Exercice 1 - Création d'un projet en C++

sous Visual : Créer un projet en C++ de la manière suivante :

- Aller dans le menu "Fichier / Nouveau / Projet".
- Sélectionner "Projet vide".
- Entrer un nom pour le projet, par exemple "TD1".
- Sélectionner un emplacement où le projet sera enregistré (par exemple sur une clé USB).
- Cliquer sur "OK".

Créer un nouveau fichier source "fonction.cpp" de la manière suivante :

- Aller dans le menu "Projet / Ajouter un nouvel élément".
- Sélectionner "Fichier C++ (.cpp)" dans la liste.
- Entrer le nom du fichier en face de "Nom :" (ici `fonction`).
- Cliquer sur le bouton "Ajouter".

sous Code : :Blocks : Créer un projet en C++ de la manière suivante :

- Aller dans le menu "File / New / Project".
- Sélectionner "Console Application".
- Cliquer "Next>".
- Sélectionner le langage C++ puis cliquer sur "Next>".
- Entrer un titre pour le projet, par exemple "TD1".
- Sélectionner un emplacement où le projet sera enregistré (par exemple sur une clé USB).
- Cliquer sur "Next>" puis "Finish". Noter qu'un fichier "main.cpp" a déjà été créé avec une fonction `main` : **supprimer le pour cette fois**.

Créer un nouveau fichier source "fonction.cpp" de la manière suivante :

- Aller dans le menu "File / New / File...".
- Sélectionner "C/C++ source" puis cliquer sur "Go".
- Sélectionner le langage C++ puis cliquer sur "Next>".
- Entrer le nom du fichier (ici `fonction`).
- Cocher les contextes de compilation dans lesquels seront utilisés les fichiers (a priori aussi bien "Debug" que "Release" (si vous ne faites pas cette opération, votre fichier source sera ignoré à la compilation).
- Cliquer sur "Finish".

Question 1

Ajouter le bout de code suivant :

```
#include<iostream>
#include<string>
using namespace std;
void bonjour() {
 cout<<"Entrez votre prenom :";
 string prenom;
 cin>>prenom;
 cout<<"Bonjour "<<prenom<<"\n";
}
```

fonction.cpp

Compiler le projet avec "Générer/Générer *nom_du_projet*". Que se passe t-il ? Expliquer.

Question 2

À la suite de la fonction `bonjour()`, ajouter le bout de code suivant :

```
/*...*/
int main() {
 bonjour();
 return 0;
}
```

fonction.cpp

Re-compiler le projet. Que se passe t-il ? Expliquer.

Exercice 2 - Fichier source *vs.* fichier d'en-tête

Reprendre l'exercice précédent en ajoutant un fichier d'entête `fonction.h` de la manière suivante :

sous Visual :

- Aller dans le menu "Projet / Ajouter un nouvel élément".
- Sélectionner "Fichier d'en-tête (.h)" dans la liste.
- Entrer le nom du fichier en face de "Nom :" (ici "fonction").
- Cliquer sur le bouton "Ajouter".

sous Code : :Blocks :

- Aller dans le menu "File / New / File...".
- Sélectionner "C/C++ header" puis cliquer sur "Go".
- Sélectionner le langage C++ puis cliquer sur "Next>".
- Entrer le nom du fichier (ici `fonction`).
- Cocher les contextes de compilation dans lesquels seront utilisés les fichiers.
- Cliquer sur "Finish".

Question 1

Ajouter le texte suivant :

```
Une université de technologie,
est un établissement à caractère scientifique, culturel et professionnel
qui a pour mission principale la formation des ingénieurs,
le développement de la recherche et de la technologie.
```

fonction.h

Compiler le projet. Que se passe t-il ? Expliquer.

Question 2

Ajouter l'instruction « `#include "fonction.h"` » en haut du fichier `fonction.cpp`. Compiler le projet. Que se passe t-il ? Expliquer.

