COMP 1633: Intro to CS II

Basics of C++

Charlotte Curtis

January 10, 2024

Where we left off

- Course outline, policies, etc
- Hello world
- Tracing without explanation
- Git + CLI adventures

```
int main() {
    int x = 0;
    int z = 0;
    while (x < 5) {
        z += x * x;
        ++x;
    }
    cout << z << '\n';
    return 0;
}
```

Today's topics

- C++ program basics
  - Format and layout
  - Compiling vs interpreting
- Simple output
- Variables and data types

> *Textbook Sections 1.3, 2.1-2.3*

A C++ Program

Every C++ program has exactly one "main" function

```
int main() {
    // execution begins here
    ...
    return 0; // ends here
}
```

> *Unlike Python, C++ will not run without a `main` function!*

Python vs C++

```python
def main() -> None:
    print("Hello World!")
```

```cpp
int main() {
    cout << "Hello World!\n";
    return 0;
}
```

Key points:

- Indentation is only for readability, with grouping indicated by braces `{}`
- Return type comes *before* function name (and is required)
- **Statements** are terminated with a semicolon `;`

Recall: statements vs expressions

Which of the following are true about **statements**? Select all that apply.

A. Can contain expressions.

B. Are instructions to the computer to do something.

C. Can be assigned to a variable.

D. Can include function calls

E. Can be nested.

Recall: statements vs expressions

Which of the following are true about **expressions**? Select all that apply.

A. Can contain expressions.

B. Are instructions to the computer to do something.

C. Can be assigned to a variable.

D. Can include function calls

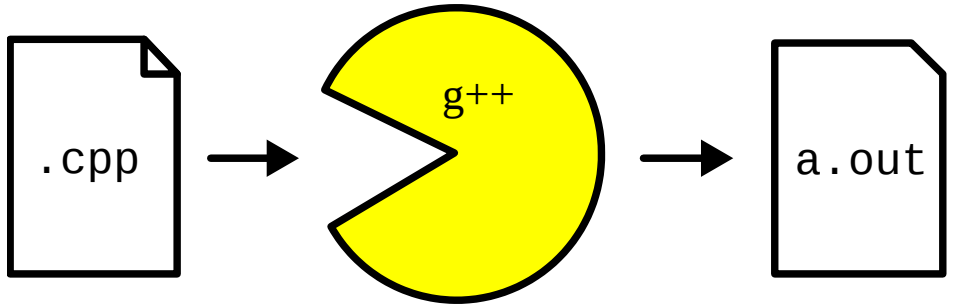
E. Can be nested.

Compiling vs Interpreting

- All computers only understand machine code, a sequence of 0s and 1s
- **Compiling** is when C++ is translated to a **binary executable**
- **Interpreting** is when Python is translated to machine code **line-by-line**

> - *Compiled code does not need the interpreter to run, and is often faster*
> - *Interpreted code can be easier to debug*

The g++ Compiler



- `g++` is the GNU C++ compiler
- We feed our source code `.cpp` file into the compiler, and it spits out an executable (by default named `a.out`)
- To change the output name, specify `-o` flag, for example:
  `g++ hello.cpp -o hello`

Recall: error types

Syntax, runtime, and logic errors

```python
def abs_val(some_var: int) -> int:
    if some_var < 0:
        soem_var = 0

    return some_var
```

```java
int abs_val(int some_var) {
    if (some_var < 0) {
        soem_var = 0;
    }

    return some_var;
}
```

> *The compiler is your friend! Compile-time errors are the easiest to fix.*

Simple Output

- While Python provides a lot of built-in functions, C++ is more modular
- To enable output, we need the following:
  - `#include <iostream>`
  - `using namespace std;`

Then we can use `cout` to print to the terminal

```
cout << "Hello World!\n";
```

> *Note that C++ does not include a newline by default*

Side Tangent: Preprocessor directives and namespaces

- Lines beginning with `#` are **preprocessor directives**
- The preprocessor is a program that runs **before** the compiler
- `#include` tells the compiler that the named file should be included

> *Preprocessor directives are not statements, and do not end with a semicolon!*

- `using namespace std;` *is* a C++ statement
- Optionally, you could omit this line and write `std::cout` every time

Variables and Types

At a high level, variables in C++ and Python are similar.

- Both allow you to refer to a value by a convenient name
- Both can only store one thing at a time, with a new value replacing the old

Python

```python
x = 5    # int
x = "Meep" # str
```

C++

```cpp
int x = 5;
x = "Meep"; // error!
```

- In C++ variables must be **declared** with a type
- The type **cannot** change!

