Support de TP C++ - M1

André Garenne - 2021

1 Variables et opérateurs

1.1

Ecrire un programme qui initialise quatre variables aShort, aInt, aFloat et aDouble avec le type correspondant et les valeurs respectives 1, 2, 1.75 et 2.75. Puis ce programme affiche les types et valeurs de la façon suivante :

```
type aShort : [s] et valeur = 1
type aInt : [i] et valeur = 2
type aFloat : [f] et valeur = 1.75
type aDouble : [d] et valeur = 2.75
type de aShort*aDouble : [d] et valeur = 2.75
type aShort (si aShort=aShort*aDouble) : [s] et valeur = 2
```

1.2

Qu'affiche le programme suivant et pourquoi ?

```
#include <iostream>
#include <typeinfo>
using namespace std;
    int main(void){
    int i1=0;int i2=1;
    short s1=1;
    float f1=2.75;
    double d1=2.75;
    cout << i1*f1 << "\t" << f1*d1 << "\t" << (10%3==s1) << endl;
    cout << (s1==i2)*2 << "\t" << (i1==1 && ++i2==2) << "\t" << i2 << endl;
    cout << ((((f1*i2==d1*s1)*50)%9)==5*(1<=2))*123.456 << endl;
    return 0;
}
```

1.3

Qu'affiche le programme suivant ?

```
#include <iostream>
#include <typeinfo>
using namespace std;
    int main(void){
    int a[]={65,76,65,66,65,77,65};
    char b[]={65,108,97,98,97,109,97};
    cout << char(a[0]) << char(a[1]) << char (a[2]) << char(a[3]) << char(a[4])
<< char(a[5]) << char(a[6]) << endl;
    cout << b << endl;
    return 0;
}
```

1.4

Ecrire un programme qui affiche la liste des couples `code ascii : caractère` pour les valeurs 0 à 255. Pour mémoire le prototype du programme reste :

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

int main(void){
    ...
    return 0;
}
```

Ci-dessous un exemple pour les valeurs de 52 à 70 :

```
52 : 4
53 : 5
54 : 6
55 : 7
56 : 8
57 : 9
58 : :
59 : ;
60 : <
61 : =
62 : >
63 : ?
64 : @
65 : A
66 : B
67 : C
68 : D
69 : E
70 : F
```

1.5

Ecrire un programme qui demande à saisir une chaine de caractères sans espace et en utilisant un tableau de `char` de taille 50. Le programme affiche ensuite chacun des caractères avec sa position puis la chaine lue à l'envers. Exemple :

```
Saisir une chaine : salut
0 : s
1 : a
2 : l
3 : u
4 : t
Chaine inversee : tulas
```

Indications :

1 - Pour connaitre la longueur d'un tableau de `char` il est possible d'utiliser la commande `strlen()` disponible dans la bibliothèque `<cstring>` :

```
strlen("salut") // renvoie la valeur 5
```

2 - une fois que la chaine inversée a été réalisée il faut lui ajouter en position finale (après la dernière lettre) un caractère de terminaison. Sous Windows par exemple c'est la valeur 0.

```cpp
#include <iostream>
#include <cstring> // ajout de la bibliothèque cstring pour avoir strlen()
using namespace std;

int main(void){
    char str_1[50]; // création d'un tableau de 50 caractères
    ...
    return 0;
}
```

2 Entrées-sorties, structures de contrôle

2.1

Ecrire un programme qui attend la saisie d'une série de notes entre 0 et 20. Quand une valeur négative est saisie le programme s'arrête et affiche le nombre de notes, la note minimale, la note maximale et la moyenne des notes. Le programme ne doit pas utiliser de variable dimensionnée. Exemple :

```
Saisir la note (-1 pour arrêter) : 12
Saisir la note (-1 pour arrêter) : 14
Saisir la note (-1 pour arrêter) : 7
Saisir la note (-1 pour arrêter) : 5
Saisir la note (-1 pour arrêter) : 1
Saisir la note (-1 pour arrêter) : 8
Saisir la note (-1 pour arrêter) : 17
Saisir la note (-1 pour arrêter) : -1
Nombre de notes : 7
Note minimale : 1
Note maximale : 17
Moyenne des notes : 9.14286
```

2.2

Même exercice que précédemment mais en ajoutant des noms et toujours sans utiliser de variable dimensionnée. Exemple :

