**Program 1**

**<Primitive Mastermind>**

**<4-letter code game>**

Class: CSC 5

Instructor: Dr. Lehr

DATE: July 16, 2015

Name: Aguirre, Jose

**Introduction**

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. I wanted to create a primitive version of the game 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 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 she moves on if incorrect she gets an predetermined number of tries before exiting program)*

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:

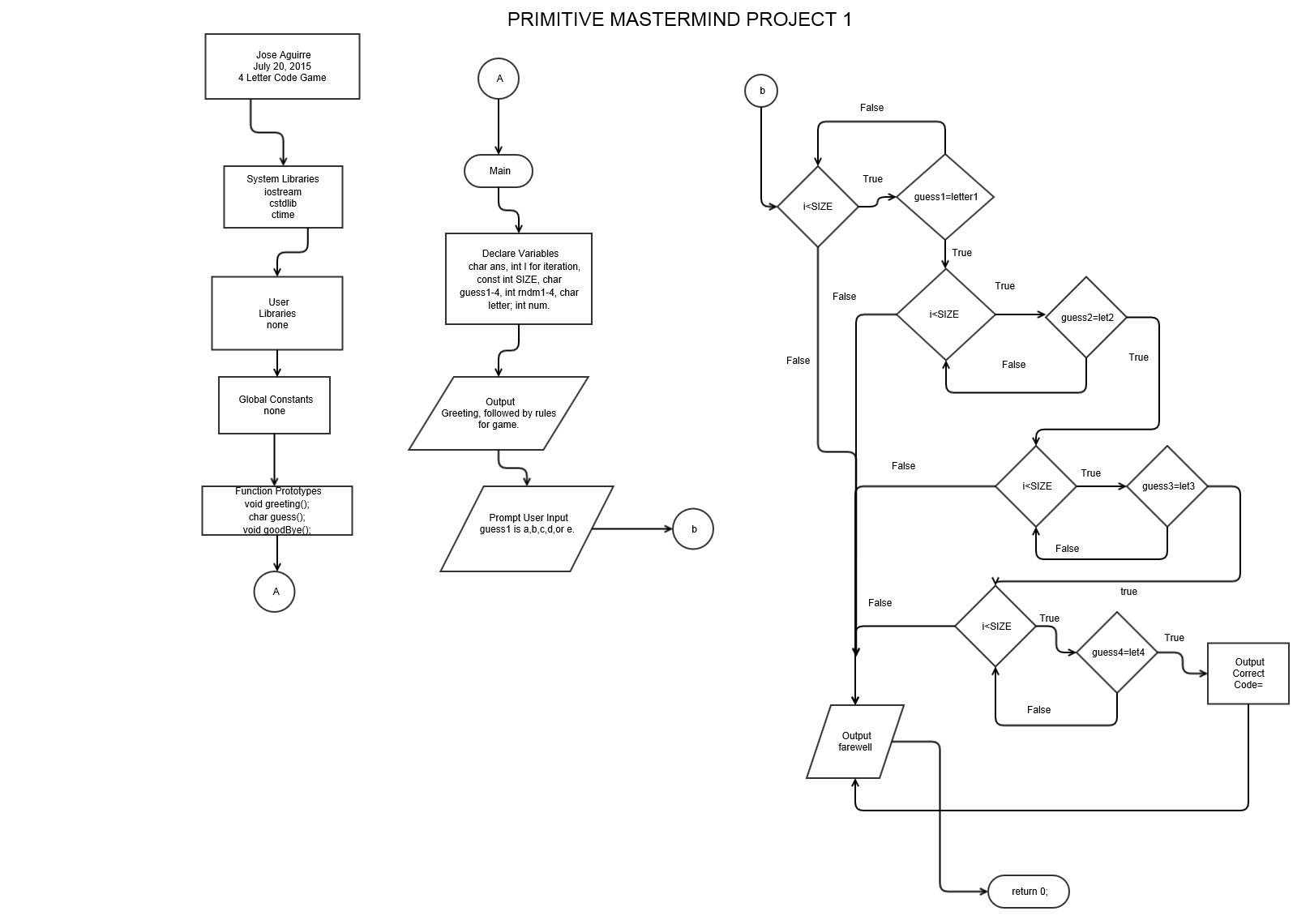
Correct! The code was DOIZ.

