TE 6

Structures de contrôle en fortran et C

Références
– Polycopié FORTRAN : chap. **4** ; chap. 2, section 2 ; annexe C
– Polycopié C : chap. 6
– Transparents : cours 3

A Structures de contrôle

A.1 Structures conditionnelles

Exercice 1 : Affectation et test A

Compiler les programmes `affectation-test.c` [1] et `affectation-test.f90` suivants, Dans le cas où un exécutable est produit, tester le programme avec les valeurs $(2, 2)$, $(0, 0)$, $(2, 0)$ puis $(0, 2)$; expliquer. Corriger les programmes pour obtenir le comportement attendu.

affectation-test.c

```
#include <stdio.h>
#include <stdlib.h>
int main(void){
int i, j;
printf("saisir deux entiers\n");
scanf("%d %d", &i, &j);
printf("i=%d  j=%d\n", i, j);
if (i=j) {
  printf("i et j sont egaux : i=%d  j=%d\n", i, j);
}
exit(EXIT_SUCCESS);
}
```

affectation-test.f90

```
program affectation
implicit none

integer :: i, j
write(*,*) "saisir deux entiers"
read(*,*) i, j
write(*,*) "i=", i, " j=", j
if (i=j) then
  write(*,*) "i et j sont egaux, i=", i, " j=", j
end if

end program affectation
```

Exercice 2 : A

Écrire un programme qui saisit deux nombres **entiers** `x` et `y`, les affiche, puis les compare et affiche (selon les cas) un des trois messages suivants : `x` est plus grand que `y`, `x` est plus petit que `y` ou `x` est égal à `y`.

1. À l'aide de `if` sans `else`.
2. À l'aide de structures de type `if... else ...` imbriquées.
3. B Écrire une deuxième version de ce programme avec une seule structure `if ...` comportant un `else if`.
4. B Dans quel langage peut-on aussi utiliser une structure d'énumération de cas (`case`) ?
5. Transformer le programme pour utiliser des réels (`float` ou `real`). Tester alors avec les valeurs `9.8765430` et `9.8765435`

Exercice 3 : Années bissextiles A

On rappelle qu'une année est bissextile :
– si elle est multiple de 4 mais n'est pas multiple de 100, ou
– si elle est multiple de 400.

1. En C, compiler d'abord avec `gcc` seul, puis avec les options de l'alias `gcc-mni-c99` et noter l'intérêt des options.

1. Écrire un programme `bissextile.c` (`bissextile.f90`) qui détermine si une année est bissextile, en utilisant des tests imbriqués. Le programme prendra en charge la saisie de l'année.
2. En faire une version `bissextile2` où la condition est évaluée sous forme de booléen avec des opérateurs logiques.
3. Copier l'un des programmes précédents pour construire un programme `liste_bissextile` affichant la liste des années bissextiles entre les années 1880 et 2020 en utilisant une structure itérative avec compteur.

A.2 Structures itératives

Exercice 4 : Sommes A

1. À l'aide d'une structure itérative avec compteur, écrire un programme qui imprime la suite des entiers de 1 à 10. Le compléter progressivement pour qu'il affiche côte à côte la somme des entiers de 1 à 10, leurs carrés et la somme de ces carrés. Compléter le programme en affichant de plus les sommes calculées grâce aux expressions analytiques suivantes :

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \quad \text{et} \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

   Comparer les résultats obtenus.
2. AB À l'aide d'une structure `while` en C ou `do while` en fortran, écrire un programme qui imprime la suite des entiers positifs, la somme de ces entiers, leurs carrés et la somme de ces carrés, en arrêtant le calcul dès que la somme des carrés dépasse 500.

Exercice 5 : Somme de la série alternée harmonique A

On se propose de calculer la somme de la série suivante :

$$1 - \frac{1}{2} + \frac{1}{3} + \cdots + \frac{(-1)^{n+1}}{n} + \cdots = \ln 2$$

Notations :

$$S_n = \sum_{i=1}^{n} u_i \qquad S = \lim_{n\to\infty} S_n = \sum_{i=1}^{\infty} u_i$$
$$R_n = S - S_{n-1} = \sum_{i=n}^{\infty} u_i \qquad \varepsilon_n = \frac{-R_n}{S} = \frac{S_{n-1} - S}{S}$$

On rappelle que, pour une série alternée, le reste R_n, caractérisant l'erreur de troncature de la série, peut être majoré en valeur absolue par le premier terme omis dans la somme finie : $|R_n| \leqslant |u_n|$.

