C#を用いたプログラミング教育について

箕原辰夫*1
Email: minohara@cuc.ac.jp

*1: 千葉商科大学基盤教育機構

◎Key Words　C#, Unity, .NET, プログラミング教育, プログラミング言語

1. はじめに

ゼミナールで、Unity(1)の開発環境を用いた 3D プログラミングを行なう前段階として、C#(2)のプログラミング言語教育を行なっている。C#は、Java(3)から発展して、独自の仕様なども採り込んでおり、現在の主要なプログラミング教育言語の一つになっている。型推論も採り入れた C#のテンプレートを用いた多相型対応のメソッド定義など、C#言語自体の実践的な内容について扱った講義での注意すべきトピックについて報告する。ここ数年のゼミナールでは、開発環境として VSCode(4)を使い、ターミナルベースのアプリケーションを作成するためのプログラミングを対象にして実習を行なってきた。

2. 講義での授業構成

2.1 開発環境構築について

最初の回で開発環境として VSCode をダウンロードして、以下の拡張機能をインストールさせた。

C# Dev Kit…C#の開発を支援する拡張機能
C#…C#文法を支援する拡張機能
.NET Install Tool…SDK をインストールする拡張機能

また、ターミナルベースのアプリケーションを開発するには、基本ライブラリが入った.NET SDK(5)が必要なので、各年度で以下の版のライブラリをダウンロードして利用することにした。

2024 年度　.NET 8
2025 年度　.NET 9
2026 年度　.NET 10

2.2 プログラムの作成の仕方

ゼミナールでのアプリケーション開発のために、学生に 1 つのプロジェクトとして作成させた。たとえば、プロジェクト名が TerminalProject であれば、PowerShell やターミナルから、以下のコマンドを入力して作成する。

```
dotnet new console -n TerminalProject
```

プロジェクトのフォルダが作成され、その中に Program.cs というファイルができる。このファイルの中のプログラムは、クラスを定義しなくても、直接書かれた内容が実行される。プロジェクトを実行するこのプログラムから実行が始まるので、ここから、各プログラムファイル（クラス定義をする）の Main 関数を呼び出す。たとえば、Practice01.cs という名前のプログラムを実行させるには、以下のように記述する。

```
Practice01.Main( [ ] );
```

また、プログラムのビルドが出来てしまえば、ターミナルからでも、以下のようにして、直接該当のプログラムを実行することができる。

```
dotnet Practice01.cs
```

なお、Practice01.cs には、次のようにして Main 関数をクラスメソッドとして定義しておく。こちらの記述の仕方は Java と共通になっている。

```
public class Practice01 {
    public static void Main( string [ ] args ) { .... }
}
```

2.3 各授業回の構成

各授業回は、以下のように行なった。

表 1 各授業回での講義内容

授業回	講義内容
第 1 回	開発環境の構築、プログラムの定義と実行
第 2 回	それぞれの型のリテラル
第 3 回	型と型変換
第 4 回	カスタムフォーマット指定子と文字列補完
第 5 回	C#の整数演算子
第 6 回	オプショナル型と演算子
第 7 回	条件分岐と繰返し
第 8 回	try 文と using 文
第 9 回	関数定義
第 10 回	値を返す関数
第 11 回	再帰関数・高階関数
第 12 回	Linq ライブラリと List クラス
第 13 回	Hashtable, Collection

学生の中には、タイピングが極度に遅い学生もいるために、結局、半期 13 回の授業で終わらないので、次の半期のゼミナールの前半 1/3 を使って、Unity のプログラミ

ングに入る前に、以下の内容を追加で講義している。

表2 追加授業回での講義内容

授業回	講義内容
第1回	クラスの定義
第2回	コンストラクタ・総合的なクラス定義
第3回	プロパティの設定
第4回	継承とサブクラス定義
第5回	インターフェースの定義と利用

3. C#の文法的な注意事項

ここでは、C#を扱うプログラミングの講義において、気をつけるべき注意点を列挙していくが、特に Python[6] や Java との差分について説明する。ゼミナールでは、Java との差分については、学生と共に Gemini[7] (Google の AI 検索機能) や Perplexity[8]などの AI を参照しながら、ゼミナール中で調べながら、講義を行なっている。

