- A program consists of instructions
    - Instruction types
        - Input: program gets data
        - Process: program performs computations on data
        - Output: program puts that data somewhere
    - Variables refer to data
- First C program:
    - Starts in main(), executes statements within main's braces {} one at a time.
    - Each statement is usually 1 per line and ends with semicolon ;
    - When creating variables must create them first then assign them values
        - Int wage;
        - Wage = 20
    - Return 0 statement ends the program (the 0 tells the OS the program ended without error)
    - **Scanf() statement gets input value** from keyboard and puts that value into wage variable. **Dont forget the &**
        - Scanf("%d", &wage);
    - Outtputting an integer variables value is achieved via printf("%d", x) the %d in the string indicates that a **decimal number** should be printed there.
    - If you want to make a printf statement with multiple variables just use the %d operator as placeholder then seperate variable names from string respectively. Even if the same variable is used twice.
        - Ex.
            - printf("In %d, the driving age is %d.\n", drivingYear, drivingAge);
            - printf("%d states have exceptions.", numStates);
        - Outputs
            - `In 2014, the driving age is 18.`
`10 states have exceptions.`
        - Ex. Doing mathematical operations with multiple variables in a printf statement.
            - printf("%d\n", rectangleLength * rectangleWidth);

- **<u>Comments and Whitespace</u>**
    - Single-line comment starts with //
    - Multi-line starts with /* and ends with */
        - Error ex.

```
/*
   numKids = 2;  /* Typical number */
   numCars = 5;
*/
```

- Compiler will generating an error when reaching the second /* that is ending a comment.

  - Whitespace
    - Use blank line to separate conceptually distinct statements
    - Indent lines the same amount
    - Align items to reduce visual clutter
    - Uses single spaces between operators

- **Error Messages**
  - Sometimes the error message refers to a line that is after that actual error line. If there are no problems at the line indicated look at previous lines.
  - Focus on the first error message, if an error exists on that line or before it, correct it and compile again.
- **Logic errors**
  - A program may compile completely fine but not work properly. This means there was no syntax errors (compile-time errors) but there is a **logic error/bug.** Ex. NumJars + numBeans instead of NumJars * NumBeans
- **Compiler Warnings**
  - Compilers can produce warnings that do not stop the compiler from creating the program but indicate a possible logic error. Good practice is to write programs the compile without warnings and also configure your compiler to produce even more warnings.
- **Bits**- either 1 or 0
  - Can be used in combination for complex calculations. Processors created to execute a list of desired calculations (instructions). Instructions specified by configuring specified switches.
- **Memory**
  - Instructions are stored in memory, circuits that can store 0 and 1s in thousands of different locations.

- **Writing Computer Programs**
  - In the 40s programmers wrote each instruction in binary (0 and 1s). These were called **machine instructions,** a sequence of machine instruction formed an **executable program.**

    - After this programs were created called **assembly language** that automatically translated human readable instructions into machine instruction. Ex. “Mul 97, #9, 98” Multiply the value stored in memory 97 by the number and 9 and store that value in memory 98.
    - In the 60-70s high-level languages were create to support programming using formulas or algorithms. Ex. F = (9 / 5) * C + 32 some languages were FORTRAN formula translator and ALGOL or algorithmic language.
    - Compilers were created to automatically translate high-level language in executable programs.
- **Common IDEs for C**
    - Visual Studio Code, XCode, CLion
- **Console**
    - Also called a terminal. Text-based interface that allows a user to run programs, enter input, and view the programs output. A consoles **command line interface (CLI)** is an interface that allows user to enter commands. Can be used to run a program with command line arguments enter by the user. Ex. Ls –a (ls is the name of the program and –a is a CLI.
- ***1.7 COMPUTER TOUR***
    - Input/output devices: screen, keyboard, mouse etc. Also called peripherals
    - Storage: SSD uses flash memory to store files and data like program files, documents. SSDs are non-volatile., meaning they maintain their contents even when powered off. Stores 0 and 1s by tunneling electrons into circuits then extracting them with a flash of electricity.
    - Memory: RAM temporarily holds data read from storage and is designed so that any address can be accessed much faster than an SSD. RAM is able to access any memory location quickly going in arbitrary order rather than having to look in a specified location.
    - Processor: runs computers programs. Reads and executes instructions from memory and performs operations like reading and writning data. When powered on teh processor starts executing programs at memory 0 (usually the BIOS). Then begins executing the OS allowing user to run other programs. Processor may contain a small amount of RAM on its own chip, called cache memory, accessible in one clock tick.
    - Clock: instructions execute at rate governed by processors clock. Tick rates can vary from 1 MHz (1 M ticks a second) to 1 GHz (1 B ticks a second).
    - Switches evolved into transistors. Which can be integrated into single chip called an integrated circuit.
- **1.8 FIRST LANGUAGES**

    - In 1978 C was created. Became the dominant language of the 80s and 90s.
    - In 1985 C++ created, added constructs to support object-oriented programming.
- **CHAPTER 1 LABS**
    - All lab programs must end with a newline unless otherwise stated.