ソフトウェア工学 第9回
― プログラミング ―

大連理工大学・立命館大学 国際情報ソフトウェア学部

大森 隆行

講義内容

- プログラミング
  - 概説
  - フローチャート
  - 構造化プログラミング
  - データ構造に基づく設計
  - プログラミングの実践

プログラミング

- プログラム(program)
  - プログラマが記述したソースコード
  - システム上で動作するオブジェクトコード(マシンコード)
- プログラミング(programming)
  - 仕様(要求仕様や設計仕様)に基づきプログラムを作成する作業
  - 実装(implementation) とほぼ同義
  - 要求が正しく記述されたとしても、実装工程で誤りが混入すると、システムは要求を満たさない
    → 簡潔でわかりやすいプログラムを、誤りなく確実に作ることが重要

プログラミングに関連する作業

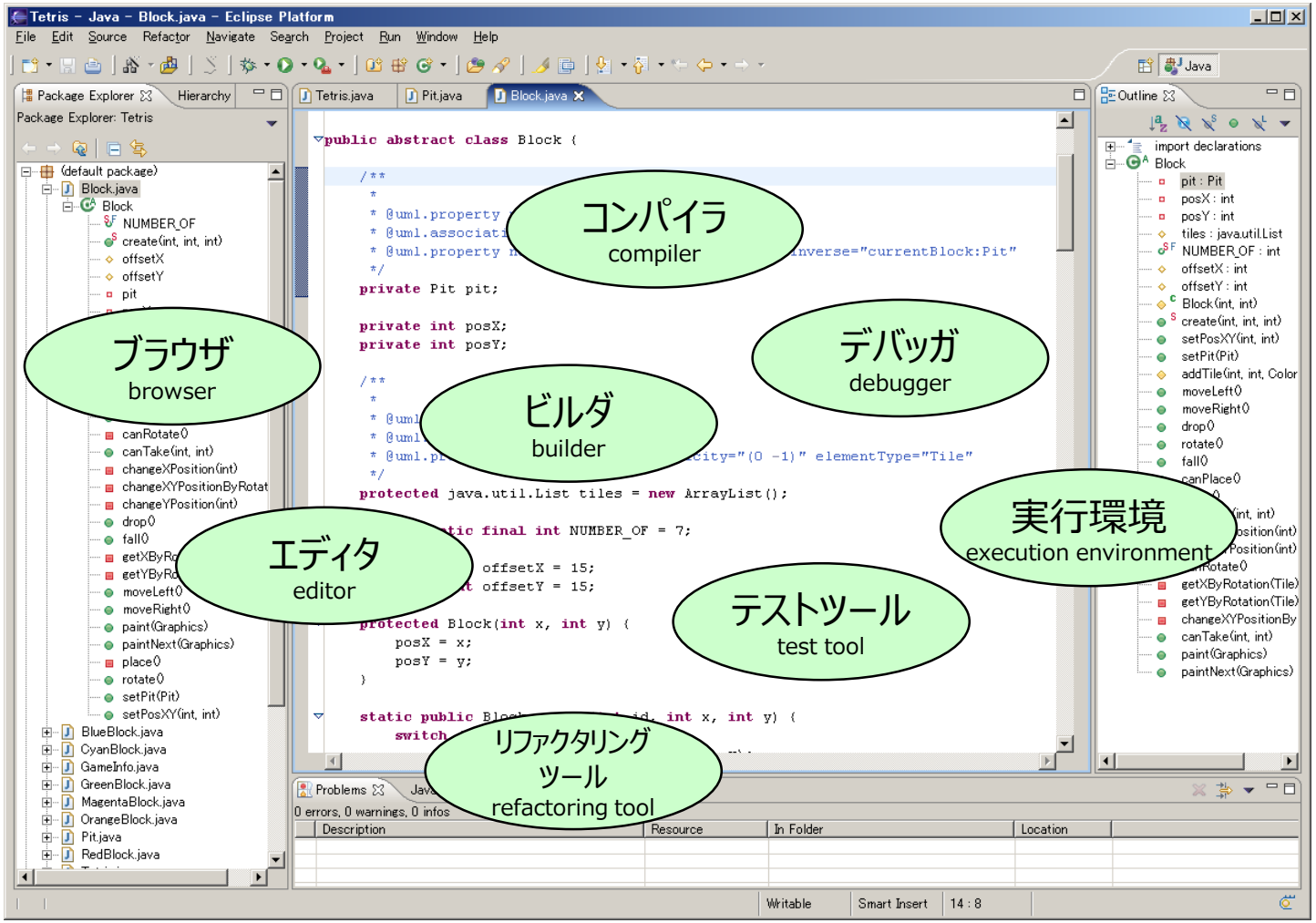
- モジュール設計
  - (前回参照)
- コーディング (coding)
  - ソースコードを記述する
  - プログラミング言語による
- デバッグ (debug)
  - プログラムのバグを除去する
- 単体テスト (unit test)
  - テストケースを用いてプログラムが
    正しく動作するかモジュールごとに確認する

ソフトウェア開発環境

- バッチ型プログラミングツール（1950～60年代）
  - 高級言語コンパイラ
- 対話型プログラミングツール（1970年代）
  - ドキュメント作成支援、エディタ、デバッガ
- 統合プログラミング環境（1980年代後半）
  - 構造化技法支援、ビジュアル化
  - CASE(computer-aided software engineering)
- 自動化（1990年代）
  - 統合開発環境(IDE: integrated development environment)
  - リポジトリ(repository)の活用
    - ソフトウェア開発時の情報を集中管理するための保管庫
- オープン化（1990年代後半～）
  - 分散開発支援環境
    - CVS, Subversion, Mercurial, Gitなど
  - プラグインの作成と公開

統合開発環境(IDE)

- Eclipse, IntelliJ IDEA, MS Visual Studio, ...



プログラミングパラダイム

- プログラミングパラダイム(programming paradigm)
  - プログラムの作り方に関する規範
  - 設計手順、プログラム構造、プログラムの記述方法を規定
  - プログラミングの際に、何に着目して問題を整理するのか、何を中心にプログラムを構成するのかの方向付けを与えるもの
- パラダイムの例
  - 手続き型：処理手順を命令文で記述
    - FORTRAN, COBOL, BASIC, PASCAL, C, Ada等
  - 関数型：入出力関係を表現する関数とその呼び出しで記述
    - Lisp, Scheme, ML, Haskell等
  - 論理型：入出力関係を述語論理(事実と規則)で記述
    - Prolog等
  - オブジェクト指向：データとその操作をカプセル化したオブジェクトとその間のメッセージ通信で記述
    - Smalltalk, C++, Java, C#, Ruby等

確認問題

- 以下の説明に合う語を下の枠内の語群から選択せよ。
  - 仕様(要求仕様や設計仕様)に基づきプログラムを作成する作業
  - バグを除去する作業
  - 様々な開発支援ツールを一つの環境に統合したソフトウェア開発環境
  - プログラミングの際に、何に着目して問題を整理するのか、何を中心にプログラムを構成するのかの方向付けを与えるもの

プログラミング、プログラミングパラダイム、デバッグ、統合開発環境(IDE)

- 以下の説明に合うプログラミングパラダイムを下の枠内の語群から選択せよ。
  - 処理手順を命令文で記述
  - 入出力関係を述語論理で記述
  - 入出力関係を表現する関数とその呼び出しで記述
  - データとその操作をカプセル化したオブジェクトとオブジェクトの間のメッセージ通信で記述