Écrire et tester à chaque étape un programme `altern.f90` et un programme `altern.c` qui :

1. pour n fixé, calculent et affichent la somme finie S_n ainsi que l'écart relatif ε_n avec la limite théorique ;
2. déterminent le nombre `nmax` de termes suffisant pour assurer une erreur relative de **troncature** majorée par `reltol`, valeur que l'on saisira au clavier dans le domaine $[10^{-4},\ 10^{-2}]$;
3. AB effectuent la sommation dans l'*ordre* qui limite les erreurs d'arrondi ; on testera l'effet du changement de l'ordre de sommation pour des précisions relatives jusqu'à 10^{-7}.

On demande de travailler en simple précision (`REAL` en fortran et en `float` en C).

Exercice 6 : Recherche des nombres premiers AB

1. Écrire un programme qui détermine et affiche si un nombre entier fourni par l'utilisateur est premier ; le tester.
   Puis en faire une deuxième version qui optimise la recherche des diviseurs entiers (quel pas choisir et où s'arrêter ?).
2. Transformer ce programme pour qu'il affiche et compte tous les nombres premiers inférieurs à un entier fourni par l'utilisateur (on pourra comparer la durée d'exécution suivant l'optimisation avec la commande `time` à condition de rediriger les entrées/sorties).

Exercice 7 : Précision des flottants : recherche de ε AB

Les réels représentés en virgule flottante sont codés sur un nombre fixe de bits répartis entre :
– signe,
– mantisse (qui détermine la précision),
– exposant (qui détermine le domaine).
Les réels sont donc représentés approximativement et dans un intervalle fini par un ensemble discret et fini de valeurs. Dans chaque octave $[2^n, 2^{n+1}]$, l'exposant est fixe le nombre de flottants est donné par le nombre de combinaisons possibles entre les bits de la mantisse. Pour des flottants sur 32 bits, 23 bits sont consacrés à la mantisse et les 2^{23} réels de l'octave $[2^n, 2^{n+1}]$ sont en progression arithmétique de raison $2^n\varepsilon$ où $\varepsilon = 2^{-23}$. En particulier, pour $n = 0$, la valeur `1.` est représentée exactement, et son successeur est $1 + \varepsilon$. Mais son prédécesseur, dans l'octave $[1/2, 1]$, est $1 - \varepsilon/2$. L'écart relatif entre deux réels consécutifs est donc compris entre $\varepsilon/2$ et ε. La précision relative de représentation des réels est donc majorée par ε.

epsilon.c

```c
#include <stdio.h>
#include <stdlib.h>
// definition des caracteristiques des flottants
#include <float.h>
int main(void){
  float eps;
  eps = FLT_EPSILON;
  printf("FLT_EPSILON  %g\n", eps);
  exit(EXIT_SUCCESS);
}
```

epsilon.f90

```
program real_eps
implicit none
real :: eps
eps = epsilon(1.)
write(*,*) "epsilon(1.) = ", eps
end program real_eps
```

1. Exécuter les programmes `epsilon.c` (`epsilon.f90`) afin d'afficher la valeur de ε pour le type `float` en C et pour le type `REAL` en fortran respectivement. Que se passe-t-il si on imprime cette valeur en format `%f` en C ou en format `f8.6` en fortran ?
2. Justifier les valeurs affichées sachant que pour les flottants sur 32 bits, 23 bits sont consacrés à la mantisse, 8 à l'exposant et 1 au signe.
3. B On se propose dans la suite de rechercher un encadrement de ε par une méthode de dichotomie. On considère des intervalles $[f_1, f_2]$ de borne inférieure fixe $f_1 = 1.$, et de largeur $\delta = f_2 - f_1$ variable. La borne inférieure, f_1, est représentée exactement mais la borne supérieure, f_2, sera arrondie au flottant le plus proche. On part de $\delta = 1.$ et on divise δ par 2 à chaque étape de l'itération jusqu'à ce que la représentation approximative en flottant de la borne supérieure f_2 soient confondue avec f_1.

   Compléter le code fourni dans `rech-epsilon` pour mettre en œuvre l'itération [2] et afficher la largeur de l'intervalle. Comparer avec ε affiché précédemment. Expliquer à l'aide d'un schéma figurant $f_1 = 1.$ et son successeur, ainsi que f_2 et son approximation en flottant pour les dernières étapes de la dichotomie. On pourra essayer des divisions par des valeurs inférieures à 2 pour affiner l'encadrement de ε ; la recherche sera alors plus longue.

