C 言語の学習
制御文

山本昌志*

2006 年 5 月 9 日

概 要

C 言語の基本制御構造について述べている．教科書の 9 章の内容の全てを説明している．ここでは，多くの図を用いて，分かり易く説明しているつもりである．プログラマーが考えたアルゴリズムをプログラムに反映させるためには，ここの内容をきちんと理解しなくてはならない．

1 本日の学習内容

教科書の 9 章の制御文を学習する．細かい説明を行っているが，以下のことが理解できればよい．

9 章 制御文

- `if`〜`else` の使い方がわかる．
- `for` 文の使い方がわかる．

2 プログラムの基本制御構造

C 言語のような構造化プログラミング言語では，**順次**，**選択**，**繰り返し** (反復，ループ) を組み合わせて処理を記述する．C 言語のプログラムは，この 3 つの基本形に分解できる．これが理解できれば，容易にプログラムの作成ができるようになるであろう．これらのフローチャートを図 1〜図 3 に示す．

順次 これはプログラムの文を上から下へと実行する構造で，特にこれを表す命令はない．

選択 値により，実行する文が異なる構造である．C 言語には `if` 文と `switch` 文がある．`switch` 文はプログラムの構造が分かり難くるので，`if` 文を使うことが望ましい．

繰り返し 同じ文を繰り返す構造である．C 言語には，`for` 文と `while` 文 `do-while` 文がある．

*独立行政法人 秋田工業高等専門学校 電気工学科

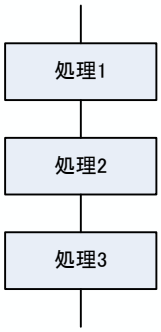

図 1: 順次の構造

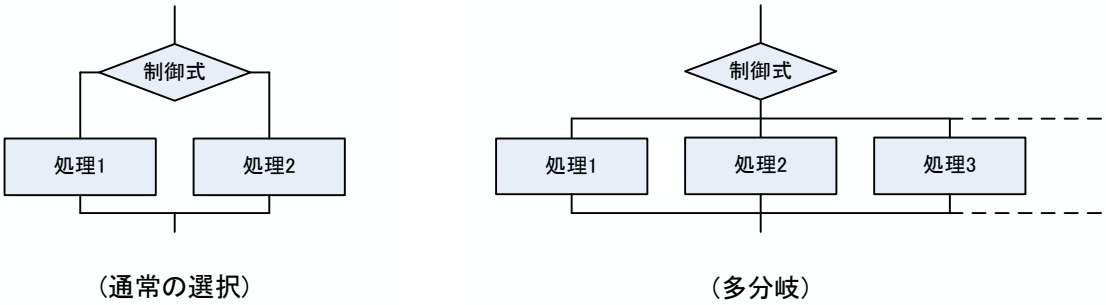

図 2: 選択の構造

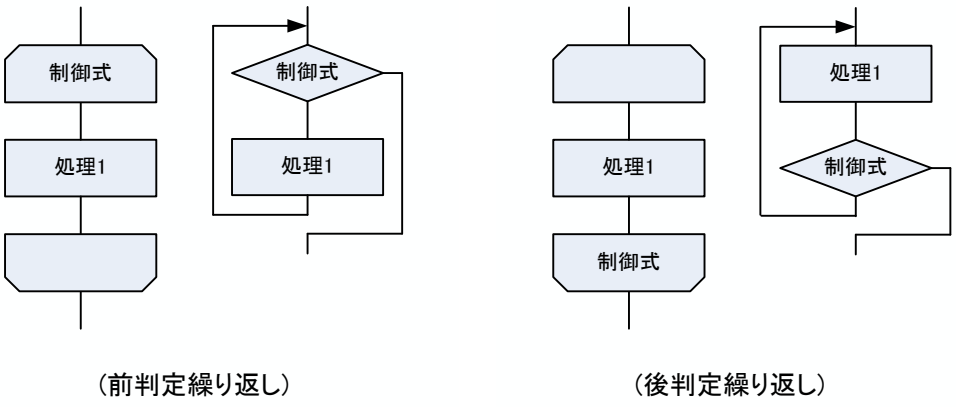

図 3: 繰り返しの構造

3　基本制御構造 (順次)

これは，もっとも基本的な構造で，文を上から下へと順次，実行していく．いつもおなじみの構造で，以下のように記述する．

書式

```
文 1;
文 2;
文 3;
```

文の数は，いくら書いてもよい．フローチャートで書くと，図 4 のようになる．以下のようなプログラムが，この構文の使用例である．

```
a=3;
b = a*a;
printf("%d*%d=%d\n",a,a,b);
```

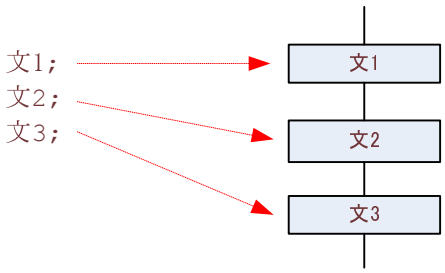

図 4: 順次の構文のフローチャート

4　基本制御構造 (選択)

