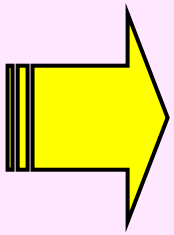
プログラミング入門
第10回講義

- 関数(その1)
  - 長いプログラムの問題
  - 関数とは
  - 関数の説明
  - 関数の例

 マークのあるサンプルプログラムは
`/home/course/prog0/public_html/2006/lec/lec10/`
下に置いてありますから、各自自分のディレクトリに
コピーして、コンパイル・実行してみてください

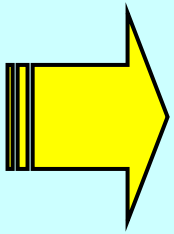
長いプログラムを分割

プログラムが長く、複雑になってくると



- 処理の流れや詳細を一覧して理解するのは大変
- 同じような処理を何回も書かなくてはならない
- 修正や改良が困難

長いプログラムを小さな部品に分割すると



- プログラムが理解しやすくなる
- 同じ部品を繰り返して使える
- 異なるプログラムにも、部品の再利用が可能
- 修正や改良がやりやすい

関数とは(p.162)

- 何度も使用できるプログラムのまとまり。
  与えられた情報を基に結果を計算する。
  数学の関数のようなもの

$$y=f(x)$$

- 関数には標準で用意されているものもある。
  これまでに出てきたものとしては:
  - 数学関数(詳しくはp.127参照)
    - sin, cos, sqrt(二乗根) 等
  - 標準入出力関数
    - scanf, printf 等

関数のサンプル

引数の2乗を返すプログラム

```
#include <stdio.h>

float nijou(float);

main()
{
  float x = 2.0, a1, a2, a3;
  a1 = nijou(3.0);      /*値*/
  a2 = nijou(x);        /*変数*/
  a3 = nijou(x * 2.0); /*式*/
  printf("a1:%f a2:%f a3:%f\n",
          a1, a2, a3);
}

float nijou(float x)
{
  float y;
  y = x * x;
  return y ;
}
```

実行例:

```
std1dc1{s1000000}1: ./a.out
a1:9.000000 a2:4.000000 a3:16.000000
std1dc1{s1000000}2:
```

ここが関数の本体(定義)
mainから3回呼ばれる

/home/course/prog0/public_html/2006/lec/source/lec10-1.c

関数宣言と引数：概略(p.166)

```
#include <stdio.h>

float nijou(float);

main()
{
   ..
   a2 = nijou(x);
   ..
}
float nijou(float x)
{
   float y;
   y = x * x;
   return y ;
}
```

プロトタイプ宣言:
Lec10-13

関数の呼び出し:Lec10-6

引数:Lec10-7,8

仮引数:Lec10-7,8

関数の定義:Lec10-11,12

戻り値:Lec10-9

標準的なプログラムの構成：
① インクルード
② マクロ定義
③ プロトタイプ
④ メイン
⑤ 関数

関数の呼び出し(p.172)

- 関数はmain又は他の関数の中からその関数の名前を書く事で呼び出すことが出来る。
- この時のデータのやり取りとして戻り値(もどりち)と引数(ひきすう)がある(次に説明する)

```
..
main()
{
   ..
   a2 = nijou(x);
   ..
}

float nijou(float x)
{
   ....
}
```

引数、仮引数(1)(p.180)

- 引数：関数を呼び出す側で使われる、関数に渡す情報
- 仮引数：関数内で用いられる、呼び出し側の変数に対応する変数
- 引数・仮引数は複数あっても良い。カンマで区切って型と名前を列挙する。
- 順序と型が同じなら、呼び出し側と関数で変数の名前が違っていてもかまわない。

```
main()
{
  float a,b;
   ...
  b = nijou(a, b);
   ...
}
```

