ECE 209
Fall 2021

Program 3: `golf`

Due Monday, Nov 29 @ 11:59pm

This programming assignment must be completed individually. Do not share your code with or receive code from any other student. Evidence of copying or other unauthorized collaboration will be investigated as a potential academic integrity violation. **The minimum penalty for cheating on a programming assignment is a grade of 0 on the assignment.** If you are tempted to copy because you're running late, or don't know what you're doing, you will be better off missing the assignment and taking a zero. Providing your code to someone is cheating, just as much as copying someone else's work.

DO NOT copy code from the Internet, or use programs found online or in textbooks as a "starting point" for your code. Your job is to design and write this program from scratch, on your own. Evidence of using external code from any source will be investigated as a potential academic integrity violation.

This program manages information about a fictional golf tournament. Information about the course and the players is provided in a file. The program will read the players' shots on each hole, and will extract statistics about each player's performance and the overall rankings in the tournament.

The learning objectives of the program are:

- Implement a program that is spread across multiple files, using both source code files and header files.
- Use struct values, struct pointers, and typedef.
- Create and manipulate a linked list data structure.
- Read data from a file and interpret that data.
- Write code to test whether a function is implemented correctly.

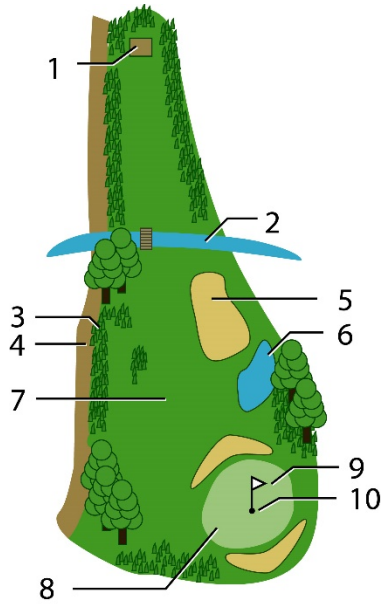
Background

This final programming assignment is built around the notions of linked lists, structs, and abstract data types. You are required to create a linked list, but the operations are fairly simple: just searching/traversing a list and inserting in a specific order. There are no deletions or other reorganization operations.

The program is designed around the notion of a golf tournament. If you are unfamiliar with golf terminology, I will try to define things as I go along, but you should also consult Appendix XXX if you want to see all the terms in one place.

The idea is that an unlimited number of players compete in one round of golf at a particular golf course. A file contains information about the course, and the results of each shot taken by each player. This information is interpreted by your program to produce the total score for each player, the average score of the field, and other interesting statistics.

The game of golf involves striking a ball with a club to advance it from a starting point (the tee) to a regulation-sized cup. The entire field of play from tee to cup is known as a *hole*, but we also talk about the cup being "the hole" where the ball will come to rest. The anatomy of a hole is shown in the figure below. For this program, we are only interested in the following features: (1) tee, (7) fairway, (3) rough, (5) bunker, (8) green, and (10) hole.



The *tee* (1) is the place from which the golfer hits the first stroke. The *fairway* (7) consists of a closely-mowed grass surface from which it is easy to strike the ball. Keeping the ball "in the fairway" makes it easier to control the distance and direction of the next shot. The *green* (8), or putting green, has even shorter grass that makes easy to roll the ball; this type of stroke is known as a *putt*. The hole (10), or cup, is the target: when the ball rolls into the cup, the play for that hole is done.

The area around the fairway is known as the rough (3). In the rough, the grass is higher and the stroke is more difficult to control. There are other features of the hole known as *hazards*, because they make the strokes more challenging. A *bunker* (5) is filled with sand, and a water hazard (2, 6) is a man-made or naturally-occurring pond or stream. We will not have any water hazards for this program. We will also not have an out-of-bounds area (4).

A golf *course* consists of multiple holes, usually 9 or 18. We will always assume 18 holes for this program. For one *round*, the golfer plays the holes in sequential order, starting at 1 and ending at 18. For each hole, the number of strokes required to move the ball from the tee into the cup is the score for that hole, and the total strokes over all holes is the golfer's score for that round.

Each hole has a specified number of strokes that is expected, known as *par*. The par score for a hole is largely dependent on the distance from tee to green, and holes are designated as par-3, par-4, or par-5. If a hole's par is *p*, then the expected play is for the golfer to get the ball onto the green using *p*-2 strokes, followed by two putts to put the ball into the hole.

