```
1  //      grader.cpp
2  //
3  //      Rhythmic Learning Tool
4  //      Fall 2017
5  //      Kenneth Hall
6  //
7  //      This program is part of a project that uses a web-app
8  //      and a Raspberry Pi based controller to teach
9  //      rhythmic structure in music.
10 //
11 //      It must deconstruct an array of strings into
12 //      a lesson key and key press history.
13 //      Afterwards, it evaluates and returns a grade.
14 //
15
16 #include    ...
17 #define     ...
18
19 long long int key[KEY_SIZE][KEY_NUM_COLUMNS];
20 long long int history[LARGE_NUMBER][HISTORY_NUM_COLUMNS];
21
22 //  debugging key formatting:
23 //  [button #]  [note len]  [low start time][high start time]   [low stop time] [high stop time]
24 //  [0]         [1]         [419871]        [519871]            [619871]        [719871]
25 //  [0]         [1]         [790000]        [810000]            [840000]        [850000]
26 void printKey() { ...
27
28
29 //  history formatting:
30 //  [button number][time button pressed][time button released]
31 //  [...]
32 void printHistory(int size) { ...
33
34
35 /* format and store a lesson key into global: key */
36 void parseKey(char* argv[], int bpm) { ...
37
38 /* convert string to int */
39 long long int atoi(char* a[], int b) { ...
40
41 /* format and store a lesson key into global: history */
42 void parseHistory(char* key[], int asize) { ...
43
44
45 //  main parameter formatting:
46 //
47 //      1. argc = array length
48 //
49 //      2. argv is a array of strings:
50 //          [64 #s indicating duration(0 - 4 only),
51 //          bpm,
52 //          {(pi button, up / down, time pressed in microsec), (repeated for all recorded
     events), ...}]
53 //
54 extern "C" int main(int argc, char **argv) {
55
56     // check arguments for valid formatting
57
58     printf("argv: \n");
59     for (size_t i = 0; i < argc; i++) {
60         printf("%d: %lld\n", i, atoi(argv, i));
```

```
61          assert(atoi(argv, i) <= 600000000 && "string too large... contains a value > 10 min
              (600000000)");
62      }
63      printf("argc: %d\n\n", argc);
64
65      assert(argc >= 65 && "input array size is too small");
66      assert((argc - 65) % 3 == 0 && "recording data is not div 3");
67
68      int bpm = atoi(argv, 64);
69      assert(bpm >= 40 && bpm <= 208 && "bpm must be between 40 and 208");
70
71      int sum = 0;
72      int sumrow = 0;
73      for (size_t i = 0; i < 64; i++) {
74          sum += atoi(argv, i);
75
76          if (i % 4 == 0 && i > 0) {
77              //printf("i: %d\n", i);
78              //printf("row duration: %d\n\n", sumrow);
79              sumrow = 0;
80          }
81          sumrow += atoi(argv, i);
82          assert(sumrow >= 0 && sumrow <= 4 && "key: a row's total duration was incorrect");
83      }
84      //printf("total duration: %d\n\n", sum);
85      assert(sum >= 0 && sum <= 64 && "key: total note length was incorrect");
86
87      assert(HISTORY_SIZE <= LARGE_NUMBER && "history size too large; increase 'LARGE_NUMBER' in
          grader.cpp");
88
89      parseKey(argv, bpm);
90      printKey();
91
92      parseHistory(argv, argc);
93      printHistory(HISTORY_SIZE);
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
```

```
120
121     //  grade
122     //
123     //  compare all entries in HISTORY_SIZE against KEY_SIZE
124     //  check press and releases times are w/i limits
125
126     float correct = 0.0;
127     for (size_t i = 0; i < HISTORY_SIZE; i++) {
128         for (size_t j = 0; j < KEY_SIZE; j++) {
129             if (history[i][0] == key[j][0] && // same button number
130                     (key[j][1] != 0) &&
