関数をつくってみる

- 複雑なプログラムを作る場合、
「ヤヤコシイ」部分を関数として切り分けると、
意外と簡単になることも

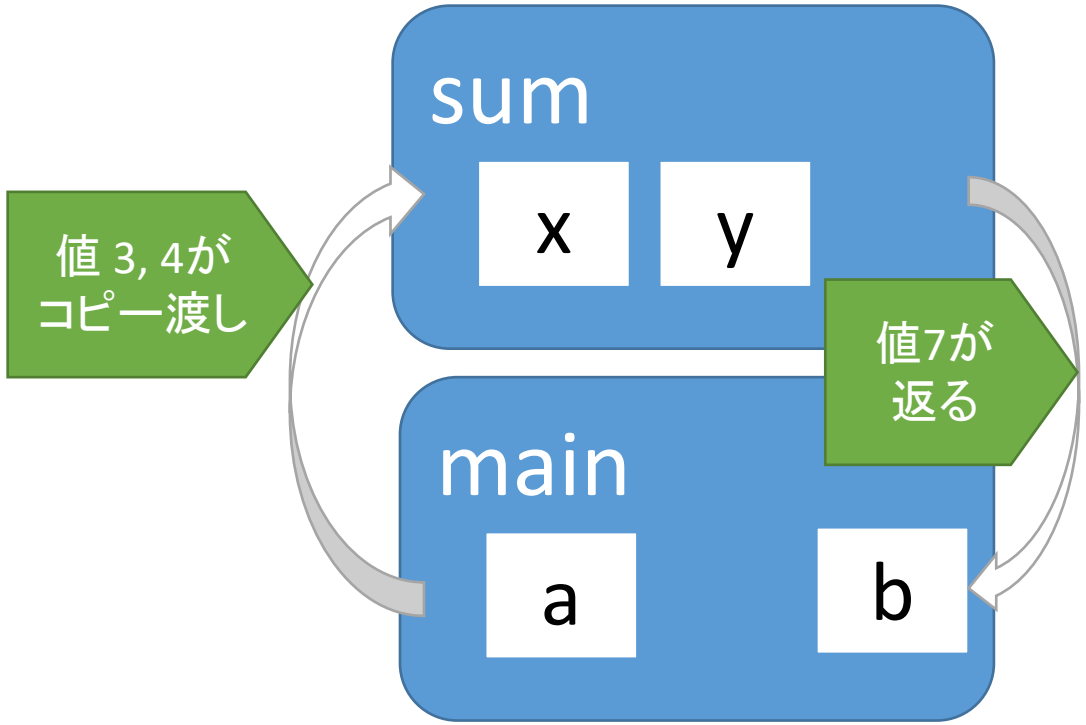
練習

- まずは、関数の実行イメージを身につける
- で、関数つかって面倒そうな問題を解いてみる
- 少しヤヤコシイ「引数の渡し方」の解説

関数呼び出し

引数の扱いについては、
後で詳しく

- main 関数が sum 関数を呼び出す例（ x + y の結果を返す）




```
int sum(int x, int y) {
    int result = x + y;
    return result;
}
int main(void)
{
    int a = 3;
    int b = sum(a, 4);
    cout << "main0:" << a << "," << b << endl;
    return 0;
}
```

実行結果
main0:3,7

ちょっとヤヤコシイ問題

2013年パソコン甲子園予選3
（本家 pdf）（レギオ用AtCoder）

- 野菜の苗に、雑草が混ざってしまった。
  - 野菜の長さは等差数列、雑草は違う
- 入力例
  - 1行目：野菜の本数、2行目：野菜＋雑草の長さ

```
5
1 2 3 6 4 5
```

```
4
5 7 9 11 12
```

