

情報処理概論

(Basic Theory of Information Processing)

第 5 回：制御構造 (条件分岐 , 繰り返し)

概要

- if 文
- for 文
- while 文
- break 文 , continue 文
- プログラムを考えてみよう。

8 制御構造

- プログラムは通常は先頭から順番に実行される。
- 制御構造とは、プログラムの実行の流れを変えるためのもの。

制御構造のために使われる文

- 条件分岐：if 文，switch 文：
ある条件が満たしているときだけ実行する。
- ループ文：for 文，while 文，do while 文：
条件を満たしている間，繰り返して実行する。

8.1 if文 (p.52)

8.1.1 if文だけの場合

文1

if (条件式) 文2;

文3

- まず, 文1を実行する。
- 条件式がtrueならば文2を実行し, 文3を実行する。
- falseならば, 文2を実行しないで文3を実行する。

8.1.2 if-else文の場合 (p.54)

```
if (条件式) 文1;  
else 文2;
```

- もし条件式がtrueならば文1を実行し，文2を実行せずに次の文に進む。
- falseならば文1を実行せずに，文2を実行し，次の文に進む。
- 条件の成立 / 非成立したがい，文1 / 文2のいずれか1つだけが実行される。
- 1つのif文に対して，else文は1ひとつだけである。

8.1.3 if-else 文の組み合わせ

```
if (条件式 1) 文 1;  
else if (条件式 2) 文 2;  
else if (条件式 3) 文 3;  
else 文 4;
```

- もし条件式 1 が true ならば文 1 を実行する。
- 条件式 1 が false で条件式 2 が true ならば文 2 を実行する。
- 条件式 1 と条件式 2 が false で条件式 3 が true ならば文 3 を実行する。
- どの条件も false ならば文 4 を実行する。
- 文 1 , 文 2 , 文 3 , 文 4 のいずれか 1 つだけが実行される。
- 1 つの if 文に対して , else if 文は複数あっても良い。
- else 文はなくてもよい。

8.1.4 複数の文を {} にくくれば, 1つの文として扱われる

- 今までの例では, 条件が成立した場合に実行する文は 1 つしか書けない。
- 複数の文を書きたいときは, 「文」のところを以下のように置き換える。
(詳細は, 次ページ例の方がわかりやすい)

```
{  
  文 1;  
  文 2;  
  ...  
  ...  
  文 N;  
}
```

{文 1; ... ; 文 N }で, 1つの文とみなされる。

プログラム例 (if文だけ)

```
public class SampleIf2 {  
    public static void main(String args[]) {  
        int num = 3;  
        if (num == 3) System.out.println("num は3である。");  
    }  
}
```

または、以下のようにも書ける。

```
public class SampleIf {  
    public static void main(String args[]) {  
        int num = 3;  
        if (num == 3) {  
            System.out.println("num は3である。");  
        }  
    }  
}
```

プログラム例 (if-else 文)

```
public class SampleIfElse {  
    public static void main(String args[]) {  
        int num = 3;  
  
        if (num == 3) {  
            System.out.println("num は3である。");  
        } else {  
            System.out.println("num は3でない。");  
        }  
    }  
}
```

プログラム例 (if-else 文でネストするしている)

```
public class SampleIfElseNest {  
    public static void main(String args[]) {  
        int i = 2, j = 3;  
        if (i == 1) {  
            if (j == 2) {  
                System.out.println("i は1で , j は2である。");  
            } else {  
                System.out.println("i は1で , j は2でない。");  
            }  
        } else {  
            if (j == 3) {  
                System.out.println("i は1でないが , j は3である。");  
            } else {  
                System.out.println("i は1でなく , j も3でない。");  
            }  
        }  
    }  
}
```

8.2 switch case 文 (p.97)

- 場合分けを簡単に書くことができる。
- `switch(i) {case 文を含んだ文の並び}` で , 変数 `i` の値を等しいものを探することができる。
- `i` の値が , `case 定数:` で指定した定数と一致すれば , それ以下の文が実行される。
- `default:` 文は , どの `case` 文の定数にも当てはまらないときに実行される。
- `break` 文で , `switch` の外へ出る。

注意 :

- `case` 文で指定する値は , 定数 (リテラル) でなくてははいけない。
- 場合分けにおいては , `else if` 文を連ねていくよりも処理が速い。
ハッシュを使ってインプリメントされる。

```
public class SampleSwitch {  
    public static void main(String args[]) {  
        int i = 2;  
        switch(i) {  
            case 1:  
                System.out.println("i は1である。");  
                break;  
            case 2:  
                System.out.println("i は2である。");  
            case 3:  
                System.out.println("i は3である。(いや, 2かもしれない。)");  
                break;  
            case 4:  
                System.out.println("i は4である。");  
                break;  
            default:  
                System.out.println("i は1,2,3,4でない。");  
        }  
    }  
}
```

if文で書いてみると、以下のようなになる。

```
public class SampleSwitchIf {  
    public static void main(String args[]) {  
        int i = 2;  
        if (i == 1) {  
            System.out.println("i は1である。");  
        } else if (i == 2) {  
            System.out.println("i は2である。");  
            System.out.println("i は3である。(いや、2かもしれない。)");  
        } else if (i == 3) {  
            System.out.println("i は3である。(いや、2かもしれない。)");  
        } else if (i == 4) {  
            System.out.println("i は4である。");  
        } else {  
            System.out.println("i は1,2,3,4でない。");  
        }  
    }  
}
```

