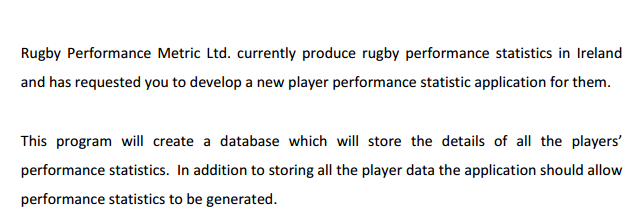
**Sarah Carroll**

**G00330821**

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

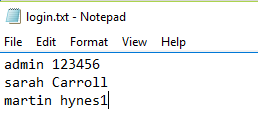
**Advanced Procedural Programming - Project 2017**

**Requirement**



**Application Initialisation**

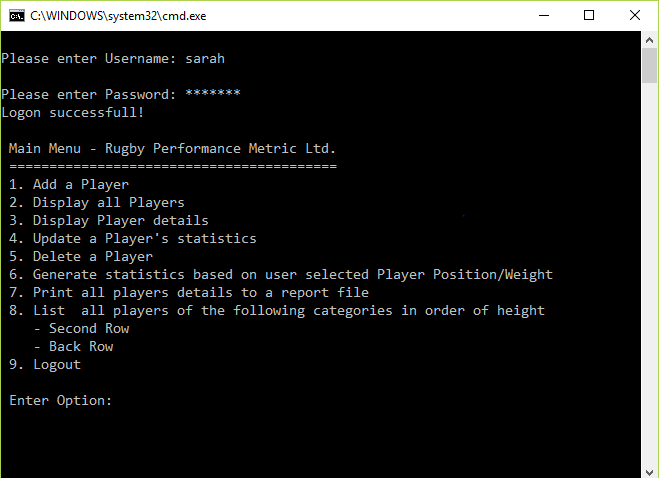
This application requires a user login. The users permitted to use this application are stored in a file called login.txt. A sample login.txt file is included. The contents of which are displayed below.



The user is prompted for username and password on application start-up. While the user name is displayer as is, the password appears as asterisks (\*) as a security measure. The logon code is maintained separately in its own login.c and login.h files.

The login.txt file is read into a singly linked list – and accessed from there.

Once login is successful the following main menu appears to the user.



We will now examine each of these options and highlight some of the design decisions and techniques used to implement the functionality.

**Player Database**

The player information is stored in a C structure with the following definition:

struct node

{

int irfuNumber;

char firstName[30];

char secondName[30];

int age;

int height;

int weight;

char club[20];

char emailAddress[40];

int playerPosition;

int tackles;

int metresRan;

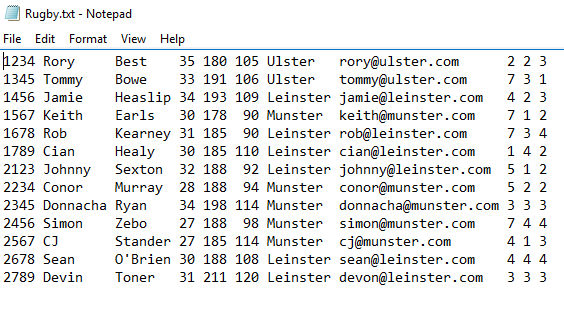
struct node\* NEXT;

};

This is implemented in a singly linked list, maintained in ascending irfuNumber order. The head of the list is stored in a global pointer defined as:

struct node\* head\_ptr = NULL;

The file Rugby.txt contains data saved from this linked list and can be reloaded automatically each time the application is run. A sample Rugby.txt is included. See sample screenshot below:



**Add Player**

When adding a new player the user is prompted for a new IRFU number. It does a search of the database (linked list) for this number and will only allow continuation if the number does not already exist. The search() function is called at this point and at various other points throughout the code:

struct node\* search(int irfuSearch)

{

struct node \*temp;

temp = head\_ptr;

while (temp != NULL)

{

if (irfuSearch == temp->irfuNumber) {

return temp;

}

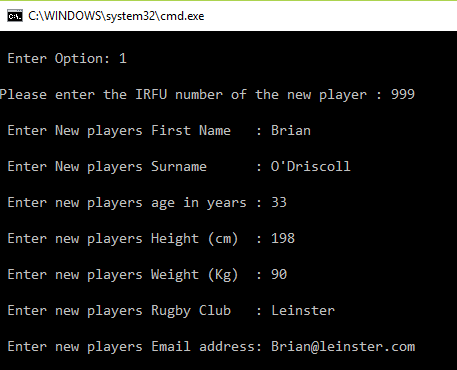
temp = temp->NEXT;

