プログラミング言語論 #01

Introduction

2018-04-09

値が等しければ結果は同じはずという考え方を用いることにより，動作するプログラムを意味を変えずに書き方の違う別のプログラムへと変換することができる．

さらにこのアイデアを，型が等しければコンパイルエラーにはならないはずというように応用することで，プログラムをエラーを起こすことなく計算結果の違う別のプログラムへと変換することができる．

このように値や型にはある種の数学的な性質があり，それゆえプログラムそのものの数学的な取り扱いを可能にする．型の概念はプログラミング言語の研究において重要なテーマであり，様々な発展を遂げてきた．

型に関する理解を深めること，それがこの科目の目的である．

型に関する計算

```
1    int z = 10 + k;
2    printf("%d", z);
```

型に関する計算

```
1    int z = 10 + k;
2    printf("%d", z);
```

C1:???は一意に決まるか

```
1    int x = 10;
2    printf("%???\n", x);
```

型に関する計算

```
1    int z = 10 + k;
2    printf("%d", z);
```

C1:???は一意に決まるか

```
1    int x = 10;
2    printf("%???\n", x);
```

C2:???は一意に決まるか

```
1    double x;
2    x = sin(31.415);
3    printf("%???\n", x);
```

printf とは何引数なのか，何型なのか？

型に関する計算

```
1    int z = 10 + k;
2    printf("%d", z);
```

C1:???は一意に決まるか

```
1    int x = 10;
2    printf("%???\n", x);
```

C2:???は一意に決まるか

```
1    double x;
2    x = sin(31.415);
3    printf("%???\n", x);
```

printf とは何引数なのか，何型なのか？

```
1    printf("please input: \n");
```

型には同値関係が存在する

従って推移律が成り立つ

C3 : ???は一意に決まるか

```
1  ??? f1 () {
2    ??? a = 0;
3    double X[10];
4
5    for (i = 0;  i < 10; i++) {
6      a = a + X[i];
7    }
8    return a;
9  }
```

型には同値関係が存在する

従って推移律が成り立つ

C3 : ???は一意に決まるか

```
1  ??? f1 () {
2    ??? a = 0;
3    double X[10];
4
5    for (i = 0;  i < 10; i++) {
6      a = a + X[i];
7    }
8    return a;
9  }
```

C4 : ???は一意に決まるか

```
10 // C3の続き
11
12 int main ()
13 {
14   printf("kekka = ???\n", f1());
15   return 0;
16 }
```

型には同値関係が存在する

従って推移律が成り立つ

C3 : ???は一意に決まるか

```
1  ??? f1 () {
2    ??? a = 0;
3    double X[10];
4
5    for (i = 0;  i < 10; i++) {
6      a = a + X[i];
7    }
8    return a;
9  }
```

C4 : ???は一意に決まるか

```
10 // C3の続き
11
12 int main ()
13 {
14   printf("kekka = ???\n", f1());
15   return 0;
16 }
```

この計算において，プログラムの意味を考えることは全く不要

数学的には次の興味として当然，順序関係が持ち込めるかどうかが気になるのだが

型と値の違い

C3 再掲

```
1  ??? f1 () {
2    ??? a = 0;
3    double X[10];
4
5    for (i = 0;  i < 10; i++) {
6      a = a + X[i];
7    }
8    return a;
9  }
```

型と値の違い

C3 再掲

```
1  ??? f1 () {
2    ??? a = 0;
3    double X[10];
4
5    for (i = 0;  i < 10; i++) {
6      a = a + X[i];
7    }
8    return a;
9  }
```

C5 : ???は一意に決まるか

```
1  ??? f2 () {
2    ??? a = 0;
3    double X[10];
4
5    for (i = 0;  i < 10; i++) {
6      a = a + X[i];
7    }
8    return 8;
9  }
```

型と値の違い

C3 再掲

```
1  ??? f1 () {
2    ??? a = 0;
3    double X[10];
4
5    for (i = 0;  i < 10; i++) {
6      a = a + X[i];
7    }
8    return a;
9  }
```

C5 : ???は一意に決まるか

```
1  ??? f2 () {
2    ??? a = 0;
3    double X[10];
4
5    for (i = 0;  i < 10; i++) {
6      a = a + X[i];
7    }
8    return 8;
9  }
```

f1 と f2 はとても良く似ているのに，

型と値に関する複雑な計算例

方程式としての側面

C3 再掲

```
1  ??? f1 () {
2    ??? a = 0;
3    double X[10];
4
5    for (i = 0;  i < 10; i++) {
6      a = a + X[i];
7    }
8    return a;
9  }
```

型と値に関する複雑な計算例

方程式としての側面

C3 再掲

```
1  ??? f1 () {
2    ??? a = 0;
3    double X[10];
4
5    for (i = 0;  i < 10; i++) {
6      a = a + X[i];
7    }
8    return a;
9  }
```