Instead of the total number of strokes, a golf score is often reported relative to par. For example, a score of -1 means one stroke below par, and a score of +2 means two strokes over par. Par scores can be reported for each hole, as well as for the course as a whole. An 18-hole course typically has a par of 72 strokes, but it could be 71 or 70, depending on the specific mix of par-3, par-4, and par-5 holes on the course.

Once a round is complete, after all golfers have played all holes, the winner is the golfer with the lowest score.

For a more complete description of golf, including rules that I have not described, see this Wikipedia page: https://en.wikipedia.org/wiki/Golf

In addition to score, there are two more statistics that will be tracked by this program.

- *Fairways hit.* Since it is helpful to keep the ball in the fairway, this statistic is designed to capture a golfer's accuracy from the tee. If the first shot ends in the fairway[1], this is called a "hit." The statistic is usually reported as two numbers, such as 10/15 or "10 out of 15," where the second number of the number of par-4 and par-5 holes on the course. A par-3 hole does not have a fairway: the golfer is expected to hit directly from the tee to the green. Therefore, par-3 holes do not count in this statistic. A professional golfer hits the fairway around 65% of the time.
- *Greens in regulation.* As described above, the expectation is that the golfer will get the ball on the green and have two putts to make par. When the golfer ends up on the green[2] in *p*-2 strokes or less, this is known as a "green in regulation" or GIR, because it is the expected outcome. If the golfer hits all greens in regulation, that means their strokes from the tee and the fairway were generally good, and they have a better chance to score par or below on the hole. This statistic is reported as a number of holes, e.g. 15, or as a percentage out of 18 holes, e.g., 83%. A professional golfer hits a green in regulation around 60-70% of the time.

Finally, there are special names for certain scores on a single hole, relative to par:

- Birdie = one stroke under par (-1)
- Eagle = two strokes under par (-2)
- Bogey = one stroke over par (+1)
- Double-bogey = two strokes over par (+2)
- Ace = "hole-in-one," which is the same as Eagle on a par-3 hole

There are other special names that are not important for this program.

Problem Description

This program is concerned with one round of golf, played on one particular day, at one particular golf course. A data file is provided that give the name of the course, the par scores for each hole, and a description of each golfer's play on every hole. A course will always have 18 holes, and the file will always contain information for each golfer on every hole. The number of golfers is not known by the program ahead of time and is considered unbounded. (Your program must work if there are zero golfers or if there are a million.)

A golfer's play on a hole is given by a string of letters, which each letter represents the resting place of the ball after a stroke. The abbreviations are as follows:

- f = fairway

[1] If the ball lands on the green on the first shot of a par-4 or part-5 hole, that would also count as a fairway hit. This is rare, but it can happen.

[2] If the ball lands in the cup within p-2 strokes, without hitting the green, that would also count as a GIR. For example, an ace is a GIR, or the golfer could hit the ball into the cup from the fairway on a par-4 or par-5 hole.

- g = green
- r = rough
- b = bunker
- h = hole

For example, the string "frbggh" means the following sequence: (1) The golfer hits from the tee to the fairway (f). (2) The next stroke ends with the ball in the rough (r). (3) The next stroke ends with the ball in a bunker (b). (4) The next stroke ends with the ball on the green (g). (5) The next stroke also ends on the green (g). (6) The next stroke ends with the ball in the hole. Therefore, the golfer has a score of 6 strokes on this hole.

Based on this information in the file, your program will record the golfer's score for each hole, both as the number of strokes and relative to par. For example, if the hole in the previous paragraph has par = 4, then the stroke score is 6 and the par score is 2 (or +2).

For this program, there is a 10-stroke limit per hole. (This is not the case in real life!) Therefore, every string will end in 'h' unless the golfer takes 10 strokes without getting into the hole.

Note that you can also track fairways hit and greens in regulation using this stroke information.

Program Specification

This program is implemented using three files: main.c, golf.h, and golf.c. You *may not change anything* in **main.c** and **golf.h**. All of your code will be in **golf.c**.

The **main.c** file contains the **main** function, the top-level user interface (UI) for the program. The program will ask the user to enter the name of a file[3] containing course data. After the file data is read, the program enters a command loop, where the user enters commands to extract information about the round. Each command is one character, and some commands also require an argument

This user interface is only provided to help you run your code for testing purposes. It is not run by any of the zyBook tests. For grading, each function described below will be tested independently.