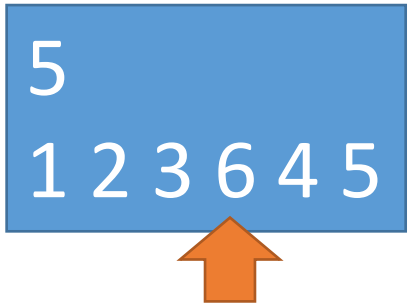
- 出力：雑草の長さ

解けそうな気はする？

- とりあえず、入力を取り込むのは簡単
  - 配列か、vector にいれておこう！

```
cin >> n;
vector<int> vec(n+1);
for (int i = 0; i < n + 1;  i++) {
    cin >> vec[i];
}
```

- さて、どうやって解く？
  - 意外に悩ましい。。。。
- 単純な案
  - i 番目の要素を無視して、等差数列になっているかチェックしよう！



プログラムの概略は？

- 雑草を仮決めしては、
  等差数列か
  確認すればいいよね。

```
for(int skip=0; skip<n+1; skip++) {
    skip が雑草かどうか、等差数列チェック;
    雑草だったら、vec[skip] 出力して終了
}
```

- とはいえ、雑草を考慮した等差数列チェックって？
  - ごちゃごちゃして分からん。。。

そういうときは、仕事を切り分けると、
頭の整理がつくかも。

等差数列チェックは作れそう？

- 与えられた vector が、等差数列かチェックする
  - vec[1] - vec[0] が
  - vec[i+1] - vec[i] (for i= 1..n-1) になってれば OK.

- で、skip の扱いどうするか？
  - 案1: skip を除いた vector を作ってみる
  - 案2: skip を考慮して「i番目」にアクセスする
    - 今回はこちらでいってみましょう

ややこしいものは関数に切り分け

- 関数の作ることにしよう！
  - bool check(skip, n, vec):　skip 番目が雑草として、vec が等差数列か判定。

- 呼び出し側はスッキリ
  - あとは、関数つくるだけ

```
    for(int skip=0; skip<n+1; skip++) {
        if(check(skip, n, vec)) {
            cout << vec[skip] << endl;
            return 0;
        }
    }
```

check 関数作るぞ

- 与えられた vector が、等差数列かチェックする（再掲）
  - vec[1] - vec[0] が
  - vec[i+1] - vec[i] (for i= 1..n-1) になってりゃOK.

- skip 考慮したi番目もヤヤコシイ

```
int check(int skip, int n, vector<int> & vec) {
    int d = 1番目 - 0番目;
    for(int i=1; i< n; i++) {
        if(「i+1番目 - i番目」!= d)
            return false;
    }
    return true;
}
```

再度、ヤヤコシイ物は切り分け

- skip 考慮した i 番目って、何番目？
  - int nth(int skip, int i)
- 呼ぶ側は、簡単に。

```
bool check(int skip, int n, vector<int> & vec) {
    int d = vec[nth(skip,1)]-vec[nth(skip,0)];
    for(int i=2; i< n; i++) {
        if(vec[nth(skip,i)]-vec[nth(skip, i-1)]!= d)
            return false;
    }
    return true;
}
```