オブジェクト指向、手続き型、論理型、関数型

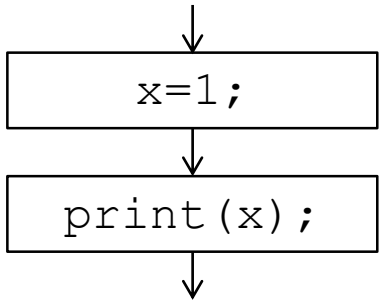
講義内容

- プログラミング
  - 概説
  - フローチャート
  - 構造化プログラミング
  - データ構造に基づく設計
  - プログラミングの実践

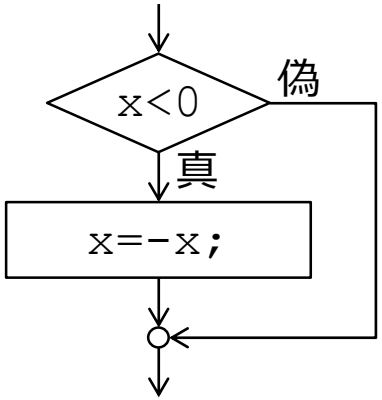
フローチャート

- プログラムの実行の流れを図示する

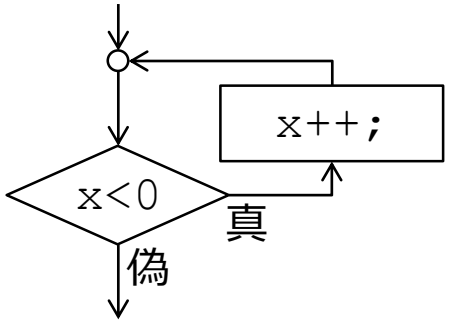
```
x = 1;
print(x);
```




```
if(x<0){
   x = -x;
}
```

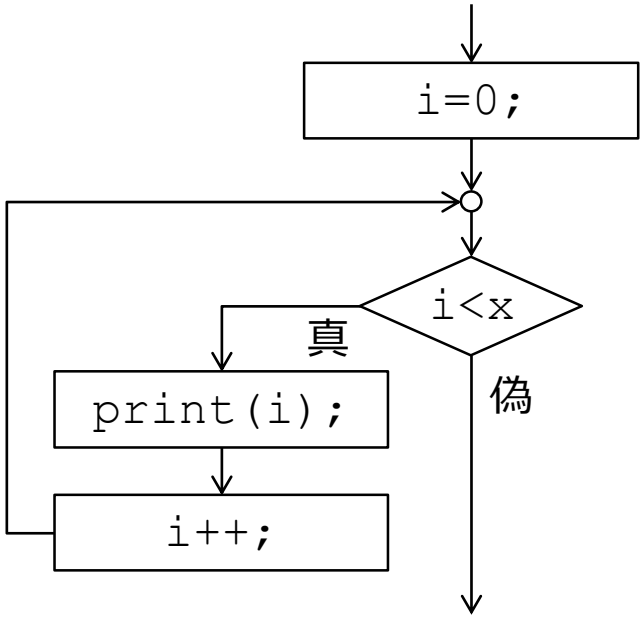


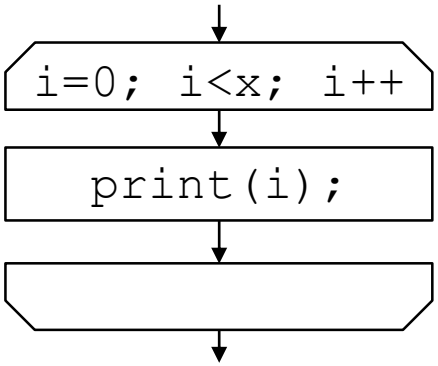

```
while(x<0){
   x++;
}
```



フローチャート

```
for(i=0; i<x; i++){
  print(i);
}
```





確認問題

- 以下のC言語のプログラムの制御の流れを示すフローチャートを記述せよ。

```
int a=4;
int b=0;

while(a>=0){
  if(b>100){
    break;
  }
  if(a%2==0){
    b+=a;
  }
  a--;
}
print(b);
```

講義内容

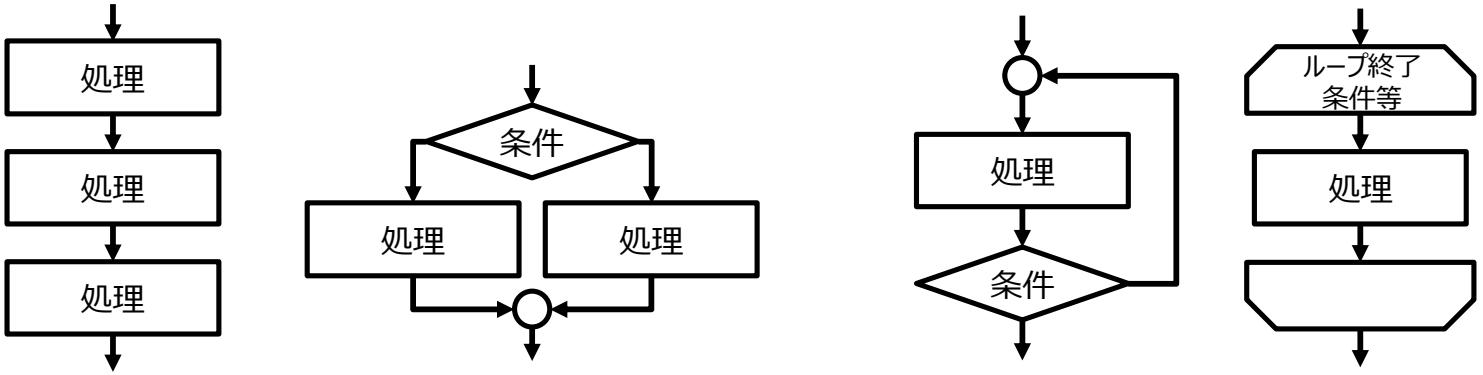
- プログラミング
  - 概説
  - フローチャート
  - 構造化プログラミング
  - データ構造に基づく設計
  - プログラミングの実践

プログラミングに求められるもの

- 昔の記述規範
  - メモリの使用量が少ない
  - 処理時間が早いこと
    - ⇩ But,
  - 理解しにくくなる傾向がある
- 近年のプログラミング
  - 理解しやすいプログラムの記述を目指す
    - 構造や振る舞いを容易に把握できる
    - 正しさを検査しやすい
    - バグの特定が容易

構造化定理

- すべての適正プログラムが連接、選択、反復の組み合わせで記述可能であることを示したもの
  - 適正プログラム
    - 制御の流れに対して、入口と出口が必ず1つずつある
    - 入口から出口への制御の流れに、すべての命令が関与する(絶対に実行されない命令は存在しない)

