```
// variables
var leftchannel = [];
var rightchannel = [];
var recorder = null;
var recording = false;
var recordingLength = 0;
var volume = null;
var audioInput = null;
var sampleRate = null;
var audioContext = null;
var context = null;
var outputElement = document.getElementById('output');
var outputString;

// feature detection
if (!navigator.getUserMedia)
    navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia ||
                  navigator.mozGetUserMedia || navigator.msGetUserMedia;

if (navigator.getUserMedia){
    navigator.getUserMedia({audio:true}, success, function(e) {
    alert('Error capturing audio.');
    });
} else alert('getUserMedia not supported in this browser.');

// when key is down
window.onkeydown = function(e){

    // if R is pressed, we start recording
    if ( e.keyCode == 82 ){
        recording = true;
        // reset the buffers for the new recording
        leftchannel.length = rightchannel.length = 0;
        recordingLength = 0;
        outputElement.innerHTML = 'Recording now...';
    // if S is pressed, we stop the recording and package the WAV file
    } else if ( e.keyCode == 83 ){

        // we stop recording
        recording = false;

        outputElement.innerHTML = 'Building wav file...';

        // we flat the left and right channels down
        var leftBuffer = mergeBuffers ( leftchannel, recordingLength );
        var rightBuffer = mergeBuffers ( rightchannel, recordingLength );
        // we interleave both channels together
        var interleaved = interleave ( leftBuffer, rightBuffer );

        // we create our wav file
        var buffer = new ArrayBuffer(44 + interleaved.length * 2);
        var view = new DataView(buffer);

        // RIFF chunk descriptor
        writeUTFBytes(view, 0, 'RIFF');
        view.setUint32(4, 44 + interleaved.length * 2, true);
        writeUTFBytes(view, 8, 'WAVE');
        // FMT sub-chunk
        writeUTFBytes(view, 12, 'fmt ');
        view.setUint32(16, 16, true);
        view.setUint16(20, 1, true);
        // stereo (2 channels)
        view.setUint16(22, 2, true);
        view.setUint32(24, sampleRate, true);
        view.setUint32(28, sampleRate * 4, true);
        view.setUint16(32, 4, true);
        view.setUint16(34, 16, true);
        // data sub-chunk
        writeUTFBytes(view, 36, 'data');
        view.setUint32(40, interleaved.length * 2, true);

        // write the PCM samples
        var lng = interleaved.length;
        var index = 44;
        var volume = 1;
        for (var i = 0; i < lng; i++){
            view.setInt16(index, interleaved[i] * (0x7FFF * volume), true);
            index += 2;
        }

        // our final binary blob
        var blob = new Blob ( [ view ], { type : 'audio/wav' } );

        // let's save it locally
        outputElement.innerHTML = 'Handing off the file now...';
        var url = (window.URL || window.webkitURL).createObjectURL(blob);
        var link = window.document.createElement('a');
        link.href = url;
        link.download = 'output.wav';
        var click = document.createEvent("Event");
        click.initEvent("click", true, true);
        link.dispatchEvent(click);
    }
}

function interleave(leftChannel, rightChannel){
  var length = leftChannel.length + rightChannel.length;
  var result = new Float32Array(length);

  var inputIndex = 0;

  for (var index = 0; index < length; ){
    result[index++] = leftChannel[inputIndex];
    result[index++] = rightChannel[inputIndex];
    inputIndex++;
  }
  return result;
}

function mergeBuffers(channelBuffer, recordingLength){
  var result = new Float32Array(recordingLength);
  var offset = 0;
  var lng = channelBuffer.length;
  for (var i = 0; i < lng; i++){
    var buffer = channelBuffer[i];
    result.set(buffer, offset);
    offset += buffer.length;
  }
  return result;
}
```

```
function writeUTFBytes(view, offset, string){
  var lng = string.length;
  for (var i = 0; i < lng; i++){
    view.setUint8(offset + i, string.charCodeAt(i));
  }
}

function success(e){
    // creates the audio context
    audioContext = window.AudioContext || window.webkitAudioContext;
    context = new audioContext();

        // we query the context sample rate (varies depending on platforms)
    sampleRate = context.sampleRate;

    console.log('succcess');

    // creates a gain node
    volume = context.createGain();

    // creates an audio node from the microphone incoming stream
    audioInput = context.createMediaStreamSource(e);

    // connect the stream to the gain node
    audioInput.connect(volume);

    /* From the spec: This value controls how frequently the audioprocess event is
    dispatched and how many sample-frames need to be processed each call.
    Lower values for buffer size will result in a lower (better) latency.
    Higher values will be necessary to avoid audio breakup and glitches */
    var bufferSize = 2048;
    recorder = context.createScriptProcessor(bufferSize, 2, 2);

    recorder.onaudioprocess = function(e){
        if (!recording) return;
        var left = e.inputBuffer.getChannelData (0);
        var right = e.inputBuffer.getChannelData (1);
        // we clone the samples
        leftchannel.push (new Float32Array (left));
        rightchannel.push (new Float32Array (right));
        recordingLength += bufferSize;
        console.log('recording');
    }

    // we connect the recorder
    volume.connect (recorder);
    recorder.connect (context.destination);
}
```