# Project 2:

## Title:

## Left Center Right V.2

## Dice Game

## Class:

## Cis 5 - 41367

## Due Date:

## February 14, 2021

## Author:

## Aamir Khan

# Introduction:

**Title: Left Center Right V.2**

**Goal**: The goal of the game is to be the last one standing with tokens. This game may be played for fun or may be played with consequences on the line that make for a more competitive game.

**Rules**: This version can either be played with 3 or 5 players. Each player will take a turn rolling 3 dice to begin with, and on successive turns, will roll as many dice as they have tokens. The game goes on until only one remains with tokens.

*1. Roll a 4 - Give a token to your opponent on the left.*

*2. Roll a 5 - A token is lost to the center.*

*3. Roll a 6 - Give a token to your opponent on the right.*

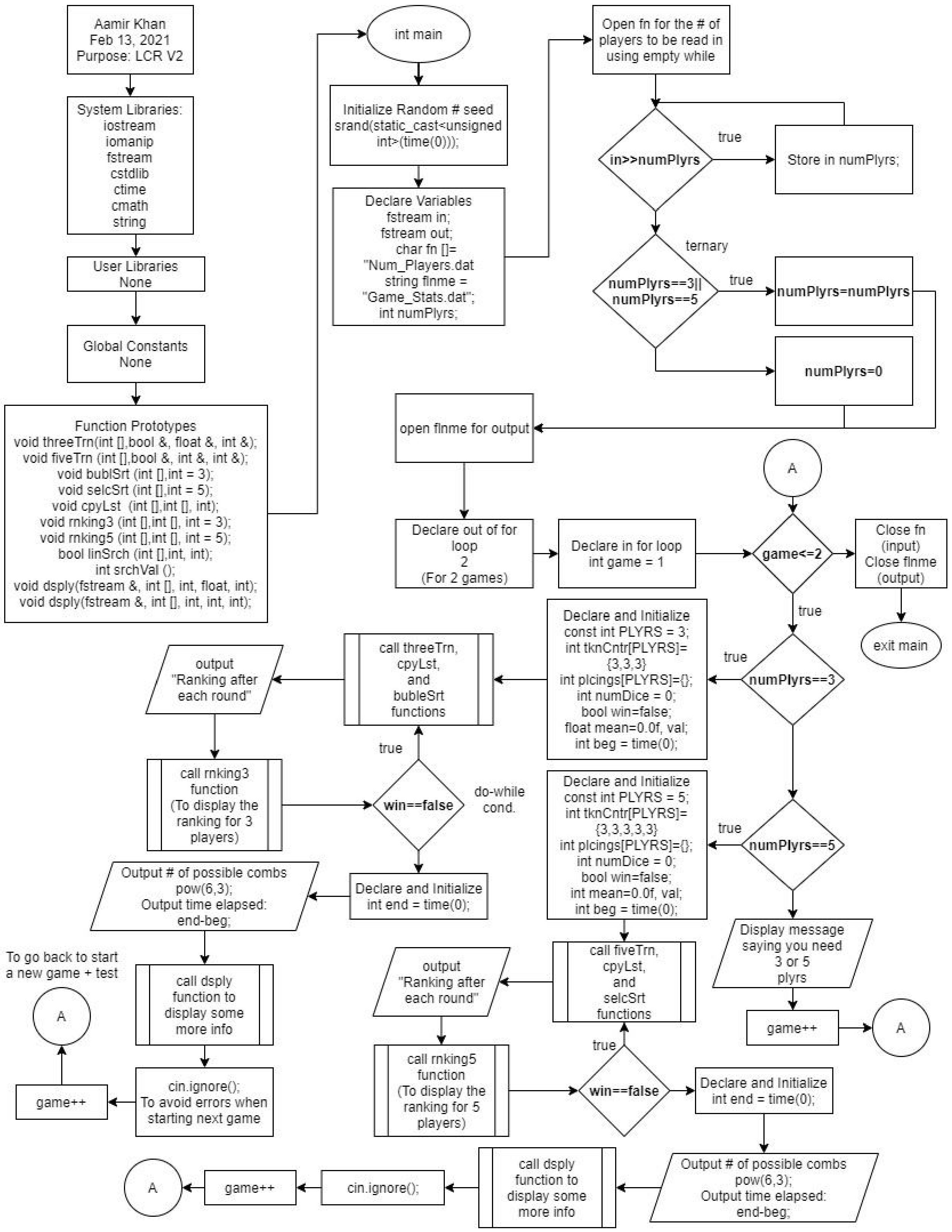
*4. Roll a 1-3 - The game goes on and no token is lost for that die.*

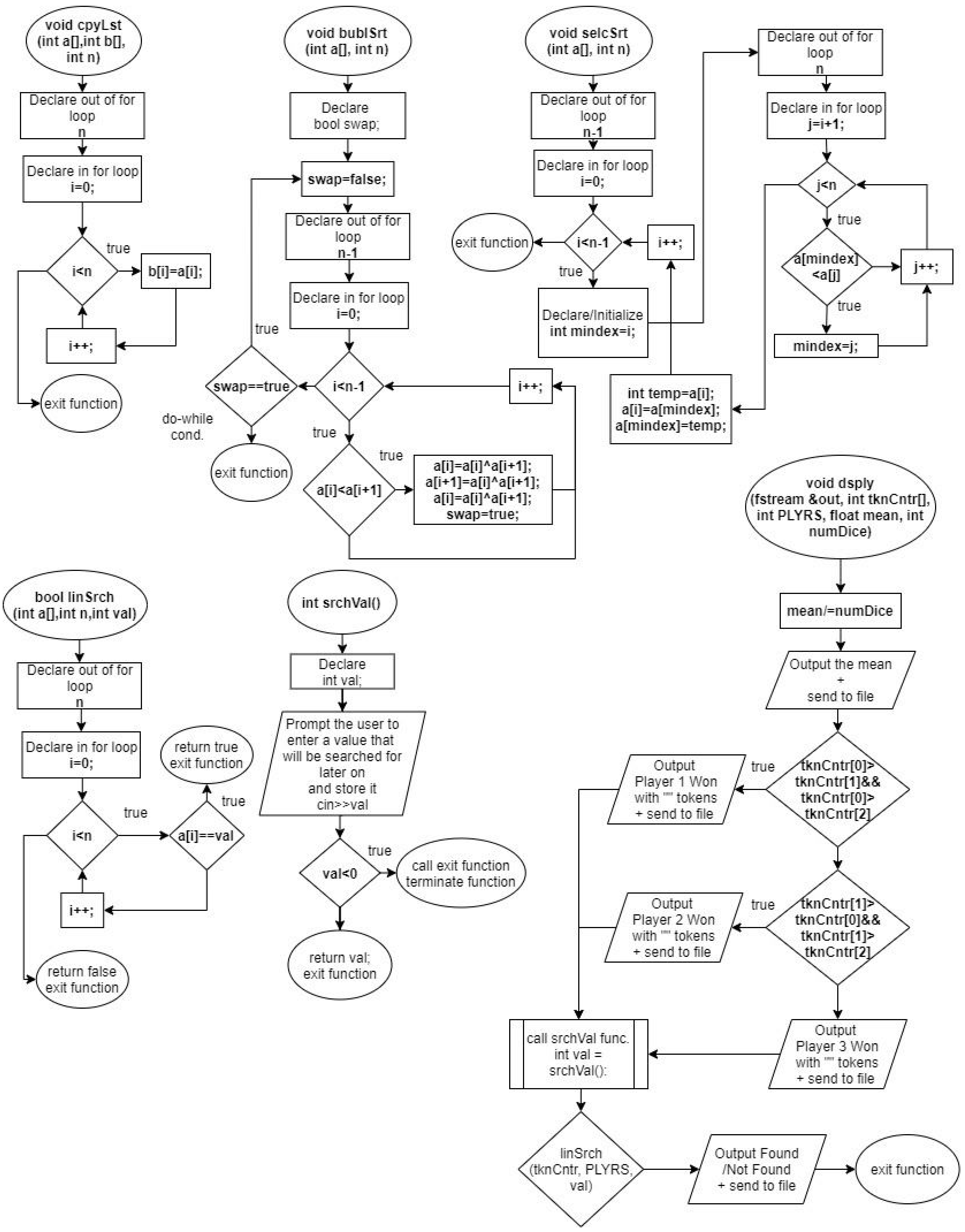
# Summary:

