1. Getting Started
   1. **Programming**: creates software that controls hardware.
      1. Uses a language
         1. **C**: “Latin” of programming.
      2. **Programming Cycle**
         1. Edit
         2. Compile and link or “build”
         3. Test run
   2. **Source code**: the “code” you write.
      1. Plain text file
      2. Contains programming instructions
      3. Ends with .c
      4. Can use any text editor
   3. **Object code**: source code after being compiled by a **compiler**.
      1. Contains tokenized instructions
      2. Ends with .o
   4. **Program file**: end result of the object code being combined by a **linker** with C library.
      1. **C library file**: “workhorse” of the program which contain routines to control the program.
      2. Can test run the program.
   5. **Integrated Development Environment (IDE)**: combines tool for editing, compiling/linking, and other special tools.
2. Programming Basics
   1. ***main()* function**: where the program execution starts; required for any program.
      1. “Dummy” if by itself
   2. Parts of the C Language
      1. **Keywords**: 44 “language” parts that accomplish basic tasks.
         1. In practice, may only use ~22
      2. **Function**: thousands of “workhorses”.
         1. Held in C language
            1. Linker takes Object code with C library to create a program.
         2. Defined by **I/O** **header** **files**.
            1. Included in the source code by including the preprocessor directive.
            2. #<stdio.h>

Includes *puts()*

* + - 1. Types
         1. **Integer function**: returns a value to the operating system.
         2. **Output function**: returns a string to the operating system.
    1. **Operators**: symbols used to manipulate data in the program.
       1. Mathematical
       2. Comparison
       3. Assignment
       4. Logical
       5. Bitwise
       6. Unary
    2. **Data type**: variables and values.
       1. **Values**
          1. **Characters**: letter or symbol.

**Array**: category of variables.

**String**: array of characters enclosed by quotation marks.

**Escape characters**: added to a string to represent a character which would normally affect it.

**\n**: newline

**\t**: tab

**\’**

**\”**

**\\**

Must be declared by an **immediate assignment**.

**Array declaration**

[] dictates that there are multiple char variables.

Compiler calculates the **buffer** (storage) space based on the string.

Otherwise, the number is within the [] as [n+1], to include space for the null.

* + - * 1. **Numeric values**

**Integer values**: whole numbers and negative numbers.

**Floating-point values**: contain decimal point or fraction; very small or very large.

**Memory locations**

* + - 1. **Variables**: placeholders/containers for a value which may vary.
         1. Requirements

Be declared of a specific type, matching

Be given a unique name

Cannot be the name of a function

Used within the code

* + - * 1. Types

**int**: integer values.

Mathematical operations can be done.

**float**: floating-point numbers.

Must add .0 even if a whole number.

**char**: single characters.

Single quotes used around characters.

Mathematical operations can be done on alphabetical characters.

**double**: type of variable with twice the precision of float.

**long**:

**short**:

**signed**:

**unsigned**:

* + - * 1. Format:

variabletype variablename

variable = value

* + 1. **Structure**: manner/order in which the language is used.
       1. **Preprocessor directives**: control program flow.
       2. ***main()* function**: first function required to be executed in all C programs.
       3. **Curly brackets**: used to enclose a function’s contents.
       4. **Statements**: “sentences” in C which include key words, functions, math, logical comparisons, etc.
       5. **Comments**: items that part of the source code but not compiled.
          1. Marked by /\* \*/ for an interval, or // for a line.

