プログラミング言語

名倉 正剛
日本大学 工学部 情報工学科
70号館7044号室

プログラミング言語　第4回

- 識別子，変数

前回の練習問題

- 次の C 言語の記述 (for文)を受理する BNF 記述を作成せよ. ただし, 簡単化のため, 下に示す制限を加えて良い.

例:1から100までの総和を求めるプログラム

```
for (i = 1; i <= 100; i++){
    sum += i;
}
```

制限

次の事項を考慮しなくて良い:

- インデントの空白と改行.
- 整数以外の数 (小数).
- 他の関数への呼び出し.
- ブロック内での, 入れ子構造.
- break 文, continue 文.
- 変数名の記号文字の存在.

```
<文>::= <for文>
<for文>::='for('<初期化文>?';'<条件>?';'<処理列>?'){'<ブロック>'}'
<初期化文>::=？？？？
<条件>::=？？？？
<処理列>::=？？？？
<ブロック>::=？？？？
```

解答例

1から100までの総和を求めるプログラム

```
for (i = 1; i <= 100; i++){
    sum += i;
}
```

<文>::= <for文>
<for文>::='for('<初期化文>?';'<条件>?';'<処理列>?'){'<ブロック>'}'
<初期化文>::=<変数><代入演算子><式>
<条件>::=<数字列>|<変数> (<比較演算子> <数字列>|<変数>)?
<処理列>::=<処理>(,<処理>)*
<比較演算子>::= (('<' | '>' | '!' | '=')? '=') | ('<' | '>')
<代入演算子>::= ('+' | '-' | '*' | '/' | '%')? '='
<算術演算子>::= '+' | '-' | '*' | '/' | '%'
<ブロック>::=(<処理>';')*
<変数>::=('('<変数>')') | <変数名>
<変数名>::=*[*]<アルファベット>+(<数字>|<アルファベット>)*
<数字列>::=('('<数字列>')')|<整数>
<整数>::=('+'|'-')?<数字>+
<数字>::='1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'|'0'
<アルファベット>::= 'A'|'B'|'C'|…(略)…|'Z'|'a'|'b'|'c'|…(略)…|'z'
<処理>::= <変数><代入演算子><式>
<式>::=('('<式>')')|<算術式>
<算術式>::=(<変数>|<数字列>|<式>)<算術演算子>(<変数>|<数字列>|<式>)

識別子 (identifier)

- プログラム内で変数や関数などを識別する，トークン（文字列）
  - スコープ（特定の範囲）内で，ユニークでなければならない．

    ex) 同一関数の中で，同じ名称の変数を作成できない．
  - 曖昧さが許されないため，文法上定まった形式に沿った名前にする必要がある．
  - 利用できる文字種は，言語によって異なる．
    - 大文字，小文字を区別しない場合もある（BASIC 等）
    - 日本語の識別子（全角文字，Unicode）を利用できる場合もある（Java 等）
    - Unicode 文字で，絵文字を使える場合もある（Swift 等）

識別子と命名ルール

- 識別子の名称となる文字列には，決定ルールがある.

- 仕様により決定されるルール：多くの場合は，プログラミング言語仕様に依存する
  - 大文字，小文字を許可するかどうか.
  - 識別子名称の長さの制限.
- 開発チームで決定されるルール：プログラムを読んだ人が理解しやすいようにルール化する.
  - ハンガリアン記法
  - キャメル記法

識別子の長さ

- 識別子が短すぎても長すぎても，プログラムの保守性を下げる.
  - 短い識別子
    - 識別が困難，同じ名前の識別子が再登場しやすい
    - 部分的に省略した結果，暗号的な名前になる
  - 長い識別子
    - タイピングしにくい
    - 識別はできるが，判別が困難
- 初期のリンカでは，メモリ使用量を抑えるため，識別子長を6文字に制限していたものもある.

