C++プログラミングI

第4回 繰り返し処理

成蹊大学理工学部

繰り返しを学ぶ前に考えること

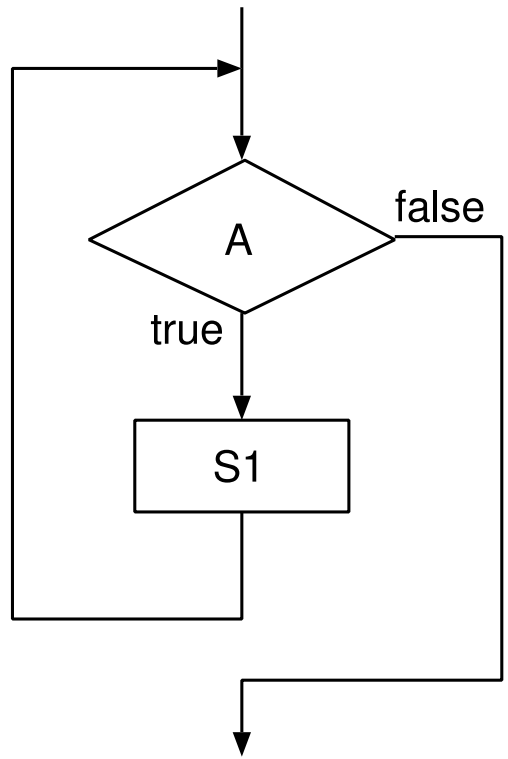
- 複数の仕事を仕上げるには
  - 一つ一つ順番におこなす
  - 似ている作業をまとめると良い
  - 仕事を分担できると早く終わる
  - 分担も作業の分類が大切
- 現代のコンピュータの特徴
  - 繰り返し処理が基本
  - 並列処理もできる時代になった
    - 繰り返し処理をパターン化しておくと良い
- 処理パターンの手がかり
  - 対象データ
    - 全体が手元にある
    - 数式で得られる
    - 入力で得られる（量が不明）
  - 処理の目的
    - 探す, まとめる, 各値を変更する

while 文

while 文

- 条件 A を満たす限り文 S1 を繰り返す
  - 条件 A は bool 型の式
  - 文 S1 が単一文（複文も使用できる）
- 注意点
  - 条件 A の計算に変化があるかを確認する

```
while (条件 A)
    文 S1
```



while 文の例 1: 基本

```
// 指定の数値以下になるまで値を半分にする
double x {0.0};
std::cin >> x;
while (x > 0.01) {  // x が 0.01 より大きい
    std::cout << x << "\n";
    x /= 2;
}
std::cout << x << "\n";
```

```
7                   <<-- これを入力
7
3.5
1.75
0.875
0.4375
0.21875
0.109375
0.0546875
0.0273438
0.0136719
0.00683594
```

while 文の例 2: 入力条件

- `cin>>x` は演算で結果は `cin`
- `bool` として評価すると入力の成功と失敗が分かる
- 入力失敗の可能性
  - 入力し尽した
    - EOF(ファイルの終端) に到達した
    - `Ctrl-D`（キーボードによる EOF）
  - 不正な文字列を入力した

```
int sum {0};
int x;
while (std::cin >> x) // 入力成功が条件
    sum += x;
std::cout << sum << "\n";
```

cin と >> 演算子について

- std::cin は大域変数 (オブジェクト)
- >>は左結合の二項演算子
- cin >> x の式の結果は cin 自身
  - 入力によって内部状態が変更される
- 連続する入力をまとめて指定できる

```
std::cin >> x >> y >> z;
((std::cin >> x) >> y) >> z; // 同じ意味
```

