Laboratoire 4

Les structures, les objets

Étape 1 : Coder un message en code morse

Vous devez faire un programme permettant de traduire en code Morse un message saisie au clavier ne dépassant pas une ligne (127 caractères maximum). Chaque caractère du message doit être affiché en code morse. Lorsqu'on saisie plus d'un mot, on doit afficher un saut de ligne entre les mots à la verticale. Si le texte contient d'autres caractères que ceux pouvant être traduit en morse, le programme affichera simplement des points d'interrogation à la place du code.

Voici un exemple de la saisie:

Entrer votre message à coder en morse (1 ligne max) :
Bonjour!

Voici la traduction de votre message:
<B> -...
<o> ---
<n> -.
<j> .---
<o> ---
<u> ..-
<r> .-.
<!> ???

Pour saisir une ligne jusqu'au enter, l'instruction getline() s'utilise différemment avec les tableaux de char qu'avec les strings. Vous pouvez utiliser une string pour votre message.

Voici un exemple d'utilisation :

```
char strMessage[128];
cout << "Entrer votre message à coder en morse (1 ligne max) : ";
cin.getline(strMessage,127);
//avec les tableaux de char, getline est une méthode de istream

string nom;
getline(cin, nom); //avec les string, getline est une fonction
```

Tableau des codes morses

Il y a en tout 37 codes morses. Vous devez déclarer une **structure code morse** qui contiendra la lettre et le code morse correspondant. Vous déclarez un tableau de code morse qui sera initialisé avec les codes morses suivants :

.	.-.-.-	0	-----	1	.----	2	..---
3	...--	4	....-	5	.....	6	-....
7	--...	8	---..	9	----.		
A	.-	B	-...	C	-.-.	D	-..
E	.	F	..-.	G	--.	H	....
I	..	J	.---	K	-.-	L	.-..
M	--	N	-.	O	---	P	.--.
Q	--.-	R	.-.	S	...	T	-
U	..-	V	...-	W	.--	X	-..-
Y	-.---	Z	--..				

Le fichier codeMorses.txt comprend toutes les données à lire pour remplir votre tableau.

Bien découper en fonction

L'algorithme de ce programme est bien simple, mais souvent, les fonctions sont trop volumineuses et deviennent peu réutilisable.

Faites une fonction pour

- **initialiser les codes morses** à l'aide du fichier texte (cette fonction appelle la fonction qui ouvre le fichier, test s'il existe, s'il est vide et le ferme après la lecture)
- **saisir le message à coder en morse**
- **rechercher le caractère** dans le tableau de code morse.

Voici l'algorithme pour ce programme: (le main doit ressembler à cet algorithme)

Initialiser le tableau de code morse
message = **saisirMessage**
Pour i de 1 au nombre de caractère du message
    Si message(i) = espace
        Faire un saut de ligne
    Sinon
        pos = **Rechercher** la majuscule de message(i) dans le tableau de 37
        Si pos = 37
            Afficher <message(i)> et ???
        Sinon
            Afficher <message(i)> et codeMorse(pos)

Montrez-moi cette étape dès qu'elle est terminée.

Étape 2 : Création du premier objet

Dans le programme du laboratoire 3 qui calcule l'âge d'une personne, nous avons fait une structure, mais ce programme aurait dû avoir une classe. Pourquoi? Contrairement à la struct code morse qui ne fait que regrouper les 2 informations liées aux codes morses, la date créée fait des actions, comme générer la date du jour, calculer la différence entre 2 dates ou l'entier correspondant au nombre d'années écoulés à partir d'une date.

Nous allons donc modifier la struct pour en faire un objet à part entière réutilisable.

1. Créez un nouveau projet et copiez-y le code de l'étape 1 du lab. 3 pour faire les modifications suivantes.

2. Modifier la structure date pour une class

```
class date
{
      int jour;
      int mois;
      int annee;
};
```

3. Ensuite, tentez de compiler votre programme. Vous verrez rapidement que ça ne fonctionne plus, car dans une **struct** tout est **public** par défaut, tandis que dans une **class**, tout est **privé** par défaut. C'est la force des objets, l'encapsulation des données qui assure la sécurité de celle-ci en empêchant leurs utilisations directement de l'extérieur. Mettez donc le code du main en commentaire sauf la déclaration des dates et vous pourrez compiler de nouveau.