Exercice 3 - Fichier d'en-tête et déclaration

Reprendre l'exercice précédent en ajoutant un fichier source `main.cpp`. On repart alors des trois fichiers `fonction.h`, `fonction.cpp` et `main.cpp` qui contiennent respectivement les codes suivants :

```
// vide pour l'instant
```

fonction.h

```
#include<iostream>
#include<string>
using namespace std;
void bonjour() {
 cout<<"Entrez votre prenom :";
 string prenom;
 cin>>prenom;
 cout<<"Bonjour "<<prenom<<"\n";
}
```

fonction.cpp

```
int main() {
 bonjour();
 return 0;
}
```

main.cpp

Question 1

Compiler le projet. Que se passe t-il ? Expliquer.

Question 2

Corriger le problème de deux façons différentes : une fois en modifiant uniquement le fichier `main.cpp`, une fois en modifiant aussi le fichier `fonction.h`.

Exercice 4 - Instructions d'inclusions conditionnelles

- Ajouter au projet un fichier d'entête `essai.h`.
- Ajouter l'instruction **`#include`** `"fonction.h"` dans le fichier `essai.h`.
- Ajouter l'instruction **`#include`** `"essai.h"` dans le fichier `fonction.h`.
- Compiler le projet.

Que se passe t-il ? Expliquer. Corriger le problème.

Exercice 5 - E/S en C++, définition de variables

Réécrire le programme suivant en ne faisant appel qu'aux nouvelles possibilités d'entrées-sorties de C++ (*c.-à-d.* en évitant les appels à `printf` et `scanf`). Définir le plus tard possible les variables. Utiliser une constante plutôt que l'instruction **#define** du préprocesseur.

```
#include<stdio.h>
#define PI 3.14159
void exerciceA(){
 int r; double p, s;
 printf("donnez le rayon entier d'un cercle:");
 scanf("%d",&r);
 p=2*PI*r;
 s=PI*r*r;
 printf("le cercle de rayon %d ",r);
 printf("a un perimetre de %f et une surface de %f\n",p,s);
}
```

fonction.cpp

Exercice 6 - Définition - Initialisation - Affectation

Dans la fonction `main()`, définir une variable `x` de type **`double`** en l'initialisant avec la valeur `3.14` et afficher sa valeur. Définir une variable `y` de type **`double`** et l'affecter avec la valeur `3.14`. Tentez d'afficher sa valeur avant et après affectation. Que se passe-t-il ?

Exercice 7 - Variables constantes

Définir une variable constante `pi` de type **`double`**, lui donner la valeur `3.14`, et afficher sa valeur. Tenter ensuite d'affecter cette variable avec une autre valeur.

Exercice 8 - Espaces de noms

Voici deux fonctions qui portent le même nom :

```
void bonjour() {
 cout<<"nichao\n";
}
```

et

```
void bonjour() {
 cout<<"hello\n";
}
```

À partir d'un nouveau projet avec 3 fichiers `fonction.h`, `fonction.cpp`, `main.cpp`, déclarer les deux fonctions `bonjour` dans le fichier `fonction.h` en utilisant deux namespaces différents. Définir ces fonctions dans le fichier `fonction.cpp`. Enfin, utiliser ces deux fonctions dans la fonction `main`. Modifier le programme pour ne pas utiliser l'instruction **`using namespace`** `std;`.

Exercice 9 - Surcharge de fonction - Fonctions `inline`

Soit les fonctions de nom `fct` suivantes :

```
/* ...*/
int fct(int x);
int fct(float  y);
int fct(int x, float  y);
float fct(float x, int  y);
```

