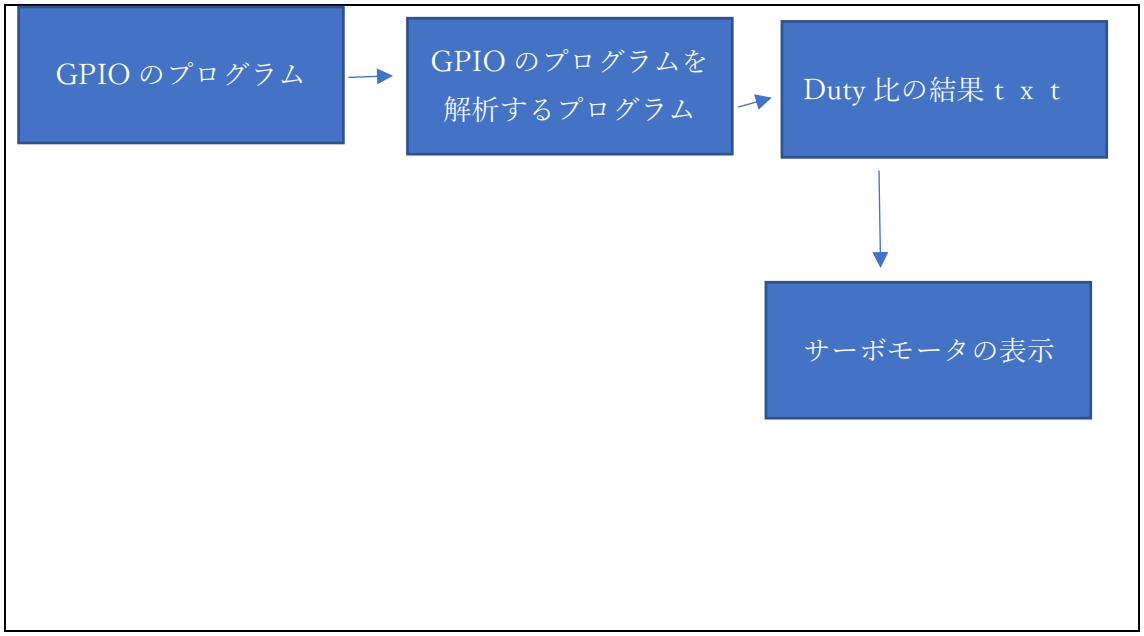
2021 年度組込み技術特論 PWM 班の提案

作成者 1CJNM001 井上駿佑

PWM 班ではサーボモータの可視化と仮想 gpio の作成を目標にしている。



そして 2021 年 5 月 31 日現在サーボモータの表示を processing で行ってきた。
今の課題は gpio プログラムをどうやって解析していくかになる。そして私は GPIO のファイルのトランスレータを lex 言語と yacc 言語で duty 比の結果の txt にするシステムを作りたいと考える。

