アルゴリズムとデータ構造 円周率課題

所属: 長野工業高等専門学校工学科 3 年
情報エレクトロニクス系情報コース
学籍番号:22120
名前: 塚田 勇人

2024 年 2 月 3 日

1　目的

多倍長ライブラリを自作し，円周率を計算することでアルゴリズムとデータ構造の理解を深めることを本課題の目的とする．

2　原理

本課題を実施するにあたって，必要な原理を説明する．

2.1　多倍長

多倍長は，プログラミングで用いられる 8bit や 16bit などの固定長の整数型では表現できない桁数の整数を扱うために用いられる可変長のデータ型である．メモリの許す限り桁数を増やすことができるため，任意の桁数の整数を扱うことができる．このように，多倍長では桁数を変えて扱うことができるため，任意精度演算とも呼ばれている．多倍長を用いて扱っている数値のことを**多倍長整数**，多倍長を用いてする計算のことを**多倍長演算**と呼ぶ．本課題では多倍長整数で扱える桁数をあらかじめ決めて，実装する．

2.2　円周率

円周率は，円の周の長さと円の直径の比率である．円周率は，π で表される．円周率は，無限に続く無理数であり，その値は 3.1415926535... となる．今までに，円周率を求めるために様々な数学的手法が提案されている．本課題では式 (1) を用いて円周率を求める．

$$\pi = 6\sqrt{3}\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)3^{n+1}} \tag{1}$$

式 (1) は，$\tan^{-1} x$ のテイラー展開を用いることで，導出される．詳細な導出方法は章末の付録の節 7.1 の式 1 の導出 に記載する．

2.3　ニュートンラフソン法

ニュートンラフソン法では，関数 $f(x) = 0$ の解を求めるために用いられる．具体的には，関数 $f(x)$ の導関数 $f'(x)$ を用いて，式 (2) の漸化式で解を求める．

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2}$$

本課題ではニュートンラフソン法を用いて，逆数を求める．a の逆数を求めるために式 (3) の関数をニュートンラフソン法を用いて解くことで，逆数を求める．

$$f(x) = \frac{1}{x} - a \tag{3}$$

これをニュートンラフソン法に適用すると，式 (4) のようになる．

$$x_{n+1} = x_n \left(2 - a x_n + (a x_n - 1)^2\right) \tag{4}$$

この式の詳細な導出方法は章末の付録の節 7.2 の式 4 の導出 に記載する．

3 実験環境

本課題を実施するにあたって，使用した実行環境を表 1 に示す．

表 1: 実験環境

OS(オペレーションシステム)	Windows 11 Pro
CPU(中央処理装置)	Ryzen5 5600
メモリ	24GB
開発環境	Ubuntu 22.04 on Windows 11 Pro
コンパイラ	GCC(GNU Compiloer Collection) version 11.4.0

4 プログラムの設計と説明

本課題で作成したプログラムに実装した関数や定数について説明する．なお，本課題で作成したプログラムでは，各定数や関数のプロトタイプ宣言を *pi.h* に記述し，各関数の内容を *pi.c* に記述する．ファイル全体は章末の付録の節 7.3 の pi.h と節 7.4 の pi.c に記載する．

4.1 定数

定数はソースコードを見たときに意味のわからない数値が直接記述されることを防ぐために定義するものである．また，定数を使うことで，プログラムの修正が容易になる．定数は大文字で記述することが一般的である．本課題で使用した定数を表 2 に示す．

表 2: 定数の定義

定数名	意味
DIGIT	求めたい円周率の桁数
RADIX	多倍長整数の基数
RADIX_LEN	多倍長整数の各桁に格納できる桁数
MARGIN	円周率を求めるためにとる余裕の桁数
KETA	多倍長整数の桁数
PLUS	正の値を表す定数
ZERO	0 を表す定数
MINUS	負の値を表す定数
TRUE	真を表す定数
FALSE	偽を表す定数
RADIX_T	多倍長整数を定義するために使う型

これらの定数の定義の記述を `pi.h` から抜粋してリスト 1 に示す.

リスト 1: 定数の定義 (`pi.h` から一部抜粋)

```
1  #define DIGIT 10000
2
3  #define RADIX 1000000000
4  #define RADIX_LEN 9
5  #define MARGIN 100
6
7  #define KETA (DIGIT + MARGIN) * 4 / RADIX_LEN + 1
8
9  #define PLUS 1
10 #define ZERO 0
11 #define MINUS -1
12
13 #define TRUE 1
14 #define FALSE 0
15
16 #define RADIX_T long long int
```

4.2 多倍長整数の実装

本課題では構造体を用いて多倍長整数を実装する．構造体では，多倍長整数の各桁の値を格納する `n` と符号を格納する `sign` の 2 つの変数を持つ．配列 `n` の値は `RADIX_T` 型で定義し，多倍長整

数の桁数は KETA で定義する．また各要素には RADIX を基数として RADIX_LEN 桁の値を格納する．

つまりこの多倍長整数は $RADIX\_LEN \times KETA$ **桁の整数**を格納することができる．例えば基数を 100，桁数を 10 とすると，この多倍長整数は 20 桁の整数を表すことができる．基数が 100 のときに 123456789 を多倍長整数に格納する様子を模式的に表した図 1 に示す．

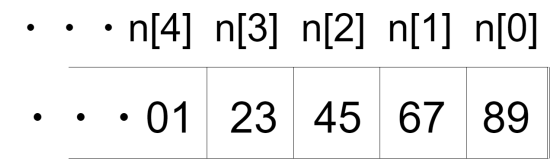

図 1: 多倍長整数の構造

変数 *sign* は多倍長整数の符号を表す．定義の PLUS．ZERO，MINUS はそれぞれ正，0，負を表しており，*sign* にはこれらのいずれかが格納される．

多倍長整数の実装を pi.h から抜粋してリスト 2 に示す．

リスト 2: 多倍長整数の実装

```
1  typedef struct NUMBER {
2      RADIX_T n[KETA]; // 各桁の変数
3      int sign; // 符号変数-1: 負，0: 0，1: 正
4  } Number;
```

4.3 関数

本課題で実装した関数を表 3 に示す．これらの関数について各項で説明する．

表 3: 関数一覧

関数名	機能
setSign	符号を設定する
getSign	符号を取得する
clearByZero	多倍長整数を初期化する
dispNumber	多倍長整数を表示する
copyNumber	値をコピーする
getAbs	絶対値を求める
isZero	0 かどうかを判定する
mulBy10SomeTimes	何回か 10 倍する
divBy10SomeTimes	何回か 10 で割る
setInt	int 型の値を多倍長整数に代入する
getInt	多倍長整数を int 型に変換する
numComp	2 つの多倍長整数を比較する
numCompWithInt	多倍長整数と int 型の値を比較する
add	2 つの多倍長整数を加算する
sub	2 つの多倍長整数を減算する
multiple	2 つの多倍長整数を乗算する
inverse3	多倍長整数の逆数を求める
divideByInverse	2 つの多倍長整数を除算する
sqrtThree	$\sqrt{3}$ を求める
getLen	多倍長整数の桁数を取得する
comparePi	多倍長整数が円周率と等しいかどうか比較する
compareRootThree	多倍長整数が $\sqrt{3}$ と等しいかどうか比較する

4.3.1 setSign 関数

`setSign` 関数は表 4 のように宣言されている.

表 4: `setSign` 関数

関数名	`setSign`
概要	符号を設定する
引数 1	`Number *a`: 符号を設定する多倍長整数
引数 2	`int s`: 設定する符号
戻り値	`int`: 成功した場合は `TRUE`, 失敗した場合は `FALSE`

setSign 関数は引数 s の値を多倍長整数 a の符号に設定する関数である．それを次の手順でプログラム上で実装する．

1. 引数 s が $PLUS$,$ZERO$,$MINUS$ のいずれかの場合，それぞれ引数 a の符号を $PLUS$,$ZERO$,$MINUS$ に設定し，$TRUE$ を返す．
2. 引数 s が $PLUS$,$ZERO$,$MINUS$ のいずれでもない場合，$FALSE$ を返す．

setSign 関数のソースコードをリスト 3 に示す．

リスト 3: setSign 関数

```
1  int setSign(Number *a, int s) {
2    switch (s) {
3        case PLUS:
4            a->sign = PLUS;
5            break;
6        case ZERO:
7            a->sign = ZERO;
8            break;
9        case MINUS:
10           a->sign = MINUS;
11           break;
12       default:
13           return FALSE;
14   }
15   return TRUE;
16 }
```

4.3.2 getSign

getSign 関数は表 5 のように宣言されている．

表 5: getSign 関数

関数名	getSign
概要	符号を取得する
引数	const Number *a: 符号を取得する多倍長整数
戻り値	int: 符号

const とは，引数の値を変更しないことを明示するための修飾子である．関数内で上書き処理を行わない関数では引数に *const* をつけることで，引数の値を変更しないことを明示する．これはプログラムの保守性を高めるための記述方法であり，以降の関数にも同様の記述がある．

getSign 関数は次の手順でプログラム上で多倍長整数の符号を戻り値として返すように実装する．

getSign 関数のソースコードをリスト 4 に示す．

リスト 4: getSign 関数

```
1  int getSign(const Number *a) { return a->sign; }
```

4.3.3 clearByZero

clearByZero 関数は表 6 のように宣言されている．

表 6: clearByZero 関数

関数名	clearByZero
概要	多倍長整数を初期化する
引数	Number *a: 初期化する多倍長整数
戻り値	なし

リスト 5 のように clearByZero 関数を呼び出すと，リスト 6 のように動作する．RADIX は 100, RADIX_LEN は 2 とする．また，以降も関数の動作を説明する際にはこの値を用いる．

リスト 5: clearByZero 関数の呼び出し

```
1  #include <stdio.h>
2  #include "pi.c"
3  int main (void){
4      Number a;
5      clearByZero(&a);
6      printf("a: %d\n", a.n[0]);
7      printf("sign: %d\n", getSign(&a));
8      return 0;
9  }
```

リスト 6: リスト 5 の実行結果

```
1  a: 0
2  sign: 0
```

値が 0 で初期化され，符号も 0 に設定されていることがわかる．

clearByZero 関数は for 文のループを使って多倍長整数を初期化するように実装する．*clearByZero* 関数のソースコードをリスト 7 に示す．

リスト 7: clearByZero 関数

```
1  void clearByZero(Number *a) {
2      for (int i = 0; i < KETA; i++) {
3          a->n[i] = 0;
4      }
5      setSign(a, ZERO);
6  }
```

4.3.4 dispNumber

dispNumber 関数は表 7 のように宣言されている.

表 7: dispNumber 関数

関数名	dispNumber
概要	多倍長整数を表示する
引数	const Number *a: 表示する多倍長整数
戻り値	なし

リスト 8 のように dispNumber 関数を呼び出すと，リスト 9 のように動作する.

リスト 8: dispNumber 関数の呼び出し

```
1  #include "pi.c"
2  int main (void){
3      Number a;
4      clearByZero(&a);
5      dispNumber(&a);
6      a.n[0] = 12;
7      a.n[2] = 34;
8      dispNumber(&a);
9      printf("\n");
10     setSign(&a, MINUS);
11     dispNumber(&a);
12     printf("\n");
13     return 0;
14 }
```

リスト 9: リスト 8 の実行結果

```
1  + 0
2  + 34 00 12
3  - 34 00 12
```

リスト 9 から，dispNumber 関数は +，0，-を表示し，その後ろに多倍長整数の各要素を 1 要素ずつ表示することがわかる．

次の手順で dispNumber 関数を実装する．

1. 多倍長整数の符号を判定し，正ならば +，0 ならば +0，負ならば − を表示する．
2. $a_i(i = \texttt{KETA} - 1, \texttt{KETA} - 2...)$ で 0 でない要素を見つけるまで i を減らす．
3. a_i を表示し，i を減らす．
4. i が 0 になるまで 3 を繰り返す．

この手順をプログラム上では次の手順で実装する．

1. 引数 a の符号を判定し，正ならば +，0 ならば +0，負ならば − を表示する．
2. 0 を付けて表示する書式を設定する．
3. for 文を用いて $i = \texttt{KETA} - 1$ から，a->n[i] が 0 ではない要素を見つけるまで i を減らす
4. for 文を用いて i から 0 まで a->n[i] を表示する．

dispNumber 関数のソースコードをリスト 10 に示す．

リスト 10: dispNumber 関数

```
1  void dispNumber(const Number *a) {
2      int i;
3      char format[8];
4      sprintf(format, " %%0%dd", RADIX_LEN); // format = " %0RADIX_LENd"
5      switch (getSign(a)) {
6          case PLUS:
7              printf("+");
8              break;
9          case ZERO:
10             printf("+ 0");
11             return;
12         case MINUS:
13             printf("-");
14             break;
15     }
16     for (i = KETA - 1; i >= 0; i--) {
17         if (a->n[i] > 0) {
18             break;
19         }
20     }
21     for (; i >= 0; i--) {
22         printf(format, a->n[i]);
23     }
```

```
24  }
```

4.3.5 copyNumber

copyNumber 関数は表 8 のように宣言されている．

表 8: copyNumber 関数

関数名	copyNumber
概要	値をコピーする
引数	Number *a: コピー先, const Number *b: コピー元
戻り値	なし

リスト 11 のように copyNumber 関数を呼び出すと，リスト 12 のように動作する．

リスト 11: copyNumber 関数の呼び出し

```
1   #include "pi.c"
2   int main (void){
3       Number a, b;
4       clearByZero(&a);
5       clearByZero(&b);
6       a.n[0] = 12;
7       a.n[2] = 34;
8       setSign(&a, PLUS);
9       copyNumber(&b, &a);
10      dispNumber(&a);
11      printf("\n");
12      dispNumber(&b);
13      printf("\n");
14      return 0;
15  }
```

リスト 12: リスト 11 の実行結果

```
1   + 34 00 12
2   + 34 00 12
```

リスト 12 の実行結果から，copyNumber 関数は引数 b に引数 a の値をコピーすることがわかる．

copyNumber 関数は関数内で引数 a のアドレスの値を引数 b のアドレスにコピーして実装する．copyNumber 関数のソースコードをリスト 13 に示す．

