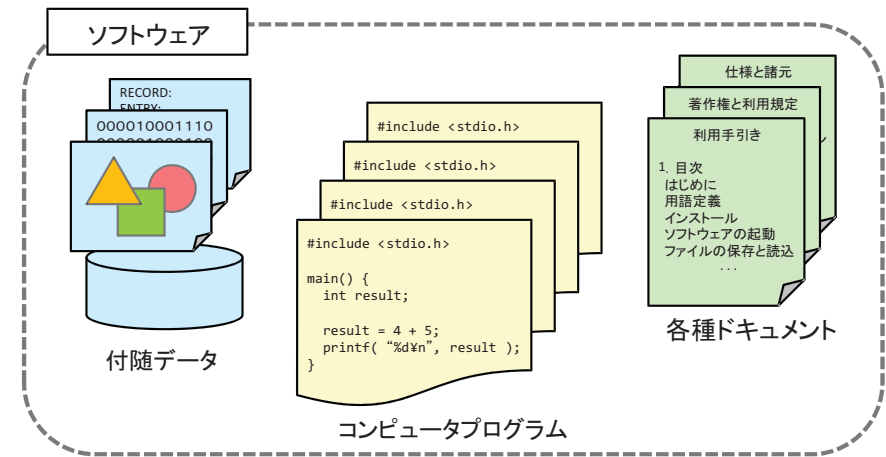


コンピュータ科学II

担当：武田敦志 <takeda@cs.tohoku-gakuin.ac.jp>
http://takeda.cs.tohoku-gakuin.ac.jp/

ソフトウェア

■ソフトウェアの構成



プログラミング(1)

■プログラミング言語

実行ファイルの内容は『**機械語** + 演算用のデータ』
コンピュータは**機械語**の内容に従って動作する
人間にとって、**機械語**を理解することは難しい

人が最も使いたい言語は**自然言語**

人間は**自然言語**を使って物事を考える
コンピュータは**自然言語**を理解できない



実行ファイルを作成するために**プログラミング言語**が必要

プログラミング(2)

■プログラミング言語の種類

●低水準言語

機械語

アセンブリ言語

●高水準言語

手続き型プログラミング言語

オブジェクト指向プログラミング言語

関数型プログラミング言語

論理型プログラミング言語

プログラミング(3)

■高水準言語と低水準言語

●高水準言語

人の考え方に近いプログラミング言語

⇒ コンピュータへの論理的な命令を表現する

C言語, Java, Perl, Lisp, Python など

●低水準言語

コンピュータの処理に適したプログラミング言語

⇒ コンピュータへの具体的な命令を表現する

機械語, アセンブリ言語, 中間言語

プログラミング(4)

■プログラミング言語

●人が理解しやすい言語

⇒ **自然言語**

●コンピュータで実行できる言語

⇒ **機械語**



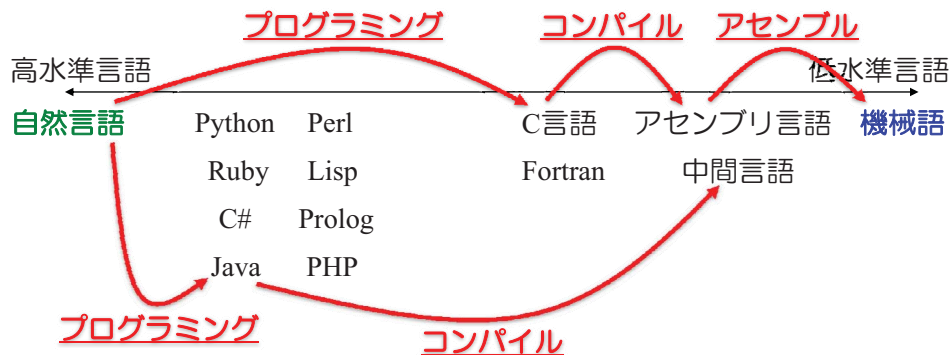
●人が理解できる+コンピュータも理解できる言語

⇒ **プログラミング言語**

通常は**プログラミング言語**で書かれたプログラムを**機械語**に翻訳して実行する

プログラミング(5)

