# Project
# CONSOLE TETRIS

Jonathan Holt
48130
12/9/14

INTRODUCTION:

I am currently working on a console based Tetris game. I felt this game would be a great start to a building block of the kind of programs I want to start making when I get my degree eventually. The reason I wanted to build this game is because I have future ideas on possibly creating programs or games that will help my work (UPS). I started with a console style Tetris game because I thought of it has like loading a trailer or truck at UPS. Basically, the point of the game is try and get as many points as the player user can. The user is constantly prompted with the table to be played on and the object or block that they need to place on the table. To do this, the player needs to get a single or multiple rows of numbers lined up horizontally to then be erased and rewarded with points. If I could then take this and possibly turn it into a 3D Tetris game, I would feel incorporating these kinds of games would keep taking me to the goals I have in mind of programming things for UPS. These are my current goals going into my computer science major.

SUMMARY:

-Size – Project currently stands at 1064 lines without including my Tetris table class which would add more.

-Criteria - it meets the requirements for this project because I utilize everything that was asked of for the project. Also, used enumerated data in the game. I constantly create and return dynamic 2D arrays/pointers in the program particularly with the Tetris blocks and Tetris table. In my classes, I try to show the same aspect. The use of structures comes with lines with my file structure system. Strings are utilized especially with user input validation then incorporated the new idea of exceptions in the program. Converted objects to classes in my program with a base class, derived class, and an abstract class. Finally, a template was added to the program.

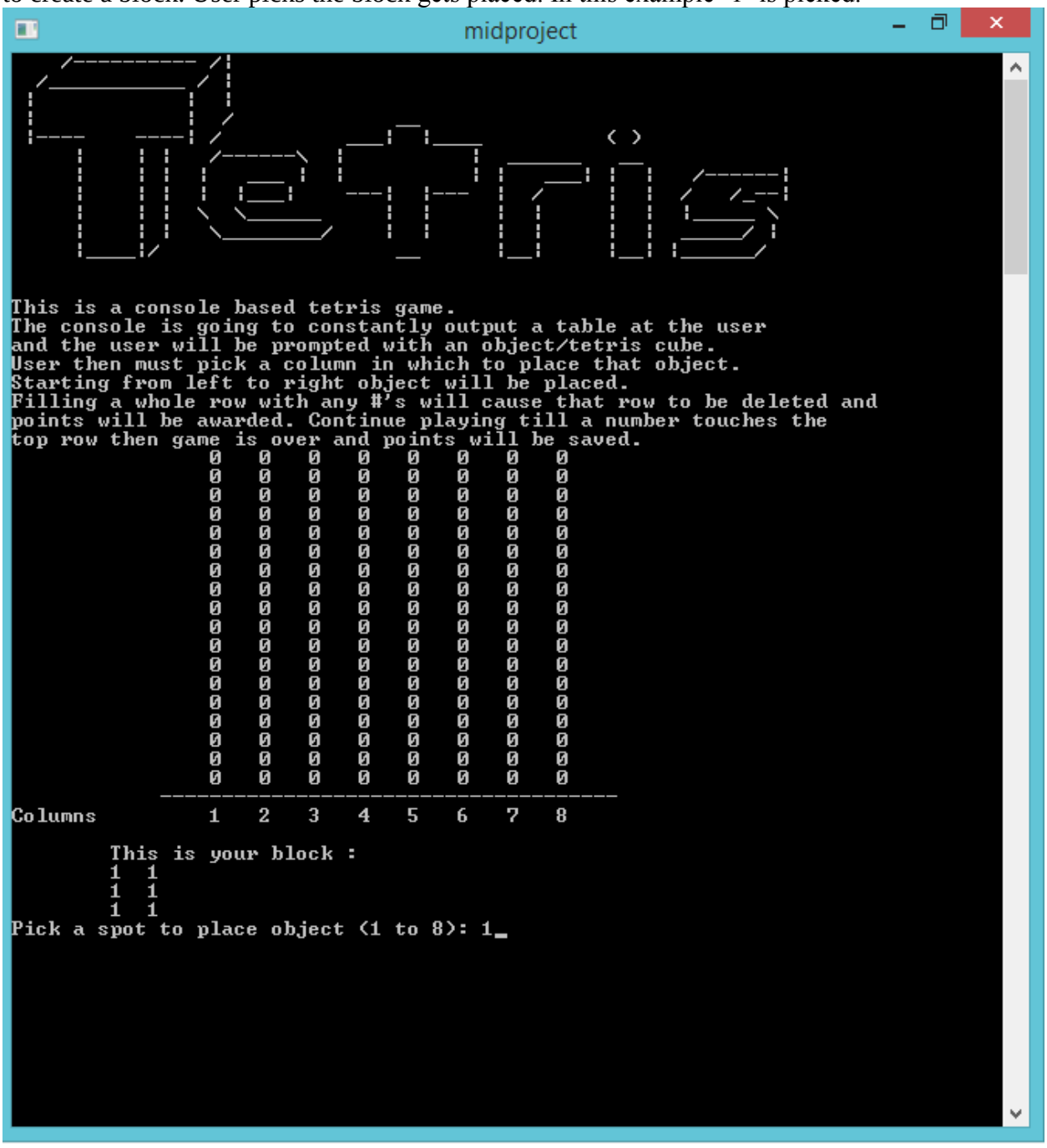
-Variables – 10 main variables created in the program.

-Production - Started working on program about more than half the semester and have been constantly updating the program. I believe I spent about 60 hours of actual coding, but a lot more of thinking of ways to incorporate things. Wasn't too challenging but somewhat difficult, I would've definitely been screwed if I waited till the last minute to try and write this program. I was able to get the game to delete multiple rows at once and then including a times multiplier for multiple rows at once. Also, added a rotation option to the user to rotate blocks if they wish to. I also do utilize a break to end the game which I couldn't get any other way to work without bounds issue screwing the game when blocks were stacked at the top of table. Got classes to work for this program. In addition, did a simple binary file to be integrated. Finally, a template was introduced.

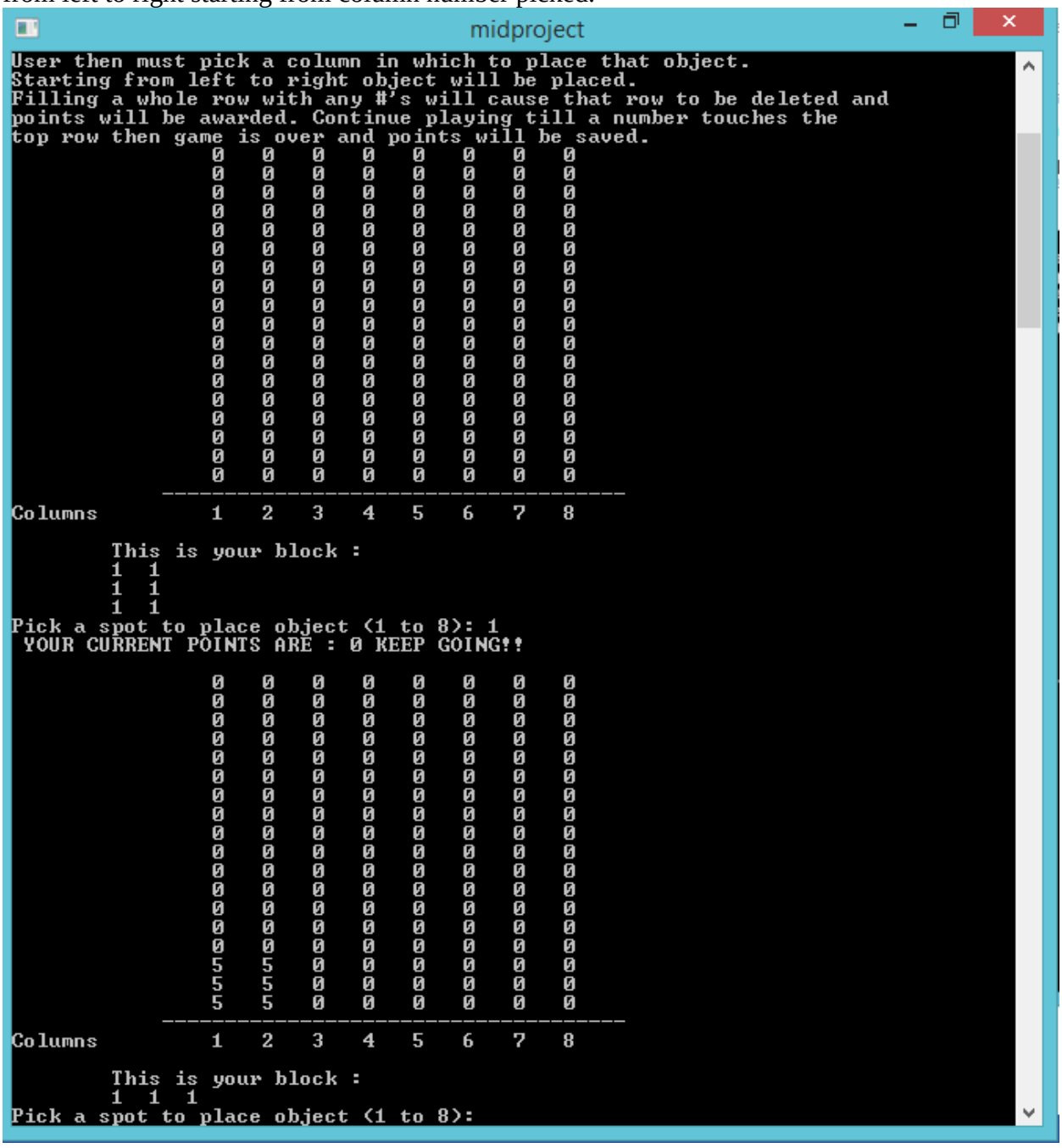
DESCRIPTION:

Started off creating a 2D pointer/array table in which user would try and place blocks given to them randomly. Problem was getting the objects to stack and playing without getting bounds issues. Reward points when a row was filled and shifting the table down 1 row. Create some kind of high score system.