リスト 13: copyNumber 関数

```
1  void copyNumber(Number *a, const Number *b) { *a = *b; }
```

4.3.6 getAbs

getAbs 関数は表 9 のように宣言されている．

表 9: getAbs 関数

関数名	getAbs
概要	絶対値を求める
引数	const Number *a: 絶対値を求める多倍長整数, Number *b: 絶対値を代入する多倍長整数
戻り値	なし

リスト 14 のように getAbs 関数を呼び出すと，リスト 15 のように動作する．

リスト 14: getAbs 関数の呼び出し

```
1  #include "pi.c"
2  int main (void){
3      Number a, b;
4      clearByZero(&a);
5      clearByZero(&b);
6      a.n[0] = 12;
7      a.n[2] = 34;
8      setSign(&a, MINUS);
9      getAbs(&a, &b);
10     dispNumber(&a);
11     printf("\n");
12     dispNumber(&b);
13     printf("\n");
14     return 0;
15 }
```

リスト 15: リスト 14 の実行結果

```
1  - 34 00 12
2  + 34 00 12
```

リスト 15 の実行結果から，getAbs 関数は引数 a の絶対値を引数 b に代入することがわかる．
getAbs 関数は次の手順で実装する．

1. 多倍長整数 a の値を多倍長整数 b にコピーする．

2. 多倍長整数 a の符号が 0 ならば b の符号を 0 に設定する．
3. 多倍長整数 a の符号が 0 でない場合，b の符号を正に設定する．

この手順をプログラム上では次の手順で実装する．

1. copyNumber 関数を用いて引数 a を b にコピーする．
2. 引数 a の符号が $ZERO$ ならば引数 b の符号を $ZERO$ に設定する．
3. 引数 a の符号が $ZERO$ でない場合，引数 b の符号を正に設定する．

getAbs 関数のソースコードをリスト 16 に示す．

リスト 16: getAbs 関数

```
1  void getAbs(const Number *a, Number *b) {
2    copyNumber(b, a);
3    if (getSign(a) == ZERO) {
4        setSign(b, ZERO);
5    } else {
6        setSign(b, PLUS);
7    }
8  }
```

4.3.7 isZero

isZero 関数は表 10 のように宣言されている．

表 10: isZero 関数

関数名	isZero
概要	0 かどうかを判定する
引数	const Number *a: 判定する多倍長整数
戻り値	int: 0 ならば TRUE，0 でないならば FALSE

リスト 17 のように isZero 関数を呼び出すと，リスト 18 のように動作する．

リスト 17: isZero 関数の呼び出し

```
1  #include "pi.c"
2  int main (void){
3      Number a;
4      clearByZero(&a);
5      a.n[0] = 12;
6      a.n[2] = 34;
7      setSign(&a, PLUS);
```

```
8       printf("%d\n", isZero(&a));
9       clearByZero(&a);
10      printf("%d\n", isZero(&a));
11      return 0;
12  }
```

リスト 18: リスト 17 の実行結果

```
1   0
2   1
```

リスト 18 の実行結果から，isZero 関数は引数 a が 0 でないときは TRUE を，0 のときは FALSE を返することがわかる．

isZero 関数は次の手順で実装する．

1. 多倍長整数の符号が 0 ならば *TRUE* を返す．
2. 多倍長整数の符号が 0 でないならば *FALSE* を返す．

この手順をプログラム上では次の手順で実装する．

1. getSign 関数を用いて引数 a の符号を取得する．
2. 符号が *ZERO* ならば *TRUE* を返す．
3. 符号が *ZERO* でないならば *FALSE* を返す．

isZero 関数のソースコードをリスト 19 に示す．

リスト 19: isZero 関数

```
1   int isZero(const Number *a) {
2     if (getSign(a) == ZERO) {
3         return TRUE;
4     } else {
5         return FALSE;
6     }
7   }
```

4.3.8 mulBy10SomeTimes

mulBy10SomeTimes 関数は表 11 のように宣言されている．

表 11: mulBy10SomeTimes 関数

関数名	mulBy10SomeTimes
概要	何回か 10 倍する
引数 1	const Number *a: 10 倍する多倍長整数
引数 2	Number *b: 10 倍した結果を代入する多倍長整数
引数 3	int k: 10 倍する回数
戻り値	int: 成功した場合は TRUE，失敗した場合は FALSE

リスト 20 のように mulBy10SomeTimes 関数を呼び出すと，リスト 21 のように動作する．

リスト 20: mulBy10SomeTimes 関数の呼び出し

```
1  #include "pi.c"
2  int main (void){
3      Number a, b;
4      clearByZero(&a);
5      clearByZero(&b)
6      a.n[0] = 12;
7      a.n[2] = 34;
8      setSign(&a, PLUS);
9      mulBy10SomeTimes(&a, &b, 2);
10     dispNumber(&a);
11     printf("\n");
12     dispNumber(&b);
13     printf("\n");
14     return 0;
15 }
```

リスト 21: リスト 20 の実行結果

```
1  + 34 00 12
2  + 34 00 12 00
```

リスト 21 の実行結果から，mulBy10SomeTimes 関数は引数 a を引数 k 回だけ 10 倍して引数 b に代入することがわかる．

mulBy10SomeTimes 関数は次の手順で実装する．この手順を模式的に示した図を図 2 に示す．

1. 多倍長整数を 10^k 倍したときの桁数が多倍長整数に格納できる桁数を超える場合，*FALSE* を返す．(図 2 の手順 1)
2. 多倍長整数を k / *RADIX_LEN* 要素数，上位要素に移動させる．(図 2 の手順 2)
3. 多倍長整数を k % RADIX_LEN 桁数，上位桁に移動させる．(図 2 の手順 3)

4. 移動後の多倍長整数の下位桁を 0 で埋める．
5. 符号を設定する．
6. TRUE を返す．

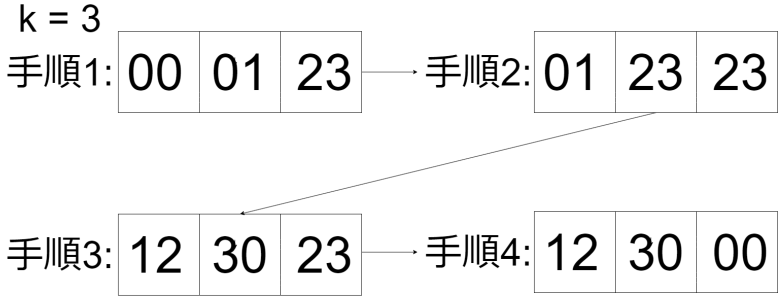

図 2: `mulBy10SomeTimes` 関数の手順

この手順をプログラム上では次の手順で実装する．

1. k が RADIX_LEN よりも大きい場合，k / RADIX_LEN 要素数，上位要素に移動させても多倍長整数に値を格納できるかどうかを判定する．
   格納できない場合，FALSE を返す．
2. 1 に加えて，RADIX_LEN - k % RADIX_LEN 桁数，上位桁に移動させても多倍長整数に値を格納できるかどうかを判定する．
   格納できない場合，FALSE を返す．
3. 1 と 2 が共に格納できる場合，以下の処理を繰り返す．(前述している手順の手順 2 にあたる)
   (a) *a*->*[i + k / RADIX_LEN](i=*多倍長整数の要素数*)* に *a*->[i] を代入する．
   (b) i を 1 減らす．
   (c) i が 0 になるまで繰り返す．
4. 以下の処理を繰り返す．(前述している手順の手順 4 にあたる)
   (a) *a*->[i](i=0) に 0 を代入する．
   (b) *i* を 1 増やす．
   (c) *i* が *k / RADIX_LEN* になるまで繰り返す．
5. 以下の処理を繰り返す．(前述している手順の手順 3 にあたる)
   (a) *a*->[i](i=0) に $10^{k\%RADIX\_LEN}$ かける．
   (b) 前の桁からの繰り上がり変数 *carry* を *a*->[i] に加える．
   (c) *a*->[i] が RADIX より大きい場合，*a*->[i] / *RADIX* を *carry* に代入し，*a*->[i] % *RADIX* を代入する．
   (d) a_i が RADIX より小さい場合，carry に 0 を代入する．
   (e) i を 1 増やす．

(f) i が a の要素数になるまで繰り返す.

6. 符号を設定する.
7. TRUE を返す.

10 倍する処理を手順 3 と手順 5 に分けているのは，一回の処理で要素を 10^k 倍すると，k が大きい場合に数が配列に格納できる最大値を超え，オーバーフローが発生するためである.

mulBy10SomeTimes 関数のソースコードをリスト 22 に示す.

リスト 22: mulBy10SomeTimes 関数

```
1   int mulBy10SomeTimes(const Number *a, Number *b, int k) {
2       int rtn = -2;
3       int i, j;
4       copyNumber(b, a);
5       if (isZero(a)) {
6           rtn = TRUE;
7       } else if (k == 0) {
8           rtn = TRUE;
9       } else {
10          int digit;
11          RADIX_T carry;
12          digit = k / RADIX_LEN;
13          int length = getLen(b);
14          j = 0;
15          i = KETA - 1;
16          while (1) {
17              if (digit <= j) {
18                  break;
19              } else if (b->n[i] != 0) {
20                  printf("mulBy10SomeTimes: overflow: -1A\n");
21                  rtn = FALSE;
22                  break;
23              }
24              j++;
25              i--;
26          }
27          if (rtn == FALSE) {
28              printf("mulBy10SomeTimes: overflow: -1B\n");
29          } else if (b->n[i] /
30                          (int)pow(10, (RADIX_LEN - (k % RADIX_LEN))) !=
31                      0) {
32              printf("mulBy10SomeTimes: overflow\n");
33              rtn = FALSE;
```

```
34              } else {
35                  if (digit != 0) {
36                      for (i = length / RADIX_LEN; i >= 0; i--) {
37                          b->n[i + digit] = b->n[i];
38                      }
39                      for (i = 0; i < digit; i++) {
40                          b->n[i] = 0;
41                      }
42                  }
43                  length += digit * RADIX_LEN;
44                  carry = 0;
45                  for (i = 0; i < length / RADIX_LEN + 2; i++) {
46                      b->n[i] *= (int)pow(10, (k % RADIX_LEN));
47                      b->n[i] += carry;
48                      if (b->n[i] >= RADIX) {
49                          carry = b->n[i] / RADIX;
50                          b->n[i] %= RADIX;
51                      } else {
52                          carry = 0;
53                      }
54                  }
55                  rtn = TRUE;
56              }
57          }
58          return rtn;
59      }
```

4.3.9 divBy10SomeTimes

divBy10SomeTimes 関数は表 12 のように宣言されている.

表 12: divBy10SomeTimes 関数

関数名	divBy10SomeTimes
概要	何回か 10 分の 1 する
引数 1	const Number *a: 10 分の 1 する多倍長整数
引数 2	Number *b: 10 分の 1 した結果を代入する多倍長整数
引数 3	int k: 10 分の 1 する回数
戻り値	なし

リスト 23 のように divBy10SomeTimes 関数を呼び出すと，リスト 24 のように動作する.

リスト 23: divBy10SomeTimes 関数の呼び出し

```
1  #include "pi.c"
2  int main (void){
3      Number a, b;
4      clearByZero(&a);
5      clearByZero(&b)
6      a.n[0] = 12;
7      a.n[2] = 34;
8      setSign(&a, PLUS);
9      divBy10SomeTimes(&a, &b, 3);
10     dispNumber(&a);
11     printf("\n");
12     dispNumber(&b);
13     printf("\n");
14     return 0;
15 }
```

リスト 24: リスト 23 の実行結果

```
1  + 34 00 12
2  + 03 40
```

リスト 24 の実行結果から，divBy10SomeTimes 関数は引数 a を引数 k 回だけ 10 分の 1 して引数 b に代入する動作をすることがわかる．

divBy10SomeTimes 関数は次の手順で実装する．

1. 多倍長整数を k % *RADIX_LEN* 桁数，下位桁に移動させる．
2. 多倍長整数を k / *RADIX_LEN* 要素数，下位要素に移動させる．
3. 移動後の多倍長整数の上位桁を 0 で埋める．
4. 符号を設定する．

この手順をプログラム上では次の手順で実装する．

1. *b*->[0] を 10^k で割った値を *b*->[0] に代入する．
2. 以下の処理を繰り返す (前述する手順の手順 1 にあたる)．
   (a) *b->[i](i=1)* を 10^k で割った値を *carry* に代入する．
   (b) *b->[i - 1]* に $carry \times 10^{RADIX\_LEN-(k\%RADIX\_LEN)}$ を加える．
   (c) *b->[i]* から *carry* を引き，$10^{k\%RADIX\_LEN}$ で割った値を *b->[i]* に代入する．
   (d) i を 1 増やす．
   (e) i が *KETA* になるまで繰り返す．
3. 以下の処理を繰り返す (前述している手順の手順 2 にあたる)．
   (a) *b->[i](i=0)* に *b->[i + k / RADIX_LEN]* を代入する．

(b) i を 1 増やす.

(c) i が *KETA - k / RADIX_LEN* になるまで繰り返す.

4. 以下の処理を繰り返す (前述している手順の手順 3 にあたる).

(a) *b->[i](i=KETA - k / RADIX_LEN)* に 0 を代入する.

(b) i を 1 増やす.

(c) i が *KETA* になるまで繰り返す.

divBy10SomeTimes 関数のソースコードをリスト 25 に示す.

リスト 25: divBy10SomeTimes 関数

```
1    void divBy10SomeTimes(const Number *a, Number *b, int k) {
2        int i;
3        int digit;
4        int carry;
5        digit = k / RADIX_LEN;
6        copyNumber(b, a);
7        b->n[0] -= b->n[0] % (int)pow(10, k % RADIX_LEN);
8        b->n[0] /= (int)pow(10, k % RADIX_LEN);
9        for (i = 1; i < KETA; i++) {
10           carry = b->n[i] % (int)pow(10, k % RADIX_LEN);
11           b->n[i - 1] +=
12               carry * (int)pow(10, RADIX_LEN - (k % RADIX_LEN));
13           b->n[i] -= carry;
14           b->n[i] /= (int)pow(10, k % RADIX_LEN);
15       }
16       if (digit > 0) {
17           for (i = 0; i < KETA - digit; i++) {
18               b->n[i] = b->n[i + digit];
19           }
20           for (i = KETA - digit; i < KETA; i++) {
21               b->n[i] = 0;
22           }
23       }
24       return;
25   }
```

4.3.10 setInt

setInt 関数は表 13 のように宣言されている.

