Information, Calcul, Communication (partie programmation) : Structures de contrôle en C++ (2) : boucles / itérations

Jean-Cédric Chappelier

Laboratoire d'Intelligence Artificielle
Faculté I&C

Objectifs de la leçon d'aujourd'hui

- Rappels :
  - **structures de contrôle** en C++ : boucles, itérations
  - et sur la portée
- Complément sur les structures de contrôle : les **sauts**
- Etudes de cas
- Questions

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Rappel du calendrier

		MOOC	décalage / MOOC	exercices prog. 1h45 Jeudi 8-10	cours prog. 45 min. Jeudi 10-11
1	11.09.25	--	-1	prise en main	Bienvenue/Introduction
2	18.09.25	1. variables	0	variables / expressions	variables / expressions
3	25.09.25	2. if	0	if – switch	if – switch
4	02.10.25	3. for/while	0	for / while	for / while
5	09.10.25	4. fonctions	0	fonctions (1)	fonctions (1)
6	16.10.25		1	fonctions (2)	fonctions (2)
-	23.10.25				
7	30.10.25	5. tableaux (vector)	1	vector	vector
8	06.11.25	6. string + struct	1	array / string	array / string
9	13.11.25		2	structures	structures
10	20.11.25	7. pointeurs	2	pointeurs	pointeurs
11	27.11.25		-	entrées/sorties	entrées/sorties
12	04.12.25		-	erreurs / exceptions	erreurs / exceptions
13	11.12.25		-	révisions	théorie : sécurité
14	18.12.25	8. étude de cas	-	révisions	Révisions
					(ne sont pas sur le MOOC)

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Les différentes structures de contrôle

On distingue 3 types de structures de contrôle :

les branchements conditionnels : *si ... alors ...*

Si $\Delta = 0$
| $x \leftarrow -\frac{b}{2}$
Sinon
| $x \leftarrow \frac{-b-\sqrt{\Delta}}{2}$, $y \leftarrow \frac{-b+\sqrt{\Delta}}{2}$

les boucles conditionnelles : *tant que ...*

Tant que pas arrivé
| avancer d'un pas

Répéter
| poser la question
jusqu'à réponse valide

les itérations : *pour ... allant de ... à ...* , *pour ... parmi ...*

$$x = \sum_{i=1}^{5} \frac{1}{i^2}$$

$x \leftarrow 0$
Pour i de 1 à 5
| $x \leftarrow x + \frac{1}{i^2}$

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Boucles et itérations

Les boucles permettent la mise en œuvre **répétitive** d'un traitement.

La répétition est **contrôlée** par une **condition de continuation**.

- boucles conditionnelles *a priori*

```
while (condition) {
    instructions;
}
```

- boucles conditionnelles *a posteriori*

```
do {
    instructions;
} while (condition);
```

- itérations générales (« à la C »)

```
for (initialisation ; condition ; mise_à_jour)
```

- itérations sur des ensembles (C++11)

```
for (déclaration : ensemble)
```

☞ plus tard (tableaux)

double h(3.0);
while ( )
h modifié
{
h

Rappels sur la portée

La **portée** d'une variable c'est l'ensemble des lignes de code où cette variable est accessible / est définie / existe / a un sens.

- les variables déclarées à l'intérieur d'un bloc sont appelées **variables locales** (au bloc). Elles ne sont accessibles qu'à l'intérieur du bloc.
- les variables déclarées en dehors de tout bloc (même du bloc `main(){}`) seront appelées **variables globales** (au programme). Elles sont accessibles dans l'ensemble du programme.
- en cas d'ambiguïté : résolution à la portée la plus proche.

Conseils :

① Ne jamais utiliser de variables globales (sauf peut être pour certaines constantes).
② Déclarer les variables au plus près de leur utilisation.
③ Evitez d'utiliser le même nom pour des variables différentes.

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Plan

- Rappels :
  - structures de contrôle en C++ : boucles, itérations
  - et sur la portée
- Complément sur les structures de contrôle : les sauts
- Etudes de cas
- Questions

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Sauts : `break` et `continue`

C++ fournit deux instructions prédéfinies, `break` et `continue`, permettant de contrôler de façon plus fine le déroulement d'une boucle.

