ESライブラリ＆＆
ゲームプログラミング

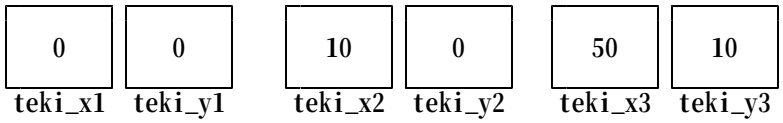
○　荷物勇者編 - 第７回　地形データで壁の描画３　○

配列

プログラムでは、データを変数に入れて扱います。基本的に１つのデータにつき１つの変数を宣言する必要がありますが、この方法ではプログラム作成上、困難になる場合があります。
たとえば、敵キャラクターの座標が、teki_x1, teki_y1, teki_x2, teki_y2…というint型の変数に格納されているとします。

```
int   teki_x1 =  0, teki_y1 =  0;   // 敵キャラNo.1
int   teki_x2 = 10, teki_y2 =  0;   // 敵キャラNo.2
int   teki_x3 = 50, teki_y3 = 10;   // 敵キャラNo.3
```

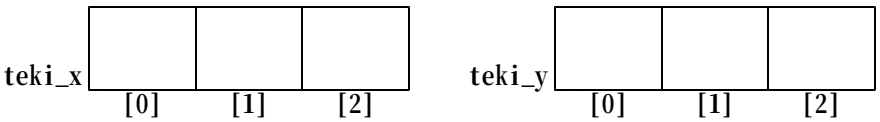
これは、以下のように個々の変数を用意していることになります。



ここで、敵キャラクターの数が多くなった場合、たとえば、1000になったとしたら、teki_x1000とteki_y1000まで２０００個もの変数を定義しなければいけなくなり、かなり面倒です。

このような時、「配列」の出番となります。配列は、「個々の変数をひとかたまりに宣言できるもの」といえます。いままでは、プログラムの中で値を格納するためには、必要な分だけ変数を作ってきました。配列を使うと、同じ型のデータをまとめて宣言できるようになります。

配列を図で表現すると、以下のように変数が連続しているイメージになります。上の図との違いは、変数が連続している点にあります。



配列とは、まとめて箱を確保し、位置(先頭からの距離)を指定してアクセスできる変数といえます。

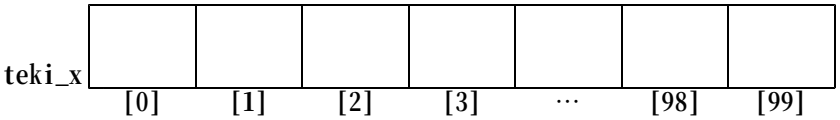
配列の定義

変数を使用するには定義が必要であったのと同じように、配列を使うためにも定義が必要です。

```
int   teki_x[100];
```

これは、teki_xという名前のint型の配列を定義しています。変数名の後に角括弧[]でくくって必要な個数を記述することで、配列の数(これを要素数といいます)を定義することができます。この例ではint型の１００個のデータが格納できる配列を定義しています。

配列を定義すると、メモリ上には以下の図のような変数の格納領域ができあがります。ちょうど、teki_x[0], teki_x[1], …, teki_x[99]という名前の１００個の変数が並んだ形になります。先頭がteki_x[1]ではなく、teki_x[0]になっていることに注意してください。



配列の定義の最初のintは、この配列がint型の変数を格納することを意味しています。もちろん、double型の配列やchar型の配列、そしてポインタや構造体などさまざまな変数を格納する配列を定義できます。

ただし、１つの配列を構成する要素は、すべて同じ型である必要があります。たとえば、１個目から３個目まではint型で、４個目から７個目まではdouble型にしたい、というような定義はできません。
配列を定義する際の括弧[]で囲まれた配列の要素数は、正の整数の定数式でなければなりません。小数や変数を指定することはできません。

```
int    array1[0.5];        // 小数はエラー

int    i = 100;
int    array2[i];          // iは変数なのでエラー

int    array3[310 / 3];    // 定数式なのでＯＫ
int    array4[0];          // 要素数が０の配列はエラー
```

