アルゴリズムとデータ構造 レポート

多倍長演算

提出日　令和２年２月３日

組番号　407
学籍番号　17406

氏名　金澤雄大

1　計算内容

授業で作成した多倍長演算のプログラムを用いて, ゼータ関数 $\zeta(s)$ の $s = 4$ の場合から π を 1000 桁程度計算する. しかし, 1000 桁を求めるには非常に時間がかかるため, 時間が許す限り可能な桁を求めることにする.

1.1　開発環境

開発環境を表 1 に示す.

表 1: 開発環境

CPU	Intel Core i7-6500 2.50Ghz
メモリ	16.0GB DDR3
OS	Windows 10 Pro
コンパイラ	GCC 7.4.0

2　プログラムの説明

本章では計算に用いた定数, および構造体, および関数の説明に述べる.

2.1　NUMBER 構造体と定数

多倍長変数の桁数 KETA と基数 RADIX をオブジェクト形式マクロで定義する. 桁数は多倍長演算を行うときに必要な桁数を設定する. 基数は 10000 とする.

多倍長変数を NUMBER 構造体として定義する. 各桁の値を保存する int 型の配列 n と, 多倍長変数の符号 (正, または負) を表す int 型の変数 sign を構造体のメンバとする. 多倍長変数の配列は長さが KETA で,RADIX を 10000 としているので各要素が 0〜9999 までの値を保持する. 配列の i 番目の要素を a_i と書くことにすると, 多倍長変数の配列は a_0 が 0〜1000 の位であり, a_{KETA-1} が最上位桁である. 多倍長変数の符号は表 2 に示すように,1 のとき正,-1 のとき負として扱う. 多倍長変数の値が 0 のとき符号が正であることに注意する必要がある.-0 と $+0$ を混同して扱うとエラーの原因になるから, 本プログラムでは一律して $+0$ で扱う.

表 2: 符号変数の値と多倍長変数の値の符号

符号変数の値	多倍長変数の値の符号
1	正または 0
-1	負

リスト 1 に桁数 KETA, 基数 RADIX, および NUMBER 構造体のプログラムを示す. リスト 1 では桁数 KETA を 150 桁, 基数 RADIX を 10000 と定義している. また多倍長変数として NUMBER 構造体を定義している.

リスト 1: 定数と NUMBER 構造体のソースコード

```
1 #define KETA 150 //桁数
2 #define RADIX 10000 //基数
3 struct NUMBER{
```

```
4     int n[KETA];
5     int sign;
6 };
```

2.2 isZero 関数

多倍長変数が 0 であることを確認する関数として isZero 関数を作成する. 表 3 に isZero 関数の機能, および引数, および返り値を示す.isZero 関数では第一引数で渡された多倍長変数 a が 0 であるかを確認し,0 であれば 0 を返す.a が 0 でないときは −1 を返す.

表 3: isZero 関数の概要

機能	多倍長変数が 0 であるかを調べる関数.
引数	struct NUMBER *a
返り値の型	int
返り値	多倍長変数 a が 0 のとき 0,0 でなければ-1

リスト 2 に isZero 関数のソースコードを示す.isZero 関数の内部ではリスト 2 の 4〜8 行目に示すように for 文を用いて多倍長変数の配列を 1 桁ずつ 0 であることを確認している.0 でない桁が 1 つでも見つかればその時点で for 文から脱出し,-1 を返す. 全ての桁が 0 であれば 0 を返す.

リスト 2: isZero 関数のソースコード

```
1  //aが0かをチェック
2  //0->return 0 else->return -1
3  int isZero(struct NUMBER *a){
4      int i;
5      for(i=0; i<KETA; i++){
6          if(a->n[i]!=0){
7              return -1;
8          }
9      }
10     return 0;
11 }
```

2.3 setSign 関数

多倍長変数に符号をセットする関数として setSign 関数を作成する. 直接符号を設定することができるが, セッターを用いることで符号がおかしな値 (例えば 999,-1234) に設定されるのを防ぐことができる. 表 4 に setSign 関数の機能, および引数, および返り値を示す. 第一引数で与えた多倍長変数 a の符号を第二引数で与えた符号変数 s の値に対応する多倍長変数の符号に設定することで setSign 関数の機能を実現する.

表 4: setSign 関数の概要

機能	多倍長変数の符号を設定する関数.
引数	struct NUMBER *a,int s
返り値の型	int
返り値	成功 (0), 失敗 (-1).

リスト 3 に setSign 関数のソースコードを示す.setSign 関数の内部では最初に, 第一引数の多倍長変数が 0 であるか確認する. もし a が 0 で負にセットしようとしているとき,−0 が作成されてしまうのでエラーとして −1 を返して処理を終了する. リスト 3 では 5〜8 行目がこの処理にあたる部分である. 次に s の値によっ

て場合分けをして符号をセットする処理を行う.$s = 1$ のとき a の符号を正に,$s = -1$ のとき a の符号を負にセットして処理を終了する.s がその他の値であるときエラーとして -1 を返して処理を終了する. リスト 3 では 10〜18 行目がこの処理にあたる部分である.

リスト 3: setSign 関数のソースコード

```
1  //aに符号をセット
2  //aが0のときに-をセットしようとするとreturn -1
3  int setSign(struct NUMBER *a,int s){
4
5      if((isZero(a)==0)&&(s==-1)){
6          printf("aは0です\n");
7          return -1;
8      }
9
10     if(s==1){
11         a->sign = 1;
12         return 0;
13     }else if(s==-1){
14         a->sign = -1;
15         return 0;
16     }else{
17         return -1;
18     }
19 }
```

2.4 getSign 関数

多倍長変数の符号を取り出す関数として getSign 関数を作成する.setSign 関数同様にゲッターを設けることでおかしな入力が混入し, 符号に異常が発生することを防ぐ. 表 5 に getSign 関数の機能, および引数, および返り値を示す. getSign 関数は第一引数で渡された多倍長変数の符号を int 型の返り値によって取得する関数である.