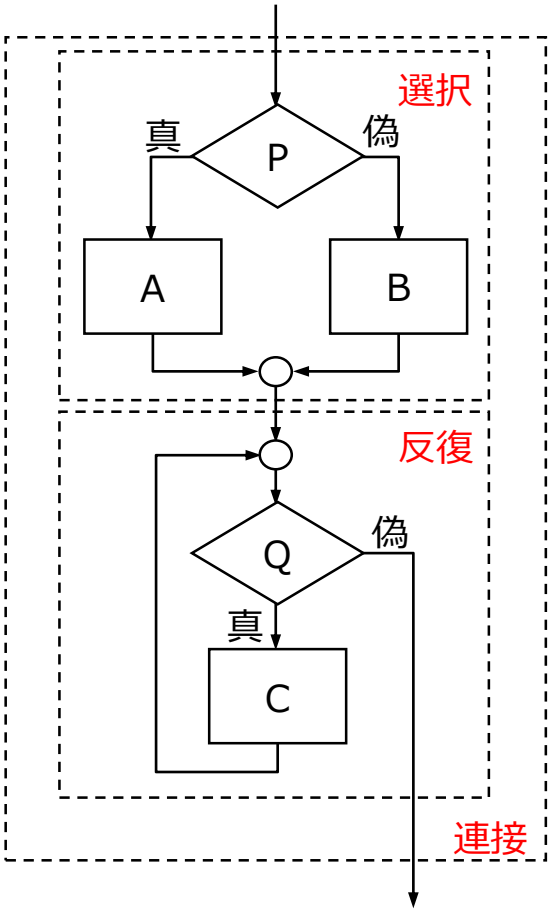


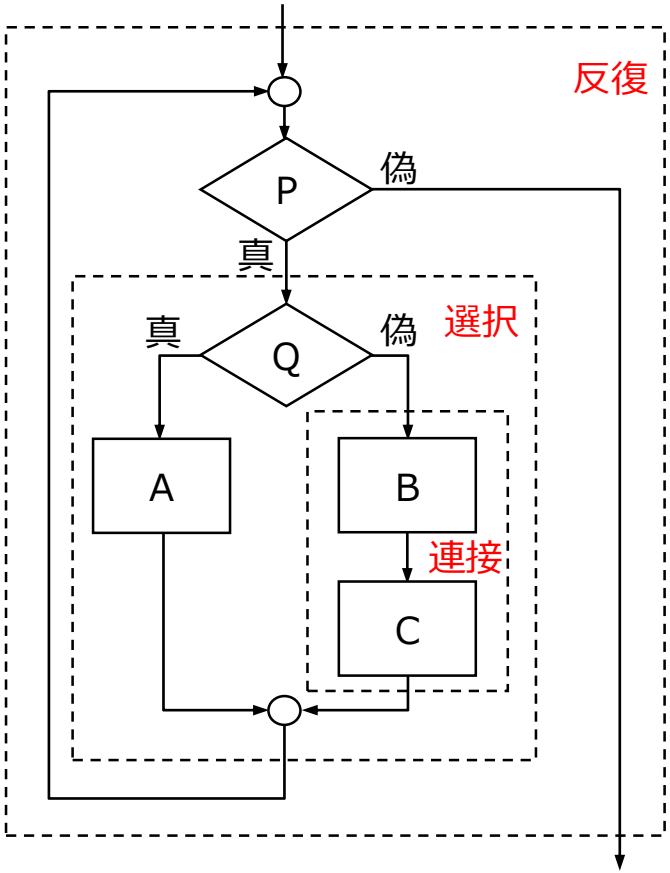
連接　選択　反復

※分岐の接合点を示す○を省略する記法もある
※分岐先には真偽等の分岐条件を明記する必要がある

適正プログラムの例

- 制御の流れに対して、入口と出口が必ず1つずつある
- 入口から出口への制御の流れに、すべての命令が関与する
(絶対に実行されない命令は存在しない)





構造化プログラミング

- 段階的詳細化等によりプログラムを構造化するための
  プログラミング技法
- ダイクストラ(E.W. Dijkstra)により提唱

- GOTO文の是非
  - 任意の命令に制御を移動可能なGOTO文はできる限り使わない
  - 関数からの戻り(return)、繰り返しからの脱出(break)、
    例外処理等は必要
- 言語との関係
  - 多くの手続き型言語で構造化プログラミングの
    考え方が取り入れられている
  - オブジェクト指向言語等を採用した場合でも
    制御構造は記述するため、考え方として有効

プログラミングスタイル

- コーディング規約
  - コーディングにおける約束事
    - 命名規則(変数名、関数名の付け方)
    - 字下げ(indentation)やスペースの配置
    - イディオム(idiom)(=特定のプログラミング言語でよく用いられる実装コードやアルゴリズム)の活用
- デザインパターン(design pattern)
  - オブジェクト指向プログラムにおける良い設計の集まり
    - GoF(E.Gamma, R.Helm, R.Johnson, J.Vlissides)によるデザインパターン 23種が有名

確認問題

- 以下の各文は正しいか。○か×で答えよ。
  - 近年では、プログラムの理解しやすさは重視されなくなりつつある。
  - 構造化プログラミングでは、GOTO文の乱用は避けるべきだとされている。
  - 構造化プログラミングでは、GOTO文は絶対に使ってはならない。
- 以下の各文の空欄を埋めよ。
  - 構造化定理とは、すべての適正プログラムが(１)、(２)、(３)の組み合わせで記述可能であることを示したものである。
  - 適正プログラムとは、制御の流れに対して、(４)と(５)が必ず1つずつあるプログラムである。
- 以下の各説明に合う語を答えよ。
  - コーディングにおける約束事。
    例えば、変数名の付け方や字下げの仕方など。
  - オブジェクト指向プログラムにおける良い設計の集まり。
    GoFによるものが有名。

講義内容

- プログラミング
  - 概説
  - フローチャート
  - 構造化プログラミング
  - データ構造に基づく設計
  - プログラミングの実践

データ構造に基づく設計

- システム内で扱うデータの基本的な特性に着目した
  モジュール設計技法

- データ中心アプローチ (DOA: data-oriented approach)
  - 業務(プロセス)の変更に強い
  - e.g., ジャクソン法、ワーニエ法

- プロセス中心アプローチ (POA: process-oriented -)
  - 業務(プロセス)の変更に弱い
  - e.g., STS分割、TR分割

ジャクソン法

- 入力データ構造と出力データ構造の対応関係からプログラムの論理構造を導出

手順

1. データ構造の定義
   - 入力データ構造と出力データ構造を分析
     → 4つの構成要素（基本、連接、反復、選択）でデータ構造図を作成
2. データ構造の対応付け
   - 入力データ構造図と出力データ構造図の構成要素間の対応関係を決定
3. プログラム論理構造の決定
   - 出力データ構造を基に論理構造を定義
     - 入力データから出力データへ変換する処理を記述
   - 構造が不一致のとき、中間のデータ構造を導入
     - 入出力データに対する処理を追加

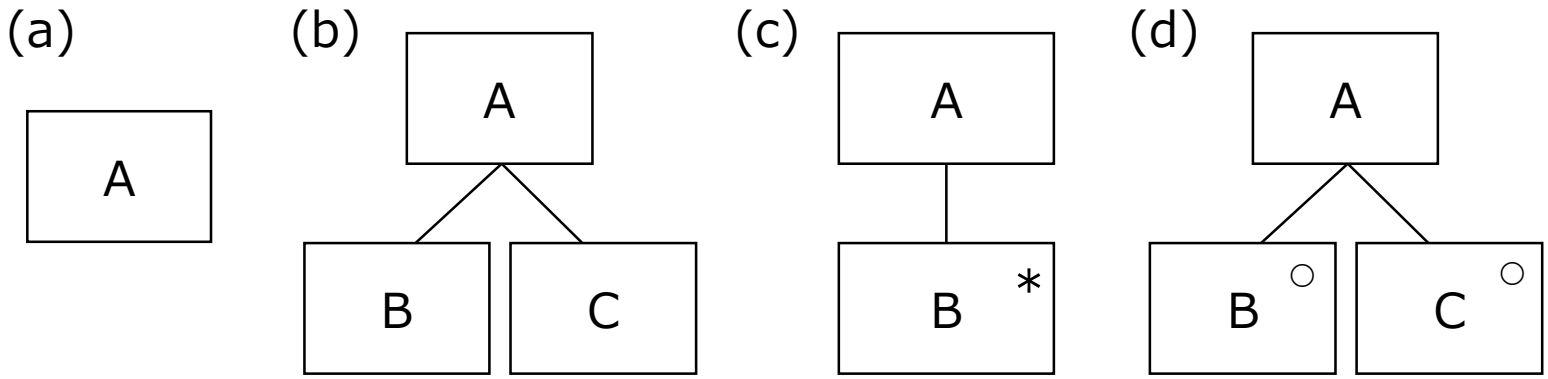
ジャクソン法 - データ構造図の構成要素

(a) 基本
- これ以上分割できない構成要素（1つのデータ項目に対応）

(b) 連接
- 異なる複数の基本要素からなる構成要素
  （複数の項目を持つレコードに対応）
- それぞれの構成要素は左から右に順に1度だけ出現
  (下図であれば、Aの内部には、Bが1回出現し、次にCが1回出現する)

