

C++プログラミングI

- 第4回：繰り返し処理
- 担当：二瓶芙巳雄

繰り返しを学ぶ前に考えること

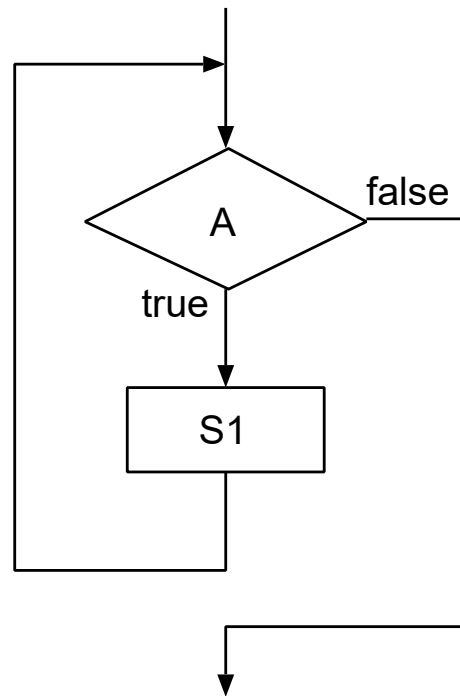
- 複数の仕事を仕上げるには
 - 一つ一つ順番におこなう
 - 似ている作業をまとめると良い
 - 仕事を分担できると早く終わる
 - 分担も作業の分類が大切
- 現代のコンピュータの特徴
 - 繰り返し処理が基本
 - 並列処理もできる時代になった
 - 繰り返し処理をパターン化しておくが良い
- 処理パターンの手がかり
 - 対象データ
 - 全体が手元にある
 - 数式で得られる
 - 入力で得られる（量が不明）
 - 処理の目的
 - 探す, まとめる, 各値を変更する

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

```
1 while (条件A)
2     文S1
```



while 文の例1: 基本

```
1  int main() {  
2      // 指定の数値以下になるまで値を半分にする  
3      double x {0.0};  
4      std::cin >> x;  
5  
6      while (x > 0.01) { // xが0.01より大きい  
7          std::cout << x << "\n";  
8          x /= 2;  
9      }  
10     std::cout << x << "\n";  
11 }
```

- 複数行の処理の場合, `{ }` で複文にする
- `if` 文と同じく, インデントに注意

```
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` (キーボード入力によるEOF)
 - ※コントロールキー (`Ctrl`, キーボードの左下) を押しながら `D` を押すと, EOFを送信できる
 - 不正な文字列を入力した (右例では, `int` 以外の値を入力した場合)

```
1  int main() {
2      int sum {0};
3      int x;
4      while (std::cin >> x) // 入力成功が条件
5          sum += x;
6      std::cout << "sum: " << sum << "\n";
7  }
```

```
% ./a.out
10 10 10 (エンターキーで確定, 続けてCtrl-Dを入力)
sum: 30
```

```
% ./a.out
1 2 3 4 5 6 7 8 9 (エンターキーで確定, 続けてCtrl-Dを入力)
sum: 45
```

```
% ./a.out
hello (エンターキーで確定)
sum: 0
```

```
% ./a.out
1 2 hello 3 4 (エンターキーで確定)
sum: 3
```

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

```
1 std::cin >> x >> y >> z;  
2 ((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

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

```
1 // 補足説明プログラム  
2 int main() {  
3     int x;  
4  
5     if( std::cin >> x )  
6         std::cout << x << ": success!\n";  
7     else  
8         std::cout << "error...\n";  
9  
10    // std::cinはboolにキャストできる  
11    // bool b = bool( std::cin >> x );  
12  
13    std::cout << "done\n";  
14 }
```

```
% ./a.out # 入力に成功したパターン  
100  
100: success!  
done
```

```
% ./a.out # 入力に失敗したパターン  
hello  
error...  
done
```

