プログラミング論I
(8) 関数を作る

電子情報工学専攻　日浦 慎作

ログインしておいてください

中間試験日程

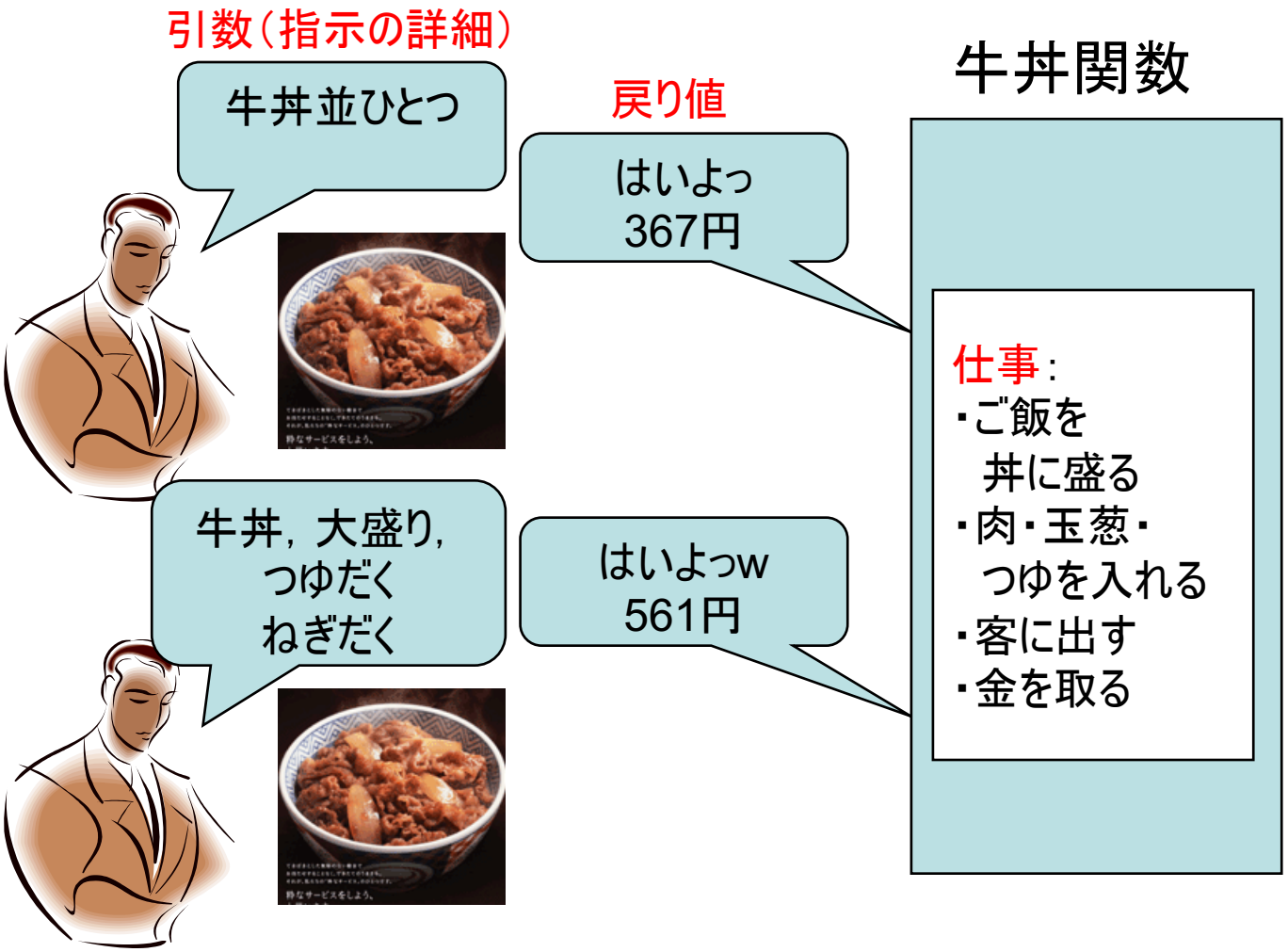
- 来週　11月26日　1限
- A棟A101教室（大きい教室）

- 前中先生の電気回路と連続

再掲

関数とは？

- 一連の仕事をひとつにまとめたもの



- 簡単な指示で複雑な仕事をさせることができる
  - 数学の「関数」は計算結果を返すだけだが，プログラミングでは計算以外の仕事もできる
- 数値を1つ，返すことができる（戻り値）
- 指示を出すほうは作業方法を知らなくてもよい
- あちこちから何度でも同じ仕事を依頼できる
- 少し違う要求を引数で指示することができ，それに応えることが出来る

関数の作り方・使い方(1)

```
#include <stdio.h>

int square(int x);

int main(void) {
    int y;

    y = square(5);
    printf("result is %d¥n", y);
    return 0;
}

int square(int x) {
    return x * x;
}
```

