CS 161 Lab #6 – Pointers and Memory Allocation

- Get checked off for (up to) 3 points of work from the previous lab in the first 10 minutes.
- **To get credit for this lab, you must be checked off by a TA by the end of lab.**
- **This lab should be done solo, not via pair programming.**

Goals:

- Refresh what references are and how/when to use them
- Practice using pointers
- Practice dynamic memory allocation and understanding the computer memory model

---

(1 pt) A. Passing Multiple Function Arguments by Reference

Create a file called `lab6_ref.cpp` with a `main()` function (and a file header with your name, the lab, and the date).

1. Above `main()`, define a function that takes in a string and counts its punctuation marks:

```
void count_punctuation(string s, int& n_periods, int& n_commas);
```

Notice that this function is void. In passing values by reference, you can obtain multiple results from the computation.

Useful string functions:

- `s.length()` : the number of characters in string `s`
- `s[i]` : the character at position (index) `i` in string `s`

Inside `main()`, read in a string from the user and use your function to count its punctuation marks. You will need to declare the integer variables before passing them to your function. in `main()`, print out both counts you obtain for the user's string.

2. Identify at least 3 test cases, write them down (with expected output), and test each one.

3. Try this experiment: What happens if you add an ampersand (&) in front of the `n_periods` argument when you call the function? Why? (If you aren't sure, discuss with another student or TA.)

(1 pt) B. Passing Multiple Function Pointer Arguments

1. Write a new function that does the same thing, this time using pointers:

```
void count_punctuation2(string s, int* n_periods, int* n_commas);
```

Add a call to this function in `main()` and print out the result to ensure it works just like your first function. Run it with the same test cases you created.

2. What needs to be changed when you make the function call? What needs to change inside the function?

3. What is the difference (in computer memory) between an ampersand (&) and an asterisk (*) added in front of a parameter in the function definition? Draw a diagram to explain.

4. Since this function can be written with either references or pointers, what are the advantages/disadvantages to each approach? Which one (at this time) makes more intuitive sense to you?

5. Run both functions using PythonTutor: http://pythontutor.com/visualize.html
- Select “C++” from dropdown menu
- Paste your program into the box. PythonTutor does not support user input, so modify your code to initialize the string to a specific value you want to test, instead of using cin.
- Click “Visualize Execution”
- Click “next” to move forward and examine what we see in memory (“stack”).
- You can click on a particular line of code to jump to that point in the visualization.

(4 pts) C. Hangman

To help you practice using pointers in functions, you will write a short program that allows two users to play Hangman. The program prompts player 1 to enter a line of text (while player 2 is looking away), then prints `\n` or **`endl`** 50 times to clear the screen. The program then outputs a "hidden" version of the string in each letter has been replaced with an asterisk. Player 2 is prompted to guess a letter. Any matches for that letter are revealed, and play continues until player 2 guesses the full string. (For a full Hangman implementation, some more functionality would be needed, like limiting the total number of guesses and checking whether the player already guessed the letter. You do not need to add these capabilities.)

Design a program to create a Hangman game and include these exact functions:

-
  ```
  /* Read a line of text from the user into string s */
  void get_input(string* s);
  ```

-
  ```
  /* Make a new copy of string s and replace all letters
   * with asterisks; put the result in s_hidden */
  void hide_letters(const string& s, string* s_hidden);
  ```

-
  ```
  /* Given a letter guess, update s_hidden to reveal any
   * occurrences of that letter in s; return the number of matches */
  int guess_letter(char letter, const string& s, string* s_hidden);
  ```

Example:

```
Enter a secret string using only lower-case letters:  one small step for
man.
<50 blank lines to clear the screen>
Your string is: *** ***** **** *** ***.
Please guess a letter :  s
2 matches!
Your string is: *** s**** s*** *** ***.
Please guess a letter:  a
```

```
2 matches!
Your string is: *** s*a** s*** *** *a*.
Please guess a letter:  z
0 matches!
Your string is: *** s*a** s*** *** *a*.
[more guesses happen until there are no more asterisks]
Congratulations!  You win the game!
```

(2 pts) Design first. You can have more functions, but you must have at least the three functions listed above with the EXACT function prototypes. Write the function headers/descriptions for each of the functions, as well as any additional functions you create. This includes information about parameters, return values, and pre/post conditions.

Design – Give as much detail as possible for the main function and all the functions above.

- Why do the function prototypes have the specific parameter types on them?
- Why do the functions have void or a return value?
- How do all the functions interact together?
- How will the program determine that the game is over?

Testing – Provide testing cases with expected results.

- What do you plan to use as bad values? Good values?
- What do you expect to happen with bad values? Good values?

Get checked off by a TA before beginning to implement. This will help you avoid logic or function mistakes.

(2 pts) Implementation. Write your functions, main() method in a file called `lab6_hangman.cpp` . When finished, ask another student to play your game with you.

(4 pts) Dynamic Memory Allocation

1. Create a new file (lab6_mem.cpp).

Here is one way to write a function to dynamically allocate an integer:

```
int* create_int() {
  int* ptr = new int;
  return ptr;
}
```

Using what you've learned about pointers, implement 2 additional functions that allocate a new integer from the heap:

```
void create_int(int*& ptr);    /* pass the pointer by reference */
void create_int(int** ptr);    /* pass a pointer to the pointer */
```

Hint: You will need to assign the address to a pointer that was created outside the function. Call each function in your main().

2. Run **`valgrind`** to see what a memory leak looks like (allocated memory that is not freed):

```
$ valgrind program_exe
```

3. What if you want to change the contents of what the pointer points to? **Write a function that will set the contents of your newly created integer**. What should you pass to this function? A value, a reference, a pointer, or an address of a pointer?

4. Update your program to clean up the heap when you are done using the newly allocated integer, so that you do not have any memory leaks. How will you delete the memory off the heap? Try doing it outside a function, then inside a function. **Make sure your delete function sets the pointer back to NULL**, since it is not supposed to be pointing anywhere anymore. Use **`valgrind`** as you write to inspect the state of memory.

Ask a TA to check your work (get points) and submit your programs to TEACH

1. Transfer your .cpp file from the ENGR servers to your local laptop.
2. Connect to TEACH here: https://teach.engr.oregonstate.edu/teach.php
3. In the menu on the right side, go to **Class Tools -> Submit Assignment.**
4. Select **CS 161 020 Lab_6** from the list of assignments and click “SUBMIT NOW”.
5. Select your .cpp files (`lab6_ref.cpp, lab6_hangman.cpp, lab6_mem.cpp`).
6. Click the **Submit** button.
7. You are done!

Point totals: 2 pts (reference/pointers) + 4 pts (Hangman) + 4 pts (dynamic memory)

If you finish the lab early, this is a chance to work on your Assignment 4 design (with TAs nearby to answer questions!).