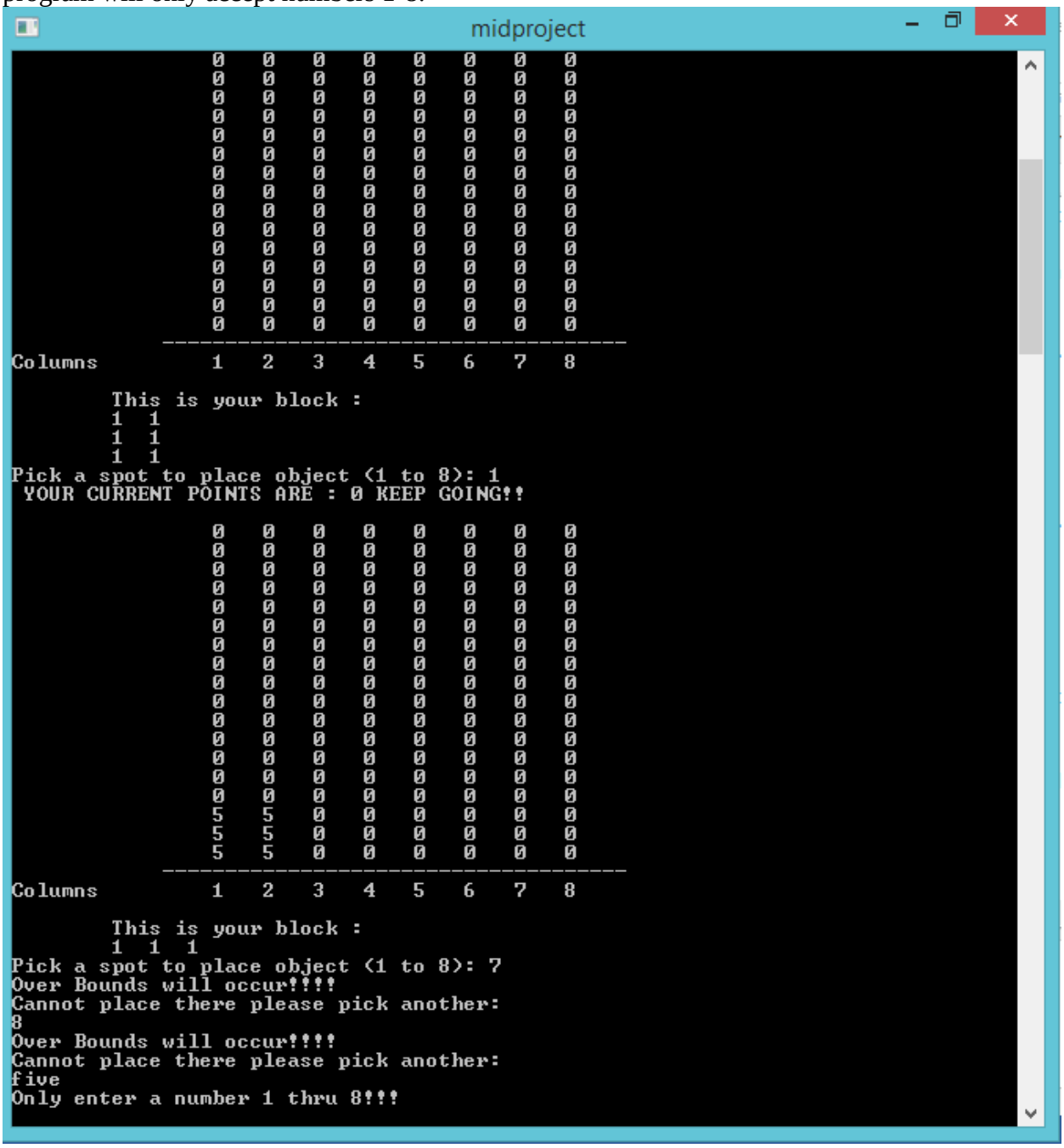
This is beginning of game where user is prompted with placement of the block. The class is used to create a block. User picks the block gets placed. In this example ‘1’ is picked.



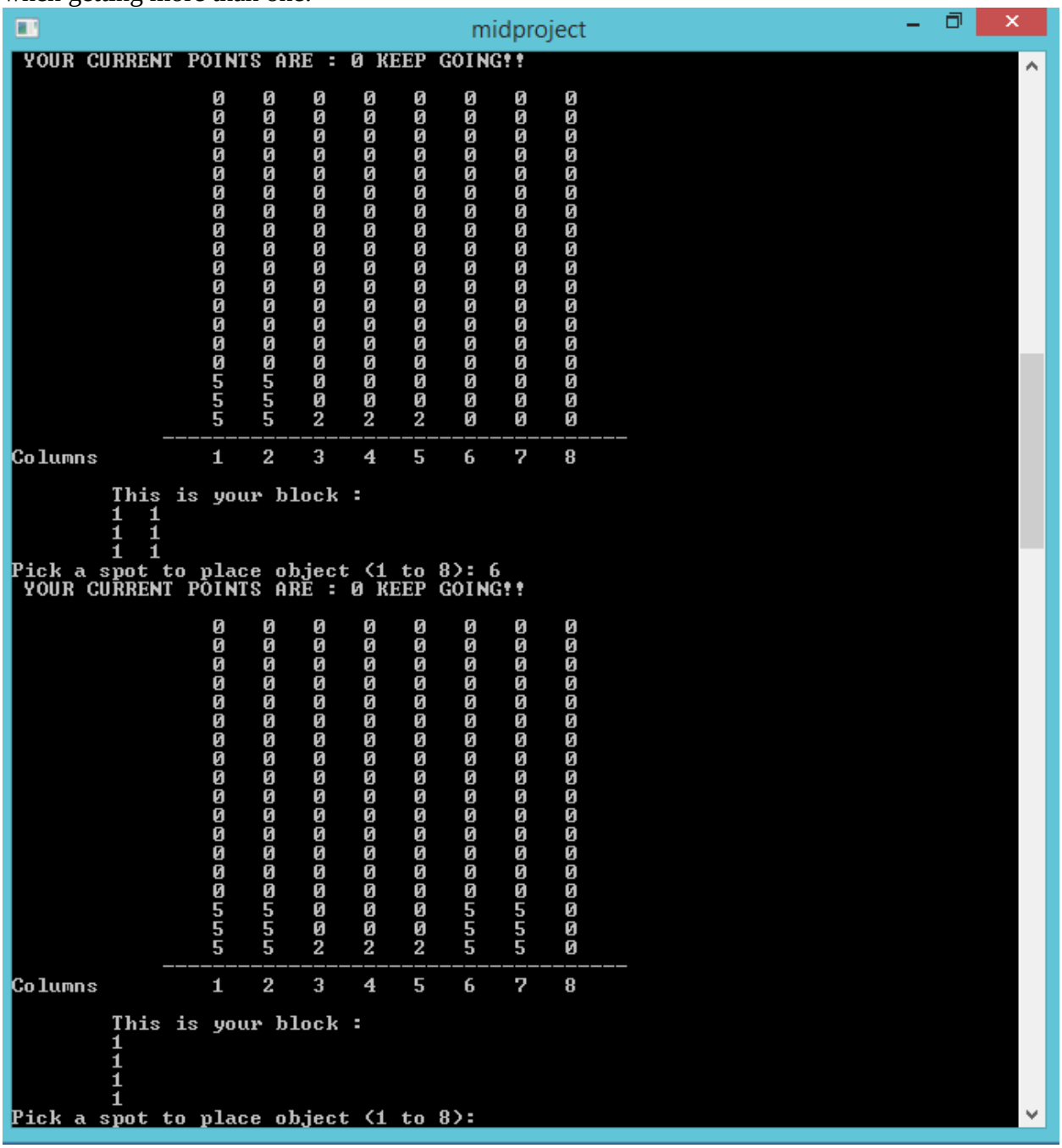
Object is then placed according to column picked by user. When column is picked always placed from left to right starting from column number picked.