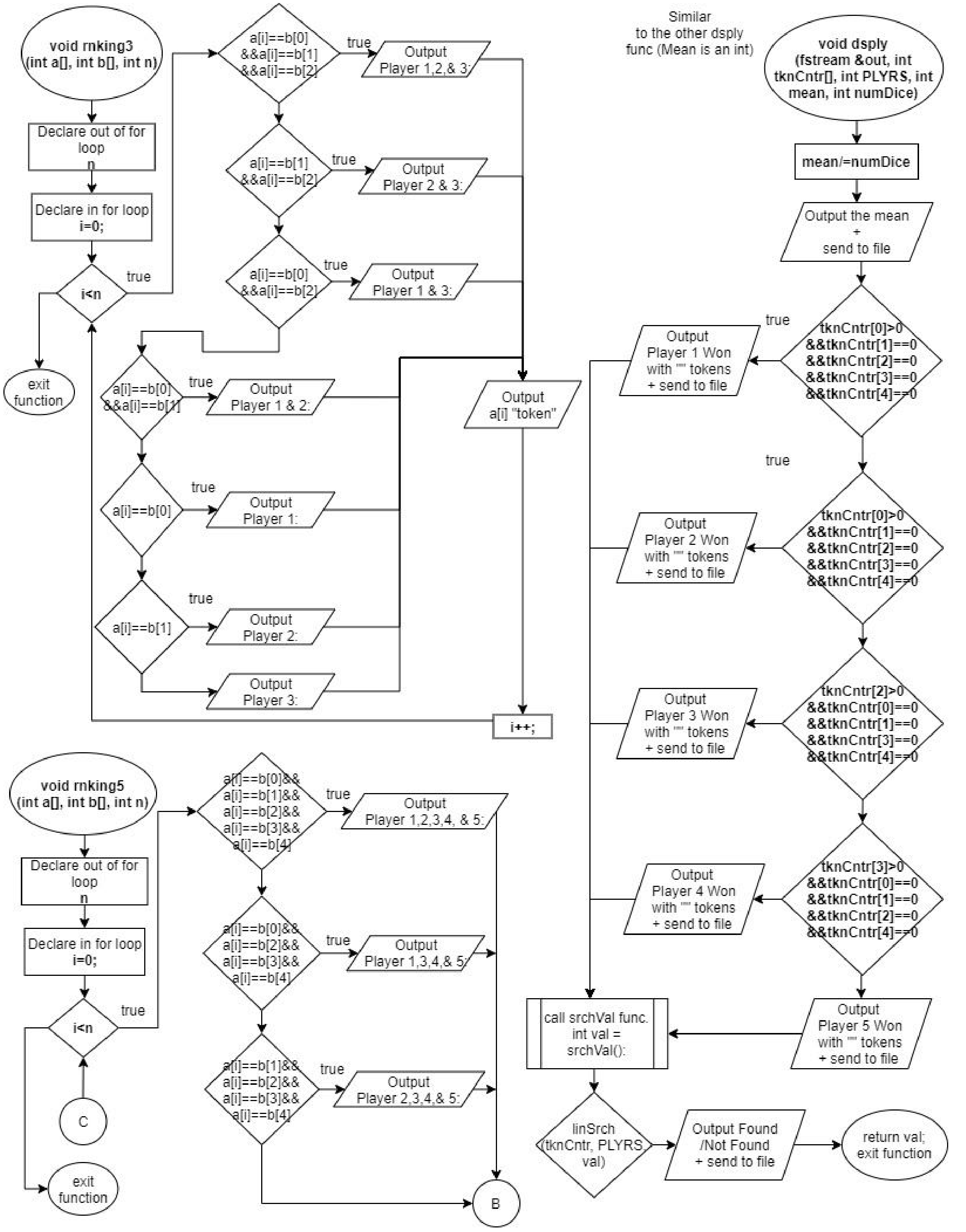
Project Length: About 570 Lines

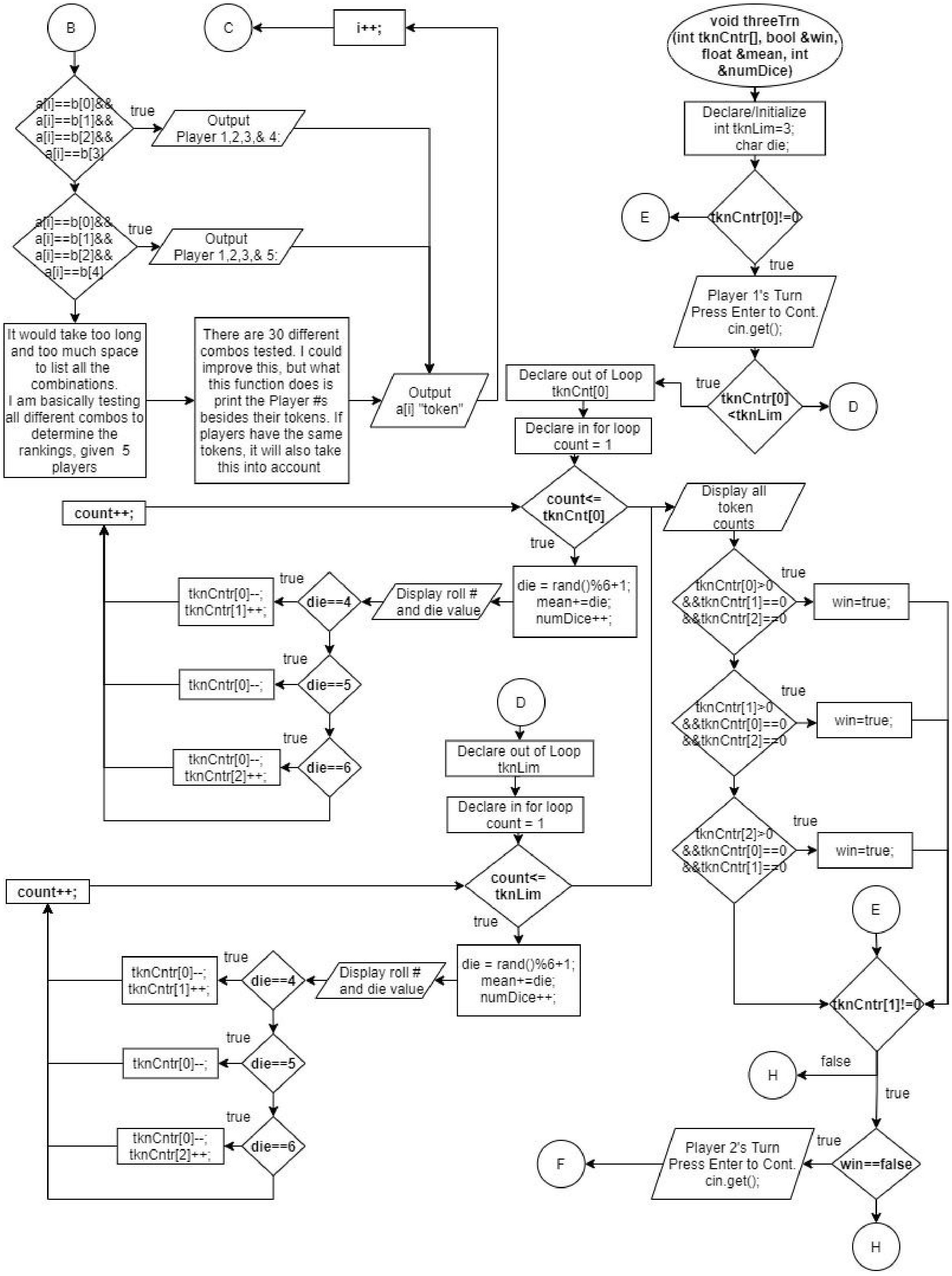
For this project, I expanded on Version 1 by implementing various concepts we covered in class past Chapter 5, such as searching, sorting, etc. The game takes advantage of for loops and dependent if statements, switch cases, and more for the calculation side of things. A do-while is used for repetition as needed and two games are played in one run. As far as concepts are concerned, I used various branching constructs and data types that I picked up on throughout Gaddis, ranging from Chapters 1-9. My code is very lengthy, which is I believe it could've been improved in more ways than one. The lines added up because I chose to make another function that allows 5 players to play the game. As a result, I also incorporated function overloading that dealt with both 3 and 5 players, respectively.

