# TP1 4R-IN1A

Essayez de n'utiliser que le terminal pour créer vos répertoires et fichiers, compiler, et lancer vos programmes.

## Préparation

Créer un répertoire "4r-in1a" et un sous répertoire "tp1".
Dans ce repertoire, mettre en place un "projet" avec quatre fichiers :

- makefile
- test.c (contient le main())
- tp1.c
- tp1.h

Votre makefile doit vous permettre de compiler sans erreur (tp1.c/tp1.h sont vides pour l'instant).

*Commandes utiles* : mkdir, cp, ls, touch.
*Programmes d'éditions des fichiers* : gedit, vim, nano, … si vous utilisez un programme qui occupe la fenetre du terminal, ouvrez en un autre pour ne pas avoir a fermer votre éditeur à chaque fois que vous souhaitez compiler / tester.

## Rappels de base

- ajoutez aux fichiers tp1.c/tp1.h la fonction `print_message(char*);` qui affiche la chaine passée en paramètre sur la sortie standard.
- modifiez le main() dans test.c pour appeler la fonction avec la chaine "hello" dans un premier temps, testez, puis faites évoluer le programme pour qu'il affiche la première chaine de caractères passée en paramètre sur la ligne de commande si il y en a une, ou le message "Message par defaut" sinon. Vous ne devez pas modifier la fonction print_message() a ce stade, uniquement le main().
- faites évoluer encore le programme pour qu'il demande à l'utilisateur de saisir une chaine de caractère si il n'y en a pas de passée sur la ligne de commande
- si il y a plusieurs chaines de caractères passées sur la ligne de commande, le programme doit toutes les afficher (plusieurs appels à print_message())
- ajoutez la fonction `int calcule(int a, int b, char op);` qui teste la valeur de *op* (avec un switch), si c'est un des caractères + - * / la fonction réalise l'opération correspondante avec les deux opérandes a et b et renvoie le résultat. Sinon, le programme demande à l'utilisateur de saisir un caractère à l'aide de scanf. Si l'utilisateur ne saisi pas un caractère correct, le programme lui signifie et redemande la saisie d'un autre caractère et ainsi de suite.

- l'affichage du résultat se fait dans la fonction main() : testez toutes les possibilités (un caractère valide, puis un caractère non valide)
- modifiez le main() pour qu'il lise les paramètres de la fonction sur la ligne de commande. Utilisez sscanf pour lire des entiers dans des chaines de caractères.
  Exemple d'appel correct du programme :

```
$> ./a.out test 33 404 +
437
$>
```

- si il n'y a qu'un argument sur la ligne de commande, le programme essaye de lire dedans les opérandes avec plusieurs sscanf, dans l'ordre de notation préfixée "classique".
  Exemple :

```
$> ./a.out test "33 + 404"
437
$>
```

## Récurssivité (1/2)

Écrivez la fonction `nb_partition(int n, int t);` qui calcule récursivement le nombre de décompositions additives d'un nombre n en utilisant les nombres de 1 à t.
Par exemple, il y 10 façons de faire 6 avec les nombres de 1 à 5 (5+1, 4+2, 4+1+1, 3+3, 3+2+1, 3+1+1+1, 2+2+2, 2+2+1+1, 2+1+1+1+1+1, 1+1+1+1+1+1). On utilisera le
principe suivant :

- si n = 0, on retournera 1
- si n <0 on retournera 0
- si n >0 et t vaut 1, on retournera 1
- si n >0 et t plus grand que 1 :
  - on calculera récursivement le nombre de façons de faire n en utilisant t (qui est égal au nombre de façons de faire n-t avec les nombres de 1 à t)
  - et le nombre de façons de payer n sans utiliser t (qui est égal au nombre
    de façons de faire n avec les nombres de 1 à t-1)
  - et on retournera la somme des deux.

## Rappels tableaux

Écrivez les fonctions :

- print_int_tab(); qui affiche un tableau d'entiers,
- print_char_tab(); qui affiche un tableau de char,

- print_string(); qui affiche une chaine de caractère.

Testez les deux premières avec des tableaux déclarés sur la pile du main(), et pour la troisième faites afficher par votre programme la ligne de commande qui l'a lancé.