while 文の例3: vector

- `push_back()` メンバ関数の利用
- EOFが入力されるまで、値を `v` に追加できる
- 蓄えられるデータ数は有限であることに注意

```
1  int main() {
2      std::vector<int> v;
3      int x;
4
5      while (std::cin >> x)
6          v.push_back(x);
7
8      // for文は次で説明
9      for ( int i=0; i < v.size(); i++ )
10         std::cout << i <<"th: " << v[i] << ", ";
11     std::cout << "\n";
12 }
```

```
% ./a.out
10 20 30 (エンターキーで確定, Ctrl-Dを入力)
10 20 30
0th: 10, 1th: 20, 2th: 30,

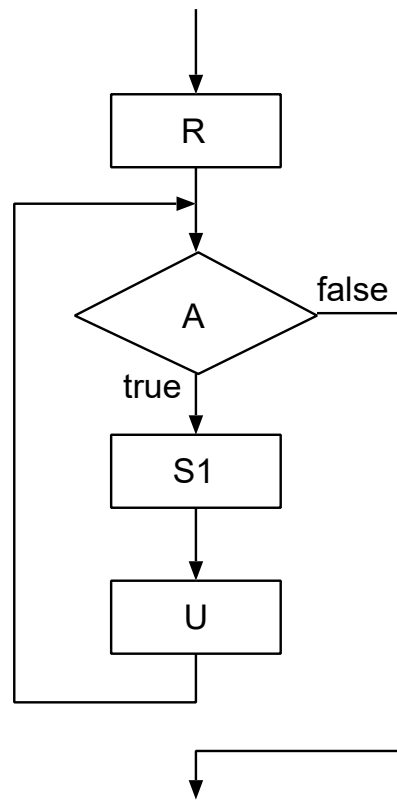
% ./a.out
2 3 5 7 11 13 (エンターキーで確定, Ctrl-Dを入力)
0th: 2, 1th: 3, 2th: 5, 3th: 7, 4th: 11, 5th: 13,
```


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

```
1  for (初期化つき変数宣言R; 条件A; 式U)
2      文S1
3
4  または
5
6  for (代入R; 条件A; 式U)
7      文S1
```



for 文の例1

- 3行目の宣言 `i` は3-4行目が有効範囲
- 宣言は `int i{1};` と書いても良い
- 8行目の宣言 `i` は11行目までが有効範囲
- 3行目と8行目で宣言された `i` は別変数

```
1  int main() {
2      int sum {0};
3      // for (初期化つき変数宣言; ループ条件; 更新式)
4      for (int i = 1; i <= 30; i++)
5          sum += i;
6      std::cout << sum << "\n"; // 465
7
8      sum = 0;
9      int i;
10     // for (代入; ループ条件; 更新式)
11     for (i = 1; i*i*i <= 1357; i++)
12         if (i % 2) sum += i;
13     std::cout << i-1 << ":" << sum << "\n"; // 11:36
14 }
```

```
% ./a.out
465
11:36
```

補足：色々なfor文

- for 文を見慣れましょう
- ループする回数を意識するのが大事かも
- 表を書くとわかりやすい

A (`for (int i = 0; i < 5; i++)`) の場合

i	0	1	2	3	4	5	6
i < 5	true	true	true	true	true	false	false

E (`for (int i = 0; i < 5; i += 2)`) の場合

i	0	2	4	6	8	10	12
i < 5	true	true	true	false	false	false	false

```
1  int main() {
2      // A: n回ループ (最も基本)
3      for (int i = 0; i < 5; i++) std::cout << i << " ";
4
5      // B: 1スタートで5回のループ (やや変則的)
6      for (int i = 1; i <= 5; i++) std::cout << i << " ";
7
8      // C: Bと本質的に同じループ
9      for (int i = 0; i < 5; i++) std::cout << i+1 << " ";
10
11     // D: 減らす方向のループ
12     for (int i = 4; i >= 0; i--) std::cout << i << " ";
13
14     // E: 更新式は工夫できる
15     for (int i = 0; i < 5; i+=2) std::cout << i << " ";
16
17     // F: 複雑な更新式 (読むのが大変)
18     for (int i = 0; i < 100; i=(i+1)*2) std::cout << i << " ";
19 }
```