C 言語の選択には，if と switch がある．ここでは，それぞれについて説明する．

4.1　if else 文 (p.138)

プログラム中で,「もし○○ならば，△△する」というような処理をしたい場合，if という命令を使う．また,「もし，○○ならば△△する，さもなければ□□する」という場合は，if と else を使う．ここでは，この if や else の使い方を学習する．

4.1.1 処理が 1 つの場合

最初は一番単純な,「もし○○ならば，△△する」という構文を示す．とくに，△△の部分が 1 つの文で表せる場合である．このような場合は，次のように，書く．

書式

```
if(制御式) 文;
```

条件を表す○○の部分が制御式で，△△の部分を文で表すのである．これは,「制御式が正しい (真) ならば，文を実行する」となる．もし，制御式が誤り (偽) であれば，この文は実行されず次の行に移る．図 5 にこの構文のフローチャートを示す．

以下のようなプログラムが，この構文の使用例である．

```
if(a<=10) printf("a は，10 以下です\n");
```

- a が 10 以下ならば，
  - 「a は，10 以下です」と表示する．

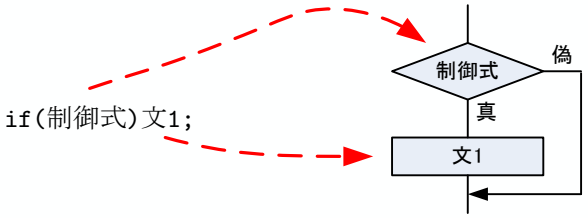

図 5: 制御式が真の場合，1 つの処理を実施する if 文

4.1.2 ブロックで処理する場合

先ほどの構文では，実行できる文は 1 個に限られる.「もし○○ならば，△△し，□□し，･･･」のように複数の文を実行したい場合がある．このようなときは，次に示すように，括弧 { } でくくり，ブロック化して，複数の文を書く．ブロック内には，任意の数の文を書くことができる．また，このブロック内には，順次や選択，繰り返しの文を書くことも可能である．

書式

```
if(制御式){
    文 1;
    文 2;
    文 3;
}
```

これは,「制御式が正しい (真) ならば，文 1 と文 2，文 3 を実行する」となる．もし，制御式が誤り (偽) であれば，これら文は実行されず，ブロックの外側に出る．図 6 にこの構文のフローチャートを示す．

以下のようなプログラムが，この構文の使用例である．

```
if(0<=a && a<=10){
  printf("a は，0 以上\n");
  printf("かつ\n");
  printf("a は，10 以下です\n");
}
```

- もし，a が 0 以上，かつ，10 以下ならば，
  - 「a は，0 以上」と表示する．
  - 「かつ」と表示する．
  - 「a は，10 以下です」と表示する．

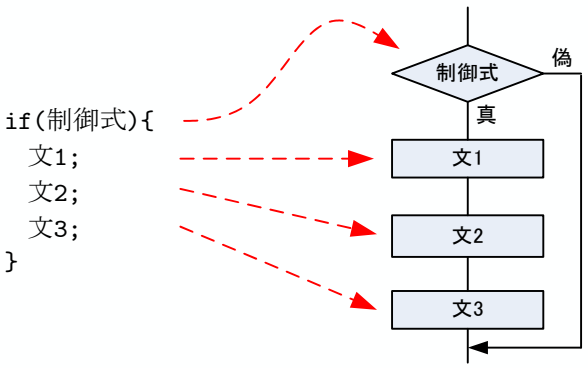

図 6: 制御式が真の場合，ブロックで処理を実施する if 文

4.1.3 2 分岐の場合

「もし○○ならば△△し，さもなければ□□する」というように，条件により二者択一の選択処理が必要な場合がある．これは，次のように書く．

書式

```
if(制御式){
   文 1;
   文 2;
   文 3;
}else{
   文 4;
   文 5;
   文 6;
}
```

これは,「制御式が正しい (真) ならば，文 1 と文 2，文 3 を実行する．さもなければ，文 4 と文 5，文 6 を実行する.」となる．実行される文が複数であるので，ブロックになっていることに注意．文が 1 つの場合,, { と } でくくり，ブロック化しなくても良い．図 7 にこの構文のフローチャートを示す.

