これまでの復習 (前期末試験に向けて)

山本昌志*

2006 年 9 月 20 日

概 要

前期末試験に向けて，これまで学習した内容をまとめる．このプリントは試験対策用である．

1 前期末試験の傾向と対策

試験の範囲は，以下の通り．

- 第 9 回の講義から本日の第 16 回の講義に配布したプリント
- 教科書は，p.60–152 が範囲となる．2〜3 回位は読み直した方がよい．教科書の範囲で講義で触れなかった部分は試験には出さない．
- このプリントの内容を良く理解すること．分からなければ，私を含めた他の人に聞くこと．

2 変数と式

2.1 基本事項

- コンピューターは，データを処理—計算—する機械である．そのためには，データを記憶することが必要となる．コンピューターのメモリーに記憶する．メモリーの一部に名前をつけ，記憶領域として使うことができる．名前をつけた記憶領域の一部を変数といい，その名前が変数名である．
- データの種類によって，記憶する方法や必要なメモリーの大きさが異なる．コンピューターがデータを適切に処理するために，型を指定しなくてはならない．
  - 1 文字を取り扱う場合は，文字型の `char` を使う．
  - 整数を取り扱う場合は，整数型の `int` を使う．
  - 実数を取り扱う場合は，倍精度実数型の `double` を使う．
- 変数を使うためには，おまじない `int main(void)` の次の中括弧 { に引き続いて，その定義を行わなくてはならない．使いたい変数の型と変数名を指定する．

*独立行政法人秋田工業高等専門学校電気工学科

```
int main(void){
  char a, b, hoge;
  int i, j, fuga;
  double x, y, foo;

        ⋮
```

- キーボードから読み込み込んだデータを変数に代入するためには，型と変数を指定しなくてはならない．文字型には%c，整数型には%d，実数型には%lf と型の指定をする．変数名の前に&を忘れてはならない．

```
scsnf("%c",&hoge);
scanf("%d",&fuga);
scanf("%lf",&foo);
```

- 変数に格納されている値を表示するときも，型を指定しなくてはならない．文字型には%c，整数型には%d をつかう．実数型の場合，%f や%lf，%e が使える．

```
printf("%c",hoge);
printf("%d",fuga);
printf("%f",foo);
```

- 文字型の変数に，文字を代入する場合，シングルクォーテーションで囲む．

```
hoge='A';
```

- コンピューターの内部では，文字は整数のデータとなっている．整数と文字との対応が決まっているので，整数として表すことができる．その対応が書かれたものがアスキー (ASCII) コード表である．例えば，文字 A はコンピューター内部では整数の 65 として表せる．
- 変数には格納できる値の範囲がある．情報量が有限のため，無限のデータの格納は不可能である．

表 1: 変数の型と情報量．ビット数が情報量を表している．

型名	データ型	ビット数	範囲
文字型	char	8	-128〜127
整数型	int	32	-2147483648〜2147483647
実数型	double	64	正負とも約 10^{-308}〜約 10^{+308} 精度約 15 桁

2.2 変数の表示と定数

- 浮動小数[1]は，以下のように表すことができる．

[1]整数でない実数と考えてよい．

```
x=0.00000023456;
y=2.3456e-7;
```

いずれも同じ値，数学では 2.3456×10^{-7} を表している．この例の e-7 は，10^{-7} を表している．非常に大きな数や小さな数を表す場合，ゼロを書かなくて済む分，簡単だし間違いが少なくなる．このような表しかたを指数形式と言う．

- 浮動小数の様々な表示方法について，学習した．例えば，mu=1.25663706144e-6 とする．printf では次のように表示される．

```
printf("%f\n",mu);              ⇒       0.000001
printf("%0.10f\n",mu);          ⇒       0.0000012566
printf("%20.10f\n",mu);         ⇒                0.0000012566
printf("%e\n",mu);              ⇒       1.256637e-06
printf("%0.10e\n",mu);          ⇒       1.2566370614e-06
printf("%20.10e\n",mu);         ⇒           1.2566370614e-06
```

