

MUSC 208 Winter 2014  
John Ellinger, Carleton College

## Lab 4 Mapping the HID Keyboard, MIDI Output, Recording

### Do This Lab On An iMac in the Lab

Lab 4 is an extended version of Lab 3. In Lab 3 we defined four functions to say a word when you pressed m, 2, o, and 8. In Lab 4 we'll map most of the keys on the keyboard. We'll write a function called `playIt( keyPress )` that is called every time a key is pressed. What gets played depends on what key is pressed.

### Setup

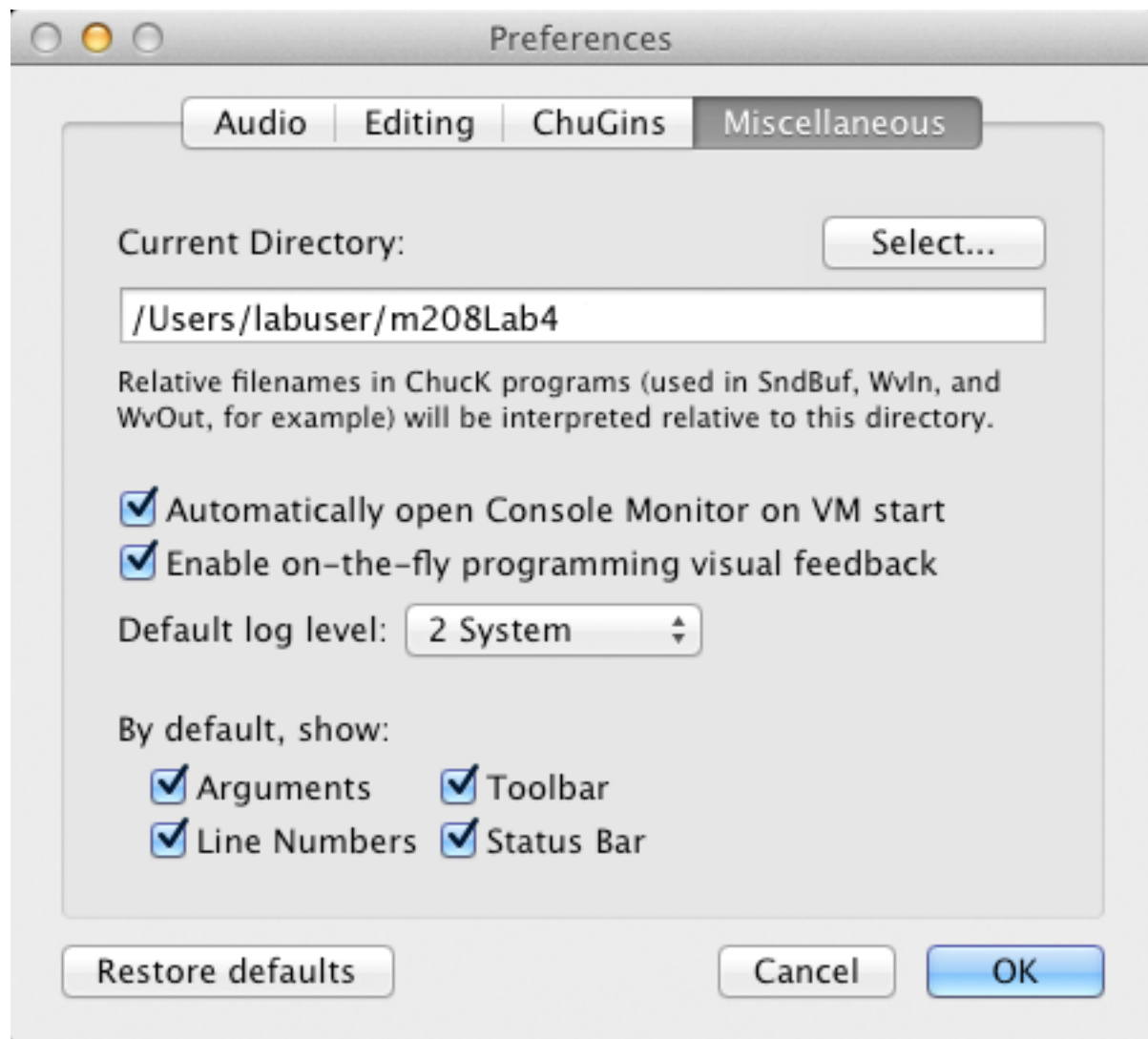
Download and unzip this file to your Desktop: m208Lab4.zip.

Directions in this lab refer to the folder named m208Lab4 and assume it's on the Desktop. The m208Lab4 folder should contain these files:



### Set the miniAudicle Working Directory

Open /Application/miniAudicle and choose Preferences from the miniAudicle menu. Select the Miscellaneous tab, click the Select button and choose the m208Lab4 folder on the Desktop. Lab iMac pathname is shown.



## Map the ASCII Keyboard Layout

Lab 3 responded to key presses using this code.

```
// check for action type
if( msg.isButtonDown() )
{
    if (msg.ascii == ascii_M)
        playMusic();
    else if (msg.ascii == ascii_2)
        playTwo();
    else if (msg.ascii == ascii_0)
        play0h();
    else if (msg.ascii == ascii_8)
        playEight();
}
```

Lab 4 is going to map most of the keys the keyboard into four horizontal rows using command line (Terminal) ChuckK and TextWrangler to help write the code. Do this portion of Lab 4 on an iMac using those two programs. You can easily copy the code to your laptop when finished. The finished code can be reused in future ChuckK programs.



## Modify kb.ck

The kb.ck file can be found in the m208Lab4 folder. kb.ck is one of many code examples that are distributed with ChuckK. Open it in TextWrangler.

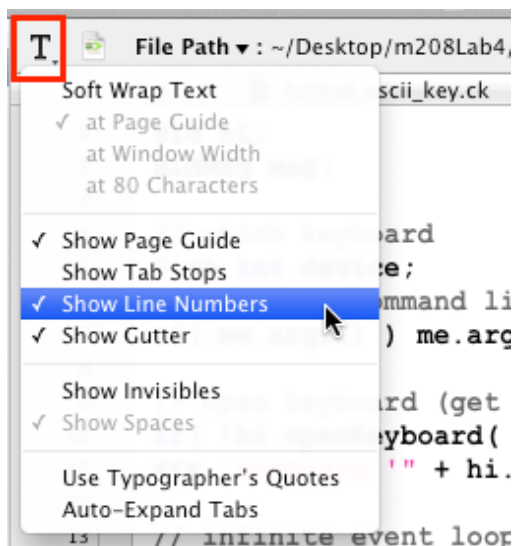
```

1  Hid hi;
2  HidMsg msg;
3
4  // which keyboard
5  1 => int device;
6  // get from command line
7  if( me.args() ) me.arg(0) => Std.atoi => device;
8
9  // open keyboard (get device number from command line)
10 if( !hi.openKeyboard( device ) ) me.exit();
11 <<< "keyboard '" + hi.name() + "' ready", "" >>>;
12 <<< "Type Ctrl-C to exit.", "" >>>;
13
14 // infinite event loop
15 while( true )
16 {
17     // wait on event
18     hi => now;
19
20     // get one or more messages
21     while( hi.recv( msg ) )
22     {
23         // check for action type
24         if ( msg.isButtonDown() )
25         {
26             <<< "\tdown:", msg.which, "(code)", msg.key, "(usb key)", msg.ascii, "(ascii)" >>>;
27         }
28
29         else if ( msg.isButtonUp() )
30         {
31             <<< "\tup:  ", msg.which, "(code)", msg.key, "(usb key)", msg.ascii, "(ascii)" >>>;
32         }
33     }
34 }

```

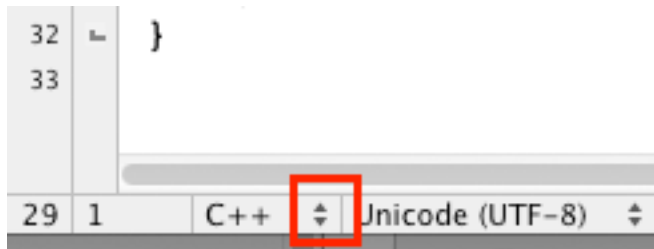
## Display TextWrangler Line Numbers

Choose Show Line Numbers from the "T" popup menu. Another useful item in this menu is Show Invisibles.