配列の要素数は、プログラム実行中に変更することはできません。コンパイル時に決定されるため、プログラムの設計時によく考えておくことが必要です。

配列の添字

配列の個々の変数(これを配列の要素と呼びます)は、括弧[]を使ってアクセスすることができます。たとえば、前節の配列teki_xでは、teki_x[0]やteki_x[1]とすることで各要素にアクセスできます。この括弧でくくられた数字の部分を配列の添字と呼びます。Ｃ言語で配列の要素を扱う場合、次のことを注意してください。

配列の最初の要素は添字が０であり、最後の添字は「配列の要素数－１」である

配列の最初の要素には、添字を0にして(たとえば、teki_x[0])としてアクセスします。その次が1でアクセスします。0から始まることに注意してください。
１００個の要素数の場合、最後の１００番目の要素にアクセスするためには、添字を99とします。100にしてはいけません。0から始まるので、そこから100数えると最後は99になるわけです。また、添字には変数を使用することもとできます。たとえば、

```
int    array[100];
int    i = 9;
int    j = array[i];
```

というように変数を使ってアクセスすることもできます。

配列の値の参照と代入

配列の要素は、添字が必要ということを除けば、通常の変数と同じように使用することができます。たとえば、「int array[100]」として定義した場合、array[添字]はint型の変数として扱われます。
array[50]に0を代入したければ、

```
array[50] = 0;
```

とします。配列は要素ごとにしかアクセスできません。=演算子である配列を別の配列にコピーしたり、+=演算子で「配列＋配列」などとすることはできません。

```
int    a[5], b[5];

a[0] = 1;
a[1] = 2;
a[2] = 3;
a[3] = 4;
a[4] = 5;

b  = a;      // 配列のコピー？→エラー
b += a;      // 配列の加算？  →エラー
```

このような場合、次のように要素１つずつ処理を行う必要があります。

```
b[0] = a[0];
b[1] = a[1];
b[2] = a[2];
b[3] = a[3];
b[4] = a[4];
```

配列をまとめて処理する場合は、ループを使うと効率よいプログラムを作成することができます。また、配列のコピー程度の操作であれば、メモリ操作関数でも行うことができます。

ところで、前述の配列a，bをまちがえて

```
b[10] = a[10];
```

とした場合、配列a，bともに要素数が５であり、10は存在しません。Ｃ言語では、配列の境界チェックを行わないため、コンパイルエラーにはなりません。このようなプログラムでは、メモリを破壊しながら処理を続けてしまいます。まったく関係のない変数やプログラムの一部が壊されてしまいます。

配列を使用するときは、範囲外にならないように、プログラム側で充分考慮する必要があります。

課　　題

地形データmapData0からmapData9を配列にまとめましょう。

(１)１０行１０列の地形データを格納する領域を配列に変更しましょう。

現在、以下のように宣言されています。

```
// 地形データ
tstring     mapData0;     // ０行目のデータ(一番上の行)
tstring     mapData1;     // １行目のデータ
tstring     mapData2;     // ２行目のデータ
tstring     mapData3;     // ３行目のデータ
tstring     mapData4;     // ４行目のデータ
tstring     mapData5;     // ５行目のデータ
tstring     mapData6;     // ６行目のデータ
tstring     mapData7;     // ７行目のデータ
tstring     mapData8;     // ８行目のデータ
tstring     mapData9;     // ９行目のデータ
```

文字列を格納できるtstring型を１０個宣言しています。同じ型の変数で、同じ目的に使う変数は、配列でまとめて宣言した方が効率の良いプログラムを作成できます。

以下のプログラムの足りない部分(?の部分)を補い、適切な場所に追加しましょう。

```
// 地形データ
tstring     mapData????;
```

上記により、madData0からmapData9は使わなくなりますので、削除もしくはコメントにしましょう。

(２)地形データの設定部分(マップデータ初期化)を配列に変更しましょう。

ヒント１：配列は先頭が添字０、最後が宣言数マイナス１となります
ヒント２：配列mapDataの場合は、先頭がmapData[0]、最後がmapData[9]です
ヒント３：これまでの０行目のデータ(mapData0)に対応するのはmapData[0]になります
ヒント４：同様に、１行目のデータmapData1→mapData[1]、２行目はmapData2→mapData[2]です