- bool 式として使うと入力の成功/失敗が分かる

```
while (std::cin >> x)
  std::cout << x <<"\n";
```

while 文の例 3: vector

- push_back() メンバ関数の利用
- 蓄えられるデータ数は有限であることに注意

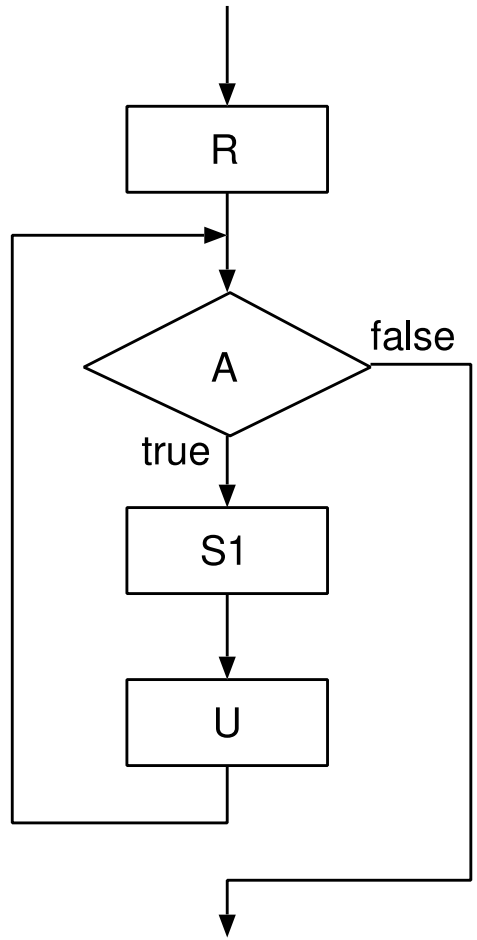
```
std::vector<int> v;
int x;
while (std::cin >> x)
    v.push_back(x);
```

for 文

for 文

- while 文に処理 R と U を加えた形
- よくあるパターンを明示する目的
- R と U はループ制御変数に関係する処理が多い
- R での宣言された変数の有効範囲は for 文内のみ

```
for (初期化つき変数宣言 R; 条件 A; 式 U)
    文 S1

または

for (代入 R; 条件 A; 式 U)
    文 S1
```



for 文の例 1

- 2 行目の宣言 i は 2-3 行目が有効範囲
- 宣言は int i{1}; と書いても良い
- 7 行目の宣言 i は 10 行目までが有効範囲
- 2 行目と 7 行目で宣言された i は別変数

```
1  int sum {0};
2  for (int i = 1; i <= 30; i++)
3      sum += i;
4  std::cout << sum <<"\n"; // 465
5
6  sum = 0;
7  int i;
8  for (i = 1; i*i*i <= 1357; i++)
9      if (i % 2) sum += i;
10 std::cout << i-1 <<":"<< sum <<"\n"; // 11:36
```

for 文と vector

- 制御変数が配列の添字として使用される
- 半開区間 [0, 要素数) を示すように for 文を書く

```
std::vector v {1.5, 8.4, 2.3, 4.6, 3.5};
double sum {0.0};
for (int i = 0; i < v.size(); i++)
    sum += v[i];
std::cout <<"sum: "<< sum <<"\n";
std::cout <<"avg: "<< sum/v.size() <<"\n";
```

for 文と入力

- 入力用の一時変数の有効範囲を狭める
  - 一時変数の宣言に行数をとられない効果もある
- for 文の更新処理を書かないこともある
  - セミコロン (;) は必要

```
int sum {0};
for (int x=0; std::cin >> x; )
    sum += x;
std::cout << sum <<"\n";
```

範囲 for 文

- 複数要素を持つデータ構造用の `for` 文
  - `string`, `vector` 以外にも利用可
- 変数 `v` の要素を一つずつ `e` に代入させて繰り返し
- 変数 `v` の要素を一つずつ `e` に対応させて繰り返し

for (型名 要素用の変数名 e : 複数データの変数 v)
または
for (型名& 要素用の変数名 e : 複数データの変数 v)

