TE 5

Exercices d'introduction en fortran et C

Éléments de correction

Dans votre répertoire `mni`, créer un répertoire `f90+c`, puis un sous-répertoire `te5`. Ne pas oublier de nettoyer votre répertoire de travail à la fin de la session, en effaçant les fichiers binaires objets et exécutables.

Options des compilateurs Afin de profiter pleinement de l'analyse effectuée sur le fichier source par le compilateur, il est fortement conseillé de préciser le standard choisi et de lui demander d'avertir l'utilisateur dès qu'une instruction présente un risque d'erreur. Ce choix se traduira notamment par l'usage des options de compilation suivantes :

en C89 : `gcc -std=c89 -W -Wall`

en C99 : `gcc -std=c99 -W -Wall`

en fortran95 avec g95 : `g95 -std=f95 -Wall`

ou, avec gfortran `gfortran -std=f95 -Wall`

Ces options seront automatiquement appliquées en utilisant les alias fournis par les fichiers de configuration de l'UE :
- `gcc-mni-c89` ou `gcc-mni-c99` pour le C
- `g95-mni` (ou `g2003-mni`) avec `g95` et `gfortran-mni` (ou `gfortran03-mni`) avec `gfortran` pour le fortran 95 (ou le fortran 2003),

A) Notions fondamentales

Exercice 1 : Premier programme (édition, compilation, exécution) A

Cet exercice a pour but de vous présenter, sur un exemple simple, les étapes de base allant de l'écriture d'un programme, à sa compilation, son édition de lien et son exécution. Elles s'appliqueront dans toute la suite des TEs à des programmes de plus en plus compliqués.

1. Édition du fichier source
   (a) Indiquez ce qu'est un code source.
   (b) Avec un éditeur de texte (`vi`, `emacs`, ...), recopiez le code source suivant dans des fichiers nommés `bonjour` et dont l'extension permet de déterminer si le programme est écrit en C ou fortran. On lancera la commande d'édition depuis une console pour préciser dès le départ le nom du fichier : son extension est reconnue par l'éditeur qui active alors la colorisation syntaxique.

bonjour.c

```c
/* Premier programme en C89 */
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Bonjour ! \n");

    exit(0);
}
```

bonjour.f90

```
! Mon premier programme en fortran

program essai

implicit none

write(*,*) "Bonjour !"
stop

end program essai
```

   (c) Comptez le nombre de lignes, de mots, de caractères et d'octets des fichiers de code source.
2. Compilation
   (a) Compilez le code source à l'aide du compilateur `gcc` / `g95` :
   `gcc-mni-c89 bonjour.c`, `g95-mni bonjour.f90` ou `gfortran-mni bonjour.f90`
   (b) Quel est le rôle du compilateur ?
   (c) Quel est le nom du fichier produit par la compilation dans chaque langage ?
   (d) Quelle est la taille en octets de l'exécutable (comparer `g95` et `gfortran`) ? Quels sont ses droits d'accès ? Vérifier que ce fichier binaire contient le texte `Bonjour`.
3. Exécution
   (a) Exécutez le fichier produit (à l'invite du système, entrez : `./nom_de_fichier`). Commentez le résultat obtenu en sortie standard.
   (b) Si vous n'obtenez pas le résultat attendu, que vous recevez des messages d'avertissement ou d'erreur, repartez de l'étape édition, identifiez les erreurs et corrigez-les, puis recompilez le programme. Ne relancez d'exécution que si la compilation réussit (comparez les dates du source et de l'exécutable). Reprenez ce cycle jusqu'à une exécution correcte.
4. Compilation seule, puis édition de liens
   Trouvez l'option vous permettant de conserver le fichier objet produit à la compilation (fichier d'extension `.o`, e.g. `bonjour.o`). Quelle est la différence entre le fichier objet et l'exécutable ? Générez un fichier exécutable à partir du fichier objet. Exécutez-le et vérifiez que le résultat est le même qu'en 3a.
5. Trouvez l'option vous permettant de produire un fichier exécutable de nom `bonjour1` à partir du fichier objet `bonjour.o`. Vérifiez que le résultat de l'exécution est le même qu'en 3a et 4.
6. À partir du fichier source `bonjour.c` ou `bonjour.f90`, produire un exécutable de nom `bonjour2` en une seule ligne de commande. Après exécution, vérifiez que le résultat est le même qu'en 3a, 4 et 5.
7. Si vous voulez donner à quelqu'un un programme que vous avez écrit, quel fichier devez-vous lui fournir sachant que vous ne savez pas sur quelle machine il doit l'utiliser ?
   Connectez-vous sur `sappli1` et essayez de lancer l'exécutable C compilé sur la machine virtuelle. Expliquez en utilisant la commande `file`. Recompilez sur `sappli1` et vérifiez.
   B Essayez d'exécuter sur la machine virtuelle le fichier compilé sur `sappli1`. Expliquez. Reprendre la question avec l'option `--static` des compilateurs.

