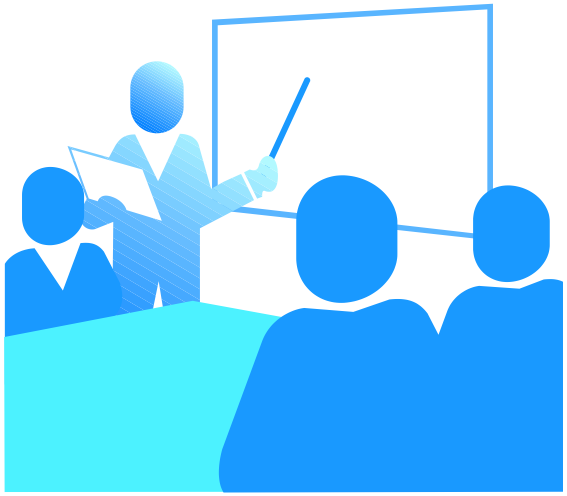
冬期インターン課題

株式会社オリエンタルインフォーメイションサービス

インターンの目的

I. システム開発業務における「プロ」と
「アマチュア」のちがいを理解する

II. 「設計」 「コーディング」の
基礎技術を修得する



I. プロとアマチュアの違いとは

1. 品質の追求

2. 時間のかけどころ

3. 可読性

4. 期待しない条件下での動作

I. プロとアマチュアの違いとは

1. 品質の追求

- バグ
  - 正常に動作するのは当たり前
  - 異常発生時の対応によって品質が決まる
- パフォーマンス
  - 省メモリ
  - 処理速度
- 拡張性
  - 仕様変更などにより、追加（削除）が発生
  - ロジックを変えずに変更

2. 時間のかけどころ

設計時間　＞　実装時間

- 設計：フローチャート，関数の構成・処理
- 実装：コーディング

設計で品質を高める

未然のバグ解消

仕様変更などに対して柔軟に対応可能

2. 時間のかけどころ

- 使用環境や目的に応じて決定

スマートフォンアプリ

早くリリース

→　頻繁に改修・機能追加など

車・金融・医療システム

バグ発生 ＝ 大問題

→　生命に関わる，社会問題

3. 可読性

- 複数人が関わる（レビュー・担当者変更）
  - 誰が読んでも理解できること

- 再読性（バグ対応・実装変更）
  - 数ヶ月，数年前に実装したものの変更
    （自分が作成していても、期間が空くと別人が書いたものと同じ）

■ **可読性上昇**（一例）

- 関数・変数の命名規則
- コーディング規約
- インデント

3. 可読性

- 関数・変数の命名規則
  - **変数名**

    一目見て「何を表している変数」なのか
    わかること

  - **関数名**

    「手段」ではなく「目的」を名前にすること

    例）引数が偶数か奇数かを判定する関数

    ○ `boolean isEvenNumber(int num)`
    関数名が「偶数であるか」（目的）

    × `boolean isModZeroDividedByTwo(int num)`
    関数名が「2で割った余りが0であるか」（手段）

3. 可読性

- コーディング規約
  - **コメント（関数など）**
    コメントを読むだけで処理内容がわかること
  - **プレフィックス（接頭辞）**
    企業名・プロジェクト名
    関数のまとまり
  - **文字コード，改行コード**
    UTF-8・Shift-JIS
    CRLF・CR・LF
  - **タブポリシー，空白**
    タブの空白サイズ・タブまたはスペース

3. 可読性

- インデント

```
for(i=0; i<5; i++) {
if(i<3) {
for(j=0; j<5; j++) {
if(j==0) {
}
else if(j!=1) {
}
}
}
else {
}
}
```

```
for(i=0; i<5; i++) {
  if(i<3) {
    for(j=0; j<5; j++) {
      if(j==0) {
      }
      else if(j!=1) {
      }
    }
  }
  else {
  }
}
```

4. 期待しない条件下での動作

- フェイルセーフ
  - 車のペダルのシステム
  - ペダルは左から順に[クラッチ,ブレーキ,アクセル]の３つのみ

```
switch(pedal) {
    /* クラッチ */
    case Clutch:
        doClutching();
        break;

    /* ブレーキ */
    case Brake:
        speedDown();
        break;

    /* アクセル */
    default:
        speedUp();
        break;
}
```

```
switch(pedal) {
    /* クラッチ */
    case Clutch:
        doClutching();
        break;

    /* アクセル */
    case Accel:
        speedUp();
        break;

    /* ブレーキ */
    default:
        speedDown();
        break;
}
```

