関数

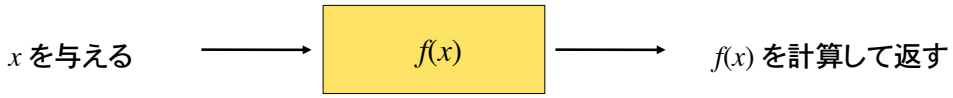
C 言語では、関数を組み合わせてプログラムを構成する

pritnf(), scanf() などは、処理系があらかじめ備えている標準ライブラリ関数

math.h で定義されている算術関数も標準ライブラリ関数の 1 つ

データを与えて、それに基づき何か動作をおこなうものが関数。

数学の関数 $y = f(x)$ のイメージ



$f(x)$ はある意味ブラックボックス。内部でどのような計算をするかは具体的な処理内容に依存。数学の場合は、引き数は実数、返却値も実数の場合がほとんど。

C 言語の関数例

```
#include <math.h>
main()
{
  double x, y;
  scanf("%lf", &x);
  y = sin(x);
  printf("%f\n",y);
}
```

算術ライブラリ関数 sin() を x を引数として呼び出して(call)、変数 y に sin(x) の返却値(戻り値)を代入。

```
main()
{
  int code;
  code = getchar();
  printf("%c\n", code);
}
```

ライブラリ関数 getchar() を引数無しで呼び出し、変数 code に getchar() の返却値を代入。

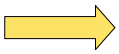
標準ライブラリ関数およびユーザー関数を組み合わせて、プログラムを作成するのが C 言語の特徴。

自分で関数を定義する

C 言語を含む多くの言語では、ユーザが自分の関数を定義してプログラム中で使用出来る（ユーザー関数）。

長いプログラムは、個々の個別処理作業を行う部品の集合として捉えるとプログラムしやすい。必要に応じて、メインプログラムから呼び出す部品（部分）をサブルーチンという。

```
main()
{
  /* 長い main 文 */
  int i;
  ...
  ...
  ...
}
```



```
main()
{
  data_yomikomi();
  data_syori();
  data_syuturyoku();
}
```

C 言語の関数（ユーザー関数を含む）はサブルーチンと似ているが返却値（戻り値）を持ちうる点が異なる。

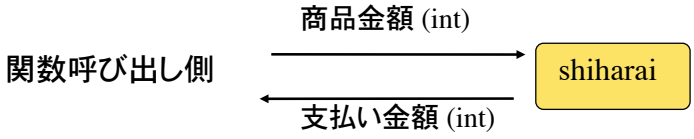
具体例

商品の金額を入力し、消費税込みの支払い金額を計算するプログラム

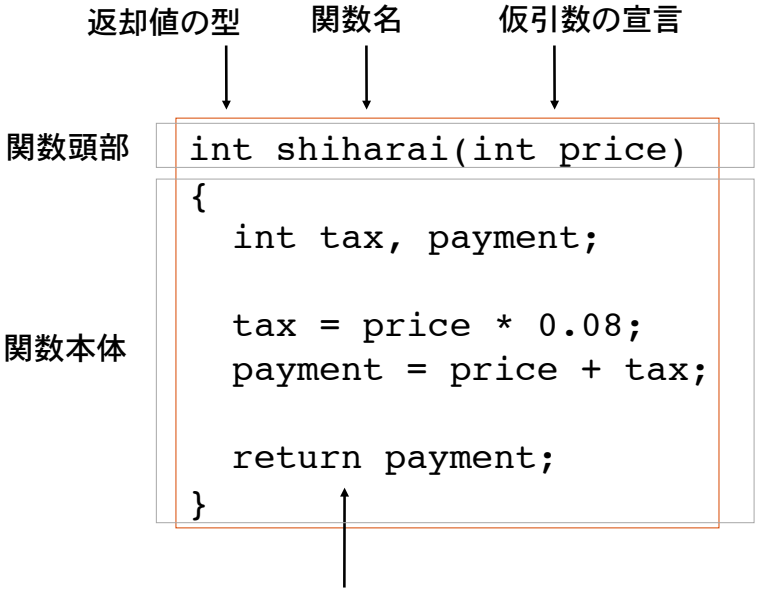
```
main()
{
  int price, tax, payment;

  scanf("%d", &price);

  tax = price*0.05;
  payment = price + tax;
  printf("%d\n", payment);
}
```

