Lecture 2 Notes

Review from Lecture 1

All computer programs are composed of statements. The types of statements that you should know are:

- Declaration: Produces a variable
- Expression: Produces a value

Values have data and a data type. Variables have a name and a value.
These are the types of expressions you should know:

- Literal value: Produces a value that is already known before compiling
- Assignment: Assigns a value to a variable
- Arithmetic expression: Produces a value through a mathematical calculation
- I/O expression: Transfers data

Comparison Expressions

A comparison expression is an expression that compares two values. Comparison expressions make use of the following 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.

Logical expressions

Logical expressions perform a logical evaluation on one or more boolean values. A logical expression may use any of these operators:

Operator	Description	Usage
`and`	Logical And	`a and b`
`or`	Logical Or	`a or b`
`not`	Negation	`not a`

The Logical And operator returns true if both of it's operands are true, otherwise it returns false. The Logical Or operator returns true if at least one of it's operands are true, otherwise it returns false. The Negation operator only takes one boolean operand, it returns the inverse value of it's operand.

Here are the "truth tables" for the logical operators:

Logical And		
Left Operand	Right Operand	Result
false	false	false
false	true	false
true	false	false
true	true	true

Logical Or		
Left Operand	Right Operand	Result
false	false	false
false	true	true
true	false	true
true	true	true

Negation	
Operand	Result
false	true
true	false

Here is an example of a logical expression being used in an assignment expression:

```
bool should_hire = ((knows_cpp and knows_english) or
    can_clean_toilets) and age >= 18 and not a_felon;
```

More Assignment expressions

You already know about the simple assignment expression using the = operator. Here are some more assignment operators:

Operator name	Usage	Description
Addition assignment	`a += b`	Adds a and b, then assigns result to a
Subtraction assignment	`a -= b`	Subtracts b from a, then assigns result to a
Multiplcation assignment	`a *= b`	Multiplies a and b, then assigns result to a
Division assignment	`a /= b`	Divides a by b, then assigns quotient to a
Modulo assignment	`a %= b`	Divides a by b, then assigns remainder to a

`a += b` produces the same effect as `a = a + b`. The same is true for all the other arithmetic assignment operators.

Increment and Decrement expressions

C++ supports short-hand ways to increment/decrement a variable:

Operator name	Usage	Description
Prefix Increment	`++x`	Increment x and return the new value of x
Prefix Decrement	`--x`	Decrement x and return the new value of x
Postfix Increment	`x++`	Increment x and return the old value of x
Postfix Decrement	`x--`	Decrement x and return the old value of x

`++x` produces the same effect as `x = x + 1` and `--x` produces the same effect as `x = x - 1`.

Selection statements

A selection statement is not an expression, it is a type of statement that selects the next statement to execute depending on the value of an expression. C++ supports two types of selection statements: the if statement, and the switch statement. We will only cover the if statement in this course, here is an example of it:

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

The if statement follows this general form:

`if (` *boolean expression* `)` *statement* `;`

The if statement executes it's inner statement if the provided boolean expression has the value `true`. Alternatively, you may provide a compound statement to the if statement. A compound statement contains multiple statements, it follows this form:

{ *statement* ; *statement* ; ... }

Thus, the an if statement can look like this:

`if (` *boolean expression* `)` { *multiple statements* }

Using this form, you can execute multiple statements if the boolean expression returns `true`. Each of the statements inside the compound statement must be terminated by semicolons.

The if statement can optionally have an else clause which allows you to select a statement to execute if the boolean expression evaluates to false. It follows this form:

`if (` *boolean expression* `)` *statement* `else` *statement*

The statement for the else clause can be another if statement, so you may chain another if statement to the else clause of the containing if statement like this:

```
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 {
    cout << "You are old enough." << endl;
}
```

The final else clause is, of course, optional.

Iteration statements

An iteration statement executes a statement multiple times, usually while a boolean expression (boolean expressions are also known as conditions) yields true. There are three types of iteration statements in C++: while statement, do-while statement, and for statement.

This is the syntax for the while statement:

`while (` *boolean expression* `)` *statement* `;`

The while statement will repeatedly do 2 things in this order: evaluate the boolean expression, execute the statement. If the boolean expression becomes false during it's evaluation, the while loop completes and the inner statement does not get executed again.

The do-while statement follows this form:

`do` *statement* `while (` *boolean expression* `);`

It is similar to the while statement except that the statement gets executed before the boolean expression gets evaluated.

The for statement follows this form:

`for (` *statement-1* `;` *boolean expression* `;` *statement-2* `)` *statement-3* `;`

The for statement executes *statement-1*, then it repeatedly performs the following in this order:

- Evaluate *boolean expression*
- Execute *statement-3*
- Execute *statement-2*

The for loop stops when the boolean expression becomes false by the time it is evaluated. Note that it is *statement-3* that initially gets executed first before *statement-2*. The for loop's *statement-3* can be a single statement or a compound statement, but *statement-1* and *statement-2* cannot be compound statements.

Listing 1: Example of calculating factorial of 100 using a while loop

```
int fac;
int n;
fac = n = 100;
while (n > 1) {
    fac *= --n;
}
```

Listing 2: Example of calculating 2 raised to the 100th power using a for loop

```
int i;
int n = 2;
for (i = 0; i < 100; ++i)
    n *= 2;
```