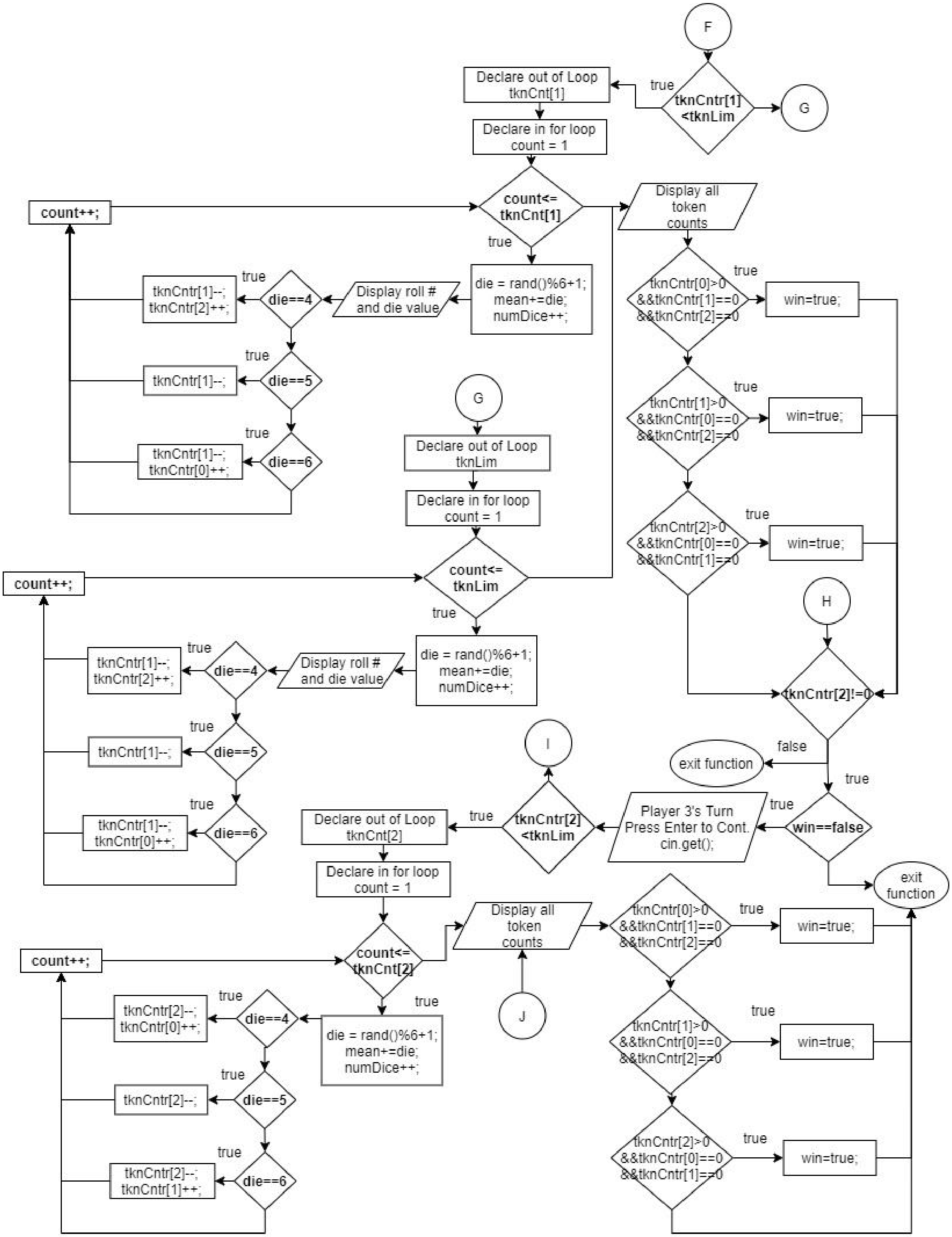
# LCR V.2 Flowchart:

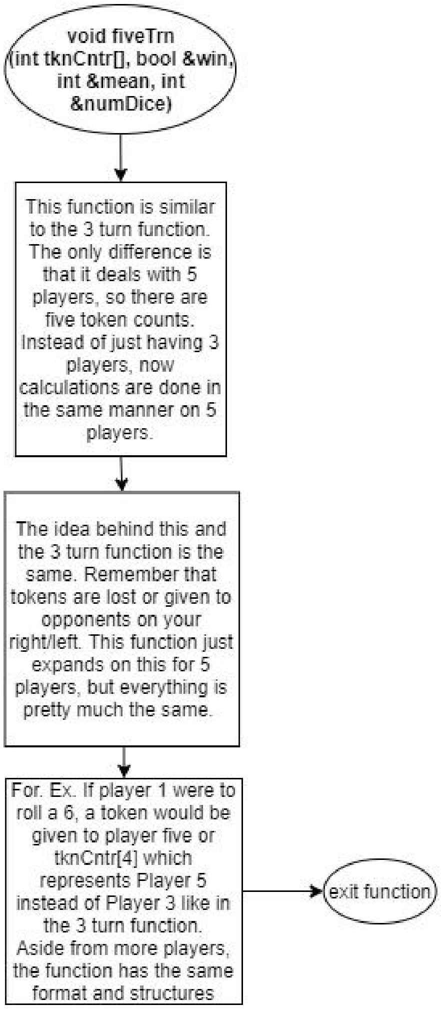


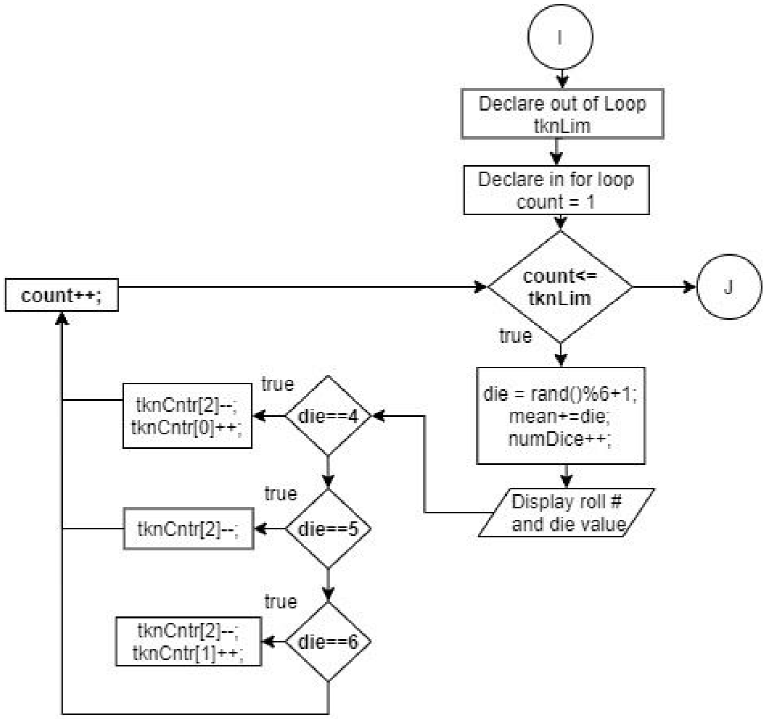

void cpyLst (int a[],int b[], int n)
Declare out of for loop n
Declare in for loop i=0;
i<n
true
b[i]=a[i];
i++;
exit function
void bublSrt (int a[], int n)
Declare bool swap;
swap=false;
Declare out of for loop n-1
Declare in for loop i=0;
i<n-1
true
a[i]<a[i+1]
true
a[i]=a[i]^a[i+1];
a[i+1]=a[i]^a[i+1];
a[i]=a[i]^a[i+1];
swap=true;
i++;
swap==true
true
do-while cond.
exit function
void selcSrt (int a[], int n)
Declare out of for loop n-1
Declare in for loop i=0;
i<n-1
exit function
true
Declare/Initialize int mindex=i;
i++;
Declare out of for loop n
Declare in for loop j=i+1;
j<n
true
a[mindex] <a[j]
true
mindex=j;
j++;
int temp=a[i];
a[i]=a[mindex];
a[mindex]=temp;
void dsply (fstream &out, int tknCntr[], int PLYRS, float mean, int numDice)
mean/=numDice
Output the mean + send to file
tknCntr[0]> tknCntr[1]&& tknCntr[0]> tknCntr[2]
true
Output Player 1 Won with "" tokens + send to file
tknCntr[1]> tknCntr[0]&& tknCntr[1]> tknCntr[2]
true
Output Player 2 Won with "" tokens + send to file
Output Player 3 Won with "" tokens + send to file
call srchVal func. int val = srchVal():
linSrch (tknCntr, PLYRS, val)
Output Found /Not Found + send to file
exit function
bool linSrch (int a[],int n,int val)
Declare out of for loop n
Declare in for loop i=0;
i<n
true
a[i]==val
true
return true exit function
i++;
return false exit function
int srchVal()
Declare int val;
Prompt the user to enter a value that will be searched for later on and store it cin>>val
val<0
true
call exit function terminate function
return val; exit function


