Overview

1. Casting
2. Math Libraries
3. Strings
4. Next Time
1 Casting

2 Math Libraries

3 Strings

4 Next Time
What is Casting?

Consider the example code

```c
int main() {
    int x = 6;
    double y = 1.5;
    double z = y*x;
}
```

Will the compiler throw an error since we are multiplying a double and an integer?
What is Casting?

Consider the example code

```c
int main() {
    int x = 6;
    double y = 1.5;
    double z = y * x;
}
```

Will the compiler throw an error since we are multiplying a double and an integer?

The answer is no, due to *implicit casting*.

- Last time, we learned that every variable has a *type* associated with it, and associated with each type are some operations like addition, division, concatenation, etc. *Casting* addresses the situation where we wish to perform a valid operation on different types, like multiplying a double and an integer.
Which type of casting do you think is explicit? Implicit?

```java
int x = 6;
float y = x;

int k = 20;
float f = (float) k/10;
```
Which type of casting do you think is explicit? Implicit?

```java
int x = 6;
float y = x;

int k = 20;
float f = (float) k / 10;
```

The first case is called implicit, and is valid whenever the two types are compatible. Learning which types are compatible can be done through trial and error, like converting `char` to `long` is not.

Sometimes a loss of precision can occur, like `double` to `float`. This will cause a compiler warning if done implicitly, but not if it is done explicitly.

**Tip:** Always use a decimal when you want to do decimal operations, `double x = 1/2` will be zero, but `double x = 1.0/2` will be 0.5.
The main points of casting are:

- Casting is performed when a valid operation between different types is performed, i.e. addition between decimals and integers.
- Casting can be explicitly forced by the programmer, or handled implicitly by the compiler.
- Casting is also referred to as conversions or type casting.
- More advanced C++ classes may cover additional forms of casting like dynamic, static, and constant casting.
1. Casting

2. Math Libraries

3. Strings

4. Next Time
Math Libraries

Math is important, but having the computer do math for you is even more important. That is why some poor souls\(^1\) spent many hours developing a \texttt{c++} library consisting of all common math functions. Some functions include:

- \(\exp(x), \log(x), \sin(x), \text{etc.}\)
- \(\texttt{pow}(x, y)\) - computes \(x^y\)
- ceil, floor, max, min, etc.
- \(\texttt{fmod}(x, y)\) - computes remainder after division of \(x/y\)


\(^1\)Probably graduate students back in the day.
Although it contains syntax we haven’t seen, try to guess what this code does:

```c++
for (int k = 0; k < N; k++){
    result += pow(x, k)/factorial(k);
}
cout << result << endl;
```

This code computes and prints out an approximation to $e^x$:

$$\sum_{k=0}^{N} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \cdots + \frac{x^N}{N!}.$$

Sure, `exp(x)` is already part of `cmath`, but its just an example.
Although it contains syntax we haven’t seen, try to guess what this code does:

```cpp
for (int k = 0; k < N; k++){
    result += pow(x, k)/factorial(k);
}
cout << result << endl;
```

This code computes and prints out an approximation to $e^x$

$$result = \sum_{k=0}^{N} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \cdots + \frac{x^N}{N!}.$$ 

Sure, $\exp(x)$ is already part of `cmath`, but its just an example.
A quick note: We have seen using namespace std; before. The C++ standard library (you have it downloaded on your computer, if you installed Visual Studio with C++ capability) is absolutely huge and useful.

To see the content of the C++ standard library, go to https://en.cppreference.com/w/cpp/header. Notice that many of the “header” files shown start with “c—”. This indicates that they were originally libraries for the C language, and have been made compatible with C++.
1. Casting

2. Math Libraries

3. Strings

4. Next Time
When we want our variables to hold strings of letters rather than numeric values, we need to declare them as having type string.

- We must include the string header file in our program as `#include <string>`
- Declare a variable as a string, `string mssge = "Hello";
- Then we can use any and all of the operators defined on the string type

Strings are objects that represent sequences of characters. A string is a `class` that contains a `char` array, but automatically manages it for you.

Strings vs. Chars

There is a more primitive type of variable called a char, and then a more useful a container of such characters called a char array. Such containers of chars are annoying to manage due to memory allocation concerns, freeing up memory when finished with the array, and so on. The string class does all of this for you, making your life easier. Some useful functions explained through example:

```c++
#include <iostream>
#include <string>
using namespace std;

int main() {
    string x = "10";
    string y = "20";
    string z = x + y; // z will be "1020"
    cout << "The length z: " << z.length(); // 4
    cout << z[1] << " " << z[0]; // 0 1
}
```
Casting

Math Libraries

Strings

Next Time
Next class, we will start in on control flow and relational operators. Additionally I’ll say a few words about classes.

Questions?

Have a great weekend!