UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Méthodes informatiques pour physiciens

introduction à C++ et
résolution de problèmes de physique par ordinateur

Corrigé 7

Professeur : Alessandro Bravar
Alessandro.Bravar@unige.ch

Université de Genève
Section de Physique

Semestre de printemps 2015

Références :

E. Lazard
Architecture de l'ordinateur

H. Michels
DISLIN : A Data Plotting Library

`http://dpnc.unige.ch/~bravar/C++2015/L7` :
pour les notes du cours, les exercices et les corrigés

7.1 Exercices

Exercice 1 : Exprimez les nombres décimaux 94, 141, 163, 191 en base 2, 8 et 16 !

La représentation en base *2* peut être obtenue par divisions successives par 2. La valeur du reste donne la valeur du bit à partir par le bit plus a droit (poids faible).
La valeur maximal que peut être représentait avec n bits est $2^n - 1$. Par exemple pour reprśenter le nombre 94 en base 2 on a besoin de 7 bits ($\text{int}(\log_2 94)+1 = 7$). Pour obtenir la valeur du bit $b-1$ il faut diviser le nombre par 2^{b-1}.

Bit	Division	Quotient	Reste
n=7 ($2^7 = 128$)			
0	94/2	47	0
1	47/2	23	1
2	23/2	11	1
3	11/2	5	1
4	5/2	2	1
5	2/2	1	0
6	1/2	0	1
Nombre binaire			1011110

Bit	Division	Quotient	Reste
n=8 ($2^8 = 256$)			
0	141/2	70	1
1	70/2	35	0
2	35/2	17	1
3	17/2	8	1
4	8/2	4	0
5	4/2	2	0
6	2/2	1	0
7	1/2	0	1
Nombre binaire			10001101

Bit	Division	Quotient	Reste
n=8 ($2^8 = 256$)			
0	163/2	81	1
1	81/2	40	1
2	40/2	20	0
3	20/2	10	0
4	10/2	5	0
5	5/2	2	1
6	2/2	1	0
7	1/2	0	1
Nombre binaire			10100011

Bit	Division	Quotient	Reste
n=8 ($2^8 = 256$)			
0	191/2	95	1
1	95/2	47	1
2	47/2	23	1
3	23/2	11	1
4	11/2	5	1
5	5/2	2	1
6	2/2	1	0
7	1/2	0	1
Nombre binaire			10111111

Si la base est 8 le problème est résolu à partir du nombre binaire : il suffit de grouper les bits 3 par 3. Par exemple, le nombre 94 écrit en base 8 est composé de 3 chiffres octales :

nombre binaire = $\overbrace{1}^{\text{chiffre 3}}\ \underbrace{011}_{\text{chiffre 2}}\ \overbrace{110}^{\text{chiffre 1}}$ $\qquad 1011110_2 = 136_8$

	chiffre 3	chiffre 2	chiffre 1
Nombre binaire	10	101	110
Nombre octal	1	3	6

	chiffre 2	chiffre 2	chiffre 1
Nombre binaire	10	001	101
Nombre octal	2	1	5

	chiffre 3	chiffre 2	chiffre 1
Nombre binaire	10	100	011
Nombre octal	2	4	3

	chiffre 3	chiffre 2	chiffre 1
Nombre binaire	10	111	111
Nombre octal	2	7	7

Si la base est 16 le problème est résolu dans la même manière sauf qui il faut grouper le bits 4 par 4. Il faut aussi se souvenir que dans la base 16, 10 est représenté avec A, 11 est

représenté avec B, etc.

	chiffre 2	chiffre 1
Nombre binaire	1010	1110
Nombre hexadécimal	5	E

	chiffre 2	chiffre 1
Nombre binaire	1000	1101
Nombre hexadécimal	8	D

	chiffre 2	chiffre 1
Nombre binaire	1010	0011
Nombre hexadécimal	A	3

	chiffre 2	chiffre 1
Nombre binaire	1011	1111
Nombre hexadécimal	B	F

Exercice 2 : Donnez sur 8 bits les représentations "signe et valeur absolue" et complément à 2 des nombres décimaux -47, -99, -102 et -118 !

Pour obtenir le complément à 2 d'un nombre binaire, on inverse tous les bits puis on y rajoute 1. Prenons l'exemple de 47 exprimé sur 8 bits :
$47_{10} = 00101111_2$ et on inverse les bits : $00101111 \rightarrow 11010000$ puis on y ajoute 1 :
$C_2(47) = 11010001$. On a laiss tomber le retenu.

	signe et valeur absolue	complément à 2
-47	10101111	11010001
-99	11100011	10011101
-102	11100110	10011010
-118	11110110	10001010

Exercice 3 : Démontrez les propriétés du complément à 2 !