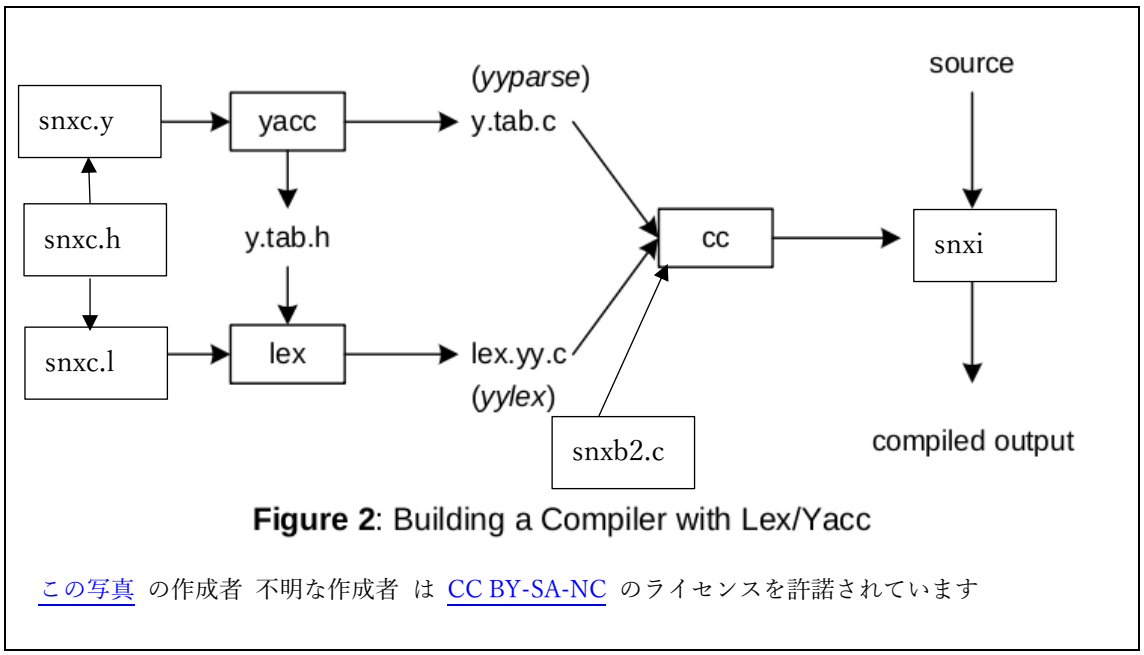


Figure 2: Building a Compiler with Lex/Yacc

その前に yacc と lex 言語について lex および yacc プログラム情報 - IBM Documentation を見て実際のプログラムを見て動かした。
pigpio library (abyz.me.uk)の pigpio の
Yacc と lex を動かす前に以下のコマンドでインストールできる。
$sudo apt-get install bison
$sudo apt-get install flex
今回システムでは if 文 for 文 while 文 print()と pigpio を使えるようなものにする。

その上以下の Makefile を作成する。

```
CC = gcc
YACC = bison -y
LEX = flex

CFLAGS = -Wall -g

CLEANS = snxb2.o y.tab.o lex.yy.o y.tab.c lex.yy.c y.tab.h
OBJI = snxb2.o y.tab.o lex.yy.o
SRCS = snxb2.c snxc.y snxc.l snxc.h

all: snxi

snxi: $(OBJI)
	$(CC) $(CFLAGS) $(OBJI) -o snxi

snxb2.o: snxb2.c snxc.h y.tab.c
	$(CC) $(CFLAGS) -c snxb2.c

y.tab.o: y.tab.c snxc.y
	$(CC) $(CFLAGS) -c y.tab.c

lex.yy.o: lex.yy.c y.tab.h
	$(CC) $(CFLAGS) -c lex.yy.c

y.tab.c: snxc.y snxc.h
	$(YACC) -d snxc.y
```

```
lex.yy.c: snxc.l snxc.h
	$(LEX) snxc.l

clean:
	-rm $(CLEANS)
```

Makefile

また snxc.l(lex ファイル)と snxc.y(yacc ファイル)と snxb2 と snxc.h を使う。

```
%{
#include <stdlib.h>
#include "snxc.h"
#include "y.tab.h"
void yyerror(char *s);
int Line = 1;
%}

%%

[a-z]		{
			yylval.Symbol = *yytext - 'a';
			return VARIABLE;
		}

[0-9]+		{
			yylval.IntVal = atoi(yytext);
			return INTEGER;
		}

[-()<>=+*,/;{}.¥[¥]] {
			return *yytext;
		}

"++"		return PP;
"--"		return MM;
">="		return GE;
"<="		return LE;
```

```
"=="                    return EQ;
"!="                    return NE;
"for"                   return FOR;
"while"                 return WHILE;
"if"                    return IF;
"else"                  return ELSE;
"return"                return RETURN;
"print"                 return PRINT;
"mem"                     return MEM;
"io"                    return IO;
"arg"                   return ARG;
"lo"                    return LO;
"void"                  return DEF;
"int"                   return DEF;
"foo"                   return FUNCNAME;
"halt"                  return HALT;
"pi.set_PWM_dutycycle" return FUNCNAMEPWM;
"time.sleep"        return SLEEP;
[ ¥t]+              ;           /* ignore whitespace */
[¥n]                    Line++;

.                       yyerror("Unknown character");
%%
int yywrap(void) {
      return 1;
}
```

snxc.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "snxc.h"
#define YYDEBUG 1
#define STACKTOP 127
```

