**Project 2**

**<Yahtzee>**

**CIS-17A-48290**

**Name: Gillam, Sean**

**Date: 11/2/2022**

**Introduction**

Title: Yahtzee

This is a dice rolling game where you are trying to roll set patterns with the dice. You roll 5 dice on the first iteration and then on subsequent rolls you may choose any number of the dice to reroll. However, you only get 2 sets of rerolls. The game has multiple categories to aim for including number of each dice face, three of a kind, four of a kind, small straight(4 dice in a row), large straight(5 dice in a row), full house(3 of a kind and 2 of a kind), 5 of a kind (which is called a Yahtzee, and chance(total of all dice.

For the number of a particular dice face patterns, you just add the like faces so for 2 dice that rolled a 3 their score is worth 6 in the threes category. For the “of Kind” sets you simply add all the dice if the criteria for the category is met. For small straight you get 30 points, for a large straight you get 40 points, and for a Yahtzee you get 50 points. The points are totaled at the end. If you don’t meet a category for points, then that category is worth zero. You can only select a category once per game.

There are bonus point systems in place, however they do not exist in all versions of the game and are not included in this program.

I chose this game because it is something my grandmother and I used to play.

**Summary**

Project size: around 1000 if you take out the whitespace.

# Variables: ~50

# Functions: 13 excluding the main.

This game meets the criteria for this project because it is both a board game and has some interesting yet complicated rules. Performing the scoring automatically for the player each needed a different set of code in order to perform a check. In addition rerolling particular dice and not the whole set further adds to the complexity of the game code.

This project took me a few days to complete working on it on and off. Towards the end of this class I began to lose focus so this project is missing a lot of the requirements. I did the best I could with what I understood of classes.

**Description**

This program was developed largely using multiple functions in addition to structures. The main function contains a menu which allows a player to choose to continue a game that already existed, start a new game, or change the setting for the number of players.

Text

Description automatically generated

Once you press 0 and enter for new game the following will display

Text

Description automatically generated

The program will take in a player’s name and then using a cstring pull it apart to get the first 3 letters to us as initials for the player. I had planned to use this to further make a score board, but I never had time to implement it.

From there the game rolls dice for the first player and asks how many they would like to reroll

Text

Description automatically generated

The player can reroll up to 5 dice up to 2 times. If a player enters zero the program will stop rolling dice and ask a player to choose a point category to store the points in (if any points were earned).

Text

Description automatically generated

Once a player chooses a point category the game will go to the next player. In this case there is only one player so the program simply loops. If a player chooses to exit the game will save and return to the main menu.

Text

Description automatically generated

A player can see their totaled points from previous rounds.

Text

Description automatically generated

Input validation is being considered so a player can only choose a category once.

Graphical user interface, text, application, website

Description automatically generated

Once all 13 categories have been filled the game ends and totals the points for the player.

Text

Description automatically generated

Going to the settings menu asks the player if they want to change the number of players

Text

Description automatically generated

The code allows for any number of players to be input and it is saved to a file for next time the game is opened.

Text

Description automatically generated

If the scoreboard option is selected it will pull the current scoreboard from the file.

Text

Description automatically generated

**Flowchart**

Diagram

Description automatically generated

Diagram

Description automatically generated with medium confidence

Shape, polygon

Description automatically generated

Diagram

Description automatically generated

Shape, polygon

Description automatically generated

**Pseudo Code**

*Bring in user libraries*

*Declare Namespace*

*Initialize global constants*

*Initialize enum for main menu*

*Initialize enum for settings menu*

*Initialize Nested Data Struct*

*Initialize Player Struct*

*Function prototypes*

*Enter Main*

*Declare random seed for dice rolling*

*Initialize variables and filestream*

*Do*

*Open file stream*

*If it fails output an error*

*Else*

*Input file into variable for number of players*

*Close the file*

*Call to menu function to display menu*

*Take in user choice*

*Choices options*

*Case play game results in a call to the game function with the the continue game flag set to false*