  - %f とすると小数形式で値を表示する．%e とすると指数形式で値を表示する．いずれの場合も小数点以下 6 桁である．
  - %0.nf あるいは%0.ne と表示した場合, それぞれの形式で小数点以下 n 桁で表示する．
  - %m.nf あるいは%m.ne と表示した場合，全体で少なくとも m カラム用意して，小数点以下 n 桁で，それぞれの形式で表示する．カラムと言うのは文字が書ける枠のこと．

- コンピュータープログラムではデータは変数の中に格納される．変数を使うためには，型と変数名を指定—変数の定義—を行わなくてはならない．定義を行うとデータを格納することができるようになる．ただし，変数を定義しただけでは，その中に格納されている値は不定—とんでもない値—である．次のようにすれば，変数の定義と初期化—定義と同時に値の格納—ができる．

```
int hoge=123, fuga=456;
double foo=3.1415, bar=1.6e-19;
```

- 型修飾子 const を付けることにより，定数のように扱うことができる．例えば，次のように変数の定義を行う．

```
const int hoge=123;
const double fuga=9.8765e+43;
```

このように変数を定義すると，プログラム中で変数 hoge や fuga に値の代入ができなくなる．コンパイラーがエラーメッセージを出す．

2.3 式と型

- 数学と同じように C 言語でも，積や商は和や差の演算よりも優先される．C 言語でも数学と同様，括弧によって演算の順序を変えることができる．ただし，C 言語の式の演算で使えるのは小括弧 ( ) のみで，中括弧 { } や大括弧 [] は使えない．

- 数学のイコール (=) は左辺と右辺が等しい—ということを表している．それに対して，C 言語のイコールは，(1) 右辺の式を計算して，(2) 左辺の変数に代入する—というコンピューターの動作を表す．このように変数に値を代入するものを代入演算子と言う．とくに，イコール (=) は単純代入演算子と呼ばれる．

- C 言語では，イコールの他に様々な代入演算子が用意されている．表 2 に示すようなものがあり，複合代入演算子と呼ばれる．

表 2: 代表的な複合代入演算子．表中の a の値は，演算の結果である．演算の前の a の値は 7，hoge は 3 とする．いずれも整数型の変数とする．

複合代入演算子	動作	例	単純代入演算子の記述	a の値
+=	加算して代入	a+=hoge	a=a+hoge	10
-=	減算して代入	a-=hoge	a=a-hoge	4
*=	積算して代入	a*=hoge	a=a*hoge	21
/=	除算して代入	a/=hoge	a=a/hoge	2
%=	剰余算して代入	a%=hoge	a=a%hoge	1

- 代入演算子の右辺の式の計算結果を右辺値，左辺の変数の値は左辺値と言う．

- 整数型 (int) と倍精度実数型 (double) の両方が含まれる二項演算では，計算精度の高い方に自動的に型が変換される．

- また，代入演算子の左辺と右辺の型が異なる場合，右辺値は左辺の型に自動的に変換される．

- 変数に格納されている型を強制的に変換したい場合，キャストによる明示的な型変換—強制型変換—を使う．括弧を付けて型名を指定すると，その右の変数や式の値が，指定した型に変換できる．例えば，整数 (hoge=10) を整数 fuga=3 で割って，倍精度実数の値にしたい場合は，

```
a=(double)hoge/fuga;
b=hoge/(double)fuga;
c=(double)hoge/double(fuga);
```

のようにする．倍精度実数型の変数 a,b,c には，いずれも 3.33333$\cdots$ が代入される．

3 制御の流れ

3.1 制御文 (if−else)

- 正しいあるいは誤りを整数の 0 と 1 で表す．正しい場合が整数の 1 で，誤りの場合が整数の 0 である．コンピューターは，なんでもかんでも数字で表すのである．C 言語では特別に，0 を誤り，それ以外—大体の場合 1— を正しいとして取り扱う．

