```
001 //Власов Роман Евгеньевич
002 //группа 250541
003
004 // Определение макроса uspeech_h
005 #define uspeech_h
006 // Подключение заголовочного файла Arduino.h для доступа к функциям
    платформы Arduino
007 #include "Arduino.h"
008 // Подключение библиотеки для работы со строками
009 #include <string.h>
010 // Подключение математической библиотеки для использования математических
    функций
011 #include <math.h>
012 // Подключение стандартной библиотеки для общего функционала
013 #include <stdlib.h>
014
015 // Определение порога тишины
016 #define SILENCE 2000
017 // Определение константы F_DETECTION для обнаружения фонемы /f/
018 #define F_DETECTION 3
019 // Определение константы F_CONSTANT (порог для /f/)
020 #define F_CONSTANT 350
021 // Определение константы MAX_PLOSIVETIME (максимальное время для
    определения взрывного звука)
022 #define MAX_PLOSIVETIME 1000
023 // Определение константы PROCESS_SKEWNESS_TIME (период анализа данных)
024 #define PROCESS_SKEWNESS_TIME 15
025
026 /**
027  * Класс для распознавания звука
028  */
029 class signal {
030 public:
031     // Объявление массива для хранения аудиоданных (буфер из 32 выборок)
032     int arr[32];
033     // Объявление переменной для средней мощности сигнала
034     int avgPower;
035     // Объявление тестового коэффициента для отладки
036     int testCoeff;
037     // Объявление минимального значения громкости, после которого сигнал
    считается готовым к распознанию
038     int minVolume;
039     // Объявление порога для распознавания фонемы /f/ (настраиваемый)
040     int fconstant;
041     // Объявление порога для распознавания фонем, таких как /ee/ или /i/
042     int econstant;
043     // Объявление порога для распознавания фонем, таких как /a/, /o/, /r/
    или /l/
044     int aconstant;
045     // Объявление порога для распознавания фонем, таких как /z/, /v/ или
    /w/
046     int vconstant;
047     // Объявление порога для распознавания фонем, таких как /sh/ или /ch/
    (значения выше - фонема /s/)
048     int shconstant;
049     // Объявление флага, разрешающего распознавание фонемы /f/
050     bool f_enabled;
051     // Объявление коэффициента усиления для регулировки чувствительности
052     int amplificationFactor;
053     // Объявление порога мощности микрофона (значения ниже игнорируются)
054     int micPowerThreshold;
055     // Объявление масштабного коэффициента для входного сигнала
056     int scale;
```

```
057     // Объявление переменной для хранения распознанной фонемы (символ)
058     char phoneme;
059     // Объявление конструктора класса, принимающего номер порта
060     signal(int port);
061     // Объявление переменной для хранения мощности микрофона
062     int micPower;
063     // Объявление метода для выборки звука
064     void sample();
065     // Объявление метода для определения максимальной мощности сигнала
066     unsigned int maxPower();
067     // Объявление метода для вычисления общей мощности сигнала
068     unsigned int power();
069     // Объявление метода для вычисления отношения сигнал/шум (SNR)
070     int snr(int power);
071     // Объявление метода для калибровки микрофона (определение фонового
    уровня)
072     void calibrate();
073     // Объявление метода для распознавания фонемы (возврат символа)
074     char getPhoneme();
075     // Объявление переменной для хранения значения калибровки (средний
    уровень фонового шума)
076     int calib;
077 private:
078     // Объявление переменной для хранения номера порта, к которому
    подключён микрофон
079     int pin;
080     // Объявление переменной для хранения времени (миллисекунды) - может
    использоваться для отладки или тайминга
081     int mil;
082     // Объявление переменной для хранения позиции в массиве с максимальной
    амплитудой
083     int maxPos;
084     // Объявление флага, указывающего на наличие тишины
085     bool silence;
086     // Объявление массива для хранения истории коэффициентов
    (используемого для фильтрации шума)
087     unsigned int overview[7];
088     // Объявление приватного метода для вычисления «сложности» сигнала
    (отношение суммы модулей производных к мощности)
089     unsigned int complexity(int power);
090 };
091
092 /**
093  * Класс для накопления статистических показателей
094  */
095 class statCollector {
096 public:
097     // Объявление переменных для количества выборок, среднего значения и
    моментов 2-го, 3-го и 4-го порядка
098     int n, mean, M2, M3, M4;
099     // Объявление конструктора класса
100     statCollector();
101     // Объявление метода для вычисления куртоза (остроты распределения)
102     int kurtosis();
103     // Объявление метода для вычисления асимметрии распределения