*Case continue game calls to game function with the flag set to true*

*Case settings is a function call to settings menu*

*Case scoreboard call highScore function*

*While choice is greater than 0 and less than 2*

*Exit main*

*Enter menu*

*Output menu options to console*

*Exit menu*

*Enter highScores*

*Call Display Class*

*Print ScoreBoard*

*Exit highScores*

*Enter game*

*Initialize values and filestreams*

*If continue game flag is true*

*Load all save information from files*

*Declare dynamic structure*

*If continue game flag is true*

*Fill dynamic structure with data from game file*

*If continue game is false*

*Ask player name*

*Get player initials using initials function*

*Set all initial game values*

*Do*

*Roll player dice*

*Ask for how many dice to reroll*

*Reroll that number of dice and repeat until total rolling is 3 iterations*

*Output scoreboard for individual player and ask player for point selection*

*The player can only select a category they haven’t chosen yet*

*Using a switch statement allow for each option to have its own logic*

*If ones count ones*

*If twos count twos*

*If threes count threes*

*If fours count fours*

*If fives count fives*

*If sixes count sixes*

*If three of a kind check for that condition*

*If four of a kind check for that condition*

*If full house*

*Count each of the numbers on the dice.*

*Check for 3 of one kind and 2 of another*

*If small straight*

*Check for 1234*

*Check for 2345*

*Check for 3456*

*If large straight*

*Check for 12345*

*Check for 23456*

*If Yahtzee check for 5 of a kind*

*If chance add up all the dice*

*Iterate player counter unless it is more than the number of players then reset it*

*If end of game phase tally points and output and then exit the game loop*

*Else ask player if they would like to save the game and exit. If not not then continue the loop*

*While exit flag is false loop*

*Delete the pointer declaration for player structure*

*Exit game function*

*Enter settings function*

*Initialize values*

*Allow user to change number of players*

*Save choice to file for next play*

*Exit settings function*

*Enter roll dice function*

*Copy player structure*

*Roll each dice using a random number*

*Exit roll dice function*

*Enter roll function*

*Reroll individual dice using random numbers*

*Exit roll function*

*Enter char getInitials function*

*Return char*

*Exit function*

*Enter searchNum function*

*Search for the number using a for loop*

*Return true if found*

*Otherwise return false*

**Major Variables**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | Variable Name | Description | Location |
| **Integer** | fullOne | Holds number of ones found in a fullhouse check loop | struct Data  switch statement inside of game() |
|  | fullTwo | Holds number of twos found in a fullhouse check loop | struct Data  switch statement inside of game() |
|  | fullThree | Holds number of threes found in a fullhouse check loop | struct Data  switch statement inside of game() |
|  | fullFour | Holds number of fours found in a fullhouse check loop | struct Data  switch statement inside of game() |
|  | fullFive | Holds number of fives found in a fullhouse check loop | struct Data  switch statement inside of game() |
|  | fullSix | Holds number of sixes found in a fullhouse check loop | struct Data  switch statement inside of game() |
|  | totalScore | Holds total score for display at the end of a game | struct Player  game(), used when play of the game is over at the end of the game loop |
|  | dice[] | holds the value of dice for any given player | Struct Player  game()  rollDice()  roll() |
|  | ones | points for ones | Struct Player  game() |
|  | twos | points for twos | Struct Player  game() |
|  | threes | points for threes | Struct Player  game() |
|  | fours | points for fours | Struct Player  game() |
|  | fives | points for fives | Struct Player  game() |
|  | sixes | points for sixes | Struct Player  game() |
|  | threeKind | points for three of a kind | Struct Player  game() |
|  | fourKind | points for four of a kind | Struct Player  game() |
|  | fullHouse | points for full house | Struct Player  game() |
|  | smallSrt | points for small straight | Struct Player  game() |
|  | largest | points for large straight | Struct Player  game() |
|  | Yahtzee | points for Yahtzee | Struct Player  game() |
|  | numPlayers | holds number of players | main()  game()  settings() |
|  | choice | allows menu choice from user | main() |
|  | playerCount | counts what player the game loop is on | game() |
|  | reroll | holds players choice on how many dice to reroll | game() |
|  | diceNum | hold players choice on which dice to reroll | game() |
|  | gameChoice | holds players choice on which point category to score for | game() |
|  | iterationCounter | Holds a number that counts down to track when game should end | game() |
|  | kindCounter | used to track how many of a like kind of a dice was found | game() |
|  | kindIt | kind iterator for dice numbers | game() |
| **Constant Integer** | NUM\_DICE | Number of dice in a game | Global |
|  | NUM\_INITIALS | Number of letters for initials to store | Global |
|  | CHANCES | Number of re-rolls allowed | Global |
|  | SCORES | Number of scoring options | Global |
| **String** | name | Holds player names | struct Player  game() |
| **Character** | initials[] | holds player initials | struct Player  game() |
| **bool** | flag[] | stores the flags for scoring conditions | struct player  game() |
|  | exit | checks for exit condition for game | game() |
|  | exitChoice | input validation for exit choice at end of game loop | game() |
| **menuChoices** | playgame | holds 0 for menu | enum |
|  | contGame | holds 1 for menu | enum |
|  | settings | holds 2 for menu | enum |
| **settingsChoice** | playerCount | holds 0 for menu | enum |
| **Player** | \*player | hold dynamic game structure | game() |
|  | \*p | holds dynamic game structure copy for return | rollDice() |
| **fstream** | numPlayersFile | filestream for numPlayers.dat | main()  game() |
|  | numSavePlayersFile | hold save games number of players or continue game condition | game() |
|  | gameFile | holds the structure of the game so that data can be reloaded later | game() |
|  | iterationFile | hold the number of iterations for game | game() |
|  | playerCountFile | tracks which player game is on for save games | game() |