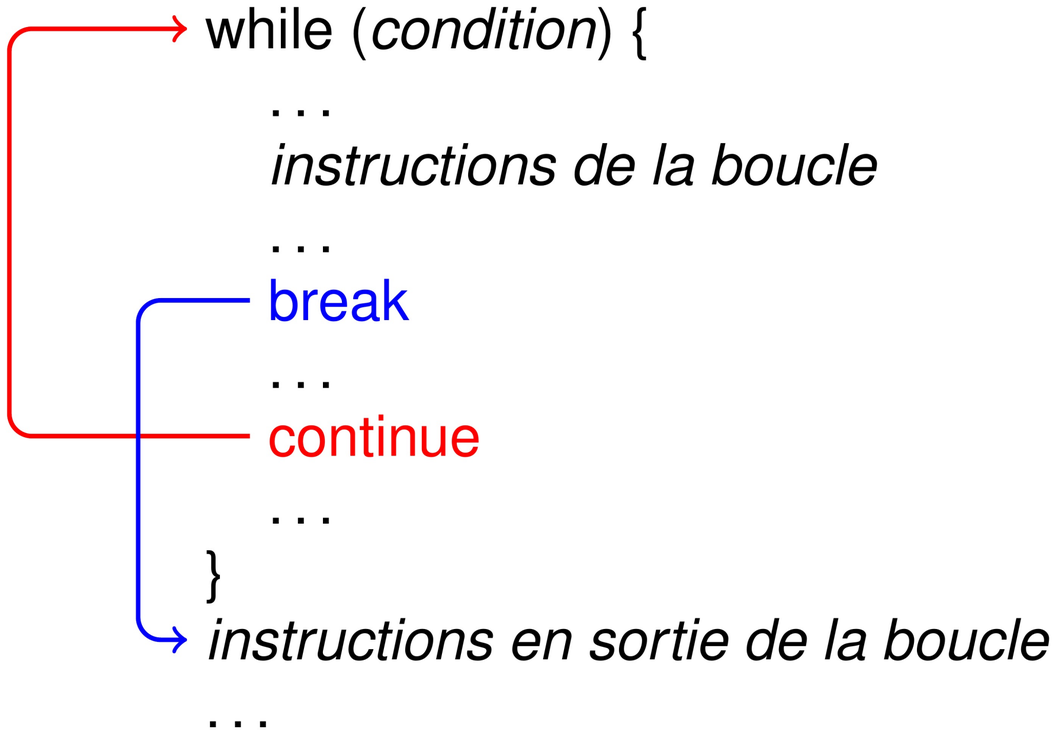
- Si l'instruction `break` est exécutée au sein du bloc intérieur de la boucle, l'exécution de la boucle est interrompue (quelque soit l'état de la condition de contrôle) ;
- Si l'instruction `continue` est exécutée au sein du bloc intérieur de la boucle, l'exécution du bloc est interrompue et la condition de continuation est évaluée pour déterminer si l'exécution de la boucle doit être poursuivie.
  Dans le cas d'un `for` la partie mise à jour est également effectuée (avant l'évaluation de la condition).

Conseil : En toute rigueur on n'aurait pas besoin de ces intructions, et tout bon programmeur évite de les utiliser.

> Pour la petite histoire, un bug lié à une mauvaise utilisation de `break;` a conduit à l'effondrement du réseau téléphonique longue distance d'AT&T, le 15 janvier 1990. Plus de 16'000 usagers ont perdu l'usage de leur téléphone pendant près de 9 heures. 70'000'000 d'appels ont été perdus.
> [P. Van der Linden, *Expert C Programming*, 1994.]

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Instructions `break` et `continue`



Jean-Cédric Chappelier
& Jamila Sam
EPFL

Instruction `break` : exemple

Exemple d'utilisation de `break` :
une **mauvaise** (!) façon de simuler une boucle avec condition d'arrêt

```
while (true) {
   Instruction 1;
   ...
   if (condition d'arrêt)
      break;
}
autres instructions;
```

Question : quelle est la bonne façon d'écrire le code ci-dessus ?

