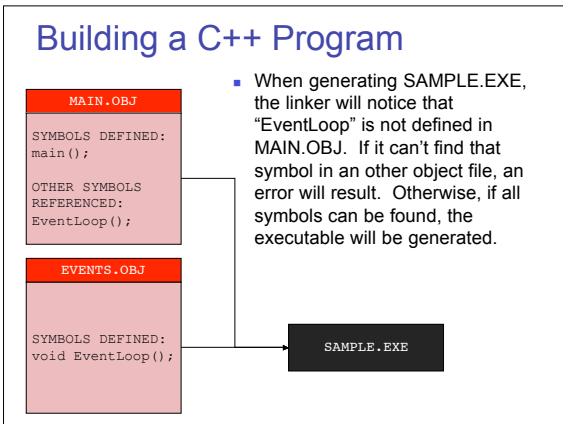
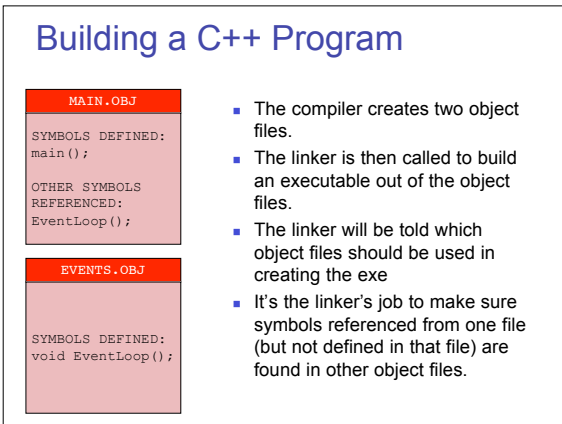
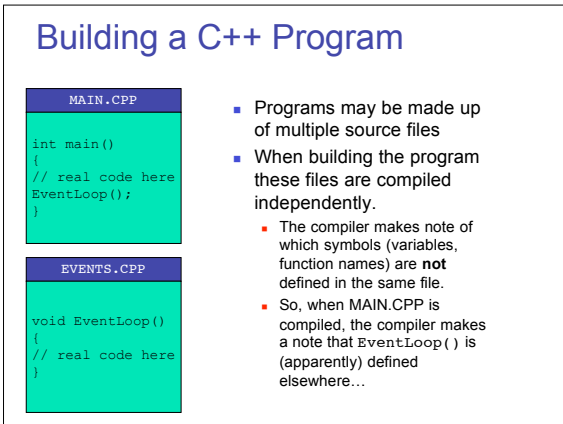
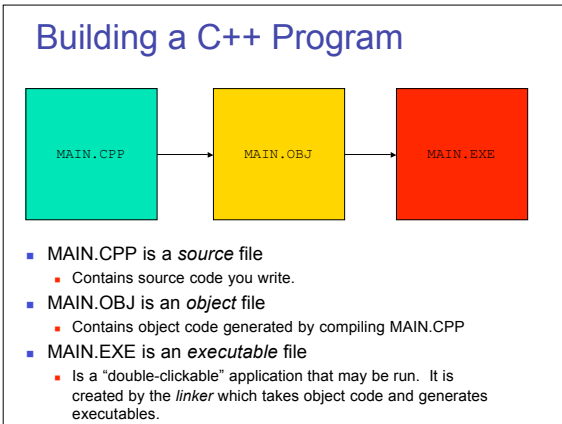


Lecture 1

The Basics (Review of Familiar Topics)

Building a C++ Program

- A C++ Program consists of
 - Source files
 - Header files
- Source Files
 - Usually have a .cpp or .cp extension
 - Compiled individually
- Header Files
 - Usually have a .h extension (some use of .hpp)
 - Contain declarations which are "included" by source files.
- Let's take a quick look at how a C/C++ program is built
 - "From 10,000 feet" (I'll omit some details to keep it simple)



A Simple C++ Program

```
#include <iostream> // header file

void main()

{
    cout << "Hello World!" << endl;
}
```

- `#include <iostream>` -- needed to access I/O streams (console)
- `void main()` -- main function-Entry point into your program
- `{,}` -- Scope delimiters
- `cout` -- the standard output identifier (console)
- `<<` -- Special operator which takes contents to the right and sends them to the left
- `endl` -- special identifier which sends a newline

Demonstration #1

Let's compile it!

Some Simple C++ Type Declarations

```
int j;

float interestRate;

char aLetter;

string userName;
```

- `int` -- integer type: range is implementation dependent
 - usually 32-bits -- +/- 2,147,483,648
 - 16-bits on older systems -- +/- 32,768
- `float` -- floating point number
- `char` -- a single character
- `string` -- more than an array of characters (a class)
 - we'll look at these in more detail later...

How to Assign Values

```
main()
{
    int j = 0;
    int k = 1;
    float pi;

    pi = 3.14159;
}
```

- **Assignment at declaration time**
 - insert an equals sign followed by an initial value
- **Assignment of previously declared variable**
 - start with the variable name, follow with equals sign, end with value to be assigned.