■コンパイルと実行



プログラミング(6)

■問題 (考えてみよう)

現在までに、多くの種類の高水準プログラミング言語が開発されてきた。なぜ、これらのプログラミング言語が必要とされているのだろうか？

高水準プログラミング言語には「手続型」「オブジェクト指向」「関数型」「論理型」などがあるが、プログラミングを行うときにどの言語を選べばよいのだろうか？

プログラミング(7)

■スクリプト言語とインタプリタ

インタプリタ：
高水準言語のプログラムを実行するソフトウェア

```
#!/usr/bin/env python
def calc_sum(num):
    if num == 1:
        return 1
    else:
        return calc_sum(num - 1) + num
if __name__ == "__main__":
    print("sum of [1,100] = {}".format(calc_sum(100)))
```

解釈



インタプリタ
(Python)

コンパイルされた
プログラム

実行

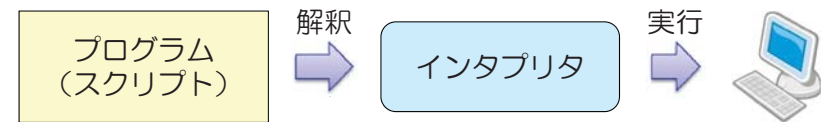


page 9

プログラミング(8)

■インタプリタの特徴

- 欠点
 - 実行速度が遅い
 - プログラムを解釈する必要がある
- 利点
 - 異なるハードウェア上で実行できる
 - プログラムはインタプリタに依存



page 10

プログラミング(9)

■中間言語と仮想マシン

インタプリタの「欠点」を解消するための仕組み

```
public class Sum {
    public static int sum(int num){
        if(num == 1){
            return 1;
        }
        else{
            return sum(num - 1) + num;
        }
    }

    public static void main(String[] args){
        System.out.println("sum of [1,100] = " + sum(100));
    }
}
```

コンパイル



```
CA FE BA BE 00 00 00 32
00 30 0A 00 0C 00 18 0A
00 0B 00 19 09 00 1A 00
1B 07 00 1C 0A 00 04 00
18 08 00 1D 0A 00 04 00
1E 0A 00 04 00 1F 0A 00
04 00 20 0A 00 21 00 22
07 00 23 07 00 24 01 00
...
```

読み込み



実行



仮想マシン
(Virtual Machine)



page 11

プログラミング(10)

■目的に応じてプログラミング言語を選ぶ

- ハードウェアを制御したい
 - C言語, アセンブリ言語 など
- 小規模のWebサービスを開発したい
 - PHP, Ruby など
- GUIを持つアプリケーションを作成したい
 - C#, Objective-C など
- 大勢が利用するWebアプリケーションを開発したい
 - Java など

page 12

ソフトウェアの設計と開発(1)

■ソフトウェア開発の流れ

- 要求分析 : ソフトウェアの目的を定義する
- ↓
- 設計 : ソフトウェアの全体像を決める
- ↓
- 実装 : プログラムを作成する
- ↓
- テスト : 目的を満たしているかを検証する
- ↓
- 運用・保守 : 実際の環境で動作させる

page 13

ソフトウェアの設計と開発(2)

■要求分析

- 機能要件
インタフェース（入力・出力）を決定する
- 性能要件
動作条件（ハードウェア・利用者数など）を決定する

■設計

- 概要設計
利用者から見たソフトウェアの概要を設計する
- 詳細設計
ソフトウェアの具体的な実現方法を設計する

page 14

ソフトウェアの設計と開発(3)

■実装

プログラミングする！

■テスト

- 単体テスト
プログラムの一部（コンポーネントなど）をテストする
- 結合テスト
プログラムの各部品が連携するかをテストする
- 総合テスト
機能要件・性能要件を満たしていることをテストする

