

# Jeu du 421 [it07] - Exercice

Bruno Adam, Karine Zampieri, Stéphane Rivière

Unisciel  algoprogram  Version 17 mai 2018

## Table des matières

<b>1</b>	<b>Énoncé</b>	<b>2</b>
<b>2</b>	<b>Algorithmique, Programmation</b>	<b>3</b>
2.1	Classement des combinaisons . . . . .	3
2.2	Nombre de jetons et valeur d'une combinaison . . . . .	4
2.3	Tour d'un joueur . . . . .	6
2.4	Comparaison des performances . . . . .	8
2.5	Mise en place du tout . . . . .	9

## C++ - Jeu du 421 (TP)



**Mots-Clés** Schéma itératif ■

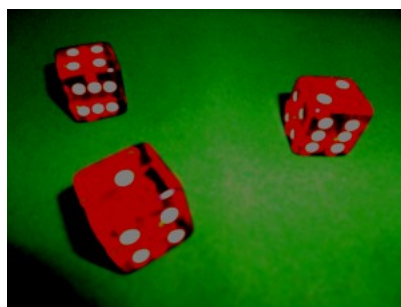
**Requis** Structures de base, Structures conditionnelles, Algorithmes paramétrés, Structures répétitives, Schéma itératif ■

**Difficulté** ●●○ (2 h à 2 h 30) ■



### Objectif

Cet exercice programme le jeu du 421 entre deux joueurs.

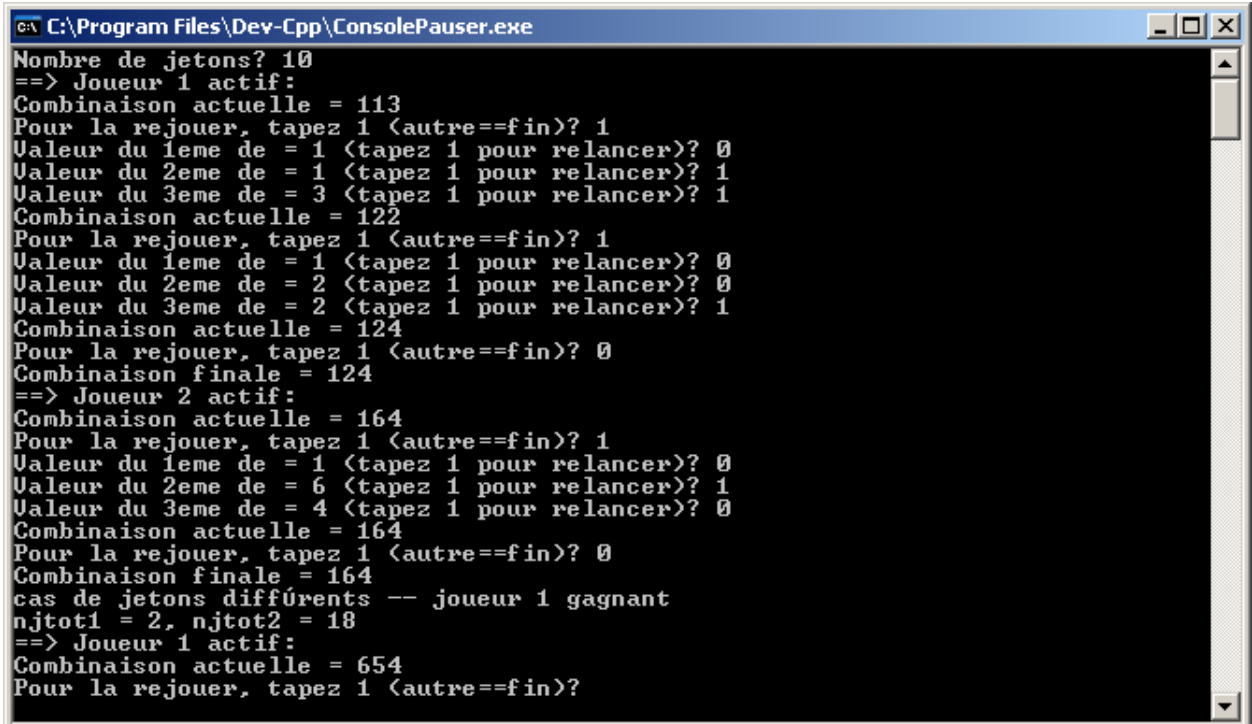


# 1 Énoncé

## Jeu du 421

Il se joue entre deux adversaires qui disposent de 3 dés à 6 faces et du même nombre de jetons chacun. A chaque tour de jeu, l'objectif de chaque joueur consiste à se débarrasser de quelques-uns de ses jetons en faveur de son adversaire en lançant les dés. Le jeu s'arrête dès qu'un des deux joueurs n'a plus de jetons.

