	Les pointeurs ne manquent pas d'adresse	**DEV1** TP7

1. Swap

- **Écrire une fonction `swap` qui a comme paramètres deux pointeurs vers des entiers et qui échange le contenu des deux entiers pointés.**
- **Tester cette fonction en écrivant un programme qui échange le contenu de deux entiers a et b en appelant cette fonction.**

2. Min-Max

- **Écrire une fonction qui a comme paramètres un tableau d'entiers de taille quelconque, la taille du tableau, et 2 pointeurs vers des entiers min et max.**

La fonction doit renvoyer dans les entiers pointés par min et max respectivement les plus petits et les plus grands entiers du tableau.

3. Initialisation d'un tableau

- **Ecrire une fonction `createArray` qui prend en paramètre un entier n. Cette fonction doit retourner un tableau de nombres flottants double précision (type double).**
- **La fonction doit donc renvoyer un pointeur sur double. Ce tableau sera alloué dynamiquement (pourquoi ?)**

Pour le moment il s'agit donc de réserver un espace mémoire que l'on va pouvoir utiliser pour la suite. Nous ne nous occupons pas ici de savoir quelles sont les valeurs de chaque élément.

- **Ecrire une fonction `getKbdNumber` qui ne prend rien en paramètre et qui renvoie un double que l'utilisateur aura entré au clavier au cours de l'exécution**
- **Ecrire une fonction `getRandNumber` qui ne prend rien en paramètre et qui renvoie un nombre aléatoire de type double compris entre 0 et 1.**
- **Ecrire une fonction `initArrayKbd` qui prend un tableau de double en paramètre ainsi qu'un entier n. Chaque élément de ce tableau sera initialisé par l'utilisateur qui entrera les valeurs au cours de l'exécution du programme utilisant cette fonction. On utilisera la fonction `getKbdNumber`**
- **Ecrire une fonction `initArrayRand` qui prend un tableau de double en paramètre ainsi qu'un entier n. chaque élément de ce tableau sera initialisé avec un nombre aléatoire. On utilisera la fonction `getRandNumber`.**

On se rend bien compte que le code écrit précédemment est redondant et pourrait être amélioré en le rendant plus générique.

- **Pour ce faire, écrire une fonction `initArray` qui prend 3 paramètres. Un tableau de double, un entier n, ainsi qu'un pointeur sur une fonction qui ne prend rien en paramètre et qui renvoie un double. On appellera alors la fonction pointée pour chaque élément du tableau.**

Le tableau de double pourrait provenir par exemple du résultat d'un appel à la fonction : `createArray`. Le paramètre n représente quant à lui le nombre d'élément du tableau.

- **Ecrire une fonction main qui fait appel aux différentes briques que nous avons mis en place : `createArray`, `getKbdNumber`, `getRandNumber` et `initArray`.**

4. Tableau de résultats

On veut écrire un programme qui permet de gérer les résultats obtenus par des étudiants à un module.

Un résultat est défini par une note (un réel) ainsi que le nom et le prénom de l'étudiant. La structure de données qui contiendra les résultats est un tableau de pointeurs sur résultats : Result *DEV1[ ]. On souhaite que les résultats soient toujours stockés par ordre alphabétique avec le nom en majuscule et le prénom en minuscule (sauf la première lettre).

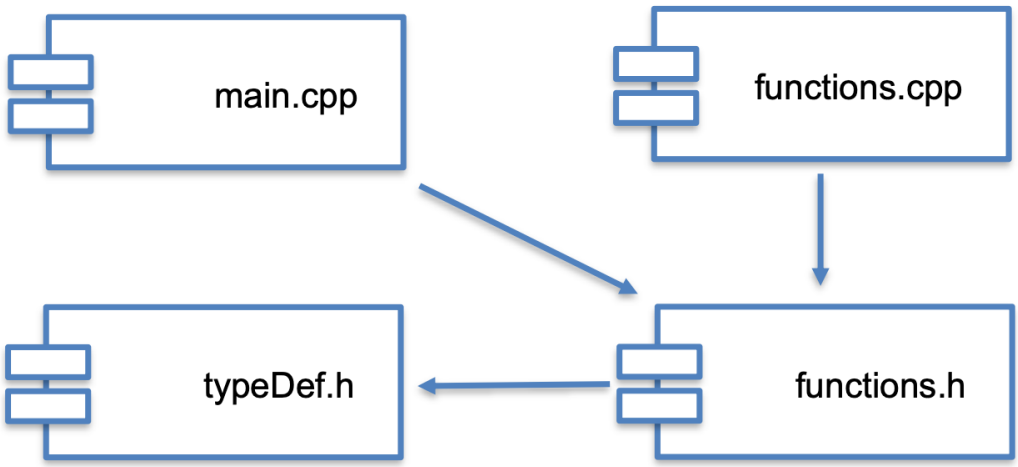
Notre tableau de notes sera géré par le menu suivant :