ユーザ関数を定義しないで、全て main文として書いたプログラム。

商品金額を引数として受けとり、支払い金額を返却値として返すユーザ関数 shiharai を定義してみる。



ユーザー関数の定義



関数頭部および関数本体に現れる引数 price は、具体的な値が格納されていない仮引数。

関数本体では、普通にプログラムを書く(変数宣言など)。

返却値を持つ関数の場合、返却値を return 文により、関数呼び出し元に返す必要がある。

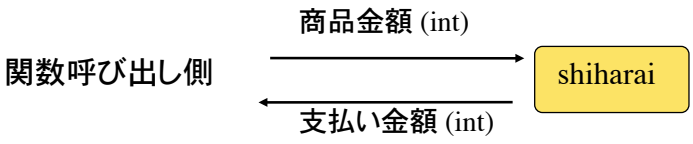
関数本体で計算した payment を、関数 shiharai の返却値として呼び出し元に返す。return 文は、後ろに書かれた式を評価し、その値を関数の返却値として呼び出し側へ返す。

1) 返却値をもつ関数は、返却値の型を明示しなければならない。

2) 関数名は自由に付けてよい(C 言語のキーワードを除く)。

3) 関数の呼び出し側から関数へデータを受け渡すには引数を用いる。
呼び出し側で指定する引数を実引数、関数側で指定する引数を仮引数と呼ぶ。

```
int shiharai(int price)
{
  int tax, payment;

  tax = price * 0.08;
  payment = price + tax;

  return payment;
}
```



関数が呼び出されるまでは、関数側の引き数には具体的なデータ(値)は格納されていないので仮引数と呼ぶ。左の例では price が仮引き数。

ユーザ関数で宣言された変数は、ユーザ関数の内部でのみ有効(局所変数)。

ユーザー関数の呼び出し

```
int shiharai(int);

main()
{
  int price, payment;

  scanf("%d", &price);

  payment = shiharai(price);
  printf("%d\n", payment);
}

int shiharai(int price)
{
  int tax, payment;
  tax = price * 0.08;
  payment = price + tax;
  return payment;
}
```

関数返却値の型、関数名、引数の型を指定するプロトタイプ宣言

関数 shiharai の呼出し (function call) 。この price は実引数（具体的な値が格納されている）。
返却値を変数 payment に代入。

関数呼出に際して、仮引数 price は実引数 price の値で初期化される。

main 文の変数 price, payment と、関数定義部の変数 price, payment は別物であることに注意！

関数のプロトタイプ宣言

ユーザが自前の関数を定義するとき、一般に関数プロトタイプの宣言が必要。

関数の名前、関数が返却するデータの型、および引き数の型と個数、を関数プロトタイプと呼ぶ。（プロトタイプ prototype＝原型、模範、手本の意）

```
int fn1(int);

double fn3(double, double);

void fn4(char);
```

