Lecture 2 Notes

Comparison Operators

You have already been breifly introduced to the arithmetic operators. Another class of operators you should know are the comparison operators:

Operator	Description
==	Is equal
!=	Is not equal
<	Is less than
>	Is greater than
<=	Is less than or equal to
>=	Is greater than or equal to

All of these operators are used for comparing the left operand with the right operand. An expression that uses this operator results in a value of type bool. For example:

```
bool result;
int x;
x = 10;
result = x > 9;
```

The variable result will have the final value "true".

```
bool result;
int a;
int b;
a = 5;
b = 4;
result = a == b;
```

In the above example, the variable result will have the final value "false". It is important to remember that the == operator is used for comparing two values while the = operator is for assigning to a variable.

In general, the arithmetic operators have greater precedence than the comparison operators, and the comparison operators have greater precedence than the assignment operator.

Conditional statements

The comparison operators are not very useful on their own without conditional statements. Conditional statements are used for executing certain blocks of code depending on a the value of a boolean expression.

The one conditional statement you should know in C++ is the "if statement". Here is an example:

```
int age;
cout << "Enter your age: ";
cin >> age;
if (age < 21) {
    cout << "Too young." << endl;
}
```

The "if statement" needs 2 things: a boolean expression (also known as a condition), and a block of code to execute if that condition is true. The condition goes inside a pair of parenthesis and the code block goes inside a pair of curly brackets.

Optionally, you may add another code block to execute if the condition passed into the "if statement" is determined to be false:

```
int age;
cout << "Enter your age: ";
cin >> age;
if (age < 21) {
    cout << "Too young." << endl;
} else {
    cout << "You are old enough." << endl;
}
```

You may also add additional code blocks after your initial condition and before your "else" clause for alternative conditions to be checked in order:

```
int age;
cout << "Enter your age: ";
cin >> age;
if (age < 21) {
    cout << "Too young." << endl;
} else if (age < 0) {
    cout << "Not born yet." << endl;
} else if (age > 120) {
    cout << "Dead." << endl;
} else {
    cout << "You are old enough." << endl;
}
```

Logical Operators

Logical operators allow you to check for multiple conditions in one "if statement" block. The first one is the Logical Or operator:

```
if (age < 21 or age > 120) {
    cout << "Too young or too old." << endl;
}
```

This statement translates to: If the age is less than 21 or the age is greater than 120 then display "Too young or too old.".

Complementary of the Logical Or operator is the Logical And operator:

```
if (age > 21 and age < 120) {
    cout << "You are old enough, but not too old." << endl;
}
```

The logical operators have a lower precedence than the comparison operators. Their left and right operands are treated as boolean values and the result of using a logical operator is a value of type boolean.

Here is a table of the possible results from the Logical And operator:

Left Operand	Right Operand	Result
false	false	false
false	true	false
true	false	false
true	true	true

From this table, you can see that the Logical And operator only yields true when both of it's operands are true. Here is the table for the Logical Or operator:

Left Operand	Right Operand	Result
false	false	false
false	true	true
true	false	true
true	true	true

The Logical Or yields true of either or both of it's operands are true.

Looping

Since programs are frequently created for processing input, programmers need to have a way to continuously execute code until some condition is met. C++ provides three statements for this purpose, one of them is the "while statement":

```
int i = 0;
int x = 2;
while (i < 8) {
    x = x * 2;
    i = i + 1;
}
cout << x;
```

The above code is a way to raise the value of variable x to the power of 8. The while loop requires a condition (in parenthesis) and a block of code (in curly brackets). The while loop will execute it's block of code while it's condition is true. This condition will be checked each time before the code block is executed.

In our while loop example, we have a variable i which gets incremented at the end of the code block. The while loop will stop when variable i reaches the value 8. Thus, since i starts at 0 and stops at 8, then the loop will iterate 8 times.

Since we multiply x by 2 eight times, starting at 2. Then this code is equivalent to 2^8. The final value of x is 256.

A simpler example using a while loop:

```
int n = 0;
while (n != 8) {
    cout << "Guess the magic number: ";
    cin >> n;
}
cout << "You won!" << endl;
```

In this example, the program asks the user to guess a number until they guess right.

An alternative to the "while" statement is the "for" statement. It looks like this:

```
int i;
for (i = 0; i < 100; i = i + 1) {
    cout << i << endl;
}
```

The above example just displays the numbers 0 through 99. Like the while statement, the for statement takes in a condition and a code block. Unlike the while statement, the for statement takes two additional statements, one is the initial statement ($i = 0$) and the other is the iteration statement ($i = i+1$). The initial statement gets exeuted exactly once before the for loop starts iterating. The condition statement gets executed at the beginning before each iteration, and the iteration statement gets executed at the end of each iteration.