nfl design

WO1 Clayton E. Williams

September 2023

1 Project Overview

nfl loads a player/team database of NFL players from 1960 to 2019. Through the use of command line options, nfl will display statistics and metrics of specified teams or players. Three of the options rely on a BFS, with a requirement to run in a reasonable time. For this reason, a hashtable was determined to be the best data structure to store the player and team structs to provide constant time access. Players will have a list of teams they have played on, and teams will have a roster of players. These are best suited as linked-lists for constant access time to pull each team or player. Original idea for printing teams was to place unique teams in a linked-list and dequeue from there, however, this modifies the head of the linked-list of teams that are shared by other players, leading to persistence issues, so leveraging the previously written trie library that inherently works with unique data was chosen as the better option to print unique teams.

2 Project Structure

Project structure breakdown:

```
.
/doc
   design.pdf
   init_design.pdf
   init_testplan.pdf
   nfl.1
   testplan.pdf
   writeup.pdf
/include
   hashtable.h
   io_helper.h
   llist.h
   long_opts_helper.h
   player.h
```

```
  trie.h
Makefile
README.md
/src
  hashtable.c
  io_helper.c
  llist.c
  long_opts_helper.c
  nfl_driver.c
  player.c
  trie.c
/test
  test_all.c
  test_io_helper.c
  test_llist.c
  test_trie.c
  /test_data
```

doc - documentation for the project, design plan, writeup, man page, and test plan.
include - custom header files for C source code files.
src - C source code files for project.
test - C source code files for unit testing.
test/test_data - various files of size and type to test against, to include default file.

3 Data Structures Needed

Overall, the data structures for players and teams varied very little from the original design plan.

```c
typedef struct player_t {
  char *id;
  char *name;
  char *position;
  char *birthday;
  char *college;
  llist_t *teams;
  int level; // This member was added for the purpose of BFS
  void *parent; // This member was added for the distance flag
} player_t;

typedef struct team_t {
  char *team_name;
  char *year; // Kept as a string to reduce need to convert
```

```
    llist_t *roster;
    void *parent; // This member was added for the distance flag
    int level; // This member was added for the purpose of BFS
  } team_t;
```

4 Functions Needed

The basic functionality to interact with the linked-list, hashtable, and trie remained unchanged from previously written libraries. However, specific void helper functions were added to aid in interpreting void pointers.

```
/*
 Used to find player name matches within the hashtable.
*/
static int compare_player(player_t * player, char *val);

/*
 Used to find player name or college matches within the
 hashtable.
*/
int compare_fields(player_t * player, char *val);

/*
 Called by hashtable_destroy to free the linked-list
 nodes of players teams, as well as the player id, while
 leaving linked-list node data intact.
*/
void player_destroy(player_t * player);

/*
 Called by hashtable_destroy to free the linked list nodes
 of team rosters, leaving the linked list node data intact.
*/
void team_destroy(team_t * team);

/*
 Resets player level to 0 after conducting BFS in support
 of oracle option.
*/
static void reset(player_t * player);

/*
 Resets team level to 0 after conducting BFS in support of
 oracle option.
*/
```

```
static void reset_team(team_t * team);

/*
 This allows hashtable lookup when a player name is provided,
 rather than the player id, which is used as the key.
 compare_player is the custom compare function used here.
*/
void *find_no_key(hash_t * table, char *val, comp_f compare);

/*
 Creates an iterable node from the linked-list, which is
 stack allocated. Allows for iterating over the linked-list
 without modifying head.
*/
int llist_create_iter(llist_t * llist, llist_iter_t * iter);

/*
 Gets the next iterable from the linked-list by referencing
 the next element, without redefining where next points to.
*/
void *llist_iter_next(llist_iter_t * iter);
```