## Display Textwrangler Color Syntax Coding

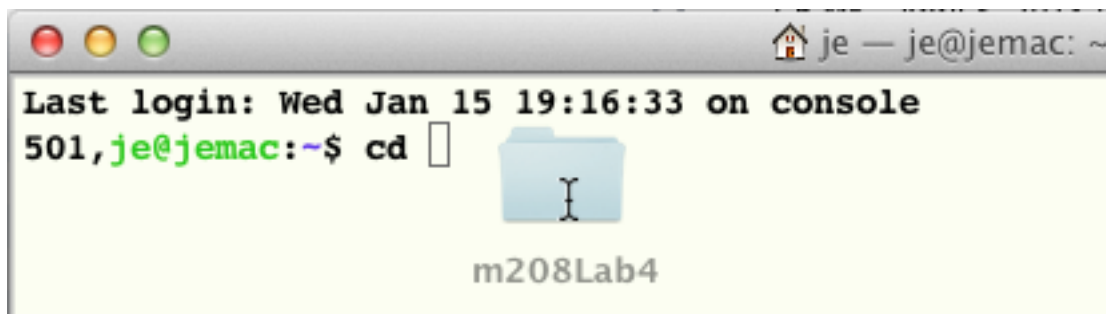
Choose C++ syntax from the formatting popup menu.



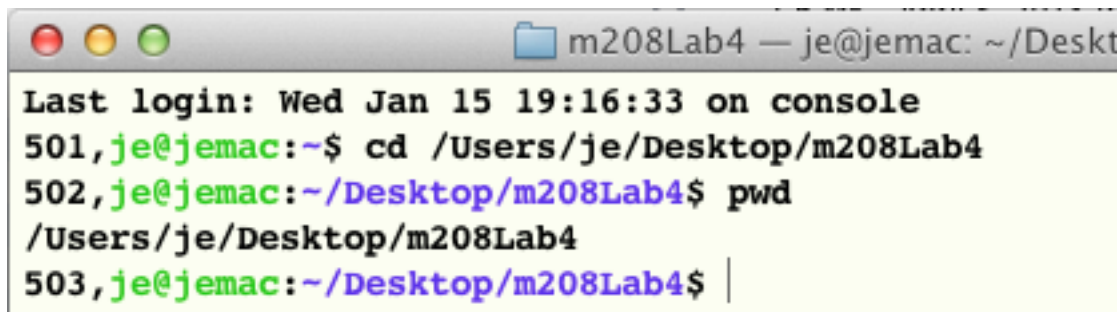
Save the file.

## Open Terminal

At the Terminal prompt type `cd` followed by space, and then drag the m208Lab4 icon into the terminal window.



Release the mouse button and the pathname of the m208Lab4 will appear after `cd`. Type return to change directory to the m208Folder and then execute `pwd` to verify the change.



## Run kb.ck from the Terminal

You can run any ChuckK file in the Terminal by typing `chuck` followed by the file name. The file must exist in the working directory (`pwd`).

```
503, je@jemac:~/Desktop/m208Lab4$ chuck kb.ck
```

Type Enter to execute. You should see

```
506, je@jemac:~/Desktop/m208Lab4$ chuck kb.ck
keyboard 'Keyboard' ready
Type Ctrl-C to exit.
```

## msg.which, msg.key, and msg.ascii

The msg.isButtonDown() and msg.isButtonUp() methods return three data items for each key press: the raw key code, the USB key code, and the ASCII value. Which of the three msg values you use in your program is up to you. The ASCII codes are most consistent. However, the key code values can vary between Mac/Windows and desktop/laptop keyboards.

### Experiment 1 - type a b c d and A B C D

Both upper and lower case letters return the same ASCII code. The capital letters are correct. The lower case ASCII codes are off by 32, "a" = 97.

<b>a</b>	<b>up:</b>	<b>4 (code)</b>	<b>4 (usb key)</b>	<b>65 (ascii)</b>
<b>b</b>	<b>up:</b>	<b>5 (code)</b>	<b>5 (usb key)</b>	<b>66 (ascii)</b>
<b>c</b>	<b>up:</b>	<b>6 (code)</b>	<b>6 (usb key)</b>	<b>67 (ascii)</b>
<b>d</b>	<b>up:</b>	<b>7 (code)</b>	<b>7 (usb key)</b>	<b>68 (ascii)</b>
<b>A</b>	<b>up:</b>	<b>4 (code)</b>	<b>4 (usb key)</b>	<b>65 (ascii)</b>
<b>B</b>	<b>up:</b>	<b>5 (code)</b>	<b>5 (usb key)</b>	<b>66 (ascii)</b>
<b>C</b>	<b>up:</b>	<b>6 (code)</b>	<b>6 (usb key)</b>	<b>67 (ascii)</b>
<b>D</b>	<b>up:</b>	<b>7 (code)</b>	<b>7 (usb key)</b>	<b>68 (ascii)</b>

### Experiment 2 - type Shift key

If your code depended on distinguishing between upper and lower case letters, you'd have to define a separate variable that kept track of the state of the Shift key. The up and down values are the same.

```
up: 225 (code) 225 (usb key) 0 (ascii)
```

Other keyboard modifier keys like capsLock, control, alt, command also return ASCII value zero. You'll need to be check for those keys with msg.which data.

### Experiment 3 - type space bar

```
up: 44 (code) 44 (usb key) 32 (ascii)
```

Experiment 4 - type backspace

```
up: 42 (code) 42 (usb key) 8 (ascii)
```

Experiment 5 - type Enter and 2<sup>nd</sup> Enter key if present

Both enter keys return ASCII 10, but return different key codes.

```
up: 40 (code) 40 (usb key) 10 (ascii)
```

```
up: 88 (code) 88 (usb key) 10 (ascii)
```

Experiment 6 - type some Function keys ( F1-F6 here)

The function keys on my Microsoft keyboard all return ASCII zero but differ by key code.

```
^[OP up: 58 (code) 58 (usb key) 0 (ascii)
```

```
^[OQ up: 59 (code) 59 (usb key) 0 (ascii)
```

```
^[OR up: 60 (code) 60 (usb key) 0 (ascii)
```

```
^[OS up: 61 (code) 61 (usb key) 0 (ascii)
```

```
^[15~ up: 62 (code) 62 (usb key) 0 (ascii)
```

```
^[17~ up: 63 (code) 63 (usb key) 0 (ascii)
```

Experiment 7 - type 1 2 3 4 on the upper row of the keyboard and on the keypad.

The ASCII values are the same, but the key codes differ.

```
1 up: 30 (code) 30 (usb key) 49 (ascii)
```

```
2 up: 31 (code) 31 (usb key) 50 (ascii)
```

```
3 up: 32 (code) 32 (usb key) 51 (ascii)
```

```
4 up: 33 (code) 33 (usb key) 52 (ascii)
```

---

```
1 up: 89 (code) 89 (usb key) 49 (ascii)
```

```
2 up: 90 (code) 90 (usb key) 50 (ascii)
```

```
3 up: 91 (code) 91 (usb key) 51 (ascii)
```

```
4 up: 92 (code) 92 (usb key) 52 (ascii)
```