void rnking3
(int a[], int b[], int n)
Declare out of for loop
n
Declare in for loop
i=0;
i<n
true
exit function
a[i]==b[0]
&&a[i]==b[1]
&&a[i]==b[2]
true
Output
Player 1,2,& 3:
a[i]==b[1]
&&a[i]==b[2]
true
Output
Player 2 & 3:
a[i]==b[0]
&&a[i]==b[2]
true
Output
Player 1 & 3:
a[i]==b[0]
&&a[i]==b[1]
true
Output
Player 1 & 2:
a[i]==b[0]
true
Output
Player 1:
a[i]==b[1]
true
Output
Player 2:
Output
Player 3:
Output
a[i] "token"
i++;
void rnking5
(int a[], int b[], int n)
Declare out of for loop
n
Declare in for loop
i=0;
i<n
true
C
exit function
a[i]==b[0]&&
a[i]==b[1]&&
a[i]==b[2]&&
a[i]==b[3]&&
a[i]==b[4]
true
Output
Player 1,2,3,4, & 5:
a[i]==b[0]&&
a[i]==b[2]&&
a[i]==b[3]&&
a[i]==b[4]
true
Output
Player 1,3,4,& 5:
a[i]==b[1]&&
a[i]==b[2]&&
a[i]==b[3]&&
a[i]==b[4]
true
Output
Player 2,3,4,& 5:
B
Similar
to the other dsply
func (Mean is an int)
void dsply
(fstream &out, int
tknCntr[], int PLYRS, int
mean, int numDice)
mean/=numDice
Output the mean
+
send to file
tknCntr[0]>0
&&tknCntr[1]==0
&&tknCntr[2]==0
&&tknCntr[3]==0
&&tknCntr[4]==0
true
Output
Player 1 Won
with "" tokens
+ send to file
true
tknCntr[0]>0
&&tknCntr[1]==0
&&tknCntr[2]==0
&&tknCntr[3]==0
&&tknCntr[4]==0
Output
Player 2 Won
with "" tokens
+ send to file
tknCntr[2]>0
&&tknCntr[0]==0
&&tknCntr[1]==0
&&tknCntr[3]==0
&&tknCntr[4]==0
Output
Player 3 Won
with "" tokens
+ send to file
tknCntr[3]>0
&&tknCntr[0]==0
&&tknCntr[1]==0
&&tknCntr[2]==0
&&tknCntr[4]==0
Output
Player 4 Won
with "" tokens
+ send to file
Output
Player 5 Won
with "" tokens
+ send to file
call srchVal func.
int val =
srchVal():
linSrch
(tknCntr, PLYRS,
val)
Output Found
/Not Found
+ send to file
return val;
exit function


B
a[i]==b[0]&&
a[i]==b[1]&&
a[i]==b[2]&&
a[i]==b[3]
true
Output
Player 1,2,3,& 4:
a[i]==b[0]&&
a[i]==b[1]&&
a[i]==b[2]&&
a[i]==b[4]
true
Output
Player 1,2,3,& 5:
It would take too long and too much space to list all the combinations.
I am basically testing all different combos to determine the rankings, given 5 players
There are 30 different combos tested. I could improve this, but what this function does is print the Player #s besides their tokens. If players have the same tokens, it will also take this into account
Output
a[i] "token"
C
i++;
void threeTrn
(int tknCntr[], bool &win,
float &mean, int
&numDice)
Declare/Initialize
int tknLim=3;
char die;
E
tknCntr[0]!=0
true
Player 1's Turn
Press Enter to Cont.
cin.get();
true
tknCntr[0]
<tknLim
D
Declare out of Loop
tknCnt[0]
Declare in for loop
count = 1
count<=
tknCnt[0]
true
count++;
die = rand()%6+1;
mean+=die;
numDice++;
Display roll #
and die value
true
die==4
tknCntr[0]--;
tknCntr[1]++;
true
die==5
tknCntr[0]--;
true
die==6
tknCntr[0]--;
tknCntr[2]++;
Display all
token
counts
tknCntr[0]>0
&&tknCntr[1]==0
&&tknCntr[2]==0
true
win=true;
tknCntr[1]>0
&&tknCntr[0]==0
&&tknCntr[2]==0
true
win=true;
tknCntr[2]>0
&&tknCntr[0]==0
&&tknCntr[1]==0
true
win=true;
D
Declare out of Loop
tknLim
Declare in for loop
count = 1
count<=
tknLim
true
count++;
die = rand()%6+1;
mean+=die;
numDice++;
Display roll #
and die value
true
die==4
tknCntr[0]--;
tknCntr[1]++;
true
die==5
tknCntr[0]--;
true
die==6
tknCntr[0]--;
tknCntr[2]++;
E
tknCntr[1]!=0
false
H
true
win==false
true
Player 2's Turn
Press Enter to Cont.
cin.get();
F
H


F
tknCntr[1]
<tknLim
true
G
Declare out of Loop
tknCnt[1]
Declare in for loop
count = 1
count<=
tknCnt[1]
count++;
true
die = rand()%6+1;
mean+=die;
numDice++;
Display roll #
and die value
die==4
true
tknCntr[1]--;
tknCntr[2]++;
die==5
true
tknCntr[1]--;
die==6
true
tknCntr[1]--;
tknCntr[0]++;
Display all
token
counts
tknCntr[0]>0
&&tknCntr[1]==0
&&tknCntr[2]==0
true
win=true;
tknCntr[1]>0
&&tknCntr[0]==0
&&tknCntr[2]==0
true
win=true;
tknCntr[2]>0
&&tknCntr[0]==0
&&tknCntr[1]==0
true
win=true;
G
Declare out of Loop
tknLim
Declare in for loop
count = 1
count<=
tknLim
count++;
true
die = rand()%6+1;
mean+=die;
numDice++;
Display roll #
and die value
die==4
true
tknCntr[1]--;
tknCntr[2]++;
die==5
true
tknCntr[1]--;
die==6
true
tknCntr[1]--;
tknCntr[0]++;
H
tknCntr[2]!=0
false
exit function
true
win==false
true
Player 3's Turn
Press Enter to Cont.
cin.get();
tknCntr[2]
<tknLim
I
true
exit
function
Declare out of Loop
tknCnt[2]
Declare in for loop
count = 1
count<=
tknCnt[2]
count++;
true
die = rand()%6+1;
mean+=die;
numDice++;
die==4
true
tknCntr[2]--;
tknCntr[0]++;
die==5
true
tknCntr[2]--;
die==6
true
tknCntr[2]--;
tknCntr[1]++;
Display all
token
counts
J
tknCntr[0]>0
&&tknCntr[1]==0
&&tknCntr[2]==0
true
win=true;
tknCntr[1]>0
&&tknCntr[0]==0
&&tknCntr[2]==0
true
win=true;
tknCntr[2]>0
&&tknCntr[0]==0
&&tknCntr[1]==0
true
win=false;


void fiveTrn (int tknCntr[], bool &win, int &mean, int &numDice)
This function is similar to the 3 turn function. The only difference is that it deals with 5 players, so there are five token counts. Instead of just having 3 players, now calculations are done in the same manner on 5 players.
The idea behind this and the 3 turn function is the same. Remember that tokens are lost or given to opponents on your right/left. This function just expands on this for 5 players, but everything is pretty much the same.
For. Ex. If player 1 were to roll a 6, a token would be given to player five or tknCntr[4] which represents Player 5 instead of Player 3 like in the 3 turn function. Aside from more players, the function has the same format and structures
exit function


I
Declare out of Loop tknLim
Declare in for loop count = 1
count<= tknLim
J
true
die = rand()%6+1; mean+=die; numDice++;
Display roll # and die value
count++;
true
die==4
tknCntr[2]--; tknCntr[0]++;
true
die==5
tknCntr[2]--;
true
die==6
tknCntr[2]--; tknCntr[1]++;

Note: The first page of the flowchart contains the code written in main. This is what is actually being executed at run-time. The rest of the flowchart contains the function definitions and applications. In other words, the remaining five pages explain what the function does. For the five turn function, I chose to not write it out as it would've been somewhat repetitive of the three turn function. The same ideas and constructs are used, and there isn't much of a difference in terms of what the function is meant to do. However, the only real difference is that there are two more players to account for, and it follows that the three turn function was modified to add these two players in. Understanding the three turn function is enough to understand the five turn function, which is why I only flowcharted the three turn function. (Refer to the code to see why.)

# Pseudo-Code:

```
/*
* File:   main.cpp
 * Author: Aamir
 * Created on February 6, 2021, 1:00 PM
 * Purpose:  LCR V.7
 */

//System Libraries
//I/O Library
//Formatting Library
//String Library
//File Library
//Rand # Generator
//Time set to seed
//Math Library
//Namespace std of system libraries

//User Libraries

//Global Constants

//Main -> Execution Begins Here
    //Initialize the Random # Generator Seed
    //Declare Variables
    //Input.open/close() functions
    //Output.open/close() functions
    //fn file for the Num_Plyrs file
    //flnme file for game stats
    //Open fn for input & flnme for output
    //Input the # of players and loop through file to read in content
    //Limit the # of players to 3 or 5

    //Use a for loop to play 2 games

    //Check if players == 3, then create and initialize an array with 3 for token counters

    //Create another array that will be used to hold the sorted copy for rankings

    //Declare some variables, like the # of dice, a bool win for repetition, value to search for,

    //and another variable to hold the mean value rolled

    //Declare and Initialize beg variable to keep track of time -> beg = time(0);

    //Use a do-while to repeat as long as win == false

        //call threeTrn func
        //call cpyLst func
        //call bublSrt func
        //call ranking func and print out the rankings
        //After the game is finished, print out some info and send to file
        //Declare and Initialize end variable to keep track of time -> end = time(0);
        //call display func to display the info

    //Else if Check if players == 5, then create and initialize an array with 3 for counters
    //Create another array that will be used to hold the sorted copy for rankings
    //Declare some variables, like the # of dice, a bool win for repetition, value to search for,
    //and another variable to hold the mean value rolled
    //Declare and Initialize beg variable to keep track of time -> beg = time(0);
    //Use a do-while to repeat as long as win == false
        //call fiveTrn func
        //call cpyLst func
        //call selcSrt func
        //call ranking func and print out the rankings
        //After the game is finished, print out some info and send to file
        //Declare and Initialize end variable to keep track of time -> end = time(0);
        //call display func to display the info
    //Else display a message that says you need 3 or 5 players
//Close the remaining two files and exit main!
```

Note:

The ranking function used for three players is different from the ranking function used for five players. The display function is also different for both, but they share the same name as an example of function overloading. I used two different sort functions: a bubble sort for when there are three players and a selection sort for when there are five players. The display function displays some general information such as the mean value rolled, whether the value search for was found, the winner, and their count.