再度、ヤヤコシイ物は切り分け（続）

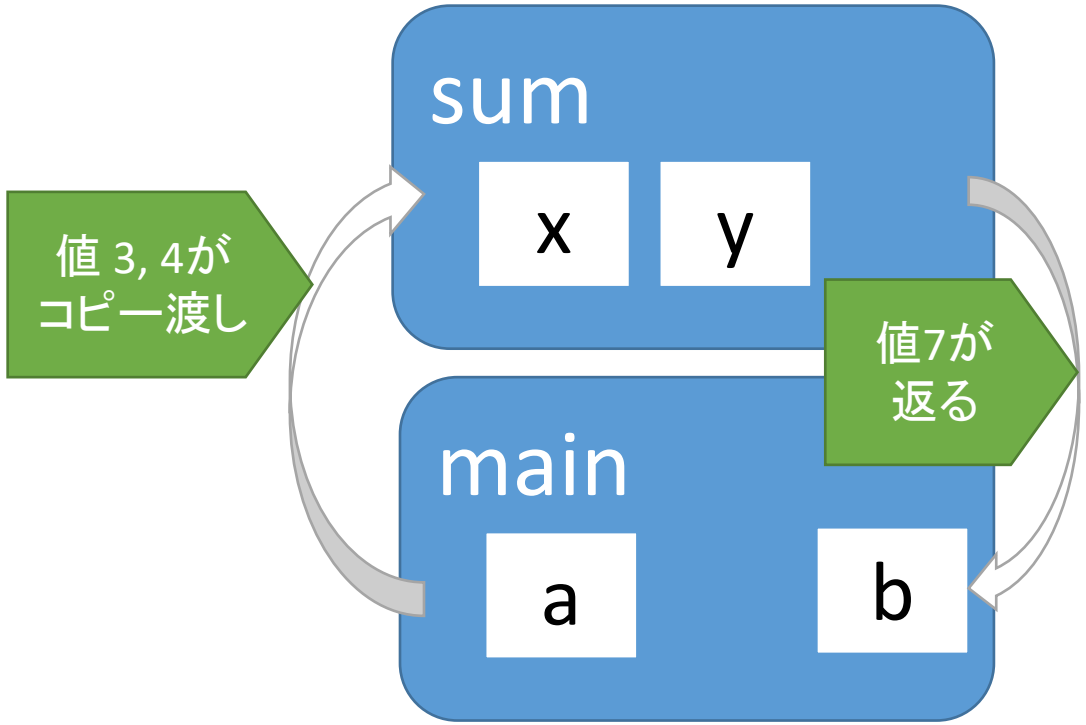
- skip 考慮した i 番目って、何番目？
  - int nth(int skip, int i)
- この関数はそれほど難しくない
  - i < skip なら、素直に i 番目
  - i>= skip なら、i+1番目

```
int nth(int skip, int i) {
    if(i<skip) return i;
    return i+1;
}
```

関数呼び出し(値渡し)

デバッガで確認

- main 関数が sum 関数を呼び出す例(x + y の結果を返す)



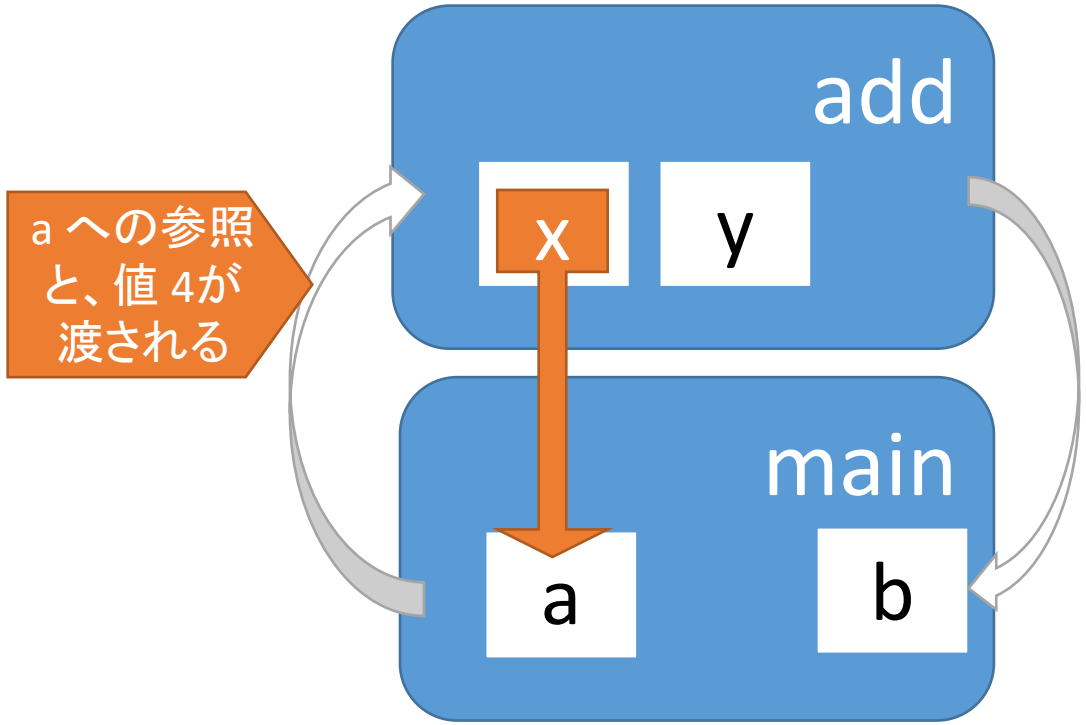

```
int sum(int x, int y) {
    int result = x + y;
    x++; // x を変化させてみる
    cout << "in sum() " << x << endl;
    return result;
}
int main(void)
{
    int a = 3;
    int b = sum(a, 4);
    cout << "main0:" << a << "," << b << endl;
    return 0;
}
```

