C++ Objects

Java and C++ are very similar

- **Similar in:**
  - Syntax: Java used syntax similar to C++ to ease adoption
  - Principles: Both are object-oriented languages
  - Execution: Many similarities when run on a machine
    - Compiled down to similar assembly language

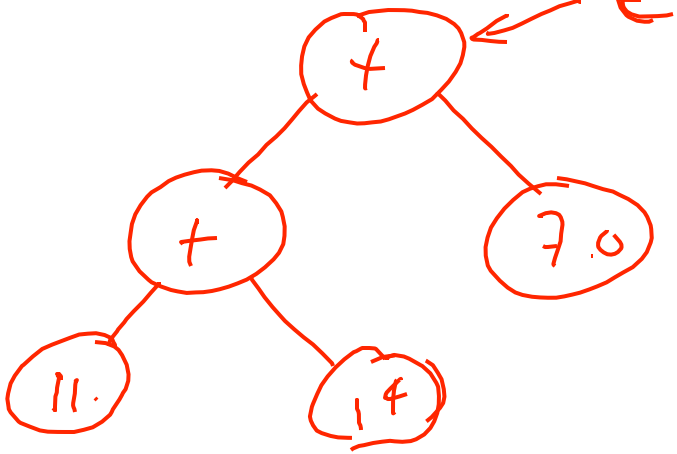
- **Different in goals:**
  - Java designed for: safety and portability
  - C++ designed for: performance and control

As a result, C++ exposes aspects of execution that Java hides

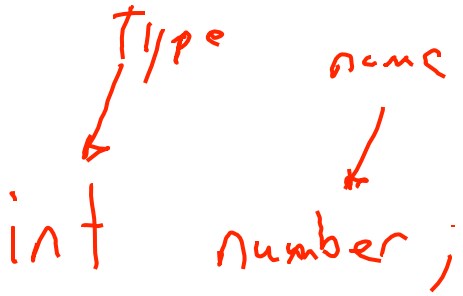
Object declaration

- **.h**
  - Declare any member variables
    - Google style guide specifies names should end in underscore (_)
  - Declare any functions

- **.cpp**
  - Specify the implementation of those methods



Allocating objects



- **Two ways (unlike Java)**
  - On the stack
    - E.g., ExpressionVariable expressionVariable;
  - On the heap
    - E.g., new ExpressionVariable();

C++ exposes info about your program!

- **sizeof()**
  - Tells you in bytes how big a type or variable is

- **& (address of operator)**
  - Tells you the memory location of a given variable

C++ language spec vs. implementation

3.4.1 **implementation-defined behavior** unspecified behavior where each implementation documents how the choice is made

- EXAMPLE An example of implementation-defined behavior is the propagation of the high-order bit when a signed integer is shifted right.

3.4.3 **undefined behavior** behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements

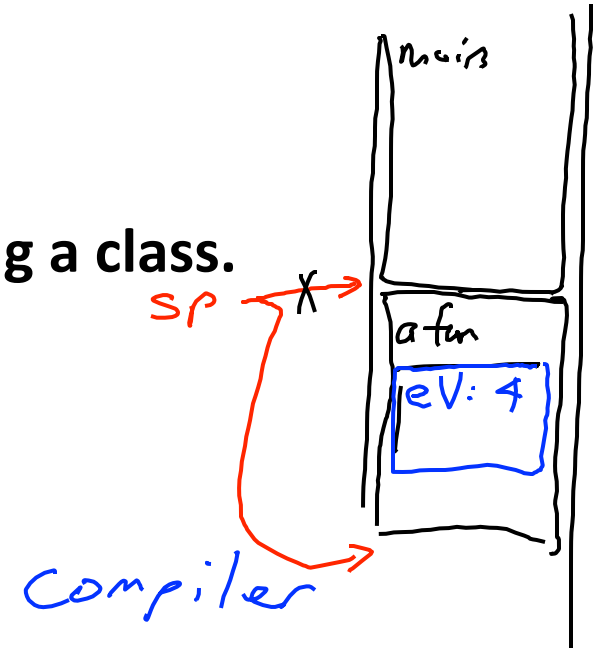
- NOTE Possible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message).
- EXAMPLE An example of undefined behavior is the behavior on integer overflow.

3.4.4 **unspecified behavior** use of an unspecified value, or other behavior where this International Standard provides two or more possibilities and imposes no further requirements on which is chosen in any instance

- EXAMPLE An example of unspecified behavior is the order in which the arguments to a function are evaluated. `x=0; f(x++, x++);` ← `f(0,1)` -or- `f(1,0)`

Constructor

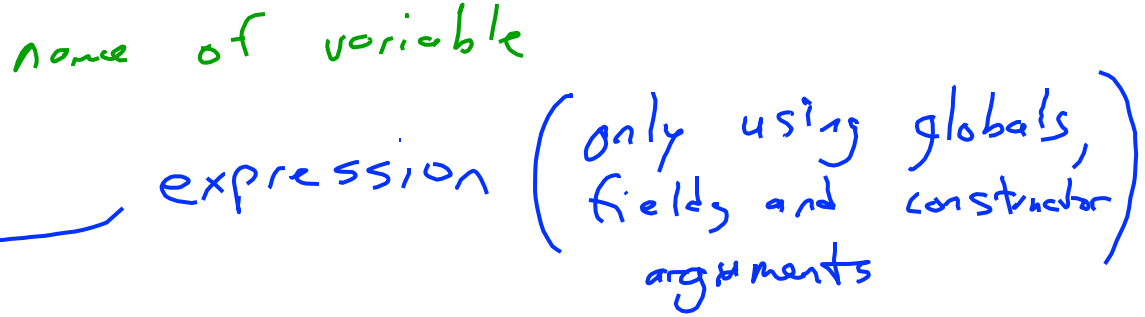
- **Always the first function run when creating a class.**
  - Run after memory is allocated



- **Automatic Default Constructor**
  - *Automatic:* created by the constructor
  - *Default:* takes no arguments
  - Calls default constructors of object fields, **but not primitives**
    - Values of primitives are **undefined**
  - Only generated when no constructors are defined

Constructor, cont.

- **Initialization list**
  - name(value)
  - Comma-separated
  - Run before body of constructor




```
cs126::ExprValue::ExprValue(double value) : value(value) {
  // body
}
```

- **Again, values are only initialized if you initialize them…**
  - So initialize everything!

Constructors can call other constructors

- Helps avoid redundant code
- Invoke the constructor as the first item of the initialization list

```
cs126::ExprValue::ExprValue() : ExprValue(1.0) {

}
```

Heap allocation

- **Use the "new" operation: allocates memory, calls constructor**

new ExprValue(7.0);

- **Returns a pointer to the object** address
  - Pointers are declared with a * before the variable name
    - * is associated with the name
  - E.g., ExprValue *exprValue = new ExprValue(7.0);
  - C++ treats pointers as primitives (doesn't init them)

- **You are responsible for deallocating heap memory**
  - delete operation
    - E.g., delete exprValue;
  - Good practice to null out pointer afterwards