Appendix 3

Rustの基本構文まとめ

Rustの基本構文は書籍の中でも詳しく解説しています。それでも、一覧になっていると確認しやすいことでしょう。そこで、ここにまとめてみました。

main関数の記述方法

Rustのプログラムはmain関数から始まります。main関数は以下のように記述します。

```
fn main() {
    // ここにプログラム
}
```

コメントについて

一行コメントと範囲コメントが使えます。

```
// Rustの一行コメント

/*
Rustの範囲コメント
*/
```

条件分岐について

条件に応じて処理を分岐するif構文は以下のように記述します。

```
if 条件 {
  // 条件が真のとき
} else {
  // 条件が偽のとき
}
```

複数に順次分岐するには下記のように記述します。

```
if 条件1 {
  // 条件1が真の時の処理
} else if 条件2 {
  // 条件2が真の時の処理
} else {
  // 条件1も条件2も偽だった時の処理
}
```

また、式の値としてif文を指定できます。以下のように記述します。

```
let 変数名 = if 条件 { 真の値 } else { 偽の値 };
```

繰り返しのfor文

特定の範囲を順番に繰り返したい場合、for文が使えます。

```
for 変数 in 開始値 .. 終了値+1 {
  // ここに繰り返す処理
}
```

なお、for文のin以降にはイテレーターを指定します。そのため、配列型やベクター型を指定できます。また、イテレーターについて詳しい内容は、4章4節をご覧ください。

変数や定数の宣言と初期化（束縛）

変数の宣言と初期化するには、let文を使います。なお、型推論の機能により「:型名」は省略できます。

```
// イミュータブル(不変)な変数の宣言
let 変数名: 型名 = 値;

// ミュータブル(可変)な変数の宣言
let mut 変数名: 型名 = 値;
```

定数の宣言にはconstを利用します。定数の宣言では型名を省略できません。

```
const 定数名: 型名 = 値;
```

変数の型を変換するにはasを利用します。

```
変数 = 変数 as 型
```

配列変数の定義と初期化

指定個数の配列を指定の値で初期化するには以下のように記述します。

```
let mut 変数名 = [初期値; 配列要素数];
```

配列変数の型と要素数を指定するには以下のように記述します。

```
let 変数名: [要素の型; 要素数];
```

関数の定義方法

関数を定義するにはfnを利用します。戻り値が必要な関数の宣言では戻り値型を省略できません。戻り値を持たない関数であれば省略できます。

```
fn 関数名(引数宣言) -> 戻り値型 {
    // ここで関数の宣言
}
```

クロージャー（無名関数）の定義は次のように記述します。

```
let 名前 = |引数| 定義;
```

パターンマッチングのmatch文

条件に応じて処理を分岐するのにmatch文を使えます。以下のように記述します。

```
match 条件 {
    値1 => 値1の時の処理,
    値2 => 値2の時の処理,
    値3 => 値3の時の処理,
    _ => その他の処理,
}
```

match文はResult型やOption型などを判定するのに便利です。Option型を使う場合、次のように記述できます。

```
match Option型の値 {
  None => 値がない場合の時の処理,
  Some(v) => 値があった時の処理,
}
```

match文もif文のように式の値を返せます。

```
let 変数名 = match 条件 {
    値1 => 値1の処理と値,
    値2 => 値2の処理と値,
    _ => その他の値
};
```

構造体の定義

構造体は以下のように定義できます。

```
struct 構造体名 {
    フィールド1: 型1,
    フィールド2: 型2,
    フィールド3: 型3,
   ...
    フィールドN: 型N, // ←末尾のカンマは省略しなくても良い
}
```

そして構造体を初期化するには以下のように記述します。

```
let 変数名 = 構造体名 {
    フィールド1: 値1,
    フィールド2: 値2,
    フィールド3: 値3,
    ...
};
```

構造体のメソッドを定義するには以下のように記述します。

```
impl 構造体の名前 {
    // コンストラクターの定義
```

```
    fn new(引数1, 引数2, ...) -> Self {
        Self { 初期値 }
    }
    fn メソッド1(&self, 引数1, 引数2, ...) {
        // ここにメソッド1の定義
    }
    fn メソッド2(&self, 引数1, 引数2, ...) {
        // ここにメソッド2の定義
    }
    // ...
}
```

トレイトについて

トレイトを定義するには下記のように記述します。

```
trait トレイト名 {
  fn メソッド名1(&self, 引数1, 引数2, ...) -> 戻り値型;
  fn メソッド名2(&self, 引数1, 引数2, ...) -> 戻り値型;
  ...
}
```

構造体でトレイトを実装するには以下のように記述します。

```
impl トレイト名 for 構造体名 {
    fn メソッド名(&self, 引数1, 引数2, ...) -> 戻り値の型 {
        // メソッドの処理
    }
    ...
}
```