```
Saisir le nom (stop pour arreter) : machin
Saisir la note : 12
Saisir le nom (stop pour arreter) : truc
Saisir la note : 16
Saisir le nom (stop pour arreter) : bidule
Saisir la note : 5
Saisir le nom (stop pour arreter) : stop
Nombre de notes : 3
La note minimale c'est bidule avec 5
La note maximale c'est truc avec 16
Moyenne des notes : 11
```

2.3

En vous aidant du paragraphe 1.12 du cours, écrire un programme qui demande à l'utilisateur de saisir un nombre n puis une loi de distribution puis les paramètres de cette loi de distribution. Il affichera ensuite n valeurs aléatoires générées selon la loi de distribution choisie :

> Remarque : pour créer un générateur suivant une loi normale de moyenne m et d'écart-type sd il faudra utiliser la commande **normal_distribution < > nd(m,sd);**. De même pour créer un générateur suivant une loi uniforme donc les valeurs seront comprises dans l'intervalle $[A, B[$ il faudra utiliser la commande **uniform_real_distribution < > urd(A,B);**

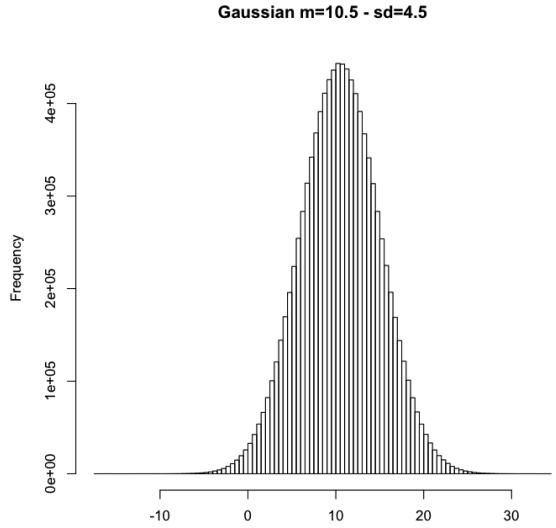
Exemple d'exécution 1 :

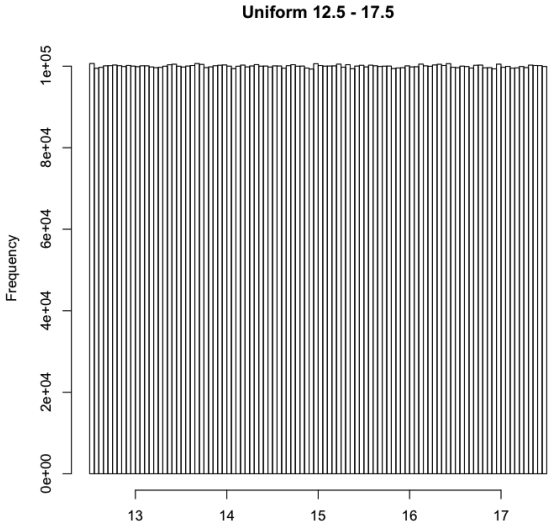
```
Nombre de valeurs aleatoires : 6
(1) distribution normale, (2) distribution uniforme : 1
Moyenne : 10.5
Ecart-type : 4.5
4.92119
8.66638
17.754
12.299
16.7226
10.7354
```

Exemple 2 :

```
Nombre de valeurs aleatoires : 6
(1) distribution normale, (2) distribution uniforme : 2
Min : 12.5
Max : 17.5
12.9252
16.9581
13.4484
14.49
16.2176
15.3019
```

Si nous traçons ces valeurs de paramètres pour 10^7 valeurs nous obtenons les résultats suivants :





2.4

Le jeu du nombre secret. Le programme doit générer un nombre entre 1 et 20 sans vous le communiquer. Il attend ensuite que vous saisissiez des valeurs entières dans cet intervalle. Il répond alors selon les cas que votre nombre est trop grand ou trop petit. Quand le nombre est trouvé le programme s'arrête en vous signalant le nombre d'essais que vous avez réalisé :

> Remarque : pour initialiser un générateur de nombres pseudo-aléatoires (**default_random_engine**) sur des valeurs différentes à chaque fois (imprévisibles) une solution est de passer un objet **random_device** : **random_device rd; default_random_engine de(rd());**

Le programme doit se comporter comme suit :

