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, sections MA et PH
Chappelier J.-C.

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

5 novembre 2021

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**.
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 des assistants.
6. L'examen comporte 8 exercices indépendants sur 12 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 150 points).
   Tous les exercices comptent pour la note finale.

Question 1 – Coder [11 points]

Un étudiant a écrit l'algorithme suivant :

Machin

entrée : n, *entier strictement positif*
sortie : *? ? ?*

$p \longleftarrow 0$
$a \longleftarrow 1$
$b \longleftarrow 0$
Tant que $a < n$
  $b \longleftarrow b + 1$
  Si b est un multiple de 3 ou de 5
    $p \longleftarrow p + b$
    $a \longleftarrow a + 1$
Sortir : p

① **[4 points]** Quelle valeur sort cet algorithme pour $n = 8$? Expliquez cette valeur en une phrase/formule.

② **[7 points]** Ecrire une fonction C++ qui implémente cet algorithme.

Réponses :

① somme des 7 premiers multiples communs :
$60 = 3 + 5 + 6 + 9 + 10 + 12 + 15$.

Commentaire : Très bien en général, pas beaucoup d'erreurs ; sauf celles et ceux qui n'ont pas pris en compte l'inégalité stricte et d'autres qui calcule le produit des n premiers multiples des deux nombre (« et » au lieu de « ou »).
Parfois aussi des « do-while » à la place du `while` ou initialisation de `a` à `0` (au lieu de `1`).

Réponses (suite) :

②

```
int f(int n)
{
  int p(0);
  int a(1);
  int b(0);
  while (a < n) {
    ++b;
    if ((b % 3 == 0) or (b % 5 == 0)) {
      p += b;
      ++a;
    }
  }
  return p;
}
```

Commentaire : Beaucoup d'élèves ont écrit un « code complet », avec demande de `n` avec `cin`, `main()`, `#include` et un `cout`, au lieu d'une *fonction*.

Question 2 – Ecriture d'algorithmes [45 points]

① **[20 points]** Ecrivez un algorithme qui effectue la fusion triée de deux listes triées.

Par exemple, si en entrée on a les listes $L_1 = (3, 7, 9, 12, 54)$ et $L_2 = (2, 9, 65)$, alors en sortie on aura la liste $(2, 3, 7, 9, 9, 12, 54, 65)$.

② **[5 points]** Quelle est la complexité de votre algorithme ? Justifiez votre réponse.

Réponses :

Commentaire : Globalement : assez agréablement surpris par les réponses à la question ③, même si c'est encore loin d'être parfait : pas mal d'étudiant(e)s ont réussi à trouver un algo qui en principe pourrait fonctionner ; mais, étrangement, assez déçu des réponses à la première question qui était vraiment simple (quelque soit l'approche choisie).

On peut globalement dire que vous n'êtes pas encore assez rigoureux/rigoureuses dans votre démarche ou dans vos écritures et faites (ou ne vérifiez pas) encore trop souvent des hypothèses trop générales, fausses ou inutiles.

Et beaucoup utilisent des variables sans les définir/initialiser.

① Il y a plein de façons d'écrire un tel algorithme. Par exemple :

fusion1

entrée : *Deux listes triées* L_1 *et* L_2
sortie : *La fusion triée de* L_1 *et* L_2

$t_1 \longleftarrow$ **taille**(L_1)
$t_2 \longleftarrow$ **taille**(L_2)
$i \longleftarrow 1$
$j \longleftarrow 1$
$L \longleftarrow ()$ // Liste vide
Tant que $(i \leq t_1)$ **ou** $(j \leq t_2)$
  Tant que $(i \leq t_1)$ **et** $((j > t_2)$ **ou** $(L_1[i] < L_2[j]))$
    $L \longleftarrow L \oplus (L_1[i])$ // ajout en fin de liste
    $i \longleftarrow i + 1$
  Tant que $(j \leq t_2)$ **et** $((i > t_1)$ **ou** $(L_1[i] > L_2[j]))$
    $L \longleftarrow L \oplus (L_2[j])$ // ajout en fin de liste
    $j \longleftarrow j + 1$
Sortir : L

(en notant $\oplus$ la concaténation de listes ; d'autres écritures sont possibles pour cela.)