int 型の引数 1 つを受け取り int を返す関数 fn1

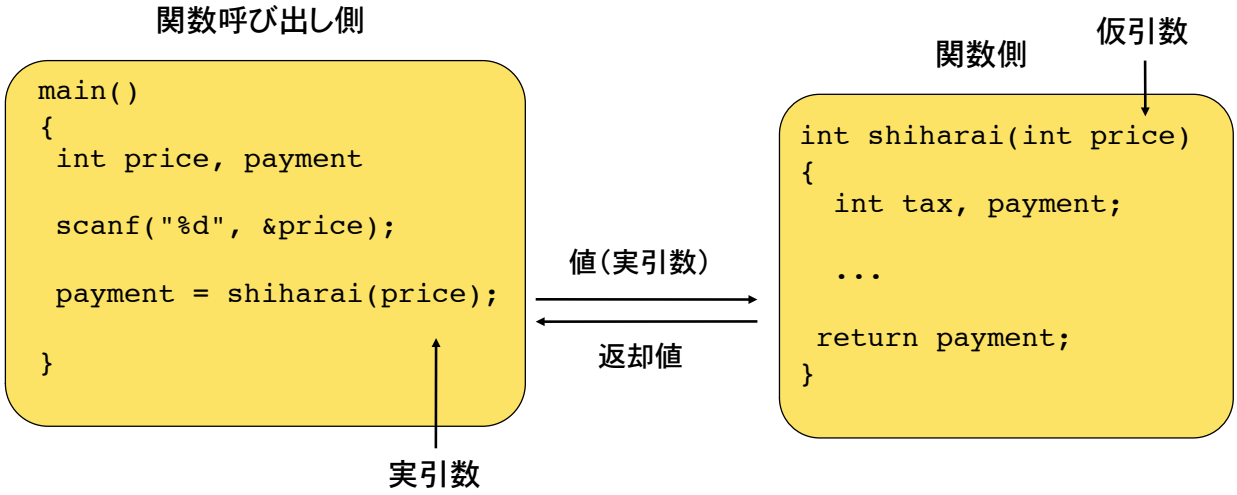
double 型の引数 2 つを受け取り、double を返す関数 fn3

char 型の引数 1 つを受け取り、返却値を持たない関数 fn4

関数プロトタイプ宣言によって、処理系（コンパイラ）は、関数が正しく呼び出されているかのチェックを行う。

関数プロトタイプ宣言をしないと、定義した関数の返却値の型は暗黙のうちに int と解釈される。（思わぬ結果を招く場合があるので注意）

関数呼び出しの値渡し



関数 shiharai 側の仮引数は、関数呼び出し側の実引数で初期化される(実引数の値が仮引数にコピーされる)。これを値渡し(call by value)という。

関数呼び出し側で宣言された変数と、関数側で宣言された変数は互いに独立。
同じ変数名であっても別の変数として取り扱われる(局所変数)。

局所変数

関数定義部(関数頭部と関数本体)で宣言された変数は、関数(もしくはブロック)の中でのみ有効。これを局所変数(ローカル変数)と呼ぶ。

```
main()
{
  int x, y, z;
  ...
}

int function_1(int x)
{
  int i, y;
  ...
}
```

変数 x, y, z は main 文の中でのみ有効な局所変数。
function_1 で宣言されている変数 i と y は main文からは見えない。

変数 x, y, i はこのブロックの中でのみ有効。
main 文で宣言されている変数とは独立無関係。

変数が有効な範囲をスコープ (scope) という。局所変数のスコープは、それが宣言された関数本体に限られる。これをブロックスコープという。

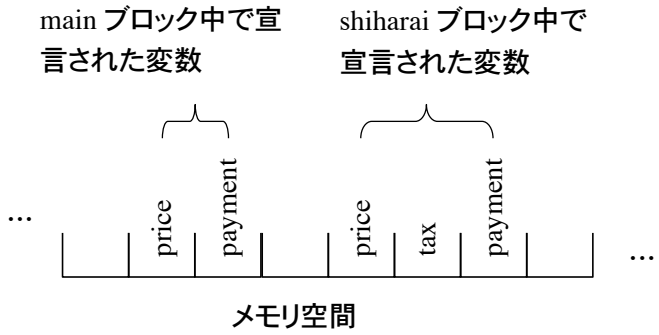
局所変数を用いることにより、外部(他の関数など)から影響を受けない変数の使用が可能になる。(変数の隠ぺい)