表 5: getSign 関数の概要

機能	多倍長変数の符号を取得する関数.
引数	struct NUMBER *a
返り値の型	int
返り値	a が正または 0(1),a が負 (-1).

リスト 4 に getSign 関数のソースコードを示す.getSign 関数の内部では多倍長変数 a の符号変数 s を返す処理を行っている.

リスト 4: getSign 関数のソースコード

```
1  //aの符号を取得
2  int getSign(struct NUMBER *a){
3      if(a->sign==1){
4          return 1;
5      }else{
6          return -1;
7      }
8  }
```

2.5 clearByZero 関数

多倍長変数をゼロクリアする関数として clearByZero 関数を作成する. 表 6 に clearByZero 関数の機能, および引数, および返り値を示す. 表 6 に示すように clearByZero 関数は第一引数で渡す多倍長変数をゼロクリアする. 失敗することはないと仮定して, 返り値は void とする.

表 6: clearByZero 関数の概要

機能	多倍長変数をゼロクリアし, 符号を正に設定する関数.
引数	struct NUMBER *a
返り値の型	void
返り値	-

リスト 5 に clearByZero 関数のソースコードを示す.clearByZero 関数の内部では, 最初に for 文を用いて多倍長変数の値を保存している配列の要素に 0 を代入している (リスト 5 の 5〜7 行目). 次に setSign 関数を用いて符号を正に設定して処理を終了している.

リスト 5: clearByZero 関数のソースコード

```
1  //aをゼロクリア
2  void clearByZero(struct NUMBER *a){
3      int i;
4
5      for(i=0; i<KETA;i++){
6          a->n[i] = 0;
7      }
8
9      setSign(a,1);
10 }
```

2.6 getKeta 関数

多倍長変数の桁数を取得する関数として,getKeta 関数を作成する. 表 7 に getKeta 関数の機能, および引数, および返り値を示す. getKeta 関数は第一引数で渡された多倍長変数の桁数を返り値とする関数である.

表 7: getKeta 関数の概要

機能	多倍長変数の桁数を取得する関数.
引数	struct NUMBER *a
返り値の型	int
返り値	多倍長変数 a の桁数.

リスト 6 に getKeta 関数のソースコードを示す.getKeta 関数の内部では, 最初に配列の最上位から 1 桁ずつ 0 でない桁を探す.0 でない桁が見つからない場合は a が 0 桁であるので 0 を返す. 0 でない桁が見つかった場合について説明する.i 桁で 0 でない桁を見つけたとする.0 でない桁の値は 1,2,3,4 桁のいずれかである. この桁数を t とすると全体の桁数は $i \times 4 - (4 - t)$ 桁であるからこの値を返す. リスト 6 のソースコードでは $4 - t$ を j として求めている.

リスト 6: getKeta 関数のソースコード

```
1  // aの桁数を返す
2  int getKeta(struct NUMBER *a){
3      int i,j,base,temp,acopy;
4      base = RADIX/10;
5      for(i=KETA-1; i>=0; i--){
6          if(a->n[i]!=0){
7              //printf("a->n[%d] = %4d\n",i,a->n[i]);
8              acopy=a->n[i];
9              for(j=0;j<(int)log10(RADIX);j++){
10                 temp = acopy/base;
11                 acopy %= base;
12                 if(temp!=0){
13                     //printf("j = %d\n",j);
14                     return (i+1)*(int)log10(RADIX)-j;
15                 }
```

```
16                  base/=10;
17              }
18          }
19      }
20      return 0;
21  }
```

2.7 dispNumber 関数

多倍長変数の符号と値を表示する dispNumber 関数を作成する. 表 8 に dispNumber 関数の機能, および引数, および返り値を示す. dispNumber 関数は, 第一引数で渡された多倍長変数の符号 (+,− のいずれか) と値を表示する関数である.

表 8: dispNumber 関数の概要

機能	多倍長変数の符号および各桁の値を表示する.
引数	struct NUMBER *a
返り値の型	void
返り値	-

リスト 7 に dispNumber 関数のソースコードを示す.dispNumber 関数の内部では, 最初に getSign 関数で多倍長変数の符号を取得し, 表示する. 次にリスト 7 の 11〜13 行目に示すように, 多倍長変数の値を上位桁 a_{KETA-1} から順番に a_0 まで表示している.

リスト 7: dispNumber 関数のソースコード

```
1  // aを全桁表示
2  void dispNumber(struct NUMBER *a){
3      int i;
4
5      if(getSign(a)==1){
6          printf("+ ");
7      }else{
8          printf("- ");
9      }
10
11     for(i=KETA-1; i>=0; i--){
12         printf("%04d ",a->n[i]);
13     }
14 }
```

2.8 dispNumberZeroSuppress 関数

桁数の大きい多倍長変数を表示するときに上位桁の 0 を省略したい. この操作をゼロサプレスという. そこで上位桁の 0 を省略して多倍長変数の符号と値を表示する dispNumberZeroSuppress 関数を作成する. 表 9 に dispNumberZeroSuppress 関数の機能, および引数, および返り値を示す. 表示方法が dispNumber 関数と異なるだけであるから, 引数および返り値は dispNumber 関数と同じである.