The commands are summarized in the following table:

Command	Meaning	Details
p	Players	Prints scores for all players, in alphabetical order.
l *num*	Leaders	Prints scores for the top *num* players plus ties, in scoring order.
f *name*	Find	Prints score card for the specified player.
s	Statistics	Prints statistics for the course and the round.
q	Quit	Ends the program.

[3] It is also possible to provide the file name as a command-line argument, in which case there is no prompt. See Appendix XXX for instructions on how to do this in CLion.

The program stays in a loop until the user enters the 'q' command. If any other character is entered, the prompt will be printed again. This code may be a bit fragile, so please follow the syntax properly when using the **main** function to test your code.

The **golf.h** file contains function declarations, struct definitions, and typedefs required for your code. This file may not be changed at all, and you will not be allowed to upload **golf.h** when submitting your code to zyBook.

The **golf.c** will contain the implementation of all the functions described below, as well as the definition of the `struct golfCourse` data type, which is needed for the **Course** abstract data type. This is described in the next section.

Data Structures

There are three user-defined data structures used in this program, all of them implemented using C structs. For the first two types, the struct is provided in **golf.h**. For the third, you will be responsible for defining a struct in **golf.c**.

Player

The **Player** type represents a single golfer. It is defined as follows:

```
struct golfPlayer {
    char name[16];   // player's name (max string length is 15)
    Course course;   // course on which this round is being played
    // note: for the following arrays, info about hole N will be
    // stored in array element N-1
    char * strokes[18];   // a string of strokes for each hole
    int strokeScore[18];  // score (# of strokes) for each hole
    int parScore[18];     // score (rel. to par) for each hole
};

typedef struct golfPlayer Player;
```

When a Player is created, the name and the course are provided. The other information should all be initialized to zero, since no stroke information has been entered yet.

To complete the Player data type, the following functions must be implemented:

```
Player * newPlayer(const char *name, Course course);
```

Create and return a new Player value using dynamic memory allocation (malloc). Initialize the name and course according to the arguments, and initialize all of the score and stroke arrays to show that no holes have been played so far.

```
int scoreHole(Player *p, int hole, const char* strokes);
```

Use the strokes string to record information for the specified player on the specified hole. The hole will be an integer between 1 and 18. The strokes string is described above, indicating the location of the ball after each stroke. A new copy of the stroke string must be dynamically allocated (using malloc) and a pointer to the string must be stored in the player's strokes array. The player's strokeScore and parScore arrays must also be updated according to this information.

You may assume that score information will be entered only once for each hole. You may also assume that score information will be entered sequentially for a given player (hole 1, then hole 2, etc.). The sequence is not important for this function, but it could matter on some of the functions below.

```
int totalStrokeScore(const Player *p);
```

Return the total number of strokes for all holes played by the specified player. Because hole information is entered sequentially, you can assume that when you reach a hole with zero score, there are no more scores to be found later in the array.

```
int totalParScore(const Player *p);
```

Return the score relative to par for all holes played by the specified player. Because hole information is entered sequentially you can assume that when you reach a hole with a stroke score (not par score) of zero, there are no more scores to be found later in the array.

```
int greensInReg(const Player *p);
```

Return the number of greens reached in regulation on all holes played by the specified player. See the description of *greens in regulation* in the Background section above.

```
void fairwaysHit(const Player *p, int *hit, int *holes);
```

Determine the number of fairways hit, and the number of holes with fairways, for all holes played by the specified player. Store the number of hits to using the hit pointer, and store the number of fairways holes (par-4 or par-5 holes) using the holes pointer.

```
int countScores(const Player *p, int parScore);
```

Return the number of holes played by the specified player with the specified par score. For example, if parScore is -1, this will return the number of birdies.

PlayerNode

The **PlayerNode** type is used to build a linked list of golfers. It is defined as follows:

```
struct golfPlayerNode {
    Player* player;
    struct golfPlayerNode * next;
};

typedef struct golfPlayerNode PlayerNode;
```

There are no specific functions relating to PlayerNode, but several of the Course functions will require manipulation of a linked list.

Course

The **Course** type is an abstract data type. It is defined as a pointer to a struct, and the user of the type has no information about the details of the struct. The struct must be defined in your **golf.c** file and may be used directly by any code in that file. Any code outside that file, however, can only use the type by calling its functions.