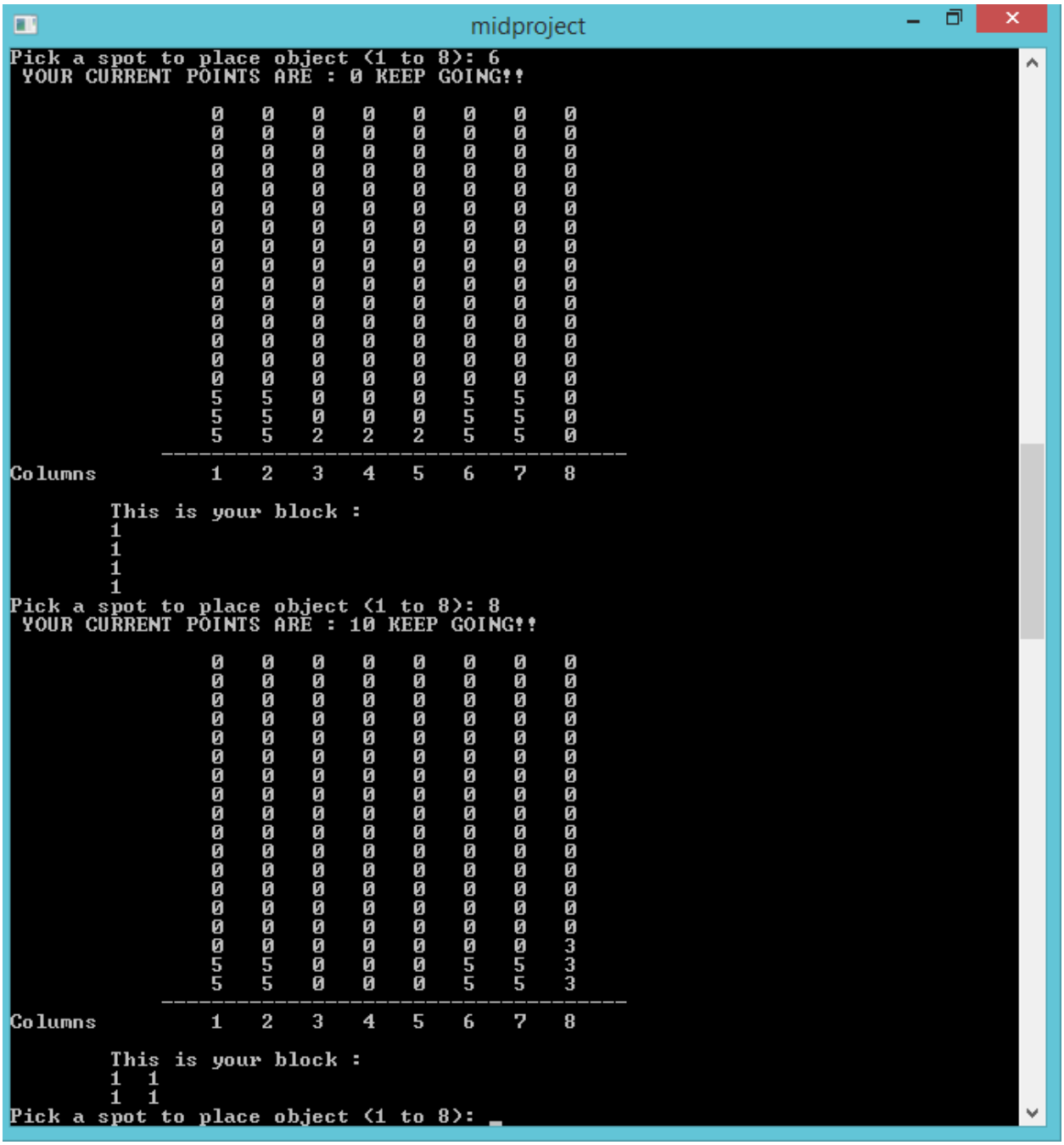
If trying to pick a place where object will go out of the table user is prompted with sentence “Over bounds will occur!” also checks for proper user input notice I typed ‘five’ into the program will only accept numbers 1-8.



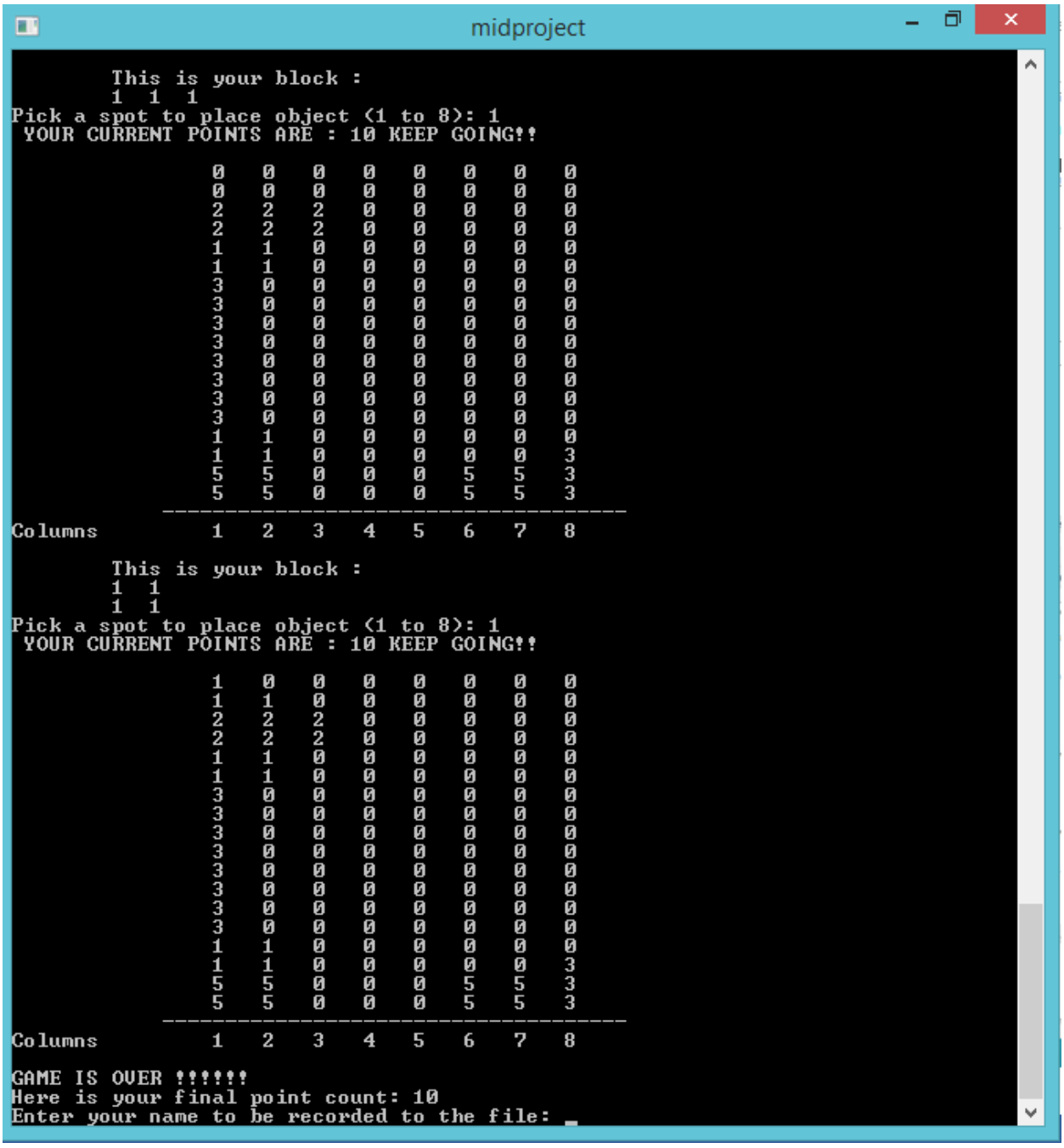
The point is to get a whole row filled with numbers. Different number represent the different blocks that were placed by the user. Once a row is completely full of numbers, row is deleted and user is rewarded with 10 points. Able to fix multiple rows problem and added point multiplier when getting more than one.



PLUS 10 points for completing a row! Everything from top of that row is copied and placed down 1 row.



Game is over once a block reaches the top row. Then user enters their name to be recorded to a file where a structure is used to get names and points of previous games and outputs the high score table to the user at the very end.



Pseudo Code:
*Start program by outputting details of game to user*
*create of 2D array/pointer table and fill it with 0's*

*Start game loop*
*call a random number for number of blocks that can be created this game currently is 1-6*
*if 1-6*
  *create and fill a 2D block dynamically to be played in table*

*user is promtped with choice of where to place object*
*user validation input is checked by entering acceptable number for placement*
  *also checks for multiple columns objects to not cause bounds issues*

*once user choice has been picked game looks for a non 0 number to place block*
  *on top of any numbers that could be blocking it from falling all the way down.*
*After that it will place block vertically where it should go however*
*if block reaches top row game is ended*
*else block is set on table accordingly*