(c) 反復
- 同一の構成要素が繰り返し現れる構成要素

(d) 選択
- 複数の構成要素のうちどれか1つを選択する構成要素



ジャクソン法 - 入力データ構造図の例

学生ファイル
(学籍番号順)

入力データ構造図

10001	A	2	SE	合
10001	A	2	OS	否
10002	B	2	NW	合
⋮				
10055	X	3	SE	否
10056	Y	3	SE	合
10058	Z	3	OS	合

同一学生

同一学年

学生レコード

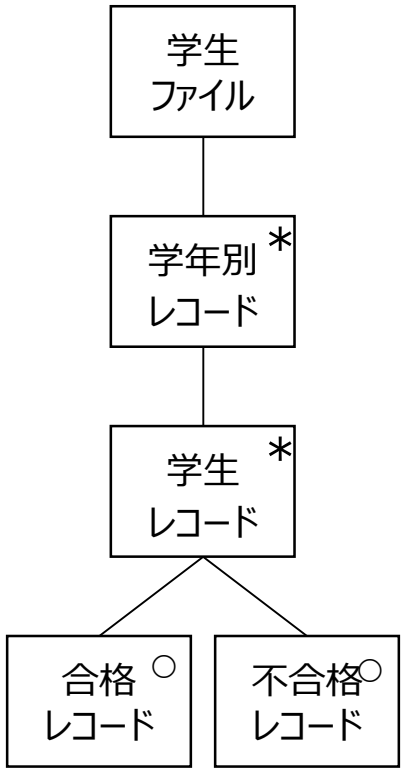
学籍番号

氏名

学年

科目

合否



ジャクソン法 - 出力データ構造図の例

合格者一覧(印刷結果)