# Stages of Development:

**Version 1 - Rolling1Die**

This was where my game began. The idea behind this version was to get a single working die and some stats to come with. Not much was covered here except using a random number generator to get a random number between 1 and 6.

**Version 2 - Rolling3Die**

This was where I got 3 working dice to go with my game. I added onto the previous version to get three random values, each between 1 and 6 to simulate rolling 3 dice. (Since die is declared as a char, it's necessary to type cast each time we roll a die as an int.)

**Version 3 - TokenCounters**

This version prioritized creating 3 working counters, and actually getting started on the token calculation side of things. In this version, one round is experienced, but the 3 token counters are incremented and decremented appropriately based on what die was rolled and who rolled it. With each turn, current token counts would then be displayed. If Player 1 rolled a 4, Player 1 would lose a token and it would go to Player 2. When a 5 is rolled, the token is lost for good, and when a 6 is rolled, the token is lost and given to the opponent to the right. All of this was implemented in this version, and would then go on to version 5 alongside looping. This version utilized dependent if statements to accomplish logic.

**Version 4 - TokenCounters**

This is where I began incorporating and implementing new concepts that were covered later on past chapter 5. It is here that I actually throw in the three player turns into a function that carries out what we had in Version 3, but now in a function. I also utilized an array for the token counters this time. The array had 3 elements, each initialized with 3 for that starting token count. The three turn function takes the array of token counters as an argument, and this array is appropriately tweaked given the turns and rolls that take place. As far as progress is concerned, the purpose of this version was to begin using newer concepts and taking an array as an argument.

**Version 5 - Game Begin**

This version introduced looping into the equation by using a do-while loop. The focus for this version was to introduce file input and reading. Additionally, a while loop was used to read in the contents of the file. This input was then validated using a ternary operator. In this version, I read in the number of players playing, and used it as input to decide whether to allow the game to proceed or not. The goal was to get a working function with file input and output. Some other information was added, such as determining the mean and the winner. This would later be put into a display function that would display all the general information about the game played.

**Version 6 - GameBegin**

In this version, sorting is introduced. A copy of the token counters array is made and then sorted. Independent if statements are also used after Player 1's turn as well as Player 2's turn to determine whether the game should be continued or not and also to skip the player's turn if their counter is equal to 0. In order to do the sort, a copy function was made to copy the contents of the token counters array to another array. This array was then used to display the rankings through a ranking function. At the very end, the winner is displayed using a nested ternary operator. Versions 1-6 now cover most of what we've learned in Gaddis Chs. 1-7, aside from some missing concepts and other small modifications that will be added onto version 7 as finishing touches. From here on, the game's logic is complete, but other necessary components needed for the project will be thrown in, such as searching and even implementing a 2D array.

**Version 7 - GameBegin**

I added many components that were previously left out, such as defaulted arguments and searching. I used a linear search to search the token counters array for a value of the user's choice. I then went on to add an entire function used to play with five players, so now there's two options to choose from, regarding the number of players playing. As such, I needed to write another ranking function specifically for five players. I then introduced two different sorting functions to meet the check-off list requirements. I used a bubble sort for when there are three players and a selection sort for when there are five players. I even threw in the info that was to be displayed into a function, so that the code in main was more simplified and short. The info that is displayed includes: a mean calculator that calculates the mean die value rolled throughout one complete game, the time elapsed each game played in seconds, whether the value searched for was found, and the number of possible outcomes one could have in using 3 6-sided dice. This version concludes the game's stages of development and marks the end of LCR V.2.

**Version 7 Additions:**

These are copies of version 7 that contain additional concepts that we learned. More of what we learned in chapters 7-8 are covered here in these additional versions.

**Version 7 - Game Begin - 2D Array**

This version took what we had and threw it into a 2-Dimensional array. The array had 2 columns: the first would hold the token counters and the second would hold the sorted copy of the token counters column. This second column would then be used in the ranking functions. Aside from the changes I made to the arrays, I also had to make various changes to the functions I wrote to accommodate the new changes for a 2D array. The previous version was carried over and changed into a 2D array, but nothing else was changed with respect to the game's functionality and function ideas. It is in this version where I take a 2D array as arguments.

**Version 7 - Game Begin - Vectors**

This version, similar to the 2D array version, takes version 7 and implements vectors instead of arrays. The actual code wasn't touched or changed as far as what I had in version 7. The only changes I made to this version were those of syntax where I brought in the vector header file and tweaked the function definitions to take those vectors and work with them. It is in this version where I take vectors as arguments.

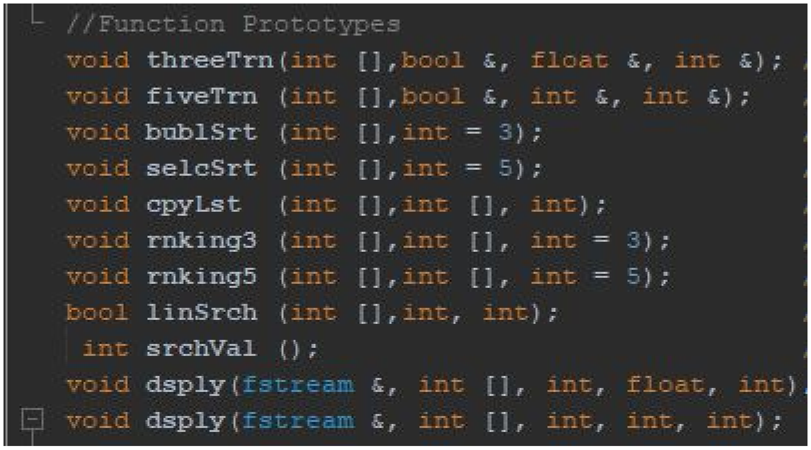
**Version 7 - Game Begin - Arrays & Vectors**

In case the two previous versions weren't enough to meet the requirements, I used all of the above in this version. For whatever dealt with three players, I used vectors and those functions took vectors as arguments. For whatever dealt with five players, I used a 2-Dimensional array that was used as arguments for the five player functions. As for the single dimensional array, I used an array of strings to hold the player's names and when the winner is displayed, the name would be used instead. This version would be a great example of function overloading because I had to write some functions that performed the same thing, but dealt with different parameters. For example, the copy function I had was made into two functions, one for vectors and one for working with a 2D array.

**Version 7 - Game Begin - Pointers and Dynamic Memory**

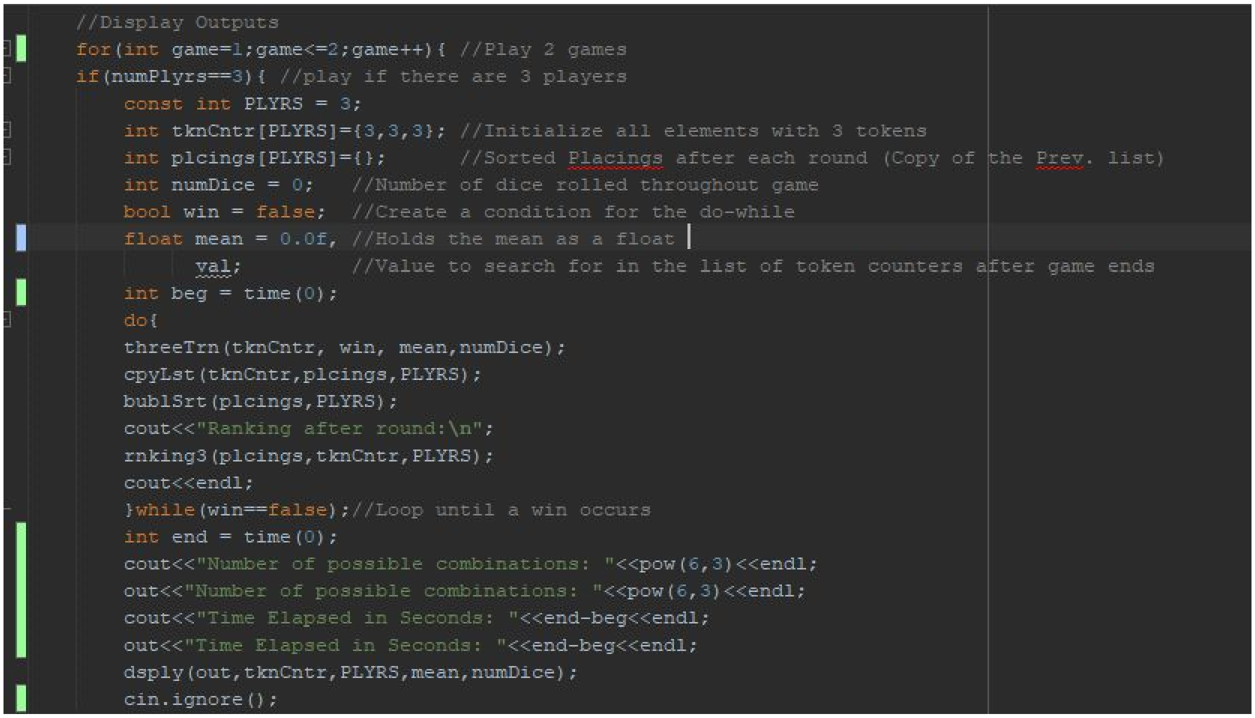
This version was an extra that wasn't required, but done for practice. In this version, I used concepts covered in chapter 9 of Gaddis, specifically pointers and dynamic memory allocation. I used the new keyword to dynamically create arrays to be used, and passed in pointers into my functions. Since pointers work like arrays, the code itself didn't need to be changed, only the syntax. All I did was tweak version 7 to work with pointers and dynamic arrays.