2. En C, on veillera à n'utiliser que des constantes de type `float` pour éviter des opérations en `double`.

rech-epsilon.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <float.h> // necessaire pour FLT_EPSILON
int main(void){
  float eps;
  float f1, f2, delta;
  eps = FLT_EPSILON;
  printf("FLT_EPSILON  %g\n", eps);
  // recherche de epsilon float par dichotomie
  // initialisation
  delta = 1.f;
  f1 = 1.f;
  f2 = f1 + delta;

  // a completer par la boucle de dichotomie

  exit(EXIT_SUCCESS);
}
```

rech-epsilon.f90

```
program real_eps
! recherche de epsilon real
implicit none
real :: eps
real :: f1, f2, delta
eps = epsilon(1.)
write(*,*) "epsilon(1.) = ", eps
! initialisation
delta = 1.
f1 = 1.
f2 = f1 + delta

! a completer par la boucle de dichotomie

end program real_eps
```

B Compléments (facultatifs)

Exercice 8 : Nombre d'or AB

L'objectif est d'écrire un programme calculant le Nombre d'Or. Celui-ci peut être obtenu à partir de la suite de Fibonnacci u_n définie par la récurrence suivante :

$$u_{n+1} = u_n + u_{n-1} \quad \text{avec} \quad u_0 = 1 \quad \text{et} \quad u_1 = 1$$

La suite des rapports $v_n = \left(u_{n+1}/u_n\right)$ converge vers le Nombre d'Or de valeur théorique $\frac{1+\sqrt{5}}{2}$.

1. Quelle structure de contrôle permet de piloter le calcul des termes de la suite jusqu'à ce que le rapport v_n de deux termes consécutifs converge à la précision relative η près ?
2. Écrire le programme de calcul des suites u_n et v_n.
3. Tester le programme avec $\eta = 10^{-4}$ en `real` en fortran et en `float` en C. Combien d'itérations sont nécessaires ? Comparez la valeur trouvée à $\frac{1+\sqrt{5}}{2}$ calculée dans le même type.
4. Le reprendre avec $\eta = 10^{-6}$. Combien d'itérations sont nécessaires ?
5. B Que doit-on modifier pour atteindre la précision de 10^{-10} ? Quel est alors le nombre d'itérations nécessaires ? Reprendre la comparaison théorique avec attention.

Exercice 9 : Somme de la série entière de argsh(x) B

La fonction argument sinus hyperbolique admet le développement en série entière suivant lorsque $|x| \leqslant 1$:

$$\begin{aligned} \operatorname{argsh} x &= x - \frac{1}{2\times 3}\, x^3 + \frac{1\times 3}{2\times 4\times 5}\, x^5 - \frac{1\times 3\times 5}{2\times 4\times 6\times 7}\, x^7 + \cdots = \sum_{n=0}^{\infty} a_n(x) \\ \text{où} \quad a_n(x) &= \frac{(-1)^n}{2n+1}\, \frac{(2n)!}{2^{2n}(n!)^2}\, x^{2n+1} \end{aligned} \tag{6.1}$$

1. Indiquer pourquoi programmer le calcul de cette série en calculant le terme général $a_n(x)$ tel qu'il est écrit ci-dessus n'est ni numériquement sûr, ni numériquement efficace.
2. En calculant $\dfrac{a_{n+1}(x)}{a_n(x)}$, montrer qu'il existe une relation de récurrence liant $a_{n+1}(x)$ à $a_n(x)$.
3. Programmer le calcul de la série en utilisant cette récurrence. Laisser le choix de x à l'utilisateur et calculer par défaut les 100 premiers termes de la série. Comparer à la fin le résultat obtenu avec celui fourni par la fonction `asinh` en C, et en utilisant le fait que $\operatorname{argsh}(x) = \ln\left(x + \sqrt{x^2+1}\right)$ en fortran [3].
4. En fait, il ne sert à rien de rajouter des termes dans la somme dès que le rapport entre le terme à ajouter et la somme partielle devient inférieur en valeur absolue à ϵ, donné en fortran par la fonction intrinsèque `EPSILON(1.)` et en C par la constante `FLT_EPSILON` définie dans le fichier d'entête `<float.h>` Modifier le programme pour arrêter la sommation dès que ce rapport est atteint.

On demande de travailler en simple précision (`REAL` en fortran et en `float` en C).

3. La fonction intrinsèque `asinh`, extension de type `gnu` sous `gfortran` norme 2003, est intégrée à la norme du fortran 2008.