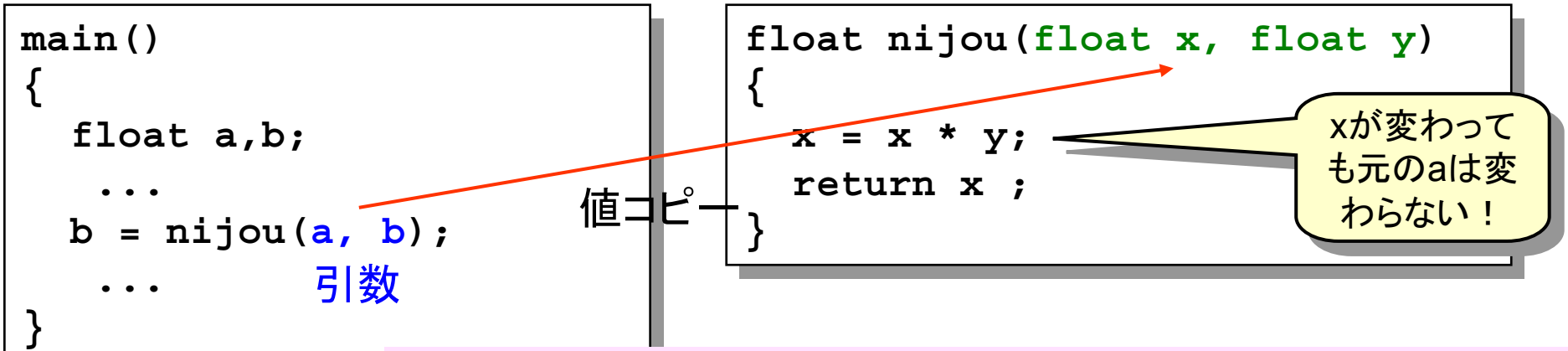
引数

```
float nijou(float x, float y)
{
  x = x * y;
  return x ;
}
```

仮引数

◆引数、仮引数(2)

- 情報は引数から仮引数にコピーされて関数内で使用される。
- 関数内で仮引数の値が変わっても引数の値は変わらない。
- 引数がない場合は「void」型とする（→サンプル）



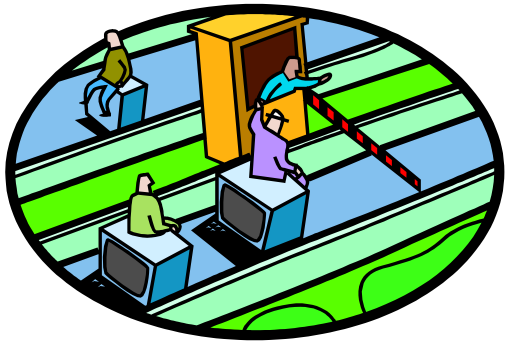
/home/course/prog0/public_html/2006/lec/source/lec10-2.c

関数の戻り値

- 関数の定義で指定した型（関数の型）の値を返す
- 戻す値を「return 文」で指定する
- 戻せる値は一つだけ
- 下例では、int型の変数yの値がfloat 型に変換（暗黙のキャスト）されて戻り値になる
- 値を戻さない場合は型を「void」にする（→lec10-2.c）

```
float nijou(int x)
{
   int y;
   y = x * x;
   return y;
}
```

この場合、暗黙のキャストが行われる



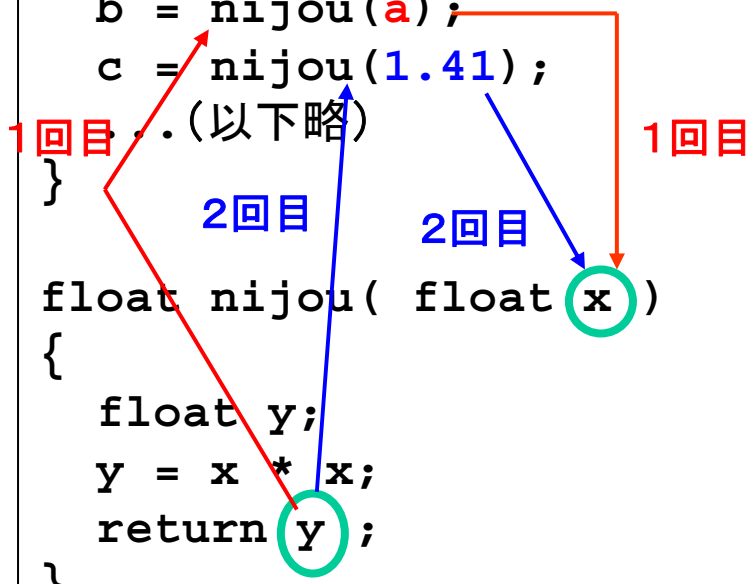
関数の注意1

```
#include <stdio.h>

float nijou(float);

main()
{
   float a = 1.73 , b , c ;
   b = nijou(a);
   c = nijou(1.41);
   ...(以下略)
}

float nijou( float x )
{
   float y;
   y = x * x;
   return y ;
}
```