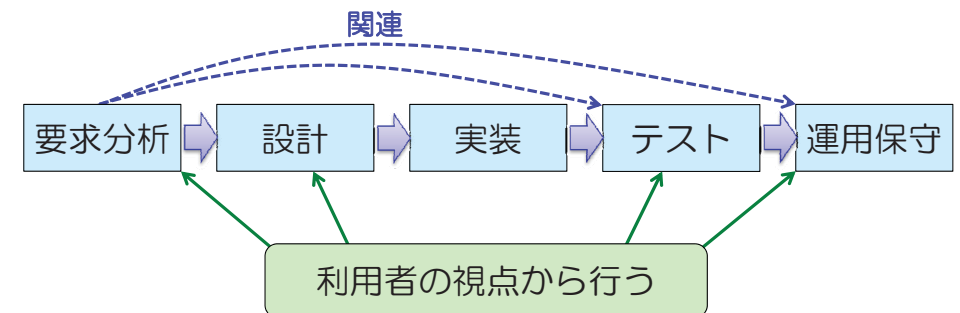
page 15

ソフトウェアの設計と開発(4)

■運用と保守

実際の動作環境に適用し、これを運用する
必要であればソフトウェアの修正を行う

■全体の流れ



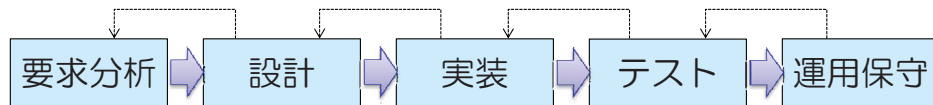
page 16

ソフトウェアの設計と開発(5)

■ウォーターフォールモデル

各工程を順番に行う開発モデル

- 基本的には、前の工程に戻ることはない
- 要求分析を正確に行う必要がある
- 長い期間使用するシステムの開発に使われることが多い



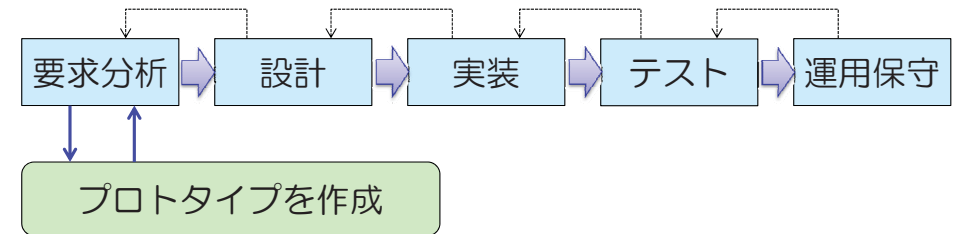
page 17

ソフトウェアの設計と開発(6)

■プロトタイプングモデル

設計前に「プロトタイプ」を作成して全体像を確認する

- 全体像を確認した後はウォーターフォールと同じ
- 機能要件や性能要件を確認しやすい
- 最終目標をイメージしやすい



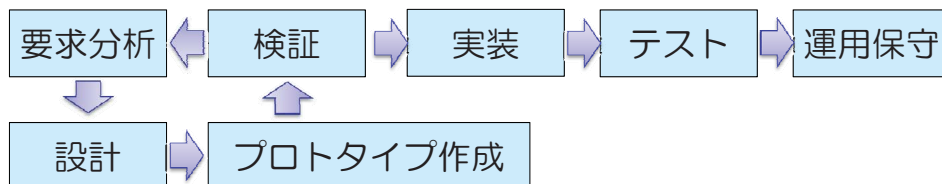
page 18

ソフトウェアの設計と開発(7)

■スパイラルモデル

プロトタイプの試作を繰り返しながら開発を進める

- 要求分析を正確にできる
- 設計の妥当性を確認できる
- 開発工程の管理が難しい



page 19

ソフトウェアの設計と開発(8)

■成長モデル

必要となる機能を逐次的に開発する方法

- 実環境での検証を基に要求分析と設計を修正できる
- あらかじめ拡張性のあるシステム設計が必要となる
- 開発の最終目標をイメージしにくい



page 20

ソフトウェアの設計と開発(9)

■アジャイル開発

設計と試作の繰り返しによりソフトウェアを開発する

- 利用者の意見を反映させたシステムを開発できる
- 開発者と利用者の意思疎通が必要不可欠
- 拡張性の高い設計と柔軟な開発体制が必要

