

# Goals for This Lecture:

- Makefile macros
- Introduction to C++

# Makefile rules

- Each stanza of the Makefile has the following form:  
`target: dependencies`  
`command` (preceded by a tab character)
- Target can be a name of a file that will be created when one of its dependent files changes
  - Targets can be things other than file names as we will see shortly
- Dependencies is a list (separated by spaces) of files which the target depends on
- Command is a series of commands that are needed to accomplish a task (usually the creation of the target file from the dependency files)

# Executing make

- Make will try to execute the commands associated with first target listed in the Makefile
- If necessary, make will execute additional targets associated with the dependency files
- You can also have make execute a specific target by giving make an argument
- Example:
  - > `make sub1.o`
- It is common to create a “clean” target in a Makefile to cleanup after the compilation process

`clean:`

`/bin/rm main.o sub1.o sub2.o`

>`make clean` executes the “clean” target

# Makefile macros

- make allows the definition of macros (think “variables”) which simplify the construction and maintenance of makefiles
- Changing compiler flags for an entire code is simplified to changing a single line

```
F90 = ifort -c
LINK = ifort -o
main: main.o sub1.o sub2.o
    $(LINK) main main.o sub1.o sub2.o
main.o: main.f90
    $(F90) main.f90
sub1.o: sub1.f90
    $(F90) sub1.f90
sub2.o: sub2.f90
    $(F90) sub2.f90
```

# Welcome to C++

- C++ evolved from the C programming language
- Developed by Bjarne Stroustrup of AT&T Bell Labs in early 1980s
- Most of C++ is a superset of C
- C evolved from B in the 1970's as the programming language associated with Unix
- C quickly became available for many other operating systems
- C++ is designed as an Object-Oriented Programming (OOP) language
- Much more so than F95
- Fortran 2003 will has many of the same OOP features of C++

# C++ Terminology

- All program units in C++ are called functions
- Subprograms, procedures, methods, and functions are all the same thing in C++
- A main program is just a function called `main`
- When you execute the program you can think of the function as being called by the shell
- The program will return a value to the shell in accordance with its type (more on this later)
- **Important note: C++ is case sensitive unlike FORTRAN**
- This means that variables named `Xvel`, `XVEL`, and `xvel` are all different variables!

# Hello World in C++

```
// File:      hellow.cpp
// Program:   A hello world example
// Author:    Doug Swesty
// Date:      11/09/2005
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello World!\n";
    return 0;
}
```

- Compilation & linking: `>icc -o hellow hellow.cpp`

# A Few Important Lexical Rules

- `//` denotes a comment statement
- Anything that follows `//` is considered a comment
- Another form of comment delimiters is `/*` to begin a comment and `*/` to end a comment
- Most statements must end with a semicolon `;` to indicate the end of the statement

# Variables

- Variable names, or anything else defined in a C++ program, are called identifiers
  - A C++ identifier must start with either a letter (A-Z or a-z) or an underscore character ( \_ )
  - Other characters in identifiers must be letters, digits, or underscores
  - Remember: C++ is case sensitive!
- Certain identifiers, such as `cout` and `if` are called keywords and may not be used as variable names

# Variable Types

C++ Type	FORTRAN Equivalent
<code>short</code> (also called <code>short int</code> )	<code>INTEGER(KIND=2)</code>
<code>int</code>	<code>INTEGER</code>
<code>long</code> (also called <code>long int</code> )	<code>INTEGER</code>
<code>float</code>	<code>REAL</code>
<code>double</code>	<code>REAL(KIND=8)</code>
<code>long double</code>	<code>REAL(KIND=8)</code>
<code>char</code>	<code>CHARACTER(LEN=1)</code>
<code>bool</code>	<code>LOGICAL</code>
<code>unsigned short</code>	none
<code>unsigned int</code>	none
<code>unsigned long</code>	none
none	<code>complex</code>