仮引数には引数の値がコピーされる

一度目：xはa
（つまり1.73）
二度目：xは1.41

戻り値が関数の値になる

一度目：nijou(a)は約3.0
二度目：nijou(a)は約2.0

関数の定義(1)

- 関数の実際の中身を記述する
- 関数への受け渡しには戻り値と引数を使用する
- 関数中の書き方はmain()と同じ

```
float mult(float x, float y)
{
  float result;
  result = x * y;
  return result ;
}
```

関数定義(2)

- 関数内で宣言される変数と引数はローカル(局所)変数である
- 関数の戻り値を作るには
  `return 値;`
  のように書く
- (必要な場合)戻り値は関数の型に変換される

```
float mult(float x, float y)
{
  float result;
  result = x * y;
  return result;
}
```

[ローカル(局所)変数]
- これまで使ってきた普通の変数のこと
- 関数内で宣言された変数は、その関数内でのみ有効
- その関数が呼ばれる度に領域が確保される。
- 暗黙に初期化されない(初期化する必要がある)

プロトタイプ宣言(p.168)

- プログラムで使用する関数の名前や型、引数の順序と型を最初に宣言する
- 最後がセミコロン";"で終わっていなければならない
- プロトタイプ宣言を書かない書き方も可能（本によってはそういう形式の物もある）だが　→　最新のCの規格(C99)では必ず書くようになっているので、書くよう習慣づけよう！
- 引数の名前は書く必要がない（型のみを記述）
（書いても良いが、通常は書かないことの方が多い）

```
double func1(float, int);
```

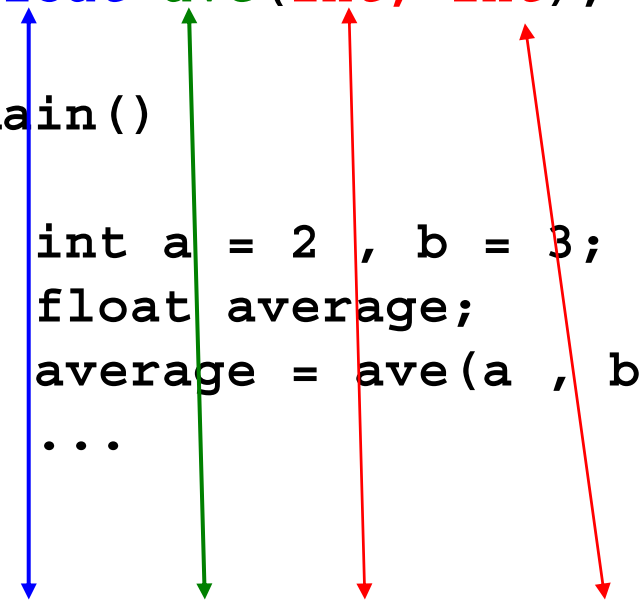
関数の注意3

```
#include <stdio.h>

float ave(int, int);

main()
{
   int a = 2 , b = 3;
   float average;
   average = ave(a , b);
   ...
}

float ave(int x, int y)
{
   float z;
   z = (x + y)/ 2.0;
   return z ;
}
```