4. Ajouter les étiquettes private et public à votre class, mettez des soulignements devant les propriétés et ajoutez les 2 constructeurs.

```
class date
{
private:              //ajouter private pour plus de précision
                      //même si c'est privé par défaut
     int _jour;
     int _mois;
     int _annee;

public:               //nous ajouterons ici les méthodes qui
                      //permettront de manipuler la date
     date();
     date(int jour, int mois, int annee);
};
//constructeur sans paramètre
date::date()
{
     _jour = _mois = 1;
     _annee = 1900;       //initialisons la date à 01/01/1900
}
//constructeur avec paramètres
date::date(int jour, int mois, int annee)
{
     _jour = jour;
     _mois = mois;
     _annee = annee;
}
```

On pourrait coder les constructeurs dans la classe, mais il est plus intéressant de le faire à l'extérieur pour plus de lisibilité.

5. Dans le main, vous pouvez maintenant tester ces 2 constructeurs comme ceci

```
date dateJour();                  //appel du constructeur sans paramètre
date dateNaissance(28, 01, 2000); //appel du constructeur avec 3 paramètres
```

Lorsqu'un objet est créé, un constructeur est appelé automatiquement, il choisi le bon selon le nombre de paramètre. Si aucun ne correspond, la compilation génère une erreur. Si aucun constructeur n'est codé, il y en a un par défaut qui existe qui n'a aucun paramètre et qui ne fait rien. C'est pour cela qu'à l'étape 3 vous pouvez compiler. Dès qu'au moins un constructeur est crée, le constructeur par défaut n'existe plus.

6. C’est la même chose pour le destructeur. Lorsque l'accolade fermante du main est rencontrée, les variables locales du main sont détruites, il en est de même pour les objets. Lorsqu’un objet est détruit, le destructeur est appelé automatiquement. S’il n’y en a pas de codé, il en existe un par défaut qui est vide, mais c’est une bonne pratique d’en coder un nous même.

   Ajouter ce destructeur à votre classe. Mettez le prototype dans la classe et la définition en dehors de la classe avec la résolution de portée (date::) Cette résolution de portée permet de dire au compilateur que la méthode appartient à la classe date.

```
~date();                    //prototype du destructeur

//définition du destructeur qui reset (nettoie) le contenu des propriétés
date::~date()
{
     _jour = 0;
     _mois = 0;
     _annee = 0;
}
```

7. Maintenant, nous aurons besoin de Getteurs/Setteurs (ou Accesseurs/Mutateurs). Ces interfaces sont très importantes pour accéder à chacune des propriétés. Les getteurs permettre de recevoir les valeurs pour les utiliser à l’extérieur et les setteurs permettent de modifier les valeurs. Ajouter les 3 getteurs et 3 setteurs pour la classe date.

   Voici le getteur de jour :

```
int getJour()const;            //prototype du getteur

//définition du getteur qui retourne le jour
int date::getJour()const
{
     return _jour;
}
```

   Les getteurs auront toujours un const après les paramètres pour empêcher la modification des propriétés à l’intérieurs du code. Ce const protège le paramètre implicite qui est toujours passé par référence, donc qui pourrait être modifié sans qu’on le veuille.

   Voici le setteur de jour :

```
void setJour(int jour);        //prototype du setteur

//définition du setteur qui reçoit un int et modifie le jour
int date::setJour(int jour)
{
     assert(jour >=1 && jour <= 31); //robustesse avec <cassert>
     _jour = jour;
}
```

   Les setteurs auront souvent à intégrer de la robustesse pour protéger la modification de la propriété avec des données incohérentes dans le paramètre.

