* + Slide Set #1: Basics
* Parts of a computer
  + **Central Processing Unit (CPU)**: executes instructions
  + **Main Memory (RAM)**: running programs and their data are stored in main memory.
    - **Volatile**: wiped off when computer is turned off.
  + **Graphics Processor (GPU)**: additional processor charged with processing graphics.
  + **External Memory**: persistent, long-term memory.
    - **Non-volatile**: data persists after shutdown.
  + **Registers**: fast, hardware memory locations in which to place instructions about to be executed and relevant data; volatile.
  + **Arithmetic Unit (ALU)**: additional processor for heavy “number crunching” computations.
  + **Operating System (OS)**: performs basic functions related to operating computer, file manipulation, queuing instructions, etc.
  + **Compiler**: translates high-level source code to machine language.
  + **Application program**: programs intended by end-user on a computer.
* Programming Cycle
  + **Edit**: compose C statements that make up the program.
  + **Compile** the C statements.
  + **Link** the program to other libraries as appropriate.
  + **Execute** the “object code” program.
  + **Object code**: instructions in machine language.
  + **Obtain** results.
  + **Examine** results for “mistakes”.
* Basic C Programming Structure
  + **Comments**: parts of the code written purely to describe things to other humans and not for the compiler to translate them.
  + **C Preprocessor**: special commands that indicate things to be done to the program prior to compilation.
    - **Header files**: allow programmer to break up program across several files.
    - **Macros**: can replace code.
  + Global data declarations: #define thing (value)
  + Function prototypes
  + **main()** function: begins and ends the program.
  + Definition of other functions
* Slide Set #2: Variables, Operators
  + **Compilation errors**: mistakes that the compiler catches.
    - Missing the “”s; not closing {}s or ()s properly; forgetting ; at the end of each statement, or & before the variable (not string) when calling scan(f); using %f instead of %lf for doubles.
  + **Execution errors**: program compiles and runs initially, but terminates abnormally; ÷0 or infinite loops
  + **Logic errors**: happen normally when the program compiles and terminates normally, but produces incorrect output.
  + Binary-decimal:
  + Decimal-binary: divide by 2 and use carry as the binary value
  + Variables
    - **Legal names/variable identifiers**: certain rules one must follow when choosing names for variables (identifiers).
      * Only case-sensitive letters, numbers, and \_
      * First character must either be a letter or \_
      * Must not be one of **37 keywords** reserved by C.
    - **Declaration**: statement that tells compiler that the programmer wants to have a variable of a particular type, with a name.
      * Requests an **index** (map) of its address to the name of the variable.
    - **Scope**: extent of validity of a variable with regards to parts of the program.
      * **Local variables**: only recognized within the block of code (i.e., inside a function) in which they are defined.
      * **Global variables**: valid everywhere in a single-file program**.**
    - **Casting**: temporarily changing a variable type.
    - Characters are mapped onto characters through the ASCII; each takes up 1 byte.
    - Operators
      * Floating point division: a/b
      * Integer division: a/b; a%b
      * **Boolean Operators**: check if a Boolean value is true or false.

AND(&&): disjunction; true if and only if all arguments are true.

OR (||): true if at least one argument is true.

NOT (!): inverts the truth value of its only argument.

* + - * Operation precedence: left 🡪 right, PEMDAS
      * Pre/post increments: ++a increments a by 1 before used in expression; a++ increments a by 1 after.
        + a(operation)=b 🡪 a = a(operation)b
  + **Complex data structures**: permit more sophisticated ways to store and manipulate data.
    - **Arrays**: collection of similar variables.
    - **Strings**: sequence of characters.
    - **Structures**: collection of dissimilar variables.
    - **Unions**: collection of dissimilar variables sharing the same addresses.
  + Computer memory
    - **Bits**: binary elements represented by 0s and 1s.
    - **Byte**: 8 consecutive bits; cach has an **address** (way to locate it within that long string).
      * **Addressable memory**: pinpoint a particular location in the long string and store/retrieve values in them.
    - **Word**: 16 consecutive bits.
* Slide #3: Control Structures

|  |  |  |  |
| --- | --- | --- | --- |
| **Conditional Structures** | | | |
| **Single-selection (if statements)**  if (condition)  {  } | **Double selection (if-else statements)**  if (condition)  {  }  else  {  } | **Multiple selection (if-else-if-else statements)**  if (condition)  {  }  else if (condition)  {  }  else  {  } | **Switch case**: similar but different structure to multiple selection.  switch (controlling expression that returns a value)  {  case label\_1:  {  break;  }  default:  {  }  } |
| **Repetition Structures** | | | |
| **Counter-controlled**: repetition structure for a fixed number of times. | | **Sentinel-controlled**: loops which continue running so long as a condition is true, after which the next line is read. | |
| for: carries out the loop continuation test immediately after the counter is incremented at the top of the loop.  for (<init>; <condition>; <increment>)  {  } | | while: entry condition; specifies action to be repeated so long as a condition remains true.  control\_var = <initial val>;  while (loop continuation test on control\_var)  {  <statement1>;  <update control\_var somehow>;  } | |
|  | | do/while: exit condition; same as while, except that loop continuation test is done after the body of the loop has been executed.  int cont\_var = <initial val>;  do  {  <statements>  <update control variable>  } while (continuing condition); | |

* + **break**: causes an immediate exit to the first statement after the loop.
  + **continue**
    - In for: causes increment expression to be executed and the loop continuation test is evaluated thereafter.
    - In while: loop continuation test is evaluated immediately after continue.
    - In do/while: loop continuation test is evaluated immediately after continue.
  + **Strings**: arrays of characters; char str [128] (128 is max)
    - **String literals**: string enclosed in “”.; **Character literals**: character enclosed in ‘’.
    - Includes **null terminator**: special character which indicates end of a string.
* Slide Set #4: Programmer-defined functions
  + **Define**: what they are to do.
    - *Type of value returned*
    - *Type and name of arguments* (inputs)
    - *Declaration* *of automatic* (local) *variables*
    - *Defining the body of the function*
    - *Return a value of correct type*
    - **Header**: first line of function definition. <return type> <function name>(<parameter list of paired inputs passed to function>)
    - **Body**: series of C statements, which describe the instructions to be executed on the variables defined.
  + **Call**: ask them to do what they do.
    - **Prototype**: optional features that describe the format of a function (name, inputs, outputs).
      * **Name** of function
      * **Pre-conditions**: arguments expected by function.
      * **Post-conditions**: what the function will do when it terminates.
    - **Call-by-value**: passing value of a variable as an argument to the function when called.
    - **Call-by-reference**: passing the **address** of the variable being referenced.
    - **Nesting**: functions being called from within others.
    - **Runtime stack**: list of instructions followed by processor.
    - **Formal parameters**: listed in the function definition, acting as local variables within the function.
    - **Actual parameters**: arguments; actual values passed in to the function when it is called.