```
/* prototypes */
Pnode *opr(int oper, int nops, ...);
Pnode *id(int i);
Pnode *con(int value);
void freeNode(Pnode *p);
void sinit(int val);
int ex(Pnode *p, int reg, int pres);
extern int Line;
extern char * yytext;

void yyerror(char *s);

int sym[65536];                         /* symbol table + memory*/
%}

%union {
      int IntVal;                       /* integer value */
      char Symbol;                      /* symbol table index */
      Pnode *Node;                      /* node pointer */
};
%token <IntVal> INTEGER
%token <Symbol> VARIABLE
%token WHILE FOR IF PRINT MRD MWT OUT IN MEM IO LO FUNCNAME FDEF
RETURN FUNC FUNCNAMEPWM SLEEP
%token DEF FDEFA HALT
%token <Node> ARG ARGV
%nonassoc IFX
%nonassoc ELSE

%left GE LE EQ NE '>' '<'
%left '+' '-'
%left '*' '/'
%right UMINUS
%left MM PP
%type <Node> astmt stmt expr stmt_list
```

```
%%

program:
        init function                   { exit(0); }
        ;

init:
        /* NULL */                      { sinit(STACKTOP);}
        ;

function:
          function stmt                 { ex($2,1,0); freeNode($2); }
        | /* NULL */
        ;

stmt:
          ';'                           { $$ = opr(';', 2, NULL, NULL); }
        | astmt ';'                     { $$ = $1; }
        | DEF FUNCNAME '(' ARG ')' stmt          { $$ = opr(FDEFA,1, $6); }
        | DEF FUNCNAME '(' DEF ARG ')' stmt          { $$ = opr(FDEFA,1, $7); }
        | DEF FUNCNAME '(' ')' stmt             { $$ = opr(FDEF, 1, $5); }
        | PRINT expr ';'                { $$ = opr(PRINT, 1, $2); }
        | FOR '(' astmt ';' expr ';' astmt ')' stmt
                { $$ = opr(FOR, 4, $3, $5, $7, $9); }
        | WHILE '(' expr ')' stmt       { $$ = opr(WHILE, 2, $3, $5); }
        | RETURN '(' expr ')' ';'       { $$ = opr(RETURN, 1, $3); }
        | RETURN expr ';'                   { $$ = opr(RETURN, 1, $2); }
        | RETURN ';'                        { $$ = opr(RETURN, 0); }
        | IF '(' expr ')' stmt %prec IFX { $$ = opr(IF, 2, $3, $5); }
        | IF '(' expr ')' stmt ELSE stmt { $$ = opr(IF, 3, $3, $5, $7); }
        | '{' stmt_list '}'             { $$ = $2; }
        | HALT ';'                          { $$ = opr(HALT, 0); }
        | FUNCNAMEPWM '('expr','expr')'   { $$ = opr(FUNCNAMEPWM,2,$3,$5);}
        | SLEEP '('expr')'              { $$ = opr(SLEEP,1,$3);}
        ;
```

```
astmt:
          expr                                  { $$ = $1; }
        | MEM '[' expr ']' '=' expr     { $$ = opr(MWT, 2, $3, $6); }
        | VARIABLE '=' expr                 { $$ = opr('=', 2, id($1), $3); }
        | VARIABLE   PP     { $$ = opr('=',2,id($1),opr('+',2,id($1),con(1))); }
        | VARIABLE   MM     { $$ = opr('=',2,id($1),opr('-',2,id($1),con(1))); }
        | VARIABLE '[' expr ']' '=' expr     { $$ = opr(MWT, 3, $3, $6, id($1)); }
        ;

stmt_list:
          stmt                          { $$ = $1; }
        | stmt_list stmt            { $$ = opr(';', 2, $1, $2); }
        ;