識別子長の例

- OSS の3つのプロジェクトを分析，識別子のうちの変数名の長さをヒストグラムに表した
  - 阿萬裕久他，”変数名とスコープの長さ及びコメントに着目したフォールト潜在性に関する定量的調査”，ソフトウェアエンジニアリングシンポジウム2015

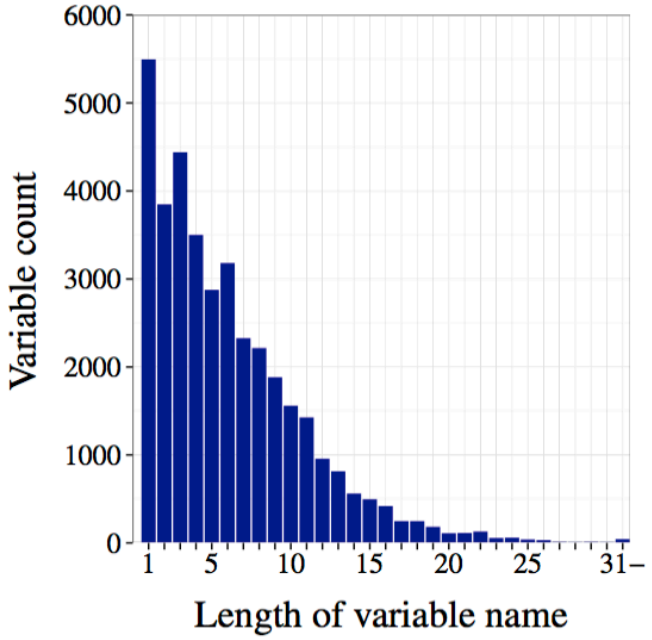


図 1　ローカル変数名の長さのヒストグラム

実際の開発現場での命名ルール

- 普通の開発プロジェクト
  - 大文字と小文字に意味を与える
  - 単語（もしくは略語）を組み合わせる
  - 特定の意味に対して，特定の記法を指定する場合も多い
    - 定数の場合は，すべて大文字にする
    - 外部から参照させることを想定している場合は，特定の接頭辞をつける　など．
- ハンガリアン記法
  - データ型+意味で命名（例：nTotal，sName)
    - データ型を変更した場合に，変数名を直す必要があり，保守性が低下する．
- キャメル記法（ラクダのコブに例えた呼び名）
  - McDoonald’s のように，複合語をひとまとまりにして，要素の先頭を大文字で表す (例：currentTimeMillis, getProperties, BufferedReader).

識別子と予約語・キーワード

- 予約語，キーワード
  - 識別子としてのルールを満たしているにもかかわらず，識別子にならない字句
  - 利用すると，コンパイルエラーや，実行時エラーになる．

- 予約語
  - プログラミング言語が特別な用途に利用するために，確保している字句
  - ただし，必ずしも利用されている必要はない．

  例）Java 言語における goto 文
- キーワード
  - プログラミング言語が特別な用途に利用している字句
  - 予約されているか，ではなく，実際に利用されている

  例）C 言語における for 文（同時に，予約語でもある）

変数

- データを格納しておく場所のこと.
- ほとんどの言語では，識別子によって，名前付けをする.

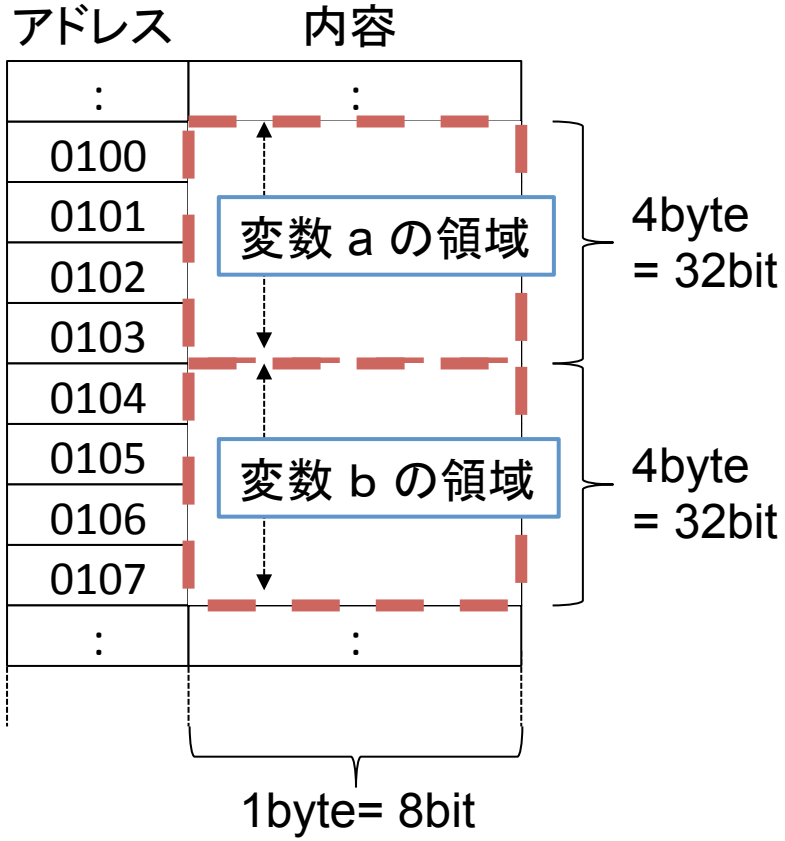
- 変数＝記憶装置上に確保された一定長の領域.
  - 利用前に，確保する処理が必要になる.
  - 利用し終わったら，解放する処理が必要になる.

