前回の関数のまとめ(1)

復習

- ✓プログラムはmain()関数の先頭から実行される
- ✓関数はmain()関数または他の関数から呼び出されて実行される.
- ✓関数を呼び出す側の実引数の値が関数内の仮引数(変数)にコピーされる
- ✓関数内で求めた値はreturn文によって関数値として呼び出し側に戻される.

関数の定義

スタートポイント

関数の呼び出し

```
#include <stdio.h>

float menseki(float a, float b)
{
        return (a * b / 2);
}

int main(void)
{
        float x, y, kekka;
        printf("底辺は?"); scanf(" %f ", &x);
        printf("高さは?"); scanf(" %f ", &y);

        kekka = menseki(x, y);
        printf("面積は%fです. ¥n", kekka);
}
```

a, b: 仮引数
仮引数もローカル変数
→関数内でのみ有効

x, y:実引数

復習

ローカル変数(1)

```
#include      <stdio.h>

float menseki(float a, float b)
{
        float s;
        s = a * b / 2;
        return s;
}

int main(void)
{
        float a, b, kekka;
        printf("底辺は?"); scanf(" %f ", &a);
        printf("高さは?"); scanf(" %f ", &b);
        kekka = menseki();
        printf("面積は%fです. ¥n", kekka);
}
```

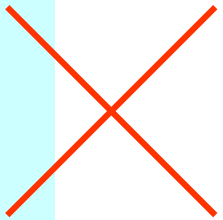
menseki()関数

menseki()関数内のローカル変数aとb

ローカル変数
同じ変数名a,bでも，関数が違うと，別の変数と見なされる
⇒赤色a,bと青色a,bは別の変数

main()関数

main()関数内のローカル変数aとb



関数の返却値

```
float menseki(float a, float b)
{
    float s;
    s = a * b / 2;
    return s;
}

int main(void)
{
    float x, y;
    printf("底辺は?"); scanf("%f", &x);
    printf("高さは?"); scanf("%f", &y);
    menseki(x, y);
    printf("計算はしたよ¥n");
}
```

menseki()関数の返却値が変数に代入されていない！

問題 これはコンパイルエラーになるか？

答え エラーにはならない．関数の返却値は代入されなくてもかまわない．

```
printf(・・・)
scanf(・・・)
main()
```

これらは全て関数
但し，一般に返却値は利用されない

void キーワード

返却値を持たない(関数値のない)関数の定義

```
void kansu(float a, float b)
{
        ....
        return;
}
```

voidは返却値が無いことを宣言

何も返さないので, return 式 ; とは書かない.
return文の省略も可能！

```
int main(void)
{
    int x;
    x = kansu(3, 4);
    printf("x=%d, %d", x, kansu(2,5));
}
```

void 宣言された関数の返却値を利用しようとするとエラー

引数としてvoidと書くと, 引数が無いことを宣言

```
int func(void)
{
    return 5;
}
int main(void)
{
    printf("値は%d", func());
}
```

main()関数

関数値は
int型
(返却値あり)

```
int func(void)
{
    return 5;
}

int main(void)
{
    printf("値は%d", func());
}
```

voidなので
引数は無い

void 宣言されていない関数ではreturn文は省略できないはず！！

main()関数は特殊な関数である

関数の返却値のまとめ

✓返却値がある関数でも，必要がなければ，呼び出し側でそれを使わなくても(代入しなくても)かまわない．

✓返却値が無い関数を定義する場合，あらかじめvoid宣言をする．この場合，return文で値を戻さない．また，return文自体も省略可．

✓void宣言された関数で返却値を利用しようとするとエラーになる．

```
void mul10(int a)
{
    a = a * 10;
    return;
}
int main(void)
{
    int x = 12;
    mul10(x);
    printf("x = %d", x);
}
```

引数を10倍する関数

```
x = 12
```

??

疑問

なぜこのmul10()関数は意図どおりに働かないか？

引数の値渡し

引数を10倍する関数

main()関数

```
int main(void)
{
    int x = 12;
    mul10(x);
    printf("x = %d", x);
}
```

