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

授業の日程　※変更する可能性あり

9/17	ガイダンス・復習
9/24	プログラム言語の歴史
10/1	構文，意味，BNF
10/8	識別子，変数
10/15	文，式
10/22	データ型，スコープ
10/29	手続き，制御構造
11/5	中間試験

11/12	モジュール
11/19	データ抽象化
11/26	（月曜授業日）
12/3	（土曜授業日）
12/10	例外処理
12/17	並行，排他制御
12/24	関数型言語，OO言語，言語の分類
1/14	まとめ
1/21	期末試験

プログラミング言語　第6回

- （前回の残り）
  - 式評価
  - 文
- データ型
  - シラバスに書き忘れてました．ごめんなさい．
- スコープ
  - たぶん，無理なので，次回に回します．
- 中間試験の話

復習：式

- 値を算出する目的で記述

- 構成要素
  - 定数
  - 変数
  - 演算子
  - 関数呼び出し

- 式の評価は，値の代入や，演算子の処理により実施される．
- 式の実行前後で変数の値が変わる場合も，変わら無い場合もある．

復習：演算子の評価順序

- 演算子の評価順序は，言語によって規定されている．
  数学，工学，科学全般での優先順位と共通．
  1. かっこ内の項
  2. べき乗，累乗根
  3. 乗法，除法
  4. 加法，減法

- 式の評価は，左から右に行う．

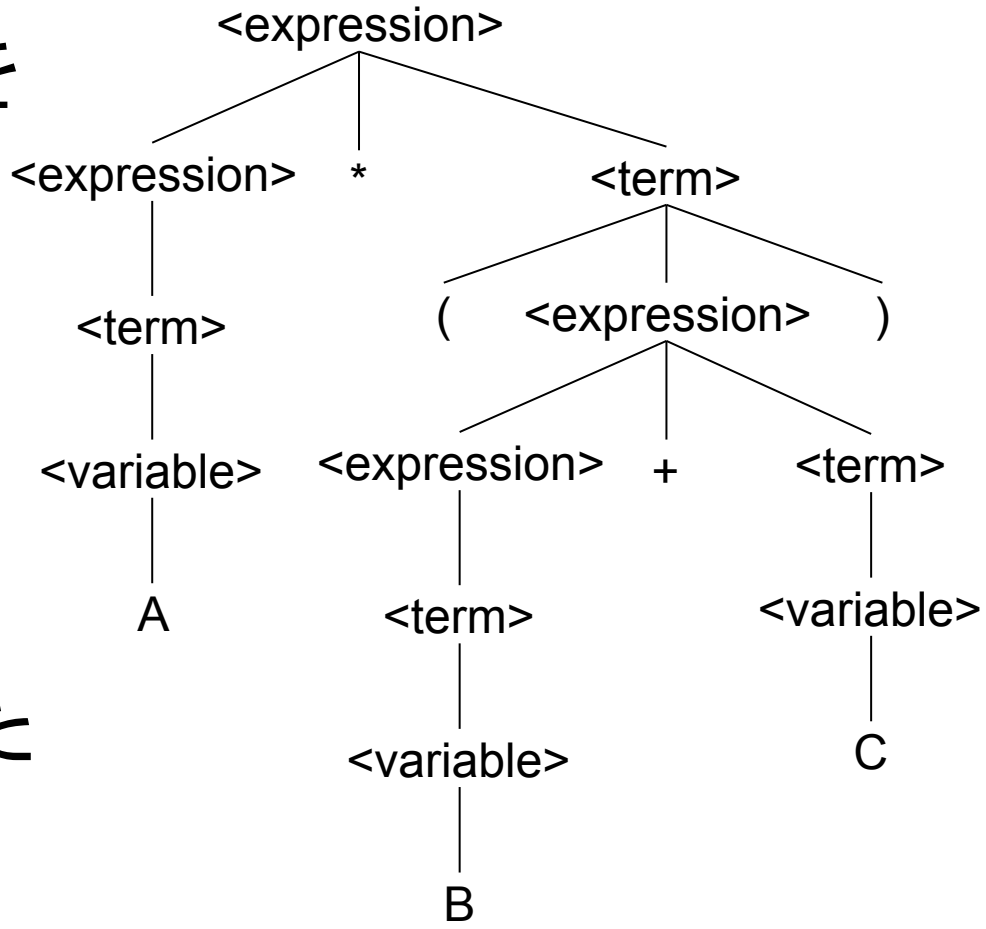
式評価の BNF 表現

- A*(B+C) を受理できるBNF

```
<expression> ::= <term> | <expression> ‘+’ <term>
                                | <expression>  ‘*’ <term>
<term> ::= <variable> | ( <expression> )
<variable> ::= 'A' | 'B' | 'C’
```

