

C++ - TP

Debugage (2h)

L'objectif de ce TP est de vous familiariser avec les outils de base de debugage proposés dans l'IDE.

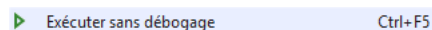
1 Le débogueur

Le *débogueur* est un outil présent dans de nombreux IDE (environnements de développement) afin d'aider les développeurs lors de la réalisation de leurs programmes. Ce débogueur permet de dérouler le programme par étape (bloc par bloc, ligne par ligne,...) et permet de voir le contenu des variables en temps réel. Ainsi le développeur peut voir l'état du programme au cours de son déroulement et détecter plus facilement les erreurs commises.

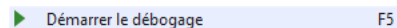
Ce débogueur **doit être** un outil que vous devez utiliser régulièrement pour corriger vos programmes. Dans ce TP, nous chercherons à déboguer des programmes grâce à cet outil. Pour la première partie du TP, orientée tutoriel, importez le programme `exo1_tutoriel.cpp`. Ce programme ne contient pas d'erreurs, mais il nous permettra de voir différentes possibilités du débogueur.

1.1 Exécuter en mode débogueur

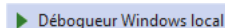
Dans Visual, le lancement d'un projet en mode *normal* se fait habituellement avec (menu *Débuguer*)



Pour lancer l'exécution du programme en mode **débogueur**, il faut utiliser (menu *Débuguer*)



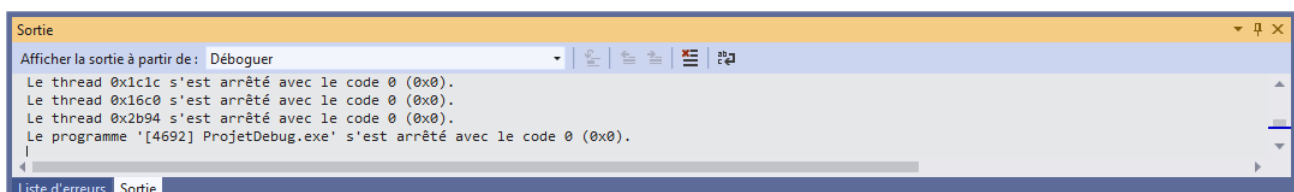
Le raccourci de ce mode d'exécution est donc **F5**, et on retrouve un bouton dans la barre de menu principale



Lancez le programme en mode debug

Telle quelle, l'exécution du programme se déroule comme en mode normal mais vous aurez accès, dans ce mode, aux fonctionnalités de debugage.

Vous devriez également voir que la fenêtre de *sortie* (dans l'IDE) affiche désormais le résultat d'une exécution en mode debugage, avec les différents éléments chargés lors de l'exécution, les threads qui se sont arrêtés et le code de sortie du programme




1.1.1 Barre de menu *Débuguer*

Lors de l'exécution en mode *debug*, une nouvelle barre de menu apparaît dans l'IDE. Les principaux boutons que nous verrons dans les parties suivantes sont les suivants



1.1.2 Arrêter l'exécution en mode *debug*

Pour arrêter l'exécution d'un programme en mode debug, utilisez le bouton  (raccourci Maj+F5). Vous en aurez besoin lors des debugages.

1.1.3 Point d'arrêt

En mode *debug*, il est possible d'arrêter l'exécution du programme en plaçant un *point d'arrêt*. Un point d'arrêt se place **sur une action** (et pas sur une déclaration de variable par exemple).

Pour placer un point d'arrêt, il faut cliquer dans la marge à gauche des numéros de ligne.

- Placez un point d'arrêt sur la ligne 38, comme ci-dessous



```
38 cout << "TP de debugage, ";
39 cout << "tutoriel" << endl;
```

- Lancez l'exécution du programme en mode *debug*.

Lors de l'exécution du programme (en mode *debug*), le programme s'arrêtera sur cette ligne (avant son exécution) (voir ci-dessous). Le programme n'est pas arrêté, il est simplement en stand-by et attend la suite des instructions de la part du développeur.

```
38 cout << "TP de debugage, ";
39 cout << "tutoriel" << endl;
```

Lors de l'exécution en mode debug, une flèche jaune indique la prochaine ligne à exécuter.



