C++ Review: List of Concepts Covered (since midterm 1)

Other Function Topics

- Function overloading - functions with same name and different parameter lists
- Default parameters (new C++ feature, not available in C)
  - optional parameters, by giving them default values
  - Must be last in the parameter list
- Understand how default parameters on functions affects function overloading
- Random number generation (using library functions)

Pass by Reference

- Reference variables
  - use & notation when declaring
  - an alias for another variable (i.e. nickname)
  - useful when variables are in different scopes (i.e. functions)
  - used in parameter passing and return values
- Pass by Value - local copies of parameters are made, and copies of returns are sent back
- Pass by Reference - Copies of parameters and returns are not made. Local parameters are references to the originals
- Use of `const` with reference parameters to prevent a function from changing the original (but avoid overhead of making a copy - faster execution)

Misc -- `cctype` library

- A useful C library of character handling functions
- `toupper()`, `tolower()`
- Understand the boolean functions whose names start with `is`, for determining if a character fits in a certain given category

---

Arrays

Array Properties

- indexed collection of data elements of same type
- consecutive storage locations
- default indexing is 0 through `size-1` (where `size` is the number of elements in the array)

Declaring Arrays

- format: `typeName variableName[size];`
  - Example: `int list[10]`
- The type can be any basic type or any user-defined type
- the size must be known by the compiler, so it must be a positive integer literal or constant.
- 2-dimensional arrays
  - Example: `double table[5][10]`

Initializing Arrays

- Can initialize arrays in the same line as declaration
- Format: `type name[size] = { list of elements };`
  - Example: `int list[5] = {1, 3, 5, 9, 10};`
- The list of elements goes in { } and is separated by commas
- may leave size box empty when **initializing** on the declaration line - compiler sets size.
- Can also initialize with for loops (good with regular patterns)
- **Special case**
  - strings: null-terminated character arrays
  - can initialize on the declaration with a string literal
    - Example: `char name[7] = "Marvin";`
  - size must leave room for null-character '\0'

Using Arrays

- valid indices are 0 through `size-1`.
- may use any of these index numbers to access a single array element:
- may use any positive integer r-value to index arrays (i.e. variables, expressions, etc)
- it is the programmer's job to check for out-of-bounds index!
- Copying Arrays
  - Assignment between array names does not copy one array to another
  - If you want to copy one array to another, do it element by element (easy with a loop)

Using c-strings

- A c-string can be used like a normal array (of characters)
- cout and cin objects also work with c-strings (for output and input of words)
- >> operator for input stops at white space (space, tab, newline, etc.)
  - only good for one word at a time
- `get` and `getline` for reading strings from input
  - `get`, `getline` read up to specified delimiter -- can read entire sentences

Arrays as function parameters

- Know how to pass an array into a function
- Usually a good idea to pass in a size as well
- Function always has access to the array contents -- only the address is sent in
  - There's no pass-by-value vs. pass-by-reference with arrays

- Use `const` on the array parameter when the function shouldn't change the array

Array Usage and Algorithms

- Understand how to handle arrays that are declared to a certain size, but are not always "full" to their capacity
- Understand common array algorithms and patterns, including (but not limited to):
  - iterating through an array elements with a loop
  - Printing array contents
  - adding or counting array elements
  - finding largest/smallest element of an array
  - initializing array contents, with either formulas, user entry, or initializer list
  - Using parallel arrays
  - Swapping or moving around array elements

<cstring> library functions:

- `strlen (string length)`
- `strcpy (string copy)`
- `strcmp (string compare)`
- `strcat (string concatenation)`
- `strncat, strncpy, strncmp`

`string` objects

- Built with the `string` class library
- variable length, flexible
- Supports more intuitive operator notations, like assignment, comparisons, + for concatenation, etc
- Understand the difference between c-strings and string objects
- Know the commonly used operators, as well as the usage of basic member functions discussed in class

---

Pointers

Basics

- A pointer is a variable that stores an address
- declaration format: `typeName * variableName;`
- target -- the item that a pointer points to
- dereferncing the pointer (to get to the target)
  - if p is a pointer, *p is the target (dereference the pointer with *)

Initializing pointers

If p is a pointer, then how can we fill in the blank?