- 関係演算子は大小関係や等しいか否かについて，計算を行う．これは四則演算子 (+,-,*,/) と同じ演算子なので，計算結果がある．それは，正しい場合 (真) に 1，誤り (偽) の場合 0 となる．

  – 大小関係を表す演算子は 4 つ (<, <=, >, >=) ある．それぞれ，数学の記号の $<$，$\leqq$，$>$，$\geqq$ に対応する．キーボードに $\leqq$ の記号がないので，<=で代用しているのである．ただし，=<と書いてはならない．

    * 例えば，演算 3>5 の結果は 0 となる．なぜならば，この式が示している大小関係は誤りだからである．
    * 一方，演算 3<5 の結果は 1 となる．なぜならば，この式が示している大小関係は正しいからである．

  – 大小関係を示す関係演算子とともに，等しいか否かを表す演算子も重要な役割を果たす．それには，等しいことを表す==と等しくないことを表す!=がある．これらは，それぞれ，数学記号の $=$ と $\neq$ に対応する．イコールひとつだと代入演算子になるので，イコールをふたつ続けて等しいことを表す演算子として使用している．$\neq$ はキーボードに記号がないので，!=を使っている．

    * 例えば，演算 3==5 の結果は 0 となる．なぜならば，この式が示している等価関係は誤りであるからである．
    * 一方，演算 3!=5 の結果は 1 となる．なぜならば，この式が示している等価関係は正しいからである．

- 論理演算子は論理が正しいか否かについて，演算を行う．演算結果は，論理が正しければ 1，誤りであれば 0 となる．論理演算子には，論理和 || と論理積 &&，論理否定 ! の 3 つがある．

  – 論理和は，日本語では「または」英語では「or」と表現される．論理和演算子をはさむ 2 つの論理のどちらか一方が正しい，あるいは両方が正しい場合，演算結果が 1 となる．両方の演算が誤りの場合のみ 0 となる．

    * 例えば，演算 9<7 || 5<3 の結果は 0 となる．なぜならば，演算子||の両側の式は誤りであるからである．
    * 次に，演算 9<7 || 3<5 の結果は 1 となる．なぜならば，論理和演算子をはさむ片方の式が正しいからである．
    * 言うまでもないが，演算 7<9 || 3<5 のように，両方の式が正しい場合も，演算の結果は 1 となる．

– 論理積は，日本語では「かつ」英語では「AND」と表現される．論理積演算子をはさむ 2 つの論理の両方の演算が正しい場合のみ 1 となる．どちらか一方が誤り，あるいは両方が誤りの場合，演算結果が 0 となる．

  * 例えば，演算 `3<5 && 7<9` の結果は 1 となる．なぜならば，演算子||の両側の 2 つの式は正しいからである．
  * 次に，演算 `3<5 || 9<7` の結果は 0 となる．なぜならば，論理和演算子をはさむ片方の式が誤りだからである．
  * 言うまでもないが，演算 `5<3 || 9<7` のように，両方の式が誤り場合も，演算の結果は 0 となる．

– 論理否定は，論理を反転させる．英語では,「NOT」と表現される．

  * 例えば，演算 `!(3<5)` の結果は 0 となる．なぜならば， `3<5` の演算の結果は 1，そして否定演算子 `!` でそれを反転させているので，`!(3<5)` の演算結果は 0 となる．
  * 一方，演算 `!(5<3)` の結果は 1 となる．なぜならば， `5<3` の演算の結果は 0，そして否定演算子 `!` でそれを反転させているので，`!(5<3)` の演算結果は 1 となる．

