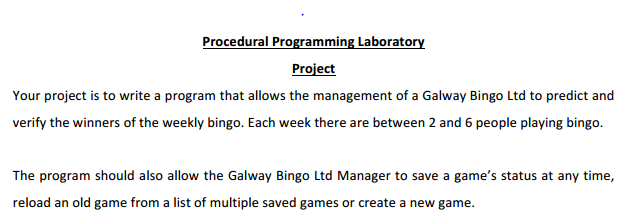
**Sarah Carroll**

**G00330821**

[**G00330821@gmit.ie**](mailto:G00330821@gmit.ie)

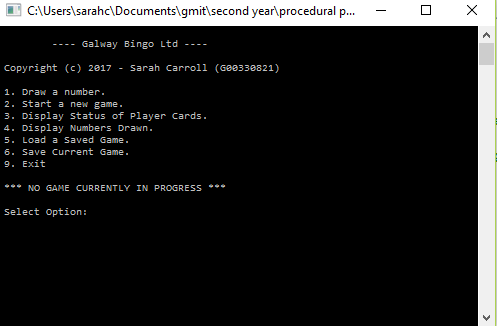
**Procedural Programming - Project 2016**

**Objective:**



**User Interface:**

This project presents the user with a menu of options.



This is achieved by a simple main() function which displays the menu and then handles the option selected.

**Code:**

int main() {

int option;

// Main Program Loop

while (1) {

option = DisplayMenu();

switch (option) {

case 1: DrawNumber();

break;

case 2: InitialiseGame();

break;

case 3: DisplayAllCards();

break;

case 4: DisplayNumbersDrawn();

break;

case 5: LoadGame();

break;

case 6: SaveGame();

break;

case 9: SaveGame();

TidyUp();

printf("\n\nProgram Ended Normally!\n");

exit(0);

};

};

}

**Design Decisions:**

* Player Bingo card data structure.
* Initialisation considerations
* Display layouts
* Drawing and checking bingo numbers
* Loading and Saving games

***Player Bingo Card Data Structure***

A basic bingo card has three lines by nine columns. Therefore, a simple integer array can accommodate this e.g.

int card[3][9];

However recording additional information will make checking numbers easier at a later stage. Thus I have devised a C structure to hold this data as well as the array.

// Bingo Card Structure

typedef struct {

int player; // Player #

int tlc; // Top Left Value

int trc; // Top Right Value

int blc; // Bottom Left Value

int brc; // Bottom Right Value

int card[3][9]; // Bingo Card

} BC;

***Initialisation Considerations***

A number of global variables will be used to hold important information during execution of the program:

// Global Variables

BC \*players = NULL; // Pointer to list of Players

int numPlayers = 0; // Number of Players ( 2 to 6 )

int numbersDrawn = 0; //count numbers drawn

int drawn[90]; // Drawn numbers - each entry set to 1 if number already drawn, zero otherwise

int saveGame = 0; // flag to indicate if game changed and is thus eligible for saving

The InitialiseGame() function will do the following :

* Set all entries in the drawn[90] array to zeroes.

// Initialise Drawn Numbers array to zeroes

for (i = 0; i<90; i++) {

drawn[i] = 0;

}

* Prompt the user for the number of players in the game (2 to 6 inclusive).

// How many players?

printf("Enter number of Bingo Players: ");

scanf("%d", &numPlayers);

if (numPlayers < 2 || numPlayers > 6) {

printf("Number of Players must be bewteen 2 and 6 inclusive!\n\n");

numPlayers = 0;

return;

}

* Allocate enough storage to accommodate the number of players multiplied by the size of the BC structure (using malloc).