Attention : Il n'est pas utile de retirer les points d'arrêt pour exécuter le programme en mode *normal*.

1.1.4 Pas à pas principal

Le *pas à pas principal* (bouton , raccourci F10), permet d'exécuter les instructions une à une.



Avancez jusqu'à la ligne 47 (avant de l'exécuter).



Exécutez un nouveau pas à pas principal.

Avec un pas à pas principal supplémentaire, l'appel de la fonction, ligne 47, se fait complètement (sans arrêt dans la fonction) et le programme passe à la ligne suivante.


```
38 cout << "TP de debugage, ";
39 cout << "tutoriel" << endl;
40
41 //on remplit le tableau
42 for (int i = 0; i < N; ++i) {
43     v = 44 * pow(-1, i) + i;
44     tab[i] = v;
45 }
46 //on affiche le tableau
47 afficherTableau(tab, N);
48
49 //on modifie le tableau
50 modifierTableau(tab, N);
```

Avant exécution ligne 47

```
38 cout << "TP de debugage, ";
39 cout << "tutoriel" << endl;
40
41 //on remplit le tableau
42 for (int i = 0; i < N; ++i) {
43     v = 44 * pow(-1, i) + i;
44     tab[i] = v;
45 }
46 //on affiche le tableau
47 afficherTableau(tab, N);
48
49 //on modifie le tableau
50 modifierTableau(tab, N);
```

Après exécution ligne 47 avec F10

1.1.5 Redémarrage du programme


Sans arrêter l'exécution en mode *debug* il est possible de redémarrer le programme (bouton , raccourci Ctrl+Maj+F5). Le redémarrage peut se faire même si vous avez modifié le code de votre programme (le pro-

gramme est recompilé).



Redémarrez l'exécution du programme.

1.1.6 Pas à pas détaillé

Le *pas à pas détaillé* (bouton , raccourci **F11**), permet également d'exécuter les instructions (de base) une à une, mais lors de l'appel de fonctions, le debugage se fera également dans la fonction.



Avancez jusqu'à la ligne 47 (avant de l'exécuter), avec un pas à pas principal ou détaillé.



Exécutez un pas à pas détaillé.

Avec un pas à pas détaillé lors de l'appel de la fonction, ligne 47, le debugueur va exécuter la fonction ligne par ligne.

```

38 cout << "TP de debugage, ";
39 cout << "tutoriel" << endl;
40
41 //on remplit le tableau
42 for (int i = 0; i < N; ++i) {
43     v = 44 * pow(-1, i) + i;
44     tab[i] = v;
45 }
46 //on affiche le tableau
47 afficherTableau(tab, N);
48
49 //on modifie le tableau
50 modifierTableau(tab, N);
    
```

Avant exécution ligne 47

```

12 void afficherTableau(int t[], int taille) {
13     for (int i = 0; i < taille; ++i) {
14         cout << t[i] << " , ";
15     }
16     cout << endl;
17 }
    
```

Après exécution ligne 47 avec **F11**



Continuez à exécuter la fonction avec en pas à pas détaillé ou principal.

Après la fonction, vous devriez revenir au programme principal, à la ligne 47 (l'appel de la fonction a été exécuté).

```

12 void afficherTableau(int t[], int taille) {
13     for (int i = 0; i < taille; ++i) {
14         cout << t[i] << " , ";
15     }
16     cout << endl;
17 }
    
```

Avant exécution ligne 17

```

38 cout << "TP de debugage, ";
39 cout << "tutoriel" << endl;
40
41 //on remplit le tableau
42 for (int i = 0; i < N; ++i) {
43     v = 44 * pow(-1, i) + i;
44     tab[i] = v;
45 }
46 //on affiche le tableau
47 afficherTableau(tab, N);
48
49 //on modifie le tableau
50 modifierTableau(tab, N);
    
```

Après exécution ligne 17 avec **F10** ou **F11**



Remarque : L'alternance pas à pas principal/détaillé est possible. Tout dépend si vous voulez entrer ou non dans les fonctions rencontrées.