**Concepts**

|  |  |  |
| --- | --- | --- |
| **Chapter** | **Concept** | **Location** |
| **13** | **Classes** |  |
|  | Instance of a Class | Display.h/scoreboard.h |
|  | Private Data Members | scoreboard.h |
|  | Specification vs. Implementation | scoreboard.h/scoreboard.cpp |
|  | Inline | scoreboard.h |
|  | Constructors | scoreboard.h |
|  | Destructors | scoreboard.h |
|  | Arrays of Objects |  |
|  | UML | Display.h |
| **14** | **More about classes** |  |
|  | Static | scoreboard.h |
|  | Friends | scoreboard.h/Display.h |
|  | Copy Constructors |  |
|  | Operator Overloading |  |
|  | Aggregation |  |
| **15** | **Inheritance** |  |
|  | Protected members | scoreboard.h |
|  | Base Class to Derived |  |
|  | Polymorphic associations |  |
|  | Abstract Classes |  |
| **16** | **Advanced Classes** |  |
|  | Exceptions |  |
|  | Templates |  |
|  | STL |  |

**Reference**

1. Textbook
2. Yahtzee Rule Book(see below)

Text, timeline

Description automatically generated

Table

Description automatically generated

A close-up of a document

Description automatically generated with medium confidenceA picture containing text, newspaper, receipt

