

CSE 1320 - Introduction to C++

Introduction

Alex Dillhoff

University of Texas at Arlington

History of C++

C++ was created by Bjarne Stroustrup.

It began as a superset of C which includes the concept of a *class*.

Classes provide a multitude of features for user-defined types, abstraction, and modularity.

History of C++

Since its inception, C++ has gone through several iterations.

Modern day C++ is almost unrecognizable from its original version.

In this class, we will cover the C++11 standard.

Transitioning from C

Any function written in C can be written in C++.

In fact, C code can be written directly in a C++ program.

However, part of programming in C++ means adhering to modern standards and practices.

The First Program

Even with the most basic program, there are obvious similarities and differences between C++ and C.

```
#include <iostream>
```

```
int main() {  
    std::cout << "Hello, CSE1320.\n";  
}
```

The First Program

The statement `std::cout << "...";` is probably the most obvious difference.

C++ uses the concept of *streams* for input and output.

The inclusion of `iostream` should be a sign that the C++ Standard Library is different.

Variables in C++

Variables in C++ are created almost identically to how they would be in C.

The major difference will be seen in what can be done with the variables and how user-defined types are established.

Types in C++

Any type that could be used in C is also in C++.

One of the first welcome additions to C++ is the basic data type `bool`.

This can take on `true` or `false` as a value.

Types in C++

Another useful addition to types is the `auto` type specifier.

Using `auto` as a variable type will infer the type of that variable from the value given to it.

Initialization

Initialization in C++ can be done similar to that of C.

However, one recommended practice is to use list initialization instead of using `=`.

Initialization

Consider the following statements.

```
int a = 4.3;
```

```
int b {4.3};
```

The first statement `int a = 4.3;` behaves just as it would in C.

The value is truncated to 4 and assigned to the variable a.

Initialization

The second statement `int b{4.3};` will actually throw an error due to floating-point to integer conversion.

However, C++ provides the programmer with the tools to handle errors and exceptions that can occur.

range-**for** loops

C++ supports the loops available in C. That is, **while** and **for**.

It also includes the range-**for** loop.

range-for loops

Consider the following code.

```
int arr[] = {0, 1, 2, 3, 4, 5, 6};  
  
for (auto val : arr) {  
    std::cout << val << std::endl;  
}
```

The range-for loop will iterate through all values in a collection, such as an array.

range-for loops

In the previous example, every value in `arr` will be copied into the variable `val`.

If, instead, we want `val` to simply refer to each value, we can use the following syntax.

```
for (auto& val : arr) {  
    std::cout << val << std::endl;  
}
```

Pointers

Pointers in C++ will also be familiar. However, the best practices for using them have changed greatly.

The first major addition is the inclusion of a proper way to determine if a pointer is null.

In C, this is done with the preprocess definition `NULL`. This definition is defined as the integer 0.

Pointers

In C++, the keyword `nullptr` can be used. This value accurately depicts a null value for a pointer instead of the integer 0.

The other major addition to pointers in C++ are **smart pointers**.

These provide protections against memory leaks. We will look at these more closely in a later lecture.

The vector

With the addition of classes comes a useful implementation: the vector class.

This is a general aggregate type that is used to represent a collection of any type, even user-defined types.

The vector

Consider the following statement.

```
std::vector<double> v({1.1, 2, 3.2});
```

This creates a collection of type `double` with 3 values.

The vector

One of the biggest conveniences of the `vector` class is that it manages its own memory.

It is possible to add, remove, and resize `vector` instances without explicitly working with memory allocation calls.

There are several other useful functions in this class that we will study later.

General Advice

- ▶ C++ is not "C with classes and more features."
- ▶ It is generally not optimal to write C code within C++.
- ▶ Avoid pointer arithmetic if necessary.
- ▶ Stick to C++ strings and vectors.

We will study many more cases in which the C++ standard is the optimal solution.