# Arduino UNO を用いた電卓の製作

## Making a calculator with Arduino UNO

BQ23077　　奥山悠大

BQ23077　　Yudai Okuyama

芝浦工業大学　無線研究部

Shibaura Institute of Technology, Ham radio club

## 1.　動機

電子工作を行ったことがなかったため,以前から興味のあった電卓の自作をすることにした。電卓は誰もが持っており,日常的に使用される電子機器である．四則演算を行う数式で一般的に用いられる方法は中置記法と呼ばれる．この方法とは異なり演算子が非演算子のあとに来る,後置記法（逆ポーランド記法）を用いた電卓を製作したいと考えた.

## 2.　目的

ArduinoUNO を用いて逆ポーランド電卓を製作し,電子工作や電卓,Arduino についての理解を深める.

## 3.　製作

一般的に使用される中置記法とは異なる順番で記述する,後置記法を使用する電卓を制作する.

回路図(別紙図 1)の作成には KiCAD を使用した.キーパッドはユニバーサル基盤,タクトスイッチ,抵抗器を用いて作成した(別紙図 2)． 押したスイッチによって電圧が変化する抵抗分圧方式を用いることで,使用する I/O ピンを 1 つにすることができた．各スイッチには数字や演算子,指示がそれぞれ対応する(別紙図 3)． Arduino 本体とは液晶シールドを経由して GND,A1(I/O),5V の 3 つとワイヤで接続されている.(別紙図 4) GND と A1 の電圧差でどのスイッチが押されたかを認識する．キーが正常に認識されるかを確認し,問題が無かったためコードの制作に取り掛かった.

コードの作成,コンパイル,書き込みには ArduinoIDE を使用した.

入力された文字が数字,演算子,指示のどれかによって分岐する(別紙図 5).

数字が入力された場合,文字列の後ろに追加する.演算子の場合,入力されていた数字を float 型に変換してスタック．スタックされた数字二つを取り出して計算．計算結果をスタック.

指示ならばその指示を実行する.

作成したコードは別紙に記載した.

## 4.　結果

演算子が一つの場合,問題なく演算することができた． 演算子を複数含む式の計算時には,演算子を入力するたびに表示ボタンを押さないと指示通りに計算しない.

演算子を入力したときすぐに演算しているため,一度入力すると変更できない.

複数のスイッチを同時に押すと番号が小さいほうが優先して読み込まれるが,電卓として利用するには全く問題がない.

## 5.　展望

今の状態では複雑な計算を行うとき,余計な工程が多く不便である． 式をすべて入力して,最後に表示を押すだけで計算できるように改良したい.

現状本体とキーパッドがワイヤで繋がっているだけであり,持ち運びに不便である． 持ち運びやすくするためにケースを制作したい.

## 6.　参考文献

1） しなぷす, 2015, Arduino を使った電卓の製作, https://synapse.kyoto/hard/calculator/page001.html,(2023/12/3)

2） 斎藤公輔, 2021, 君は逆ポーランド電卓を知っているか？～そして自作へ～, (2023/12/1)
https://dailyportalz.jp/kiji/RPN-calculator

3） syumokuzame, 2021, 構造体でスタックを簡単に実装してみた,
https://same.blog/2021/01/10/c%e8%a8%80%e8%aa%9e%e6%a7%8b%e9%80%a0%e4%bd%93%e3%81%a7%e3%82%b9%e3%82%bf%e3%83%83%e3%82%af%e3%82%92%e7%b0%a1%e5%8d%98%e3%81%ab%e5%ae%9f%e8%a3%85%e3%81%97%e3%81%a6%e3%81%bf%e3%81%9f%e3%82%b3/,(2023/12/4)