Description automatically generatedBackground pattern

Description automatically generated with low confidenceA close-up of a document

Description automatically generated with low confidence

**Program**

|  |
| --- |
| /\*  \* File: main.cpp  \* Author: Sean Gillam  \*  \* Created on October 28, 2022, 6:23 PM  \*/  #include <iostream> //For user in/output  #include <ctime> //For random dice rolls  #include <cstdlib> //Standard Library  #include <cstring> //For Players Initials  #include <string> //For Players Name  #include <fstream> //For save games  #include "scoreboard.h"  #include "Display.h"  using namespace std;  //Global Constants  const int NUM\_DICE = 5; // Number of dice in a game  const int NUM\_INITIALS = 3; // Number of letters for initials to store  const int CHANCES = 3; // Number of re-rolls allowed  const int SCORES = 13; //Number of scoring options  //For menu choices  enum menuChoices  {  playgame,  contGame,  settings,  scoreb  };  //For settings choices  enum settingsChoices  {  playerCount  };  //Structure for checking full house condition  struct Data  {  int fullOne;  int fullTwo;  int fullThree;  int fullFour;  int fullFive;  int fullSix;  };  //Player data for file storage  struct Player  {  //Player Details  string name;  char initials[NUM\_INITIALS];  //Overall trackers  int totalScore;  int dice[NUM\_DICE];  //Score trackers  int ones;  int twos;  int threes;  int fours;  int fives;  int sixes;  int threeKind;  int fourKind;  int fullHouse;  int smallSrt;  int largeSrt;  int yahtzee;  int chance;  //Boolean flags so scores cant be double counted  bool flag[SCORES];  //Nested structure  Data full;  };  //Menu Option Prototypes  void menu();  void game(int,bool);  void setting(fstream&);  void highScores();  //Supporting function prototypes  Player\* rollDice(Player \*,int,bool,bool,bool,bool,bool);  void roll(Player \*,int,int);  char getInitials(Player \*,int,int);  void dispDice(Player \*,int);  bool searchNum(Player \*,int,int);  int main(int argc, char\*\* argv)  {  //Setting random seed  srand(static\_cast<unsigned int>(time(0)));    //numPlayer declarations  fstream numPlayersFile;  int numPlayers;    //Menu Do While Loop  int choice;  do  {  //Read in numPlayers data (Important for save game)  numPlayersFile.open("numPlayers.dat", ios::in);  if(!numPlayersFile)  cout<<"ERROR: Reading Player Count";  else  numPlayersFile>>numPlayers;  numPlayersFile.close();    //begin menu  menu();  cin>>choice;  cin.ignore();    switch(choice)  {  case playgame : {game(numPlayers,false);break;}  case contGame : {game(numPlayers,true);break;}  case settings : {setting(numPlayersFile);break;}  case scoreb : {highScores();break;}  default: cout<<endl<<endl<<"Exiting menu. Good Bye!";  }  }while(choice>=0&&choice<=3);    //Exit Stage Right  return 0;  }  void menu()  {  cout<<endl<<endl;  cout<<"Main Menu:"<<endl;  cout<<"Type 0 for new game."<<endl;  cout<<"Type 1 to continue game."<<endl;  cout<<"Type 2 for settings."<<endl;  cout<<"Type 3 for scoreboard"<<endl;  cout<<"Type any other number to quit game."