Ce à quoi il faut particulièrement prêter attention est :
- de ne pas « déborder » des listes (index inférieurs aux tailles avant accès) ;
- de ne pas se tromper dans les connecteurs logigues (**et** , **ou**).

On peut aussi utiliser l'algorithme de tri vu en cours :

fusion2

entrée : *Deux listes triées* L_1 *et* L_2
sortie : *La fusion triée de* L_1 *et* L_2

$L \longleftarrow L_1$
Pour tout élément e de L_2
  $L \longleftarrow L \oplus (e)$
Sortir : trier(L)

(Note : j'ai écrit ici explicitement la boucle pour mieux montrer la complexité ; la concaténation de listes n'étant pas une opération élémentaire. Mais écrire « **Sortir : trier**$(L_1 \oplus L_2)$ » est aussi valide (si tant est que la notation $\oplus$ ait été clairement définie).

Ou encore écrire une version récursive :

fusion3

entrée : *Deux listes triées* L_1 *et* L_2
sortie : *La fusion triée de* L_1 *et* L_2

Si L_1 est vide
  Sortir : L_2
Si L_2 est vide
  Sortir : L_1
Si $L_1[1] < L_2[1]$
  $t_1 \longleftarrow$ **taille**(L_1)
  Sortir : $(L_1[1]) \oplus$ **fusion3** $((L_1[2], ..., L_1[t_1]), L_2)$
Sinon
  $t_2 \longleftarrow$ **taille**(L_2)
  Sortir : $(L_2[1]) \oplus$ **fusion3** $(L_1, (L_2[2], ..., L_2[t_2]))$

Commentaire : Beaucoup d'algorithmes en $\Theta(n_1 \times n_2)$ où la liste finale est de taille $n_1 \times n_2$. Regarder la taille de la liste finale me semble être un bon sanity check que beaucoup n'ont pas utilisé.

Aussi énormément d'erreurs d'index "out of bound", pas suffisamment d'étudiant(e)s font attention à la taille des listes qu'ils manipulent.

Et aussi beaucoup s'arrêtent trop tôt, quand une (souvent la première) des deux listes est traitée, oubliant de traiter le reste de l'autre liste.

Enfin, trop oublient de traiter les listes vides (ou font l'hypothèse erronée qu'elles ne le sont pas).

Remarque : certain(e)s utilisent la suppression d'éléments de la liste, ce qui, en algorithmique, n'est pas possible en soit (ce sont des mathématiques) et pas forcément désirable en programmation (immu-

tabilité – argument const)

② La taille n de l'entrée de cet algorithme est la somme des taille de L_1 et L_2. Parcourant chacune des listes une seule fois, la complexité temporelle pire cas de cet algorithme est en $\Theta(n)$.

Attention à la complexité de ceux qui utilisent le tri !! Elle est en $\Theta(n \log n)$.

③ **[20 points]** Ecrivez un algorithme *récursif* qui, prenant en entrée deux listes, effectue la fusion alternée de la première liste et du renversement (ordre de lecture inverse) de la seconde.

Par exemples :

- si en entrée on a les deux listes $L_1 = (1, 2, 3, 4)$ et $L_2 = (-11, -22, -33)$ (dans cet ordre), alors en sortie on aura la liste $(1, -33, 2, -22, 3, -11, 4)$;
- si en entrée on a les deux listes $L_1 = (12, 3, -5, 31, 6)$ et $L_2 = (8, 23, 11)$ (dans cet ordre), alors en sortie on aura la liste $(12, 11, 3, 23, -5, 8, 31, 6)$;
- par contre si en entrée on a les deux listes dans l'autre ordre : $L_1 = (8, 23, 11)$ et $L_2 = (12, 3, -5, 31, 6)$, alors en sortie on aura la liste $(8, 6, 23, 31, 11, -5, 3, 12)$.

Réponse :

fusion4

entrée : *Deux listes* L_1 *et* L_2
sortie : *La fusion de* L_1 *et* L_2 *renversée*

Si L_2 est vide
  Sortir : L_1

$t_2 \longleftarrow$ **taille**(L_2)
Si L_1 est vide
  Sortir : $(L_2[t_2]) \oplus$ **fusion4**$(L_1, L_2[1 : t_2 - 1])$

$t_1 \longleftarrow$ **taille**(L_1)
Sortir : $(L_1[1], L_2[t_2]) \oplus$ **fusion4**$(L_1[2 : t_1], L_2[1 : t_2 - 1])$

avec les notations :

- $L[i : j]$ comme sous-liste de L allant des éléments $L[i]$ à $L[j]$ si $i \leq j$ et la liste vide sinon ; on peut aussi l'écrire : $L[i], \cdots, L[j]$;
- et $\oplus$ pour la concaténation de deux listes.