% ./a.out

A: 0 1 2 3 4

B: 1 2 3 4 5

C: 1 2 3 4 5

D: 4 3 2 1 0

E: 0 2 4

F: 0 2 6 14 30 62

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
- 条件を `i <= v.size()-1;` と書くより, `i < v.size();` のほうがよい

```
1  int main() {
2      std::vector v {1.5, 8.4, 2.3, 4.6, 3.5};
3      double sum {0.0};
4      for (int i = 0; i < v.size(); i++)
5          sum += v[i];
6      std::cout << "sum: " << sum << "\n";
7      std::cout << "avg: " << sum/v.size() << "\n";
8  }
```

```
% ./a.out
sum: 20.3
avg: 4.06
```

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
- ※上下のプログラムは全く同じ挙動. お好みで.

```
1  int main() {  
2      int sum {0};  
3  
4      for ( int x=0; std::cin >> x; )  
5          sum += x;  
6      std::cout << sum << "\n";  
7  }
```

```
1  int main() {  
2      int sum {0};  
3      int x;  
4      while (std::cin >> x)  
5          sum += x;  
6      std::cout << "sum: " << sum << "\n";  
7  }
```

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

```
1  for (型名 要素用の変数名 e : 複数データの変数 v)
2
3  または
4
5  for (型名& 要素用の変数名 e : 複数データの変数 v)
```

- 一つ目の例：変数 v の要素を一つずつ e にコピーする
- 二つ目の例（& がある方）：変数 v の要素**そのもの**を一つずつ e として取り出す
 - &：リファレンス、のちの授業で詳細を説明。
- ※変数 v の要素を直接書き換えたい場合は & をつける。直接書き換えたくない場合は & をつけない。

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

```
% ./a.out
```

```
1.2 3.4 5.6 7.8 9
```

```
2.4 6.8 11.2 15.6 18
```

```
2.4 6.8 11.2 15.6 18
```

```
a b c d e f g
```

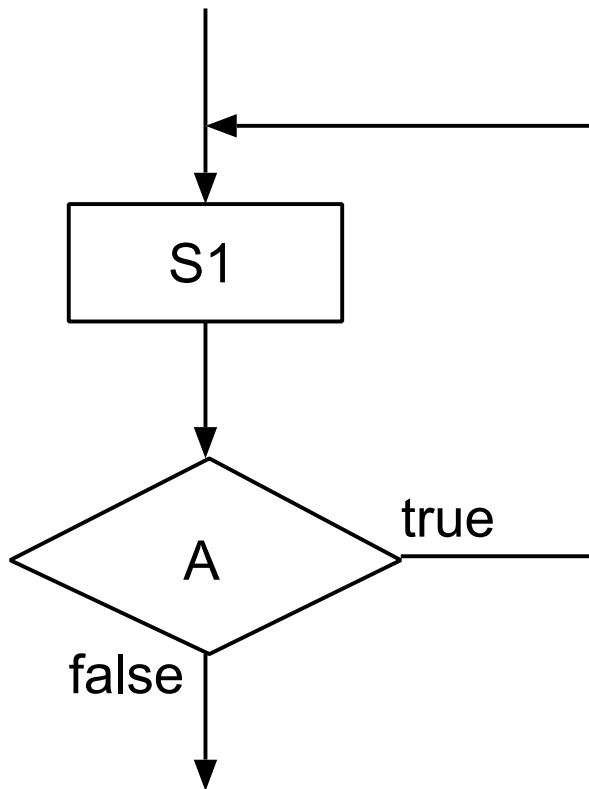
```
1  int main() {
2      std::vector<double> v{1.2, 3.4, 5.6, 7.8, 9.0};
3      for (double d : v) std::cout << d << " ";
4      std::cout << "\n";
5
6      // vを更新する場合
7      for (double& d : v) d *= 2.0;
8      for (double d : v) std::cout << d << " ";
9      std::cout << "\n";
10
11     // 以下ではvは更新できない
12     for (double d : v) d *= 2.0;
13     for (double d : v) std::cout << d << " ";
14     std::cout << "\n";
15
16     std::string s {"abcdefg"};
17     for (char ch : s) std::cout << ch << " ";
18     std::cout << "\n";
19 }
```