# Assignment Statements

- Like FORTRAN, in C++ the `=` is the assignment operator
  - Must add a `;` to indicate the end of the statement
- Example:
  - `x_coordinate = 2.5 ;`
- In C++ assignment statements can be used as expressions
- Example:
  - `x_coordinate = (y_coordinate) = 2.5 ;`
  - or
  - `x_coordinate = y_coordinate = 2.5 ;`
- Don't do this (bad programming style)
- A simple typo will create problems. The compiler will accept both:
  - `x_coordinate = y_coordinate = 2.5 ;`
  - and
  - `x_coordinate = y_coordinate + 2.5 ;`

# Declaration Statements

- Declarations in C++ are nearly the same as they are in FORTRAN

- Form:

```
type variable1_name, variable2_name, ...
```

- Example:

```
int counter;  
float x_coordinate, y_coordinate;
```

- Variables can be initialized in declaration statements just as in FORTRAN

- Examples:

```
int counter = 0;  
float x_coordinate = 2.5, y_coordinate = 2.5;
```

- Just as in FORTRAN, uninitialized variables in C++ cannot be used in expressions or to pass in values to a function.

- Good programming style tip: Use lower case variables for variable names in C++ code except for uppercase letters to distinguish the beginning of a word in a name

- Example:

```
int ssnOfStudent;
```

# Assignment Compatibility

- Don't assume the same rules for assignment conversion that apply to FORTRAN will apply to C++
- In particular, statements like:  

```
int counter = 2.5 ;  
float velocity=2 ;
```

should be avoided
- Results may vary from compiler to compiler
- We will discuss another method is type conversion, called casting, shortly

# Literals

- Literals can be specified similarly to how they are specified in FORTRAN

- Examples:

```
int counter = 2 ;
```

```
char key= 'A' ; // Must be single quotes
```

```
float velocity=2.0 ;
```

```
double gravity=6.672e-8;
```

- Character strings are denoted with double quotes

- Example:

```
"Hello World"
```

- Note:

```
"A" is not the same as 'A'
```

# Escape Sequences

- Character Strings can contain Escape Sequences that control input and output

Sequence	Meaning
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Tab character
<code>\a</code>	Alert (causes a beep)
<code>\\</code>	Backslash (allows a backslash in character string)
<code>\'</code>	Single quote (allows a single quote inside single quotes)
<code>\"</code>	Double quote (allows a double quote inside double quotes)
<code>\v</code>	Vertical tab
<code>\b</code>	Backspace
<code>\f</code>	Form-feed
<code>\?</code>	Question Mark

# Constants

- Constants, the C++ equivalent of FORTRAN parameters can be declared by adding the constant modifier to the declaration statement
- Examples:

```
const int COUNTER = 2 ;  
const char KEY= 'A' ;  
const float VELOCITY=2.0 ;  
const double GRAVITY=6.672e-8;
```
- Programming style tip: Use uppercase names to distinguish constants in C++ code

# Arithmetic Operators

- Most arithmetic operators in FORTRAN are also the same in C++

Purpose	C++ Operator	FORTRAN Operator
Addition	+	+
Subtraction	-	-
Multiplication	*	*
Division	/	/
Exponentiation	pow(,)	**
Modulo or remainder	%	mod(,)

# More Assignment Statements

- A shorthand notation exists for combining the assignment with an arithmetic operator

- Form:

```
Variable operator = Expression;
```

Which is equivalent to

```
Variable = Variable operator Expression ;
```

- Examples:

```
i += 2
```

Which is equivalent to

```
i = i+2;
```

```
i *= 2
```

Which is equivalent to

```
i = i*2;
```

# Integer and Floating-point division

- Integer and floating point division works like it does in FORTRAN

Expression Form	Result
int/double	double
double/int	double
double/double	double
int/int	int

# Reading Assignment

- Read the “Introduction to make” tutorial linked on the course home page
- Read Chapter 1 of Savitch
- Get your Hello World! Example to run