```
1: //implemented in startup.c
2:     /*
3:     Author: Stefan Mathies
4:     No parameters.
5:     Returns a float corresponding to the distance the user entered on the screen.
6:     */
7:     float getUserDistance();
8:
9:     /*
10:    Author: Emily D'Silva
11:    No parameters.
12:    Initializes and resets sensors, then returns 1 if it succeeded and 0 if it failed.
13:    */
14:    void initializeSensors();
15:
16:
17: //implemented in logger.c
18:    #include "fileLib/PC_FileIO.c" //includes the file library provided by the teaching team
19:
20:    /*
21:    Author: Samuel Mailhot
22:    No parameters.
23:    Creates, then returns, a logfile with the proper name.
24:    */
25:    TFileHandle prepLog();
26:
27:    /*
28:    Author: Kiran Ghanekar
29:    Parameters: int time (the log's timestamp)
30:                            string msg (the message to be written)
31:                            (optional) float numarg or int numarg
32:                                (a numerical argument to log, may or may not be needed)
33:    No returns. Writes a log message to the file returned by prepLog().
34:    */
35:    void sendLog(TFileHandle &logfile, int &time, string &mesg);
36:    void sendLog(TFileHandle &logfile, int &time, string &mesg, float &numarg);
37:    void sendLog(TFileHandle &logfile, int &time, string &mesg, int &numarg);
38:
39:
40: //implemented in tasks.c
41:    /*
42:    Author: Emily D'Silva
43:    Parameters: float dist (the distance to drive)
44:                bool direction (1 for forward, 0 for backward)
45:                bool toStop (1 to stop at the end, 0 to keep moving)
46:                int speed (between 0 and 100)
47:                float &currentdist (reference to a variable containing
48:                                    how far the robot has advanced)
49:                int &time (reference to an integer corresponding to the current time value)
50:                TFileHandle &logfile (reference to a logfile to write to)
51:    Returns whether the robot was or was not moving after driving 'dist' in 'direction'.
52:    */
53:    bool drive(float dist, bool direction, bool toStop, int speed,
54:               float &currentdist, int &time, TFileHandle &logfile);
55:
56:    /*
57:    Author: Samuel Mailhot
58:    Parameters: int &pastRotations (reference to a variable holding the number of
59:                                    times the lead screw has already turned)
60:                bool spinDown (if true, spin the leadscrew all the way back to 0 rotations)
61:    No returns - rotates the lead screw a certain amount based on
```

```
62:                  pastRotations to tension the wheels further.
63:     */
64:     void tensionWheels(int &pastRotations, bool spinDown);
65:
66:     /*
67:     Author: Samuel Mailhot
68:     Parameters: float &currentdist (reference to a variable containing
69:                                     how far the robot has advanced)
70:                 int &time (reference to an integer corresponding to the current time value)
71:                 TFileHandle &logfile (reference to a logfile to write to)
72:     Returns 1 if the procedure succeeded, and 0 if it failed.
73:     */
74:     bool clean(float &currentdist, int &time, TFileHandle &logfile);
75:
76:     /*
77:     Author: Stefan Mathies
78:     Parameters: float &currentdist (reference to a variable containing
79:                                     how far the robot has advanced)
80:                 int &time (reference to an integer corresponding to the current time value)
81:                 TFileHandle &logfile (reference to a logfile to write to)
82:     No returns - runs the escape procedure in accordance with the flowchart.
83:     */
84:     void escape(float &currentdist, int &time, TFileHandle &logfile);
85:
86:     /*
87:     Author: Stefan Mathies
88:     Parameters: int &pastRotations (reference to a variable holding the number of
89:                                     times the lead screw has already turned)
90:                 int &time (reference to an integer corresponding to the current time value)
91:                 TFileHandle &logfile (reference to a logfile to write to)
92:     No returns - runs the shutdown procedure in accordance with the flowchart.
93:     */
94:     void shutdown(int &pastRotations, int &time, TFileHandle &logfile);
95:
96:
97: //implemented in checks.c
98:     /*
99:     Author: Kiran Ghanekar
100:    No parameters.
101:    Returns whether or not a blockage is within range or not.
