OCaml 言語の調査

新井 海風 202103254　小川 大輔 202103264

1　はじめに

OCaml とは，1973 年にイギリスで開発された ML をもとに 1996 年に INRIA(フランス国立情報学自動制御研究所) で開発された関数型言語の一種である．OCaml 言語のベースとなった ML とは何か．ML は，「Meta Language(メタ言語)」のことである．1973 年にイギリスの科学者であるロビン・ミルナー氏によって開発されたプログラミング言語であり, メタ言語＝“ある対象について記述や定義するための言語”とされている．OCaml は ML にオブジェクト指向 (処理を部品化して部品を組み合わせることで 1 つのプログラムを作る考え) を追加して取り入れた関数型言語であるため，「簡単な記述」，「高い信頼性」かつ「高速な動作」といった特徴がある．簡単な理由として，より人間に近い思考レベルに近い記述が可能である．安全性が高いためバグの少ないプログラム作成が可能で高速に動作するため OCaml はウェブアプリケーション開発やソフトウェア開発など幅広く使用されている．

OCaml のメリット

1. 型システムの強力さと高い実行性能
2. 高い安全性と信頼性
3. 柔軟なプログラミングスタイル

.

1. OCaml は静的型付けの言語であり，OCaml の型システムは開発の安全性と効率性を高める多くの特性を持っている．特性の一つに「型推論」がある．型推論とはプログラミング言語の機能の一つで変数や関数を明示的に宣言しなくても自動的に型に導いてくれるものだ．型推論のメリットとして，型の宣言が必須ではなくなるためコードを簡潔かつ読みやすくすることが可能となる点がある．型の安全性にも優れているため OCaml はコンパイル時に型エラーなどを早期発見することが可能で実行時のエラーが少ないのだ．また，OCaml は高い実行性能によってパフォーマンスを重視したアプリケーションにも適している．OCaml には高速なネイティブコード (コンピュータの CPU が理解できる形で記述されたプログラム) を生成するコンパイラが存在する．これにより，高い実行速度でのパフォーマンスが可能で，かつ計算負荷が高いアルゴリズムなどにも対応可能．その他に高速なガベージコレクションだ．OCaml には軽量で効率的に動作するガベージコレクタがあるためメモリ解放のコード記述など，管理を開発者がする必要がない．
2. OCaml は高い安全性から高信頼性が求められるプロジェクトで多く活用されている．これらは高い開発効率があるからだ．OCaml で新たに型を作成する場合には「代数的データ型」と呼ばれる方法を活用する．この方法は C 言語の共用体 (union) と列挙型 (enum) を組み合わせたようなデータの記述方法だ．この方法は複雑なデータ構造を簡潔に記述することが可能となり不整合を防いでコードの明確性を向上させる．また，代数的データ型の値へのアクセスにはパターンマッチングを利用する．記述量を削減することができ，パターンに漏れなどが見つかった場合にコンパイラが警告を出してくれるため信頼性も確保することができる．
3. 「オブジェクト指向プログラミング」にはクラス，オブジェクト，継承といった概念が統合されている．簡単に，クラスとはデータ (フィールド) と動作 (メゾット) を定義するテンプレート．オブジェクトはクラスから生成された実体のこと．継承はクラスを継承し，振る舞いの追加や拡張ができること．メリットとして，他のオブジェクト指向言語との親和性が高い点がある．

OCaml のデメリット

1. 一般的な採用率の低さ
2. 学習コスト

1. アプリケーション開発やソフトウェア開発などに

おいて特定の分野で広く使用されている OCaml だが，専門的なコミュニティで利用されることが多い．そのため C++ や Python，Java と比較すると一般的な採用率は低い．私の友人も何人かプログラミングを学んでいるが OCaml のことは触れたことがなく，知らなかった．一般的な採用率が低いことで起こる影響として，OCaml を活用している企業やプロジェクトの数はさほど多くないため全体のプログラミング業界において採用が難しい部分もあるようだ．具体的に，現在は金融業界や研究開発での使用が多い．

2. OCaml には，エラーが少なく安全で信頼性のあるコードを簡潔に書ける点がメリットとしてある反面，概念や記法に慣れるまでが大変な点もある．
また，OCaml には他言語のツールに比べると独自の概念や設定が多いため OCaml を扱い始めるにはほぼ 1 からの学習が必要となる．

