ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours d'Information, Calcul et Communication, section PH
Chappelier J.-C.

Information, Calcul et Communication (SPH) : Correction de l'Examen I

1er novembre 2024

SUJET 1

INSTRUCTIONS (à lire attentivement)

IMPORTANT ! Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre examen annulé dans le cas contraire.

1. Vous disposez de deux heures quarante-cinq minutes pour faire cet examen (13h15 – 16h00).

2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur.

   N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).

3. Vous avez droit à toute documentation papier.

   En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.

4. Répondez aux questions directement sur la donnée, **MAIS** ne mélangez pas les réponses de différentes questions !

   Ne joignez aucune feuilles supplémentaires ; **seul ce document sera corrigé**.

   Si nécessaire, il y a une page de réponse supplémentaire en fin de copie (page 16).

5. Lisez attentivement et *complètement* les questions de façon à ne faire que ce qui vous est demandé. Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un(e) des assistant(e)s.

6. L'examen comporte 6 exercices indépendants sur 16 pages, qui peuvent être traités dans n'importe quel ordre, mais qui ne rapportent pas la même chose (les points sont indiqués, le total est de 147 points).

   Tous les exercices comptent pour la note finale.

Question 1 – Questions diverses [29 points]

① **[4 points]** On souhaite stocker en binaire le score d'un participant à un concours composé de n épreuves successives. Chaque épreuve est évaluée par un score s_i compris entre 0 et $m-1$. Le score final S est calculé comme le produit des scores des épreuves : $S = s_1 \times s_2 \times \cdots \times s_n$.
En fonction de n et m, de combien de bits a-t-on besoin pour représenter ce score final S ?
Justifiez *brièvement* votre réponse.

Réponse puis justification : Toutes les valeurs entre 0 et $(m-1)^n$ sont possibles. Il y a donc $(m-1)^n + 1$ valeurs différentes à coder. Donc :

$$\lceil \log_2 \left((m-1)^n + 1 \right) \rceil$$

Commentaire : Question assez mal réussie dans l'ensemble. Il semble que le nombre de bits nécessaires pour représenter K objets/valeurs ne soit pas encore vraiment intégré.

Considérez l'algorithme suivant :

algo1

entrée : *nombre entier strictement positif* p
sortie : *? ? ?*

```
q ⟵ 1
Tant que q ≤ p²
    q ⟵ 2 q
Sortir : q
```

② **[3 points]** Cochez toutes les affirmations vraies (pour tout $p \geq 1$). **Attention !** Pénalité pour de mauvaises coches.

- [ ] $\frac{p^2}{2} \leq$ **algo1**$(p) \leq p^2$
- [✔] $p^2 <$ **algo1**$(p) \leq 2\,p^2$
- [ ] $2p^2 <$ **algo1**(p)
- [✔] **algo1**(p) est pair
- [✔] **algo1**(p) est une puissance de 2
- [ ] aucune des autres propositions n'est vraie pour tout $p \geq 1$

③ **[4 points]** Quelle est la complexité de **algo1** ? Justifiez *pleinement* votre réponse.

Réponse et justification :

La boucle va de 1 à (au plus) $2\,p^2$ en multipliant par 2, d'où $\log_2 \left(2\,p^2\right)$ étapes ; tout le reste sont des opérations élémentaires, d'où :

$$\left(\alpha \log_2 \left(2\,p^2\right) + \beta\right) \in \Theta(\log p)$$

Commentaire : Bien réussi dans l'ensemble ; même si plusieurs expriment la complexité en n sans dire ce qu'est n, et/ou oublient de justifier que le reste dans la boucle ne sont que des instructions élémentaires (ce qui changerait la complexité si ce n'était pas le cas).

Considérez l'algorithme suivant, où $\lfloor x \rfloor$ représente la partie entière de x :

algo2
entrée : *nombre entier positif* p sortie : *? ? ?*
Si $p < 2$ **Sortir :** $p + 1$ **Sortir :** $3 \times$ **algo2**$(\lfloor \frac{p}{2} \rfloor) + 2$

④ **[2 points]** Quelle est la sortie de **algo2**(6) ? Justifiez *brièvement* votre réponse.