3.1 原始型と型変換

原始型としては、C/C++からの踏襲で、基本的には整数型と実数型、文字型（char/Char）および文字列型（string/String）があり、Java からの踏襲で論理型 (bool/Boolean) が用意されている。整数型と実数型には、bit サイズによって、各種の型に分かれている。表3 に挙げた以外に、整数型には、short(Int16), ushort(UInt16), byte(Byte)型も用意されている。Python のように整数型のリテラルとして、0b の前置詞をつけて2進数も扱えるようになっている。float に対応するクラスが、Float ではなくて、Single であることは注意が必要である。また、四倍精度実数型が実装されているのは、数値計算を扱う場合には望ましいと言える。

表3 C#の主な数値型

型名	対応クラス	内容
int	Int32	標準の符号付き整数型 32bit
long	Int64	符号付長桁整数型 64bit
uint	UInt32	符号なし整数型 32bit
ulong	UInt64	符号なし長桁整数型 64bit
float	Single	単精度実数型 32bit
double	Double	倍精度実数型 64bit
decimal	Decimal	四倍精度実数型 128bit

明示的な型の変換は、キャストの文法に加えて Convert クラスのクラスメソッド等で行なうことができる。

```
Convert.To クラス名( 変換する式 )
   例：int value = Convert.ToInt32( "345" );
型名.Parse( 文字列 ) // 型名.TryParse もある
   例：long value = long.Parse( "2134" );
値.ToString( ) // 文字列へ変換 ""+値でも可能
```

これらの主要な型変換メソッド以外にも、ジェネリック関数対応として、以下のようなメソッドが用意されている。

```
Convert.ChangeType( 値, クラス名 )
型名.CreateCheckd( 値 )
```

3.2 複数変数への代入

最近のプログラミング言語と同様に C#でも、Python のタプルのような記法を用いて、複数変数に一括して値を代入することができる。また、これを用いて、変数の値を交換することができる。

```
int value1, value2;
(value1, value2) = (12, 23); // 一括代入
(value1, value2) = (value2, value1); // 変数の値の交換
```

3.3 カスタムフォーマット指定子と文字列補完

C 言語の printf 関数のフォーマット指定子や、Python のフォーマット指定子に比べると指定できる形式が限られるが、以下のようなカスタム指定子が用意されていて、ToString メソッドや文字列補完の式で利用することができる。n は正の整数で指定する。

Dn…整数 n 桁、0 で右側を埋める
Fn…実数を固定小数点表記で、小数部を n 桁に制限
En…実数を指数表記で、仮数の小数部を n 桁に制限
Cn…通貨形式で、小数部を n 桁に制限
Pn…パーセント形式で、小数部を n 桁に制限
X…整数の 16 進数形式（x だと英字は小文字）
指定桁形式…#,0,.(小数点),%（パーセント）および ,（3 桁ごとのカンマ区切り）が指定可能

また、Python のフォーマット済み文字列リテラルのように、$で始まり、文字列の中の式を{}で囲むような文字列補完の形式も用意されている。

```
$"{value:D2} {value,2}" // 0・空白で埋める 2 桁指定
```

3.4 型推論と型の判定

C#では、変数の型を指定せずに初期値からその変数の型を推論する機能がある。ただし、C#は厳格な型言語であるために、決定された型はそこで完全に固定されるため、Python のような動的型推論言語とは異なり、後から別の型を代入したり、変数自体の型を変更したりすることはできない。

```
var value = 123; // value は整数型と決定される
```

この型名を指定しない変数やジェネリック関数があるために、型を同定するためのいくつかの仕組みが導入されている。

- typeof(型名)…その型の対応クラスが返される
- 値.GetType( )…その値の対応クラスが返される
- 型名または対応クラス名.isSubclassOf(対応クラス)…その型が対応クラスのサブクラスかどうか

・値 is 型名またはクラス名…その値が、その型・クラスとして利用可能かどうか（サブクラス判定も含む）

Python のように型名やクラス名を使って直接判定することはできず、一旦 typeof 関数を使って、対応クラス同士で判定をすることになる。

```
value.GetType( ) == typeof( int )
```