1. The Old I/O
   1. **Input/Output (I/O)**: taking input and generating output.
      1. Typical C programs take input, generate output.
         1. If not, then it is simply plumbing.
   2. **C Library Output Functions**: send one or more characters to standard output device.
      1. ***puts()****:* put string.
         1. Defined in stdio.h
      2. ***printf()***: print function.
         1. Defined in stdio.h
         2. Does not add a new line at the end.
            1. Must be added by \n escape character.
         3. **Placeholders**: represent a value inside the print function; defined after the string with a comma in between the string and placeholder.
            1. **%d**: integer values.
            2. **%s**: strings.
            3. **%f**: floating-point values.
            4. **%c**: single characters.
            5. **%%**: percent sign.
   3. **C Library Input Functions**
      1. ***scanf()***: fetches a string or any specific value from standard input and assigns it to a variable.
         1. Requires stdio.h
         2. Uses *printf()* placeholders
         3. Only works if strings are one word long.
         4. Format
            1. scanf(“*format*”,&variable);

*format*: formatted string.

Almost always composed of placeholder.

Determines what type of input requested.

&: **memory location operator** used for **variables**, NOT strings/arrays.

variable: variable name.

* + 1. ***fgets()***: file input function, which can also be used to read standard input.
       1. Format
          1. fgets(variable,#,stdin);

variable: variable into which input is stored.

Also called a **buffer** (storage).

#: number which indicates the size of the buffer.

stdin: indicates standard input.

* 1. **Character I/O Functions**
     1. Work with integer values only.
        1. Use %c placeholder to display the character, not its code value.
     2. Require stdio.h header
     3. **Stream-oriented**: characters are sent to output as they are typed.
        1. Computer waits to send the characters until the **buffer** (storage) is flushed (i.e., the program is finished).
     4. Types
        1. ***getchar()***: fetches character from standard input.
        2. ***putchar()***: sends character to the standard output.

1. **C Language Math Operator**s
   1. Format
      1. Normally
         1. Calculation occurs on the right
         2. Result assigned on the left.
      2. Or, result can be used immediately.
   2. Types
      1. **+**: addition
      2. **-**: subtraction
      3. **\***: multiplication
      4. **/**: division
         1. Must be done with floating point values to obtain the right amounts.
      5. **++**: increment; adds one to a variable.
      6. **--**: decrement; decreases one to a variable.
   3. **Order of precedence**: shows the order in which operators are carried out.
      1. PEMDAS
      2. Left 🡪 Right
      3. Order can be changed with parentheses.
2. **C Language Math Library Functions**
   1. Generally require math.h header file
   2. Work with real numbers and floating point values; use double variables if more precision is required.
      1. **double**: type of variable with twice the precision of float.
   3. Types
      1. ***sqrt()***: square root
      2. ***pow()***: exponential function
         1. First number is base
         2. Second number is exponent
      3. Random
         1. ***rand()***: generates pseudo-random values.
            1. Requires stdlib.h header file.
            2. Does not require arguments in its ()s.
            3. Pseudo-random: outputs the same random number.

Values can be predicted

Must use a **seed randomizer**: machine that helps generate random numbers.

* + - 1. ***srand()***: seeds *rand()* function.
         1. Requires a positive integer argument.
         2. Requires stdlib.h
         3. Must be reseeded to output different numbers.

Asking users to input a positive integer value.

Borrow a **tick-tock** value from the computer’s internal time clock, which is constantly changing.

srand((unsigned)time(NULL));

Brings out numbers that are close.

1. **C Language Comparisons**: evaluate multiple values or expressions.
   1. Types
      1. ==: is equal to
      2. <: less than
      3. >: greater than
      4. <=: less than or equal to
      5. >=: greater than or equal to
      6. !=: “not”; does not equal
2. **Flow Control Words**: alter the flow of a program.
   1. ***if()***: conditional
      1. Format: one statement split between two lines enclosed by two curly brackets.
         1. First line: sets condition in parenthesis.
         2. Second line: executed if the first line is true.
         3. **;**: statements by themselves; go after the statements, **NOT** the parentheses that hold the condition of *if*.
      2. If condition is unmet…
         1. The statement is skipped unless
         2. An *else* keyword is used.
   2. ***else***: executed if the condition is unmet.
      1. Same format as *if*.
   3. ***else if***: secondary conditional, before *else*.