プログラミング入門
第11回講義

- 関数(その2)
  - 関数の復習
  - 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/source/`
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

```
#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 ;
}
```

関数のまとめ

- 引数が仮引数にコピーされる
- 関数内で仮引数の値に従って計算(処理)
- 戻り値が関数の値となる
- mainの前にプロトタイプ宣言
- mainの後ろに関数本体の宣言

```
#include <stdio.h>

float mult(int, int);

main()
{
   int a = 2 , b = 3;
   float f;
   f = mult(a , b);
   ...(以下略)
}

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

関数の注意1

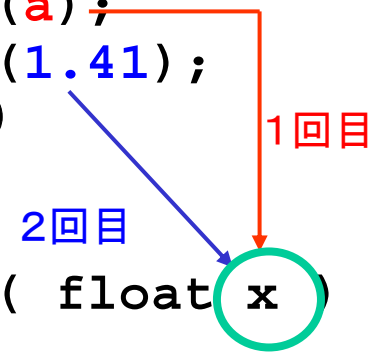
プロトタイプと関数本体の宣言の一致

- 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
- 関数名
- プロトタイプ宣言は通常、#includeとmainの間に書く

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

関数の注意2

```
#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

xの値は:
一度目:aの値
（つまり1.73）
二度目:1.41

引数がない場合はvoid型

int func(); と int func(void); というプロトタイプ宣言では意味が違うので、引数がない場合は必ずint func(void); のようにvoidを使用する。

関数の注意3

```
#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 ;
}
```

1回目

2回目

戻り値が関数の値になる

nijou(a)の値:
一度目:約3.0
二度目:約2.0

戻り値の型は宣言時の型

戻り値はfloat型

戻り値がない場合はvoid型

プロトタイプ宣言でfunc(int); のように戻り値を省略すると、戻り値はint型(int func(int);)だと自動的に解釈されるので、戻り値がない場合は必ずvoidを付けて、void func(int);のようにプロトタイプ宣言をする必要がある。

```
#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 ;
}
```

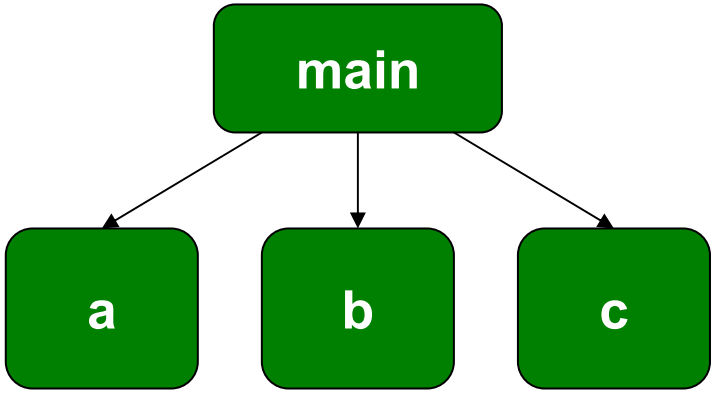
関数の動作おさらい

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

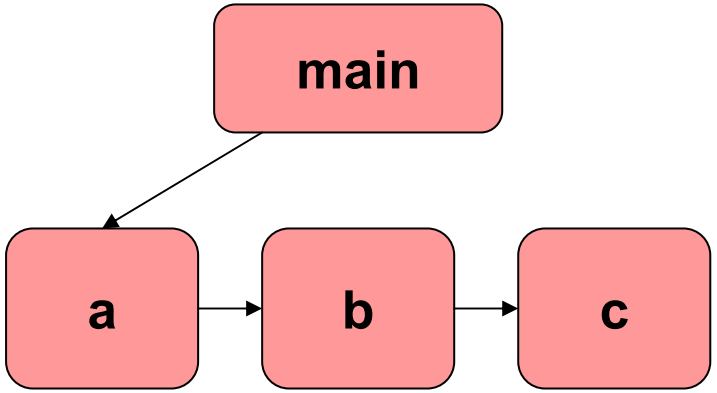
（教科書P165参照）

関数から関数を呼ぶ(p.184)

- 関数は必ずしもmainから呼ばれるとは限らない。
- 関数から呼ばれることもある。



mainから関数a,b,cを呼ぶ



mainが関数aを呼び、aがbを、bがcを順に呼んでいく

自動変数
＝普段使う変数(p.189)

