<Primitive

MASTERMIND>

Name: Jose Aguirre

Class: CSC5

Date: July 24, 2015

Purpose: Project #2

Introduction

I begin this introduction by discussing my first project. I used project 1 as a stepping stone. I realized that if I could create a code game(which I did do) it would allow me to gain insight into the programming concepts behind mastermind.

As I stated in my first project: initially I began this project, as I imagine most people do, with a big idea and little working knowledge of how to make it happen. For my first project I wanted to create a primitive version of the game mastermind with the hope that eventually I would be able to program my own version of mastermind. Mastermind is a two player coding game which allows one player to set a code and a second to break it. The game I wanted to create for my first project required one player to guess a four letter code. It would include a minimum of two players(similar to mastermind). The first player would type in a four letter code. Then, the second player would begin by guessing one letter at a time. For example, the first player might create a code called doiz

Player 1 would be prompted to enter a four letter word on the screen:

Enter a four letter word player1:

doiz (player one enters the word doiz, or some other four letter code)

Player 2 would try to guess the word one letter at a time:

(player 2 enters a letter, if correct he moves on

if incorrect he gets an undetermined number of tries)

Enter letter 1:

A

Incorrect Try Again:

B

Incorrect Try Again:

C

Incorrect Try Again:

D

Correct.

Enter letter 2:

A

Incorrect Try Again:

E

Incorrect Try Again:

I

Incorrect Try Again:

O

Correct

Enter letter 3...

If the player was able to guess all four correctly it would spew out some line on the screen like:

Correct! The code was DOIZ.

The idea was crystal clear, but the execution was full of ambiguity. If I wanted to write this game, I’d have to break it down into smaller components which might,if I put the right kind of effort in, be achievable.

I too thought that if I could get this spin-off version of mastermind working correctly I could then build on it to create the mastermind game at a later time.

One of the problems I ran into in this first version was that the ’26’ characters of the alphabet was too many for the user to play with. It is hard enough for the user to guess with just a few characters. I did change the game so that the user only had to guess a four letter code choosing from the first five letters of the alphabet in some randomly generated order.

Admittedly, to recreate mastermind, I thought that all I would have to do was create a similar program and change the letters of the alphabet to colors. However, I was wrong.

There were a few things I had to worry about. For example, the project needed to: be a minimum of 250 lines, include functions and arrays(1d and 2d), pass arrays b/w functions, functions needed pass by value, functions needed to include pass by reference, functions needed to include defaulted parameters, functions needed to return primitive data types, needed to make sure output was formatted, needed to read and write to files, and sorting game players/searching should also be included. Oh yeah, it should also include any additional concepts(not yet covered) from the final weeks of class.

Maybe I had to worry about more than just a few, but honestly, I was more worried about getting my program to work correctly. I figured I could build my program entirely in the main function and then go back and transform it to include all of the necessaries mentioned previously.

Problems with My Version of Mastermind

With this in mind I set out to design my program. Once I had a working version I ran into my first problem. Mastermind, you see, is a code game. Typically two people are needed to play, one to set the code and the other to decode it. Therefore, if player1 set up a code such as:

player1: red, blue, green, orange

Then, player2 would be responsible for decoding it. He might guess:

player2: red, green, yellow, violet

As I said earlier, my ideas exceeded my capabilities, and with time running low, I decided to simplify the game further and make it a one person game. The computer would generate the code and the person playing the game would try to decode it. This change would now look like this.

player1(computer generated): red, blue, green, orange

player2: red, green, yellow, violet

The decoder guessed two colors correctly,red and green, but only red is in the correct place. How do you let the player know this?

Reader, I will tell you how I let the player know all this stuff. All games, including mastermind, have rules. Without the rules the game ceases to be a game and is simply a state of chaos. To solve this problem and acquaint my user with the rules I created a simple menu. When the user initialized the game she would see the following:

Hello!

Welcome to Mastermind!

Enter 1 to read the instructions.

Enter 2 to play the game.

Assuming that the user did enter 1, the following would pop up:

Instructions

OBJECT OF THE GAME

The computer is the Codemaker.

It will set up a line of four Code pegs (secret code)

in the secret code holes(somewhere in the computer's memory).

The code can be made up of any combination of the 8 coloured pegs.

You may NOT use two or more pegs of the same colour.

The Decoder can begin guessing, trying to

duplicate the exact colors and positions of the hidden Code pegs.

Each guess is made by typing a row of Code pegs on the screen.

After every guess, the Codemaker(computer) informs the Decoder of their progress as follows:

Black Indicator Pegs

Black Indicator Pegs will be notated with a 'B'

a 'B' will be placed under your code if a Code peg is

of the correct colour AND in the correct position.

