

ブロック化

→ 「効率的に管理するためには」

ブロック化・モジュール化

- ・ 「構造化プログラミング」の手法
- ・ C 言語, Java 言語などにおける { ... } で囲まれた部分
- ・ HTML では、<タグ> ... </タグ> で囲まれた部分
↓
- ・ プログラミング言語における「関数」



利点

- 関係する部分をひとまとめに
- 仕事・機能の単位をブラックボックスとして扱える
- 中身について知らなくても利用できる

プログラムをわかりやすくする方法

構造化プログラミング (structured programming)

- ・ ダイクストラ (E. W. Dijkstra)
goto 文 (無条件ジャンプ) はプログラムを複雑化する原因

- ・ ヤコビーニ (Jacopini) ベーム (Böhm)
ストラクチャード定理 (構造化定理)

適正プログラム

1つの入り口と1つの出口を持つように設計される

↓

「適正プログラム」は基本的な構造単位の組み合わせで記述可

アルゴリズムの基本形

→ 「フローチャートの基本部品」

ブロック化を積極的に利用する + それぞれのブロック構造の適正化！

アルゴリズム (フローチャート) の基本形

↓

「ストラクチャード定理 (構造化定理)」

↓

基本形は全て「入り口」と「出口」が 1つ!! (2つ以上でも 0でも駄目!)

1. **順次**, 逐次, 連接 (sequence)
→ 上から下へ順番に
「プログラムにおける文の並び・処理に書いてある順に次々と実行」

2. **分岐**, 選択, 判断 (selection) if ~ then ~ else
→ 上から下へのジャンプあり
「条件に応じて2つの処理の一方を選択(if)」
「プログラミング言語によっては3つ以上の分岐あり(switch)」

3. **ループ**, 繰り返し (iteration) while ~ or do ~ while or for ~
→ 下から上へのジャンプあり
「処理の繰り返し」

ループ繰り返し判定をどこに置くか？

- | | | |
|---|----------|--------------------------------|
| { | 繰り返し処理の前 | → 3a. 前判定ループ (while) |
| | 繰り返しの内部 | → 中判定ループ (break) |
| | 繰り返し処理の後 | → 3b. 後判定ループ (do-while) |

アルゴリズム(フローチャート)は上記 1, 2, 3(= 3a, 3b) の組み合わせで記述！
(教科書により表現が若干違っているのだが・・・基本はこういうこと！)