課題

- 以下の問題に解答せよ

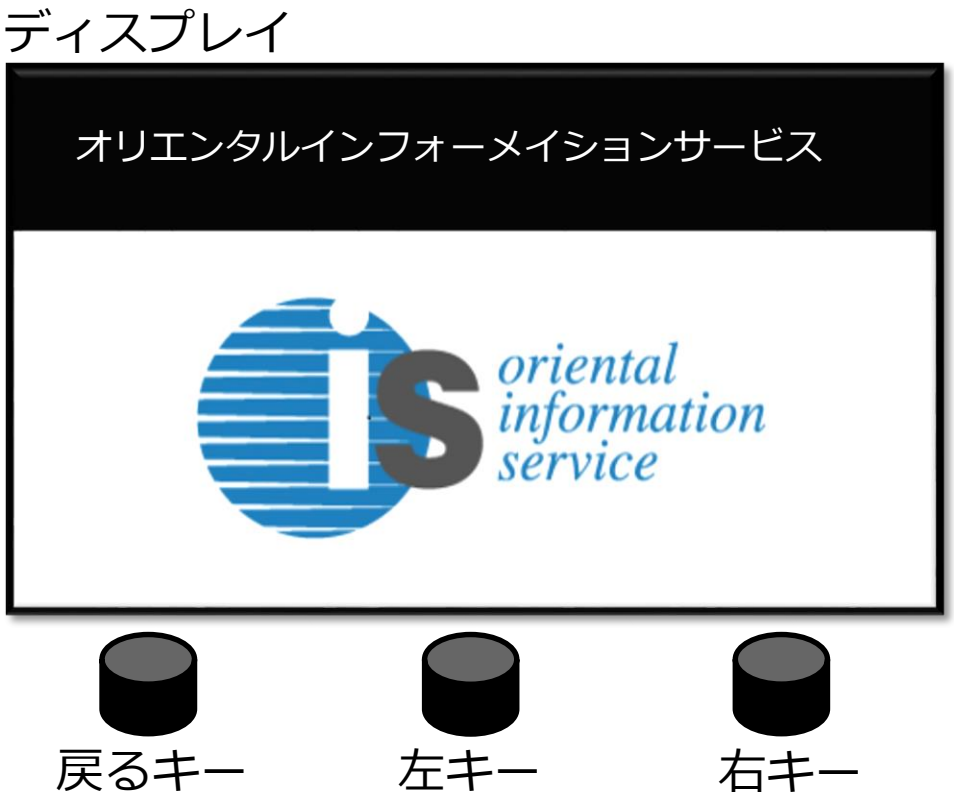
1. フローチャートを作成せよ

2. フローチャートに基づき、
   与えられたC言語のプログラムを変更せよ

■仕様（デバイス）

デバイスは以下の4つを有する

ディスプレイ、戻るキー、左キー、右キー

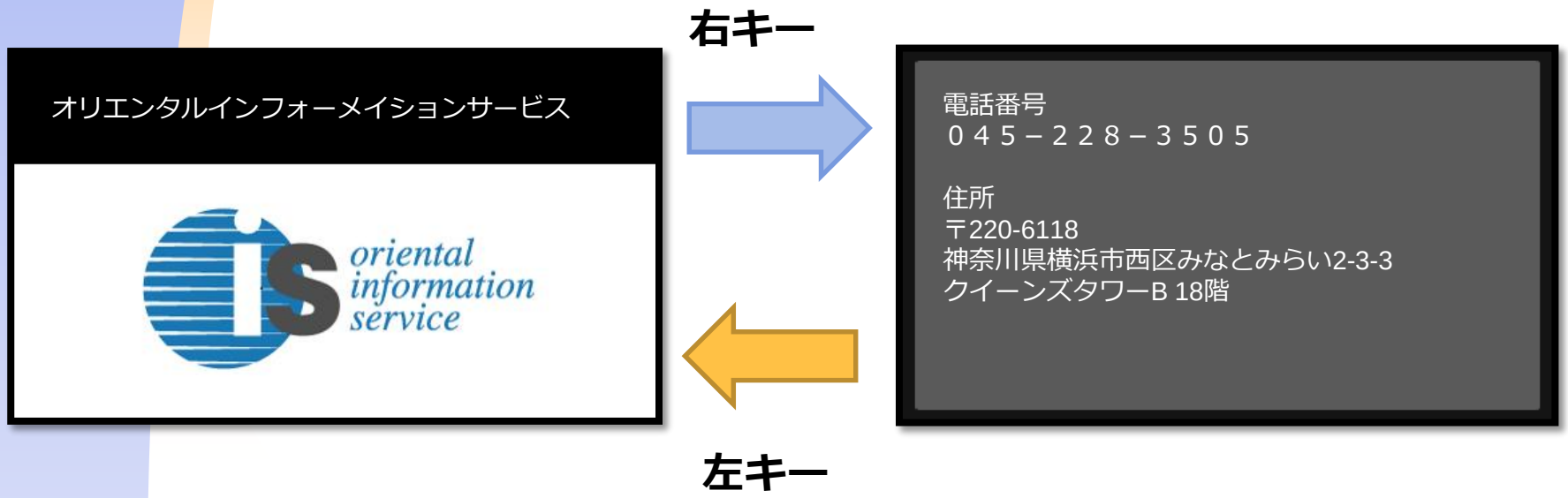


■仕様（現状）

画面が１つ表示されている

■変更内容１（課題）

左右の左右キーを押したとき、別の画面が表示されるようにしたい



設計するにあたり必要な知識

- 関数（今回の場合）
  - ハンドラ
- フローチャート書き方ルール
- 開発環境

- 関数
  - **ハンドラ**

    入力が発生した時に呼ばれる関数

    ・左または右キーを押す（表示画面変更）

    今回のハンドラは以下の１つ

    ◆ 画面切替ハンドラ

●画面切替ハンドラ

画面切替操作を行った際にコール

```
viewId_e
Intern00x00x::internKeyProcess(viewId_e viewId, keyId_e keyId)
