Final-term Project

學號：B11215024　　姓名：劉柏毅

The project is based on global variable with internal linkage and process the input of the barcode by function using global variable, the data is as follows:

```
typedef union data_s
{
    struct
    {
        uint8_t b1: 1;
        uint8_t b2: 1;
        uint8_t b3: 1;
        uint8_t b4: 1;
        uint8_t b5: 1;
    };
    char    character;
} data_t;
```

Figure 1

As Figure. 1, there is the data frame of the barcode program, with its data structure using 5-bit value, and decode to the corresponding character using 8 bits (1Byte). For filling the problem, token `DATA_SIZE` should be set to 32：

```
static data_t   data[DATA_SIZE] = {'\0'};
```

```
enum STATUS_FLAG
{
    STATUS_EMPTY_FLAG,
    STATUS_EXIT_FLAG,
    STATUS_BAD_FLAG,
    STATUS_BAD_C_FLAG,
    STATUS_BAD_K_FLAG,
};
```

The program provide 5 kind of state for the state flows of the process, which are standing for empty flag, escape flag, null flag, invalid C and invalid K verification code. There is the related variable (Initialize the state of the flag by empty flag):

```
static uint8_t  step, status;
static uint8_t  status_code = STATUS_EMPTY_FLAG;
```

And, there is the definition of the preprocessing token:

```
#define CODE_SIZE   150
#define DATA_SIZE   32
#define DATA_WIDTH  5
#define UPPER_WIDTH 150
#define LOWER_WIDTH 29
```

Respectively, standing for the upper limit of the barcode and the data, the width of the data and the upper and lower limit of the length of the barcode (The upper one is as same as upper limit of the barcode, it’s only for semantically convenience.

The process flows are divided into three categories, and using array PROCESS as the sequence of the process：

```
#define METHOD_STEP 13

void (*volatile const PROCESS[METHOD_STEP])(void);
```

The first category is for modify barcode, the main idea of the program.

```
void get_length     (void);

void get_code       (void);

void sort_code      (void);

void take_format    (void);

void calibrate_code (void);

void binary_code    (void);

void reverse_code   (void);

void get_data       (void);

void empty_buffer   (void);
```

The second category is for checking the barcodes and the data to determine whether they are the valid input.

```
void boundary_check (void);

void split_check    (void);

void C_check        (void);

void K_check        (void);
```

The third category is only for outputting the data.

```
void print_code     (void);
void print_code_buf (void);
void print_data     (void);
```

The architecture of the project use state register, also known as state flow, with other function to scan the barcode. As Figure. 2, the working sequence has been initialized:

```
void (*volatile const PROCESS[METHOD_STEP])(void) =
{
    get_length,    //Get m, n
    boundary_check,    //Check m
    get_code,      //Scan the barcode.
    sort_code,     //Sort the barcode(preprocessing).
    take_format,   //Get the format of the barcode.
    calibrate_code,    //Calibrate the barcode with format.
    binary_code,   //Transpile the barcode to binary.
    reverse_code,  //Probe the barcode whether it is reverse.
    split_check,   //Check the format of the split character..
    get_data,      //Get the data.
    C_check,       //Check the C verification code.
    K_check,       //Check the K verification code.
    print_data     //Output the result if all the thing is valid.
};
```

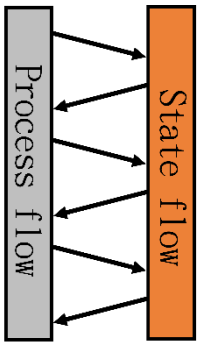

Figure 2

The definition of PROCESS is as follows: PROCESS is an array of constant and volatile pointer to function (void) returning nothing.

The main function consists of process flows and state flows:

```
int main(void)
{
    while(1)
    {
        status_code = STATUS_EMPTY_FLAG;
        for(step = 0; step < METHOD_STEP; ++step)
        {
            PROCESS[step]();
            status = check_status();

            if(status == 1)
            {
                return 0;
            }
            if(status == 2)
            {
                break;
            }
        }
        empty_buffer();
    }
}
```

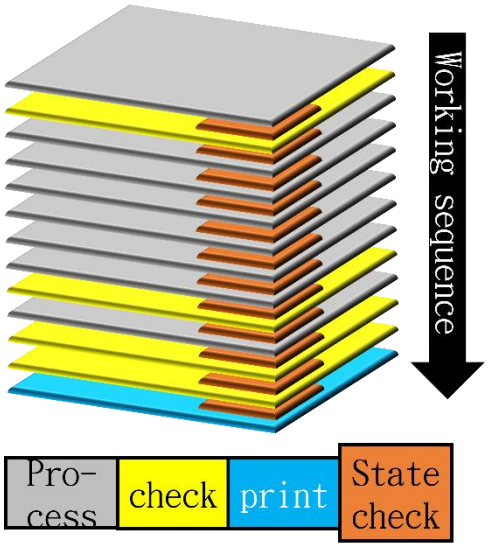

Figure 3

As Figure.3, there is always a state check for each process, jump from current process if it get bad flag and then output the result with diverse situation or cut down all the program:

```
static inline uint8_t check_status(void)
{
    switch(status_code)
    {
    case STATUS_EMPTY_FLAG:
        return 0;
    case STATUS_EXIT_FLAG:
        return 1;
    case STATUS_BAD_FLAG:
        printf("Case %d: bad code\n", count);
        return 2;
    case STATUS_BAD_C_FLAG:
        printf("Case %d: bad C\n", count);
        return 2;
    case STATUS_BAD_K_FLAG:
        printf("Case %d: bad K\n", count);
        return 2;
    default:
        return 0;
    }
}
```