以下のようなプログラムが，この構文の使用例である．

```
if(0<=a && a<=10){
  printf("a は，0 以上\n");
  printf("かつ\n");
  printf("a は，10 以下です\n");
}else{
  printf("a は，0 未満\n");
  printf("または\n");
  printf("a は，10 より大きい\n");
}
```

- もし，a が 0 以上，かつ，10 以下ならば，
  - 「a は，0 以上」と表示する．
  - 「かつ」と表示する．
  - 「a は，10 以下です」と表示する．
- さもなければ
  - 「a は，0 未満」と表示する．
  - 「または」と表示する．
  - 「a は，10 より大きい」と表示する．

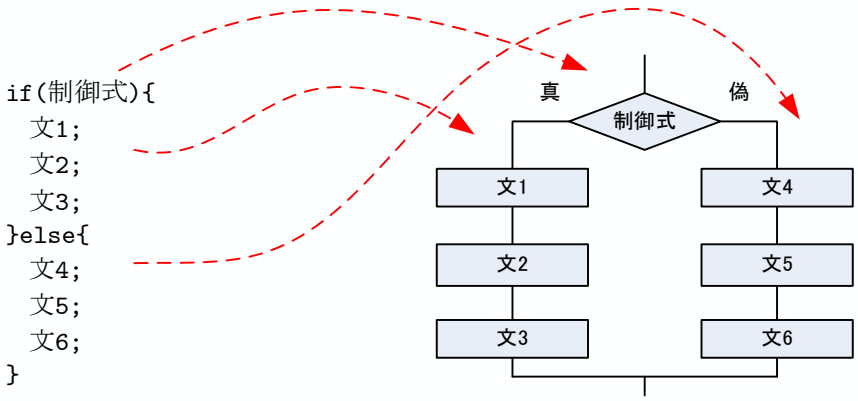

図 7: else を使って，二者択一の処理をする構文

4.1.4 連続制御の分岐

「もし○○ならば ⊛⊛ する．さもなければ，もし□□ならば ⊠⊠ する．さもなければ，もし △△ ならば ▽▽ する．さもなければ，◎◎ する.」というよう構文を書きたい場合がある．条件に合致しなければ，次の条件と，次々に条件を変ている．これは，次のように書く．

書式

```
if(制御式 1){
    文 1;
    文 2;
}else if(制御式 2){
    文 3;
    文 4;
}else if(制御式 3){
    文 5;
    文 6;
}else{
    文 7;
    文 8;
}
```

これは,「制御式 1 が正しい (真) ならば，文 1 と文 2 を実行する．さもなければ，制御式 1 が正しいならば，文 3 と文 4 を実行する．さもなければ，制御式 3 が正しいならば，文 5 と文 6 を実行する．さもなければ，

文 7 と文 8 を実行する.」となる.

この構文のフローチャートを，図 8 に示す．このフローチャートを見てわかるように，最初に真となった制御式に続くブロック内が実行されのみである．それ以降，真になっても，そのブロックは実行されない．どの制御式も真にならない場合，最後の else のブロックが実行される．即ち，実行されるブロックは 1 個のみである．else if の段数をいくらでも増やせることは，言うまでもない．

また，else が無い構文も許される．この場合，真となる制御式がない場合，どのブロックも実行されず，この構文から抜ける．

つぎのプログラムが，この構文の使用例である．

```
if(a < 0){
  printf("a は，0 以下\n");
}else if (0 <= a && a < 1){
  printf("a は，0 以上\n");
  printf("かつ\n");
  printf("a は，1 未満\n");
}else if (1 <= a && a < 10){
  printf("a は，1 以上\n");
  printf("かつ\n");
  printf("a は，10 未満\n");
}else{
  printf("a は，10 以上\n");
}
```

- もし，a が 0 未満ならば，
  - 「a は，0 以下」と表示する．
- さもなければ，もし，a が 0 以上，かつ，1 未満ならば
  - 「a は，0 以上」と表示する．
  - 「かつ」と表示する．
  - 「a は，1 未満」と表示する．
- さもなければ，もし，a が 1 以上，かつ，10 未満ならば
  - 「a は，1 以上」と表示する．
  - 「かつ」と表示する．
  - 「a は，10 未満」と表示する．
- さもなければ
  - 「a は，10 以上」と表示する．