表 13: setInt 関数

関数名	setInt
概要	整数を多倍長整数に設定する
引数 1	Number *a: 設定する多倍長整数
引数 2	long n: 設定する整数
戻り値	int: 成功した場合は TRUE，失敗した場合は FALSE

リスト 26 のように setInt 関数を呼び出すと，リスト 27 のように動作する．

リスト 26: setInt 関数の呼び出し

```
1    #include "pi.c"
2    int main (void){
3        Number a;
4        setInt(&a, 1234);
5        dispNumber(&a);
6        printf("\n");
7        return 0;
8    }
```

リスト 27: リスト 26 の実行結果

```
1    + 12 34
```

リスト 27 の実行結果から，setInt 関数は引数 n を引数 a に代入するという動作をすることがわかる．

setInt 関数は次の手順で実装する．

1. 引数 n が 0 の場合，多倍長整数 a の符号を $ZERO$ に設定する．
2. 引数 n が負の値の場合，多倍長整数 a の符号を $MINUS$ に設定し，引数 n をプラスに変換する．
3. 引数 n が正の値の場合，多倍長整数 a の符号を $PLUS$ に設定する．
4. 引数 n を多倍長整数 a に代入する．
5. TRUE を返す．

この手順をプログラム上では次の手順で実装する．

1. 引数 n が負の値の場合，*setSign* 関数で引数 a の符号を $MINUS$ に設定し，n をプラスに変換する．
2. 引数 n が 0 の場合，*setSign* 関数で引数 a の符号を $ZERO$ に設定する．
3. 引数 n が正の値の場合，*setSign* 関数で引数 a の符号を $PLUS$ に設定する．

4. 引数 n の先頭の配列要素に n を代入する．
5. TRUE を返す．

この関数では，引数 n が多倍長整数の基数を超える場合には対応していない．そのため，引数 n が多倍長整数の基数を超える場合には，正しく動作しない点に注意が必要である．setInt 関数のソースコードをリスト 28 に示す．

リスト 28: setInt 関数

```
1    int setInt(Number *a, long x) {
2        clearByZero(a);
3        if (x < 0) {
4            setSign(a, MINUS);
5            x *= -1;
6        } else if (x == 0) {
7            setSign(a, ZERO);
8        } else {
9            setSign(a, PLUS);
10       }
11       a->n[0] = x % RADIX;
12       return TRUE;
13   }
```

4.3.11 getInt

getInt 関数は表 14 のように宣言されている．

表 14: getInt 関数

関数名	getInt
概要	多倍長整数を整数に変換する
引数 1	const Number *a: 変換する多倍長整数
引数 2	int *x: 変換した整数を代入する変数
戻り値	int: 成功した場合は TRUE，失敗した場合は FALSE

リスト 29 のように getInt 関数を呼び出すと，リスト 30 のように動作する．

リスト 29: getInt 関数の呼び出し

```
1    #include "pi.c"
2    int main (void){
3        Number a;
4        int x;
```

```
5         setInt(&a, 1234);
6         dispNumber(&a);
7         printf("\n");
8         getInt(&a, &x);
9         printf("%d\n", x);
10        return 0;
11    }
```

リスト 30: リスト 29 の実行結果

```
1     + 12 34
2     1234
```

リスト 30 の実行結果から，getInt 関数は引数 a を整数に変換して引数 x に代入するという動作をすることがわかる．

getInt 関数は次の手順で実装する．

1. 多倍長整数 a の符号が *ZERO* の場合，0 を返す．
2. 多倍長整数の値を変数 x に代入する．
3. 多倍長整数の符号が *MINUS* の場合，x に-1 を掛ける．
4. TRUE を返す．

この手順をプログラム上では次の手順で実装する．

1. 引数 a が 0 の場合，引数 x に 0 を代入し，*TRUE* を返す．
2. 引数 a の桁数が RADIX_LEN よりも大きい場合，*FALSE* を返す．
3. 引数 a の先頭の配列要素を x に代入する．
4. 引数 a の符号を *getSign* で取得し，それが *MINUS* の場合，x に-1 を掛ける．
5. TRUE を返す．

この関数では，多倍長整数の値が int 型の範囲を超える場合には対応していない．そのため，多倍長整数の値が int 型の範囲を超える場合には，正しく動作しない点に注意が必要である．

getInt 関数のソースコードをリスト 31 に示す．

リスト 31: getInt 関数

```
1     int getInt(const Number *a, int *x) {
2         int rtn;
3         if (isZero(a)) {
4             *x = 0;
5             rtn = TRUE;
6         } else if (getLen(a) > RADIX_LEN) {
7             rtn = FALSE;
```

```
8         } else {
9             *x = a->n[0];
10            *x += a->n[1] * RADIX;
11            if (getSign(a) == MINUS) {
12                *x *= -1;
13            }
14            rtn = TRUE;
15        }
16        return rtn;
17    }
```

4.3.12 numComp

numComp 関数は表 15 のように宣言されている.

表 15: numComp 関数

関数名	numComp
概要	2 つの多倍長整数を比較する
引数 1	const Number *a: 比較する多倍長整数 1
引数 2	const Number *b: 比較する多倍長整数 2
戻り値	int: a が b より大きい場合は 1，a が b より小さい場合は-1，等しい場合は 0 を返す

リスト 32 のように numComp 関数を呼び出すと，リスト 33 のように動作する.

リスト 32: numComp 関数の呼び出し

```
1     #include "pi.c"
2     int main (void){
3         Number a, b;
4         int rtn;
5         setInt(&a, 1234);
6         setInt(&b, 1234);
7         rtn = numComp(&a, &b);
8         dispNumber(&a);
9         printf("\n");
10        dispNumber(&b);
11        printf("\n");
12        printf("%d\n", rtn);
13        setInt(&b, 1235);
14        rtn = numComp(&a, &b);
15        dispNumber(&a);
```

```
16          printf("\n");
17          dispNumber(&b);
18          printf("\n");
19          printf("%d\n", rtn);
20          setInt(&b, 1233);
21          rtn = numComp(&a, &b);
22          dispNumber(&a);
23          printf("\n");
24          dispNumber(&b);
25          printf("\n");
26          printf("%d\n", rtn);
27          return 0;
28      }
```

リスト 33: リスト 32 の実行結果

```
1   + 12 34
2   + 12 34
3   0
4   + 12 34
5   + 12 35
6   -1
7   + 12 34
8   + 12 33
9   1
```

リスト 33 の実行結果から，numComp 関数は引数 a と引数 b を比較して，a が b より大きい場合は 1，a が b より小さい場合は-1，等しい場合は 0 を返すという動作をすることがわかる．

numComp 関数は次の手順で実装する．

1. 多倍長整数 a と多倍長整数 b の符号が異なる場合は符号での比較を行い，比較結果に応じて 1 または-1 を返す．
2. 多倍長整数 a と多倍長整数 b の符号が同じ場合は，多倍長整数 a と多倍長整数 b の絶対値を比較し，符号に応じて結果を返す．

この手順をプログラム上では次の手順で実装する．

1. 引数 a と引数 b の符号によって場合わけをし，以下の処理を行う．
   - 引数 a の符号が $MINUS$ で引数 b の符号が $PLUS$ の場合，-1 を返す．
   - 引数 a の符号が $PLUS$ で引数 b の符号が $MINUS$ の場合，1 を返す．
   - 引数 a の符号が $ZERO$ で引数 b の符号が $ZERO$ の場合，0 を返す．
   - 引数 a の符号が $PLUS$ で引数 b の符号が $ZERO$ の場合，1 を返す．

- 引数 a の符号が $MINUS$ で引数 b の符号が $ZERO$ の場合，-1 を返す.
- 引数 a の符号が $ZERO$ で引数 b の符号が $PLUS$ の場合，-1 を返す.
- 引数 a の符号が $ZERO$ で引数 b の符号が $MINUS$ の場合，1 を返す.
- 引数 a の符号が $PLUS$ で引数 b の符号が $PLUS$ の場合や，引数 a の符号が $MINUS$ で引数 b の符号が $MINUS$ の場合，手順 2 に進む.

2. 桁数を比較し，結果に応じて 1 または-1 を返す.
3. 桁数が同じな場合，上位の桁から比較し，結果に応じて 1 または-1 を返す.
4. すべての桁が同じ場合，0 を返す.

numComp 関数のソースコードをリスト 34 に示す.

リスト 34: numComp 関数

```
1     int numComp(const Number *a, const Number *b) {
2         int rtn = 0;
3         switch (getSign(a) * 3 + getSign(b)) {
4             case -4: // とが負 ab
5                 if (getLen(a) > getLen(b)) {
6                     rtn = -1;
7                 } else if (getLen(a) < getLen(b)) {
8                     rtn = 1;
9                 }
10                for (int i = KETA - 1; i >= 0; i--) {
11                    if (a->n[i] < b->n[i]) {
12                        rtn = 1;
13                        break;
14                    } else if (a->n[i] > b->n[i]) {
15                        rtn = -1;
16                        break;
17                    }
18                }
19                break;
20            case 4: // とが正 ab
21                if (getLen(a) > getLen(b)) {
22                    rtn = 1;
23                } else if (getLen(a) < getLen(b)) {
24                    rtn = -1;
25                }
26                for (int i = KETA - 1; i >= 0; i--) {
27                    if (a->n[i] > b->n[i]) {
28                        rtn = 1;
29                        break;
30                    } else if (a->n[i] < b->n[i]) {
```

```
31                          rtn = -1;
32                          break;
33                      }
34                  }
35                  break;
36              case -3: // が負でが ab0
37              case -2: // が負でが正 ab
38              case 1: // がでが正 a0b
39                  rtn = -1;
40                  break;
41              case -1: // がでが負 a0b
42              case 2: // が正でが負 ab
43              case 3: // が正でが ab0
44                  rtn = 1;
45                  break;
46              case 0: // とが ab0
47                  break;
48          }
49          return rtn;
50      }
```

4.3.13 numCompWithInt

numCompWithInt 関数は表 16 のように宣言されている．

表 16: numCompWithInt 関数

関数名	numCompWithInt
概要	多倍長整数と int 型整数を比較する
引数 1	const Number *a: 比較する多倍長整数
引数 2	int b: 比較する int 型整数
戻り値	int: a が b より大きい場合は 1，a が b より小さい場合は-1，等しい場合は 0 を返す

リスト 35 のように numCompWithInt 関数を呼び出すと，リスト 36 のように動作する．

リスト 35: numCompWithInt 関数の呼び出し

```
1   #include "pi.c"
2   int main (void){
3       Number a;
4       int rtn;
5       setInt(&a, 1234);
```

```
6         rtn = numCompWithInt(&a, 1234);
7         dispNumber(&a);
8         printf("\n");
9         printf("%d\n", rtn);
10        rtn = numCompWithInt(&a, 1235);
11        dispNumber(&a);
12        printf("\n");
13        printf("%d\n", rtn);
14        rtn = numCompWithInt(&a, 1233);
15        dispNumber(&a);
16        printf("\n");
17        printf("%d\n", rtn);
18        return 0;
19    }
```

リスト 36: リスト 35 の実行結果

```
1     + 12 34
2     0
3     + 12 34
4     -1
5     + 12 34
6     1
```

リスト 36 の実行結果から，numCompWithInt 関数は引数 a と引数 b を比較して，a が b より大きい場合は 1，a が b より小さい場合は-1，等しい場合は 0 を返すという動作をすることがわかる．

numCompWithInt 関数は多倍長整数 a を int 型整数に変換し，引数 x と比較して，比較結果に応じて戻り値を返すことで動作する．

その動作をプログラム上で次の手順で実装する．

1. 引数 a を *getInt* 関数で int 型整数に変換する．
2. 変換に失敗した場合，1 を返す．
3. int 型に変換した整数と引数 b を比較し，比較結果に応じて戻り値を返す．

numCompWithInt 関数のソースコードをリスト 37 に示す．

リスト 37: numCompWithInt 関数

```
1     int numCompWithInt(const Number *a, int x) {
2         int num, rtn;
3         if (getInt(a, &num) == FALSE) {
4             rtn = TRUE;
5         } else if (num > x) {
6             rtn = TRUE;
```

```
7         } else if (num < x) {
8             rtn = FALSE;
9         } else {
10            rtn = 0;
11        }
12        return rtn;
13    }
```

4.3.14 add, sub

add 関数と sub 関数は表 17，表 18 のように宣言されている．

表 17: add 関数

関数名	add
概要	2 つの多倍長整数を加算する
引数 1	const Number *a: 加算する多倍長整数 1
引数 2	const Number *b: 加算する多倍長整数 2
引数 3	Number *c: 加算結果を代入する多倍長整数
戻り値	int: 成功した場合は TRUE，失敗した場合は FALSE

表 18: sub 関数

関数名	sub
概要	2 つの多倍長整数を減算する
引数 1	const Number *a: 減算する多倍長整数 1
引数 2	const Number *b: 減算する多倍長整数 2
引数 3	Number *c: 減算結果を代入する多倍長整数
戻り値	int: 成功した場合は TRUE，失敗した場合は FALSE

リスト 38 のように add 関数と sub 関数を呼び出すと，リスト 39 のように動作する．

リスト 38: add 関数と sub 関数の呼び出し

```
1     #include "pi.c"
2     int main (void){
3         Number a, b, c;
4         int rtn;
5         setInt(&a, 1234);
6         setInt(&b, 5678);
7         rtn = add(&a, &b, &c);
```

```
8           dispNumber(&a);
9           printf(" + ");
10          dispNumber(&b);
11          printf(" = ");
12          dispNumber(&c);
13          printf("\n");
14          printf("%d\n", rtn);
15          rtn = sub(&a, &b, &c);
16          dispNumber(&a);
17          printf(" - ");
18          dispNumber(&b);
19          printf(" = ");
20          dispNumber(&c);
21          printf("\n");
22          printf("%d\n", rtn);
23          return 0;
24      }
```

リスト 39: リスト 38 の実行結果

```
1       + 12 34 + + 56 78 = 69 12
2       1
3       + 12 34 - + 56 78 = - 44 44
```