<<endl;  }  void highScores()  {  Display hold;  hold.printBoard();  }  void game(int numPlayers, bool cgame)  {      int playerCount=0;  bool exit;  int reroll;  int diceNum;  int gameChoice;  bool exitChoice;  int iterationCounter;  int kindCounter;  int kindIt;    fstream numSavePlayersFile, gameFile, iterationFile, playerCountFile,  numPlayersFile;    //Reset Player number to save game size if cgame flag is true  if(cgame)  {  numSavePlayersFile.open("numSavePlayers.dat", ios::in);  numPlayersFile.open("numPlayers.dat", ios::out);  numSavePlayersFile>>numPlayers;  int temp = numPlayers;  numPlayersFile << temp;    iterationFile.open("iteration.dat", ios::in);  iterationFile>>iterationCounter;    playerCountFile.open("playerCount.dat", ios::in);  playerCountFile>>playerCount;    numSavePlayersFile.close();  numPlayersFile.close();  iterationFile.close();  playerCountFile.close();  }    Player \*player = new Player[numPlayers];    //Resetting structure from file if continue game  if(cgame)  {  gameFile.open("game.bin",ios::in|ios::binary);  for(int i=0;i<numPlayers;i++)  gameFile.read(reinterpret\_cast<char \*>(&player[i]),sizeof(Player));  gameFile.close();  }    //If not new game then set all starting conditions  if(!cgame)  {  //Getting Players names and initials  cout<<"\n\nStarting new game...\n\n";  for(int i=0;i<numPlayers;i++)  {  cout<<"Hello player #"<<i+1<<endl;  cout<<"Please enter name: ";  getline(cin,player[i].name);  for(int j=0;j<NUM\_INITIALS;j++)  player[i].initials[j]=getInitials(player,i,j);  cout<<"Hello "<<player[i].name<<" your initials for the scoreboard"  " will be: "<<player[i].initials<<endl;  }    //Settings initial values for all players  for(int i=0;i<numPlayers;i++)  {  player[i].totalScore=0;  player[i].ones=0;  player[i].twos=0;  player[i].threes=0;  player[i].fours=0;  player[i].fives=0;  player[i].sixes=0;  player[i].threeKind=0;  player[i].fourKind=0;  player[i].fullHouse=0;  player[i].smallSrt=0;  player[i].largeSrt=0;  player[i].yahtzee=0;  player[i].chance=0;  player[i].full.fullOne=0;  player[i].full.fullTwo=0;  player[i].full.fullThree=0;  player[i].full.fullFour=0;  player[i].full.fullFive=0;  player[i].full.fullSix=0;  for(int j=0;j<SCORES;j++)  player[i].flag[j]=false;  }    //Setting for new game how many times we will loop  iterationCounter = numPlayers\*13;  }        do  {  exit=false;    //Starting Dice rolling part of game loop  cout<<"Player "<<player[playerCount].initials<<"'s Turn!"<<endl;  player = rollDice(player,playerCount,1,1,1,1,1);  for(int i=0;i<CHANCES;i++)  {  cout<<"Roll "<<i+1<<":"<<endl;  if(i<CHANCES-1)  {  do  {  dispDice(player,playerCount);  cout<<"How Many dice would you like to reroll(0-5)?: ";  cin>>reroll;  cin.ignore();  if(reroll<0||reroll>5)  cout<<"Invalid number of dice!"<<endl;  }while(reroll<0||reroll>5);  if(reroll==0)  i=CHANCES-1; //Set to big number to escape  else  {  cout<<"Which Dice would you like to reroll?"<<endl;  for(int j=0;j<reroll;j++)  {  do  {  cout<<"Dice #?: ";  cin>>diceNum;  cin.ignore();  if(diceNum<1||diceNum>5)  cout<<"Invalid!"<<endl;  }while(diceNum<1||diceNum>5);  roll(player,playerCount,diceNum);  }  }  }  else  dispDice(player,playerCount);  }    //Outputing current player  cout<<player[playerCount].initials<<"'s Score Card:"<<endl;  if(player[playerCount].flag[0])  cout<<"1: Ones: "<<player[playerCount].ones<<endl;  else  cout<<"1: Ones: "<<endl;  if(player[playerCount].flag[1])  cout<<"2: Twos: "<<player[playerCount].twos<<endl;  else  cout<<"2: Twos: "<<endl;  if(player[playerCount].flag[2])  cout<<"3: Threes: "<<player[playerCount].threes<<endl;  else  cout<<"3: Threes: "<<endl;  if(player[playerCount].flag[3])  cout<<"4: Fours: "<<player[playerCount].fours<<endl;  else  cout<<"4: Fours: "<<endl;  if(player[playerCount].flag[4])  cout<<"5: Fives: "<<player[playerCount].fives<<endl;  else  