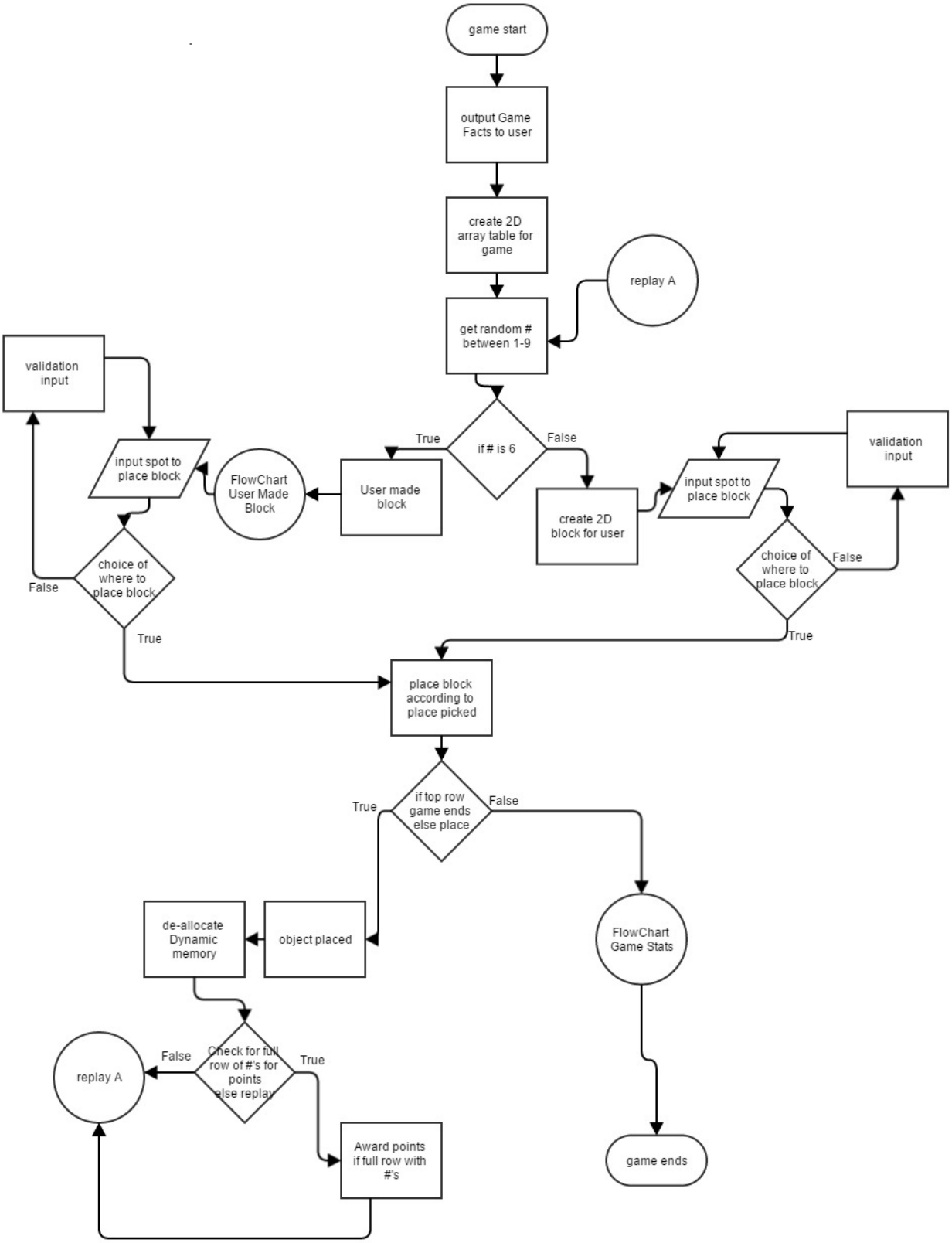
*destroy dynamically created block after it has been placed*

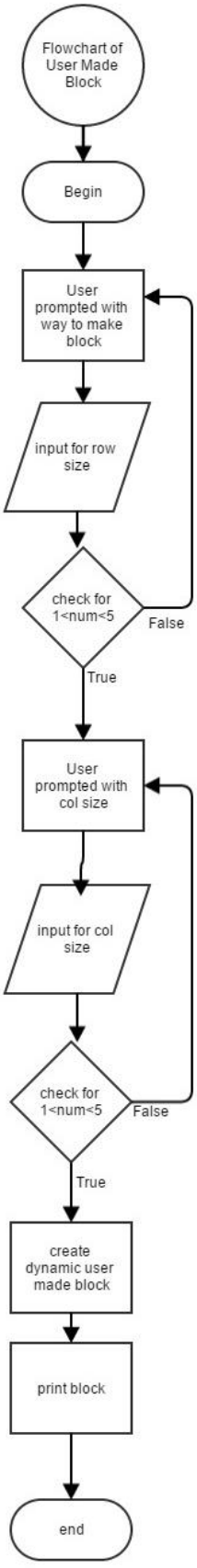
*Check for all #'s in a row*
*if all of row == # other than 0 |or| all of row != 0*
  *create and copy table dynamically with 2D array/pointer*
  *find row that contains full row of #'s then*
  *proceed to copy top of that row down exactly one row to replace*
  *all the numbers that completed a full row of #'s*
  *user is awarded 10 points*

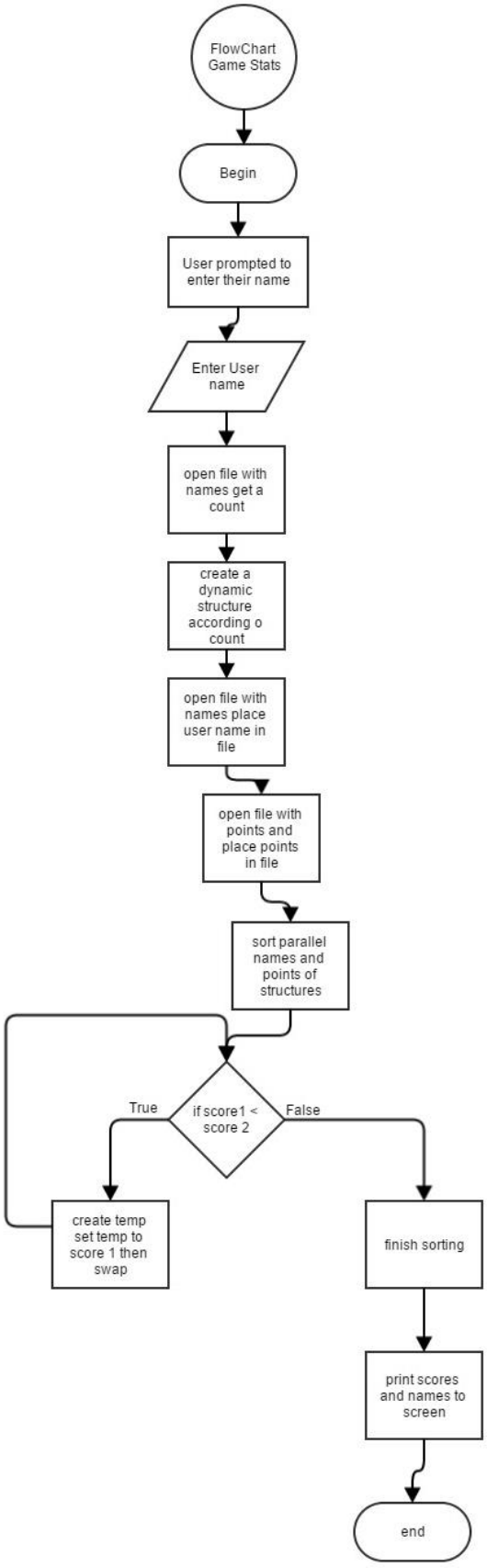
*game ends when a block reaches top row of table*

*use a structure to save names and points parallel*
  *Read in names from a file save them*
  *Read in points from a file save them*
*if one element > second element*
  *Sort*
*Output to screen the highscores with names*

Flowcharts (game)




Flowchart of User Made Block
Begin
User prompted with way to make block
input for row size
check for 1<num<5
False
True
User prompted with col size
input for col size
check for 1<num<5
False
True
create dynamic user made block
print block
end


FlowChart Game Stats
Begin
User prompted to enter their name
Enter User name
open file with names get a count
create a dynamic structure according o count
open file with names place user name in file
open file with points and place points in file
sort parallel names and points of structures
if score1 < score 2
True
False
create temp set temp to score 1 then swap
finish sorting
print scores and names to screen
end

Major Variables in the program:

| TYPE | NAME | DESCRIPTION | LOCATION | LINES(main) |
|---|---|---|---|---|
| Class | Block | Base class | Block.h | 67,71-123 |
| | CreateBlock | Derived class | CreateBlock.h | 67, 71-123 |
| | AbsBlock | Abstract class | AbsBlock.h | 67, 71-123 |
| Structure | Filetrack | Point system | void fileScores(int) | 143 |
| 2D Array (dynamic) | table | Game table | int main() | 52 |
| (dynamic) | newTble | Copied table | int **newTable(int**,int) | 126 |
| Integer | spot | Block placement | void spotChoice (int,int) | 603,614 |
| String | name | User name | void fileScores(&) | 784 |
| | valid/choice | User validation | void spotChoice (int,int) | 575 |
| Fstream | file | Read and Write to file | void fileScores(&) | 781-880 |
| | binFile | Binary File | void outputBegin() | 206-248 |
| Exceptions | BadRow{} | Exception class | Block.h | 306-510 |
| | BadCol{} | Exception class | Block.h | 306-510 |
| Template | points | Points for game | TetrisTable.h | 143, (844) |

References:

Gaddis, Tony. Starting Out With C++: From Control Structures Through Objects 8th ed. Pearson Education Inc. 2015. Text.

Joel. Gave some suggestions and helped with array table shifting.