```
Entrer une valeur : 10
Trop grand
Entrer une valeur : 5
Trop petit
Entrer une valeur : 8
Trop grand
Entrer une valeur : 7
Trop grand
Entrer une valeur : 6
Bravo, trouve en 5 essais
```

2.5

Ecrire un programme qui demande à l'utilisateur de saisir le prix d'un kg de pâtes et combien il en achète, puis celui d'un rouleau de papier toilette ainsi que le nombre acheté. Le programme demande ensuite de saisir un code TVA, soit (A) soit (B) et calcule et affiche le prix HT et le prix TTC.

Exemple :

```
Combien coute un kilo de pates : 1.5
Combien de kilos avez-vous achetes : 1
Combien coute un rouleau de papier toilette : 2
Combien de rouleaux avez-vous achetes : 2
Quelle est la TVA sur ces 2 produits : 5.5% (A), 19.6% (B) a
Saisir (A) ou (B) svp
Quelle est la TVA sur ces 2 produits : 5.5% (A), 19.6% (B) C
Saisir (A) ou (B) svp
Quelle est la TVA sur ces 2 produits : 5.5% (A), 19.6% (B) A
Le prix HT est : 5.5 euros et le prix TTC : 5.8025 euros
```

2.6

Ecrire un programme qui saisit une valeur entière puis calcule n tel que

$$\Sigma_{i=1}^{n} < value$$

Exemple :

```
Saisir une valeur : 100
Valeur de n : 13 avec une somme de : 91
```

2.7

Ecrire un programme qui demande à l'utilisateur de saisir un entier `n` et qui affiche `U(n)` tel que :

$$u_{(0)} = 1,\ u_{(1)} = 1\ et\ u_{(n+1)} = u_{(n)} + u_{(n-1)}$$

Exemple :

```
Saisir la valeur de n : 10
U[10] : 89
```

3 Fonctions

3.1

Ecrire une fonction factorielle sous la forme facto(n). La tester avec les exemples suivants :

```
factorielle(5)=120
factorielle(11)=39916800
```

3.2

Ecrire une fonction factorielle sous la forme **factoRec(n)** mais qui utilise un algorithme récursif. La tester avec les exemples suivants :

```
factorielle récursive(5)=120
factorielle récursive(11)=39916800
```

4 Pointeurs

4.1

Ecrire une fonction **change_pas()** et une fonction change() qui prennent chacune un entier en paramètre, multiplient cet entier par 3 et renvoient le résultat. La première ne doit pas changer le contenu de la variable passée mais la seconde si. La tester avec les exemples suivants :

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

...

int main(void){
    int a=12;
    cout << "change_pas(a) renvoie " << change_pas(a) << " mais : a = " << a <<
endl;
    cout << "change(a) renvoie " << change(a) << " mais : a = " << a << endl;
    return 0;
}
```

Le programme doit se comporter comme suit :

```
change_pas(a) renvoie 36 mais : a = 12
change(a) renvoie 36 mais : a = 36
```

4.2

En vous inspirant de l'exercice précédent écrire les fonctions **f1()**, f2() et **f3()**. Les tester avec les exemples suivants :

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

...

int main(void){
int* a = new int(12);
    cout << "f1(*a) renvoie " << f1(*a) << " mais : *a = " << *a << endl;
    cout << "f2(*a) renvoie " << f2(*a) << " mais : *a = " << *a << endl;
    cout << "f3(a) renvoie " << f3(a) << " mais : *a = " << *a << endl;
    delete a;
    return 0;
}
```

Le programme doit se comporter comme suit :

```
f1(*a) renvoie 36 mais : *a = 12
f2(*a) renvoie 36 mais : *a = 36
f3(a) renvoie 108 mais : *a = 108
```

5 Lecture/écriture de fichiers

5.1

Créer une fonction **write_data()** qui prend en argument deux chaines de caractères, utilise la première comme chemin + nom de fichier et la seconde comme ce que doit contenir ce fichier. La fonction renvoie un **void**. Elle écrit le contenu de la seconde chaine dans le nom de fichier désigné par la première et renvoie une erreur sur le flux d'erreur standard si l'écriture est impossible :