fonction.h

```
/* ...*/

int fct(int x){ std::cout<<"1:"<<x<<"\n"; return 0; }
int fct(float  y){ std::cout<<"2:"<<y<<"\n"; return 0; }
int fct(int x, float  y){ std::cout<<"3:"<<x<<y<<"\n"; return 0; }
float fct(float x, int  y){ std::cout<<"4:"<<x<<y<<"\n"; return 3.14; }

void exercice_surcharge(){
 int i=3,j=15;
 float x=3.14159,y=1.414;
 char c='A';
 double z=3.14159265;
 fct(i); //appel 1
 fct(x); //appel 2
 fct(i,y); //appel 3
 fct(x,j); //appel 4
 fct(c); //appel 5
 fct(i,j); //appel 6
 fct(i,c); //appel 7
 fct(i,z); //appel 8
 fct(z,z); //appel 9
}
```

fonction.cpp

Question 1

Les appels de fonction sont-ils corrects et, si oui, quelles seront les fonctions effectivement appelées et les conversions mises en place ? Expliquer les appels incorrects et les corriger.

Question 2

Transformer le programme précédent (corrigé) pour que les fonctions fct deviennent des fonctions en ligne.

Exercice 10 - Pointeurs - Pointeurs `const`

Parmi les instructions suivantes, indiquer celles qui sont ne sont pas valides et expliquer pourquoi.

```
double* pt0=0;
double* pt1=4096;
double* pt2=(double*)4096;
void* pt3=pt1;
pt1=pt3;
pt1=(double*)pt3;
double d1=36;
const double d2=36;
double* pt4=&d1;
const double* pt5=&d1;
*pt4=2.1;
*pt5=2.1;
pt4=&d2;
pt5=&d2;
double* const pt6=&d1;
pt6=&d1;
*pt6=2.78;
double* const pt6b=&d2;
const double* const pt7=&d1;
pt7=&d1;
*pt7=2.78;
double const* pt8=&d1;
pt8=&d2;
pt8=&d1;
*pt8=3.14;
```

dans une unité de compilation .cpp

Exercice 11 - Référence - Références `const`

Parmi les instructions suivantes, indiquer celles qui sont ne sont pas valides et expliquer pourquoi.

```
double& d1=36;
double d2=36;
double& ref=d2;
ref=4;
const double d3=36;
const double& d4=36;
const double& d5=d2;
d5=21;
const double& d6=d3;
double& ref2=d6;
int i=4;
double& d7=i;
const double& d8=i;
d8=3;
```

dans une unité de compilation .cpp

Exercice 12 - Passage d'argument par adresse et par référence

Question 1

Écrire la fonction qui permet d'inverser les valeurs de deux variables entière passées en argument en utilisant des passages par adresse (prototype : `void inverse(int* a, int* b);`).

Question 2

Écrire la fonction qui permet d'inverser les valeurs de deux variables entière passées en argument en utilisant des passages par référence (prototype : `void inverse(int& a, int& b);`).

Exercice 13 - Passage d'argument par adresse et par référence

Soit le modèle de structure suivant :

```
/*...*/
struct essai {
 int n;
 float x;
};
```

fonction.h

Ecrire une fonction nommée `raz` permettant de remettre à zéro les 2 champs d'une structure de ce type transmise en argument.
Faire l'exercice une fois en utilisant un passage par adresse, une fois en utilisant un passage par référence. Dans les deux cas, on écrira un petit programme d'essai de la fonction. Il affichera les valeurs d'une structure de ce type après appel de la dite fonction.

Exercice 14 - Arguments par défaut

Soit la structure suivante :

```
/*...*/
struct point {
 int x;
 int y;
 int z;
};
```

fonction.h

Après avoir placé les déclarations de cette structure dans le fichier d'en-tête `fonction.h`, faire les modifications nécessaire pour simplifier le programme suivant en utilisant les nouvelles possibilités du C++. Ajouter aussi les déclarations qui semblent utiles dans le fichier d'entête.

```
#include "fonction.h"
/*...*/
void init(point* pt, int _x, int _y, int _z) {
 pt->x=_x; pt->y=_y; pt->z=_z;
}
void init(point* pt, int _x, int _y) {
 pt->x=_x; pt->y=_y; pt->z=0;
}
void init(point* pt, int _x) {
 pt->x=_x; pt->y=0; pt->z=0;
}
void init(point* pt) {
 pt->x=0; pt->y=0; pt->z=0;
}
void essai_init() {
 point p;
 init(&p);
 init(&p,1);
 init(&p,1,2);
 init(&p,1,2,3);
}
```