関数のプロトタイプ宣言（関数の概要）

square ： 関数名
（自由に名付けて良い）

関数呼び出し

関数の定義
（関数そのもの）

整数を2乗する関数

関数の作り方・使い方(2)

```
#include <stdio.h>

int square(int x);

int main(void) {
      int y;

      y = square(5);
      printf("result is %d¥n", y);
      return 0;
}

int square(int x) {
      return x * x;
}
```

戻り値の型は int である

引数の型は int である

引数の値 5 が，関数の引数変数 x に代入される

return の後ろの数式の値が，呼び出し側へ，戻り値として返される

関数の作り方・使い方まとめ

- 最初にプロトタイプ宣言を行う
  - 実際に呼び出しが行われる行よりも上に置く
    - コンパイラに，関数の概要（呼び出し方）を教える役割を持つ
  - プロトタイプ宣言は，基本的に，関数の外側に置く
    - `main()` も関数なので，`main` の前に
- 関数そのものを定義する
  - 引数の値（実引数）は，引数変数（仮引数）で受け取る
    - 前ページの例では，実引数は`5`，仮引数は`x`
  - 関数の書き方は，mainと同じ
    - 関数内で変数の定義などができる
  - `return` で値を戻すことができる（戻り値）．

関数の「いいところ」

- 関数の中身を知らなくていい
  - 「どういう結果を生じるか」のみ知っていればOK
  - プログラムが見やすく，分かりやすくなる
- 同じ処理を何度も書かなくて良い
  - 同じ処理をするたびに，呼び出せばよい
- 処理を「一人立ち」させることが出来る
  - 関数を独立に作成・デバッグし，
    完成したものとして提供できる
  - 他人様にも簡単に使っていただける

関数の「精神」

- 良い「ブラックボックス化」とは？
  - 処理は複雑でも，提供する機能は分かりやすい
    - 例えば，sin( ) とか sqrt( ) など．説明が簡単
  - 確実に動作する
    - バグ（不具合）がよく除かれている
  - 望んだ結果以外の作用（＝副作用）がない
    - 他の変数の値が勝手に変わるなど
- 良いプログラムとは？
  - だらだらと長く書くのではなく，関数を用いて，こまめに，短くまとめる．

関数の「仕様」

- どんな引数を取るのか
  - 引数の数，型 (int, double, ..)
  - 引数に求められる条件（負の数はダメ，など）
- どんな戻り値を返すのか
  - 戻り値の有無，戻り値の型
  - 戻り値の意味（1なら成功，0なら失敗　などもある）
- どんな機能を持つのか
  - どのような引数を与えると，どのように動作し，どのような結果（画面入出力，戻り値など）をもたらすのか

関数の使い方

- どこからでも呼び出すことが出来る

```
y = square(i);
printf("val = %d¥n", square(j) );
z = square( square(2) );
```

普通の式
関数の引数で
（同上）

- 戻り値を利用しなくてもOK
  - 戻り値のない関数もある（void型関数：後で説明）
- 引数の型に注意
  - int を引数として受け取る関数に double の値を渡すと，切り捨てられてしまう
  - float を返す関数の値を int の変数に代入すると，切り捨てられてしまう
    など

引数・変数の使い方

呼び出し側（一部）

```
        double a, b = 1.2;
        int y = 3;

        a = beki(b, y + 1);
```

呼び出し側の引数には，実際の値でも，変数でも，数式でも書くことができる

実数値 1.2

整数値 4

関数定義

```
double beki(double x, int n) {
        int i;
        double b = 1.0;

        for(i = 1; i <= n; i++) {
                b *= x;
        }
        return b;
}
```

関数内部で変数を定義できる（有効範囲はこの関数内だけ）

仮引数名や，関数内で定義する変数名は，他の関数と重複しても良い

スコープ（変数の有効範囲）

```
int main(void) {
        double a, b = 1.2;
        int y = 3;

        a = beki(b, y + 1);
}
```

main 関数内の変数
a, b, y の有効範囲

```
double beki(double x, int n) {
        int i;
        double b = 1.0;

        for(i = 1; i <= n; i++) {
                b *= x;
        }
        return b;
}
```

