Introduction to C++ and Modern Fortran

Prof. Jeremy Roberts

Fall 2017

Outline

Hello World

```cpp
#include <iostream>

int main(int argc, char* argv[])
{
 /* Comments can continue
    on multiple lines */

 // or just be one-liners
 std::cout << "Hello World!"
           << std::endl;

 // Because "main" is an integer
 // function, it must return an
 // integer.
 return 0;
}
```

hello_world.cc

```fortran
program hello_world

 ! Fortran comments start with
 ! exclamation points, and there
 ! is not a multiline option

 print *, "Hello world!"

end program hello_world
```

hello_world.f90

Compiling Your First Program

For C++, use (in the command line)

$$\overbrace{\texttt{g++}}^{\text{compiler}}\ \underbrace{\texttt{hello\_world.cc}}_{\text{file to compile}}\ \overbrace{\texttt{-o}}^{\text{output as}}\ \underbrace{\texttt{hello\_world}}_{\text{this executable}}$$

For Fortran, use

$$\overbrace{\texttt{gfortran}}^{\text{compiler}}\ \underbrace{\texttt{hello\_world.f90}}_{\text{file to compile}}\ \overbrace{\texttt{-o}}^{\text{output as}}\ \underbrace{\texttt{hello\_world}}_{\text{this executable}}$$

Use `sudo apt-get install g++ gfortran` to get them. **Now try them!**

Compiler Options

`g++` and `gfortran` are part of the GNU compiler set and share several key compiler options that may (or may not) work with compilers from other vendors; these include:

- `-Wall` – warn us of anything unexpected but make the executable
- `-Werror` – turn any warning into an error
- `-O` – (that's an "Oh") use optimization (or `-ON` for $N = 0, 1, 2, 3$ for various levels of optimization)
- `-g` – produce debugging information
- `-pg` – produce profiling information

Declaring Variables

```cpp
int main()
{
  // One can declare and then
  // define variables anywhere
  int a;
  double b;
  a = 123;
  b = 3.14;

  // One can also declare and
  // define simultaneously
  const int A = 123;
  double B = 3.14;
  float C = 3.14;

  return 0;
}
```

declaring.cc

```fortran
program declare_demo
  ! All Fortran variables must be
  ! declared before execution of
  ! statements.  These variables
  ! may be initialized, too.
  integer, parameter :: a = 123
  double precision :: b
  real :: c = 3.1415926535897932
  b = 3.1415926535897932
  print *, b
  print *, c
end program declare_demo
```

declaring.f90

Simple Math

```cpp
#include <cmath>
int main()
{
 double x = 1.0;
 double y = 2.0;
 double z;
 z = x/y;
 z = sqrt(x);
 z = exp(y);
 z = pow(x, y);
 z = M_PI; // cmath has Pi
 return 0;
}
```

simple_math.cc

```fortran
program simple_math
  implicit none
  double precision :: x, y, z
  x = 2.0
  y = 3.0
  z = x/y
  z = sqrt(x)
  z = exp(y)
  z = x**y
  ! no built-in Pi definition
end program simple_math
```

simple_math.f90

Control of Program Flow – If's

```cpp
#include <iostream>
using std::cout;
using std::endl;
int main()
{
  int a = 1;
  if (a > 2)
  {
    // do something
  }
  else
  {
    // do something else
  }
  if (a == 1)
    a; // do something
  else if (a > 4)
    a;
  else
    a; // do somthing else
  if (a==1) cout<<"hi"<<endl;
  return 0;
}
```

control.cc

```fortran
program control
  integer :: a = 1
  if (a == 1) then
    print *, "a = 1"
  else if (a == 2) then
    print *, "a = 2"
  else
    print *, "a < 1 || a > 2"
  end if
  if (a == 1) print *, "hi"
end program control
```

control.f90

Control of Program Flow – Switches

```cpp
#include <iostream>
using std::cout;
using std::endl;
int main()
{
  int a = 1;
  switch (a)
  {
    case 1:
      cout << "a=1" << endl;
      break;
    case 2:
      // do something
      break;
    default:
      cout << "hi" << endl;
  }
}
```