- これまで，学習してきた演算子には，算術演算子 (`+,-,*,/,%`) と関係演算子 (`<, <=, >, >=`)，論理演算子 (`||,&&,!`) がある．これらの演算子 (オペレーター) と被演算子 (オペランド)[2]を組み合わせて式ができあがる．数学同様，演算子には優先順位があり，先に計算する演算子が決まっている．表 3 の上の演算子から計算を行う．

表 3: 演算子の優先順位 (上のほうが優先順位が高い)

種類	演算子
括弧	`()`
論理否定	`!`
乗除	`*` `/`
加減	`+` `-`
比較	`<` `<=` `>` `>=`
等価	`==` `!=`
論理積	`&&`
論理和	`||`

- 「もし○○ならば，△△する」という構文—とくに△△の部分が 1 つの文—は次のように書く．

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

- 「もし○○ならば，△△し，□□し，···」のように複数の文を実行する場合は，次のように書く．

[2]普通，値や変数がオペランドになる．

```
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");
}
```

実行させたい複数の文は，括弧 { } でくくり，ブロック化するのがこつである．教科書ではこれを複文と言っている．

- 「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は，次のように書く．

```
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");
}
```

3.2 制御文 (switch と if−else if−else)

- プログラム中で,「もし a=1 ならば○○する，a=2 ならば△△する，a=5 ならば□□する，さもなければ◎◎する」というような処理をしたい場合，switch という命令を使う．このように，値により処理が複数に分岐することを多分岐と言う．条件の部分が整数 (int)，あるいは文字 (char) で表される場合，switch を使う．この構文のフローチャートと書式を図 1 に示す．

- 「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，次々と条件を変化させる．このような構文には，if〜else if〜else 文を使う．この構文のフローチャートと書式を図 2 に示す．