3.5 オプショナル型とポインタ型

近年の新しい言語で導入されてきた null 値を持つことができるオプショナル型と、null 値がどうかを判定する演算子や強制評価の演算子などが導入されている。

```
int ? value = null; //  オプショナル型の宣言
int y = value ?? 0;   // value が null なら y に 0 を代入
int? length = message?.Length;
// message が null なら length にも null が代入される
double? x = array?[ 2 ];
// array が null でない場合、array[2]の要素を返す
string line = Console.ReadLine()!;
//  オプショナル型のラップを外し強制評価する
```

Java では回避されていたポインタ型も使うことができる。ただし、ポインタを使う場合、メソッドやコードブロックに unsafe キーワードを付ける必要がある。加えて、コンパイルするときに、/unsafe オプションを必要とする。

3.6 switch 式と関係パターン

従来からの switch 文に加えて、最近の言語で導入されてきた match 文に該当する switch 式を使うことができる。この式では、Python と同様に論理演算子を&&, ||, ~ ではなく、and, or, not での英単語での演算子で記述する。

```
var season = date.Month switch {
      > 3 and < 6 => "spring",
      >= 6 and <= 9 => "summer",
      > 9 and < 12 => "autumn",
      12 or (>= 1 and <= 3) => "winter",
    _ => "not a month"};
```

3.7 using 文によるブロック

using 文は通常は名前空間の導入に用いられるが、Python の with 文と同様に使うことができる。この文で用いられるオブジェクトには、IDisposable インターフェースを実装する必要がある。

```
using (var writer = new
System.IO.StreamWriter("sample.text"))
{ writer.WriteLine("Hello, C# using block"); }
```

3.8 内部関数

Java とは異なり、Python と同様に、メソッドの中に関数を定義することができる。Java では、クラスメソッドで統一するか、一度クラスのオブジェクトを生成してインスタンスメソッドで統一するかしなければならなくて面倒だったが、通常の関数であれば、Main クラスメソッドの中で閉じて定義できるので、実際に関数の定義の例を示すのに役立った。

```
public static void Main(string [ ] arg ) {
    int square( int x ) { return x*x; } //  内部関数
    Console.WriteLine( square( 12 ) ); }
```

3.9 オプショナル引数と名前付き実引数

Python と同様に、関数・メソッドの引数において、デフォルト値を指定しておくオプショナル引数を利用することができる。また、実引数で指定するときに、キーワード引数のように、引数名を指定することができる。

```
int cube( int x:1 ) { return x*x*x; }
int value = cube( ); //  実引数が省略された場合
value = cube( x: 17 ); //  キーワード的に指定する
```

3.10 配列の引数と ref, out 引数、params 引数

配列の引数の場合は、Java とは異なり、値渡し（コピー渡し）ではなく、Python と同様に参照渡しになる。また、1 つの変数についても、ref や out などの修飾子をつけることで参照渡しになり、関数の中で値を設定すると元の変数に値が代入される。ただし、実引数の方でも ref, out の修飾子をつける必要がある。ref は、変数に予め値が設定されている必要があるが、out は設定されていなくても構わない。

```
void ModifyRef(ref int num) {  num = 20;  }
int number = 12;
ModifyRef(ref number); // number  →  20
void InitializeOut(out int num) {  num = 30;  }
int value;
InitializeOut(out value); // value  →  30
```

Python と同様に、可変個数の引数を受け渡すことができる params 修飾子も導入された。

```
void PrintNumbers(params int[] numbers) {
  foreach ( int num in numbers ) {
      Console.Write( num + " " );
PrintNumbers( 12, 23, 34, 45 );
```

3.11 メソッドとシグネチャ

Java と同様に、メソッドの場合は、名前以外に仮引数の型や戻り値の型を用いたシグネチャで識別されるためにメソッドのオーバーロードが可能で、多相型を実現することができる。しかしながら、内部関数ではシグネチャが使えないので、オーバーロードはできないのには注意が必要である。Perplexity にそれを確かめさせた。

