

Programming Assignment 2 and 3 (20 points)

This programming assignment has two parts:

- **Preliminary Submission (PA #2)** (due Friday, April 23, beginning of discussion)
- **Final Submission (PA #3)** (due Friday, April 30, beginning of discussion)

This is an individual programming assignment. See Handout 1 (Course Information) for our policy on collaboration, plagiarism, and late polices. **Plagiarism will result in severe penalties.** Please mark the top of the programming assignment with your name, CS201, and the date. Points may be deducted if your TA has problems understanding your solution or if your solution is not legible.

This assignment is worth 20 points.

In this assignment, you will implement the popular game of Mastermind using the C programming language. Mastermind is a code-breaking game on a board (see photo below) that contains:

- A 4-peg hidden code (bottom of photo),
- Code pegs (from a variety of six colors for use in the four center columns), and
- Key pegs (white and black, for use in the smaller rightmost columns).



We do not consider the smaller leftmost columns in this assignment.

The game is played by two players: a codemaker and a codebreaker. The codemaker sets up a hidden code, which contains 4 arbitrary pegs drawn from 6 colors (repetition is allowed). Then the codebreaker makes repeated guesses (4 pegs) of the hidden code until the guessed code matches the hidden code or the number of guesses reaches the limit.

At each round, the codebreaker makes a guess (putting 4 pegs into the 4 center holes in a row). After each guess, the codemaker provides feedback regarding that guess by placing key pegs into the smaller holes at the right hand side of that row. A black key peg indicates that one guessed peg matches one hidden peg at the same position and in the same color. A white key peg indicates that one guessed peg matches one hidden peg in color but **not** in the same position. The key pegs have no order information (the positions of key pegs do not matter).

In this assignment, you will simulate the game using a program that interacts with the user (the codebreaker) on the console. The computer acts as the codemaker. The rules are as follows:

- There are 6 colored pegs in the game: R(Red), G(Green), B(Blue), Y(Yellow), P(Purple), O(Orange); 2 colored key pegs: black and white.
- The codemaker inputs a 4-peg hidden code when the game starts.
- At each round, the codebreaker attempts to guess the hidden code by typing values that represent 4 pegs (e.g., “RGBY” with no spaces allowed). The input is on a single line and followed by an “Enter” key.
- Your program should respond appropriately to invalid inputs. Two cases will be examined by the TA:
 - Invalid number of input pegs (other than 4).
 - Invalid input pegs that are not in the specified 6 colors.

For either case, your program should output an error message with error type (e.g., “Invalid Input. The number of input pegs should be 4!”) and ask the user to make another input (e.g., “Please try again: ”).

- After each guess, the computer should give feedback by comparing the guessed code with the hidden code. The feedback is then displayed on the screen. For example: “2 black pegs, 1 white pegs”.
- When the guessed code matches the hidden code (4 black key pegs represent the correct guess), the program should output “Code-breaking succeeded after # guesses” (# represents the actual number of guesses taken) and exit.
- When the number of guesses reaches the maximum (**use 12 in your program**) and the guess is not correct, the program should output: “Code-breaking failed” and exit.

The Sample Input and Output is as follows:

```
*****
elnux4>mastermind YGPP

                Welcome to the game of Mastermind!
Color Pegs: R=Red  G=Green  B=Blue  Y=Yellow  P=Purple  O=Orange
Guess Round 1: RRGG
    0 black pegs
    1 white pegs
Guess Round 2: OOB
Invalid input! The number of input pegs should be 4!
Please try again: OOB
Invalid input! Please use the specified color pegs!
Please try again: OOB
    0 black pegs
    0 white pegs
Guess Round 3: YYPP
    3 black pegs
    0 white pegs
Guess Round 4: YYPR
    2 black pegs
    0 white pegs
Guess Round 5: GYPP
    2 black pegs
    2 white pegs
Guess Round 6: YGPP
    4 black pegs
    0 white pegs
Code-breaking succeeded after 6 guesses!
*****
```

We are providing skeleton code online (the beginnings of function declarations).

Your Final Submission will be graded based on:

1. Elegant comments within the program code (5 pts)
2. Technical questions (see below) (5 pts)
3. Correct feedback at each guess round (4 pts)
4. Correct judgment of code breaking result (4 pts)
5. Proper display of information (welcome, exit, etc.) (2 pts)
6. *Extra credit*: an automatic player that uses Knuth's algorithm
[you can reference [http://en.wikipedia.org/wiki/Mastermind_\(board_game\)](http://en.wikipedia.org/wiki/Mastermind_(board_game))]
(4 pts, contact the staff before the Final Submission if you want to attempt the extra credit).

The Preliminary Submission (PA #2) will contribute to the “elegant comments” and “technical questions” portion of your points (10 of the points). The Final Submission (PA #3) covers the remaining 10 points. Your submitted code must compile without error and execute correctly on the **Ed-lab Machines**. But you are free to develop/debug your program as you like before making sure it runs correctly on Ed-lab.

Preliminary Submission Due April 23

The purpose of the Preliminary Submission is to ensure you have designed correct abstractions for your functions. By April 23, you should submit answers to the “technical questions” below along with a partial program that:

- Contains at least three new function **declarations** with comments specifying requirements of the input and function behavior/output
- Includes a *main* function that prints the welcome message to the screen
- Compiles without errors and warnings

Each function declaration must have comments descriptive enough for the reader to understand the purpose of the code, its assumptions, and its result. Our sample code provides an example of proper comments for function declarations. Submit your preliminary code in your Ed-lab course folder (name it **Mastermind**). Please also bring a hardcopy of your program to your discussion section on April 23 so that the TA can provide feedback to you. We will post sample C code on the course Web page.

Technical questions for preliminary submission:

1. Do a modified book problem 14.15 whereby you draw the run-time stack when the program is about to return from function **g** rather than function **f**.
2. Do book problem 16.8 and draw the portion of the run-time stack representing the known structure and values in the activation records for `main()` and `triple()` at the time when **triple()** has finished creating its activation record, but has not yet begun its computation.
3. Do the previous technical question if the code were instead:

```
int main ()
{
    int a, b, c;
    a = 1; b=2; c=3;
    triple(a,b,c);
}
```

4. Explain why the contents and structure of the run-time stack are different for the two cases above.

Final Submission Due April 30

Submit your final, well-documented code and a sample output of the game by putting them in your Ed-lab course folder. Comments should conform to the philosophy discussed in lecture. Difficult to follow code, bugs, and other undesired behavior will result in a loss of points. Make sure to comment at the top of your program **the compiling command** (e.g., `g++ mastermind.c -lm -o mastermind`). Each major piece of code/function should have comments descriptive enough for the reader to understand the purpose of the code, its assumptions, its effects, and its result. Avoid redundant single instruction comments when it's obvious (e.g., `i++; // Increment i` has a redundant comment).

We have zero tolerance for **plagiarism**. You may not share code with your peers, borrow code from past or present or future students, the Web, eBay, the Interwebz, etc. You may borrow code from the book if you document the borrowing. Keep in mind that your course staff specialize in fraud detection, pattern matching, and cryptanalysis. **We have automated tools to encourage students to “stay honest” about avoiding plagiarism.** See Handout 1 for how we handle cases of suspected plagiarism. The long process of prosecuting plagiarism is no fun for anyone—be it students or staff. We thank you for coding and playing with honor.

If you have any questions or concerns about the plagiarism policy or want to know if a particular external resource could be used, please contact cs201-staff@cs.umass.edu.