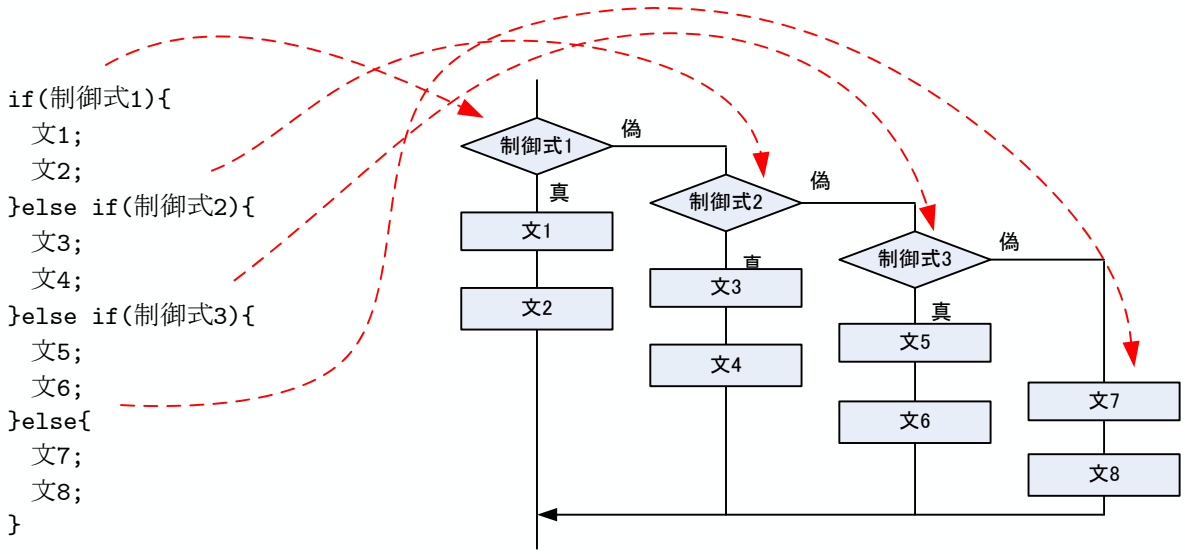

図 8: if else if else を使っての多段の選択

4.2 switch 文 (p.147)

if 文は，選択肢が少ない場合，わかりやすい記述ができる．しかし，選択肢が多くなると，記述は複雑になり，分かりにくいプログラムとなる．そのような場合は，if 文の代わりに switch 文を使うことができる．

この構文のフローチャートを，図 9 に示す．これは，式の値により，それにマッチしたブロック[1]が実行される．もし，どれもマッチしなければ，default が実行される．default 文は無くてもよいが，その場合はどのブロックも実行されない場合がある．

文の集まりのブロックの最後には，bread 文を書く．この break 文が無いと，マッチしたブロック以降の他のブロックも実行される．コードブロックを表す中括弧 { } が無いので，こうすることになっている．この break 文で switch 文の終わりを示す中括弧 (}) から

式や定数式の値の型は，int または char でなくてはならない．定数式の方は，コンパイル時に，評価できなくてはならない．

case の後の定数式は，ラベルである．ラベルの後は，コロン (:) をつける．文の終わりを示すセミコロン (;) ではない．

つぎのプログラムが，この構文の使用例である．

```
switch(a){
  case 1:
```

[1]コードブロックではないので，中括弧 ({}) がない．

```
    printf("あなたは，1と答えました．\n");
    printf("不正解です．\n");
    break;
  case 2:
    printf("あなたは，2と答えました．\n");
    printf("不正解です．\n");
    break;
  case 5:
    printf("あなたは，5と答えました．\n");
    printf("正解です\n");
    break;
  default:
    printf("質問にまじめに答えろ．\n");
}
```

- aが1ならば，以下を実行する．
  - 「あなたは，1と答えました.」と表示する．
  - 「不正解です.」と表示する．
  - switchの構文から抜ける．
- aが2ならば，以下を実行する．
  - 「あなたは，2と答えました.」と表示する．
  - 「不正解です.」と表示する．
  - switchの構文から抜ける．
- aが5ならば，以下を実行する．
  - 「あなたは，5と答えました.」と表示する．
  - 「正解です.」と表示する．
  - switchの構文から抜ける．
- どれにもマッチしなければ，以下を実行する..
  - 「質問にまじめに答えろ.」と表示する．

