This Time

- Whitespace and Input/Output revisited
- The Programming cycle
- Boolean Operators
- The “if” control structure
- LAB
  - Write a program that takes an integer from the user and prints “even” if the number is even and “odd” otherwise

Whitespace

- Whitespace consists of spaces, newlines, and tabs. C++ treats all whitespace as if it were just one space
  - e.g. int x is the same as int   x
- You do need white space to separate names and keywords in C++
  - e.g. int x is not the same as intx
- You do not need whitespace between a variable name or keyword and an operator
  - e.g. x + y, x   +   y, and x+y are all correct
  - e.g. cout<<x; is the same as cout   <<   x;

Input/Output, revisited

- Think of input and output as two streams of data.
- One stream flows out of your program to the screen
- One stream flow into your program from the keyboard
- “cout” is the name of the output stream
- “cin” is the name of the input stream

Input/Output, revisited

- In order to put things into the output stream so that they are displayed we used the stream insertion operator
  - e.g cout << x;
  - Think of the stream insertion operator as pointing x towards the "cout" output stream.

Input/Output, revisited

- To get values out of the input stream we use the stream extraction operator
  - e.g. cin >> x
  - Think of >> as pointing from the input stream “cin” towards the variable.
- You can have as many insertion and extraction operators as you want for each cout or cin.
  - So we can write cout << x << y << z;
  - Or cin >> x >> y >> z;

Input/Output, revisited

- We can also put text directly into the output stream with String Constants:
  - A string constant is a sequence of characters enclosed in double quotes
  - e.g. cout << “This is a string constant”;
  - There are special characters that we can include in our string constant that you can’t type on the keyboard.
  - These characters are represented special codes called escape sequences.

Input/Output, revisited

- If you want a newline for example:
  - cout << “Prints on line 1\nPrints on line 2”;

  Produces the output:

  Prints on line 1

  Prints on line 2
- cout << “Prints on line 1

  Prints on line 2”; is incorrect.

Input/Output, revisited

- We can also insert commands into the stream
  - e.g. cout << flush; tells the stream to add a display everything in the stream right away without adding a newline.
  - e.g. cout << endl; is the same as flush but it adds a newline as well.
  - There are many other commands that we can insert into the output stream, they generally determine the format of the output.

Current Assignments

- Homework 1 due in 5 days (June 16th)

  Variables, mathematical and logical operators, input/output, and the "if" operator.

  (After today' s class you should be able to do all the problems on Homework 1)

- Project 1 Due in 12 days (June 23rd)

  Write a binomial root solver using the quadratic equation.

  (After today' s class you should be able to write this program)

Last Time

- How to allocate memory for variables of different types.
- How to name those variables.
- The primitive mathematical operators we can apply to those variables.
- The order in which operators are applied.
- How to print the value of a variable.
- How to take a value from the keyboard and put it in a variable.

The Programming Cycle

- There are two kinds of programming language:
  - Some languages such as Lisp and Prolog have an interpreter that performs a statement as soon as you type it in. With these languages you get instant feedback on whether the statement you just wrote makes any sense or not.
  - Other languages such as Fortran and C++ are called compiled languages. They separate the process of writing the program, compiling it into machine code, and running the program into three separate steps.

The Programming Cycle

- The programming cycle for C++ consists of four main phases.

1) Design. Here the algorithm we want the program to execute is developed. There are lots of different tools we can use in the design phase: flow charts, pseudocode, UML being a few choices of how we can represent the algorithm or "logic flow" of the program.
2) "Coding" translating the algorithm into the particular syntax of the language we want to use to create the program.

The Programming Cycle

3) Compilation. Here the source file we wrote in the coding phase is checked for obvious errors by the compiler and then translated into machine code.
4) Debugging. There are two types of debugging:
   a. First we resolve any compilation errors. Compilation errors are typically the result of typos and careless mistakes with syntax.
   b. Once the program compiles and we run it we analyze its behavior to make sure that it does what we expected.