リスト 39 の実行結果から，add 関数と sub 関数は引数 a と引数 b を加算，減算して，引数 c に代入するという動作をすることがわかる．

add 関数と sub 関数はどちらも符号で場合分けをし，式を変形してから計算を行っている．add 関数の式変形は次のように行っている．

- a が正，b が正の場合，$a + b = a + b$
- a が正，b が負の場合，$a + b = a - |b|$
- a が負，b が正の場合，$a + b = b - |a|$
- a が負，b が負の場合，$(-a) + (-b) = -(|a| + |b|)$
- a が 0 の場合，$a + b = b$
- b が 0 の場合，$a + b = a$

sub 関数の式変形は次のように行っている．

- a が正，b が正の場合，$a - b = a - b$
- a が正，b が負の場合，$a - b = a + |b|$
- a が負，b が正の場合，$a - b = -(|a| + b)$
- a が負，b が負の場合，$a - b = |a| - |b|$

- a が 0 の場合，$a - b = -b$
- b が 0 の場合，$a - b = a$

なので，add 関数と sub 関数ともに，引数 a と引数 b の符号がどちらも正の場合の計算手順を示す．add 関数の計算手順は次のようになる．

1. 多倍長整数 $a_i(i = 0)$ と多倍長整数 b_i と $carry$ を加算し，変数 num に代入する
2. 変数 num が $RADIX$ より大きい場合，変数 num を $RADIX$ で割った余りを多倍長整数 c_i に代入し，繰り上がりとして変数 $carry$ に 1 を代入する．
3. 変数 num が $RADIX$ 以下の場合，変数 num を多倍長整数 c_i に代入し，繰り上がりなしとして変数 $carry$ に 0 を代入する．
4. i を 1 増やす．
5. i が多倍長整数の要素数に達するまで手順 1 から 4 を繰り返す．
6. ループを抜けた後，繰り上がりがある場合，多倍長整数に格納できないのでオーバーフローとして $FALSE$ を返す．
7. 繰り上がりがない場合，$TRUE$ を返す．

sub 関数の計算手順は次のようになる．

1. 多倍長整数 $a_i(i = 0)$ か r 繰り下がり変数 $carry$ を減算し，変数 num に代入する
2. 変数 num が b_i より小さい場合，変数 num に $RADIX$ を加え，繰り下がりとして変数 $carry$ に 1 を代入する．
3. 変数 num が b_i 以上の場合，変数 num を多倍長整数 c_i に代入し，繰り下がりなしとして変数 $carry$ に 0 を代入する．
4. i を 1 増やす．
5. i が多倍長整数の要素数に達するまで手順 1 から 4 を繰り返す．
6. ループを抜けた後，繰り下がりがある場合，多倍長整数に格納できないのでオーバーフローとして $FALSE$ を返す．
7. 繰り下がりがない場合，$TRUE$ を返す．

add 関数と sub 関数のソースコードをリスト 40，リスト 41 に示す．

リスト 40: add 関数

```
1     int add(const Number *a, const Number *b, Number *c) {
2         Number A, B;
3         RADIX_T d;
4         int i, rtn;
5         int e = 0;
6         rtn = -2;
7         int caseNum = getSign(a) * 3 + getSign(b);
8         switch (caseNum) {
```

```
9               case -4: // とが負 ab
10              case 4: // とが正 ab
11                  // clearByZero(c);
12                  if (caseNum == -4) {
13                      getAbs(a, &A);
14                      getAbs(b, &B);
15                      setSign(c, MINUS);
16                  } else {
17                      copyNumber(&A, a);
18                      copyNumber(&B, b);
19                      setSign(c, PLUS);
20                  }
21                  for (i = 0; i < KETA; i++) {
22                      d = A.n[i] + B.n[i] + e;
23                      e = 0;
24                      if (d >= RADIX) {
25                          d -= RADIX;
26                          e = 1;
27                      } else {
28                          e = 0;
29                      }
30                      c->n[i] = d;
31                  }
32                  if (e == 1) {
33                      rtn = FALSE;
34                  } else {
35                      rtn = TRUE;
36                  }
37                  break;
38              case -3: // が負でが ab0
39              case 3: // が正でが ab0
40                  copyNumber(c, a);
41                  rtn = TRUE;
42                  break;
43              case -2: // が負でが正 ab
44                  getAbs(a, &A);
45                  rtn = sub(b, &A, c);
46                  break;
47              case -1: // がでが負 a0b
48              case 1: // がでが正 a0b
49                  copyNumber(c, b);
50                  rtn = TRUE;
```

```
51                  break;
52              case 0: // とが ab0
53                  clearByZero(c);
54                  rtn = TRUE;
55                  break;
56              case 2: // が正でが負 ab
57                  getAbs(b, &B);
58                  rtn = sub(a, &B, c);
59                  break;
60          }
61          return rtn;
62      }
```

リスト 41: sub 関数

```
1       int sub(const Number *a, const Number *b, Number *c) {
2           Number A, B;
3           copyNumber(&A, a);
4           copyNumber(&B, b);
5           int i, e, num, rtn;
6           int caseNum = getSign(&A) * 3 + getSign(&B);
7           Number numA, numB;
8           rtn = -2;
9           switch (caseNum) {
10              case -4: // とが負 ab
11              case 4: // とが正 ab
12                  if (caseNum == -4) {
13                      getAbs(&A, &numB);
14                      getAbs(&B, &numA);
15                  } else {
16                      copyNumber(&numA, &A);
17                      copyNumb(&numB, &B);
18                  }
19                  e = 0;
20                  switch (numComp(&A, &B)) {
21                      case 1:
22                          for (i = 0; i < KETA; i++) {
23                              num = numA.n[i] - e;
24                              if (num < numB.n[i]) {
25                                  c->n[i] = num + RADIX - numB.n[i];
26                                  e = 1;
27                              } else {
28                                  c->n[i] = num - numB.n[i];
```

```
29                              e = 0;
30                          }
31                          setSign(c, PLUS);
32                      }
33                      break;
34                  case -1:
35                      for (i = 0; i < KETA; i++) {
36                          num = numB.n[i] - e;
37                          if (num < numA.n[i]) {
38                              c->n[i] = num + RADIX - numA.n[i];
39                              e = 1;
40                          } else {
41                              c->n[i] = num - numA.n[i];
42                              e = 0;
43                          }
44                      }
45                      setSign(c, MINUS);
46                      break;
47                  case 0:
48                      clearByZero(c);
49                      setSign(c, ZERO);
50                      break;
51              }
52              if (e > 0) {
53                  rtn = FALSE;
54              } else {
55                  rtn = TRUE;
56              }
57              break;
58          case -3: // が負でが ab0
59          case 3: // が正でが ab0
60              copyNumber(c, &A);
61              rtn = 0;
62              break;
63          case -2: // が負でが正 ab
64              getAbs(&A, &A);
65              rtn = add(&A, &B, c);
66              setSign(c, MINUS);
67              break;
68          case -1: // がでが負 a0b
69              copyNumber(c, &B);
70              setSign(c, PLUS);
```

```
71                  rtn = 0;
72                  break;
73              case 1: // がでが正 a0b
74                  copyNumber(c, &B);
75                  setSign(c, MINUS);
76                  rtn = 0;
77                  break;
78              case 0: // とが ab0
79                  clearByZero(c);
80                  rtn = 0;
81                  break;
82              case 2: // が正でが負 ab
83                  getAbs(&B, &B);
84                  rtn = add(&A, &B, c);
85                  break;
86          }
87          return rtn;
88      }
```

4.3.15 multiple

multiple 関数は表 19 のように宣言されている.

表 19: multiple 関数

関数名	multiple
概要	2 つの多倍長整数を乗算する
引数 1	const Number *a: 乗算する多倍長整数 1
引数 2	const Number *b: 乗算する多倍長整数 2
引数 3	Number *c: 乗算結果を代入する多倍長整数
戻り値	int: 成功した場合は TRUE, 失敗した場合は FALSE

リスト 42 のように multiple 関数を呼び出すと，リスト 43 のように動作する.

リスト 42: multiple 関数の呼び出し

```
1   #include "pi.c"
2   int main (void){
3       Number a, b, c;
4       int rtn;
5       setInt(&a, 1234);
6       setInt(&b, 5678);
```

```
7         rtn = multiple(&a, &b, &c);
8         dispNumber(&a);
9         printf(" * ");
10        dispNumber(&b);
11        printf(" = ");
12        dispNumber(&c);
13        printf("\n");
14        printf("%d\n", rtn);
15        return 0;
16    }
```

リスト 43: リスト 42 の実行結果

```
1     + 12 34 * + 56 78 = + 70 00 52 52
2     1
```

リスト 43 の実行結果から，multiple 関数は引数 a と引数 b を乗算して，引数 c に代入するという動作をすることがわかる．

multiple 関数は次の手順で実装する．

1. 多倍長整数 $a_i(i = 0)$ と多倍長整数 $b_j(j = 0)$ を乗算し，変数 num に代入する
2. 変数 num が 0 の場合は，後の処理をスキップし，j を 1 増やし，手順 1 に戻る．
3. 変数 num の $RADIX$ よりも小さい部分を多倍長整数 c_{i+j} に加算し，それより大きい部分を繰り上がりとして多倍長整数 c_{i+j+1} に加算する．
4. 繰り上がりを足したことにより，c_i が $RADIX$ よりも大きくなることがあるため，繰り上がりを再計算する．
5. j を 1 増やし，手順 1 に戻る．
6. j が多倍長整数の要素数に達するまで手順 1 から 3 を繰り返す．
7. i を 1 増やし，手順 1 に戻る．
8. i が多倍長整数の要素数に達するまで手順 1 から 4 を繰り返す．

multiple 関数のソースコードをリスト 44 に示す．

リスト 44: multiple 関数

```
1     int multiple(const Number *a, const Number *b, Number *c) {
2         int rtn = -2;
3         int signA, signB;
4         signA = getSign(a);
5         signB = getSign(b);
6         if (signA == ZERO || signB == ZERO) {
7             clearByZero(c);
8             rtn = 0;
```

```
9         } else {
10            RADIX_T tmp;
11            Number A, B;
12            getAbs(a, &A);
13            getAbs(b, &B);
14            clearByZero(c);
15            int ALength = getLen(&A);
16            int BLength = getLen(&B);
17            for (int i = 0; i < ALength / 9 + 1; i++) {
18                for (int j = 0; j < BLength / 9 + 1; j++) {
19                    tmp = A.n[i] * B.n[j];
20                    if (tmp == 0) {
21                        continue;
22                    }
23                    c->n[i + j] += tmp % RADIX;
24                    c->n[i + j + 1] += tmp / RADIX;
25                    c->n[i + j + 1] += c->n[i + j] / RADIX;
26                    c->n[i + j] %= RADIX;
27                }
28            }
29            switch (signA * 3 + signB) {
30                case -4: // とが負 ab
31                case 4: // とが正 ab
32                    setSign(c, PLUS);
33                    break;
34                case -2: // が負でが正 ab
35                case 2: // が正でが負 ab
36                    setSign(c, MINUS);
37                    break;
38                    // case 3:,case 1:,case 0:,case -1:,case
39                    // は最初で判定するのでここには来ない-3:
40            }
41            rtn = 0;
42        }
43        return rtn;
44    }
```

4.3.16 inverse3

inverse3 関数は表 20 のように宣言されている.

表 20: inverse3 関数

関数名	inverse3
概要	逆数を求める
引数 1	const Number *a: 逆数を求める多倍長整数
引数 2	Number *b: 逆数を代入する多倍長整数
戻り値	余裕を持っている桁数.

リスト 45 のように inverse3 関数を呼び出すと，リスト 46 のように動作する．

リスト 45: inverse3 関数の呼び出し

```
1   #include "pi.c"
2   int main (void){
3       Number a, b;
4       int rtn;
5       setInt(&a, 1234);
6       rtn = inverse3(&a, &b);
7       dispNumber(&a);
8       printfの逆数(" = ");
9       dispNumber(&b);
10      printf("\n");
11      printf("%d\n", rtn);
12      return 0;
13  }
```

リスト 46: リスト 45 の実行結果

```
1   + 12 の逆数 34 = + 81 00
2   1
```

リスト 46 の実行結果から，inverse3 関数は引数 a の逆数を求めて，引数 b に代入するという動作をすることがわかる．

inverse3 関数は章 2 の式 (24) を用いて逆数を求めている．式 (24) では，1 回試行するごとに，3 乗ずつ桁が求まる．この関数では式を適用する前と後で値が変わらなくなった場合，収束したと判断する．また，計算する時の精度を担保するために，求める逆数の小数点以下の値を求める値の桁数 +DIGIT 桁数求めるようにする．

関数内では (24) の式を $h = 1 - ax_i$ として (5) の形に変形して利用する．

$$x_{i+1} = x_i(1 + h + h^2) \tag{5}$$

初期値はできるだけ $1/a$ に近い値を選びたい．そこで次のように初期値を設定する．a を対数を使って表すと，(6) のようになる．

$$a = 10^{\log_{10} a} \tag{6}$$

このとき逆数 $1/a$ は (7) のようになる．

$$\frac{1}{a} = 10^{-\log_{10} a} \tag{7}$$

これでも十分近い初期値が得られるが，さらに (8) のように初期値を設定することで，より近い初期値を得ることができる．

$$x_0 = 0.2 * 10^{-\log_{10} a} \tag{8}$$

この 0.2 は突然出てきた値のように見えるが，実際に値を代入してみると 0.2 を使うことで最も初期値が実際の逆数の値に近くなることがわかる．表 21 に逆数を求める際の初期値の違いによる比較を示す．どの値も実際の逆数の値に近いが，0.2 を使った場合が最も安定的に近い値を得ることができる．

表 21: 逆数を求める際の初期値の違いによる比較

a	$x_0 = 0.1 * 10^{-\log_{10} a}$	$x_0 = 0.2 * 10^{-\log_{10} a}$	$x_0 = 0.3 * 10^{-\log_{10} a}$	実際の逆数の値
10	0.01	0.02	0.03	0.1
50	0.002	0.004	0.006	0.02
200	0.0005	0.001	0.0015	0.005

inverse3 関数は次の手順で実装する．

1. x_0 を設定する．
2. $x_{i+1} = x_i(1 + h + h^2)$ を計算する．
3. x_i と x_{i+1} の差が 1 桁以下になるまで手順 2 を繰り返す．
4. x_{i+1} を返す．

