

Summary of Topics Covered by Lecture

Lecture 1

- The backbone of every introductory program should look like:

```
#include <iostream>
using namespace std;

int main()
{
...
}
```

- `#include <iostream>` is included at the beginning of the program to use commands like `cin` for input and `cout` for output
 - `cout` is always followed by a left shift operator (`<<`) and ends with a semicolon (`;`)
 - `cin` is always followed by a right shift operator (`>>`) and ends with a semicolon (`;`)
- `using namespace std;` permits you to use the standard library
- `endl;` moves the output to the next line
- There are two primary types of values we need to know for this class
 - `int` is used for integers
 - `double` is used for numbers with decimal points
- Include the following code if you want all subsequent outputs to have `n` decimal places of precision

```
cout.setf(ios::fixed);
cout.precision(n);
```

- Order of operations applies
 - Operations with equal precedence is read from left to right
- The operator `*` must *a/ways* be used for multiplication

- The following figure shows the **data type** that will result from an arithmetic operation:

```
int □ int = int
double □ int = double
int □ double = double
double □ double = double
```

where □ stands for an arithmetic operator (+, -, *, /)

- If the output is a double, most compilers will restrict it to **four decimal places**
 - The compiler will not write out **trailing zeros**

[End of Lecture 1 Summary](#)

Lecture 2

- `getline(cin, identifier);` is used to input **strings** that take up more than one word
- If your inputs are a number followed by a string, you must include the following code right after the number-based `cin` line

```
cin.ignore(10000, '\n');
```

- If-statements have the following structure:

```
if (condition)
    statement-if-true;

// The 'else' statement is optional
else
    statement-if-false;
```

- Signs for conditions are written in the following way
 - Less than (<)
 - Less than or equal to (<=)
 - Greater than (>)
 - Greater than or equal to (>=)
 - Not equal to (!=)
 - Equal to (==)

Notice that this is not written as (=).
- In general, if there is a double in an arithmetic expression, the output will also be a double
- Compound statements are written in the form:

```
{statement; statement; statement;}
```

- Assignment statements are used to set a variable equal to an expression and are in the form:

```
variable = expression; // If variable is already declared
type variable = expression; // If variable is not declared
```

- Pay attention to assignment (=) vs. equals (==)

Lecture 3

- **Organize** your program so it is easier for you to read
 - Group your code into sections, each fulfilling a certain function
 - Use the marking `// sample text` to **annotate** your code
- If you want an initialized variable to be constant, use `const` before declaring it


```
const type variable = number;
```

 - If you try to change a constant later on in the program, it will not compile!!!
- When writing if-statements, `else` always pairs up with the nearest preceding `if` that is unpaired
 - Use curly braces `{}` to get around this
- For if-statements with **multiple conditions**, use `&&` (and) or `||` (or)
 - `&&` has higher precedence than `||`
 - Use parentheses `()` to group certain conditions together
 - Equal precedence is read from left to right
- Write out the conditions **completely**

```
if (citizenship == "US" || citizenship == "Canada") // Valid statement
if (citizenship == "US" || == "Canada") // Error! Won't compile.
```

- **De Morgan's Law** states that when you want the opposite, you need to switch the `&&` and `||`'s
 - `not (A AND B)` turns into `(not A) or (not B)`
 - `not (A OR B)` turns into `(not A) AND (not B)`
- We also need to consider the inclusivities of greater-than and less-than signs
 - `not (a <= b)` turns into `a > b`
 - `not (a < b)` turns into `a >= b`
 - `not (a >= b)` turns into `a < b`
 - `not (a > b)` turns into `a <= b`
 - `not (a == b)` turns into `a != b`
 - `not (a != b)` turns into `a == b`

- An **if-ladder** is a series of if-statements that produce an output once specific conditions are met
 - Read from top to bottom
 - Once the condition has been met, the computer will ignore everything else in the if-ladder that comes after it

```
if (condition)
    ...
else
{
    if (condition)
        ...
    else
    {
        if (condition)
            ...
        else
        {
            ...
        }
    }
}
```

Lecture 4

- A neater way to write an if-ladder is called a **switch statement**
 - Only works with integers
 - Very ineffective for large ranges of numbers
 - In the following example, `a`, `b`, and `c` are integers that can be thought of as 1, 2, and 3

```
switch (variableName)
{
    case a:
    case c:
    ...
    break;

    case b:
    ...
    break;

    default:
    ...
    break;
}
```

- Initiate it `switch (variableName)`, followed by curly braces
 - Write out `case`, followed by the specific number you want to assign to it
 - `break` tells the program to ignore everything else that comes afterwards *within* the switch statement
- **While-loops** are similar to if-statements and are used to run the commands again until a condition is no longer satisfied
 - **Initialize** a new variable right before
 - The **stay-in-loop condition** compares some other input variable with the new variable
 - There's also an **assignment statement** at the end of the loop to redefine the new variable

- Example of a **while-loop**:

```
int nTimes;
cin >> nTimes; // Input variable

int n = 1; // New variable
while (n <= nTimes) // Stay-in-loop condition
{
    cout << "Hello" << endl;
    n = n + 1 // Reassigns n and loops back to the beginning of the while-statement
}
```

- **For-loops** serve the same purpose as a while-loop, but have a different format

```
for (initialize-new-variable, stay-in-loop-condition, reassignment-statement)
    statement
...
```

- **Assignment statements in loops** can be shortened in the following ways:

- `n = n + 1` can be shortened to `n += 1` or `n++` or `++n`
- `n = n - 1` can be shortened to `n -= 1` or `n--` or `--n`
- `m = m * 2` can be shortened to `m *= 2`
- `k = k / 5` can be shortened to `k /= 5`