Arithmetic Expressions

```
main()
{
    int j = 0, k = 1, m = 2, n, p, q, r;
    float f;

    n = j + k; // Add j and k, place in n
    p = n * m; // Multiply n and m, store in p
    q = p / 4.0; // Divide p by 4, place in q
    r = q - 1; // Subtract 1 from q, place in r
}
```

- Can be used to calculate a value to be assigned
- What is wrong with the division expression?
- When assigning values to variables, the value is always coerced to the type of the variable it is getting assigned to.

Arithmetic Expressions (cont)

```
main()
{
    int j = 0, k = 5;

    j = j + 1;
    k = k - 5;
}
```

- **The same variable may appear on both sides of an assignment operator**
 - on the right hand side of the assignment operator, the variable in question represents its value prior to the execution of this statement.
 - on the left hand side of the assignment operator, the variable receives a new value which is the result of the evaluation on the right hand side.
 - In our example above, j ends up being "1" and k ends up being "0".

Arithmetic Expressions (shortcuts)

```
main()
{
    int j = 0, k = 5;

    j++;          // really like j = j + 1;
    k -= 5;       // really like k = k - 5;
}
```

- When incrementing an integer variable by "1", just append a ++ to the variable name.
- When decrementing by "1", just append a "--" to the variable name.
- When performing any other operation on a variable and stuffing the value back into the same variable, use a shortcut (like +=, -=, *=)

Arithmetic Expressions (prefix vs. postfix)

```
main()
{
    int j = 0, k = 0, q, r;

    q = j++;      // Postfix operation
    r = ++k;      // Prefix operation
}
```

- When the "++" appears after a variable it is said to be a "postfix operator"
 - the variable isn't incremented until all other evaluations (and assignments) have taken place
- When the "++" appears *before* a variable it is said to be a "prefix operator"
 - the variable is incremented *before* any other evaluations take place.
- What will the values of q & r be in the example above?

Demonstration #2

Arithmetic Expressions, Shortcuts and Pre/Postfix Operators

Control Structures--if/else statements

```
if (expression)
    statement1
else
    statement2
```

- *expression* is any expression that can be evaluated as an integer
 - a non zero value is taken as "true", a 0 value is taken as "false"
- *statement1* is a statement or group of statements executed if *expression* evaluates to a non-zero value
- *statement2* is a statement or group of statements executed if *expression* evaluates to a zero value
 - *statement2* is needed only if the the optional `else` keyword is present

Control Structures--if/else statements

```
if (x = 0)
    cout << "It's zero" << endl;
else
    cout << "No, it's not zero!" << endl;
```

- **WARNING!!!!**
 - While the "if" statement above may look perfectly fine it contains a very common flaw.
 - The assignment operator (=) is not used to test for equality.
 - "x=0" is an expression which evaluates to "0" along with having the side effect of storing the value 0 in the variable "x".
 - As an expression which evaluates to "0" it will always cause the "else" branch to be executed.

Control Structures--if/else statements

```
if (x == 0)
    cout << "It's zero" << endl;
else
    cout << "No, it's not zero!" << endl;
```

- This is the correct way, use the equality operator (==)
- What are some of the other comparison operators?
 - (a > b), true if "a" is greater than "b"
 - (a < b), true if "a" is less than "b"
 - (a >= b), true if "a" is greater than or equal to "b"
 - (a <= b), true if "a" is less than or equal to "b"
 - (a != b), true if "a" is not equal to "b"

Control Structures--compound expressions

```
if ((x == 0) || (y > 1))
{
    cout << "x is zero OR" << endl;
    cout << "y is greater than 1" << endl;
}
```

- An expression with the logical "or" (||) operator...
 - Evaluates to "true" if an expression on either side evaluates to "true"
- An expression with the logical "and" (&&) operator...
 - Evaluates to "true" if the expressions on **both** sides evaluate to "true"
- Note the use of curly braces ({, }) above
 - Used to group multiple statements to be executed if the "if" statement evaluates to "true"

Demonstration #3

If/else statements

Control Structures--loops

```
while (expression)
    statement(s)
```

- A while loop will continue executing as long as *expression* evaluates to a non-zero (true) value.
- How do you print your name 10 times using a while loop?

```
int x = 0;
while (x < 10)
{
    cout << "Ron DiNapoli" << endl;
    x++;
}
```

- Why is it (x < 10) and not (x <= 10) ?

Control Structures--loops

```
int x;
while (true)
{
    cin >> x;
    if (x == 0)
        break;
    cout << "You entered the number " << x;
}
```

- A while loop can be used to loop forever by having it test for an expression which will always evaluate to a non-zero value (true)
- A `break` statement can be used to break out of such a loop when the time comes
- Some think that this is bad programming style, but it is frequently used.

Control Structures--do...while loops

```
int x;
do
{
    cin >> x;
    if (x == 0)
        break;
    cout << "You entered the number " << x;
} while(true);
```

- A `do...while` loop is very similar to a regular `while` loop.
 - Terminating condition is specified at the end of the loop

Demonstration #4

loops