C++ Basics

Programming

- Computer
  - Execute sequence of simple (primitive) instructions
  - What instructions should be provided?
    - Is there a minimum set? (See Turing Machine)
  - Generic
    - Reduce future limitations
- Program
  - Describe process in the form of a sequence of instructions
    - Think recipe
- Programming Language
  - Express sequences of instructions
  - Translate to computer instructions

"Hello, World!"

```
#include <iostream>                  // pre-processor directive
using namespace std;

int main()                           // start of program
{
   cout << "Hello, World!\n"; // standard output stream
   return 0;                        // return value to operating system
}
```

Structure of a C++ Program

Computers are good at following instructions, but not at reading your mind.

- Donald Knuth

- https://isocpp.org/std/the-standard
- Grammar
  - Rules that define the language
  - Describes what is valid and what is not
    - E.g., `return:` is not valid
  - Ambiguity
- Statement
  - Smallest standalone unit that expresses an action
  - Many statements end in a semicolon
    - E.g., `return 0;`

Structure of a C++ Program

- Block (compound statement)
  - Treated as a single statement
  - Begin with { and end with } (curly braces)
  - No semicolon needed after ending curly brace
  - Can be used where a simple statement is permitted

Structure of a C++ Program

- Function
  - Section of a program performing a specific task
  - Every function body is defined inside a block
  - Body
    - Statements executed in sequence
- For a C++ *executable*, exactly one function named `main()`

Structure of a C++ Program

- Library
  - Typically pre-compiled code available to the programmer to perform common tasks
  - Two parts
    - Interface
      - header file, which contains names and declarations of items available for use
    - Implementation
      - pre-compiled definitions, or implementation code. In a separate file, location known to compiler
- Use the `#include` directive to use a library in your program (satisfies declare-before-use rule)

Namespaces

File1.cpp

```
void hello()
{
  cout << “hello\n”;
}
```

File2.cpp

```
void hello()
{
  cout << “Hello\n”;
}
```

Comments

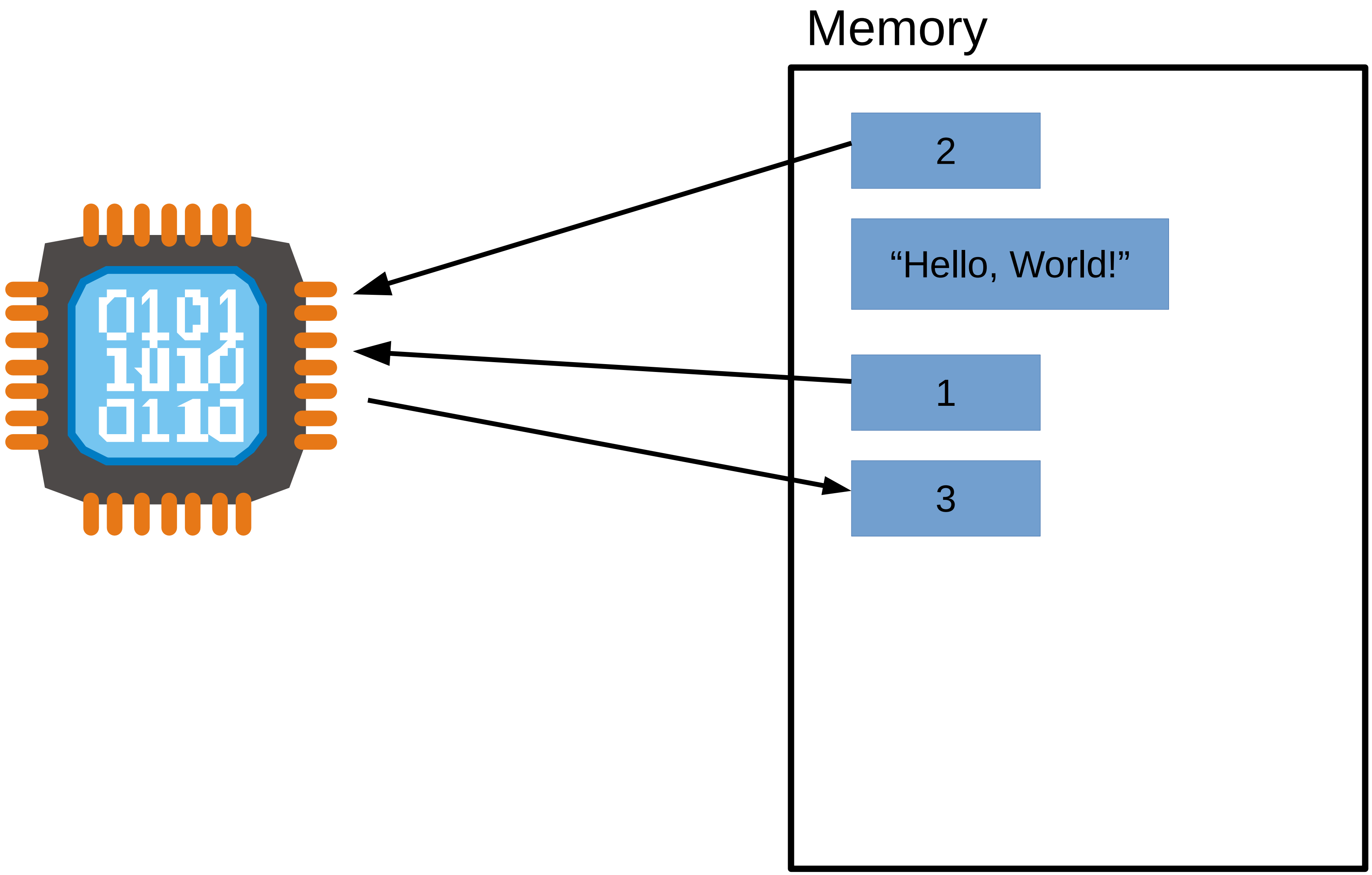
- Annotate code
  - Add information useful to humans but not to compiler
  - Comments are ignored by the compiler
  - Examples:
    - Author
    - Citation – origin of code
    - Explain code
- Block style (like C)

```
/* This is a comment.
       It can span multiple lines  */
```