## Map the four main keyboard rows

Start with the top row 1 2 3 4 5 6 7 8 9 0 - = and type every character from left to right. Do the same with the qwerty row, the asdfgh row, and the zxcvbn row.



```

506,je@jemac:~/Desktop/m208Lab4$ chuck kb.ck
keyboard 'Keyboard' ready
Type Ctrl-C to exit.
1      up:  30 (code) 30 (usb key) 49 (ascii)
2      up:  31 (code) 31 (usb key) 50 (ascii)
3      up:  32 (code) 32 (usb key) 51 (ascii)
4      up:  33 (code) 33 (usb key) 52 (ascii)
5      up:  34 (code) 34 (usb key) 53 (ascii)
6      up:  35 (code) 35 (usb key) 54 (ascii)
7      up:  36 (code) 36 (usb key) 55 (ascii)
8      up:  37 (code) 37 (usb key) 56 (ascii)
9      up:  38 (code) 38 (usb key) 57 (ascii)
0      up:  39 (code) 39 (usb key) 48 (ascii)
-      up:  45 (code) 45 (usb key) 45 (ascii)
=      up:  46 (code) 46 (usb key) 61 (ascii)
q      up:  20 (code) 20 (usb key) 81 (ascii)
w      up:  26 (code) 26 (usb key) 87 (ascii)
e      up:   8 (code)  8 (usb key) 69 (ascii)
r      up:  21 (code) 21 (usb key) 82 (ascii)
t      up:  23 (code) 23 (usb key) 84 (ascii)
y      up:  28 (code) 28 (usb key) 89 (ascii)
u      up:  24 (code) 24 (usb key) 85 (ascii)
i      up:  12 (code) 12 (usb key) 73 (ascii)
o      up:  18 (code) 18 (usb key) 79 (ascii)
p      up:  19 (code) 19 (usb key) 80 (ascii)
[      up:  47 (code) 47 (usb key) 91 (ascii)
]      up:  48 (code) 48 (usb key) 93 (ascii)
\      up:  49 (code) 49 (usb key) 92 (ascii)
a      up:   4 (code)  4 (usb key) 65 (ascii)
s      up:  22 (code) 22 (usb key) 83 (ascii)
d      up:   7 (code)  7 (usb key) 68 (ascii)
f      up:   9 (code)  9 (usb key) 70 (ascii)
g      up:  10 (code) 10 (usb key) 71 (ascii)
h      up:  11 (code) 11 (usb key) 72 (ascii)
j      up:  13 (code) 13 (usb key) 74 (ascii)
k      up:  14 (code) 14 (usb key) 75 (ascii)
l      up:  15 (code) 15 (usb key) 76 (ascii)
;      up:  51 (code) 51 (usb key) 59 (ascii)
'      up:  52 (code) 52 (usb key) 39 (ascii)
z      up:  29 (code) 29 (usb key) 90 (ascii)
x      up:  27 (code) 27 (usb key) 88 (ascii)
c      up:   6 (code)  6 (usb key) 67 (ascii)
v      up:  25 (code) 25 (usb key) 86 (ascii)
b      up:   5 (code)  5 (usb key) 66 (ascii)
n      up:  17 (code) 17 (usb key) 78 (ascii)
m      up:  16 (code) 16 (usb key) 77 (ascii)
,      up:  54 (code) 54 (usb key) 44 (ascii)
.      up:  55 (code) 55 (usb key) 46 (ascii)
/      up:  56 (code) 56 (usb key) 47 (ascii)
^C[chuck]: cleaning up...

```

## Quit kb.ck

Type Control C to quit.



# Using Grep to Write Code

## Grep - General Regular Expression Parser

Copy and paste the kb.ck data from the Terminal into TextWrangler.

1	1	up:	30	(code)	30	(usb key)	49	(ascii)
2	2	up:	31	(code)	31	(usb key)	50	(ascii)
3	3	up:	32	(code)	32	(usb key)	51	(ascii)
4	4	up:	33	(code)	33	(usb key)	52	(ascii)
5	5	up:	34	(code)	34	(usb key)	53	(ascii)
6	6	up:	35	(code)	35	(usb key)	54	(ascii)
7	7	up:	36	(code)	36	(usb key)	55	(ascii)
8	8	up:	37	(code)	37	(usb key)	56	(ascii)
9	9	up:	38	(code)	38	(usb key)	57	(ascii)
10	0	up:	39	(code)	39	(usb key)	48	(ascii)
11	-	up:	45	(code)	45	(usb key)	45	(ascii)
12	=	up:	46	(code)	46	(usb key)	61	(ascii)
13	q	up:	20	(code)	20	(usb key)	81	(ascii)
14	w	up:	26	(code)	26	(usb key)	87	(ascii)
15	e	up:	8	(code)	8	(usb key)	69	(ascii)
16	r	up:	21	(code)	21	(usb key)	82	(ascii)
17	t	up:	23	(code)	23	(usb key)	84	(ascii)
18	y	up:	28	(code)	28	(usb key)	89	(ascii)

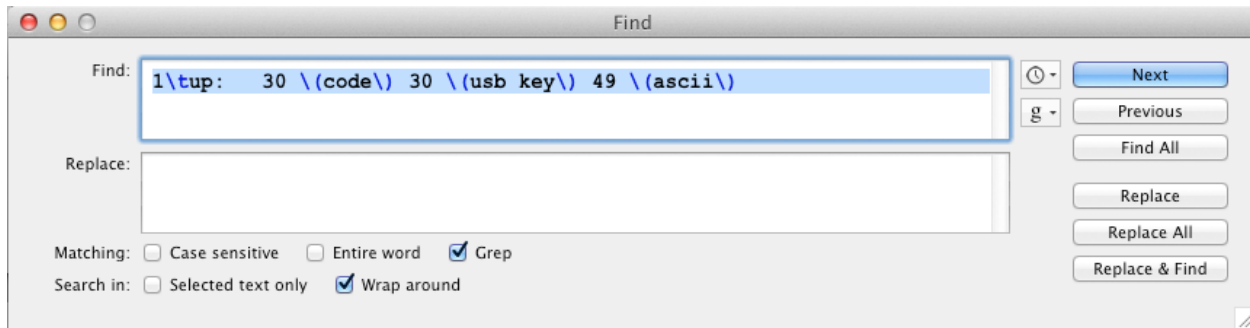
etc.

We're going to use Grep to write some ChuckK code for us. Choose the Find command in TextWrangler's Search menu (Command-F). When the Find dialog opens make sure Grep and Wrap around are selected.

Close the Find dialog and select line 1 of the text. Try to end the selection with the last closing parenthesis.

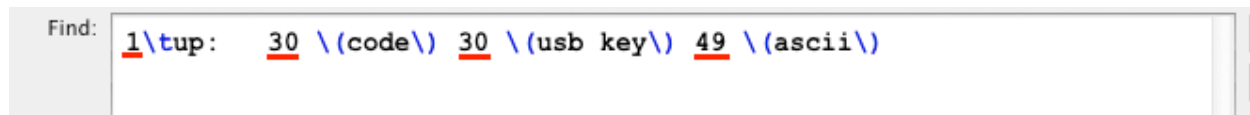
1	1	up:	30	(code)	30	(usb key)	49	(ascii)
2	2	up:	31	(code)	31	(usb key)	50	(ascii)
3	3	up:	32	(code)	32	(usb key)	51	(ascii)
4	4	up:	33	(code)	33	(usb key)	52	(ascii)
5	5	up:	34	(code)	34	(usb key)	53	(ascii)

Type Command-E to automatically enter line 1 into the Find field of the Find dialog. Open the Find dialog again and you should see this.

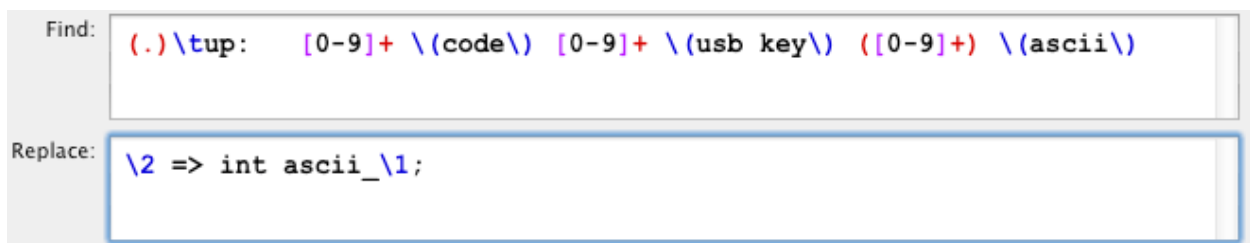


If you see the end of line character `\r` at the end of the selection, delete it.

Carefully change these four numbers



into this. Be careful not to remove any whitespace and be sure to enclose the first and fourth underlined items in parentheses. The `\1` and `\2` in the Replace field will be replaced by whatever is in the first and fourth text locations in each line.



## Click The Replace All Button

Click Replace All and you should see correctly for matted ChuckK code appear.

1	49 => int ascii_1
2	50 => int ascii_2;
3	51 => int ascii_3;
4	52 => int ascii_4;
5	53 => int ascii_5;
6	54 => int ascii_6;
7	55 => int ascii_7;
8	56 => int ascii_8;
9	57 => int ascii_9;
10	48 => int ascii_0;
11	45 => int ascii_-;
12	61 => int ascii_=;
13	81 => int ascii_q;
14	87 => int ascii_w;
15	69 => int ascii_e;
16	82 => int ascii_r;
17	84 => int ascii_t;
18	89 => int ascii_y;

etc.