この手順をプログラム上では次の手順で実装する．

1. 小数点以下の値を多倍長整数で表すために，必要な桁数である「求めたい円周率の桁数 +a の桁数」を「有効数字を表す変数/textitsigDigs + `margin`」で表す．
2. a の桁数を変数 *length* に代入する．
3. 初期値を $2^{sigDigs+margin-length}$ に設定する．
4. 定数である 1 を $10^{sigDigs+margin}$ に設定する．
5. 以下の処理を繰り返す．
6. x_{i-1} にあたる，前の結果を多倍長整数 *x0* に代入する．
7. 多倍長整数 *tmp* に a と *x0* を乗算した結果を代入する．
8. $h = 1 - ax_i$ にあたる，多倍長整数 h に *one* と *tmp* を減算した結果を代入する．
9. h の 2 乗を計算し，多倍長整数 *tmp* に代入する．
10. $10^{sigDigs+margin}$ 倍している数どうしを乗算したため，*tmp* を $10^{sigDigs+margin}$ で割る．

11. tmp を h に加算した結果を tmp に代入する．
12. tmp を one で足した結果を tmp に代入する．
13. tmp を $x0$ で乗算した結果を b に代入する．
14. 手順 10 と同様に $10^{sigDigs+margin}$ 倍している数どうしを乗算したため，b を $10^{sigDigs+margin}$ で割る．
15. 誤差を求めるために，b と $x0$ を減算した結果を g に代入する．
16. g の桁数が 1 以下になった場合，収束したと判断する．そうじゃなかった場合，手順 5 に戻る．
17. 符号を設定する．
18. 返り値を設定し，返す．

inverse3 関数のソースコードをリスト 47 に示す．

リスト 47: inverse3 関数

```
1   int inverse3(const Number *a, Number *b) {
2       int rtn;
3       if (isZero(a)) {
4           clearByZero(b);
5           rtn = 0;
6       } else if (numCompWithInt(a, 1) == 0) {
7           copyNumber(b, a);
8           mulBy10SomeTimes(b, b, DIGIT + MARGIN);
9           rtn = 0;
10      } else {
11          Number x0; // ひとつ前のx
12          Number A; // 逆数を求める数
13          Number tmp; // 作業用変数
14          Number h;
15          Number g; // 逆数の誤差
16          Number bigOne;
17          int sigDigs = DIGIT + MARGIN;
18          int margin = 0;
19          int length = getLen(a);
20          if(length >= sigDigs + margin) {
21              margin += length;
22          } else {
23              margin += sigDigs;
24          }
25          getAbs(a, &A);
26          setInt(&bigOne, 1);
27          setInt(b, 2);
```

```
28              mulBy10SomeTimes(b, b, sigDigs + margin - length); // 初期値
29              mulBy10SomeTimes(&bigOne, &bigOne, sigDigs + margin);
30              while (1) {
31                  copyNumber(&x0, b); // ひとつ前のx
32                  if (multiple(&A, &x0, &tmp) == -1) {
33                      printf("ERROR:inverse2 overflow\n");
34                      clearByZero(b);
35                      rtn = -1;
36                      break;
37                  }
38                  sub(&bigOne, &tmp, &h);
39                  multiple(&h, &h, &tmp);
40                  divBy10SomeTimes(&tmp, &tmp, sigDigs + margin);
41                  add(&tmp, &h, &tmp);
42                  add(&tmp, &bigOne, &tmp);
43                  if (multiple(&x0, &tmp, b) == -1) {
44                      printf("ERROR:inverse2 overflow\n");
45                      clearByZero(b);
46                      rtn = FALSE;
47                      break;
48                  }
49                  divBy10SomeTimes(b, b, sigDigs + margin);
50                  sub(b, &x0, &g);
51                  if (getLen(&g) < 2) {
52                      break;
53                  }
54              }
55              rtn = margin;
56              setSign(b, getSign(a));
57          }
58          return rtn;
59      }
```

4.3.17 divideByInverse

divideByInverse 関数は表 22 のように宣言されている.

表 22: divideByInverse 関数

関数名	divideByInverse
概要	逆数を求めて乗算することで除算をする
引数 1	const Number *a: 除数
引数 2	const Number *b: 逆数を求める多倍長整数
引数 3	Number *c: 除算結果を代入する多倍長整数
戻り値	int: 成功した場合は TRUE，失敗した場合は FALSE

リスト 48 のように divideByInverse 関数を呼び出すと，リスト 49 のように動作する．

リスト 48: divideByInverse 関数の呼び出し

```
1   #include "pi.c"
2   int main (void){
3       Number a, b, c;
4       int rtn;
5       setInt(&a, 12340);
6       setInt(&b, 5678);
7       rtn = divideByInverse(&a, &b, &c);
8       dispNumber(&a);
9       printf(" / ");
10      dispNumber(&b);
11      printf(" = ");
12      dispNumber(&c);
13      printf("\n");
14      printf("%d\n", rtn);
15      return 0;
16  }
```

リスト 49: リスト 48 の実行結果

```
1   + 12 34 0 / + 56 78 = + 02
2   1
```

リスト 49 の実行結果から，divideByInverse 関数は引数 a を引数 b で除算して，引数 c に代入するという動作をすることがわかる．

divideByInverse 関数は次の手順で実装する．

1. 逆数を求める．
2. 逆数を求めた結果を引数 a と乗算する．
3. 乗算結果を引数 c に代入する．

4. TRUE を返す.
5. 逆数を求める際にエラーが発生した場合，FALSE を返す.

この手順をプログラム上では次の手順で実装する

1. 引数 a,b の符号で場合分けし，c の符号を設定する.
2. *inverse3* 関数を用いて逆数を求める.
3. 逆数を求めた結果を引数 a と乗算する.
4. 乗算した結果は textitinverse3 関数の戻り値 + 求めたい円周率の桁数分だけ 10 倍されているため，10 倍している分を割る.
5. 符号を設定する.
6. TRUE を返す.

divideByInverse 関数のソースコードをリスト 50 に示す.

リスト 50: divideByInverse 関数

```
1    int divideByInverse(const Number *a, const Number *b, Number *c) {
2        Number A, B;
3        getAbs(a, &A);
4        getAbs(b, &B);
5        if (numComp(&A, &B) == -1) {
6            clearByZero(c);
7            return 0;
8        }
9        int rtn;
10       int cSign;
11       Number inv;
12       int margin = 0;
13       switch ((getSign(a) < 0) * 2 + (getSign(b) < 0)) {
14           case 0: // が正でが正 AB
15           case 3: // が負でが負 AB
16               cSign = 1;
17               break;
18           case 1: // が正でが負 AB
19           case 2: // が負でが正 AB
20               cSign = -1;
21               break;
22       }
23       margin = inverse3(&B, &inv);
24       if (margin == FALSE) {
25           printf("ERROR:divideByInverse errorA\n");
26           rtn = FALSE;
```

```
27          } else {
28              if (multiple(&A, &inv, c) == -1) {
29                  printf("ERROR:divideByInverse errorB\n");
30                  rtn = FALSE;
31              } else {
32                  divBy10SomeTimes(c, c, DIGIT + MARGIN + margin);
33                  rtn = TRUE;
34              }
35          }
36          setSign(c, cSign);
37          return rtn;
38      }
```

4.3.18 sqrtThree

sqrtThree 関数は表 23 のように宣言されている．

表 23: sqrtThree 関数

関数名	sqrtThree
概要	$\sqrt{3}$ を求める
引数 1	Number *a: $\sqrt{3}$ を代入する多倍長整数
戻り値	int: 成功した場合は TRUE，失敗した場合は FALSE

リスト 51 のように sqrtThree 関数を呼び出すと，リスト 52 のように動作する．DIGIT が 9 であるとする．

リスト 51: sqrtThree 関数の呼び出し

```
1   #include "pi.c"
2   int main (void){
3       Number a;
4       int rtn;
5       rtn = sqrtThree(&a);
6       dispNumber(&a);
7       printf("\n");
8       printf("%d\n", rtn);
9       return 0;
10  }
```

リスト 52: リスト 51 の実行結果

```
1   + 17 32 05 88 77 25
```

リスト 52 の実行結果から，sqrtThree 関数は引数 a に $\sqrt{3}$ を求めたい円周率の桁数分代入するという動作をすることがわかる．

この sqrtThree 関数は次のように求めている．$\sqrt{3}$ の整数部分が 1 であることは自明である．そのため，小数部分を求めるため式 (9) が成り立つことがわかる．

$$|1 - \sqrt{3}| < 1 \tag{9}$$

ここで (10) に変形し，(11) に一般化する．

$$(1 - \sqrt{3})^2 \tag{10}$$

$$(a_0 - b_0\sqrt{3})^2 \tag{11}$$

(11) の一般化された式で，$\sqrt{3}$ の場合 a_0 と b_0 がそれぞれ 1 のときに成り立つ．これを展開すると (12) が得られる．

$$(a_0 - b_0\sqrt{3})^2 = a_0^2 + 3b_0^2 - 2a_0b_0\sqrt{3} \tag{12}$$

ここで，(13),(14) を適用することで (15) が得られる．

$$a_1 = a_0^2 + 3b_0^2 \tag{13}$$

$$b_1 = 2a_0b_0 \tag{14}$$

$$(a_0 - b_0\sqrt{3})^2 = a_1 + b_1\sqrt{3} \tag{15}$$

これを漸化式として表すと，(16)，(17) のようになる．

$$a_{n+1} = a_n^2 + b_n^2 \tag{16}$$

$$b_{n+1} = 2a_nb_n \tag{17}$$

この漸化式を進めていくと，$\sqrt{3}$ が n を 1 増やすごとに 2 倍ずつ桁が求まる．

このアルゴリズムを次の手順で実装する．

1. $a_0 = 1$，$b_0 = 1$ とする．
2. $a_{n+1} = a_n^2 + b_n^2$，$b_{n+1} = 2a_nb_n$ を計算する．
3. 必要な桁数が求まるまで手順 2 を繰り返す．

4. a と b の符号を設定する.
5. TRUE を返す
6. 途中でエラーが発生した場合，FALSE を返す.

この手順をプログラム上では次の手順で実装する.

1. 求めたい桁数である $DIGIT + MARGIN$ が 2 の何乗かを求めめ，変数 j に代入する.
2. 多倍長変数 a と b を用意し，それぞれ 1 を代入し，初期値とする.
3. $j = 0$ とし，以下の処理を繰り返す.
4. a と b の値をどちらも前の値として $a0$ と $b0$ に代入する.
5. $a0$ と $b0$ をどちらも 2 乗して a と b に代入する.
6. b と 3 を掛けて b に代入する.
7. a と b を加算して a に代入する.
8. $a0$ と $b0$ を掛けて b に代入する.
9. b を 2 倍して b に代入する.
10. j を 1 増やす.
11. j が $DIGIT + MARGIN$ より大きくなるまで手順 4 に戻る.
12. 符号を設定する.
13. TRUE を返す.
14. 途中でエラーが発生した場合，FALSE を返す.

sqrtThree 関数のソースコードをリスト 53 に示す.

リスト 53: sqrtThree 関数

```
1     int sqrtThree(Number *a) {
2         clearByZero(a);
3         int rtn;
4         Number numA, numB;
5         Number constant;
6         Number numA0, numB0;
7         Number two;
8         int digSig = DIGIT + MARGIN;
9         int i;
10        int j = 0;
11        i = 1;
12        // がの何乗かを求める digSig2
13        while (1) {
14            if (digSig < i) {
15                break;
16            }
17            i *= 2;
```

```
18              j++;
19          }
20
21          setInt(&constant, 3);
22          setInt(&two, 2);
23          setInt(&numA, 1);
24          copyNumber(&numB, &numA);
25          for (i = 0; i < j + 1; i++) {
26              printf("\rroot3 calculate %d", i);
27              fflush(stdout);
28              copyNumber(&numA0, &numA);
29              copyNumber(&numB0, &numB);
30              if (multiple(&numA0, &numA0, &numA) == FALSE) {
31                  printf("ERROR:sqrtThree overflowA\n");
32                  clearByZero(a);
33                  rtn = FALSE;
34                  break;
35              }
36              if (multiple(&numB0, &numB0, &numB) == FALSE) {
37                  printf("ERROR:sqrtThree overflowB\n");
38                  clearByZero(a);
39                  rtn = FALSE;
40                  break;
41              }
42              if (multiple(&numB, &constant, &numB) == FALSE) {
43                  printf("ERROR:sqrtThree overflowC\n");
44                  clearByZero(a);
45                  rtn = FALSE;
46                  break;
47              }
48              if (add(&numA, &numB, &numA) == FALSE) {
49                  printf("ERROR:sqrtThree overflowD\n");
50                  clearByZero(a);
51                  rtn = FALSE;
52                  break;
53              }
54              if (multiple(&numA0, &numB0, &numB) == FALSE) {
55                  printf("ERROR:sqrtThree overflowE\n");
56                  clearByZero(a);
57                  rtn = FALSE;
58                  break;
59              }
```

```
60              if (multiple(&numB, &two, &numB) == FALSE) {
61                  printf("ERROR:sqrtThree overflowF\n");
62                  clearByZero(a);
63                  rtn = FALSE;
64                  break;
65              }
66              rtn = TRUE;
67          }
68          printf("\n");
69          if (rtn != FALSE) {
70              if (mulBy10SomeTimes(&numA, &numA, DIGIT + MARGIN) == FALSE) {
71                  printf("ERROR:sqrtThree overflowG\n");
72                  clearByZero(a);
73                  rtn = FALSE;
74              } else if (divideByInverse(&numA, &numB, a) == FALSE) {
75                  printf("ERROR:sqrtThree overflowH\n");
76                  clearByZero(a);
77                  rtn = FALSE;
78              }
79          }
80          return rtn;
81      }
```

4.3.19 getLen

getLen 関数は表 24 のように宣言されている.

表 24: getLen 関数

関数名	getLen
概要	多倍長整数の桁数を求める
引数 1	const Number *a: 桁数を求めたい多倍長整数
戻り値	int: 桁数

リスト 54 のように getLen 関数を呼び出すと，リスト 55 のように動作する.

