(){
106   // jeu de test de type_triangle
107   type_simple_triangle (1,1,1); //equi
108   type_simple_triangle (1,1,2); //iso
109   type_simple_triangle (1,2,1); //iso
110   type_simple_triangle (2,1,1); //iso
111   type_simple_triangle (1,2,3); //scalene
112
113
114
115   // jeu de test de est_triangle
116   assert(est_triangle (1,2,3));
117   assert(est_triangle (3,4,5));
118   assert(est_triangle (1,1,1));
119   assert(!est_triangle (3,4,15));
120   assert(!est_triangle (31,4,15));
121
122
123
124
125   // jeu de test de est_plat
126   assert(est_plat (1,2,3));
127   assert(!est_plat (3,4,5));
128
129
130   // jeu de test de est_rectangle
131   assert(!est_rectangle (1,2,3));
132   assert(est_rectangle (3,4,5));
133
134
135   // jeu de test de affiche_type_triangle
136   affiche_type_triangle (1,1,1); //equi
137   affiche_type_triangle (0,0,0); // equi plat = pt
138   affiche_type_triangle (1,1,2); //iso plat
139   affiche_type_triangle (1,2,1); //iso plat
140   affiche_type_triangle (2,1,1); //iso plat
141   affiche_type_triangle (3,3,5); //iso
142   affiche_type_triangle (1,2,3); //plat
143   affiche_type_triangle (3,4,5); //rectangle
144   affiche_type_triangle (3,4,6); //scalène
145   //isocèle rectangle n'arrive pas avec des longueurs entières
146
147   return 0;
148 }
```

1.2 Code du fichier `nom_fichier.c`

Ici, on insère le fichier par bloc, de telle ligne à telle ligne. On utilise même `\textbackslash newpage` pour éviter que le bloc du `main` soit à cheval sur deux pages.

```c
#include <stdio.h>
#include <stdbool.h>
#include <assert.h>

void type_simple_triangle (int a, int b, int c){
//hyp : a>=0 && b>=0 && c>=0
  if (a==b){
    if(b==c)
      printf("équilatéral");
    else
      printf("isocèle");
  }
  else{
    if((b==c) || (a==c))
      printf("isocèle");
    else
      printf("scalène");
  }
  printf("\n");
}

bool est_triangle(int a, int b, int c){
  return (a<=b+c) && (b<=a+c) && (c<=a+b) ;
}

bool est_plat(int a, int b, int c){
  return (a==b+c) || (b==a+c) || (c==a+b) ;
}

void affiche_type_triangle(int a,int b, int c){
  if (!est_triangle(a,b,c))
    printf("Ces longueurs ne sont les 3 côtés d'un même triangle\n");
  else{
    if (a==b){
      if(b==c){
         if (a==0)// alors a=b=c=0
           printf("équilatéral plat = réduit à un point");
         else
          printf("équilatéral");
        }
      else if(2*b*b == c*c)  // dans ce cas a²+b²=c²
        printf("isocèle rectangle");
      else if (a+b == c)
        printf("isocèle plat");
      else
        printf("isocèle");
    }
    else{ // ici on sait que a!=b
```

```
74        if (b == c) {
75          if(2*b*b == a*a)  // dans ce cas c²+b²=a²
76            printf("isocèle rectangle");
77          else if (b+c == a)
78            printf("isocèle plat");
79          else
80            printf("isocèle");
81        }
82        else if (a == c){
83          // ici on sait que a!=b et b!=c et a=c
84          if(2*a*a == b*b)  // dans ce cas a²+c²=b²
85            printf("isocèle rectangle");
86          else if (a+c == b)
87            printf("isocèle plat");
88          else
89            printf("isocèle");
90        }
91        else{ // ici on sait que a!=b et b!c et a!=c
92          if (est_plat(a,b,c))
93            printf("plat");
94          else if (est_rectangle(a,b,c))
95            printf("rectangle");
96          else
97            printf("scalène");
98        }
99      }
100     printf("\n");
101   }
102 }
```

```c
int main(){
  // jeu de test de type_triangle
  type_simple_triangle (1,1,1); //equi
  type_simple_triangle (1,1,2); //iso
  type_simple_triangle (1,2,1); //iso
  type_simple_triangle (2,1,1); //iso
  type_simple_triangle (1,2,3); //scalene



  // jeu de test de est_triangle
  assert(est_triangle (1,2,3));
  assert(est_triangle (3,4,5));
  assert(est_triangle (1,1,1));
  assert(!est_triangle (3,4,15));
  assert(!est_triangle (31,4,15));




  // jeu de test de est_plat
  assert(est_plat (1,2,3));
  assert(!est_plat (3,4,5));


  // jeu de test de est_rectangle
  assert(!est_rectangle (1,2,3));
  assert(est_rectangle (3,4,5));


  // jeu de test de affiche_type_triangle
  affiche_type_triangle (1,1,1); //equi
  affiche_type_triangle (0,0,0); // equi plat = pt
  affiche_type_triangle (1,1,2); //iso plat
  affiche_type_triangle (1,2,1); //iso plat
  affiche_type_triangle (2,1,1); //iso plat
  affiche_type_triangle (3,3,5); //iso
  affiche_type_triangle (1,2,3); //plat
  affiche_type_triangle (3,4,5); //rectangle
  affiche_type_triangle (3,4,6); //scalène
  //isocèle rectangle n'arrive pas avec des longueurs entières

  return 0;
}
```

2 Code de la deuxième partie

2.1 Code du fichier

```
let oppose (l:int list) : int list =
  let rec aux (ll:int list) (lres:int list) : int list =
    match ll with
    | [] -> lres
    | a::q -> aux q ((-a)::lres)
  in List.rev (aux l [])