## What Just Happened

### Find:

. means match any 1 single character

.\* means match any number of characters to the end of the line or until another regular expression symbol like a space is encountered

[0-9] means find any single number in the range zero to nine.

[a-z] means find any single lower case letter

[A-Z] means find any single upper case letter

[0-9a-zA-Z] means find any single alpha-numeric character

[0-9]+ the plus sign means find one or more numeric characters.

( ) means to remember whatever is inside the parentheses save it as a variable for later

\1, \2, ... refer to the variables surrounded by the parenthesis in left to right order.

### Replace:

\1 is the the single character surrounded by parentheses in data column 1

\2 is the characters [0-9]+ surrounded by parentheses (the ASCII code)

### Edit the Code

There are a few lines that would return errors in Chuck. You'll need to changes these.

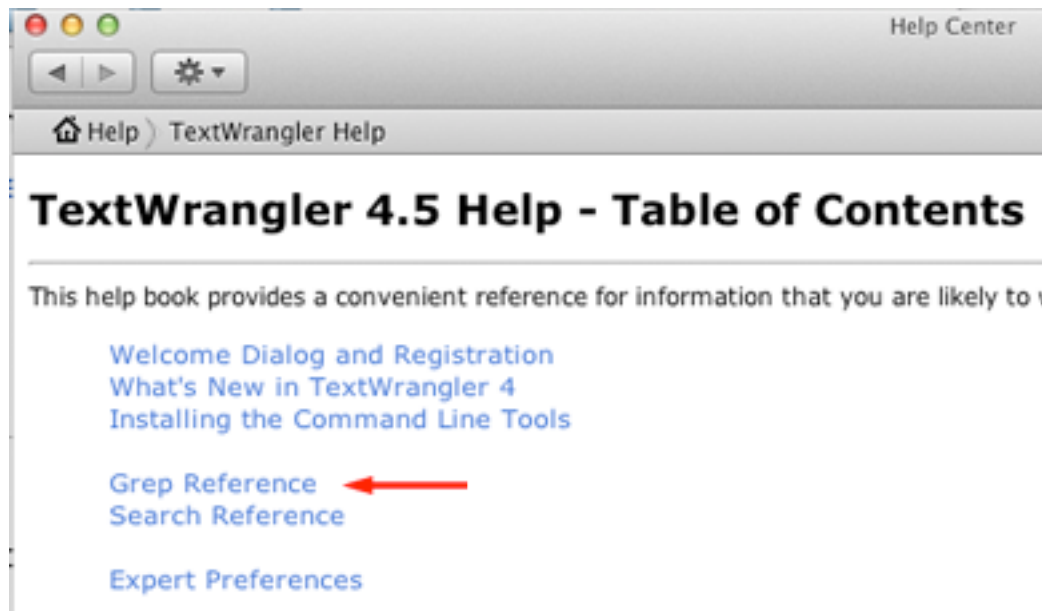
1	45 => int ascii_-; // change to ascii_hyphen
2	61 => int ascii_=; // change to ascii_equals
3	91 => int ascii_[; // change to ascii_leftBracket
4	93 => int ascii_]; // change to ascii_rightBracket
5	92 => int ascii_\; // change to ascii_backSlash
6	59 => int ascii;; // change to ascii_semicolon
7	39 => int ascii'; // change to ascii_singleQuote
8	44 => int ascii_,; // change to ascii_comma
9	46 => int ascii_.; // change to ascii_period
10	47 => int ascii_/.; // change to ascii_forwardSlash

### Save As keyboardASCII.txt

We'll use it again later.

### Learning Grep

If you plan to continue programming or manipulate text based data you need to learn Grep. TextWrangler's Help menu is a good place to start.



## Create the keyboardPlay Chuck Code

Open keyboardPlay.ck in TextWrangler from the m208Lab4 folder.

```

1  // keyboardPlay.ck
2  // started from ChuckK example kb.ck
3  Hid hi;
4  HidMsg msg;
5
6  // which keyboard
7  0 => int device;
8  // get from command line
9  if( me.args() ) me.arg(0) => Std.atoi => device;
10
11 // open keyboard (get device number from command line)
12 if( !hi.openKeyboard( device ) ) me.exit();
13 <<< "keyboard '" + hi.name() + "' ready", "" >>>;
14 <<< "Type Ctrl-C to exit" >>>;
15
16 // ASCII KEYBOARD DEFINITIONS GO HERE
17
18 // infinite event loop
19 while( true )
20 {
21     // wait on event
22     hi => now;
23
24     // get one or more messages
25     while( hi.recv( msg ) )
26     {
27         // check for action type
28         if ( msg.isButtonDown() )
29         {
30         }
31         else if ( msg.isButtonUp() )
32         {
33         }
34     }
35 }
```



## Add the ASCII variable Definitions

Copy and paste the ASCII Keyboard definitions into keyboardPlay.ck at line 16.

```

16 // ASCII KEYBOARD DEFINITIONS GO HERE
17 49 => int ascii_1;
18 50 => int ascii_2;
19 51 => int ascii_3;
20 52 => int ascii_4;
21 53 => int ascii_5;
22 54 => int ascii_6;
23 55 => int ascii_7;
24 56 => int ascii_8;
25 57 => int ascii_9;
26 48 => int ascii_0;
27 45 => int ascii_hyphen;
28 61 => int ascii_equals;
29 81 => int ascii_q;
30 87 => int ascii_w;
31 69 => int ascii_e;
32 82 => int ascii_r;
33 84 => int ascii_t;
34 89 => int ascii_y;
35 85 => int ascii_u;
36 73 => int ascii_i;
37 79 => int ascii_o;
38 80 => int ascii_p;
39 91 => int ascii_leftBracket;
40 93 => int ascii_rightBracket;
41 92 => int ascii_backSlash;
42 65 => int ascii_a;
43 83 => int ascii_s;
44 68 => int ascii_d;
45 70 => int ascii_f;
46 71 => int ascii_g;
47 72 => int ascii_h;
48 74 => int ascii_j;
49 75 => int ascii_k;
50 76 => int ascii_l;
51 59 => int ascii_semicolon;
52 39 => int ascii_singleQuote;
53 90 => int ascii_z;
54 88 => int ascii_x;
55 67 => int ascii_c;
56 86 => int ascii_v;
57 66 => int ascii_b;
58 78 => int ascii_n;
59 77 => int ascii_m;
60 44 => int ascii_comma;
61 46 => int ascii_period;
62 47 => int ascii_forwardSlash;

```

## Create the Keydown Message Code for Chuck

Open the keyboardASCII.txt file you saved.

Select the first line up through the semicolon and type Command-E.

## Find Command

Choose the Find command from TextWrangler's Search menu and enter this. Make sure Grep and Wrap around are selected. Enter these Find and Replace statements.

Find: `.* => int ascii_(.*);`

Replace: `else if ( msg.ascii == ascii_\\1 )\\n{\\n\\tplayIt( \\\"\\1\\\" );\\n}`

### Find:

We're matching line by line and remembering the suffix of ascii\_

### Replace:

My comments are indicated by #

```
else if ( msg.ascii == ascii_\\1 ) # set up the loop condition
\\n                                # new line
{                                # opening code bracket
\\n                                # new line
\\t                                # tab character
playIt( \\\"\\1\\\" );                # call function playIt()
\\n                                # new line
}                                # closing code bracket
# the \\ in line 1 and line 6 will be replaced by the ascii_ suffix from the Find text field
```

## Click The Replace All Button

You should see this.

```
1  else if ( msg.ascii == ascii_1 )
2  {
3      playIt( "1" );
4  }
5  else if ( msg.ascii == ascii_2 )
6  {
7      playIt( "2" );
8  }
9  else if ( msg.ascii == ascii_3 )
10 {
11     playIt( "3" );
12 }
13 else if ( msg.ascii == ascii_4 )
14 {
15     playIt( "4" );
16 }
17 else if ( msg.ascii == ascii_5 )
18 {
19     playIt( "5" );
20 }
21 else if ( msg.ascii == ascii_6 )
22 {
23     playIt( "6" );
24 }
```