3.12 ジェネリック関数

C++から始まりJavaでも導入されたテンプレートを使って、C#でもジェネリック関数を導入することができるが、加えて、where句で受け取れる型の制限をすることが可能になっている。このときに比較ができるクラスのオブジェクトの場合は、ICompatible, 四則演算ができるクラスのオブジェクトの場合は、INumberのインターフェースを指定することができる。

```
using System.Numerics; //INumber 導入用
T Modulo<T>(T n, T m ) //剰余を返す関数
  where T: INumber<T>, IConvertible {
  T result = n;
  while ( result > m ) { result -=m; }
  if ( result is double ) { //  実数の場合は誤差を考慮
   double r = Math.Round( Convert.ToDouble( result ),
       14, MidpointRounding.AwayFromZero );
   return (T)Convert.ChangeType(r, typeof(T));
  } else { return result; } }
```

戻り値の型が静的に定まらない関数の場合、戻り値型をobjectにしておくが、これもジェネリック関数で対応できることが多い。

```
object GetHalf( int n ) { // object 版
  return (n%2==0) ? n/2 : (double)n/2.0; }
T GetHalf<T>( int n ) { //  ジェネリック版
  return (n%2==0)
  ? (T)Convert.ChangeType(n/2, typeof(int))
  : (T)Convert.ChangeType(n/2.0, typeof(double)); }
```

ランタイムに型推論をするdynamic型、シグネチャやジェネリック関数を利用することで多相型を実装することが可能だが、Pythonのような標準で動的に型推論をするインタープリタ言語と比べると記述能力は劣るだろう。

3.13 高階関数と無名関数

Javaとは異なり、Pythonと同様に、関数やメソッドもファーストオブジェクトになっているために、高階関数を定義することができる。関数の型を指定するときには、Func <引数の型,戻り値の型>を指定する。また、高階関数には実引数として無名関数も利用することが可能になっている。無名関数は、「引数 => 戻り値の式」で指定する。

```
void PrintThree( Func<int, string> fun ){
     Console.WriteLine( fun( 3 ) ); }
PrintThree( n => "ABC:" + n ); //  無名関数
```

3.14 配列のLinqライブラリとListクラス

配列に対しては、Linqライブラリ や Listクラスのオブジェクトに変換して使うことで、Python のリストに対しての様々な関数と同様なメソッドを利用することができる。たとえば、Pythonのenumerate, zip, map, filter, reduce, index などの関数も以下のようなメソッドで実現することができる。

```
配列.Select(1 変数関数)…map, enumerate
配列.Where(論理値関数)…filter
配列.Aggregate(2 変数関数)…reduce
Array.FindIndex(配列, 論理値関数)…index
配列.Zip(配列)…zip
```

3.15 プロパティ制御

C#では、Javaのフィールドに加えて、フィールドへのアクセスを制御する仕組みのことをプロパティと呼んでいる。無名関数の記法を利用して、クラスの定義に、getterと setter の関数を定義することにより、プロパティへのアクセスの仕組みを実装することができる。

```
class Point { private double _x;
   public double X {
     get => _x;
     set => _x = value; // value は代入された値 } }
```

4.　おわりに

C#はJavaから進化して、最新のプログラミング言語で導入されている様々な仕様を取り込んできている。Unityで3次元CGのプログラミングをするために扱ってきたC#であるが、Unity 自体は JavaScript でも開発が可能である。しかしながら、ローカルなアプリケーションの開発での需要があって C#の方が正式な開発言語として用いられてきているのだと考えている。Pythonが普及し、圧倒的な記述の容易さが波及して、コンパイラ言語でもPythonで導入されてきたような機構の一部を採り入れてきている。ただ、プログラミング教育はC#から始めるのではなく、Pythonから始める方が良く、Pythonである程度学んだ学生に需要に応じてC#を学ばせた方が良い。

参考文献

(1) Unity, Unity Technologies, https://unity.com/ja.
(2) C# language reference, Microsoft Corporation, https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/.
(3) Java language specification (Java SE21 edition), Oracle Corporation, https://docs.oracle.com/javase/specs/jls/se21/html/.
(4) Visual Studio Code, Microsoft Corporation, https://code.visualstudio.com/.
(5) .NET SDK, Microsoft Corporation, https://dotnet.microsoft.com/en-us/download.
(6) Python, Python Software Foundation, https://docs.python.org/3/.
(7) Google Gemini, Google LLC, https://gemini.google.com/app.
(8) Perplexity, Perplexity AI, Inc., https://www.perplexity.ai/.