実行結果
in sum() 4
main0:3,7

関数呼び出し(参照渡し)

デバッガで確認

- main 関数が add 関数を呼び出す例(x の参照先に y を加算)



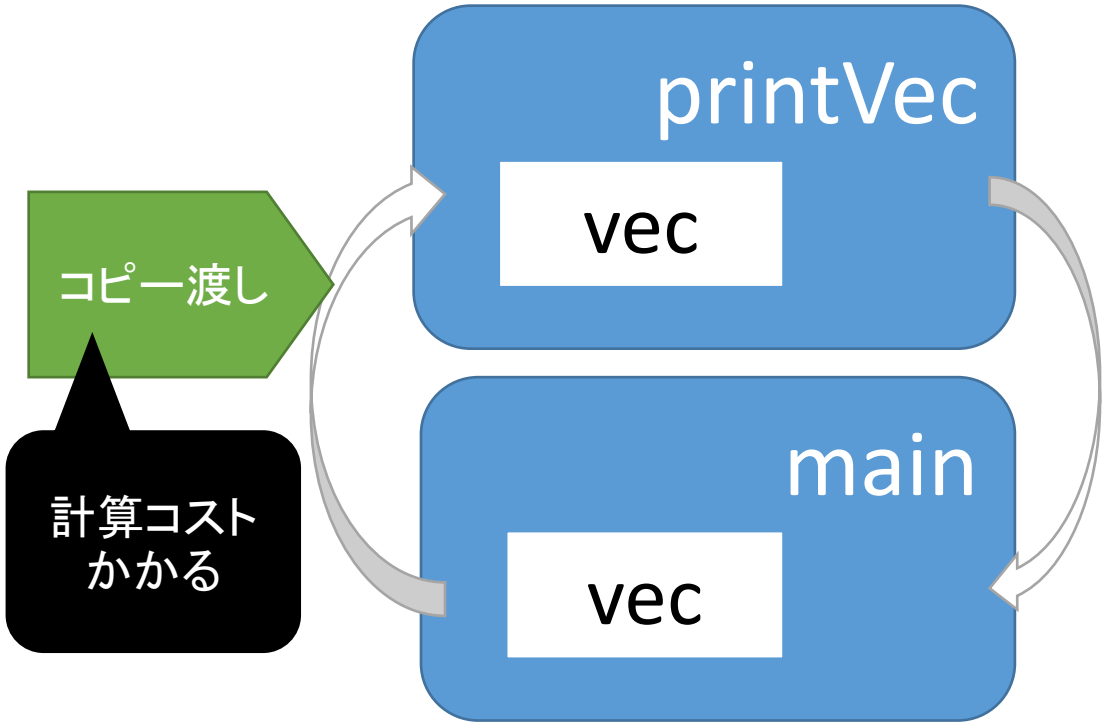

```
void add(int & x, int y) {
    x += y;
    cout << "in add() " << x << endl;
}

int main(void)
{
    int a = 3;
    int b = sum(a, 4);
    cout << "main0:" << a << "," << b << endl;
    add(a, b);
    cout << "main1:" << a << "," << b << endl;
    return 0;
}
```

