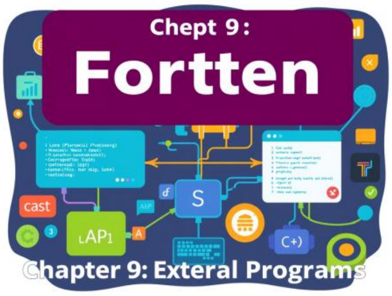
Chapter 9:
Calling External Programs



Chapter 9: Calling External Programs in Fortran

9.1 Introduction

Fortran allows interaction with external programs, enabling users to:

- Execute system commands.
- Call external scripts (Python, Bash, etc.).
- Integrate with external libraries (C, C++, etc.).
- Read and write data to external files for further processing.

This is particularly useful in scientific computing, where Fortran programs often need to interact with tools such as **MATLAB, Python, C, and Shell scripts**.

9.2 Calling System Commands

Fortran provides a built-in function SYSTEM to execute shell commands.

9.2.1 Example: Running a Shell Command

```
   IMPLICIT NONE
   INTEGER :: status
   ! Execute a system command
   status = SYSTEM("ls -l")  ! Lists files in Linux/Mac (use "dir" on Windows)
   ! Check the status of the command execution
   IF (status /= 0) THEN
      PRINT *, "Error executing command"
   ELSE
      PRINT *, "Command executed successfully"
   END IF
END PROGRAM call_system_command
```

- SYSTEM("command") executes the given shell command.
- Returns 0 if the command was successful, otherwise returns an error code.

9.2.2 Example: Running an External Python Script

```
PROGRAM call_python_script
  IMPLICIT NONE
  INTEGER :: status

  ! Run a Python script
  status = SYSTEM("python my_script.py")

  ! Check execution status
  IF (status /= 0) THEN
    PRINT *, "Python script execution failed"
  ELSE
    PRINT *, "Python script executed successfully"
  END IF
END PROGRAM call_python_script
```

9.3 Calling External Programs with Arguments

Sometimes, it is necessary to pass arguments to external programs.

9.3.1 Example: Running a Python Script with Arguments

```
PROGRAM call_python_with_args
  IMPLICIT NONE
  INTEGER :: status
  CHARACTER(LEN=100) :: command
  ! Construct command
  command = "python my_script.py 5 10"
  ! Execute command
  status = SYSTEM(command)
  ! Check status
  IF (status /= 0) THEN
    PRINT *, "Execution failed"
  ELSE
    PRINT *, "Execution successful"
  END IF
END PROGRAM call_python_with_args
```

- The Fortran program calls my_script.py and passes 5 and 10 as arguments.
- The Python script would then receive these numbers as input.

9.4 Calling C/C++ Functions from Fortran

Fortran can call **C or C++** functions using the ISO_C_BINDING module.

9.4.1 Writing a C Function

Create a C file my_c_function.c:

```
c
#include <stdio.h>

void print_message() {
   printf("Hello from C!\n");
}
```

Compile it as a shared library:

```
gcc -c -fPIC my_c_function.c
gcc -shared -o libmy_c_function.so my_c_function.o
```

9.4.2 Calling the C Function in Fortran

```
PROGRAM call_c_function
   USE, INTRINSIC :: ISO_C_BINDING
   IMPLICIT NONE
   INTERFACE
      SUBROUTINE print_message() BIND(C)
      END SUBROUTINE print_message
   END INTERFACE
   ! Call the C function
   CALL print_message()
END PROGRAM call_c_function
```

To compile and link:

```
gfortran call_c_function.f90 -L. -lmy_c_function -o call_c_function.out
```

- The ISO_C_BINDING module allows Fortran to call C functions.
- The BIND(C) attribute ensures compatibility with C.

9.5 Calling Fortran from C

We can also call Fortran functions from C.

9.5.1 Writing a Fortran Function

Create my_fortran_function.f90:

```
SUBROUTINE hello_from_fortran() BIND(C)
   IMPLICIT NONE
   PRINT *, "Hello from Fortran!"
END SUBROUTINE hello_from_fortran
```

Compile it:

```
gfortran -c -fPIC my_fortran_function.f90
gcc -shared -o libmy_fortran_function.so my_fortran_function.o
```

9.5.2 Calling Fortran from C

Create call_fortran_from_c.c:

```
#include <stdio.h>
void hello_from_fortran();
int main() {
   printf("Calling Fortran function from C...\n");
   hello_from_fortran();
   return 0;
}
```

Compile and link:

```
gcc call_fortran_from_c.c -L. -lmy_fortran_function -o call_fortran
```

9.6 Using File-Based Communication

Another method for calling external programs is **file-based communication**.

9.6.1 Example: Writing Data for a Python Script

```
PROGRAM write_data
   IMPLICIT NONE
   INTEGER :: i
   REAL :: x, y
   OPEN(10, FILE='data.txt', STATUS='REPLACE')

   ! Generate data
   DO i = 1, 100
      x = REAL(i) / 10.0
      y = SIN(x)
      WRITE(10,*) x, y
   END DO
   CLOSE(10)
   PRINT *, "Data written to file."
END PROGRAM write_data
```

Python Program (plot_data.py):

```
import numpy as np
import matplotlib.pyplot as plt

# Read data from file
data = np.loadtxt("data.txt")
x, y = data[:, 0], data[:, 1]

# Plot data
plt.plot(x, y, label="Fortran Data")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()
```

Running write_data first writes a file, then plot_data.py reads and plots it.

9.7 Summary and Best Practices

Method	Use Case	Advantages	Limitations
SYSTEM	Calling shell commands	Simple and direct	OS-dependent
Calling Python	Interfacing with Python scripts	Easy to extend	Requires Python
Calling C/C++	Using external libraries	Efficient	Requires ISO_C_BINDING
File-based	Exchanging data between programs	Works across languages	File I/O overhead

- **Use SYSTEM** for quick OS commands.
- **Use ISO_C_BINDING** for high-performance computing.
- **Use file-based communication** if data exchange is needed.

9.8 Conclusion

- Fortran can **call system commands, external programs, and scripts**.
- It can **integrate with C/C++ using ISO_C_BINDING**.
- It can **exchange data via files** for interoperability with other languages.