表 9: dispNumberZeroSuppress 関数の概要

機能	多倍長変数の符号および各桁の値を上位桁の 0 を省略して表示する.
引数	struct NUMBER *a
返り値の型	void
返り値	-

リスト 8 に dispNumberZeroSuppress 関数のソースコードを示す.dispNumberZeroSuppress 関数の内部では最初に,getSign 関数を用いて符号を取得し表示する. 次にリスト 8 の 12〜14 行目に示すようにゼロサプレスを行う多倍長変数 a が 0 であるときに画面出力が行われないことを防ぐため,isZero 関数を用いて a が 0 かを判別し,0 であれば 0 と表示して終了する.0 でなければ配列の要素上位桁から一桁ずつ 0 でないか確認し,0 であるときは画面に出力しない.0 ではない数が現れて以降は, 全ての桁を表示する処理を行う (リスト 8 の 15〜24 行目).

リスト 8: dispNumberZeroSuppress 関数のソースコード

```
1  // aを全桁表示(ゼロサプレス)
2  void dispNumberZeroSuppress(struct NUMBER *a){
3      int i;
4      int j=1;
5
6      if(getSign(a)==1){
7          printf("+ ");
8      }else{
9          printf("- ");
10     }
11
12     if(isZero(a)==0){
13         printf("0");
14     }
15     for(i=KETA-1; i>=0; i--){
16         if(j){
17             if(a->n[i] != 0){
18                 printf("%d ",a->n[i]);
19                 j=0;
20             }
21         }else{
22         printf("%04d ",a->n[i]);
23         }
24     }
25 }
```

2.9 copyNumber 関数

多倍長変数を別の多倍長変数にコピーする関数として,copyNumber 関数を作成する. 表 10 に copyNumber 関数の機能, および引数, および返り値を示す. copyNumber 関数は第一引数の多倍長変数の符号と値を別の多倍長変数にコピーする関数である.

表 10: copyNumber 関数の概要

機能	多倍長変数の符号と値を別の多倍長変数にコピーする.
引数	struct NUMBER *a,struct NUMBER *b
返り値の型	void
返り値	-

リスト 9 に copyNumber 関数のソースコードを示す.copyNumber 関数の内部では b の構造体に a の構造体を渡す処理を行っている.

リスト 9: copyNumber 関数のソースコード

```
1  //aをbにコピー
2  void copyNumber(struct NUMBER *a,struct NUMBER *b){
3      *b=*a;
4  }
```

2.10 getAbs 関数

多倍長変数の絶対値を別の多倍長変数に代入する関数として,getAbs 関数を作成する. 表 11 に getAbs 関数の機能, および引数, および返り値を示す. getAbs 関数は第一引数で渡された多倍長変数 a の絶対値を第

二引数で渡された多倍長変数 b に代入する関数である.

表 11: getAbs 関数の概要

機能	多倍長変数の絶対値を別の多倍長変数に代入する.
引数	struct NUMBER *a,struct NUMBER *b
返り値の型	void
返り値	-

リスト 10 に getAbs 関数のソースコードを示す.getAbs 関数の内部では, 最初に copyNumber 関数を用いて a の値を b にコピーする. 次に setSign 関数を用いて b の符号を正に設定する.

リスト 10: getAbs 関数のソースコード

```
1  //aの絶対値をbに代入
2  void getAbs(struct NUMBER *a,struct NUMBER *b){
3      copyNumber(a,b);
4      setSign(b,1);
5  }
```

2.11 mulBy10 関数

多倍長変数の 10 倍を別の多倍長変数に代入する関数として,mulBy10 関数を作成する. 表 12 に mulBy10 関数の機能, および引数, および返り値を示す.mulBy10 関数は第一引数の多倍長変数 a の 10 倍を第二引数の多倍長変数 b に代入する関数である. オーバーフローしたときには返り値として-1 を返す.

表 12: mulBy10 関数の概要

機能	多倍長変数の値を 10 倍する.
引数	struct NUMBER *a,struct NUMBER *b
返り値の型	int
返り値	成功 (0), 失敗 (-1)

リスト 11 に mulBy10 関数のソースコードを示す.mulBy10 関数の内部では, 最初に,setSign 関数を用いて a の符号を b に設定する. 次にリスト 11 の 6〜8 行目に示すように a の最上位桁が 0 であることを確認する. もし 0 でなければ 10 倍したときにオーバーフローしてしまうので-1 を返す. オーバーフローが起きないことを確認したので b に a の 10 倍を代入する処理を行う.

リスト 11: mulBy10 関数のソースコード

```
1  //aを10倍してbに代入
2  int mulBy10(struct NUMBER *a,struct NUMBER *b){
3      int i,temp;
4      int carry =0;
5      setSign(b,getSign(a));
6      if(a->n[KETA-1]/(int)(RADIX/10) != 0){
7          return -1;
8      }
9
10     b->n[KETA-1] = a->n[KETA-1]*10;
11     for(i=KETA-2; i>=0;i--){
12         temp = a->n[i]*10;
13         carry = temp/RADIX;
14         b->n[i] = temp % RADIX;
15         b->n[i+1]+=carry;
16     }
17     return 0;
18 }
```

2.12 mulBy10000 関数

多倍長変数の 10000 倍を別の多倍長変数に代入する関数として mulBy10000 関数を作成する. 表 13 に mulBy10000 関数の機能, および引数, および返り値を示す.mulBy10000 関数は第一引数の多倍長変数 a の 10000 倍を第二引数の多倍長変数 b に代入する関数である.