control2.cc

```fortran
program control
  integer :: a = 1
  select case (a)
    case (1)
      print *, "a = 1"
    case (2)
      print *, "a = 2"
    case default
      print *, "a < 1 || a > 2"
 end select
end program control
```

control2.f90

Loops

```cpp
int main()
{
  int j1 = 0;
  int j2 = 0;
  for (int i = 0; i < 100; ++i)
  {
    j1 = j1 + i;
    j2 += i;
  }
  int i2 = 0;
  j1 = 0;
  do
  {
    j1 += i2;
    i2++;
  }
  while(i2 < 100);
  return 0;
}
```

loops.cc

```fortran
program loops
  integer :: i, j
  j = 0
  do i = 1, 100
    j = j + i
  end do
  i = 1
  j = 0
  do while (i < 100)
     j = j + i
     i = i + 1
  end do
end program loops
```

loops.f90

Functions

```cpp
#include <iostream>
using std::cout;
using std::endl;
int add(int a, int b)
{
  cout << "add ints" << endl;
  return a + b;
}
int add(double a, double b)
{
  cout << "add doubles" << endl;
  return a + b;
}
int main(int argc, char* argv[])
{
  cout << add(1, 2) << endl;
  cout << add(1.0, 2.0) << endl;
  return 0;
}
```

functions.cc

```fortran
program functions
  interface add
    real function add_d(x, y)
      real, intent(in) :: x, y
    end function add_d
    integer function add_i(x, y)
      integer, intent(in) :: x, y
    end function add_i
  end interface
  print *, add(1, 2)
  print *, add(1.0, 2.0) !!!
end program functions
real function add_d(x, y)
  real, intent(in) :: x, y
  print *, "add reals"
  add_d = x + y
end function add_d
integer function add_i(x, y)
  integer, intent(in) :: x, y
  print *, "add ints"
  add_i = x + y
end function add_i
```

functions.f90

Eclipse with C++ and Fortran

- Go to `eclipse.org` and download the Eclipse installer
- Install Eclipse for Parallel Application Developers

Command Line Arguments

```cpp
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
int main(int argc, char* argv[])
{
 if (argc != 2)
 {
  cout << "usage: " << argv[0]
       << " <arg>" << endl;
 }
 else
 {
  std::string s = argv[1];
  cout << "arg = " << s << endl;
  int n = 1;
  if (!(istringstream(s) >> n))
    n = 0;
  cout << "n = " << n << endl;
 }
 return 0;
}
```

cl.cc

```fortran
program command_line
 implicit none
 character(80)  :: s
 integer :: n = 1, io
 if (command_argument_count() &
     .lt. 1) then
  stop "usage: a.out <arg>"
 else
  call get_command_argument(1,s)
  print *, "s = ", s
  read (s, *, iostat=io) n
  if (io .ne. 0) n = 0
  print *, "n = ", n
  end if
end program command_line
```

cl.f90

File I/O

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
int num_lines(std::string name){
 std::ifstream f(name.c_str());
 std::string line; int i = 0;
 for(;std::getline(f, line);++i)
  continue;
 return i;
}
int main(){
 int n = num_lines("data.txt");
 std::ifstream f("data.txt");
 std::vector<double> T(n),rho(n);
 for (int i = 0; i < n; ++i)
 {
  f >> T[i] >> rho[i];
  std::cout << T[i] << "\n";
 }
 f.close();
}
```

file_io.cc

```fortran
program file_io
  integer :: i, n
  real,allocatable :: T(:), rho(:)
  n = num_lines("data.txt")
  allocate(T(n), rho(n))
  open (unit=5, file="data.txt", &
        action="read")
  do i = 1, n
   read(5, *) T(i), rho(i)
  end do
end program file_io

integer function num_lines(s)
  character(len=*) :: s
  integer :: io=0
  num_lines=0
  open(unit=5,file=s,action="read")
  do while (1 .eq. 1)
    read(5, *, iostat=io)
    if (io < 0) exit
    num_lines = num_lines + 1
  end do
  close(unit=5)
end function num_lines
```

file_io.f90

File I/O

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
int num_lines(std::string name){
 std::ifstream f(name.c_str());
 std::string line; int i = 0;
 for(;std::getline(f, line);++i)
  continue;
 return i;
}
int main(){
 int n = num_lines("data.txt");
 std::ifstream f("data.txt");
 std::vector<double> T(n),rho(n);
 for (int i = 0; i < n; ++i)
 {
  f >> T[i] >> rho[i];
  std::cout << T[i] << "\n";
 }
 f.close();
}
```