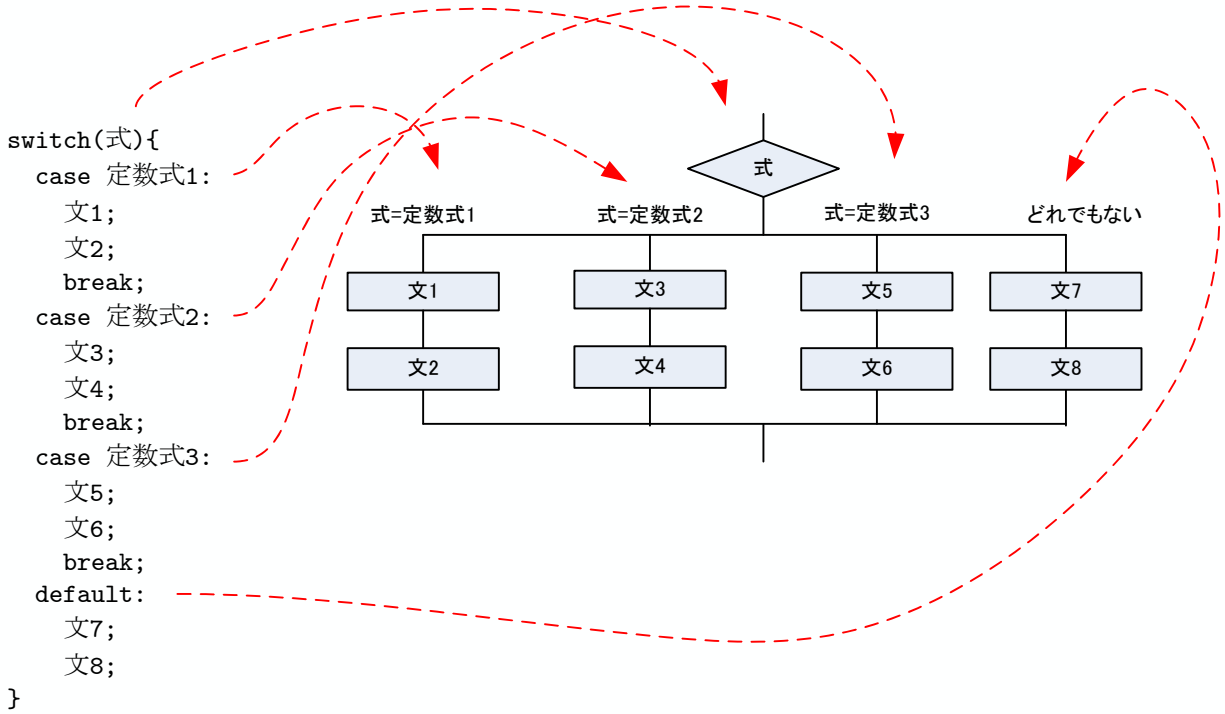

図 9: switch 文を使った構文．多くの選択肢がある場合．

5　基本制御構造 (繰り返し)

5.1　for 文 (p.142)

繰り返しの回数が予め分かっているとき，for 文がつかわれる．「初期値は○○，条件式が正しければ，ループを繰り返し，条件を再設定する．これは，条件式検査 → ループ → 条件の再設定を繰り返す．条件式が誤りになれば，そのループから抜け出す」という構文に使われる．これは，次のように，書く．

書式

```
for(初期値設定式; 継続条件式; 再設定式){
    文 1;
    文 2;
    文 3;
}
```

これは,「継続条件が正しい限り，文 1 と文 2，文 3 を実行する」となる．もし，制御式が誤り (偽) であれば，これら文は実行されず，ブロックの外側に出る．図 10 にこの構文のフローチャートを示す．

以下のようなプログラムが，この構文の使用例である．

```
for(a=1; a<=1000; a++){
   sum=sum+a;
   printf("a=%d sum=%d\n",a,sum);
}
```

この構文の実行直前まで，sum=0 ならば，1〜1000 までの和を計算することができる．見慣れない a++は，a=a+1 と同じで，a の値を 1 増加させている．これをインクリメントと言う．

構文の内容は，次の通りである．

- 初期値として,「a=1」と設定する．
- もし，a が 1000 以下ならば，
  - 「sum+a を計算し，その結果を sum に代入」を実行する．
  - 「a=値　sum=値」と表示する．
  - 「a の値を+1 増加」を実行する．
- 一つ前の，アイテム「もし，a が 1000 以下ならば ···」に戻る．

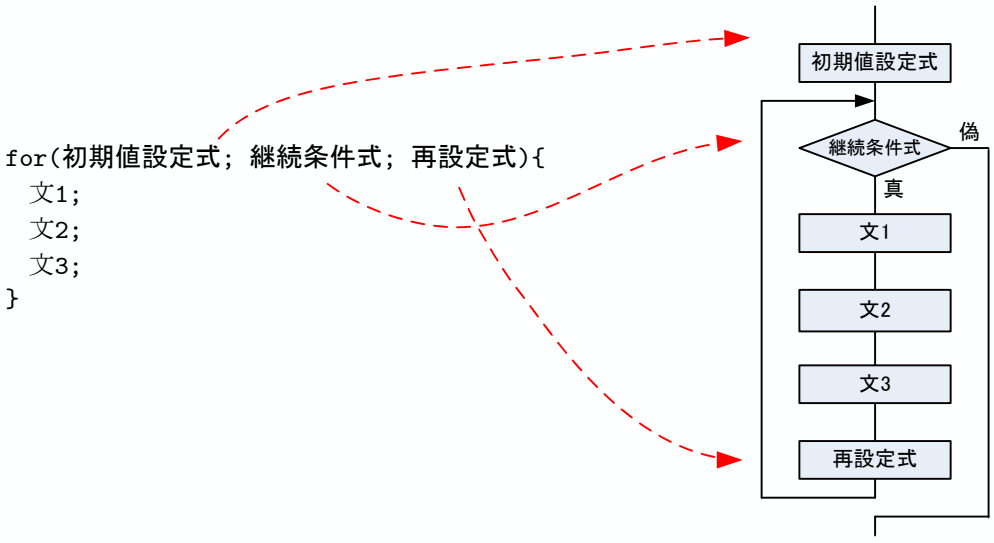

図 10: for の前判定繰り返し文

[**練習 1**] 次の動作をするプログラムを作成せよ．最後の c の値はどのような値になるか?

1. 実数 a, b の初期値をそれぞれ，1.0 と 2.0 とする．
2. 実数 c を $c = (a + b)/2.0$ とする．
3. もし，c^2 が 2 より小さければ，c の値を a に代入する．反対に 2 よりも大きければ，c の値を b に代入する．
4. 操作 2〜3 を 100 回繰り返す.