リスト 54: getLen 関数の呼び出し

```
1       #include "pi.c"
2       int main (void){
3           Number a;
4           int rtn;
5           setInt(&a, 12340);
```

```
6           rtn = getLen(&a);
7           dispNumber(&a);
8           printfの桁数は("%d\n", rtn);
9           return 0;
10      }
```

リスト 55: リスト 54 の実行結果

```
1       + 01 23 の桁数は 405
```

リスト 55 の実行結果から，getLen 関数は引数 a の桁数を求めていることがわかる．

getLen 関数は for 文を用いて，多倍長整数の最上位要素から順に 0 でない要素が見つかるまでカウントすることで桁数を求めている．

getLen 関数のソースコードをリスト 56 に示す．

リスト 56: getLen 関数

```
1       int getLen(const Number *a) {
2           int i;
3           if (isZero(a)) {
4               return 1;
5           }
6           for (i = KETA - 1; i >= 0; i--) {
7               if (a->n[i] != 0) {
8                   break;
9               }
10          }
11          return i * RADIX_LEN + (int)log10(a->n[i]) + 1;
12      }
```

5 検算

本課題では計算結果の正当性を証明するために，$\sqrt{3}$ の計算結果と円周率の計算結果を検算する．

5.1 $\sqrt{3}$ の検算

compareRootThree 関数を使って $\sqrt{3}$ の検算を行う．

この関数では https://ryo.blue/archive/4-%e3%83%ab%e3%83%bc%e3%83%883-10%e4%b8%87%e6%a1%81/ から取得した $\sqrt{3}$ の値を root3.txt に保存し，その値と計算した $\sqrt{3}$ の値を比較する．

compareRootThree 関数のソースコードをリスト 57 に示す．

リスト 57: compareRootThree 関数

```
1   int compareRootThree(const Number *a) {
2       FILE *fp;
3       int num;
4       int length;
5       char format[10];
6       length = getLen(a);
7       fp = fopen("multiple/text/root3.txt", "r");
8       for (int i = 0; i < length % 9; i++) {
9           format[i] = fgetc(fp);
10      }
11      num = atoi(format);
12      if (a->n[length / 9] != num) {
13          printf一致しません("\n");
14          printf("a->n[%d]: %lld, num: %d\n", length / 9, a->n[length /
                9], num);
15          fclose(fp);
16          return FALSE;
17      }
18      // 桁ずつ比較する9
19      for (int i = length / 9 - 1; i >= 0; i--) {
20          if (fgets(format, 10, fp) == NULL) {
21              fclose(fp);
22              printf("fgets error\n");
23              return FALSE;
24          }
25          num = atoi(format);
26          if (a->n[i] != num) {
27              printf一致しません("\n");
28              printf("a->n[%d]: %lld, num: %d\n", i, a->n[i], num);
29              fclose(fp);
30              return FALSE;
31          }
32      }
33      fclose(fp);
34      printf("same\n");
35      return 0;
36  }
```

5.2 円周率の検算

comparePi 関数を使って円周率の検算を行う.

この関数では https://miniwebtool.com/ja/first-n-digits-of-pi/?number=1000 から取得した円周率の値を pi.txt に保存し，その値と計算した円周率の値を比較する.

comparePi 関数のソースコードをリスト 58 に示す.

リスト 58: comparePi 関数

```
1   int comparePi(const Number *a) {
2       FILE *fp;
3       int num;
4       int length;
5       char format[10];
6       length = getLen(a);
7       fp = fopen("multiple/text/pi.txt", "r");
8       for (int i = 0; i < length % 9; i++) {
9           format[i] = fgetc(fp);
10      }
11      num = atoi(format);
12      if (a->n[length / 9] != num) {
13          printf一致しません("\n");
14          printf("a->n[%d]: %lld, num: %d\n", length / 9, a->n[length /
                9], num);
15          fclose(fp);
16          return FALSE;
17      }
18      // 桁ずつ比較する9
19      for (int i = length / 9 - 1; i >= 0; i--) {
20          if (fgets(format, 10, fp) == NULL) {
21              printf("fgets error\n");
22              fclose(fp);
23              return FALSE;
24          }
25          num = atoi(format);
26          if (a->n[i] != num) {
27              printf一致しません("\n");
28              printf("a->n[%d]: %lld, num: %d\n", i, a->n[i], num);
29              fclose(fp);
30              return FALSE;
31          }
32      }
33      fclose(fp);
```

```
34          printf("same\n");
35          return 0;
36      }
```

6 実行結果

本実験では，円周率の計算を行い，その結果を検算する．円周率計算には，式 (1) を用いて計算を行う．

基数を 10^9 とし，求める桁数を 8000 桁とした．

次の手順で円周率を求める．

1. $\sqrt{3}$ を求める．
2. $6\sqrt{3}$ を求める．
3. $n = 0$ として下の処理を繰り返す．
4. 2n + 1 に 3 をかけた値を a とする．
5. $6\sqrt{3}$ を a で割った値を tmp とする．
6. n が偶数の場合，tmp を x に加える．
7. n が奇数の場合，tmp を x から引く．
8. tmp が 0 の場合，次の手順に進む．そう出なければ手順 3 に戻る．
9. x を表示する．

式を実装する main 関数のソースコードをリスト 59 に示す．

リスト 59: main 関数

```
1       // の公式により円周率を求める sharp
2
3       #include <stdio.h>
4       #include <sys/time.h>
5       #include <time.h>
6
7       #include "../pi.h"
8
9       int main(int argc, char **argv) {
10          printf("%dLengthpi\n", DIGIT);
11          struct timeval tv;
12          double tstart, tend;
13          gettimeofday(&tv, NULL);
14          tstart = (double)tv.tv_sec + (double)tv.tv_usec * 1.e-6;
15
16          Number digitNum; // 求める桁数
```

```
17          Number constant; // 定数
18          Number x; // 答え
19          Number tmp; // 作業用多倍長整数
20          Number a, b; // 項 a,b
21          Number three;
22          int n;
23          int numA = 0;
24          n = 0;
25          setInt(&constant, 6);
26          setInt(&three, 3);
27          setInt(&b, 1);
28          // ルートを求める 3
29          sqrtThree(&digitNum);
30          printf("root3 = ");
31          dispNumber(&digitNum);
32          printf("\n");
33          printf("root3Len = %d\n", getLen(&digitNum));
34          compareRootThree(&digitNum);
35          // ルートを求める 63
36          multiple(&digitNum, &constant, &constant);
37          // 初期値
38          clearByZero(&x);
39          n = 0;
40          while (1) {
41              printf("\r%d times", n);
42              fflush(stdout);
43              // を求める a
44              numA = 2 * n + 1;
45              setInt(&a, numA);
46              // を求める b
47              multiple(&b, &three, &b);
48              // a * を求める b
49              if (multiple(&a, &b, &tmp) == FALSE) {
50                  printf("overflow\n");
51                  break;
52              }
53              // を求める x
54              if (divideByInverse(&constant, &tmp, &tmp) == FALSE) {
55                  printf("overflow\n");
56                  break;
57              }
58              if (isZero(&tmp)) {
```

```
59                  break;
60              }
61              if (n % 2 == 0) {
62                  if (add(&x, &tmp, &x) == FALSE) {
63                      printf("overflow\n");
64                      break;
65                  }
66              } else {
67                  if (sub(&x, &tmp, &x) == FALSE) {
68                      printf("overflow\n");
69                      break;
70                  }
71              }
72              n++;
73          }
74          printf("\n");
75          divBy10SomeTimes(&x, &x, MARGIN);
76          printf("pi = ");
77          dispNumber(&x);
78          printf("\n");
79          printf("piLen = %d\n", getLen(&x));
80          comparePi(&x);
81
82          gettimeofday(&tv, NULL);
83          tend = (double)tv.tv_sec + (double)tv.tv_usec * 1.e-6;
84          printf("time: %d h %d m %d s\n", (int)(tend - tstart) / 3600,
85                 (int)(tend - tstart) % 3600 / 60, (int)(tend - tstart) % 60);
86          printf("%fs\n", tend - tstart);
87          return 0;
88      }
```

このメイン関数を次のコマンドでコンパイルする．

```
gcc -Wall -O2 -o multiple/.exe/n2.exe multiple/main/n2.c multiple/pi.c -lm
```

使っている gcc のコンパイルオプションは次の通りである．

- -Wall: 警告を表示する．
- -O2: 最適化レベル 2 でコンパイルする．
- -o: 出力ファイル名を指定する．
- -lm: math.h の数学関数を使うためのオプション．

このコンパイルオプションでコンパイルしたファイルの実行結果をリスト 6 に示す．なお，実行

中の CPU 使用率は 20% 程度であった.

```
1   tsukada@tsukadaPC:/mnt/c/vscode$ gcc -Wall -O2 -o multiple/.exe/n2.exe
        multiple/main/n2.c multiple/pi.c -lm
2   tsukada@tsukadaPC:/mnt/c/vscode$ ./multiple/.exe/n2.exe
3   8000Lengthpi
4   root3 calculate 13
5   root3 = + 000000001 732050807 568877293 5274 ...//
6   root3Len = 8101
7   same
8   16969 times
9   pi = + 314159265 358979323 846264338 3279 ...//
10  piLen = 8001
11  same
12  time: 0 h 41 m 4 s
13  2464.599008s
```

それぞれの実行結果は次の通りである.

- 3 行目は求める桁数を表示している.
- 4 行目は $\sqrt{3}$ の計算をするためにループした回数を表示している.
- 5 行目は $\sqrt{3}$ の計算結果を表示している.
- 6 行目は $\sqrt{3}$ の計算結果の桁数を表示している.
- 7 行目は $\sqrt{3}$ の計算結果と検算した結果を表示している.
- 8 行目は円周率の計算をするためにループした回数を表示している.
- 9 行目は円周率の計算結果を表示している.
- 10 行目は円周率の計算結果の桁数を表示している.
- 11 行目は円周率の計算結果と検算した結果を表示している.
- 12 行目は計算にかかった時間を表示している.
- 13 行目は計算にかかった時間を秒で表示している.

実行結果によって 8000 桁の円周率が 41 分で求まり，正しい値であることがわかった.

7 付録

7.1 式 (1) の導出

式 (1) の導出は $\tan^{-1} x$ のテイラー展開を用いて行う．$\tan^{-1} x$ のテイラー展開は式 (18) で表される.

$$\tan^{-1} x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1} \tag{18}$$

この級数において，$x = 1 \,/\, \sqrt{3}$ とすると，式 (19) が得られる．

$$\tan^{-1}\frac{1}{\sqrt{3}} = \sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1}\left(\frac{1}{\sqrt{3}}\right)^{2n+1} \tag{19}$$

式 (19) に式 (20) を適用すると，式 (21) が得られる．

$$\left(\frac{1}{\sqrt{3}}\right)^{2n+1} = \frac{1}{3^{n+\frac{1}{2}}} \tag{20}$$

$$\tan^{-1}\frac{1}{\sqrt{3}} = \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)3^{n+\frac{1}{2}}} \tag{21}$$

ここで，$\tan^{-1}(1/\sqrt{3})$ は $\pi/6$ であるため，式 (22) が得られる．

$$\frac{\pi}{6} = \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)3^{n+\frac{1}{2}}} \tag{22}$$

式 (22) の両辺に 6 を掛けると，式 (23) が得られる．

$$\pi = 6\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)3^{n+\frac{1}{2}}} \tag{23}$$

形を見やすくするために分子と分母に $\sqrt{3}$ を掛けると，式 (1) が得られる．

7.2 式 4 の導出

式 (3) をニュートンラフソン法の式に適用するためにまず，$f(x)$ の導関数を求める．導関数を式 (24) に示す．

$$f'(x) = 1 - \frac{1}{x^2} \tag{24}$$

次に，式 (24) を式 (4) に代入すると (25) が得られる．

$$x_{n+1} = x_n - \frac{\frac{1}{x_n} - a}{-\frac{1}{x_n^2}} \tag{25}$$

整理すると，式 (26) が得られる．

$$x_{n+1} = x_n(2 - ax_n) \tag{26}$$

ここで収束誤差 e_n について考える．e_n は式 (27) で表される．

$$e_n = x_n - \frac{1}{a} \tag{27}$$

これを式 (26) に適用すると，式 (28) が得られる

$$x_{n+1} = (\frac{1}{a} + e_n)(2 - a(\frac{1}{a} + e_n)) \tag{28}$$

式 (28) を展開する．

$$x_{n+1} = (\frac{1}{a} + e_n)(2 - 1 - ae_n) \tag{29}$$

$$x_{n+1} = (\frac{1}{a} + e_n)(1 - ae_n) \tag{30}$$

$$x_{n+1} = \frac{1}{a} - \frac{1}{a}ae_n + e_n - ae_n^2 \tag{31}$$

$$x_{n+1} = \frac{1}{a} - ae_n^2 \tag{32}$$

式 (32) から，より誤差は n^2 ずつ小さくなることがわかる．より誤差を試行回数に対して線形に小さくするためには，収束速度を速める必要がある．まず，x_n を e_n を使って展開する．

$$x_n = \frac{1}{a} + e_n \tag{33}$$

$$ax_n = 1 + ae_n \tag{34}$$

$$ax_n - 1 = ae_n \tag{35}$$

両辺を 2 乗し，高次に展開すると，式 (36) が得られる．

$$(ax_n - 1)^2 = (1 + ae_n - 1)^2 = (ae_n)^2 \tag{36}$$

これを (32) に加えると，式 (37) が得られる．

$$x_{n+1} = x_n(2 - ax_n + a^2e_n^2) \tag{37}$$

(27) を (37) に適用すると，式 (38) が得られる．

$$x_{n+1} = x_n(2 - ax_n + a^2(x_n - \frac{1}{a})^2) \tag{38}$$

(38) を展開し，整理すると，式 (4) が得られる．

$$x_{n+1} = x_n(2 - ax_n + a^2x_n^2 - 2ax_n + 1) \tag{39}$$

$$x_{n+1} = x_n\left(2 - ax_n + (ax_n - 1)^2\right) \tag{40}$$

(40) を (36) と同じように x_n に e_n を代入すると，式 (41) が得られる．

$$x_{n+1} = \frac{1}{a} - ae_n^3 \tag{41}$$

式 (41) から，誤差は n^3 ずつ小さくなることがわかる．

7.3 pi.h

pi.h のソースコードをリスト 60 に示す.

