MLA Header

Mastermind

Team 5

The goal of our final C++ project was to demonstrate topics learned in class through the application of a console program. Suggested projects mainly included variations of games or calculators. Choosing a C++ project is truly challenging because anything can be done with C++ (source). C++ is preferential for banking and preserving privacy because of principals like encapsulation the protect data (source). We chose a project that we were confident we could complete, and that could be continually expanded upon as we learned new concepts. A console game was a good fit because it allowed the use of control flow, loops, and evaluating user input. We chose the 1970’s board game, Mastermind (source on date of game).

Mastermind is a code breaking game created by Mordecai Meirowitz, who was an Israeli postmaster (source). There are two players in Mastermind, a code maker and a code breaker. The players would alternate making and breaking a code. The game is played with a board and colored pegs. The ‘maker’ sets the code in colored pegs, hidden from the code breaker, and the the ‘breaker’ also makes their guesses in colored pins. Clues are given to the player in the form of a colored pin showing whether a guess is correct or not (source on gameplay). The game can also be played with pencil and paper, but goes by the name ‘Bulls and Cows’ (source on ‘Bulls and Cows’).

There were several challenges in converting the traditional board game to a console application. While accepting and evaluating user input was straightforward, communicating information back to the user was a larger challenge. We had to create a method that would convey hints back to the user, but not give the user more information than they should have. This proved a problem when evaluating the guess array. Initially the data was evaluated and printed in the original array order. This told the user information about specific array items, when they should only know generally how many guesses were correct or incorrect. The clue array was obfuscated by ordering the hints numerically.

A bottom up method was used to write the program. We initially started out with a, basic but functional, program to generate a random array, and take user input. That base gameplay was built upon and grew as the project continued. Using bottom up programming allowed us to start out with something we knew we could complete, and adding in more functions let different programmers easily work on separate parts.

 Through this method, gameplay and “User Experience” elements of the game were developed separately. Gameplay had to be completed before Introduction and Exit screen functions were created. The information screens explain the gameplay to the user and allow the user to control what the program does. Several of the information screens are reused. The goal of the introduction and exit was to be informative but do as little coding as possible.

The built in function, int rand(), was used to generate a pseudo random number. The numbers seem random to the user but are based off a seeded number initialized by, srand(time(NULL));. The range needed to be limited to the numbers, 0-9. Double digit number would complicate input and make guessing a four digit array unlikely. The range is limited by modifying the function.

(Insert code with % 10, center on page).

The secretcode() function, contains the rand() function and uses a for loop to assign the random number to the array code[4], the array the user is trying to guess.

 Aside from the rand() function, exit function, and sort function, all other functions were defined by the team. Checktwo() evaluates user input against the randomized array and stores a hint to be relayed back to the user. An if/else statement was used to assign either a:

* + - * 2. Correct number in the correct position
			* 1. Incorrect position, but number is somewhere in array
			* 0. Incorrect position, and number is not anywhere in array

Both the randomized number sequence and user guess are both stored as arrays. The use of an array speeds up the program because the data is stored in sequentially (source on array increasing speed), and works well with a for loop.

Readability in code is important when working as part of a team. We followed the format discussed in class. Space was used to aid in reading. C++ ignores white space, which lets spaces be added to differentiate different sections of code (source on white space). Variables and functions were also named in a way that was easy type and informative in what they did. Most variable were user input or arrays.

Int int code[4]; // secrect code to be guessed

int guess[4]; // user guess

int clue[4]; // shows clue to user

int hold[4]; // an array to keep track of which elements of the user guess had been accounted for

 Notes were especially important as the program evolved. Notes were made on what functions and variables do, what should be changed in the program, references, and as a placeholder for suggestions.

Backups of the program were created at several points and exist at various stages. Most larger sections began as new programs and ran entirely on their own before they were inserted into the Mastermind program. This allowed sections to be tested and fully functional before introduced.

 The user is able to select the difficulty of the game through int diff. The user has the ability to choose hard, normal, or easy settings. The easy setting gives the user more attempts at solving the code. By default the user will be given the normal setting if they do not correctly choose easy or hard.

 If/Else statements were used, as previously described, to evaluate user input. These worked out well because there were only three possibilities depending on user input: fully correct, correct number with incorrect position, or incorrect number and position. User input of a character instead of a number would also result in the condition of ‘incorrect number and position’. If guess[i] == code[i], then the clue[i] would be assigned ‘2’. Else if, guess[i] is somewhere in code[], then clue[i] would be assigned ‘1’. Else, the input is incorrect and receives a ‘0’.

Switch case statements were used in the introduction and exit screens. The user has several options for how they would like to proceed. They can: read the rules, begin gameplay, or exit. The user enters the matching number to their option. Switch cases worked well because it gives the user options, but gameplay is defined by the program. The default case means the program continues whether the user follow the instructions or not.

For loops were used in creating the random array being guessed and collecting user guesses. The code being guessed is four numbers long. The program only needs four numbers from the user. The loop used to collect guesses is inside gameplay().

Conclusion

The main thing we would change if we were to make this program again is to see if we could have certain functions actually return values. The way the program runs now, most functions are of void type, and they manipulate variables and arrays within themselves rather than returning a value to an array or variable. If checktwo() were into for example, it could return certain values to the array rather than just changing the values of the array within the function.