Instruction `break` : exemple

Exemple d'utilisation de `break` :
une **mauvaise** (!) façon de simuler une boucle avec condition d'arrêt

```
while (true) {
  Instruction 1;
  ...
  if (condition d'arrêt)
      break;
}
autres instructions;
```

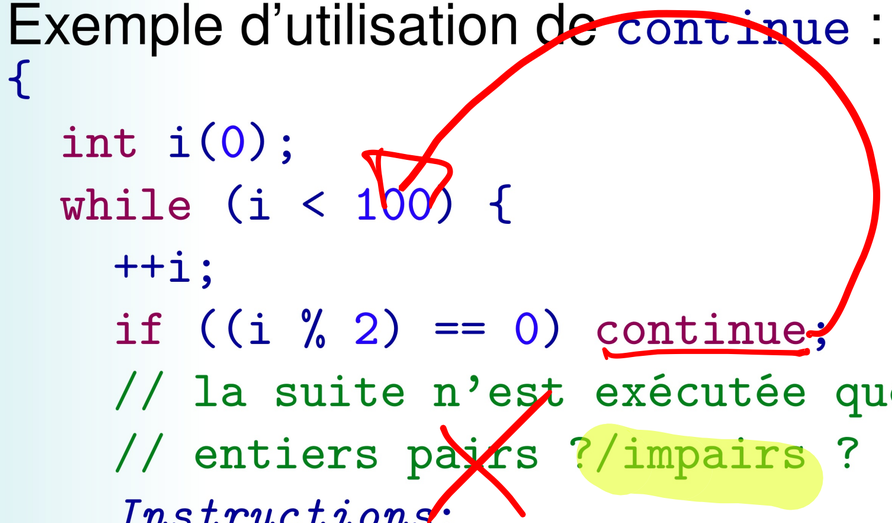
Question : quelle est la bonne façon d'écrire le code ci-dessus ?

```
do {
  Instruction 1;
  ...
} while (not (condition d'arrêt));
autres instructions;
```


Jean-Cédric Chappelier
& Jamila Sam
EPFL

Instruction `continue` : exemple

Exemple d'utilisation de `continue` :



```
{
  int i(0);
  while (i < 100) {
    ++i;
    if ((i % 2) == 0) continue;
    // la suite n'est exécutée que pour les
    // entiers pairs ?/impairs ?
    Instructions;
    ...
  }
}
```

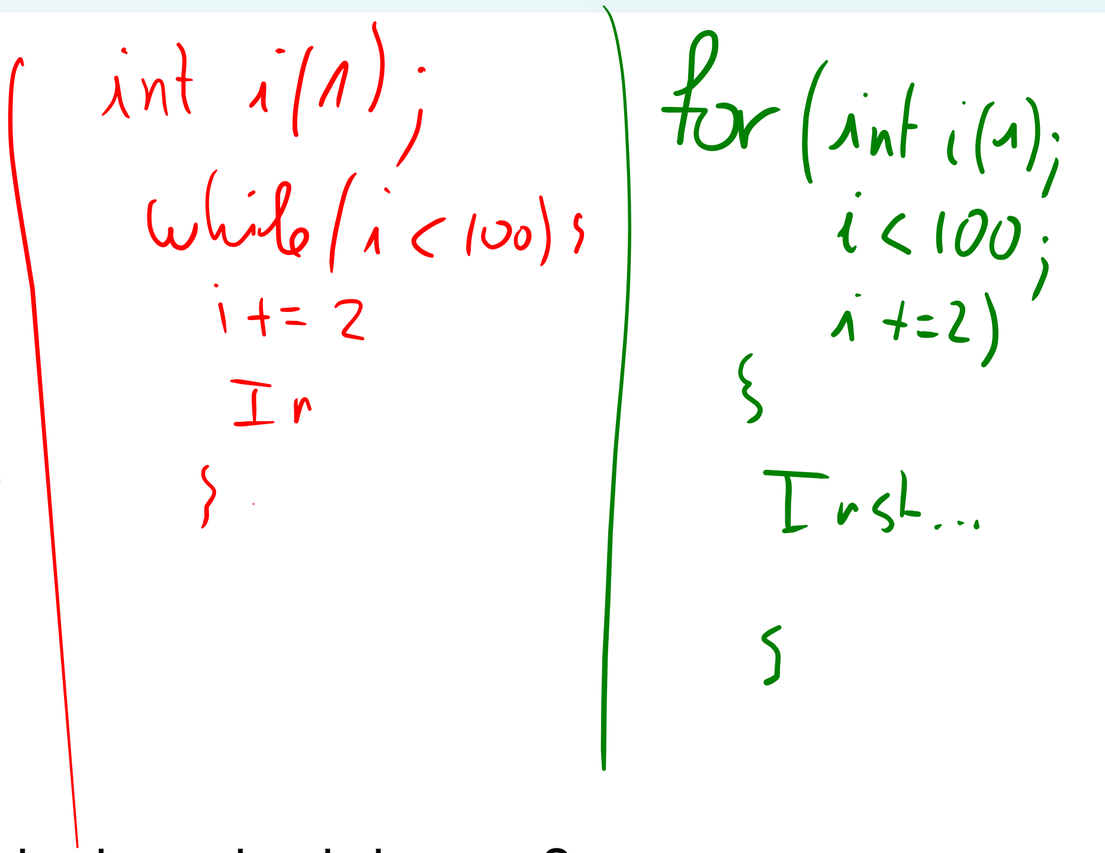
Question : quelle est une meilleure façon d'écrire le code ci-dessus ?
(on suppose que *`Instructions; ...`* ne modifie pas la valeur de `i`)