## Change the `playIt()` Parameter in These Lines

Find and change these `playIt` parameters to use the actual character typed rather than the `ascii_` definition.

```
1  else if ( msg.ascii == ascii_hyphen )
2  {
3      playIt( "-" );
4  }
5  else if ( msg.ascii == ascii_equals )
6  {
7      playIt( "=" );
8  }
9  else if ( msg.ascii == ascii_leftBracket )
10 {
11     playIt( "[" );
12 }
13 else if ( msg.ascii == ascii_rightBracket )
14 {
15     playIt( "]" );
16 }
17 else if ( msg.ascii == ascii_backSlash )
18 {
19     playIt( "\" );
20 }
21 else if ( msg.ascii == ascii_semicolon )
22 {
23     playIt( ";" );
24 }
25 else if ( msg.ascii == ascii_singleQuote )
26 {
27     playIt( "'" );
28 }
29 else if ( msg.ascii == ascii_comma )
30 {
31     playIt( "," );
32 }
33 else if ( msg.ascii == ascii_period )
34 {
35     playIt( "." );
36 }
37 else if ( msg.ascii == ascii_forwardSlash )
38 {
39     playIt( "/" );
40 }
```

## Delete the else from the first line

```

1  if ( msg.ascii == ascii_1 )
2  {
3      playIt( "1" );
4  }
5  else if ( msg.ascii == ascii_2 )
6  {
7      playIt( "2" );
8  }

```

## Add the KeyDown Message Statements

### Process a Key Press in the While Loop

Key presses are processed in the `msg.isButtonDown()` block inside the `while(true)` loop.

```

// infinite event loop
while( true )
{
    // wait on event
    hi => now;

    // get one or more messages
    while( hi.recv( msg ) )
    {
        // check for action type
        if( msg.isButtonDown() )
        {
        }
        else
        {
        }
    }
}

```

## Copy Paste the Keydown Messages

Copy and paste the keydown messages from TextWrangler into the `msg.isButtonDown()` code block.

```
69 // infinite event loop
70 while( true )
71 {
72     // wait on event
73     hi => now;
74
75     // get one or more messages
76     while( hi.recv( msg ) )
77     {
78         // check for action type
79         if ( msg.isButtonDown() )
80         {
81             if ( msg.ascii == ascii_1 )
82             {
83                 playIt( "1" );
84             }
85             else if ( msg.ascii == ascii_2 )
86             {
87                 playIt( "2" );
88             }
89             else if ( msg.ascii == ascii_3 )
90             {
91                 playIt( "3" );
92             }
```

etc.



## Create The playIt() Function

The first version of playIt looks like this and used to see if the code can be run free of errors. Add it to keyboardPlay.ck.

```

62 47 => int ascii_forwardSlash;
63
64 function void playIt( string str )
65 {
66     <<< "\tyou typed:\t", str >>>;
67 }
68
69 // infinite event loop
70 while( true )

```

## Run keyboardPlay.ck in the Terminal

If everything works you should see this

```

511,je@jemac:~/Desktop/m208Lab4$ chuck "keyboardPlay.ck"
keyboard 'Keyboard' ready
"Type Ctrl-C to exit" : (string)
      you typed:      1
12      you typed:      2
3       you typed:      3
4       you typed:      4
5       you typed:      5
6       you typed:      6
7       you typed:      7
8       you typed:      8
9       you typed:      9
0       you typed:      0
^C[chuck]: cleaning up...

```

1 shows up on next line  
just ignore it

## Save A Copy As keyboardPlayTemplate.ck

Congratulations. You've created a skeleton program that you can reuse again and again whenever you want Chuck to respond to the keyboard.

## The ABC Song

### Save keyboardPlay.ck as ABCSong.ck

The data folder in the m208Lab4 folder contains 26 wave file samples of each alphabet letter. We'll play them by typing the keys using the SndBuf object we learned about in Lab 3.

### Create A Sndbuf Object And Modify The playIt() Function

```
SndBuf buf => dac;|
function void playIt( string str )
{
    // samples are in the data folder
    "data/" + str + ".wav" => string filename;
    filename => buf.read;

    0 => buf.pos;
    buf.length() => now;
}
```

### Make Chuck Sing the Song

Type a b c d e f g ... to the tune of "Twinkle Twinkle Little Star."

### Let's Record It

Chuck's WvOut object can be used to save the samples you hear to a wave file. Modify this line.

```
SndBuf buf => dac;
```

to read

```
SndBuf buf => WvOut w => dac;
"abcSong.wav" => w.wavFilename;
```

### WvOut Documentation

From: [http://chuck.cs.princeton.edu/doc/program/ugen\\_full.html#WvOut](http://chuck.cs.princeton.edu/doc/program/ugen_full.html#WvOut)

## [ugen]: WvOut (STK Import)

- *STK audio data output base class.*

This class provides output support for various audio file formats. It also serves as a base class for "realtime" streaming subclasses.

WvOut writes samples to an audio file. It supports multi-channel data in interleaved format. It is important to distinguish the tick() methods, which output single samples to all channels in a sample frame, from the tickFrame() method, which takes a pointer to multi-channel sample frame data.

WvOut currently supports WAV, AIFF, AIFC, SND (AU), MAT-file (Matlab), and STK RAW file formats. Signed integer (8-, 16-, and 32-bit) and floating-point (32- and 64-bit) data types are supported. STK RAW files use 16-bit integers by definition. MAT-files will always be written as 64-bit floats. If a data type specification does not match the specified file type, the data type will automatically be modified. Uncompressed data types are not supported.

Currently, WvOut is non-interpolating and the output rate is always `Stk::sampleRate()`.

by Perry R. Cook and Gary P. Scavone, 1995 - 2002.

(control parameters)

- **.matFilename** - ( string , WRITE only ) - *open matlab file for writing*
- **.sndFilename** - ( string , WRITE only ) - *open snd file for writing*
- **.wavFilename** - ( string , WRITE only ) - *open WAVE file for writing*
- **.rawFilename** - ( string , WRITE only ) - *open raw file for writing*
- **.aifFilename** - ( string , WRITE only ) - *open AIFF file for writing*
- **.closeFile** - ( string , WRITE only ) - *close file properly*

See: <http://wiki.cs.princeton.edu/index.php/Chuck/Record>  
for more information.

## Run The Program

Play the complete song.

**Important:** When finished press the Stop Virtual Machine button in the Virtual Machine window.

You should see an abcSong.wav file in the m208Lab4 folder. Open and play it in Audacity.

## Chuck MIDI

I'll post instructions for getting this lab to work on your laptops for both Mac and Windows on the course web page as homework assignments. It should be there by Sunday morning.

### Save keyboardPlay.ck as keyboardPlayMIDI.ck

The MIDI programs you need to use with ChucK are already setup in the Lab. I've placed instructions on the web page to help you use MIDI on your laptop. It's easier on Windows.

### Open Au Lab

Open /Applications/Au Lab



AU Lab.app

### Open Terminal and Type `chuck --probe`

```
513,je@jemac:~/Desktop/m208Lab4$ chuck --probe
[chuck]: found 7 device(s) ...
[chuck]: -----( audio device: 1 )-----
```

At the end of the output you'll see all MIDI inputs and outputs ChucK can find. In this case it found one input and two outputs.

```
[chuck]: -----( chuck -- 1 MIDI inputs )-----
[chuck]:      [0] : "IAC Driver Bus 1"
[chuck]:
[chuck]: -----( chuck -- 2 MIDI outputs )-----
[chuck]:      [0] : "IAC Driver Bus 1"
[chuck]:      [1] : " DLSMusicDevice"
[chuck]:
```

### IAC Driver Bus 1

Input and Output [o] use the IAC driver. IAC stands for Inter Application Communication and is part of MacOS. The IAC Driver allows you to connect ChuckK to many commercial MIDI applications like GarageBand, LogicPro, Ableton Live, Reaktor, Reason, and others. ChuckK can use this.

## DLSMusicDevice

This is the General MIDI software synthesizer supplied by Apple and is part of MacOS. ChuckK can use this.

## Initialize Midi Output

Add these lines above the current playIt() function.

```

61 46 => int ascii_period;
62 47 => int ascii_forwardSlash;
63
64 MidiOut mout;
65 MidiMsg mmsg;
66
67 mout.open( 1 ) => int amIOpen;
68 if (!amIOpen )
69     me.exit();
70
71 function void playIt( string str )
72 {

```

### Lines 64-69

The mout variable is a MidiOut object that represents one of the available MIDI Output ports on the computer. The output from chuck —probe showed that we could use either output port 0 or 1. Line 67 attempts to open port 1, the DLS Music Device (the built in MAC MIDI synthesizer). If the port was opened successfully the variable amIOpen will be set to 1. If the port could not be opened it will be set to 0. Line 68 checks that result. The ! means not, so if "not amIOpen" then exit. No point in trying to hear MIDI if we can't talk to it.

