

*IUT d'Annecy, 9 rue de l'Arc en Ciel, 74940 Annecy*

Télécharger et décompresser le squelette du TP :

```
wget https://github.com/IUT-GEII-Annecy/squelettes/releases/download/branch-2025/tp5.zip
unzip tp5.zip
rm tp5.zip
```

## Sommaire

<b>1 Première factorisation</b>	<b>3</b>
1.1 Premières fonctions . . . . .	3
1.2 Généralisation des fonctions . . . . .	3
<b>2 Pierre, feuille, ciseaux</b>	<b>4</b>
<b>3 Le retour des Tacos</b>	<b>4</b>

## Un peu de théorie



### Concevoir une fonction

Pour être bien conçue, une fonction doit respecter les critères suivants :

- ne doit faire qu'une seule tâche (principe de responsabilité unique) ;
  - ◊ Rend le code explicite,
  - ◊ Facilite la maintenance,
  - ◊ Facilite les tests unitaires.
  - ◊ Favorise la réutilisabilité.
- être la plus générique possible :
  - ◊ Cela signifie qu'elle doit être la plus indépendante possible du contexte dans lequel elle est utilisée.
- avoir un nom explicite qui reflète son rôle ;
- avoir des paramètres clairs et pertinents ;
- avoir un type de retour approprié.



### Point méthodologique

Avant de se lancer dans le codage, il est important de bien réfléchir à la conception des fonctions. Voici quelques étapes à suivre :

- Identifier les différentes tâches dans le code existant ;
- Regrouper les tâches similaires ou liées ;
- Généraliser les tâches identifiées
- Définir les fonctions en fonction des tâches identifiées ;
- Déterminer les paramètres nécessaires pour chaque fonction ;

- Définir le type de retour pour chaque fonction.

## 1 Première factorisation

Dans cette partie, on propose de factoriser le code d'un duel de dé entre plusieurs joueurs.

### 1.1 Premières fonctions

#### Manipulation 1 : Test du programme

**Étape 1** Tester le code duel

**Étape 2** Lire l'architecture du programme

On propose de découper ce code selon les fonctions suivantes :

**Lancer\_de** : Simule le lancement d'un dé

**Arguments** : Aucun

**Valeur de retour** : Le résultat du lancer de dé

**Afficher\_gagnant** : Affiche le gagnant selon les scores donnés en entrée.

**Arguments** : les scores des joueurs

**Valeur de retour** : Aucune

**Comportement annexe** : Affiche le gagnant sur la console.

Le programme utilisera alors ces fonctions tout en conservant le même comportement :

1. Deux appels de la fonction `lancer_de`
2. Appel de la fonction `Afficher_gagnant`

#### Manipulation 2 : Factorisation du programme

**Étape 3** Déclarer les fonctions en écrivant les prototypes en début de fichier.

**Étape 4** Définir les fonctions correspondantes en fin de fichier

**Étape 5** Modifier la fonction `main` pour qu'elle appelle les fonctions

**Étape 6** Faire vérifier par l'enseignant.

### 1.2 Généralisation des fonctions

On propose à présent d'améliorer les fonctions précédentes pour les rendre plus générale. Dans tous les cas, l'idée est de conserver le même fonctionnement mais de **généraliser** les fonctions pour les rendre les plus utiles possibles. Les modifications proposées sont les suivantes :

**lancer\_de** Cette fonction doit maintenant permettre de lancer un dé avec un nombre de face variable. On pourrait même proposer le tirage d'un nombre compris entre deux bornes données en paramètres.

**Afficher\_gagnant** Cette fonction doit maintenant afficher le gagnant parmi un nombre quelconque de joueurs.

#### Manipulation 3 : Généralisation des fonctions

**Étape 7** Modifier les fonctions pour obtenir les généralisations présentées ci-dessus.

**Étape 8** Faire vérifier par l'enseignant.

## 2 Pierre, feuille, ciseaux

On vous propose de coder un jeu de **Pierre Feuille Ciseau** contre l'ordinateur. Votre code sera écrit avec les contraintes suivantes :

- Aucune variable globale
- Le main ne devra contenir que des déclarations de variables et appels à des fonctions définies pour l'exercice
- Chaque fonction devra suivre strictement les principes données au début de ce TP :
  - ◊ Responsabilité unique
  - ◊ Explicite du nom
  - ◊ Généralité

Manipulation 4 : Pierre, feuille, ciseaux !