プロトタイプと関数本体の宣言が一致していないといけない！

- 引数の型、個数、順番
  - プロトタイプ宣言には変数名は必要ない
- 戻り値の型
- 名前

例えば、戻り値の型、引数の型・数などが一致していないと「conflicting types for '関数名'」と言うようなメッセージが出てコンパイルエラーとなる

関数の動作おさらい(p.165)

```
#include <stdio.h>

float nijou(float);

main()
{
  float x = 2.0, a1, a2, a3;
  a1 = nijou(3.0);      /*値*/
  a2 = nijou(x);        /*変数*/
  a3 = nijou(x * 2.0); /*式*/
  printf("a1:%f a2:%f a3:%f¥n",
          a1, a2, a3);
}

float nijou(float x)
{
  float y;
  y = x * x;
  return y ;
}
```

1. mainの最初の実行文から開始
2. nijou(3.0)が評価される
3. 3.0が関数の仮引数xにコピーされ、関数nijouに実行が移る
4. 計算の結果、yに9.0が入り、「return y」で9.0がnijou関数の結果となる
5. nijou関数の結果がa1に代入される。
6. 次にnijou(x)が評価される
7. x(2.0)が関数のxにコピーされ、関数nijouに実行が移る
8. 計算の結果yに4.0が入り、「return y」で4.0がnijou関数の結果となる
9. nijou関数の結果がa2に代入される
10. a3も同様に計算される
11. printfでa1,a2,a3が表示される
12. 終了

標準入出力関数も関数

- `<stdio.h>`で定義される
- `printf("%d",i);`の場合、`"%d"`と`i`が引数になる。
- 戻り値の例: `scanf`
  - 整数型の戻り値を持つ
  - 正常に入力された変数の数を返す
  - ファイルの終端を読みこむと、`EOF(-1)`を返す

`/usr/include/stdio.h`
で定義されているマクロ

- なぜプロトタイプ宣言が必要ないのか？
  → それは`stdio.h`に記述されているから
  (`/usr/include/stdio.h` を参照のこと)

正確には/usr/include/stdio.hから更に
includeされている
/usr/include/iso/stdio_iso.h にある

scanfの戻り値を利用したプログラム

```
#include <stdio.h>
main()
{
  int data[100],result,i,n;
  for(i = 0 ; i < 100 ; i++){    /* data 入力 */
    result = scanf("%d",&data[i]);
    if (result == 1) continue;  /* scanf 成功 */
    else if (result == EOF) break;/* 入力終了 */
    else{ /* scanf 失敗 */
      printf("Input data error!\n");
      break;
    }
  }
  n = i;  /* i に正常に入力された個数が入っている */
  for(i = 0 ; i < n ; i++){    /* データ出力 */
    printf("data[%2d] : %d\n",i,data[i]);
  }
}
```

実行結果

```
std1dc1{s1000000}1: ./a.out
1 2 3 4 (改行)
Control+D
data[ 0] : 1
data[ 1] : 2
data[ 2] : 3
data[ 3] : 4
std1dc1{s1000000}2: ./a.out
1 2 3 k 4 5 (改行)
Input data error!
data[ 0] : 1
data[ 1] : 2
data[ 2] : 3
std1dc1{s1000000}3:
```

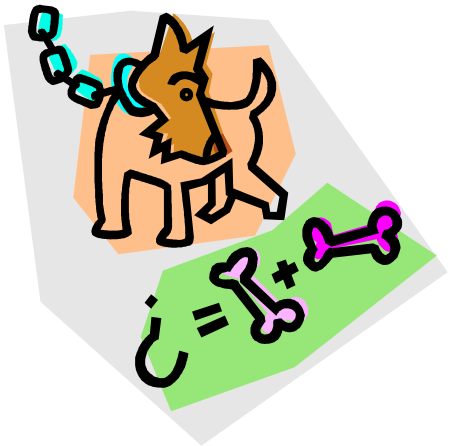
/home/course/prog0/public_html/2006/lec/source/lec10-3.c

例題—ニュートン法(p.178)