構文木

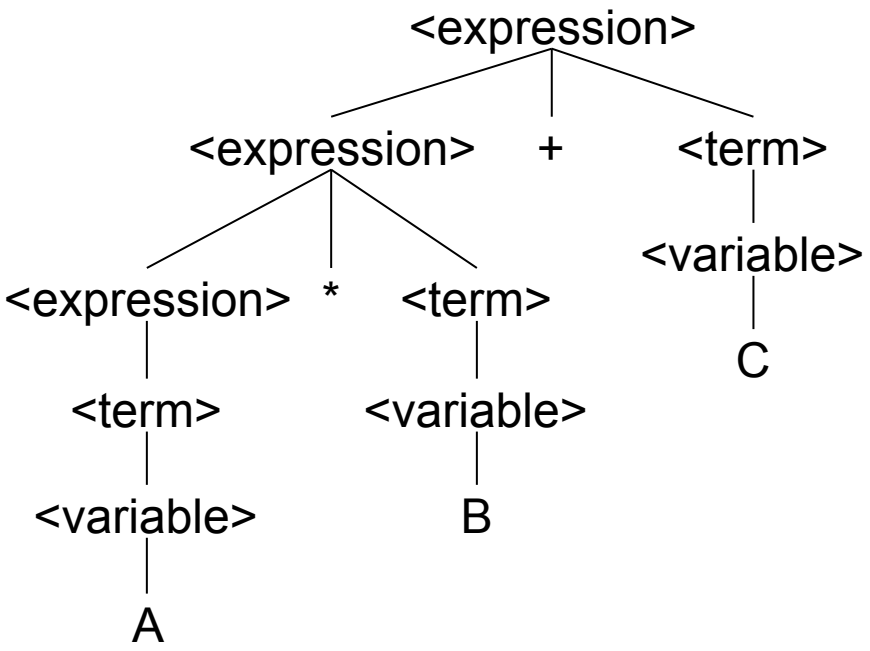
- 特定の言語定義を使って実際の言語を解析した結果を表すための木構造図



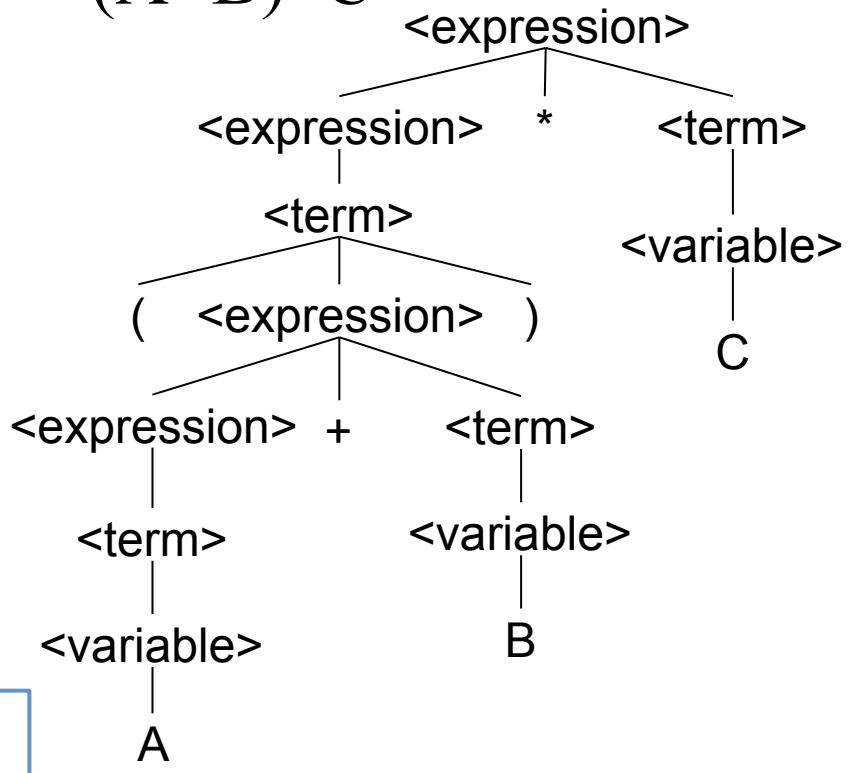
- 前の BNF で A*(B+C) を解析した場合:

構文木と演算順序

- 解析対象により，構文木の形状は異なる．
  - A*B+C



  - (A+B)*C



この BNF では，
<expression> '+' <term>
<expression> '*' <term>
の順序で表さないと受理しないことに注意

曖昧な文法

- 次の文法定義があったとする

```
<expression> ::= <variable> | <expression> ‘+’ <expression>
                             | <expression>  ‘*’ <expression>
<expression> ::= <expression> | ( <expression> )
<variable> ::= 'A' | 'B' | 'C’
```

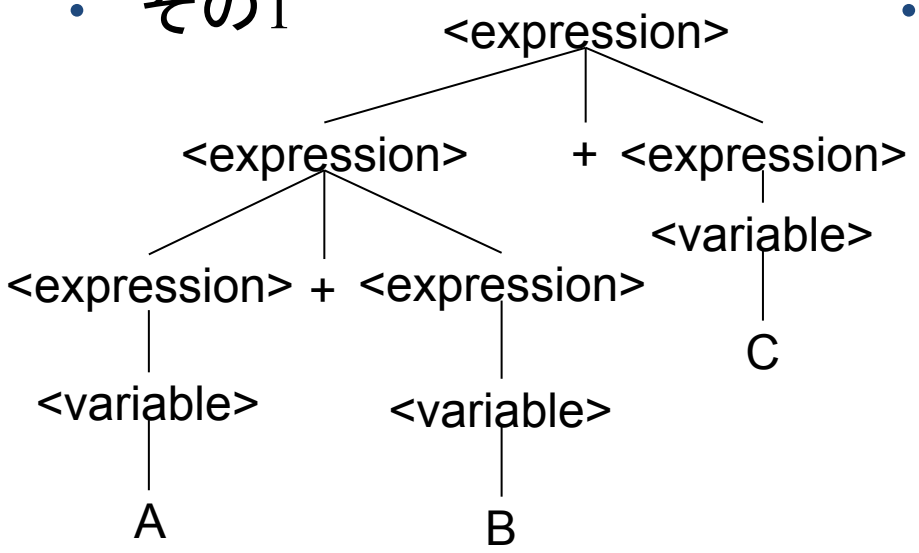
- 上の BNF で A+B+C を受理することは「曖昧」さを含んでいる.
- 先ほどの BNF（下記）と比較してみてください

```
<expression> ::= <term> | <expression> ‘+’ <term>
                        | <expression>  ‘*’ <term>
<term> ::= <variable> | ( <expression> )
<variable> ::= 'A' | 'B' | 'C’
```

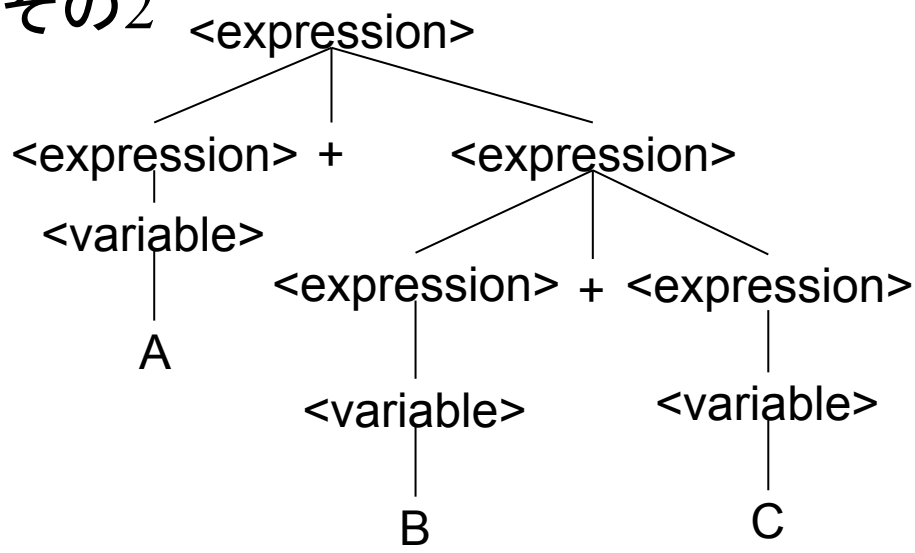
構文の曖昧さ

- 同じA+B+Cの受理に対して，構文木が複数かける．
  - 同じ言語の受理に対して，解析処理が定まらない→ 曖昧な文法
    → 構文解析の効率が大幅に低下する