Solution

1. (a) Code source : série d'instructions (par exemple en C ou fortran) ou de commandes (par exemple sous UNIX) exprimées selon la syntaxe d'un langage pour demander à l'ordinateur d'effectuer certaines tâches.
   (b) Extension ou suffixe des fichiers source : `.c` (langage c) ou `.f90` (fortran 90)
   (c) `wc bonjour.c` ou `wc bonjour.f90`
2. (a)
   (b) Compilateur : programme qui traduit le code source en langage machine.
   (c) Fichier exécutable produit par défaut : `a.out`.
   (d) `ls -l a.out` affiche la taille et les droits d'accès de l'exécutable binaire. Les exécutables `g95` sont notablement plus gros car l'édition de liens est statique. Le droit d'exécution (`x`) est positionné par défaut (sans appel à `chmod`). `strings a.out` permet d'extraire la partie texte du fichier exécutable (on peut rechercher le message via grep : `strings a.out |grep Bonjour`.)
3. (a) Résultat : `Bonjour !`
   (b) Les erreurs à l'exécution sont souvent plus difficiles à analyser que les erreurs à la compilation ; sur des programmes comportant des entrées/sorties, elles peuvent provenir de saisies erronées au clavier.
4. `gcc-mni -c bonjour.c` ⇒ `bonjour.o` ou `g95-mni -c bonjour.f90` ⇒ `bonjour.o`
   Lors de l'utilisation de fonctions externes (de librairie comme `printf()`), le fichier objet généré par la compilation doit être lié au code objet des fonctions externes pour créer l'exécutable final (passage de l'objet à l'exécutable au moyen d'un éditeur de liens).
   `gcc-mni bonjour.o` ⇒ `a.out` ou `g95-mni bonjour.o` ⇒ `a.out`
5. `gcc-mni bonjour.o -o bonjour` ⇒ `bonjour` ou `g95-mni bonjour.o -o bonjour` ⇒ `bonjour`

6. `gcc-mni bonjour.c -o bonjour` ⇒ `bonjour1` ou `g95-mni bonjour.f90 -o bonjour` ⇒ `bonjour2`
7. Le fichier source, qui peut être recompilé sur la machine cible à condition qu'il respecte une norme de langage[1].
   Les exécutables ne sont pas portables, en particulier des machines virtuelles vers le serveur `sappli`, notamment à cause de l'édition de liens dynamique : il faudrait que les bibliothèques dynamiques en particulier la `GLIBC` soient compatibles.
   Par exemple, si on compile un source avec `gcc` sur la machine virtuelle mandriva 2009 et que l'on essaye de l'exécuter sur `sappli`, on obtient l'erreur
   `a.out: /lib/tls/libc.so.6: version 'GLIBC_2.4' not found (required by a.out)`
   La commande `file` indique la version de la `glibc` que requiert l'exécutable. L'opération réussit en sens inverse car `sappli` utilise une version plus ancienne que mandriva 2009 (compatibilité ascendante).
   En revanche, si on effectue une édition de liens avec des bibliothèques statiques, par exemple avec `gcc --static`, l'exécutable compilé sous mandriva peut fonctionner sous `sappli1`.

Exercice 2 : Découpage des lignes affichées en sortie A

Sans compiler les programmes `lignes.c` et `lignes.f90` suivants, prédire combien de lignes ils doivent afficher, ainsi que le contenu de ces lignes.

lignes.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void){
  printf("Combien de lignes\n\n sont affichees"
         " par ce programme et ");
  printf("\nquel est le contenu des ces lignes ?\n");
  printf("Observer ... la fin !");
  exit(EXIT_SUCCESS);
}
```

lignes.f90