cout<<"5: Fives: "<<endl;  if(player[playerCount].flag[5])  cout<<"6: Sixes: "<<player[playerCount].sixes<<endl;  else  cout<<"6: Sixes: "<<endl;  if(player[playerCount].flag[6])  cout<<"7: Three of a Kind: "<<player[playerCount].threeKind<<endl;  else  cout<<"7: Three of a Kind: "<<endl;  if(player[playerCount].flag[7])  cout<<"8: Four of a Kind: "<<player[playerCount].fourKind<<endl;  else  cout<<"8: Four of a Kind: "<<endl;  if(player[playerCount].flag[8])  cout<<"9: Full House: "<<player[playerCount].fullHouse<<endl;  else  cout<<"9: Full House: "<<endl;  if(player[playerCount].flag[9])  cout<<"10: Small Straight: "<<player[playerCount].smallSrt<<endl;  else  cout<<"10: Small Straight: "<<endl;  if(player[playerCount].flag[10])  cout<<"11: Large Straight: "<<player[playerCount].largeSrt<<endl;  else  cout<<"11: Large Straight: "<<endl;  if(player[playerCount].flag[11])  cout<<"12: Yahtzee: "<<player[playerCount].yahtzee<<endl;  else  cout<<"12: Yahtzee: "<<endl;  if(player[playerCount].flag[12])  cout<<"13: Chance: "<<player[playerCount].chance<<endl;  else  cout<<"13: Chance: "<<endl;    //Ask for player input  do  {  cout<<"Please select an option too score your dice(1-13): ";  cin>>gameChoice;  cin.ignore();  if(gameChoice<1||gameChoice>13)  cout<<"Invalid!"<<endl;  else if(player[playerCount].flag[gameChoice-1])  {  cout<<"Already Scored!"<<endl;  gameChoice=-1;  }  }while(gameChoice<1||gameChoice>13);    //Switch statement handling scoring by player choice.  switch(gameChoice)  {  case 1 :  player[playerCount].flag[0]=true;  for(int i=0;i<NUM\_DICE;i++)  if(player[playerCount].dice[i]==1)  player[playerCount].ones++;  break;  case 2 :  player[playerCount].flag[1]=true;  for(int i=0;i<NUM\_DICE;i++)  if(player[playerCount].dice[i]==2)  player[playerCount].twos++;  player[playerCount].twos=player[playerCount].twos\*2;  break;  case 3 :  player[playerCount].flag[2]=true;  for(int i=0;i<NUM\_DICE;i++)  if(player[playerCount].dice[i]==3)  player[playerCount].threes++;  player[playerCount].threes=player[playerCount].threes\*3;  break;  case 4 :  player[playerCount].flag[3]=true;  for(int i=0;i<NUM\_DICE;i++)  if(player[playerCount].dice[i]==4)  player[playerCount].fours++;  player[playerCount].fours=player[playerCount].fours\*4;  break;  case 5 :  player[playerCount].flag[4]=true;  for(int i=0;i<NUM\_DICE;i++)  if(player[playerCount].dice[i]==5)  player[playerCount].fives++;  player[playerCount].fives=player[playerCount].fives\*5;  break;  case 6 :  player[playerCount].flag[5]=true;  for(int i=0;i<NUM\_DICE;i++)  if(player[playerCount].dice[i]==6)  player[playerCount].sixes++;  player[playerCount].sixes=player[playerCount].sixes\*6;  break;  case 7 :  player[playerCount].flag[6]=true;  kindIt=0;  do  {  kindCounter=0;  kindIt++;  for(int i=0;i<NUM\_DICE;i++)  if(player[playerCount].dice[i]==kindIt)  kindCounter++;  }while(kindCounter<3&&kindIt<7);  if(kindCounter>2)  for(int i=0;i<NUM\_DICE;i++)  player[playerCount].threeKind=  player[playerCount].threeKind+  player[playerCount].dice[i];  break;  case 8 :  player[playerCount].flag[7]=true;  kindIt=0;  do  {  kindCounter=0;  kindIt++;  for(int i=0;i<NUM\_DICE;i++)  if(player[playerCount].dice[i]==kindIt)  kindCounter++;  }while(kindCounter<4&&kindIt<7);  if(kindCounter>2)  for(int i=0;i<NUM\_DICE;i++)  player[playerCount].fourKind=  player[playerCount].fourKind+  player[playerCount].dice[i];  break;  case 9 :  player[playerCount].flag[8]=true;  for(int i=0;i<NUM\_DICE;i++)  {  if(player[playerCount].dice[i]==1)  player[playerCount].full.fullOne++;  else if(player[playerCount].dice[i]==2)  player[playerCount].full.fullTwo++;  else if(player[playerCount].dice[i]==3)  player[playerCount].full.fullThree++;  else if(player[playerCount].dice[i]==4)  player[playerCount].full.fullFour++;  else if(player[playerCount].dice[i]==5)  player[playerCount].full.fullFive++;  else if(player[playerCount].dice[i]==6)  player[playerCount].full.fullSix++;  }  if((player[playerCount].full.fullOne<2)&&(  (player[playerCount].full.fullTwo>1)||  (player[playerCount].full.fullThree>1)||  (player[playerCount].full.fullFour>1)||  (player[playerCount].full.fullFive>1)||  (player[playerCount].full.fullSix>1)))  player[playerCount].fullHouse=25;  else if((player[playerCount].full.fullTwo<2)&&(  (player[playerCount].full.fullOne>1)||  (player[playerCount].full.fullThree>1)||  (player[playerCount].full.fullFour>1)||  (player[playerCount].full.fullFive>1)||  (player[playerCount].full.fullSix>1)))  player[playerCount].fullHouse=25;  else if((player[playerCount].full.fullThree<2)&&(  (player[playerCount].full.fullOne>1)||  (player[playerCount].full.fullTwo>1)||  (player[playerCount].full.fullFour>1)||  (player[playerCount].full.fullFive>1)||  (player[playerCount].full.fullSix>1)))  player[playerCount].fullHouse=25;  else if((player[playerCount].full.fullFour<2)&&(  (player[playerCount].full.fullOne>1)||  (player[playerCount].full.fullTwo>1)||  (player[playerCount].full.fullThree>1)||  (player[playerCount].full.fullFive>1)||  (player[playerCount].full.fullSix>1)))  player[playerCount].fullHouse=25;  else if((player[playerCount].full.fullFive<2)&&(  (player[playerCount].full.fullOne>1)||  (player[playerCount].full.fullTwo>1)||  (player[playerCount].full.fullThree>1)||  (player[playerCount].full.fullFour>1)||  (player[playerCount].full.fullSix>1)))  player[playerCount].fullHouse=25;  else if((player[playerCount].full.fullSix<2)&&(  (player[playerCount].full.fullOne>1)||  (player[playerCount].full.fullTwo>1)||  (player[playerCount].full.fullThree>1)||  (player[playerCount].full.fullFour>1)||  (player[playerCount].full.fullFive>1)))  player[playerCount].fullHouse=25;  break;  case 10 :  player[playerCount].flag[9]=true;  if(searchNum(player,playerCount,1))  {  if(searchNum(player,playerCount,2))  if(searchNum(player,playerCount,3))  if(searchNum(player,playerCount,4))  player[playerCount].smallSrt=30;  }  if(searchNum(player,playerCount,2))  {  if(searchNum(player,playerCount,3))  if(searchNum(player,playerCount,4))  if(searchNum(player,playerCount,5))  player[playerCount].smallSrt=30;  }  if(searchNum(player,playerCount,3))  {  if(searchNum(player,playerCount,4))  if(searchNum(player,playerCount,5))  if(searchNum(player,playerCount,6))  player[playerCount].smallSrt=30;  }  break;  case 11 :  player[playerCount].flag[10]=true;  if(searchNum(player,playerCount,1))  {  if(searchNum(player,playerCount,2))  if(searchNum(player,playerCount,3))  if(searchNum(player,playerCount,4))  if(searchNum(player,playerCount,5))  player[playerCount].largeSrt=40;  }  if(searchNum(player,playerCount,2))  {  if(searchNum(player,playerCount,3))  if(searchNum(player,playerCount,4))  if(searchNum(player,playerCount,5))  if(searchNum(player,playerCount,6))  player[playerCount].largeSrt=40;  }  break;  case 12 :  player[playerCount].flag[11]=true;  if(player[playerCount].dice[0]==player[playerCount].dice[1])  if(player[playerCount].dice[0]==player[playerCount].dice[2])  