- 非線型方程式 $f(x)=0$ の解を求める方法
- 例題として、ニュートン法で

$$x^2 - a = 0$$

の解を求めることで $\sqrt{a}$ を求める



ニュートン法の原理

初期値x_0とし、そこでの接線とx軸との交点を順次求めて行く

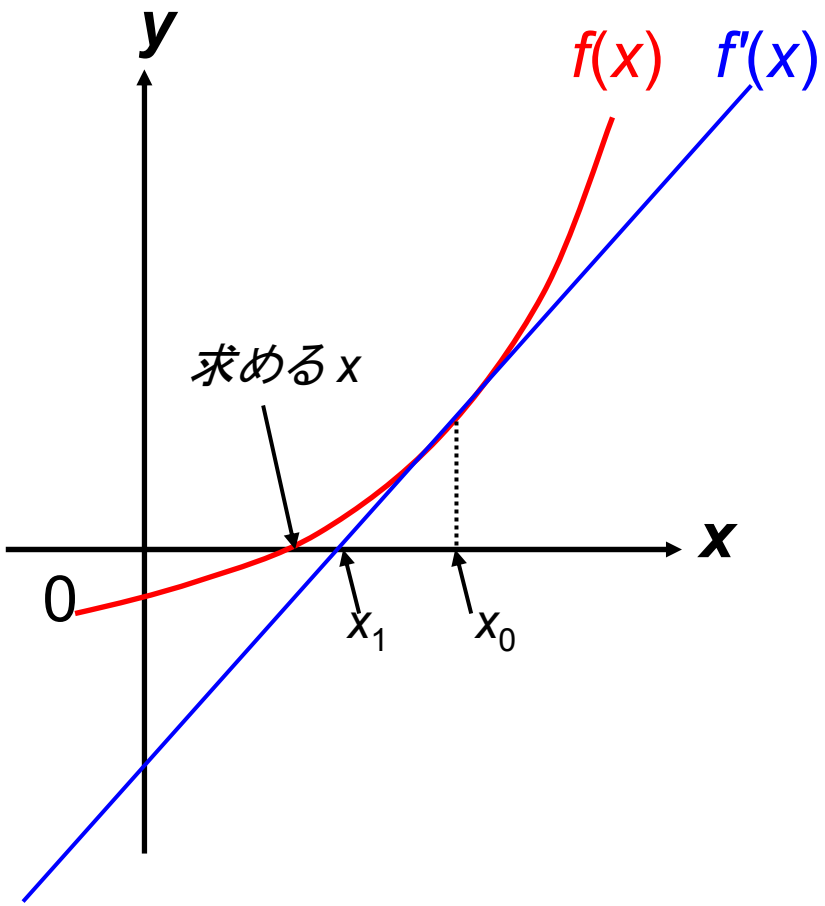
$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}$$

ここで、

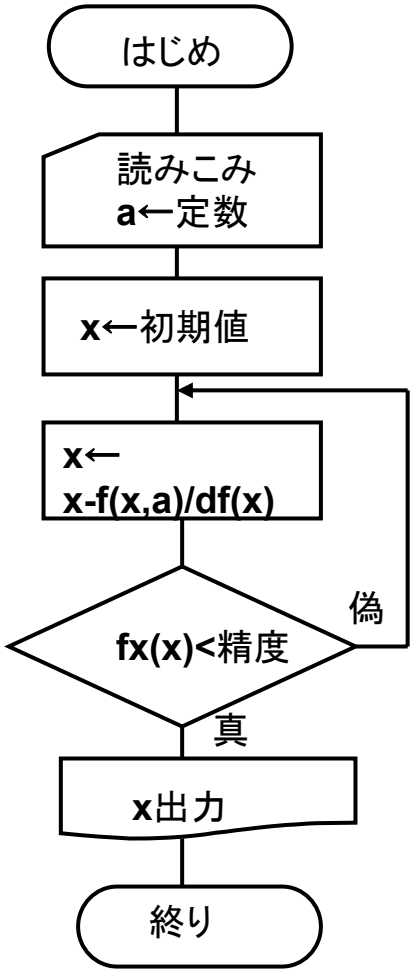
$$f'(x) = \frac{\mathrm{d}\, f(x)}{\mathrm{d}\, x}$$