出力データ構造図

合格者一覧（2013年度）

2回生

学籍番号	氏名	科目
10001	A	SE
10002	B	NW
…		

合格者数： 40

3回生

学籍番号	氏名	科目
10056	Y	SE
10058	Z	OS
…		

合格者数： 25

総合格者数： 65

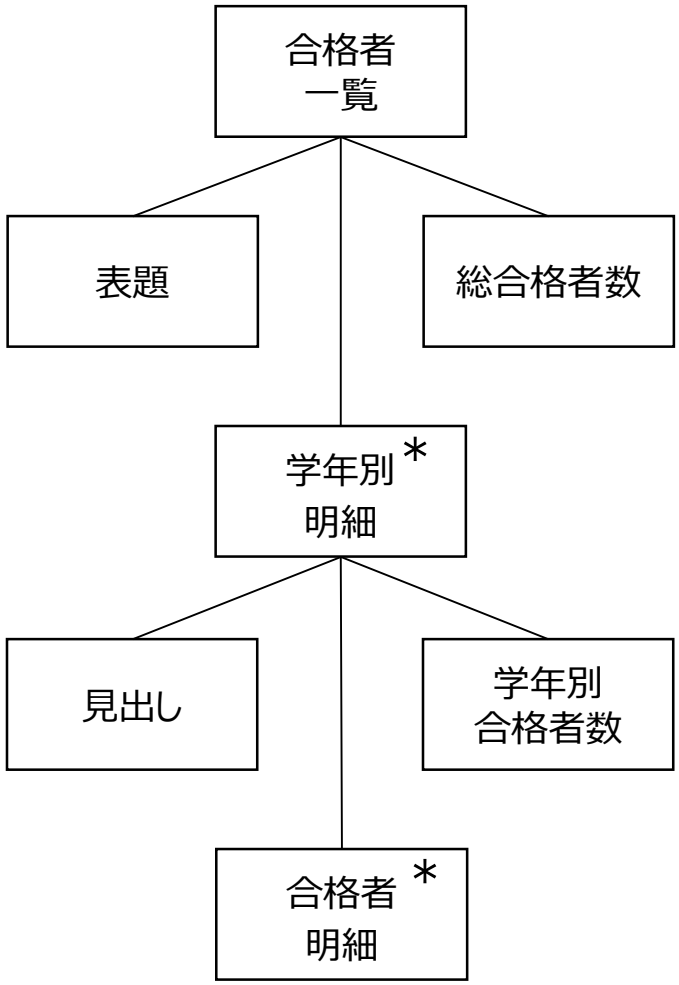
表題

見出し

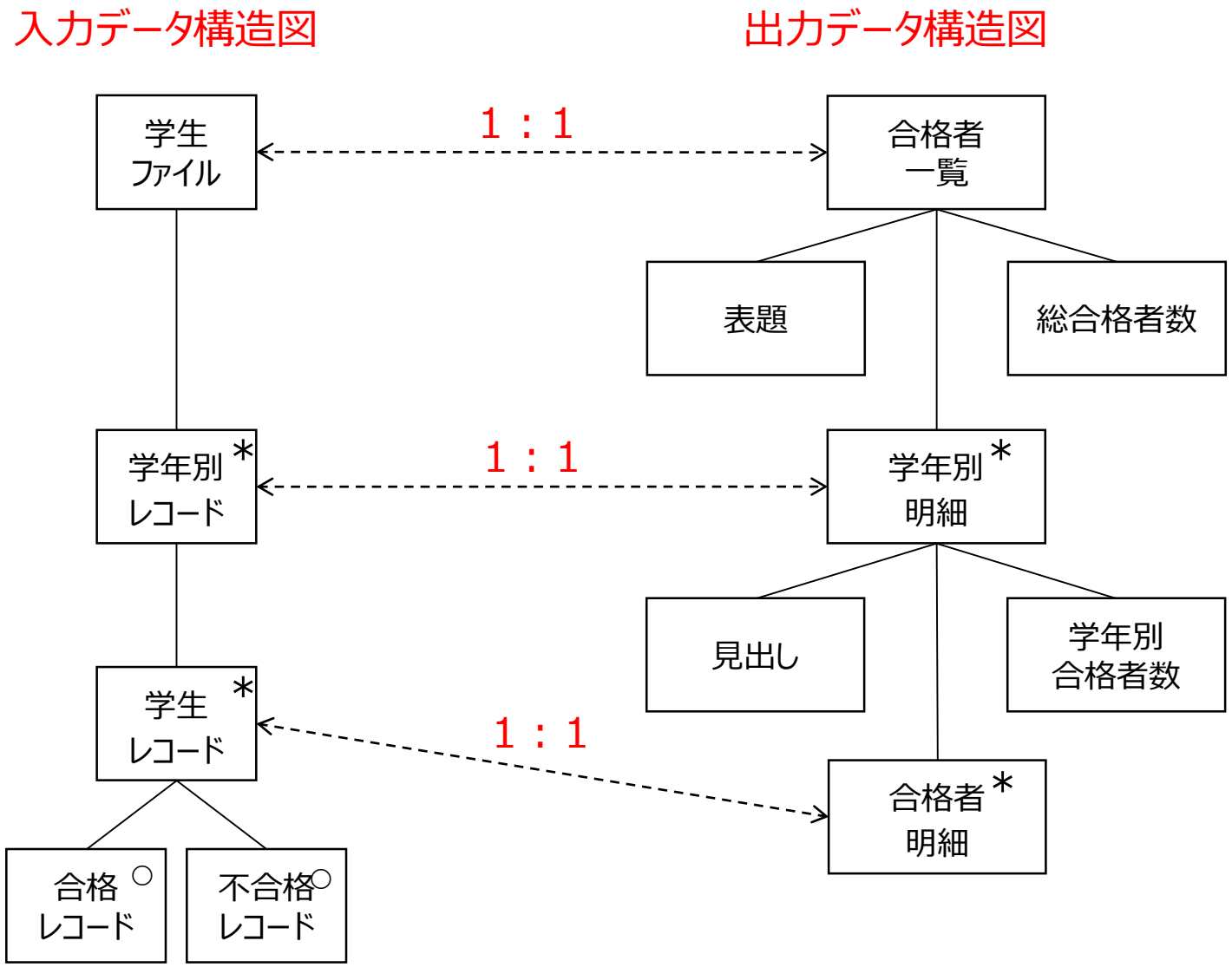
合格者明細

学年別合格者数

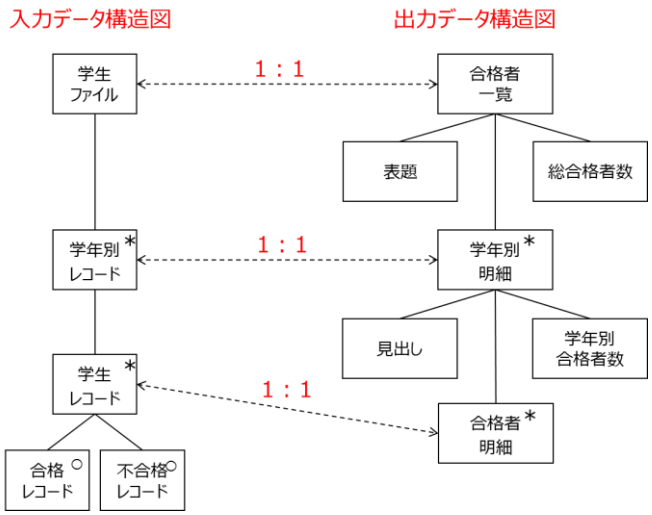
総合格者数

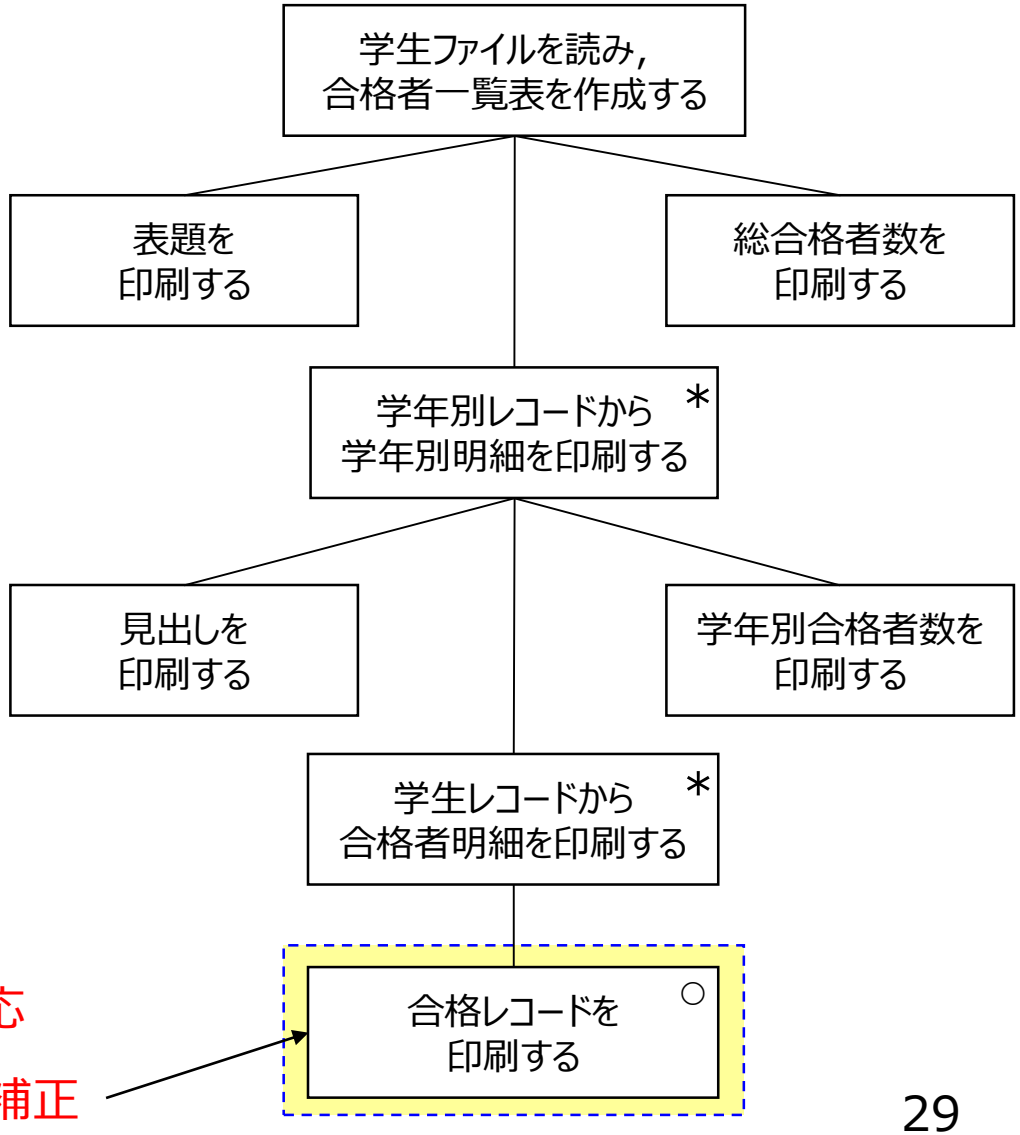


ジャクソン法 - 構成要素間の対応



ジャクソン法 - プログラム構造図の導出





基本的には出力データ構造図と対応

入力データ構造図からの補正

ワーニエ法

- 入力データ構造と出力データ構造から直接プログラムの論理構造を決定

→cf. ジャクソン法：入力データ構造と出力データ構造の対応関係からプログラムの論理構造を導出

手順

(1)データ構造の定義

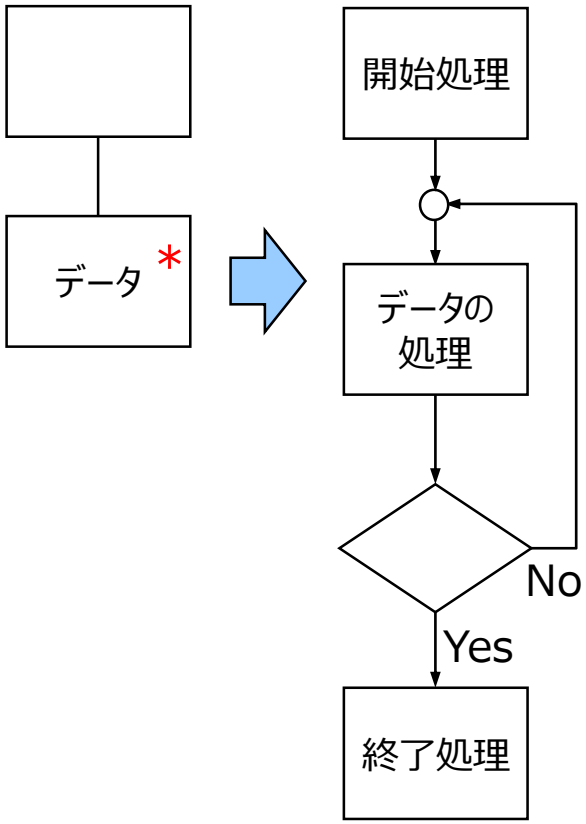
- 入力データ構造と出力データ構造を分析し、
  4つの構成要素（基本、連接、反復、選択）でデータ構造図を作成