⑤ **[3 points]** Quelle est la complexité de **algo2** ? Justifiez *pleinement* votre réponse.

Réponses et justifications :

$$3\,\textbf{algo2}(3) + 2 = 3\,(3\,\textbf{algo2}(1) + 2) + 2 = 3 \times 8 + 2 = 26$$

On part de p pour arriver à 1 en divisant à chaque fois par 2. Il y a donc autant d'étapes que de fois qu'on peut diviser p par 2, soit $\log_2(p)$. Tout le reste sont des opérations élémentaires. La complexité est donc en $\Theta(\log p)$.

Considérez l'algorithme suivant :

algo3
entrée : *liste* L *de nombres entiers positifs* sortie : *nombre entier positif* p

$n \longleftarrow$ **taille**(L)
$p \longleftarrow 0$
Pour i de 1 à $n - 1$
  Pour j de $i + 1$ à n
  Si $L[i] < L[j]$
  $p \longleftarrow p + 1$
Sortir : p

⑥ **[2 points]** Quelle est la sortie de **algo3** pour la liste $(2, 4, 1, 5)$?
Justifiez *brièvement* votre réponse.

⑦ **[3 points]** Quelle est la complexité de **algo3** ? Justifiez *brièvement* votre réponse.

Réponses et justifications : 4.

Cet algorithme compte le nombre de couples de valeurs $(L[i], L[j])$ tels que $L[i] < L[j]$.
Il y a quatre tels couples : $(2, 4)$, $(2, 5)$, $(4, 5)$ et $(1, 5)$.

La boucle en i est de l'ordre de n (où n est la taille de la liste) et la boucle en j aussi et tout le reste sont des opérations élémentaires, donc le tout est de l'ordre de n^2.

Formellement :

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1} n - i = n\,(n-1) - \frac{n\,(n-1)}{2} = \frac{n(n-1)}{2}$$

⑧ **[3 points]** Quelle est l'écriture en virgule flottante (binaire) avec 4 bits d'exposant et 4 bits de mantisse, du nombre x solution de l'équation $2^{-5} \times x = -\frac{21}{16}$?
Justifiez *pleinement* votre réponse.

Réponse puis justification :

$$x = -2^5 \times \left(1 + \frac{1}{4} + \frac{1}{16}\right)$$

qui est *directement* l'écriture définissant le format utilisé pour la représentation en virgule flottante.

x est négatif, donc son bit de signe est à `1`. L'exposant est 5, noté `0101` sur 4 bits. Et la mantisse vaut donc : `0101`

Ce qui donne au final par exemple (dans l'ordre signe, exposant, mantisse) : `1 0101 0101`

Commentaire : L'écriture en virgule flottante possède *forcément* un bit de signe (cf le cours).

⑨ **[2 points]** Si un nombre entier s'écrit `011010` en binaire signé sur **6** bits, comment s'écrit son opposé (toujours sur 6 bits) ?
Justifiez *brièvement* votre réponse.

Réponse puis justification :

`100110`, en appliquant simplement le complément à deux : flip tous les bits, puis ajouter 1.

⑩ **[3 points]** Comment s'écrit la valeur -200 en binaire signé sur **9** bits ?
Justifiez votre réponse.

Réponse puis justification :

`100111000`

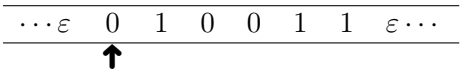
Soit par écriture binaire de 200 (`011001000`) puis complément à deux, soit par $512 - 200 = 312$.