White Indicator Pegs

White Indicator Pegs will be notated with a 'W'"

a 'W' will be placed under your line of code if a Code peg is

of the correct colour but NOT in the correct position.

Blank Space

Blank Space will just be an empty space

If decoder gets a blank line under code

it means that none of the colors entered were correct.

Decoders: if you break the secret code, the Codemaker(computer)

        places four Black Indicator pegs below that row and adjacent to

        that row reveals the hidden code.

Remember, you will only get 10 guesses to break the code.

If you are unable to, then you receive the message: YOU LOSE!

Good Luck!

End of instructions.

We’ve already discussed my first problem(making it a one player game and the computer the generator) and my second problem(creating a menu so user can familiarize themselves with the instructions of the game). However, there were more problems to this game. One of the biggest problems was dealing with duplicates. You may have wondered, what if the computer generates the following code instead:

player1(computer generated): red, red, green, orange

That is, what if it chooses the same color twice(in this case red). Or maybe three or four times. Some versions of mastermind, after all, do allow for duplicates. In any case, I purposefully decided that I would not allow my game to generate duplicates. Probably the coward’s way out, but sometimes it’s better just to get things working, than to get things working correctly. These three problems were all issues I would deal with. There were more problems which I will talk about later in this paper, and I am certain that there are even more problems, which to date, I am not aware of, but nevertheless still inhabit my program. By the way, sorry for that!

Let me remind you that the user is prompted with the following screen:

Hello!

Welcome to Mastermind!

Enter 1 to read the instructions.

Enter 2 to play the game.

If the user decides to play the game, then she will have initialized the program. The program has various components. Most of what the programmer writes, that is, the code, is never seen by the decoder. Nevertheless, it is important that I walk you through this process. Programs are designed, by convention, much like a recipe, from top-to-bottom. They have a heading, system libraries, user libraries, global constants, function prototypes, and then the main program.

The Heading

A heading in the c++ language is very similar to a heading in any other language or class. You start with a name, date, and any other information you feel is important. Here is an example of what you would see in a regular class(1), followed by the c++ way of writing a heading(2).

Example 1 Regular Heading:

 Author: Jose Aguirre

 Created on July 11, 2015, 11:28 AM

 Class: C++

Example 2 c++ Heading:

/\*Author: Jose Aguirre

 \*Created on July 11, 2015, 11:28 AM

 \*Class: C++

\*/

Alternate way of writing Example 2:

//Author: Jose Aguirre

//Created on July 11, 2015, 11:28 AM

//Class: C++

In C++ we begin a heading(or any multiline comment) with the characters /\* similarly we end the comment with \*/.

note: although the alternate method would also work in a c++ program, it is advisable that alternate method is only used for a single line of code and the method used in Example 2 be used on blocks(multiple lines) of code. After you are done writing the heading you can begin with the program.

**System Libraries**

The compiler doesn’t allow you just to type in regular English and translate that into machine language. For example, if you wanted the computer to output the message:

“Yo, WAZZUP?”

You couldn’t simply type in Yo, WAZZUP?

You would need to include a few lines of code first.

At the very top of the program are our system libraries. There are many different libraries, below are the ones used in this program. Note, if we wanted to write our message mentioned earlier we would have to include the iostream library which is responsible for output.

#include <iostream> //I/O Library

#include <cstdlib>//header library includes random numbers

#include <ctime>//header file for time 0 since 1970

#include <fstream>//files

using namespace std;//Namespace for iostream

Above us are the words using namespace std; This line of code too is essential in outputting a message.

**User Libraries**

The next item in our program is the user libraries. Unfortunately(or maybe fortunately) for you and me, I have no idea what these are yet, but I know they go after our system libraries. So let’s move on.

**GLOBAL CONSTANTS**

Global constants go after the user libraries. This is reserved for conversions or numbers that are universally agreed upon. For example we all agree that pi begins with 3.14152653589... and thus it could be declared as a global constant. Again though, my code game does not use global constants, so let’s keep on keepin’ on.

**Function Prototypes**

When we prototype a function we write that function’s name right below the global constants. I could have probably created my program without making separate functions, however, creating functions aids in overall readability of the program. For example, when you bake a cake, you could probably just put all of the ingredients straight into a pan, mix it, and bake, but we don’t do this. We split the recipe into different sections that say what to do, or functions.

driedIngredients();

wetIngredients();

combineIng();

There is one function which combines all the dried ingredients. A second that combines all the wet ingredients. Finally, a third that combines both the wet with the dry.