104     int skew();
105     // Объявление метода для получения среднего значения
106     int _mean();
107     // Объявление метода для получения стандартного отклонения
    (возвращается M2)
108     int stdev();
109     // Объявление метода для накопления статистических данных (обновление
    моментов) с новым значением x
```

```
110     void collect(int x);
111 };
112 
113 /**
114  * Класс для аккумуляции фонем (слогов)
115  */
116 class syllable {
117 public:
118     // Объявление аккумуляторов для счёта появлений фонем: f, e, o, s, h,
    v
119     int f, e, o, s, h, v;
120     // Объявление переменных для хранения максимальных позиций (или
    показателей) для каждой фонемы
121     int maxf, maxe, maxo, maxs, maxh, maxv;
122     // Объявление переменных для модальности каждой фонемы (индикатор
    наличия двух пиков в распределении)
123     int modalityf, modalitye, modalityo, modalitys, modalityh, modalityv;
124     // Объявление переменной для хранения длины произнесённого слога
    (количество фонем)
125     int length;
126     // Объявление счётчика взрывных звуков (плозивов)
127     int plosiveCount;
128     // Объявление конструктора класса syllable
129     syllable();
130     // Объявление метода для сброса аккумуляторов (вызывается при
    обнаружении тишины)
131     void reset();
132     // Объявление метода для классификации символа (фонемы) с обновлением
    аккумуляторов в зависимости от входного символа
133     void classify(char c);
134     // Объявление метода для возврата аккумуляторов в виде указателя на
    массив int
135     int* tointptr();
136     // Объявление метода для отладочного вывода накопленных данных на
    Arduino
137     void debugPrint();
138     // Объявление метода для расчёта «расстояния» (схожести) между двумя
    слогами
139     void distance(syllable s);
140 private:
141     // Объявление временных аккумуляторов для промежуточного подсчёта
    фонем
142     char cf, ce, co, cs, ch, cv;
143     // Объявление переменных для хранения предыдущих значений временных
    аккумуляторов (для вычисления пиков и модальностей)
144     char prevf, preve, prevo, prevs, prevh, prevv;
145     // Объявление переменной для хранения текущего пика (фонемы) и флага
    ожидания пробела (для разделения слогов)
146     char currPeak, expectSp;
147     // Объявление переменной для хранения времени последнего обновления
    (определение интервала между слогами)
148     unsigned long lastTime;
149 };
150 
151 #include "uspeech.h"
152 
153 // Реализация метода распознавания фонемы getPhoneme класса signal
154 char signal::getPhoneme() {
155     // Выполнение выборки звуковых данных
156     sample();
157     // Вычисление суммарной мощности (энергии) сигнала
158     unsigned int pp = power();
159 
```

```
160     // Сравнение вычисленной мощности с пороговым значением тишины
161     if (pp > SILENCE) {
162         // Вычисление "сложности" сигнала
163         int k = complexity(pp);
164         // Сдвиг значений в истории коэффициентов (низкочастотная
    фильтрация)
165         overview[6] = overview[5];
166         overview[5] = overview[4];
167         overview[4] = overview[3];
168         overview[3] = overview[2];
169         overview[2] = overview[1];
170         overview[1] = overview[0];
171         // Запись нового значения сложности в начало массива истории
172         overview[0] = k;
173         // Инициализация переменной для расчёта среднего коэффициента
174         int coeff = 0;
175         // Итерация по элементам массива для суммирования значений
176         for (uint8_t f = 0; f < 6; f++) {
177             coeff += overview[f];
178         }
179         // Вычисление среднего коэффициента фильтрации
180         coeff /= 7;
181         // Расчёт мощности микрофона с экспоненциальным сглаживанием
182         micPower = 0.05 * maxPower() + (1 - 0.05) * micPower;
183         // Сохранение вычисленного коэффициента для отладки
184         testCoeff = coeff;
185         // Классификация фонемы на основе среднего коэффициента
186         if (coeff < econstant) {
187             // Выбор фонемы 'e'
188             phoneme = 'e';
189         } else if (coeff < aconstant) {
190             // Выбор фонемы 'o'
191             phoneme = 'o';
192         } else if (coeff < vconstant) {
193             // Выбор фонемы 'v'
194             phoneme = 'v';
195         } else if (coeff < shconstant) {
196             // Выбор фонемы 'h'
197             phoneme = 'h';
198         } else {
199             // Выбор фонемы 's'
200             phoneme = 's';
201         }
202         // Проверка разрешения распознавания фонемы /f/
203         if (f_enabled) {