// Allocate a Bingo Card structure for each player

players = (BC\*)malloc(numPlayers \* sizeof(BC));

if (players == NULL)

{

printf("Error! Memory for Player Bingo cards cannot be allocated.\n\nProgram ended!");

exit(0);

}

* Initialise the storage for each player using the following steps:

1. Set the player number.
2. Set all four corner variables to zero.
3. Set all 27 cells of the card[3][9] array to -1.

// Initialise basic cards

for (z = 0; z<numPlayers; z++) {

players[z].player = z + 1; // Set Player number

players[z].tlc = 0; // Zeroise the corners

players[z].trc = 0;

players[z].blc = 0;

players[z].brc = 0;

for (i = 0; i<3; i++) {

for (j = 0; j<9; j++) {

players[z].card[i][j] = -1; }

}

}

* Initialise the random seed to ensure random values will be returned subsequently.

srand(time(NULL)); // initialise rand seed

* Randomly allocate four cells on each line to be interpreted as spaces. Set the value of these cells in the array to zero.

// Initialse 4 random space cells on each line of each card

for (z = 0; z<numPlayers; z++) {

for (i = 0; i<3; i++) {

blanks = 4; //4 blanks on each line only

while (blanks>0) {

r = rand() % 9;//get a random integer between 0 and 8

if (players[z].card[i][r] == -1) {

players[z].card[i][r] = 0;// SET CELL TO 0 (SPACE)

blanks--;

}

}

}

* Initialise each column of each player’s card with appropriate random numbers in the correct ranges, ensuring there are no duplicates. Also set the corner elements of the BC structure to the suitable values as they are being generated.

// Initialise each column of each card with appropriate random numbers

for (z = 0; z<numPlayers; z++) {

for (i = 0; i<3; i++) {

for (j = 0; j<9; j++) {

while (1) {

r = (rand() % 10) + 1 + (j \* 10);

//returns a pseudo-random integer between 1 and 10, 11 and 20, 21 and 30, etc

if ((players[z].card[0][j] == r) || (players[z].card[1][j] == r) || (players[z].card[2][j] == r)) {

continue; // get new random #, dont use same number twice!

}

if (players[z].card[i][j] == -1) { // if cell not occupied currently

players[z].card[i][j] = r; // Set CELL random number in specified range

if (i == 0 && players[z].tlc == 0) // set top left corner

players[z].tlc = r;

if (i == 0 && players[z].trc >= 0) // set top right corner

players[z].trc = r;

if (i == 2 && players[z].blc == 0) // set bottom left corner

players[z].blc = r;

if (i == 2 && players[z].brc >= 0) // set bottom right corner

players[z].brc = r;

}

break;

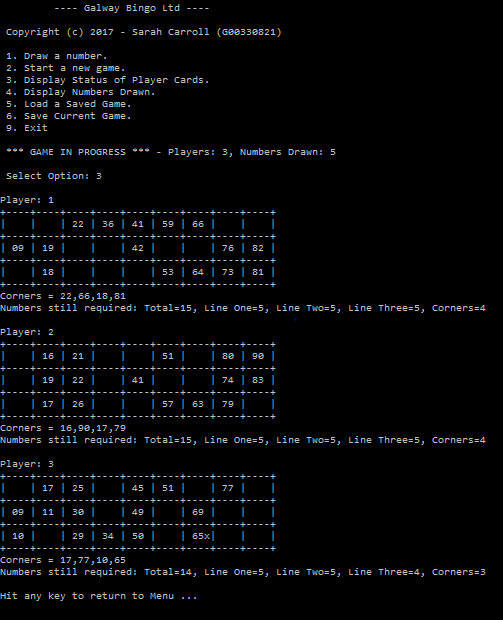
}

}

}

}

***Display layouts***



The above screenshot shows a typical display of bingo cards while a game is in progress. Each player’s card is displayed on a 3x9 grid. It shows four spaces and five numbers on each line as generated randomly by the InitialiseGame() function. It also shows the four numbers at each corner of the grid and the numbers still required to win a prize (one line, two lines, four corners and full house ).

All cards can be displayed ( DisplayCards() function ) or individual player cards ( DisplayCard() ). An ‘x’ appears beside a number in the grid which has already been drawn.

***Drawing and Checking Bingo Numbers***

The DrawNumber() function will pick a random number between 1 and 90. Duplicates are not allowed so it will reselect if a number is generated a second time. The number selected is then marked as used in the drawn[90] array. The function CheckCards() is then called to check if the numbers appear on player cards and notify the user of any winners.