局所変数のメモリ上の配置

```
int shiharai(int);

main()
{
  int price, payment;

  scanf("%d", &price);

  payment = shiharai(price);
  printf("%d\n", payment);
}

int shiharai(int price)
{
  int tax, payment;
  tax = price * 0.08;
  payment = price + tax;
  return payment;
}
```

ブロックスコープを持つ変数は変数名が同じであっても互いに独立。

メモリ上の変数の配置（処理系によって異なる）



ブロックスコープのお陰で、外部に隠ぺいされた変数が利用できる。関数内部で定義された変数はその関数内だけのみ有効。

様々なユーザ関数例 1

```
double fn(double);

main()
{
  double x = 0.0, y;

  do{
    y = fn(x);
    printf("%f %f\n", x, y);
    x += 0.1;
  }while(x <= 10);
}
double fn(double x)
{
  double tmp;

  tmp = exp(-0.1*x)*sin(x);
  return tmp;
}
```

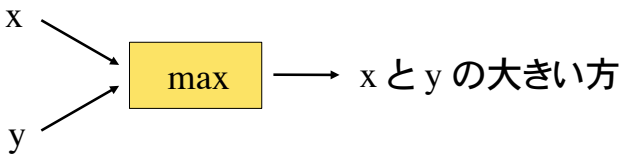
数学の関数 $y = \exp[-0.1*x] \sin(x)$ である。

計算結果をリダイレクションを用いてファイルに書き出し、gnuplot でグラフに描く。

```
% ./a.out > data
% gnuplot

        G N U P L O T
        Linux version 3.7
        ......
gnuplot > plot "data"
gnuplot > quit
%
```

様々なユーザ関数例 2

double 型の値を 2 つ受け取り、大きな方を返却値として返す関数




```
double max(double x, double y)
{
  double tmp;

  if(x>=y)
    tmp = x;
  else
    tmp = y;

  return tmp;
}
```

返却値は double 型、2 つの仮引数 x, y も double 型。

変数 tmp は、この関数の内部でのみ有効な局所変数。

どちらが大きいか判定して、値が大きな方を return 文で返す。

様々なユーザ関数例 2 関数呼出

```
double max(double, double);

main()
{
  double x, y, z;
  scanf("%lf %lf", &x, &y);
  z = max(x, y);
  printf("%f\n", z);
}

double max(double x, double y)
{
  double tmp;

  if(x>=y)
    tmp = x;
  else
    tmp = y;
  return tmp;
}
```

関数 max のプロトタイプ宣言。
引数の型をコンマで区切って宣言。

関数 max の呼び出し。実引数 x, y の値は、関数 max の仮引数 x, y にコピーされる。関数 max へ処理が移り、返却値を変数 z に代入。

様々な関数

値を返却しない（返却値がない）関数を void 関数と呼ぶ。

```
void graph(int n)
{
  int i;
  for(i=0; i<n; i++)
    printf("*");
  printf("\n");
}
```

返却値の型として void を指定。

引数として受け取った整数値分の * を出力して改行。

引数を持たない関数も定義できる。

```
void hello(void)
{
  printf("Hello!\n");
  printf("How are you?\n");
}
```

仮引数の宣言として void を指定。

様々な関数具体例

```
void graph(int);

main()
{
  int score[10], i;

  /* 配列 score へ 10 人分の成績を入力 */

  /* 入力結果の視覚化 */
  for(i=0; i<10; i++)
    graph(score[i]);
}

void graph(int n)
{
  int i;
  for(i=0; i<n; i++)
    printf("*");
  printf("\n");
}
```