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.

In the following pages I will walk you through a flowchart of the finished version, a brief explanation of C++, the code to the finished version, my attempts at creating the different versions of the program(Appendix 0), and finally, the code for all of the versions(Appendix 1).

**Flowchart**



**The CODE TO Version 7**

**The Four Letter Code GAME Is Complete!**

This particular program is written in a computer programming language known as C++. There are many types of computer programming language, but essentially they all do the same thing.

The computer, you see, is just a simple machine. It only knows one kind of language, **machine language**. Machine language is just a bunch of strings of **binary code.** Binary codeis just a combination of 0’s and 1’s. For example, if I am looking for the number 2, the machine language doesn’t recognize it, b/c there are only two digits it can recognize, 0 or 1. So how does it translate the two. Well, as I am writing this answer I am realizing that a lot of material covered in this paper will

require many more technical issues than I’m sure my reader is prepared to deal with. Nevertheless, I will attempt a quick explanation at binary. You see there are many base systems, we humans use a decimal system b/c we have 10 fingers. We count the following way:

**0,1,2,3,4,5,6,7,8,9, and 10**

However, If we only had 9 fingers(or digits), nine would be our last finger and we would count to our new 9 as follows:

**0,1,2,3,4,5,6,7,8, and 10**

Therefore, in base 9 there is no 9, it is our last finger so that finger = 10, but we don’t pronounce it ten, we pronounce it one zero base nine.

Similarly, if we were an animal with only two fingers(although I’m not sure they exist), we would count.

**0,1, and ?**

**0,1, and 10**

Remember this is read one zero base two.

As you can see, this machine language business is really tedious stuff, and this is probably why languages such as C++ were invented. C++ allows you to write your program using English(along with some additional commands), and this is translated to machine language. The **compiler** which is responsible for the translation then translates the machine code back into a format familiar to the user(hopefully).

Now that I’ve briefly introduced C++, let’s look at the code. The code is made up of different sections. I will introduce it from top-to-bottom(the same way the computer reads it).

**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++

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.

**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*

*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 construct in our program are 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**

The next construct is the function prototypes. Although a program can be written with out function prototypes, it is advisable that function prototypes are used, b/c it helps with overall readability. For example, you could write a book without chapters and just have a reader read it from beginning to end. However, if you include chapters, it greatly improves the readability and framework of the overall book. There are different types of functions, but since our program only uses the **void** and **char** type, this will be the only types discussed in this paper. Void means, not that our function is empty, but rather that it does not return a value. That is, our function isn’t performing any mathematical operation so it won’t be returning a sum, difference, product, or anything similar to this. Our void functions are just greeting and outputting farewells. The char function is going to return a type, and as you might have guessed the char stands for character. In this case the characters we are dealing with are the first five letters of the alphabet.

I’ve used three void functions in my program. The first,

***void greeting();//greets players***

is a greeting. I preprogrammed five greetings, so anytime the game is initiated the user is greeted.

The second,

***char guess();//#game***

is the actual program. This is the meat and potatoes. Here the user enters a guess and is allowed a number of chances(or **iterations**) to guess correctly.

Finally, the third function,

***void goodBye();//says bye to players***

is very similar to the first. However, instead of a greeting it outputs a farewell message.

