CSC 362 Programming Assignment #3
Due Date: Monday, March 18

You are probably familiar with the story of the hare and the tortoise. The hare, being much faster, should win the race. The hare is over confident and takes a nap. While sleeping, the tortoise overtakes him. In this assignment, you will write a program that will simulate a race between the hare and tortoise, with some variations on the story. The primary purpose of this assignment is to learn how to use pointers, so you will use pointers for the hare and the tortoise and their location in the race, which will be denoted as a string variable we will call path. The path is narrow and will consist of open spaces (‘ ’), rocks (‘R’), streams (‘S’) and carrot patches (‘C’). The game will operate as follows.

In one turn, each animal gets to move. The tortoise will move a random amount from 1-3 per turn. The hare has a 1/3 probability of napping during this turn and thus not moving. Otherwise, if the hare is in a carrot patch and there is a carrot available, it will eat the carrot instead of moving. Otherwise, the hare moves from 1-8. Unlike the tortoise, if the move would place the hare either on top of a rock or in a stream, the hare must move back to the space before the rock or stream (the tortoise can climb the rock or swim the stream but the hare cannot). If either animal’s move lands on the other animal, it must move back one position. If the hare lands on the tortoise and moving back causes the hare to be either on a rock or in a stream, it must move back further to before the rock or stream. Turns continue until someone has crossed the finish line. As both could potentially cross the finish line in the same turn, the winner of the race is whichever animal made it further.

To implement the program, you will write four functions (more can be written if desired) as follows:

- main: initialize the path and the hare and tortoise pointers. The path is a string (see the next page to see its contents) and the hare and tortoise will point into the path. Initially, the hare and tortoise will points at the beginning of the path (e.g., hare = path; or hare = &path[0];) and this is the only time both can be in the same position. Also have three int variables, turn (to count the turn number, starting at 1 and incremented after each turn), collision (a flag used to determine if a collision occurred in this turn) and nap (a flag used to determine if the hare is napping in this turn). main will have a while loop that will iterate while both hare and tortoise are still in the path (their addresses are < path+60). Each animal take its turn by calling separate move functions (described below). Then call an output function to output the current status of the race (again, described below). After the loop, determine and output who won the race.
- moveTortoise: this function moves the tortoise a random amount from 1-3, and determines if the move causes the tortoise to land on the hare and if so, move the tortoise back 1 position. This function must receive both tortoise and hare pointers and the collision flag. It must return the new location of the tortoise. This will be an address in the string (that is, a char *). The function prototype might look like this: `char *moveTortoise(char *, char *, int *);` where the two char * are the hare and tortoise pointers and the int * is the collision flag. The function will end with `return tortoise;`
- moveHare: this function moves the hare. First, it determines if the hare is napping (a 1 in 3 chance) and if so, set the nap flag. Otherwise, if the hare is currently on a ‘C’, the hare takes this turn to eat the carrot (it does not move). Remove the carrot from the path string by setting the character to a blank (‘ ’) (if you don’t do this, the hare will stay here forever). Otherwise, the hare moves a random amount from 1-8. If this lands the hare on the tortoise, move the hare back 1 position. If the initial move, or the move backward, lands the hare on a rock or in the stream, move the hare backward until the hare is not on the rock or stream. If in moving backward, this puts the hare on the tortoise, move it back one position. You will not have to test for another rock/stream here as the path is set up in such a way that streams/rocks are separated by at least a couple of spaces. Return the new location of the hare and set the collision and nap flags as needed.
- print: this function receives the path, the hare and tortoise pointers, the turn, and the two flags (collision and nap). First, output the turn. Then, go through the path character-by-character to output the character, or, if the current position is equal to the hare pointer or tortoise pointer, output

‘H’ or ‘T’. Do not use printf(“%s” path) in this function. After outputting the path, on the same line, output if a collision occurred and/or the hare took a nap. End the function by outputting a ‘\n’.

All access to the path must be handled through pointers, not array references That is, it is not legal to use path[i]. For instance, to test to see if the hare is on a carrot, you would test `*hare==‘C’`. Use pointer arithmetic to move the hare and tortoise and use dereferencing to test where the hare is (carrot, rock, stream) and to print out the path. The only time you should use [ ] is when declaring the path itself.