8.3 for 文

- 文の繰り返し実行 (だんだん , 難しくなってくる)

for (初期設定 ; 条件式 ; 繰り返しの最後に実行する文) 文 ;
--

1. 「初期設定」を実行する。
2. 「条件式」を評価する。
 - 「条件式」が , true ならば ,
 - (a) 「文」を実行する。
 - (b) 「繰り返しの最後に実行する文」を実行する。
 - (c) 2にもどる
 - 「条件式」が , false ならば , for 文の次の文を実行する。

注意 :

- 「文」を { 文1 ; 文2 ; ... } とすれば , 「文」の実行において , 文1 , 文2 , ... を繰り返し置き換えて実行する。

例：5から8まで加算するプログラム

```
class SampleFor {  
    public static void main(String[] args) {  
        int i, sum = 0;  
        for ( i = 5 ; i <= 8 ; ++i ) {  
            sum = sum + i  
        }  
        System.out.println("5 + 6 + 7 + 8= " + sum);  
    }  
}
```

例：5から8まで加算するプログラム

```
class SampleFor {  
    public static void main(String[] args) {  
        int i, sum = 0;           ⇐ sum が0になる  
        for ( i = 5 ; i <= 8 ; ++i ) {  
            sum = sum + i  
        }  
        System.out.println("5 + 6 + 7 + 8= " + sum);  
    }  
}
```

例：5から8まで加算するプログラム

```
class SampleFor {  
    public static void main(String[] args) {  
        int i, sum = 0;  
        for ( i = 5 ; i <= 8 ; ++i ) {  
            sum = sum + i  
        }  
        System.out.println("5 + 6 + 7 + 8= " + sum);  
    }  
}
```

⇐ iが5になる，条件が成立するので，
⇐ が実行される

例：5から8まで加算するプログラム

```
class SampleFor {  
    public static void main(String[] args) {  
        int i, sum = 0;  
        for ( i = 5 ; i <= 8 ; ++i ) {  
            sum = sum + i  
        }  
        System.out.println("5 + 6 + 7 + 8= " + sum);  
    }  
}
```

⇐sumは, $0 + 5 = 5$ になる

例：5から8まで加算するプログラム

```
class SampleFor {  
    public static void main(String[] args) {  
        int i, sum = 0;  
        for ( i = 5 ; i <= 8 ; ++i ) {  
            sum = sum + i  
        }  
        System.out.println("5 + 6 + 7 + 8= " + sum);  
    }  
}
```

⇐ iが6になる，条件が成立するので，
⇐ が実行される

例：5から8まで加算するプログラム

```
class SampleFor {  
    public static void main(String[] args) {  
        int i, sum = 0;  
        for ( i = 5 ; i <= 8 ; ++i ) {  
            sum = sum + i  
        }  
        System.out.println("5 + 6 + 7 + 8= " + sum);  
    }  
}
```

⇐sumは, 5 + 6 = 11になる

例：5から8まで加算するプログラム

```
class SampleFor {  
    public static void main(String[] args) {  
        int i, sum = 0;  
        for ( i = 5 ; i <= 8 ; ++i ) {  
            sum = sum + i  
        }  
        System.out.println("5 + 6 + 7 + 8= " + sum);  
    }  
}
```

⇐ iが7になる，条件が成立するので，
⇐ が実行される

例：5から8まで加算するプログラム

```
class SampleFor {  
    public static void main(String[] args) {  
        int i, sum = 0;  
        for ( i = 5 ; i <= 8 ; ++i ) {  
            sum = sum + i  
        }  
        System.out.println("5 + 6 + 7 + 8= " + sum);  
    }  
}
```

⇐sumは, 11 + 7 = 18になる

例：5から8まで加算するプログラム

```
class SampleFor {  
    public static void main(String[] args) {  
        int i, sum = 0;  
        for ( i = 5 ; i <= 8 ; ++i ) {  
            sum = sum + i  
        }  
        System.out.println("5 + 6 + 7 + 8= " + sum);  
    }  
}
```

⇐ iが8になる，条件が成立するので，
⇐ が実行される

例：5から8まで加算するプログラム

```
class SampleFor {  
    public static void main(String[] args) {  
        int i, sum = 0;  
        for ( i = 5 ; i <= 8 ; ++i ) {  
            sum = sum + i  
        }  
        System.out.println("5 + 6 + 7 + 8= " + sum);  
    }  
}
```