CheckCards() performs the following checks:

* Is there a match on all four corners of a players card?
* Checking each line to see if all the number on the line have been drawn.
* Have numbers been matched on any two lines.
* If all numbers match we have a full house winner.

See code for details of the above checks….

***Loading and Saving games***

Options 5 and 6 on our main menu allow us to load a previously saved game or save the current status of a game in progress.

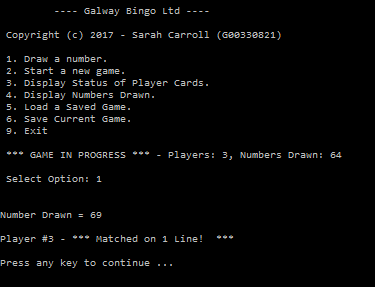
The data saved includes:

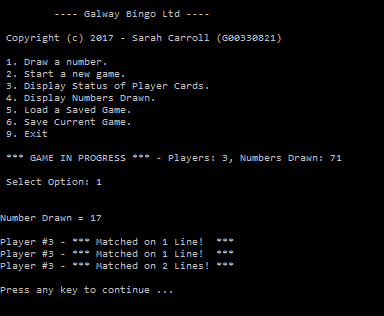
* Number of players in the game (2 to 6).
* The contents of the BC structures for each player.
* Amount of numbers already drawn in the game.
* The contents of the drawn[90] array – zeroes and ones.

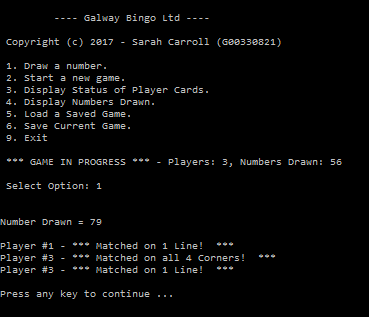
SaveGame() function uses fprintf() to save the values to a file name selected by the user.

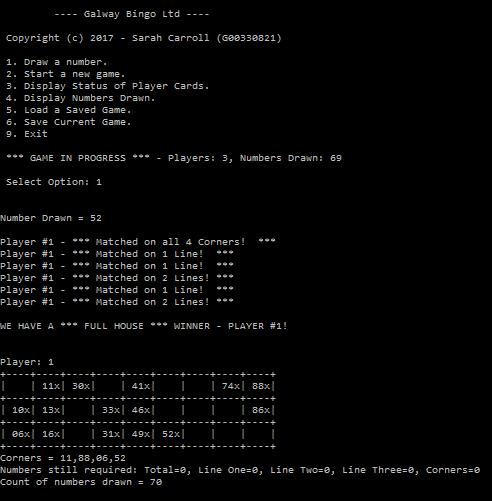
LoadGame() function uses fscanf() to read in the values from the predefined file name.

Sample screenshots showing winning combinations:









**Complete Source Code for project.c:**

// Sarah Carroll

// G00330821

//

// 40440 -- PROCEDURAL PROGRAMMING

//

// Procedural Programming Project 2016

//

#include <time.h>

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

// Bingo Card Structure

typedef struct {

int player; // Player #

int tlc; // Top Left Value

int trc; // Top Right Value

int blc; // Bottom Left Value

int brc; // Bottom Right Value

int card[3][9]; // Bingo Card

} BC;

// Global Variables

BC \*players = NULL; // Pointer to list of Players

int numPlayers = 0; // Number of Players ( 2 to 6 )

int numbersDrawn = 0; //count numbers drawn

int drawn[90]; // Drawn numbers - each entry set to 1 if number already drawn, zero otherwise

int saveGame = 0; // flag to indicate if game changed and is thus eligible for saving

