05 STEP

関数

このSTEPでは、関数について学習しましょう。
関数を使ってSTEP04のプログラムを改良していきます。

5.1 関数

関数とは、入力したデータに対して結果を出力するものです。例えば、一次関数 y=2x+3 に x の値を入力すると y の値が決まりましたね。C 言語での関数も同じことです。具体的にどのように使うか見ていきましょう。

5.2 関数化

まずは、図 5-1 をご覧ください。
図 5-1 はプログラム 4-3 の永久ループを抜粋したものです。

```
// 永久ループ
while (1)
{
    P4.DR.BYTE = 0x80; // 右上の LED を点灯

    for (j = 0; j < 500; j++) // 待ち時間 (1[m 秒 ] × 500)
    {
        for (i = 0; i < 1560; i++) // 待ち時間 (1[m 秒 ])
            k++;
    }

    P4.DR.BYTE = 0x00; // 全消灯

    for (j = 0; j < 500; j++) // 待ち時間 (1[m 秒 ] × 500)
    {
        for (i = 0; i < 1560; i++) // 待ち時間 (1[m 秒 ])
            k++;
    }
}
```

図 5-1　関数化するプログラム

STEP 05 関数

図 5-1 の白抜き枠で囲まれた部分はプログラムの内容が全く同じ「待ち時間処理」です。この待ち時間処理のように、同じ内容を何回も記述するのはあまり望ましいことではありません。なぜなら、同じ処理を繰り返し書くことによって、それだけ誤って記述してしまう可能性が増えるからです。また、待ち時間を調整するにも修正箇所が多く手間が掛かってしまいます。
そこで、この待ち時間処理を関数にしてみましょう。

【 関数の定義 】

C 言語での関数とは、分かりやすく言えば「処理をまとめて名前をつけたもの」ですが、本来は「入力したものに対して処理を行い、結果を出力するもの」です。関数に入力するものを引数、出力される結果を戻り値と言います。

関数の記述は、変数と同じように、関数を定義するところから始まります。

```
戻り値のデータ型　関数名　（引数のデータ型　引数）
{
    処理
}
```

「戻り値のデータ型」、「関数名」、「引数のデータ型」、「引数」には以下のような記述をします。
・戻り値のデータ型は、関数から出力するデータの型を書きます。
・関数名は、変数名と同様の規則にのっとって付けます。
・引数のデータ型は、関数に入力するデータの型を書きます。
・処理には、実行する処理を書きます。

※ 関数は main 関数より前に記述します。
　また、関数内で宣言した変数は、その関数の中でしか使えません。

```
例 1）void waitMs(int ms)
```

例 1 で定義されている関数は、戻り値のデータ型「void」、関数名「waitMs」、引数のデータ型「int」、引数「ms」です。void は、返す戻り値がない場合に定義するデータ型です。

```
例 2）main 関数
    int main(void)
```

今までのプログラムで組んできた main も関数です。
C 言語では、プログラムを実行すると main 関数から処理していくことが決まっているので、main 関数は少し特殊な関数と言えます。
さて、main 関数ですが、この関数は return を返しています。return は戻り値を返す場合に使います。戻り値が無ければ、

```
return ;
```

と、記述します。
例 2 の main 関数は、戻り値のデータ型を「int」で定義し、戻り値を return で返します。

5.3 関数を作ってみよう

図 5-1 の共通部分を関数化すると、になります。

```
/*
 *  待ち時間関数
 */
void waitMs(int ms)
{
    int i, j, k;
    for (j = 0; j < ms; j++)
    {
        for (i = 0; i < 1560; i++)
            k++;
    }
}
```

図 5-2　待ち時間関数のプログラム

【 関数の呼び出し 】

C 言語では main 関数が一番始めに実行されますが、定義した関数は関数の呼び出しを行わなければ実行されません。

C 言語では、関数の呼び出しを以下のように記述します。
(　) 内には引数を書きますが、引数が無い場合は記述する必要はありません。

```
関数名(　);
```