204             // Сравнение мощности микрофона с порогом для фонемы /f/
205             if (micPower > fconstant) {
206                 // Возврат фонемы 'f'
207                 return 'f';
208             }
209         }
210         // Возврат распознанной фонемы
211         return phoneme;
212     }
213     else {
214         // Обнуление мощности микрофона при отсутствии звука
215         micPower = 0;
216         // Обнуление тестового коэффициента
217         testCoeff = 0;
218         // Возврат пробела (отсутствие звука)
219         return ' ';
220     }
221 }
```

```
222
223 #include "uspeech.h"
224
225 // Реализация конструктора класса signal
226 signal::signal(int port) {
227     // Присвоение номера порта для микрофона
228     pin = port;
229     // Инициализация порога для фонемы /f/ значением константы F_CONSTANT
230     fconstant = F_CONSTANT;
231     // Установка порога для фонем 'e'
232     econstant = 2;
233     // Установка порога для фонем 'o'
234     aconstant = 4;
235     // Установка порога для фонем 'v'
236     vconstant = 6;
237     // Установка порога для фонем 'h'
238     shconstant = 10;
239     // Установка коэффициента усиления для вычисления сложности сигнала
240     amplificationFactor = 10;
241     // Задание порога, ниже которого мощность микрофона считается слишком
    низкой
242     micPowerThreshold = 50;
243     // Установка масштаба для входного сигнала
244     scale = 1;
245 }
246
247 // Реализация метода калибровки микрофона на основе усреднения фонового
    уровня шума
248 void signal::calibrate(){
249     // Инициализация переменной калибровки нулем
250     calib = 0;
251     // Инициализация переменной для суммирования измерений
252     uint32_t samp = 0;
253     // Цикл для сбора 10 000 выборок фонового шума
254     for (uint16_t ind = 0; ind < 10000; ind++){
255         // Считывание значения с аналового входа, масштабирование и
    накопление
256         samp += analogRead(pin) * scale;
257     }
258     // Вычисление среднего значения фонового шума
259     calib = samp / 10000;
260 }
261
262 // Реализация метода выборки звукового сигнала с вычитанием калибровочного
    значения
263 void signal::sample(){
264     // Инициализация счётчика выборки
265     int i = 0;
266     // Цикл для сбора 32 значений с микрофона
267     while (i < 32){
268         // Считывание значения с аналога, масштабирование, вычитание
    калибровки и запись в массив
269         arr[i] = (analogRead(pin) * scale - calib);
270         // Инкремент счётчика
271         i++;
272     }
273 }
274
275 // Реализация метода для вычисления общей мощности сигнала
276 unsigned int signal::power(){
277     // Инициализация переменной для накопления суммы модулей значений
    сигнала
278     unsigned int j = 0;
```

```
279     // Инициализация счётчика
280     uint8_t i = 0;
281     // Цикл по всем элементам массива с выборками
282     while (i < 32){
283         // Прибавление абсолютного значения текущей выборки к сумме
284         j += abs(arr[i]);
285         // Инкремент счётчика
286         i++;
287     }
288     // Возврат суммарной мощности сигнала
289     return j;
290 }
291
292 // Реализация метода для вычисления «сложности» сигнала (отношение суммы
    модулей разностей соседних значений к мощности)
293 unsigned int signal::complexity(int power){
294     // Инициализация переменной для накопления суммы модулей разностей
    соседних выборок
295     unsigned int j = 0;
296     // Установка счётчика с началом со второго элемента массива
297     uint8_t i = 1;
298     // Цикл по элементам массива, начиная со второго значения
299     while (i < 32){
300         // Прибавление модуля разности текущей и предыдущей выборки
301         j += abs(arr[i] - arr[i - 1]);
302         // Инкремент счётчика
303         i++;
304     }
305     // Вычисление и возврат значения «сложности» сигнала с учётом
    коэффициента усиления
306     return (j * amplificationFactor) / power;
307 }
308
309 // Реализация метода для определения максимальной амплитуды сигнала
310 unsigned int signal::maxPower() {
311     // Инициализация счётчика для обхода массива
312     int i = 0;
313     // Инициализация переменной для хранения максимальной амплитуды
314     unsigned int max = 0;
315     // Цикл по всем 32 выборкам сигнала
316     while (i < 32){
317         // Сравнение текущего максимального значения с модулем текущей
    выборки
318         if (max < abs(arr[i])){
319             // Обновление максимального значения амплитуды
320             max = abs(arr[i]);
321             // Запись позиции, на которой зафиксирован максимум
322             maxPos = i;
323         }
324         // Инкремент счётчика
325         i++;
