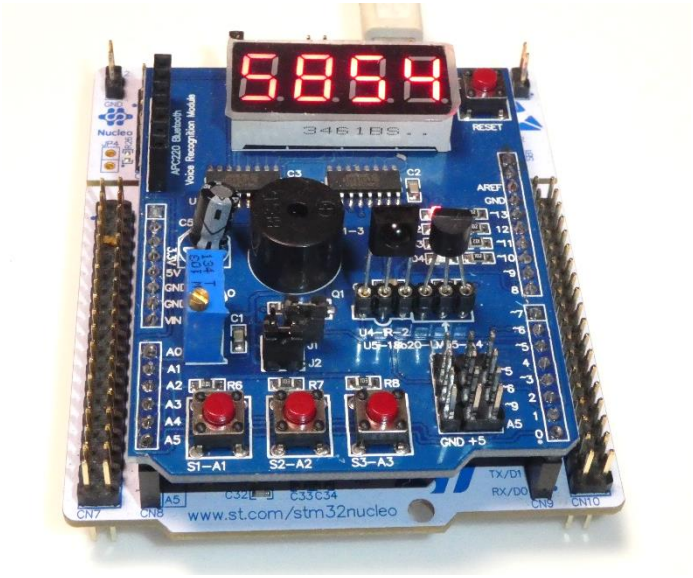
STM Nucleo で学ぶ C プログラミング入門



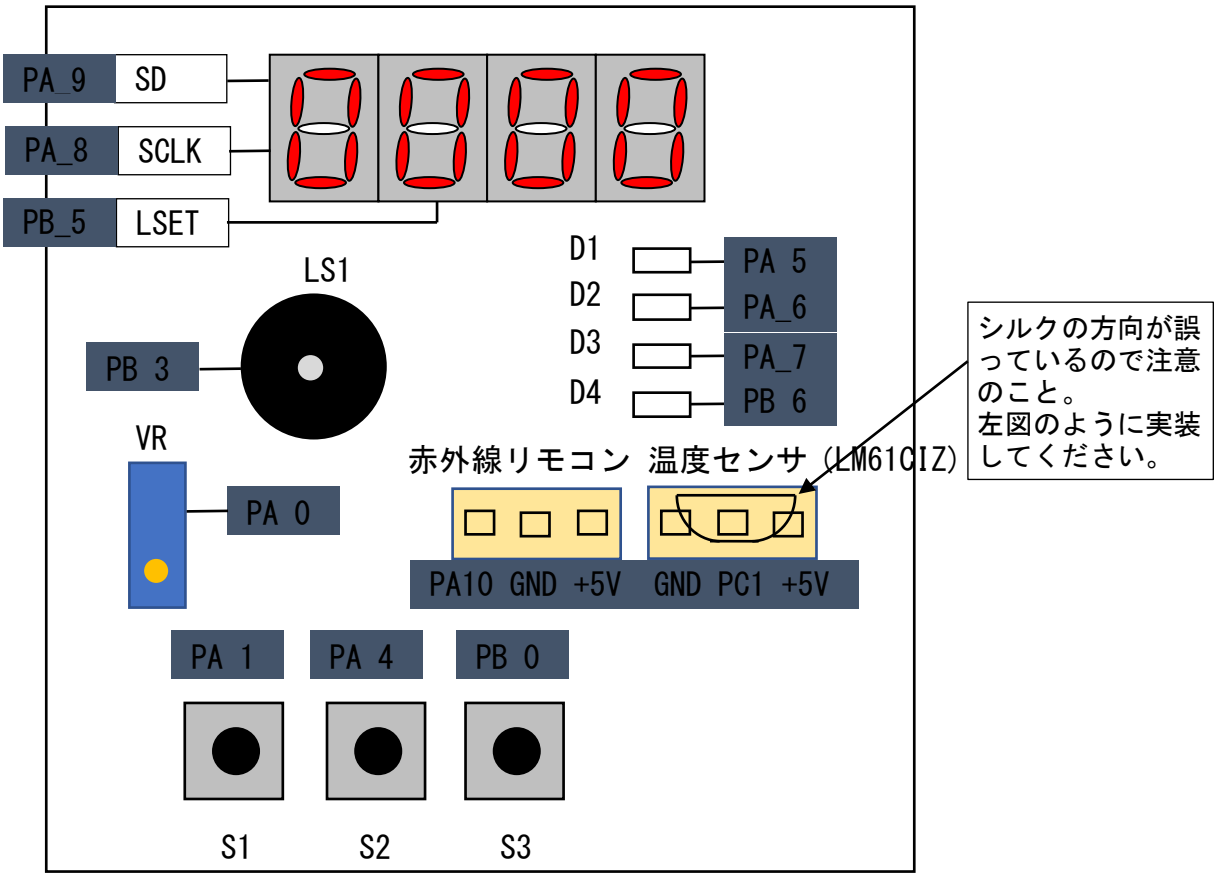
ZEP

1.2.3　　研修ボードのピン接続

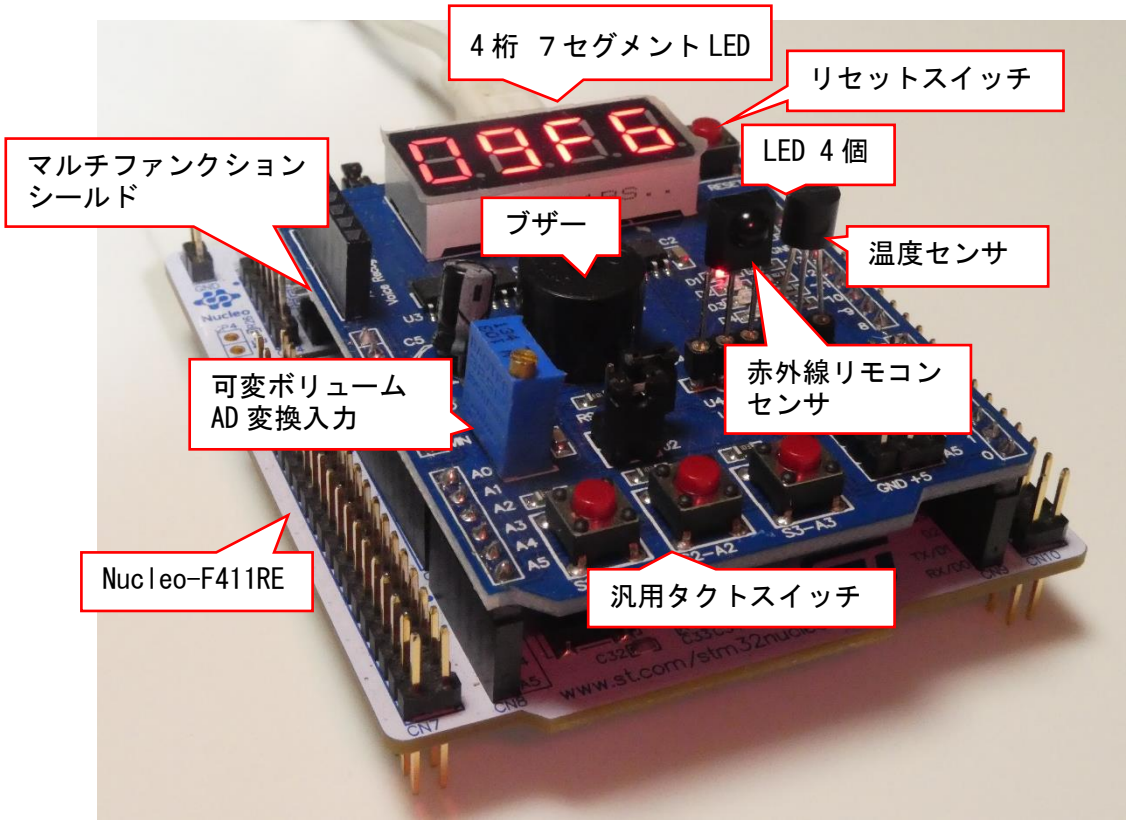
Nucleo-F411RE　入出力ピン構成

CN6						CN5					
						PB_8	SPI5_MOSI		PWM4/3		I2C1_SCL
						PB_9	SPI2_SSEL		PWM4/4		I2c1_SDA
						AVDD					
				NC		GND					
				IOREF		PA_5	SPI1_SCLK		PWM2/1	ADC1/3	
				RESET		PA_6	SPI1_MISO		PWM3/1	ADC1/6	
				3V3		PA_7	SPI1_MOSI		PWM1/1N	ADC1/7	
				5V		PB_6		UART1_TX	PWM4/1		I2C1_SCL
				GND		PC_7	SPI2_SCLK	UART6_RX	PWM3/2		
				GND		PA_9		UART1_TX	PWM1/2		
				VIN		PA_8			PWM1/1		I2C3_SCL
						PB_10	SPI2_SCLK		PWM2/3		I2C2_SCL
ADC1/0	PWM2/1	UART2_CTS		PA_0		PB_4	SPI3_MISO		PWM3/1		I2C3_SDA
ADC1/1	PWM2/2	USRT2_RTS	SPI4_MOSI	PA_1		PB_5	SPI3_MOSI		PWM3/2		
ADC1/4			SPI3_SSEL	PA_4		PB_3	SPI3_SCLK	UART1_RX	PWM2/2		I2C2_SDA
ADC1/8	PWM1/2N		SPI5_SCLK	PB_0		PA_10	SPI5_MOSI	UART1_RX	PWM1/3		
ADC1/11				PC_1		PA_2		UART2_TX			
ADC1/10				PC_0		PA_3		UART2_RX			