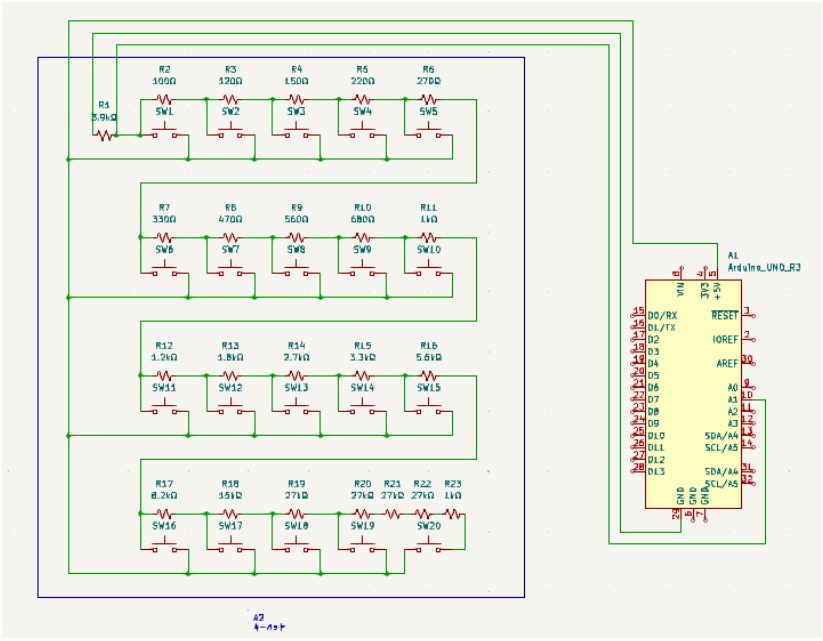


図 1. キーパッド回路図

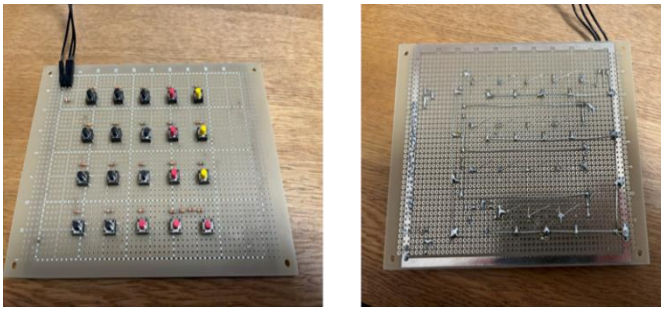

図 2. 制作したキーパッド

| | | | | |
|---|---|---|---|---|
| 7 | 8 | 9 | / | すべてクリア |
| 4 | 5 | 6 | * | 入力をクリア |
| 1 | 2 | 3 | - | 一文字消す |
| 0 | 小数点 | ENTER | + | 計算結果を表示 |