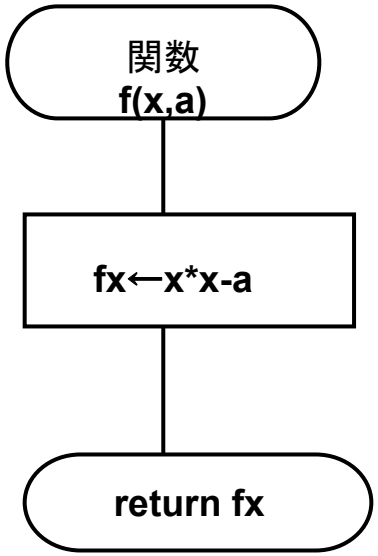
とする。

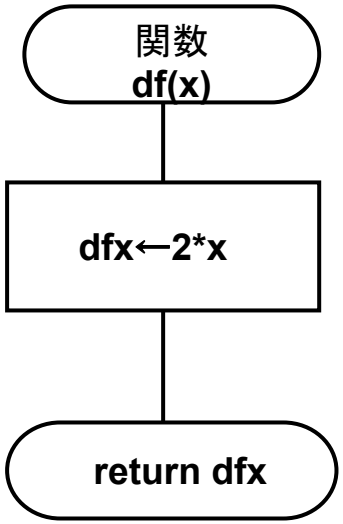
求めた$f(x_k)$の値が一定の値（精度）より小さければ終了



フローチャート







- 初期値は一般的に解より少し大きい値にするので、aとする。
- 精度は10^{-6}とする
- また、一般的には「fx(x)<精度」に関して、本当はfx(x)の絶対値を取る必要があるが、初期値を解より大きいaとしたので、負になることは考えない

プログラム

```
#include <stdio.h>
#define EPS 1.0e-6

float f(float, float);
float df(float);

main()
{
  float a, x, fx, dfx;
  printf("input a number:");
  scanf("%f",&a);
  x = a;
  printf("x\t\tfx\t\tdfx\n");
  while((fx = f(x,a)) > EPS ){
    dfx = df(x);
    x = x - fx/dfx;
    printf("%f\t%f\t%f\n",x,fx,dfx);
  }
  printf("sqrt(%f) = %f\n",a,x);
}
```

精度

初期値

タブ

これ以外の方法でも書ける

代入してから比較

```
float f(float x, float a)
{
  float fx;
  fx = x*x - a;
  return fx;
}

float df(float x)
{
  float dfx;
  dfx = 2.0*x;
  return dfx;
}
```

ループが違うだけ

/home/course/prog0/public_html/2006/lec/source/lec10-4{a,b,c}.c

実行結果

```
std1dc1{s1000000}1: ./a.out
input a number:2
x           fx          dfx
1.500000    2.000000    4.000000
1.416667    0.250000    3.000000
1.414216    0.006944    2.833333
1.414214    0.000006    2.828431
sqrt(2.000000) = 1.414214
std1dc1{s1000000}2:
```

平方根を3乗根に変えた場合

```
float f(float x, float a)
{
   float fx;
   fx = x*x*x - a;
   return fx;
}

float df(float x)
{
   float dfx;
   dfx = 3.0*x*x;
   return dfx;
}
```

- main関数での変更は最小限で済む
- 簡単な変更で、もっと複雑な方程式の解を求めることも可能

/home/course/prog0/public_html/2006/lec/source/lec10-5.c

この授業(講義・演習)で使用される関数

- <stdio.h>
  - printf,scanf, getchar(文字入力),
    fprintf(stderr...)(エラー出力)
- <stdlib.h>
  - exit(強制終了),rand(乱数発生)
- <math.h>
  - sqrt(二乗根),fabs(絶対値),pow(べき乗)