```
/*****************************************************************************************************************
*
 * Program Name:        Cookie Collector v? (fproject.exe)
 * Course:              CIS-61, C++ Language Programming
 * Instructor:          C. Polen
 * Project:             Final Project
 * Created Date:        December 13th, 2010
 * Due Date:            December 14th, 2010
 * Created By:          Chad Philip Johnson
 * Purpose:             A silly game where you must collect all of the cookies in a small area to save the princess
 * Editor/IDE:          Notepad++
 * Resoluton:           1024x768
 * Compiler:            MinGW C++
 * Acknowledgements:    None
*****************************************************************************************************************
*/

#include <iostream>
#include <cstdlib>      //for rand() and srand()
#include <stdlib.h>     //for system()
#include <conio.h>      //for getche()
#include <time.h>
using namespace std;

const int       MAX_HEIGHT = 20;            //height of the grid
const int       MAX_WIDTH = 40;             //width of the grid

const int       MAX_COOKIES = 10;           //number of cookies to be collected
const char      COOKIE_CHARACTER = '@';     //character for the cookies

/*****************************************************************
 * Class: PickUpGame
 * Purpose: To store the grid and the current x and y position of the
 * user. Also has memeber functions to intialize the grid and print it.
 * Allows the user to move around the grid but provides no out of
 * bounds checking.
 *****************************************************************/
class PickUpGame
{
protected:
      char Screen[MAX_HEIGHT][MAX_WIDTH]; //The grid to print to the screen
      int xPos, yPos; //The current x and y position of the users cursor on the grid

public:
      //Constructor that will intialize the screen and x and y positions
      PickUpGame() : xPos(0), yPos(MAX_WIDTH - 1)
      {
            SetupScreen(); //Initalize the grid
      }

```

```
49        //Initialize the screen with all '.' characters and set the intial user cursor position on the grid
50        void SetupScreen()
51        {
52             for(int height = 0; height < MAX_HEIGHT; height++) {
53                  for(int width = 0; width < MAX_WIDTH; width++) {
54                       Screen[height][width] = '.'; //Initialize each grid position
55                  }
56             }
57             Screen[xPos][yPos] = '<'; //Set the users initial cursor position
58        }
59
60        //Print the grid to the screen
61        void Print()
62        {
63             for(int height = 0; height < MAX_HEIGHT; height++) {
64                  for(int width = 0; width < MAX_WIDTH; width++) {
65                       cout << Screen[height][width]; //Print the character at this location in the grid
66                  }
67                  cout << endl; //After each row is printed, print a newline character
68             }
69        }
70
71        //Take in user input to move around the grid
72        void Move(char Direction)
73        {
74             switch(static_cast<int>(Direction)) //Don't know the ASCII characters for the arrow keys so use the ASCII numbers
75             {
76                  case 72: //Up arrow
77                       Screen[xPos][yPos] = ' '; //Wipe out the users current cursor
78                       xPos--; //Move the users x position on the grid
79                       Screen[xPos][yPos] = '^'; //Move the users cursor
80                       break;
81                  case 80: //Down arrow
82                       Screen[xPos][yPos] = ' ';
83                       xPos++;
84                       Screen[xPos][yPos] = 'V';
85                       break;
86                  case 75: //Left arrow
87                       Screen[xPos][yPos] = ' ';
88                       yPos--;
89                       Screen[xPos][yPos] = '<';
90                       break;
91                  case 77: //Right arrow
92                       Screen[xPos][yPos] = ' ';
93                       yPos++;
94                       Screen[xPos][yPos] = '>';
95                       break;
96             }
97        }
98   };
```

```
99
100 /*****************************************************************************************************************
    *
101 * Class:                FindCookies
102 * Purpose:              Inherited from PickUpGame class to place cookies randomly around the playing field and also provide
103                         out of bounds checking
104 *****************************************************************************************************************
    */
105
106 class FindCookies : public PickUpGame   {
107     private:
108         int intTotalCookies;        //total number of cookies for the game
109         int intRemainingCookies;    //remaining cookies still not found by the player
110
111     public:
112         //use FindCookies inherited class constructor to distribute cookies across playing field