Question 2 – Responsum ex machina [16 points]

On considère la machine de Turing ayant pour table de transition :

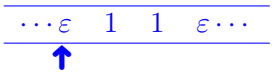
	`0`	`1`	ε
1	$(2, \varepsilon, +)$	$(1, \varepsilon, +)$	$(10, \texttt{0}, -)$
2	$(3, \varepsilon, +)$	$(1, \varepsilon, +)$	$(10, \texttt{0}, -)$
3	$(3, \varepsilon, +)$	$(4, \varepsilon, +)$	$(6, \texttt{1}, -)$
4	$(2, \varepsilon, +)$	$(5, \varepsilon, +)$	$(10, \texttt{0}, -)$
5	$(2, \varepsilon, +)$	$(1, \varepsilon, +)$	$(6, \texttt{1}, -)$
6	$(10, \texttt{1}, -)$	$(10, \texttt{1}, -)$	$(10, \texttt{1}, -)$

① **[5 points]** Quel est l'état de la bande et la position de la tête de lecture lorsque la machine s'arrête, si elle a démarré avec sa tête de lecture positionnée comme suit :



② **[11 points]** Justifiez votre réponse en trois ou quatre phrase(s), puis dites en une phrase ce que fait cette machine.

Réponses :



On peut peut être commencer par remarquer que cette machine efface tout (ε partout), sauf dans l'état 6, lequel ajoute simplement un `1` devant.

On peut ensuite remarquer que cette machine ne sort que sur lecture de ε, soit en écrivant `0` (et c'est tout), soit en écrivant `1` puis allant dans l'état 6 (donc écrit `11` au final).

L'état 1 attend (tout en effaçant comme nous l'avons déjà dit) un `0` ou un ε. En cas de `0`, on voit en suivant les états 2, 3, puis éventuellement 4,5 qu'elle avance jusqu'à l'état 6, soit sur `00`, soit sur `0011`. Dans tous les autres cas, soit elle recommence (en 1 ou en 2), soit elle sort en écrivant `0`.

Cette machine vérifie donc que l'entrée termine soit par `00`, soit par `0011`. Elle répond `11` si c'est le cas, et `0` sinon.

Commentaire : Beaucoup réussissent très bien le déroulement (à part quelques erreurs de placement de la tête), mais trop peu arrivent à abstraire les états (et encore moins le comportement de la machine).

Par exemple, peu font le parrallèle entre l'état 1 et un `if`, ou entre revenir à l'état 1 et faire une récursion.

suite au dos ☞

Question 3 – Réflexions algorithmiques [25 points]

Note : lisez la sous-question ③ (au dos) avant de répondre à ①.

Soit L une liste quelconque de nombres entiers. On cherche à écrire un algorithme qui remplace toute séquence de nombres égaux consécutifs par la valeur commune suivie du compte de répétitions.

Par exemple, pour la liste $L = (5, 5, 5)$ en entrée, l'algorithme sortira la liste $(5, 3)$, puisque la valeur 5 est répétée trois fois. De façon similaire, pour la liste $(5, 5, 5, 1, 1, 1, 1, 8, 8, 1)$, il sortira $(5, 3, 1, 4, 8, 2, 1, 1)$.

① **[8 points]** Écrivez un algorithme pour résoudre ce problème.

Réponse :

Voici un premier algorithme, simple :

RLEi

entrée : *liste L telle que décrite*
sortie : *liste L' « comprimée » (RLE)*

```
n ⟵ taille(L)
Si n = 0
    Sortir : () // liste vide
L' ⟵ (L[1])
s ⟵ 1
Pour i de 1 à n − 1
    Si L[i] = L[i + 1]
        s ⟵ s + 1
    Sinon
        L' ⟵ L' ⊕ (s, L[i + 1]) // ajout en fin de liste
        s ⟵ 1
L' ⟵ L' ⊕ (s) // ajout en fin de liste
Sortir : L'
```

Voici une autre solution :