(2)プログラム論理構造の決定

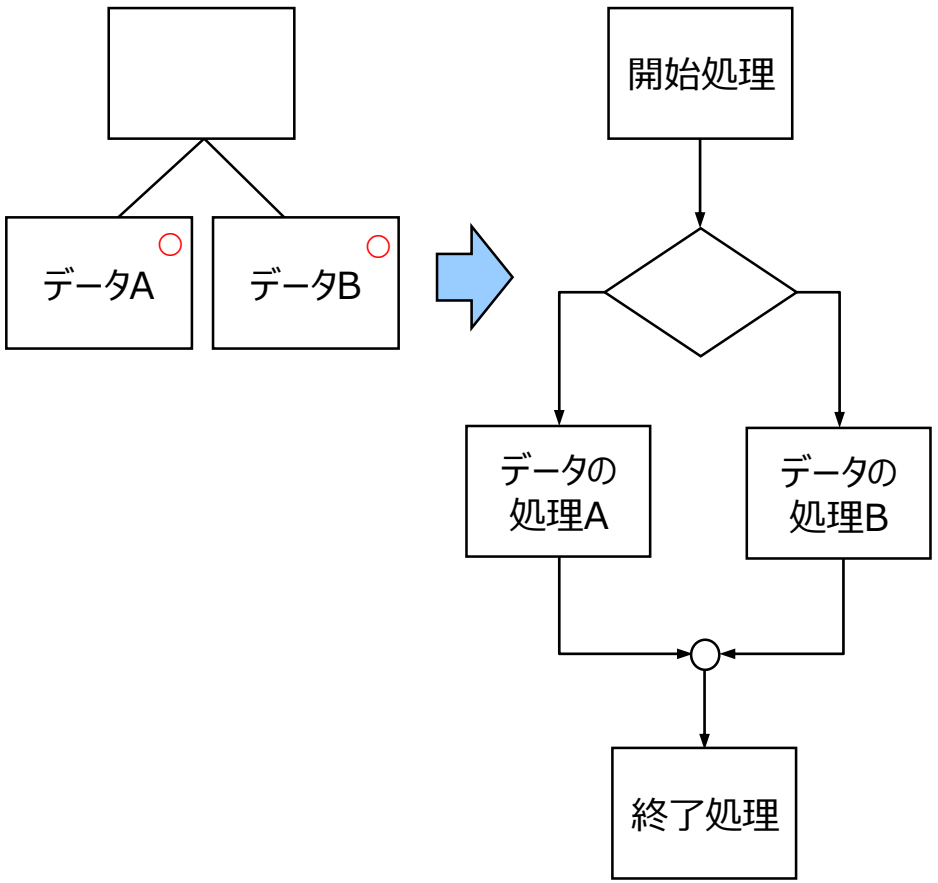
- 入力データ構造を基にプログラムの論理構造を決定
  - 反復のデータ構造　→　反復の論理構造
  - 選択のデータ構造　→　選択の論理構造
  - 入力データ構造にあり、出力データ構造になし　→　無視
  - 入力データ構造になし、出力データ構造にあり　→　入力データの加工
  - 各論理構造の前後に開始部と終了部を付加