file_io.cc

```fortran
program file_io
  integer :: i, n
  real,allocatable :: T(:), rho(:)
  n = num_lines("data.txt")
  allocate(T(n), rho(n))
  open (unit=5, file="data.txt", &
        action="read")
  do i = 1, n
   read(5, *) T(i), rho(i)
  end do
end program file_io

integer function num_lines(s)
  character(len=*) :: s
  integer :: io=0
  num_lines=0
  open(unit=5,file=s,action="read")
  do while (1 .eq. 1)
    read(5, *, iostat=io)
    if (io < 0) exit
    num_lines = num_lines + 1
  end do
  close(unit=5)
end function num_lines
```

file_io.f90

C++ Arrays - Main Program I

```
// demonstration of basic arrays in C++

// system-level includes
#include <vector>
#include <cstdio>

// local includes
#include "basic_arrays_functions.hh"

int main(int argc, char* argv[])
{
    // A fixed-sized array of floating-point values
    int n = 100;
    double a[n];
    for (int i = 0; i < n; ++i)
        a[i] = 1.0;

    // A dynamically-sized array of the same
    double *b = new double[n];
    for (int i = 0; i < n; ++i)
        b[i] = 2.0;

    // A dynamically-sized "array" using std::vector
    std::vector<double> c(n, 3.0);

    // How to pass arrays?
    passing_dumb_arrays(a, n);
    passing_dumb_arrays(b, n);
```

C++ Arrays - Main Program II

```
    passing_dumb_arrays(&c[0], n); // dumb pointer under the hood!

    // How about vectors?
    std::printf("           original value of c[1]: %6.2f\n", c[1]);
    pass_vector_by_value(c);
    std::printf("value of c[1] after pass by value: %6.2f\n", c[1]);
    pass_vector_by_reference(c);
    std::printf("  value of c[1] after pass by ref: %6.2f\n", c[1]);
    pass_vector_by_pointer(&c);
    std::printf("  value of c[1] after pass by ptr: %6.2f\n", c[1]);

    // what about 2-D arrays?
    double a2[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};
    std::printf("  a2[1][1] is %6.2f\n", a2[1][1]);
    double **b2;
    b2 = new double*[3]; // an array of pointers;
    int k = 1;
    for (int i = 0; i < 3; ++i)
    {
      b2[i] = new double[3];
      for (int j = 0; j < 3; ++j)
      {
        b2[i][j] = ++k; // 6? what if k++?
      }
    }
    std::printf("  b2[1][1] is %6.2f\n", b2[1][1]);

    // are we forgetting something?
```

C++ Arrays - Main Program III

```
    return 0;
}
```

basic_arrays.cc

C++ Arrays - Function Header I

```cpp
// include guard
#ifndef basic_arrays_functions_hh
#define basic_arrays_functions_hh

// system-level includes
#include <vector>

// declare helpful typedef (a "shortcut")
typedef std::vector<double> vec_dbl;

// declare functions, etc.
void passing_dumb_arrays(double *a, const int n);
void pass_vector_by_value(vec_dbl a);
void pass_vector_by_reference(vec_dbl &a);
void pass_vector_by_pointer(vec_dbl *a);

#endif // basic_arrays_functions_hh
```

basic_arrays_functions.hh

C++ Arrays - Function Definitions I

```
#include "basic_arrays_functions.hh"

#include <cstdio>

// Demonstrate how to pass dumb arrays.  Note, prefer to "const" all
// incoming scalars (ints, floats, etc.) that are not to change inside
// the function.  Pedantic, but "defensive"
void passing_dumb_arrays(double *a, const int n)
{
    // do something with the array
    std::printf("a[1] = %6.2f\n", a[1]);
}

void pass_vector_by_value(vec_dbl a)
{
    // do something with the array
    std::printf("a[1] = %6.2f\n", a[1]);
    // change an element
    a[1] = 99;
}

void pass_vector_by_reference(vec_dbl &a)
{
    // do something with the array
    std::printf("a[1] = %6.2f\n", a[1]);
    // change an element
    a[1] = 99;
}
```

C++ Arrays - Function Definitions II

```

void pass_vector_by_pointer(vec_dbl *a)
{
    // do something with the array
    std::printf("a[1] = %6.2f\n", (*a)[1]);
    // change an element
    (*a)[1] = 101;
}
```

basic_arrays_functions.cc

Compiling

Go ahead, try `g++ basic_array.cc`.