In class examples.

```
/**
 * Jonathan Holt
 * C++ objects
 * project
 * i certify this is my work
 */

#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <cctype>
#include <cstring>
#include <ctime>
#include <fstream>
#include "tettable.h"
#include "createblock.h"

using namespace std;

/**
 * Constant variables to build the table in this program which is used as
 * 2D array/pointer table.
 */
const int ROWS = 18;
const int COLS = 8;
/**
 * Enumerator data.
 * Used to create the sizes of objects throughout the program.
 */
enum ObSpots {ONE = 1, TWO, THREE, FOUR, FIVE};/**< Enum value starting at 1.*/

struct Filetrack{/**< Structure that takes in names and scores from files.*/
   string name;
   int score;
};
void outputBegin ();
int randObject ();
int **objectNum (CreateBlock &, int num);
int **objectNum1 (CreateBlock &,int num);
void outputTbl(int **, int, int);
int **fillGrid (int , int);
bool isOver (int **);
void destroy (int**, int);
void objtPlcmnt(int **, int spot, CreateBlock &, int num);
```

```
int **newTable (int **, int &pts);
void spotChoice (int &spot, CreateBlock &, int);
int realNum (int n);
void fileScores (TetrisTable<int> &);

int main(int argc, char** argv) {
   //making table
   int **table = fillGrid (ROWS, COLS); /**< Table to be used for game. */
   TetrisTable<int> tble(18,8);/**< Table to be used for game. */
   srand (time(NULL));
   int **object; /**< 2D ptr that user will be prompted with.*/
   bool game; /**< Bool check to end the game officially. */
   int spot; /**< User choice of placement of objects.*/
   int points = 0;/**< Points to be tracked while playing. */

   //begin function
   outputBegin ();
   tble.outputTable();
   //outputTbl(table, ROWS, COLS);
   do
   {
      int num = randObject ();
      CreateBlock block;/**< pre-made block incorporated with a class.*/
      switch (num)
      {
         case 1:
            object = objectNum (block, num);
            spotChoice (spot, block, 1);
            objtPlcmnt(table, spot, block, 1);
            tble.placeBlock(spot, block, 1);
            //destroy (object, TWO);
            break;
         case 2:
            object = objectNum (block, num);
            spotChoice (spot, block,2);
            objtPlcmnt(table, spot, block, 2);
            tble.placeBlock(spot, block, 2);
            break;
         case 3:
            object = objectNum (block, num);
            spotChoice (spot, block,3);
            objtPlcmnt(table, spot, block, 3);
            tble.placeBlock(spot, block, 3);
            break;
         case 4:
            object = objectNum (block, num);
```

```
            spotChoice (spot, block,4);
            objtPlcmnt(table, spot, block, 4);
            tble.placeBlock(spot, block, 4);
            break;
         case 5:
            object = objectNum (block, num);
            spotChoice (spot, block,5);
            objtPlcmnt(table, spot, block, 5);
            tble.placeBlock(spot, block, 5);
            break;
         case 6:
            object = objectNum1 (block, num);
            spotChoice (spot, block,6);
            objtPlcmnt(table, spot, block, 6);
            tble.placeBlock(spot, block, 6);
            break;
         case 7:
            object = objectNum (block, num);
            spotChoice (spot, block,7);
            objtPlcmnt(table, spot, block, 7);
            tble.placeBlock(spot, block, 7);
            break;
         case 8:
            object = objectNum (block, num);
            spotChoice (spot, block,8);
            objtPlcmnt(table, spot, block, 8);
            tble.placeBlock(spot, block, 8);
            break;
         case 9:
            object = objectNum (block, num);
            spotChoice (spot, block,9);
            objtPlcmnt(table, spot, block, 9);
            tble.placeBlock(spot, block, 9);
            break;
      }
      tble.newTable();
      table = newTable (table, points);
      cout << " YOUR CURRENT POINTS ARE : " << points << " KEEP GOING!!\n\n";
      cout << " YOUR CURRENT POINTS ARE* : " << tble.getPoints() << " KEEP
GOING!!\n\n";
      //int **object = objectNum (num);
      //spotChoice (spot, TWO);
      tble.outputTable();
      //outputTbl(table, ROWS, COLS);
      //checking if lost
      //game = isOver(table);
```

```
        game = tble.game();
    }while(game != false);
    destroy(table, ROWS);

    cout << "GAME IS OVER !!!!!!" << endl;
    cout << "Here is your final point count: " << points << endl;
    cout << "HERE IS YOUR FINAL POINT COUNT: " << tble.getPoints() <<endl;
    fileScores (tble);

    return 0;
}
/**
 * This function is used to output the 2D table to the console that user will
 * be interacting with by placing objects in it. Outputs this table constantly
 * so user can always see it.
 * @param ptr 2D pointer.
 * @param rows size for rows.
 * @param cols size for columns.
 */
void outputTbl(int **ptr, int rows, int cols)
{
    int count = 0;
    for (int i = 0; i < ROWS; i++)
    {
        cout << "\t\t";
        for(int j = 0; j < COLS; j++)
        {
            cout << ptr[i][j] << "   ";
            count++;
            if (count == COLS)
            {
                cout << endl;
                count = 0;
            }
        }
    }
    cout << "\t    ------------------------------------" << endl;
    cout << "Columns \t1   2   3   4   5   6   7   8    "<< endl<<endl;
}
/**
 * Function creates a dynamic two dimensional table which is created by the
 * size of ROWS and COLS and sets all elements to 0. After that it then returns
 * the 2D pointer.
 * @param ROWS size for rows.
 * @param COLS size for columns.
 * @return 2D dynamic pointer.
```

```
 */
int **fillGrid (int ROWS, int COLS)
{
   int **array=new int*[ROWS];
   //creating 2D array
   for(int i=0;i<ROWS;i++)
   {
      array[i]=new int[COLS];
   }
   //setting to 0
   for (int row = 0; row < ROWS; row++)
   {
      for (int col = 0; col < COLS; col++)
      {
         array[row][col] = 0;
      }
   }
   return array;
}
/**
 * Information of beginning of game to give a brief description for how the
 * game is played.
 */
void outputBegin ()
{
   string str1= "    /---------- /|" ;//<< endl;
   string str2= "  /_____________ / | " ;//<< endl;
   string str3= " |             |  | " ;//<< endl;
   string str4= " |             |  /              __ " ;//<< endl;
   string str5= " |----     ----| /          ___|  |____          ( )" ;//<< endl;
   string str6= "      |    | |   /------\\  |          | _______   __ " ;//<< endl;
   string str7= "      |    | | |   ___ | |          | |   ___| | |   /------|";//\n";
   string str8= "      |    | | | |___|    ---|  |---  | /     | | |   /   /_--|";//\n";
  string str9= "      |    | |  \\  \\_______     | |     | |     | |  |______ \\";//\n";
   string str10= "      |    | |   \\__________/    | |     | |     | |  ______/ |";//\n";
   string str11= "      |_____|/                    __      |__|     |__| |_______/";//\n";
   fstream file;
   file.open("binFile.dat", ios::out | ios::binary);
   cout << "Writing to file..."<<endl;
   file.write(reinterpret_cast<char *> (&str1), sizeof(str1));
   file.write(reinterpret_cast<char *> (&str2), sizeof(str2));
   file.write(reinterpret_cast<char *> (&str3), sizeof(str3));
   file.write(reinterpret_cast<char *> (&str4), sizeof(str4));
   file.write(reinterpret_cast<char *> (&str5), sizeof(str5));
   file.write(reinterpret_cast<char *> (&str6), sizeof(str6));
   file.write(reinterpret_cast<char *> (&str7), sizeof(str7));
```

```
    file.write(reinterpret_cast<char *> (&str8), sizeof(str8));
    file.write(reinterpret_cast<char *> (&str9), sizeof(str9));
    file.write(reinterpret_cast<char *> (&str10), sizeof(str10));
    file.write(reinterpret_cast<char *> (&str11), sizeof(str11));
    file.close();
    file.open("binFile.dat", ios::in | ios::binary);
    cout << "Reading from the file..."<<endl;
    file.read(reinterpret_cast<char *> (&str1), sizeof(str1));
    file.read(reinterpret_cast<char *> (&str2), sizeof(str2));
    file.read(reinterpret_cast<char *> (&str3), sizeof(str3));
    file.read(reinterpret_cast<char *> (&str4), sizeof(str4));
    file.read(reinterpret_cast<char *> (&str5), sizeof(str5));
    file.read(reinterpret_cast<char *> (&str6), sizeof(str6));
    file.read(reinterpret_cast<char *> (&str7), sizeof(str7));
    file.read(reinterpret_cast<char *> (&str8), sizeof(str8));
    file.read(reinterpret_cast<char *> (&str9), sizeof(str9));
    file.read(reinterpret_cast<char *> (&str10), sizeof(str10));
    file.read(reinterpret_cast<char *> (&str11), sizeof(str11));
    cout << str1 <<endl <<str2<< endl<<str3<< endl<<str4<< endl<<str5<< endl
         <<str6<< endl<<str7<< endl<<str8<< endl<<str9<< endl<<str10<< endl
         <<str11<<endl<<endl;
    file.close();
    cout << "This is a console based tetris game." << endl << "The console is"
         " going to constantly output a table at the user" << endl << "and"
         " the user will be prompted with an object/tetris cube." << endl <<
         "User then must pick a column in which to place that object."<< endl
         <<"Starting from left to right object will be placed." <<endl
         << "Filling a whole row with any #'s will cause that row to be"
         " deleted and"<<endl << "points will be awarded. Continue playing"
         " till a number touches the" << endl <<"top row then game is over"
         " and points will be saved. Bonus points \nfor multiple rows "
         "completed at once. Imagine that the blocks get dropped"<<endl<<
         "automatically from the top of the column of your choice down to"
         " where it"<<endl<< "either stacks or falls to the very bottom."
         "\n\n";
}
/**
 * destroy function is used to delete all of the dynamically created objects
 * in the program which most are 2D so need to delete by rows first then delete
 * the entire thing.
 * @param array the dynamic 2D pointer/array.
 * @param rows rows to be deleted first.
 */
void destroy(int **array,int rows)
{
    //Destroy in reverse order of creation
```

```
   for(int i=0;i<rows;i++)
   {
      delete []array[i];
   }
   delete []array;
}
/**
 * Rand function is user to get a number between 1 and 6. It is for the
 * different objects to be randomized by particularly 6 of them.
 * @return an integer value to be used to choose a random object.
 */
int randObject ()
{
   int num;
   num = rand()%9+1;
   return num;
}
/**
 * This function has multiple objects from which takes a number 1 to 5 and
 * dynamically creates the object. Then outputs that object to the screen
 * for user to see what that dynamic 2D pointer looks like and then
 * to be placed on table.
 * @param blk class used to create a block for user
 * @param num a random integer value used to pick object.
 * @return the object chosen by integer num returns a 2D dynamic array.
 */
int **objectNum (CreateBlock &blk, int num)
{
   int count =0;
   int **object;
   if (num == 1)
   {
      try{
         blk.setRow(TWO);
      }
      catch(Block::BadRow){
         cout << "There is an invalid row manual input for tetris block 1."
              <<endl << "Please refix this issue if you wish to play the"
              " game correctly."<<endl<<"Setting row to default == 1.\n";
         blk.setRow(1);
      }
      try{
         blk.setCol(TWO);
      }
      catch(Block::BadCol){
         cout << "There is an invalid column manual input for tetris block 1."
```

```
            <<endl << "Please refix this issue if you wish to play the"
            "game correctly."<<endl <<"Setting col to default == 1.\n";
        blk.setCol(ONE);
    }
    //blk.setRow(TWO);
    //blk.setCol(TWO);
    blk.makeBlock();
    blk.print(blk, 1);
    object = blk.getBlock();
}
if (num == 2)
{
    try{
        blk.setRow(ONE);
    }
    catch(Block::BadRow){
        cout << "There is an invalid row manual input for tetris block 2."
            <<endl << "Please refix this issue if you wish to play the"
            " game correctly."<<endl<<"Setting row to default == 1.\n";
        blk.setRow(ONE);
    }
    try{
        blk.setCol(THREE);
    }
    catch(Block::BadCol){
        cout << "There is an invalid column manual input for tetris block 2."
            <<endl << "Please refix this issue if you wish to play the"
            "game correctly."<<endl <<"Setting col to default == 1.\n";
        blk.setCol(ONE);
    }
    //blk.setRow(ONE);
    //blk.setCol(THREE);
    blk.makeBlock();
    blk.print(blk,2);
    object = blk.getBlock();
}
if (num == 3)
{
    try{
        blk.setRow(FOUR);
    }
    catch(Block::BadRow){
        cout << "There is an invalid row manual input for tetris block 3."
            <<endl << "Please refix this issue if you wish to play the"
            " game correctly."<<endl<<"Setting row to default == 1.\n";
        blk.setRow(1);
```

```
    }
    try{
       blk.setCol(ONE);
    }
    catch(Block::BadCol){
       cout << "There is an invalid column manual input for tetris block 3."
            <<endl << "Please refix this issue if you wish to play the"
            "game correctly."<<endl <<"Setting col to default == 1.\n";
       blk.setCol(ONE);
    }
    //blk.setRow(FOUR);
    //blk.setCol(ONE);
    blk.makeBlock();
    blk.print(blk,3);
    object = blk.getBlock();
}
if (num == 4)
{
    try{
       blk.setRow(TWO);
    }
    catch(Block::BadRow){
       cout << "There is an invalid row manual input for tetris block 4."
            <<endl << "Please refix this issue if you wish to play the"
            " game correctly."<<endl<<"Setting row to default == 1.\n";
       blk.setRow(1);
    }
    try{
       blk.setCol(THREE);
    }
    catch(Block::BadCol){
       cout << "There is an invalid column manual input for tetris block 4."
            <<endl << "Please refix this issue if you wish to play the"
            "game correctly."<<endl <<"Setting col to default == 1.\n";
       blk.setCol(ONE);
    }
    //blk.setRow(TWO);
    //blk.setCol(THREE);
    blk.makeBlock();
    blk.print(blk,4);
    object = blk.getBlock();
}
if (num == 5)
{
    try{
       blk.setRow(THREE);
```

```
    }
    catch(Block::BadRow){
       cout << "There is an invalid row manual input for tetris block 5."
            <<endl << "Please refix this issue if you wish to play the"
            " game correctly."<<endl<<"Setting row to default == 1.\n";
       blk.setRow(1);
    }
    try{
       blk.setCol(TWO);
    }
    catch(Block::BadCol){
       cout << "There is an invalid column manual input for tetris block 5."
            <<endl << "Please refix this issue if you wish to play the"