- その1



- その2



<expression> ::= <variable> | <expression> ‘+’ <expression> | <expression> ‘*’ <expression>
<expression> ::= <expression> | (<expression>)
<variable> ::= 'A' | 'B' | 'C’

文の構造の種類

- 分岐構造
  - IF-THEN-ELSE 構造
  - CASE-WHEN（SWITCH-CASE) 構造
- 繰り返し（反復）構造
  - WHILE-DO-END 構造
  - DO-WHILE (REPEAT-UNTIL) 構造
  - LOOP-REPEAT 構造
- GOTO 文とラベル

分岐構造

※ Ada の場合

- IF-THEN-ELSE 構造

```
if C1 then S1
elsif C2 then S2
elsif C3 then S3
……
elsif Cn then Sn
else Sn+1
endif;
```

- C 言語では，一部をブロック構造で表現.
- elsif を示す専用キーワードが存在する言語もある.

- CASE-WHEN 構造

```
case V is
  when C1 S1
  when C2 S2
……
  when Cn Sn
  when others Sn+1
end case;
```

- C 言語では，一部をブロック構造で表現.
  - switch-case 構造

繰り返し（反復）構造

※ Pascal の場合

- WHILE-DO-END 構造

```
while COND do
  STATEMENT
end
```

- REPEAT-UNTIL 構造

```
repeat
  STATEMENT
until COND
```

- それぞれ，前判定，後判定
  - C 言語だと，後判定は DO-WHILE.

- LOOP-REPEAT 構造

```
loop
  if not COND then exit
  STATEMENT
repeat
```

```
loop
  STATEMENT
  if COND then exit
repeat
```

- 処理実行前に，条件を満たさない場合に終了 or処理実行後に，条件を満たす場合に終了.
- C 言語だと，無限ループと if + break で書く.

GOTO 文とラベル

- ラベルを定義して，goto 文で自由にジャンプできる
  （右はC言語の場合．
  "a\n"は表示されない）

```
int main(void){
        goto test;
        printf("a\n");
    test:
        printf("b\n");
        return 0;
}
```

- 前ページまでにあげた
  制御構造内から，自由
  に外に飛ぶことが可能
- 逆に，制御構造内に入ることも可能．

→ 便利ではあるが，制御構造を崩してしまう．
（スパゲッティプログラム）

データ型

- すべての言語は，データ型のセットを有している.

- 一般的に次のものは存在することが多い:
  - 整数 (C 言語では integer)
  - 実数 (double や float)
  - 文字型 (char)
  - 論理型，ブール型 (C 言語では該当なし)

データ構造による分類

- 基本データ型
- 列挙型
- ポインタ型
- 構造体型
- オブジェクト型

基本データ型（プリミティブ型）

- プログラミング言語組み込みで提供されるデータ型 ⇔ 対義語：複合型
- 種類は，プログラミング言語により異なる．
  - C 言語では文字列が複合型，BASIC では，基本データ型．
- コンピュータメモリ上のオブジェクトと1対1対応の場合もあるが，そうでないこともある．
  - もっとも高速な演算を行える言語要素である場合が多い．
- 代表的なプリミティブ型
  - 整数型（int），文字型（char），浮動小数点数型（double）

列挙型

- 識別子をそのまま有限集合として持つデータ型.
- 例:トランプの4つの記号を識別したい時
  方法1）type = 1, 2, 3, 4 として識別
  方法2）type = Clubs, Diamonds, Hearts, Spades として識別
  - どちらがわかりやすいか？？
- 列挙型では，識別するための番号を意識せずに，識別子のみを列挙して識別可能.
- C 言語の例:

```
#include <stdio.h>
enum cardtype { CLUBS, DIAMONDS, HEARTS, SPADES };
int main (void){
        printf("%d, %d, %d, %d\n", CLUBS, DIAMONDS, HEARTS, SPADES );
        return 0;
}
```

実行結果: 0, 1, 2, 3

ポインタ型

- あるオブジェクトに対して，論理位置情報（アドレス）で参照する方法.
  - 例は省略.
- C 言語は，UNIX を記述するために開発.
  - アセンブラが実行できるほぼすべての操作を行える必要があった.
  - 特定のメモリ領域を直接参照してアクセスできる必要性があったため，強力なポインタ機能を備える.
- デメリット：不正な領域でも参照できてしまうため，バグや深刻なセキュリティ問題を引き起こす.
  - 直接アドレスを参照する言語は，今後は減少する傾向