[**練習 2**] 以下の級数展開式を用いてネピア数 (e) を計算せよ．少し難しいので，余裕のある者のみ実施せよ．

$$\begin{aligned} \mathrm{e} &= 1 + 1 + \frac{1}{2} + \frac{1}{3 \times 2 \times 1} + \frac{1}{4 \times 3 \times 2 \times 1} + \cdots \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} \end{aligned} \tag{1}$$

5.2 不定回数繰り返し

繰り返し回数が予め分かっていない場合は，`do while` 文か `while` 文を用いる．それぞれの違いは，以下の通りである．

- `while` 文は，ループ入り口で継続条件の判断を行う.
- `do-while` 文は，ループ出口で継続条件の判断を行う.

5.2.1 `while` 文 (p.141)

これも，`for` 文同様，前判定繰り返しであるが，予め繰り返し回数が分からないときには，`while` 文が使われることが多い.「条件式が正しければ，ループ繰り返す．条件式が誤りになれば，そのループから抜け出す」という構文に使われる．次のように，書く.

書式

```
while(継続条件式){
   文 1;
   文 2;
   文 3;
}
```

これは,「継続条件が正しい限り，文 1 と文 2，文 3 を実行する」となる．もし，制御式が誤り (偽) であれば，これら文は実行されず，ブロックの外側に出る．図 11 にこの構文のフローチャートを示す.

以下のようなプログラムが，この構文の使用例である.

```
while(a<=1000){
   sum=sum+a;
   printf("a=%d sum=%d\n",a,sum);
```

```
    a++;
}
```

この構文の実行直前まで，sum=0 かつ a=1 ならば，

$$\texttt{sum} = 1 + 2 + 3 + \cdots + 1000$$

を計算する．構文の内容は，次の通りである．

- もし，a が 1000 以下ならば，
  - 「sum+a を計算し，その結果を sum に代入」を実行する．
  - 「a=値　sum=値」と表示する．
  - 「a の値を+1 増加」を実行する．
- 一つ前の，アイテム「もし，a が 1000 以下ならば $\cdots$」に戻る．

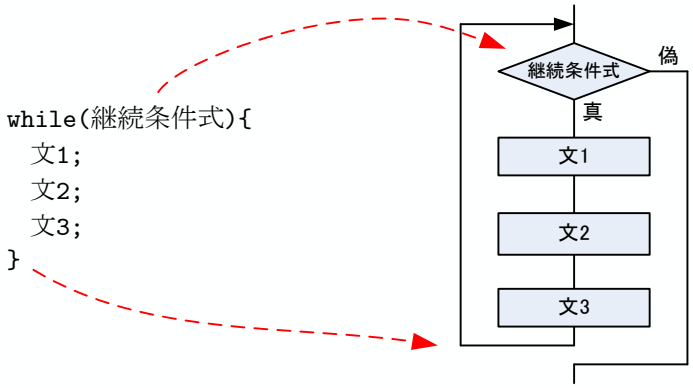

図 11: while の前判定繰り返し文

5.2.2　do while 文 (p.145)

これは後判定繰り返しで，予め繰り返し回数が分からないときに使われることが多い.「ループ内を実行し，継続条件式が正しければ，さらにループを繰り返す．条件式が誤りになれば，そのループから抜け出す」という構文に使われる．次のように，書く．

書式

```
do{
   文 1;
   文 2;
   文 3;
}while(継続条件式);
```

これは,「文 1 と文 2，文 3 を実行し，継続条件が正しければ，これを繰り返す」となる．もし，制御式が誤り (偽) であれば，ブロックの外側に出る．図 12 にこの構文のフローチャートを示す.

以下のようなプログラムが，この構文の使用例である．

```
do{
   sum=sum+a;
   printf("a=%d sum=%d\n",a,sum);
   a++;
}while(a<=1000);
```

この構文の実行直前まで，sum=0 かつ a=1 ならば，

$$\texttt{sum} = 1 + 2 + 3 + \cdots + 1000$$

を計算する．構文の内容は，次の通りである．

- 以下を実行する．
  - 「sum+a を計算し，その結果を sum に代入」を実行する．
  - 「a=値　sum=値」と表示する．
  - 「a の値を+1 増加」を実行する．
- もし，a が 1000 以下ならば，一つ前の，アイテム「以下を実行する」に戻る．

do while 文と whil 文の動作はよく似ているが,「最初から継続条件式が誤り」の場合に違いが生じる．違いは，

do while	最初から条件式が誤りでも，ループブロックを 1 回は実行する．
while	最初から条件式が誤りの場合，ループブロックは実行されない．

である．

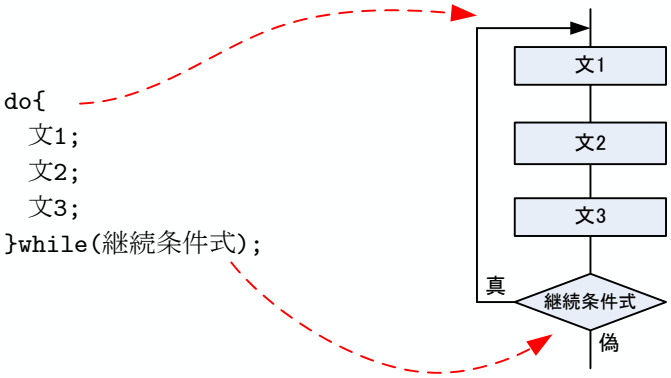

図 12: do while の後判定繰り替え詩文

5.3 ループのスキップと脱出

5.3.1 スキップ (continue)(p.150)

