CalcVox - Journal 2

Ty Gillespie

December 19, 2023

Goals:

1. Round the tinyexpr result to two decimal places using C++ streams by the end of Monday. Completed
2. Implement support for multiple pin layouts using #define statements by the end of Monday. Completed
3. Implement clearing the entire equation with a button press by the end of Wednesday. Completed
4. Implement deletion by character by the end of Friday. Completed
5. Implement different audio stream types for different versions of the hardware. Completed
6. Debug the software on our new H1 hardware. Partially completed (we hit issues)
7. Meet with advisors in-person on Monday. Completed
8. Port the software to use Flite as a text-to-speech provider by the end of Wednesday. Completed
9. Work on Journal 2 and debug hardware problems by the end of Friday.

My Research and What I Learned: *Rounding the Calculation Results:* For evaluating basic mathematical expressions, I'm using a C++ library called TinyExpr. This library works extremely well, but it has one problem. The output is always given as a massive floating point number, with lots of trailing zeros. For example, if I typed

```
35+42
```

it would give me

```
77.00000000
```

Although accurate, this is a lot of unnecessary information that very often gets in the way of doing math productively. As a result, I wrote a custom eval function that automatically rounds the result to two decimal places. I want to make this more dynamic and smarter later (i.e. only trim numbers that are zeros), but it works for now. It looks like the following:

```
std::string eval(
const std::string& expression,
const int precision
) {
    te\_variable vars[] = {};
    int err;
    te\_expr* expr = te\_compile(
expression.c_str(), vars, 0, &err
);
    if (!expr) {
     return "Error";
    }
    double result = te\_eval(expr);
    te\_free(expr);
    std::ostringstream result\_stream;
    result\_stream <<
    std::fixed << std::setprecision(precision)
    << result;
    return result\_stream.str();
}
```

This does the following:

1. Attempts to compile the expression using TinyExpr.
2. If this fails (most likely due to a syntax error), it returns the string "Error", which will be spoken to the user.
3. Otherwise, the expression is evaluated then freed.
4. After that, I create a C++ string stream and write the result to it, setting the precision of the number as I go.
5. And finally, the result is returned to the user.

This works quite well, however as I said I definitely want to make it more dynamic at some point. My current thoughts on how to do this are to parse the string, and read every character after the decimal point. If the next character is a zero and there's not another number for at least 4 characters, stop adding characters.

Supporting multiple Pin Layouts: We have multiple prototypes and boards for this project, and as such, needed a way to differentiate between them in code. I came up with a basic system using C++ macros (A.K.A. conditional defines). I have a file, called pins.h, that looks like this:

```
#pragma once

#if defined(CALCVOX_PROTOTYPE)
#define USE_ANALOG_OUTPUT
const byte ROWS = 3;
const byte COLUMNS = 5;
byte cols[COLUMNS] = {23, 22, 14, 32, 15};
byte rows[ROWS] = {33, 27, 12};
std::string keys[COLUMNS][ROWS] = {
{".", "0", "="},
{"1", "2", "3"},
{"4", "5", "6"},
{"7", "8", "9"},
{"+", "delete", "all_clear"}
};

#elif defined(CALCVOX_H1)
#define USE_I2S_OUTPUT
const byte ROWS = 8;
const byte COLUMNS = 8;
byte cols[COLUMNS] = {3, 19, 18, 5, 4, 33, 15, 32};
byte rows[ROWS] = {34, 39, 36, 14, 22, 23, 1, 21};
std::string keys[COLUMNS][ROWS] = {
{"quit", "options", "p1", "p2", "p3", "p4", "p5", "p6"},
// Emitted for brevity.
{"on", "0", ".", "(-)", "=", "2nd", "apps", "alpha"}
};
int i2s_ws = 25;
int i2s_din = 26;
int i2s_bclk = 2;


#elif defined(CALCVOX_H2)

#endif
```

If CALCVOX_PROTOTYPE is defined (i.e. we're running on the prototype board), we use analog audio output, and also have a much smaller set of pins and buttons. Meanwhile, if we're on CALCVOX_H1 (the big PCB with an 8x8 button matrix), we use I2S output, and define information needed for all the buttons. I also have an empty section for CALCVOX_H2, if/when we get around to making a second version.

Implementing Deletion: Deleting (both by symbol and entirely) is an extremely important feature of a calculator. As such, I went about adding it to CalcVox. Clearing the entire equation turned out to be just about as simple as I thought; simply null the string. The code to do so looks like this:

```
} else if (key == "all_clear") {
if (!current_equation.empty()) {
current_equation = "";
flite.say("All clear");
} else {
flite.say("Empty");
}
} else if (key == "delete") {
```

The way this works is actually extremely simple. When the key to clear all is pressed, the code checks if the equation is already empty. If it is, it informs the user. Otherwise, it clears the equation, and informs the user that it did so. Small sidenote: I've never seen a talking calculator that differentiates between clearing an already empty equation and clearing any other equation – they all just say "clear" or "all clear" when you press that key ,no matter if the equation is empty or not. CalcVox having this is such a small feature, but it truly is the small features that can make or break it.

Switching Text-to-Speech Providers: The original text-to-speech provider we were using (Espeak) proved to be problematic starting around week 5. While Espeak is a very mature project that has had a lot of work done on it, the Arduino port was a direct port from Unix. As such, it was extremely bloated, including things such as a simulation of an entire Posix filesystem. This started causing us problems around week 5, when we suddenly became unable to flash to the new hardware due to running out of flash. As such, we started exploring other options. After a lot of research, we eventually settled on Flite (Festival-Lite). Updating the code to use it was actually fairly straightforward. The same person who ported Espeak also ported Flite, so the API was very similar. I mainly had to change how audio streams were initialized, change all the calls to Flite* to Espeak*, and it compiled. Just because the code compiles doesn't mean it works, though. On Calcvox H1 (the current prototype with the massive button matrix), it speaks just fine. However, on the original prototype, I only get gibberish for output. I suspect it has something to do with different audio stream types – the prototype uses analog (mainly due to a lack of hardware), and H1 uses I2S. Based on how the audio sounds my best guess is that it's a sample rate/bitrate mismatch, although I'm still uncertain. More research needs to be done here.

Accomplishments:

1. Successfully implemented basic rounding for calculation results.
2. Made the code support multiple pin layouts simply by changing one line to whatever pin layout you're compiling for.
3. Implement equation deletion (both by character and entirely).
4. Support multiple audio stream types using the same mechanism I used for different pin layouts.

5. Met with an advisor in-person to discuss plans for the project, as well as to explore new types of hardware.

6. Switch away from using Espeak to Flite for text-to-speech.

Progress on Timeline: My timeline has gotten all sorts of out-of-sync with where I am currently, primarily due to extremely unexpected issues like us running out of flash super early in the project. It didn't help either that whenever a massive issue that I just did not know how to fix would pop up it would entirely zap my motivation for a day or two. All that said, though, I'm still extremely confident that the project will be completed on time, even earlier if we're able to make more massive leaps like we did in the first four weeks. Despite all the setbacks we're still making amazing progress.

Sources:

- https://github.com/pschatzmann/arduino-Flite
- https://www.geeksforgeeks.org/round-in-cpp/
- https://en.cppreference.com/w/cpp/numeric/math/round
- https://stackoverflow.com/questions/14596236/rounding-to-2-decimal-points
- https://cplusplus.com/reference/string/string/substr/
- https://github.com/pschatzmann/arduino-audio-tools/wiki/Introduction
- https://github.com/pschatzmann/arduino-audio-tools/wiki/Working-with-PlatformIO