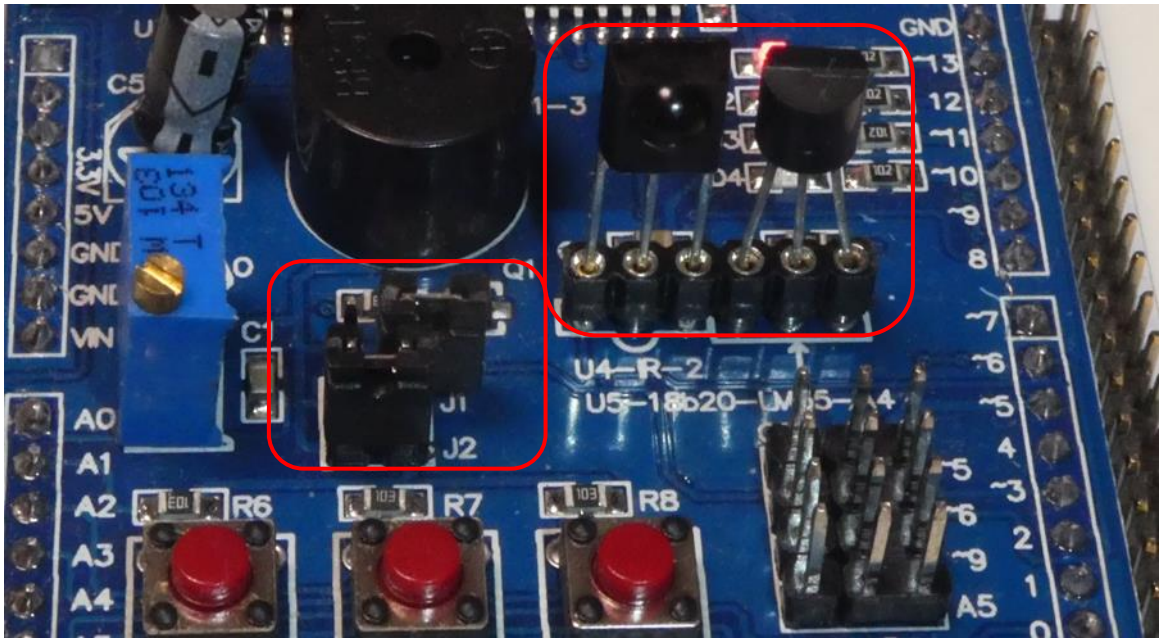
マルチファンクションシールドのピン接続



■　研修機材外観



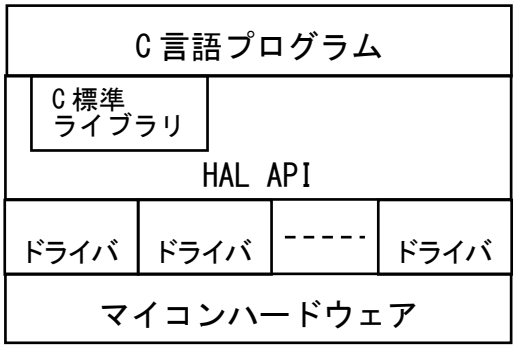
■　ジャンパの設定/センサの実装



1.2.4　　Mbed OS

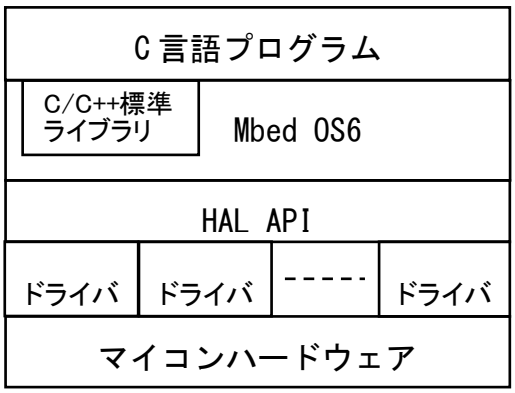
パソコン上で行う C 言語研修では、Windows や Linux などのパソコンで動作している OS 上で動くアプリケーションを作成します。パソコン画面への出力やキーボードからの入力は、Windows の API（アプリケーション・プログラミング・インタフェース）を通して行います。

一方、性能があまり高くなく、使用できるメモリが少ないマイコンボード上で動作する C 言語プログラムは、下記の様にマイコンボードのハードウェアを直接に C 言語プログラムから動かしたり、ハードウェアを仮想化した HAL（ハードウェア・アブストラクション・レイヤ）を通して IO を制御します。printf()や scanf()のような標準入出力関数は、HAL とインタフェースする例が多いです。



マイコンボード上の C 言語プログラム

mbed OS は、2009 年にマイコンボード上で動くプログラムを手早く作成するインタフェースとして誕生しました。発表当初は、mbed は、オープンソースではありませんでしたが、2013 年 2 月に「mbed 2.0」の発表とあわせてオープンソースになりました。この mbed2.0 は上図のような構成でしたが、2015 年にリリースされた RTOS 機能を取り入れた mbed OS 3 と mbed 2.0 を統合し、Mbed OS5が、2016 年にリリースされました。現在は、Mbed OS5 を組み込みマイコン用途に機能を整理した Mbed OS6がリリースされています。



Mbed OS6 を使ったソフトウェア構成

1.2.5　プログラム作成と実行

■　ソースプログラムの作成

IDE 上のエディタ（文書入力ソフト）を使用して、C 言語の文法に従って、ファイルを作成します。このファイルを「ソースプログラム」と言います。、今回使用した IDE では、C 言語の拡張版である C++の仕様でコンパイルが行われるため、「～.cpp」のファイル名で作成されます。

■　コンパイル

ソースプログラムをマシン語に翻訳します。翻訳するためのソフトウェアのことを「コンパイラ」と言い、翻訳することを「コンパイル」と言います。

コンパイルされたソースプログラムは「オブジェクトプログラム」というマシン語のファイルになり、「～.o」というファイル名になります。

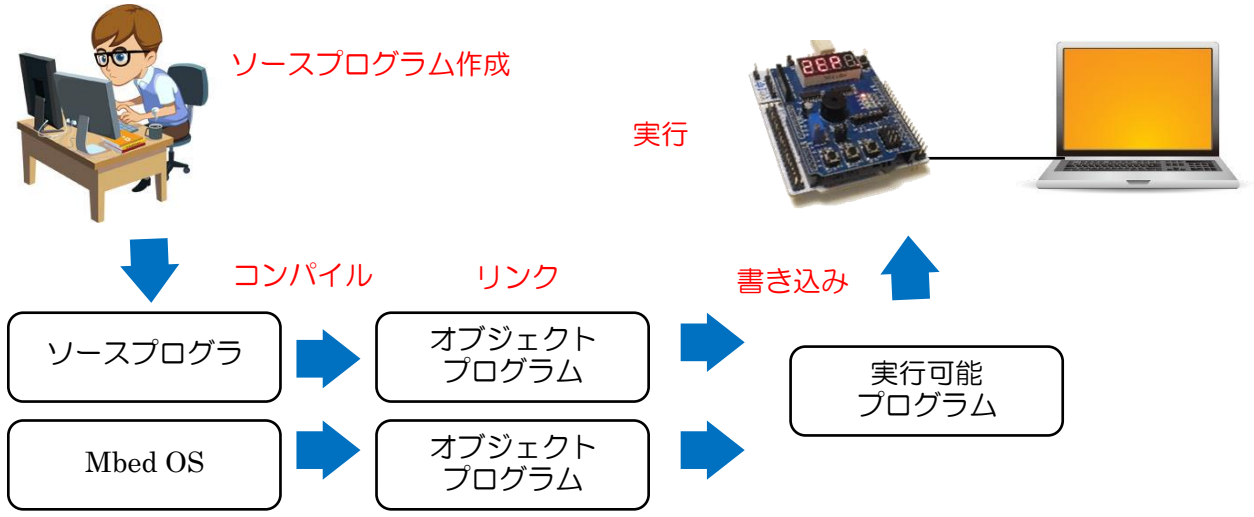
Mbed OS もソースコードで提供されており、作成したソースプログラムと同様に使用するマイコンボードの構成に合わせて、コンパイルが行われます。

■　リンク

作成したプログラムのオブジェクトファイルや mbed OS のオブジェクトは、実行するときのアドレスを割り付けたりする「リンカ」というソフトウェアで、「リンク」という作業を行います。

■　実行

「コンパイル」とリンクが終了すると実行可能プログラムが作成されます。これらのコンパイルとリンクの一連の指示を行うのが「ビルド」コマンドです。ビルドが完了すると実行可能プログラムは、ST-LINK を通してマイコンのフラッシュメモリに書き込まれます。