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Instruction `continue` : exemple

Exemple d'utilisation de `continue` :

```
{
  int i(0);
  while (i < 100) {
    ++i;
    if ((i % 2) == 0) continue;
    // la suite n'est exécutée que pour les
    // entiers impairs !
    Instructions;
    ...
  }
}
```



Question : quelle est une meilleure façon d'écrire le code ci-dessus ?
(on suppose que *Instructions*; ... ne modifie pas la valeur de `i`)

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Instruction `continue` : exemple

Exemple d'utilisation de `continue` :

```
{
  int i(0);
  while (i < 100) {
    ++i;
    if ((i % 2) == 0) continue;
    // la suite n'est exécutée que pour les
    // entiers impairs !
    Instructions;
    ...
  }
}
```

Question : quelle est une meilleure façon d'écrire le code ci-dessus ?
(on suppose que `Instructions; ...` ne modifie pas la valeur de `i`)

```
for (int i(1); i < 100; i += 2) {
  Instructions;
  ...
}
```


Jean-Cédric Chappelier
& Jamila Sam
EPFL

Les structures de contrôle

les branchements conditionnels : *si ... alors ...*

```
if (condition)
    instructions
.........................
if (condition 1)
    instructions 1
...
else if (condition N)
    instructions N
else
    instructions N+1
```

```
switch (expression) {
   case valeur:
       instructions;
       break;
   ...
   default:
       instructions;
}
```

les boucles conditionnelles : *tant que ...*

```
while (condition) {
    instructions;
}
```

```
do {
    instructions;
} while (condition);
```

les itérations : *pour ... allant de ... à ...* , *pour ... parmi ...*

```
for (initialisation ; condition ; increment)
    instructions
```

```
for (déclaration : valeurs)
    instructions
```

les sauts : `break;` et `continue;`

Note : `instructions` représente une instruction élémentaire ou un bloc.

`instructions;` représente une suite d'instructions élémentaires.

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Etudes de cas

- lien avec ICC : début du calcul des $\binom{n}{k}$ (version programmation dynamique)
- jeu de devinette : trouver (par dichotomie) le nombre imaginé

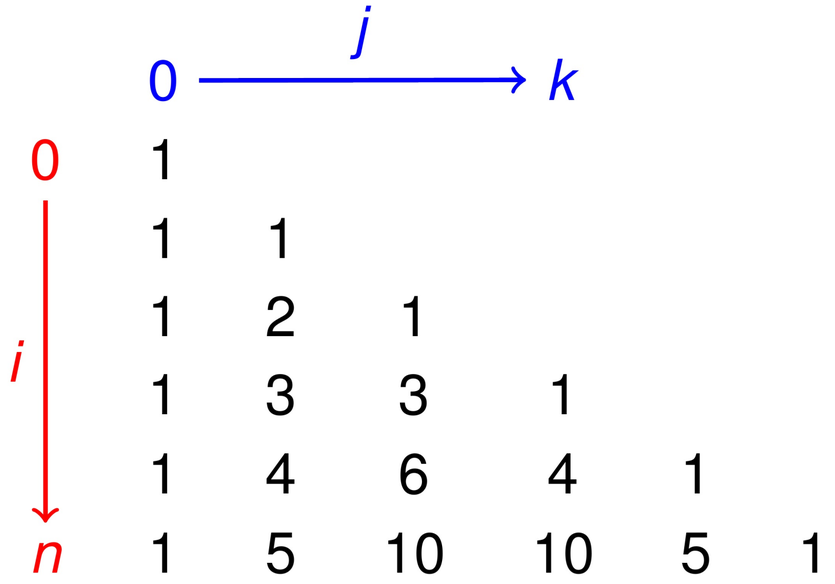
Jean-Cédric Chappelier
& Jamila Sam
EPFL