```

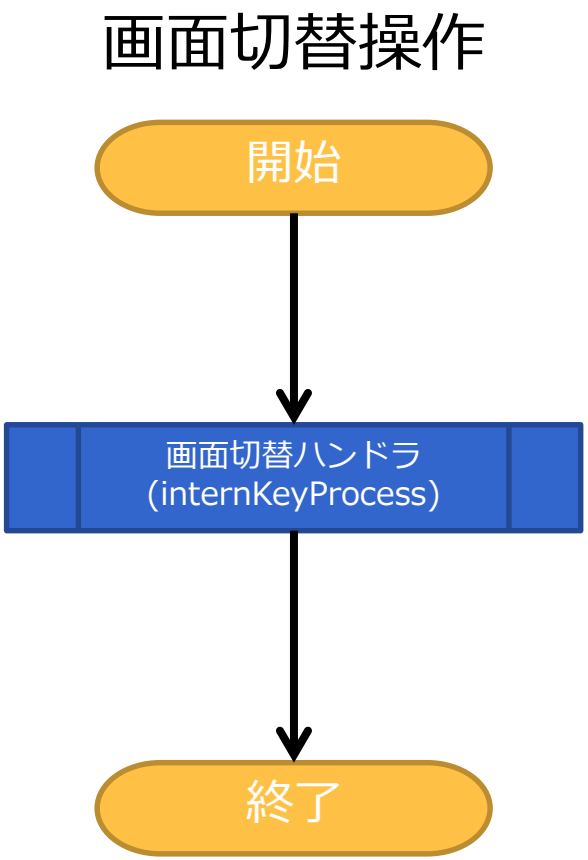
- **パラメーター**

viewId	現在表示中の画面
型：viewId_e	画面ごとに割り当てるID
keyId	押下したボタン（右 または 左）
型：keyId_e	キーごとに割り当てるID

- **戻り値**

  遷移先の画面ID

- 動作の流れ



- 型の定義
  - *viewId_e*
  - *keyId_e*

intern00x00x.h参照

```
typedef enum {
    VIEW_UNKNOWN,
    VIEW_1ST,
    VIEW_2ND,
    VIEW_3RD,
} viewId_e;
```

```
typedef enum {
    KEY_UNKNOWN,
    KEY_LEFT,
    KEY_RIGHT,
} keyId_e;
```

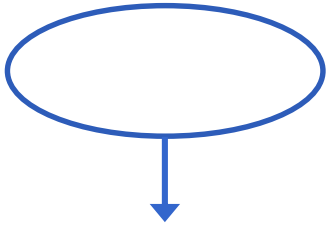
例）画面の判定

```
if(viewId == VIEW_1ST) { }
```