Commentaire : Comme dit dans le premier commentaire, une bonne partie de la classe semble maintenant assez à l'aise avec la récursion, ce qui me réjouit. Ceci dit, cette question présente de très gros écarts entre étudiant(e)s (il y a donc aussi de gros échecs ici).

Au niveau des erreurs : surtout des problèmes classiques, comme ne pas bien définir les conditions d'arrêt, et ne pas utiliser la valeur de retour de l'appel récursif (ou au contraire ne pas utiliser ce que fait l'appel courant et juste retourner le résultat de l'appel récursif).

Et aussi, trop oublient d'inverser la liste L_2 lorsque L_1 est vide.

Question 3 – Plan de quartier [10 points]

Une commune considère le problème « *boiser le quartier* » suivant :
elle veut offrir à chaque parcelle un arbre parmi 3 espèces (chêne, sapin ou hêtre) tout en s'assurant

— que deux parcelles voisines ne recoivent pas un arbre de la même espèce ;
— et qu'aucun arbre ne soit à moins de 25 m d'un autre arbre d'une même espèce.

Pour résoudre ce problème, elle envisage l'algorithme suivant :

1. décider si l'on peut effectivement donner un arbre à chaque parcelle de sorte que deux parcelles voisines ne recoivent pas un arbre de la même espèce ;
2. si c'est le cas, considérer une solution et calculer la distance minimale entre chaque paire de parcelles ayant reçu un arbre de la même espèce ;
3. vérifier que chacun de ces minima est supérieur à 25 m.

Que pouvez-vous dire de la complexité du problème « *boiser le quartier* » ?
Soyez le plus précis possible et **justifiez** votre réponse.

Réponse :

Ce problème contient comme sous-problème le problème de la 3-coloration de graphe planaire, qui est dans NP et dont on ne sait pas s'il est dans P.

« *boiser le quartier* » est donc au moins aussi difficile qu'un problème de NP.

Pour savoir si « *boiser le quartier* » est dans NP, il faudrait pouvoir *vérifier* une bonne solution en un temps polynomial.
En suivant l'approche proposée :

1. on sait vérifier en un temps polynomial que l'on a une bonne affectation des arbres (3-coloration est dans NP)
2. vérifier ensuite que 2 à 2 les arbres sont à au moins 25 m se fait en $\Theta(n^2)$ (n étant le nombre d'arbres)

On a donc aussi une vérification en un temps polynomial. Donc « *boiser le quartier* » est dans NP.

Par contre, on ne sait pas non plus si « *boiser le quartier* » est dans P.

Commentaire : C'était volontairement un exercice atypique pour évaluer à quel degré la théorie de la complexité algorithmique avait été assimilée.

Certain(e)s confondent complexité de problèmes et complexité d'algorithmes.

Voir que ce problèmen est **au moins** dans NP : beaucoup citent la 3-coloration et savent qu'elle est dans NP, *mais* trop supposent que si la partie 1 résout un problème dans NP, alors le reste de l'algorithme est au moins aussi complexe que NP, donc cela n'est presque jamais affirmé explicitement.

Voir que c'est dans NP : beaucoup ne prennent pas le temps de dire si c'est *vérifiable* en temps polynomial ; parmi ceux qui l'ont fait, beaucoup ont donné une analyse de complexité des étapes 2 et 3, utilisé la plupart des fois pour montrer la vérifiabilité d'une solution dans un temps polynomiale, donc dans le bon contexte.
Par contre, quand une complexité est exprimée en fonction d'une quantité (par ex. n), celle-ci n'est souvent pas définie (certaines fois elle représente le nombre de parcelles, d'autres fois le nombre de couples de parcelles).

Certain(e)s ont consideré le problème sur une seule dimension (c'est-à-dire des parcelles alignées).

L'étape 2 a été souvent considerée comme algorithme de plus court chemin, alors que c'est juste un contrôle de paires.
(Les problèmes de plus court chemin ne sont donc pas compris en soi).