実引数

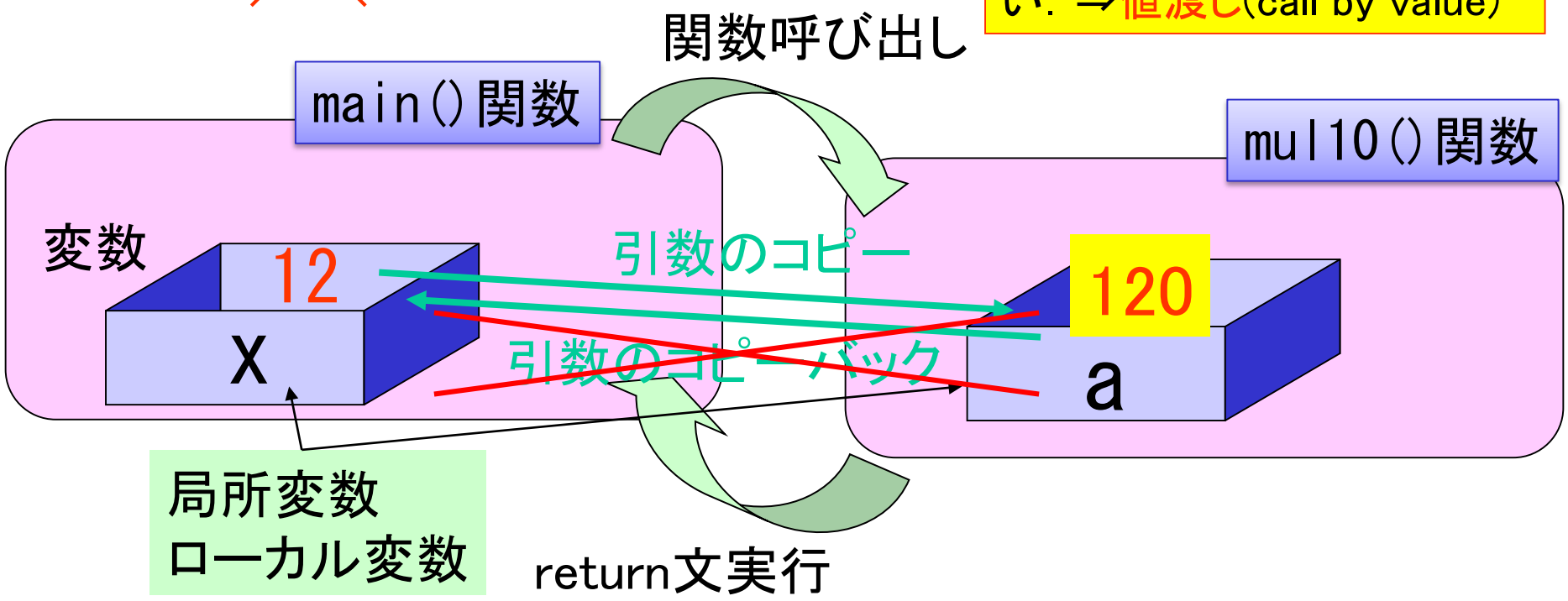
```
x = 12
```

mul10()関数

```
void mul10(int a)
{
    a = a * 10;
    return;
}
```

仮引数

原則 引数は関数からコピーバックされない．呼び出し側の変数(実引数)は変化しない．⇒値渡し(call by value)



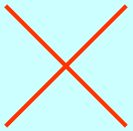
引数の値渡し

値渡し(基本)

- ✓ 実引数の内容が仮引数にコピーされる. コピーバックはされない(一方通行のコピー)
  - ⇒ 関数を呼び出した側の実引数の値が変化しない.
- ✓ 原則として全ての引数は値渡し

しかし, それだけでは不便なことも・・・

```
void InputXY(int x, int y)
{
    printf("xの値?"); scanf("%d", &x);
    printf("yの値?"); scanf("%d", &y);
}
int main(void)
{
    int a, b;
    InputXY(a, b);
    //・・・ a, bを用いた処理・・・
}
```



二つの値をユーザー入力してもらう関数

二つ以上の値を呼び出し側に戻す関数は作れない！
(一つだけならreturn文で返却値として戻せる)

引数の参照渡し

参照渡し(例外)

関数を呼び出した側の実引数の値が変化する.
以下の二つの場合のみ.

- ポインタ引数(2年で学習)
- 配列引数

参照渡しでは，仮引数は実引数の別名として働く
⇒ 呼び出し側の実引数の値を変化できる.

配列引数(参照渡し)を用いた関数

n個のデータの値を10倍する関数

```
#inc  e  <stdio.h>

void mul10(int data[], int n)
{
    int i;
    for (i = 0; i < n; i++)
    {  data[i] = data[i]*10;  }
}

int main(void)
{
    int i, aa[30], s;

    // 配列aaに30個のデータを入力する
    for (i = 0; i < 30; i++)
    {
        printf("%d番のデータ:", i);
        scanf("%d", &aa[i]);
    }

    mul10(aa, 30);
    printf("データを全て10倍すると");
    //データの表示(省略)
}
```

配列仮引数:かっこ[ ]のみ付ける. 要素数を空欄にした配列
→data[n]× data[0]× data×

理由:コンパイラから見たら普通の変数に見えるので×

理由:配列全体でなく要素を指定しているので×

データの個数(n個)を引数とする

理由:この関数を再利用することを想定.
データの個数はいつも30個とは限らないので, ループ回数を30回に固定したくない

配列引数は参照渡しであるため, 呼び出し側の配列の値が変化する!