RLEi

entrée : *liste L telle que décrite*
sortie : *liste L' « comprimée » (RLE)*

```
n ⟵ taille(L)
L' ⟵ () // liste vide
i ⟵ 1
Tant que i ≤ n
    s ⟵ 1
    j ⟵ i + 1
    Tant que j ≤ n et L[j] = L[i]
        s ⟵ s + 1
        j ⟵ j + 1
    L' ⟵ L' ⊕ (L[i], s) // ajout en fin de liste
    i ⟵ j
Sortir : L'
```

Remarques :
- on peut supposer le **et** comme étant « paresseux » (la seconde condition n'est pas évaluée si la première est fausse ; sinon, écrire cette seconde condition dans un « **Si** ») ; dans tous les cas, attention à ne pas déborder avec j ;
- la sortie désirée pour la liste vide n'est pas spécifiée ; on pourra accepter soit la liste vide (comme ici), soit la liste contenant seulement 0 : (0), soit éventuellement $(x, 0)$ avec x un nombre entier quelconque.

Commentaire : Beaucoup d'algorithmes ne fonctionnaient pas sur l'un ou l'autre des exemples donnés (p.ex. par oubli de sortir le compte de la dernière valeur rencontrée) : pensez à tester vos algorithmes sur les exemples donnés (ils n'ont pas été choisi au hasard).

Aussi beaucoup de solutions proposées n'étaient pas correctes pour la liste vide ou la liste réduite à un seul élément : pensez aussi à vérifier vos algorithmes sur les cas limites.

Les erreurs les plus fréquentes sont :
- faire des boucles (`while`) qui ne changent pas leur condition de vérité (p.ex. qui n'incrémentent pas la valeur testée) : cela donne des boucles infinies ;
- des variables utilisées sans être initialisées (typiquement la liste de retour non initialisée à vide) ;
- des accès erronés ($L[i-1]$ ave i qui vaut 0 ou 1 ; $L[i+1]$ avec i qui vaut la taille de la liste ; ...)

② **[4 points]** Quelle est la complexité de votre algorithme proposé en ① ? **Justifiez** votre réponse.

Réponse et justification : La complexité de l'algorithme précédent est en $\Theta(n)$ où n est la taille de la liste. On parcourt en effet la liste en entier par bloc de valeurs consécutives (et tout le reste sont des instructions élémentaires). L'instruction clé ici est l'affectation de j à i qui fait sauter i du parcourt effectué par j (que l'on ne refait donc pas).

Commentaire : Il ne suffit pas d'avoir une boucle pour conclure : il faut aussi compter ce qui est *dans* la boucle, p.ex. ici dire que ce sont des opérations élémentaires. (On peut très bien avoir des choses compliquées dans des boucles, ce qui augment d'autant la complexité !)

Il est aussi important de dire par rapport à quoi on mesure la complexité (c'est quoi « n » ?).

③ **[9 points]** Écrivez un algorithme *récursif* et sans boucle pour résoudre le problème proposé.

Si votre réponse à ① est déjà un algorithme récursif, alors vous n'avez rien de plus à faire ici que le dire (et serez, bien entendu, noté(e) sur la somme des points des deux sous-questions).

Réponse : Voici un algorithme récursif pour résoudre cette tâche :

RLEr
entrée : *liste L telle que décrite*
sortie : *liste L' « comprimée » (RLE)*

$n \longleftarrow$ **taille**(L)
Si $n = 0$
  Sortir : L
Si $n = 1$
  Sortir : $(L[1], 1)$
$L' \longleftarrow$ **RLEr**$(L[2], \cdots, L[n])$
Si $L[1] = L[2]$
  $L'[2] \longleftarrow L'[2] + 1$
Sinon
  $L' \longleftarrow (L[1], 1) \oplus L'$ // ajout en début de liste
Sortir : L'

Remarque : si l'écriture « $L'[2] \longleftarrow L'[2] + 1$ » ne vous plait/parle pas, on peut aussi écrire : $L' \longleftarrow (L'[1], L'[2] + 1, L'[3], ..., L'[m])$ avec m la taille de L' (ou tout autre variante similaire à base de $\oplus$).