表 13: mulBy10000 関数の概要

機能	多倍長変数の値を 10000 倍する.
引数	struct NUMBER *a,struct NUMBER *b
返り値の型	int
返り値	成功 (0), 失敗 (-1)

リスト 12 に mulBy10000 関数のソースコードを示す.mulBy10000 関数の内部では, 最初に setSign 関数を用いて b に a の符号をセットする. 次にリスト 12 の 5〜7 行目に示すように a の最上位桁が 0 であることを確認する. 最上位桁が 0 でなければ 10000 倍したときにオーバーフローしてしまうため-1 を返す. オーバーフローが起きないことを確認したので b に a の 10000 倍を代入する処理を行う.

リスト 12: mulBy10000 関数のソースコード

```
1  //aを10000倍してbに代入
2  int mulBy10000(struct NUMBER *a,struct NUMBER *b){
3      int i;
4      setSign(b,getSign(a));
5      if(a->n[KETA-1]!=0){
6          return -1;
7      }
8
9      for(i=0; i<KETA-1;i++){
10         b->n[i+1] = a->n[i];
11     }
12     b->n[0]=0;
13     return 0;
14 }
```

2.13 setInt 関数

int 型の変数を多倍長変数に変換する関数として,setInt 関数を作成する. 表 14 に setInt 関数の機能, および引数, および返り値を示す.setInt 関数は第二引数の int 型の値 x を第一引数の多倍長変数 a に代入する関数である. 多倍長変数の桁数が足りない場合は int 型の変数を変換しきれないので-1 を返す.

表 14: setInt 関数の概要

機能	int 型の変数を多倍長変数に設定する.
引数	struct NUMBER *a,int x
返り値の型	int
返り値	成功 (0), 失敗 (-1)

setInt 関数の処理内容は次の通りである.

1. 多倍長変数の桁数が足りることを確認する. 足りない場合は-1 を return する.
2. x が負の時 x の絶対値を作業用変数 (int 型) に代入する.
3. a の下位桁から順に x の作業用変数の値を設定する.
4. setInt 関数を用いて a に x の符号を設定する.

リスト 13 に setInt 関数のソースコードを示す.setInt 関数の内部では, 最初にリスト 13 の 4〜7 行目に示すように多倍長変数の桁数が足りるか確認を行う. 足りなければ-1 を返して終了する. 次に a をゼロクリアして int 型の値 x の絶対値を a にセットする. 最後に x の符号を setSign 関数を用いて a の符号にセットする.

リスト 13: setInt 関数のソースコード

```
1  //xをaに代入
2  int setInt(struct NUMBER *a,int x){
3      int temp,i;
4      if(KETA<3){
5          printf("KETAが足りません\n");
6          return -1;
7      }
8
9      clearByZero(a);
10
11     temp=x;
12     if(x<0){
13         temp*=-1;
14     }
15     for(i=0;i<3;i++){
16         a->n[i]=temp%RADIX;
17         temp-=a->n[i];
18         temp/=RADIX;
19     }
20
21     if(x<0){
22         setSign(a,-1);
23     }else{
24         setSign(a,1);
25     }
26     return 0;
27 }
```

2.14 numComp 関数

多倍長変数の大小を比較する関数として numComp 関数を作成する. 表 15 に numComp 関数の機能, および引数, および返り値を示す.numComp 関数は第一引数の多倍長変数 a と第二引数の多倍長変数 b のの大小を比較する関数である.

表 15: numComp 関数の概要

機能	多倍長変数の大小を比較する.
引数	struct NUMBER *a,struct NUMBER *b
返り値の型	int
返り値	a¿b(1),a¡b(-1),a=b(0), 不明なエラー (623)

リスト 14 に numComp 関数のソースコードを示す.

リスト 14: numComp 関数のソースコード

```
1  //多倍長の大小比較 a==b:0 a>b:1 a<b:-1
2  int numComp(struct NUMBER *a,struct NUMBER *b){
3      int i,Sa,Sb;
4      Sa = getSign(a);
5      Sb = getSign(b);
6
7      // a+ b-
8      if((Sa==1)&&(Sb==-1)){
9          return 1;
10     }
11
12     //a- b+
13     if((Sa==-1)&&(Sb==1)){
14         return -1;
15     }
16     //a+ b+
17     if((Sa==1)&&(Sb==1)){
18         for(i=KETA-1; i>=0; i--){
19             if(a->n[i] > b->n[i]){
```

```
20              return 1;
21          }
22
23          if(a->n[i] < b->n[i]){
24              return -1;
25          }
26      }
27      return 0;
28  }
29
30  if((Sa==-1)&&(Sb==-1)){
31      for(i=KETA-1; i>=0; i--){
32          if(a->n[i] > b->n[i]){
33              return -1;
34          }
35
36          if(a->n[i] < b->n[i]){
37              return 1;
38          }
39      }
40      return 0;
41  }
42  return 623;
43 }
```

numComp 関数の処理内容は次の通りである.a,b の符号によって場合わけをして大小を比較する.

1. a および b の符号を getSign 関数を用いて取得する.
2. a が正,b が負のとき a¿b であるから 1 を return する.
3. a が負,b が正のとき b¡a であるから-1 を return する.
4. a,b がともに正のとき, 最上位桁から順に比較し, 大小関係が確定させる.
5. a,b がともに負のとき, 最上位桁から順に比較し, 大小関係が確定させる.