102:    */
103:    bool ultrasonicDist();
104:
105:    /*
106:    Author: Kiran Ghanekar
107:    Parameters: float &currentdist (reference to a variable holding the distance
108:                                    moved by the robot since power-on)
109:                float &endpoint (reference to the user-input endpoint)
110:                bool didDrive (whether or not the robot moved, from the accelerometer)
111:                int &failures (reference to a variable containing the number of
112:                               health checks that have already failed)
113:                float drivedist (the distance to drive before each health check)
114:                int &time (reference to the current time counter)
115:                TFileHandle &logfile (reference to a logfile to write to)
116:    Returns: 0 if everything passes
117:             1 if the robot should start to leave the pipe
118:             5 if the robot should start cleaning operations
119:             10 if the robot should tension the wheels more
120:    */
121:    int healthCheck(float &currentdist, float &endpoint, bool didDrive, int &failures,
122:                    float drivedist, int &time, TFileHandle &logfile);
```

```
1: //sensor locations
2:     const tSensors USPORT = S1;
3:     const tSensors ACCPORT = S3;
4:     const tSensors TOUCHPORT = S2;
5:
6: //motor locations
7:     const tMotor FDRIVE = motorA;
8:     const tMotor RDRIVE = motorB;
9:     const tMotor BRUSH = motorC;
10:    const tMotor LDSCREW = motorD;
11:
12: //holds the distance, in cm, driven between health checks
13:    const float DRIVEDIST = 5;
14:
15: //holds the distance from the ultrasonic sensor to the front of the robot
16:    const float USOFFSET = 15;
17:
18: //holds the conversion factor for motor encoder count to distance
19:    const float CONV = 180/(PI*2.75);
20:
21: //speed constants
22:    const int SPEEDRAM = 100;
23:    const int SPEEDHIGH = 50;
24:    const int SPEEDLOW = 50;
25:
26: //minimum accelration to be considered moving
27:    const int MINACCEL = 50;
28:
29: //number of times to attempt cleaning
30:    const int HITS = 5;
31:
32: //distance for something to be considered a blockage
33:    const int BLOCKDIST = 40;
34:
35: //how many times to rotate motor for tensioning
36:    const int LDSCREWROTS = 10 * 360 * 24;
37:
38: //what distance to leave at the end while escaping
39:    const int DISTTOLEAVE = 10;
40:
41: //max number of drive check failures during cleaning phase
42:    const int MAXFAIL = 25;
43:
```

```
1: //main.c
2: #include "consts.h"
3: #include "funcs.h"
4: #include "startup.c"
5: #include "tasks.c"
6: #include "checks.c"
7: #include "logger.c"
8:
9: /*
10: Main function.
11: No returns, no parameters.
12: */
13: task main(){
14:     TFileHandle logfile = prepLog();
15:
16:     initializeSensors();
17:
18:     float endpoint = getUserDistance();
19:     wait1Msec(5000);
20:
21:     float currentdist = 0;
22:     int state = 0, time = 0, pastRotations = 0, failures = 0;
23:     bool go = true, didDrive = false;
24:
25:     string mesg = "Robot initialized. Endpoint is ";
26:     sendLog(logfile, time, mesg, endpoint);
27:
28:     //main loop
29:     while (go) {
30:         time++;
31:         didDrive = drive(DRIVEDIST, 1, 0, SPEEDHIGH, currentdist, time, logfile);
32:         state = healthCheck(currentdist, endpoint, didDrive, failures, DRIVEDIST, time, logfile);
33:
34:         if (state == 10){
35:             tensionWheels(pastRotations, 0);
36:             state = 1;
37:         }
38:         else if (state == 5){
39:             if(clean(currentdist, time, logfile)){
40:                 mesg = "Cleaned blockage.";
41:                 sendLog(logfile, time, mesg);
42:             }
43:             else {
44:                 state = 1;
45:                 mesg = "Failed cleaning.";
46:                 sendLog(logfile, time, mesg);
47:             }