Avancez encore d'un pas pour vous retrouver sur la ligne 50.

```

50 modifierTableau(tab, N);
    
```

1.1.7 Pas à pas sortant




Avance en pas à pas détaillé pour entrer dans la fonction `modifierTableau` (comme ci-dessous).

```

6 void modifierTableau(int t[], int taille) {
7     for (int i = 0; i < taille; ++i) {
8         t[i] += 2;
9     }
10 }

```

Le *pas à pas sortant* permet de sortir directement d'une fonction (bouton , raccourci **Maj+F11**). Cela peut être utile si vous avez terminé une vérification et passer à la suite du programme principal sans avoir à exécuter pas à pas la fonction.



Exécutez un pas à pas sortant pour rejoindre le programme principal.

```

6 void modifierTableau(int t[], int taille) {
7     for (int i = 0; i < taille; ++i) {
8         t[i] += 2;
9     }
10 }

```

Avant exécution pas à pas sortant

```

50 modifierTableau(tab, N);

```

Après exécution pas à pas sortant

1.1.8 Continuer l'exécution

Il est possible de placer plusieurs points d'arrêt dans le programme.




Sans arrêter l'exécution, placez un point d'arrêt sur la ligne 59.

```

49 //on modifie le tableau
50 modifierTableau(tab, N);
51 //on affiche le tableau
52 afficherTableau(tab, N);
53
54 //on calcule la moyenne des éléments
55 moy = moyenne(tab, N);
56 cout << "Moyenne=" << moy << endl;
57
58 //test: 2 appels à une fonction
59 sum = somme(tab, 2) + moyenne(tab, N);
60 cout << sum << endl;

```

Pour rejoindre ce second point d'arrêt, soit vous avancez pas à pas, soit vous pouvez reprendre l'exécution normale jusqu'à un prochain point d'arrêt.

Pour reprendre l'exécution du programme jusqu'au prochain point d'arrêt, vous pouvez utiliser le bouton  **Continuer** (raccourci **F5**).

```

49 //on modifie le tableau
50 modifierTableau(tab, N);
51 //on affiche le tableau
52 afficherTableau(tab, N);
53
54 //on calcule la moyenne des éléments
55 moy = moyenne(tab, N);
56 cout << "Moyenne=" << moy << endl;
57
58 //test: 2 appels à une fonction
59 sum = somme(tab, 2) + moyenne(tab, N);
60 cout << sum << endl;

```

Exécution en cours

```

49 //on modifie le tableau
50 modifierTableau(tab, N);
51 //on affiche le tableau
52 afficherTableau(tab, N);
53
54 //on calcule la moyenne des éléments
55 moy = moyenne(tab, N);
56 cout << "Moyenne=" << moy << endl;
57
58 //test: 2 appels à une fonction
59 sum = somme(tab, 2) + moyenne(tab, N);
60 cout << sum << endl;

```

Après exécution jusqu'au prochain point d'arrêt (**F5**)

1.1.9 Plusieurs appels de fonctions sur une même ligne

Si une ligne contient plusieurs appels de fonctions (la même fonction ou des fonctions différentes), un pas à pas détaillé rentrera dans chaque fonction, les uns après les autres, dans l'ordre.



Exécutez la ligne 59 en pas à pas détaillé. Vous devez constater que vous passez d'abord par la fonction `somme` puis par la fonction `moyenne`.


1.1.10 Exécution jusqu'à une ligne donnée

Il est possible d'exécuter le programme jusqu'à une ligne particulière, sans placer de point d'arrêt.



Redémarrez l'exécution en mode *debug*. Votre programme doit s'arrêter sur le point d'arrêt ligne 38.

Pour choisir jusqu'à quelle ligne exécuter votre programme :

- placez le curseur au-dessus des instructions de la ligne souhaitée,
- le bouton  va apparaître au début de la ligne,
- cliquez sur ce bouton.