326         // Накопление значения для расчёта средней мощности
327         avgPower += arr[i];
328     }
329     // Вычисление средней мощности сигнала
330     avgPower /= 32;
331     // Возврат максимальной амплитуды
332     return max;
333 }
334
335 // Реализация метода для вычисления отношения сигнал/шум (SNR)
336 int signal::snr(int power){
337     // Инициализация счётчиков для обхода массива
```

```
338     uint8_t i = 0, j = 0;
339     // Вычисление среднего значения сигнала
340     int mean = power / 32;
341     // Цикл по всем выборкам сигнала
342     while (i < 32){
343         // Накопление квадрата разности между выборкой и средним значением
344         j += sq(arr[i] - mean);
345         // Инкремент счётчика
346         i++;
347     }
348     // Вычисление и возврат соотношения (корень из среднего квадрата
    ошибки к мощности)
349     return sqrt(j / mean) / power;
350 }
351
352 #include "uspeech.h"
353
354 // Реализация конструктора класса syllable
355 syllable::syllable(){
356     // Инициализация аккумуляторов для каждой фонемы
357     f = 0;
358     e = 0;
359     o = 0;
360     s = 0;
361     h = 0;
362     v = 0;
363     // Инициализация длины слога и временных аккумуляторов фонем
364     length = 0;
365     cf = 0;
366     ce = 0;
367     co = 0;
368     cs = 0;
369     ch = 0;
370     cv = 0;
371     // Инициализация модальности каждой фонемы
372     modalityf = 0;
373     modalitye = 0;
374     modalityo = 0;
375     modalitys = 0;
376     modalityh = 0;
377     modalityv = 0;
378     // Установка флага ожидания пробела в исходное состояние
379     expectSp = 1;
380     // Инициализация счётчика взрывных звуков
381     plosiveCount = 0;
382 }
383
384 // Реализация метода для сброса накопленных значений слога
385 void syllable::reset(){
386     // Сброс аккумуляторов для фонем
387     f = 0;
388     e = 0;
389     o = 0;
390     s = 0;
391     h = 0;
392     v = 0;
393     // Сброс длины слога и временных аккумуляторов
394     length = 0;
395     cf = 0;
396     ce = 0;
397     co = 0;
398     cs = 0;
399     ch = 0;
```

```
400     cv = 0;
401     // Сброс модальности для каждой фонемы
402     modalityf = 0;
403     modalitye = 0;
404     modalityo = 0;
405     modalitys = 0;
406     modalityh = 0;
407     modalityv = 0;
408     // Сброс флага ожидания пробела в исходное состояние
409     expectSp = 1;
410     // Сброс счётчика взрывных звуков
411     plosiveCount = 0;
412 }
413 
414 /***
415  * Метод для классификации входного символа (фонемы) с обновлением
    соответствующих счетчиков
416  */
417 void syllable::classify(char c){
418     // Увеличение длины слога (количество обработанных символов)
419     length++;
420     // Проверка флага ожидания пробела
421     if (expectSp == 0){
422         // Проверка, является ли текущий символ не пробелом
423         if (c != ' '){
424             // Сброс флага ожидания пробела (объединение символов в один
    слог)
425             expectSp = 1;
426             // Проверка, попадает ли интервал между символами в допустимый
    порог для взрывных звуков
427             if ((millis() - lastTime) < MAX_PLOSIVETIME){
428                 // Увеличение счётчика взрывных звуков
429                 plosiveCount++;
430             }
431         }
432     }
433     // Обработка символа в конструкции switch
434     switch (c) {
435         // Обработка символа 'f'
436         case 'f':
437             // Увеличение основного счётчика для 'f'
438             f++;
439             // Увеличение временного счётчика для 'f'
440             cf++;
441             // Выход из оператора switch для 'f'
442             break;
443         // Обработка символа 'e'
444         case 'e':
445             // Увеличение счётчика для 'e'
446             e++;
447             // Увеличение временного счётчика для 'e'
448             ce++;
449             break;
450         // Обработка символа 'o'
451         case 'o':
452             // Увеличение счётчика для 'o'
453             o++;
454             // Увеличение временного счётчика для 'o'
455             co++;
456             break;
457         // Обработка символа 'v'
458         case 'v':
459             // Увеличение счётчика для 'v'
```

```
460             v++;
461             // Увеличение временного счётчика для 'v'
462             cv++;
463             break;
464         // Обработка символа 'h'
465         case 'h':
466             // Увеличение счётчика для 'h'
467             h++;
468             // Увеличение временного счётчика для 'h'
469             ch++;
470             break;
471         // Обработка символа 's'
472         case 's':
473             // Увеличение счётчика для 's'
474             s++;