expr:
          INTEGER                       { $$ = con($1); }
        | VARIABLE                      { $$ = id($1); }
        | ARG                           { $$ = opr(ARGV, 0); }
        | '-' expr %prec UMINUS { $$ = opr(UMINUS, 1, $2); }
        | MEM '[' expr ']'          { $$ = opr(MRD, 1, $3); }
        | VARIABLE '[' expr ']'          { $$ = opr(MRD, 2, $3, id($1)); }
        | FUNCNAME '(' expr ')' { $$ = opr(FUNC, 1, $3); }
        | expr '+' expr             { $$ = opr('+', 2, $1, $3); }
        | expr '-' expr             { $$ = opr('-', 2, $1, $3); }
        | expr '<' expr             { $$ = opr('<', 2, $1, $3); }
        | expr '>' expr             { $$ = opr('>', 2, $1, $3); }
        | expr GE expr              { $$ = opr(GE, 2, $1, $3); }
        | expr LE expr              { $$ = opr(LE, 2, $1, $3); }
        | expr NE expr              { $$ = opr(NE, 2, $1, $3); }
        | expr EQ expr              { $$ = opr(EQ, 2, $1, $3); }
        | '(' expr ')'              { $$ = $2; }
        ;

%%
```

```
Pnode *con(int value) {
	Pnode *p;

	/* allocate node */
	if ((p = malloc(sizeof(Const))) == NULL)
		yyerror("out of memory");

	/* copy information */
	p->type = typeCon;
	p->con.value = value;

	return p;
}

Pnode *id(int i) {
	Pnode *p;

	/* allocate node */
	if ((p = malloc(sizeof(Ident))) == NULL)
		yyerror("out of memory");

	/* copy information */
	p->type = typeId;
	p->id.i = i;

	return p;
}

Pnode *opr(int oper, int nops, ...) {
	va_list ap;
	Pnode *p;
	size_t size;
	int i;

	/* allocate node */
	size = sizeof(Operator) + (nops - 1) * sizeof(Pnode*);
```

```
    if ((p = malloc(size)) == NULL)
        yyerror("out of memory");

    /* copy information */
    p->type = typeOpr;
    p->opr.oper = oper;
    p->opr.nops = nops;
    va_start(ap, nops);
    for (i = 0; i < nops; i++)
        p->opr.op[i] = va_arg(ap, Pnode*);
    va_end(ap);
    return p;
}

void freeNode(Pnode *p) {
    int i;

    if (!p) return;
    if (p->type == typeOpr) {
        for (i = 0; i < p->opr.nops; i++)
            freeNode(p->opr.op[i]);
    }
    free (p);
}

void yyerror(char *s) {
    fprintf(stdout, "%s(%s) at %d¥n", s, yytext, Line);
}

extern int yydebug;
int main(int argc, char *argv[]) {
    if(argc==2 && !strcmp(argv[1],"-d")) yydebug=1;
    yyparse();
    return 0;
}
```

snxc.y

```
#include <stdio.h>
#include<math.h>
#include <stdlib.h>
#include <setjmp.h>
#include "snxc.h"
#include "y.tab.h"
#define FUNMAX    127

static Pnode *foo;
static int sp=FUNMAX;
static jmp_buf funbuf[FUNMAX];
static int val, jv;

int sinit(int init) {return 0;}
int ex(Pnode *p) {
      if (!p) return 0;
      switch(p->type) {
      case typeCon:           return p->con.value;
      case typeId:            return sym[p->id.i + 1];
      case typeOpr:
            switch(p->opr.oper) {

            case FDEF:
            case FDEFA:       foo=p->opr.op[0];
                              p->opr.op[0] = NULL;
                              return 0;
            case ARGV:
                              return sym[sp+1];
            case FUNC:        if(p->opr.nops>0) {
                              val = ex(p->opr.op[0]);
                              sp -=2;
                              sym[sp+1] = val;
                              jv = setjmp(funbuf[sp]);
                              if(jv == 0)
                                    val=ex(foo);
```

