Monday, July 17, 2017

C++ Primer Plus

Functions: C++'s Programming Modules

- In libraries
  - C++ classes and ANSI C library

FUNCTION REVIEW

- Function definition and function prototype
- Return
  - Return statement is optional for void functions
  - Return value can't be an array
    - Everything else is possible - pointers, structures and objects
  - Function returns a value by copying the return value to CPU register (or memory location)
    - Then calling program examines a location
    - Returning and calling function have to agree on the type of data at that location
    - Function prototype tells the calling program what to expect
    - Function definition tells the program what to return
- Why prototypes?
  - Prototype describes the function interface to the compiler
    - Describes return value, arguments
  - Return values in CPU register
  - Without this information compiler can only guess
    - It can't catch errors
    - It can't look further in the file and see how those functions are defined
      - Not efficient, it would have to stop compiling the file and search rest of the file
        - Function doesn't even have to be in one file with name

        - Files can be compiled independently -> main() doesn't have to have access to the function
        - Fiction can be in the library
    - You can avoid using a prototype by placing function definition before it is used
- Prototype syntax
    - Function prototype is a statement so it must have a terminating semicolon
    - You don't need to provide names for the variables
        - List of types is enough
        - It acts like a placeholder
- C++ versus ANSI C prototyping
    - ANSI C borrowed prototyping from C++
    - ANSI C
        - Prototyping optional because of compatibility with classic C
    - C++
        - Makes prototyping mandatory
- What prototypes do for you
    - Helps compiler and you to reduce program errors
        - Compiler correctly handles the function return value
        - Compiler checks that you use correct number of function arguments
        - Compiler checks correct type of arguments (or convert arguments to the correct type if possible)
    - If cube(int a) is supposed to have an argument and you write just cube()
        - It uses whatever value happens to be there
    - If you provide function a wrong type
        - Weird errors
        - If function expects int and you pass double -> function pass just 16 bits

- Prototyping results in type conversion only when it makes sense
  - It won't convert structure to int
- Prototyping takes place during compile time
  - = static type checking

FUNCTION ARGUMENTS AND PASSING BY VALUE

- C++ normally passes by value
  - Numeric value of the argument is passed to the function
- Formal argument
  - = formal parameter
  - = parameter
  - Variable that is used to receive passed value
- Actual argument
  - = actual parameter
  - = argument
  - The value passed to the function
- Variables declared within a function are private to a function
  - Function is called -> allocation of memory, creating variable
  - Function terminated -> freeing of memory, destroying variable
  - Local variables
  - Preserves data integrity
  - Automatic variables
    - Allocated and relocated automatically during program execution
- Multiple arguments
  - You can also use it in for loop

FUNCTIONS AND ARRAYS

- Passing array

- Arr is not an array, it is a pointer
  - But you can write rest of the function as if it was an array

- How pointers enable array processing functions
  - C++ and C in most contexts treats the name of an array as a pointer
    - Array name is an address of its first element
    - 2 exceptions to this rule
      - Array declaration uses array name to label the storage
      - Applying size of to an array name
        - Size of whole array in bytes

- Function header
  - Int *arr and int arr[] have the same meaning
    - Only when used in function header or prototype
    - Because

- The implications of using arrays as arguments
  - When array is passed to function, its pointer is actually passed
    - information
      - It tells the function where the array is (address)
      - what kind of elements it has (type)
      - How many elements it has (n variable)
    - The function uses the original array
      - If you pass ordinary variable, function uses copy
      - It doesn't violate pass by value approach
        - Function still passes a value that is assigned to a new variable
          - But the value is single address, not contents of an array
    - Is it good to pass pointer instead of the actual array?

    - Saves the time and memory that is needed to copy an entire array
    - But it the possibility of inadvertent data corruption
        - That is problem in C
        - In C++ is const modifier that provides remedy
- Cookies and arr evaluate to the same address
- Sizeof cookies is 16 and arr is 4
    - Because cookies is a size of whole array and arr is size of a pointer to that array
    - That is the reason why you have to explicitly pass the size of the array, rather than sizeof
- More array function examples
    - Filling the array
- Showing the array and protecting it with const
    - Guarantee that the display function doesn't alter the original array
    - If you try to modify the array
- Functions using array ranges
    - You can pass pointer and size
    - You can pass two pointers
        - One identifying start and second end
    - Count number of elements
    - Functions calls

POINTERS AND CONST

- You can use the const keyword two different ways with pointers
    - Make pointer to point to a constant object
        - That prevents you from using the pointer to change the pointed-to value
    - Make pointer itself constant
        - That prevents you from changing where the pointer points

