```
// Amount of time to let the robot run.
#define RUNTIME 30.0
// Direction control flags.
#define LEFT 0
#define RIGHT 1
// Motor power level definitions.
#define HIGH 80
#define MID 60
#define LOW 40

int irRead[5] = {0,0,0,0,0}; // Array used to store the IR sensor readings.
int hit = -1; // Variable used to indicate where a light source has been detected
int oldHit = hit;
int timeup = 0; // Flag used to signal when time has expired.

// Variables used to store the process ID's of the three main processes.
int timerPID;
int lightLocatorPID;
int controlMotorsPID;

// The timer function waits for a specified length of time and then switches the
// timeup flag.
void timer( float t ){
    float startTime = seconds();
    while( seconds() - startTime < t ){  // While time has not expired.
        defer();  // Relinquish control of the processor.
    }
    timeup = 1; // Once time has expired, change the flag, and exit the function.
} // timer( float t )

// The lightLocator function sets the variable hit to an integer representing
// the IR sensor that is detecting the most IR light.
// -1 indicates that no meaningful hit was detected.
// 0 through 4 indicate which sensor has a hit.
void lightLocator() {
    int i;        // A counter index.
    int minRead;  // The lowest IR reading.
    int maxRead;  // The highest IR reading.
    int maxIndex; // The index (sensor number) of the highest IR reading.

    while(!timeup){
        // Reset tracking variables.
        minRead = 256;
        maxRead = -1;
        maxIndex = -1;
        // Loop through each sensor to obtain the readings.
        for(i=0; i<=4; i++){
            irRead[i] = 255 - analog(i); // ****

            if( irRead[i]<minRead ){     // ****
                minRead = irRead[i];
            }
            if( irRead[i]>maxRead ){     // ****
                maxIndex = i;
                maxRead = irRead[i];
            }
        }
        // Process the readings to see if a meaningful hit was found.
        if( /* Enter a Boolean statement that recognizes a true hit. */ ){
            hit = maxIndex;
        }
        else{ hit = -1; } // ****
    }
} // lightLocator()


```

```
 67: // The controlMotors function controls the motors based on the value of "hit"
 68: // which is continuously updated by lightLocator().
 69: void controlMotors()
 70: {
 71:     while(!timeup){ // As long as time is left, continue updating the motor commands.
 72:         if ( hit != oldHit){ // ****
 73:             oldHit = hit;
 74:             if(hit == -1){ // No hit; what should you do?
 75:                 defer();
 76:             }
 77:             if(hit == 0){  // Turn left quickly
 78:                 defer();
 79:             }
 80:             if(hit == 1){  // Turn left slowly
 81:                 defer();
 82:             }
 83:             if(hit == 2){  // Go forward
 84:                 defer();
 85:             }
 86:             if(hit == 3){  // Turn right slowly
 87:                 defer();
 88:             }
 89:             if(hit == 4){  // Turn right quickly
 90:                 defer();
 91:             }
 92:         }
 93:     }
 94:     // Once time has expired kill the engines and end the function.
 95:     alloff();
 96: } // controlMotors()
 97:
 98: // Main simply starts processes for the controlling functions above.
 99: void main()
100: {
101:     while(1){
102:         printf("Press start to  chase light.\n");
103:         while( !start_button() ){  } // ****
104:
105:         printf("Chasing!!\n");
106:         reset_system_time();  // ****
107:         timeup = 0;           // ****
108:
109:         timerPID = start_process( timer( RUNTIME ) );
110:         lightLocatorPID = start_process( lightLocator() );
111:         controlMotorsPID = start_process( controlMotors() );
112:
113:         while(!timeup && !stop_button() ){  // ****
114:             // Add debugging commands here as needed.
115:             defer();
116:         }
117:
118:         // ****
119:         kill_process(controlMotorsPID);
120:         kill_process(lightLocatorPID);
121:         kill_process(timerPID);
122:         alloff();
123:         beep();
124:     }
125: } // main()
126:
```