場合によっては，ループブロックの文を実行させたくない場合がある．このとき，continue 文を使う．通常は if 文を伴って，次のように書く．

書式

```
while(継続条件式){
   文 1;
    if(制御式) continue;
   文 2;
   文 3;
}
```

これは,「継続条件が正しい限り，文 1 と文 2，文 3 を実行する．ただし，制御式が正しければ文 2 と文 3 はスキップする.」となる．当然，制御式が誤り (偽) であれば，これら文は実行されず，ブロックの外側に出る．図 13 にこの構文のフローチャートを示す．ここでは，while 文に，continue を用いているが，for や do while 文にも使える．いずれの構文でも，contine 文に出会うと，それ以降のループブロックが実行されない．

以下のようなプログラムが，この構文の使用例である．

```
while(sum<=10000){
   sum=sum+n;
```

```
    n++;
    if(sum <= 9000)contine;
    printf("sum=%d\n",n);
}
```

この構文の実行直前まで，sum=0 かつ n=1 ならば，

$$\mathtt{sum} = 1 + 2 + 3 + \cdots + n$$

を計算する．ただし，この繰り返し文を抜けたときには，sum の値は 10000 を越えている．

構文の内容は，次の通りである．

- もし，sum が 10000 以下ならば，
  - 「sum+n を計算し，その結果を sum に代入」を実行する．
  - 「n の値を+1 増加」を実行する．
  - もし，sum が 9000 以下ならば，ループブロックの最後にスキップする．
  - 「sum=値」と表示する．
- 一つ前の，アイテム「もし，sum が 10000 以下ならば $\cdots$」に戻る．

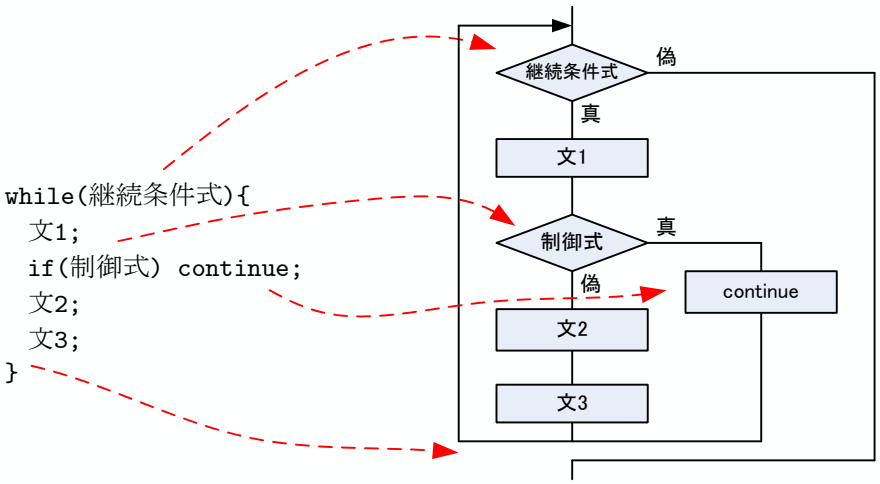

図 13: contine を使って，ループブロックをスキップする構文

5.3.2 脱出 (break)(p.149)

場合によっては，継続条件式が正しくても，構文から抜け出たい場合がある．このとき，break 文を使う．通常は if 文を伴って，次のように書く．

書式

```
while(継続条件式){
   文 1;
    if(制御式) break;
   文 2;
   文 3;
}
```

これは,「継続条件が正しい限り，文 1 と文 2，文 3 を実行する．ただし，制御式が正しければ，この構文から抜ける」となる．当然，制御式が誤り (偽) であれば，これら文は実行されず，ブロックの外側に出る．図 14 にこの構文のフローチャートを示す．ここでも，while 文に，break 文を用いているが，for や do while 文にも使える．いずれの構文でも，break 文に出会うと，構文から抜け出る..

以下のようなプログラムが，この構文の使用例である．

```
while(1){
   sum=sum+n;
   n++;
   if(sum >=  10000)break;
   printf("sum=%d\n",n);
}
```

この構文の実行直前まで，sum=0 かつ n=1 ならば，

$$\texttt{sum} = 1 + 2 + 3 + \cdots + n$$

を計算する．ただし，break により，sum の値が 10000 以上になると，この構文から完全に抜け出す..