**int main()**

We can now begin our execution of the program. Following the function prototypes is our main program. This part of the program always begins with **int main().** Following the parentheses() there are always two brackets {}. Your program goes in b/w the two brackets. Before you end your program w/ the final bracket ‘}’ you must include the following line of code:

return 0;

which basically means, this program is finished. Here is an example:

int main()

{

//your program goes here

return 0;

}

Now, below is the actual code used in my program. Notice that my program includes some extra stuff, like the variable declarations and the function prototypes mentioned earlier.

int main()

{

//variable declarations

    char ans;

    //loops greeting and game

    do{

        greeting();//this function generates a random hello

        guess();

    //prompt user to continue playing

    cout << "Play Again?" << endl

         << "Enter Y to retry," << endl

         << "any other key to end game." << endl;

    //user inputs y to play again and any other key to stop

    cin >> ans;

    }

    while(ans == 'y' or ans == 'Y');

    //generate a random bye-bye message

    goodBye();

    return 0;

}

Much like in math class, variables are used here to represent something. In this case the variable ans is used to represent the answer the user will input at the end of the program. If she wishes to repeat the program he will enter a character, which has been set to ‘y’ or ‘Y’, if she does not wish to continue she will input any other key or character.

After the variable has been declared there is a loop. A do-while loop to be specific. There are many types of loops, but I went with a do-while loop because once the program is initiated the user has to play at least once. I’m kinda biased, but I think my game is pretty cool, and I want them to play it at least once so that they can see how cool it is too. After the do there are also two brackets and anything in those brackets is repeated, so long as the condition for the while statement is met. The condition in the while statement is known as a **boolean** statement. A boolean statement can be true or false, if true the *do* part is repeated, if false it exits the loop. In our case above the user types in ‘y’ for a true and any other key for false. Simplified it looks like this:

do

{

//it will keep doing this if the while expression is met

}while(condition);

Within our do brackets the program calls on our previously mentioned function prototypes. Once the user exits the loop the program initiates the final function prototype. The user is given a farewell message and the program is terminated. We know it ends b/c of the return 0; and final ‘}’.

**After int main();**

This section also aids in readability. At the top of the program, you’ll remember, I declared the function prototypes. I could of typed in the entire code for the prototypes, but that would of been messy. So instead, I only declare them, then at the end of the program I type in all of the code. This is the final part of my program. Here is the code:

Note: all of the code is bold to aid with readability

It begins with the

***char guess();//#game***

I’ll remind you this is the meat and potatoes. Here the user enters a guess and is allowed a number of chances(or **iterations**) to guess correctly. Before the function is a multi line comment which we use the /\*\*/ characters for. This comment tells the person reading the program what the function does. The purpose, the input, and the output of the function are listed inside of the comment.

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

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* guess \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Purpose: Runs the game and the iterations(guesses user gets).**

**\* Input: User inputs: a or b or c or d or e**

**\* Output: If first input is correct user inputs**

**\* second guess, or else user keeps trying the first letter**

**\* If user does not guess in the amount required, the game ends.**

**\***

**\*/**

The function char guess, is where the magic is happening. There are variable declarations just like in the main program. Before each declaration there is a comment which tells what the variable is for. The comment is a single line and thus the // characters are used.

**char guess()**

**{**

**//variable declarations**

**//for loop, i is used for the number of iterations**

**int i;**

**//for the size of our array**

**const int SIZE=5;**

**//the 4 different letters the user will input**

**char guess1,guess2,guess3, guess4;**

**//the 4 dif letters the computer will generate**

**int rndm1,rndm2,rndm3,rndm4;**

**//the letters used in this game**

**char letter[SIZE]={'a','b','c','d','e'};**

After the declarations, the program generates the five letters using the rndm function. It then outputs the purpose of the program and prompts the user to enter a guess.

**//time 0 makes rand truly random**

**//rndm1-4 will correspond to an element in the array**

**srand(time(0));**

**rndm1 = (rand()%5);**

**rndm2 = (rand()%5);**

**rndm3 = (rand()%5);**

**rndm4 = (rand()%5);**

**//output program's purpose and prompt user input**

**cout<<"This is a guessing game."<<endl**

**<<"The objective of the game is that you"<<endl**

**<<"guess a code which is generated by the computer."<<endl**

**<<"The code is four characters in length and is"<<endl**

**<<"made up of only the first five letters of the alphabet."<<endl**

**<<"Those characters are: a, b, c, d, e"<<endl**

**<<endl**

**<<"For example, a four letter code might be: "<<endl**

**<<"adec"<<endl<<"or"<<endl<<"adda"<<endl<<"or even"<<endl<<"bbbb"<<endl;**

**cout<<"Enter your first letter: "<<endl;**

At this point there is no returning. The user has entered my labyrinth. As noted earlier, it is not very readable, however, it is working, and sometimes workability is preferable to simple workability. Primarily, because it consumes less of the programmer’s time.