```
program lignes

  write(*,*) "Combien de lignes"
  write(*,*)
  write(*,*) " sont affichees&
         & par ce programme et "
  write(*,*) "quel est le contenu des ces lignes ?"
  write(*,'(a)',advance="no" ) "Observer ... "
  write(*,'(a)',advance="no" ) "la fin !"
end program lignes
```

Compiler et exécuter pour vérifier. Modifier le programme pour que l'invite unix apparaisse en début de ligne.

Solution

En C, une instruction se termine par un point virgule et non en fin de ligne. On ne peut pas écrire une chaîne sur plusieurs lignes dans le fichier source sans refermer les «"». En revanche, deux chaînes de caractères entre «"», séparées par un ou plusieurs blancs (espace, tabulation, retour ligne) sont concaténées par le compilateur. L'instruction `printf` ne fait pas passer à la ligne suivante ; il faut pour cela un `\n` dans la chaîne de caractères à afficher.

En fortran, la fin de ligne signifie la fin d'une instruction, sauf si elle se termine par une esperluette (`&`). Il est possible de couper une ligne à l'intérieur d'une chaîne : alors les blancs initiaux ne sont pas pris en compte dans la suite de la chaîne qui commence après la première esperluette. Chaque instruction `write` passe à la ligne suivante.

Observer en particulier la position de l'invite unix en fin d'affichage.

affichage en C

```
Combien de lignes

 sont affichees par ce programme et
quel est le contenu des ces lignes ?
Observer ... la fin !invite unix
```

affichage en fortran

```
Combien de lignes

 sont affichees par ce programme et
quel est le contenu des ces lignes ?
Observer ... la fin !invite unix
```

Rappel des syntaxes élémentaires d'entrées-sorties standard

En fortran, en format libre :

```
write(*,*) "entrer un entier (i) et un réel (r)"
read(*,*) i, r
write(*,*) "valeurs lues i = ", i, "  r = ", r
```

Chaque ordre `read` ou `write` provoque un changement de ligne (sauf avec `advance=no`).

[1] Donner l'exécutable suppose que le destinataire ait une très grande confiance en l'auteur du programme, ce qui en retour, permet de supposer que l'auteur n'a aucune raison de ne pas donner le fichier source.

En C :

```
printf("entrer un entier (i) et un float (r)\n") ;
scanf("%d %g", &i, &r) ;
printf("valeurs lues i = %d , r = %g \n", i, r) ;
```

Ne pas oublier le `&` en entrée. Insérer `\n` dans le format d'affichage pour passer à la ligne.

Exercice 3 : Saisie, calcul, affichage A

1. Saisissez avec un éditeur de texte et sauvegardez dans un fichier de suffixe approprié le code source suivant :

```c
// Titre de mon programme en C99

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int x, y, z, t, u;
    double a, b;
    /* Mon premier commentaire  */
    printf("Entrez un nombre entier \n");
    scanf("%d", &x);
    /* Mon
       deuxieme
       commentaire */
    printf("Entrez un deuxieme nombre entier \n");
    scanf("%d", &y);

    z = x * y;
    t = x / y  u = x % y; // 3ieme commentaire
    printf("%d * %d = %d  t = %d u = %d\n",
            x, y, z, t, u);
    // decrire la suite
    printf("Saisir deux doubles \n");
    scanf("%lg %lg", &a, &b);
    c = a / b;
    printf("a=%g   b=%g   c=%g\n", a, b, c);
    printf("fin\n");

    exit(0);
}
```

```f90
! Titre de mon programme en fortran

program calcul

implicit none

integer :: x, y, z, t,  u
real :: a, b

! Mon premier commentaire
  write(*, *) "Entrez un nombre entier:"
  read(*, *) x

! Mon deuxieme
   !  commentaire
  write(*, *) "Entrez un deuxieme entier"
  read(*, *) y

  z = x * y ! 3ieme commentaire
  t = x / y   u = mod(x,y)
  write(*, *) x, " * ", y, " = ", z, &
    " t = ", t , " u = ", u
  ! decrire la suite
  write(*,*) "saisir deux reels "
  read (*,*) a, b
  c = a / b
  write(*,*) "a=", a, "    b=", b, "   c=", c
  write(*,*) "fin"