48:         }
49:
50:         if (state == 1){
51:             mesg = "Health check failure";
52:             sendLog(logfile, time, mesg);
53:             go = false;
54:         }
55:     }
56:
57:     escape(currentdist, time, logfile);
58:     shutdown(pastRotations, time, logfile);
59: }
60:
```

```
1: //startup.c
2:
3: float getUserDistance() {
4:     float maxdist = 0;
5:
6:     while (!getButtonPress(buttonEnter)) {
7:         displayString(10, "Distance to clear: %1.2f", maxdist);
8:         if (getButtonPress(buttonUp)) {
9:             maxdist += 10;
10:            while(getButtonPress(buttonAny)){}
11:        } else if(getButtonPress(buttonDown) && maxdist >= 10) {
12:            maxdist -=10;
13:            while(getButtonPress(buttonAny)){}
14:        }
15:    }
16:    displayString(5, "Distance to clear: %f", maxdist);
17:    return maxdist;
18: }
19:
20: void initializeSensors() {
21:    SensorType[USPORT] = sensorEV3_Ultrasonic;
22:    wait1Msec(50);
23:
24:    SensorType[TOUCHPORT] = sensorEV3_Touch;
25:    wait1Msec(50);
26:
27:    nMotorEncoder[FDRIVE] = 0;
28:    nMotorEncoder[RDRIVE] = 0;
29:    nMotorEncoder[LDSCREW] = 0;
30:    nMotorEncoder[BRUSH] = 0;
31: }
32:
```

```
1: //tasks.c
2:
3: bool drive(float dist, bool direction, bool toStop, int speed,
4:            float &currentdist, int &time, TFileHandle &logfile) {
5:     bool isMoving = true;
6:     string mesg = "";
7:
8:     nMotorEncoder[FDRIVE] = 0;
9:
10:    if(direction) {
11:        motor[FDRIVE] = speed;
12:        motor[RDRIVE] = -speed;
13:    } else {
14:        motor[FDRIVE] = -speed;
15:        motor[RDRIVE] = speed;
16:    }
17:
18:    while((abs(nMotorEncoder[FDRIVE]) <= dist * CONV)) {}
19:
20:    float acc2 = abs(SensorValue[ACCPORT]);
21:    sendLog(logfile, time, mesg, acc2);
22:
23:    if (direction){
24:        if (acc2 > MINACCEL) {
25:            isMoving = true;
26:            currentdist += dist;
27:        }
28:        else {
29:            mesg = "Failed to drive.";
30:            sendLog(logfile, time, mesg);
31:            isMoving = false;
32:        }
33:    }
34:    else {
35:        currentdist -= dist;
36:    }
37:
38:    if(toStop) {
39:        motor[FDRIVE] = motor[RDRIVE] = 0;
40:    }
41:
42:    return isMoving;
43: }
44:
45: void tensionWheels(int &pastRotations, bool spinDown) {
46:    if(!spinDown) {
47:        nMotorEncoder[LDSCREW] = 0;
48:        motor[LDSCREW] = -100;
49:        while(nMotorEncoder[LDSCREW] < LDSCREWROTS){}
50:        motor[LDSCREW] = 0;
51:        pastRotations += nMotorEncoder[LDSCREW];
52:    }
53:    else {
54:        nMotorEncoder[LDSCREW] = 0;
55:        motor[LDSCREW] = 100;
56:        while(abs(nMotorEncoder[LDSCREW]) < pastRotations){}
57:        motor[LDSCREW] = 0;
58:    }
59: }
60:
61: bool clean(float &currentdist, int &time, TFileHandle &logfile) {
```

```
62:     string mesg = "";
63:     int failures = 0;
64:
65:     for(int i = 0; i < HITS; i++) {
66:         mesg = "Started cleaning";
67:         sendLog(logfile, time, mesg);
68:
69:         drive(15, 0, 1, SPEEDLOW, currentdist, time, logfile);
70:         mesg = "Reversing: ";
71:         sendLog(logfile, time, mesg, i);
72:
73:         motor[BRUSH] = 100;
74:         mesg = "Spinning up brush";
75:         sendLog(logfile, time, mesg);
76:
77:         wait1Msec(1000);
78:
79:         while (SensorValue(TOUCHPORT) != 1){
80:             mesg = "Ramming";
81:             sendLog(logfile, time, mesg);
82:             if(!drive(DRIVEDIST, 1, 0, SPEEDRAM, currentdist, time, logfile)){
83:                 failures++;
84:                 mesg = "Clean failures now at: ";
85:                 sendLog(logfile, time, mesg, failures);
86:                 if (failures >= MAXFAIL){
87:                     motor[BRUSH] = 0;
88:                     return false;
89:                 }
90:             }
91:         }
92:         drive(5, 1, 1, SPEEDRAM, currentdist, time, logfile);