- 関数内のみで通用
- 関数が呼ばれると生成され、リターンすると消える
- 別関数なら（通用範囲が違うので）同じ名前でも構わない

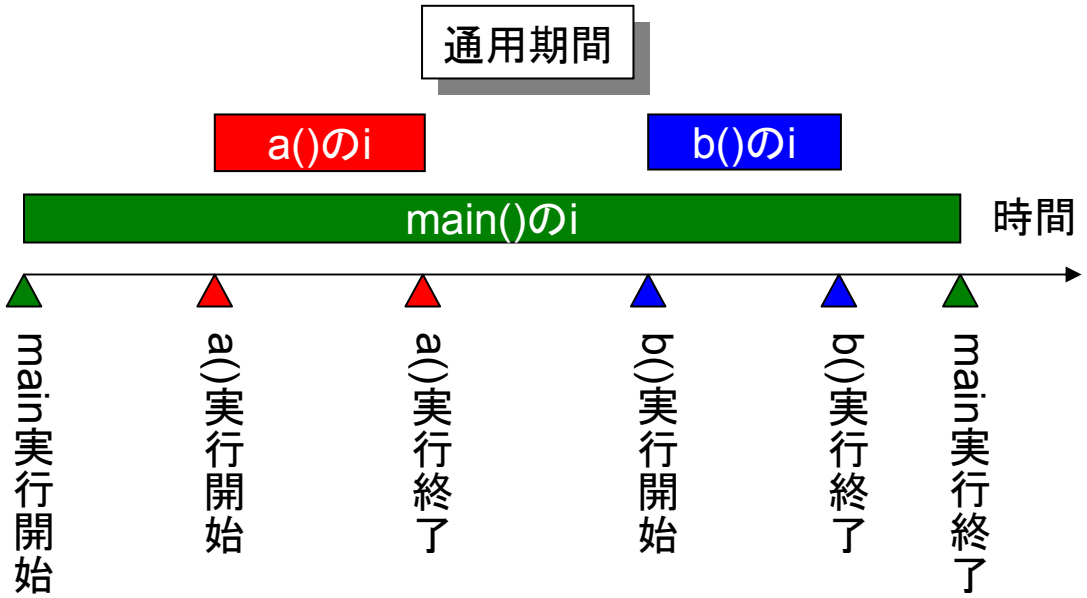
```c
#include <stdio.h>

void a(void);
void b(void);

main()
{
  int i = 1;

  a();
  b();
}

void a(void)
{
  int i = 2;
  printf("i : %d\n",i);
}

void b(void)
{
  int i = 3;
  printf("i : %d\n",i);
}
```

通用範囲

main()のi

a()のi

b()のi

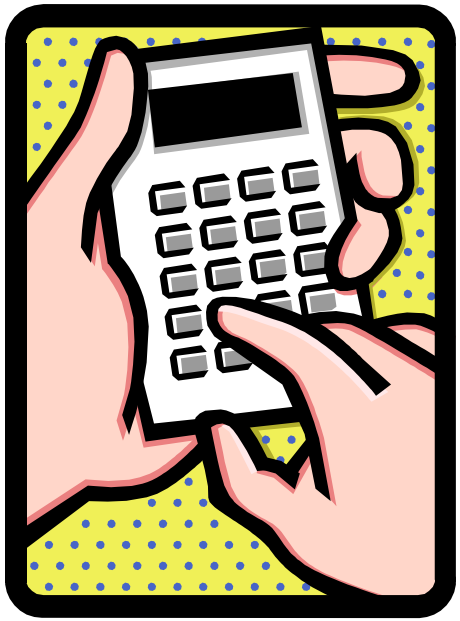


/home/course/prog0/public_html/2006/lec/source/lec11-2.c

関数の例1：桁数を知る(1)

- 入力された数字の桁数を返す関数を作る
  （例えば、「12345678」は8桁）
- 受け渡しの要件：1入力・1出力
  - 入力：数字 ⇨ int型
  - 出力：桁数 ⇨ int型
- 名前は桁数（digits）からdigitsとする。
  digitsのプロトタイプ宣言は以下のようになる。

```
int digits(int);
```



桁数を知る(2)

初期値として　桁数＝1，y＝10とする
xをyで割った答えが0以外の間ループ
　　yを10倍する(つまり今度は一桁上を見る)
　　桁数に1加える
ループここまで
関数の値として桁数をリターン