Note: Function names, such as, driedIngredients, can be written different ways. They, however, are never capitalized, that is reserved for constants. I’ve chosen to capitalize the first letter of the second word without using a space b/w the two; this is known as camelCase. Function prototypes all end with parentheses() followed by semicolon(;).

This is similar to what happens in mastermind(at least the way I programmed mastermind). First the user is welcomed with a greeting and brief introduction. The computer then generates the colors randomly and prompts the user for input. As soon as the user enters her first guess at the four color choices, the game is initialized. If the user wins or loses, she is prompted to try the game one more time. If she decides to quit, then a farewell message appears on screen. All of the highlighted words in this paragraph are functions in my game, and thus have to be prototyped underneath the global constants. This is what they look like prototypes:

greeting();

introduction();

generateColor();

userInput();

game();

farewell();

So far we have looked at the design of the program, we covered the heading, system libraries, user libraries, global constants, and function prototypes. The main part of our program, however, has yet to be covered. What follows is the main part of our program.

**int main()**

The main function is the main part of our program. It is declared using int main(). The int stands for integer and means that the function will return an integer. Typically when int main is initiated it ends with the line of code:

return 0;

ours is of no exception. This value is returned and means that the program terminated okay. Note, it is returning an integer, zero, which I will bring up later. The important thing here is that there are all types of functions and they return different things, in this case an integer. After int main() the body of our program is executed within two brackets:

{

//body goes here

}

Our body includes many lines inside of the two brackets, the first is a function call.

    enterName();

This function call prompts the user to enter their name, it is explained in detail after the main part of our program. Here when a function appears, it is just being called.

The following line of code is a comment followed by a variable being declared.

     //variable declaration

    char ans;

The variable is of type character, which in our case will be used to type in some key from the keyboard. The name of the variable ans is short for answer. Later the user will enter their answer when they are prompted to repeat.

The next line of code is the beginning of a do-while loop. It is initiated with the code do. A do-while loop is used to ensure that the user will go through the program at least one time. The code you want it to repeat is all kept in the body of two more brackets. It ends with a while construct. After the code while there are two parentheses which provide our condition or test, if the user puts in a true value the program is repeated, otherwise it is terminated. The following is a simplified version of the code inside the do-while loop:

do

{

//stuff you want to repeat goes here

}while(do you want to repeat?)

Here is the code inside of my program:

    do

    {

        //variable declarations

        int choice;//for user choices

        //prompt user input to choose from menu

        cout<<"Enter 1 for mastermind."<<endl

            <<"Enter 2 for mastermind instructions."<<endl

            <<"Enter 3 for guess the number game."<<endl

            <<"Enter 4 for order whole numbers game."<<endl

            <<"Enter your choice: "<<endl;

        cin>>choice;

        switch(choice)

        {

            case 1: mastermind();

            break;

            case 2: instructions();

            break;

            case 3: guessNum();

            break;

            case 4: orderNum();

            break;

        }

        //prompt user input to repeat program

        cout<<"Would you like to return to the menu?"<<endl

            <<"Type y for yes, any other key to end."<<endl;

        cin>>ans;

    }while(ans=='y'||ans=='Y');//create a condition to repeat

    return 0;

}

Inside of our do-while loop the user is prompted to choose an option. There are four options.

Play Game Mastermind

Read Instructions

Play Number Guessing Game

Sort Numbers

The user then inputs one of the choices. The choices are executed using a switch statement. A switch statement is useful when you want a user to decide between different options. The syntax for it is as follows:

switch(the users choice)

{

case 1: callsomefunction();

break;

case 2: callsomeotherfunction();

break;

}

Note, that after the function is called and executed, there is the word break followed by a semicolon. Whenever you are in a loop the code

break;

helps you break out of that loop, or the body of the two brackets{}. If those are the only two brackets in the program then it will terminate the program. However, in our case, those two brackets have been inserted inside of another two brackets, so it only exits the first two brackets.

From here the user can initialize various functions. However, in this paper I will only discuss the mastermind game as that is what this paper is about after all.

Therefore, if the user inputs a 1, mastermind is initiated.

The call is to the function mastermind(); This leaves the main part of the program and goes below the main, where the functions are usually written, not to be confused with the function prototypes which are at the top of the program. You see, you could write the function name and purpose at the top of the program, that is define and prototype a function all at the top, however, this is not typical programmer practice and the function is usually only prototyped at the top and defined at the bottom. This greatly enhances readability. All of the functions below the main part of our program begin with a comment which looks similar to the one below:

/\*\*\*\*\*\*\*\*\*\*\*\*\*functionName\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*Purpose:

 \*

 \* input:

 \* output:

 \*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

These comments state the function name, its purpose, the input and the output. I will, however, refrain from discussing this too much, as I will be explaining all of this within the code I’ve created.