131                 history[i][1] >= key[j][2] && // start time above start_min
132                 history[i][1] <= key[j][3] && // start time below start_max
133                 //history[i][2] >= key[j][4] && // stop time above stop_min
134                 //history[i][2] <= key[j][5] &&  // stop time below stop_max
135
136                 // &&   i != j
137                 key[j][2] != 0 && // ignore blank key entries
138                 key[j][3] != 0 &&
139                 key[j][4] != 0 &&
140                 key[j][5] != 0
141                 ) {
142                 correct += 1.0;
143                 printf("correct row: %d\n", i);
144                 break;
145             }
146         }
147     }
148     int keysize = KEY_SIZE;
149     for (size_t i = 0; i < KEY_SIZE; i++) {
150         if (key[i][1] == 0) {
151             keysize -= 1;
152         }
153     }
154
155     printf("\ncorrect: %.0f of %d\n", correct, keysize);
156     printf("grade: %.2f %%\n", correct / keysize * 100.0);
157     printf("return value: %d\n", (int)(correct / keysize * 100.0));
158     return (int)(correct / keysize * 100.0);
159 }
160
```

```
/* format and store a lesson key into global: key */
void parseKey(char* argv[], int bpm) {
    long long int DELTA = ((DELTA_BASE * 60) / 2) / bpm;
    long long int spb = MU_S * 60 / bpm;
    for (size_t i = 0; i < KEY_SIZE; i++) {
        for (size_t j = 0; j < KEY_NUM_COLUMNS; j++)
        {
            key[i][j] = 0;
        }
    }

    for (size_t i = 0; i < KEY_SIZE; i++) {
        key[i][0] = i % 16; // button #
        key[i][1] = argv[i][0] - 48; // note type

        key[i][2] = (i + 1) * spb - DELTA + latency1;// -(60 * latency1 / bpm); //start lo
        key[i][3] = (i + 1) * spb + DELTA + latency1;// -(60 * latency1 / bpm); //start hi

        long long int offset = (spb * key[i][1]);
        key[i][4] = (i + 1) * spb - DELTA + offset + latency2;// -(60 * latency2 / bpm); // stop lo
        key[i][5] = (i + 1) * spb + DELTA + offset + latency2;// -(60 * latency2 / bpm); // stop hi
    }
}


/* format and store a lesson key into global: history */
void parseHistory(char* key[], int asize) {
    for (size_t i = 0; i < LARGE_NUMBER; i++) {
        for (size_t j = 0; j < HISTORY_NUM_COLUMNS; j++) {
            history[i][j] = 0;
        }
    }

    long long int  history_temp[LARGE_NUMBER][3];
    int history_size;

    int i = 0;
    for (size_t j = 65; j < asize; j += 3) {
        history_temp[i][0] = atoi(key, j); // button #
        history_temp[i][1] = atoi(key, j + 1); // up/down // time pressed
        history_temp[i][2] = atoi(key, j + 2); // time     //time released
        i += 1;
    }

    history_size = i;
    int k = 0;
    for (size_t i = 0; i < history_size; i++) { // take pi record and pair up key presses. assumptions: cant press same key twice without releasing inbetween
        if (history_temp[i][1] == 1) { // if down press
            for (size_t j = 0; j < history_size; j++) { // look for matching release
                if (history_temp[j][1] == 0 && // is a release
                    history_temp[j][0] == history_temp[i][0] && // same button numbers
                    history_temp[i][2] < history_temp[j][2] // start is less than stop
                    ) {
                    history[k][0] = history_temp[i][0];
                    //history[k][1] = history_temp[i][2]; // LEAVE ALONE
                    history[k][1] = history_temp[i][2] - 1184000 + 2285000 - 310000;//815000; // LEAVE ALONE
                    history[k][2] = history_temp[j][2] - 1335000 + 2420000 - 250000;//886000;
                    k += 1;
                    break;
```

```
60                  }
61              }
62          }
63      }
64  }
```