// DisplayMenu function

int DisplayMenu() {

int option;

system("cls"); // clear screen

printf("\n ---- Galway Bingo Ltd ----\n");

printf("\n Copyright (c) 2017 - Sarah Carroll (G00330821)\n");

printf("\n 1. Draw a number.");

printf("\n 2. Start a new game.");

printf("\n 3. Display Status of Player Cards.");

printf("\n 4. Display Numbers Drawn.");

printf("\n 5. Load a Saved Game.");

printf("\n 6. Save Current Game.");

printf("\n 9. Exit");

if (numPlayers == 0)

printf("\n\n \*\*\* NO GAME CURRENTLY IN PROGRESS \*\*\*");

else

printf("\n\n \*\*\* GAME IN PROGRESS \*\*\* - Players: %d, Numbers Drawn: %d", numPlayers, numbersDrawn);

printf("\n\n Select Option: ");

scanf("%d", &option);

return option;

}

// Tidy Up function

void TidyUp() {

if (players != NULL)

free(players);

numPlayers = 0;

numbersDrawn = 0;

}

// DisplayCard function

void DisplayCard(int z) {

int i, j, x;

int required = 0;

int line[3] = { 0,0,0 };

int corners = 0;

printf("\nPlayer: %d ", players[z].player);

printf("\n+----+----+----+----+----+----+----+----+----+\n");

for (i = 0; i<3; i++) {

for (j = 0; j<9; j++) {

x = players[z].card[i][j];

if (x != 0) {

if (drawn[(x - 1)] != 0)

printf("| %02dx", x);

else {

printf("| %02d ", x);

required++;

line[i]++;

}

}

else

printf("| ");

}

printf("|\n+----+----+----+----+----+----+----+----+----+\n");

}

printf("Corners = %02d,%02d,%02d,%02d\n", players[z].tlc, players[z].trc, players[z].blc, players[z].brc);

if (drawn[players[z].tlc - 1] == 0) corners++;

if (drawn[players[z].trc - 1] == 0) corners++;

if (drawn[players[z].blc - 1] == 0) corners++;

if (drawn[players[z].brc - 1] == 0) corners++;

printf("Numbers still required: Total=%d, Line One=%d, Line Two=%d, Line Three=%d, Corners=%d\n", required, line[0], line[1], line[2], corners);

}

// DisplayAllCards function

void DisplayAllCards() {

int z;

for (z = 0; z<numPlayers; z++) {

DisplayCard(z);

}

printf("\nHit any key to return to Menu ...");

getch(); // pause for key enter

}

void InitialiseGame() {

// Variables

int z, i, j, blanks, r;

// Tidy Up any previous game

TidyUp();

// Initialise Drawn Numbers array to zeroes

for (i = 0; i<90; i++) {

drawn[i] = 0;

}

// How many players?

printf("Enter number of Bingo Players: ");

scanf("%d", &numPlayers);

if (numPlayers < 2 || numPlayers > 6) {

printf("Number of Players must be bewteen 2 and 6 inclusive!\n\n");

numPlayers = 0;

return;

}

// Allocate a Bingo Card structure for each player

players = (BC\*)malloc(numPlayers \* sizeof(BC)); //memory allocated using malloc

if (players == NULL)

{

printf("Error! Memory for Player Bingo cards cannot be allocated.\n\nProgram ended!");

exit(0);

}

// Turn on Save Flag to be checked later ...

saveGame = 1;

// Initialise basic cards

for (z = 0; z<numPlayers; z++) {

players[z].player = z + 1; // Set Player number

players[z].tlc = 0; // Zeroise the corners

players[z].trc = 0;

players[z].blc = 0;

players[z].brc = 0;

for (i = 0; i<3; i++) {

for (j = 0; j<9; j++) {

players[z].card[i][j] = -1; // Initialise all card cells to -1

}

}

}

//DisplayAllCards();

srand(time(NULL)); // initialise rand seed

// Initialse 4 random space cells on each line of each card

for (z = 0; z<numPlayers; z++) {

for (i = 0; i<3; i++) {

blanks = 4; //4 blanks on each line only

while (blanks>0) {

r = rand() % 9; //returns a pseudo-random integer between 0 and 8

if (players[z].card[i][r] == -1) {

players[z].card[i][r] = 0; // SET CELL TO 0 (SPACE)

blanks--;