数が一桁の場合は
一度もループに入らない

ループ回数	y	x/y	keta
1	10	1234567	1
2	100	123456	2
3	1000	12345	3
4	10000	1234	4
5	100000	123	5
6	1000000	12	6
7	10000000	1	7
8	100000000	0	8

引数に12345678が渡された
時のループの様子

```
int digits(int x)
{
  int keta = 1 , y = 10;

  while((x / y) > 0){
    y *= 10;
    keta++;
  }
  return keta;
}
```

右表はこの
時点の値

一つ上の桁を見る準備

割った答えが0でない→その桁が存在
することを意味するので桁数に1加える

```c
#include<stdio.h>

int digits(int);

main()
{
  int i, j;

  scanf("%d",&i);
  j = digits(i);
  printf("%d の桁数は %d です\n",i,j);

}

int digits(int x)
{
  int keta = 1 , y = 10;

  while((x / y) > 0){
    y *= 10;
    keta++;
  }
  return keta;
}
```

プログラムと実行結果

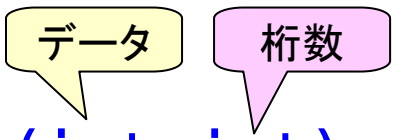
実行結果

```
std1dc1{s1000000}1: ./a.out
12345678
12345678 の桁数は 8 です
std1dc1{s1000000}2:
```

/home/course/prog0/public_html/2006/lec/source/lec11-3.c

関数の例2：ある桁の数を知る(1)

- 入力された数字の指定された桁の数を返す関数を作る（例えば、654321の下から2桁目は2）
- 受け渡しの要件：2入力・1出力
  - 二つの入力：
    - データ ⇨ int型
    - 桁数 ⇨ int型
  - 結果（その桁の数） ⇨ int型
- 名前はget_1_digitとした。digitsのプロトタイプ宣言は以下のようになる。




```
int get_1_digit(int,int);
```

ある桁の数を知る(2)

- 例えば654321の（下から）2桁目は2

  654321 % 100 = 21

  21 / 10 = 2

- つまり

  求める数　＝　(データ % $10^{桁数}$)／ $10^{(桁数-1)}$

- $10^{(桁数-1)}$ をどう作るか？

  ⇨ 10を（桁数−1）回掛け合わせる（ループにて）

◆プログラムと実行結果

```c
#include<stdio.h>

int get_1_digit(int, int);

main()
{
  int i, j = 2, result;

  scanf("%d",&i);
  result = get_1_digit(i, j);
  printf("%d の %d 桁目は %d です¥n",i,j,result);
}

int get_1_digit(int x, int pos)
{
  int i, j, k = 1;

  for(i = 1 ; i < pos ; i++){
    k *= 10;
  }
  j  =  x % (k * 10) / k;
  return j;
}
```

実行結果

```
std1dc1{s1000000}1: ./a.out
654321
654321 の 2 桁目は 2 です
std1dc1{s1000000}2:
```

10(桁数−1)のためのループ

(データ%10桁数)/10(桁数−1)の計算

/home/course/prog0/public_html/2006/lec/source/lec11-4.c

エラーチェック

- エラーチェックとは予想される間違いを検出すること
- このプログラムの場合、以下のような入力誤りが予想される
  - 負の数が入力される（例：－101）
  - 桁数が8を超える（例：101010101）
  - 2進数の入力のはずなのに0と1以外の数が含まれる（例：10123）
- エラーを検出した場合はエラーに応じた処理を行う（「エラー処理」と言う）
- エラーの重大さによって、処理が異なることがある（例えば軽度なエラーは処理を続行させ、重度なエラーは処理を中止するなど）
- このプログラムの場合
  - 負の数又は最大変換可能数11111111以上だった場合：「変換出来る範囲を越えています」と表示して再度データ入力からやり直す
  - 0と1以外の数が含まれる場合、「データが0か1ではありません」と表示してプログラムを強制終了（次ページ参照）させる
- エラーチェックをしっかりしておくとプログラムの誤動作を未然に防ぐことが出来る

