C, C++, Fortran: Basics

Bruno Abreu Calfa

Last Update: September 27, 2011

Table of Contents

Outline

Contents

1 Introduction and Requirements

Compiled Languages

- C, C++, and Fortran are **compiled** languages
- What you will need:
  - Compiler
    - C: GNU C Compiler (`gcc`), Intel C Compiler (`icc`)...
    - C++: GNU C++ Compiler (`g++`), Intel C++ Compiler (`icpc`)...
    - Fortran: GNU Fortran Compiler (`gfortran`), Intel Fortran Compiler (`ifort`)...
- What you may use:
  - Integrated Development Environments (IDEs)
    - All Platforms: NetBeans, Eclipse...
    - Mac OS: XCode
    - Windows: Visual Studio

Languages Standards

- C, C++, and Fortran have been revised and enhanced throughout the years
- It is *highly recommended* to adhere to ANSI/ISO standards
- The standards adopted in this presentation are:
  - C: ISO C99
  - C++: ISO C++98

  - Fortran: ISO Fortran 90/95 (some Fortran 2003)

- Fortran 2003 received several enhancements including:
  - Better interoperability with C
  - Object-Oriented Programming
  - Procedure pointers

Possible References

- Google
- Books
  - C:
    * Kernighan, B. W., Ritchie, D. M. (1988) The C Programming Language. 2nd Edition. Prentice Hall.
  - C++:
    * Stroustrup, B. (1997) The C++ Programming Language. 3rd Edition. Addison-Wesley Professional.
    * Yang, D. (2000) C++ and Object-Oriented Numeric Computing for Scientists and Engineers. Springer.
  - Fortran:
    * Chapman, S. J. (2003) Fortran 90/95 for Scientists and Engineers. 2nd Edition. McGraw-Hill.
    * Metcalf, M., Reid, J., Cohen, M. (2004) Fortran 95/2003 Explained. Oxford University Press.

2 Basic Programming Elements

Basic Differences

- C and C++ are *case sensitive*, while Fortran is *case insensitive*
- Primitive Data Types

C	C++	Fortran	Meaning
`bool`	`bool`	`LOGICAL`	Logical values (TRUE or FALSE)
`char`	`char`	`CHARACTER`	Strings of alphanumeric characters
`int`	`int`	`INTEGER`†	Integer values
`float`	`float`	`REAL`†	Real values (single precision)
`double`	`double`	`REAL`†	Real values (double precision)

- † Selecting the precision in Fortran involves more details (compiler and processor dependent)
- Recommended: use the `KIND` parameters in the `ISO_FORTRAN_ENV` module, *e.g.* `REAL(KIND=REAL64)` means 64-bit real value
- C and C++ programs require a "main" function in either of the forms:

```
int main();
int main(void);
int main(int argc, char* argv[]);
```

'Hello, world!' Program

- C:

```
#include <stdio.h>
int main() {
    printf("Hello, world!\n");
    return 0;
}
```

- C++:

```cpp
#include <iostream>
int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

- Fortran:

```fortran
PROGRAM main
    WRITE(*,*) 'Hello, world!'
END PROGRAM main
```

Vectors

- C and C++:

```cpp
int a[5] = {1, 2, 3, 4, 5};
float b[] = {3.14, 2.72};
a[0] // First element of vector 'a' = 1
b[1] // Second (and last) element of vector 'b' = 2.72
```

- Fortran:

```fortran
INTEGER, DIMENSION(5) :: a = (/ 1, 2, 3, 4, 5 /)
REAL(KIND=REAL32), DIMENSION(2) :: b = (/ 3.14, 2.72 /)
a(1) ! First element of vector 'a' = 1
b(2) ! Second (and last) element of vector 'b' = 2.72
a(1:5:2) ! Array subset: first:last:step = (/ 1, 3, 5 /)
```

- Arrays in C and C++ are *zero-based*, while in Fortran they are *one-based* (default, but can be changed)

Matrices

- C and C++:

```cpp
double c[2][3] = {{1.0, -2.3, 3.1},{0.2, 7.5, 4.8}};
c[0][1] // Element in first row and second column = -2.3
```

- Fortran:

```fortran
REAL(KIND=REAL64), DIMENSION(2,3) :: c = RESHAPE((/1.0, 0.2, -2.3, 7.5, 3.1,
    4.8/), (/2, 3/))
c(1,2) ! Element in first row and second column = -2.3
```

- Arrays in C and C++ are stored in *row-major order*, while in Fortran they are stored in *column-major order*

Strings

- C and C++:

```cpp
char d[6] = "Hello";
char e[] = "Programming is awesome! :-P";
d[0] // First element of string 'd' = 'H'
e[10] // Eleventh element of string 'e' = 'g'
```

- Fortran:

```fortran
CHARACTER(len=5) :: d
CHARACTER(len=27) :: e
d = 'Hello'
e = 'Programming is awesome! :-P'
d(1) ! First element of string 'd' = 'H'
e(11) ! Eleventh element of string 'e' = 'g'
```

- All languages have functions and operators for strings manipulation (length, concatenation, comparison...)

Dynamic Memory Allocation

- Preallocate arrays at runtime (useful if dimension is unknown a priori)
- C:

```c
#include <stdlib.h>
...
int m = 1000;
double* f = (double*) malloc(m*sizeof(double)); // Allocate array with 1000 elements
free(f); // Deallocate memory
```

- C++:

```cpp
int m = 1000;
double* f = new double[m]; // Allocate array with 1000 elements
delete[] f; // Deallocate memory
```

- Fortran:

```fortran
INTEGER :: m = 1000
REAL(KIND=REAL64), ALLOCATABLE, DIMENSION(:) :: f
ALLOCATE(f(m)) ! Allocate array with 1000 elements
DEALLOCATE(f) ! Deallocate memory
```

Arithmetic, Relational, and Logical Operators

- Arithmetic Operators:

C and C++	Fortran	Meaning
+	+	Addition
-	-	Subtraction
*	*	Multiplication
/	/	Division
N/A	**	Exponentiation

- Relational Operators:

C and C++	Fortran	Meaning
>, <	>, <	greater than, smaller than
>=, <=	>=, <=	greater or equal than, smaller or equal than
==, !=	==, /=	equal to, not equal to

- Logical Operators:

C and C++	Fortran	Meaning
&&	.AND.	Logical AND
||	.OR.	Logical OR
!	.NOT.	Logical NOT

`if-elseif-else` Statements: Flow Control

- C and C++:

```cpp
if (logical_expr_1) {
    statements_1
}
else if (logical_expr_2) {
    statements_2
}
else {
    statements_3
}
```

- Fortran:

```fortran
IF (logical_expr_1) THEN
    statements_1
ELSE IF (logical_expr_2) THEN
```

```
    statements_2
ELSE
    statements_3
END IF
```

`for`/`DO` Loop Statements

- C and C++:

```
for (index = istart; condition; index update) {
    statements
}
```

- Fortran:

```
DO index = istart, iend, istep
    statements
END DO
```

The `istep` is optional (default: 1)

I/O Concepts

- Two main input/output devices (streams)
  - Standard (console, terminal)
  - File in hard disk
- Main I/O functions
  - C: (`<stdio.h>`)

Function	Meaning
`scanf`	Reads (formatted) data from standard input
`printf`	Writes (formatted) data to standard output
`fscanf`	Reads (formatted) data from file
`fprintf`	Writes (formatted) data to file

  - C++:
    - Higher-level objects and operators
    - For standard I/O, include `<iostream>` and use `namespace std`
    - For files, include `<ifstream>` and use `namespace std`
    - Examples: `cin` (standard input), `cout` (standard output), `endl` (newline), « ('put to' operator), » ('get from' operator)
    - Formatting can be achieved through 'set' methods (width, precision, number format) or inline with `<iomanip>` functions
    - Example: print `x = 1.23` to standard output
      `cout « fixed « setprecision(2) « 1.2345 « endl;`
  - Fortran:
    - Two basic functions: `READ` and `WRITE`
    - List-directed input: `READ(*,*) input_list`
    - List-directed output: `WRITE(*,*) output_list`
    - Formatting can be achieved by specifying the second argument for `READ` and `WRITE` or through labeled `FORMAT` statements
    - Example: print `X = 1.23` to standard output
      `WRITE(*,100) 1.2345`
      `100 FORMAT('X = ',F5.2)`

Procedures: Functions and Subroutines

- C and C++:

```
out_type fcn_name(in_type_1 arg_1, ...) {
    statements
    [return var;]
}
```

where `out_type` can be `void` (no return value)

- Example:

```
double square(double a) {
    return a*a;
}
```

- Fortran:
  - Subroutines: receive and return values through an argument list

```
SUBROUTINE subroutine_name(argument_list)
    statements
    [RETURN]
END SUBROUTINE [subroutine_name]
```

Use the reserved word `CALL` to "call" the subroutine

  - Example:

```
SUBROUTINE add2(a, b, c)
    REAL, INTENT(IN) :: a, b
    REAL, INTENT(OUT) :: c
    c = a + b
END SUBROUTINE add2
```

  - Functions: result is a single value

```
[OUT_TYPE] FUNCTION function_name(argument_list)
    statements
    [RETURN]
END FUNCTION [function_name]
```

The `function_name` **must** appear on the LHS of at least one assignment statement in the function

  - Example:

```
REAL FUNCTION square(a)
    REAL, INTENT(IN) :: a
    square = a*a
END FUNCTION square
```

3 Application: Numerical Linear Algebra

Successive Over-Relaxation (SOR)

- Given the linear system

$$Ax = b$$

where $A \in \mathbb{R}^{n^2}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^n$, and initial guess $x^{(0)}$, compute the solution x using the SOR algorithm

- Iteration

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j>i} a_{ij}x_j^{(k)} - \sum_{j<i} a_{ij}x_j^{(k+1)}\right)$$
$$i = 1, 2, \ldots, n$$

where $1 < \omega < 2$

- Input: text file in the following format

$$
\begin{array}{|ccc|}
\hline
n & & \\
a_{11} & \cdots & a_{1n} \\
\vdots & \ddots & \vdots \\
a_{n1} & \cdots & a_{nn} \\
b_1 & & \\
\vdots & & \\
b_n & & \\
x_1^{(0)} & & \\
\vdots & & \\
x_n^{(0)} & & \\
\hline
\end{array}
$$

- Output: text file in the following format

$$
\begin{array}{|c|}
\hline
iter \\
x_1 \\
\vdots \\
x_n \\
\hline
\end{array}
$$

- Solution strategy
  - Create procedure to read input data
  - Create procedure to apply the algorithm
  - Create procedure to write the output results