「関数の呼び出し」をフローチャートでは四角枠の両端の縦線を二重に描きます。p.64 図 4-3 の「待ち時間」は右図のように描くのが正解です。

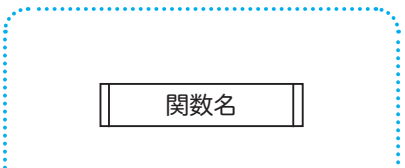

図 5-3　関数の呼び出し

プログラム 5-1 の白抜き枠で囲まれた部分を見てください。
この記述は関数の呼び出しを行っている部分です。関数が呼び出されることによって初めて実行されます。関数を定義しているだけでは実行されません。待ち時間関数内の処理が行われた後は、main 関数の処理を続けて行っていきます。

プログラム例　5-1

```
05  #include <3052f.h> // 3052F 固有の定数
06
07  /*
08   *  待ち時間関数
09   */
10  void waitMs(int ms)
11  {
12      int i, j, k;
13      for (j = 0; j < ms; j++)
14      {
15          for (i = 0; i < 1560; i++)
16              k++;
17      }
18  }
19
20  /*
21   *  main 関数
22   */
23  int main(void)
24  {
25      // 入出力設定
26      P4.DDR = 0xFF; // 出力 LED 横行
27      PB.DDR = 0xFF; // 出力 LED 縦行
28
29      // 出力レベル設定
30      PB.DR.BYTE = 0xFE; // 1111 1110 カソード
31
32      // 永久ループ
33      while (1)
34      {
35          P4.DR.BYTE = 0x80; // 右上の LED を点灯
36          waitMs(500);       // 待ち時間関数の呼び出し
37
38          P4.DR.BYTE = 0x00; // 全消灯
39          waitMs(500);       // 待ち時間関数の呼び出し
40      }
41
42      return 0;
43  }
```

5.4 プログラムの応用

ここからは、プログラムの応用です。
今まで DR にはバイト単位でアクセスしてきましたが、DR には 2 進数で書き込むことができないので、DR に書き込む値を考えるのが面倒だったと思います。
STEP02 でも少し言いましたが、DR にはバイト単位でアクセスする方法とビット単位でアクセスする方法があります。課題のように右上の LED のみを点灯させる場合はバイト単位で DR に書き込む値を考えるより、ビット単位で書き込む値を考える方が楽ですね。

【 ビット単位で DR にアクセスする 】

「3052f.h」によって、P4DR、PBDR のビットには次のような名前でアクセスできます。

P4DR は	PBDR は
P4.DR.BIT.B0	PB.DR.BIT.B0
P4.DR.BIT.B1	PB.DR.BIT.B1
P4.DR.BIT.B2	PB.DR.BIT.B2
P4.DR.BIT.B3	PB.DR.BIT.B3
P4.DR.BIT.B4	PB.DR.BIT.B4
P4.DR.BIT.B5	PB.DR.BIT.B5
P4.DR.BIT.B6	PB.DR.BIT.B6
P4.DR.BIT.B7	PB.DR.BIT.B7

例）

PBDR にバイト単位でアクセス

```
PB.DR.BYTE = 0xFE;
```

＝

PBDR にビット単位でアクセス

```
PB.DR.BIT.B0 = 0;
PB.DR.BIT.B1 = 1;
PB.DR.BIT.B2 = 1;
PB.DR.BIT.B3 = 1;
PB.DR.BIT.B4 = 1;
PB.DR.BIT.B5 = 1;
PB.DR.BIT.B6 = 1;
PB.DR.BIT.B7 = 1;
```

では、プログラム 5.1 で DR にビット単位でアクセスする場合、どのような記述になるのか考えてみましょう。

問題　5-1

プログラム 5-1 では、P4DR にバイト単位でアクセスしています。これをビット単位でアクセスすると、どのような記述になるのか考えましょう。四角に当てはまる数字を書いてください。