- Line comments -- use the double-slash `//`

```
int x;    // This is a comment
x = 3;    // This is a comment
```

More C++ Primitives


Memory
2
“Hello, World!”
1
3

Variables

- Stores data
- Type of data (e.g., string, number)
- Name
- Declare Before Use
  - Variables must be declared before they can be used in other statements
- Examples:

```
int page_number;

string title;
```

Variables

- Declare and Define

```
int x;
```

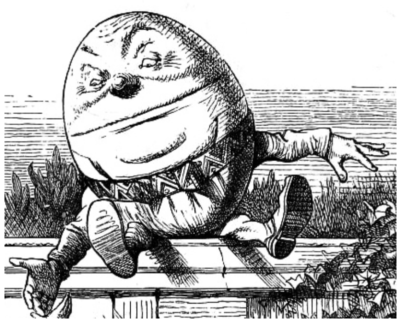
- Assign value

```
x=5;
```

- Arithmetic operations

```
x=x+5;
```

Identifiers



`When I use a word,' Humpty Dumpty said, in rather a scornful tone, `it means just what I choose it to mean -- neither more nor less.'

— Lewis Carroll, Through the Looking Glass

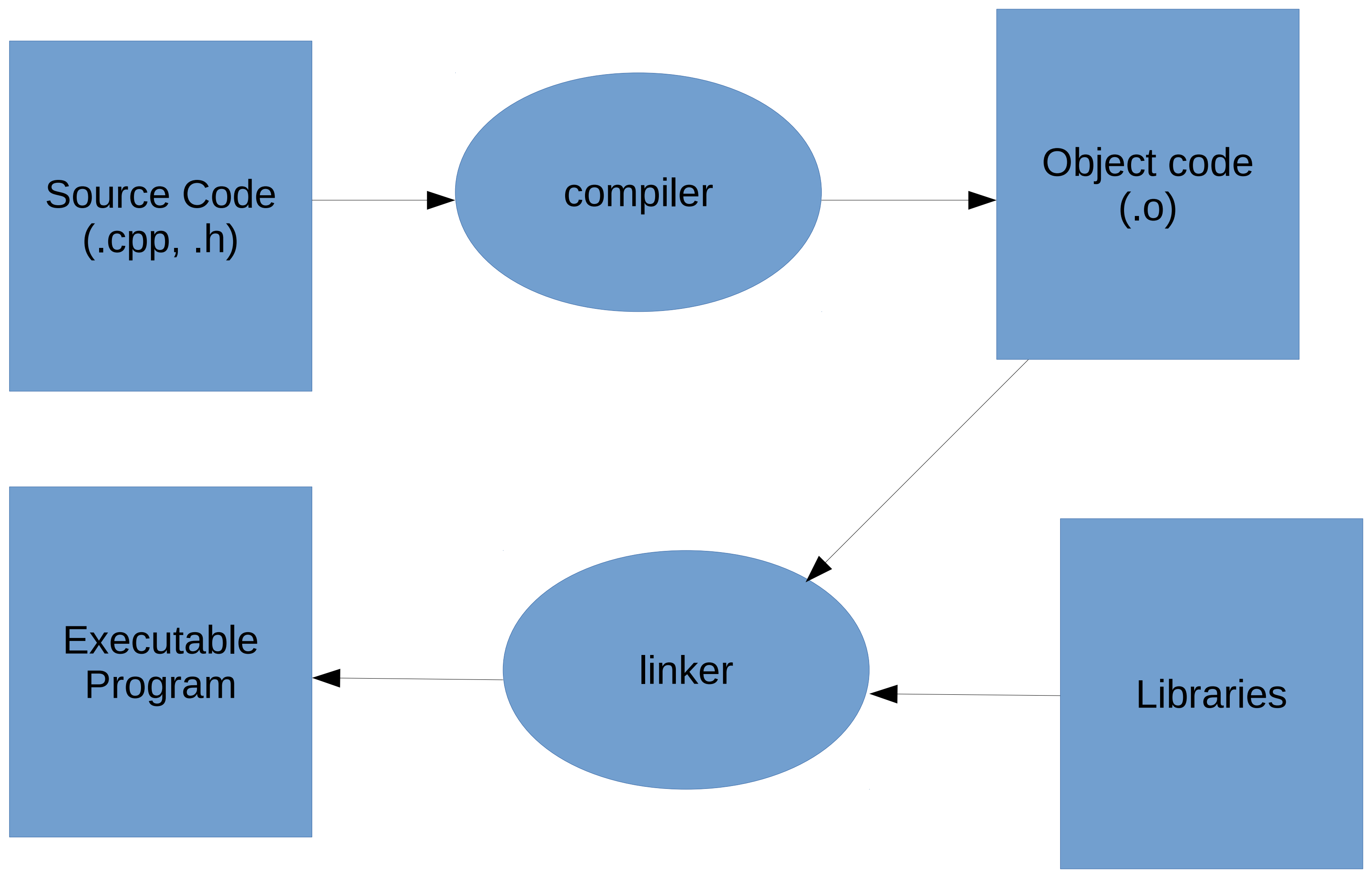
- Need a way to refer to variables, functions, etc.
- Choose names that are descriptive
- Can use multiple words
    - E.g., FirstName, last_name
  - Function that performs an action – use predicate-like
    - E.g., ComputeGrade(...), display_text(...)
- Be consistent

Programming Strategies

- How do I go about writing a program?
- Top-down programming
  - Start with description and divide it into sufficiently small units corresponding to available components
- Bottom-up programming
  - Start with small components and build from them

In-Class Example

Building and Running a C++ Program



Building and Running a C++ Program

- Pre-processing
  - The #include directive is an example of a pre-processor directive (anything starting with #).
  - #include <iostream> tells the preprocessor to copy the standard I/O stream library header file into the program
- Compiling
  - Syntax checking, translation of source code into object code (i.e. machine language). Not yet an executable program
- Linking
  - Puts together any object code files that make up a program, as well as attaching pre-compiled library implementation code (like the standard I/O library implementation, in this example)
  - End result is a final target -- like an executable program
- Run it!