第2章　C プログラミングの基本

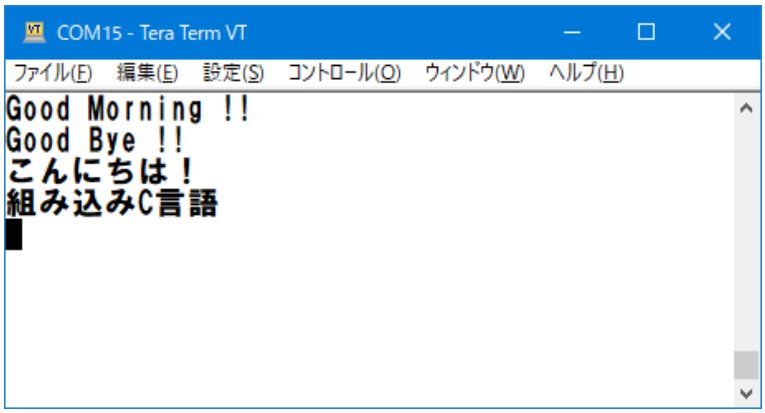
2.1　main 関数

【例題１】画面にメッセージを出力します。

【z_sample2_1.h】

```
#include <mbed.h>
#include <stdio.h>

void app_main()
{

    printf("Good Morning !! ¥n");    /*「Good Morning !!」を表示し改行する */
    printf("Good Bye !!¥n");         /*「Good Bye !!」を表示し改行する */
                                     /*「¥n」は改行を意味する */
    printf("こんにちは！¥n");          //  日本語も書けます
    printf("¥r 組み込み C 言語¥n");    //  漢字もOKです

}
```

【実行結果】



※実行はボード上のリセットボタンを押すことで何回でも行えます。

2.1.1　　main 関数の基本形

C 言語では、プログラムに実行させる処理を、「関数」という単位にまとめて記述します。

関数には、「標準ライブラリ関数」（システムが提供している関数）と「ユーザ関数」（ユーザ自身が作成する関数）の２つがあり、これらを組み合わせてプログラムを作成します。

ユーザ関数の中に、「main 関数」という特別な関数があります。ソースプログラムは main 関数の先頭から実行するようにビルドされるためです。ソースプログラムには、必ず１つの main 関数を作成します。

プログラム以下のように記述します。

main 関数　　main.cpp

```
#include "ユーザ関数記述ファイル名"
void app_main();
int main(void)
{
    処理１
    処理２
    処理３

  return  0;
}
```

ユーザ関数をインクルードする
プロトタイプ宣言
「main」関数であることを示す
関数の始まりを示す

main 関数に実行させる処理

関数の呼出先(mbed OS) に帰る文
関数の終わりを示す

ユーザ関数　　xxxx.h

```
#include <mbed.h>
void app_main()
{
    処理 1;
    処理 2;
    処理 3;
}
```

Mbed OS をインクルードする
ユーザ関数であることを示す
関数の始まりを示す

ユーザ関数に実行させる処理

関数の終わりを示す

※main(void)の void は省略可能です。

※この研修では、main 関数の処理としては、ユーザ関数「app_main()」のみを記述し、実際の処理は下記のようにユーザ関数に記述します。

※#include 文で実際の処理を行っているアプリケーション処理を挿入します。

※include 文、プロトタイプ宣言、ユーザ関数の詳細は６章で説明します。

第3章　ビット操作

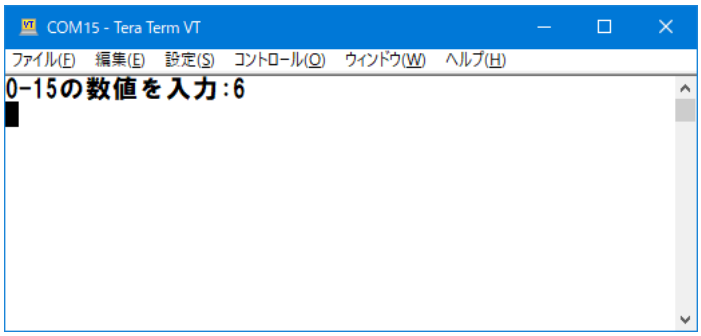
3.1　ビットデータの表現方法

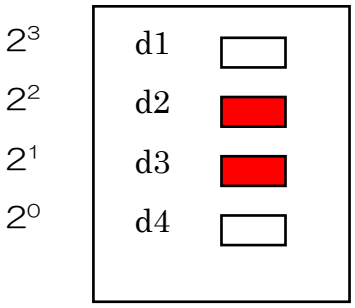
【例題１】　入力した0～15 の数値に対応して、４ つの LED に２進数で数値を表示します。

【z_sample3_1.h】

```
#include <mbed.h>
#include <iostream>
using namespace std;

BusOut led(PB_6, PA_7, PA_6, PA_5);     //LED を 4bit バス出力に設定（下位 bit->上位 bit）

void app_main(){
    int data;
    led = ~0;    //0 を反転して led に出力
    cout <<"0-15 の数値を入力:";
    cin >> data;     //整数を入力
    led = ~data;     //入力した数値を反転して led に出力
}
```

【実行結果】





入力した数値（0-15）を 2 進数で LED 表示する

日常生活では、日付や時間などのごく一部を除いて１０進数を用いるのが一般的ですが、マイコン内部では、０と１だけからなる２進数が使用されています。今回の研修のようにマイコンボードを動かすようなプログラムでは、マイコンに接続したハードウェアを動かすために２進数の知識が必須となります。C 言語での１０進数と２進数及び 16 進数の相互変換やビットの演算方法などを紹介します。

3.1.1　　2進数、16進数

2 進数	１６進数	2 進数	１６進数
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F
		10000	10

■　2進数の特長

- ０と１の２種類の数値で表現します。
- ２になると、桁上がりします。
- 桁上がりは、２ⁿごとに発生します。

２進数は、頻繁に桁上がりするため、文字や数値を２進数で表現すると大きな桁数になります。従って、コンピュータ内部で表現されるデータを簡潔に表現するために、１６進数に変換することがあります。