En base binaire sur n bits, la valeur X est représenté par :

$$X = \sum_{i=0}^{n-1} x_i 2^i$$

où x_i peut prendre seulement les valeurs 0 ou 1. Le complément à 2 revient à inverser tous les bits et ajouter 1. $C_2(X)$ sera représenté par :

$$C_2(X) = \sum_{i=0}^{n-1} \left(1 - x_i 2^i\right) + 1 \; .$$

La somme de X et de son complément sont alors donnés par :

$$X + C_2(X) = \sum_{i=0}^{n-1} x_i 2^i + \sum_{i=0}^{n-1} (1 - x_i) 2^i + 1 = \sum_{i=0}^{n-1} x_i 2^i + \sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} x_i 2^i + 1 = (2^n - 1) + 1 = 2^n$$

donc

$$C_2(X) = 2^n - X$$

et

$$C_2(C_2(X)) = 2^n - (2^n - X) = X \; .$$

Exercice 4 : Variable `short`

Le type `short` est enregistré sur 2 octets (16 bits). Le plus grand entier négatif que l'on peut ainsi enregistrer est 1 suivi de 15 zéros, c'est-á-dire -32768. Si dans un programme la valeur enregistrée dépasse la plage correspondante au type de la variable considérée, le programme retourne des valeurs erronées pour les entiers, ±inf ou NaN (e.g. division par zéro) pour les *floats*.

Short.cpp

```cpp
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  short int m = 0;
  int n = 0;
  cout << "short \t\t long" << endl;
  for (int i=0; i<100; i++) {
    cout << m << "\t\t" << n << endl;
    m = m - 10000;
    n = n - 10000;
  }

  return 0;
}
```

Les premières 6 lignes imprimées sont affichées ci dessous. Dans le 5ème itŕation on est à -40000, qui dépasse la plage admissible pour le variables de type `short` (-32768). La différence -40000 - (-32768) = -7232 est ajoutée au plus grand entier positif `short` (+32768) et ça nous donne 25536.

short	long
0	0
-10000	-10000
-20000	-20000
-30000	-30000
25536	-40000

Exercice 5 : Erreurs d'arrondi

Les erreurs d'arrondi viennent du fait que les variables `float` (ou `double`) ne sont que des approximations des nombres réels. Dans cet exercice le calcul algébrique donne `d = a`, mais le programme affiche `335544`. Ça se produit parce que les `float` ont une précision limitée. Quand le programme calcule `c = 1/b - 1` le résultat est très petit ($\sim 3 \cdot 10^{-6}$), donc il y a un erreur d'arrondi. Si on utilise `double` au lieu de `float` le résultat est correct.

Arrondi.cpp

```cpp
//erreur d'arrondi
#include <iostream>

using namespace std;

int main() {
  float a = 333333.;
  float b = 1. - 1./a;
  float c = 1./b - 1.;
  float d = 1./c + 1.;
  cout << "La valeur algebrique de d est 333333 " << endl
       << "tandis que la valeur calculee (float) est " << d << endl;

  double w = 333333.;
  double x = 1. - 1./w;
  double y = 1./x - 1.;
  float z = 1./y + 1.;
  cout << "La valeur algebrique de z est 333333 " << endl
```

```
19        << "tandis que la valeur calculee (double) est " << z << endl;
20
21    return 0;
22  }
```

Exercice 6 : Erreurs d'arrondi

Pour $U_0 = 3$ (1/3) on trouve $U_1 = 3$ (1/3), c'est-á-dire $U_{n+1} = U_n$, la suite est donc constante. Le programme **Suite.cpp** montre pourtant que la suite converge toujours vers 3! En effet, 1/3 n'est pas un point fixe de la suite, il suffit d'une petite variation pour que la suite converge vers le point fixe obtenu avec la valeur 3.

Suite.cpp

```
1  #include <iostream>
2  #include <iomanip>
3
4  using namespace std;
5
6  int main() {
7    double first;
8    cout << "Entrez le premier element de la suite : ";
9    cin >> first;
10   cout << endl << setw(25) << setprecision(20) << first << endl;
11
12   int steps = 100;
13   double next = first;
14   for (int i=0; i<steps; i++) {
15     cout << i << " : " << setw(25) << setprecision(20) << next << endl;
16     next = -3.*next*next/8. + 9.*next/4. - 3./8.;
17     //next = -4.*next*next/9. + 26.*next/9. - 4./9.;
18   }
19
20   return 0;
21 }
```

7.2 Exercices DISLIN

1. **Dessinez la fonction $f(x) = \frac{1}{1+x^2}$ entre 1 et 100 avec 100 points ! Au lieu de `qplot`, essayez `qplsca`.**

   Le programme **Fonc_D.cpp** définit une constante (`POINTS`) au début du programme qui indique le nombre des points des graphiques. Ensuite, il calcule les valeurs de la fonction donnée avec une boucle (tabulation). Après le programme utilise la fonction `metafl("XWIN");` pour dessiner à l'écran les graphiques en utilisant la fonction `qplsca(x,y, POINTS);` qui dessine à chaque point un symbole et `qplot(x,y,POINTS);` qui relie les points avec une ligne.