Exécutez votre programme directement jusqu'à la ligne 50.

```

38  cout << "TP de debugage, ";
39  cout << "tutoriel" << endl;
40
41  //on remplit le tableau
42  for (int i = 0; i < N; ++i) {
43      v = 44 * pow(-1, i) + i;
44      tab[i] = v;
45  }
46  //on affiche le tableau
47  afficherTableau(tab, N);
48
49  //on modifie le tableau
50  modifierTableau(tab, N);
    
```

Point d'arrêt ligne 38

```

38  cout << "TP de debugage, ";
39  cout << "tutoriel" << endl;
40
41  //on remplit le tableau
42  for (int i = 0; i < N; ++i) {
43      v = 44 * pow(-1, i) + i;
44      tab[i] = v;
45  }
46  //on affiche le tableau
47  afficherTableau(tab, N);
48
49  //on modifie le tableau
50  modifierTableau(tab, N);
    
```

Après utilisation du bouton ligne 50



- Arrêtez l'exécution en mode *debug*.
- Retirez les points d'arrêt.



Attention : Le programme s'arrêtera sur la ligne sauf si un point d'arrêt est rencontré avant (le programme s'arrêtera sur le point d'arrêt).

1.2 Inspecter l'état des variables

Le débogueur permet également de voir l'état des variables en temps réel. Cela permet de voir si le comportement du programme est normal ou si les valeurs que prennent les variables sont incohérentes.

1.2.1 Inspecter ponctuellement une variable



- Mettez un point d'arrêt ligne 44.
- Lancez l'exécution du programme en mode *debug*.
- Placez le curseur de la souris (**sans cliquer**) au-dessus de la variable `v`. Vous devriez voir la valeur de la variable apparaître.
- Idem pour voir l'état du tableau `tab`.

```
42 for (int i = 0; i < N; ++i) {
43     v = 44 * pow(-1, i) + i;
44     t v 44
45 }
```

Curseur au-dessus de v

```
42 for (int i = 0; i < N; ++i) {
43     v = 44 * pow(-1, i) + i;
44     tab[i] = v;
45 }
```

Curseur au-dessus de tab



Remarque : Si vous souhaitez voir seulement `tab[i]` (et pas tout le tableau), vous pouvez surligner la variable choisie (ici `tab[i]`) et placer le curseur au-dessus de la sélection.

1.2.2 Inspecter une variable en continu

Il est également possible de voir l'état des variables en continu dans une fenêtre dédiée. Plusieurs onglets sont disponibles dans cette fenêtre.



- Cette fenêtre devrait déjà être ouverte dans l'IDE (si ce n'est pas le cas : menu *Débuguer* → *Fenêtres* → *Automatique*).
- Allez dans l'onglet *Automatique*.

Cet onglet vous montre l'état des variables actuellement concernées par le programme (les variables `may` et `sum` ne devraient pas apparaître).

Automatique		
Rechercher (Ctrl+E)	Profondeur de recherche : 3	A
Nom	Valeur	Type
N	5	const int
i	0	int
tab	0x00b3f978 {-858993460, -858993460, -858993460, -858993460, -858993460}	int[5]
tab[i]	-858993460	int
v	44	int



- Avancez jusqu'à la ligne 47 pour voir l'évolution de l'affichage (les variables modifiées par la ligne apparaissent en rouge).
- En ligne 47, faites un pas à pas détaillé pour rentrer dans la fonction et voir l'évolution de l'affichage de cet onglet.

1.2.3 Créer ses propres espions

Il est possible de créer ses propres espions, afin de visualiser des variables que l'on choisit, tout au long de l'exécution du programme.



- Redémarrez l'exécution en mode *debug* - Lorsque le programme s'arrête ligne 47, nous allons ajouter un espion sur la variable `v` et sur le tableau `tab`. Plusieurs solutions :
 - Depuis le code
 - Sélectionnez la variable à inspecter
 - *clic droit* sur la sélection → *Ajouter un espion*
 - Depuis la fenêtre d'inspection, onglet *Espion 1*
 - Si l'onglet *Espion 1* n'est pas encore ouvert : menu *Débuguer* → *Fenêtres* → *Espion* → *Espion 1*,
 - Cliquez sur la ligne *Ajoutez un élément à espionner*, puis saisissez le nom de la variable à inspecter.