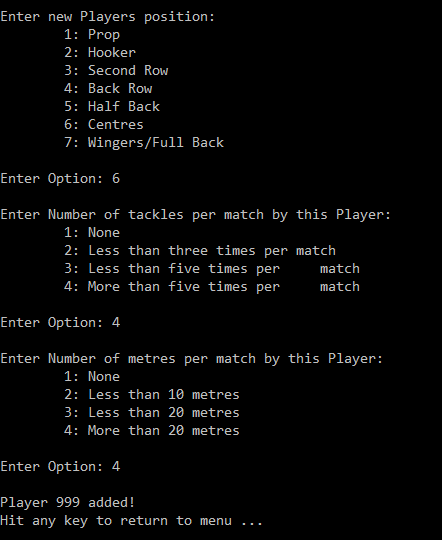
}

return NULL;

}

The following screenshots show typical data entry required for a new player:





Some of the major design decisions used in the addPlayer() function include:

1. The storage for the player node structure is obtained using malloc().
2. Most fields are just populated from scanf() calls, with strings and integers.
3. The email address of the player is validated with the function validateEmail(). This checks for @ symbol and .com in the entered string.
4. Three fields have their own functions to check ranges of values entered -these are player position (e.g. prop, hooker, etc.), tackles made (options 1-4), metres run (options 1-4).
5. The new player is inserted into the linked list based on his IRFU number.

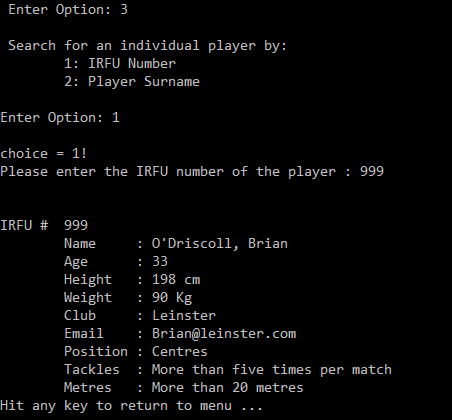
**Display All Players**

This option is implemented in a single function printPlayers(). This loops through the linked list and displays each player’s details.



**Display Player Details**

This option allows the user to display details of an individual player selecting him by either his IRFU number or his surname. Note, if surname is used it will display only the first occurrence of a player with that name. The search for the surname is case insensitive using the stricmp() function.





**Update a Player’s Statistics**

This option allows the user to update the tackles made and metres run by an individual player. The user is asked to input a valid IRFU number and function updatePlayer() is called.

void updatePlayer(struct node \*updNode) {

updNode->tackles = getPlayerTackles();

updNode->metresRan = getPlayerMetres();

change = 1; //mark as changed so it will be saved.

printf("\n\nPlayer %d (%s,%s)updated!",

updNode->irfuNumber,

updNode->secondName,

updNode->firstName);

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

getch();

}//end updatePlayer()

**Delete a Player**

This option removes the entry for a player from the linked list, and frees associated memory.

void deletePlayer(int irfunum)

{

struct node \*temp, \*prev;

temp = head\_ptr;

while (temp != NULL)

{

if (temp->irfuNumber == irfunum)

{

if (temp == head\_ptr)

{

head\_ptr = temp->NEXT;

free(temp);

return;

}

else

{

prev->NEXT = temp->NEXT;

free(temp);

return;

}

}

else

{

prev = temp;

temp = temp->NEXT;

}

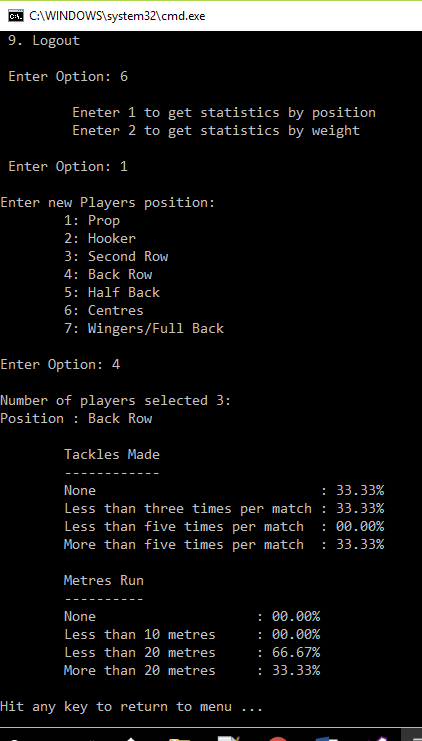
}

return;

}

**Generate statistics based on user selected Player Position/Weight**

This option generates percentage statistics for player position and/or weight. Sample output:



The code for printPositionStats() and printWeightStats() is very similar. They loop through the linked list to find matching position or qualifying weights. They total the number of occurrences of tackles made and metres run for each category. They then produce percentage statistics for each category.

void printPositionStats(int pos) {

int i, j, pcount;

int posStatsA[5];

int posStatsB[5];

struct node\* temp;

for (i = 0; i<5; i++) { // Initialise arrays to zero

posStatsA[i] = posStatsB[i] = 0;

}

temp = head\_ptr;

pcount = 0;

while (temp != NULL) {

if (temp->playerPosition == pos) {

pcount++; // just count players in this POSITION category

posStatsA[temp->tackles]++; // increment appropriate bucket

posStatsB[temp->metresRan]++; // depending on value in linklist

}

temp = temp->NEXT;