１６進数の特長

- ０～１５までの１６種類の数値で表現します。
- １０～１５迄を１桁であらわすために、A、B、C、D、E、F を使用します。
- １６になると、桁上がりします。
- 桁上がりは、１６ⁿごとに発生します。

第4章　制御用構文

4.1　制御構文とは

通常、C 言語のプログラムは、main 関数の先頭から順に処理を実行します。制御用構文を利用することで、処理を条件によって分岐したり、処理を繰り返し実行したりすることができます。

このようにプログラムを先頭から順に実行、条件で分岐、繰り返しの３つの構造で作成する手法を構造化プログラミングと呼びます。

C 言語の制御用構文には、分岐に使用する「if 文」と「switch 文」、繰り返しに使用する「while 文」、「for 文」、「do～while 文」があります。

分岐文

if 文	条件式が「成立」か「不成立」かによって処理を分岐する（２分岐）
switch 文	変数か計算式の値を条件に、処理を多方向に分岐する（多分岐）

繰り返し（ループ）文

while 文	条件式が「成立」の間、処理を繰り返す • while 文は条件式のみを記述する • for 文は条件式の他に前処理の式、後処理の式を記述する
for 文	
do～while 文	処理を実行したあと条件式を判定し、「成立」の間、処理を繰り返す

～構造化プログラミング～

「構造化プログラミング」とは、「順次」「分岐」「繰り返し」の３つの構造のみで構成する手法です。

- 順次　……　上から順番に実行する
- 分岐　……　処理を分岐する
- 繰り返し……　処理を繰り返す

これにより、プログラムが複雑な記述になるのを防止し、プログラムの構造を明確にすることができます。

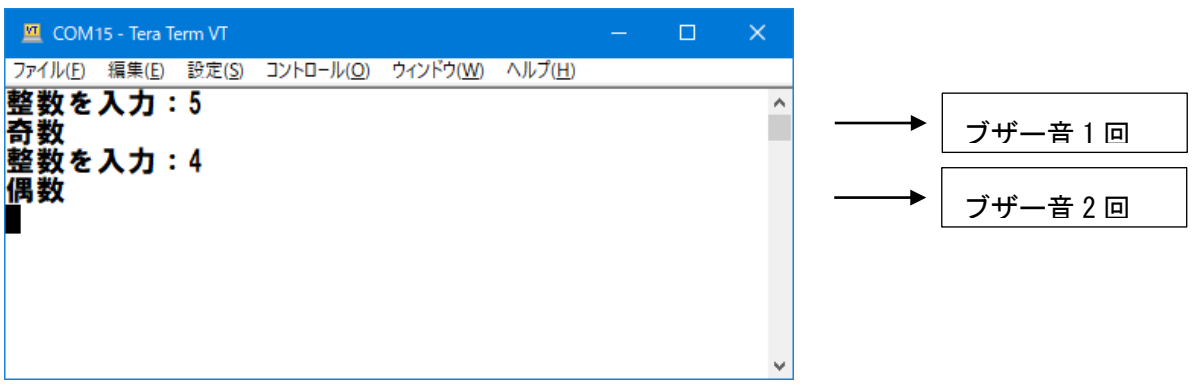
4.2　if 文

【例題１】入力した整数が偶数の場合は２つ、奇数の場合は１つブザーを鳴らします。

【z_sample4_1.h】

```
#include <mbed.h>
#include <iostream>
using namespace std;
DigitalInOut ls1(PB_3); //D3 を入出力に設定

void app_main()
{
    int data;
    ls1 = 0;//ブザー ON
    ls1.input();//ブザー OFF(High-z)

    cout << "整数を入力:";
    cin >> data;
    if(data % 2 != 0){
        cout << "奇数¥n";
        ls1.output();
        wait_us(100000);
        ls1.input();
    }else{
        cout << "偶数¥n";
        ls1.output();
        wait_us(100000);
        ls1.input();
        wait_us(100000);
        ls1.output();
        wait_us(100000);
        ls1.input();
    }
}
```

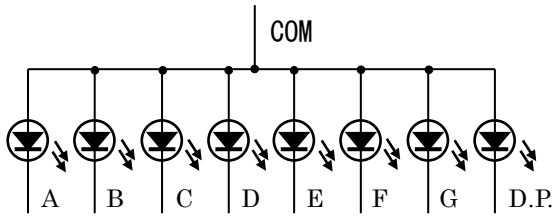
【実行結果】



■　7セグメント LED の点灯

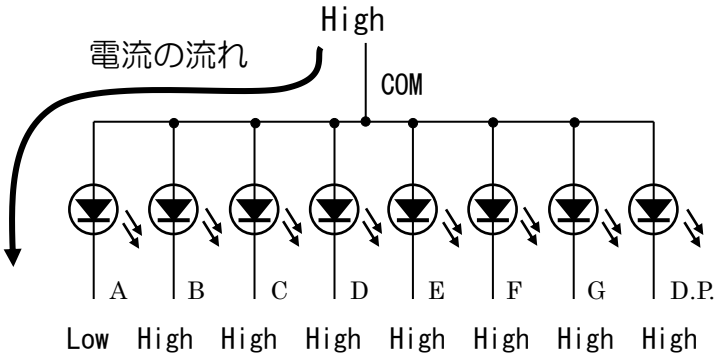
７セグメント LED には、LED の色が「赤」「緑」「青」「橙」「白」のものや、1 桁表示のもの、複数桁の表示ができるものなどがあります。構造上は、アノードコモン、カソードコモンの２つのタイプがあります。マルチファンクションシールドは、下記のようなアノードコモンタイプを使用しています。