図 3. スイッチに対応する数字,記号,指示

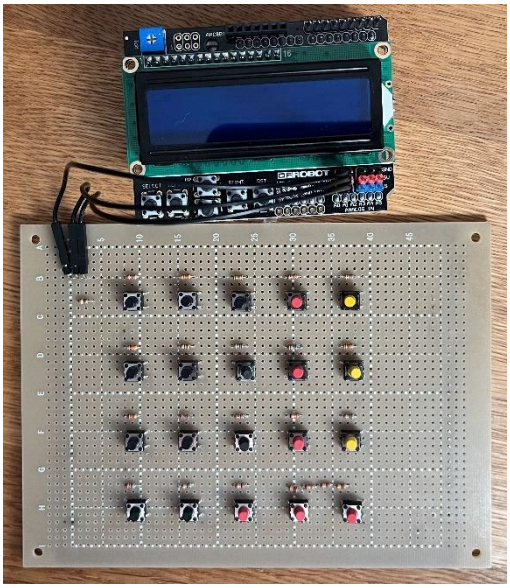

図 4. 制作した電卓

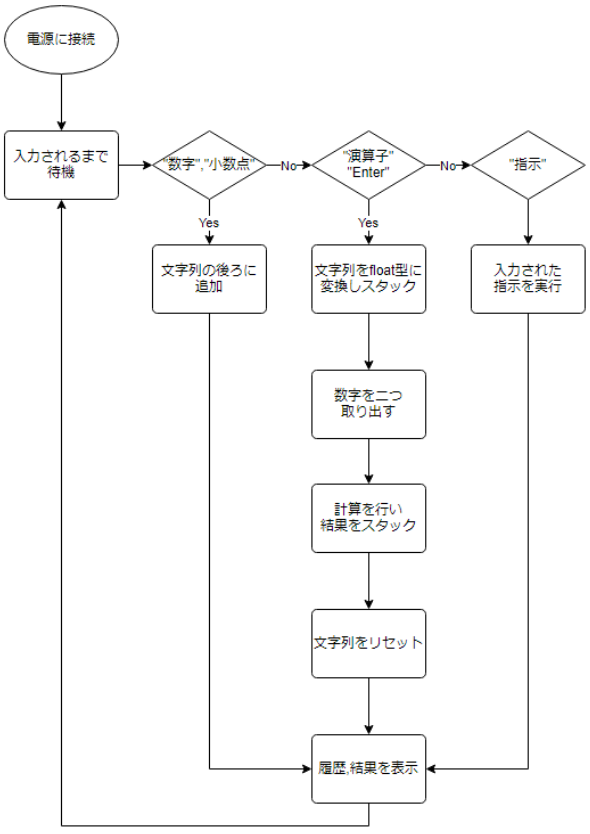


図 5. フローチャート

```
//calc_2

#include <LiquidCrystal.h> // 液晶用のライブラリ
#include <ResKeypad.h> // キーパッド用のライブラリ

#define STACK_SIZE  100
typedef float  stackdata;
stackdata stack_data[STACK_SIZE];
int stack_num;

const int KeyNum=20; // キーの数
const char PROGMEM
KeyChar[KeyNum]={'7','8','9','/','c','4','5','6','*','e','1','2','3','-','b','0'
,'.',' ','+','s'}; // キー割り当て
ResKeypad keypad(A1,KeyNum,RESKEYPAD_4X5,KeyChar); //入力用のライブラリ
LiquidCrystal lcd(8, 9, 4, 5, 6, 7); // 液晶への出力

const int columns=16; // 表示最大桁数
const int digits=6; // 入力最大桁数
const char ErrorMessage[]="ERROR"; // エラー時の表示
float MaxNum; // 扱える数の上限
float NumF;//float型に変換した被演算子
float result=0.0;//途中の演算結果
const float ErrorFloat=1.0E30; // エラーを表す数
int count_history=0;
int operator_n=0;//演算子の個数
int operand_n=0;//被演算子の個数

String NumStr="waitingforkeyin"; // 下の行に表示される文字列('='ボタンを押した直後なら計
算結果を表わす文字列、それ以外なら入力中の数字を表わす文字列)
String history=""; // 上の行に表示される入力履歴の文字列

int push(stackdata push_data)
{
    if (stack_num < STACK_SIZE) {
        stack_data[stack_num] = push_data;
        stack_num ++;
        return 1;
    } else {
        return 0;
    }
}//push

int pop(stackdata *pop_data)
{
    if (stack_num > 0) {
        stack_num --;
        *pop_data = stack_data[stack_num];
        return 1;
    } else {
        return 0;
    }
}//pop
```

```
void DisplayStrings(String hist, String num,int DelayTime=0)
{
  lcd.clear(); // 画面を消去
  // DelayTimeが0でなければ、指定された時間だけ処理を止める
  if(DelayTime!=0) {
    delay(DelayTime);
  } //if
  lcd.setCursor(columns-hist.length(),0); // 右詰めで表示するためにカーソル移動
  lcd.print(hist); // 入力履歴を表示
  lcd.setCursor(columns-num.length(),1); // 右詰めで表示するためにカーソル移動
  lcd.print(num); // 現在入力中の数を表示
} // DisplayStrings


float StF(String s)//文字型をfloat型へ
{
  int index; // 現在処理中の文字のインデックス
  boolean negative; // sが'-'で始まっているかどうかのフラグ
  float result; // 変換結果の数
  // 文字列が'-'で始まっていればnegativeをtrueに、そうでなければfalseにする
  if(s.length()>0 && s[0]=='-') {
    negative=true;
    index=1;
  } else {
    negative=false;
    index=0;
  } // if
  // 空の文字列または'-'一文字の文字列ならばエラーを返す
  if(s.length()<=index) { // if(s=="" || s=="-") {
    return ErrorFloat;
  } // if
  result=0.0; // 変換結果を初期化
  // 整数部(小数点の左側)を処理
  while(index<s.length() && s[index]!='.') {
     // 数字でなかったらエラーを返す
    if(s[index]<'0' || s[index]>'9') {
      return ErrorFloat;
    } // if
    result=result*10.0+(s[index++]-'0'); // 変換結果に一桁追加
  } // while