  → 確保，解放については，コンパイラが自動で実施するコードを付加するような言語もある.

  例）C 言語では malloc でメモリ空間を確保したら free で解放する必要があるが，Java では，解放処理が不要.

変数の概念

- 32bit 処理系で2変数を定義した場合 `int a, b;`
  - メモリ上のどこかの領域に，32bit 分の領域を2つ確保.
  - 確保した領域を，それぞれ a, b という名前で参照できるようにする.
  - 参照した領域に，int (32bit 整数) の値を入出力できるようにする.



変数の4要素

- 変数を定義した場合，次の4要素を定義したことになる.
  - 名前：変数を示す識別子（a とか，b とか）.
  - 属性：参照で示される場所に記憶されている内容(0, 1 のビット列) を，我々の世界の概念 (整数型とか) にどうマッピングするか，どういう演算の種類が許されるのかを規定するもの.
  - 参照：記憶装置上の先頭のアドレス
  - 値： 上記の属性で示された範囲内の値

- たとえば，int a = 100 の場合：

  名前："a"，属性：32ビット整数型，参照：0x0100，値：100

変数宣言

- プログラム内でどのような変数を用いるかを，変数の使用前に明確に示すこと.
  - プログラミング言語によっては，データ型の指定も必要

- 一般的に，次の形式で記述：

(修飾子) (型名) 識別子名

例）
private int a; (Java 言語の場合)
static int i ; (C言語の場合)

変数と型

- 一般的な言語では，変数はその宣言時に，型が規定される.
  - 例：C言語
  - （最適化の存在を無視すれば）宣言された時点で，記憶装置上に，型に応じた領域を確保される，と説明できる.
- 言語によっては，型を規定しない場合もある.
  - 例：Perl 言語や，BASIC では，明示的な変数宣言を必要としない.
  - 変数に最初に代入され時点で，代入される値に応じて，型が決定される（型推論という）
    - 整数が代入されれば，整数型
    - 小数が代入されれば，浮動点小数型

定数

- 変数のうち，中身を変更できないもの．
  - 一度初期化すると，内容を変更することができない．
  - 内容が変換しないことを保証する必要があるときに利用される．
    - 不用意に書き換えられることを防止する．
- 一般的な言語で，変数宣言時に修飾子で定数であることを明示する．

例）
final int a=100; (Java 言語の場合)
const int i=5; (C言語の場合)

- 定数を示す修飾子を付加した場合：
  - 変数宣言時に，値を代入する必要がある．
  - コンパイル言語の場合は，別の箇所で代入すると，コンパイルエラーになる．
  - インタプリタ言語の場合は，実行に失敗する

マクロ

- プログラミング言語によっては，定数と似た機能として「マクロ」が用意されている．

例）
#define MAX 100 (C言語の場合)

- コンパイラがコンパイルを実施する前に，機械的に文字列置換を実施する．
  - 文字列置換を行うプログラムを，プリプロセッサ（preprocessor：前処理を実施するプログラム，という意味）という．
  - C言語の場合は，機械的な文字列置換のため，セミコロンがない．
    - コンパイラによる解釈でないため，C言語の文法と異なる

変数と代入

- 変数に対する代入文の例　x = x + 1; (C言語の場合)

- 数学的な「等式」と考えると，おかしい．
  - 代入，という異なる概念．代入演算子．
  - 等式ならば，比較演算子を利用する．
- 両辺の変数 x の意味は異なる．
  - 右辺：変数 x の現在の値を示す．
  - 左辺：変数 x の領域への参照をあらわす．
- 代入演算子は2引数を持つ関数であると言える．
  - 代入すべき値（右辺）
  - 代入すべき場所への参照（左辺）