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

```
1  do {  
2      文S1  
3  } 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

```
% ./a.out
温度を入力: 10
まだある?(y/n) y
温度を入力: 20
まだある?(y/n) n
avg: 15
```

```
1  int main() {
2      double sum {0.0};
3      int cnt {0};
4      char ch;
5
6      do {
7          double x {0.0};
8          std::cout << " 温度を入力: ";
9          std::cin >> x;
10         sum += x;
11         ++ cnt;
12         std::cout << " まだある?(y/n) ";
13         std::cin >> ch || (ch = 'n');
14     } while (ch == 'y');
15
16     std::cout << "avg: " << sum/cnt << "\n";
17 }
```

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

```
1  int main() {  
2      while (true) {  
3          // 無限に行う処理  
4          std::cout << "hello";  
5      }  
6  
7      for (;;) {  
8          // 無限に行う処理  
9      }  
10 }
```

無限ループを止めたいときは **Ctrl-C** or **Ctrl-Z**. プログラムの実装ミスで意図せず無限ループになってしまったときにも、これで止められます.

入れ子の繰り返し指定

- 二重ループがよく使われる
- 素数の判定
 - エラトステネスのふるい法
- 二重ループにより、素数ではない（`false`）値を決める
- `2 <= i < 120` の範囲で `i` に対して、その `i` の2倍, 3倍, ... の値は `i` で割りきれ値なので、その添字が対応する配列の要素を `false` にする

```
1  int main() {
2      // 120未満の素数一覧
3      std::vector<bool> a(120, true); // 初期値は120個のtrue
4
5      // 条件に合わない数を探す
6      a[0] = a[1] = false;
7      for (int i = 2; i < a.size(); i++) {
8          for (int j = 2; i*j < a.size(); j++) {
9              a[i*j] = false;
10         }
11     }
12
13     // 結果の出力
14     for (int i = 2; i < a.size(); i++)
15         if (a[i]) std::cout << i << " ";
16     std::cout << "\n";
17 }
```

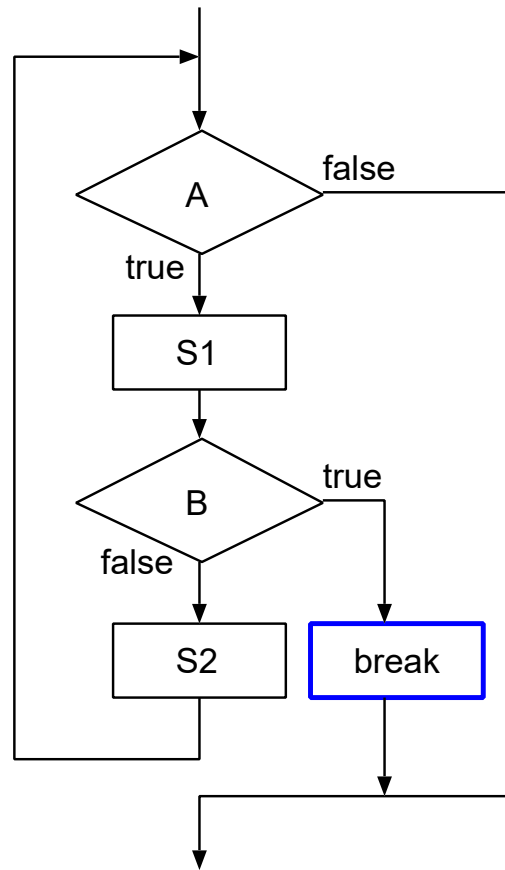
% ./a.out

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43
47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113
```