Commentaire :

Très peu réussie, mais c'était la difficulté majeure de l'exercice. Il est important de retravailler cet aspect (récursivité) pour celles et ceux qui ont encore de la peine.

Voici quelques conseils (sur cet exercice) :

- plusieur(e)s étudiant(e)s ajoutent des entrées qui ne correspondent pas au problème de départ (typiquement ici, un compte) : dans ce cas là, il faut faire un autre petit algorithme préalable qui appelle, avec les bons arguments, l'algorithme récursif à arguments supplémentaires ;
- vérifiez tous les cas de condition d'arrêt : liste vide ou liste à un élément ; l'algorithme doit fonctionner pour tous les cas ;
- appels récursifs non utilisés : il ne suffit pas d'écrire l'appel, il faut l'utiliser, p.ex. le mettre dans une liste que l'on manipule ensuite, ou alors le mettre dans un « **Sortir** » ;
- pensez à vérifier/rester compatible avec les arguments lors de l'appel récursif : même nombre, mêmes types ;
- de même, attention au type de retour : ne pas faire les opérations illicites sur ce qui est retourné (p.ex. faire une addition (nombre) sur une liste, ou dans l'autre sens ajouter des éléments à un nombre).

④ **[4 points]** Quelle est la complexité de votre algorithme proposé en ③ ? **Justifiez** votre réponse.

Si votre réponse à ① était déjà un algorithme récursif, et que vous avez répondu à sa complexité en ②, alors vous n'avez rien de plus à faire ici (et serez, bien entendu, noté(e) sur la somme des points des deux sous-questions).

Réponse et justification : On peut ici sans perte de généralité supposer que le calcul de la taille est en $\Theta(1)$ (on pourrait simplement la calculer auparavant et la passer comme argument supplémentaire, il n'y a aucune difficulté à en garder la trace/calculer la nouvelle valeur). La complexité de l'algorithme précédent est alors en $\Theta(n)$.

Si l'on suppose `taille` en $\Theta(n)$, alors l'algorithme est en $\Theta(n^2)$.

En effet, chaque appel récursif diminue de 1 la longueur de la liste. Il y donc au plus n appels récursifs. Et tout le reste est en $\Theta(1)$.

Commentaire : Comme pour ②, plusieurs justifications manquent de rigueur (arguments incomplets).

suite au dos ☞

Question 4 – Sommes nulles [35 points]

Le problème SSP est un problème de décision sur un ensemble de nombres entiers, qui cherche à déterminer s'il existe un sous-ensemble non-vide de somme nulle.

Par exemple, pour l'ensemble $\{3, 4, 12, -1, 6, -2\}$ la réponse est « oui » puisque le sous-ensemble $\{3, -1, -2\}$ somme bien à 0.
Par contre, pour l'ensemble $\{3, 2, 14, -8, 1\}$ la réponse est « non » puisqu'aucun de ses sous-ensembles de valeurs ne somme à 0.

SSP est connu pour être dans NP. Mais qu'est-ce que cela veut dire concrètement ? Nous allons regarder cela d'un peu plus près.

Note : si cela vous aide, vous pouvez, sans perte de généralité, considérer qu'un ensemble est une liste.

① **[2 points]** Pour commencer, pour écrire des algorithmes sur des sous-ensembles d'un ensemble, il est parfois plus simple de voir un sous-ensemble comme l'écriture binaire d'un entier positif : `1` si l'élément est présent dans le sous-ensemble, et `0` s'il ne l'est pas

Par exemple, dans l'exemple ci-dessus ($\{3, 4, 12, -1, 6, -2\}$), le sous-ensemble $\{3, -1, -2\}$ serait représenté par le nombre binaire `100101` : 3 est présent, 4 et 12 ne le sont pas, -1 est présent, 6 ne l'est pas et -2 l'est.

À quelle valeur décimale correspond `100101` ? Justifiez *brièvement* votre réponse.

Réponse et justification : 37 ($= 1 + 4 + 32$).