## Replace The Playit() Function

Use this code.



```

70
71 function void playIt( int note, int vel )
72 {
73     0x90 => mmsg.data1;
74     note => mmsg.data2;
75     vel => mmsg.data3;
76     mout.send( mmsg );
77     <<< "note vel\t", note, "\t", vel >>>;
78 }

```

**Lines 74-81**

These three lines construct the MIDI message.

**Line 79**

Send the message so you can hear it.

**Line 80**

Display the note number and velocity in the Console Monitor window.

**MIDI Messages**

Most MIDI messages consist of three numbers referred to as status, data1, and data2. All three numbers are restricted to an 8 bit range, 0-255, called a byte in computer terms. The data1 and data2 bytes occupy the lower half of the range (0-127) and the status byte covers the upper half (128-255).

**Status Bytes are Hexadecimal Numbers - Base 16**

Status bytes are easiest to understand in the hexadecimal number system (base 16) which counts from 0-15 using the numbers 0123456789ABCDEF. You enter a hex number in Chuck by prefixing with 0x (zero x). There are 8 status messages. All status and data bytes can be written as two hex digits. The first hex digit (8 - F) of a status byte determines the type of MIDI message. The second hex digit (0 - F) determines which of the 16 MIDI channels the message applies to. In Lab 4 we'll use channel 0.

Status Byte in Hex	Message Type
80 - 8F	Note Off
90 - 9F	Note On
A0 - AF	After touch applies to all notes on a channel
B0 - BF	Controller messages for knobs, pedals, sliders, etc.
C0 - CF	Patch change messages change instruments
D0 - DF	After touch applies to individual notes on a channel
E0 - EF	Pitch bend

F0 - FF	System messages
---------	-----------------

We'll begin with the two most used MIDI messages : the message for note on (NON), and the message for the note off (NOF).

## Non Messages - Status Data1 Data2

Note On messages turn on a MIDI note that will continue to sound until it receives a matching Note Off message. Some instrument sounds, like a guitar pluck, die away naturally because of built in decay. Other instruments, like an organ, continue to sound until the Note Off message is received. You should always test MIDI programs you write with an organ sound so you can hear if there are any "stuck" notes, or notes that don't turn off.

Both data1 and data2 bytes are numbers in the range 0-127. The data1 byte is the MIDI note number based on the piano keyboard layout. Middle C is note number 60. The next higher key, C#, is 61, followed by D, 62, etc. The same note name an octave higher adds 12 to the MIDI note number. The same note name an octave lower subtracts 12 from the MIDI note number. The C below middle C is 48 and the C above middle C is 72.

The data2 byte represents how loud or soft the note is played and is termed the velocity. Zero is silence and 127 maximum loudness.

### Example NON message - 0x90 60 100

0x90 - Attention computer! this is a note on message for channel 0  
 60 - Turn on the current instrument sound for middle C (261 Hz)  
 100 - Set its volume to 100 using a range from 0 (silence) to 127 (maximum).

## Nof Messages - Status Data1 Data2

Note Off messages are used to turn a previous Note On, off. We'll turn notes off by sending a Note On message with a velocity of 0.

### Example NOF message - 0x90 60 0 (this is the one used in today's lab)

0x90 - Attention computer! this is a Note On message for channel 0  
 60 - Turn on the current instrument sound for middle C (261 Hz)  
 0 - Set its volume to 0 (silence)

### Example NOF message - 0x80 60 0

0x80 - Attention computer! this is a Note Off message for channel 0  
 60 - Turn off the current instrument sound for middle C (261 Hz)

- o - can be anything, it's already a note off message

## Chuck's MidiMsg Object

MIDI data holder

- .data1 (int, READ/WRITE) first byte of data (member variable)
- .data2 (int, READ/WRITE) second byte of data (member variable)
- .data3 (int, READ/WRITE) third byte of data (member variable) (see MIDI tutorial)

From the ChuckK manual: [http://booki.flossmanuals.net/chuck/\\_full/](http://booki.flossmanuals.net/chuck/_full/)

ChuckK is contrary to most MIDI references and calls the status byte data1, the note number data2, and the velocity data3.

## Change the Parameters of All playIt()'s Inside The While Loop

The playIt function for MIDI expects to see two parameters, one for the MIDI note number and one for the velocity (volume of that note). We'll need to modify every playIt() function inside the while loop to take a two int parameters. There are a lot of them and they all need to be done by hand. This is what you need to do. Notice that the MIDI note number is increasing by 1 each time. You'll eventually reach note number 81, producing an ascending chromatic scale, or scale of half steps. A half step on the piano is the interval between any note and its nearest neighbor. The velocity vales are all 100.

## Open TextWrangler's Find Dialog

Enter these statements and click Replace All.



The screenshot shows the Find dialog box in TextWrangler. The 'Find:' field contains the text `playIt.*;`. The 'Replace:' field contains the text `playIt( , 100 );`. Below the fields, there are checkboxes for 'Matching: Case sensitive', 'Entire word', and 'Grep' (which is checked). At the bottom, there are checkboxes for 'Search in: Selected text only' and 'Wrap around' (which is checked).

You should see this.

```
89 // check for action type
90 if ( msg.isButtonDown() )
91 {
92     if ( msg.ascii == ascii_1 )
93     {
94         playIt( , 100 );
95     }
96     else if ( msg.ascii == ascii_2 )
97     {
98         playIt( , 100 );
99     }
100     else if ( msg.ascii == ascii_3 )
101     {
102         playIt( , 100 );
103     }
```

All playIt velocities are set to 100. Next set the note numbers. Start with 36 and increment by one until done.

```

92         if ( msg.ascii == ascii_1 )
93         {
94             playIt( 36 , 100 );
95         }
96         else if ( msg.ascii == ascii_2 )
97         {
98             playIt( 37 , 100 );
99         }
100        else if ( msg.ascii == ascii_3 )
101        {
102            playIt( 38 , 100 );
103        }
104        else if ( msg.ascii == ascii_4 )
105        {
106            playIt( 39 , 100 );
107        }
108        else if ( msg.ascii == ascii_5 )
109        {
110            playIt( 40 , 100 );
111        }

```

We're almost ready to test it, but first we're going to write a function that will allow us to test different instrument sounds.

## General MIDI (GM) Instrument Sound Set

The General MIDI specification defines 128 instruments in 16 different families as well as the MIDI patch number assigned to that instrument.

```

=====
List all GM instruments
=====

```

Piano =====

- 0: Acoustic Grand Piano
- 1: Bright Acoustic Piano
- 2: Electric Grand Piano
- 3: Honky-tonk Piano
- 4: Electric Piano 1

5: Electric Piano 2  
 6: Harpsichord  
 7: Clavinet  
 Chromatic Percussion =====  
 8: Celesta  
 9: Glockenspiel  
 10: Music Box  
 11: Vibraphone  
 12: Marimba  
 13: Xylophone  
 14: Tubular Bells  
 15: Dulcimer  
 Organ =====  
 16: Drawbar Organ  
 17: Percussive Organ  
 18: Rock Organ  
 19: Church Organ  
 20: Reed Organ  
 21: Accordion  
 22: Harmonica  
 23: Tango Accordion  
 Guitar =====  
 24: Acoustic Guitar (nylon)  
 25: Acoustic Guitar (steel)  
 26: Electric Guitar (jazz)  
 27: Electric Guitar (clean)  
 28: Electric Guitar (muted)  
 29: Overdriven Guitar  
 30: Distortion Guitar  
 31: Guitar harmonics  
 Bass =====  
 32: Acoustic Bass  
 33: Electric Bass (finger)  
 34: Electric Bass (pick)  
 35: Fretless Bass  
 36: Slap Bass 1  
 37: Slap Bass 2  
 38: Synth Bass 1  
 39: Synth Bass 2  
 Strings =====



40: Violin  
41: Viola  
42: Cello  
43: Contrabass  
44: Tremolo Strings  
45: Pizzicato Strings  
46: Orchestral Harp  
47: Timpani