2

簡単な OCaml の紹介

```
#3;;                    //整数の場合
- : int = 3
#3.14;;                 //実数の場合
- : float = 3.14
#"Kanagawa";;           //文字列の場合
-: string = "Kanagawa"
# not (false || (not false) && true);;
- : bool = false
# 2 < 3 ;;              //真偽値の場合
- : bool = true
```

3 例

ここで実際に 2 つ問題に取り組んでみようと思う．

- ＜例題＞

とある会社でアルバイトを雇っている．時給 850 円で A さんは週に計 25 時間，B さんは週に 28 時間，C さんは週に 31 時間働いていたとすると，この会社は週に合計いくらのアルバイト代を出すことになるだろうか．

解き方

この問題に限ると，

```
#(25 + 28 + 31) * 850 ;;        \\850 が時給である
-：int = 71400
```

で表わすことができる．

しかし，給与体系が人によって変化してきた場合はどうなるだろうか．また，同じ給与体系だとしてももっと多くの雇用人数で複数の場所に同じ情報を書いている場合，変更を加える際変更し忘れる部分がある可能性がある．ここで変数定義を使うと，例題で使用した時給が複数の場所に書かれる問題を回避することが可能．
OCaml で変数を定義する際には，

```
let 変数 = 式
```

を使用する．この際，変数名はアルファベット小文字で書く必要がある．　この問題では時給が 850 円で定義されているため，

```
# let jikyu = 850 ;;
```

と表現できる．このようにしておくと，例え時給が変化したとしても時給の値を変更するだけで式全体の値を変更することが可能となる．

```
#let jikyu = 1000 ;;
# (25 + 28 + 31) * jikyu ;;
- : int = 84000
```

例えば時給 1000 円に変更する場合，このように表現できる．

- フィボナッチ数列

フィボナッチ数列とは以下のように定義される数列だ．

・F(0)=0
・F(1)=1
・F(n)=F(n − 1)+F(n − 2)（n は 2 以上）

解き方

基本ケース:
n=0 の場合，結果は 0 を返す．
n=1 の場合，結果は 1 を返す．
再帰ケース：
n が 2 以上の場合、フィボナッチ数列の性質に従って

F(n − 1)+F(n − 2) を計算する．
let rec は再帰関数を定義する際に使う．
＜コード＞

```
#(* 再帰的なフィボナッチ関数の定義 *)
let rec fibonacci n =
  if n = 0 then 0
  else if n = 1 then 1
  else fibonacci (n - 1) + fibonacci (n - 2)
val fibonacci : int -> int = <fun>
#fibonacci(10);;
- : int = 55
```

と表現できる．
上記の再帰的実装では，計算が指数的に増えるため，効率が非常に悪い．
たとえば，fibonacci(50) を実行すると，膨大な再帰呼び出しが発生し，計算に時間がかかる．
非効率の原因として再帰的な呼び出しで同じ値を何度も計算するため，計算量が増大する．
例：fibonacci 5 を計算する際に fibonacci 3 が 2 回呼び出される．
ここで末尾再帰による効率化として OCaml では末尾再帰（Tail Recursion）を最適化できるため、計算を効率化できる．

```
#let fibonacci_tail_recursive n =
  let rec aux n a b =
    if n = 0 then a
    else aux (n - 1) b (a + b)
  in
  aux n 0 1 ;;
val fibonacci_tail_recursive : int -> int = <fun>
# fibonacci_tail_recursive(50);;
- : int = 298632863
```

//aux は補助関数で，引数として現在のカウント n，前の値 a，現在の値 b を持つ．
基本ケースとして，n=0 になったら a を返す．//

- ここで紹介した変数は，ほかのプログラミング言語でいうところの定数に近いものになる．C など命令型言語の変数は，一度，値を代入したあとで別の値に書き変えるの普通．しかし，関数型言語での変数は基本的に書き換えることはできず，一度定義したら永遠にその値を持ち続ける．命令型言語の信頼性が減少してるのは，変数の値の書き変えである．変数の値が書き変わることによってプログラムの動きを把握しにくくなり間違いが多くなる．関数型言語では，変数の値の書き変えをしないことにより，信頼性の高いプログラム作成を実現させてる．