# Sample Screenshots:



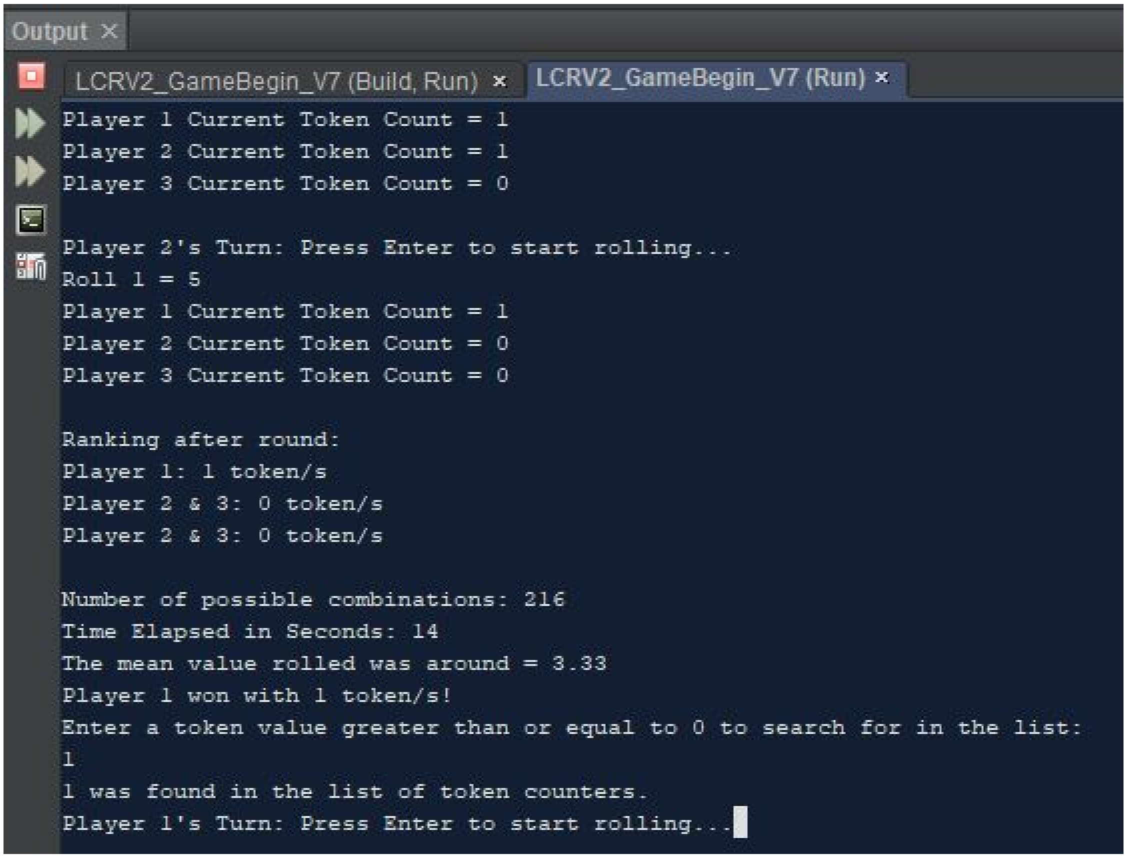
```
//Function Prototypes
void threeTrn(int [],bool &, float &, int &);
void fiveTrn (int [],bool &, int &, int &);
void bublSrt (int [],int = 3);
void selcSrt (int [],int = 5);
void cpyLst  (int [],int [], int);
void rnking3 (int [],int [], int = 3);
void rnking5 (int [],int [], int = 5);
bool linSrch (int [],int, int);
 int srchVal ();
void dsply(fstream &, int [], int, float, int)
void dsply(fstream &, int [], int, int, int);
```

An image of all the function prototypes I used throughout this single program.
An example of function overloading at the bottom can be seen where I have two dsply functions.



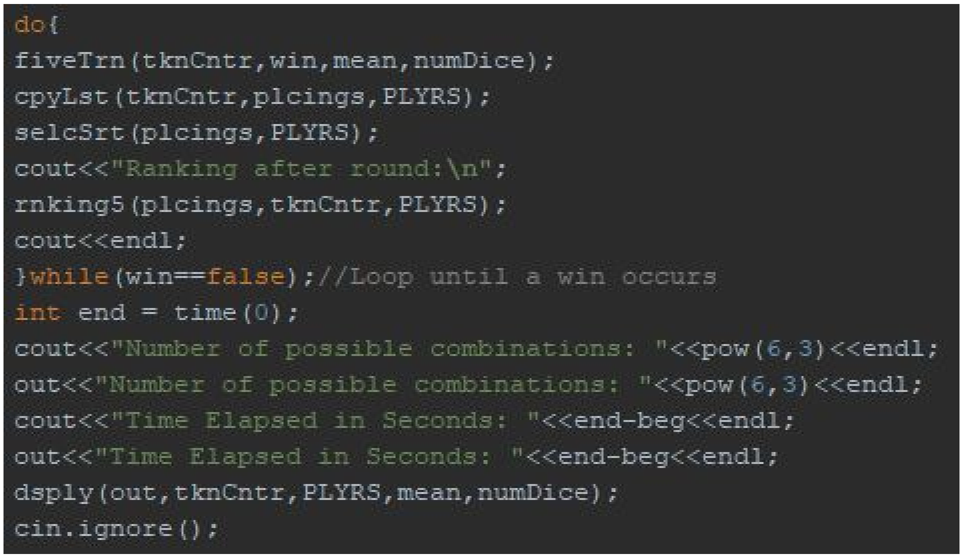
```
//Display Outputs
for(int game=1;game<=2;game++){ //Play 2 games
if(numPlyrs==3){ //play if there are 3 players
    const int PLYRS = 3;
    int tknCntr[PLYRS]={3,3,3}; //Initialize all elements with 3 tokens
    int plcings[PLYRS]={};      //Sorted Placings after each round (Copy of the Prev. list)
    int numDice = 0;   //Number of dice rolled throughout game
    bool win = false;  //Create a condition for the do-while
    float mean = 0.0f, //Holds the mean as a float
          val;         //Value to search for in the list of token counters after game ends
    int beg = time(0);
    do{
    threeTrn(tknCntr, win, mean,numDice);
    cpyLst(tknCntr,plcings,PLYRS);
    bublSrt(plcings,PLYRS);
    cout<<"Ranking after round:\n";
    rnking3(plcings,tknCntr,PLYRS);
    cout<<endl;
    }while(win==false);//Loop until a win occurs
    int end = time(0);
    cout<<"Number of possible combinations: "<<pow(6,3)<<endl;
    out<<"Number of possible combinations: "<<pow(6,3)<<endl;
    cout<<"Time Elapsed in Seconds: "<<end-beg<<endl;
    out<<"Time Elapsed in Seconds: "<<end-beg<<endl;
    dsply(out,tknCntr,PLYRS,mean,numDice);
    cin.ignore();
```

An image of the three turn function as well as other functions being used to carry out one game.
Each game reinitializes the arrays appropriately, so the game can repeat.