let test_oppose : unit =
  assert(oppose [] = []);
  assert(oppose [-2; 3; 4; -5] = [2; -3; -4; 5]);
  assert(oppose [10; -10; 10] = [-10; 10; -10])


let intervertit (l: ('a*'b) list) : ('b*'a) list =
  let rec aux (ll:('a*'b) list) (lres:('b*'a) list) : ('b*'a) list  =
    match ll with
    | [] -> lres
    | (x,y)::q -> aux q ((y,x)::lres)
  in List.rev (aux l [])

let test_intervertit : unit =
  assert (intervertit [] = []);
  assert (intervertit [(1,2); (3,4)] = [(2,1); (4,3)]);
  assert (intervertit [('a',1); ('b',3); ('z',18)] = [(1,'a'); (3,'b');
  ↪  (18,'z')] )


let demultiplie (l: ('a*int) list) : 'a list =
  let rec aux (ll:('a*int) list) (lres:'a list) : 'a list  =
    match ll with
    | [] -> lres
    | (elem,k)::q -> if k<=0 then aux q lres else aux ((elem,k-1)::q)
    ↪  (elem::lres)
  in List.rev (aux l [])


let test_demultiplie : unit =
  assert (demultiplie [] = []);
  assert (demultiplie [('a',0)] = []);
  assert (demultiplie [(true,3)] = [true; true; true]);
  assert (demultiplie [('a',1); ('c',3); ('a',2)] = ['a'; 'c'; 'c'; 'c'; 'a';
  ↪  'a'])


let compresse (l:'a list) : ('a*int) list =
  (*hyp : l <> [] *)
  let rec aux (ll:'a list) (lres:('a*int) list) (elem: 'a) (nb:int) : ('a*int)
  ↪  list  =
```

```
    match ll with
    | [] -> (elem,nb)::lres
    | a::q -> if a=elem then aux q lres elem (nb+1) else aux q
    ↪  ((elem,nb)::lres) a 1
  in match l with
  | [] -> failwith "on ne compresse pas la liste vide"
  | e1::l1 -> List.rev (aux l1 [] e1 1)

let test_compresse : unit =
  assert (compresse [true; true; true] = [(true,3)] );
  assert (compresse [1; 1; 2; 2; 2; 3; 3] = [(1,2); (2,3); (3,2)] );
  assert (compresse ['a'; 'c'; 'c'; 'c'; 'a'; 'a'] = [('a',1); ('c',3);
  ↪  ('a',2)] )



let moyenne (l:int list) : float =
  (*hyp : l <> [] *)
  let rec aux (ll:int list) (somme:int) (cpt:int) : int*int  =
    match ll with
    | [] -> (somme, cpt)
    | a::q -> aux q (somme+a) (cpt+1)
  in let s,c = aux l 0 0
  in float_of_int(s)/.float_of_int(c)


let test_moyenne : unit =
  assert (moyenne[1; 1; 1; 1; 1] = 1.);
  assert (moyenne[1; 2; 3] = 2.);
  assert (moyenne[-10; 10; 5] = (5./.3.))

let moyennes_listes (l: int list list) : float list =
  (*hyp : les éléments de l sont non vides *)
  let rec aux (ll: int list list) (lres:float list) : float list  =
    match ll with
    | [] -> lres
    | a::q -> aux q ((moyenne a)::lres)
  in List.rev (aux l [])

let test_moyennes_listes : unit =
  assert (moyennes_listes[] = []);
  assert (moyennes_listes [[1; 1; 1; 1; 1]; [2;3]] = [1.; 2.5]);
  assert (moyennes_listes [[1; 2; 3]; [3; 2; 1]; [2; 1; 3]] = [2.; 2.; 2.]);
  assert (moyennes_listes[[-10; 10; 5]] = [5./.3.])


let moyenne_bis (l:int list) : float option =
  match l with
  | [] -> None
  | _ ->
    let rec aux (ll:int list) (somme:int) (cpt:int) : int*int  =
      match ll with
```

```
95        | [] -> (somme, cpt)
96        | a::q -> aux q (somme+a) (cpt+1)
97      in let s,c = aux l 0 0
98      in Some (float_of_int(s)/.float_of_int(c))
99
100
101 let moyennes_listes_bis (l: int list list) : float list =
102   (*hyp : les éléments de l sont non vides *)
103   let rec aux (ll: int list list) (lres:float list) : float list  =
104     match ll with
105     | [] -> lres
106     | a::q -> let m = (moyenne_bis a) in
107       match m with
108       |  None -> aux q lres
109       | Some moy -> aux q (moy::lres)
110   in List.rev (aux l [])
111
112
113 let test_moyennes_listes_bis : unit =
114   assert (moyennes_listes_bis[] = []);
115   assert (moyennes_listes_bis [[1; 1; 1; 1; 1]; [];  [2;3]] = [1.; 2.5]);
116   assert (moyennes_listes_bis [[1; 2; 3]; [3; 2; 1]; [2; 1; 3]; [];[]] = [2.;
      ↪  2.; 2.]);
117   assert (moyennes_listes_bis[[-10; 10; 5]; [0]; []] = [5./.3.; 0.])
118
119 let moyennes_listes_ter (l: int list list) : float list =
120   List.filter_map moyenne_bis l
121
122 let test_moyennes_listes_ter : unit =
123   assert (moyennes_listes_ter[] = []);
124   assert (moyennes_listes_ter [[1; 1; 1; 1; 1]; [];  [2;3]] = [1.; 2.5]);
125   assert (moyennes_listes_ter [[1; 2; 3]; [3; 2; 1]; [2; 1; 3]; [];[]] = [2.;
      ↪  2.; 2.]);
126   assert (moyennes_listes_ter[[-10; 10; 5]; [0]; []] = [5./.3.; 0.])
```