C6: 常に C3 の関数 f1 と同じ結果を返すこと

```
1  ??? f3 () {
2    ??? a = 0
3    ??? b = ???;
4    double X[10];
5
6    for (i = 0;  i < 10; i++) {
7      a = a + 1;
8      ? = ? + X[i];
9    }
10   return b;
11 }
```

値の等価性，型の等価性

○正しく動く元プログラム

```
1  x1 = 80808 * 19 + a1;
2  x2 = 80808 * 19 + a2;
3  x3 = 80808 * 19 + a3;
```

○正しく動く元プログラム

```
1  printf("kekka = %d\n", x1);
2  printf("kekka = %d\n", x2);
3  printf("kekka = %d\n", x3);
```

○改良を試みたプログラム

```
1  a = 80808 * 19;
2  x1 = a + a1;
3  x2 = a + a2;
4  x3 = a + a3;
```

○改良を試みたプログラム

```
1  ??? m = "kekka = %d\n";
2  printf(m, x1);
3  printf(m, x2);
4  printf(m, x3);
```

△間違えても × エラーなし○

```
1  x1 = 80808 * 19 + a1;
2  x2 = 80088 * 19 + a2;
3  x3 = 88008 * 18 + a3;
```

△間違えても × エラーなし○

```
1  printf("kekka = %d\n", 1);
2  printf("keka = %d\n", x2);
3  printf("kekka = %dn", x3);
```

プログラミング言語における型

値の法則

- 任意のものは値が等しいものと置換可能=実行結果は等しい

型の法則

- 任意のものは型が等しいものと置換可能=エラーにならない

型は値とは違う．また，文法とは違う種類のルールが存在する．

型の意義

- 実行前にエラーを発見できる
- ある場所に書けるものを限定できる，指定できる

プログラミングにおける部分点

1. 白紙（文法なしアルゴリズムなし）＝ 0 点

プログラミングにおける部分点

1. 白紙（文法なしアルゴリズムなし）＝ 0 点
2. コンパイルできないプログラムもどき＝ 30 点

プログラミングにおける部分点

1. 白紙（文法なしアルゴリズムなし）＝ 0 点
2. コンパイルできないプログラムもどき＝ 30 点
3. 型に間違いのないプログラム

プログラミングにおける部分点

1. 白紙（文法なしアルゴリズムなし）＝ 0 点
2. コンパイルできないプログラムもどき＝ 30 点
3. 型に間違いのないプログラム
4. 値を正しく計算するプログラム＝ 100 点

プログラミングにおける部分点

1. 白紙（文法なしアルゴリズムなし）＝ 0 点
2. コンパイルできないプログラムもどき＝ 30 点
3. 型に間違いのないプログラム ⇐ 中間地点
4. 値を正しく計算するプログラム＝ 100 点

プログラミングにおける部分点

1. 白紙（文法なしアルゴリズムなし）＝ 0 点
2. コンパイルできないプログラムもどき＝ 30 点
3. 型に間違いのないプログラム ⇐ 中間地点
4. 値を正しく計算するプログラム＝ 100 点

数学は値の正しさに主眼．プログラミングするにはその前に型の正しさも必要．

プログラミングにおける部分点

1. 白紙（文法なしアルゴリズムなし）＝ 0 点
2. コンパイルできないプログラムもどき＝ 30 点
3. 型に間違いのないプログラム ⇐ 中間地点
4. 値を正しく計算するプログラム＝ 100 点

数学は値の正しさに主眼．プログラミングするにはその前に型の正しさも必要．

プログラムができない一因は一気に 4 を目指すやり方しか知らないから

この科目の目的

プログラミング言語における型について深く理解する

- プログラミング言語における型の意義を理解し，説明できる
- その過程で C 言語以外の進んだ最近の言語が理解できる
  - Haskell 言語（型→多相→種→依存型という進化中）
  - JavaScript 言語（型がないという反対の例）
  - C++言語（C の延長のオブジェクト指向言語）
  - 他にも Python 言語, Java 言語なども対象

議論の土台：集合論，等価性，推移律といった数学的基礎

型と言語

- どういう型を許すか→様々なプログラミング言語
- 近年のプログラミング言語の進化は型から見ると説明できることが多い
  - コンパイラに型を使わせる
  - 書きにくさ，難しさの型の応用による解決
  - 関数の進化
- 型の知識は，新しい言語の理解に使える

より実利的な方向なので，世の中の「プログラミング言語論」よりは型に集中した内容

応用：型等式（等しい物は等しい）

正解を選べ（理解の確認）

```
1  int x;
2  // xに値を代入したいです
3  scanf("%d", x);     // 案1
4  scanf("%d", &x);    // 案2
```

正解を選べ

```
1  int A[10];
2  // A[0]に値を代入したいです
3  scanf("%d", A);      // 案1
4  scanf("%d", A[0]);   // 案2
5  scanf("%d", &A[0]); // 案3
```

応用：型等式（等しい物は等しい）

正解を選べ（理解の確認）

```
1  int x;
2  // xに値を代入したいです
3  scanf("%d", x);     // 案1
4  scanf("%d", &x);    // 案2
```