変数と有限性

- 普通の人間は，有限なものしか認識できない（はず）
  - コンピュータで扱える情報も，有限なもののみ．

- 無限に桁が存在する数値を表現できない．
  - Q1：循環小数はどうなる？
  - Q2：無理数はどうなる？？
  - Q3：非常に大きい数はどうなる？？？

Q1：循環小数はどうなる？

- そもそも，循環小数ってなに？
  - ある桁から先で，同じ数字の列が無限に繰り返される小数
    - 例：　1/3 = 0.33333…（3の繰り返し）
      1/7=0.142857142857…（142857 の繰り返し）
- どのように表現されるか？
  - 当然，分数では表現されない．
  - データのビット数に合わせて表現できるうちで一番近い浮動小数点に近似して表される．
  - 浮動小数点はあとで．

Q2：無理数はどうなる？？

- そもそも，無理数ってなに？
  - 有理数ではない実数．分子・分母ともに，整数であるような分数で表せない実数．
  - 小数部分が循環しない無限小数．
    - 例：　$\sqrt{2} = 1.41413562373095\ldots$

      $\pi = 3.141592653589793\ldots$

- どのように表現されるか？
  - データのビット数に合わせて表現できるうちで．．（以下略）．

Q3 : 非常に大きい数はどうなる？？？

- そもそも，非常に大きい数ってどのくらい？
  - 言語処理系によっては，64 ビット整数型が用意されているものもある．
    - long long int 型 (C99 仕様)
    - 符号なしならば，18446744073709551616 まで表現可能 (1844京6744兆737億955万1616) まで表現可能
      - ちなみに，全世界の預金額は3000兆円
      - アメリカの国家予算は，209兆5250億円
      - 日本の国家予算は，172兆1250億円
      - 全世界の国家予算ならば足りそう？
- それより大きい数を表したいとき，どのように表現されるか？
  - データのビット数に合わせ．．（以下略）．．．．

浮動小数点数型による表現

- 循環小数や，無理数や，大きな数を表現したいとき，浮動小数点数型で近似して表す．
  - 言語によっては，用意されていないものもある．
  - C 言語，Java 言語：float (32bit 単精度)，double (64bit 倍精度)
  - BASIC：Double (64bit 倍精度)
  - 単精度? 倍精度??
- 丸め誤差が生じる

浮動小数点数型

- コンピュータ内で，整数以外の有限桁の数を表現するための形式
- 物理や化学で，非常に大きな数や，非常に小さな数を表現する場合と同じ方法.

  例）1mol は，$6.022140857 \times 10^{23}$

- コンピュータ内部では，$A \times 2^B$ で表現する.
  - A を仮数部，B を指数部という.
  - 単精度の場合，仮数部と指数部の合計が 32bit
    - 符号部 1 ビット・指数部 8 ビット・仮数部 23 ビット
  - 倍精度の場合，仮数部と指数部の合計が 64bit
    - 符号部 1 ビット・指数部 11 ビット・仮数部 52 ビット
  - いずれにせよ，仮数部の桁数分の精度しか表現できない ⇒ 丸め誤差が発生

丸め誤差

- 数値を，どこかの桁で端数処理した場合に生じる誤差のこと.

- 整数型の場合
  - int 型の変数に，小数を代入した時に，小数点以下の桁のデータが切り捨てられる.
- 浮動小数点数型の場合
  - 仮数部で表すことのできる桁数分だけの桁を超えてしまう部分のデータが切り捨てられる.

浮動小数点での丸め誤差

- コンピュータ内では2進数表現：思わぬ部分で丸め誤差が影響する場合あり.

- 0.1 (10進数) を $A \times 2^B$ で表現するために，2 進数に変換すると，
  - 0 . 0001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 100
    (1001 が循環，有効数字 52 桁)
- これを，再度 10 進数で表現すると，
  - 0.09999999999999964472863211994990706443786621093750
  - 0.1 にならない.
- 演算結果に誤差が生じることになる.

浮動小数点での丸め誤差を生じるプログラム例

- C 言語で次のようなプログラムを記述する

```
#include<stdio.h>

int main(void){
  double d = 0.1 + 0.2;
  if (d == 0.3) printf("Variable d is 0.3.\n");
  else printf("Variable d is not 0.3.\n");
}
```

- 実行結果は，どちらになるか?

