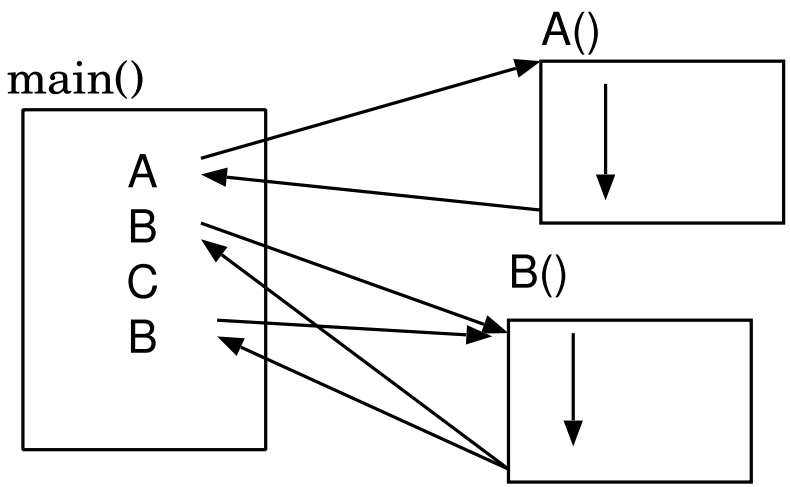
C++プログラミングI

関数定義と変数のスコープ

成蹊大学理工学部

関数とは

- 一連の処理に名前をつけたもの
- プログラムを区切って読みやすくする
- 用途：
  - 似たような処理を一つにまとめる
  - 一般的な関数を作り再利用（ライブラリ関数）
  - プログラムの管理を関数ごとに



関数定義

- 戻り値の型、関数名、仮引数
  - 仮引数: 呼び出される関数側の引数
  - 実引数: 呼び出し側で指定する引数
- 値渡し：実引数の値が仮引数に渡される（代入）
- 戻り値：return 文で関数が返す値を指定
  - 戻り値は呼出し側で変数に代入できる
  - 関数呼び出しは式を構成する変数と同じ場所で指定

```
int sumup(int from, int to)
{
    ...
    return sum;
}

s1 = sumup(1, 10);
int s2 {sumup(s1, s1+10)} ;
std::cout << sumup(s2, s2+10) << "\n";
```

void 関数

- 戻り値のない関数には void を指定する
- 予約語 void は無効を表す
- 値指定なしの return を使う

```
void print(int x)
{
    if (x == 0) return;
    std::cout << x <<"\n";
}
```

関数の引数

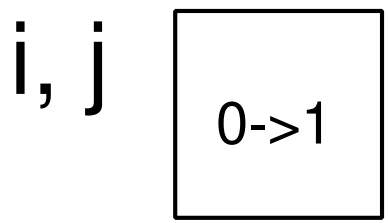
- 関数の引数は 0 個でも複数でも良い
- 引数 0 個の場合も丸括弧 () が必要
- 同一型の複数引数でもそれぞれに型指定が必要
  - ○ : int x, int y
  - × : int x, y
- 実引数には代入可能な式（右辺値）を指定する

```
int zero()                          { ... }
int one(int x)                      { ... }
int two(int x, int y)               { ... }
double dist(double x, double y) { ... }
int mix(int x, double d)            { ... }
```

lvalue リファレンス

- リファレンス：参照によってある変数の別名となる
- リファレンスは lvalue(左辺値）となれる変数が対象
- 一度初期化したら他の変数に変更できない

```
int  i {0};  // i は左辺値にできる
int& j {i};  // j は i の別名
++ j;
cout << i <<" "<< j <<"\n";
```

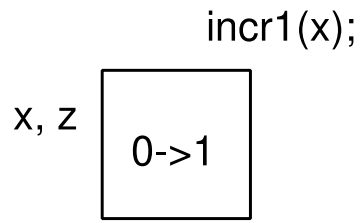


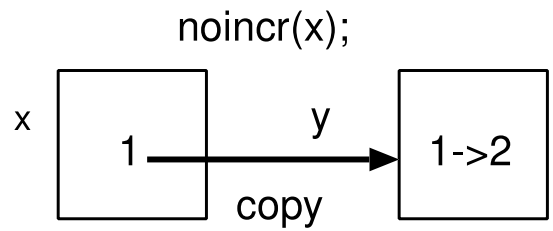
リファレンス引数

- lvalue リファレンスを引数に適用
- 関数の呼び出し先で実引数の変数を変更する

```
void incr1(int& z) { ++z; }    // 参照渡し
void noincr(int y) { ++y; }    // 値渡し

incr1(x);     // x の値が更新される
noincr(x);    // x は変更なし
```





変数の入れ替え関数 (swap)