Etude de cas : les $\binom{n}{k}$

Comment commencer le calcul des $\binom{n}{k}$ tel que vu en cours ICC ?
(version « programmation dynamique » ; on ne fait ici que le début, nous continuerons lorsque nous aurons vu les tableaux en C++)

Rappel du problème :

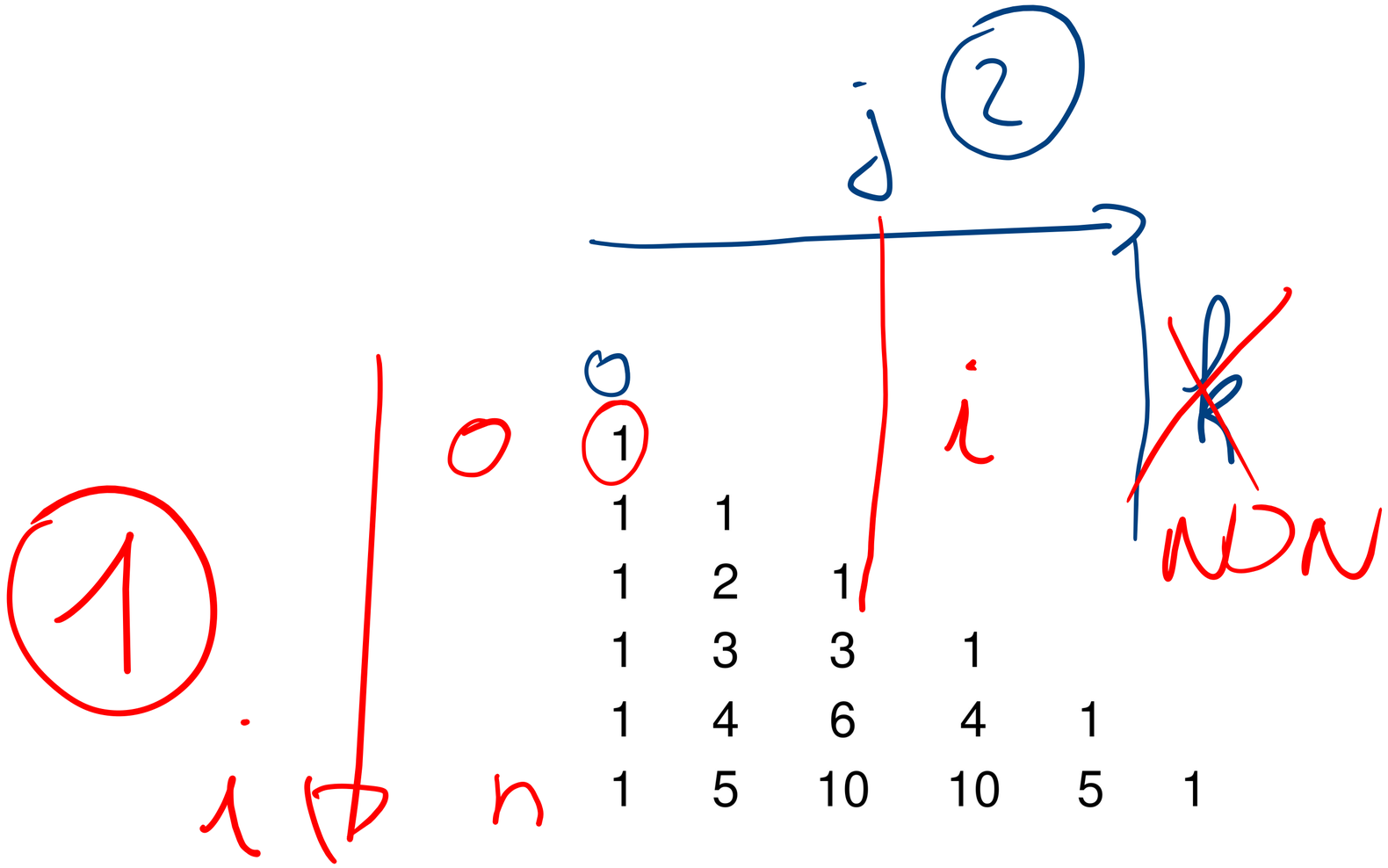
- on nous donne n et k (corrects)
- on veut calculer $\binom{n}{k}$ par deux boucles sur le « triangle de Pascal » :