beki 関数内の変数
x, n, i, b の有効範囲

スコープとは？

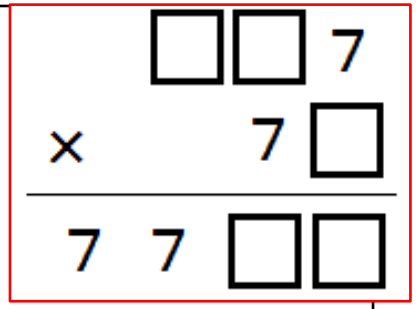
- 変数が使える，見える範囲のことをスコープという
- C言語では，スコープは関数ごと
  - 関数内で定義した変数は，その関数内しか見えない
  - 関数の仮引数は，その関数内でしか見えない
- 関数が異なれば，変数名は同じでもよい
  - 変数名が同じでも，別の変数として扱われる
    - 家が違う．櫻井翔と哀川翔は別人，みたいなもの？
- ブロック内もスコープになる
  - ただしブロックを包含するブロック（外側のブロック）の変数にはアクセスできる
  - 名前が重複した場合，外側の変数は見えなくなる

```
#include <stdio.h>

int main(void) {
  int i, j, k;

  for(i = 1; i <= 9; i++) { //最上位桁は0はダメ
    for(j = 0; j <= 9; j++) { //2桁目以降は0でも良い
      for(k = 0; k <= 9; k++) {
        int l, m, n;

        l = 100 * i + 10 * j + 7; //1行目の値
        m = 70 + k; //2行目の値
        n = l * m; // 積を求める
        if(n / 100 == 77) { //積が 77xx かどうか確認
          printf("%d * %d = %d¥n", l, m, n);
        }
      }
    }
  }
  return 0;
}
```



変数 l, m, n の有効範囲
（ブロック内で定義した変数は，
そのブロック内がスコープとなる）

戻り値のない関数

```
void dispGCP(int i) {
        if(i == 0) {
                printf(“グー”);
        }
        else if(i == 1) {
                printf(“チョキ”);
        }
        else if(i == 2) {
                printf(“パー”);
        }
        else {
                printf(“エラー”);
        }

        return;          省略可
}
```

引数の値に応じてグーチョキパーを表示する関数（値を返す必要がない）

引数のない関数

```
int enterInt(void) {
        int i;

        scanf(“%d”, &i);

        return i;
}
```

キーボードから整数を1つ入力させ，その値を返す関数（引数をうける必要がない）

戻り値や引数がない場合， void を使う

これまでに学んだ標準関数では，例えば，
乱数を初期化する srand() は戻り値がなく，
乱数を返す rand() は引数がない

前ページの2つの関数の，呼び出し側では・・

```
#include <stdio.h>

void dispGCP(int i);
int enterInt(void);

int main(void) {
      int i;

      i = enterInt();
      dispGCP(i);

      return 0;
}
```

前のページの関数を呼び出す側
（main関数）

プロトタイプ宣言は，関数定義の1行目と同じにする

enterIntは引数を受け取らないので，( )の間には何も書かない

- void と書くのもダメ
- ( )は省略できない

dispGCPは戻り値を戻さないので，

a = dispGCP(i);

のような代入はしない．

引数の「値渡し」について

```
#include <stdio.h>

void minus(int i);

int main(void) {
        int i = 5;

        minus(i);
        printf("main : %d¥n", i);

        return 0;
}

void minus(int i) {
        i = -i;
        printf("minus : %d¥n", i);
}
```

値 5 が代入される

引数 i の正負を反転する関数（のつもり・・）

```
minus : -5
main : 5
```

実行結果

- main関数中の i と，minus関数中の i は別もの
- minus関数の i の値を変更しても，main関数の i の値には影響しない

なぜ？

- 関数呼び出しのときには，引数の値（実際の数値，このプログラムでは 5）が，仮引数 i に代入される
- これを「値渡し」という

関数の注意事項

- 関数内で変数の値を変更しても，
  呼び出し側には影響しない
  - 副作用を防ぐための重要な仕組みの1つ
- 呼び出し側に値を返す，４つの方法
  - 戻り値で情報を戻す
  - ポインタを使う（この講義では扱わない．２年前期）
    - `scanf` で，変数に `&` をつけなければならないのはこのため．
  - 配列を使う（次回，次々回）
  - グローバル変数を使う（関数の外側で定義する変数）

練習：この円周率を求めるプログラムでは，ユークリッド距離
$\sqrt{x^2 + y^2}$
を求める部分が2箇所あるので，
これを関数を用いて簡単にせよ．

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

int main(void) {
  int n;
  double x = 0, y = 1, cx, cy, dist, pi;

  for(n = 8; n < 10000000; n*=2) {
    cx = (x + 1) / 2;
    cy = y / 2;
    dist = sqrt (cx * cx + cy * cy);
    x = cx / dist;
    y = cy / dist;
    pi = (sqrt((x - 1) * (x - 1) + y * y) * n) / 2;
    printf("%.12f¥n", pi);
  }
  return 0;
}
```

解答例

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

double length(double x, double y);                    プロトタイプ宣言

int main(void) {
  int n;
  double x = 0, y = 1, cx, cy, dist, pi;

  for(n = 8; n < 10000000; n*=2) {
    cx = (x + 1) / 2;
    cy = y / 2;
    dist = length(cx, cy);
    x = cx / dist;
    y = cy / dist;
    pi = (length(x – 1, y) * n) / 2;
    printf("%.12f¥n", pi);
  }
  return 0;
}

double length(double x, double y) {
        return sqrt(x * x + y * y);          関数 length の定義
}
```