- Assigning the address of regular variable to a pointer-to-const
  - Pointer points to a const int (39), *pt is constant
  - But it doesn't mean that value to which it points is a constant
    - You can change value age directly
    - But you can't change value indirectly using pt
- Assigning the address of a const variable to a pointer-to-const
  - Valid
- Assigning the address of a const to a regular pointer
  - Invalid
  - You can use a type cast to override the restriction
- Pointers to pointers
  - Assigning a non-const pointer to a const pointer is okay
    - If you are dealing with just one level of direction
      - Pointer assignments that mix const and non-const
        - Not so safe
- Arrays and pointers
  - The prohibition against assigning the address of a constant array means that you can't pass the array name as an argument to a function by using a non-constant formal argument
  - Function attempts to assign a const pointer (months) to a non-const pointer (arr)
- Using const when you can
  - Why to declare pointer arguments as pointers to constant data
    - Protects you against programming errors that inadvertently alter data
    - Using const allows a function to process both const and non-const arguments
      - Function that omits const in the prototype can accept only non-const data
- Example

  - Const only prevents you from changing the value to which pt points (39)
  - It doesn't prevent you from changing the value of pt itself
- How to make it impossible to change the value of the pointer itself
  - Finger points only to sloth
    - But it allows you to use ps to change the value of sloth
  - The middle declaration doesn't allow you to use ps to change value of sloth
    - But it permits you to have ps point to another location
- You can declare a const pointer to a const object
  - Stick can point only to trouble and stick cannot be used to change the value of trouble

FUNCTIONS AND TWO-DIMENSIONAL ARRAYS

- Data is the name of an array with 3 elements (arrays of 4)
  - The type of data is pointer-to-array-of-four-int
  - Its prototype
    - Parentheses are needed because
      - This would declare an array of four pointers-to-int
        - Function parameter can't be an array
  - Prototype can also be declared
    - Function only works with arrays with 4 columns
    - Number of roes is specified by variable size
    - Possible function definition
- Array notation
  - Work out the meaning

FUNCTIONS AND C-STYLE STRINGS

- Functions with C-Style string arguments
  - 3 choices of representing a string

    - Array of char
    - Quoted string constant
        - = string literal
    - Pointer-to-char set to the address of a string
  - All are pointer-to-char type
    - So you can use all 3 as arguments to string-processing functions
  - You are passing the address of the first character in the string
  - Difference between C-Style string and regular array
    - String has a build in terminating character (null)
    - So you don't have to pass the size of the string as an argument
- Functions that return C-Style strings
  - Function can't return a string
  - Returns address of a string - more efficient
  - To create string of n visible characters, you need storage for n+1

FUNCTIONS AND STRUCTURES

- Structures behave like normal variable
- Pass by value
- C didn't allow passing by value for structures
- C++ provides third alternative
  - Passing by reference
- Passing and returning structures
  - You can get two inputs
    - If you type something else than numbers, loop will terminate
    - If you want to continue you need to cin.clear()
- Passing structure addresses
  - Pass its address - &pplace

- Declare the formal parameter to be pointer-to-polar
  - Function shouldn't modify the structure
- Because formal parameter is pointer, use indirect membership operator (->)
- To use it more efficiently, you should return pointer instead of structure
  - You need to pass 2 pointer to the function
    - First points to the structure to be converted, const
    - Second points to the structure that is to hold the conversion, not const
  - Instead of returning a new structure, function modifies an existing structure in the calling function

RECURSION

- C++ functions can call itself
- Recursion with a single recursive call
  - Break chain of recursive calls with if statement
  - Each recursive call creates its own set of variables
- Recursion with multiple recursive calls
  - Divide and conquer strategy
  - Number of calls grows geometrically

POINTERS TO FUNCTIONS

- Function address
  - Memory address at which the stored machine language code for the function begins
- Normally it isn't important nor useful that user knows the address
  - It can be useful to a program
  - You can write function that takes the address of another function as an argument
    - That enables the first function to find the second function and run it
- Function pointer basics

- Suppose you want to design an estimate function that estimate the amount of time necessary to write a given number of lines of code
  - Part of code will be same for all, part will be different for each programmer (algorithm)
  - So you pass estimate the address of the particular algorithm
    - To do so you need
      - Obtain the address of function
        - Use function name without parentheses
        - Think()… function, Think… address
      - Declare a pointer to a function
        - Declaration has to specify to what type of function it points
          - Identify the return type and signature (argument list) - same info as prototype
        - *pf is function, pf is a pointer to function
        - Parentheses are required
        - After declaration, you can assign it the address of matching function
        - Compiler rejects non matching assignments
        - Declaration of the function
        - Pass the pointer to function to the second function
      - Use a pointer to a function to invoke the function
        - C++ allows you to use pf as if it were a function name
          - how can pf and (*pf) be equivalent?
            - History reasons