2.15 add 関数

2 つの多倍長変数の和を計算する関数として,add 関数を作成する. 表 16 に add 関数の機能, および引数, および返り値を示す.add 関数は第一引数の多倍長変数 a と第二引数の多倍長変数 b の和 $a+b$ を計算し, 第三引数の多倍長変数 c に計算結果を代入する関数である.$a+b$ の計算結果がオーバーフローしたときには c をゼロクリアして終了する. 関数の返り値は関数の実行が正常に行われたか検知するために使用する.

表 16: add 関数の概要

機能	a+b を計算し,c に計算結果を代入する.
引数	struct NUMBER *a,struct NUMBER *b,struct NUMBER *c
返り値の型	int
返り値	成功 (0), 失敗 (-1), 不明なエラー (623)

リスト 16 に add 関数のソースコードを示す.

リスト 15: add 関数のソースコード

```
1  // c<- a+b
2  int add(struct NUMBER *a,struct NUMBER *b,struct NUMBER *c){
3      int Sa = getSign(a);
4      int Sb = getSign(b);
5      struct NUMBER Nd,Ne;
6      int d,i;
7      int e=0;
8      //a+ b+
9      if((Sa==1)&&(Sb==1)){
10         for(i=0;i<KETA;i++){
11             d = a->n[i] + b->n[i] +e;
12             c->n[i] = d%RADIX;
13             e = (d - c->n[i]) /RADIX;
14         }
15         if(e==1){
16             clearByZero(c);
```

```
17              return -1;
18          }
19          setSign(c,1);
20          return 0;
21          }
22
23          //a+ b-
24      if((Sa==1)&&(Sb==-1)){
25          getAbs(b,&Nd);
26          sub(a,&Nd,c);
27          return 0;
28      }
29
30      //a- b+
31      if((Sa==-1)&&(Sb==1)){
32          getAbs(a,&Nd);
33          sub(b,&Nd,c);
34          return 0;
35      }
36
37      //a- b-
38      if((Sa==-1)&&(Sb==-1)){
39          getAbs(a,&Nd);
40          getAbs(b,&Ne);
41          add(&Nd,&Ne,c);
42          setSign(c,-1);
43          return 0;
44      }
45      return 623; //不明なエラー検出
46  }
```

add 関数の処理内容について説明する. add 関数は a,b の符号によって処理が異なる.a,b がともに正のとき次の手順で計算する. この手順は授業中に紹介された筆算の方法である.

1. i(int 型), および e(int 型) を 0 にセットする.
2. a の i 桁目 a_i を取り出す.
3. b の i 桁目 b_o を取り出す.
4. d に $a_i + b_i + e$ を代入する.
5. d の 1〜1000 の位の値を取り出して c の i 桁目にセットする.
6. d の 10000 の位を取り出して e に代入する.
7. i をインクリメントする.i≦ KETA-1 のとき, 手順 2 に戻る.
8. e が 0 でない場合オーバーフローであるので clearByZero 関数を用いて c をゼロクリアして終了する.
9. c の符号を setSign 関数を用いて正にセットし終了する.

次に a が正,b が負の場合を考える. この場合 $a+b$ は式 (1) に示すように変形できるから, a と b の絶対値をとった値を減算関数 (sub 関数) に渡す処理をしている. ただし,sub 関数については次節で実装している.

$$c = a + b = a - |b| \tag{1}$$

a が負,b が正の場合を考える. この場合 $a+b$ は式 (2) に示すように変形できるから b と a の絶対値をとった値を sub 関数に渡す処理をしている.

$$c = a + b = b - |a| \tag{2}$$

a,b がともに負のときを考える. この場合 $a+b$ は式 (3) に示すように変形できるから, 再度 add 関数を呼び出し a の絶対値を取った値と,b の絶対値を取った値を渡し, 計算結果の符号を負に設定する処理をしている.

$$c = a + b = -(|a| + |b|) \tag{3}$$

2.16 sub 関数

2 つの多倍長変数の差を計算する関数として,sub 関数を作成する. 表 17 に sub 関数の機能, および引数, および返り値を示す.sub 関数は第一引数の多倍長変数 a と第二引数の多倍長変数 b の差 $a-b$ を計算し, 第

三引数の多倍長変数 c に計算結果を代入する関数である.$a - b$ の計算結果がオーバーフローしたときには c をゼロクリアして終了する. 関数の返り値は関数の実行が正常に行われたか検知するために使用する.

表 17: sub 関数の概要

機能	a-b を計算し,c に計算結果を代入する.
引数	struct NUMBER *a,struct NUMBER *b,struct NUMBER *c
返り値の型	int
返り値	成功 (0), 失敗 (-1), 不明なエラー (623)

リスト 16: sub 関数のソースコード

```
1  // c<- a-b
2  int sub(struct NUMBER *a,struct NUMBER *b,struct NUMBER *c){
3      struct NUMBER Nd,Ne;
4      int Sa = getSign(a);
5      int Sb = getSign(b);
6      int i;
7      int h=0;
8      //a+ b+
9      if((Sa==1)&&(Sb==1)){
10     if(numComp(a,b)==1){
11         setSign(c,1);
12         for(i=0;i<KETA;i++){
13             a->n[i]-=h;
14             if(a->n[i]>=b->n[i]){
15                 c->n[i] = a->n[i] - b->n[i];
16                 h=0;
17             }else{
18                 c->n[i] = RADIX + a->n[i] - b->n[i];
19                 h=1;
20             }
21         }
22         return 0;
23     }
24
25     if(numComp(a,b)==-1){
26         for(i=0;i<KETA;i++){
27             b->n[i]-=h;
28             if(b->n[i]>=a->n[i]){
29                 c->n[i] = b->n[i] - a->n[i];
30                 h=0;
31             }else{
32                 c->n[i] = RADIX
33                  + b->n[i] - a->n[i];
34                 h=1;
35             }
36         }
37         setSign(c,-1);
38     }
39
40     if(numComp(a,b)==0){
41         clearByZero(c);
42     }
43     return 0;
44     }
45
46     //a+ b-
47     if((Sa==1)&&(Sb==-1)){
48         getAbs(b,&Nd);
49         add(a,&Nd,c);
50         return 0;
51     }
52
53     //a- b+
54     if((Sa==-1)&&(Sb==1)){
55         getAbs(a,&Nd);
56         add(&Nd,b,c);
57         setSign(c,-1);
58         return 0;
59     }
60
61     //a- b-
62     if((Sa==-1)&&(Sb==-1)){
63         getAbs(b,&Ne);
64         add(a,&Ne,c);
65         return 0;
66     }
67
```

```
68     return 623;//不明なエラー検出
69 }
```