}

}

}

}

//DisplayAllCards();

// Initialise each column of each card with appropriate random numbers

for (z = 0; z<numPlayers; z++) {

for (i = 0; i<3; i++) {

for (j = 0; j<9; j++) {

while (1) {

r = (rand() % 10) + 1 + (j \* 10); //returns a pseudo-random integer between 1 and 10, 11 and 20, 21 and 30, etc

//printf("\nz=%d,i=%d,j=%d,r=%d",z,i,j,r);

if ((players[z].card[0][j] == r) || (players[z].card[1][j] == r) || (players[z].card[2][j] == r)) {

continue; // get new random #, dont use same number twice!

}

if (players[z].card[i][j] == -1) { // if cell not occupied currently

players[z].card[i][j] = r; // Set CELL random number in specified range

if (i == 0 && players[z].tlc == 0) // set top left corner

players[z].tlc = r;

if (i == 0 && players[z].trc >= 0) // set top right corner

players[z].trc = r;

if (i == 2 && players[z].blc == 0) // set bottom left corner

players[z].blc = r;

if (i == 2 && players[z].brc >= 0) // set bottom right corner

players[z].brc = r;

}

break;

}

}

}

}

}

void DrawNumber() {

int r, result;

char ch[5];

if (numPlayers == 0) {

printf("\nSorry, no game currently in progress!\nStart a new game or load an existing game!\nPress any key to return to Menu.\n\n");

getch();

return;

}

if (numbersDrawn == 0)

printf("\nEyes Down ... let's play BINGO!\n\n");

while (numbersDrawn < 90) {

r = rand() % 90; //returns a pseudo-random integer between 0 and 89

if (drawn[r] != 0)

continue; // already drawn - try again

drawn[r] = 1; // mark as drawn

numbersDrawn++; // increment drawn numbers count

saveGame = 1; // new data added - game eligible to be saved.

printf("\n\nNumber Drawn = %d\n", r + 1);

result = CheckCards(); // Anyone won?

if (result > -1) {

printf("\n\nWE HAVE A \*\*\* FULL HOUSE \*\*\* WINNER - PLAYER #%d!\n\n", players[result].player);

DisplayCard(result);

printf("Count of numbers drawn = %d\n", numbersDrawn);

getch();

exit(0);

}

printf("\n\nPress any key to continue ...");

getch();

break;

}

}

int CheckCards() {

//DisplayAllCards();

int z, i, j, x;

int oneline, twolines, fullhouse;

// Check if Numbers on Bingo card have been drawn:

// (Corners, Full House, One Line, Two Lines)

for (z = 0; z<numPlayers; z++) {

// Check for all four corners ...

if (drawn[players[z].tlc - 1] == 1 &&

drawn[players[z].trc - 1] == 1 &&

drawn[players[z].blc - 1] == 1 &&

drawn[players[z].brc - 1] == 1) {

printf("\nPlayer #%d - \*\*\* Matched on all 4 Corners! \*\*\*", players[z].player);

}

// Reset values for each player ...

twolines = 0; // zeroise twolines flag

fullhouse = 1; // expect fullhouse!

for (i = 0; i<3; i++) { // Look at each Line

oneline = 1; // Expect an one line match

for (j = 0; j<9; j++) { // Look at each cell in line

x = players[z].card[i][j];

if (x != 0 && drawn[x - 1] == 0) { // if NOT on card, not FULL HOUSE or ONE LINE

fullhouse = 0;

oneline = 0;

}

}

if (oneline == 1) {

printf("\nPlayer #%d - \*\*\* Matched on 1 Line! \*\*\*", players[z].player);

twolines++;

if (twolines > 1) {

printf("\nPlayer #%d - \*\*\* Matched on 2 Lines! \*\*\*", players[z].player);

}

//DisplayCard(z);

}

}

if (fullhouse == 1)

return z;

}

//printf("\n\nHit enter to continue");