93:         wait1Msec(1000);
94:
95:         if(!ultrasonicDist()){
96:             motor[BRUSH] = 0;
97:             return true;
98:     }
99:     }
100:
101:    motor[BRUSH] = 0;
102:    return false;
103: }
104:
105: void escape(float &currentdist, int &time, TFileHandle &logfile) {
106:    bool acceltrue = true;
107:    int failures = 0;
108:
109:    drive(0, 0, 1, 0, currentdist, time, logfile);
110:
111:    while ((currentdist > DISTTOLEAVE) && acceltrue) {
112:        if (!drive(DRIVEDIST, 0, 0, SPEEDHIGH, currentdist, time, logfile)) {
113:            if (failures > MAXFAIL){
114:                acceltrue = false;
115:            }
116:            else {
117:                failures++;
118:            }
119:        }
120:    }
121:
122:    if (!acceltrue) {
```

```
123:         string message = "Mission Failure: Shutting Down.";
124:         sendLog(logfile, time, message);
125:     }
126:     else {
127:         string message = "Escaping.";
128:         sendLog(logfile, time, message);
129:
130:         while(!getButtonPress(buttonAny)) {
131:             drive(DRIVEDIST, 0, 0, SPEEDLOW, currentdist, time, logfile);
132:         }
133:     }
134:     drive(0, 0, 1, 0, currentdist, time, logfile);
135: }
136:
137: void shutdown(int &pastRotations, int &time, TFileHandle &logfile) {
138:     tensionWheels(pastRotations, 1);
139:     string mesg = "Shut down.";
140:     sendLog(logfile, time, mesg);
141: }
142:
```

```
1: //checks.c
2:
3: bool ultrasonicDist(){
4:     return (SensorValue[USPORT]-USOFFSET) < BLOCKDIST;
5: }
6:
7: int healthCheck(float &currentdist, float &endpoint, bool didDrive, int &failures,
8:                 float drivedist, int &time, TFileHandle &logfile){
9:     int state = 0;
10:    string mesg = "";
11:
12:    if (currentdist >= endpoint){
13:        state = 1;
14:        mesg = "Reached endpoint.";
15:        sendLog(logfile, time, mesg);
16:    }
17:    else if (didDrive){
18:        if (ultrasonicDist()){
19:            state = 5;
20:            mesg = "Ultrasonic triggered";
21:            sendLog(logfile, time, mesg);
22:        }
23:        else {
24:            state = 0;
25:            mesg = "Business as usual";
26:            sendLog(logfile, time, mesg);
27:        }
28:    }
29:    else {
30:        if (failures > MAXFAIL){
31:            didDrive = drive(DRIVEDIST, 0, 1, SPEEDHIGH, currentdist, time, logfile);
32:            if (didDrive) {
33:                mesg = "Backwards driving worked.";
34:                sendLog(logfile, time, mesg);
35:                state = 1;
36:            }
37:            else {
38:                mesg = "Time to tension.";
39:                sendLog(logfile, time, mesg);
40:                state = 10;
41:            }
42:        }
43:        else {
44:            failures++;
45:            mesg = "HC failures now at: ";
46:            sendLog(logfile, time, mesg, failures);
47:        }
48:    }
49:
50:    return state;
51: }
52:
```

```
1: //Logger.c
2:
3: TFileHandle prepLog() {
4:     TFileHandle fout;
5:     openWritePC(fout, "log.txt");
6:     return fout;
7: }
8:
9:
10: void sendLog(TFileHandle logfile, int time, string mesg){
11:     writeLongPC(logfile, time);
12:     writeTextPC(logfile, "\t");
13:     writeTextPC(logfile, mesg);
14:     writeEndlPC(logfile);
15: }
16:
17: void sendLog(TFileHandle logfile, int time, string mesg, float &numarg){
18:     writeLongPC(logfile, time);
19:     writeTextPC(logfile, "\t");
20:     writeTextPC(logfile, mesg);
21:     writeTextPC(logfile, "\t");
22:     writeFloatPC(logfile, numarg);
23:     writeEndlPC(logfile);
24: }
25:
26: void sendLog(TFileHandle logfile, int time, string mesg, int &numarg){
27:     writeLongPC(logfile, time);
28:     writeTextPC(logfile, "\t");
29:     writeTextPC(logfile, mesg);
30:     writeTextPC(logfile, "\t");
31:     writeLongPC(logfile, numarg);
32:     writeEndlPC(logfile);
33: }
34:
```