The first function called is mastermind. It is a void function which just means that, unlike our int main(), it will not return a value, hence the name, VOID. The first line is the declaration type, name and two parentheses:

void mastermind()

This is followed by two brackets which include the structure of the mastermind game in it.

{

//structure of game goes here

}

I am afraid it is a bit more complicated than that though. Here is the actual code:

{

//time 0 calculates secs from beginning of comp time

    srand(time(0));

At the very top of the program is the srand(time(0)) which will help in generating a random number. That in turn will help in generating a random color from our array later.

After this line of code, the variables used are declared. There are const int variables, which stands for a constant integer. This is used for the size of an array. So my array will always have SIZE elements in it. In this case I’ve initialized SIZE to 8, so the array always has 8 elements in it. This is followed by the array with our 8 color choices. After the array are the 4 clrs that will be generated by the computer. I’ve initialized them to z because I’ve heard you always need to initialize variables, however, I am not certain it would make a difference here.

    //declare variables

    const int SIZE=8;

    char clrs[SIZE]={'r','o','y','g','b','i','v','w'};

    char clr1='z',clr2='z',clr3='z',clr4='z';//for code

Once the variables are declared even more functions are called. Some of the include values that are passed into them, like the introduction function.

    //call functions

    introduction(clrs,SIZE);

After the introduction, the colors are set. I’ve used a lot of do-while loops here as well to ensure that there are no duplicate colors. Remember, the original mastermind did have duplicates, however, in this case it will not, it is primitive mastermind afterall.

    clr1=setClr1(clrs,SIZE);

    //cout<<endl<<clr1<<endl;

After color 1 is set, color two is set to any color remaining, but color 1.

    do

    {

        clr2=setClr2(clrs,SIZE);

    }while(clr1==clr2);//ensures that there are no duplicates

Similarly, after color 2 is set, color 3 is set to any color but color 1 or 2.

    //cout<<clr2<<endl;

    do

    {

        clr3=setClr3(clrs,SIZE);

    }while((clr2==clr3)||(clr1==clr3));

    //cout<<clr3<<endl;

Finally, color 4 is set to any color as long as it was not used by color 1, 2, or 3.

    do

    {

        clr4=setClr4(clrs,SIZE);

    }while(clr3==clr4||(clr2==4)||(clr1==clr4));

    //cout<<clr4<<endl;

Once the colors have been set the game function is called. This is where the user will input her guesses.

    game(clr1, clr2, clr3, clr4);

}

This is the end of the mastermind function.

Recall that our mastermind function include several functions integrated in the body. The first was the introduction function. This function was responsible for displaying the array and a brief intro. It is also a void function because it does not return a value. The code follows:

void introduction(char clrs[], int SIZE)

{

    //output colors used

    cout<<"This is mastermind!"<<endl;

    cout<<"Your color choices are the following: "<<endl;

    for(int i=0;i<SIZE;i++)

    cout<<clrs[i]<<" ";

}

The user sees:

This is mastermind!

Your color choices are the following:

roygbivw

This is the end of the intro function.

Additionally, in the mastermind function the game function was also called. The game function consists of, well, the game. It is here that functions to set the colors are called, and it is also here that the player enters her guesses. This function has a lot going on. It has character types that are being passed by reference, this is done within the two parentheses using an ampersand symbol. The four color choices are passed in this way. The following is the code for it:

int game(char& clr1, char& clr2, char& clr3, char& clr4)

After this the program is built within two brackets. Like before the variables are declared. In this function we also use a string. This helps output how many tries the user has left. Our guesses are also declared here. Recall that the colors where set as character types, therefore, our guesses will too be of type character.

{

    //declare variables

    const int numSIZE=10;

    string num[numSIZE]={"first","second","third","fourth","fifth","sixth","seventh","eight","ninth","tenth"};

    char guess1,guess2,guess3,guess4;//user inputs their guess

After the variables have been declared, the user is prompted to enter four color choices and press enter.

    //Prompt user input

    cout<<"\nEnter your four color choices, then press enter."<<endl;

Then the loop is initiated. It first outputs enter your \_\_\_\_\_ guess. In the blank line our string is initiated to say first, second, third, ... etc...

It runs through the first of the loops.

Is color 1 equal to guess 1, if so output a B which stands for a black peg, if not don’t output anything. Unless it is the right color in the wrong place, in that case output a W.