ワーニエ法 - データ構造とプログラム制御構造

(a) 繰返し

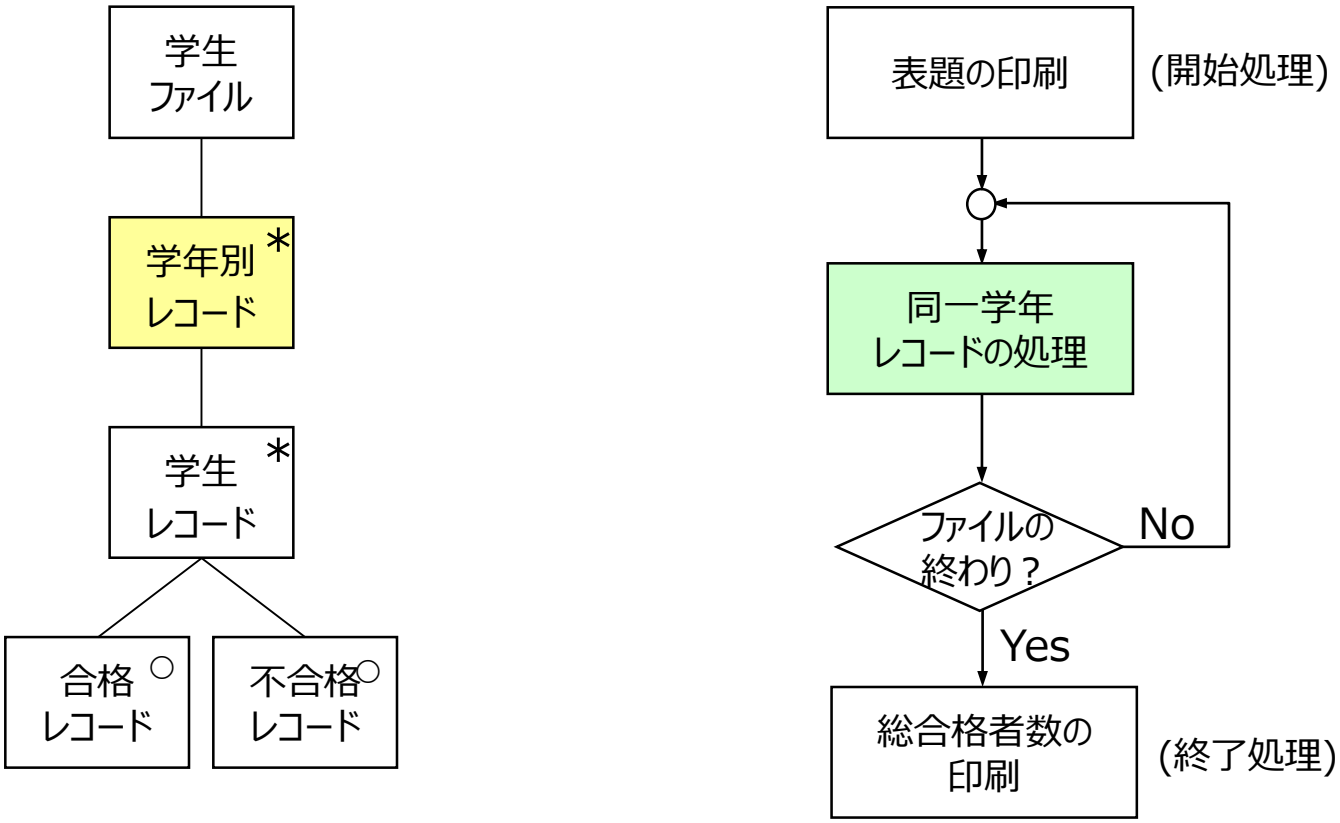


(b) 選択

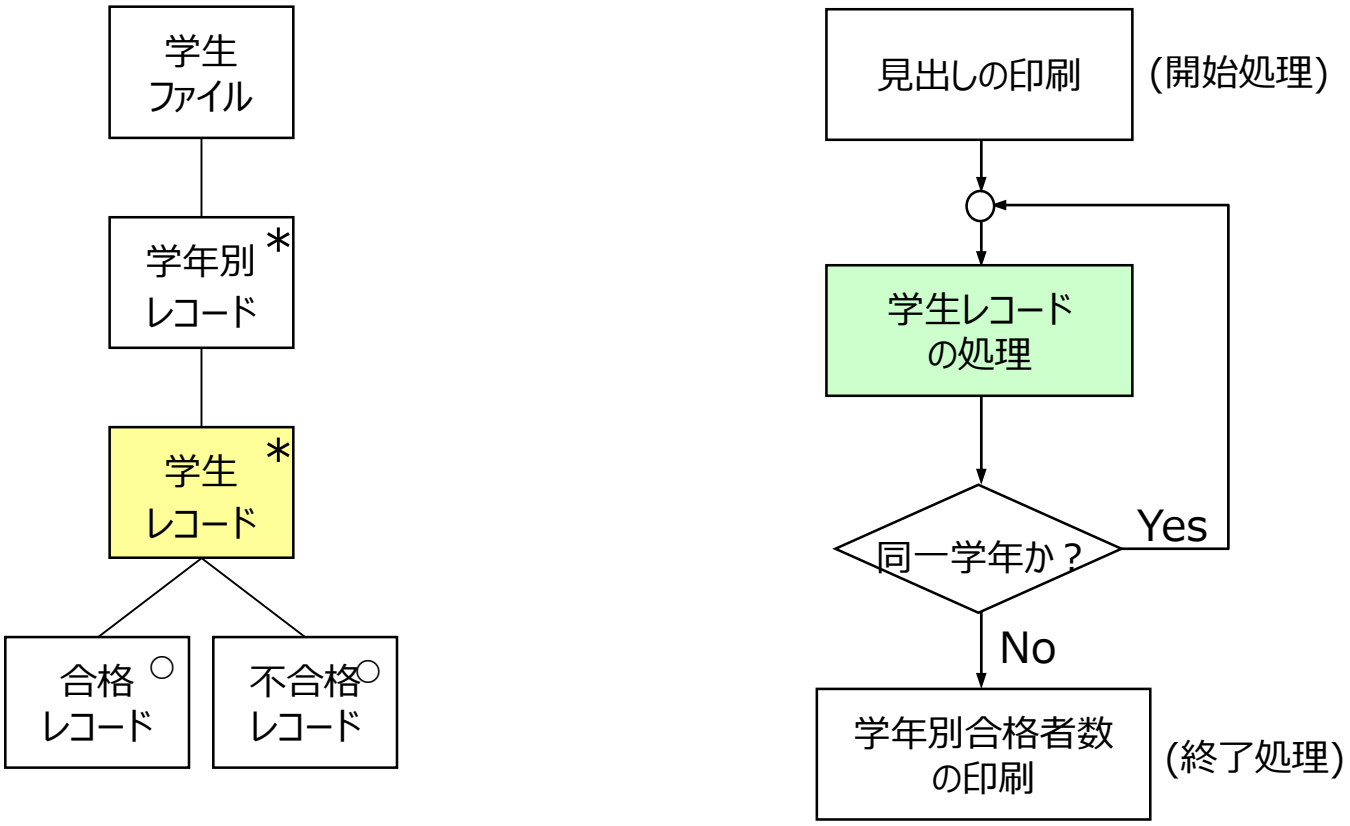


ワーニ工法 - 制御構造への変換例

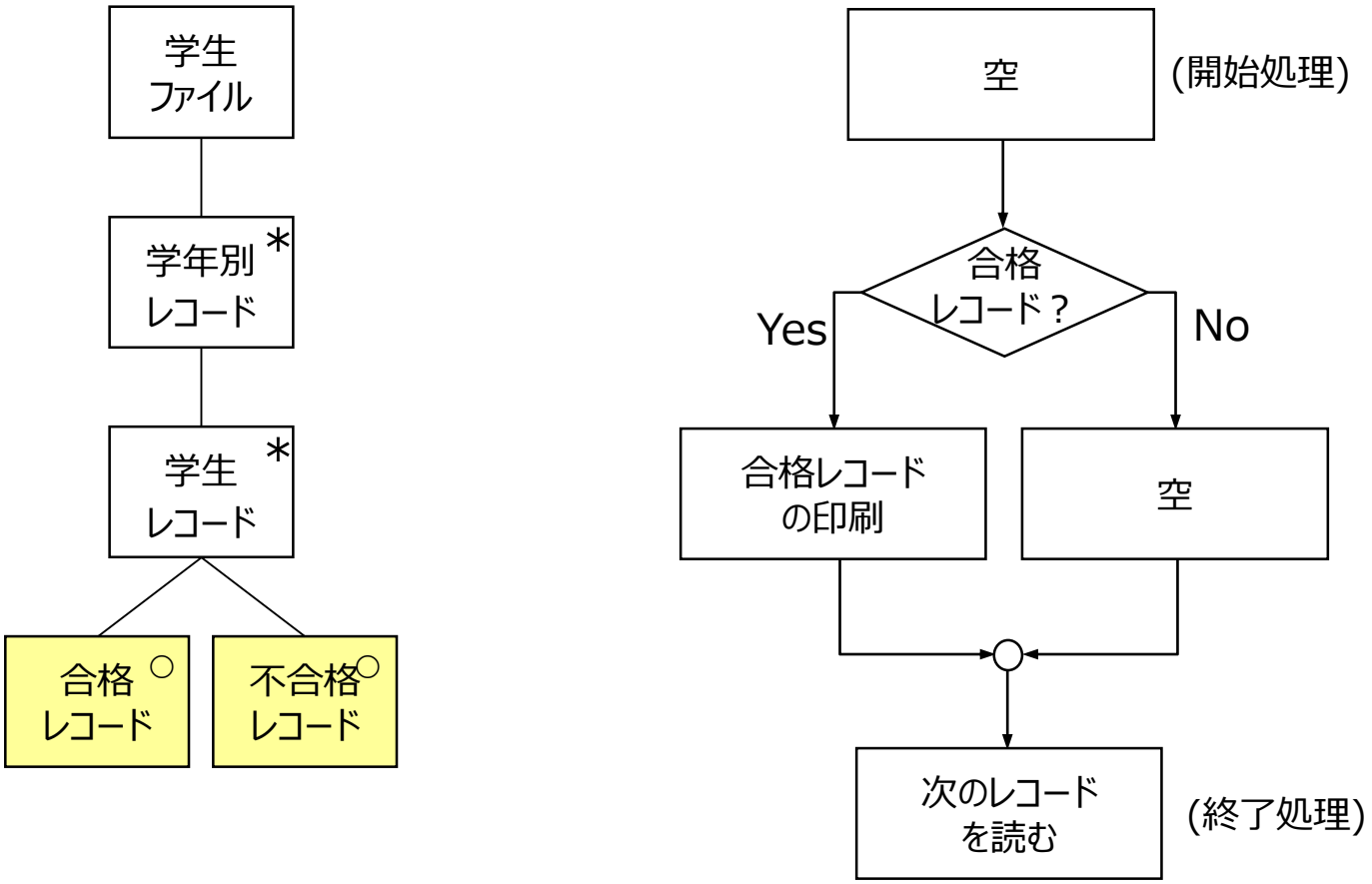
入力データ構造図 → プログラム制御構造



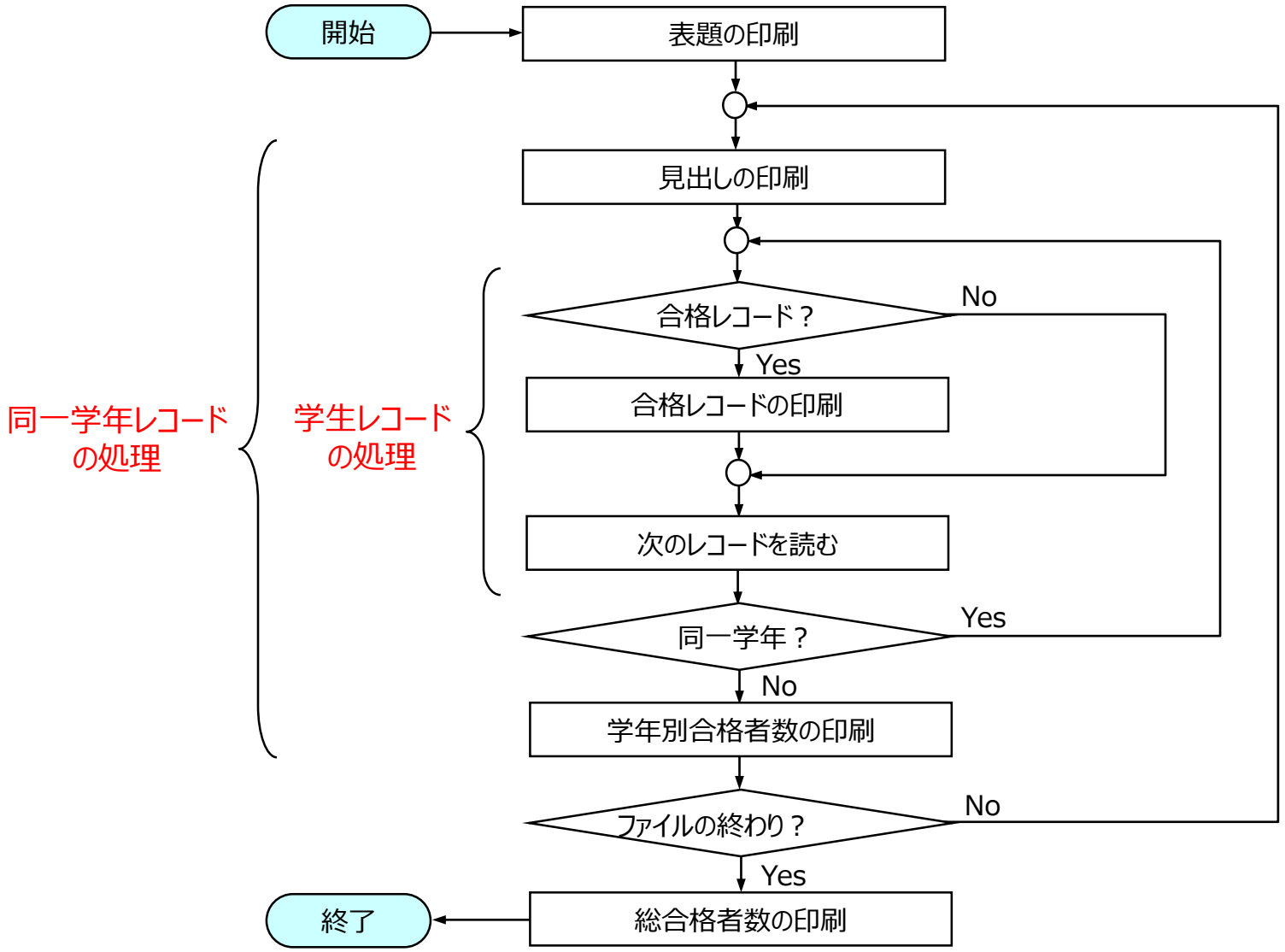
ワーニ工法 - 制御構造への変換例



ワーニ工法 - 制御構造への変換例



ワーニ工法 - 制御構造への変換例



ジャクソン法・ワーニエ法を用いる際の注意

- あらかじめデータ構造が十分に標準化されている必要
  - 重複データをなくす等の処理(正規化)
- ジャクソン法はモジュール分割技法と捉えることもできる
  - 導出される構成要素は非常に小さいため、モジュール設計というよりプログラミング技法と捉えることも可能
- ワーニエ法では命令レベルでプログラム構造が導出される
  →プログラミング技法と捉える方が自然

確認問題

- システム内で扱うデータの基本的な特性に着目したモジュール設計技法を(　１　)アプローチと呼ぶ
- ジャクソン法では、入力データ構造と出力データ構造を分析し、4つの構成要素（基本、(　２　)、(　３　)、(　４　)）でデータ構造図を作成する。
- ワーニエ法では、(　５　)データ構造から直接、プログラムの制御構造を導出する。

講義内容

- プログラミング
  - 概説
  - フローチャート
  - 構造化プログラミング
  - データ構造に基づく設計
  - プログラミングの実践

ビルド (build)

- コンパイル、テスト、ドキュメント生成、パッケージング(実行ファイルを配布形式にまとめる)、リリース等を一括して行う
  - ソースコード修正のたびに発生する定型処理を自動化
- ビルドツール
  - Make
    - UNIX系OSにおける標準コマンド (makeコマンドで実行)
    - Makefileにファイル間の依存関係や処理内容を記述
  - Ant
    - Java用のビルドツール
    - ビルドスクリプト(ビルドの手続き)をXMLで記述
  - Maven
    - POM (project object model) という概念を導入