プロトタイプ宣言

関数呼出。
返却値はない。

main 文の変数 i と関数 graph 内の変数 i は別物であることに注意（ブロックスコープ）

問題 1

正の実数を受け取り、小数点以下のみを取り出す関数を定義して、以下の動作をするプログラムを作れ。

ヒント：int 型の変数に double 型の値を代入すると小数点以下が切り捨てられることを用いよ。

```
% ./a.out
正の実数を入力：5.1234
少数部分は 0.1234 です。
%
```

main 文はすでに完成している。関数 my_function を完成せよ。

```
double my_function(double);
main()
{
  double input, output;
  printf("正の実数を入力:");
  scanf("%lf", &input);
  output = my_function(input);
  printf("少数部分は %f です\n", output);
}
```

問題 2

成績（100点満点の整数値）を受け取り、優、良、可、不可、を出力する関数を定義して、以下の動作をするプログラムを作れ。

100~80 : 優
79 ~ 70 : 良
69 ~ 60 : 可
59 ~ 0 : 不可

```
% ./a.out
成績を入力:90
貴方の成績は優です。
%
```

main 文の骨格はすでに完成している。
関数 hantei を定義せよ。

```
void hantei(int);
main()
{
  int score;

  scanf("%d", &score);
  hantei(score);
}
```

関数 hantei は成績を引数として受け取り、上記の判定にしたがって、優・良・可・不可を表示する。
（返却値無し）

問題 3

正の整数を受け取り、それが素数であれば 1 (int) を、素数でなければ 0 (int) を返却値として返す関数 prime を完成させよ。

```
% ./a.out
正の整数を入力:13
13 は素数です
%
```

main 文の骨格部分はすでにでき上がっている。

```
int prime(int);
main()
{
  int i;
  scanf("%d", &i);
  if( prime(i) )
    printf("%d は素数です\n", i);
  else
   printf("%d は素数ではない\n", i);
}
```

関数 prime は整数値の引数を受け取る。返却値は int 0 もしくは int 1 である。

問題 4

文字を引き数として受け取り、大文字に変換して返却する関数 ToUpperCase を定義して、文字入力を大文字に変換して表示するプログラム。

```
% ./a.out
all alphabets in lowercase should be converted to uppercase
ALL ALPHABETS IN LOWERCASE SHOLD BE CONVERTED TO UPPERCASE
%
```

関数 ToUpperCase は、アルファベットの文字コードを受け取ると、これを大文字に変換して返却。アルファベット以外の文字コードはそのまま返却する。

main 文の骨格部分はすでにでき上がっている。

```
int ToUpperCase(int);

main()
{
  int code, code2;
  while( (code=getchar()) != EOF ){
    code2 = ToUpperCase(code);
    putchar(code2);
  }
}
```

問題 5

gnuplot 等の視覚化ツールを用いて次の関数をグラフに描け。

$$z = f(x, y) = \sin(x^2 + y^2)/(1 + x^2 + y^2) \quad -4 \le x \le 4, -4 \le y \le 4$$

2 次元平面上の点 (x, y) の高さが $z = f(x, y)$ で与えられる 3 次元空間内の局面。
2 つの引き数をもつ関数を自分で定義して次の形式で出力する。

```
% ./a.out
–4.0  –4.0   0.0167099
–3.9  –4.0 –0.00634816
....
```

← x, y, z の形式で出力（スペースで区切って出力）

```
% ./a.out > data
% gnuplot
...
gnuplot > splot "data"
gnuplot > quit
%
```

← リダイレクションで計算結果をファイルへ書き込む。

← gnuplot からデータファイルを読み込み視覚化する。

問題 6

局所変数のアドレスを表示することにより、変数のブロックスコープを確認せよ

```
int shiharai(int);

main()
{
  int price, payment;

  scanf(“%d”, &price);

  payment = shiharai(price);
  printf(“%d\n”, payment);
}

int shiharai(int price)
{
  int tax, payment;
  tax = price * 0.08;
  payment = price + tax;
  return payment;
}
```

左のプログラムで使用される全ての局所変数のアドレスを表示せよ。

アドレスを表示する際の変換指定は %x

プログラム中にアドレスを表示する printf を追加

```
printf("Address of price in main() is %x\n", &price);
```