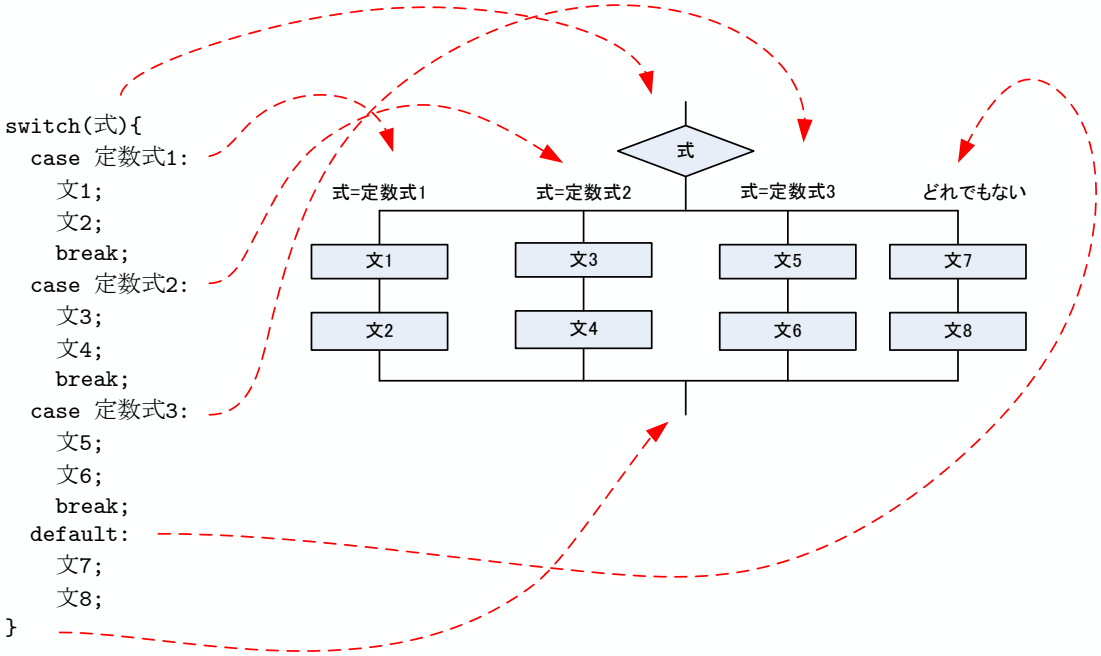

図 1: switch 文を使った構文．多くの選択肢がある場合．

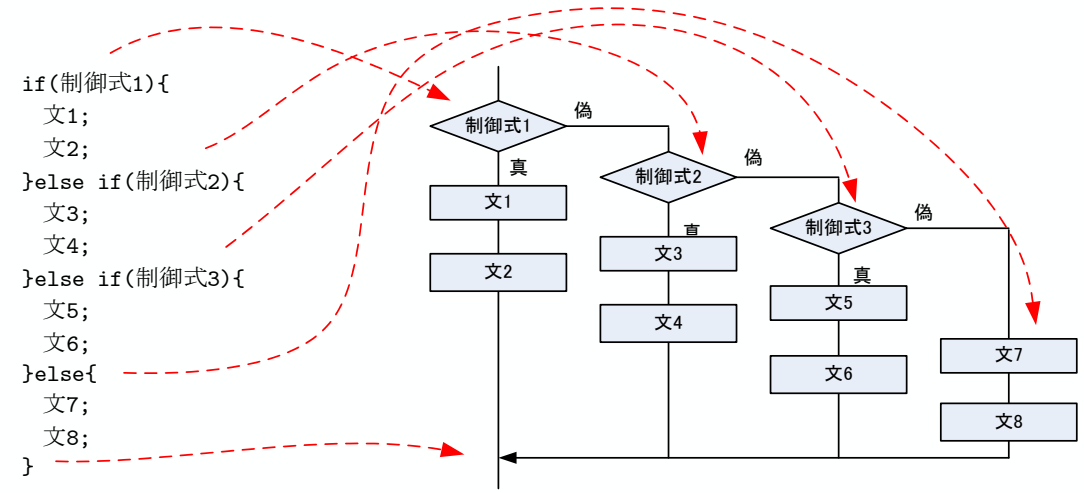

図 2: if〜else if〜else を使った多分岐

3.3 ループ処理 (while と for，do--while)

何回も実行する繰り返し文 (ループ文) についての学習を行った．繰り返し文に使われる while と for，do–while の使い方を理解しなくてはならない．それぞれの文のプログラムの書き方を図 3〜5 に示す．また，表 4 にそれぞれの構文の特徴を示す．表に示したよく使う場面と言うのは大まかな目安である．どれでも同じようにプログラムは可能である．

表 4: それぞれの繰り返し文の特徴.

ループ	判断	よく使う場面
while	ループ処理の前	ループ回数不明，処理の後に判断するとき
for	ループ処理の前	予めループ回数が分かっているとき
do–while	ループ処理の後	判断の前にとりあえず実行するとき