//getch();

return -1;

}

void DisplayNumbersDrawn() {

int i, x, first = 1;

printf("\n\nNumbers Drawn To Date: ");

for (i = 0; i<90; i++) {

if (drawn[i] == 1) {

x = i + 1;

if (first) {

printf("%d", x);

first = 0;

}

else {

printf(",%d", x);

}

}

}

printf("\n\nHit enter to return to Menu ...");

getch();

}

// Prompt user for filename and save game

void SaveGame() {

int p, i, j;

char reply[2];

char filename[100];

FILE \*fp;

if (saveGame == 1) {

// code to prompt for filename and save game data

printf("Do you wish to save this game (Y/N): ");

scanf("%s", &reply);

if (reply[0] == 'Y' || reply[0] == 'y') {

printf("\nEnter name of file to save game to: ");

scanf("%s", filename);

fp = fopen(filename, "w");

if (fp == NULL) {

printf("\nSorry - cannot open file name supplied");

}

else {

// write out player data - Bingo Card details

fprintf(fp, "%d\n", numPlayers);

for (p = 0; p<numPlayers; p++) {

fprintf(fp, "%d ", players[p].player);

fprintf(fp, "%d ", players[p].tlc);

fprintf(fp, "%d ", players[p].trc);

fprintf(fp, "%d ", players[p].blc);

fprintf(fp, "%d\n", players[p].brc);

for (i = 0; i<3; i++) {

for (j = 0; j<9; j++) {

fprintf(fp, "%d ", players[p].card[i][j]);

}

fprintf(fp, "\n"); // newline after each line on card

}

}

// write out drawn number details

fprintf(fp, "%d\n", numbersDrawn);

for (i = 0; i<90; i++) {

fprintf(fp, "%d ", drawn[i]);

}

fprintf(fp, "\n"); // final newline

fclose(fp); // close file

saveGame = 0; // reset save game flag in all cases

printf("\n\nGame saved!\nHit enter to return to Menu ...");

getch();

}

}

}

}

// Prompt user for filename to read and load saved game

void LoadGame() {

FILE \*fp;

char filename[100];

int p, i, j;

if (saveGame == 1)

SaveGame(); // save existing game if one in progress

TidyUp(); // Tidy up existing game, if one exists

printf("\nEnter name of file of game to load: ");

scanf("%s", filename);

fp = fopen(filename, "r");

if (fp == NULL) {

printf("\nSorry - cannot open file name supplied");

}

else {

// read in number of players in saved game

fscanf(fp, "%d", &numPlayers);

players = (BC\*)malloc(numPlayers \* sizeof(BC)); //memory allocated using malloc

if (players == NULL)

{

printf("Error! Memory for Player Bingo cards cannot be allocated.\n\nProgram ended!");

exit(0);

}

// read player data - Bingo Card details

for (p = 0; p<numPlayers; p++) {

fscanf(fp, "%d", &players[p].player);

fscanf(fp, "%d", &players[p].tlc);

fscanf(fp, "%d", &players[p].trc);

fscanf(fp, "%d", &players[p].blc);

fscanf(fp, "%d", &players[p].brc);

for (i = 0; i<3; i++) {

for (j = 0; j<9; j++) {

fscanf(fp, "%d", &players[p].card[i][j]);

}

}

}

// read in drawn number details

fscanf(fp, "%d", &numbersDrawn);

for (i = 0; i<90; i++) {

fscanf(fp, "%d", &drawn[i]);

}

fclose(fp); // close file

saveGame = 0; // ensure save game flag reset

}

}

int main() {

int option;

// Main Program Loop

while (1) {

option = DisplayMenu();

switch (option) {

case 1: DrawNumber();

break;

case 2: InitialiseGame();

break;

case 3: DisplayAllCards();

break;

case 4: DisplayNumbersDrawn();

break;

case 5: LoadGame();

break;

case 6: SaveGame();

break;

case 9: SaveGame();

TidyUp();

printf("\n\nProgram Ended Normally!\n");

exit(0);

};

};

}