  if(index<s.length()) { // 小数部(小数点の右側)があった
    float f=0.0; // 小数部
    for(int i=s.length()-1; i>index; i--) { // 小数部を右側から処理(左側から処理をすると丸め誤差が増える)
      // 数字でなかったらエラーを返す
      if(s[i]<'0' || s[i]>'9') {
        return ErrorFloat;
      } // if
      f=(f+(s[i]-'0'))*0.1; // 一桁追加
    } // for i
    result+=f; // 変換結果に小数部の変換結果を加える
  }
  // 文字列が'-'で始まっていれば、変換結果に負号を付ける
  if(negative) {
```

```
    result=-result;
  } // if
  return result;
}//StF

String FtS(float f)//float型を文字型へ
{
  String result=""; // 変換後の文字列
  boolean negative; // fが負の数かどうかのフラグ
  long li; // fの整数部
  int cnt; // fの整数部の桁数(必ず1以上)
  float ff; // 作業用のfloat型変数
  int index; // 文字列処理の作業用変数
  // fが負かどうかを判定してnegativeを設定し、fの絶対値を取る
  if(f<0.0) {
    f=-f;
    negative=true;
  } else {
    negative=false;
  } // if
   // fが扱える桁数を超えていたら"?"を返す
  if(f>MaxNum) {
    return "?";
  } // if
  // fの整数部が何桁か調べる
  ff=10.0;
  cnt=1;
  while(f>=ff) {
    ff*=10.0;
    cnt++;
  } // while
  // 10進数から2進数に変換した時の丸め誤差を目立たない様にするために、小さな数をfに足す
  ff*=0.2;
  for(int i=0; i<digits; i++) {
    ff/=10.0;
  } // for i;
  f+=ff;
   // fが扱える桁数を超えていたら"?"を返す
  if(f>MaxNum) {
    return "?";
  } // if
  li=long(f); // fの整数部を取得する
  f-=li; // fの小数部を取得する
  // 整数部を文字列に変換する(整数部が0の場合はresult==""となる)
  while(li>0) {
    result=char('0'+(li%10))+result;
    li=li/10;
  } // while
  // 整数部が0だった場合はresultに"0"を代入する
  if(result=="") {
    result="0";
  } // if
  cnt=digits-result.length(); // 小数部の有効な桁数を取得
  result+="."; // resultに小数点を付加
```

```
  // 小数部を変換する
  for(int i=0; i<cnt; i++) {
    f=f*10;
    result+=String(int(f));
    f-=int(f);
  } // for i
  // 小数点以下の無駄な0や無駄な小数点を取り除く(例:123.400→123.4、123.00→123)
  index=result.length()-1; // resultの一番右側の文字の添え字の取得
  while(result[index]=='0') index--; // resultの右端から走査し、'0'がどこまで続くか調べる
  if(result[index]=='.') index--; // resultの右端が小数点なら、indexをデクリメント
  result=result.substring(0,index+1); // 末尾の不要な文字を取り除く
  // もしnegative==trueなら、resultの前に'-'を追加する
  if(negative) {
    result="-"+result;
  } // if
  return result;
}//FtS

void setup() {
  lcd.begin(columns,2);
  DisplayStrings(history,NumStr); // 表示の初期化
  // 扱える最大の数を計算する
  MaxNum=1.0;
  for(int i=0; i<digits; i++) {
    MaxNum*=10.0;
  } // for i
  MaxNum-=0.6;
} // setup

void loop() {
  char c; //最後に入力した文字
  int additional; // NumStrに含まれる'-'と'.'の数
  int DelayTime=0; // 再描画の遅延時間
  char last_history;
  float x1=0.0,x2=0.0;//演算時に取り出した被演算子2つ

   c=keypad.WaitForChar(); // 入力されたらcに代入
   switch(c) {
   case 'c': // 'C'のボタンが押された
    history=""; // 入力履歴を消す
    NumStr="waitingforkeyin"; // NumStrを消去