```
                    sp +=2;
                    return val;
                    } else
                    {
                    jv = setjmp(funbuf[sp]);
                    if(jv == 0)
                        val = ex(foo);
                    return val;
                    }
case RETURN:        if(p->opr.nops>0) {
                    val = ex(p->opr.op[0]);
                    } else
                    {
                    val = 0;
                    }
                    longjmp(funbuf[sp], -1);
case FOR:           for(ex(p->opr.op[0]);
                        ex(p->opr.op[1]);
                        ex(p->opr.op[2])) ex(p->opr.op[3]); return 0;
case WHILE:         while(ex(p->opr.op[0])) ex(p->opr.op[1]); return 0;
case HALT:          exit(0);
case IF:            if (ex(p->opr.op[0]))
                        ex(p->opr.op[1]);
                    else if (p->opr.nops > 2)
                        ex(p->opr.op[2]);
                    return 0;
case PRINT:         printf("%d¥n", ex(p->opr.op[0])); return 0;
case ';':           ex(p->opr.op[0]); return ex(p->opr.op[1]);
case '=':           return sym[p->opr.op[0]->id.i + 1] = ex(p->opr.op[1]);
case OUT:           //printf("Port[%d] <- %d¥n",
                        //sym[ex(p->opr.op[0])] , ex(p->opr.op[1]));
                    return 0;
case MWT:           return sym[ex(p->opr.op[0])
                        +((p->opr.nops>2)?p->opr.op[2]->id.i+1:0)] =
                        ex(p->opr.op[1]);
case MRD:           return sym[ex(p->opr.op[0])+
```

```
                                        ((p->opr.nops>1)?p->opr.op[1]->id.i+1:0)];
        case IN:              //printf("Enter value for Port[%d]: ", ex(p->opr.op[0]));
                              return getchar();
        case UMINUS:        return -ex(p->opr.op[0]);
        case '+':           return ex(p->opr.op[0]) + ex(p->opr.op[1]);
        case '-':           return ex(p->opr.op[0]) - ex(p->opr.op[1]);
        case '*':           return ex(p->opr.op[0]) * ex(p->opr.op[1]);
        case '/':           return ex(p->opr.op[0]) / ex(p->opr.op[1]);
        case '<':           return ex(p->opr.op[0]) < ex(p->opr.op[1]);
        case '>':           return ex(p->opr.op[0]) > ex(p->opr.op[1]);
        case GE:            return ex(p->opr.op[0]) >= ex(p->opr.op[1]);
        case LE:            return ex(p->opr.op[0]) <= ex(p->opr.op[1]);
        case NE:            return ex(p->opr.op[0]) != ex(p->opr.op[1]);
        case EQ:            return ex(p->opr.op[0]) == ex(p->opr.op[1]);
        case FUNCNAMEPWM:
                              printf("%d,",ex(p->opr.op[1]));
                              return 0;
        case SLEEP:
                              printf("%d¥n",ex(p->opr.op[0]));
                              return 0;
        }
    }
 return 0;
}
```

snxb2.c

```
typedef enum { typeCon, typeId, typeOpr } nodeType;

/* constants */
typedef struct {
    nodeType type;                  /* type of node */
    int value;                      /* value of constant */
} Const;

/* identifiers */
typedef struct {
```

```
    nodeType type;                  /* type of node */
    int i;                          /* subscript to ident array */
} Ident;

/* operators */
typedef struct {
    nodeType type;                  /* type of node */
    int oper;                       /* operator */
    int nops;                       /* number of operands */
    union PnodeTag *op[1];      /* operands (expandable) */
} Operator;

typedef union PnodeTag {
    nodeType type;                  /* type of node */
    Const con;              /* constants */
    Ident id;               /* identifiers */
    Operator opr;               /* operators */
} Pnode;

extern int sym[65536];
```

snxc.h

これらを使って以下のプログラムを実行した。

```
a=30;
pi.set_PWM_dutycycle(18,a);
time.sleep(44);
pi.set_PWM_dutycycle(18,a+10);
time.sleep(44);
```

duty.c

このプログラムは c 言語と pigpio の関数を解析できるようにした。
そのため";"を処理の終端につける。
そして