The Course data type is defined as follows:

```
typedef struct golfCourse * Course;
```

The Course-related functions that must be implemented are listed below.

```
Course readCourse(const char * filename);
```

Open and read the specified file to create a Course. If the file cannot be opened, this function must return NULL. See the specifications for the file in the next section.

```
const char * courseName(Course c);
```

Return a pointer to a string containing the name of the course.

```
const int * courseHoles(Course c);
```

Return a pointer to an array of integers that holds the par scores for each hole of the course. The par for hole *x* will be in array element *x-1*.

```
const PlayerNode * coursePlayers(Course c);
```

Return the head of a list of players, in alphabetical order. (All names in the file are unique.) The return value is a const pointer, so this list may be kept in the Course data structure, and the pointer returned by this call. (In other words, there's no need to sort the list when this function is called; just keep the list in sorted order.)

```
PlayerNode * courseLeaders(Course c, int n);
```

Return the head of a list of the top-n players plus ties. This is a newly-created list, and the caller is allowed to alter or destroy the nodes in the list. (In other words, don't use the same nodes in the data structure's list of players.) However, the caller promises to not alter or destroy any player information in the list, so this list can point to the same players as in the list in the previous function. It's just the PlayerNode values that may be destroyed by the caller.

```
int numPlayers(Course c);
```

Return the number of players participating in the tournament.

```
Player * findPlayer(const char * name, Course c);
```

Find and return a player matching the specified name. If no such player is found, return NULL. This is not a const pointer, and the caller may use this pointer to alter information about the player.

```
void addPlayer(Player *p, Course c);
```

Add a new player to the list of players. If the player is already present in the list, no action is taken. While there is no requirement for a specific order, it is recommended that the player be inserted in alphabetical order; this will simplify the **coursePlayers** function above.

```
double avgTotalScore(Course c);
```

Return the average total stroke score for all players.

```
double avgParScore(Course c);
```

Return the average par score for all players.

```
double avgHoleScore(Course c, int hole);
```

Return the average stroke score for all players on the specified hole.

Course File Format

The data file provided for the course will have the structure specified below. There will be no file that does not satisfy these requirements. In other words, you don't need to worry about errors in the file.

The course name is on the first line. The string will be no more than 41 characters, including the linefeed at the end of the line. I recommend that you use **fgets** to read this string, because it may contain spaces.

After the name will be 18 decimal integers, giving the par score for each hole of the course. There will be one more spaces and/or linefeed characters between each integer.

After the hole information will be player information. Information for each hole is provided in the following format: *name*:*hole*:*strokes.*

Each player info will be a contiguous string with no spaces. The fields of the string are separated by a single colon character. The name is at most 15 character. The hole is a decimal integer between 1 and 18. The strokes string is up to 10 characters, described in the Problem Description section.

Recommendation: Read as single string and then extract the information from the string using the colon character as a separator.[4] Use the **atoi** function to convert the hole from a string to an integer.

It is guaranteed that each player will have 18 holes of information in the file, and that each player's information will occur in the order of holes played. In other words, you will see hole 1 before hole 2, and so forth. However, a player's hole information may be interleaved with information for other players. (You may see multiple hole 1's before you see a hole 2, and any interleaving of hole information is allowed as long as each player's holes are sequential. As a model, think about a gaggle of reporters following the players around and calling in the information to the person creating the file.)

It is guaranteed that all player names will be unique.

NOTE: When you store the stroke information, you must allocate a new string. If you're not careful, you will end up overwriting previous strings.

NOTE: When you read a player's name, check whether that player is already in the list. If not, create a new player and add it to the list. If the player is already on the list, then this hole information should be added to that existing player. (Given the information above, you could also use the hole number to detect a new player.)

Developing and Submitting the Program

You will submit this code as a zyBook lab, but you will develop the program outside of the zyBook. It is expected that you will use CLion, but you are free to use whatever development environment that you want.

1. In CLion, create a new project. Specify a C Executable, and use the C11 standard. Name the project whatever you like, but I recommend something meaningful like "prog3" or "golf". This will create a directory in the place where your CLion projects are kept.
2. Get the **main.c**, **golf.h** and **golf.c** files from the zyLab or from the Moodle site. Replace the project's **main.c** file with the downloaded **main.c** file, and add the other files to the top-level project directory.
3. Get the **TorreyPines.txt** file from the Moodle site or the zyLab, and store it in the top-level project directory. Set the Working Directory for the project to make sure the program can find the data files. There will be other files provided by the zyLab.
4. In order to compile and execute the code, you will need to provide some implementation of all required functions, such as readCourse, etc. You can provide "dummy" code that just returns zero or something similar, to get the code to run. This will allow you to work incrementally. You do not need to implement the entire program before testing it. (See suggestions on testing functions below.) To run the zyBooks tests, you will need dummy implementations of ALL of the functions specified above.
5. When you are ready, upload your **golf.c** file to the zyBook assignment and submit. This will run the tests.