If the user guesses correctly she moves on to next letter, otherwise he continues guessing until she runs out of guesses and loses the game. 

**//enter the loop**

**//all four letters**

**//iterations is set < SIZE**

**for(i=1; i<SIZE; i++)**

**{**

**cin>> guess1;**

**if(guess1!=letter[rndm1])**

**{**

**cout<<"Try again!"<< endl;**

**}**

**if(guess1==letter[rndm1])**

**{**

**for(i=1;i<SIZE;i++)**

**{**

**cout<<"1st letter correct."<<endl;**

**cout<<"1st letter: "<<guess1<<endl;**

**cout<<"2nd letter: ";**

**cin>>guess2;**

**if(guess2!=letter[rndm2])**

**{**

**cout<<"Try again!"<<endl;**

**}**

**if(guess2==letter[rndm2])**

**for(i=1;i<SIZE;i++)**

**{**

**cout<<"1st & 2nd letter correct."<<endl;**

**cout<<"1st letter: "<<guess1<<endl;**

**cout<<"2nd letter: "<<guess2<<endl;**

**cout<<"3rd letter: ";**

**cin>>guess3;**

**if(guess3!=letter[rndm3])**

**cout<<"Try again!"<<endl;**

**if(guess3==letter[rndm3])**

**for(i=1;i<SIZE;i++)**

**{**

**cout<<"1st & 2nd & 3rd letter correct."<<endl;**

**cout<<"1st letter: "<<guess1<<endl;**

**cout<<"2nd letter: "<<guess2<<endl;**

**cout<<"3rd letter: "<<guess3<<endl;**

**cout<<"4th letter: ";**

**cin>>guess4;**

**if(guess4!=letter[rndm4])**

**cout<<"Try again!"<<endl;**

**if(guess4==letter[rndm4])**

**{**

**cout << "Winner!"<<endl;**

**cout<<"CODE: ";**

**cout<<letter[rndm1]<<letter[rndm2]<<letter[rndm3]<<letter[rndm4];**

**cout<<endl;**

**return -1;**

**}**

**}**

**}**

**}**

**}**

**}**

**cout<<"You loose"<<endl;**

**}**

Again, before the function is a multi-line comment which we use the /\*\*/ characters for. This comment tells the person reading the program what the function does. The purpose, the input(none in this case), and the output of the function are listed inside of the comment.

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

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* greeting \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Purpose:  To display a greeting message**

**\* Input:**

**Void funnction, no input necessary**

**\* Output:**

**\*      greeting**

**\*/**

In this case I designed the program to output 5 different cases. I could’ve chosen more, but 5 was enough. The fifth entry is never generated, but it was used as a default when I was developing the different cases. I used the **switch** statement to generate the greetings. The switch statement allows you to develop many different cases. The program, however, only chooses one, which is all I need.

**void greeting()**

**{**

**srand(time(0));//to make the program truly generate a random greeting**

**int num;**

**num = (1+rand()%4); //+1 bc 0 is not a valid entry**

**//cout << num; I did this to trace bc could not figure out where 0 was coming from(%)**

**switch (num)**

**{**

**case 1:**

**cout << "Hi!" << endl;**

**break;**

**case 2:**

**cout << "Hola!" << endl;**

**break;**

**case 3:**

**cout << "Welcome!" << endl;**

**break;**

**case 4:**

**cout << "Yo! Yo! Yo!" << endl;**

**break;**

**default:**

**cout <<"Not valid entry" << endl; //it will generate zero**

**}**

**}**

Again, before the function is a multi line comment which we use the /\*\*/ characters for. This comment tells the person reading the program what the function does. The purpose, the input, and the output of the function are listed inside of the comment.