    operand_n=0;
    operator_n=0;
    break;

   case 'b': // ←
   if(count_history!=0){
    last_history=history[count_history - 1];//historyの最後の文字を取得
   }
   if(NumStr=="waitingforkeyin"||NumStr==""){
      if(last_history==" "){
        operator_n=operator_n-1;
      }//if
      history=history.substring(0,history.length()-1);
```

```
      break;
    }//if
    if(NumStr.length()>1) {//2文字以上
      NumStr=NumStr.substring(0,NumStr.length()-1); // 最後の文字を消す
    } else if(NumStr.length()==1) { // 1文字だけ
     if(history==""){
       NumStr="waitingforkeyin"; // NumStrを消去
     }else{
       NumStr="";
     }//if
    } else if(NumStr.length()==0&&history!="") {
      if(last_history==","){
        operator_n=operator_n-1;
      }//if

if(last_history=="/"||last_history=="*"||last_history=="-"||last_history=="+"){
      break;
      }
      history=history.substring(0,history.length()-1);
    }else{
     NumStr="waitingforkeyin"; // NumStrを消去
    }//if
    break;

  case ' ':
  operator_n=operator_n+1;
  if(NumStr=="waitingforkeyin"){
    break;
  }else{
  history=history+NumStr+c;//履歴に被演算子NumStrと演算子cを追加
  NumF=StF(NumStr);//被演算子をfloat型へ変換
  push(NumF);//被演算子をスタック
  NumStr="";//被演算子を消去
  break;
  }//if

  case '+'://演算子+を入力
  case '-'://演算子-を入力
  case '/'://演算子/を入力
  case '*'://演算子*を入力
  if(operator_n==0){//被演算子が二つ以上ない場合
    break;
  }else{
    history=history+NumStr+c;//履歴に被演算子NumStrと演算子cを追加
    NumF=StF(NumStr);//被演算子をfloat型へ変換
    push(NumF);//被演算子をスタック
    NumStr="";//被演算子を消去
    pop(&x2);
    pop(&x1);//被演算子の取り出し
  break;
  }//if

  case '0':
  case '1':
```

```
  case '2':
  case '3':
  case '4':
  case '5':
  case '6':
  case '7':
  case '8':
  case '9':
  case '.'://数字,小数点を入力
  if(NumStr=="waitingforkeyin"){
    NumStr="";
  }
   NumStr=NumStr+c;
   break;

  case 'e':
  NumStr="waitingforkeyin";
  break;

  case 's'://演算を行う
  operator_n=1;
  history=FtS(result)+" ";
  NumStr="";
  result=0.0;
  break;
}//swich

switch(c) {//演算子が入力されたと時,演算を行う
  case '+'://演算子+を入力
    result=result+(x1+x2);
    push(result);
    history=history+" ";
  break;
  case '-'://演算子-を入力
    result=result+(x1-x2);
    push(result);
    history=history+" ";
  break;
  case '/'://演算子/を入力
    result=result+(x1/x2);
    push(result);
    history=history+" ";
  break;
  case '*'://演算子*を入力
    result=result+(x1*x2);
    push(result);
    history=history+" ";
  break;
  default:
  break;
}
   DisplayStrings(history,NumStr,DelayTime);
   int count_history=history.length();
}//繰り返す
```