## Exemple d'exécution



```
C:\Program Files\Dev-Cpp\ConsolePauser.exe
Nombre de jetons? 10
==> Joueur 1 actif:
Combinaison actuelle = 113
Pour la rejouer, tapez 1 <autre==fin>? 1
Valeur du 1eme de = 1 <tapez 1 pour relancer>? 0
Valeur du 2eme de = 1 <tapez 1 pour relancer>? 1
Valeur du 3eme de = 3 <tapez 1 pour relancer>? 1
Combinaison actuelle = 122
Pour la rejouer, tapez 1 <autre==fin>? 1
Valeur du 1eme de = 1 <tapez 1 pour relancer>? 0
Valeur du 2eme de = 2 <tapez 1 pour relancer>? 0
Valeur du 3eme de = 2 <tapez 1 pour relancer>? 1
Combinaison actuelle = 124
Pour la rejouer, tapez 1 <autre==fin>? 0
Combinaison finale = 124
==> Joueur 2 actif:
Combinaison actuelle = 164
Pour la rejouer, tapez 1 <autre==fin>? 1
Valeur du 1eme de = 1 <tapez 1 pour relancer>? 0
Valeur du 2eme de = 6 <tapez 1 pour relancer>? 1
Valeur du 3eme de = 4 <tapez 1 pour relancer>? 0
Combinaison actuelle = 164
Pour la rejouer, tapez 1 <autre==fin>? 0
Combinaison finale = 164
cas de jetons différents -- joueur 1 gagnant
njtot1 = 2, njtot2 = 18
==> Joueur 1 actif:
Combinaison actuelle = 654
Pour la rejouer, tapez 1 <autre==fin>? 0
```

## Objectif

Programmer une version du jeu.

...(suite page suivante)...

## 2 Algorithmique, Programmation

### 2.1 Classement des combinaisons

Afin de **comparer efficacement** le jet des trois dés des deux joueurs, nous allons ordonner les combinaisons par ordre **décroissant**.



Écrivez le **profil** d'une procédure `permuter2i(a,b)` qui échange les contenus de deux entiers `a` et `b`.

#### Orientation

Les paramètres formels `a` et `b` sont des paramètres mixtes **Donnée/Résultat**. En effet, ils ont des valeurs **avant** l'appel et ils seront **modifiés** lors de l'échange.



Écrivez le corps de la procédure.

#### Rappel de cours

Pour permuter deux variables, il faut passer par une variable intermédiaire.



Écrivez une procédure `decroitre2i(a,b)` qui classe **deux** entiers `a` et `b` par ordre **décroissant**, c.-à-d. qu'à l'issue de la procédure, `a` doit contenir le plus grand entier et `b` le plus petit de `(a,b)`.



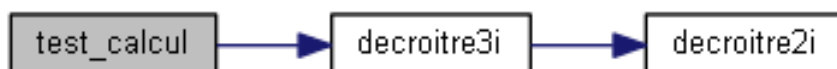
Déduisez une procédure `decroitre3i(a,b,c)` qui classe **trois** entiers `a`, `b` et `c` par ordre décroissant, en appelant **trois** fois la procédure `decroitre2i` :

- Classez `a` et `b` en ordre décroissant.
- Puis classez `b` et `c` en ordre décroissant.
- Puis classez `a` et `b` en ordre décroissant.



Écrivez une procédure `test_calcul` qui saisit trois entiers, les classe par ordre décroissant puis les affiche. Affichez l'invite :

Trois entiers?





Testez. Exemples d'exécution :

```
Trois entiers? 3 2 6
==> 6 3 2
```

```
Trois entiers? 3 2 1
==> 3 2 1
```

## 2.2 Nombre de jetons et valeur d'une combinaison



### Attention

Dans tout ce problème, la **combinaison**  $(a,b,c)$  du jet de trois dés est classée par **ordre décroissant**, c.-à-d.  $a \geq b \geq c$ .



### C/C++

Écrivez les fonctions qui suivent avant la procédure de test `test_calcul`.



Une **tierce** est une combinaison de trois dés qui se suivent.