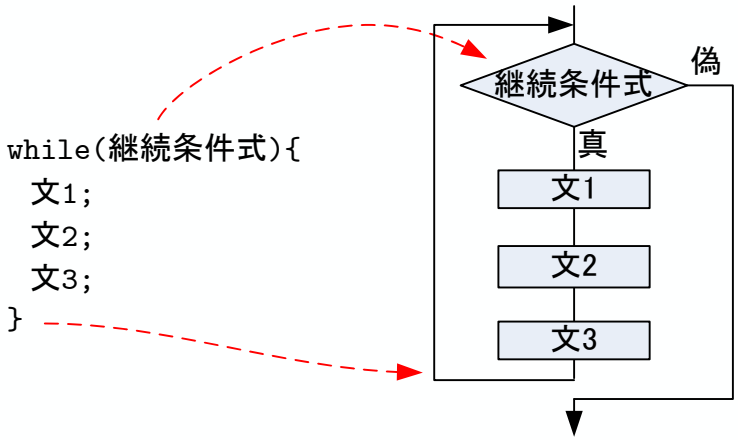

図 3: while を使ったループ処理の方法

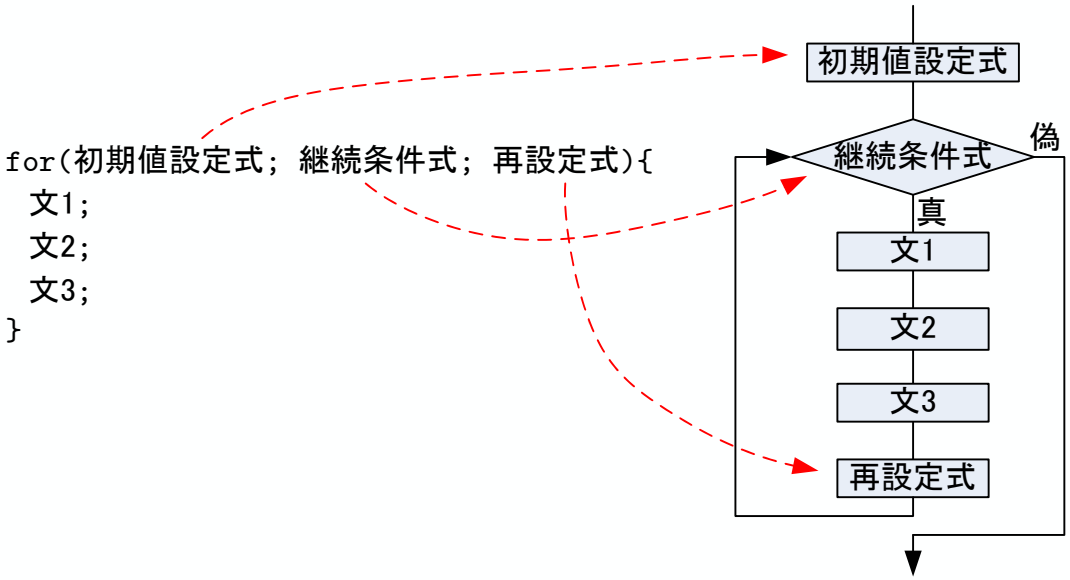

図 4: for を使ったループ処理の方法

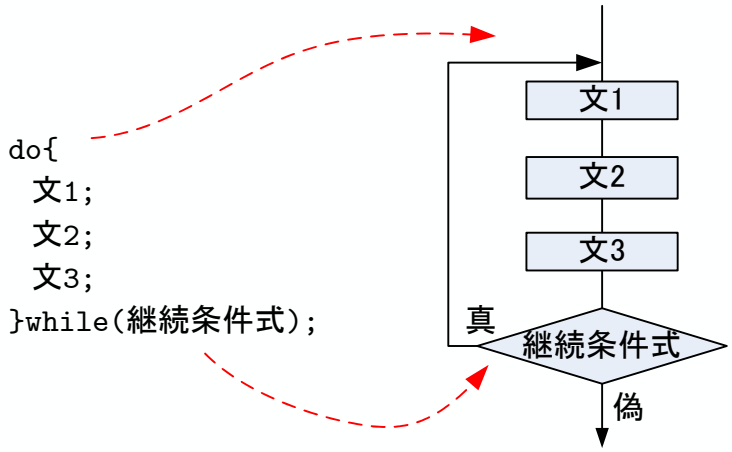

図 5: do while の後判定繰り返し文の方法

4　プログラム例

4.1　1〜1000 までの和の計算プログラム

3 つの繰り返し文を使った 1〜1000 までの和を計算するプログラムを，リスト 1–??に示す.