Ensemble =====

48: String Ensemble 1  
49: String Ensemble 2  
50: SynthStrings 1  
51: SynthStrings 2  
52: Choir Aahs  
53: Voice Oohs  
54: Synth Voice  
55: Orchestra Hit

Brass =====

56: Trumpet  
57: Trombone  
58: Tuba  
59: Muted Trumpet  
60: French Horn  
61: Brass Section  
62: SynthBrass 1  
63: SynthBrass 2

Reed =====

64: Soprano Sax  
65: Alto Sax  
66: Tenor Sax  
67: Baritone Sax  
68: Oboe  
69: English Horn  
70: Bassoon  
71: Clarinet

Pipe =====

72: Piccolo  
73: Flute  
74: Recorder  
75: Pan Flute

76: Blown Bottle  
 77: Shakuhachi  
 78: Whistle  
 79: Ocarina  
 Synth Lead =====  
 80: Lead 1 (square)  
 81: Lead 2 (sawtooth)  
 82: Lead 3 (calliope)  
 83: Lead 4 (chiff)  
 84: Lead 5 (charang)  
 85: Lead 6 (voice)  
 86: Lead 7 (fifths)  
 87: Lead 8 (bass + lead)  
 Synth Pad =====  
 88: Pad 1 (new age)  
 89: Pad 2 (warm)  
 90: Pad 3 (polysynth)  
 91: Pad 4 (choir)  
 92: Pad 5 (bowed)  
 93: Pad 6 (metallic)  
 94: Pad 7 (halo)  
 95: Pad 8 (sweep)  
 Synth Effects =====  
 96: FX 1 (rain)  
 97: FX 2 (soundtrack)  
 98: FX 3 (crystal)  
 99: FX 4 (atmosphere)  
 100: FX 5 (brightness)  
 101: FX 6 (goblins)  
 102: FX 7 (echoes)  
 103: FX 8 (sci-fi)  
 Ethnic =====  
 104: Sitar  
 105: Banjo  
 106: Shamisen  
 107: Koto  
 108: Kalimba  
 109: Bag pipe  
 110: Fiddle  
 111: Shanai

**Percussive =====**

112: Tinkle Bell  
 113: Agogo  
 114: Steel Drums  
 115: Woodblock  
 116: Taiko Drum  
 117: Melodic Tom  
 118: Synth Drum  
 119: Reverse Cymbal

**Sound Effects =====**

120: Guitar Fret Noise  
 121: Breath Noise  
 122: Seashore  
 123: Bird Tweet  
 124: Telephone Ring  
 125: Helicopter  
 126: Applause  
 127: Gunshot

**Patch Change Messages - Status Data1 Data2**

Patch Change messages change instruments. The data2 byte is used in ChuckK but is ignored.

**Example Patch Change message - 0xC0 0 0**

0xC0 - Attention computer! this is a Note On message for channel 0  
 0 - Use patch number 0 "Acoustic Grand Piano"  
 0 - ignored

**Example Patch Change message - 0xC0 18 0**

0xC0 - Attention computer! this is a Note On message for channel 0  
 18 - Use patch number 18 "Rock Organ"  
 0 - ignored

## Add The patchChange Function

```

71 function void playIt( int note, int vel )
72 {
73     0x90 => mmsg.data1;
74     note => mmsg.data2;
75     vel => mmsg.data3;
76     mout.send( mmsg );
77     <<< "note vel\t", note, "\t", vel >>>;
78 }
79
80 function void patchChange( int patchNum )
81 {
82     0xC0 => mmsg.data1;
83     patchNum => mmsg.data2;
84     0 => mmsg.data3;
85     mout.send( mmsg );
86 }
87 // Change to Rock Organ
88 patchChange( 18 );
89
90 // infinite event loop
91 while( true )
92 {

```

## Run The Program

Type a few keys and —

## Houston, We've got a problem- Stuck Notes!

You'll have to quit Au Lab to turn them off.

## Save Your Work And Quit Terminal

Whenever you close Au Lab it's a good idea to close other programs that are using it.

## Restart Au Lab And Then Restart Terminal In That Order

Many, but not all, MIDI programs look for Au Lab at startup so you should get in the habit of starting Au Lab first.

## All Notes Off

Most MIDI programs have a Panic button to turn all notes off. Let's write a ChuckK function to do that.

```

90  function void allNotesOff()
91  ▼ {
92      for ( 0 => int ix; ix < 128; ix++ )
93  ▼ {
94      playIt( ix, 0 );
95  └ }
96  └ }
97
98  // infinite event loop
99  while( true )
100 ▼ {

```

## Use the Space Bar to Turn All Notes Off

Now we need to find a key to press to turn all notes off. Let's use the space bar (ASCII 32). Add these lines at the beginning of the key press statements in the while loop. Since we're panicking we might as well do it first. Then add and else in front of the `ascii_1` line.

```

// check for action type
if ( msg.isButtonDown() )
{
    if ( msg.ascii == 32 ) // space bar ASCII code
    {
        allNotesOff();
    }
    else if ( msg.ascii == ascii_1 )
    {
        playIt( 36 , 100 );
    }
}

```

## Run The Program Again

This time when you hear the stuck notes, press the space bar.

## The Real Fix

The stuck note problem occurred because we forgot to send Note Off messages. I used the organ patch to make stuck notes obvious. Inside the while loop the if (msg.IsButtonDown) block processes all key downs and sends Note On messages.

We'll everything inside the msg.isButtonDown() block between the opening and closing curly brass and and paste them into the msg.isButtonUp() block.

Set the velocity parameter in every isButtonUp line to zero. An easy way to do it is to copy the buttonDown messages into a new document and use the Find dialog to replace all 100's with 0's.

After this fix key downs will send Note On messages and key ups will send Note Off messages. Notes will continue to sound until you let the key up. You can play chords.

```

else if ( msg.isButtonUp() )
{
if ( msg.isButtonDown() )
{
    if ( msg.ascii == 32 ) // space bar ASCII code
    {
        allNotesOff();
    }
    else if ( msg.ascii == ascii_1 )
    {
        playIt( 36 , 0 );
    }
    else if ( msg.ascii == ascii_2 )
    {
        playIt( 37 , 0 );
    }
    else if ( msg.ascii == ascii_3 )
    {
        playIt( 38 , 0 );
    }
}
}

```

## Run The Program

Test it out.

## The Drum Channel

Sending messages to channel 10 ( status byte `0x99` ) will produce drum sounds.  
Channel 1 in english is channel zero in MIDI.

## General MIDI Drum Note Assignments

Individual drum sounds are assigned to these MIDI note numbers on channel 10.

```
=====
List GM drum kit MIDI note numbers.
```

```
All GM drum notes are assigned to MIDI channel 10
=====
```

```
35: Bass Drum 2
36: Bass Drum 1 // Lab 4 starts with 36
37: Side Stick
38: Snare Drum 1
39: Hand Clap
40: Snare Drum 2
41: Low Tom 2
42: Closed Hi-hat
43: Low Tom 1
44: Pedal Hi-hat
45: Mid Tom 2
46: Open Hi-hat
47: Mid Tom 1
48: High Tom 2
49: Crash Cymbal 1
50: High Tom 1
51: Ride Cymbal 1
52: Chinese Cymbal
53: Ride Bell
54: Tambourine
55: Splash Cymbal
56: Cowbell
57: Crash Cymbal 2
```

58: Vibra Slap  
59: Ride Cymbal 2  
60: High Bongo  
61: Low Bongo  
62: Mute High Conga  
63: Open High Conga  
64: Low Conga  
65: High Timbale  
66: Low Timbale  
67: High Agogo  
68: Low Agogo  
69: Cabasa  
70: Maracas  
71: Short Whistle  
72: Long Whistle  
73: Short Guiro  
74: Long Guiro  
75: Claves  
76: High Wood Block  
77: Low Wood Block  
78: Mute Cuica  
79: Open Cuica  
80: Mute Triangle  
81: Open Triangle

Modify the playIt() function to use status byte 0x99. To be consistent you should also change the status byte in the allNotesOff function. You do not have to modify the patchChange function, the drum channel ignores it.