      - 規約(ライブラリの依存関係等)を pom.xml に記述し、これに基づいてビルド
      - ライブラリの依存関係をMaven Centralで管理
  - Gradle
    - スクリプト言語Groovyで規約を記述

実装に役立つ知識

- プログラミング言語、パラダイム
- ライブラリ、API (application programming interface)
  - API: ソフトウェアが機能を提供する際のインタフェース (クラスや関数等)
  - 通常はリファレンスも公開されている
- 各種勧告(technical report)、RFC(request for comments)
  - 例えば https://www.w3.org/TR/ には様々なWeb関係のプロトコル仕様が掲載されている
  - TR：確定した仕様、RFC：意見受付中の仕様
- コードの性能
  - 時間計算量、空間計算量
  - 適切なデータ構造とアルゴリズムやAPI、依存ライブラリの選択

コーディングスタイル・標準

- 個人開発者や開発チームの中で、
  標準的なソースコードの書き方を決めたもの
- 一般的に推奨されるコーディングスタイルを標準化したコーディング標準も存在
  - CERT C コーディングスタンダード
  - IPA 組込みソフトウェア開発向けコーディング作法ガイド[C言語版]
- 例
  - 命名法
    - スネークケース snake_case、キャメルケース camelCase
  - 省略しすぎていないか
  - 英単語である (× eitango → ○ word)
  - 必要がない箇所でグローバル変数を使わない
  - 適切なサイズにモジュール分割する
    - Blob(God class)を避ける
  - コードクローン(code clone)を減らす

確認問題

- 以下の各説明に合う語句を語群から選べ。
  - ソフトウェアが他のソフトウェアに機能を提供する際のインタフェース
  - コンパイルや配布ファイルの生成等をまとめて自動化する
  - 確定した技術仕様をまとめた文書
  - 標準的なソースコードの書き方、作法

語群：コーディングスタイル、
プログラミングパラダイム、ビルドツール、
勧告、RFC、API

参考文献

- 「ソフトウェア工学」
  高橋直久、丸山勝久 著、森北出版、2010
- 「Gradle徹底入門」
  綿引琢磨、須江信洋、林政利、今井勝信 著、翔泳社、2014