ヒント５：

```
mapData[0] ##########  (mapData0)
mapData[1] #        #  (mapData1)
mapData[2] # ###### #  (mapData2)
mapData[3] #      # #  (mapData3)
mapData[4] # #  # # #  (mapData4)
mapData[5] # #    # #  (mapData5)
mapData[6] # #    # #  (mapData6)
mapData[7] # ###### #  (mapData7)
mapData[8] #        #  (mapData8)
mapData[9] ##########  (mapData9)
```

(３)０行目の地形データから１文字ずつ読み取り、座標(0.0, 0.0, 0.1)から(288.0, 0.0, 0.1)に'#'なら壁、' 'なら何も表示しないプログラムを配列に変更します。
以下ように該当部分を変更しましょう。

```
// 地形描画
// マップデータ０行目
for(unsigned int x = 0; x < 10; x++) {
    if(mapData0[x] == '#')
        SpriteBatch.Draw(*wallSpr, Vector3(※ここは各自考えてください※, 0.0f, 0.1f));
}
```

↓

```
// 地形描画
// マップデータ０行目
for(unsigned int x = 0; x < 10; x++) {
    if(mapData[0][x] == '#')
        SpriteBatch.Draw(*wallSpr, Vector3(※ここは各自考えてください※, 0.0f, 0.1f));
}
```

(４)(３)を参考にmapData1からmapData9の表示プログラムも配列に変更しましょう。

(５)地形データの表示プログラムをfor文を使った繰り返しで表示してみましょう。

地形データは、mapDataの[0]から[9]より１文字ずつ読み取り、表示してます。ここで、mapData[0]の'0'の部分(添字の部分)は数字です。つまり、mapData[0]からmapData[9]は、数字の部分だけが異なり、ほかは同じプログラムだということです。
ここで、縦方向は、

```
mapData[0] → ｙ座標 0.0 → 0 * 32.0
mapData[1] → ｙ座標32.0 → 1 * 32.0
mapData[2] → ｙ座標64.0 → 2 * 32.0
mapData[3] → ｙ座標96.0 → 3 * 32.0
      ⋮
```

というように、[]の中の数値とｙ座標に関連があります。for文でまとめるのに最適といえます。
さらに、横方向も

```
０文字目 → ｘ座標 0.0 → 0 * 32.0
１文字目 → ｘ座標32.0 → 1 * 32.0
２文字目 → ｘ座標64.0 → 2 * 32.0
３文字目 → ｘ座標96.0 → 3 * 32.0
      ⋮
```

となっているので、横方向もfor文でまとめることができます。

以上をまとめ、プログラム風に記述すると、

```
for(mapData[0]からmapData[9]まで) {
    mapDataのｙ行目から１文字ずつ読み取って描画 …※
}
```

※の部分は、

```
for(mapData[ｙ行目]の先頭から最後の文字まで) {
    mapData[ｙ行目]から１文字読み取り、該当地形を描画
}
```

となります。２つを併せると、

```
for(mapData[0]からmapData[9]まで) {
    for(mapData[ｙ行目]の先頭の文字から最後の文字まで) {
        mapData[ｙ行目]の１文字を読み取り、該当する地形を描画
    }
}
```

と、for文が２つ必要になります。縦のループと横のループということです。このような構造を多重ループ(入れ子)と呼びます。

以下のプログラムの足りない部分を補い、地形データ表示プログラムと差し替えましょう。

```
// 地形描画
for(unsigned int y = 0; y < 10; y++) {
    for(unsigned int x = 0; x < 10; x++) {
        if(※ここは各自考えましょう)
            SpriteBatch.Draw(※ここは各自考えましょう);
    }
}
```

ヒント：配列mapDataのｙ行目のｘ文字にアクセスするには、mapData[y][x]と記述します