```
$ make clean
$ ls
duty.c  Makefile  snxb2.c  snxc.h  snxc.l  snxc.y
$make all
```

$ls

duty.c lex.yy.c lex.yy.o Makefile snxb2.c snxb2.o snxc.h snxc.l snxc.y snxi y.tab.c y.tab.h y.tab.o

$./snxi<duty.c

30,44

40,44

で動かした。

実行ファイルは. /snxi となる。

duty.c は import pigpio と pi.set_mode(18, pigpio.OUTPUT)

pi = pigio.pi()を行ったとして構成している。

$./snxi<duty.c＞duty.txt

$cat duty.txt

30,44

40,44

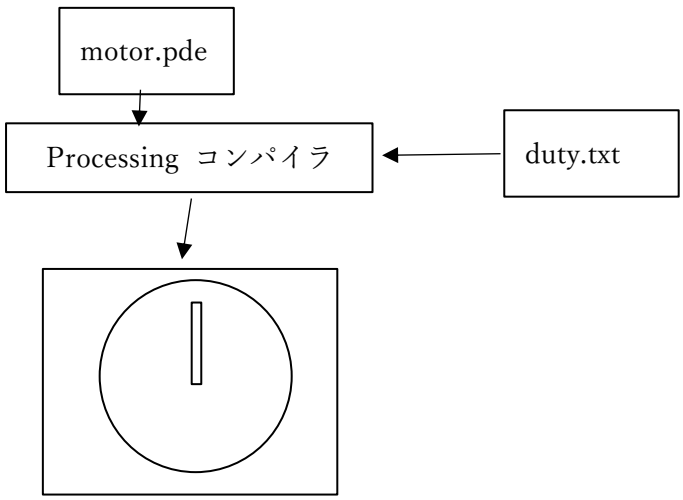
```
int bit;
int d=0;
int motor_pwm;
float angle=0;
int mode=0;
int count=0;
int yoko=0;
int sleep=0;
int wait_motor=0;
int gyo;
void setup(){
      size(400,300);
      background(255);
}
void draw(){
   delay(1);
   background(255);

   Table tbl =loadTable("duty.csv","csv");
   gyo=tbl.getRowCount();
   d=tbl.getInt(count,0);
   sleep=tbl.getInt(count,1);
   if(d<0){
      motor_pwm=-d*2-50;
   }else{
      motor_pwm=d*2-50;
   }
   println(d);
   println(wait_motor);
   println(sleep);
   wait_motor++;
   if((wait_motor/100)>=sleep){
      if(count<gyo-1){
```

```
        count++;
      }
      wait_motor=0;
    }
    if((bit&(1<<0))>0&&d<50){
      d+=1;
      println(d);
    }
    if((bit&(1<<1))>0&&d>-50){
      d-=1;
      println(d);
    }
    for(int i=-1;i<=3;i++){
      line(30+100*i,270,30+100*i,240);
      line((30+100*i),240,(80+100*i)+motor_pwm,240);
      line((80+100*i)+motor_pwm,270,(80+100*i)+motor_pwm,240);
      line((80+100*i)+motor_pwm,270,(130+100*i),270);
    }
    translate(200,100);
    rotate(radians(angle));
    ellipse(0,0,90,90);
    rect(-5,-5,40,10);
    ellipse(0,0,5,5);
    if(mode==0){
       angle=170+(50+d)*2;
    }else if(mode==1){
      angle+=(d)/10;
    }



}
    void keyPressed(){
      if(keyCode==RIGHT){
        bit|=(1<<0);
      }
```

```
    if(keyCode==LEFT){
      bit|=(1<<1);
    }
    if(key=='d'){
      mode=1;
    }
    if(key=='s'){
      mode=0;
    }
  }
    void keyReleased(){
      if(keyCode==RIGHT){
        bit&=~(1<<0);
      }
      if(keyCode==LEFT){
          bit&=~(1<<1);
      }
    }
```

motor.pde

を使って行えば可視化できる。



現状だとモーターを何秒間かどのくらいのスピードで回転させるかしかできていない。
これをもっと拡張すべきだと考える。