Feuille de travaux pratiques n° 1
Langage C++ et environnement

Exercice 1.1

Écrire, compiler et tester le programme C++ demandant à l'utilisateur son prénom et son nom en une seule fois et sur la même ligne, et affichant la phrase « `Untel, votre nom comporte` x `lettres` » où « `Untel` » est remplacé par le prénom de la personne et x par le nombre de lettres de son nom.

Exercice 1.2 (Positions géographiques)

On considère le fichier texte coordonnees.txt suivant (disponible sur madoc) organisé en trois colonnes :

Ancenis	47.22	1.10
Auray	47.40	3.00
Brest	48.23	4.29
Carhaix	48.17	3.34
Chateaubriant	47.43	1.23
Chateaulin	48.12	4.06
Clisson	47.05	1.17
Concarneau	47.52	4.00

La première colonne est une chaîne de caractères désignant une ville ; la deuxième colonne est un `double` désignant une latitude ; la troisième colonne est un `double` désignant une longitude.
Dans la suite, on s'attachera à respecter les principes de la compilation séparée : toutes les fonctions de manipulation de `ville_ts` seront définies dans un fichier indépendant du fichier contenant le programme principal.

1. Définir une structure `ville_t` composée d'un nom, d'une latitude et d'une longitude ;
2. Écrire la fonction `lire_fichier()` prenant en entrée un « *flux* » de type `ifstream` sur un fichier texte ayant la structure ci-dessus et un tableau t de `ville_ts`, et insérant dans t les informations relatives aux villes apparaissant dans le fichier ;
3. Écrire la fonction `chercher_par_nom()` prenant en entrée un nom de ville, un tableau de `ville_ts`, et retournant un pointeur sur la structure correspondante si la ville se trouve dans le tableau, ou `nullptr` sinon ;
4. Écrire la fonction `chercher_par_position()` prenant en entrée une position (i.e. une latitude et une longitude), un tableau de `ville_ts`, et retournant la structure correspondant à la ville la plus proche de cette position. Étant donnés deux points $A = (l_A, L_A)$ et $B = (l_B, L_B)$, on utilisera la notion de distance suivante entre A et B :
$$\text{dist}(A, B) = \sqrt{(l_A - l_B)^2 + (L_A - L_B)^2}\,;$$
5. Écrire un programme affichant un menu offrant à l'utilisateur la possibilité de rechercher les coordonnées d'une ville en donnant son nom, d'afficher le nom de la ville la plus proche d'un point donné, ou de quitter le programme

Exercice 1.3

1. Écrire la définition de la structure `noeud_t` représentant une cellule de liste doublement chaînée de réels (type `double`) ;
2. Écrire la fonction `afficher_liste()` affichant à l'écran la liste dont la tête est passée en paramètre ;

3. Écrire la fonction `ajouter_en_tete()` ajoutant un **`double`** en tête d'une liste chaînée et retournant la nouvelle tête de la liste ;
4. Écrire la fonction `chercher_element()` retournant un pointeur sur le premier nœud contenant un **`double`** passé en paramètre, ou `nullptr` si le **`double`** n'est pas présent ;
5. Écrire la fonction `ajouter_element()` ajoutant un **`double`** d_1 passé en paramètre après un **`double`** d_2 passé en paramètre. Si d_2 n'est pas dans la liste, l'ajout de d_1 doit se faire en tête de la liste ;
6. Écrire la fonction `retirer_element()` retirant un **`double`** passé en paramètre s'il est présent dans la liste ;
7. Écrire un programme principal utilisant toutes les fonctions ci-dessus.

Exercice 1.4

On considère une application où les désallocations de mémoire se font toujours dans l'ordre inverse des allocations. Pour ce type de programme, il est possible d'optimiser la gestion de la mémoire en écrivant un allocateur spécialisé avec une stratégie « en pile » : à l'initialisation, on alloue un grand espace mémoire suffisant pour tous les besoins du programme, c'est le *tampon*. L'allocateur alloue la mémoire en gérant un « pointeur » sur le début de l'espace encore libre du tampon. Le désallocateur ramène le « pointeur » à sa position précédente.

1. Définir le type `tampon_t` ;
2. Écrire une fonction `init_tampon()` prenant en entrée le nombre total d'octets à allouer et retournant un objet de type `tampon_t` ;
3. Écrire une fonction `allouer()` prenant en entrée un tampon et le nombre d'octets à allouer et retournant un pointeur de type **`void`**`*` sur le début de l'espace alloué. La fonction devra retourner `nullptr` si la place restante est insuffisante pour répondre à la requête ;
4. Écrire une fonction `desallouer()` prenant en entrée un tampon et récupérant la place occupée par le dernier objet alloué ;
5. Écrire une fonction `delete_tampon()` détruisant l'espace alloué dans le tampon ;
6. Écrire un programme principal pour tester les fonctions écrites précédemment.