② **[10 points]** En supposant qu'il existe un algorithme **somme** qui prend en entrée un ensemble E et un nombre entier k positif non nul et qui donne en sortie la somme des éléments de E correspondant au sous-ensemble de E décrit par l'écriture binaire de k, proposez un algorithme permettant de répondre au problème SSP.

Réponse :

SSP

entrée : *ensemble E de nombres entiers*
sortie : *un entier décrivant un sous-ensemble de somme nulle, 0 sinon*

```
n ⟵ taille(E)
Pour i de 1 à 2^n − 1
    Si somme(E, i) = 0
        Sortir : i
Sortir : 0
```

Note : on peut aussi ici accepter juste la sortie « oui » ou « non ».

Commentaire : Attention aux bornes !

③ **[3 points]** En supposant que la complexité de **somme** est en $\Theta(n)$ où n est la taille de E, quelle est la complexité de l'algorithme que vous avez proposé en ② ? **Justifiez** votre réponse.

④ **[3 points]** Cette complexité est-elle en contradiction avec l'affirmation que SSP est dans NP ? Détaillez votre réponse.

Réponses et justifications : Sa complexité est (au moins) exponentielle (« $\Theta(2^n)$ ») parce qu'il parcourt toutes les configurations possibles (tous les sous-ensembles non vides).

Celà n'est pas du tout contraditoire avec le fait d'être dans NP, qui ne parle que de vérification, pas de résolution.

(Et de plus cet algorithme n'est pas polynomial, ce qui prouverait que SSP est dans P. La donnée ne dit rien à ce sujet et aurait pu en effet considérer un problème qui est dans P. En fait, on ne sait pas à ce jour si SSP est dans P et s'il l'était, alors on saurait que NP=P.)

⑤ **[3 points]** Quel serait un certificat pour une instance (positive) de ce problème ?

Illustrez votre explication dans le cas concret donné en exemple au début.

Réponse : un sous-ensemble (F) non vide de l'instance (E) tel que la somme de ses éléments (de F) soit nulle.

Comme dans l'exemple du début, avec $E = \{3, 4, 12, -1, 6, -2\}$ et $F = \{3, -1, -2\}$.

⑥ **[8 points]** Proposez un algorithme qui, recevant un ensemble et un certificat tel que vous l'avez défini en ⑤, permet de vérifier qu'une instance positive de SSP en est bien une.

Vous **ne** pouvez **pas** utiliser l'algorithme **somme** ici.

Réponse :

```
SSP
entrée : ensemble E et ensemble F
sortie : oui ou non
    s ⟵ 0
    Pour tout élément f de F
    |   Si recherche(E, f)
    |   |   s ⟵ s + f
    |   Sinon
    |   |   Sortir : « non »
    Sortir : s = 0 et taille(F) > 0
```

Notez qu'il est important de vérifier que le F reçu est bien un sous-ensemble non-vide de E (si vous donnez F encodé sous la forme d'un entier tel que fait en ①, alors il faut vérifier qu'il est entre 1 et $2^n - 1$ (où n est la taille de E)).

⑦ **[3 points]** Quelle est la complexité de l'algorithme que vous avez proposé en ⑥ ? Justifiez votre réponse.

⑧ **[3 points]** Cette complexité est-elle en contradiction avec l'affirmation que SSP est dans NP ? Détaillez votre réponse.

Réponses et justifications : La complexité est en $n \times m$ avec n la taille de E et m la taille de F, puisque l'on parcourt tous les éléments de F et que la recherche dans E est en $\Theta(n)$ (et que tout le reste sont des opérations élémentaires).

Le pire cas est quand $F = E$: on a alors un algorithme en $\Theta(n^2)$.