Écrivez une fonction `tierce(a,b,c)` qui teste et renvoie `Vrai` si une combinaison  $(a,b,c)$  est une tierce, `Faux` sinon. Exemple :

```
tierce(3,2,1) ==> Vrai
tierce(5,3,2) ==> Faux
```



Un **brelan** est une combinaison de trois dés identiques.

Écrivez une fonction `brelan(a,b,c)` qui teste et renvoie `Vrai` si une combinaison  $(a,b,c)$  est un brelan, `Faux` sinon.



Un **421** est une combinaison gagnante : un dé avec 4, un autre avec 2 et un avec 1.

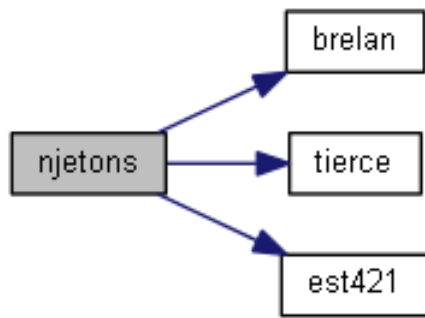
Écrivez une fonction `est421(a,b,c)` qui teste et renvoie `Vrai` si une combinaison  $(a,b,c)$  est un 421, `Faux` sinon.



Le **nombre de jetons** correspondant à la combinaison obtenue est calculé comme suit :

- A la combinaison gagnante 421 : on associe 8 jetons.
- Pour un brelan : 5 jetons.
- Pour une tierce : 2 jetons.
- Pour toutes les autres combinaisons dites mineures : 1 jeton.

Écrivez une fonction `njetons(a,b,c)` qui calcule et renvoie le nombre de jetons d'une combinaison  $(a,b,c)$ .



La **valeur d'une combinaison** est définie par :

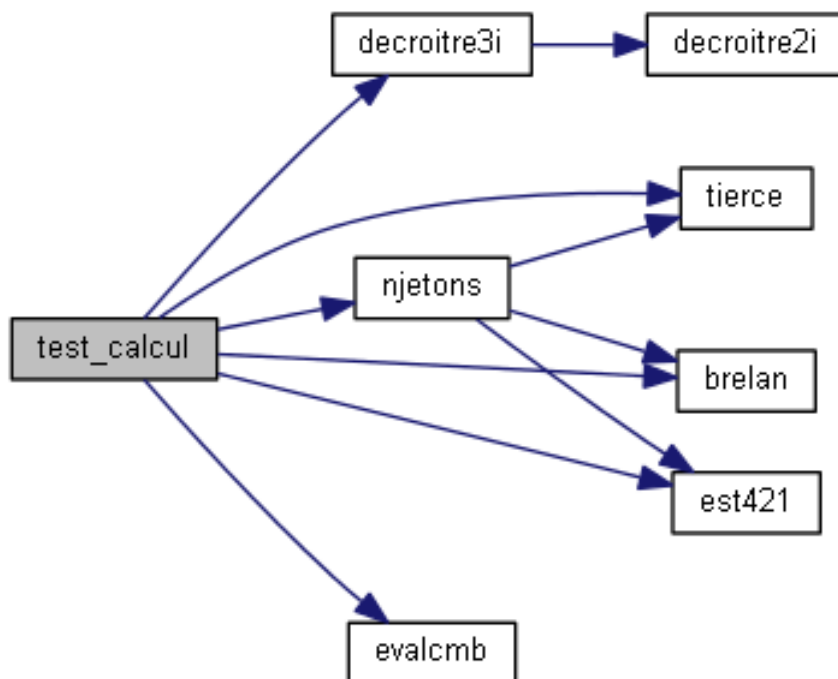
$$s = 100a + 10b + c \text{ avec } a \geq b \geq c$$

Écrivez une fonction `evalcmb(a,b,c)` qui calcule et renvoie la valeur d'une combinaison `(a,b,c)`. Exemple :

```
evalcmb(6,5,1) ==> 100*6+10*5+1 = 651
```



**Complétez** votre procédure `test_calcul` de sorte à tester toutes les fonctions de ce problème.



Testez. Exemple d'exécution :

```
Trois entiers? 4 5 3
==> 5 4 3
tierce = 1
brelan = 0
est421 = 0
```

```
njetons = 2
evalcmb = 543
```

(Le C/C++ affiche 1 pour **Vrai** et 0 pour **Faux**.)

## 2.3 Tour d'un joueur

Chaque tour d'un joueur se déroule en **un, deux ou trois lancers** de dé(s).