end program calcul
```

   Compiler ce code source en examinant attentivement les messages du compilateur. Corriger pas à pas les erreurs dans l'ordre des lignes du code[2]. Que fait le programme ?

2. – Lister les variables et leur type.
   – Modifier les commentaires de manière à décrire les actions du programme.
   – Repérer les opérateurs utilisés.
   – B Repérer deux opérateurs présents dans le programme en C et pas dans celui en fortran ; citer un opérateur qui existe en fortran mais pas en C.
   – **(C)** `#include <stdio.h>`
   Quel est le rôle de `#include` ? Qu'est ce que le fichier : `stdio.h` ? Pourquoi ce fichier permet le bon fonctionnement du programme ?
   – B **(f90)** Quel est le rôle de la déclaration `implicit none` ? Où peut-on la placer ?

Correction

1. C : Erreur sur la ligne de division et de modulo des variables `t = x / y  u = x % y;` (il manque le `;` pour terminer la première instruction).
   Fortran : erreur à la ligne de division et de modulo des variables `t= x/y  u = mod(x,y)` il manque un `;` pour séparer plusieurs instructions sur une même ligne.
   Autre erreur dans les deux langages : `c` n'est pas déclarée
   Le programme lit deux entiers, calcule leur produit, effectue leur division entière et calcule le reste ; il affiche toutes ces valeurs. Il lit ensuite deux réels `a` et `b` et place dans `c` les résultats de la division de `a` par `b` et affiche `a`, `b` et `c`.

[2] C'est le débogage du programme – vient de `bug` = anomalie en anglais, terme francisé en bogue.

2. **C :** `main(){ }` fonction principale de type entier. `printf()` et `scanf()` fonctions de librairie.
   fortran : appel de `write` et `read`
   - 5 entiers et deux nombres en virgule flottante dans le programme principal **C :** 5 `int` et 2 `double` déclarés dans `main`.
     Fortran : 5 `integer` et 2 `real` déclarés.
     Tous les calculs sauf celui de `a/b` sont ici effectués en entier, en particulier la division `x/y`.
   - en C : délimiteurs de commentaires (multilignes éventuellement) : `/* ... */`
     introducteur en C99 : `//`
     Fortran : introducteur de commentaires `!` (hors chaîne de caractères)
   - Les opérateurs communs aux deux langages : `=` affectation ; `+` addition ; `-` soustraction ; `*` multiplication ; `/` division ;
     En C, mais pas en fortran : `%` opérateur binaire reste de la division entière
     `*` opérateur unaire d'indirection ou de déréférencement.
     `&` opérateur unaire d'adresse ou de référencement.
     En fortran `**` (élévation à la puissance), mais pas en C (utiliser `pow` + `tgmath.h` + `-lm`)
   - **C :** les instructions se terminent par `;`
     les blocs sont délimités par des `{ }`.
     Fortran : la fin de ligne marque la fin d'instruction sauf si elle se termine par `&`, ce qui indique que l'instruction continue sur la ligne suivante.
   - **C :** `#include` demande au préprocesseur d'insérer le contenu d'un fichier avant la compilation. `<stdio.h>` est l'exemple d'un tel fichier (`.h` pour header file). Sans `<stdio.h>`, le prototype des fonctions de librairie comme `printf()` n'est pas connu : le compilateur suppose alors un typage implicite pour leurs paramètres.
   - **fortran :** l'instruction `implicit none` interdit l'emploi de variables non déclarées ; ce restriction est très fortement conseillée.

Exercice 4 : Échange A

Sans connaître les valeurs de `a` et `b`, compléter le code des programmes ci-après de façon à placer la valeur de la variable `a` dans `b` et réciproquement.

echange.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
 double a, b;
 a = 5.2;
 b = 9.3;
 printf("La variable 'a' contient la valeur %g\n", a);
 printf("La variable 'b' contient la valeur %g\n", b);
 // a completer
 printf("La variable 'a' contient la valeur %g\n", a);
 printf("La variable 'b' contient la valeur %g\n", b);
 exit(0);
}
```

echange.f90

```
program echange
implicit none
real :: a, b

a = 5.2
b = 9.3
write(*,*) "La variable 'a' contient la valeur ", a
write(*,*) "La variable 'b' contient la valeur ", b
! a completer
write(*,*) "La variable 'a' contient la valeur ", a
write(*,*) "La variable 'b' contient la valeur ", b

end program echange
```

Copier les fichiers source associés. Compléter le code et tester.