1. Ajouter un résultat.
2. Supprimer un résultat en tapant son nom et son prénom.
3. Afficher les résultats : NOM Prénom (note).
4. Afficher la moyenne des résultats.
5. Afficher la liste des résultats par ordre décroissant des notes : (note) NOM Prénom.

0. Quitter.

On vérifiera qu'il n'y a pas 2 résultats avec le même nom ni le même prénom dans le tableau. Il faudra trier le tableau de résultats d'abord par rapport au nom, ensuite par rapport au prénom au fur et à mesure des ajouts de résultats.

Vous veillerez à bien décomposer ce problème en menant une réflexion sur les fonctions nécessaires et l'organisation de votre projet (*.h *.cpp).



PJ : fichiers main.cpp et functions.h

main.cpp

```cpp
#include <iostream>
using namespace std;

#include "functions.h"

int main()
{
  cout << "Hello World!" << endl;

  Result* AP12[SIZE]={nullptr};

  AP12[0]=new Result({"TARANTINO",  "Quentin", 11});
  AP12[1]=new Result({"JACKSON",    "Samuel",  12});
  AP12[2]=new Result({"WILLIS",     "Scout",   14});
  AP12[3]=new Result({"WILLIS",     "Bruce",   15});
  AP12[4]=new Result({"JACKSON",    "Mickael", 13});
  AP12[5]=new Result({"JACKSON",    "Janet",   13});
  AP12[6]=new Result({"JACKSON",    "Five",    13});

  //int nbResults = 7;
  cout << "NB resultats = " << nbResults(AP12) << endl;

  displayModuleByName(AP12);
  sortModuleByName(AP12);
  displayModuleByName(AP12);
  sortModuleByNameAndFirstname(AP12);
  displayModuleByName(AP12);

  Result res={"JACKSONE", "Karl", 15};
  int place = findAPlace(AP12, res);
  cout << "place to be = " << place << endl ;

  if(place==-1){
    cout << "Result still exists !" << endl;
  }
  else{
    if(SIZE==nbResults(AP12)){
      cout << "Module's full !" << endl;
    }
    else{
      addAResult(AP12, res, place);
    }
  }
  displayModuleByName(AP12);

  deleteResult(AP12);
  displayModuleByName(AP12);

  cout << "Average = " << calculateAverageScore(AP12) << endl ;

  displayModuleByScore(AP12);

  displayModuleByName(AP12);


  // kidinioudidélaite
  int i=0;
  while(AP12[i]!=nullptr){
    delete AP12[i];
    ++i;
  }

  return 0;
}
```

functions.h

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include "typeDef.h"

// formate aString en majuscule
void nameFormat(string& aString);

// formate aString en minuscule avec la première lettre en majuscule
void firstnameFormat(string& aString);

// affiche le nombre de résultats
int nbResults(Result* aModule[]);

// affiche les nom, prénom et note sur une ligne du module aModule
void displayModuleByName(Result* aModule[]);

// tri les aResultsNb résultats du module aModule par ordre alphabétique
du nom
void sortModuleByName(Result* aModule[]);

// tri les aResultsNb résultats du module aModule par ordre alphabétique
du nom et du prénom
void sortModuleByNameAndFirstname(Result* aModule[]);

// recherche la place du résultat aResult dans le module aModule
int findAPlace(Result* aModule[], Result& aResult);

// ajoute le résultat aResult dans le module aModule à la place aPlace
void addAResult(Result* aModule[], Result& aResult, int aPlace);

// supprime le résultat du module aModule après saisie des nom et prénom
void deleteResult(Result* aModule[]);

// calule la moyenne du module aModule
float calculateAverageScore(Result* aModule[]);

// affiche les note, nom et prénom sur une ligne pour chaque résultat du
module aModule
void displayModuleByScore(Result* aModule[]);

#endif // FUNCTIONS_H
```