**Étape 9** Compléter les fichiers `pierre_feuille_ciseaux.c`, `utils.h` et `utils.c` pour coder le jeu.

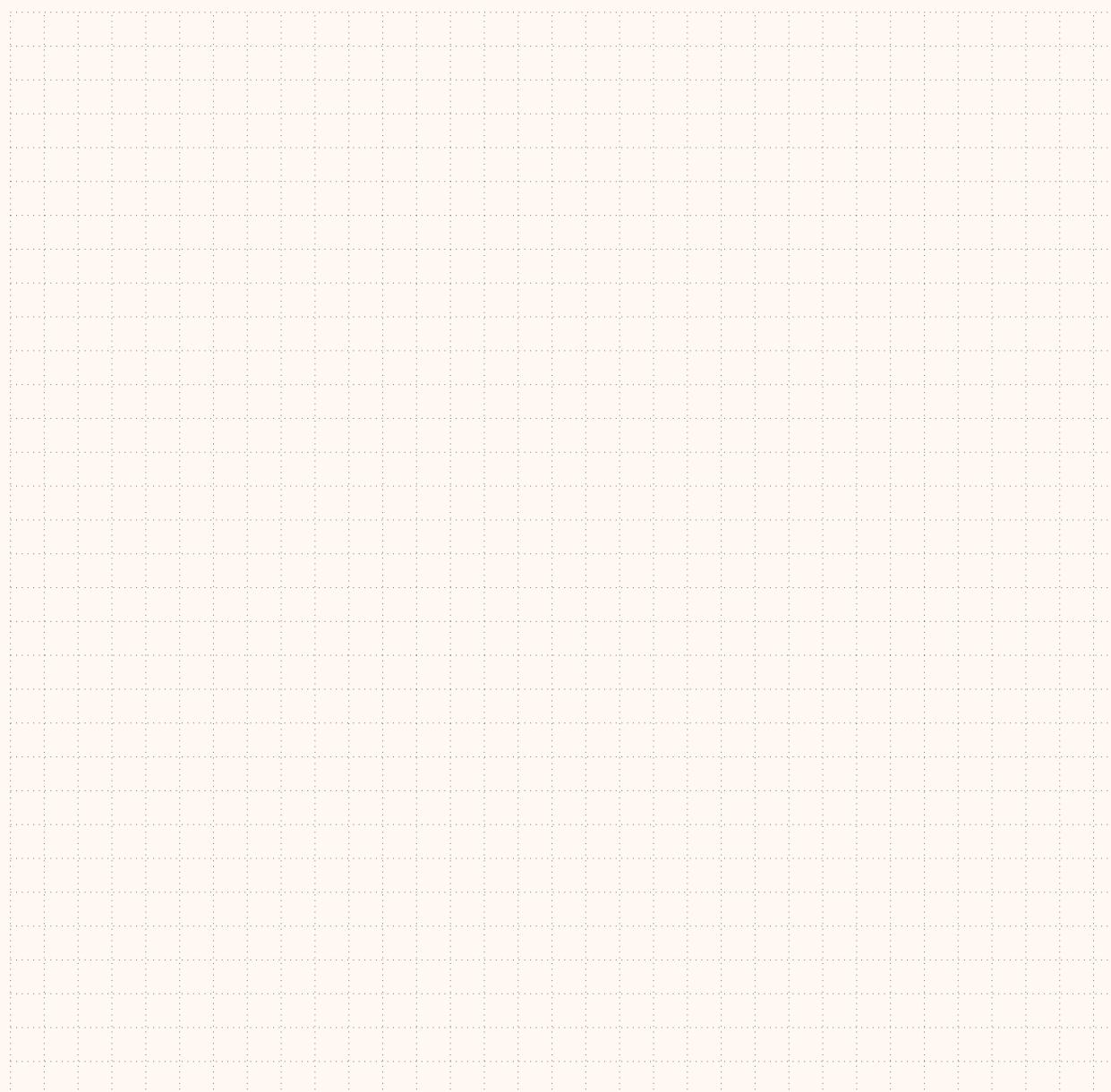
## 3 Le retour des Tacos

Dans cette partie, nous allons reprendre l'exercice du magasin de tacos pour en factoriser le code à l'aide de fonctions. L'objectif est donc de rendre le code plus lisible et propres à l'aide de fonctions pertinentes.

Préparation 1 : Conception des fonctions

**Question 1** En reprenant le cahier des charges et le code sur les Tacos, proposer une liste de fonctions à concevoir.

- Chaque fonction devra-t-être la plus générique possible
- Une fonction peut appeler une autre fonction (la fonction prenant la commande du client, par exemple.)
- Pour chaque fonction, préciser :
  - ◊ son nom ;
  - ◊ son rôle (ce qu'elle fait) ;
  - ◊ ses paramètres (type et nom) ;
  - ◊ son type de retour.



**Question 2** Faire valider par l'enseignant

## Manipulation 5 : Fonctions et réutilisation

**Étape 10** Recoder le programme des Tacos en utilisant les fonctions définies précédemment.

- Placer les prototypes des fonctions dans un fichier **boutique.h**
- Placer les définitions des fonctions dans un fichier **boutique.c**
- Placer le **main** dans un fichier **main.c**

**Étape 11** En utilisant le plus de fonction possible du programme précédent, coder un programme pour une boulangerie qui propose les produits suivants :

- |  |   |
|--|---|
| <ul style="list-style-type: none"><li>• Baguette : 1.00 €;</li><li>• Croissant : 0.90 €;</li></ul> | <ul style="list-style-type: none"><li>• Pain au chocolat : 1.20 €;</li><li>• Pain aux raisins : 1.30 €.</li></ul> |
|--|---|