8. Dans la fonction qui saisie la date de naissance, vous pouvez maintenant faire la saisie des informations et modifier la date de naissance à l'aide des setteurs.

```
do
{
        valid = true;
        viderBuffer();
        cout << endl << "Entrer un votre date de naissance (jj mm aaaa) : ";
        cin >> jour >> mois >> annee;
        ...
}while(!valid);     //on tourne tant que le flux fail, donc pas un int

dateNaissance.setJour(jour);      //dateNaissance est le paramètre implicite
dateNaissance.setMois(mois);      //et il est toujours passé par référence
dateNaissance.setAnnee(annee);
```

9. Il serait intéressant pour l'objet d'offrir un interface de modification d'une date avec 3 paramètres au lieu d'imposer la modification des 3 informations d'une date avec les 3 setteurs séparément. Ajoutons donc une méthode setDate qui appellera les 3 setteurs.

   Comme avec la définition des fonctions, on ajoute toujours un commentaire au dessus de la définition de nos méthodes.

10. Il serait intéressant pour l'objet d'offrir une méthode qui retourne la date au complet pour avoir accès à toute la date sans devoir appeler les 3 getteurs. . Ajoutons donc une méthode getDate qui retournera la date. Dans la méthode, on n'a qu'à écrire

```
return *this;  //this est un pointeur vers l'objet, *this est l'objet
```

11. On veut maintenant générer la date du jours, cette fonction pourrait rester dans le programme, mais il serait bien plus intéressant de la mettre dans notre objet, comme ça, dans le futur, tout ceux qui utiliseront l'objet date pourront générer facilement une date du jour comme ceci:

```
dateJour.today();   //appel la méthode today qui génère la date du jour et
                    //l'affecte dans les propriétés de l'objet implicite dateJour
```

12. On aimerait bien afficher la date dans un format standard. Faites une fonction print qui reçoit un ostream en paramètre et qui affiche la date de façon standard `01/01/2020`

13. Pour le calcul de l'âge, nous conserverons la fonction dans le programme, car pour une date, l'âge n'a pas de signification, il serait intéressant pour l'objet d'offrir la fonctionnalité de calculer le nombre de jours écoulés entre 2 dates, mais pas le calcul de l'âge. Modifier donc votre fonction pour calculer l'âge avec les getteurs pour accéder aux différentes parties de vos dates.

14. Finalement, vous allez mettre votre objet dans un fichier d'entête. Créez un fichier date.h dans la section des fichiers d'entête et copiez-y votre classe. Créez un fichier source date.cpp dans la section des fichiers sources et copiez y les méthodes. Pour que tout compile bien, vous devez inclure en haut de votre .h les librairies propres à la classe. (attention, ne pas mettre toutes les librairies du main, mais seulement celles utilisées par l'objet). En haut de date.cpp et du main, vous devez inclure date.h et retirer les librairies qui ne sont plus utilisées dans le main.

Exemple

date.h

```
/*****************************************************************************
Auteur: Julie Gagnon
Date: 01/02/2020
Programme: date.h
But: objet date (jour, mois, annee) qui offre de générer de la date du jour
*****************************************************************************
/

#pragma once
#include <time.h>
#include <cassert>
#include <iostream>
#include <iomanip>
using namespace std;

class date
{

};
```

date.cpp

```
#include <date.h>

//constructeur sans paramètre qui initialise la date à 01/01/1900
date::date()
{
	jour = mois = 1;
	annee = 1900;
}
//constructeur avec paramètres
date::date(int jour, int mois, int annee)
{
	_jour = jour;
	_mois = mois;
	_annee = annee;
}
… on place les méthodes dans le même ordre que leur prototype dans la classe
```

calculerAge.cpp

```
/*****************************************************************************
Auteur: Julie Gagnon
Date: 01/02/2020
Programme: calculerAge.cpp
But: programme qui calcul l'âge à partir de la date de naissance saisie
*****************************************************************************
/

#include <date.h>
#include <iostream>
using namespace std;

int main()
{
}
```

Montrez-moi ces 2 programmes terminés avant le cours où le lab.5 sera présenté. (semaine 5)