## pointeurs / malloc

- Écrivez une fonction `int* new_int_tab(int size);` qui prend en paramètre un entier, alloue un tableau de cette taille, et renvoie son adresse. La fonctino renvoie NULL en cas de problème d'allocation. Le main() devra donc tester le retour avant d'utiliser le tableau.
- Ecrivez une fonction `void free_int_tab(int*);` qui libère la mémoire associée au tableau. À quoi sert cette fonction ? pourquoi ne pas utiliser directement free(); ?
- Écrivez la procédure `void swap(int* a, int* b);` qui échange les valeurs des variables (de type int) pointées par a et b.
- Écrivez la procédure `void swap(void* a, void* b, int size_t);` qui échange les valeurs des variables (de type inconnu) pointées par a et b. Pour la copie, vous utiliserez la fonction memcpy().
- Testez vos fonctions !

## structures

Définir les types suivants :

- `t_Date` : représente une date avec trois entiers (jours, mois, an)
- `t_Identite` : représenter une personne avec deux chaines de caractères (nom, prenom). La taille des tableaux stockant les chaines peut être fixée à 10 (définissez une constante avec #define).
- `t_Adresse` : une adresse composée d'un entier (code postal) et d'une chaine de caractères (ville), limité à 9 lettres encore pour simlifier.

Écrivez (et testez) les fonctions suivantes :

- `int est_bissextile(t_Date d1);` qui retourne 1 si l'année est bissextile, 0 sinon
- `int nb_jours_par_an(t_Date d1);` qui calcule et retourne le nombre de jours de l'année de d1
- `nb_jours_par_mois(t_Date d1);` qui retourne
  - le nombre de jours du mois (de d1) s'il est compris entre 1 et 12
  - -1 sinon
- `char* nom_jour(t_Date d1);` qui retourne le nom du jour de la date d1. Il suffira d'appeler la fonction pour connaître le nom du jour d'une date.
- `int cmp_dates( t_Date d1 , t_Date d2);` qui
  - renvoie 0 si d1 égal d2
  - -1 sinon

- `int cmp_identites( t_Identite id1, t_Identite id2);` qui
  - renvoie 0 si id1 égal id2
  - -1 sinon
- `int cmp_adresses(t_Adresse ad1, t_Adresse ad2);` qui
  - renvoie 0 si ad1 égal ad2
  - -1 sinon
- modifiez les trois dernières fonctions pour qu'elles renvoient
  - 0 si égalité,
  - un nombre <0 si le premier paramètre est plus petit que le deuxième
  - un nombre >0 si le premier paramètre est plus grand que le deuxième
    Pour les dates, il suffit de compter le nombre de jour entre les deux dates. Pour les identités on compare d'abord les nom puis les prenoms, dans l'ordre lexicographique (le dictionnaire), pour les adresses, l'ordre est doné par les code postaux.

## Annuaire

- Créer une 1 ère structure t_Eleve pouvant contenir les informations suivantes :
  - t_Identite id
  - t_Date naiss ;
- Créer une 2 ème structure qui va représenter l'annuaire. Cette structure Annuaire contiendra :
  - un tableau tab de 10 éléments (pointeurs t_Eleve initialisés à NULL)
  - et un compteur cpt (initialisé à zéro) indiquant le nombre d'élèves dans le tableau.
- Créer une fonction `t_Eleve* saisie_eleve(t_Identite id, t_Date naiss);`
  - qui effectue une allocation (malloc ) d'un t_Eleve
  - initialise ses champs avec les informations founies en paramètre
  - retourne le pointeur vers le t_Eleve allouée

Le pointeur retouné par saisie_eleve sera rangé dans la première place libre du tableau de l'annuaire et le compteur cpt incrémenté de 1.

- Créer ensuite une procédure afficher_eleve qui affiche les informations contenues dans une structure t_Eleve e1 passée en argument (pointeur pour éviter les copies) : `void afficher_eleve(t_Eleve * e );`
- Créer une fonction qui recherche un t_Eleve e dans l' annuaire selon le critère nom de l'élève donné en argument : `int ceherche_eleve(char *nom );`. La fonction retourne
  - la position dans le tableau si trouvé

    - -1 sinon
- Créer une fonction `int anniversaires(t_date d1);` qui retourne le nombre d'élèves dont l'anniversaire est d1