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

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* goodBye \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Purpose:  To display a farewell message and exit game**

**\* Input:**

**any key other than 'y'**

**\* Output:**

**\*      farewell message**

**\*/**

Much like the greeting, the farewell is designed the same way. It also includes a switch statement. However, this function can be initialized one of two ways, either the user exits the program, or loses her game and thus the goodBye function is automatically called.

**void goodBye()**

**{**

**srand(time(0));//to make the program truly generate a random greeting**

**int num;**

**num = (1+rand()%5); //+1 bc 0 is not a valid entry**

**//cout << num; I did this to trace bc could not figure out where 0 was coming from(%)**

**switch (num)**

**{**

**case 1:**

**cout << "Good-bye!" << endl;**

**break;**

**case 2:**

**cout << "Smell ya later!" << endl;**

**break;**

**case 3:**

**cout << "See you later!" << endl;**

**break;**

**case 4:**

**cout << "Hasta la vista" << endl;**

**break;**

**case 5:**

**cout << "Thank you, come again!" << endl;**

**break;**

**default:**

**cout <<"Not valid entry" << endl; //it will generate zero**

**}**

**}**

**Appendix 0**

**Version 1 Guessing Number Game**

I began my program and quickly realized that my idea far succeeded my capability. Like any normal person, I decided to break the problem down into smaller parts. “The gist of the game, is that the second player guess,” I said to myself.

A guessing game!

Eureka!

I could just create a game in which a user(yes, I went from a min of two to only one user), would guess a number generated by the computer. The computer would generate a number 1-9 and the user would get three shots to guess correctly. The program was working, and since I was getting the hang of the **rand()**, don’t worry, that’s just a word that when typed into the **IDE** generates a random number, kinda(I’ll explain later, I promise), I decided to create two more random **functions.** A function is just a smaller component of a program, within the larger program. These two smaller parts of my program greeted my user when she arrived with 1 of 5 greetings randomly selected by, well, by **rand()**, and said a farewell message when they were finished with the game.

*The corresponding code can be found in Appendix 1.*

**Version 2 Repeating Guessing Number Game**

I realized that it sorta sucked that my game did not allow my user to repeat the game over and over and over and over again, until they decided it was time to call it quits. So, in my second version I decided to allow them to repeat the program over and over again. In computer programming this is called a **loop**, there are dif. kinds of loops, but I simply used the famous **do-while loop.** I preferred this loop because it ensures that your player will play the game at least one time. I’m kinda biased, but I think my game is pretty cool, and I want them to play it at least once so that they can see how cool it is too.

*The corresponding code to version 1 can be found in Appendix 1.*

**Version 3 Guessing Vowel Game**

Alright, so I still wasn’t any closer to producing my initial game, wait!

Yes I was!

I broke the game into parts, the first and fundamental part of my game was that it was a guessing game. However, I was not interested in numbers, I was interested in guessing a four letter code. I didn’t exactly know how to go about getting there yet, because four letters was a lot of different combinations, but even letters can be broken into parts. For example, there are consonants(lots of them), and there are vowels.

VOWELS!

What if my third version was a guessing vowel game; that is, what if the player only had to guess one of the five vowels: A, E, I, O, U. Simple enough, but the random number generator only generates, well, numbers. This was the tricky part of version three, is there a random letter generator?

short answer: NO!

long answer: kinda. If you’re willing to make one, and I    was!

I began searching the web, and I kept reading a whole lot about these things called **arrays,** don’t worry, I’m not gonna define this b/c I don’t know what it is either. I’d remembered reading about these things called **enumerations** which basically allow you to create your own type. I thought since my type only had five characters(a,e,i,o,u) this would be simple enough.