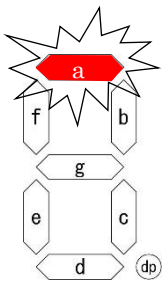
アノードコモンタイプ



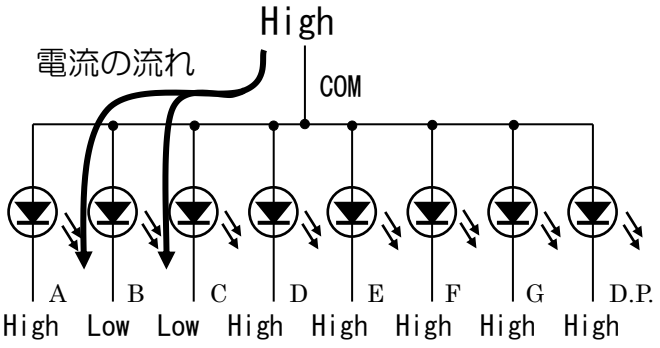
■　7 セグメント LED の点灯

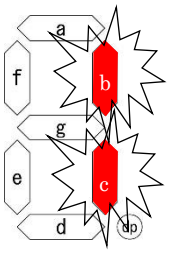
アノードコモンタイプの場合、共通化しているアノード部分（=COM 端子）を「High」に設定し、LED に接続されている端子を「Low」に設定することで対応した LED が点灯します。





数字の「1」に見える点灯をさせたい場合、「b」と「c」を同時に点灯させれば良いので、下図のように COM 端子を「High」にし、「b」と「c」の端子を「Low」にします。



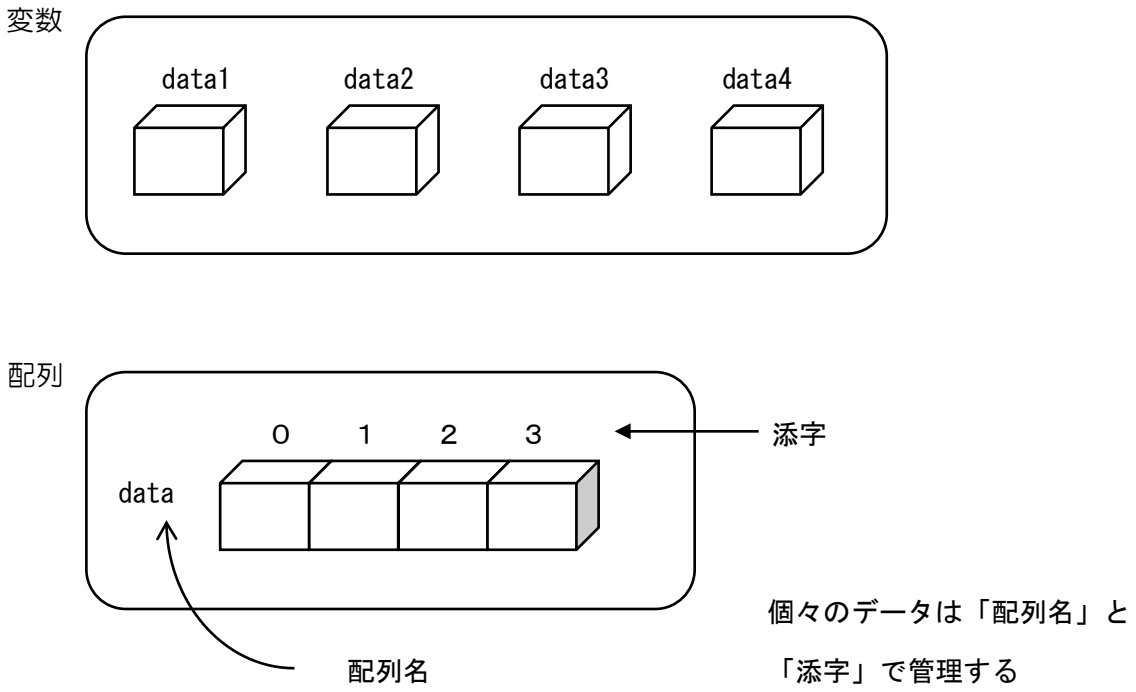


第5章　配列

5.1　配列

プログラムで大量のデータを扱う場合、データを効率よく管理するため「配列」を利用します。

配列は、同じデータ型の変数をまとめて扱うものです。各変数（配列の要素）は、先頭から何番目（０、１、２．．．）データであるかを表す「添字」をつけて管理します。



5.1.1　　配列の基本

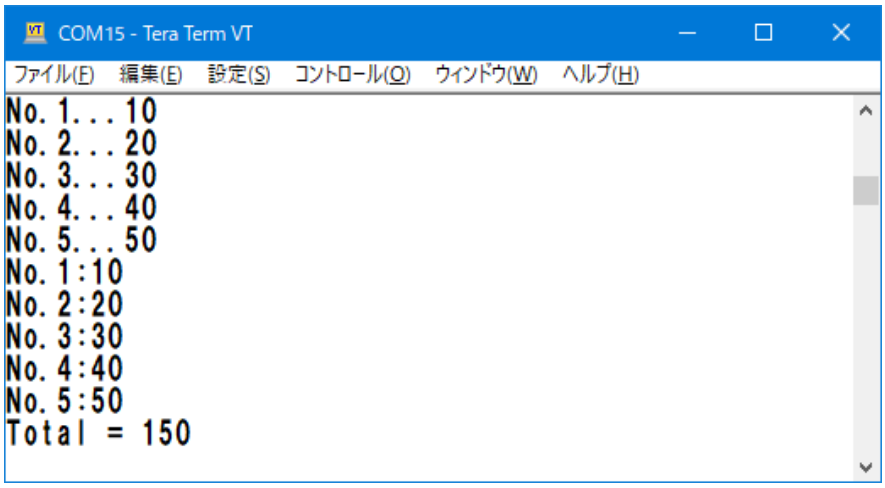
【例題１】５人の得点をキーボードから入力し、入力したデータを順に出力します。

また、入力したデータの合計値を計算します。

【z_sample5_1.h】

```
#include<mbed.h>
#include<iostream>
using namespace std;

void app_main(void)
{
    int data[5];                /* 配列の定義 */
    int cnt, total = 0;

    for(cnt=0; cnt<5; cnt++){           /* data[0]～data[4]を順に参照する */
        cout <<"No." << cnt+1 <<"...";
        cin >> data[cnt];   /* 入力したデータを順に配列に保存 */
        total = total + data[cnt];      /* 合計値の計算 */
    }

    for(cnt=0; cnt<5; cnt++){           /* data[0]～data[4]を参照する */
        cout <<"No." << cnt+1 <<":" <<data[cnt] <<"¥n";
    }
    cout <<"Total = " << total <<"¥n";      /*  合計値を出力する */
}
```