Don’t do anything with empty flag (STATUS_EMPTY_FLAG), the reset process flows is initialized with empty flag; Return 1 to finish the program with exit flag (STATUS_EXIT_FLAG); Output the error messages with bad flags(STATUS_BAD_???), and return 2 to reset process flows.

The function used by other process flows is as following 3 category:

1. Function for printing:

```
void print_code     (void);
void print_code_buf (void);
void print_data     (void);
```

They are used for print data; print_code can print the scanned barcode when you want, it is for debugging as print_code_buf does so; print_data is used for outputting the result.

2. Function for checking:

```
void boundary_check (void);
void split_check    (void);
void C_check        (void);
void K_check        (void);
```

They are used for checking data; According input barcode, boundary_check checks whether its data the follows the barcode format; split_check checks whether all the split characters are narrow bar (note that the light bar has been excluded.); C_check and K_check check whether the verification code is valid.

3. Function for checking: the main algorithm the project use.

```
void take_format(void)
{
    for(each = 0; each < m - 1; ++each)
    {
        if((code_buf[each + 1] - code_buf[each]) >
        (bias_upper(code[each + 1]) - bias_lower(code[each + 1])))
            break;
    }

    min = ((code_buf[0] * 2 < code_buf[each + 1])?
              code_buf[0] * 2: code_buf[each + 1]);
    max = ( code_buf[each] * 2  > code_buf[m - 1])?
              code_buf[each] * 2:  code_buf[m - 1];

    wide_bar    = (min + max) / 2;
    narrow_bar  = (min + max) / 4;
}
```

Above-mention function is for finding the value of light-bar format calibrating the barcode. (The barcode has to be sorted firstly.) The for-statement will break when the next bar too greater than the current bar. (Namely, probing whether the difference between the current bar and next bar is greater than ±5% of the value.) Then, you can calculate the average value of the next bar and the twice of the minimum one to find the format of the narrow bar and you can do so again with the current bar and the maximum one to find the format of the wide bar.

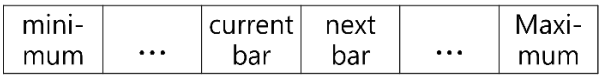

Figure 4

As Figure. 4, the for-statement break if the current bar is greater ±5% than next bar, so we make next bar (when for-statement break) as a new format, and because the next bar is seen as wide bar as well (which is as twice as light bar) we should compare it to the twice of the minimum one to get the format, so do current bar and maximum one.

(Note that it will get wrong and set the state to bad flag when there are three kind of the format of the barcode.)

```
void reverse_code(void)
{
    if(code[0] != code[m - 5] || code[1] != code[m - 4] ||
       code[2] != code[m - 3] || code[3] != code[m - 2] ||
       code[4] != code[m - 1]) {
        status_code = STATUS_BAD_FLAG;
        return;
    }
    if(!code[0] && !code[1] && code[2] && code[3] && !code[4])
        return;
    if(!code[0] && code[1] && code[2] && !code[3] && !code[4])
        for(i = 0, j = m-1; i < j; ++i, --j)
            SWAP(code[i], code[j]);
    return;
    status_code = STATUS_BAD_FLAG;
}
```

The function check whether START and STOP are identical, Returning and setting flag to bad code when they aren't the same. Otherwise, check the format of the START and STOP and determine whether they should be reverse. Finally, setting bad code when all the above-mention checking don't match, it's stand for another kind of invalid input format.

```
void get_data(void)
{
    for(i = 1 + DATA_WIDTH, j = 0; i < m - DATA_WIDTH;
        i += DATA_WIDTH + 1, ++j)
        code_to_data(j, code + i);
    for(each = 0; each < n; ++each)
        if(!data_decode(each)) {
            status_code = STATUS_BAD_FLAG; return;
        }
    n -= 2;
}
```

The function gets the weight of the corresponding binary code to takes the decoded message. Return when the binary code cannot be decoded. Furthermore, the n -= 2; means that the C and K verification code should be excluded because all we want is the only message not including these two.

About this project:

There are so many kinds of semantics on the format to calibrate barcode, so I really get some confused by them. But there is my first time for designing a software architecture, I hope the object-oriented programming could support me to modify this project to make it much useability and readability when I had learned the skill on the courses “object-oriented programming” and “software engineering.”