            "game correctly."<<endl <<"Setting col to default == 1.\n";
       blk.setCol(ONE);
    }
    //blk.setRow(THREE);
    //blk.setCol(TWO);
    blk.makeBlock();
    blk.print(blk,5);
    object = blk.getBlock();
 }
 if (num == 7){
    try{
       blk.setRow(ONE);
    }
    catch(Block::BadRow){
       cout << "There is an invalid row manual input for tetris block 5."
            <<endl << "Please refix this issue if you wish to play the"
            " game correctly."<<endl<<"Setting row to default == 1.\n";
       blk.setRow(1);
    }
    try{
       blk.setCol(TWO);
    }
    catch(Block::BadCol){
       cout << "There is an invalid column manual input for tetris block 5."
            <<endl << "Please refix this issue if you wish to play the"
            "game correctly."<<endl <<"Setting col to default == 1.\n";
       blk.setCol(ONE);
    }
    //blk.setRow(ONE);
    //blk.setCol(TWO);
    blk.makeBlock();
    blk.print(blk,7);
    object = blk.getBlock();
```

```
}
if (num == 8)
{
   try{
      blk.setRow(TWO);
   }
   catch(Block::BadRow){
      cout << "There is an invalid row manual input for tetris block 5."
           <<endl << "Please refix this issue if you wish to play the"
           " game correctly."<<endl<<"Setting row to default == 1.\n";
      blk.setRow(1);
   }
   try{
      blk.setCol(FOUR);
   }
   catch(Block::BadCol){
      cout << "There is an invalid column manual input for tetris block 5."
           <<endl << "Please refix this issue if you wish to play the"
           "game correctly."<<endl <<"Setting col to default == 1.\n";
      blk.setCol(ONE);
   }
   //blk.setRow(THREE);
   //blk.setCol(TWO);
   blk.makeBlock();
   blk.print(blk,8);
   object = blk.getBlock();
}
if (num == 9)
{
   try{
      blk.setRow(FOUR);
   }
   catch(Block::BadRow){
      cout << "There is an invalid row manual input for tetris block 5."
           <<endl << "Please refix this issue if you wish to play the"
           " game correctly."<<endl<<"Setting row to default == 1.\n";
      blk.setRow(1);
   }
   try{
      blk.setCol(TWO);
   }
   catch(Block::BadCol){
      cout << "There is an invalid column manual input for tetris block 5."
           <<endl << "Please refix this issue if you wish to play the"
           "game correctly."<<endl <<"Setting col to default == 1.\n";
      blk.setCol(ONE);
```

```
      }
      //blk.setRow(THREE);
      //blk.setCol(TWO);
      blk.makeBlock();
      blk.print(blk,9);
      object = blk.getBlock();
   }

   return object;
}
/**
 * A function that creates a 2 dimensional pointer through a structure. Allows
 * user to make it and checks for valid user input. User enters a row size
 * then a column size not allowing to be bigger than 5 or less than 1 for both.
 * Keeps track of row and column integers to be used for delete later.
 * @see objectNum().
 * @param blk class UserObject
 * @param num  a random integer value used to pick object.
 * @param rowOb an integer to save the row value.
 * @param colOb an integer to save the column value.
 * @return the object chosen by integer number returns a 2D dynamic array.
 */
int **objectNum1 (CreateBlock &block, int num)
{
   string valid;
   if (num == 6)
   {
      cout << "This is a BONUS PLAY!" << endl << "User is allowed to enter"
           " an object of their own choice." << endl << "The max size"
           " you may enter is a 5x5." << endl;

      cout << "Enter a row(s) number between 1 n 5 (height)" << endl;
      cin >> valid;
      while (valid != "1" &&valid != "2" &&valid!= "3" &&valid != "4" &&
         valid != "5")
      {
         cout << "Only enter a number 1 thru 5!!!" << endl;
         cin >> valid;
         //object.rows = realNum (valid[0]);
      }
      //object.rows = realNum (valid[0]);
      int row = realNum (valid[0]);
      block.setRow(row);

      cout << "Enter a column(s) number between 1 n 5 (width)" << endl;
      cin >> valid;
```

```
      while (valid != "1" &&valid != "2" &&valid!= "3" &&valid != "4" &&
         valid != "5"){
         cout << "Only enter a number 1 thru 5!!!" << endl;
         cin >> valid;
      }
      //object.cols = realNum (valid[0]);
      int col = realNum (valid[0]);
      block.setCol(col);
   //creating user object
      block.makeBlock();
      block.print(block,6);

   return block.getBlock();
   }
}
/**
 * This function spotChoice allows user to input data that will then decide
 * where they wish to place their object. Checks for user validation by taking
 * a string first thing converting it to an integer. The integer spot is
 * referenced to be changed throughout the program. Returns nothing.
 * @param spot an integer value that represents spot to be placed on table.
 * @param b class used for col value needed to check for bounds issues.
 */
void spotChoice (int &spot, CreateBlock &b, int num){
   string choice;
   /**
    * this do while loop lets the user be able to rotate the object to be able
    * to place the block on the table in a rotation of favorable fashion.
    */
   do{
      cout << "Pick a spot to place object (1 to 8): \n" ;
      cout << "Do you wish to rotate the object type ('r') or place object:"
         <<endl;
      cin >> choice;

      while (choice != "1" &&choice != "2" &&choice != "3" &&choice != "4" &&
            choice != "5" &&choice != "6" &&choice != "7" &&choice != "8"||
            choice == "r")
      {
         if (choice == "r"){
         int temp = b.getRow();
         int temp1 = b.getCol();
         b.setRow(temp1);
         b.setCol(temp);
      //b.rotateBlock(b)
         b.makeBlock();
```

```
            b.print(b, num);
        }
            cout << "Only enter a number 1 thru 8 or ('r')!!!" << endl;
            cin >> choice;
        }

        spot =  realNum (choice[0]);
    //cout << "SPOT: "  << spot << "SPPOT!!"<<endl;
    //IMPORTANT BOUNDS CHECKING AND PLACEMENT
    //first area checks spot choice compared to column size of object
    //rest makes sure number is 1 - 8
        while ((spot-1) + b.getCol() > 8 || spot < 1 || spot > 8&&choice != "1"
            &&choice != "2" &&choice != "3" &&choice != "4" &&
            choice != "5" &&choice != "6" &&choice != "7" &&choice != "8" ){
                cout << "Over Bounds will occur!!!!" << endl;
                cout << "Cannot place there please pick another:" << endl;
                cin >> choice;
                spot =  realNum (choice[0]);
          }
    }while (choice == "r");
}
/**
 * The function realNum takes an integer number between 1 and 8 that then
 * references it back to the ascII table to be able to convert it to its
 * actual number.
 * @param n an integer value that inputted by user.
 * @return returns an integer value.
 */
int realNum (int n){
    int realOne;
    if (n == 49){
        realOne = 1;
    }if (n == 50) {
        realOne = 2;
    }if (n == 51){
        realOne = 3;
    }if (n == 52){
        realOne = 4;
    }if (n == 53){
        realOne = 5;
    }if (n == 54){
        realOne = 6;
    }if (n == 55){
        realOne = 7;
    }if (n == 56){
        realOne = 8;
```

```
    }
    if (n == 57){
        realOne = 9;
    }
    return realOne;
}
/**
 * The function isOver is checking for a non 0 value on the first row of the
 * table. If it finds this value it instantly breaks from stacking objects on
 * the table that way over bounds issues will not come into play.
 * @param tbl 2D pointer that represents a table of elements.
 * @return condition of true or false.
 */
bool isOver (int **t)
{
    bool lose;
    //for (int i =0; i < ROWS; i++){ actually dont need to check columns
    int i = 0;
    for (int col=0; col < COLS;col++){
        //[0][j] because this would start from the top left of table
        if (t[i][col] != 0)
        {
            lose = false;
            break;
        }
        else
            lose = true;
    }
    return lose;
}
/**
 * The function objtPlcmnt is first going take 2D table then since object is
 * of a certain size looking at size of column spots to check for a #. If there
 * is a # in either spot i set rows to that spot to then place object on top
 * of the numbers in table. Uses a break statement to break from going over
 * bounds when placing an object.
 * @param tble 2D table.
 * @param spot integer value entered by user.
 * @param b class block used for rows and cols of block
 * @param num integer value to represent numbers in the object.
 */
void objtPlcmnt(int **tble, int spot, CreateBlock &b, int num)
{
    //t.placeBlock(spot, b, num);
    //columns
    int user_C = spot-1;
```

```
    int row = ROWS;
    //starting from bottom left to top
    for (int i = ROWS-1; i >= 0; i--){
       for (int k = 0; k < b.getCol(); k++)
       {
         if (tble[i][user_C+k] != 0 )//||
         {
           //setting row
           row = i;
         }
       }
     }

     for (int i=row-1; i >= row-b.getRow(); i--){
         for(int j=0; j < b.getCol(); j++)  {
            //checking if spot[col] top of table = 1 if so break from placing
            //one
            if (tble[0][user_C] != 0){
               //breaking from loop cycle
               break;
            }
            else
             tble [i][j+spot-1] = num;
         }
      }
}
/**
 * This function newTable is designed to act as the point system for the game.
 * It creates a new dynamic 2D table and then copies the one currently being
 * use by user. It loops through all the rows in the table by checking for
 * non zero #s and if it finds that row saves that row. After that goes
 * through two loops to copy the row above that row and everything above it to
 * be placed where the row with all #'s were.
 * @param tble 2D pointer of table for game play.
 * @param pts integer value that is used to keep track of user points.
 * @return the 2D pointer that represents new table.
 */
int **newTable (int **tble, int &pts){
    int dstryRow =0; /**< Integer value of row where replacing happens.*/
    int count=0;/**< Integer value of number of rows filled at a time.*/

    for (int i = 0; i < ROWS; i++) {
          if (tble[i][0] != 0 && tble[i][1] != 0 &&tble[i][2] != 0 &&
                tble[i][3] != 0 &&tble[i][4] != 0 &&tble[i][5] != 0 &&
                tble[i][6] != 0 &&tble[i][7] != 0)
          {
```

```
            count++;
        }
    }
    cout << "YOU GOT " << count << " ROWS AT ONCE."<<endl;

        for (int i = 0; i < ROWS; i++)
        {
            if (tble[i][0] != 0 && tble[i][1] != 0 &&tble[i][2] != 0 &&
                tble[i][3] != 0 &&tble[i][4] != 0 &&tble[i][5] != 0 &&
                tble[i][6] != 0 &&tble[i][7] != 0)
            {
                pts += 10*count;
                dstryRow=i;
                /**
                 * Had to create table in loop or else same table was getting
                 * copied this way new table gets copied with deleted row.
                 */
                int **newTble = fillGrid (ROWS, COLS);/**< Creating new table to copy.*/
                for (int i =0; i < ROWS; i++){
                    for (int j =0; j < COLS; j++){
                        newTble[i][j] = tble[i][j]; /**< Copying new table.*/
                    }
                }
                //setting row above to = to the row to be deleted
                for (int k = 0; k < dstryRow; k++){
                     for (int j =0; j < COLS ; j++){
                         tble[k+1][j] = newTble[k][j];
                     }
                }
                destroy (newTble, ROWS);
            }
        }
    return tble;
}
/**
 * The function fileScores serves the purpose of a high score tracking system.
 * It first goes through the name file to get a count to find out how big of
 * a dynamic structure needs to be allocated. Then 2D dynamic structure is
 * created to keep track of parallel names and points. First, names are read
 * and saved and then last person played name is saved and written back to the
 * file. After, points are read in and then points of last played game are also
 * recorded and then all sent back to the files. After that, names and
 * points are then sorted parallel to find highest to lowest then printed to
 * user.
 * @param points integer value of user's points after game is done.
 */
```

```
void fileScores (TetrisTable<int> &t){
    fstream file;
    string output;
    string name;
    int scores;
    int count=1;

    cout << "Enter your name to be recorded to the file: ";
    cin.ignore();
    getline (cin, name);

    file.open("names.txt", ios:: in | ios::out);// ios::app |

    if(file){
          while (getline(file, output))
          {
             count++;
          }
       }
    file.close();
    //allNames = new string [count];
    //allpoints = new int [count];
    Filetrack *stats = new Filetrack [count];
    file.open("names.txt", ios:: in );//| ios::ate
    int nameCounter=0;
    if(file){
       getline(file, output); //WORKING ON GETLINE
       while (file)
       {
          stats[nameCounter].name=output;
          nameCounter++;
          getline(file, output);
       }
    }

    file.close();
    //setting name entered to last element
    stats[count-1].name=name;

    cout<<endl;
    file.open("names.txt", ios:: out);
    for (int i=0;i<count;i++){

       if (i != count-1)
       {
          file << stats[i].name <<endl;
```

```
    }
    else
        file << stats[i].name;
}
file.close();
file.open("scores.dat", ios:: in );//| ios::ate
int pointCounter=0;
if(file){
        while (file >> scores)
        {
            stats[pointCounter].score=scores;
            pointCounter++;
        }
    }

file.close();
//setting points of last person played to last element
stats[count-1].score=t.getPoints();

file.open("scores.dat", ios:: out);
for (int i=0;i<count;i++){

    if (i != count-1)
    {
        file << stats[i].score <<endl;
    }
    else
        file << stats[i].score;
}
file.close();
int temp = 0;
string temp1="";
for (int i = 0; i < count - 1; i++)
{
    for (int j = i + 1; j < count; j++)
    {
        if (stats[j].score < stats[i].score)
        {
            temp = stats[j].score;
            stats[j].score = stats[i].score;
            stats[i].score = temp;
            temp1 = stats[j].name;
            stats[j].name = stats[i].name;
            stats[i].name = temp1;
        }
    }
```

```
    }
    cout << "\nHIGH SCORES!!!!!!!!!!!!!!!!!!!" <<endl;
    cout << "NAME:            SCORES:"<<endl;
    for (int i=count-1; i >=0;i--){
        cout<<left<<setw(20)<<stats[i].name << stats[i].score <<endl;
    }
    delete []stats;
}
```