範囲 for 文の例

- 読み出しと更新で&の指定の有無を決める
- v から要素を d に一つずつ取り出す
- s から文字を ch に一つずつ取り出す

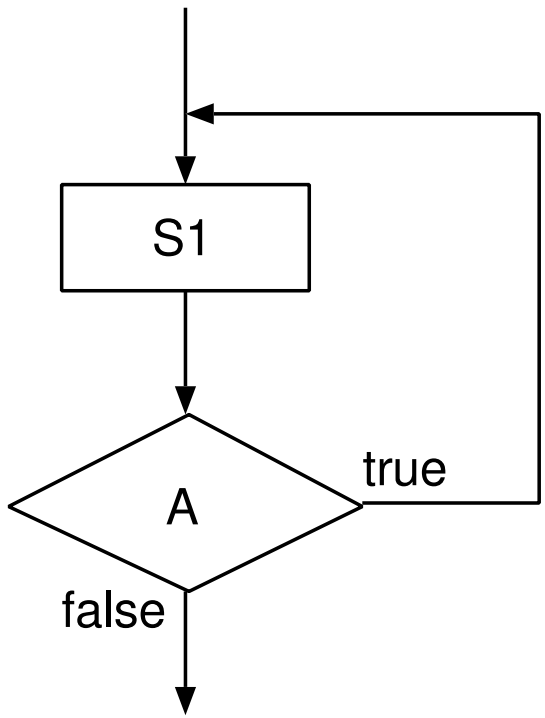
```
std::vector v {1.2, 3.4, 5.6, 7.8, 9.0 };
for (double d : v)
    std::cout << d <<" ";
std::cout <<"\n";

for (double& d : v) // v を更新する場合
    d *= 2.0;

std::string s {"abcdefg"};
for (char ch : s)
    std::cout << ch <<"\n";
```

do-while 文

do-while 文

- 最低 1 回は処理をする繰り返し
- 文 S1 の後に条件 A を評価する
- フローチャートはシンプルで処理効率も良い
- あまり使われない
  - 最低 1 回行う処理を使う場面が少ない
  - 条件 A に使用できる変数の制限

```
do {
  文 S1
} while ( 条件 A );
```



do-while の例

- 必ず 1 回は入力することにした。
- ch は do-while より前に宣言が必要

```
double sum {0.0};
int cnt {0};
char ch;
do {
    double x {0.0};
    std::cout << " 温度を入力: ";
    std::cin >> x;
    sum += x;
    ++ cnt;
    std::cout << " まだある?(y/n) ";
    std::cin >> ch || (ch = 'n');
} while (ch == 'y');
std::cout <<"avg: "<< sum/cnt <<"\n";
```

その他

無限ループ

- あえて終了させないループ
- 2種類の書き方がよく使われる

```
while (true) {
    // 無限に行う処理
}

for (;;) {
    // 無限に行う処理
}
```

入れ子の繰り返し指定

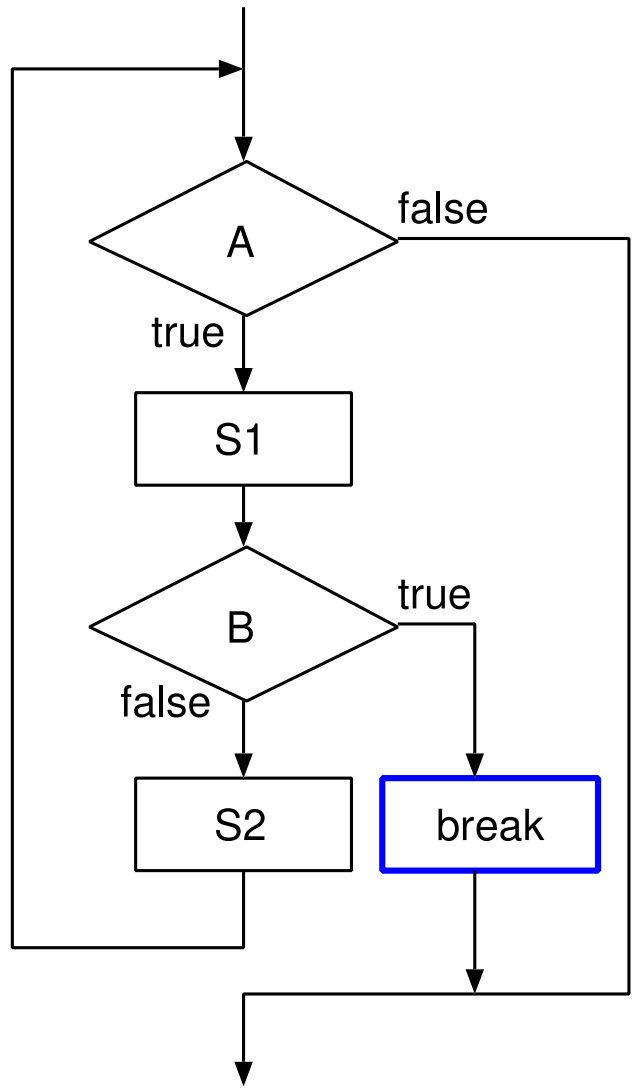
- 二重ループがよく使われる

```
// 120 未満の素数一覧
std::vector<bool> a(120, true);

// 条件に合わない数を探す
a[0] = a[1] = false;
for (int i = 2; i < a.size(); i++) {
    for (int j = 2; i*j < a.size(); j++) {
        a[i*j] = false;
    }
}

// 結果の出力
for (int i = 2; i < a.size(); i++)
    if (a[i]) std::cout << i <<" ";
std::cout <<"\n";
```