A)

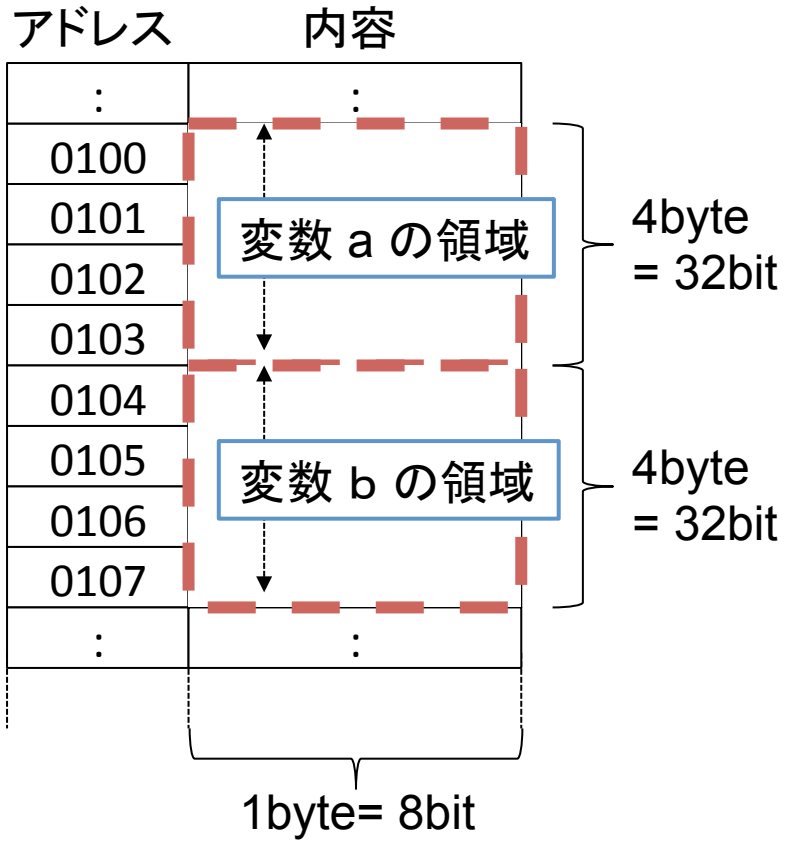
```
Variable d is 0.3.
```

B)

```
Variable d is not 0.3.
```

変数に対する参照方法

- 変数を読み出す場合，次の2通りの参照方法がある．
  - 値を直接参照
  - 変数領域のアドレスから領域内の変数を参照
- 値を代入していると思って代入先を書き換えた場合に，同一の領域を書き換える場合あり．



アドレスの参照

- C 言語で次のようなプログラムを記述する

```
#include<stdio.h>

int main(void){
  int a =1;
  int *b = &a;
  a = 1;
  *b = 100;
  printf(" Variable a is %d\n", a);
}
```

- 実行結果は，どうなるか?

ポインタとアドレス

- C 言語のポインタの記法は，変数のアドレスを自由に設定するための仕掛け．
  - & を変数名の前に付加することで，アドレス参照
  - * を設定した変数では，変数の参照先のアドレス (これをポインタという) を自由に変更できる．

  ⇒ アドレスを代入すると，参照を代入することになるため，同じ領域を参照する．

- Java ではポインタがない．
  - が，しかし，オブジェクトの参照が同じ意味を持つ．

Java の場合

- Java 言語で次のようなプログラムを記述する

```
public class Main {
  public static void main(String args[]){
    Test t1 = new Test();
    Test t2 = t1;
    t1.str = "String1";
    t2.str = "String2";
    System.out.println("t1 is " + t1.str);
    System.out.println("t2 is " + t2.str);
  }
}
```

```
public class Test {
  public String str;
}
```

- 実行結果は，どうなるか?

ヒント

- 第2回のスライド（後半部分に注意）

Java 言語

- オブジェクト指向プログラミングの考え方に基づいて設計された言語(1991，米サン・マイクロシステムズ *James Gosling*)
  - 完全にクラスベースなオブジェクト指向プログラミング
  - 高水準言語（C など）では，機種間でのプログラムの互換性はあったが，機械語プログラムに変換すると互換性がなかった
    → 機種間でバイナリ互換性を保つために，仮想マシン（VM）上で動作する．
- C/C++ の文法から多くを引き継いでいる．
  - 文法的には，ポインタはない．
  - しかし，オブジェクトインスタンスの代入は，参照により行われる．

```
List list1 = new ArrayList();
list1.add("A");
list1.add("B");
List list2 = list1;
list2.clear();
if (list1.isEmpty())
  System.out.println("empty");
```

真になる
（empty が表示）