Introduction to C++
[Appendix B]

Introduction to C++

- C++ is a superset of C
- (Almost) any legal program in C is also a legal C++ program
- The biggest changes are those to support "object-oriented" programming
  - We'll introduce these gradually

A Few Simple Extensions

- **cout** for output to screen
- Comments
- Declaration Statements
- Symbolic Constants
- Reference Parameters
- Enumerated Types

cout for output to screen

```
#include <iostream.h>
cout << "All mimsy were the borogroves"
     << endl;
```

- puts a message on the screen
- It's part of the stream I/O package
- We'll study in much more detail soon
- Meanwhile -- try it out!

Two Styles of Comments

- Old C-style comments

```
/* this is a comment */
```

- Double-slash comments (comment extends from the `//` to the end of the line)

```
int id; // Student ID number
```

- Which form is better?

Declarations Are Statements

- C++ declarations are statements, and can appear anywhere a normal statement can

```
void aFunction( int x )
{
     if( x == 10 )
          x = x / 2;
     int y; // Declaration OK
     …
}
```

Extended **for** Loop Syntax

- Initializer of **for** loop can be statement (in particular, declaration)

```
…
for( int row = 0; row < 4; row++ ) {
     for( int col = 0; col < 4; col++ ) {
          matrix[ row ][ col ] = 0;
     }
}
…
```


Symbolic Constants

- Symbolic constants in C: use **#define**

```
#define PI 3.1415926
```

- Why not? Because no type checking is performed.
- C++ **const** good for symbolic constants

```
const double PI = 3.1415926;
```

- Lets the compiler do more work

Parameter Passing

- Pass by value (as in C):
  - Passes a copy of the actual parameter
  - Assignments to formal don't affect actual parameters
- Pass by reference:
  - Makes formal an **alias** for actual parameter
  - Like passing address of actual, without messy **&** and *
  - Assignments to formal **do** change actual

Pass by Reference

- Place **&** between type and parameter name

```
void swap( int& x, int& y )
{
     int tmp = x; x = y; y = tmp;
}

int a = 15; int b = -21; swap( a, b );
```

- What if **a** and **b** were passed by value?

Enumerated Types

- User-defined type whose constants are meaningful identifiers, not just numbers

```
enum Colour { RED, GREEN, CHARTREUSE };
```

- Declare like other types; use like other integer types

```
Colour pants_colour;
if( pants_colour == CHARTREUSE ) {
     cout << "Yuck!" << endl;
} else if ( pants_colour == RED ) { ... }
```


Type Casting

- Don't use it unless really necessary.
- C style casts (prefix notation):

```
anInt = (int)aFloat;
```

- C++ style casts (functional notation):

```
anInt = int(aFloat);
```

New bool type

- C convention is 0 is false, non-0 is true
- C++ **bool** has two legal values: **true** and **false**
  - **bool**, **true** and **false** are all reserved words now
  - Direct implementation of the "Boolean" concept
- Not supported in earlier C++ compilers
- Not mentioned in earlier textbooks

int vs. bool (I)

- Under the hood, **bool** is an **int** type
- Use **bool** where Boolean values are natural

```
int i; bool b = true;
b = (mass >= 10.8);       // b is true or
                          // false
if( b ) { … }             // OK
while( b && !(i < 15) ) {// OK
     …
}
```


int vs. bool (II)

- Things to avoid:

```
i = b;     // marginally OK: value
           // is 0 or 1
i = true; // OK, but bad style
b = i;     // ill-advised (warning)
```

- **cout <<**
  - displays 0 or 1 for **bool** values
- How can we convert between **int** and **bool**?