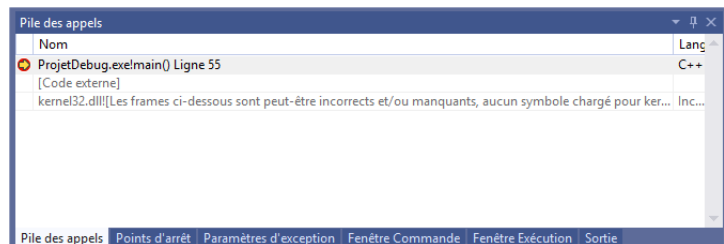


1.3.4 Fenêtre de la pile des appels de fonctions

Cette fenêtre permet de visualiser l'empilement des appels de fonctions. Pour afficher cette fenêtre : menu *Débuguer* → *Fenêtres* → *Pile des appels*.

```
54 //on calcule la moyenne des éléments
55 moy = moyenne(tab, N);
56 cout << "Moyenne=" << moy << endl;
```

Avant appel de la fonction `moyenne`



On est dans le programme principal `main`

```
27 float moyenne(int t[], int taille) {
28     float s = somme(t, taille);
29     float m = s / taille;
30     return m;
31 }
```

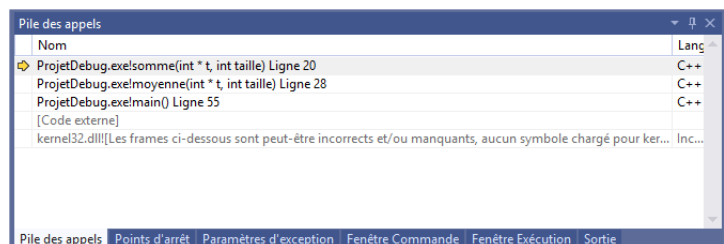
Avant appel de la fonction `somme`



On est dans la fonction `moyenne`

```
19 float somme(int t[], int taille) {
20     float somme = 0.0;
21     for (int i = 0; i < taille; ++i) {
22         somme += t[i];
23     }
24     return somme;
25 }
```

Exécution de la fonction `somme`



On voit la pile des appels

```
27 float moyenne(int t[], int taille) {
28     float s = somme(t, taille);
29     float m = s / taille;
30     return m;
31 }
```

Après l'appel de la fonction `somme`



On est de nouveau dans la fonction `moyenne`

```
54 //on calcule la moyenne des éléments
55 moy = moyenne(tab, N);
56 cout << "Moyenne=" << moy << endl;
```

Après l'appel de la fonction `moyenne`




On est de nouveau dans le programme principal `main`

1.4 Options sur les points d'arrêt

Dans cette section, nous présentons quelques options sur les points d'arrêt. Pour chaque option, il est indiqué comment la mettre en place dans la fenêtre principale du code, mais toutes ces options sont également accessibles via la fenêtre des points d'arrêt (section 1.3.2).

1.4.1 Désactivation d'un point d'arrêt


Au lieu d'ajouter/supprimer sans cesse des points d'arrêt, il peut être utile de simplement les désactiver. Pour cela,

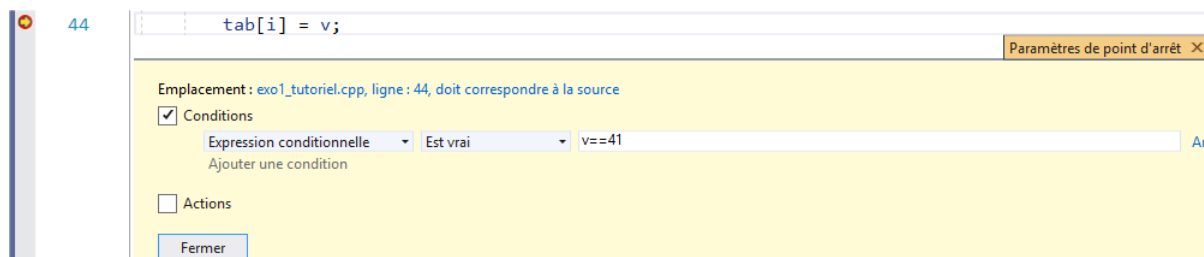
- Survoler le point d'arrêt (sans cliquer sinon il sera supprimé), un menu doit apparaître,
- Utiliser le bouton d'activation/désactivation 

1.4.2 Arrêt sous condition

Il est parfois intéressant d'arrêter le programme sur une ligne donnée, mais sous condition. Exemple, on sait qu'une boucle fonctionne pour les 1000 premières cases d'un tableau mais qu'il y a une erreur lors du 1001ème tour de boucle ; il est inconcevable de faire du pas à pas sur les 1000 premiers tours de boucle.