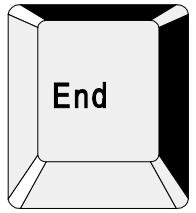
◆プログラムの強制終了

- プログラムの強制終了の方法
  - `<stdlib.h>` をインクルードする。
  - `exit(整数);` でどこからでもプログラムを強制終了させることが出来る。
  - 引数には自由な整数を渡すことが出来る。
  - プログラム実行後この数はシェル変数に渡される（例えば tcsh だと「`$?`」、csh だと「`$status`」というシェル変数に格納される）
  - このようにプログラムからシェル変数に値が渡る事で、
    - 値によってエラー理由を知ることが出来る
    - シェルスクリプトを使用して値によって動作を変える事が可能
  - 例えば以下のような非常に簡単なプログラム（exit(8);で強制終了）の実行終了後`$status`を見ると8という数字が入っていることが分かる。(csh)

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

main()
{
  exit(8);
}
```

実行結果(cshの場合)

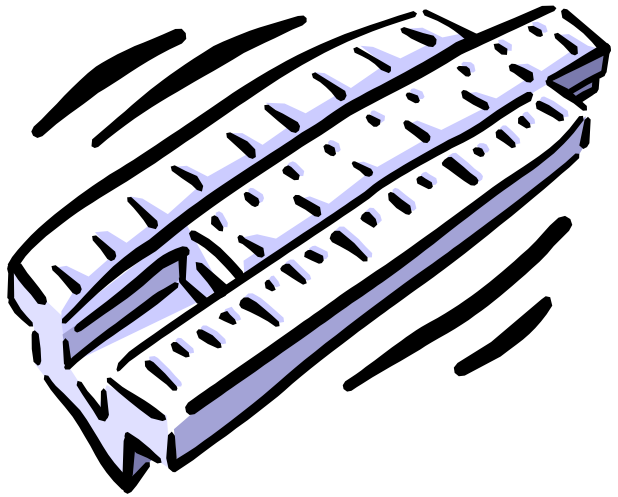
```
std1dc1{s1000000}1: ./a.out
std1dc1{s1000000}2: echo $status
8
std1dc1{s1000000}3:
```



`/home/course/prog0/public_html/2006/lec/source/lec11-5.c`

関数の例3:
2進数から10進数への変換(1)

- 入力された数字を2進数とみなして10進数に変換するプログラム(これまで作った関数を使う)
- 構成
  - **main**
  - **digits**(桁数を求める)
  - **get_1_digit** (数のうち一桁だけを取り出す)
- main()で行うこと:
  - 0が入力されるまで無限ループをして数を読む
  - 2進数から10進数への変換(次ページ)



2進数から10進数への変換(2)

- 各桁の重みと各桁の数をかけた物を加え合わす　例:101011(2進数) ⇨ 43

  $2^5*1+2^4*0+2^3*1+2^2*0+2^1*1+2^0*1 = 43$

- 処理:

桁数を求める(digitsを使用)
合計を0にする。
桁の重みの初期値を2^0(＝1)にする
桁数回ループして各桁について計算する
　　その桁の数を求める(get_1_digitを使用)
　　桁の重みと桁の数(0または1)を掛けて合計に足し込む
　　次のループに備えて次の桁の重みを計算する(重みを2倍する)
ループここまで

```
keta = digits(data)
total = 0;
exp = 1;
for(i = 1 ; i <= keta ; i++){
  n = get_1_digit(data, i);
  total += n * exp;
  exp *= 2;
}
```

ループの様子

n	exp	total
1	1	1
1	2	3
0	4	3
1	8	11
0	16	11
1	32	43

この場所での値

2進数変換プログラム(1)

```
#include<stdio.h>
#include<stdlib.h>
int digits(int);
int get_1_digit(int, int);
main()
{
  int data, keta, i, n, exp, total;
  while(1){
    /* データ読み込みとチェック */
    printf("8桁以下の2進数を入力 ==> ");
    scanf("%d",&data);
    if (data == 0) exit(0);
    if (data > 11111111 || data < 0){
      printf("変換出来る範囲を越えています\n");
      continue;
    }
    /* 桁数計算 */
    keta = digits(data);
    /* 10進数に変換 */
    total = 0;
    exp = 1;
    for(i = 1 ; i <= keta ; i++){
      n = get_1_digit(data, i);
      if((n != 0) && (n != 1)){ /*データチェック*/
        printf("データが0か1ではありません\n");
        exit(8);
      }
      total += n * exp;
      exp *= 2;
    }
    printf("2進 : %d  -> 10進 : %d\n",data,total);
  }
}
```

右に続く⇨

⇨左から続く