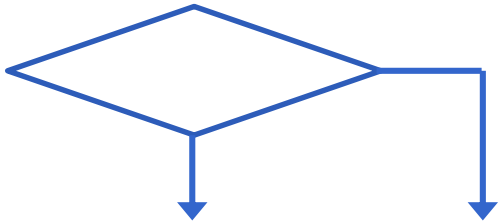
●フローチャート

- 作成するフローチャートは2種類
  - **画面切替処理**

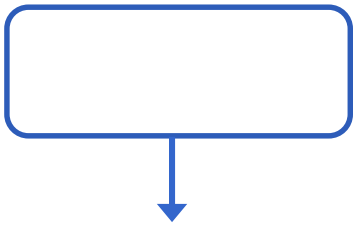
開始・終了



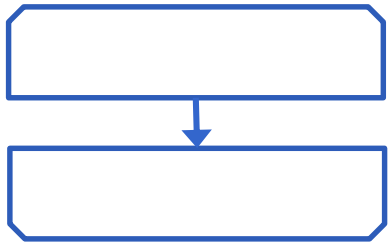
条件分岐



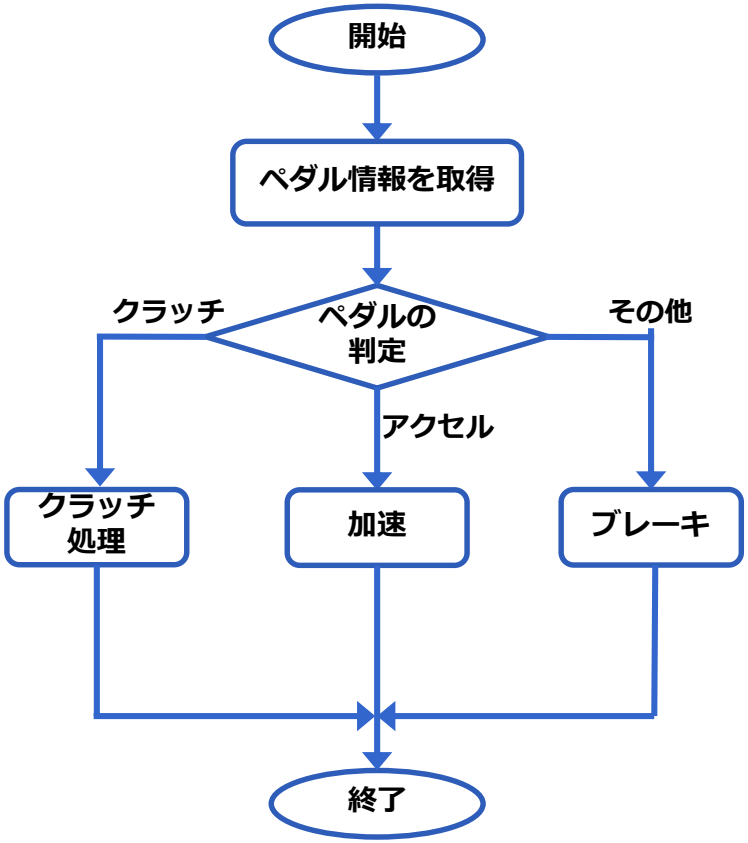
処理（表示・値設定など）



繰り返し



- フローチャート書き方
  - 例）車のペダルのシステム



- 開発環境
  - フローチャート作成
    - 手書き
  - 実装
    - 使用言語：C言語
    - 開発環境：Visual Studio 2013

設計・実装
実機での動作確認

課題（仕様変更）

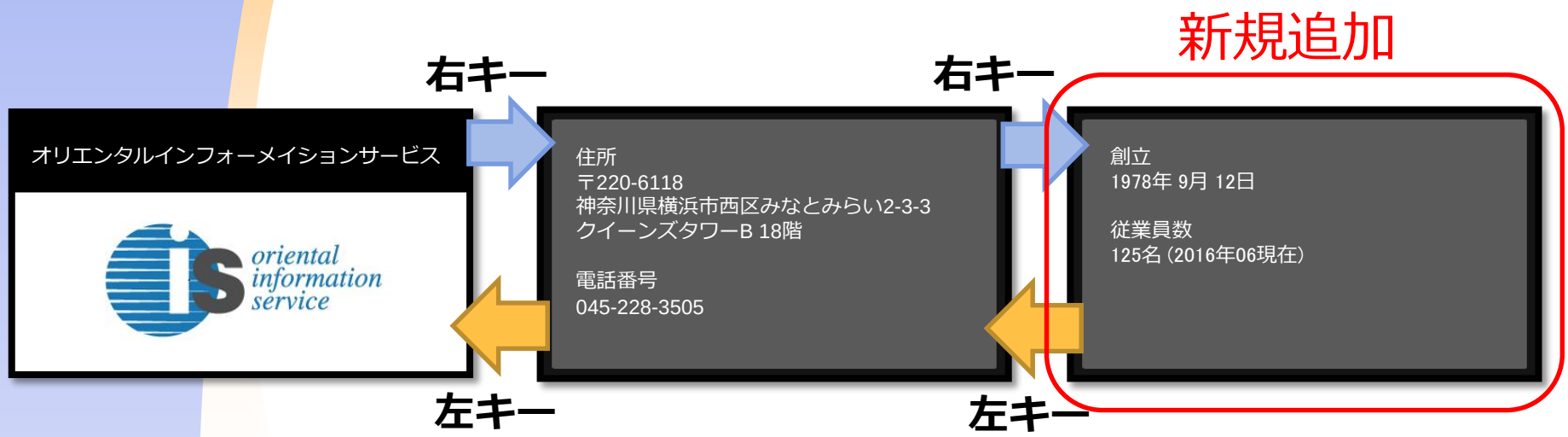
プログラムに仕様変更が生じ、本来、1画面追加であったが
2画面追加することとなった