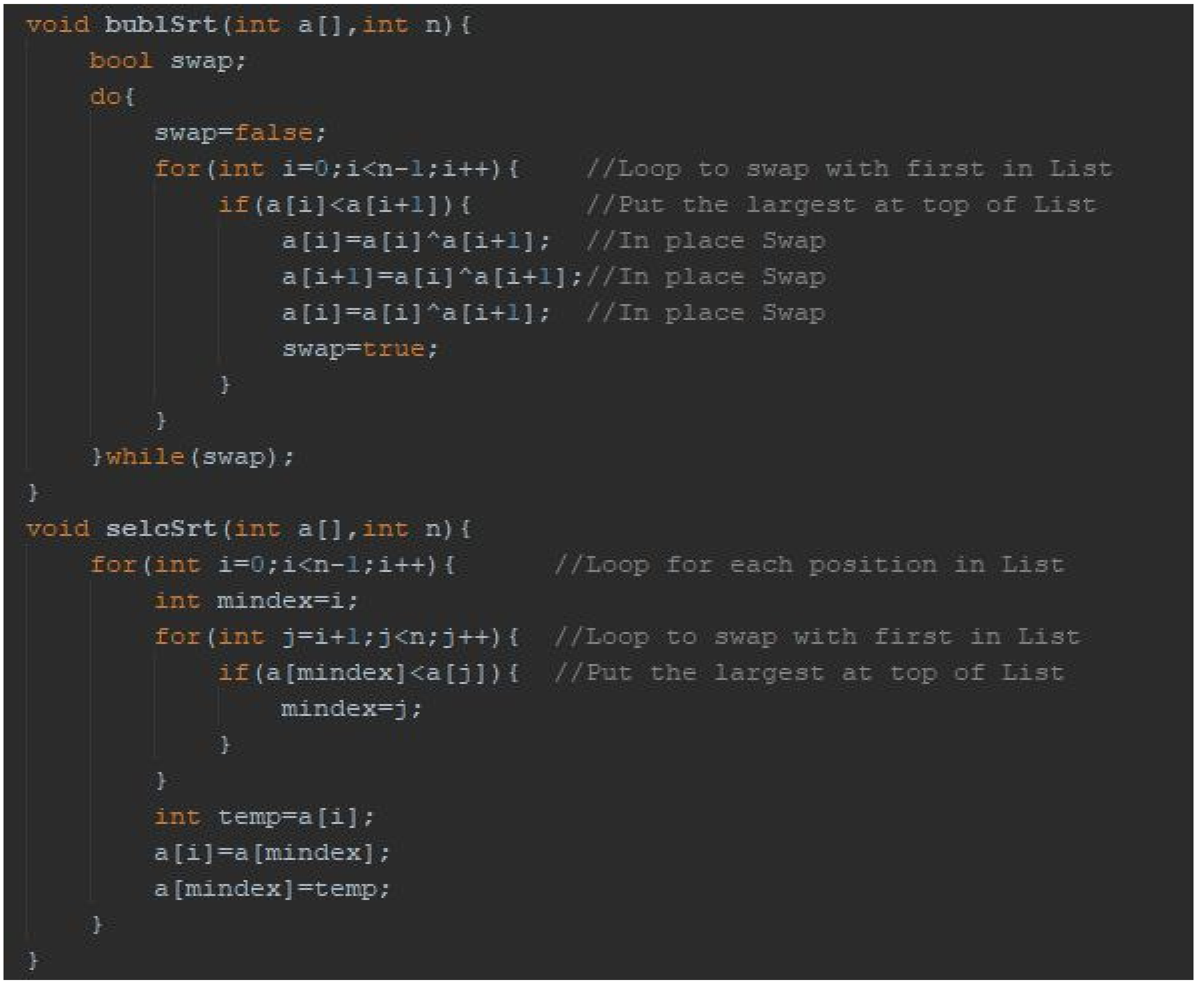
```
Player 1 Current Token Count = 1
Player 2 Current Token Count = 1
Player 3 Current Token Count = 0

Player 2's Turn: Press Enter to start rolling...
Roll 1 = 5
Player 1 Current Token Count = 1
Player 2 Current Token Count = 0
Player 3 Current Token Count = 0

Ranking after round:
Player 1: 1 token/s
Player 2 & 3: 0 token/s
Player 2 & 3: 0 token/s

Number of possible combinations: 216
Time Elapsed in Seconds: 14
The mean value rolled was around = 3.33
Player 1 won with 1 token/s!
Enter a token value greater than or equal to 0 to search for in the list:
1
1 was found in the list of token counters.
Player 1's Turn: Press Enter to start rolling...
```

An image of one complete game with displayed information is shown. The terminal also displays the rankings after each round or after everyone has had their turn. An example of the linear search is also present which is used to search the token counters array for a value of the user's choice.



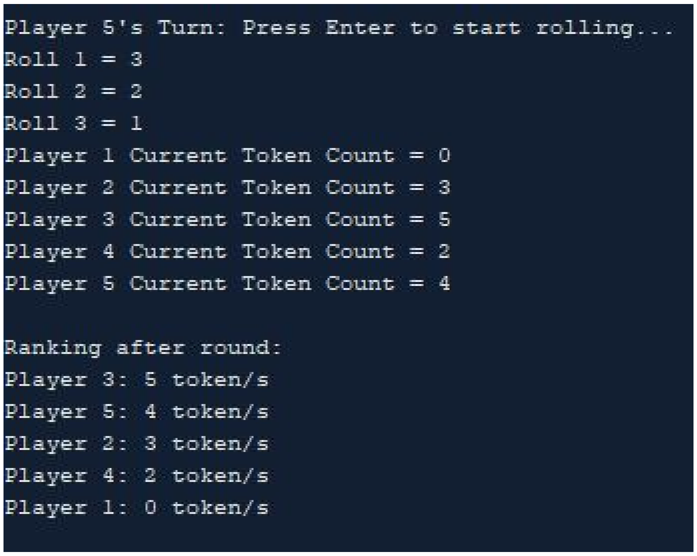
```
do{
fiveTrn(tknCntr,win,mean,numDice);
cpyLst(tknCntr,plcings,PLYRS);
selcSrt(plcings,PLYRS);
cout<<"Ranking after round:\n";
rnking5(plcings,tknCntr,PLYRS);
cout<<endl;
}while(win==false);//Loop until a win occurs
int end = time(0);
cout<<"Number of possible combinations: "<<pow(6,3)<<endl;
out<<"Number of possible combinations: "<<pow(6,3)<<endl;
cout<<"Time Elapsed in Seconds: "<<end-beg<<endl;
out<<"Time Elapsed in Seconds: "<<end-beg<<endl;
dsply(out,tknCntr,PLYRS,mean,numDice);
cin.ignore();
```

An image of the five player alternative is shown. Notice that the functions are different.



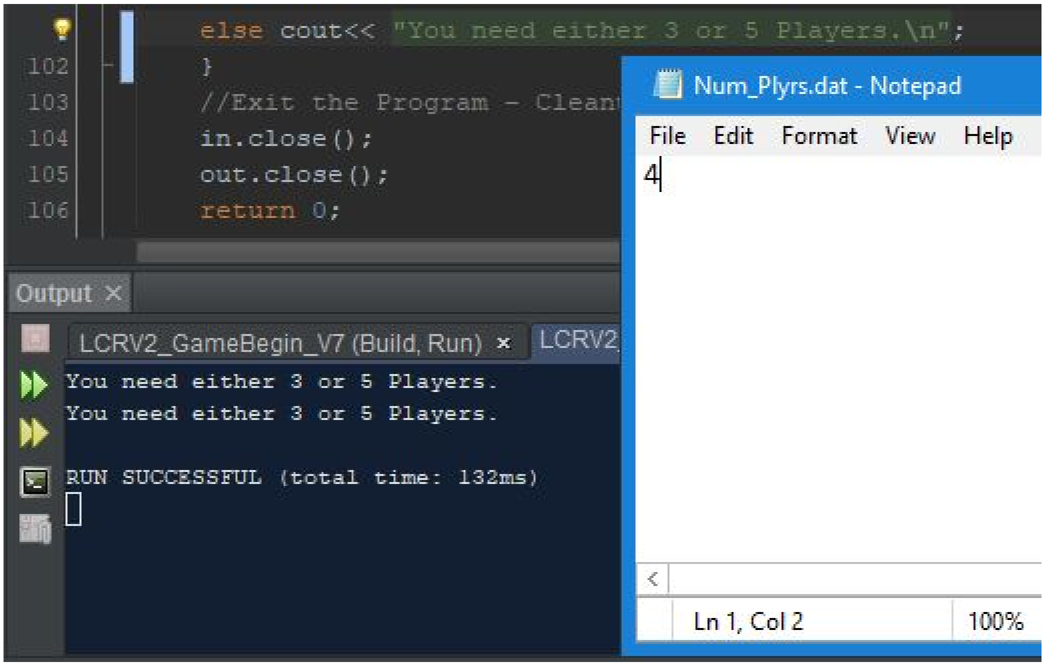
```
void bublSrt(int a[],int n){
    bool swap;
    do{
        swap=false;
        for(int i=0;i<n-1;i++){    //Loop to swap with first in List
            if(a[i]<a[i+1]){       //Put the largest at top of List
                a[i]=a[i]^a[i+1];  //In place Swap
                a[i+1]=a[i]^a[i+1];//In place Swap
                a[i]=a[i]^a[i+1];  //In place Swap
                swap=true;
            }
        }
    }while(swap);
}
void selcSrt(int a[],int n){
    for(int i=0;i<n-1;i++){      //Loop for each position in List
        int mindex=i;
        for(int j=i+1;j<n;j++){  //Loop to swap with first in List
            if(a[mindex]<a[j]){  //Put the largest at top of List
                mindex=j;
            }
        }
        int temp=a[i];
        a[i]=a[mindex];
        a[mindex]=temp;
    }
}
```

An image of the two sort functions written out. These were functions that we wrote in class that were brought over to my program.



```
Player 5's Turn: Press Enter to start rolling...
Roll 1 = 3
Roll 2 = 2
Roll 3 = 1
Player 1 Current Token Count = 0
Player 2 Current Token Count = 3
Player 3 Current Token Count = 5
Player 4 Current Token Count = 2
Player 5 Current Token Count = 4

Ranking after round:
Player 3: 5 token/s
Player 5: 4 token/s
Player 2: 3 token/s
Player 4: 2 token/s
Player 1: 0 token/s
```

A better example of the ranking being displayed.