# Absblock.h

```
/*
 * File:   absblock.h
 * Author: Jonathan
 *
 * Created on December 4, 2014, 9:39 PM
 */

#ifndef ABSBLOCK_H
#defineABSBLOCK_H
class AbsBlock{
   public:
      virtual int getRow() = 0;
      virtual int getCol() = 0;
};


#endif  /* ABSBLOCK_H */
```

# Block.h

```
/*
 * File:   block.h
 * Author: Jonathan
 *
 * Created on November 30, 2014, 8:47 PM
 */

#ifndef BLOCK_H
#defineBLOCK_H
#include "absblock.h"
class Block: public AbsBlock{
   private:
      int row;
      int col;
   public:
      //exceptions
      class BadRow{};
      class BadCol{};
      Block ();
      Block (int, int);
      void setRow (int);
      void setCol (int);
      int getRow(){return row;}
```

```
        int getCol(){return col;}
};

#endif  /* BLOCK_H */
```

## Block.cpp

```
///implementation of block class

#include "block.h"

Block::Block(){
   row = 1;
   col = 1;
}
Block::Block(int r, int c){
   if (r < 1)
      throw BadRow();
   else
      row = r;
   if (c < 1)
      throw BadCol();
   else
      col = c;
}
void Block::setRow(int r){
   if (r < 1)
      throw BadRow();
   else
      row = r;
}
void Block::setCol(int c){
   if (c < 1)
      throw BadCol();
   else
      col = c;
}
```