One last comment. In any turn, it is possible that the hare could move beyond the end of the array. In the moveHare function, you test *hare against ‘C’, ‘R’ and ‘S’. If hare is beyond the end of the array, doing *hare may cause either a runtime error or logical error. Before testing hare, make sure the hare is still within the path. You can test this using `hare < path + 60`. But, while the path is 60 long in this program, you may change the path in the future, so use `#define MAX 60`. Use MAX throughout the program rather than 60 (note: since you are using #define, you do not need to pass MAX as a parameter). Your program can be placed in a single file if desired and you do not need a separate header file, but if you want to split this into more than one file, that is your choice.

Run your program several times until you have at least one win by each animal. Both animals should have close to equal chances of winning (tortoise will probably win more often). Capture two outputs, one where each animal wins) and submitted your source code (commented) and two outputs. Below is the path variable. This is also placed on the website so you can copy and paste it directly into your code. Below is also a sample output. Notice how some of the ‘C’s disappear after the hare lands on them. Although not shown, there could be turns where both the hare is napping and a collision occurs.

```
char path[] = " R  R  R  SSSSS   R  R CCCC  R   R   CCCCCCCCCC  R  SSSS  R ";
```

```
Turn   1: HT  R  R  SSSSS   R  R CCCC  R   R   CCCCCCCCCC  R  SSSS  R  -collision-
Turn   2: HR  T  R  SSSSS   R  R CCCC  R   R   CCCCCCCCCC  R  SSSS  R
Turn   3:  R  R TRH SSSSS   R  R CCCC  R   R   CCCCCCCCCC  R  SSSS  R
Turn   4:  R  R  T HSSSSS   R  R CCCC  R   R   CCCCCCCCCC  R  SSSS  R
Turn   5:  R  R  R HTSSSS   R  R CCCC  R   R   CCCCCCCCCC  R  SSSS  R  -hare napping-
Turn   6:  R  R  R HSTSSS   R  R CCCC  R   R   CCCCCCCCCC  R  SSSS  R
Turn   7:  R  R  R  SSSST H R  R CCCC  R   R   CCCCCCCCCC  R  SSSS  R
Turn   8:  R  R  R  SSSSS  TR HR CCCC  R   R   CCCCCCCCCC  R  SSSS  R
Turn   9:  R  R  R  SSSSS   RT R CHCC  R   R   CCCCCCCCCC  R  SSSS  R  -collision-
Turn  10:  R  R  R  SSSSS   R  T CHCC  R   R   CCCCCCCCCC  R  SSSS  R
Turn  11:  R  R  R  SSSSS   R  RTC CCH R   R   CCCCCCCCCC  R  SSSS  R
Turn  12:  R  R  R  SSSSS   R  R CTCC  R  HR   CCCCCCCCCC  R  SSSS  R
Turn  13:  R  R  R  SSSSS   R  R C TC  R   R   HCCCCCCCCC  R  SSSS  R
Turn  14:  R  R  R  SSSSS   R  R C CT  R   R   HCCCCCCCCC  R  SSSS  R
Turn  15:  R  R  R  SSSSS   R  R C CCT R   R   HCCCCCCCCC  R  SSSS  R  -hare napping-
Turn  16:  R  R  R  SSSSS   R  R C CC  RT  R    CCCCHCCCC  R  SSSS  R
Turn  17:  R  R  R  SSSSS   R  R C CC  R T R    CCCCHCCCC  R  SSSS  R
Turn  18:  R  R  R  SSSSS   R  R C CC  R   T    CCCC HCCC  R  SSSS  R
Turn  19:  R  R  R  SSSSS   R  R C CC  R   R T  CCCC HCCC  R  SSSS  R  -hare napping-
Turn  20:  R  R  R  SSSSS   R  R C CC  R   R   TCCCC HCCC  R  SSSS  R
Turn  21:  R  R  R  SSSSS   R  R C CC  R   R    CCTC  CCC  RH SSSS  R
Turn  22:  R  R  R  SSSSS   R  R C CC  R   R    CCCC TCCC  R HSSSS  R
Turn  23:  R  R  R  SSSSS   R  R C CC  R   R    CCCC  TCC  R HSSSS  R  -hare napping-
Turn  24:  R  R  R  SSSSS   R  R C CC  R   R    CCCC  CTC  R HSSSS  R  -hare napping-
Turn  25:  R  R  R  SSSSS   R  R C CC  R   R    CCCC  CCCT R HSSSS  R  -hare napping-
Turn  26:  R  R  R  SSSSS   R  R C CC  R   R    CCCC  CCC TR HSSSS  R
Turn  27:  R  R  R  SSSSS   R  R C CC  R   R    CCCC  CCC  RT SSSS  RH
Turn  28:  R  R  R  SSSSS   R  R C CC  R   R    CCCC  CCC  R  STSS  R
Hare wins!
```