Solution

Il faut définir une variable intermédiaire du même type que `a` et `b`

```c
double c;
c = b;
b = a;
a = c;
```

```
REAL :: c
c=b
b=a
a=c
```

B) Entrées sorties standard élémentaires

Exercice 5 : Format de sortie `f` en virgule fixe A

Compiler les programmes `format_f.c` et `format_f.f90`. Exécuter ces programmes et expliquer l'affichage. Modifier le programme pour obtenir un affichage plus robuste.

format_f.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    float r1, r2;
    r1 = 1.234e-5f;
    r2 = r1 / 100.f;
    printf("r1 vaut %f et r2 vaut %f\n", r1, r2);
    exit (EXIT_SUCCESS);
}
```

format_f.f90

```
program format_f
implicit none

    real :: r1, r2
    r1 = 1.234e-5
    r2 = r1 / 100.
    write(*, "(a, f12.6, a, f12.6)") &
         "r1 vaut ", r1, " et r2 vaut ", r2
end program format_f
```

Solution

Le format `f` soit virgule fixe peut faire apparaître un nombre trop faible comme nul : c'est le cas de `r2` ici. On préférera le format `g` (général) qui s'adapte à la valeur : il cherche à afficher en virgule fixe sauf s'il y a soit dépassement du nombre total de chiffres (valeur trop grande) soit affichage de zéro à cause d'un nombre insuffisant de chiffres après la virgule. Dans ces cas il bascule en virgule flottante (format `e`).

Exercice 6 : Formats de sortie erronés A

Compiler les programmes `erreur_format.c` (avec `gcc` sans option ou avec l'alias qui implique `-Wformat`) et `erreur_format.f90`. Prendre connaissance des messages et s'assurer que l'on a bien créé un exécutable (comment faire ?). Lancer l'exécution et expliquer. Modifier le programme pour obtenir un affichage correct.

erreur_format.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    float r;
    int i;
    r = 5.25f;
    i = 12;
    printf("r vaut %e et  i vaut %e\n", r, i);
    printf("r vaut %d et  i vaut %d\n", r, i);
    exit (EXIT_SUCCESS);
}
```

erreur_format.f90

```
program err_format
  implicit none
  real :: r
  integer :: i

  r = 5.25
  i = 12
  write(*, "(a, i12, a, i12)") &
       "r vaut ", r, " et  i vaut ", i
  write(*, "(a, e12.3, a, e12.3)") &
       "r vaut ", r, " et  i vaut ", i
end program err_format
```

Solution

L'affichage de l'entier `i` est demandé dans un format de type `e` réservé aux flottants. Il faudra passer en format `%d` en C et `i` en fortran (par exemple `i0` en fortran 2003). De même, l'affichage de `r` en format `d` en C ou `i` en fortran est réservé aux entiers.

Avec `gcc` sans options sévères, la compilation réussit et l'exécution ne provoque pas d'erreur, mais l'affichage se fait avec une conversion inadaptée dans les deux `printf` : double pour un entier, ou entier pour un float converti en double. On remarque que l'affichage de `i` en flottant n'est pas un flottant valide sur 32 bits (première ligne). Sur la deuxième ligne, l'affichage en entier pour `r` donne 0 et vient corrompre de plus l'affichage de `i` en entier !

```
r vaut 5.250000e+00 et  i vaut -2.273810e-39
r vaut 0 et  i vaut 1075118080
```

En revanche, l'option `-Wformat` (impliquée par `-Wall`) suffit à détecter le problème dès la compilation.

```
gcc -Wformat erreur_format.c
erreur_format.c: In function 'main':
erreur_format.c:9: warning: format '%e' expects
type 'double', but argument 3 has type 'int'
erreur_format.c:10: attention : format '%d' expects
type 'int', but argument 2 has type 'double'
```

N.-B. : les arguments de `printf`, de type `float`, sont passés par copies et sont convertis en type `double` lors de l'appel, ce qui explique que les `double` et non les `float` soient invoqués dans les messages. Il n'en serait pas de même avec `scanf` à qui on passe des pointeurs et qui dispose de deux formats distincts pour les types `float` et `double`.