- 2 つの実引数の値を入れ替える

```
void swap_int(int& a, int& b)
{
    int t {a};
    a = b;
    b = t;
}

swap_int(x, y);  // x と y の値の入れ替え
std::swap(x, y); // ライブラリ関数がある
```

vector と引数

- vector 変数の値渡しは全要素のコピー
  - 要素数が多ければ時間が無視できなくなる
- 以下のどちらかの参照渡しが望まれる

```
void f(const vector<T>& a); // 実引数変更不可
void f(vector<T>& a);       // 実引数変更可
```

main 関数の仮引数

- コマンド引数の利用
- 利用には C 配列とポインタの知識が必要
  - argc:コマンド引数の数,
  - argv:文字列へのポインタ配列
- vector<std::string>に変換できる

```
int main(int argc, char *argv[])
{
   // argv(コマンド引数) を a に取り込む
   vector<std::string> a(argv, argv+argc);

   for (const std::string& s : a)
      cout << s <<"\n";
   return 0;
}
```

局所変数のスコープ (有効範囲)

関数の中の変数：局所変数、有効範囲がある

- {}の範囲をブロックと呼ぶ
- ブロック内にそのブロック用の局所変数を宣言できる
- ブロックは内部に別のブロックを持てる (入れ子構造)
- 内側のブロックの変数名が外側のものと同じならば、外側を一時的に隠す

```
void func() {
    int x {1};     // 宣言 a)
    if (x > 0) {
        int x {2}; // 宣言 b) 宣言 a) の x を隠す
    }
    cout << x <<"\n"; // 再び宣言 a) の x が使える
    ...
}
```

例外的なルール

- 仮引数は関数本体のブロックがそのスコープ
- 制御文の文末までのスコープ
  - `for` 文の初期設定で宣言された変数
  - `if`, `while`, `switch` 文の条件部で宣言された変数
  - 複文でない場合は中括弧 (`{}`) がないので注意

```
for (int i = 0; i < 10; i++) {
    cout << i <<"\n";
}
if (int n = abs(x)) {
    cout << 10/n <<"\n";
}
while (char ch = s[i++]) {
    cout << ch <<"\n";
}
```

宣言と条件 (C++17)

- if と switch の変数宣言と条件指定
- 複雑な論理式が指定できる
- セミコロンで宣言と論理式を区切る

```
if (int z {std::min(x, y)}; z > 10)
    std::cout << z << " is too large\n";

switch (int z {std::max(x, y)}; z*3) {
    case 3: std::cout << "three\n"; break;
    case 6: std::cout << "six\n";   break;
    case 9: std::cout << "nine\n";  break;
}
```

スコープの利用

- 変数のスコープを限定して読みやすいコード作成を!

```
int count1() { // cnt が重要であることが明確
    int cnt {0};
    for (int i = 1; i <= 100; i++) {
        if (int x {(i*i) % 3}; x == 0)
            ++ cnt;
    }
    return cnt;
}

int count2() { // 局所変数の用途が明確でない例
    int i, x, cnt = 0;      // 3 個の変数が同等
    for (i = 1; i <= 100; i++) {
        x = (i*i) % 3;
        if (x == 0) ++ cnt;
    }
    return cnt;
}
```

同じ名前の変数

- スコープが異なれば変数に同じ名前をつけても良い

```
void func1() {
    int x{};          // func1 に局所的な変数 x
    x = 1;            // func の局所変数を更新
}
void func2(int x){ // func2 に局所的な仮引数 x
    x = 2;            // 仮引数 x を更新
}
int func3(int x){ // 仮引数 x は func3 に局所的
    return x+1; // 戻り値
}
int main() {
    int x {0}; // main に局所的な変数 x
    func1();
    func2(x+1); // x+1 は実引数
    x = func3(2); // 2 は実引数
}
```

大域変数

- 大域変数：関数間で共通に使える変数
  - 宣言した場所より下側関数で利用できる
  - 以下では sum, from, to
- 局所変数：関数の中で宣言された変数
  - 宣言した場所の関数でだけ利用できる
  - 仮引数も局所変数

```
int sum {0};    // 大域変数の宣言
int from, to;
void sumup()
{
    sum = 0;    // 大域変数 sum の利用
    for (int i = from; i <= to; i++)
        sum += i;
}
```

変数の名前づけの慣習

- 局所変数には短めの名前
  - 使用範囲が限定的のため
- 大域変数には長めの名前
  - 使用範囲が広いので混乱を避ける目的

変数の初期値

- 初期値を指定しない組み込み型の変数
  - 大域変数は実行開始までに 0 に初期化される
  - 仮引数以外の局所変数の初期値は不定
  - 仮引数は実引数で初期値が指定される
- ユーザ定義型の変数の初期値はそれぞれで異なる
  - string 型の変数, cin, cout など

```
int    globalint;
double globalreal;
std::string globalstr;

void func(int p1, double p2) {
   int    l1;              // 値は不定
   double l2;              // 値は不定
   std::string l3;                 // 空文字列で初期化
}
```

大域変数と局所変数の関係

- 大域変数と局所変数が同じ名前ならば局所変数が優先
- 同じ名前とならないようにする
- 大域変数はスコープ解決演算子（::）で明示可能

```
int x {0};  // 大域変数
void func1() {
    x = 1;   // 大域変数の更新
}
void func2() {
    int x{}; // 局所変数
    x = 2;   // 局所変数の更新
}
void func3() {
    int x{}; // 局所変数
    ::x = 3; // 大域変数の更新
}
```