break 文

- break : 途中でループを抜ける

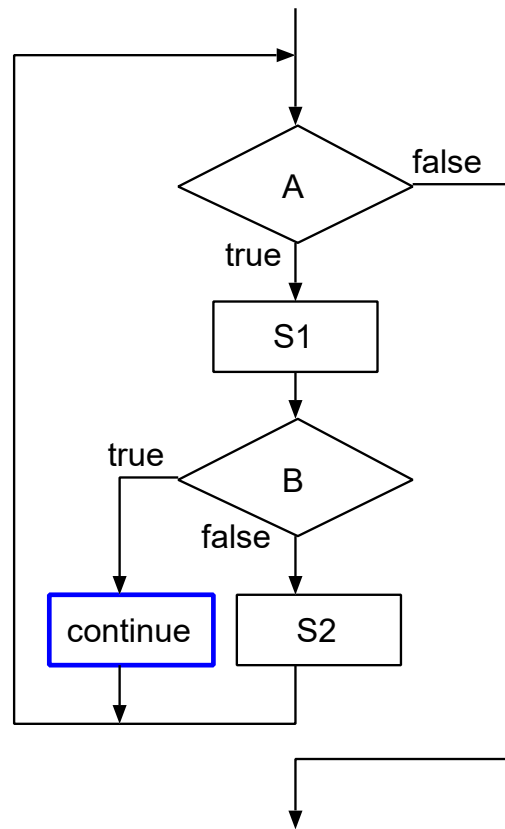
```
1 while ( 条件A ) {  
2   文S1  
3   if ( 条件B ) break;  
4   文S2  
5 }
```



continue 文

- continue : 途中で次の繰り返しに進む

```
1 while ( 条件A ) {  
2   文S1  
3   if ( 条件B )  
4     continue;  
5   文S2  
6 }
```



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

```
1  int main() {
2      for (int i = 0; i < 5; i++) {
3          std::cout << "i: " << i << ", j: ";
4
5          for (int j = 0; j < 7; j++) {
6              std::cout << j << " ";
7              if ( j == 2 ) break; // 抜けるのは j のループ
8          }
9          std::cout << "\n";
10     }
11 }
```

```
% ./a.out
i: 0, j: 0 1 2
i: 1, j: 0 1 2
i: 2, j: 0 1 2
i: 3, j: 0 1 2
i: 4, j: 0 1 2
```

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、見つければ終了

```
1  int main() {
2      std::vector a {3, 6, 2, 8, 1, 5, 2, 9, 3, 7};
3      const int x {5}; // 探す対象
4
5      int i;           // 見つけた場所
6      for (i = 0; i < a.size(); i++)
7          if (a[i] == x) break;
8
9      std::cout << ( i < a.size() ? "found" : "not found" ) << "\n";
10 }
```

```
% ./a.out
found
```

pythonとの比較： 繰り返し文

```
1 // cppのプログラム, includeは省略
2 int main() {
3     int x {1};
4     while( x < 10 ) { // 条件を満たす間ループ
5         x *= 2;
6         std::cout << x << ", ";
7     }
8
9     for( int i = 0; i < 3; i++ ) { // 3回ループ
10         std::cout << i << ", ";
11     }
12
13     std::vector<int> array { 2, 3, 5, 7 };
14
15     for( int i = 0; i < array.size(); i++ ) { // 配列の要素ごとのループ①
16         std::cout << array[i] << ", ";
17     }
18
19     for( int e : array ) { // 配列の要素ごとのループ②
20         std::cout << e << ", ";
21     }
22 }
```

```
1 # pythonのプログラム
2
3 x = 1
4 while x < 10:
5     x *= 2
6     print( x, end="," )
7
8
9 for i in range(3):
10     print( i, end="," )
11
12
13 array = [2, 3, 5, 7]
14
15 for i in range( len(array) ):
16     print( array[i], end="," )
17
18
19 for e in array:
20     print( e, end="," )
```

2,4,8,16, 0,1,2, 2,3,5,7, 2,3,5,7,