Avec `g95` ou `gfortran`, pas d'option `-Wformat`, la compilation réussit mais une erreur se produit à l'exécution (le message pointe vers le format incriminé).

```
At line 7 of file erreur_format.f90 (Unit 6)
Fortran runtime error: Expected REAL for item 4
in formatted transfer, got INTEGER (a, e12.3, a, e12.3)
                                                ^
```

C) Types et conversions

Exercice 7 : Conversions A

Sans éditer et avant de compiler les programmes suivants, indiquer tout d'abord les affichages que doivent produire leur exécution.

real_int_conv.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    float q;
    float a;
    int g;

    q = 5.25;
    g = 5;

    a = q / g;
    printf("La valeur de a est %g\n", a);
    a = a + g;
    printf("La valeur de a est %g\n", a);
    g = a - q;
    a = g + q;
    printf("La valeur de a est %g\n", a);
    exit(0);
}
```

real_int_conv.f90

```
program real_int_conv
implicit none

real :: q, a
integer :: i

q = 5.25
i = 5

a = q / i
write(*,*) "valeur de a = ", a

a = a + i
write(*,*) "valeur de a = ", a

i = a - q
a = i + q
write(*,*) "valeur de a = ", a

end program real_int_conv
```

Copier les fichiers source associés. Compiler ce code source avec `gcc` (`g95` ou `gfortran`) sans option.

Exécuter ce code et comparer les affichages avec vos prévisions. Afin de rechercher les points délicats sur le code fortran, recompiler avec les options conseillées des alias. Repérer les avertissements et expliquer.

Solution

Les valeurs affichées pour `a` sont successivement 1.05, 6.05 et 5.25

L'instruction `g = a-q` en C ou `i = a-q` en fortran provoque une conversion implicite en entier par troncature. Le compilateur fortran signale cette perte de précision : on peut faire disparaître l'avertissement en rendant explicite la conversion.

Exercice 8 : Divisions AB

Compiler les programmes `div_int.c` et `div_int.f90` avec respectivement `gcc` et `g95` sans option. Les exécuter et expliquer les résultats affichés.

Recompiler avec les options des alias. Modifier le programme pour que cette compilation exigeante se fasse sans avertissement.

div_int.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i, j, k;
    float a, b, c, d;
    i = 15;
    j = 4;
    a = i /j ;
    b = (float ) (i /j) ;
    printf("a = %g   b = %g \n", a, b);
    c = (float) i / (float) j ;
    d = (float) i / j ;
    printf("c = %g   d = %g \n", c, d);
    k = d;
    printf("k = %d\n", k);
    exit(0);
}
```

div_int.f90

```
program divisions

implicit none

integer :: i, j, k
real :: a, b, c, d

  i = 15
  j = 4
  a = i / j
  b = real(i / j)
  write(*,*) "a=", a, "  b=", b
  c = real(i) / real(j)
  d = i / real(j)
  write(*,*) "c=", c, "  d=", d
  k = d
  write(*,*) "k=", k

end program divisions
```

Solution

Si les deux opérandes de la divison sont entiers, la division est une division entière qui donne 3, même si on stocke le résultat dans une variable de type réel. Ainsi `a = b = 3`, mais dès qu'un des opérateurs est réel, la division se fait en type réel, quitte à convertir un des opérandes. Ainsi `c = d = 3.75` ; mais par conversion en entier `k=3`

Exercice 9 : Domaine des entiers AB

Copier et compiler les programmes ci-après. Expliquer le résultat des additions. Retrouver par le calcul les valeurs maximales des entiers par défaut (stockés sur 32 bits). Vérifier les calculs à l'aide de `kcalc` qui permet d'effectuer les conversions entre décimal et binaire.

int_range.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int main(void){
int i, j, k1, k2, k3;
int m1;
i =  1073741824 ;   //  2**30
j = i - 1;
k1 = i + j;
k2 = k1 + 1;
k3 = k1 + 2;
printf(" i= %d, i-1 = %d\n", i, j);
printf("k1= %d k1+1= %d  k1+2= %d\n" , k1,k2, k3);
m1 = INT_MAX;
printf("plus grand entier via INT_MAX %d\n", m1);
exit(EXIT_SUCCESS);
}
```

int_range.f90

```
program int_range
implicit none
integer :: i, j, k1, k2, k3
integer :: m1
i = 2**30
j = i - 1
k1 = i + j
k2 = k1 + 1
k3 = k1 + 2
write(*, *) "i=", i, " i-1=", j
write(*, *) "k1=", k1, " k1+1=", k2, " k1+2=" , k3
! utilisation de fonctions d'interrogation
m1 = huge(i)
write(*,*) "plus grand entier via huge", m1
end program int_range
```

Solution