Au cours du premier lancer, le joueur jette les trois dés. Si le résultat lui convient, il peut décider de la garder ; sinon il peut décider de rejeter 1, 2 ou les 3 dés, en un ou deux coups supplémentaires (on ne peut pas lancer les dés plus de trois fois).

Une fois que le nombre autorisé de lancers de dés a été effectué ou que le joueur ait décidé de garder la combinaison, on calcule le nombre de jetons et la valeur de la combinaison.

Voici un exemple :

```
D:\Z_XALGOPROG\algorithme2\s15iteratif\00slexec-cpp\pg-jeu421A1c.exe
Combinaison actuelle = 665
Pour la rejouer, tapez 1 <autre==fin>? 1
Valeur du 1eme de = 6 <tapez 1 pour relancer>? 0
Valeur du 2eme de = 6 <tapez 1 pour relancer>? 0
Valeur du 3eme de = 5 <tapez 1 pour relancer>? 1
Combinaison actuelle = 665
Pour la rejouer, tapez 1 <autre==fin>? 1
Valeur du 1eme de = 6 <tapez 1 pour relancer>? 0
Valeur du 2eme de = 6 <tapez 1 pour relancer>? 0
Valeur du 3eme de = 5 <tapez 1 pour relancer>? 1
Combinaison actuelle = 666
Pour la rejouer, tapez 1 <autre==fin>? 0
Combinaison finale = 666
njetons = 5, evalcmb = 666
Appuyez sur une touche pour continuer...
```



Écrivez une fonction `jetDe` qui renvoie la valeur d'un lancer de dé, c.-à-d. un entier pseudo-aléatoire dans `[1..6]`.

### Outil C++

La fonction `rand()`, définie dans la bibliothèque `<random>`, renvoie un entier pseudo-aléatoire positif ou nul. Utilisez le modulo pour projeter l'entier dans l'intervalle souhaité.

### Outil C++

L'initialisation du générateur de nombres pseudo-aléatoires (ici avec l'horloge système) s'effectuera dans le **programme principal** avec l'instruction :

```
srand(time(NULL));
```

La procédure `srand` est définie dans la bibliothèque `<random>` et la fonction `time` dans la bibliothèque `<ctime>`.



Écrivez une procédure `relancer(de,n)` qui affiche la valeur du dé `de` (entier) numéro `n`, puis demande et relance si c'est le cas. Affichez (où `[x]` désigne le contenu de `x`) :

Valeur du [n]eme de = [de] (tapez 1 pour relancer)?

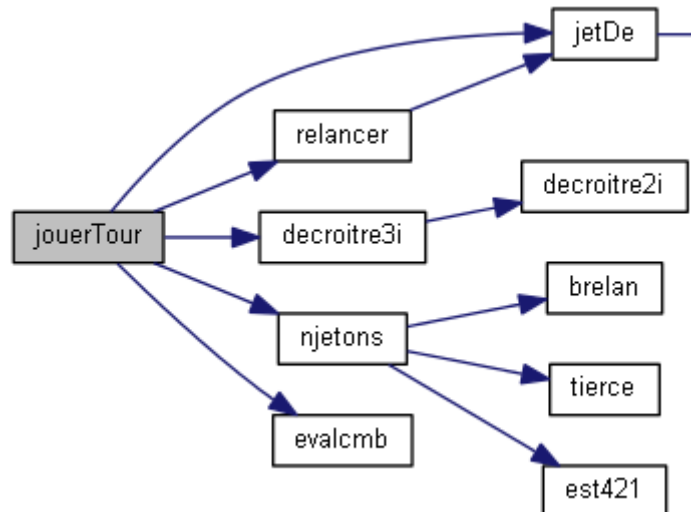


### Aide simple

Affichez le message puis saisissez la réponse de l'utilisateur dans un entier. Si la valeur de celui-ci vaut 1, alors relancez et **actualisez** la valeur du dé `de`.



Écrivez une procédure `jouerTour(nj,s)` qui effectue un tour de jeu d'un joueur. La procédure doit **restituer** le nombre de jetons du joueur actif dans `nj` (entier) **ainsi que** la valeur de sa combinaison dans `s` (entier).



### Aide simple

Les paramètres formels `nj` et `s` sont donc des paramètres **Résultat**.

### Aide méthodologique

Le principe de cette procédure est le suivant :

- Lancez les trois dés `d1`, `d2`, `d3`.
- Affichez la combinaison obtenue.
- Initialisez le nombre de lancers : `nlancers <-- 1`.
- Demandez au joueur s'il veut garder ou relancer la combinaison obtenue.
- **TantQue** le joueur veut relancer Et `nlancers <= 3` Faire
  - `relancer(d1,1)` et de même `d2` puis `d3`.
  - Incrémentez le nombre de lancers `nlancers`.