リスト 60: pi.h

```
1   #define DIGIT 8000
2
3   #define RADIX 1000000000
4   #define RADIX_LEN 9
5   #define MARGIN 100
6
7   #define KETA (DIGIT + MARGIN) * 4 / RADIX_LEN + 1
8
9   #define PLUS 1
10  #define ZERO 0
11  #define MINUS -1
12
13  #define TRUE 1
14  #define FALSE 0
15
16  #define RADIX_T long long int
17
18  typedef struct NUMBER {
19      RADIX_T n[KETA]; // 各桁の変数
20      int sign; // 符号変数-1: 負, 0: 0, 1: 正
21  } Number;
22
23  void clearByZero(Number *);
24  void dispNumber(const Number *);
25  void copyNumber(Number *, const Number *);
26  void getAbs(const Number *, Number *);
27  int isZero(const Number *);
28  int mulBy10SomeTimes(const Number *, Number *, int);
29  void divBy10SomeTimes(const Number *, Number *, int);
30  int setInt(Number *, long);
31  int getInt(const Number *, int *);
32  int setSign(Number *, int);
33  int getSign(const Number *);
34  int numComp(const Number *, const Number *);
35  int numCompWithInt(const Number *, int);
36  int add(const Number *, const Number *, Number *);
37  int sub(const Number *, const Number *, Number *);
38  int multiple(const Number *, const Number *, Number *);
```

```
39      int inverse3(const Number *, Number *);
40      int divideByInverse(const Number *, const Number *, Number *);
41      int sqrtThree(Number *);
42      int getLen(const Number *);
43      int comparePi(const Number *);
44      int compareRootThree(const Number *);
```