Et trop affirment encore que « ce n'est pas dans P » !! (grosse erreur)

Question 4 – Complexité [20 points]

① **[5 points]** Pour trier un tableau de nombres de dimensions $n \times n$, on trie d'abord séparément chaque colonne avec un algorithme de tri optimal, puis ensuite chaque ligne avec ce même algorithme.
Quelle est la complexité temporelle de cet algorithme de tri du tableau en entier ?
Justifiez votre réponse.

Réponse :

- la taille d'une colonne est de n et son tri optimal est en $\Theta(n \log n)$;
- il y a n colonnes ; donc la première phase de tri est en $\Theta(n^2 \log n)$;
- de même la seconde phase, sur les lignes est en $\Theta(n^2 \log n)$;
- le tout est donc en $\Theta(n^2 \log n)$.
- Si l'on veut l'exprimer par rapport à la taille de l'entrée $t = n^2$, la complexité est alors en $\Theta(t \log \sqrt{t}) = \Theta(t \log t)$.

Commentaire : La plupart des étudiant(e)s connaissent la complexité de l'algorithme de tri, ce qui est bien, mais certain(e)s d'entre elles/eux, après avoir fait un bon raisonnement, négligent le $\log(n)$ par rapport au n^2 et donnent juste n^2 comme réponse (alors qu'ils se *multiplient* ici, et non pas s'additionnent).

---

② Considérez le programme suivant (que l'on appliquera qu'à des entiers positifs ou nuls) :

```
int f(int i, int j)
{
  if (i <= j) return 1;
  int s(i + j);
  while (i > j) {
    s = s + 2;
    --i;
  }
  return s + f(j, s);
}
```

a. **[3 points]** Calculer `f(5, 3)`.
b. **[12 points]** Quelle est la complexité de l'algorithme implémenté par `f` ? **Justifier** votre réponse.

Réponses :

① 13 $(= 3 \times 5 - 3 + 1)$

② L'algorithme implémenté par `f()` s'arrête si `i <= j`. Si `i > j`, alors

- `s` vaut `i+j+2(i-j)`, soit `3i-j`
- on rappelle donc `f(j, 3i-j)`
- mais `j <= 3i - j` car `i > j`
- donc on s'arrête

Il n'y a donc pas de récursion plus qu'une fois, soit $\Theta(1)$ fois.

L'exécution du reste est dominée par la boucle `while`, qui dans le pire cas s'exécute en $\Theta(i - j)$, avec $i > j$, qui est le pire lorsque j vaut 0, donc en $\Theta(i)$.

Noter qu'au final `f(i, j)` vaut donc : 1, si $i \leq j$; et $3i - j + 1$, sinon.

Commentaire : Trop semblent très hésitants à utiliser une autre variable que n dans la complexité. Souvent, ils/elles utilisaient n sans le définir.

Et cette partie est souvent mal justifiée même si la majorité semble avoir compris ce que fait l'algorithme.
Certain(e)s ignorent carrément l'aspect récursif.

Question 5 – Petites machines [15 points]

① **[6 points]** On considère la machine de Turing dont la table de transition est :

	0	1	ε
1	$(2, \varepsilon, +)$	$(3, \varepsilon, +)$	$(5, \varepsilon, +)$
2	$(2, 0, +)$	$(2, 1, +)$	$(4, 0, -)$
3	$(3, 0, +)$	$(3, 1, +)$	$(4, 1, -)$
4	$(4, 0, -)$	$(4, 1, -)$	$(5, \varepsilon, +)$

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

$\cdots\varepsilon$	1	0	1	0	1	0	1	0	$\varepsilon\cdots$
	↑								

Expliquez votre réponse en une ou deux phrase(s).

Réponse :

$\cdots\varepsilon$	0	1	0	1	0	1	0	1	$\varepsilon\cdots$
	↑								

Cette machine reporte le premier symbole en fin (« *left bit-rotate* ») :

- l'état 1 sert de « `if` » pour savoir si l'on a lu (et effacé) un `0` ou un `1` ;
- les états 2 et 3 vont chacun à la fin du mot écrit et y écrivent respectivement `0` ou `1` ;
- l'état 4 revient au début du mot.

Commentaire : Environ la moitié n'a pas compris ce que fait vraiment la machine, certaines fois le complément à 1 est donné comme réponse.