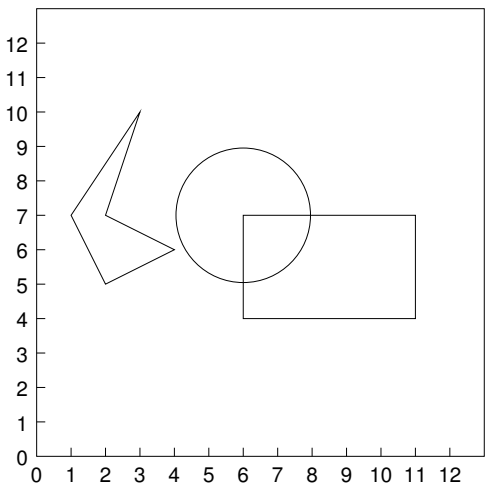
Exercice 1.5

Un fichier texte contient la description d'une image composée de primitives graphiques. On considère trois primitives :

— `cercle` : un cercle défini par les coordonnées de son centre et son rayon ;
— `rectangle` : un rectangle défini par les coordonnées des points "haut gauche" et "bas droit" ;
— `polygone` : un polygone défini par le nombre de côtés et les coordonnées des points le définissant ;

Les coordonnées des points sont des paires de nombres entiers de type **`int`**. Le rayon d'un cercle est aussi de type **`int`**.
Le fichier est constitué comme illustré dans l'exemple ci-dessous :

Fichier figure.dat
```
cercle
6,7 2
rectangle
6,7 11,4
polygone
5 1,7 3,10 2,7 4,6 2,5
```



Le type de primitive est écrit seul sur une ligne ; la ligne suivante contient les informations définissant la primitive dans l'ordre indiqué plus haut. Le fichier est donc constitué d'un ensemble de taille variable de paires de deux lignes.

1. Définir les types `cercle_t`, `rectangle_t` et `polygone_t` ;
2. Définir le type `image_t` correspondant à la description d'une image en terme de liste de primitives ;
3. Écrire la fonction `lire_point()` prenant en entrée un flux `ifstream` et une référence sur un `point_t p`, et lisant dans le flux un point dont les coordonnées sont sauvées dans `p` ;
4. Écrire la fonction `lire_image()` prenant en entrée un flux `ifstream` correspondant à un fichier de description ouvert en lecture et retournant un pointeur sur un objet de type `image` contenant les informations relatives à l'image décrite dans le fichier ;
5. Écrire la fonction `detruire_image()` prenant en entrée une image et désallouant la mémoire associée ;
6. Écrire la fonction `affiche_image()` affichant à l'écran la liste des primitives de l'image passée en paramètre ainsi que leurs coordonnées ;
7. Écrire le programme prenant un nom de fichier en paramètre et chargeant en mémoire la description de l'image décrite dans le fichier. Le programme devra retourner un code d'erreur 1 en affichant un message d'erreur adéquat sur la sortie standard d'erreur si aucun paramètre n'a été passé sur la ligne de commande ou si le fichier ne peut être ouvert en lecture.

Exercice 1.6

On souhaite écrire une librairie de manipulation de tableaux de `double`s dont la taille peut être modifiée dynamiquement. Pour cela, on se propose de créer un type `tabdyn_t` contenant la taille courante du tableau et une liste chaînée de ces éléments.

1. Définir le type `tabdyn_t` ;
2. Écrire la fonction `creer_tabdyn()` prenant en entrée un tableau `T` de `double`s et sa taille, et retournant un `tabdyn` contenant tous les éléments de `T` dans le même ordre ;
3. Écrire la fonction `creer_tab()` créant un tableau de **`double`**s classique à partir d'un `tabdyn_t` ;
4. Écrire la fonction `afficher()` prenant en entrée un `tabdyn_t t` et un flux `os` de type `ostream` et affichant le contenu de `t` sur `os` ;
5. Écrire la fonction `inserer()` prenant en entrée un tableau `t`, un flottant `f` et une position `i` et insérant `f` à la position `i` dans `t` :
   — S'il existe déjà un élément à la position `i`, il est remplacé par `f` ;
   — Si `i` fait référence à une position à l'extérieur du tableau, celui-ci est étendu (voir figure ci-dessous) ;
6. Écrire la fonction `supprimer()` prenant en entrée un `tabdyn_t t` et un indice `i` et supprimant tous les éléments du tableau à partir de `i` ;
7. Écrire un programme principal créant le tableau de **`double`**s `td={4.5 , 6.7 , -2.1}`, créant un `tabdyn_t T1` à partir de `td`, affichant `T1` sur l'écran, insérant la valeur **`9.8`** dans `T1` à l'indice **`5`**, créant un tableau de **`double`**s `td2` à partir de `T1` et affichant `td2` à l'écran.