7.4 pi.c

pi.c のソースコードをリスト??に示す.

```
1       #include "pi.h"
2
3       #include <limits.h>
4       #include <math.h>
5       #include <stdio.h>
6       #include <stdlib.h>
7
8       /// @brief 符号を設定する
9       /// @param a 符号を設定する構造体
10      /// @param s 1: 正, 0: 0, -1: 負
11      /// @return 成功: 0, エラー: -1
12      int setSign(Number *a, int s) {
13          switch (s) {
14              case PLUS:
15                  a->sign = PLUS;
16                  break;
17              case ZERO:
18                  a->sign = ZERO;
19                  break;
20              case MINUS:
21                  a->sign = MINUS;
22                  break;
23              default:
24                  return FALSE;
25          }
26          return TRUE;
27      }
28
29      /// @brief 符号を取得する
30      /// @param a 符号を取得する構造体
31      /// @return 1: 正, 0: 0, -1: 負
```

```
32      int getSign(const Number *a) { return a->sign; }
33
34      /// @brief 構造体の中身をで初期化する 0
35      /// @param a 初期化する構造体
36      void clearByZero(Number *a) {
37          for (int i = 0; i < KETA; i++) {
38              a->n[i] = 0;
39          }
40          setSign(a, ZERO);
41      }
42
43      /// @brief 先頭のを抜いて表示する 0
44      /// @param a 表示する構造体
45      void dispNumber(const Number *a) {
46          int i;
47          char format[8];
48          sprintf(format, " %%0%dd", RADIX_LEN); // format = " %02d"
49          switch (getSign(a)) {
50              case PLUS:
51                  printf("+");
52                  break;
53              case ZERO:
54                  printf("+ 0");
55                  return;
56              case MINUS:
57                  printf("-");
58                  break;
59          }
60          for (i = KETA - 1; i >= 0; i--) {
61              if (a->n[i] > 0) {
62                  break;
63              }
64          }
65          for (; i >= 0; i--) {
66              printf(format, a->n[i]);
67          }
68      }
69
70      /// @brief 値をコピーする
71      /// @param a コピー先
72      /// @param b コピー元
73      void copyNumber(Number *a, const Number *b) { *a = *b; }
```

```
74
75      /// @brief 絶対値を求める
76      /// @param a 絶対値を求める構造体
77      /// @param b 絶対値を代入する構造体
78      void getAbs(const Number *a, Number *b) {
79          copyNumber(b, a);
80          if (getSign(a) == ZERO) {
81              setSign(b, ZERO);
82          } else {
83              setSign(b, PLUS);
84          }
85      }
86
87      /// @brief がかどうかを判定する a0
88      /// @param a 判定する構造体
89      /// @return true: 0, false: でない 0
90      int isZero(const Number *a) {
91          if (getSign(a) == ZERO) {
92              return TRUE;
93          } else {
94              return FALSE;
95          }
96      }
97
98      /// @brief を何回か倍してに代入する a10b
99      /// @param a 倍する構造体 10
100     /// @param b 倍した値を代入する構造体 10
101     /// @param k 倍する回数 10
102     /// @return 0: 正常終了, -1: オーバーフロー
103     int mulBy10SomeTimes(const Number *a, Number *b, int k) {
104         int rtn = -2;
105         int i, j;
106         copyNumber(b, a);
107         if (isZero(a)) {
108             rtn = TRUE;
109         } else if (k == 0) {
110             rtn = TRUE;
111         } else {
112             int digit;
113             RADIX_T carry;
114             digit = k / RADIX_LEN;
115             int length = getLen(b);
```

```
116             j = 0;
117             i = KETA - 1;
118             while (1) {
119                 if (digit <= j) {
120                     break;
121                 } else if (b->n[i] != 0) {
122                     printf("mulBy10SomeTimes: overflow: -1A\n");
123                     rtn = FALSE;
124                     break;
125                 }
126                 j++;
127                 i--;
128             }
129             if (rtn == FALSE) {
130                 printf("mulBy10SomeTimes: overflow: -1B\n");
131             } else if (b->n[i] /
132                             (int)pow(10, (RADIX_LEN - (k % RADIX_LEN))) !=
133                         0) {
134                 printf("mulBy10SomeTimes: overflow\n");
135                 rtn = FALSE;
136             } else {
137                 if (digit != 0) {
138                     for (i = length / RADIX_LEN; i >= 0; i--) {
139                         b->n[i + digit] = b->n[i];
140                     }
141                     for (i = 0; i < digit; i++) {
142                         b->n[i] = 0;
143                     }
144                 }
145                 length += digit * RADIX_LEN;
146                 carry = 0;
147                 for (i = 0; i < length / RADIX_LEN + 2; i++) {
148                     b->n[i] *= (int)pow(10, (k % RADIX_LEN));
149                     b->n[i] += carry;
150                     if (b->n[i] >= RADIX) {
151                         carry = b->n[i] / RADIX;
152                         b->n[i] %= RADIX;
153                     } else {
154                         carry = 0;
155                     }
156                 }
157                 rtn = TRUE;
```

```
158             }
159         }
160         return rtn;
161     }
162 
163     /// @brief を何回かで割ってに代入する a10b
164     /// @param a で割る構造体 10
165     /// @param b で割った値を代入する構造体 10
166     /// @param k で割る回数 10
167     /// @return 剰余
168     void divBy10SomeTimes(const Number *a, Number *b, int k) {
169         int i;
170         int digit;
171         int carry;
172         digit = k / RADIX_LEN;
173         copyNumber(b, a);
174         b->n[0] -= b->n[0] % (int)pow(10, k % RADIX_LEN);
175         b->n[0] /= (int)pow(10, k % RADIX_LEN);
176         for (i = 1; i < KETA; i++) {
177             carry = b->n[i] % (int)pow(10, k % RADIX_LEN);
178             b->n[i - 1] +=
179                 carry * (int)pow(10, RADIX_LEN - (k % RADIX_LEN));
180             b->n[i] -= carry;
181             b->n[i] /= (int)pow(10, k % RADIX_LEN);
182         }
183         if (digit > 0) {
184             for (i = 0; i < KETA - digit; i++) {
185                 b->n[i] = b->n[i + digit];
186             }
187             for (i = KETA - digit; i < KETA; i++) {
188                 b->n[i] = 0;
189             }
190         }
191         return;
192     }
193 
194     /// @brief 型の値を構造体に代入する int
195     /// @param a 代入する構造体
196     /// @param x 代入する値
197     /// @return 成功: 0, エラー (overflow): -1
198     int setInt(Number *a, long x) {
199         clearByZero(a);
```

```
200         if (x < 0) {
201             setSign(a, MINUS);
202             x *= -1;
203         } else if (x == 0) {
204             setSign(a, ZERO);
205         } else {
206             setSign(a, PLUS);
207         }
208         a->n[0] = x % RADIX;
209         return TRUE;
210     }
211 
212     /// @brief 構造体の中身を型に変換する int
213     /// @param a 値を読み取る構造体
214     /// @param x 型に変換した値を代入する変数 int
215     /// @return 成功: 0, エラー (overflow): -1
216     int getInt(const Number *a, int *x) {
217         int rtn;
218         if (isZero(a)) {
219             *x = 0;
220             rtn = TRUE;
221         } else if (getLen(a) > RADIX_LEN) {
222             rtn = FALSE;
223         } else {
224             *x = a->n[0];
225             *x += a->n[1] * RADIX;
226             if (getSign(a) == MINUS) {
227                 *x *= -1;
228             }
229             rtn = TRUE;
230         }
231         return rtn;
232     }
233 
234     /// @brief つの多倍長整数を比較する 2
235     /// @param a 比較する構造体
236     /// @param b 比較する構造体
237     /// @return 1: a > b, 0: a = b, -1: a < b
238     int numComp(const Number *a, const Number *b) {
239         int rtn = 0;
240         switch (getSign(a) * 3 + getSign(b)) {
241             case -4: // とが負 ab
```

```
242                 if (getLen(a) > getLen(b)) {
243                     rtn = -1;
244                 } else if (getLen(a) < getLen(b)) {
245                     rtn = 1;
246                 }
247                 for (int i = KETA - 1; i >= 0; i--) {
248                     if (a->n[i] < b->n[i]) {
249                         rtn = 1;
250                         break;
251                     } else if (a->n[i] > b->n[i]) {
252                         rtn = -1;
253                         break;
254                     }
255                 }
256                 break;
257             case 4: // とが正 ab
258                 if (getLen(a) > getLen(b)) {
259                     rtn = 1;
260                 } else if (getLen(a) < getLen(b)) {
261                     rtn = -1;
262                 }
263                 for (int i = KETA - 1; i >= 0; i--) {
264                     if (a->n[i] > b->n[i]) {
265                         rtn = 1;
266                         break;
267                     } else if (a->n[i] < b->n[i]) {
268                         rtn = -1;
269                         break;
270                     }
271                 }
272                 break;
273             case -3: // が負でが ab0
274             case -2: // が負でが正 ab
275             case 1: // がでが正 a0b
276                 rtn = -1;
277                 break;
278             case -1: // がでが負 a0b
279             case 2: // が正でが負 ab
280             case 3: // が正でが ab0
281                 rtn = 1;
282                 break;
283             case 0: // とが ab0
```

```
284                 break;
285         }
286         return rtn;
287     }
288 
289     /// @brief 多倍長整数と型の値を比較する int
290     /// @param a 比較する構造体
291     /// @param x 比較する値
292     /// @return 1: a > x, 0: a = x, -1: a < x
293     int numCompWithInt(const Number *a, int x) {
294         int num, rtn;
295         if (getInt(a, &num) == FALSE) {
296             rtn = 1;
297         } else if (num > x) {
298             rtn = 1;
299         } else if (num < x) {
300             rtn = -1;
301         } else {
302             rtn = 0;
303         }
304         return rtn;
305     }
306 
307     /// @brief つの多倍長整数を加算する同じ変数を関数内に入れてはいけない 2()
308     /// @param a 加算する構造体
309     /// @param b 加算する構造体
310     /// @param c 加算した値を代入する構造体
311     /// @return オーバーフロー: -1, 正常終了: 0
312     int add(const Number *a, const Number *b, Number *c) {
313         Number A, B;
314         RADIX_T d;
315         int i, rtn;
316         int e = 0;
317         rtn = -2;
318         int caseNum = getSign(a) * 3 + getSign(b);
319         switch (caseNum) {
320             case -4: // とが負 ab
321             case 4: // とが正 ab
322                 // clearByZero(c);
323                 if (caseNum == -4) {
324                     getAbs(a, &A);
325                     getAbs(b, &B);
```

```
326                     setSign(c, MINUS);
327                 } else {
328                     copyNumber(&A, a);
329                     copyNumber(&B, b);
330                     setSign(c, PLUS);
331                 }
332                 for (i = 0; i < KETA; i++) {
333                     d = A.n[i] + B.n[i] + e;
334                     e = 0;
335                     if (d >= RADIX) {
336                         d -= RADIX;
337                         e = 1;
338                     } else {
339                         e = 0;
340                     }
341                     c->n[i] = d;
342                 }
343                 if (e == 1) {
344                     rtn = FALSE;
345                 } else {
346                     rtn = TRUE;
347                 }
348                 break;
349             case -3: // が負でが ab0
350             case 3: // が正でが ab0
351                 copyNumber(c, a);
352                 rtn = TRUE;
353                 break;
354             case -2: // が負でが正 ab
355                 getAbs(a, &A);
356                 rtn = sub(b, &A, c);
357                 break;
358             case -1: // がでが負 a0b
359             case 1: // がでが正 a0b
360                 copyNumber(c, b);
361                 rtn = TRUE;
362                 break;
363             case 0: // とが ab0
364                 clearByZero(c);
365                 rtn = TRUE;
366                 break;
367             case 2: // が正でが負 ab
```

```
368                 getAbs(b, &B);
369                 rtn = sub(a, &B, c);
370                 break;
371         }
372         return rtn;
373     }
374 
375     /// @brief つの多倍長整数を減算する同じ変数を関数内に入れてはいけない2()
376     /// @param a 減算する構造体
377     /// @param b 減算する構造体
378     /// @param c 減算した値を代入する構造体
379     /// @return オーバーフロー: -1, 正常終了: 0
380     int sub(const Number *a, const Number *b, Number *c) {
381         Number A, B;
382         copyNumber(&A, a);
383         copyNumber(&B, b);
384         int i, e, num, rtn;
385         int caseNum = getSign(&A) * 3 + getSign(&B);
386         Number numA, numB;
387         rtn = -2;
388         switch (caseNum) {
389             case -4: // とが負ab
390             case 4: // とが正ab
391                 if (caseNum == -4) {
392                     getAbs(&A, &numB);
393                     getAbs(&B, &numA);
394                 } else {
395                     copyNumber(&numA, &A);
396                     copyNumber(&numB, &B);
397                 }
398                 e = 0;
399                 switch (numComp(&A, &B)) {
400                     case 1:
401                         for (i = 0; i < KETA; i++) {
402                             num = numA.n[i] - e;
403                             if (num < numB.n[i]) {
404                                 c->n[i] = num + RADIX - numB.n[i];
405                                 e = 1;
406                             } else {
407                                 c->n[i] = num - numB.n[i];
408                                 e = 0;
409                             }
```

```
410                         setSign(c, PLUS);
411                     }
412                     break;
413                 case -1:
414                     for (i = 0; i < KETA; i++) {
415                         num = numB.n[i] - e;
416                         if (num < numA.n[i]) {
417                             c->n[i] = num + RADIX - numA.n[i];
418                             e = 1;
419                         } else {
420                             c->n[i] = num - numA.n[i];
421                             e = 0;
422                         }
423                     }
424                     setSign(c, MINUS);
425                     break;
426                 case 0:
427                     clearByZero(c);
428                     setSign(c, ZERO);
429                     break;
430             }
431             if (e > 0) {
432                 rtn = FALSE;
433             } else {
434                 rtn = TRUE;
435             }
436             break;
437         case -3: // が負でが ab0
438         case 3: // が正でが ab0
439             copyNumber(c, &A);
440             rtn = TRUE;
441             break;
442         case -2: // が負でが正 ab
443             getAbs(&A, &A);
444             rtn = add(&A, &B, c);
445             setSign(c, MINUS);
446             break;
447         case -1: // がでが負 a0b
448             copyNumber(c, &B);
449             setSign(c, PLUS);
450             rtn = TRUE;
451             break;
```

```
452             case 1: // がでが正 a0b
453                 copyNumber(c, &B);
454                 setSign(c, MINUS);
455                 rtn = TRUE;
456                 break;
457             case 0: // とが ab0
458                 clearByZero(c);
459                 rtn = TRUE;
460                 break;
461             case 2: // が正でが負 ab
462                 getAbs(&B, &B);
463                 rtn = add(&A, &B, c);
464                 break;
465         }
466         return rtn;
467     }
468 
469     /// @brief つの多倍長整数を掛け算する 2
470     /// @param a 掛け算する構造体
471     /// @param b 掛け算する構造体
472     /// @param c 掛け算した値を代入する構造体
473     /// @return オーバーフロー: -1, 正常終了: 0
474     int multiple(const Number *a, const Number *b, Number *c) {
475         int rtn = -2;
476         int signA, signB;
477         signA = getSign(a);
478         signB = getSign(b);
479         if (signA == ZERO || signB == ZERO) {
480             clearByZero(c);
481             rtn = TRUE;
482         } else {
483             RADIX_T tmp;
484             Number A, B;
485             getAbs(a, &A);
486             getAbs(b, &B);
487             clearByZero(c);
488             int ALength = getLen(&A);
489             int BLength = getLen(&B);
490             for (int i = 0; i < ALength / 9 + 1; i++) {
491                 for (int j = 0; j < BLength / 9 + 1; j++) {
492                     tmp = A.n[i] * B.n[j];
493                     if (tmp == 0) {
```

```
494                     continue;
495                 }
496                 c->n[i + j] += tmp % RADIX;
497                 c->n[i + j + 1] += tmp / RADIX;
498                 c->n[i + j + 1] += c->n[i + j] / RADIX;
499                 c->n[i + j] %= RADIX;
500             }
501         }
502         switch (signA * 3 + signB) {
503             case -4: // とが負 ab
504             case 4: // とが正 ab
505                 setSign(c, PLUS);
506                 break;
507             case -2: // が負でが正 ab
508             case 2: // が正でが負 ab
509                 setSign(c, MINUS);
510                 break;
511                 // case 3:,case 1:,case 0:,case -1:,case
512                 // は最初で判定しているのでここには来ない-3:
513         }
514         rtn = TRUE;
515     }
516     return rtn;
517 }
518
519 /// @brief 多倍長整数の逆数を求める次収束(3)
520 /// @param a 逆数を求める構造体
521 /// @param b 逆数を代入する構造体
522 /// @return ゼロ除算: -1, 正常終了: 余裕を持っている桁数
523 int inverse3(const Number *a, Number *b) {
524     int rtn = TRUE;
525     if (isZero(a)) {
526         clearByZero(b);
527         rtn = TRUE;
528     } else if (numCompWithInt(a, 1) == 0) {
529         copyNumber(b, a);
530         mulBy10SomeTimes(b, b, DIGIT + MARGIN);
531         rtn = TRUE;
532     } else {
533         Number x0; // ひとつ前のx
534         Number A; // 逆数を求める数
535         Number tmp; // 作業用変数
```

```
536             Number h;
537             Number g; // 逆数の誤差
538             Number bigOne;
539             int sigDigs = DIGIT + MARGIN;
540             int margin = 0;
541             int length = getLen(a);
542             if(length >= sigDigs + margin) {
543                 margin += length;
544             } else {
545                 margin += sigDigs;
546             }
547             getAbs(a, &A);
548             setInt(&bigOne, 1);
549             setInt(b, 2);
550             mulBy10SomeTimes(b, b, sigDigs + margin - length); // 初期値
551             mulBy10SomeTimes(&bigOne, &bigOne, sigDigs + margin);
552             while (1) {
553                 copyNumber(&x0, b); // ひとつ前のx
554                 if (multiple(&A, &x0, &tmp) == -1) {
555                     printf("ERROR:inverse2 overflow\n");
556                     clearByZero(b);
557                     rtn = FALSE;
558                     break;
559                 }
560                 sub(&bigOne, &tmp, &h);
561                 multiple(&h, &h, &tmp);
562                 divBy10SomeTimes(&tmp, &tmp, sigDigs + margin);
563                 add(&tmp, &h, &tmp);
564                 add(&tmp, &bigOne, &tmp);
565                 if (multiple(&x0, &tmp, b) == -1) {
566                     printf("ERROR:inverse2 overflow\n");
567                     clearByZero(b);
568                     rtn = FALSE;
569                     break;
570                 }
571                 divBy10SomeTimes(b, b, sigDigs + margin);
572                 sub(b, &x0, &g);
573                 if (getLen(&g) < 2) {
574                     rtn = margin;
575                     break;
576                 }
577             }
```

```
578             setSign(b, getSign(a));
579         }
580         return rtn;
581     }
582 
583     /// @brief 多倍長整数を割り算する逆数使用()
584     /// @param a 被除数
585     /// @param b 除数
586     /// @param c 商を代入する構造体
587     /// @return ゼロ除算: -1, 正常終了: 0
588     int divideByInverse(const Number *a, const Number *b, Number *c) {
589         Number A, B;
590         getAbs(a, &A);
591         getAbs(b, &B);
592         if (numComp(&A, &B) == -1) {
593             clearByZero(c);
594             return 0;
595         }
596         int rtn;
597         int cSign;
598         Number inv;
599         int margin = 0;
600         switch ((getSign(a) < 0) * 2 + (getSign(b) < 0)) {
601             case 0: // が正でが正 AB
602             case 3: // が負でが負 AB
603                 cSign = 1;
604                 break;
605             case 1: // が正でが負 AB
606             case 2: // が負でが正 AB
607                 cSign = -1;
608                 break;
609         }
610         margin = inverse3(&B, &inv);
611         if (margin == FALSE) {
612             printf("ERROR:divideByInverse errorA\n");
613             rtn = FALSE;
614         } else {
615             if (multiple(&A, &inv, c) == -1) {
616                 printf("ERROR:divideByInverse errorB\n");
617                 rtn = FALSE;
618             } else {
619                 divBy10SomeTimes(c, c, DIGIT + MARGIN + margin);
```

```
620                 rtn = TRUE;
621             }
622         }
623         setSign(c, cSign);
624         return rtn;
625     }
626 
627     /// @brief の平方根を求める 3
628     /// @param a の平方根を代入する構造体 3
629     /// @return エラー: -1, 正常終了: 0
630     int sqrtThree(Number *a) {
631         clearByZero(a);
632         int rtn;
633         Number numA, numB;
634         Number constant;
635         Number numA0, numB0;
636         Number two;
637         int digSig = DIGIT + MARGIN;
638         int i;
639         int j = 0;
640         i = 1;
641         // がの何乗かを求める digSig2
642         while (1) {
643             if (digSig < i) {
644                 break;
645             }
646             i *= 2;
647             j++;
648         }
649 
650         setInt(&constant, 3);
651         setInt(&two, 2);
652         setInt(&numA, 1);
653         copyNumber(&numB, &numA);
654         for (i = 0; i < j + 1; i++) {
655             printf("\rroot3 calculate %d", i);
656             fflush(stdout);
657             copyNumber(&numA0, &numA);
658             copyNumber(&numB0, &numB);
659             if (multiple(&numA0, &numA0, &numA) == FALSE) {
660                 printf("ERROR:sqrtThree overflowA\n");
661                 clearByZero(a);
```

```
662                 rtn = FALSE;
663                 break;
664             }
665             if (multiple(&numB0, &numB0, &numB) == FALSE) {
666                 printf("ERROR:sqrtThree overflowB\n");
667                 clearByZero(a);
668                 rtn = FALSE;
669                 break;
670             }
671             if (multiple(&numB, &constant, &numB) == FALSE) {
672                 printf("ERROR:sqrtThree overflowC\n");
673                 clearByZero(a);
674                 rtn = FALSE;
675                 break;
676             }
677             if (add(&numA, &numB, &numA) == FALSE) {
678                 printf("ERROR:sqrtThree overflowD\n");
679                 clearByZero(a);
680                 rtn = FALSE;
681                 break;
682             }
683             if (multiple(&numA0, &numB0, &numB) == FALSE) {
684                 printf("ERROR:sqrtThree overflowE\n");
685                 clearByZero(a);
686                 rtn = FALSE;
687                 break;
688             }
689             if (multiple(&numB, &two, &numB) == FALSE) {
690                 printf("ERROR:sqrtThree overflowF\n");
691                 clearByZero(a);
692                 rtn = FALSE;
693                 break;
694             }
695             rtn = TRUE;
696         }
697         printf("\n");
698         if (rtn != FALSE) {
699             if (mulBy10SomeTimes(&numA, &numA, DIGIT + MARGIN) == FALSE) {
700                 printf("ERROR:sqrtThree overflowG\n");
701                 clearByZero(a);
702                 rtn = FALSE;
703             } else if (divideByInverse(&numA, &numB, a) == FALSE) {
```

```
704                 printf("ERROR:sqrtThree overflowH\n");
705                 clearByZero(a);
706                 rtn = FALSE;
707             }
708         }
709         return rtn;
710     }
711 
712     /// @brief 多倍長整数の桁数を求める
713     /// @param a 桁数を求める構造体
714     /// @return 桁数
715     int getLen(const Number *a) {
716         int i;
717         if (isZero(a)) {
718             return 1;
719         }
720         for (i = KETA - 1; i >= 0; i--) {
721             if (a->n[i] != 0) {
722                 break;
723             }
724         }
725         return i * RADIX_LEN + (int)log10(a->n[i]) + 1;
726     }
727 
728     /// @brief 多倍長整数がと等しいか判定する pi
729     /// @param a 判定する構造体
730     /// @return 等しい: 0, 等しくない: -1
731     int comparePi(const Number *a) {
732         FILE *fp;
733         int num;
734         int length;
735         char format[10];
736         length = getLen(a);
737         fp = fopen("multiple/text/pi.txt", "r");
738         for (int i = 0; i < length % 9; i++) {
739             format[i] = fgetc(fp);
740         }
741         num = atoi(format);
742         if (a->n[length / 9] != num) {
743             printf一致しません("\n");
744             printf("a->n[%d]: %lld, num: %d\n", length / 9, a->n[length /
                    9], num);
```

```
745             fclose(fp);
746             return FALSE;
747         }
748         // 桁ずつ比較する 9
749         for (int i = length / 9 - 1; i >= 0; i--) {
750             if (fgets(format, 10, fp) == NULL) {
751                 printf("fgets error\n");
752                 fclose(fp);
753                 return FALSE;
754             }
755             num = atoi(format);
756             if (a->n[i] != num) {
757                 printf一致しません ("\n");
758                 printf("a->n[%d]: %lld, num: %d\n", i, a->n[i], num);
759                 fclose(fp);
760                 return FALSE;
761             }
762         }
763         fclose(fp);
764         printf("same\n");
765         return 0;
766     }
767 
768     /// @brief 多倍長整数がと等しいか判定する root3
769     /// @param a 判定する構造体
770     /// @return 等しい: 0, 等しくない: -1
771     int compareRootThree(const Number *a) {
772         FILE *fp;
773         int num;
774         int length;
775         char format[10];
776         length = getLen(a);
777         fp = fopen("multiple/text/root3.txt", "r");
778         for (int i = 0; i < length % 9; i++) {
779             format[i] = fgetc(fp);
780         }
781         num = atoi(format);
782         if (a->n[length / 9] != num) {
783             printf一致しません ("\n");
784             printf("a->n[%d]: %lld, num: %d\n", length / 9, a->n[length /
                    9], num);
785             fclose(fp);
```

```
786             return FALSE;
787         }
788         // 桁ずつ比較する 9
789         for (int i = length / 9 - 1; i >= 0; i--) {
790             if (fgets(format, 10, fp) == NULL) {
791                 fclose(fp);
792                 printf("fgets error\n");
793                 return FALSE;
794             }
795             num = atoi(format);
796             if (a->n[i] != num) {
797                 printf一致しません ("\n");
798                 printf("a->n[%d]: %lld, num: %d\n", i, a->n[i], num);
799                 fclose(fp);
800                 return FALSE;
801             }
802         }
803         fclose(fp);
804         printf("same\n");
805         return 0;
806     }
```