475             // Увеличение временного счётчика для 's'
476             cs++;
477             break;
478         // Обработка символа пробела
479         case ' ':
480             // Проверка активности флага ожидания пробела
481             if (expectSp != 0){
482                 // Сброс флага ожидания пробела для разделения слогов
483                 expectSp = 0;
484                 // Фиксация времени, используемого для расчёта интервала
    между слогами
485                 lastTime = millis();
486             }
487             break;
488         // Обработка любых остальных символов
489         default:
490             break;
491     }
492     // Периодический анализ для определения пиков и модальности
493     if ((length & PROCESS_SKEWNESS_TIME) == 0){
494         // Анализ временного счётчика для 'f'
495         if ((cf > prevf) & (prevf < PROCESS_SKEWNESS_TIME)){
496             // Обновление предыдущего значения для 'f'
497             prevf = cf;
498             // Фиксация длины слога при достижении пика 'f'
499             maxf = length;
500             // Установка текущего пика и увеличение модальности для 'f'
501             if (currPeak != 'f'){
502                 currPeak = 'f';
503                 modalityf++;
504             }
505         }
506         // Анализ временного счётчика для 'e'
507         if ((ce > preve) & (preve < PROCESS_SKEWNESS_TIME)){
508             preve = ce;
509             maxe = length;
510             if (currPeak != 'e'){
511                 currPeak = 'e';
512                 modalitye++;
513             }
514         }
515         // Анализ временного счётчика для 'o'
516         if ((co > prevo) & (prevo < PROCESS_SKEWNESS_TIME)){
517             prevo = co;
518             maxo = length;
519             if (currPeak != 'o'){
520                 currPeak = 'o';
521                 modalityo++;
```

```
522             }
523         }
524         // Анализ временного счётчика для 's'
525         if ((cs > prevs) & (prevs < PROCESS_SKEWNESS_TIME)){
526             prevs = cs;
527             maxs = length;
528             if (currPeak != 's'){
529                 currPeak = 's';
530                 modalitys++;
531             }
532         }
533         // Анализ временного счётчика для 'h'
534         if ((ch > prevh) & (prevh < PROCESS_SKEWNESS_TIME)){
535             prevh = ch;
536             maxh = length;
537             if (currPeak != 'h'){
538                 currPeak = 'h';
539                 modalityh++;
540             }
541         }
542         // Анализ временного счётчика для 'v' с порогом 15
543         if ((cv > prevv) & (prevv < 15)){
544             prevv = cv;
545             maxv = length;
546             if (currPeak != 'v'){
547                 currPeak = 'v';
548                 modalityv++;
549             }
550         }
551         // Сброс временных счётчиков для нового интервала анализа
552         cf = 0;
553         ce = 0;
554         co = 0;
555         cs = 0;
556         ch = 0;
557         cv = 0;
558     }
559 }
560
561 // Реализация метода для возврата указателя на массив с накопленными
    данными
562 int* syllable::tointptr(){
563     // Объявление статического массива для хранения накопленных значений
564     static int matrix[20];
565     // Запись значения аккумулятора 'f' в массив
566     matrix[0] = f;
567     // Запись значения аккумулятора 'e' в массив
568     matrix[1] = e;
569     // Запись значения аккумулятора 'o' в массив
570     matrix[2] = o;
571     // Запись значения аккумулятора 'v' в массив
572     matrix[3] = v;
573     // Запись значения аккумулятора 's' в массив
574     matrix[4] = s;
575     // Запись значения аккумулятора 'h' в массив
576     matrix[5] = h;
577     // Запись значения модальности для 'f' в массив
578     matrix[6] = modalityf;
579     // Запись значения модальности для 'e' в массив
580     matrix[7] = modalitye;
581     // Запись значения модальности для 'o' в массив
582     matrix[8] = modalityo;
583     // Запись значения модальности для 'v' в массив
```

```
584     matrix[9] = modalityv;
585     // Запись значения модальности для 's' в массив
586     matrix[10] = modalitys;
587     // Запись значения модальности для 'h' в массив
588     matrix[11] = modalityh;
589     // Запись значения пика для 'f' в массив
590     matrix[12] = maxf;
591     // Запись значения пика для 'e' в массив
592     matrix[13] = maxe;
593     // Запись значения пика для 'o' в массив
594     matrix[14] = maxo;
595     // Запись значения пика для 'v' в массив
596     matrix[15] = maxv;
597     // Запись значения пика для 's' в массив
598     matrix[16] = maxs;
599     // Запись значения пика для 'h' в массив