1. フローチャートを作成せよ

2. フローチャートに基づき、
   与えられたC言語のプログラムを変更せよ

■変更内容３（仕様変更）

左右のキーを押したとき、別の画面が表示されるようにしたい



設計・実装
実機での動作確認

追加課題

追加画面数が100画面の場合の処理の設計を行う

■ 制限

- 画面数の変更（追加・削除）が発生しても
  設計（フローチャート）に変更が不要であること

- if文, switch文などの**条件分岐を増やさない**こと

  ※100通り全ての条件分岐を作成する では×

1. フローチャートを作成せよ
2. フローチャートに基づき、
   与えられたC言語のプログラムを変更せよ

※実装は3画面で行うこと

設計・実装
実機での動作確認

- 実装（模範解答）
  - 画面IDに対応する切替画面や通知方法のテーブルを作成する

```
typedef struct {
    viewId_e    viewId;          /* 画面ID */
    viewId_e    leftViewId;      /* 左キー押下時の画面ID */
    viewId_e    rightViewId;     /* 右キー押下時の画面ID */
} viewInfo_t;
```

表示中の画面	左キー押下時の画面	右キー押下時の画面

- 実装（模範解答）
  - 画面IDに対応する切替画面や通知方法のテーブルを作成する

```
static const viewInfo_t viewInfoList[] = {
    {
        VIEW_1ST,
        VIEW_UNKNOWN,
        VIEW_2ND,
    },
    {
        VIEW_2ND,
        VIEW_1ST,
        VIEW_3RD,
    },
    {
        VIEW_3RD,
        VIEW_2ND,
        VIEW_UNKNOWN,
    },
};
```

- 実装（模範解答）
  - 画面IDに対応する切替画面や通知方法のテーブルを作成する

```
{ // 左画面
    VIEW_1ST,
    VIEW_UNKNOWN,
    VIEW_2ND,
},
```

```
{ // 中央画面
    VIEW_2ND,
    VIEW_1ST,
    VIEW_3RD,
},
```

```
{ // 右画面
    VIEW_3RD,
    VIEW_2ND,
    VIEW_UNKNOWN,
},
```

表示中の画面	左キー押下時の画面	右キー押下時の画面
左画面	なし	中央画面
中央画面	左画面	右画面
右画面	中央画面	なし

実装（模範解答）

- 画面切替後の画面を取得する処理

```
viewId_e
Intern001001::internKeyProcess(viewId_e viewId, keyId_e keyId)
{
    viewId_e            next      = VIEW_UNKNOWN;
    const viewInfo_t    *viewInfo = NULL;
    uint8_t             i         = 0;

    /* 表示中の画面IDを探す */
    for (i = 0; i < viewInfoListSize; i++) {
        if (viewInfoList[i].viewId == viewId) {
            viewInfo = &viewInfoList[i];
            break;
        }
    }

//続く
```

実装（模範解答）

- 画面切替後の画面を取得する処理

```
//続き

    /* 表示中の画面が見つかった場合 */
    if (viewInfo) {
        /* 左キー押下時、左画面をセット */
        if (keyId == KEY_LEFT) {next = viewInfo->leftViewId;}
        /* 右キー押下時、右画面をセット */
        else if (keyId == KEY_RIGHT) {next = viewInfo->rightViewId;}
        /* その他の時、VIEW_UNKNOWNをセット */
        else {next = VIEW_UNKNOWN;}
    }

    /* キー押下後に遷移する次画面の画面IDを返却 */
    return next;
}
```

実際に業務でやっていること

- **ユーザーインターフェース**

  LUI ・ RUI