`p = _____`

Four ways:

1. NULL pointer
    - pointer that stores address 0. Has no valid target.
    - 0 is the only literal number that can be assigned to a pointer
2. Another pointer of the **same type**
    - a pointer is thought of as a "pointer to a" specific type
    - different pointer types can NOT be assigned to each other (automatic type conversions like on the basic types do not apply)
    - An array name counts as a pointer (to whatever type the array is built from)
    - A string **literal** (e.g. `"Hello"`) is an r-value that counts as a `(const char * )`
3. The "address of" an existing variable
    - Using & on a variable means "address of" that variable.
        - Note this is NOT the same as using & in a declaration (with a type in front of it). That's a reference variable -- used in Pass By Reference
    - Example: `&x` means "address of x"
    - May assign an address to a pointer (of matching type)
        - `p = &x;`
4. a `new` operation (Dynamic Allocation -- See below)

Pointer Arithmetic:

- can subtract two pointers
- can add or subtract integers to or from pointers
    - does not use literal integer. adds or subtracts that many units of pointer **type**
    - i.e. add x to a pointer-to-int actually adds (x * (size of an int))

Pointers and Arrays:

- the name of an array is a pointer to the first element of the array
- array access through a pointer can be done with bracket operator p[3]
- array access through pointer can be done with pointer arithmetic *(p+3)
- May assign array name to a pointer of matching type:

```
int list[10];
int * ptr;
ptr = list;
```

`ptr` could now be used to access array elements, like the array name

- Name of statically declared array cannot be assigned another value (it's like a pointer that is a constant -- it's bound to the array)

Pass by Address:

- third type of parameter passing -- pass in the pointer, or address

- Parameter type is pointer type (e.g. int *)
- passes in copy of address, function can use address to find original data
- very useful for passing arrays in and out of functions (by their name)
- Can use to accomplish same as pass by reference. Reference parameters easier when using single variables, though
- For arrays, the following two notations are equivalent in meaning:

```
void Function1(int * arr );
void Function1(int arr[] );
```

- **Const:**
  - declaring a parameter as const on a pass by reference or address
  - does not make copies of the data, but prevents changing original data
  - `const int * p;` -- pointer p CAN be changed. Target cannot be changed.
  - different combinations with "const" can make pointer constant, too.

Dynamic Memory Allocation

Memory Allocation Categories

- **Static** -- compile time. size and types known in advance
- **Dynamic** -- run time. sizes and amounts can be set while program running

Dynamic Allocation, Deallocation

- create dynamic space with the operator `new`.
  - always use a TYPE after the word `new`.
- The `new` operator also returns the address of the allocated space
  - use a pointer to store this address
- can dynamically allocate basic variables and arrays
- deallocate dynamically allocated memory with operator `delete`
  - apply `delete` to the pointer, and it de-allocates the target.
  - Use `delete []` for arrays

Dynamically resizing an array (application example):

1. dynamically create a new array of the needed size (need another pointer for this)
2. copy the data from the old array to the new one (use a for-loop)
3. delete the old dynamic array (keyword delete)
4. change the pointer so that the new array has the original name

---

Process Management

- Multitasking and multiprocessing
- Changing Commands with the chmod command
- Processes
    - run in their own memory and address space
    - multiple processes scheduled to run on same processor (by the OS)
- Creation of processes in Unix - just know generally
    - A parent process creates a new process with the *fork* command
    - New process a clone of the parent at first
    - Child process makes an *exec* system call to start up a new program on the process
    - ALL processes in Unix created this way, starting with one initialization process when a system is booted
- Job control commands can be used to view, manipulate, and halt processes in a Unix system
    - ps, jobs, kill, fg, Ctrl-C, Ctrl-Z
    - Use `&` to launch a process as a background job

Input/Output redirection

- Understand what is meant by:
    - standard input
    - standard output
    - standard error
- Know how to use these symbols in unix commands:
    - `<`
    - `>`
    - `>>`
    - `|`
- Specifically, know how to:
    - Re-direct standard input into a command to come f$
    - Re-direct the standard output from a command to g$ of the screen)
    - Re-direct the standard output from a command to b$ file
    - "Pipe" the output from one command to become the $ another command

Shell Commands Covered

Command	Description	Flag options covered	You should be able to

`grep`	Search for patterns within files	`-i`
		`-n`
		`-l`
		`-P`
			ignore case
			line numbers
			display filenames only

`ps`	report on current processes	`-e`
		`-f`
		`-a`
		`aux`
			view all processes on the system (-e)
			view the "full format" (-f)
			all processes except for session leaders (-a)

		`-l`	view the "long format" (-l)
`jobs`	Shows background processes		view the current background processes you are running
`fg`	put a job into foreground		put one of your background processes into the foreground
`bg`	run a job in the background		put one of your current jobs to run in the background
`Ctrl``-Z`	suspend the current foreground process		suspend a process without killing it
`Ctrl``-C`	kill foreground job		kill, or cancel, the current foreground process
`kill`	kill a running process, through its process ID (PID) number	`-KILL` or `-9`	send a KILL signal to override and halt processes that can't be killed with normal kill command
`sleep`	a delay for a specified time		cause current process to sleep, or delay, for specified number of seconds