600     matrix[17] = maxh;
601     // Запись значения длины слога в массив
602     matrix[18] = length;
603     // Запись значения счётчика взрывных звуков в массив
604     matrix[19] = plosiveCount;
605     // Возврат указателя на заполненный массив
606     return matrix;
607 }
608
609 #include "uspeech.h"
610
611 // Реализация конструктора класса statCollector для накопления
    статистических данных
612 statCollector::statCollector(){
613     // Инициализация переменных для количества выборок, среднего значения
    и моментов нулевыми значениями
614     n = 0;
615     mean = 0;
616     M2 = 0;
617     M3 = 0;
618     M4 = 0;
619 }
620
621 // Реализация метода для возврата среднего значения
622 int statCollector::_mean(){
623     // Возврат рассчитанного среднего значения
624     return mean;
625 }
626
627 // Реализация метода для получения стандартного отклонения (в данном
    случае дисперсии - M2)
628 int statCollector::stdev(){
629     // Возврат значения второго центрального момента
630     return M2;
631 }
632
633 // Реализация метода для вычисления куртоза (измерение «остроты»
    распределения)
634 int statCollector::kurtosis(){
635     // Вычисление куртоза по формуле (нормированный 4-й момент минус 3)
636     int kurtosis = (n * M4) / (M2 * M2) - 3;
637     // Возврат вычисленного значения куртоза
638     return kurtosis;
639 }
640
641 // Реализация метода для вычисления асимметрии распределения
642 int statCollector::skew(){
```

```
643     // Вычисление асимметрии по формуле (нормированный 3-й момент минус 3)
644     int kurtosis = (n * M3) / (M2 * M2 * M2) - 3;
645     // Возврат полученного значения асимметрии
646     return kurtosis;
647 }
648
649 // Реализация метода для накопления статистических данных с новым
    значением x
650 void statCollector::collect(int x) {
651     // Сохранение предыдущего количества выборок
652     int n1 = n;
653     // Увеличение счётчика выборок
654     n = n + 1;
655     // Вычисление разницы между новым значением и текущим средним
656     int delta = x - mean;
657     // Вычисление поправочного коэффициента для среднего
658     int delta_n = delta / n;
659     // Вычисление квадрата поправочного коэффициента
660     int delta_n2 = delta_n * delta_n;
661     // Вычисление вспомогательного терма для обновления статистических
    моментов
662     int term1 = delta * delta_n * n1;
663     // Обновление среднего значения
664     mean = mean + delta_n;
665     // Обновление четвёртого момента распределения
666     M4 = M4 + term1 * delta_n2 * (n * n - 3 * n + 3) + 6 * delta_n2 * M2 -
    4 * delta_n * M3;
667     // Обновление третьего момента (асимметрия)
668     M3 = M3 + term1 * delta_n * (n - 2) - 3 * delta_n * M2;
669     // Обновление второго момента (сумма квадратов отклонений)
670     M2 = M2 + term1;
671 }
672
673 #include <Wire.h>
674 // Подключение библиотеки для работы с интерфейсом I²C
675 #include <LiquidCrystal_I2C.h>
676 // Подключение заголовочного файла uspeech.h (библиотека для распознавания
    речи)
677 #include <uspeech.h>
678
679 // Определение номера пина для зелёного светодиода
680 #define ledGreen 7
681 // Определение номера пина для оранжевого светодиода
682 #define ledOrange 6
683 // Определение номера пина для белого светодиода
684 #define ledWhite 5
685
686 // Определение макроса MIN3 для вычисления минимального значения из трёх
    аргументов
687 #define MIN3(a, b, c) ((a) < (b) ? ((a) < (c) ? (a) : (c)) : ((b) < (c) ?
    (b) : (c)))
688
689 // Создание объекта voice класса signal с подключением микрофона к
    аналоговому пину A0
690 signal voice(A0);
691
692 // Создание объекта lcd класса LiquidCrystal_I2C для работы с LCD-дисплеем
    по I²C
693 // Указание адреса дисплея (0x27) и размера дисплея (16 столбцов, 2
    строки)
694 LiquidCrystal_I2C lcd(0x27, 16, 2);
695
```

```
696 // Определение константы для размера буфера, используемого для
    формирования слова из фонем
697 const int BUFFER_MAX_PHONEMES = 32;
698 // Объявление массива для накопления распознанных символов
699 char inputString[BUFFER_MAX_PHONEMES];
700 // Инициализация переменной индекса текущей позиции в буфере
701 byte index = 0;
702
703 // Определение количества элементов в словаре (паттернов) фонем
704 const int DICT_MAX_ELEMNTS = 3;
705 // Объявление словаря паттернов фонем для распознавания команд
706 char dict[DICT_MAX_ELEMNTS][BUFFER_MAX_PHONEMES] = {
707   "vvvoeeeeeeeeofff",   // Паттерн для команды green
708   "hhhhhvoovvvvf",      // Паттерн для команды orange
709   "booooooffffffff"     // Паттерн для команды white
710 };
711
712 // Определение порогового значения для расстояния Левенштейна (для
    распознавания слова)