【実行結果】



第6章　関数とクラス

6.1　関数によるモジュール化

【例題１】４ 章の例題 ８ のマイコンボードのハードウェアを操作する部分をモジュール化して、multi.h に集約し、同じ動作をするプログラムを記述しました。

【z_sample6_1.h】

```
#include "multi.h"

void app_main(void){
    char data, s, data0 = 0xf7, data1 = 0xfe;   //data0:Low data1:High
    char col = 0x01;    //桁:01 4 桁目を ON
    multiini();
    while(true){
        switch(col){    //col に合わせて s1,s2,s3 の状態読み出す
            case 1:s = s1; break;
            case 2:s = s2; break;
            case 4:s = s3; break;
            case 8:s = 0;
        }
        if(s == 0){     //表示の切り替え
            data = data0;
        }else{
            data = data1;
        }
        int leddata = data << 8 | col;
        sevenseg(leddata);
        wait_us(1000);        //1ms 待ち
        col <<= 1;           //桁を右にシフト
        if(col == 0x10) col = 0x01; //右端に来たら左端に戻す
    }
}
```

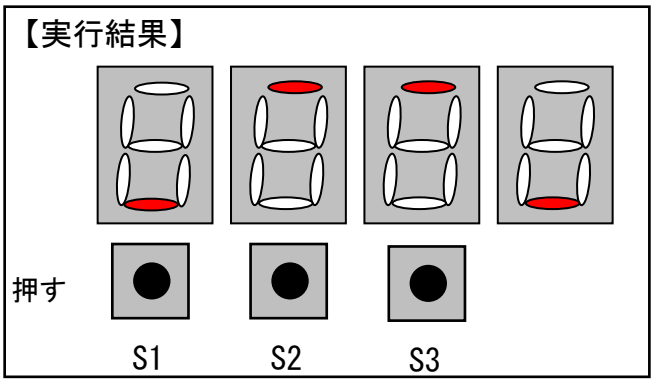
【multi.h】

```
#include <mbed.h>
#include<iostream>
using namespace std;

DigitalOut d1(PA_5);    //D13 を出力に設定
DigitalOut d2(PA_6);    //D12 を出力に設定
DigitalOut d3(PA_7);    //D11 を出力に設定
DigitalOut d4(PB_6);    //D10 を出力に設定

DigitalInOut ls1(PB_3); //D3 を入出力に設定

DigitalIn s1(PA_1);
DigitalIn s2(PA_4);
DigitalIn s3(PB_0);

AnalogIn a0(PA_0);

DigitalOut sd(PA_9);
DigitalOut sclk(PA_8);
DigitalOut lset(PB_5);
```

```
void sevenseg(int leddata){
    for(int n = 0; n < 16; n++){     //16bit 分ループ
        if((leddata & 0x8000) != 0){   //最上位 bit が 1？
            sd = 1;      //シリアル入力を 1
        }else{
            sd = 0;      //シリアル入力を 0
        }
        sclk = 1;        //クロックを送出
        sclk = 0;
        leddata <<= 1;
    }
    lset = 1;            //REG をセットして LED に送出
    lset = 0;
}

void beep(int tim){
    ls1.output();
    wait_us(tim);
    ls1.input();
}

void multiini(){
    sevenseg(0);
    d1 = 1;
    d2 = 1;
    d3 = 1;
    d4 = 1;
    ls1 = 0;
    ls1.input();
}
```

【実行結果】

例題８と同じ

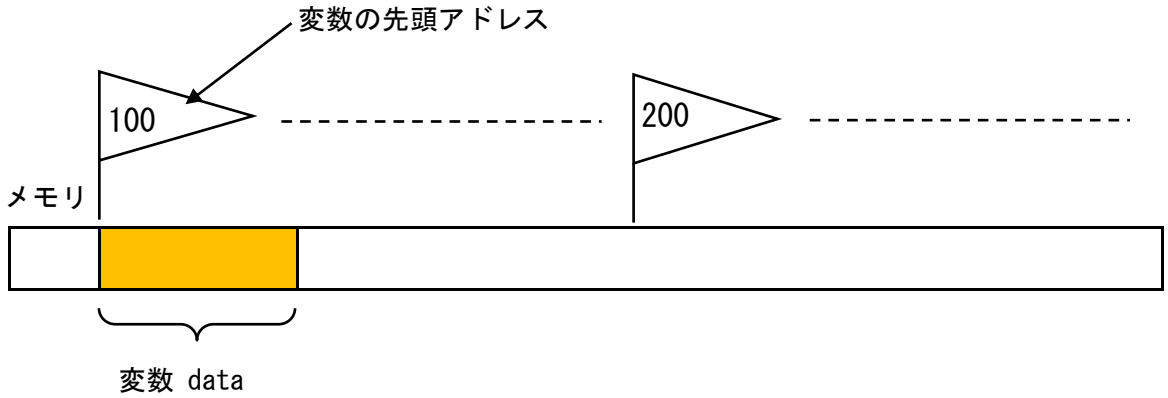


第7章　ポインタ

7.1　ポインタの概要

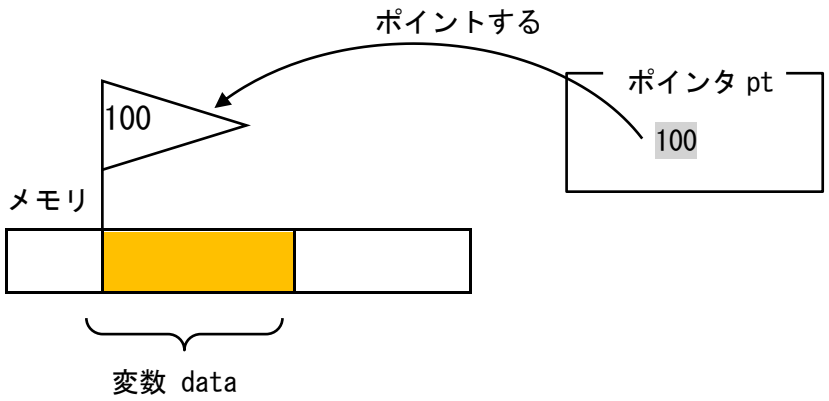
コンピュータの記憶領域であるメモリは、「アドレス」によりデータの格納場所を管理しています。