if(player[playerCount].dice[0]==player[playerCount].dice[3])  if(player[playerCount].dice[0]==player[playerCount].dice[4])  player[playerCount].yahtzee=50;  break;  case 13 :  player[playerCount].flag[12]=true;  for(int i=0;i<NUM\_DICE;i++)  player[playerCount].chance=player[playerCount].chance+  player[playerCount].dice[i];  }    //iterate to next players turn  playerCount++;  if(playerCount>numPlayers-1)  playerCount=0;    //Exit Game condition when over  iterationCounter--;  if(iterationCounter==0)  {  for(int i=0;i<numPlayers;i++)  {  player[i].totalScore=player[i].ones+player[i].twos+  player[i].threes+player[i].fours+player[i].fives+  player[i].sixes+player[i].threeKind+player[i].fourKind+  player[i].fullHouse+player[i].smallSrt+  player[i].largeSrt+player[i].yahtzee+player[i].chance;  }  cout<<"Final Totals!"<<endl;  for(int i=0;i<numPlayers;i++)  {  cout<<player[i].initials<<"'s total points = "  <<player[i].totalScore<<endl;  }    for(int i=0;i<numPlayers;i++)  {  Scoreboard sb;  sb.getScore(player[i].totalScore);  }    cout<<"Thanks for playing!"<<endl<<endl;  exit=true;  }  else  {  do  {  cout<<"Would you like to exit and save the game(yes=1,no=0)?: ";  cin>>exitChoice;  if(exitChoice<0||exitChoice>1)  cout<<"Invalid!"<<endl;  }while(exitChoice<0||exitChoice>1);  if(exitChoice)  {  exit=true;  exitChoice=false;    numSavePlayersFile.open("numSavePlayers.dat", ios::out);  numSavePlayersFile<<numPlayers;  numSavePlayersFile.close();    iterationFile.open("iteration.dat", ios::out);  iterationFile<<iterationCounter;  iterationFile.close();    playerCountFile.open("playerCount.dat", ios::out);  playerCountFile<<playerCount;  playerCountFile.close();    gameFile.open("game.bin",ios::out|ios::binary);  for(int i=0;i<numPlayers;i++)  gameFile.write(reinterpret\_cast<char \*>(&player[i]),sizeof(Player));  gameFile.close();  }  else  cout<<"Continuing Game..."<<endl;  }  }while(!exit);      //delete pointer  delete [] player;  }  void setting(fstream& f)  {  int choice;  int num;    f.open("numPlayers.dat",ios::in);  if(!f)  cout<<"ERROR: Reading Player Count";  else  f>>num;  f.close();    cout<<endl<<endl;  do  {  cout<<"Settings:"<<endl;  cout<<"Type 0 to change number of players"<<endl;  cout<<"Type any other number to return to main menu"<<endl;  cin>>choice;  cin.ignore();  switch(choice)  {  case playerCount :  cout<<"Current Setting: "<<num<<endl;  cout<<"Enter number of players: ";  cin>>num;  cout<<"New Setting: "<<num<<endl<<endl;    break;  default : cout<<"Returning to main menu...";  }  }while(choice==0);    f.open("numPlayers.dat",ios::out);  if(!f)  cout<<"ERROR: Writing Player Count";  else  f<<num;  f.close();    cout<<endl;  }  Player\* rollDice(Player \* pR,int num,bool d1,bool d2,bool d3,bool d4,bool d5)  {  Player \*p = new Player[num];  p = pR;  if(d1)  p[num].dice[0]=1+(rand()%6);  if(d2)  p[num].dice[1]=1+(rand()%6);  if(d3)  p[num].dice[2]=1+(rand()%6);  if(d4)  p[num].dice[3]=1+(rand()%6);  if(d5)  p[num].dice[4]=1+(rand()%6);  return p;  }  void roll(Player \* rr,int num,int d)  {  rr[num].dice[d-1]=1+(rand()%6);  }  void dispDice(Player \* pD,int num)  {  for(int i=0;i<NUM\_DICE;i++)  cout<<" Dice "<<i+1<<": "<<pD[num].dice[i]<<endl;  }  char getInitials(Player \* pI,int i,int j)  {  return toupper((\*(pI+i)).name[j]);  }  bool searchNum(Player \* sN,int num,int j)  {  for(int i=0;i<NUM\_DICE;i++)  if(sN[num].dice[i]==j)  return true;  return false;  } |