[4] You are allowed to use the **strtok** function from the standard library, but there's not much advantage over the functions that we've covered in class. It would literally save you about 3-4 lines of code, and you'd have to learn something new. Just go with what you know and save yourself some time.

If some tests fail, work on your program some more. The output for the failed tests will give you some idea of what's not correct, but use the debugger to figure out what's happening with your implementation.

Several iterations might be necessary. In fact, it's a reasonable strategy to write a program that only passes the first test (or some tests), then improve it to pass the next test, etc. (This even has a fancy name: test-driven development.)

Each function will be tested independently, using what is known as a "unit test." This is a separate test program that only calls your function and checks whether it returns the right information. To write your own test programs, you can add more executable files to your Clion project; I have posted a video to describe how to do this. If you don't want to this in CLion, you can certainly use zyBooks to do this testing for you.

There is no limit to the number of times you can submit. Each submission will overwrite the earlier ones, so make sure that your new code does not break tests that passed earlier.

Hints and Suggestions

- Don't overcomplicate the program. Do not use anything that we have not covered in class.
- Work incrementally. Get one part of the code working, and then move to the next. Each function you add will pass more and more of the tests.
- For compiler errors, look at the source code statement mentioned in the error. Try to figure out what the error is telling you. Try to fix the first error first, and then recompile. Sometimes, fixing the first error will make all the other errors go away. (Because the compiler got confused after the first error.)
- Use a source-level debugger to step through the program if the program behavior is not correct. If you are using CLion on your own computer, the debugger is integrated with the editor and compiler, so there's no excuse for not using it.
- For general questions or clarifications, use Piazza, so that other students can see your question and the answer. For code-specific questions, post a private message to Piazza and attach your code as a file. (Do not copy and paste!)

Administrative Info

Updates or clarifications on Piazza:

Any corrections or clarifications to this program spec will be posted on Piazza. It is important that you read these postings, so that your program will match the updated specification.

What to turn in:

- Submit your **golf.c** file to the zyLab assignment in *15.3 Program 2: golf*.

Grading criteria:

Points	Tests	Functions being tested	Description
20	1	`readCourse, courseName, courseHoles, numPlayers, coursePlayers, findPlayer`	Reads a file with no golfers. Checks that course information is correctly returned.
10	2	`newPlayer, addPlayer, readCourse, coursePlayers`	Reads file with no golfers. Call **newPlayer** to create a new player; this must also call **addPlayer** to add to the course's list of players. Course will be queried to maker sure player is added. Checks that player info is properly initialized.
20	3	`readCourse, numPlayers, coursePlayers, findPlayer`	Reads various files with multiple golfers. Checks player list for names only (scores are ignored for this test).
15	4	`scoreHole, totalStrokeScore, totalParScore, readCourse`	Reads file with no golfers. Adds a player (see above). Adds scores for holes and checks totals and strokes array. NOTE: The holes will be added sequentially, but the totals functions may be called before all 18 holes have been entered. In other words, you must return correct information if less than 18 holes have been scored.
10	5,6	`greensInReg, fairwaysHit, countScores`	Reads various files with multiple players. Checks scoring for various players.
15	7	`avgTotalScore, avgParScore, avgHoleScore, totalStrokeScore, totalParScore`	Reads various files with multiple players. Checks course totals.
10	8,9,10	`courseLeaders`	Reads various files with multiple players. Checks leader list for correct ordering, correct number, correct scores. (Requires all player score functions to work correctly.)

NOTE: The ZyBook tests will total 100 points. The other 10 points for style will be assigned manually by the grader. Please follow the style guidelines, but style will not be graded.

NOTE: Points may be deducted for errors, even for zyBook tests that pass. This will be rare, but it may happen if it is obvious to the grader that the program is written specifically to pass only these tests, and would not pass other similar tests.

NOTE: You will not receive points for tests that are passed by dummy functions. For example, if your function always returns 0, you will not receive credit for tests in which returning 0 is a correct answer.

Do not make assumptions. Write your code to pass any test consistent with this specification. We reserve the right to run your code on tests that were not provided ahead of time.