113         FindCookies()   {
114             SetupScreen(); //initalize the grid with COOKIES!
115         }
116
117
            /*****************************************************************************************************************
            *
118         * Function Name:        SetupScreen
119         * Parameters:           None
120         * Return Value:         None
121         * Purpose:              Modified version of inherited SetupScreen function that scatters cookies across the playing
122         *                           field to be collected by the player
123
            *****************************************************************************************************************
            */
124         void SetupScreen()  {
125
126             //NOTE:  original playing field with default values is invoked automatically by PickUpGame default constructor
127             //uncomment line below to reset the playing field to default values
128             //PickUpGame::SetupScreen();
129
130             //temporary variables for random values to check for valid location to place a cookie in the array
131             int intTempVerticalLocation, intTempHorizontalLocation;
132
133             intTotalCookies         = (rand() % MAX_COOKIES) + 1;   //random value for number of cookies to appear on playing
                field
134             intRemainingCookies     = intTotalCookies;  //assign same number to remaining number of cookies to collect
135
136             do {
137                 intTempVerticalLocation     = (rand() % MAX_HEIGHT);    //random horizontal location
138                 intTempHorizontalLocation       = (rand() % MAX_WIDTH); //random vertical location
139
140                 //only place a cookie and decrement remaining cookies when random h/v position does not already have
141                 //a cookie and is not the player's starting position
```

```
142                 if ( Screen[intTempVerticalLocation][intTempHorizontalLocation] != COOKIE_CHARACTER &&
143                         Screen[intTempVerticalLocation][intTempHorizontalLocation] != Screen[xPos][yPos] ) {
144                     Screen[intTempVerticalLocation][intTempHorizontalLocation] = COOKIE_CHARACTER;  //place a cookie here
145                     intRemainingCookies--;  //decrement the number remaining to be placed on the playing field
146                 }
147 
148             } while (intRemainingCookies > 0);  //continue loop until no more cookies remain
149 
150             //reassign total cookies to remaining cookies to use when player is hunting for cookies in the game
151             intRemainingCookies = intTotalCookies;
152 
153         }
154 
155 
            /*****************************************************************************************************************
            *
156         * Function Name:        Move
157         * Parameters:           None
158         * Return Value:         None
159         * Purpose:              Modified version of inherited Move function that provides out of bounds checking before moving the
160         *                           player's position
161 
            *****************************************************************************************************************
            */
162         void Move(char* chrTestLocation)    {
163 
164             bool blnContinue    = false;    //value to confirm change of player's position
165 
166             switch( static_cast<int>(*chrTestLocation) )    {
167                 case 72:    //up arrow
168                     //test boundary by temporarily decrementing xPos value by one
169                     //okay when greater than or equal to zero
170                     if (xPos - 1 >= 0)
171                         blnContinue = true; //flag to continue player movement
172 
173                     fncDidIFindACookie(-1, 0);  //check for a cookie, xPos - 1
174                     break;
175 
176                 case 80:    //down arrow
177                     //test boundary by temporarily incrementing xPos value by one
178                     //okay when less than MAX_HEIGHT value
179                     if (xPos + 1 < MAX_HEIGHT)
180                         blnContinue = true; //flag to continue player movement
181 
182                     fncDidIFindACookie(1, 0);   //check for a cookie, xPos + 1
183                     break;
184 
185                 case 75:    //left arrow
186                     //test boundary by temporarily decrementing yPos value by one
187                     //okay when greater than or equal to zero
```

```
                if (yPos - 1 >= 0)
                    blnContinue = true; //flag to continue player movement

                fncDidIFindACookie(0, -1);  //check for a cookie, yPos - 1
                break;

            case 77:    //right arrow
                //test boundary by temporarily incrementing yPos value by one
                //okay when less than MAX_WIDTH value
                if (yPos + 1 < MAX_WIDTH)
                    blnContinue = true; //flag to continue player movement

                fncDidIFindACookie(0, 1);   //check for a cookie, yPos + 1
                break;

        }