構文の内容は，次の通りである．

- 継続条件式はいつも 1(真) なので，以下を実行する．
  - 「sum+n を計算し，その結果を sum に代入」を実行する．
  - 「n の値を+1 増加」を実行する．
  - もし，sum が 10000 以上ならば，構文から抜ける．
  - 「sum=値」と表示する．
- 一つ前の，アイテム「継続条件式はいつも 1(真) なので，$\cdots$」に戻る．

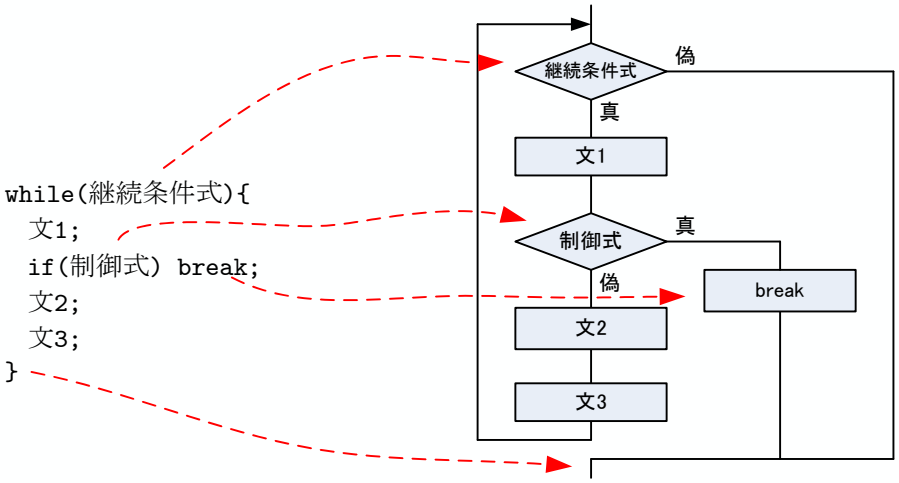

図 14: break 文を用いた構文からの脱出

6　基本制御構造ではない構造

6.1　goto 文とラベル (p.152)

強制的にプログラムの制御を移す．goto 文が示すラベルに実行が移る．if 文と共に用いられることが多い．もちろん，単独で使用することも可能である．

ただし，goto 文はプログラムの流れがわかりにくくなりますので，使わないほうが良いとされている．行儀の良いプログラムを書くためには goto 文は使わないことになっている．ただし，初心者が書くような短いプログラムであれば使っても良いでろう．簡単だし，行儀が悪くてもプログラムを書くことに慣れる方が重要である．上達したら，goto 文を書かないようにすればよい．

ラベル名の後ろには，セミコロンではなく文の前に書いてコロンをつける．

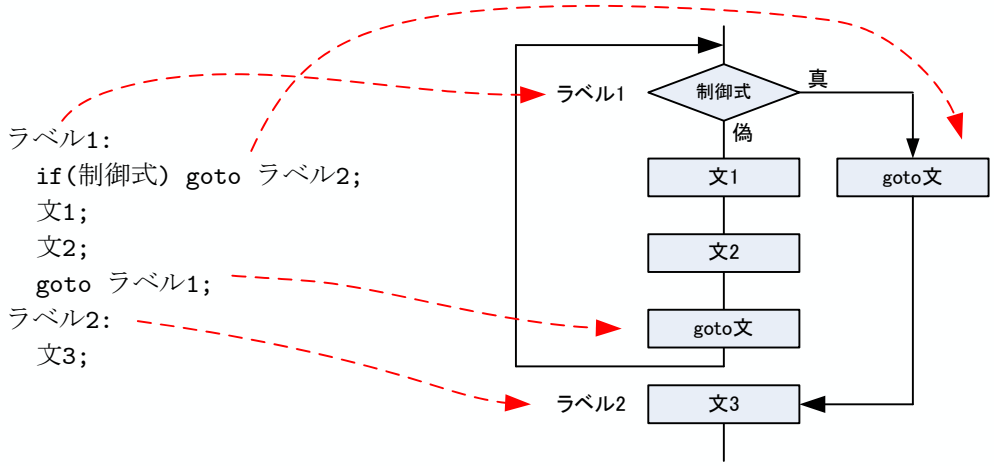

図 15: goto 文とラベルによる制御

7 練習問題

[**練習 1**] 3 種類の繰り返し文 (for,while,do-while) を使って，1〜10000 の整数の和を計算するプログラムを作せよ．

[**練習 2**] キーボードから整数を読み込み，それが素数か否か，判定するプログラムを作成せよ．

- 判定結果は「素数です」，あるいは「素数でない」と表示する．
- 負の値が入力されるまで，連続して判定を行う．
- 変数 test に，キーボードから整数を代入する文は，以下のとおり (p.337)．もちろん，この文に先だって，変数宣言を行う必要がある．

```
scanf("%d%*c",&test);
```

[**練習 2**] 1〜1000000 までの素数を全て，書き出すプログラムを作成せよ．

[**練習 3**] 以下の級数を使って，円周率 π の値を計算せよ．

$$\pi = 16\arctan\left(\frac{1}{5}\right) - 4\arctan\left(\frac{1}{239}\right)$$

$$\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots = \sum_{k=1}^{\infty}(-1)^{k-1}\frac{x^{2k-1}}{2k-1}$$