正解を選べ

```
1  int A[10];
2  // A[0]に値を代入したいです
3  scanf("%d", A);      // 案1
4  scanf("%d", A[0]);   // 案2
5  scanf("%d", &A[0]); // 案3
```

一歩々々考えると難しいことはないのだが

```
1  int x;
2  int A[10];
3  // 一気にするのは無理でした
4  scanf(...
5  A[0] =...
```

応用：型等式（等しい物は等しい）

scanf の第 2 引数はポインタでなければならない：

この情報を基に

```
1  int scanf(...., int *); // プロトタイプ宣言（注意：次ページ脚注
       参照）
```

正解を選べ

```
2    int x, *y;
3   // xに値を代入したいです
4   scanf("%d", &x);    // これはよく知っている
5   // yに値を代入したいです
6   scanf("%d", y);      // 案1
7   scanf("%d", &y);     // 案2
8   scanf("%d", *y);     // 案3
9   scanf("%d", **y);    // 案4
10  scanf("%d", &*y);    // 案5
```

応用：型等式（等しい物は等しい）

scanf の第 2 引数はポインタでなければならない：

この情報を基に

```
1  int scanf(...., int *); // プロトタイプ宣言（注意：次ページ脚注
       参照）
```

正解を選べ

```
2    int x, *y;
3   // xに値を代入したいです
4   scanf("%d", &x);    // これはよく知っている
5   // yに値を代入したいです
6   scanf("%d", y);      // 案1
7   scanf("%d", &y);     // 案2
8   scanf("%d", *y);     // 案3
9   scanf("%d", **y);    // 案4
10  scanf("%d", &*y);    // 案5
```

```
1    int x, *y;
2    // xに値を代入してyにコピー
3    scanf("%d", &x);    // これは簡単
4    *y = x;
```

数学の導入：再び等価関係

もっとも簡単な場合を考える[1]

1. 関数宣言

```
1  int scanf(...., int *);
```

2. 変数定義

```
1    int x;
```

3. 呼出し

```
1    scanf("%d", &x);
```

& が必要な理由を答えよ

- &はポインタにする演算子
- *はポインタをたどる演算子
- &*，*&は何もしない

2'. 変数定義'

```
1    int &*x; または int *&x;
```

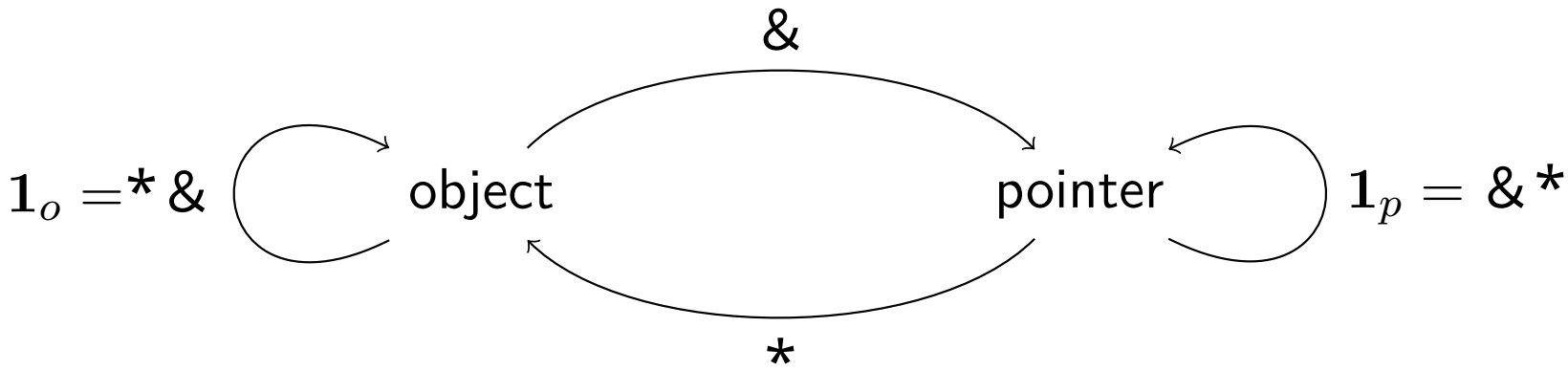
3. 呼出し

```
1    scanf("%d", &x);
```

※ 何もしない＝値も型も変えない

[1] scanf の本当の関数宣言はもっと複雑で，これ以外の型も受け付けるのだがここでは無視して簡単化した

数学の導入：$Id = f^{-1} \circ f$



関数宣言と変数定義と呼出しに関する法則の存在

scanf だけではない

C7

```c
// プロトタイプ宣言
int f(int);

// 変数定義
  int ******n;

// 呼出し
  f(???n);
```

C8

```c
// プロトタイプ宣言
void super(int ***);

// 変数定義
  int ???z;

// 呼出し
  super(*z);
```