リスト 1: while 文を使った 1〜1000 の加算プログラム

```
#include <stdio.h>

int main(void){
  int i, sum;

  sum=0;
  i=1;

  while(i<=1000){
    sum += i;
    i++;
  }

  printf("1から1000までの和は，%dです. \n", sum);

  return 0;
}
```

リスト 2: for 文を使った 1〜1000 の加算プログラム

```
#include <stdio.h>

int main(void){
  int i, sum;

  sum=0;

  for(i=1; i<=1000; i++){
    sum += i;
  }

  printf("1から1000までの和は，%dです. \n", sum);

  return 0;
}
```

リスト 3: do–while 文を使った 1〜1000 の加算プログラム

```
#include <stdio.h>

int main(void){
  int i, sum;

  sum=0;
  i=1;

  do{
    sum += i;
    i++;
  }while(i<=1000);
```

```
13
14    printf("1から1000までの和は，%dです．\n", sum);
15
16    return 0;
17  }
```

4.2 最大値を求めるプログラム

リスト 4 は，関数

$$f(x) = -5x^2 + 6x + 6\sin x \qquad -1000 \leqq x \leqq 1000 \tag{1}$$

の最大値を計算するプログラムである．計算のステップ幅—計算精度を表す— は，$dx = 0.001$ としている．すなわち，

$$x = -1000.0000, -1000.0001, -1000.0002, \quad \cdots \quad , 999.9998, 999.9999, 1000.0000 \tag{2}$$

と x の値を変化させて，関数 $f(x)$ の値を計算している．

リスト 4: 関数の最大値を検索するプログラム

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main(void){
5    double xmin, xmax, x, dx, fx;
6    double max_fx, max_x;              // 最大を格納する変数
7    int i, ncal;
8
9    xmin = -1000.0;                    // xの最小値
10   xmax =  1000.0;                    // xの最大値
11   dx = 0.0001;                       // xの計算のきざみ幅（誤差の程度）
12   ncal = (xmax-xmin)/dx;             // 計算回数
13
14   //--- 暫定最大（x=xmin を暫定最大とする）---------
15   x = xmin;
16   max_x = x;
17   max_fx = -5.0*x*x + 6.0*x + 6*sin(x);
18
19   for(i=1; i<=ncal; i++){
20     x = xmin + i*dx;                          // xの計算
21     fx = -5.0*x*x + 6.0*x + 6*sin(x);         // f(x)の計算
22
23     //---- 最大値か否かの検査 --------------
24     if(max_fx < fx){                          // 新たに最大値発見
25       max_fx = fx;
26       max_x = x;
27     }
28
29   }
30
31   printf("x = %fのときf(x)=%fで最大です\n", max_x, max_fx);
32
33   return 0;
34 }
```

4.3 連立不等式

リスト 5 は，基礎数学の教科書 p.62 の練習問題 2-A の 2.(2) の連立不等式

$$\begin{cases} x^2 - 2x - 3 \leqq 0 \\ \qquad 3x + 2 < 4x \end{cases} \qquad -1000 \leqq x \leqq 1000 \tag{3}$$

を計算するプログラムである．計算のステップ幅—計算精度を表す— は，$dx = 0.001$ としている．

リスト 5: 連立不等式を計算するプログラム

```
#include <stdio.h>

int main(void){
  double xmin, xmax, x, dx;
  int period , current;
  int i, ncal;

  xmin = -1000.0;                    // xの最小値
  xmax =  1000.0;                    // xの最大値
  dx = 0.0001;                       // xの計算のきざみ幅（誤差の程度）
  ncal = (xmax-xmin)/dx;             // 計算回数

  printf("連立不等式が成立するのは以下の範囲です\n");

  period = 0;

  for(i=1; i<=ncal; i++){
    x = xmin + i*dx;                 // 検査する x

    // --- 連立不等式が OK or NG の検査 ----
    if(x*x-2*x-3 <= 0 && 3*x+2 < 4*x){
      current = 1;                   // OKの場合
    }else{
      current = 0;                   // NGの場合
    }

    // --- OK と NG の境界の処理 ----
    if(current != period){
      if(current == 1){
        printf("%f\tから\t", x);
        period = current;
      }else{
        printf("%f\n", x);
        period = current;
      }
    }
  }

  return 0;
}
```