```
#include <iostream>
...
using namespace std; // utilisation de l'espace de nom de std

void write_data(string file, string content){
...
}

int main(){
    string filename="data_01.txt";
    string content="nom\tprenom\nAA\taa\nBB\tbb\nCC\tcc\n";
    write_data(filename,content);
    return 0;
}
```

Le programme doit se comporter comme suit :

```
nom prenom
AA  aa
BB  bb
CC  cc
```

5.2

Créer une fonction **read_data()** qui prend en argument une chaines de caractères contenant un chemin + nom de fichier, qui lit ce fichier et en retourne le contenu via une string :

```
#include <iostream>
...
using namespace std; // utilisation de l'espace de nom de std

string read_data(string file){
...
}

int main(){
    string filename="data_01.txt";
    string content=read_data(filename);
    cout << content << endl;
    return 0;
}
```

Le programme doit se comporter comme suit :

```
nom prenom
AA  aa
BB  bb
CC  cc
```

6 Chaines de caractères

6.1

Créer une fonction **check_mail()** qui prend en argument une chaine de caractères et qui renvoie **true** si la chaine contient "**@**" et **false** sinon :

```
#include <iostream>
#include <string>
using namespace std; // utilisation de l'espace de nom de std

bool check_mail(string mail){
...
}

int main(){
    string mail1="aaa.bbb@ccc.ddd";
    string mail2="aaa.bbb_ccc.ddd";
    string mail3="aaa@ccc.ddd";
    string mail4="aaa.bbb@ccc";
    if (check_mail(mail1)) cout << mail1 << " contient @" << endl;
    if (check_mail(mail2)) cout << mail2 << " contient @" << endl;
```

```
    if (check_mail(mail3)) cout << mail3 << " contient @" << endl;
    if (check_mail(mail4)) cout << mail4 << " contient @" << endl;
    return 0;
}
```

Le programme doit se comporter comme suit :

```
aaa.bbb@ccc.ddd contient @
aaa@ccc.ddd contient @
aaa.bbb@ccc contient @
```

6.2

Créer une fonction **display()** qui prend en arguments un **vector** de **string** et qui affiche son contenu à raison d'une ligne par élément. Créer une fonction **get_mails()** qui demande à l'utilisateur de saisir des adresses mails. Quand l'utilisateur saisit une chaine sans "**@**" une erreur est signalée et le contenu non conservé. Quand l'utilisateur rentre "stop" la saisie s'arrête et le vecteur des chaines de caractères est retourné par la fonction :

```
#include <iostream>
#include <string>
#include <vector>
using namespace std; // utilisation de l'espace de nom de std
bool check_mail(string mail){
...
}

void display(vector<string> value){
...
}

vector<string> get_mails(void){
...
}

int main(){
    vector<string> result=get_mails();
    display(result);
    return 0;
}
```

Le programme doit se comporter comme suit :

```
entrer le mail : abc@gh.j
entrer le mail : dgst.j
il manque un @
entrer le mail : dgst@gt.i
entrer le mail : dsr@uj.qz
entrer le mail : stop
abc@gh.j
dgst@gt.i
dsr@uj.qz
```

6.3

Créer une fonction supplémentaire **mail_tokens()** qui prend une chaine de caractères en argument, vérifie si elle contient bien un "**@**" et renvoie ensuite un **vector** de **string** contenant tous les éléments de la chaines séparés par les "**@**" ou des "." :

```
#include <iostream>
#include <string>
#include <vector>
using namespace std; // utilisation de l'espace de nom de std

bool check_mail(string mail){
...
}

void display(vector<string> value){
...
}

vector<string> mail_tokens(string mail){
...
}

int main(){
    string mail1="aaa.bbb@ccc.ddd";
    string mail2="aaa.bbb_ccc.ddd";
    string mail3="aaa@ccc.ddd";
    string mail4="aaa.bbb@ccc";
    cout << mail1 << endl;
    display(mail_tokens(mail1));
    cout << mail2 << endl;
    display(mail_tokens(mail2));
    cout << mail3 << endl;
    display(mail_tokens(mail3));
    cout << mail4 << endl;
    display(mail_tokens(mail4));
    return 0;
}
```

Le programme doit se comporter comme suit :