        //only move location when within playing field boundaries
        if (blnContinue)
            PickUpGame::Move(*chrTestLocation); //pass to original Move function from inherited class

        //return a QUIT command when there are no more remaining cookies to collect
        if (intRemainingCookies == 0)
            *chrTestLocation = 'q';

    }


    /*****************************************************************************************************************
    *
    * Function Name:        fncDidIFindACookie
    * Parameters:           None
    * Return Value:         None
    * Purpose:              Adjust the xPos and/or yPos temporarily to see if there is a cookie there and decrement remaining
    *                           cookies by one if true

    *****************************************************************************************************************
    */
    void fncDidIFindACookie(int intVerticalAdjust, int intHorizontalAdjust) {   //pass value to be
        if (Screen[xPos+intVerticalAdjust][yPos+intHorizontalAdjust] == COOKIE_CHARACTER)   {
            intRemainingCookies--;  //decrement total remaining cookies to collect by one
        }
    }


    /*****************************************************************************************************************
    *
    * Function Name:        fncReturnTotalCookies
    * Parameters:           None
    * Return Value:         Integer
```

```
232         * Purpose:              Return to calling function the current total number of cookies in the current game
233 
            ********************************************************************************************************************
            */
234         int fncReturnTotalCookies() {
235             return intTotalCookies;
236         }
237 
238 
            /********************************************************************************************************************
            *
239         * Function Name:        fncReturnRemainingCookies
240         * Parameters:           None
241         * Return Value:         Integer
242         * Purpose:              Return to calling function the current remaining cookies to be collected in the current game
243 
            ********************************************************************************************************************
            */
244         int fncReturnRemainingCookies() {
245             return intRemainingCookies;
246         }
247     };
248 
249 int main()  {
250 
251     char chrUserMove        = ' '; //variable to store the users input
252 
253     srand( time(NULL) );    //initiate random seed
254 
255     system("cls");  //clear the screen
256 
257     cout << "Welcome to Cookie Collector v?" << endl;   //game title and version
258     cout << "Get all of the cookies to save the princess!" << endl << endl; //story
259 
260     system("pause");    //pause for user to read intro
261 
262     //loop through game until player decides to stop playing
263     do {
264 
265         FindCookies* Game = new FindCookies; //create a new game object and store it in an object pointer
266 
267         //continue game until all cookies have been collected or user quits
268         do {
269             system("cls"); //clear the screen
270 
271             Game->Print();  //refresh the grid
272 
273             cout << endl;   //instructions to the user
274             cout << "What direction would you like to move in?" << endl;
275             cout << "(Move using the arrow keys or type q to quit.) ";
```

```
276
277             chrUserMove = _getche(); //get one character from the user
278
279             Game->Move(&chrUserMove); //process the user's input
280
281         //keep running the program until the game ends or user types in 'Q' or 'q'
282         } while(chrUserMove != 'Q' && chrUserMove != 'q');
283
284         system("cls"); //clear the screen
285
286         Game->Print(); cout << endl; //print the final grid out to the user and a newline
287
288         //purge last game from memory
289         delete Game;
290
291         if (Game->fncReturnRemainingCookies() == 0) {
292
293             cout << endl;   //newline
294
295             if (Game->fncReturnTotalCookies() == 1) {
296                 //proper text for only one cookie to be collected with a smiley
297                 cout << "Congratulations!  You found the cookie! " << static_cast<char>(1);
298             } else {
299                 //proper text for multiple cookies to be collected with another smiley
300                 cout << "Congratulations!  You found all " << Game->fncReturnTotalCookies() << " cookies! " << static_cast<char>(
                    2);
301             }
302
303             //display result of efforts
304             cout << endl << endl << "I'm sorry, but our princess is in another castle." << endl << endl;
305         }
306
307         cout << "Would you like to play again? (y/n) "; //prompt user to play again
308         chrUserMove = _getche();    //get one character from the user
309
310     //keep restarting game until user types in 'N' or 'n'
311     } while (chrUserMove != 'n' && chrUserMove != 'N');
312
313     //game over
314     return 0;
315
316     }
317
```