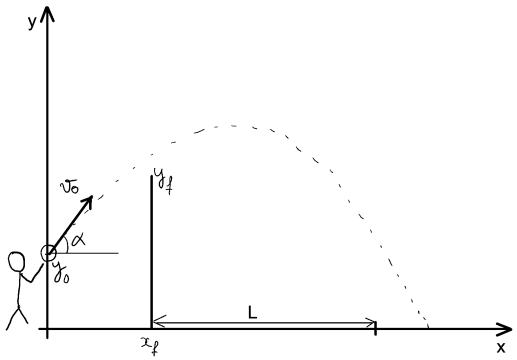
Ceci est en parfait accord avec le fait que SSP est dans NP : on a ici un algorithme de vérification (du « oui ») en un temps polynomial.

C'est une *preuve* que SSP est dans NP.

Question 5 – Simulation de tir [23 points]

On cherche à écrire *des parties* d'un programme C++ permettant de simuler le lancer d'un ballon au volley.

Le problème auquel on s'intéresse est donc un problème de simulation, en 2D, d'un objet en chute libre étant donné une vitesse initiale v_0 d'angle α par rapport à l'horizontale (voir illustration ci-contre).



Les données du problème à simuler sont :

- la hauteur y_0 d'où le joueur (de volley) lance le ballon (depuis l'origine des abscisses : $x_0 = 0$) ;
- la vitesse initiale v_0 du ballon ;
- l'angle initial α ;
- la position x_f du filet est sa hauteur y_f ;
- la longueur L du terrain (depuis le filet).

Les équations des coordonnées (x, y) du ballon sont :

$$x = v_0 \cos(\alpha)\, t$$
$$y = y_0 - \frac{g}{2} t^2 + v_0 \sin(\alpha)\, t$$

① **[2 points]** Écrire (en C++) la fonction `hauteur_ballon()` qui prend en entrée y_0, v_0, α et un temps t et qui retourne l'ordonnée y du ballon au temps t.

Réponse : Voici une version possible :

```
double hauteur_ballon(double y0, double v0, double alpha, double t)
{
  return y0 - g * t * t / 2.0 + v0 * sin(alpha) * t;
}
```

On peut soit supposer que `g` est une variable globale (le dire), soit écrire explicitement :

```
constexpr double g(9.81);
```

Commentaire : Ne *jamais* mettre de constantes « en dur » dans le code (c.-à-d. des valeurs explicites).

Par ailleurs, n'utilisez pas `const` là où `constexpr` est approprié.

② **[2 points]** Écrire (en C++) la fonction `temps_x()` qui prend en entrée v_0, α et une abscisse x du ballon, et qui retourne le temps t auquel x est atteinte.

Réponse : Voici une version possible :

```
double temps_x(double v0, double alpha, double x)
{
  return x / (v0 * cos(alpha));
}
```

③ **[7 points]** Écrire (en C++) la fonction `lancer()` qui prend en entrée y_0, v_0, α et un temps t et qui retourne l'absisse x_{fin} du ballon lorsqu'il touche le sol.

Nous vous demandons pour cela :

- de définir une constante `dt` égale à 0.1 ;
- de calculer **de proche en proche** (tous les `dt`) l'ordonnée y du ballon à partir du temps donné t et jusqu'à ce qu'elle soit nulle (le ballon touche le sol) ;
- de retourner son absisse à ce temps là.

Réponse : Voici une version possible, très compacte :

```
constexpr double dt(0.1);

double lancer(double y0, double v0, double alpha, double t)
{
  while (hauteur_ballon(y0, v0, alpha, t) > 0) t += dt;
  return v0 * cos(alpha) * t;
}
```

Voici une autre version aussi acceptée (malgré le `dt` de différence sur les temps) :

```
constexpr double dt(0.1);

double lancer(double y0, double v0, double alpha, double t)
{
  double y(0.0);
  do {
    y = hauteur_ballon(y0, v0, alpha, t);
    t = t + dt;
  } while (y > 0);

  return v0 * cos(alpha) * t;
}
```

Commentaire : On ne doit *jamais* faire de comparaison explicite de `double` (revoir le cours de théorie I.4).