```
aaa.bbb@ccc.ddd
aaa
bbb
ccc
ddd
aaa.bbb_ccc.ddd
aaa@ccc.ddd
aaa
ccc
ddd
aaa.bbb@ccc
aaa
bbb
ccc
```

6.4

Créer une fonction **split()** qui prend en argument une chaine de caractères et un caractère de séparation sep et qui renvoie un vecteur de string découpée autour des sep :

```
#include <iostream>
#include <string>
#include <vector>
using namespace std; // utilisation de l'espace de nom de std

void display(vector<string> value){
...
}

vector<string> split(string value, string sep){
...
}

int main(){
    string str1="aaa,bbb,ccc,ddd";
    string str2="111_222_333";
    cout << str1 << endl;
    display(split(str1,","));
    cout << str2 << endl;
    display(split(str2,"_"));
    return 0;
}
```

Le programme doit se comporter comme suit :

```
aaa,bbb,ccc,ddd
aaa
bbb
ccc
ddd
111_222_333
111
222
333
```

7 Tableaux dynamiques

7.1

Créer trois fonctions. Une fonction **str_to_double()** qui prend un **string** et le convertit en **double** et le retourne, une fonction **vstr_to_double()** qui prend un **vector** de **string** et renvoie le **vector** de **double** issue de leur conversion et enfin une fonction **mean()** qui prend un **vector** de **double** et retourne leur moyenne arithmétique. Pour la conversion il faudra vous référer à la gestion d'exceptions mentionnée à 1.9 :

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
using namespace std;
```

```
vector<string> split(string value, string sep){
...
}

string read_data(string file){
...
}

double str_to_double(string value){
...
}

vector<double> vstr_to_double(vector<string> values){
...
}

double mean(vector<double> values){
...
}

int main(){
    string data=read_data("data_02.txt");
    vector<double> notes=vstr_to_double(split(data,"\n"));
    cout << "moyenne : " << mean(notes) << endl;
    return 0;
}
```

Le programme doit se comporter comme suit :

```
12
20
7.5
8
11
12
2
2.5
12
7
moyenne : 9.4
```

7.2

En vous aidant du paragraphe page du cours (*Quelques compléments sur les arguments de la fonction main()*) écrire un programme qui va manipuler des fichiers de données de la façon suivante.

Le fichier **data_03.txt** contient des données sur 3 colonnes espacées avec des tabulations. Les valeurs sont des réels avec 3 chiffres après la "," qui est ici le séparateur décimal.

```
x cos(x)  sin(x)
0,000   1,000   0,000
0,100   0,995   0,100
...
9,800   -0,930  -0,366
9,900   -0,889  -0,458
```

Ecrire un programme `ex_7_2` qui va être utilisable comme suit :

```
ex_7_2 data_03.txt new_data.txt
```

la commande `ex_7_2` va donc prendre en argument deux noms de fichiers. Le premier **data_03.txt** va être lu et toutes les occurrences de "," vont être remplacées par des "." et la version modifiée du fichier va être sauvegardée dans le nom de fichier passé en second argument, ici **new_data.txt**. Si l'on affiche ce second fichier on doit donc avoir :

```
x cos(x)  sin(x)
0.000   1.000   0.000
0.100   0.995   0.100
...
9.800   -0.930  -0.366
9.900   -0.889  -0.458
```

Remarque : vous pouvez utiliser des fonctions comme `string::find()` et `string::replace()` mais ce n'est pas nécessaire pour résoudre ce problème.

7.3

En vous aidant de l'exercice précédent écrivez un programme qui prend en argument successivement un caractère `c1`, un caractère `c2`, un nom de fichier à lire `f1` et un nom de fichier à écrire `f2`. Il lit le contenu de `f1`, remplace toutes les occurrences de `c1` trouvées dedans par `c2` et sauvegarde le résultat dans `f2`. Par exemple :

```
ex_7_3 "," "." data_04.txt new_data.txt
```

doit donner le même résultat que la commande :

```
ex_7_2 data_04.txt new_data.txt
```

Remarque : il faut ajouter des "" autour des "." et "," lorsque l'on passe les arguments en ligne de commande car sinon ces caractères sont considérés comme des séparateurs.

```
data=read.table("new_data.txt",header=T)
matplot(data[,1],data[,2:6],type="l",lty=1,lw=2,xlab="Temp (h)",ylab="[ ]
microM")
legend("top", colnames(data[,2:6]),col=1:4,cex=0.8,fill=1:4)
```