sub 関数の内部処理を説明する. sub 関数は a,b の符号と大小関係によって処理が異なる.a≧b かつ両方が正のとき次の手順で計算する. a<b のとき式 (4) のように変形できるので再度 sub 関数を呼び出して $b-a$ を計算し, 計算結果に負の符号を設定する.

1. i(int 型), および h(int 型) を 0 にセットする.
2. a の i 桁目 a_i を取り出す.
3. b の i 桁目 b_i を取り出す.
4. a_i から h を引く.
5. $a_i \geq b_i$ であれば $c_i = a_i - b_i$ を計算し,h=0 とする.
6. $a_i < b_i$ であれば $c_i = 10000 + a_i - b_i$ を計算し,h=1 とする.
7. i をインクリメントする.i≦KETA-1 のとき 2 に戻る.
8. h が 1 のときオーバーフローであるから clearByZero 関数を用いて c をゼロクリアし, 終了する.
9. setSign 関数を用いて c の符号を正にセットして終了する.

$$c = a - b = -(b - a) \tag{4}$$

次に a が正,b が負の場合を考える. この場合 $a-b$ は式 (5) に示すように変形できるから, a と b の絶対値をとった値の和を add 関数で計算する.

$$c = a - b = a + |b| \tag{5}$$

a が負,b が正の場合を考える. この場合 $a-b$ は式 (6) に示すように変形できるから, b と a の絶対値をとった値の和を add 関数で計算し符号を負にセットする.

$$c = a - b = -(|a| + b) \tag{6}$$

a が b がともに負の場合を考える. この場合 $a-b$ は式 (7) に示すように変形できるから, a と b の絶対値をとった値の和を add 関数で計算する.

$$c = a - b = a + |b| \tag{7}$$

2.17 increment 関数

多倍長変数にインクリメントを行う関数として increment 関数を作成する. インクリメントとは変数の値を 1 増やす演算のことである. 表 18 に increment 関数の機能, および引数, および返り値を示す.increment 関数は第一引数の多倍長変数の値 a をインクリメントし, 第二引数の多倍長変数 b に代入する関数である. オーバーフローが起きたときには-1 を返す.

表 18: increment 関数の概要

機能	多倍長変数をインクリメントする.
引数	struct NUMBER *a,struct NUMBER *b
返り値の型	int
返り値	成功 (0), 失敗 (-1)

リスト 17 に increment 関数のソースコードを示す.increment 関数の内部では, 最初に作業用の多倍長変数 one に setInt 関数を用いて 1 を代入する. 次に add 関数を用いて $a+1$ を計算し b に代入する. オーバー

フローが起きたとき,add 関数は表 16 より返り値として成功 (0), 失敗 (−1) を返すので, これを increment 関数の返り値としている.

リスト 17: increment 関数のソースコード

```
// b<- a++
int increment(struct NUMBER *a,struct NUMBER *b){
    struct NUMBER one;
    int r;
    setInt(&one,1);
    r = add(a,&one,b);
    return r;
}
```

2.18 multiple 関数

2 つの多倍長変数の積を計算する関数として multiple 関数を作成する. 表 19 に multiple 関数の機能, および引数, および返り値を示す.multiple 関数は第一引数の多倍長変数 a と第二引数の多倍長変数 b の積を計算し, 第三引数の多倍長変数 c に代入する関数である. オーバーフローが起きたとき, 返り値として −1 を返す.

表 19: multiple 関数の概要

機能	a×b を計算し c に代入する.
引数	struct NUMBER *a,struct NUMBER *b,struct NUMBER *c
返り値の型	int
返り値	成功 (0), 失敗 (-1)

リスト 18 に multiple 関数のソースコードを示す.

リスト 18: multiple 関数のソースコード

```
// c<-a*b
int multiple(struct NUMBER *a,struct NUMBER *b,struct NUMBER *c){
    int e,h,i,j;
    int Sa = getSign(a);
    int Sb = getSign(b);
    struct NUMBER d,d2,ee;
    clearByZero(c);

    if(Sa==1 && Sb==1){
        for(i=0;i<KETA;i++){
            h=0;
            clearByZero(&d);
            for(j=0;j<KETA;j++){
                e=(a->n[j]* b->n[i])+h;
                d.n[j] = e%RADIX;
                e-=d.n[j];
                h = e/RADIX;
            }
            if(h != 0){
                clearByZero(c);
                return -1;
            }

            for(j=0;j<i*((int)log10(RADIX));j++){
                if(mulBy10(&d,&d2)==-1){
                    clearByZero(c);
                    return -1;
                }
                copyNumber(&d2,&d);
            }

            if(add(c,&d,c)==-1){
                clearByZero(c);
                return -1;
            }
        }
        return 0;
    }

    if(Sa==-1 && Sb==1){
```

```
41          getAbs(a,&d);
42          i = multiple(&d,b,c);
43          if(i==-1){
44              return -1;
45          }
46          setSign(c,-1);
47          return 0;
48      }
49
50      if(Sa==1 && Sb==-1){
51          getAbs(b,&d);
52          i = multiple(a,&d,c);
53          if(i==-1){
54              return -1;
55          }
56          setSign(c,-1);
57          return 0;
58      }
59      if(Sa==-1 && Sb ==-1){
60          getAbs(a,&d);
61          getAbs(b,&ee);
62          i = multiple(&d,&ee,c);
63          if(i==-1){
64              return -1;
65          }
66          setSign(c,1);
67          return 0;
68      }
69      return 623;
70  }
```