break 文

- break: 途中でループを抜ける

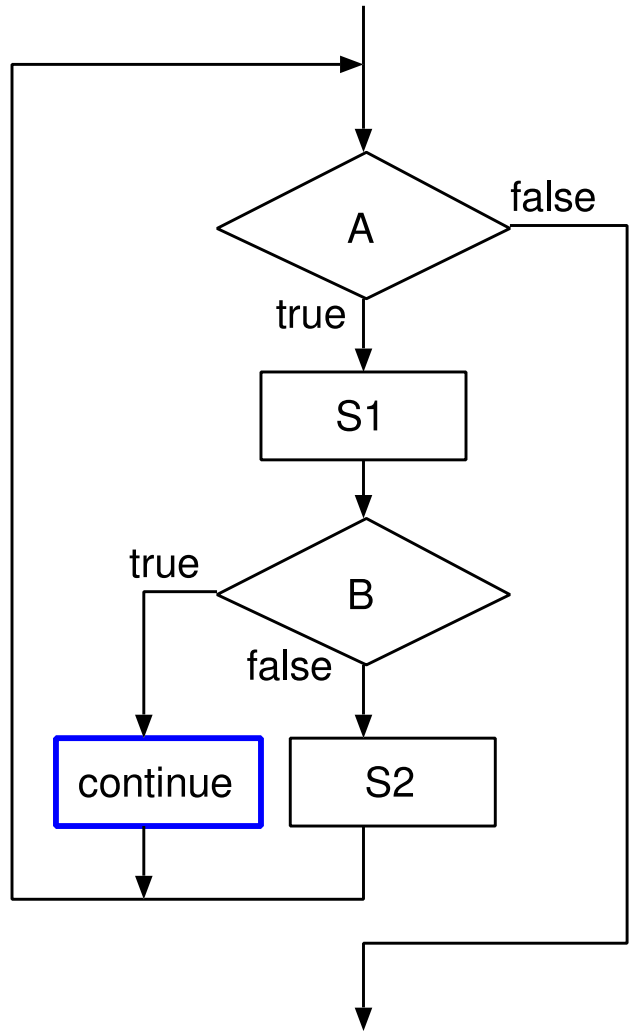
```
while ( 条件 A ) {
  文 S1
  if ( 条件 B )
    break;
  文 S2
}
```



continue 文

- continue: 途中で次の繰り返しに進む

```
while ( 条件 A ) {
  文 S1
  if ( 条件 B )
    continue;
  文 S2
}
```



break 文と continue 文の特徴

- ループの中で使う
  - break は switch 文でも使う
- if 文と組み合わせることが多い
- 入れ子ループでは、それを囲む一番内側のループが対象
  - 二重/三重ループの内側から抜けるには工夫が必要
- プログラムの流れを不規則にする

逐次探索

- データを探すための基本的な方法
- 先頭から探し、見つかれば終了

```
std::vector a {3, 6, 2, 8, 1, 5, 2, 9, 3, 7};
const int x {5}; // 探す対象

int i;              // 見つけた場所
for (i = 0; i < a.size(); i++)
    if (a[i] == x) break;

std::cout << (i < a.size()?"":"not ")
          << "found\n";
```