   Fonc_D.cpp.

```
1  #include <iostream>
2  #include "dislin.h"      //bibliotheque DISLIN
3
4  using namespace std;
5
6  //declaration la fonction a dessiner
7  double fonct(double x);
8
9  int main() {
10   cout << "Dessin d'une fonction" << endl;
```

```
11
12    //tabulation de la faonction
13    const int POINTS = 100;
14    double x[POINTS+1];
15    double y[POINTS+1];
16    double dx = 1.;
17    //boucle pour remplir les tableaux x et f(x)
18    for (int i=0; i<=POINTS; i++) {
19      x[i] = i * dx;
20      y[i] = fonct(x[i]);
21    }
22
23    //affichage sur l'ecran
24    metafl("XWIN");
25    qplot(x,y,POINTS);
26
27    return 0;
28  }
29
30  //definition de la fonction a dessiner
31  double fonct(double x) {
32    double y = 1./(1.+x*x);
33    return y;
34  }
```

2. **Dessinez sin(x) et cos(x) sur le même graphe !**
   Le programme **Sinus_D.cpp** utilise la bibliothèque **dislin** incluse avec la prèdirective `#include "dislin.h"`. Au début il faut définir une constante (`N`) qui indique le nombre des points des graphiques. Après le programme tabule la fonction **sinus** et **cosinus** avec une boucle. Pour afficher les graphiques sur l'écran on utilise la fonction `metafl("XWIN")`. Le graphe est "ouvert" par la fonction `disini()` et terminé par la fonction `disfin()`. Tout ce qui se situe entre ces deux fonctions concerne le même graphe. Après on introduit les étiquettes des axes du graphe avec les fonctions `name("axe X","X")` et `name("axe Y","Y")`. Ensuite on utilise les fonctions `titlin("Dessin de f(x)",1)` et `titlin("f(x) = sin(x)",3); titlin("f(x) = cos(x)",4)` pour afficher le titre du graphe. Les dimensions des axes sont définies par la fonction `graf(0., 2.*M_PI, 0., M_PI/4., -1., 1., -1., 0.25);`. Les premiers 4 paramètres indiquent les limites de l'axe x, les derniers 4 de l'axe y. La fonction `thkcrv(10)` définit l'épaisseur de la courbe à dessiner et `color("green")` sa couleur. Enfin, la fonction `curve(x,sinx,N+1)` dessine la courbe.
   Sinus_D.cpp.

```
1   #include <iostream>
2   #include <cmath>
3   #include "dislin.h"      //bibliotheque DISLIN
4
5   using namespace std;
6
7   int main() {
8     cout << "Dessin de sin(x) et cos(x) sur l'intervalle [0-2PI]" << endl;
9
10    //tabulation des fonctions sin et cos
11    const int N = 1000;
12    double x[N+1];
13    double sinx[N+1];
14    double cosx[N+1];
15
16    for (int i=0; i<=N; i++) {
```

```
17      x[i] = i*2.*M_PI/N;    //x
18      sinx[i] = sin(x[i]);    //sin(x)
19      cosx[i] = cos(x[i]);    //cos(x)
20    }
21
22    //initialization de dislin
23    metafl("XWIN");      //sortie sur l'ecran
24    disini();
25    name("axe X","X");
26    name("axe Y","Y");
27    titlin("Dessin de f(x)",1);
28    titlin("f(x) = sin(x)",3);
29    titlin("f(x) = cos(x)",4);
30
31    //dessin du systeme des coordonnees :
32    //xmin, xmax, xxx, xxx, ymin, ymax, yyy, yyy
33    graf(0., 2.*M_PI, 0., M_PI/4. , -1. , 1. , -1. , 0.25);
34    title();
35
36    //dessin des fonctions
37    thkcrv(10);
38    color("green");
39    curve(x,sinx,N+1);
40    color("red");
41    curve(x,cosx,N+1);
42    color("fore");
43    grid(1,1);
44
45    //fin de DISLIN
46    disfin();
47
48    return 0;
49  }
```

3. **Dessinez la fonction $f(x,y) = x^2 + y^2$ entre $-10 < x < 10$ et $-10 < y < 10$ avec 21×21 points !**
   Fonc2D_D.cpp.

```
1   //dessin d'une fonction bi-dimensionnelle
2   #include <iostream>
3   #include <cmath>
4   #include "dislin.h"      //bibliotheque DISLIN
5
6   using namespace std;
7
8   int main() {
9     const int POINTSX = 21, POINTSY = 21;
10    double z[POINTSX][POINTSX];
11    double x, y;
12    double dx = 1, dy = 1;
13    //tabulation de la fonction
14    for (int i=0; i<POINTSX; i++)
15      for (int j=0; j<POINTSY; j++) {
16        x = -10. + i*dx;
17        y = -10. + j*dy;
18        z[i][j] = x*x + y*y;
19      }
20
21    metafl("XWIN");
```

```
22    disini();
23    //dessin du graphique
24  //  titlin("f(x,y) = x*x + y*y",4);
25    graf3d(-10.,10.,-10.,5.,-10.,10.,-10.,5.,0.,250.,0.,25.);
26  //  title();
27    color("GREEN");
28
29    //passage du tableau z par pointeur
30    //&z[0][0] donne l'adresse memoire du premier element du tableau
31    qplsur(&z[0][0],POINTSX,POINTSY);
32
33    disfin();
34
35    return 0;
36  }
```