Boolean Operators

- Whereas the mathematical operators we saw before act on integers and floating point numbers boolean operators act on true/false values.

- All boolean operators return a value of the type bool which can only take on the values “true” or “false.”

Boolean Operators, Comparitors

- The primitive boolean operators are:
  - > greater than,
    e.g. 5 > 4 returns true
  - < less than,
    e.g. 5 < 4 returns false
  - >=, greater than or equal,
    e.g. 5 >= 5 returns true
  - <=, less than or equal
    e.g. 5 <= 2 returns false

Boolean Operators, Equality

- ==, equality

  e.g. 5 == 5 returns true

  5.5 == 4.3 returns false

- !=, not equal

  e.g. 5 != 5 returns false

  5.5 != 4.3 returns true

Boolean Operators, And

- &&, logical “and”

true && false returns false

true && true returns true

false && true returns false

false && false returns false

Boolean Operators, Or and Not

- ||, logical “or”

  true || false returns true

  true || true returns true

  false || true returns true

  false || false returns false
- !, logical “not”, logical negation

  !true returns false

  !false, returns true

Boolean Operators, Precedence

- ! (not) has the highest precedence of any operator
- Comparators have lower precedence than +, and –, * and /
- != and == have lower precedence than (other) comparators
- && has lower precedence than != and ==
- || has lower precedence than && but higher precedence than unary operators like +=

Boolean Operators, Precedence

- Expressions within parentheses are always evaluated first.
- If there are nested sets of parentheses the inner most parentheses are evaluated first.
- You can avoid all precedence issues in your own code by always using parentheses to force the order of evaluation.
- You have to know the order when reading other people’s code.

Boolean Operators, Example 1

```
int main
{
    bool x = true, y = false, result = true;

    result = x && y || y;
    cout << result << endl;
    return 0;
}
```

```
0
```

Boolean Operators, Example 3

```
int main
{
    int x = 6, y = 8, z = -1, w = 14,
    bool result = false;
    result = (x + z != w) && !(x > 6)
    cout << result << endl;
    return 0;
}
```

```
 1
```

Boolean Operators, Example 4

```
int main
{
    int x = 6, y = 8, z = -1, w = 14,
    bool result = false;
    result = (z == w) || (x <= 6)
    cout << result << endl;
    return 0;
}
```

```
1
```

The “if” Control Structure

- We would like our programs to be more than just calculators we want them to make decisions.
- Decision making in programming is called branching. The program goes down one branch if some condition is true and down another if that condition is false.
- Statements that make decisions about what branch of instructions to execute next are called control structures.
- The most common control structure is the “if” statement.

The "if" Control Structure

- The syntax of the if control structure is:

```
if ( boolean_expression )
{
        statements…
}
```

- If boolean_expression returns "true" then the statements inside the braces are executed. If the expression is false then those statements are skipped.

The “if” Control Structure, Example 1

```
#include <iostream>
int main()
{
    int x = 6, y = 12;
    if ( x > 5 )
    {
        cout << “y = “ << y << endl;
    }
    cout << “x = “ << x << endl;
    return 0;
}
```

The “if” Control Structure, Example 2

```
#include <iostream>
int main()
{
    int x = 6, y = 12;
    if ( x < 5 )
    {
            cout << “y = “ << y << endl;
    }
    cout << “x = “ << x << endl;
    return 0;
}
```

The “if” Control Structure, Example 2

```
#include <iostream>
int main()
{
    int x = 6, y = 12;
    if ( x < 5 || y != 15)
    {
        cout << “y = “ << y << endl;
    }
     cout << “x = “ << y << endl;
    return 0;
}
```

Lab – Parity Program

1) Login to UNIX with SSH
2) Start xwin32
3) Start emacs
4) Open a new file called parity.cpp
5) Write a program that takes an integer as input and displays "even" if the integer is even and "odd" otherwise
6) This program will require two "if" statements, one for even and one for odd
7) x % 2 returns 0 if x is even and 1 if x is odd.
8) Compile, debug, and run your program.