Feuille 1

TD de Cpp

Exercice 1 :

Ecrire un programme qui affiche les nombres de 1 à 100. Mais qui pour les nombre multiples de 3 affiche "Fizz" au lieu du nombre et pour les nombres multiples de 5 affiche "Buzz" au lieu du nombre. Pour les nombres qui sont multiples de 3 et de 5, on affichera "FizzBuzz".

Solution de l'exercice 1 :

```cpp
#include <iostream>
#include <cmath>

int main() {
    for (int n=1; n <= 100; n++) {
        if (n%3 == 0) std::cout << "Fizz";
        if (n%5 == 0) std::cout << "Buzz";
        if (n%3 != 0 && n%5 != 0) std::cout << n;
        std::cout << std::endl;
    }
    return 0;
}
```

Exercice 2 :

Ecrire de deux manières différentes un programme qui calcule x^n où x est un réel et n un entier. On n'utilisera pas la fonction `pow`.

Solution de l'exercice 2 :

```cpp
#include <iostream>
#include <cmath>

// exponentiation naive
double pow1(double x, int n){
  double p = 1;
  int i;
  for(i=0; i<n; i++)
    p *= x;
  return p;
}

//exponentiation rapide
double pow2(double x, int n){
  double v = 1.;
  double p = x;

  while(n > 0){
```

```
    if (n & 1) v *= p;
    p *= p;
    n >>= 1;
  }
  return v;
}

int main(){
  double x = 52;
  int n = 7;
  std::cout << "pow1 : " << pow1(x, n) << std::endl;
  std::cout << "pow2 : " << pow2(x, n) << std::endl;
  return 0;
}
```

Exercice 3 :

Ecrire un programme qui à partir d'un tableau d'entiers calcule et écrit

1. la valeur minimale
2. la valeur maximale
3. la moyenne (qui peut ne pas être une valeur entière !)

Solution de l'exercice 3 :

```
#include <iostream>
#include <cmath>

int main()
{
    int i;
    const int dim = 5;
    int T[dim] = {1, 2, 3, 4, 5};
    int min = T[0], max = T[0];
    double moy = T[0];

    for(i=1; i<dim; i++){
        min = (min > T[i])? T[i]: min;
        max = (max < T[i])? T[i]: max;
        moy += T[i];
    }
    moy /= dim;

    std::cout << "La valeur minimale est " << min << std::endl;
    std::cout << "La valeur maximale est " << max << std::endl;
    std::cout << "La moyenne est          " << moy << std::endl;
    return 0;
}
```

Exercice 4 :

100 portes dans un couloir sont initialement fermées. Vous allez faire 100 passages par ce couloir. La première fois, vous visitez chacune des portes et changez l'état de la porte (si elle est ouverte, vous la fermez et,

inversement, si elle est fermée, vous l'ouvrez). La seconde fois, vous visitez toutes les portes multiples de 2. La troisième fois, vous visitez toutes les portes multiples de 3, etc, jusqu'à ce que vous visitiez seulement la porte 100.
Dans quelle configuration sont les portes après le dernier passage ?

Solution de l'exercice 4 :

```
#include <iostream>

void print(int N, int *T){
    // print
    int n, no=0, nc=0;
    for (n=0; n<N; n++){
        no += T[n];
    }
    nc = N-no;
    std::cout << "Il␣y␣a␣";
    std::cout << no << "␣portes␣ouvertes␣et␣";
    std::cout << nc << "␣portes␣fermees␣!" << std::endl;
    std::cout << "Les␣portes␣ouvertes␣sont␣" << std::endl;
    int i=0;
    for (n=0; n<N; n++){
        if (T[n] == 1){
            i++;
            std::cout << n << "␣(" << i << "^2=" << i*i << ")␣";
        }
    }
    std::cout << std::endl;
}

int main(){
    const int N = 100;
    int T[N];
    int p, n;
    // initialization
    for (n=0; n<N; n++){
        T[n] = 0;
    }
    // open and close the doors
    for (n=0; n<N; n++){
        for (p=0; p<N; p += n+1){
            T[p] = 1-T[p];
        }
    }
    print(N, T);
    return 0;
}

/*
Explication :
on decompose le nombre en facteurs premiers
n = prod a_k^p_k
La porte sera ouverte ssi le nombre de diviseurs de n est impair
Or le nombre de diviseurs est
```

```
prod (p_k+1)
donc la porte est ouverte si tous les p_k sont pairs !
*/
```

Exercice 5 :

Ecrire un programme qui donne une représentation LCD d'une donnée entière en utilisant une grille 3×3 d'espace, d'underscore et de pipe pour chaque chiffre. Chaque chiffre est montré ci-dessous en remplaçant les espaces par des points.

```
 ._.   ...   ._.   ._.   ...   ._.   ._.   ._.   ._.   ._.
 |.|   ..|   ._|   ._|   |_|   |_.   |_.   ..|   |_|   |_|
 |_|   ..|   |_.   ._|   ..|   ._|   |_|   ..|   |_|   ._|
```

Solution de l'exercice 5 :

```
#include <iostream>

const int nbrow=3, nbchar=3;
char LCD[nbrow][10][nbchar+1] = {
    {"␣_␣", "␣␣␣", "␣_␣", "␣_␣", "␣␣␣", "␣_␣", "␣_␣", "␣_␣", "␣_␣", "␣_␣"},
    {"|␣|", "␣␣|", "␣_|", "␣_|", "|_|", "|_␣", "|_␣", "␣␣|", "|_|", "|_|"},
    {"|_|", "␣␣|", "|_␣", "␣_|", "␣␣|", "␣_|", "|_|", "␣␣|", "|_|", "␣_|"}
};

void affiche(int c){
    std::cout << "Nous␣allons␣afficher␣" << c << std::endl;
    int row;
    // preparation
    int ctemp=c, nbdigits=0, invc=0;
    while (ctemp>0){
        nbdigits++;
        invc *= 10;
        invc += ctemp%10;
        ctemp /= 10;
    }
    std::cout << "Nombre␣de␣chiffres␣:␣" << nbdigits << std::endl;
    std::cout << c << "␣dans␣l'autre␣sens␣s'ecrit␣" << invc << std::endl;
    // affichage ligne par ligne
    for (row=0; row<nbrow; row++){
        ctemp = invc;
        int i;
        for (i=0; i<nbdigits; i++)
        {
            std::cout << LCD[row][ctemp%10] << "␣";
            ctemp /= 10;
        }
        std::cout << std::endl;
    }

}

int main(){
```

```
    int c;
    std::cout << "Entrez un nombre : ";
    std::cin >> c;
    affiche(c);
    return 0;
}
```