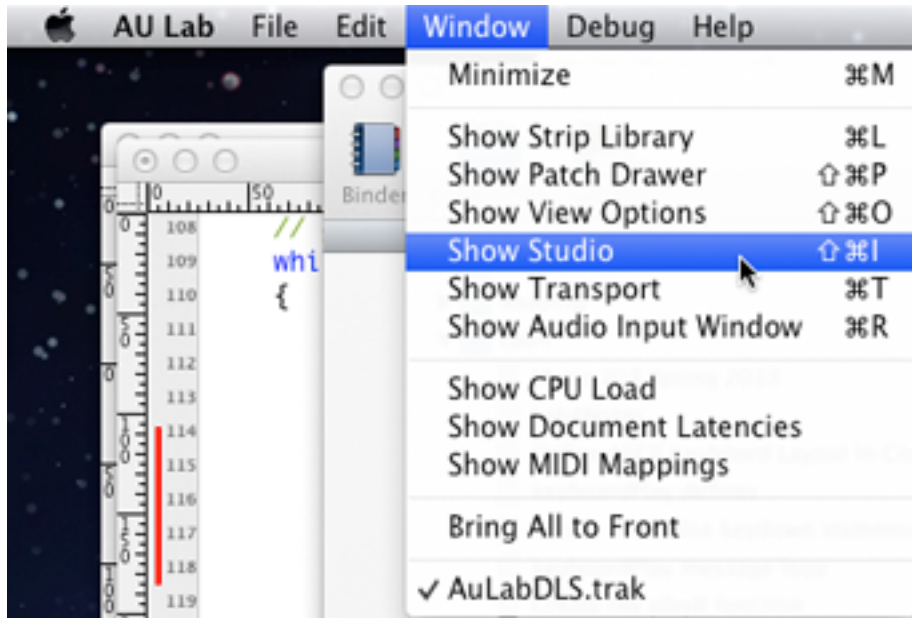
# Recording MIDI From Chuck Into Audacity

## MACINTOSH - Soundflower and Au Lab

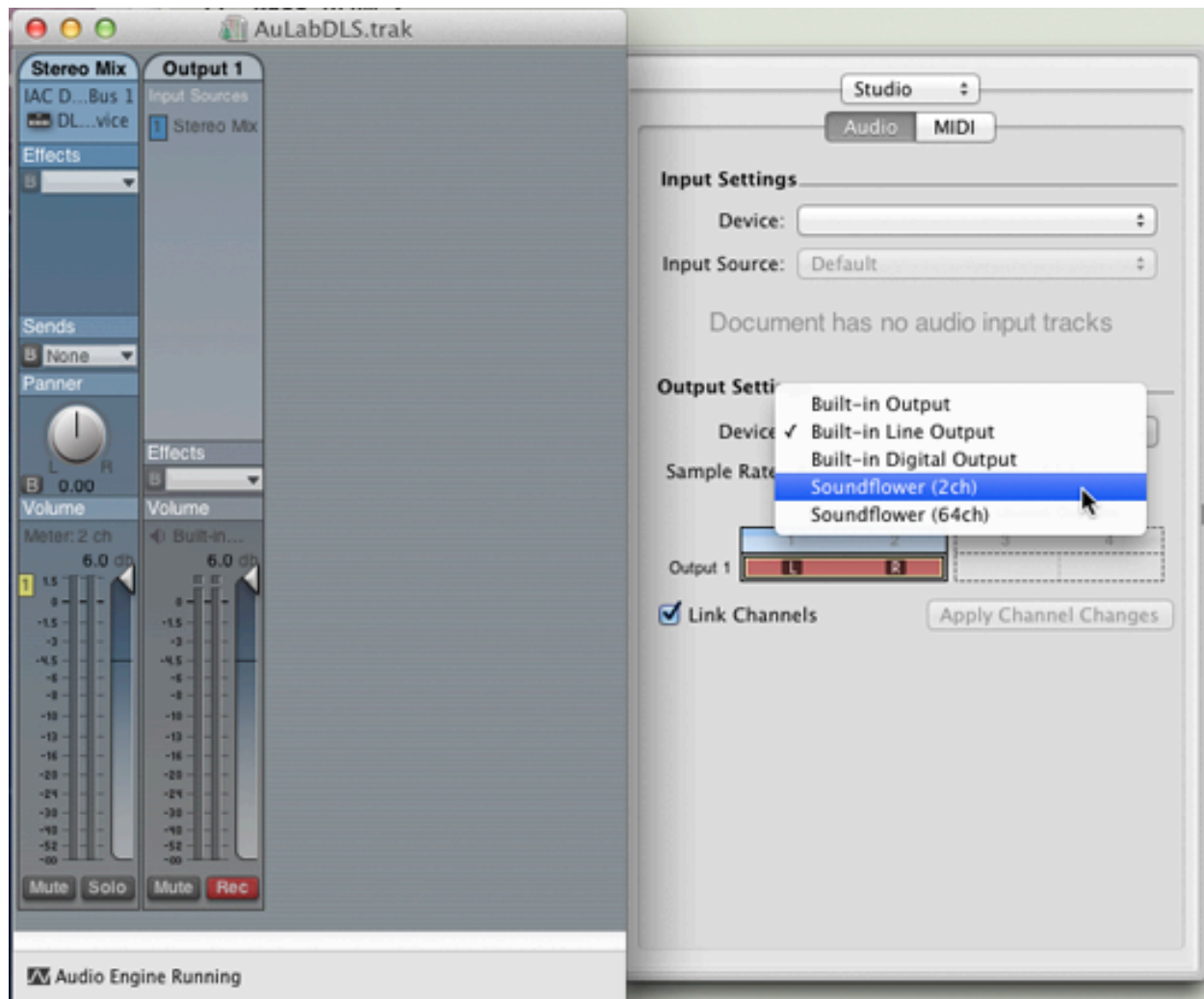
Soundflower is free utility that allows you to play audio in one application and record it in another. It's a Mac only program and is available at <http://cycling74.com/soundflower-landing-page/>. You can install it on your laptop. For

## Au Lab Setup

Choose Show Studio from the Au Lab Window menu.

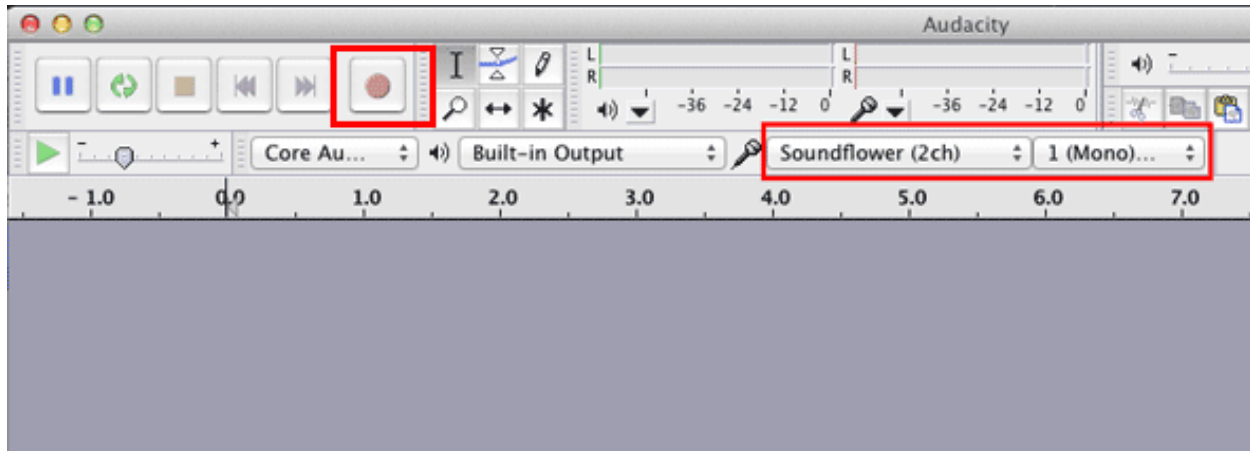


In the Studio Panel set the Output Device to Soundflower (2ch). Remember the current output device name. I took this picture at home and the lab may be different.



## Audacity Setup

Open Audacity and choose Soundflower (2ch) as the sound input device using 1 (Mono) track.



Click the Audacity record button (Red Circle) and then return to miniAudicle and start typing. You should see the recorded audio update in real time in the Audacity window.

## Set Au Lab Output Device Back To Its Original Output Device

Done with Lab 4

Windows users can watch this YouTube video to see one way to do it in Windows:  
[http://www.youtube.com/watch?v=mjQ\\_qS-LaoU](http://www.youtube.com/watch?v=mjQ_qS-LaoU)