Pour inclure une condition pour que le programme s'arrête au point d'arrêt,

- Survoler le point d'arrêt (sans cliquer sinon il sera supprimé), un menu doit apparaître,
- Utiliser le bouton de paramétrage, 
- Cocher *Conditions* et remplir les champs demandés.



Plusieurs types de conditions sont possibles, et plusieurs conditions peuvent être combinées. Dans l'exemple présenté ci-dessus, 2 solutions pourraient être proposées : *Expression conditionnelle* - *Est vrai* - $i == 1001$ ou *Nombre d'accès* - $=$ - 1001 .




Question : Dans la fonction *somme*, que vaut la variable *somme* lors de la 3ème itération ? (vous devriez trouver 53.0)

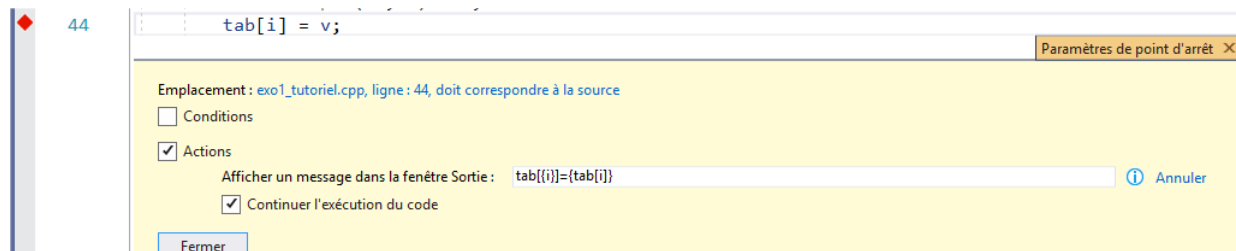
1.4.3 Affichage au point d'arrêt

Il est également possible d'afficher un message dans la fenêtre de sortie (section 1.3.3) lors du passage sur un point d'arrêt.

Pour ajouter un affichage lorsque le programme arrive à un point d'arrêt,

- Survoler le point d'arrêt (sans cliquer sinon il sera supprimé), un menu doit apparaître,
- Utiliser le bouton de paramétrage, 
- Cocher *Actions* et remplir le message. Vous pouvez afficher
 - du texte brut
 - des variables : mettre la variable entre accolades `{ }`
 - des valeurs internes : **\$motclé** (voir l'aide de l'IDE pour les différentes valeurs possibles)

Il est également possible de s'arrêter ou non sur le point d'arrêt.



Sans modifier le code, affichez dans la fenêtre *sortie* les valeurs modifiées du tableau lors de l'appel de la fonction `modifierTableau`.

2 Programmes à debugger

2.1 Calcul de moyenne

On souhaite calculer la moyenne des nombres de 1 à N . On vous propose le code du fichier `debug_exo1_etu.cpp`, avec $N = 7$.



Parcourez le programme pas à pas et remplissez le tableau ci-dessous. En remplissant les valeurs, vous devriez trouver ce qui pose problème.

	valeurs													
i														
<i>somme</i>														
<i>moyenne</i>														

2.2 Suite de Syracuse

La suite de Syracuse d'un nombre entier $N > 0$ est définie par récurrence de la manière suivante :

- $u_0 = N$,
- pour tout $n \geq 0$, $u_{n+1} = \begin{cases} \frac{1}{2}u_n & \text{si } u_n \text{ est pair,} \\ 3u_n + 1 & \text{si } u_n \text{ est impair.} \end{cases}$

Voici quelques exemples de suites pour différentes valeurs de N ($N = 14, 15, 52$).