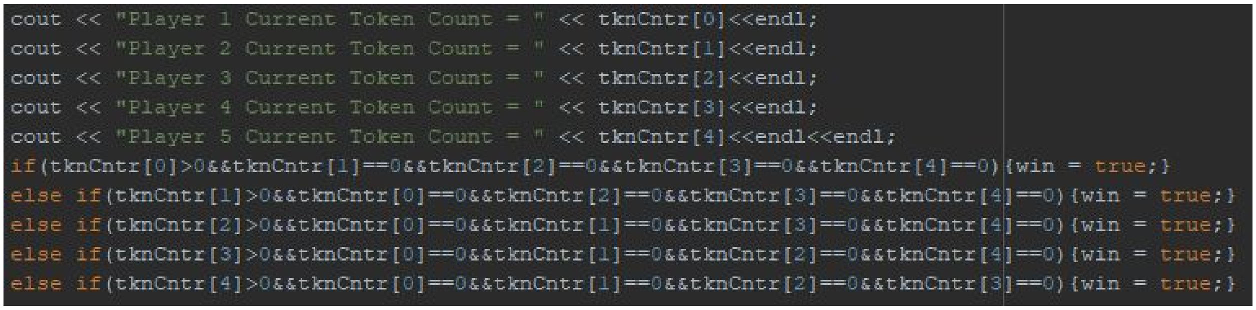


```
        else cout<< "You need either 3 or 5 Players.\n";
    }
    //Exit the Program - Clean
    in.close();
    out.close();
    return 0;
```

```
You need either 3 or 5 Players.
You need either 3 or 5 Players.

RUN SUCCESSFUL (total time: 132ms)
```

```
4
```

An example of a failed game run is shown. The file held 4 when only 3 or 5 could be used to play the game. Since there were two games, the error message printed twice.



```
cout << "Player 1 Current Token Count = " << tknCntr[0]<<endl;
cout << "Player 2 Current Token Count = " << tknCntr[1]<<endl;
cout << "Player 3 Current Token Count = " << tknCntr[2]<<endl;
cout << "Player 4 Current Token Count = " << tknCntr[3]<<endl;
cout << "Player 5 Current Token Count = " << tknCntr[4]<<endl<<endl;
if(tknCntr[0]>0&&tknCntr[1]==0&&tknCntr[2]==0&&tknCntr[3]==0&&tknCntr[4]==0){win = true;}
else if(tknCntr[1]>0&&tknCntr[0]==0&&tknCntr[2]==0&&tknCntr[3]==0&&tknCntr[4]==0){win = true;}
else if(tknCntr[2]>0&&tknCntr[0]==0&&tknCntr[1]==0&&tknCntr[3]==0&&tknCntr[4]==0){win = true;}
else if(tknCntr[3]>0&&tknCntr[0]==0&&tknCntr[1]==0&&tknCntr[2]==0&&tknCntr[4]==0){win = true;}
else if(tknCntr[4]>0&&tknCntr[0]==0&&tknCntr[1]==0&&tknCntr[2]==0&&tknCntr[3]==0){win = true;}
```

The image shows how the token counters are shown in code. This could've been done using a for loop, but it works nonetheless. Below is the nested ternary operator used to test for a win. Each player's count is tested and a win occurs when only one remains with tokens.

# Project 2: Check-Off Sheet

| Chap. | Sect. | Topic | Where Line #"s | Pts | Notes |
|---|---|---|---|---|---|
| 2 | 2 | cout | | | |
| | 3 | libraries | Begins @ Line 10 | 5 | iostream, iomanip, cmath, cstdlib, fstream, string, ctime |
| | 4 | variables/literals | | | No variables in global area, failed project! |
| | 5 | Identifiers | | | |
| | 6 | Integers | Line 45 | 1 | |
| | 7 | Characters | Line 111 | 1 | |
| | 8 | Strings | Line 43 | 1 | Character String ~File Name |
| | 9 | Floats No Doubles | Line 59 | 1 | Using doubles will fail the project, floats OK! |
| | 10 | Bools | Line 58 | 1 | |
| | 11 | Sizeof ***** | | | |
| | 12 | Variables 7 characters or less | | | All variables <= 7 characters |
| | 13 | Scope ***** No Global Variables | | | |
| | 14 | Arithmetic operators | | | |
| | 15 | Comments 20%+ | Through out Game | 2 | Model as pseudo code |
| | 16 | Named Constants | | | All Local, only Conversions/Physics/Math in Global area |
| | 17 | Programming Style ***** Emulate | | | Emulate style in book/in class repositiory |

| | | | | | |
|---|---|---|---|---|---|
| 3 | 1 | cin | | | |
| | 2 | Math Expression | | | |
| | 3 | Mixing data types **** | | | |
| | 4 | Overflow/Underflow **** | | | |
| | 5 | Type Casting | Line 121 | 1 | |
| | 6 | Multiple assignment ***** | | | |
| | 7 | Formatting output | Line 529 | 1 | |
| | 8 | Strings | Line 44 | 1 | String Object ~File Name |
| | 9 | Math Library | Line 71 | 1 | All libraries included have to be used |
| | 10 | Hand tracing ****** | | | |
| 4 | 1 | Relational Operators | | | |
| | 2 | if | Line 112 | 1 | Independent if |
| | 4 | If-else | Lines 116/126 | 1 | |
| | 5 | Nesting | Lines 112-124 | 1 | |
| | 6 | If-else-if | Lines 122-124 | 1 | |
| | 7 | Flags ***** | | | |
| | 8 | Logical operators | Lines 140-142 | 1 | |
| | 11 | Validating user input | Line 49 | 1 | |
| | 13 | Conditional Operator | Line 49 | 1 | |
| | 14 | Switch | Line 155 | 1 | |

| | | | | | |
|---|---|---|---|---|---|
| 5 | 1 | Increment/Decrement | Line 156 | 1 | |
| | 2 | While | Line 48 | 1 | |
| | 5 | Do-while | Line 62 | 1 | |
| | 6 | For loop | Line 117 | 1 | |
| | 11 | Files input/output both | Lines 47-49+75 | 2 | Dsply function sends output to file |
| | 12 | No breaks in loops ****** | | | Failed Project if included |
| ***** Not req. to show | | | Total | 30 | |
| Chap. | Sect. | Topic | Where Line #"s | Pts | Notes |
| 6 | | Functions | | | |
| | 3 | Function Prototypes | Lines 25-35 | 4 | Always use prototypes |
| | 5 | Pass by Value | Line 75 | 4 | Ex. Passing the mean into Dsply function |
| | 8 | return | Lines 33+542 | 4 | A value from a function |
| | 9 | returning boolean | Lines 32+543 | 4 | Returns a Bool |
| | 10 | Global Variables | | XXX | Do not use global variables -100 pts |
| | 11 | static variables | Begins @ Line 41 | 4 | |
| | 12 | defaulted arguments | Lines 27-28 | 4 | |

| | | | | | |
|---|---|---|---|---|---|
| | 13 | pass by reference | Line 63 | 4 | Passing an Array |
| | 14 | overloading | Lines 34-35 | 5 | |
| | 15 | exit() function | Line 523 | 4 | |
| 7 | | Arrays | | | |
| | 1 to 6 | Single Dimensioned Arrays | Line 55 | 3 | |
| | 7 | Parallel Arrays | Lines 55-56 | 2 | |
| | 8 | Single Dimensioned as Function Arguments | Line 64 | 2 | |
| | 9 | 2 Dimensioned Arrays | Line 53 | 2 | Found in Another Variation of Version 7 -> V7_2DArray |
| | 12 | STL Vectors | Line 52 | 2 | Found in Another Variation of Version 7 -> V7_Vectors |
| | | Passing Arrays to and from Functions | Lines 63-67 | 5 | |
| | | Passing Vectors to and from Functions | Lines 59-63 | 5 | Found in Another Variation of Version 7 -> V7_Vectors |
| 8 | | Searching and Sorting Arrays | | | |
| | 3 | Bubble Sort | Line 65 | 4 | |
| | 3 | Selection Sort | Line 89 | 4 | |
| | 1 | Linear or Binary Search | Line 543 | 4 | |
| ****** Not req. to show | | | Total | 70 | Other 30 points from Proj 1 first sheet tab |