Similarly, it does the same for colors 2, 3, and 4. B for correct, and W for correct color, but wrong position. The code is kinda scary:

    for(int j=0;j<numSIZE;j++)

    {

        cout<<"\nEnter your "<<num[j]<<" guess:"<<endl;

        cin>>guess1>>guess2>>guess3>>guess4;

        if(guess1==clr1)

            cout<<"B";

            else if(guess2==clr1||guess3==clr1||guess4==clr1)

                cout<<"W";

        if(guess2==clr2)

            cout<<"B";

            else if(guess1==clr2||guess3==clr2||guess4==clr2)

                cout<<"W";

        if(guess3==clr3)

            cout<<"B";

            else if(guess1==clr3||guess2==clr3||guess4==clr3)

                cout<<"W";

        if(guess4==clr4)

            cout<<"B";

            else if(guess1==clr4||guess2==clr4||guess3==clr4)

                cout<<"W";

If the user guesses the correct code within the given number of tries, the program outputs: Congratulations! You Win! and displays the code. It then returns to the menu: If the user does not guess in the given amount of tries it displays the code and ask if she would like to return to the menu to play again.

if(guess1==clr1&&guess2==clr2&&guess3==clr3&&guess4==clr4)

        {

            cout<<"\nCongratulations! You Win!"<<endl;

            return 1;

        }

    }

    cout<<"\nCODE="<<clr1<<clr2<<clr3<<clr4<<endl;

}

You may not remember this, but back in the mastermind function the colors were set. They were set using a function call called setCLr1,2,3,4. The purpose is to set each color equal to something from our options:ROYGBIVW. The code is the same for all four functions. It begins with a function type in this case, character because it is a letter we are after. Next, is our function name. The function names were setClr1, 2, 3, or 4. Inside our parameters we pass our array and the size of the array. The body of the function includes a variable declaration for the first color and it is generated randomly. Remember earlier I said that there was a weird function called

srand(time(0))

which I said I would explain later. That function generates some number, and when its cousin, the random function is called, it makes it truly random. here I’ve set the colors equal to

colors[random()%8]

Colors is the name of the array, and it is looking for an index. Which one? It that’s up to the computer. So why the percentage sign and the eight. Well that percentage sign is called a modulus in programming, and much like other math-y words in programming they don’t have the same meaning as in a math class. What it means is calculate the remainder when you divide by eight. So if I had something like

10%8 the answer is 1 remainder 2

The % sign however only returns the 2. So it will return one of eight numbers: 0-7. Which is precisely what I need since my array had eight elements.

colorArray[0]=r

colorArray[1]=o

colorArray[2]=y

colorArray[3]=g

colorArray[4]=b

colorArray[5]=i

colorArray[6]=v

colorArray[7]=w

The code in this case is a lot more simple:

char setClr1(char clrs[], int SIZE)

{

    //declare variables

    char clr1;

    //set the code

    clr1=clrs[rand()%8];

    return clr1;

}

**Problems With My Program.**

As I stated earlier, there are, I’m sure, problems in my program which I remain unaware of, but in this case it was essential that I had something which was working, even if incorrectly. There are however, problems I was aware of.

Once I had built my program, I realized that it still didn’t include a search or sort. I decided that I would create a simple number guessing game which would include a search of an array. I could in this way illustrate that I did understand that search. I was able to complete this.

I also created a file so that I could have a user input their name and store it in a file. At the end of the game I would say good-bye and tell them their name and their score. I, however, only partially completed this.

For the sort, I was going to create another little game that would ask a user to enter their name. It would then take the letters and sort them in alphabetical order. I only partially finished this. I used a basic bubble search, which at present, is the only one I understand. There are others though. There was simply not enough time to finish everything I wanted, and understand everything I learned, but overall, I feel competent in my ability to explain a program to others who do not know a thing about programming. I have a friend from the University of Redlands, who was the only person I knew that ever took a c++ course, tell me that she did not know anything about c++, although she passed with an A. I may not pass with that grade, but I know the structure of a program.

//system libraries

//user libraries

//global constants

//funct prototypes

//execution of main program

I also feel confident in my ability to work with functions. I got my feet wet and tried to demonstrate my understanding of 1-d and 2-d arrays. I also showed a short guessing game which illustrated a search. I can also output to a file and do a basic bubble sort. There are still certain things I am uncomfortable with, but I know what they are. I know what was covered, and what I do not understand. In this project and write up, I’ve tried to show my understanding of c++, a very vast language still to me, after 6 weeks of intensive study.