```
int digits(int x)
{
  int keta = 1 , k = 10;

  while((x / k) > 0){
    k *= 10;
    keta++;
  }
  return keta;
}

int get_1_digit(int x, int pos)
{
  int i, j, k = 1;

  for(i = 1 ; i < pos ; i++){
    k *= 10;
  }
  j  =  x % (k * 10) / k;
  return j;
}
```

/home/course/prog0/public_html/2006/lec/source/lec11-6.c

実行例

```
std1dc1{s1000000}1: ./a.out
8桁以下の2進数を入力 ==> 101011
2進 : 101011 -> 10進 : 43
8桁以下の2進数を入力 ==> 101010101
変換出来る範囲を越えています
8桁以下の2進数を入力 ==> -101
変換出来る範囲を越えています
8桁以下の2進数を入力 ==> 10123
データが0か1ではありません
std1dc1{s1000000}2: echo $status
8
std1dc1{s1000000}3:
```

2進数変換プログラム(2)

```c
#include<stdio.h>
#include<stdlib.h>

int digits(int);
int get_1_digit(int, int);
int b2d(int);

main()
{
  int data, result;
  while(1){
    /* データ読み込みとチェック */
    printf("8桁以下の2進数を入力 ==> ");
    scanf("%d",&data);
    if (data == 0) exit(0);
    if (data > 11111111 || data < 0){
      printf("変換出来る範囲を越えています\n");
      continue;
    }
    result = b2d(data);
    printf("2進 : %d  -> 10進 : %d\n",data,result);
  }
}
```

右に続く⇨

⇨左から続く

```c
int digits(int x)
{
  /* 同じなので省略 */
}

int get_1_digit(int x, int pos)
{
  /* 同じなので省略 */
}

int b2d(int x)
{
  int i, n, keta, total = 0, exp = 1;
  keta = digits(x);

  for(i = 1 ; i <= keta ; i++){
    n = get_1_digit(x, i);
    if((n != 0) && (n != 1)){
      printf("データが0か1ではありません\n");
      exit(8);
    }
    total += n * exp;
    exp *= 2;
  }
  return total;
}
```

exitではなくreturn -1とすることで不正データの場合も再入力が可能となる。mainをどう変更すれば良いだろう？→ lec11-7b.c

改良バージョン

-関数の中から関数を呼ぶようにすると、mainをもっと簡単に出来る。この例では変換部分をb2dとして独立させ、mainからb2dを呼び、b2dからdigitsとget_1_digitを呼ぶ。（但しあまり細かく関数を分け過ぎると、分かり難くなる）

更に後期に習うテクニックを駆使したlec11-8.cも用意したので興味のある人は見てみると良いだろう

/home/course/prog0/public_html/2006/lec/source/lec11-7.c

◆ 補足：10進数から2進数への変換

なお、10進数から2進数への変換は以下のようなプログラムによって行う事が出来る。

配列を使用

```
#include<stdio.h>
main()
{
  unsigned int data;
  int bin[32], i;

  printf("10進数を入力 ==> ");
  scanf("%u",&data);
  for(i = 0; i < 32; i++){
    bin[31 - i] = data % 2;
    data /= 2;
  }
  for(i = 0; i < 32; i++){
    printf("%1d",bin[i]);
  }
  printf("¥n");
}
```

シフト演算子を使用

```
#include<stdio.h>

main()
{
  unsigned int data;
  int i;

  printf("10進数を入力 ==> ");
  scanf("%u",&data);

  for(i = 0; i < 32; i++){
    printf("%1d",(data >> (31 - i)) & 1);
  }
  printf("¥n");
}
```

/home/course/prog0/public_html/2006/lec/source/lec11-9{a,b,c}.c

【補足】漢字コードについて(1)

- アルファベットはASCII(アスキーコード)で表される
  - 0x00～0x7Fの１２８種類
  - １バイト（最大２５６通り）で済む
- 漢字（かなも含む）
  - 第一水準３４８９字
  - 第二水準３３８８字
  - １バイトでは足りない！

【補足】漢字コードについて(2)

- 漢字コード
  - 漢字を2バイトで表現する。以下の3つが良く使われる。
- JISコード
  - アルファベットとの共存が出来ないので、特殊なコード配列で「漢字の始まり」と「終わり」を示す
- シフトJISコード(MS漢字)
  - アルファベットとの共存可（EUCも）
  - PCで使用
- EUCコード(Extended Unix Code)
  - UNIX・汎用計算機等で使用