```
i= 1073741824, i-1 = 1073741823
k1= 2147483647 k1+1= -2147483648  k1+2= -2147483647
```

Le plus grand entier signé sur 32 bits est $2^{31}-1 = 2^{30}+2^{30}-1$, soit 2147483647 qui est le résultat obtenu pour `k1`. Si on ajoute 1, le bit de plus fort poids passe à 1 et on obtient des entiers négatifs par dépassement de capacité.

D) Domaine et précision des flottants

Exercice 10 : Conversion et précision A

Compiler les programmes `precis_affect.c` et `precis_affect.f90` avec respectivement `gcc` et `gfortran` sans option. Les exécuter et expliquer les résultats affichés.
Recompiler avec les options des alias. Modifier le programme pour que cette compilation exigeante se fasse sans avertissement.

precis_affect.c

```c
#include <stdio.h>
#include <stdlib.h>
int main(void){
float a;
int i, j;

i = 1000000020;
printf("i = %d\n", i);
a = i;
j = a;
printf("j = %d\n", j);
exit(EXIT_SUCCESS);

}
```

precis_affect.f90

```
program int_real_conv
implicit none
real ::  a
integer :: i, j

i = 1000000020
write(*,*) "i = ", i
a = i
j = a
write(*,*) "j = ", j

end program int_real_conv
```

Solution

La précision des flottants sur 32 bits est de l'ordre de 2×10^{-7}, donc on ne fait pas différence entre 100000000 et 100000020. Pour éviter les avertissements, il faut rendre explicites les conversions.

precis_affect.c

```c
#include <stdio.h>
#include <stdlib.h>
int main(void){
float a;
int i, j;

i = 1000000020;
printf("i = %d\n", i);
a = (float) i;
j = (int) a;
printf("j = %d\n", j);
exit(EXIT_SUCCESS);

}
```

precis_affect.c

```
program int_real_conv
implicit none
real ::  a
integer :: i, j

i = 1000000020
write(*,*) "i = ", i
a = real(i)
j = int(a)
write(*,*) "j = ", j

end program int_real_conv
```

Exercice 11 : Constante de Stefan B

Préliminaire

Compiler et exécuter les programmes `domaine.c` et `domaine.f90`, qui affichent les limites du domaine des valeurs positives représentables en flottant sur 32 bits.

float_32_range.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
/* domaine des float sur 32 bits */
int main(void){
    printf(" valeur positive max %g\n", FLT_MAX);
    printf(" valeur positive min %g\n", FLT_MIN);
    exit(EXIT_SUCCESS);
}
```

real_32_range.f90

```
program real_32_range
! domaine des reels sur 32 bits

 implicit none
 write(*,*) " valeur positive max ", HUGE(1.)
 write(*,*) " valeur positive min ", TINY(1.)

end program real_32_range
```

Solution

```
valeur positive max  3.4028235E+38
valeur positive min  1.1754944E-38
```

```
valeur positive max 3.40282e+38
valeur positive min 1.17549e-38
```

On souhaite calculer la constante de Stefan σ qui intervient dans l'expression de l'émittance totale du corps noir :

$$M_B = \pi B = \sigma T^4 \quad \text{où} \quad \sigma = \frac{2\pi^5}{15}\frac{k^4}{c^2 h^3}$$

h	constante de Planck	$h = 6,626 \times 10^{-34}$ J s
k	constante de Boltzmann	$k = 1,38 \times 10^{-23}$ J K^{-1}
c	vitesse de la lumière	$c = 3,00 \times 10^{8}$ m s^{-1}

On doit obtenir $\sigma \approx 5,67 \times 10^{-8}$ W m^{-2} K^{-4}.

Écrire un programme `stefan.c` (respectivement `stefan.f90`) qui évalue naïvement la constante de Stefan en flottants sur 32 bits. On prendra pour simplifer $\pi = 3.141593$. En C, on développera les puissances pour éviter de faire appel à la fonction `pow`. Expliquer pourquoi on n'obtient pas la valeur attendue.

Reformuler l'expression de σ en cherchant à limiter le domaine des valeurs intermédiaires à calculer. Exploiter la nouvelle formulation.

Reprendre la première partie de l'exercice en travaillant avec des flottants sur 64 bits (`double` en C et `double precision` en fortran). Expliquer le comportement en affichant les limites du domaine des sous-types flottants sur 64 bits.

Solution

Le premier calcul, naïf, provoque des dépassements de domaine (floating underflow) dans le calcul de k^4 et de h^3 en flottants sur 32 bits. Le résultat est une valeur indéterminée $(0/0)$ signalée par un `NaN` (Not a Number). La factorisation de $(k/h)^3$ utilisée dans la deuxième formulation évite cette difficulté sans qu'il soit nécessaire de faire appel à un type au domaine plus étendu sur 64 bits par exemple.