⇐sumは, $18 + 8 = 26$ になる

例：5から8まで加算するプログラム

```
class SampleFor {  
    public static void main(String[] args) {  
        int i, sum = 0;  
        for ( i = 5 ; i <= 8 ; ++i ) { ⇐ iが9になる , 条件が成立しないので ,  
            sum = sum + i  
        }  
        System.out.println("5 + 6 + 7 + 8 = " + sum); ⇐ が実行される  
    }  
}
```

5 + 6 + 7 + 8 = 26

と表示される。

(for 文を出た後のiの値は , 9である。)

8.4 ネスティング

- for 文の中に for を入れることができる。
- 例：1 + 1 + 2 + 1 + 2 + 3 + 1 + 2 + 3 + 4 + 1 + 2 + 3 + 4 + 5
を実行するプログラム

```
class ForNesting {  
    public static void main(String[] args) {  
        int i, j, sum = 0;  
        for ( i = 1 ; i <= 5 ; ++i ) {  
            for ( j = 1 ; j <= i ; ++j ) {  
                sum = sum + j  
            }  
        }  
        System.out.println("Answer = " + sum);  
    }  
}
```

8.5 while 文

```
while ( 条件式 ) 文;
```

- 条件式がtrueの間, 繰り返し「文」を実行する。

例 :

```
class SampleWhile {  
    public static void main(String[] args) {  
        int beki = 0, num = 32;  
        while ( num > 1) {  
            num /= 2;  
            ++beki;  
        }  
        System.out.println("32 = 2^" + beki);  
    }  
}
```

無限ループ :

```
while (true) 文;
```

8.6 do while 文

```
do 文 while ( 条件式 );
```

- まず，文を実行する。
- 次に，条件式がtrueの間，繰り返し「文」を実行する。

while 文との違い

- while 文では，ループの初期状態において条件が成立しない場合は，文は1回も実行されない。
- do while 文では，ループの初期状態において条件が成立しない場合でも，1回は実行される。
- 条件を文の実行前にチェックするか，実行後にチェックするの違い。

(あまり使われない。)

例題

SampleFor を，while 文を使って同じ動作をするように書き換える。

8.7 break 文

- ループのbreak文のところからループを終了して，次の文を実行する。
- ネスティングしている場合は，そのbreakを含む最も内側のループ1つだけから抜ける。

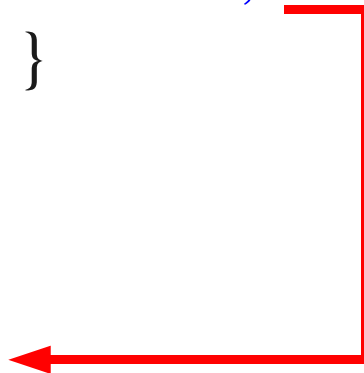
```
for (i = 0 ; i < 10 ; i++ ) {
```

```
    if (j == 5) {
```

```
        break;
```

```
    }
```

```
}
```



```
while(i < 10) {
```

```
    if (j == 5) {
```

```
        break;
```

```
    }
```

```
}
```



(例 : PrimeBreak.java)

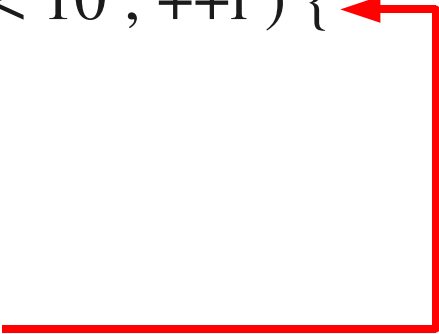
8.8 continue 文

- `continue` 文のところからループの先頭に戻り，文を実行する。

注意点：`continue` 文を実行すると，

- `for` 文のときは，「繰り返しの最後に実行する文」と「条件式」を実行し `true` ならば，`for` 文の「文」を実行する。
- `while` 文のときは，「条件式」を実行し `true` ならば，`while` 文の「文」を実行する。
- 両方の場合において，「条件式」が `false` ならば，文を実行しないでループを終了する。

```
for (i = 0 ; i < 10 ; ++i ) {  
    if (j == 5) {  
        continue;  
    }  
}
```



```
while(i < 10) {  
    if (j == 5) {  
        continue;  
    }  
    ++i;  
}
```



次の2つのプログラムの違いに注意！

```
int i, k;
k = 6;
for (i = 1 ; i < 10 ; ++i) {
    if (k % i != 0) continue;
    System.out.println(i + " は " + k + " を割り切る");
}
```

```
int i, k;
k = 6;
i = 1;
while (i < 10) {
    if (k % i != 0) continue;
    System.out.println(i + " は " + k + " を割り切る");
    ++i;
}
```

8.9 今日の最後に

- 制御構造がはいると，動作が複雑になる。
- プログラムがどのように動くか，変数の値がどのように変わっていくか，自分の頭でしっかり考えることが絶対に必要。
- あとは配列を覚えれば，基本的には計算できるものはなんでも計算できます。