}

// pcount = count(); // all players?

printf("\nNumber of players selected %d:", pcount);

if (pcount == 0) pcount = 1; //avoid division by zero!

printf("\nPosition : %s", playerPositions[pos]);

printf("\n\n\tTackles Made");

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

for (j = 1; j<5; j++) {

//printf("\n\tTackles %d:",posStatsA[j]);

printf("\n\t%s\t: %05.2f%%", playerTackles[j], ((float)posStatsA[j] / (float)pcount) \* 100.00);

}

printf("\n\n\tMetres Run");

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

for (j = 1; j<5; j++) {

//printf("\n\tMetres %d:",posStatsB[j]);

printf("\n\t%s\t: %05.2f%%", playerMetres[j], ((float)posStatsB[j] / (float)pcount) \* 100.00);

}

}//end of printPositionStats()

**Print all Player details to a report file**

This option calls function writeToReport() which simply opens an output report file and uses fprintf() to write each entry in the player link list to the file:

void writeToReport() {

FILE\* outp;

struct node \*temp;

temp = head\_ptr;

outp = fopen("Report.txt", "w");

//If file exists

if (outp == NULL)

{

printf("Error opening file!\n");

getch();

return;

}

while (temp != NULL) {

fprintf(outp, "\n\nIRFU # %d", temp->irfuNumber);

fprintf(outp, "\n\tName : %s, %s", temp->secondName, temp->firstName);

fprintf(outp, "\n\tAge : %d", temp->age);

fprintf(outp, "\n\tHeight : %d cm", temp->height);

fprintf(outp, "\n\tWeight : %d Kg", temp->weight);

fprintf(outp, "\n\tClub : %s", temp->club);

fprintf(outp, "\n\tEmail : %s", temp->emailAddress);

fprintf(outp, "\n\tPosition : %s", playerPositions[temp->playerPosition]);

fprintf(outp, "\n\tTackles : %s", playerTackles[temp->tackles]);

fprintf(outp, "\n\tMetres : %s", playerMetres[temp->metresRan]);

temp = temp->NEXT;

}

fclose(outp);

}

**List Players By Order of Height**

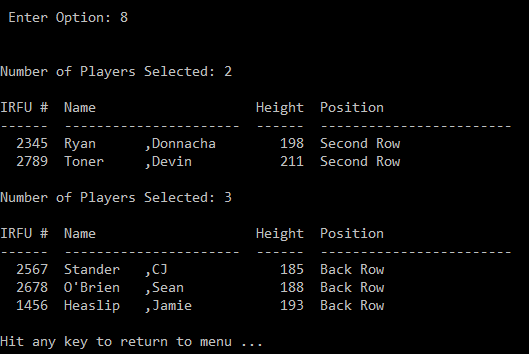
This option presented the greatest technical challenge as a way had to be found not only to select players for two specific positions but also had to reorder the data in ascending height order.

The player link list is in ascending IRFU number order and did not easily lend itself to reordering/sorting. I took a different approach.

I allocated a 2-dimensional integer array to hold the height and IRFU number for each qualifying player. I then used qsort() – a C library supplied function to sort the 2D array in place by the height value only.

It was then a case of looping through the 2D array and selecting the linked list entry for each IRFU number in order and listing the contents in height order.

The screenshot below shows the resulting output and the code which generated this output.



int cmpHeight(const void \*a, const void \*b) {

return ((const int \*)a)[0] - ((const int \*)b)[0];

}

void printPositionByHeight(int pos) {

int i;

int pcount = count(); // get number of items in linklist

int list[50][2]; // set max array size-first col holds heights other hold irfu

struct node\* temp;

for (i = 0; i<50; i++) { // Initialise array to zero

list[i][0] = list[i][1] = 0;

}

temp = head\_ptr;

pcount = 0;

while (temp != NULL) {

if (temp->playerPosition == pos) {

list[pcount][0] = temp->height; // insert height in first array element

list[pcount][1] = temp->irfuNumber; // insert irfu # in second array element

pcount++; // just count players in this POSITION category

}

temp = temp->NEXT;

}

//printf("Before sorting\n");

//for(i=0; i<pcount; i++){

// printf ("list[%d][0] = %d list[%d][1] = %d\n", i, list[i][0], i, list[i][1]);

//}

//printf("AFTER sorting\n");

qsort(list, pcount, 2 \* sizeof(int), cmpHeight);

printf("\n\nNumber of Players Selected: %d", pcount);

printf("\n\nIRFU # Name Height Position");

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

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

// printf ("list[%d][0] = %d list[%d][1] = %d\n", i, list[i][0], i, list[i][1]);

temp = search(list[i][1]);

printf("\n%6d %-10s,%-10s %6d %s",

temp->irfuNumber,

temp->secondName, temp->firstName,

temp->height,

playerPositions[pos]);

}

}//end of printPositionByHeight()

**Logout**

This option checks the global change flag and if set by ADD/UPDATE/DELETE earlier, then it writes the link list contents to Rugby.txt to update the permanent copy of the Database in saveFile() function.