713 int LOWEST_COST_MAX_THREASHOLD = 20;
714
715 // Объявление глобальных переменных для контроля бездействия
716 // Переменная lastCommandTime хранит время последнего изменения команды
717 unsigned long lastCommandTime = 0;
718 // Флаг idleDisplayed указывает, что состояние бездействия уже выведено на
    дисплей
719 bool idleDisplayed = false;
720
721 // Функция для вычисления длины строки до пробела или конца строки
722 int strLength(char const* s) {
723   // Инициализация счётчика
724   int i = 0;
725   // Цикл до достижения символа окончания строки или пробела
726   while (s[i] != '\0' && s[i] != ' ')
727     ++i;
728   // Возврат вычисленной длины строки
729   return i;
730 }
731
732 // Функция для расчёта расстояния Левенштейна между двумя строками
733 unsigned int levenshtein(char *s1, char *s2) {
734   // Объявление переменных для хранения длин строк
735   unsigned int s1len, s2len, x, y, lastdiag, olddiag;
736   // Вычисление длины первой строки
737   s1len = strlen(s1);
738   // Вычисление длины второй строки
739   s2len = strlen(s2);
740   // Объявление массива для хранения промежуточных результатов (колонка
    матрицы)
741   unsigned int column[s1len + 1];
742
743   // Инициализация первого столбца матрицы (расстояния)
744   for (y = 1; y <= s1len; y++)
745     column[y] = y;
746
747   // Основной цикл для расчёта расстояния Левенштейна
748   for (x = 1; x <= s2len; x++) {
749     // Обновление первого элемента столбца
750     column[0] = x;
751     // Внутренний цикл для перебора символов первой строки
752     for (y = 1, lastdiag = x - 1; y <= s1len; y++) {
753       // Сохранение текущего значения для использования в следующей
    итерации
```

```
754       olddiag = column[y];
755       // Вычисление минимального из вариантов: удаление, вставка или
    замена символа
756       column[y] = MIN3(
757         column[y] + 1,
758         column[y - 1] + 1,
759         lastdiag + (s1[y - 1] == s2[x - 1] ? 0 : 1)
760       );
761       // Обновление lastdiag для следующего шага
762       lastdiag = olddiag;
763     }
764   }
765
766   // Возврат конечного расстояния между строками
767   return column[s1len];
768 }
769
770 // Функция для поиска наиболее похожего паттерна в словаре, используя
    алгоритм Левенштейна
771 char* guessWord(char* target) {
772   // Проверка на прямое совпадение указателей (для статичных строк)
773   for (int i = 0; i < DICT_MAX_ELEMNTS; i++) {
774     if (dict[i] == target) {
775       return dict[i];
776     }
777   }
778
779   // Объявление массива для хранения стоимости расстояния для каждого
    паттерна
780   unsigned int cost[DICT_MAX_ELEMNTS];
781   // Вычисление расстояния Левенштейна для каждого паттерна словаря
782   for (int j = 0; j < DICT_MAX_ELEMNTS; j++) {
783     cost[j] = levenshtein(dict[j], target);
784     // Вывод стоимости для отладки через Serial
785     Serial.println("dict[j]=" + String(dict[j]) + " target=" +
    String(target) +
786                    " cost=" + String(cost[j]));
787   }
788
789   // Инициализация переменных для поиска паттерна с наименьшей стоимостью
790   int lowestCostIndex = -1;
791   int lowestCost = LOWEST_COST_MAX_THREASHOLD;
792
793   // Поиск паттерна с минимальным расстоянием, удовлетворяющим порогу
794   for (int j = 0; j < DICT_MAX_ELEMNTS; j++) {
795     if (cost[j] < lowestCost) {
796       lowestCost = cost[j];
797       lowestCostIndex = j;
798     }
799   }
800
801   // Если найден подходящий паттерн, возвращается он; иначе – пустая
    строка
802   if (lowestCostIndex > -1) {
803     return dict[lowestCostIndex];
804   } else {
805     return "";
806   }
807 }
808
809 // Функция для обработки распознанной команды (слова) с выводом на дисплей
    и управлением светодиодами
810 void parseCommand(char* str) {
```

```
811   // Получение наиболее похожего паттерна для введённой строки
812   char *gWord = guessWord(str);
813   // Вывод результата распознавания в Serial
814   Serial.println("guessed: " + String(gWord));
815 
816   // Очистка дисплея LCD
817   lcd.clear();
818   // Установка курсора в начало первой строки
819   lcd.setCursor(0, 0);
820   // Вывод метки "Word:" на LCD
821   lcd.print("Word:");
822   // Перемещение курсора на вторую строку дисплея
823   lcd.setCursor(0, 1);
824 
825   // Если паттерн не найден, дальнейшая обработка прекращается
826   if (gWord == "") {
827     return;
828   }
829   // Обработка команды "green"
830   else if (gWord == dict[0]) {
831     // Вывод текста "green" на дисплей
832     lcd.print("green");
833     // Включение зелёного светодиода
834     digitalWrite(ledGreen, HIGH);
835     // Выключение оранжевого светодиода
836     digitalWrite(ledOrange, LOW);