Souvent une simple explication pas à pas, sans vision globale.

Pas mal oublient de préciser la tête de lecture ou se trompent.

Beaucoup ont donc perdu 1.5 points par manque d'explication. La donnée n'étant pas assez explicite à ce sujet, **j'ai descendu mon barème global de 1.5 points (mais la perte de points reste marquée dans votre fiche de correction)**.

---

② **[9 points]** On considère maintenant une autre machine de Turing, dont la table de transition est :

	0	1	ε
1	$(1, 1, +)$	$(1, 0, +)$	$(2, \varepsilon, -)$
2	$(3, 1, -)$	$(2, 0, -)$	$(4, \varepsilon, +)$
3	$(3, 0, -)$	$(3, 1, -)$	$(4, \varepsilon, +)$

Complétez le corps de la fonction C++ `f` ci-dessous, de sorte à ce que, pour un `int`, elle fasse la même chose que la machine de Turing ci-dessus (ça peut être simple ; en tout cas pas plus que la place laissée libre ici) :

```
int f(int a)
{
  return -a;
}
```

Commentaire : Environ la moitié a compris ce que la machine fait.

Presque tout celles et ceux qui ont compris l'idée l'ont aussi implementé correctement ; même si certains parmis celles/ceux là écrivent de (faux) `2^n - a;` ou `2^8 - a;`
(`n` pas défini, erreur de syntaxe (`^` **n'**est **pas** la puissance en C++), mais, bien plus grave : de toutes façons « 2^n » n'est pas représentable !)

Certains, malgré le commentaire de la donnée, tentent d'implémenter la machine de Turing sur une bande imaginaire, ou en appliquant `f` avec `a` comme 0, 1 ou ε(!).

Question 6 – Représentations des nombres [19 points]

① **[4 points]** Sur une machine (moderne) où les `int` sont stockés sur 8 bits, quel est le schéma binaire en mémoire de la valeur de la variable `a` suivante :

```
int a(-35);
```

Justifiez brièvement votre réponse (p.ex. en explicitant vos calculs).

② **[5 points]** Combien vaut, en décimal, le résultat de l'opération en binaire signé par complément à 2, `01101000` + `01110101` ?

Justifiez brièvement votre réponse.

③ **[5 points]** Le nombre 0.7 (en base 10) peut il être représenté exactement (sans erreur d'arrondi) en binaire à virgule flottante avec 5 bits d'exposant et 10 bits de mantisse ?

Justifiez brièvement votre réponse.

④ **[5 points]** Quelle est la valeur décimale du nombre binaire représenté en virgule flotante sur 8 bits avec (dans cet ordre) 1 bit de signe, 2 bits d'exposant et 5 bits de mantisse, par `11001101` ?

Justifiez brièvement votre réponse (p.ex. en explicitant vos calculs).

Réponses :

① Il s'agit du complément à deux de 35.
La représentation binaire de 35 est `00100011` $(32 + 2 + 1)$.
Son complément à deux est `11011101`.
On peut aussi directement rechercher la représentation binaire non signée de $256 - 35 = 221$.

Commentaire :
- bien réussi
- quelques étudiant(e)s ont fait une erreur dans la conversion en binaire
- d'autres (très peu) ont fait une représentation sur 9, 7, ou 6 bits
- pour celles et ceux qui l'ont fait, le complément à deux est bien acquis ; quelques confusions cependant entre terminologie « complément à un » et « complément à deux »

② L'addition de `01101000` + `01110101` donne `11011101` (tiens, tiens !)
qui, en représentation signée est donc un nombre négatif, opposé de son complément à deux.
Le complément à deux de `11011101` est `00100011`, dont la valeur décimale est 35 (cf ①).
La réponse est donc -35.

Commentaire :
- beaucoup de variabilité, soit c'est bien réussi, soit l'étudiant(e) n'a pas compris
- certain(e)s convertissent en base 10, font l'addition et donnent le résultat
- env. 10% font au moins une erreur de calcul dans l'addition en binaire
- env. 10% font le complément à deux final, mais oublient de mettre le signe dans le résultat final

③ non : $0.7 = 0.5 + 2 \times 0.1$, et 0.1 n'est pas représentable exactement en binaire (écriture infinie).
Le fait que ce nombre n'est pas représentable en binaire à virgule flottante ne dépend pas de la convention (non précisée d'ailleurs dans la question !).