④ **[12 points]** Écrire (en C++) la fonction `simulation()` qui prend en entrée y_0, les coordonnées x_f, y_f du filet, et la longueur L du terrain, et qui :

- demande à l'utilisateur les valeurs v_0 et α ;
  α sera demandé en degrés, puis sera converti en radians (multiplié par 0.017453292517) ;
- calcule le temps t_f auquel le ballon passe le filet (temps auquel l'absisse est x_f) ;
- calcule la hauteur du ballon à ce temps t_f ;
- affiche « `touche le filet` » si cette ordonnée est inférieure à y_f ;
- et sinon :
  - affiche « `passe` » ;
  - calcule l'absisse à laquelle le ballon touche le sol ;
  - affiche « `out` » si cette absisse est plus grande que $x_f + L$.

Réponse :

Voici une version possible :

```
void simulation(double y0, double x_filet, double y_filet, double L)
{
  double v0;
  cout << "v0 : ";
  cin >> v0;

  double alpha;
  cout << "alpha (deg) : ";
  cin >> alpha;
  alpha *= numbers::pi / 180.0; // 0.017453292517

  const double t_filet(temps_x(v0, alpha, x_filet));

  if (hauteur_ballon(y0, v0, alpha, t_filet) < y_filet) {
    cout << "touche le filet" << endl;
  } else {
    cout << "ça passe" << endl;

    if (lancer(y0, v0, alpha, t_filet) > x_filet + L) {
      cout << "out" << endl;
    }
  }
}
```

Commentaire : *Jamais* de copié-collé (pensez à réutiliser les fonctions que vous avez faites).

suite au dos ☞

Question 6 – C'est pas juste ! [19 points]

Voici un programme C++ visant à maximiser numériquement une fonction :

```
 1  #include <iostream>
 2  #include <cmath>
 3  using namespace std;
 4
 5  constexpr double pi(3.14159265358979323846);
 6  constexpr double epsilon(1e-5);
 7
 8  double f(double x)
 9  { return   5.0 * sin(0.3*pi * x + pi/4.0)
10           - 7.0 * sin(0.4*pi * x + pi/6.0); }
11
12  double df(double x)
13  { return ( f(x+epsilon) - f(x) ) / epsilon; }
14
15  int main()
16  {
17    double a(0);
18    cout >> "Point de départ : ";
19    cin << a;
20
21    constexpr double pas(0.1);
22    double gain(0.0);
23    int nb_iter(0);
24
25    do {
26      const double b(a + pas * df(a));
27      gain = f(b) - f(a);
28      ++nb_iter;
29      cout << "    " << a << " --> " << f(a) << endl;
30    } while ((gain != epsilon) or (nb_iter > 1000));
31
32    cout << "Max : " << f(b) << " en x=" << b << endl;
33
34    return 0;
35  }
```

Mais ce programme comporte plusieurs erreurs de programmation, possiblement de différente nature (syntaxe, déroulement, conception, méthodologie, ...).
Indiquez et **corrigez** toutes les erreurs (directement sur le code ci-dessus).
Expliquez *brièvement* les erreurs/corrections à droite du code ou sur la page ci-contre.
On ôtera 1 point pour toute indication d'une erreur qui n'en est pas une.

Explications :

Les cinq/six erreurs sont :

- lignes 18 et 19 : inversion du sens de `<<` et `>>`
- lignes 25–30 : `a` n'est jamais modifié ; il faut le mettre à jour, p.ex. `a=b;`
- ligne 30 : (doublement) pas de sens de tester l'égalité de `gain` ; une correction qui fait sens serait `gain > epsilon` ;
- ligne 30 : la condition du `while` ne fait pas de sens (le `while` ne se fait pas du tout) : il faut un `and (nb_iter <= 1000)` (l'inégalité stricte va aussi) ;
- ligne 32 : `b` n'est pas dans cette portée ; ce devrait être `a` ici (ou augmenter la portée de `b`).

Commentaire : L'erreur de portée sur `b` n'a pas souvent été vue.

Et trop de fausses erreurs inventées.