837     // Выключение белого светодиода
838     digitalWrite(ledWhite, LOW);
839   }
840   // Обработка команды "orange"
841   else if (gWord == dict[1]) {
842     // Вывод текста "orange" на дисплей
843     lcd.print("orange");
844     // Выключение зелёного светодиода
845     digitalWrite(ledGreen, LOW);
846     // Включение оранжевого светодиода
847     digitalWrite(ledOrange, HIGH);
848     // Выключение белого светодиода
849     digitalWrite(ledWhite, LOW);
850   }
851   // Обработка команды "white"
852   else if (gWord == dict[2]) {
853     // Вывод текста "white" на дисплей
854     lcd.print("white");
855     // Выключение зелёного светодиода
856     digitalWrite(ledGreen, LOW);
857     // Выключение оранжевого светодиода
858     digitalWrite(ledOrange, LOW);
859     // Включение белого светодиода
860     digitalWrite(ledWhite, HIGH);
861   }
862   // Обновление времени последнего распознавания команды и флага
    бездействия
863   lastCommandTime = millis();
864   idleDisplayed = false;
865 }
866 
867 void setup() {
868   // Настройка параметров uSpeech (распознавание, пороговые значения) и
    калибровка микрофона
869   voice.f_enabled = true;
870   voice.minVolume = 2000;
871   voice.fconstant = 500;
```

```
872   voice.econstant = 2;
873   voice.aconstant = 4;
874   voice.vconstant = 6;
875   voice.shconstant = 10;
876   // Запуск калибровки фонового уровня микрофона
877   voice.calibrate();
878 
879   // Инициализация последовательного порта для отладки
880   Serial.begin(9600);
881 
882   // Настройка пинов светодиодов в режиме OUTPUT
883   pinMode(ledGreen, OUTPUT);
884   pinMode(ledOrange, OUTPUT);
885   pinMode(ledWhite, OUTPUT);
886 
887   // Инициализация дисплея LCD
888   lcd.init();
889   // Включение подсветки дисплея
890   lcd.backlight();
891   // Очистка дисплея
892   lcd.clear();
893 
894   // Вывод приветственного сообщения на дисплей
895   lcd.setCursor(0, 0);
896   lcd.print("Voice Recog");
897   lcd.setCursor(0, 1);
898   lcd.print("Say a color...");
899 
900   // Последовательное мигание светодиодов в течение 3 секунд при запуске
901   unsigned long startTime = millis();
902   while (millis() - startTime < 3000) {
903     // Включение зелёного светодиода
904     digitalWrite(ledGreen, HIGH);
905     delay(200);
906     // Выключение зелёного светодиода
907     digitalWrite(ledGreen, LOW);
908     delay(100);
909 
910     // Включение оранжевого светодиода
911     digitalWrite(ledOrange, HIGH);
912     delay(200);
913     // Выключение оранжевого светодиода
914     digitalWrite(ledOrange, LOW);
915     delay(100);
916 
917     // Включение белого светодиода
918     digitalWrite(ledWhite, HIGH);
919     delay(200);
920     // Выключение белого светодиода
921     digitalWrite(ledWhite, LOW);
922     delay(100);
923   }
924 
925   // Очистка дисплея после мигания и вывод приглашения к произнесению
    команды
926   lcd.clear();
927   lcd.setCursor(0, 0);
928   lcd.print("Say a color...");
929 
930   // Инициализация переменных для контроля бездействия
931   lastCommandTime = millis();
932   idleDisplayed = false;
933 }
```

```
934
935 void loop() {
936   // Получение новой выборки звука с микрофона
937   voice.sample();
938   // Распознавание фонемы из полученной выборки
939   char p = voice.getPhoneme();
940
941   // Если получен символ пробела или буфер заполнен, обработка
    накопленного слова
942   if (p == ' ' || index >= BUFFER_MAX_PHONEMES) {
943     // Если накопленная строка не пустая
944     if (strLength(inputString) > 0) {
945       // Вывод распознанного слова в Serial
946       Serial.println("received: " + String(inputString));
947       // Обработка команды, соответствующей накопленному слову
948       parseCommand(inputString);
949       // Очистка буфера для следующего слова
950       inputString[0] = 0;
951       // Сброс индекса буфера
952       index = 0;
953     }
954   } else {
955     // Добавление нового символа в буфер
956     inputString[index] = p;
957     // Увеличение индекса
958     index++;
959     // Обеспечение корректного завершения строки нулевым символом
960     inputString[index] = '\0';
961   }
962
963   // Если с момента последнего распознавания прошло более 5 секунд, вывод
    сообщения idle
964   if (millis() - lastCommandTime > 5000 && !idleDisplayed) {
965     // Выключение всех светодиодов
966     digitalWrite(ledGreen, LOW);
967     digitalWrite(ledOrange, LOW);
968     digitalWrite(ledWhite, LOW);
969
970     // Очистка дисплея LCD
971     lcd.clear();
972     lcd.setCursor(0, 0);
973     lcd.print("Say a color...");
974
975     // Установка флага, что режим бездействия уже выведен
976     idleDisplayed = true;
977   }
978 }
979
```