Compiling

Go ahead, try `g++ basic_array.cc`.

The error is a **linking error**.

Compiling

Go ahead, try `g++ basic_array.cc`.

The error is a **linking error**.

Comment out the `#include "basic_arrays_functions.hh"` line in `basic_array.cc` and try compiling again.

Compiling

Go ahead, try `g++ basic_array.cc`.

The error is a **linking error**.

Comment out the `#include "basic_arrays_functions.hh"` line in `basic_array.cc` and try compiling again.

The error is a **syntactical error**—all names, including functions, need to be defined before use.

Compiling

Go ahead, try `g++ basic_array.cc`.

The error is a **linking error**.

Comment out the `#include "basic_arrays_functions.hh"` line in `basic_array.cc` and try compiling again.

The error is a **syntactical error**—all names, including functions, need to be defined before use.

The right way (after uncommenting the `.hh` file):

```
g++ basic_array_functions.cc basic_array.cc
```

(where the order matters!)

Compiling with `make`

```
# Make a program all at once
basic_arrays_cc:
        $(CXX) -o $@  basic_arrays_functions.cc basic_arrays.cc

# Or do it with dependencies
basic_arrays_cc:
        $(CXX) -o $@  basic_arrays_functions.cc basic_arrays.cc
```

make_basic_arrays

Fortran Arrays - Main Program I

```fortran
! demonstration of basic arrays in Fortran
program basic_arrays

use basic_arrays_module

implicit none
integer, parameter :: n = 5
! a statically-sized array
double precision :: a(n)
! a dynamically-sized array
double precision, dimension(:), allocatable :: b
! loop variables
integer :: i, j, k

a = 1.0_8 ! yes, all at once
a(:) = 1.0_8 ! equivalent

allocate(b(n))
do i = 1, n
    b(i) = 2.0_8
end do

! How to pass arrays?
print 666, '           original value of b(2): ', b(2)
call pass_array_1(b)
print 666, ' value of b(2) after pass_array_1: ', b(2)
call pass_array_2(b)
print 666, ' value of b(2) after pass_array_2: ', b(2)
```

Fortran Arrays - Main Program II

```fortran
call pass_array_3(b)
print 666, ' value of b(2) after pass_array_3: ', b(2)

! what about 2-D arrays?
allocate(a2(3, 3))
a2 = reshape((/ 1, 2, 3, 4, 5, 6, 7, 8, 9 /), (/3, 3/))
print *, a2
print *, a2(1, 1), a2(2, 1)


666 format(a, f6.2)

end program basic_arrays
```

basic_arrays.f90

Fortran Arrays - Functions Module I

```fortran
module basic_arrays_module

implicit none

! module-wide variable declarations, et.c
integer, parameter :: m = 5
double precision, allocatable, dimension(:, :) :: a2

contains

subroutine pass_array_1(a)
    double precision, dimension(:) :: a
    ! do something with the array
    print '(f6.2)', a(2)
    ! change an element
    a(2) = 99
end subroutine pass_array_1

subroutine pass_array_2(a)
    double precision, dimension(:), intent(out) :: a
    ! do something with the array
    print '(f6.2)', a(2)
    ! change an element
    a(2) = 101
end subroutine pass_array_2

subroutine pass_array_3(a)
    double precision, dimension(:), intent(inout) :: a
```

Fortran Arrays - Functions Module II

```fortran
    ! do something with the array
    print  '(f6.2)', a(2)
    ! change an element
    a(2) = 103
end subroutine pass_array_3

end module basic_arrays_module
```

basic_arrays_module.f90