※ 但し、P4.DR の初期値はすべて 0 で、初期値になっていることが確実であるとき。

```
P4.DR.BYTE = 0x80;    →   P4.DR.BIT.□ = □;
```

答えはページ下

ビット単位で DR にアクセスする方法は分かっていただけたと思います。
さらに、C 言語の否定演算子を使うとプログラムを簡略化できます。否定演算子は 0 のものを 1 に、1 のものを 0 に反転する命令です。

【 否定演算子 】

C 言語では、0 には偽、0 以外 (一般的に 1) には真という意味があることは既に話をしました。
否定演算子は、偽を真、真を偽にするもので、記述は ! (俗に言うビックリマーク) です。

```
!a    : 変数 a が真なら偽に、偽なら真にする
```

例)

```
int a = 0;
while (1)
    a = !a;
```

変数「a」が否定演算子によって 1 なら 0 になり、0 なら 1 になる処理を繰り返し続けます。

答え：　P8.DR.BIT.B7 = 1;

プログラム　5-2

ここまで学習したことを基にプログラム 5-1 を改良してみましょう。また、ポートの入出力設定、LED 全消灯の処理を行う「初期化関数」も合わせて考えてみましょう。まずは、フローチャートを描いて整理しておきます。

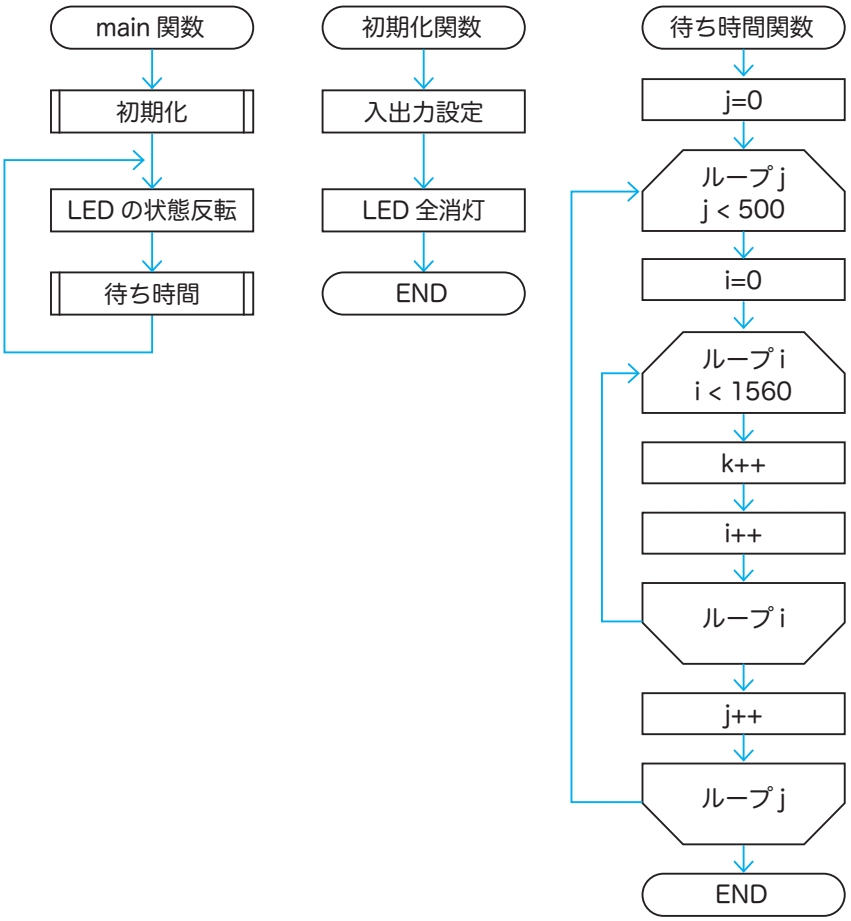

図 5-4　応用プログラムのフローチャート