multiple 関数の内部処理を説明する. multiple 関数の処理は a および b の符号によって場合分けを行う.a,b がともに正のときの処理内容は次の通りである.

1. c をゼロクリアする.
2. i=0(int 型) とする.
3. b の i 桁目 b_i を取り出す.
4. 作業用の変数として,h(int 型) および多倍長変数 d をゼロクリアする.
5. j=0(int 型) とする.
6. a の j 桁目 a_j を取り出す.
7. 作業用の変数 e(int 型) に $a_j \times b_i + h$ を代入する.
8. e の 1000 の位まで取り出し,d_j に代入する.
9. e の 10000 の位を取り出し h に代入する.
10. j=KETA-1 のとき手順 11 を行う.h が 0 でない場合, オーバーフローである.
11. j をインクリメントして手順 5 に戻る.
12. mulBy10 関数を用いて d を 10^i 倍する. ここでもオーバーフローが起こりえるので検出する.
13. c に d を加える. ここでもオーバーフローが起こりえる.
14. i=KETA-1 のときループ終了する.
15. i をインクリメントし手順 3 に戻る.

a が正,b が負のとき $a \times b$ は式 (8) に示すように書き直せるから, 再度 multiple 関数を呼び出し,a と b の絶対値をとった値を渡せばよい. 符号は setSign 関数を用いて負に設定する.

$$c = a \times b = -(a \times |b|) \tag{8}$$

a が負,b が正のとき $a \times b$ は式 (9) に示すように書き直せるから, 再度 multiple 関数を呼び出し,a の絶対値をとった値と b を渡せばよい. 符号は setSign 関数を用いて負に設定する.

$$c = a \times b = -(|a| \times b) \tag{9}$$

a,b がともに負のとき $a \times b$ は式 (10) に示すように書き直せるから, 再度 multiple 関数を呼び出し,a の絶対値をとった値と b の絶対値を渡せばよい. 符号は setSign 関数を用いて正に設定する.

$$c = a \times b = (|a| \times |b|) \tag{10}$$

2.19 divide_2 関数

多倍長変数の割り算を行う関数として divide_2 関数を作成する. 表 20 に divide_2 関数の機能, および引数, および返り値を示す.divide_2 関数は第一引数の多倍長変数の値 a, 第二引数 b, 第三引数 c, 第四引数 b とすると $a \div b$ を計算し, 計算結果の商を c, 剰余を d に代入する関数である.

表 20: divide_2 関数の概要

機能	a÷b を計算する,
引数	struct NUMBER *a,struct NUMBER *b,struct NUMBER *c,struct NUMBER *d
返り値の型	int
返り値	成功 (0), 失敗 (-1)

リスト 19 に divide_2 関数のソースコードを示す.

リスト 19: divide_2 関数のソースコード

```
1  //割り算高速化
2  // a/b = c...d
3  int divide_2(struct NUMBER *a,struct NUMBER *b,struct NUMBER *c,struct NUMBER *d){
4      struct NUMBER n,p,q,r;
5      int over;
6      int Sa = getSign(a);
7      int Sb = getSign(b);
8      clearByZero(c);
9      clearByZero(d);
10
11     if(isZero(b)==0){
12         printf("div by 0\n");
13         return -1;
14     }
15
16     copyNumber(a,&n);
17
18     if(Sa==1 && Sb==1){
19         while(1){
20             if(numComp(&n,b)==-1){
21                 break;
22             }
23             copyNumber(b,&p);
24             setInt(&q,1);
25             while(1){
26                 over = mulBy10(&p,&r);
27                if(over == -1){
28                    break;
29                }
30                if(numComp(&n,&r)==-1){
31                    break;
32                }
33                copyNumber(&r,&p);
34                mulBy10(&q,&r);
35                copyNumber(&r,&q);
36             }
37             sub(&n,&p,&r);
38             copyNumber(&r,&n);
39             add(c,&q,&r);
40             copyNumber(&r,c);
41
42         }
43         copyNumber(&n,d);
44         return 0;
45     }
46     if(Sa==1 && Sb==-1){
47         getAbs(b,&p);
48         divide_2(&n,&p,c,d);
49         setSign(c,-1);
50         return 0;
51     }
52
53     if(Sa==-1 && Sb==1){
54         getAbs(a,&p);
55         divide_2(&p,b,c,d);
56         setSign(c,-1);
57         setSign(d,-1);
58         return 0;
59     }
60
```

```
61      if(Sa==-1 && Sb==-1){
62          getAbs(a,&p);
63          getAbs(b,&q);
64          divide_2(&p,&q,c,d);
65          setSign(d,-1);
66          return 0;
67      }
68      return 623;
69  }
```

divide_2 関数の内部処理について説明する. divide_2 関数の処理内容は a および b の符号によって異なる.a および b がともに正のときの処理内容を次に示す. 最初に 0 除算が発生していないことを確認してから除算処理を行っている.