5 Project Flow

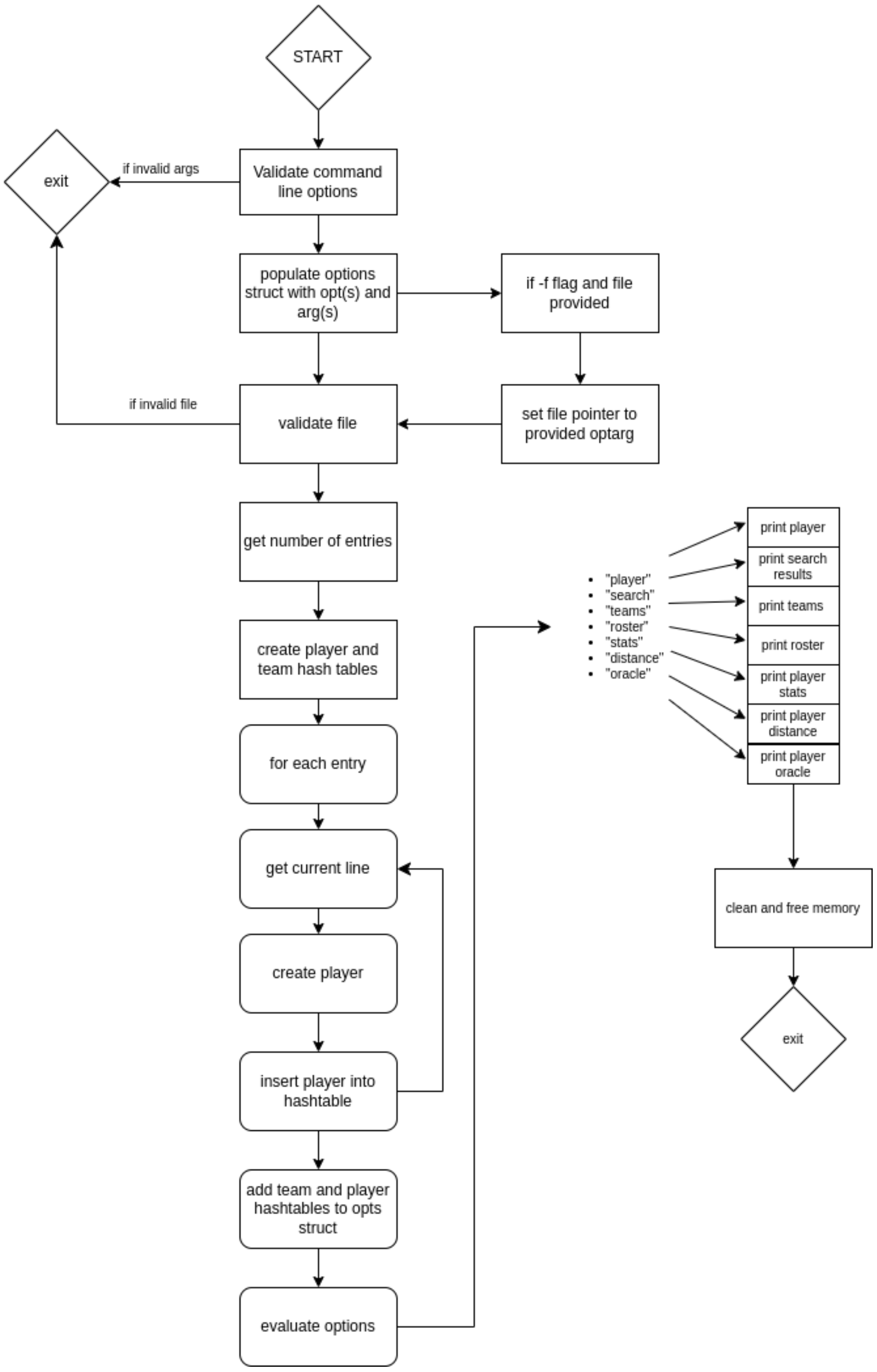

Figure 1: Enter Caption