*The corresponding code to version 3 can be found in Appendix 1.*

**Version 4 Guessing vowel game, but entered by a user.**

Once I had debugged version three, I realized I was one step closer to my four letter code guessing game. However, it was the computer generating the vowels and not one of the users who was inputting the code(Need I remind you this was suppose to be a two player game). I knew that the next step in solving this problem was to create a game which would allow for two players, one to input the vowel and the other to guess it. I did this by allowing the user to put in a vowel, sort of. I gave the user a table that said

 "Enter 1 for a, 2 for e, 3 for i, 4 for o, and 5 for u."

but then I realized that the vowel or corresponding number would be showing on the screen and it wouldn’t be much of a guessing game. So I added in several **endl** or **\n** to my code. These both do the same thing, essentially they add a new blank line to your program, hence their name, the **new line character**.

**So**

**instead**

**of**

**pressing**

**return**

**to**

**type**

**a**

**new**

**line,**

**you**

**would**

**just**

**write**

**endl.**

I could re-write the preceding lines using the statement below:

cout<<“So”<<endl<<“instead”<<endl<<“of”<<endl<<“pressing”<<endl<<“return”<<endl<<“to”<<endl<<“type”<<endl<<“a”<<endl<<“new”<<endl<<“line,”<<endl<<“you”<<endl<<“would”<<endl<<“just”<<endl<<“write”<<endl<<“endl.”

Note how much more compact it is.

*The corresponding code to version 4 can be found in Appendix 1.*

**Version 5 Creating an Enum w/ All Leters**

Version 4 was working, from this point forward, it would all be easy, or so I thought. BTW, it turns out that what I thought was an enumeration was actually an **array.** An array is very useful in coding because it allows us to group data types. It also allows us to create our own data type. In this case, the letters of the alphabet are a data type. I would just have to create an **array** which would hold all 26 letters of our alphabet. I would then have to create a table which told the first player which numbers to enter to produce the code. However, it was here that I realized that my array wasn’t really doing much. This was a real buzzkill, especially since I had already gone on to version 6 and didn’t realize my mistake until I began version 7, back to the drawing board. My users were not going to input letters, they were going to convert the letters into numbers and then type in a number. For example, if the correct letter was ‘a’, they wouldn’t type in ‘a’, they would type in the number ‘1’. There was a whole table:

Here's a table to help you out

1=a 2=b 3=c 4=d 5=e 6=f

7=g 8=h 9=i 10=j 11=k 12=l

    13=m  14=n 15=o 16=p 17=q 18=r

    19=s 20=t 21=u 22=v 23=w 24=x

         25=y   26=z

*The corresponding code to version 5 can be found in Appendix 1.*

**Version 6 Four Letter Code**

In version 5 I realized that once I added the other three letters, the code would exceed a few hundred lines. This was sort of hard to read, so I decided to create **functions** for each letter, in total there were four functions. My program was working, but not exactly the way I wanted it to. The first player had to input a letter, then the second player had to guess the first letter. Second, the first player had to input a second letter and the second player had to guess the second letter... and so on. This was inefficient. I wanted to create a program in which the first player would simply type in a string of letters, but instead a player typed in a string of numbers that corresponded to four letters. As I said earlier, this was a buzzkill. It was very disappointing to have built myself up to this version only to realize I didn’t really have a program superior to the original number generator I’d created in version 1. Worse yet was that I tried incessantly for hours to *transform* my program into something useful. I didn’t want to start all over, but after hours of frustration I realized that is exactly what I had to do. That is how I came up with version 7.

**Appendix 1**

This appendix initially included the code in its entirety. However, this document exceeded 65 pages so I ran it in ideone.com. If you’d like to view the code simply click on link.

**Version 1 Code to Guessing Number Game**

<http://ideone.com/uYfn0>O

**Version 2 Code to Repeating Guessing Number Game**

<http://ideone.com/C094NE>

**Version 3 Code to Guessing Vowel Game**

<http://ideone.com/TjRTAQ>

**Version 4 Code to Guessing Vowel Game w/ User Input**

<http://ideone.com/Zlb1bf>

**Version 5 Code to Creating an Enum w/ All Leters**

<http://ideone.com/PK8opZ>

**Version 6 Code to Four Letter Code**

<http://ideone.com/QTJimc>

**Version 7 A Simplified Version of a Big Idea**

<http://ideone.com/LDBzdN>