## CreateBlock.h

```
/*
 * File:   createblock.h
 * Author: Jonathan
 *
 * Created on November 30, 2014, 9:15 PM
 */
```

```
#ifndef CREATEBLOCK_H
#defineCREATEBLOCK_H

#include "block.h"
//Derived class from block
class CreateBlock : public Block{
   private:
      int **block;/**< 2D ptr that user will be prompted with.*/
      void fillBlock();
      int num;
   public:
      CreateBlock();
      CreateBlock (int,int);
      ~CreateBlock();
      void rotateBlock(Block &);
      void makeBlock();
      void print(Block &,int);
      int **getBlock()const {return block;};
};

#endif  /* CREATEBLOCK_H */
```

## CreateBlock.cpp

```
///implementation of create block class derived from block class

#include "createblock.h"
#include <iostream>
using namespace std;
CreateBlock::CreateBlock(){
   ///setting dimensions to 1x1
   setRow (1);
   setCol (1);
   ///dynamically allocating memory
   block = new int *[getRow()];
   for (int i=0; i < getRow();i++){
      block[i] = new int [getCol()];
   }
   fillBlock();
}
///derived class of base class block
CreateBlock::CreateBlock(int r, int c):Block(r, c){
   block = new int *[r];
   for (int i=0; i < r;i++){
      block[i] = new int [c];
   }
```

```
    fillBlock();
}
/**
 * Filling object with 1's
 */
void CreateBlock::fillBlock(){
    for(int i =0; i < getRow();i++){
        for (int j =0; j<getCol(); j++){
            block[i][j] = 1;
        }
    }
}
/**
 * Destructor
 * destroy class 2D pointer is used to delete all of the dynamically created
 * objects in the class  so need to delete by rows first then delete
 * the entire thing.
 */
CreateBlock::~CreateBlock(){
    //cout << "CALLING DESTRUCTOR"<<endl;
    for(int i=0;i<getRow();i++)
    {
        delete []block[i];
    }
    delete []block;
}
void CreateBlock::makeBlock(){
    block = new int *[getRow()];
    for (int i=0; i <getRow();i++){
        block[i] = new int[getCol()];
    }
    fillBlock();
}
/**
 * Function used to print the block and is outputted to the user to be prompted
 * to place on the table for playing
 * @param b class block used needed columns and rows
 */
//polymorphism
void CreateBlock::print(Block &b, int num) {
    int count=0;
    cout << "\tThis is your block : " << endl;
        for (int i=0; i <b.getRow(); i++){//rows
            cout << "\t";
           for(int j=0; j <b.getCol(); j++){//cols
             block[i][j]= num;
```

```
            cout << block[i][j]  << "  ";
            count++;
            if (count == b.getCol())
            {
               cout <<  endl;
               count = 0;
            }
          }
       }
}
```

## TetTable.h

```
/*
 * File:   tettable.h
 * Author: Jonathan
 *
 * Created on November 20, 2014, 10:20 AM
 */

#ifndef TETTABLE_H
#defineTETTABLE_H
#include "createblock.h"
#include "tettable.h"
#include <string>
#include <iostream>
using namespace std;

template<class T>
class TetrisTable{
   private:
      int **table;
      int row;
      int col;
      T points;
      int **fill_tble ();
      CreateBlock b;
   public:
      //constructors
      TetrisTable();
      TetrisTable(int, int);
      //copy constructor
      TetrisTable (const TetrisTable &);
      //destructor
      ~TetrisTable();
      //getters
      int getRow()const{return row;}
```

```
        int getCol()const{return col;}
        int getPoints()const{return points;}
        int **getTable()const{return table;}
        int getElem(int i, int j){return table[i][j];}
        //if(i>=0&&i<size)

        void placeBlock(int, CreateBlock &, int );
        void outputTable ()const;
        void newTable();
        bool game();
};

//constructor
template<class T>
TetrisTable<T>::TetrisTable(){
    table = new int *[row];
    for (int i=0; i<row;i++){
         table[i] = new int [col];
    }
    points =0;
}
template<class T>
TetrisTable<T>::TetrisTable(int r, int c){
    //get an execption in here
    if (r>0)
        row=r;
    if (c>0)
        col=c;
    table = new int *[row];
    for (int i=0; i<row;i++){
         table[i] = new int [col];
    }
    table = fill_tble();
    points =0;
}
//copy constructor
template<class T>
TetrisTable<T>::TetrisTable(const TetrisTable &cpyTble){
    row = cpyTble.row;
    col = cpyTble.col;
    table = new int *[row];
    //allocate
    for(int i=0;i<row;i++){
        table[i]=new int [col];
    }
    //copy contents
```

```
    for(int i=0;i <row;i++){
        for(int j=0;j<col;j++){
            table[i][j] = cpyTble.table[i][j];
        }
    }
}
/**
 * destroy class 2D pointer is used to delete all of the dynamically created
 * objects in the class  so need to delete by rows first then delete
 * the entire thing.
 */
template<class T>
TetrisTable<T>::~TetrisTable(){
    for(int i=0;i<row;i++)
    {
        delete []table[i];
    }
    delete []table;
}
/**
 * Function creates a dynamic two dimensional table which is created by the
 * size of ROWS and COLS and sets all elements to 0. After that it then returns
 * the 2D pointer filled with 0s.
 * @return 2D dynamic pointer.
 */
template<class T>
int **TetrisTable<T>::fill_tble(){
    table = new int *[row];
    for (int i=0;i<row;i++){
        table[i] = new int [col];
    }
    /**< Filling the Table. */
    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            table[i][j] = 0;
        }
    }
    return table;
}
/**
 * The function objtPlcmnt is first going take 2D table then since object is
 * of a certain size looking at size of column spots to check for a #. If there
 * is a # in either spot i set rows to that spot to then place object on top
 * of the numbers in table. Uses a break statement to break from going over
 * bounds when placing an object.
 * @param spot user picked spot of user
```

```
 * @param b aggregation of a class to use placement of block
 * @param num number represented with block
 */
template<class T>
void TetrisTable<T>::placeBlock(int spot, CreateBlock &b, int num){
   //column choice of user
   int userChoice = spot-1;
   int rowStart;
   //bRow = b.getRow();
   //bCol = b.getCol();
   //starting from bottom left to top
   for (int i = row-1; i >= 0; i--){
      for (int k = 0; k < b.getCol(); k++)
      {
        if (table[i][userChoice+k] != 0 )//||
        {
          //setting row
          rowStart = i;
        }
      }
    }

    for (int i=rowStart-1; i >= rowStart-b.getRow(); i--){
        for(int j=0; j < b.getCol(); j++)  {
           //checking if spot[col] top of table = 1 if so break from placing
           //one
           if (table[0][userChoice] != 0){
              //breaking from loop cycle
              break;
           }
           else
              table[i][j+userChoice] = num;
        }
     }
}
/**
 * This function is used to output the 2D table to the console that user will
 * be interacting with by placing objects in it. Outputs this table constantly
 * so user can always see it.
 * @param ptr 2D pointer.
 * @param rows size for rows.
 * @param cols size for columns.
 */
template<class T>
void TetrisTable<T>::outputTable() const{
   int count = 0;
```

```
    for (int i = 0; i < row; i++)
    {
       cout << "\t\t";
       for(int j = 0; j < col; j++)
       {
          cout << table[i][j] << "   ";
          count++;
          if (count == col)
          {
             cout << endl;
             count = 0;
          }
       }
    }
    cout << "\t    ---------------------------------------" << endl;
    cout << "Columns \t1   2   3   4   5   6   7   8    "<< endl<<endl;
}
template<class T>
void TetrisTable<T>::newTable(){
    int dstryRow=0; /**< Integer value of row where replacing happens.*/
    int count=0;/**< Integer value of number of rows filled at a time.*/
    for (int i = 0; i < row; i++) {
          if (table[i][0] != 0 && table[i][1] != 0 &&
                table[i][2] != 0 && table[i][3] != 0 &&
                table[i][4] != 0 && table[i][5] != 0 &&
                table[i][6] != 0 && table[i][7] != 0)
          {
             count++;
          }
    }
    cout << "YOU GOT* " << count << " ROWS AT ONCE."<<endl;

       for (int i = 0; i < row; i++)
       {
          if (table[i][0] != 0 && table[i][1] != 0 &&table[i][2] != 0 &&
                table[i][3] != 0 &&table[i][4] != 0 &&table[i][5] != 0 &&
                table[i][6] != 0 &&table[i][7] != 0)
          {
             points += 10*count;
             dstryRow=i;
             /**
              * Had to create table in loop or else same table was getting
              * copied this way new table gets copied with deleted row.
              */
             int **newTble= new int *[row];;/**< Creating new table to copy.*/
             for (int i=0;i<row;i++){
```

```
                newTble[i] = new int [col];
            }
            for (int i =0; i < row; i++){
               for (int j =0; j < col; j++){
                  newTble[i][j] = getElem(i,j); /**< Copying contents.*/
               }
            }
            /*for (int i =0; i < row; i++){
               for (int j =0; j < col; j++){
                  cout << newTble[i][j] <<" ";
               }
               cout <<endl;
            }
            cout <<endl;*/
            //setting row above to = to the row to be deleted
            for (int k = 0; k < dstryRow; k++){
                for (int j =0; j < col ; j++){
                   table[k+1][j] = newTble[k][j];
                }
            }
         }
      }
}
/**
 * The function isOver is checking for a non 0 value on the first row of the
 * table. If it finds this value it instantly breaks from stacking objects on
 * the table that way over bounds issues will not come into play.
 */
template<class T>
bool TetrisTable<T>::game(){
   bool lose;
   for (int j=0; j < col;j++){
      if (table[0][j] != 0)
      {
         lose = false;
         break;
      }
      else
         lose = true;
   }
   return lose;
}
#endif  /* TETTABLE_H */
```