- Affichez la nouvelle combinaison.
- Demandez au joueur s'il veut garder ou relancer la combinaison obtenue.
- Classez `d1`, `d2`, `d3` par ordre décroissant.
- Calculez le nombre de jetons `nj` (appel de la fonction `njetons(...)`).
- Calculez la valeur `s` (appel de la fonction `evalcmb(...)`).



Écrivez une procédure `test_tour` qui appelle la procédure `jouerTour` puis affiche les calculs.



Testez.

## 2.4 Comparaison des performances

Soient `nj` le nombre de jetons et `s` la valeur de la combinaison du joueur actif. Pour comparer les « performances » des deux joueurs, on distinguera trois cas, dans cet ordre :

1. **Égalité des combinaisons** (même `s`) : il y a `MATCHNUL` entre les deux joueurs et un autre tour est entamé.
2. Sinon **inégalité du nombre de jetons** (`nj` différents) : celui qui a obtenu le plus grand nombre de jetons gagne le tour et donne ce nombre de jetons à l'adversaire.
3. Sinon (cas d'égalité du nombre de jetons `nj` et valeur différente de la combinaison) : celui qui a obtenu la valeur de combinaison la plus grande gagne le tour et donne le nombre de jetons de la combinaison `nj` à l'adversaire.

### Exemple

En cas d'égalité de jetons :

- Si les deux joueurs ont obtenu un brelan, donc 5 points, (exemple (4, 4, 4) et (6, 6, 6)) alors celui qui a obtenu le plus grand brelan (ici  $s = 666$ ) va donner 5 jetons à son adversaire.
- Si chacun des deux joueurs a obtenu une tierce, donc 2 points, (exemple (1, 2, 3) et (5, 6, 4)), alors celui qui a obtenu la plus grande tierce (ici  $s = 654$ ) va donner 2 jetons à son adversaire.
- Si les deux joueurs ont obtenu une combinaison mineur, (exemple (4, 2, 6) et (3, 1, 6)), alors celui qui a obtenu la plus grande des deux combinaisons (ici  $s = 642$ ) va donner 1 jeton à son adversaire.



**Cas 2 :** Écrivez une procédure `traiterJetonsDiff(nj1,nj2,njtot1,njtot2)` qui traite le cas de jetons différents des joueurs `nj1` (entier) et `nj2` (entier) et actualise leurs nombre total de jetons dans `njtot1` (entier) et dans `njtot2` (entier).

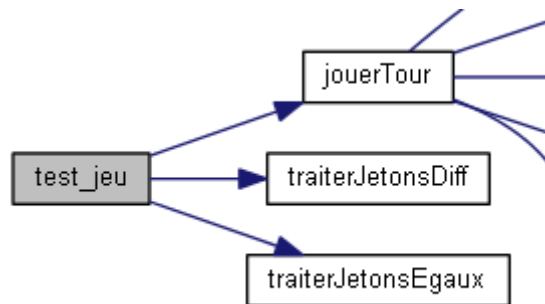


**Cas 3 :** Écrivez une procédure `traiterJetonsEgaux(nj,s1,s2,njtot1,njtot2)` qui traite le cas de jetons égaux `nj` (entier) et valeurs de combinaison différentes des joueurs `s1` (entier) et `s2` (entier) et qui actualise leurs nombre total de jetons dans `njtot1` (entier) et dans `njtot2` (entier).



## 2.5 Mise en place du tout

Ce problème réalise le jeu du 421.



Écrivez une procédure `test_jeu` qui saisit le nombre de jetons à donner à chacun des deux joueurs dans `njtot1` (entier). Affichez l'invite :

Nombre de jetons?



Affectez ce même nombre `njtot1` à `njtot2` (même nombre de jetons au départ).



Tant que `njtot1 > 0` Et `njtot2 > 0`, la procédure :

- Affiche « ==> Joueur 1 actif » et le fait `jouerTour(nj1,s1)`.
- Affiche « ==> Joueur 2 actif » et le fait `jouerTour(nj2,s2)`.
- Effectue les tests en fonction de `s1`, `s2`, `nj1`, `nj2` et calcule les nouveaux scores dans `njtot1` et dans `njtot2`.
- Affiche les nouveaux scores `njtot1` et `njtot2`.



Testez.