関数内のローカル配列変数dataを変化すると, 呼び出し側の配列aaも変化

配列を実引数とした関数呼び出し
配列名のみ書く.([ ]は不要)

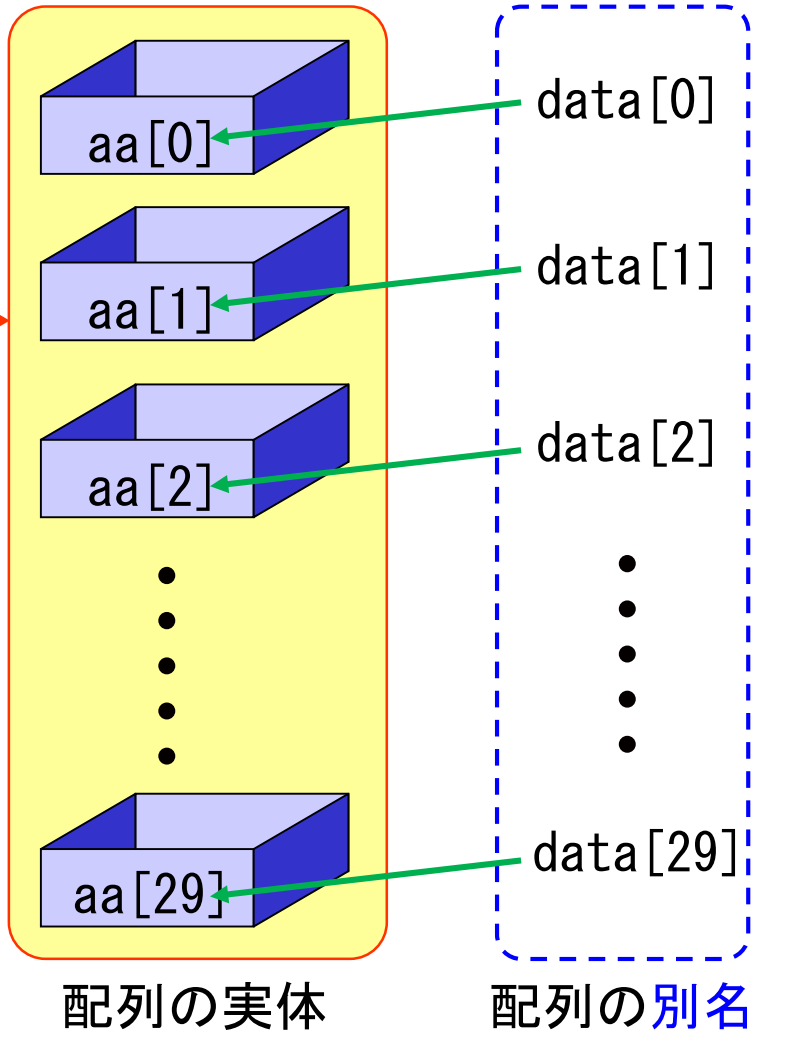
理由1: 配列全体を引数としているのでインデックス付けない
理由2: コンパイラはすでにaaが配列であることを知っている

有効なデータの個数は30個

配列引数(参照渡し)における実引数と仮引数

```
#include  <stdio.h>

void mul10(int data[], int n)
{
    int i;
    for (i = 0; i < n; i++)
    {  data[i] = data[i]*10;
}

int main(void)
{
    int i, aa[30], s;

    // 配列aaに30個のデータを入力する
    for (i = 0; i < 30; i++)
    {
        printf("%d番のデータ:", i);
        scanf("%d", &aa[i]);
    }

    mul10(aa,30);
    printf("データを全て10倍すると");
    //データの表示(省略)
}
```

参照渡しでは実引数の値は仮引数にコピーされない



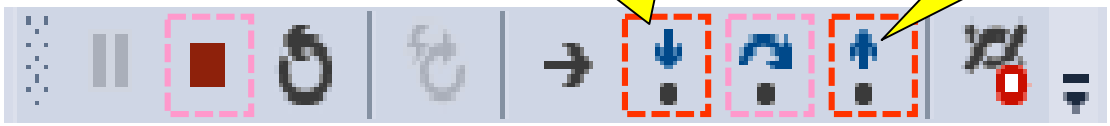
関数のデバッグ

[A] ツールバーでデバッグを始める手順
(1)　ツールボタンで右クリックして，デバッグを選ぶ
(2)　デバッグツールバーからボタンを選ぶ

ソース中の関数呼び出しの行でステップインを実行する

ステップインは関数内部に入って1行づつ実行する

ステップアウトは関数から飛び出す



(4)　(2)　(1)　(3)

(1)ステップオーバー	ソースの次の行を実行
(2) ステップイン	関数の中に入る
(3) ステップアウト	関数の中から出る
(4) デバッグの中止	デバッグをやめる

！注意！　ステップインをprintf()関数やscanf()関数等のライブラリ関数に対して実行するとデバッガが特殊な画面に遷移することがある

→ ステップアウトで関数から抜ける

(理由:ライブラリ関数はソースが存在しないため普通に1行ずつ実行することができず，機械語に対するデバッガの状態になるため)