fonction.cpp

Exercice 15 - Allocation dynamique

Écrire plus simplement en C++ les instructions suivantes, en utilisant les opérateurs **new**, **new**[], **delete** et **delete**[] :

```
void essai_alloc(){
 int* pt_int;
 double* pt_double;
 pt_int=(int*)malloc(sizeof(int));
 pt_double=(double*)malloc(sizeof(double)*100);
 free(pt_int);
 free(pt_double);
}
```

fonction.cpp

Exercice 16 - Variables et fonctions `constexpr`

Expliquer pourquoi le programme suivant ne peut pas être compilé :

```
#include<array>
using namespace std;
int calcul(int x)  { return 2 * x + 1; }
int getNumber() { return 3; }
int main() {
 const int N = getNumber();
 array<int,calcul(N)> tableau;
 return 0;
}
```

main.cpp

Transformer le programme ci-dessus en utilisant des variables et des fonctions `constexpr` de façon à pouvoir le compiler.

Exercice 17 - Structures et tableaux

Soit la structure définie dans le fichier `fonction.h` :

```
/*...*/
struct personne {
 char nom[30];
 unsigned int age;
};
```

fonction.h

Question 1

Écrire une fonction `raz` qui permet d'initialiser le champ `nom` et le champ `age` d'une variable de type `personne` transmise en argument, respectivement avec la chaîne de caractères vide et la valeur 0.

Question 2

Écrire une fonction `affiche_struct` qui permet d'afficher les attributs d'une `personne` dont l'adresse (de type `const personne*`) est transmise en argument.

Question 3

Écrire une fonction `affiche_tab` qui permet d'afficher les attributs d'un tableau de `personne` dont l'adresse et le nombre d'éléments sont passés en argument.

Question 4

Écrire une fonction `init_struct` qui permet d'initialiser une structure dont l'adresse est passée en argument avec une chaîne de caractères et un entier passés en arguments. Ne pas utiliser de fonction de type `strcpy` pour cet exercice.

Question 5

Écrire une fonction `copy_struct` qui permet de copier les différents champs d'une variable de type `personne` donnée dans une autre variable de type personne.

Question 6

Écrire une fonction `copy_tab` qui permet de copier un tableau de variables de type personne dans un autre tableau du même type.

Question 7

Écrire une fonction qui utilise l'ensemble de ces fonctions.

Exercice 18 - Classe `string`

Transformer le programme précédent de manière à utiliser des variable de type `string` (entête `<string`) plutôt que des chaînes de caractères de type `C`.

Exercice 19 - Retour de fonction par référence

Soit le modèle de structure suivant :

```
#include<string>
struct compte {
 string id;
 int solde;
};
```

fonction.h

Ecrire une fonction operation qui permet de faire les opérations suivantes en leur donnant du sens.
Le premier argument de cette fonction est un tableau de de structure de type `compte`. Le deuxième argument est une chaîne de caractères qui permet d'identifier dans ce tableau le compte particulier à utiliser : il s'agit de celui dont le champs `id` est égal à cette chaîne. Il s'agit de faire en sorte que le solde du compte soit modifié conformément à l'opération souhaitée. On supposera que le compte voulu par l'utilisateur existe bien dans le tableau.

```
void essai_comptes(){
 compte tab[4]={ {"courant", 0},{"codevi", 1500 },
      {"epargne", 200 }, { "cel", 300 } };
 operation(tab,"courant")=100;
 operation(tab,"codevi")+=100;
 operation(tab,"cel")-=50;
 for(int i=0; i<4; i++) cout<<tab[i].id<<" : "<<tab[i].solde<<"\n";
}
```

fonction.cpp