Jean-Cédric Chappelier
& Jamila Sam

Etude de cas : les $\binom{n}{k}$

- On va donc commencer par



Etude de cas : les $\binom{n}{k}$

- On va donc commencer par une boucle en *i* (de 1 en 1) :
  `for (int i(...); i ...; ++i)`
- Dans laquelle on aura une boucle en *j* (de 1 en 1) :
  `for (int j(...); j ...; ++j)`

$\binom{5}{2}$

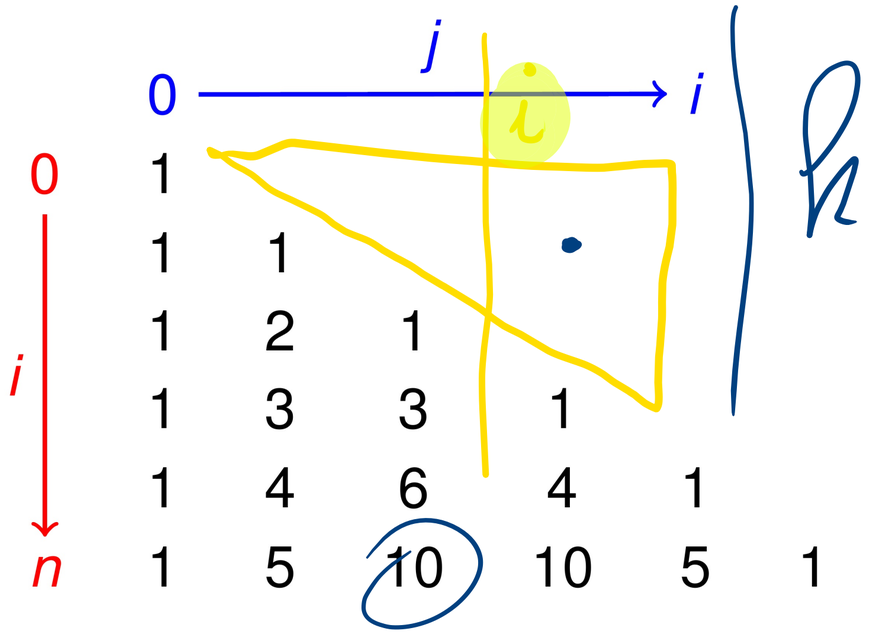
Quelles bornes pour *i* ?
☞ entre 0 et *n*

Quelles bornes pour *j* ?
☞ entre 0 et *i*

On a donc à ce stade :