APPENDIX A – Example run

Apologies for the small font and the wide display for the course stats. Hopefully it will be more readable when you run the program in CLion.

```
Welcome to today's golf tournament.
Course: Torrey Pines, CA
Number of players: 3

After each prompt (?), enter a command.
? p
Mario            70 (-2)
Peach            68 (-4)
Yoshi            86 (+14)
? f Mario
--- Scorecard for Mario ---
Hole:       1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
Par:        4   4   3   4   5   4   4   3   5   5   4   3   4   4   3   4   5   4
Strokes:    4   5   3   4   3   4   3   2   5   6   3   1   6   5   3   4   6   3
Score:      E  +1  +1  +1  -1  -1  -2  -3  -3  -2  -3  -5  -3  -2  -2  -2  -1  -2
Greens in regulation: 10/18 (55.56%)
Fairways hit: 8/14 (57.14%)
Eagles: 2, Birdies: 4, Pars: 7, Bogies: 4, Double-bogies: 1
? f Peach
--- Scorecard for Peach ---
Hole:       1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
Par:        4   4   3   4   5   4   4   3   5   5   4   3   4   4   3   4   5   4
Strokes:    3   4   3   3   5   5   3   3   4   5   5   2   4   4   3   4   4   4
Score:     -1  -1  -1  -2  -2  -1  -2  -2  -3  -3  -2  -3  -3  -3  -3  -3  -4  -4
Greens in regulation: 13/18 (72.22%)
Fairways hit: 12/14 (85.71%)
Eagles: 0, Birdies: 6, Pars: 10, Bogies: 2, Double-bogies: 0
? f Yoshi
--- Scorecard for Yoshi ---
Hole:       1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
Par:        4   4   3   4   5   4   4   3   5   5   4   3   4   4   3   4   5   4
Strokes:    6   7   4   4  10   3   4   4   5   6   5   3   5   4   4   4   5   3
Score:     +2  +5  +6  +6 +11 +10 +10 +11 +11 +12 +13 +13 +14 +14 +15 +15 +15 +14
Greens in regulation: 9/18 (50.00%)
Fairways hit: 7/14 (50.00%)
Eagles: 0, Birdies: 2, Pars: 7, Bogies: 6, Double-bogies: 1
? l 3
  1. Peach            68 (-4)
  2. Mario            70 (-2)
  3. Yoshi            86 (+14)
? l 2
  1. Peach            68 (-4)
  2. Mario            70 (-2)
? s
--- Course statistics ---
Avg strokes: 74.67
Avg score: 2.67
Hole:             1     2     3     4     5     6     7     8     9    10    11    12    13    14    15
16    17    18
Par:              4     4     3     4     5     4     4     3     5     5     4     3     4     4     3
4     5     4
Avg Strokes:   4.33  5.33  3.33  3.67  6.00  4.00  3.33  3.00  4.67  5.67  4.33  2.00  5.00  4.33  3.33
4.00  5.00  3.33
Diff:         +0.33 +1.33 +0.33 -0.33 +1.00 +0.00 -0.67 +0.00 -0.33 +0.67 +0.33 -1.00 +1.00 +0.33 +0.33
+0.00 +0.00 -0.67
? q
```

APPENDIX B – Command-line argument

To pass the file name as an argument to main, do the following:

1. At the top right of CLion, open Edit Configurations. (Just like you do to set the Working Directory.)

2. In the box marked “Program arguments,” put the file name.
3. Click OK.

This will remain in place until it is changed. If you want to change the file name, replace the text in the box. If you want to remove the command-line argument and type in the file name each time, clear the block.

APPENDIX C – Golf glossary

Birdie: A one-hole score of one under par.

Bogey: A one-hole score of one over par.

Bunker: A sand-filled region alongside the fairway or green, or in the middle of the fairway.

Double-bogey: A one-hole score of two over par.

Eagle: A one-hole score of two under par.

Fairway: A closely-mowed grass surface between the tee and the green, from which it is easy to strike the ball.

Fairways hit: For holes that are par-4 or par-5, the number of times the shot from the tee lands in the fairway (or on the green, or in the hole).

Green: A smooth area of very short grass, where it is easy to roll (putt) the ball.

Greens in regulation: The number of holes for which the golfer reaches the green (or in the hole) with at least two strokes remaining for par.

Leaderboard: A listing of players in scoring order, from lowest to highest.

Par: The number of strokes expected to get from the tee to the hole.

Rough: The area around the fairway or green that has higher grass, making it more difficult to strike the ball.

Tee: The place from which the golfer hits the first stroke.