Welcome Back

What you learned in CSCI 261 (or equivalent):
- Variables
- Types
- Arrays
- Expressions
- Conditionals
- Branches & Loops
- Functions
- Recursion
- Classes & Objects
- Streams
- Vectors
- Strings

You remember all of this, right? 😊

Hello, Let’s Review

Here’s a simple C++ program:

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

int main() {
    string hello = "Hello, world!";
    cout << hello << endl;
    return 0;
}
```

This tells the compiler that it should include symbols and types from the standard library <iostream>.

This is boilerplate that lets you use standard library symbols without extra ugly syntax. Don’t worry about it for now, just put it in whenever you #include something.

Variables

Declaration:
- int x;

Use in expressions:
- x + 10

Set via assignment operator:
- x = 4;

Declare and initialize:
- int x = 42;

Types

- Basic types
  - Integer types:
    - int: 42
    - char: ‘k’
  - Floating point types:
    - double: 3.14159, 4.5e3
  - Boolean type:
    - bool: true, false

- Pointers
- Arrays
- Class/struct types
Expressions & Operators

- Anything with a value is an expression:
  - Variables
    - x
  - Indexed array variables
    - arr[10]
  - Literals
    - 42
    - "Hello"
    - true
- Arithmetic/logical expressions using operators:
  - \(4 + 7 / 3.0\)
  - \((x * 2 + 1) \% y\)
  - count == 0
  - a | b & !c
- Mixed type expressions allowed due to type conversions

Loops

What if we want to print "Hello, world!" three times?

```cpp
for (int i = 1; i <= 3; i++) {
    cout << i << " Hello, world!" << endl;
}
```

Output:
1 Hello, world!
2 Hello, world!
3 Hello, world!

Another Loop

```cpp
int i = 3;
while (i > 0) {
    cout << i << " Hello, world!" << endl;
    i--;
}
```

Output:
3 Hello, world!
2 Hello, world!
1 Hello, world!

Conditionals

```cpp
if (true=false-expression) {
    true-block
} else {
    false-block
}
```

Arrays

```cpp
int numbers[3];
numbers[1] = 14;
numbers[2] = -3;
numbers[3] = 7093; // Oops! What's wrong here?
```

Hello, if?

Let's modify Hello to respond to an input:

```cpp
char answer;
cout << "Say (H)eello or (G)ooodbye?" << endl;
cin >> answer;
if (answer == 'H') {
    cout << "Hello, world!" << endl;
} else {
    cout << "Goodbye, world!" << endl;
}
```

What happens if the user enters "h" instead of "H"?
Arrays

```cpp
int numbers[3];
numbers[0] = 14;
numbers[1] = -3;
numbers[2] = 7093;
```

Let's print out the numbers in the array. What about in reverse order?

Loops on Arrays

```cpp
int numbers[] = {14, -3, 7093};

for (int i = 0; i < 3; i++) {
    cout << numbers[i] << endl;
}

for (int i = 2; i >= 0; i--) {
    cout << numbers[i] << endl;
}
```

Functions

We've seen one function:

```cpp
int main() { ...
```

Here's another:

```cpp
int print_it(string msg) {
    cout << msg << endl;
    return msg.length();
}
```

Hello Functions!

A silly program.

```cpp
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

int print_it(string msg) {
    cout << msg << endl;
    return msg.length();
}

int main() {
    int n;
    double nroot;
    n = print_it("Hello, world!");
    nroot = sqrt(n);
    cout << "The square root of the number of characters printed is ";
    cout << nroot << endl;
    return 0;
}
```

Recursion

Functions can call themselves.

```cpp
void print_n_times(string s, int n) {
    if (n == 0) return;
    cout << s << endl;
    print_n_times(s, n - 1);
}
```

Objects

C++ is an object-oriented (OO) language.

What's an object?

A definition:

An object is a package of data with associated behavior.

More specifically, we say that an object has properties (fields, attributes, data, state), and that it has associated methods (functions).
Classes

• Objects also have type. Objects of the same type:
  – Have a common set of properties and methods
  – Used in a similar manner to primitive types.

• Types are (usually) modeled by classes. Classes formally define the properties and methods.

• Essentially, defining classes is a way to add new types to C++.
  (Classes do some other neat things, too, but we’ll get to that later.)

Classes in C++

Classes are created via a class declaration:

```cpp
class student {  
  public:  
    string name;  
    string year;  
    double gpa;  
    bool is_hungry;  
    student();  
    void eat();  
    void sleep();  
    void program(int assignment);  
  };  
```

- Don’t forget this semi-colon!

Defining Member Functions

The declaration only gave the member function signatures (prototypes); we still have to write the functions themselves:

```cpp
void student::eat() {
  is_hungry = false;
}
void student::program(int assignment) {
  if (grade(this, assignment) == 'A') gpa++;
}
```

Etc.

Using Objects in C++

• Objects can be created just like chars, ints, etc.:
  ```cpp
  student s;
  ```

• Properties are referenced by the “.” operator:
  ```cpp
  s.name = "April";
  s.gpa = 4.0;
  double d = s.gpa;
  ```

• Methods are invoked on objects also using “.”:
  ```cpp
  s.sleep();
  ```

Some Notes on Visibility

• Many philosophies around visibility
  – “All data should be private”
  – Partly a matter of style

• Rule of thumb:
  – If it is specific to the implementation, it is private
  – Else, it is public

• Not all OO languages have visibility modifiers. (But they all have commenting systems!)

Streams

• Console I/O:
  ```cpp
  #include <iostream>
  cin >> some_var;
  cout << expression << endl;
  getline(cin, s); // must #include <string>
  ```

• File I/O:
  ```cpp
  #include <fstream>
  ifstream fin("words.txt");
  fin >> some_var;
  getline(fin, s);
  ofstream fout("output.txt");
  fout << expression << endl;
  ```

• We’ll also learn about stringstream objects (later).
Arrays and Vectors

Arrays:
```c
int foo[10];
for (int j = 0; j < 10; j++)
    foo[j] = j;
```

Vectors:
```c
#include <vector>

vector<int> foo(10);
for (int j = 0; j < 10; j++)
    foo[j] = j; // == foo.at(j) = j
```

Do More with Vectors

E.g. you can append to a vector – it automatically resizes:
```c
vector<int> foo;
for (int j = 0; j < 10; j++)
    foo.push_back(j);
```

foo contains:
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Gives an initial size to the Vector.

Declares that this vector will hold int values.

Initially size 0 without optional argument.

Something New-ish

C++ 11 added a new type of for loop:
```c
vector<int> numbers = {14, -3, 7093};
for (int x: numbers) {
    cout << x << endl;
}
```

This denotes that x is a variable of type int which will take on each value in numbers in turn.

Note vector initializer list – can be used almost like a literal in certain contexts.

About Strings

In C/C++, the literal "Hello" is called a string.
It is of type char[] (a char array).

Confusingly, C++ defines a new type, string.
A string is mostly interchangeable with a string (which in C++ is called a "C-string").

But, you can do more with string objects:
```c
#include <string>

string foo = "Hello";
string bar = "World";
string hello = foo + " ", " + bar + "!
if (foo == bar) { … }
```

More About Strings

Know/learn the string interface!
- See Help page of course website for C++ documentation websites
- Some string methods you should know:
  ```c
  length       operator[]
  size         operator+
  find         operator+=
  substr       relational operators
  ```

Up Next

- Please continue to review chapters 1 – 10 in your textbook
- Friday, August 25
  - Lab 1: Compiling and building code
  - Assignment: APT 1