4 他の言語と OCaml の違い

1. OCaml と C++ を比較
2. OCaml と Python を比較
3. OCaml と Java を比較

1.
- ＜ OCaml ＞：特徴は高レベルで抽象化された構文であるため，型安全性が保証できる．
  ・メリット：メモリの安全性が高いため，開発をする際に C++ よりも簡単かつ迅速に実行することが可能．
  ・デメリット：低レベルの制御が難しく，パフォーマンスが C++ ほどでない．
- ＜ C++ ＞: 特徴は高いパフォーマンスとハードウェアに近い低レベルの制御が可能．
  ・メリット：ネイティブコードのパフォーマンスで幅広いプラットフォームとライブラリのサポートが可能．
  ・デメリット：メモリの管理が手動のためバグが OCaml に比べ起こりやすい．
- ＜ OCaml と C++ の用途の違い＞:
  ・OCaml は抽象的なアルゴリズムや形式の検証が必要な場面で活用でき，C++ はパフォーマンス重視のゲーム開発やシステムプログラミング，ハードウェアの制御で多く活用できる．

2.
- ＜ OCaml ＞: 特徴は強い静的型付けでコンパイル型言語，関数型プログラミングのサポートがある．
  ・メリット：型安全性が高いためコンパイル時にエラーを早期発見できる．高速なネイティブコードコンパイラによって高いパフォーマンスが可能．
  ・デメリット：独自の概念などが多いことから学習コストが高い．エコシステムやライブラリなどが小規模なため限定的．
- ＜ Python ＞: 特徴は動的型付けでインタプリタ型言語，幅広いエコシステムが存在．

・メリット：学習が簡単で初心者に優しい．豊富なライブラリやフレームワーク (Numpy や Django) などで幅広い分野に対応可能．

・デメリット：実行速度が遅く，大規模プロジェクトでは型エラーの発見まで時間がかかる場合がある．

- ＜ OCaml と Python の用途の違い＞:

  OCaml は型安全性やパフォーマンスが求められるシステムプログラミングやアルゴリズム重視のプロジェクトに適していて，Python は Web 開発やデータサイエンス，スクリプト作成，プロトタイピングなどに適している．

3. - ＜ OCaml ＞: 特徴はコンパイル型言語で高速な実行速度を持つが，仮想マシンではなくネイティブコードを生成する．

     ・メリット：型推論により簡潔なコードの記述が可能で　関数型プログラミングを自然にサポートする．

     ・デメリット：一般的な利用が少ないためライブラリが少ない．

   - ＜ Java ＞: 特徴はオブジェクト指向プログラミングの代表格で，JVM 上で動作する．

     ・メリット：安定したエコシステムで幅広い商業利用，クロスプラットフォームのサポートがある．

     ・デメリット：冗長なコードになることが多く, 関数型プログラミングのサポートは限定的.

   - ＜ OCaml と Java の用途の違い＞: OCaml はアルゴリズムやプロトタイピング，形式検証などの専門的なユースケースに適していて，Java はエンタープライズアプリケーションやクロスプラットフォームなソフトウェアに適している．

5　まとめ

OCaml は，型安全性，高パフォーマンス，関数型プログラミングの様々な利点を活かしながら，多様なパラダイムをサポートするバランスの取れた言語だ．簡単な記述で高速動作，高い信頼性がある OCaml だが，その一方でエコシステムの制約や学習コストの高さといった課題も存在する．しかし，金融分野やウェブアプリケーション開発，ソフトウェア開発などの特定の用途においては非常に強力なツールとなっている．現時点ではまだ採用率が低いが，今後のマルチコア対応やエコシステムの発展がさらに広範な分野での採用を後押しするのではないかと私たちは考える．

参考文献

[1] 浅井　健一, プログラミングの基礎, 株式会社サイエンス社,2007

[2] 型推論，https://members.tripod.com/e_luw/gakko/latex1/l_list.html.

[3] プログラミング入門，https://programming-world.net/language/ocaml/

[4] ネイティブコンパイルで爆速アプリ開発！パフォーマンス向上とメリットを徹底解説，https://dx.shiro-holdings.co.jp/p6847/

[5] データサイエンスのためのトップ 10 の Python フレームワーク，https://jp.orientsoftware.com/blog/data-science-frameworks/#:~:text=Numpy,%E3%81%99%E3%82%8B%E3%81%93%E3%81%A8%E3%82%82%E3%81%A7%E3%81%8D%E3%81%BE%E3%81%99%E3%80%82

[6] プログラミング言語比較：Python、Java、C++、どれを選ぶべきか？, https://invisibletechnology.jp/compare-programming/