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}	u_{15}	u_{16}	u_{17}
14	7	22	11	34	17	52	26	13	40	20	10	5	16	8	4	2	1
15	46	23	70	35	106	53	160	80	40	20	10	5	16	8	4	2	1
52	26	13	40	20	10	5	16	8	4	2	1						

Une conjecture (conjecture de Syracuse) affirme que pour tout N , il existe un indice n tel que $u_n = 1$. L'observation de la suite montre que la suite peut s'élever assez haut avant de retomber. L'analyse des suites font penser à la chute chaotique d'un grêlon ou bien à la trajectoire d'une feuille emportée par le vent. De cette observation est né tout un vocabulaire imagé : on parlera du **vol** de la suite. On définit alors plusieurs éléments :

- **le temps de vol** : c'est le plus petit indice n tel que $u_n = 1$. Il est de 17 pour $N = 14, 15$ et de 11 pour $N = 52$.
- **le temps de vol en altitude** : c'est le plus petit indice n tel que $u_{n+1} \leq u_0$. Il est de 0 pour $N = 14, 52$ et de 10 pour $N = 15$.
- **l'altitude maximale** : c'est la valeur maximale de la suite. Il est de 52 pour $N = 14$, 160 pour $N = 15$ et 52 pour $N = 52$.

Nous souhaitons un programme qui permet, à partir d'une valeur N demandée à l'utilisateur, de calculer tous les éléments de la suite de Syracuse jusqu'au premier élément de valeur 1. On souhaite aussi avoir les valeurs des trois éléments définis au-dessus. Pierre Kiroule, étudiant émérite (en devenir), a fourni le code présent dans le fichier `debug_exo2_etu.cpp`. Malheureusement le programme comporte des erreurs.



A faire : On vous demande de :

- débbugger la **partie 1** qui doit afficher tous les éléments de la suite,
- débbugger la **partie 2** qui doit calculer et afficher les 3 paramètres.

Les 2 parties dans la boucle sont indépendantes.

2.3 Plus grand entier divisible par 2 premiers

Le plus grand entier ≤ 100 divisible seulement par 2 nombres premiers est 96 (divisible par 2 et 3, $96 = 2^5 \times 3$). Pour 2 nombres premiers distincts p et q , on note $M(p, q, N)$ le plus grand entier positif $\leq N$ seulement divisible par les 2 nombres premiers p et q (et $M(p, q, N) = 0$ si un tel entier n'existe pas).

Exemples :

$$M(2, 3, 100) = 96,$$

$$M(3, 5, 100) = 75 \text{ (et pas 90 car 90 est divisible par 2, 3 et 5),}$$

$$(2, 73, 100) = 0.$$

On appelle $S(N)$ la somme de tous les $M(p, q, N)$ distincts. Ainsi $S(100) = 2262$.

Quelques exemples :

— $S(100) = 2262$ (30 valeurs à sommer)

les valeurs sommées sont :

$$\begin{array}{llllll} M(2, 3, 100) = 96 & M(2, 19, 100) = 76 & M(2, 43, 100) = 86 & M(3, 17, 100) = 51 & M(5, 11, 100) = 55 \\ M(2, 5, 100) = 100 & M(2, 23, 100) = 92 & M(2, 47, 100) = 94 & M(3, 19, 100) = 57 & M(5, 13, 100) = 65 \\ M(2, 7, 100) = 98 & M(2, 29, 100) = 58 & M(3, 5, 100) = 75 & M(3, 23, 100) = 69 & M(5, 17, 100) = 85 \\ M(2, 11, 100) = 88 & M(2, 31, 100) = 62 & M(3, 7, 100) = 63 & M(3, 29, 100) = 87 & M(5, 19, 100) = 95 \\ M(2, 13, 100) = 52 & M(2, 37, 100) = 74 & M(3, 11, 100) = 99 & M(3, 31, 100) = 93 & M(7, 11, 100) = 77 \\ M(2, 17, 100) = 68 & M(2, 41, 100) = 82 & M(3, 13, 100) = 39 & M(5, 7, 100) = 35 & M(7, 13, 100) = 91 \end{array}$$

— $S(200) = 7838$ (56 valeurs à sommer),

— $S(500) = 49301$ (145 valeurs à sommer),

— $S(1000) = 193408$ (288 valeurs à sommer).

Question : Pouvez-vous déterminer $S(100000)$?

Sarah Croche, étudiante en 1ère année, a proposé le code fourni dans le fichier `debug_exo3_etu.cpp`. Sa proposition est intéressante car elle essaie de limiter au maximum les calculs inutiles afin d'accélérer l'exécution de son programme.



Question : Pouvez-vous vérifier le code qu'elle a donné et le corriger si besoin ?