Commentaire :

— env. 15% justifient en citant (à tord) la première représentation vue en cours
— env. 6% répondent que oui c'est possible, et donnent un calcul erronné pour justifier
— env. 30% ne justifient pas ou donnent une justification fausse (par ex : justification basée sur le fait que 0.7 est un multiple de 0.1, et que comme 0.1 n'est pas représentable, alors 0.7 ne l'est pas aussi ; contre-exemple : $5 \times 0.1 = 0.5$)
— env. 20% donnent une justification incomplète. par ex : justification qui se contente de dire qu'il faut une somme infinie pour stocker 0.7, sans montrer la décomposition

④ `11001101` se décompose alors en :
— signe : `1`, donc négatif
— exposant : `10`, soit 2
— mantisse : `01101`, représentant donc `1,01101`,
qui, en multipliant par 2^2, vaut (en binaire) : `101,101`
soit 5.625 ($= 4 + 1 + \frac{1}{2} + \frac{1}{8}$).
La réponse est donc -5.625.

Commentaire :
— très souvent une bonne méthodologie, mais quelques erreurs de calcul sur le chemin
— env. 25% (!!) ont fait au moins une erreur de calcul, ou alors se trompent en recopiant la donnée
— env. 10% se contentent de laisser le résultat tel quel, sans faire la multiplication/somme finale
— env. 5-10% se trompent dans l'exposant, par ex lisent `0b10` comme "1" ou "3", ou font comme si c'était déjà en base 10
— env. 5-10% oublient d'ajouter 1 devant la mantisse
— env. 2.5% oublient le signe

Question 7 Approximation – [15 points]

Le développement limité de la fonction « arc sinus » au voisinage de 0 est :

$$\arcsin(x) = x + \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{2 \cdot 4 \cdot 5} x^5 + \cdots + \frac{1 \cdot 3 \cdot 5 \cdots (2n-1)}{2 \cdot 4 \cdot 6 \cdots (2n) \cdot (2n+1)} x^{2n+1} + o(x^{2n+2})$$

que l'on peut donc utiliser pour calculer une valeur approchée :

$$\begin{aligned} \arcsin(x) &\simeq x + \sum_{i=1}^{N} \frac{1 \cdot 3 \cdot 5 \cdots (2i-1)}{2 \cdot 4 \cdot 6 \cdots (2i)} \cdot \frac{1}{2i+1} \cdot x^{2i+1} \\ &= x + \sum_{i=1}^{N} \left(\frac{x}{2i+1} \prod_{j=1}^{i} \left(\frac{2j-1}{2j} x^2 \right) \right) \end{aligned}$$

Ecrivez une fonction C++ `asin_approx()` qui prend en argument x et N et utilise la formule ci-dessus pour calculer une valeur approchée de $\arcsin(x)$.

Réponse : Voici quelques solutions possibles :

```
double asin_approx(double x, int N = 5)
{
  double somme(x);

  for(int i(1); i <= N; ++i) {
    double produit(1);
    for(int j(1); j <= i; ++j) {
      produit *= x*x * (2.0*j - 1.0) / (2.0*j);
    }

    somme += produit * x / (2.0*i + 1.0);
  }

  return somme;
}
```

mais notez qu'il n'est pas du tout nécessaire d'avoir deux boucles :

```
double asin_approx(double x, int N = 5)
{
  double current_approx(x); // approximation courante
  double powerx(x); // puissance de x (initialisé à x^1 = x)
  double prod(1.0); // produit des (impair divisé par pair)s

  for (int i(1); i <= N; ++i) {
    powerx *= x*x; // powerx représente x^3, x^5 ..., x^(2n+1)
    prod   *= (2.0*i - 1.0) / (2.0*i); // attention à la division entière

    current_approx += prod * powerx / (2.0*i + 1.0);
  }

  return current_approx;
}
```

ou :

```
double asin_approx(double x, int N = 5)
{
  double approx(x);
  double cumul_term(x); // (2n-1)/2n * x^(2n+1)

  for (int i(1); i <= N; ++i) {
    cumul_term *= (2*i-1) * x*x / (2*i);
    approx += cumul_term / (2*i+1);
  }

  return approx;
}
```

La valeur par défaut pour `N` (`5`) est totalement *optionnelle* ici.