Variable definition in Python

Internally, there's a whole Rube Goldberg-esque process happening when you write `x = 5` in Python. You end up with:

- namespaces
- objects
- references
- memory allocation

> *Python's ease of use means a lot is hidden from the programmer!*

If you're interested in diving down the CPython implementation rabbit hole, this is a pretty good start.

Variable declaration and initialization in C++

It looks similar to Python, but we need to be explicit about the type:

```
int x = 5;
x = "Hello world!"; // error, x can only be an int!
```

Declaration and initialization can also be separated

```
int x; // declaration
x = 5; // initialization
int x; // error! We already declared that x was an int
```

> *Beware the uninitialized variable!*

Separate or combined?

- From the compiler's perspective, there is **no difference** for primitive types
- My preference: declare and initialize on the same line
  - Less risk of uninitialized variables
  - Define variables when they are needed
  - C requires all declarations at the **top** of a function, so you may prefer this
- Multiple variables can be declared on one line, but this can be confusing

```
int x, y, z;
int a, b = 5, c; // Madness! Please don't do this.
```

- We'll revisit this idea when we discuss **reading input**

Variable naming

More or less the same restrictions as Python:

- Must start with a letter or underscore
- Can contain letters, numbers, and underscores
- Cannot be a reserved word (e.g. `int`, `double`, `return`)
- Case sensitive

> *Convention is either `snake_case` or `camelCase`, just be consistent*

Primitive Data Types

Python	C++	Size
`int`	`int`	4 bytes
`float`	`double`	8 bytes
`bool`	`bool`	1 byte
`None`	`void`, `NULL` (ish)	□
`str`	□	□
□	`char`	1 byte

- C++ has true primitive data types
- Declaring an `int` reserves just 4 bytes of memory
- `float` exists, but is rarely used
- `void` and `NULL` are special types that we'll discuss later

`str` vs `char`

- In Python, `str` is a sequence of characters
  - No difference between `'a'`, `"a"`, `'abc'`, or `'🐍'`
- In C++, `char` is a single character, or 1 byte **integer**
  - `'a'` is a `char`, but `"a"` means something entirely different
  - A **string literal** behaves the same way as in Python

```
cout << "Hello, world!\n";
```

  - ... but you can't declare and manipulate them the same way

```
cout << "Hello, " + "world!\n"; // error!
```

> *We'll talk about strings more depth later, for now we'll stick to string literals*

What do you think will happen?

This code **compiles and runs**. Predict the output from the choices below:

A. `97`

B. `aa`

C. `194`

D. `2a`

E. `2 * a`

```
int main() {
    char a = 'a';
    cout << 2 * a << '\n';
    return 0;
}
```

The assignment operator

Much like in Python, the `=` operator assigns the value of the **expression** on the right hand side to the **variable** on the left hand side:

```
int x = 5;
int y = x + 1;
x = x - 1;
y = 5 / 2;
```

> *What's going on with that `5 / 2`?*

Arithmetic operators

Python	C++	Description
`+`	`+`	Addition
`-`	`-`	Subtraction
`*`	`*`	Multiplication
`/`	`/`	Division
`//`	`/`	Integer division
`%`	`%`	Modulo (only for `int`)
`**`		Exponentiation

- Order of operations follows BEDMAS (just like Python)
- Operation depends on the data types of the **operands** and the **variable**

Arithmetic with mixed data types

- If both operands are `int`, the expression evaluates to `int`
- If at least one operand is `double`, the expression evaluates to `double`
- If a `double` is assigned to an `int` variable, it is **truncated** to an `int` (with an associated compiler warning)
- If an `int` is assigned to a `double`, it is **promoted** to a `double`

Practice!

For each of the following **expressions**, specify the data type of the result given `i` is an `int` and `d` is a `double`:

```
5 + i * 2
d + i * 2
d / 9.33
7 / i
7.0 / i
42 + 7 / (i * 1.2)
```

> *In the end, the result is cast to the variable type, but there may be intermediate loss of precisions as in* `double y = 5 / 2`

A few new operators

Like Python, C++ has the compound assignment operators `+=`, `-=`, `*=`, `/=`, and `%=`. There's a few new ones as well:

- `++` and `--`: increment and decrement by 1
  - Can be either `++x` or `x++`
- **Unary** operators: `+` and `-`
  - Finally you can write `x + -5` instead of `x - 5`!
- `++` and `--` happen first, then unary operators, then the usual BEDMAS

A full listing of operator precedence can be found here

Coming up next

- Lab: Hello world
- Lecture: Continuing with C++ basics

> *Textbook Sections 2.4-2.5*