構造体型

- 1つ，もしくは複数の値をまとめて格納できるデータ型.
  - それぞれのメンバの型が異なっても良いことが，配列と異なる.
- 関数を入れることも可能.

```
typedef struct {
        int x0, y0;
        char point_name;
} Location;
. . .
Location point1, point 2;
point1.x0 = 100;
point1.y0 = 200;
point1.point_name = "A";
point2.x0 = 300;
point2.y0 = 400;
point2.point_name = "X";
```

構造体とオブジェクト指向

```
#include <stdio.h>
#include <strings.h>

typedef struct{
  char str [150];
  void (*setName)(char *, char *);
}Name;

void setNameMr(char *to, char *from){
  strcpy(to, "Mr.");
  strcat(to, from);
}

void setNameMs(char *to, char *from){
  strcpy(to, "Ms.");
  strcat(to, from);
}
```

（左下から続く）

```
int main(void){
  Name name;

  name.setName = setNameMr;
  name.setName(name.str, "Tanaka");
  printf("Hello %s!\n", name.str);

  name.setName = setNameMs;
  name.setName(name.str, "Yamada");
  printf("Hello %s!\n", name.str);
}
```

実行結果：

```
Hello Mr.Tanaka!
Hello Ms.Yamada!
```

- データとそれを操作する関数をひとまとまりの構造体として表現 → オブジェクト指向の考えへ

オブジェクト型

- 関連するデータと，それらに対する手続きをまとめて表せるようにしたデータ型
- カプセル化により，外部から不要な操作ができないようにする
- 右の例では，getSum 関数経由でないと，CalcSum クラスの変数 sum に操作できない．

1から100までの総和を求めるプログラム

```
#include <iostream>

class CalcSum {
    int sum;
  public:
    CalcSum(int start, int end){
      sum = 0;
      for (int i = start; i <= end; i++)
        sum += i;
    }
    int getSum() { return sum; }
};
int main(void){
  CalcSum sum(1, 100);
  std::cout << sum.getSum() <<"\n";
}
```

先ほどの例 (C言語)

- Name 構造体で，文字列とそれを操作する関数をまとめている　＝ オブジェクト指向プログラム，と言える．
  - ただし，name.str に直接代入することもできてしまう．

  ⇒ カプセル化ができないことが，オブジェクト型との相違．

```
#include <stdio.h>
#include <strings.h>

typedef struct{
  char str [150];
  void (*setName)(char *, char *);
}Name;

void setNameMr(char *to, char *from){
  strcpy(to, "Mr.");
  strcat(to, from);
}
```

```
（左下から続く）

int main(void){
  Name name;

  name.setName = setNameMr;
  name.setName(name.str, "Tanaka");
  printf("Hello %s!\n", name.str);

}
```

実行結果：

```
Hello Mr.Tanaka!
Hello Ms.Yamada!
```

オブジェクト指向プログラミング言語

- 言語の枠組みとして，オブジェクト指向プログラムを書きやすくした言語.
  - オブジェクトを「指向」している
- 次の性質を備える.
  - カプセル化：データや振る舞いの隠蔽を可能にする性質
  - 継承：既存の部品を拡張した新しい部品を作成できる性質
  - 多態性（ポリモーフィズム）：同じ名前でクラスにより異なる動作の関数（メソッド）を作成できる性質
    （別の構造体に，同じ名前の関数ポインタを入れると，似たような感じになる）
  - 動的バインディング：引数型を，コンパイル時に決定せずに，プログラム実行時に決定することのできる性質（抽象データ型で受け取り，代入された引数の型で振る舞いを変える）

中間試験範囲はここまで

- 試験は，11月5日.
- 60分，持ち込み不可.

- 内容:

1. プログラミング言語の変遷について（2回）
2. Syntax，Semantics，文法，BNF について（3回）
3. 識別子，予約語，変数，定数，マクロ，内部データ表現，丸め（4回，5回）
4. 式，参照透過性，構文木，曖昧な文法 （5回，6回）

試験と成績

- 成績
  - 期末試験60%．中間試験40%
- レポート
  - 1回レポートを課す予定（たぶん11/19出題）
  - 提出する必要がある
  - 未提出　⇒　不合格（試験の受験資格を失う）
  - 再提出は無しにするつもり．内容にあまりにも不備がある場合は，未提出扱いになることがある．
  - 内容がよければ，加点扱いしようかと思っています．
- 出席（カードリーダ）
  - 原則として4回以上欠席した場合に，試験の受験資格を失う