Algorithmique et Programmation 2
Travaux Pratiques – Séance 4

À déposer à la fin de la présente séance sur Arche

1 Opérations primitives

Dans cette séance de TP nous représentons un nombre complexe $z = \texttt{re} + i \times \texttt{im} \in \mathbb{C}$ à l'aide de l'enregistrement `struct Complexe` défini comme suit :

```
struct Complexe{
  double re ;
  double im ;
};

typedef struct Complexe complexe;
```

Nous munissons cet enregistrement de plusieurs opérations primitives :

```
/* SIGNATURES DES OPERATIONS PRIMITIVES */
/* Constructeurs et */
complexe zero () ;
complexe ecrire_re (double a, complexe z) ;
complexe ecrire_im (double b, complexe z) ;
/* Acces */
double lire_re (complexe z) ;
double lire_im (complexe z) ;
```

Ces opérations primitives réalisent les axiomes suivants :

[1] `lire_re(zero()) = 0`

[2] `lire_re(ecrire_re(a,z)) = a`

[3] `lire_re(ecrire_im(b,z)) = lire_re(z)`

[4] `lire_im(zero()) = 0`

[5] `lire_im(ecrire_re(a,z)) = lire_im(z)`

[6] `lire_im(ecrire_im(b,z)) = b`

Exercice 1 — Les opérations primitives
Complétez le code `C` des fonctions primitives dans le fichier `complexe.c` disponible sur Arche.

2 Opérations non primitives

On souhaite maintenant munir cette structure de diverses fonctions *usuelles* sur les nombres complexes :

```
/* SIGNATURES DES OPERATIONS NON PRIMITIVES */
bool egaux (complexe z1, complexe z2) ;
```

```
complexe oppose (complexe z) ;
complexe conjugue (complexe z) ;
double module (complexe z) ;
complexe somme(complexe z1, complexe z2);
complexe inverse (complexe z) ;
complexe produit(complexe z1, complexe z2);
complexe quotient(complexe z1, complexe z2);
```

Indication : Afin d'utiliser la fonction `double sqrt(double x)` de la bibliothèque `math.h` il est nécessaire d'ajouter l'option `-lm` à la fin de la ligne de compilation : `gcc complexe.c -o complexe -lm`.

Exercice 2 — Les opérations non-primitives
Complétez le fichier `complexe.c` avec les codes des fonctions précédentes. Pour chacune des fonctions, vous fournirez également les procédures de test. La fonction `main` du fichier `complexe.c` aura la forme suivante :

```
int main(int argc, char** argv){

  test_egaux();
  test_oppose();
  test_conjugue();
  test_module();
  test_somme();
  test_inverse();
  test_produit();
  test_quotient();

  return EXIT_SUCCESS;
}
```

3 Équation du second degré

Soient a, b et c trois réels. On représente l'équation $az^2+bz+c=0$ à l'aide de l'enregistrement `struct Equation` suivant :

```
struct Equation_2nd_degre{
  double a, b, c ;
  unsigned int nb_solutions;
  complexe solution0, solution1;
};
typedef struct Equation_2nd_degre equation;
```

Exercice 3 — Nouvelle équation
Complétez le fichier `complexe.c` avec le code `C` de la fonction `equation nouvelle_equation(double a, double b, double c)` qui prend en entrée trois réels et renvoie un élément de type `equation` représentant l'équation $az^2+bz+c=0$.

Rappel : Soit $S \subseteq \mathbb{C}$ l'ensemble des solutions de l'équation $az^2+bz+c=0$.

1. Si $a=b=c=0$, alors $S=\mathbb{C}$.
2. Si $a=b=0$ et $c\neq 0$, alors $S=\emptyset$.

3. Si $a = 0$ et $b \neq 0$, alors $S = \left\{-\frac{c}{b}\right\}$.
4. Si $a \neq 0$, on pose $\Delta = b^2 - 4ac$:
   (a) Si $\Delta = 0$, alors $S = \left\{-\frac{b}{2a}\right\}$.
   (b) Si $\Delta > 0$, alors $S = \left\{-\frac{b-\sqrt{\Delta}}{2a}, -\frac{b+\sqrt{\Delta}}{2a}\right\}$.
   (c) Si $\Delta < 0$, alors $S = \left\{-\frac{b-i\sqrt{-\Delta}}{2a}, -\frac{b+i\sqrt{-\Delta}}{2a}\right\}$.

Le réel Δ est appelé le **discriminant** de l'équation $az^2 + bz + c$.

Exercice 4 — Discriminant
Écrire le code `C` de la fonction `double delta(equation eq)` (et celui de ses procédures de test également) qui, étant donnée une équation `eq`, calcule et retourne la valeur de son discriminant.

Exercice 5 — Résolution d'une équation
Complétez le fichier `complexe.c` avec la procédure `void resolution(equation eq)` qui prend en entrée une équation `e` et modifie les champs `nb_solutions`, (et éventuellement) `solution0` et `solution1` de `eq` après les avoir calculé.