変数を定義するとメモリ上に領域が確保され、変数の値を格納します。確保した領域の先頭に割り当てられたアドレスを「先頭アドレス」と呼びます。コンピュータの内部では「先頭アドレス」を用いて変数の処理を行います。



C 言語では先頭アドレスを用いて変数の処理を行うことができます。先頭アドレスを「ポインタ」と呼ぶ変数に格納し、プログラム中で使用します。

例）変数 data の先頭アドレスを格納したポインタ pt



7.2　ポインタの基本

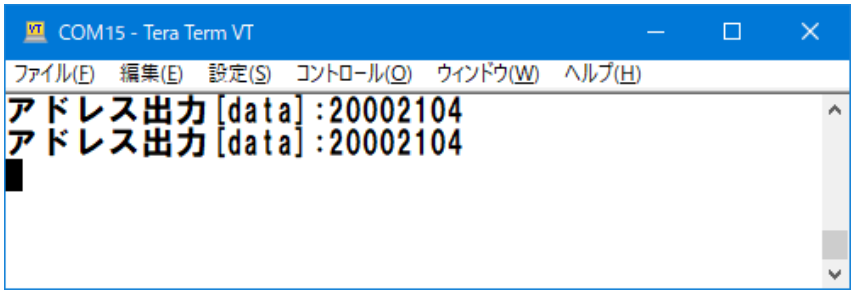
7.2.1　変数のアドレス

【例題１】int 型変数の先頭アドレスを出力します。

【z_sample7_1.h】

```
#include<mbed.h>
#include<iostream>
using namespace std;

void app_main(){
    int data;

    /* アドレス出力は、hex を使用する */
    cout << "アドレス出力[data]:" << &data << endl;
    /* アドレス出力の変換仕様は、%pを使用する */
    printf("アドレス出力[data]:%p¥n", &data);
}
```

【実行結果】



第8章　構造体

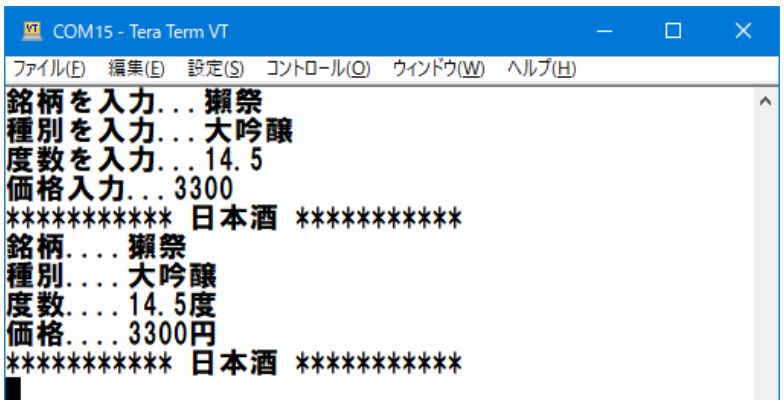
8.1　構造体の基本

【例題１】日本酒名、種別、度数、価格を入力し、を出力します。

【z_sample8_1.h】

```
#include<mbed.h>
#include<iostream>
#include<iomanip>
using namespace std;

/* 構造体の宣言 */
struct sake{
    string name;
    string type;
    double level;
    int price;
};

void app_main(){

    /* 構造体の定義 */
    sake data;

    /* 構造体の各メンバーへデータ入力 */
    cout << "銘柄を入力...";
    cin >> data.name;
    cout << "種別を入力...";
    cin >> data.type;
    cout << "度数を入力...";
    cin >> data.level;
    cout << "価格入力...";
    cin >> data.price;

    /* 入力したデータを出力 */
    cout << "********** 日本酒 **********¥n";
    cout << "銘柄...." << data.name << endl;
    cout << "種別...." << data.type << endl;
    cout << "度数...." << fixed << setprecision(1) << data.level << "度¥n";
    cout << "価格...." << data.price << "円¥n";
    cout << "********** 日本酒 **********¥n";
}
```

【実行結果】

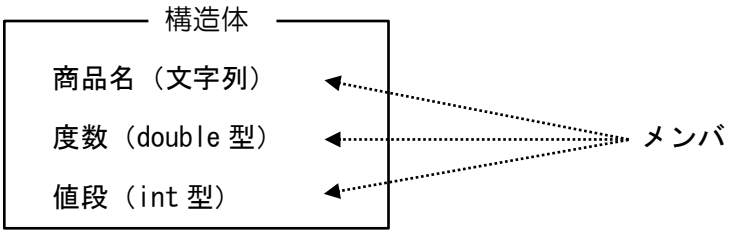


8.1.1　　構造体とは

C 言語では、配列を利用することで、同じ型のデータをまとめて扱うことができましたが、商品名（文字列）や値段（int 型）などの型の異なるデータをまとめて扱う場合には「構造体」を使います。C++では、クラスが使えるため同じことがクラスでも実現可能です。C 言語の構造体がデータのみを扱うのに対し、C++では、クラスと同様に関数もまとめて扱うことができます。

C++において、クラスと構造体は、既定のアクセス制限が private か public かの違いだけで、機能的には同じです。使い分けとしては、基本的にはクラスを使い、データ主体でメンバ変数を公開して使うようなものには構造体を使うのが便利です。

構造体の各データや関数（関数は C++のみ）を「メンバ」と呼びます。



構造体を利用する場合、どのような種類のデータをまとめて扱うかを決定する「構造体の宣言」と実際にメモリ上に構造体の領域を確保する「構造体の定義」という２つの手続きを行います。この手順はクラスと同様です。構造体の定義を行うことでメモリ上に領域が確保されます。