実行結果
main0:3,7
in add() 10
main1:10,7

関数呼び出し（vector の場合）

デバッガで確認

- printVec で vector の中身を表示



& なし

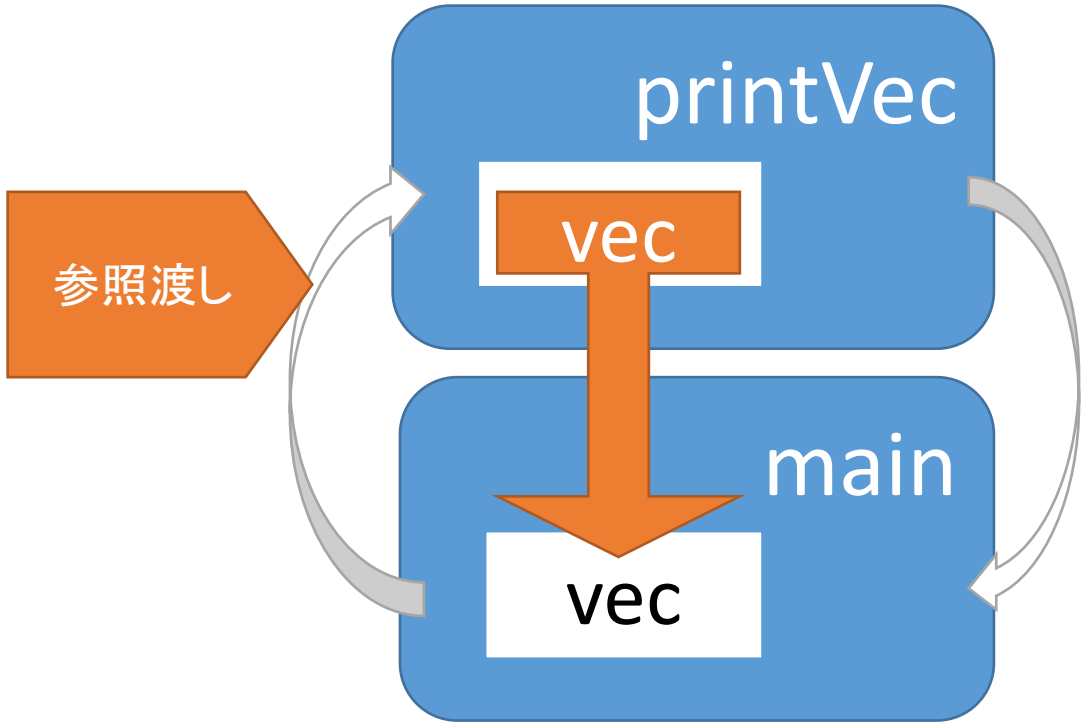
```
int printVec(vector<int>    vec) {
    vec.push_back(10); // 実験のため要素を追加
    cout << "VecX:";
    for (size_t i = 0; i < vec.size(); i++) {
        cout << vec[i] << " ";
    }
    cout << endl;
}
int main(void)
{
    vector<int> vec{1, 2};
    printVec(vec);
    return 0;
}
```

呼び出し側の vec が変わらず

関数呼び出し（vector の場合）

デバッガで確認

- printVec で vector の中身を表示



& あり

```
int printVec(vector<int> & vec) {
    vec.push_back(10); // 実験のため要素を追加
    cout << "VecX:";
    for (size_t i = 0; i < vec.size(); i++) {
        cout << vec[i] << " ";
    }
    cout << endl;
}
int main(void)
{
    vector<int> vec{1, 2};
    printVec(vec);
    return 0;
}
```

呼び出し側の vec が変更