Remarque sur l'option `-ffloat-store`

Malgré la déclaration des variables sur 32 bits, les calculs intermédiaires sont, par défaut, effectués dans l'unité de calcul en virgule flottante (Floating Processor Unit) en utilisant des registres sur 80 bits, qui correspondent à la double précision étendue de la norme IEEE 754, dont le domaine est plus grand que $[10^{-308}, 10^{+308}]$. Ainsi, l'instruction `sigma = 2*(pi**5/15.) *(k**4)/ (c**2 *h**3)` ne provoquera pas de dépassement de capacité. Mais il suffira par exemple de la formuler différemment par exemple en stockant k^4 ou h^3 dans une variable pour provoquer le dépassement. Une façon de mettre en évidence le risque de dépassement est d'utiliser l'option de compilation `-ffloat-store` qui interdit le stockage des variables en virgule flottante dans des registres à précision (et domaine) étendu. Cette option dégrade donc les performances du processeur et ne devrait être utilisée que dans des cas bien précis où domaine et précision doivent rester conformes au type des variables.

stefan.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <tgmath.h>

// compiler avec -ffloat-store pour éviter
// les registres à précision et domaine étendus

int main(void) {
float c, h, k, pi , sigma;
pi = 3.1415927f;
c = 3.e8f;
h = 6.62e-34f;
k = 1.38e-23f;

// formulation naive => underflow en 32 bits
// sur k**4 et h**3
// sigma  NotANumber
printf("formulation naive avec -ffloat-store");
printf(" => underflow => sigma=NotANumber\n");
sigma = 2.f*(pi*pi*pi*pi*pi/15.f) *
         (k*k*k*k)/(c*c *h*h*h);
printf("c=%g h=%g k=%g sigma=%g\n", c, h, k, sigma);

// formulation robuste => pas d'underflow
// réduction de dynamqiue aevc le rapport k/h
printf("formulation robuste => pas d'underflow \n");
sigma = 2.f*(pi*pi*pi*pi*pi/15.f) *
        (k/h)*(k/h)*(k/h) *k / (c*c);
printf("c=%g h=%g k=%g sigma=%g\n", c, h, k, sigma);
exit(EXIT_SUCCESS);
}
```

stefan.f90

```
program stefan
implicit none
real :: c, h, k, pi , sigma ! réels sur 32 bits

! compiler avec -ffloat-store pour éviter
! les registres à précision et domaine étendus

! passage en 64 bits en déclarant double precision ou
! par option de compilation sous g95 avec -r8
! ...             sous gfortran avec -fdefault-real-8

! calul de sigma = 2*(pi**5 /15)* k**4 /(c**2 * h**3)

pi = 3.1415927
c = 3.e8
h = 6.62e-34
k = 1.38e-23
! formulation incorrecte
! => underflow sur k**4 et h**3 en 32 bits
! => sigma = NotANumber
write(*, *) "formulation naïve avec -ffloat-store"
write(*, *) " => underflow en 32 bits"
sigma = 2*(pi**5/15.) *(k**4)/ (c**2 *h**3)
write(*, *) "c=", c, "h=", h, "k=", k, "sigma=", sigma

! formulation correcte y compris en 32 bits
write(*, *) "formulation robuste => pas d'underflow"
! réduction de dynamique avec le rapport k/h
sigma = 2*(pi**5/15.) *(k/h)**3 *k / c**2
write(*, *) "c=", c, "h=", h, "k=", k, "sigma=", sigma
end program stefan
```

double_64_range.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
/* domaine des double sur 64 bits */
int main(void){
    printf(" valeur positive max %g\n", DBL_MAX);
    printf(" valeur positive min %g\n", DBL_MIN);
    exit(EXIT_SUCCESS);
}
```

double_64_range.f90

```
program double_64_range
! domaine des double precision sur 64 bits

 implicit none
 write(*,*) " valeur positive max ", HUGE(1.d0)
 write(*,*) " valeur positive min ", TINY(1.d0)

end program double_64_range
```

```
 valeur positive max 1.79769e+308
 valeur positive min 2.22507e-308
```

```
  valeur positive max  1.7976931348623157E+308
  valeur positive min  2.2250738585072014E-308
```