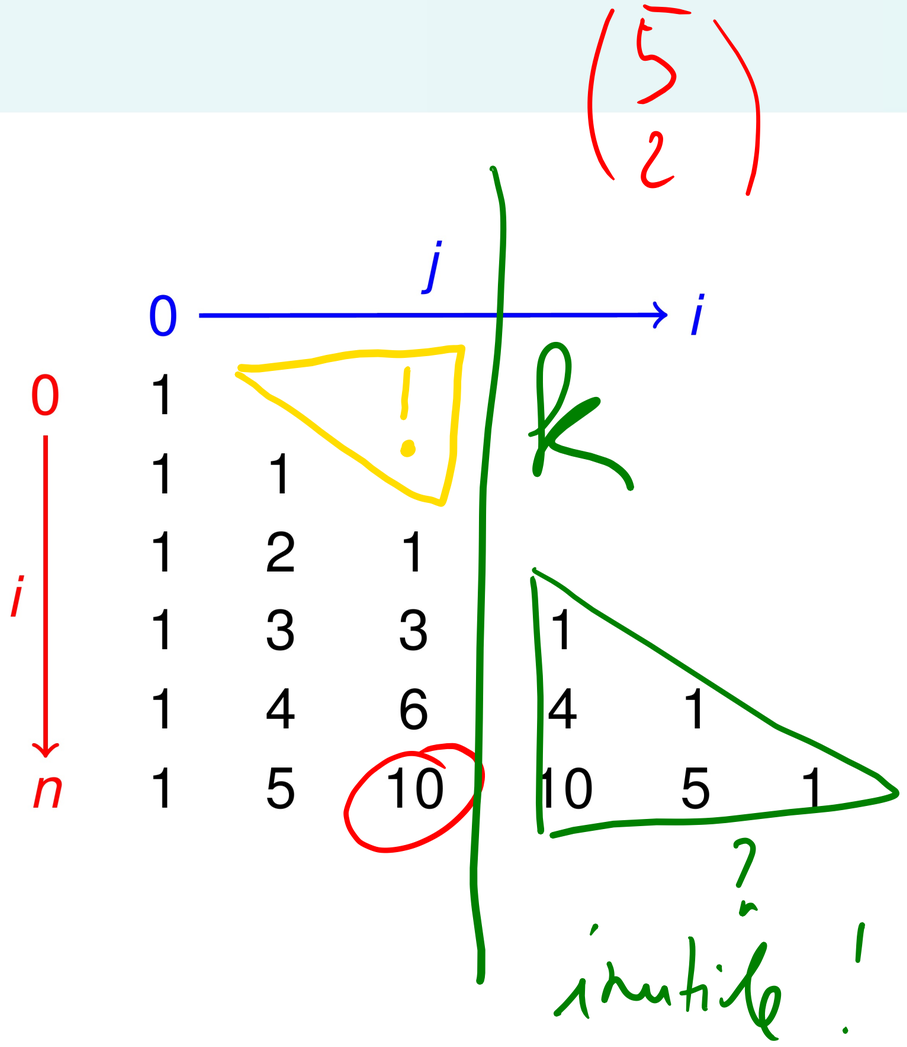
```
for (int i(0); i <= n; ++i) {
  for (int j(0); j <= i; ++j) {

  }
 }
```



Etude de cas : les $\binom{n}{k}$

Peut-on faire mieux ?



Jean-Cédric Chappelier
& Jamila Sam
EPFL

Etude de cas : les $\binom{n}{k}$

Peut-on faire mieux ?

☞ Oui : il **n'**est **pas** nécessaire d'aller au delà de k pour la boucle en j

Donc veut donc que j soit inférieur à i, mais sans non plus dépasser k.

Comment cela s'écrit-il ?

```
(j <= i) or (j <= k)
```

ou bien

```
(j <= i) and (j <= k)
```

?

Jean-Cédric Chappelier
& Jamila Sam
EPFL

`or` ou `and` ?

Il est souvent difficile de correctement choisir entre la conjonction `and` et la disjonction `or`. Quelques pistes :

- tout d'abord que veut-on : que la condition soit vraie ou qu'elle soit fausse ?
  - ☞ Ici on veut *continuer* tant qu'elle est *vraie*
- parfois il est plus facile de passer par la contraposée (sa négation) :
  - ☞ Ici on veut *s'arrêter* dès que la condition est *fausse* ;
    dès que j est plus grand que i ou k (...donc sa négation : et)
- tester pour un cas ambigu.
  - ☞ Ici : tester dans un cas où justement j est entre k et i pour i plus grand que k (puisque c'est bien ces cas là que nous voulons optimiser)
    dans ce cas ($k < j < i$) :
    - `(j <= i) or (j <= k)` est vraie (`true or false`), la boucle continuera donc ;
      ce qui n'est pas ce que nous voulons ;
    - `(j <= i) and (j <= k)` est fausse (`true and false`), la boucle aura donc été arrêtée ;
      ce que nous voulons.
- ☞ Ici c'est donc bien « `(j <= i) and (j <= k)` »

Etude de cas : les $\binom{n}{k}$

On obtient donc :

```
for (int i(0); i <= n; ++i) {
  for (int j(0); (j <= i) and (j <= k); ++j) {
    // ...
  }
}
```

On pourrait aussi écrire (avec `#include <algorithm>`) :

```
for (int i(0); i <= n; ++i) {
  for (int j(0); j <= min(i, k); ++j) {
    // ...
  }
}
```

Reste à voir quoi mettre dans la boucle...
...ce que nous aborderons plus tard une fois vus les tableaux.

Jean-Cédric Chappelier
& Jamila Sam
EPFL