Il est évident que l'utilisation de la fonction `asin()` est totalement hors sujet ici (ne calculerait pas l'approximation demandée pour N fixé).

Commentaire : En règle général cet exercice n'a pas été pas un succès, et c'est bien dommage. Un certain nombre d'étudiant(e)s n'ont même rien écrit.

Beaucoup avaient un code partiellement juste :

- Presque tous n'ont pas fait attention à la division entière.
- Trop oublient de remettre l'accumulateur à 1 à chaque itération de boucle.
- Beaucoup ont écrit les équations en écriture algébrique au lieu de C++.
- erreur (ou oubli) dans les initialisation des variables
- oubli du `*=` et/ou `+=` dans le calcul du multiplicateur ou de la somme.

Question 8 – Trouver les erreurs [15 points]

On considère l'*extrait* de programme C++ suivant, qui comporte plusieurs erreurs de programmation de différente nature (syntaxe, déroulement, conception, méthodologie, ...) :

```
1   double f(double x)
2   {
3     13 + 7*x - 2*x*x;
4   }
5
6   double derive(double x, double y, double epsilon)
7   {
8     return (y - x) / epsilon;
9   }
10
11  double df(double x)
12  {
13    const double epsilon = 1e-5;
14    return derive(f(x), f(x+epsilon), epsilon);
15  }
16
17  double d2f(double x)
18  {
19    const double epsilon = 1e-6;
20    return derive(df(x), df(x+epsilon), epsilon);
21  }
22
23  double extremum(double& a)
24  {
25    const double epsilon = 1e-4;
26    const double u(0.0);
27    const double v(0.0);
28    int security(100.0);
29
30    do {
31      u = df(a);
32      v = d2f(a);
33      a -= u / v;
34      --security;
35    } while ((not (security = 1)) and (u/v == epsilon));
36
37    return a;
38  }
```

Ceci n'étant qu'un *extrait*, l'absence de `#include`, `namespace` et `main()` ne sont pas des erreurs.

Indiquez et corrigez toutes les erreurs (directement sur le code).
Expliquez brièvement les erreurs/correction à droite du code ou sur la page supplémentaire au dos.
On ôtera 1 point pour toute indication d'une erreur qui n'en est pas une.

En bonus, vous pouvez également indiquer des suggestions d'amélioration (qui ne sont pas, strictement parlant, des erreurs).

Les six erreurs sont :

- ligne 3 : manque le `return` ;
- ligne 23 : il n'y a aucune raison de passer `a` par référence ;
- lignes 26 et 27 : `u` et `v` ne peuvent pas être `const` ;
- ligne 33 (et 35) : tester la non (preque) nullité de `v` ;
- ligne 35 : `=` au lieu de `==` (sur `security`) ;
- ligne 35 : **jamais** de test d'égalité sur des `double` (`u/v`).

En bonus (améliorations), on peut citer :

- ligne 7–8 : on pourrait vérifier qu'`epsilon` est non nul ;
- lignes 13 et 19 : `epsilon` serait mieux en tant que paramètre avec valeur par défaut ;
- la ligne 28 (`100.0`) **n'**est **pas** une *erreur*, mais ce serait mieux d'écrire `100` ;
- lignes 33 et 35 : éviter le copié-collé de `u/v` (variable `const` intermédiaire).

Commentaire : Souvent signalés : le problème avec l'instruction `return` et le quantificateur `const`. Mais sinon beaucoup d'erreurs manquées et surtout de fausses erreurs inventées (malgré la pénalité annoncée).

Certain(e)s semblent penser qu'il est impossible d'initialiser avec « `=` ». Ce n'était pas voulu et n'a pas été pénalisé (car à peine présenté en cours).

Peu de gens ont remarqué le problème de la division par zéro ou du contrôle d'égalité d'une variable à virgule flottante (leçon I.4 !).
Et beaucoup ont aussi raté le passage par référence (ligne 23).

Nombreux sont celles et ceux pensent qu'il est impossible de définir des variables constantes portant le même nom dans différentes portées (`epsilon`).

Certain(e)s n'aiment pas passer les appels aux fonctions en tant que paramètres.

Certain(e)s ont probablement mal lu la consignes et n'ont pas pris le temps de corriger toutes les erreurs repérées.

Et mettre un `double` (`100.0`) dans un `int` n'est pas une erreur.