1. b が 0 でないことを確認する.0 であれば終了する.
2. c をゼロクリアする.
3. numComp 関数を用いて a と b の大小を調べる.a<b であれば手順 11 に飛ぶ.
4. d に b を代入する.
5. 作業用の多倍長変数 p に 1 をセットする.
6. d を 10 倍する操作を繰り返して,a>d を満たす最大の d を見つける.
7. 手順 6 で d を 10 倍したのと同じ回数だけ e を 10 倍する.
8. a から d を引く.
9. c に e を足す.
10. 手順 3 に戻る.
11. d に a をコピーして終了する.

a が正,b が負のとき,$a \div b$ の商 c は負, 剰余 d は正の値になるから, 再度 divide_2 関数を呼び出して a と b の絶対値を渡し, 計算結果の商を負の値にセットする.a が負,b が正のとき,$a \div b$ の商 c は負, 剰余 d は負の値になるから, 再度 divide_2 関数を呼び出して a の絶対値と b を渡し, 計算結果の商と剰余を負の値にセットする. a,b がともに負のとき,$a \div b$ の商 c は正, 剰余 d は負の値になるから, 再度 divide_2 関数を呼び出して a の絶対値と b の絶対値を渡し, 計算結果の商と剰余を負の値にセットする.

2.20 sqrt_newton 関数

多倍長変数の平方根を計算する関数として sqrt_newton 関数を作成する. 表 21 に sqrt_newton 関数の機能, および引数, および返り値を示す.sqrt_newton 関数は Newton-Raphson 法を用いて第一引数の多倍長変数 a の平方根を計算し, 第二引数の多倍長変数 b に代入する.

表 21: sqrt_newton 関数の概要

機能	多倍長変数の平方根を計算する.
引数	struct NUMBER *a,struct NUMBER *b
返り値の型	int
返り値	成功 (0), 失敗 (-1)

Newton-Raphson 法とは方程式 $f(x) = 0$ を反復的に解く方法である. 解を $s = \alpha$,$f(x)$ の一階微分を $f'(x)$, 適当な初期値を x_0 とする. 式 (11) を x_1,x_2... と反復計算を行うことで x の近似値 x_i が真の値 α に近づいていく.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{11}$$

Newton-Raphson 法を用いて N の平方根を計算するために $f(x)$ を式 (12) のように設定する.

$$f(x) = x^2 - N \tag{12}$$

$f(x)$ の一階微分は式 (13) であるから, 反復計算に式 (16) を用いればよいことが分かる.

$$f(x) = 2x \tag{13}$$

$$\begin{aligned} x_{i+1} &= x_i - \frac{x_i^2 - N}{2x_i} && (14)\\ &= \frac{x_i}{2} + \frac{N}{2x_i} && (15)\\ &= \frac{1}{2}\left(x_i + \frac{N}{x_i}\right) && (16) \end{aligned}$$

リスト 20 に sqrt_newton 関数のソースコードを示す.

リスト 20: sqrt_newton 関数のソースコード

```
1  // b<- sqrt{a} (Newton-Raphson法)
2  int sqrt_newton(struct NUMBER *a,struct NUMBER *b){
3      struct NUMBER x,temp,temp2,c,d;
4
5      setInt(&d,2);
6      divide_2(a,&d,&x,&c);
7      if(isZero(&x)==0){
8          copyNumber(a,b);
9          return 0;
10     }
11
12     if(getSign(&x)==-1){
13         return -1;
14     }
15
16     copyNumber(&x,&d);
17
18     while(1){
19         copyNumber(&d,&c);
20         copyNumber(&x,&d);
21
22         divide_2(a,&d,&temp,&temp2);
23         add(&d,&temp,&temp2);
24         setInt(b,2);
25         divide_2(&temp2,b,&x,&temp);
26
27         if(numComp(&x,&d)==0){
28             break;
29         }
30         if(numComp(&x,&c)==0){
31             if(numComp(&d,&x)==-1){
32                 copyNumber(&d,&x);
33             }
34         }
35     }
36     copyNumber(&x,b);
37     return 0;
38 }
```

sqrt_newton 関数の処理内容は次の通りである.

1. 作業用変数 x に初期値 $(\frac{a}{2})$ を代入する.
2. x=0, または 1 のとき $\sqrt{a} = a$ であるから b に a をコピーして終了する.
3. x が負の時エラーで-1 を返す
4. 式 (16) を反復的に計算する.
5. 収束, 振動を判定する.k 回目の計算結果が $k-1$ 回目, または $k-2$ 回目の計算結果と同じであれば終了する.

2.21 zeta4 関数

zeta4 関数はゼータ関数 $\zeta(s)$ の $s=4$ のときの値から円周率 π を計算する関数である. $\zeta(s)$ は $s=4$ のとき式 (17) である.

$$\zeta(4) = \sum_{n=1}^{\infty} \frac{1}{n^4} = \frac{\pi^4}{90} \tag{17}$$

手持ちの関数では小数を扱うことができないので $10^{\alpha}(\alpha > 0)$ をかけて式 (18) を計算する. また無限回ループを繰り返すことができないので b 回ループを繰り返した時の近似値を求める.

$$10^{\alpha} \sum_{n=1}^{b} \frac{1}{n^4} = \frac{10^{\alpha} \times \pi^4}{90} \tag{18}$$

表 22 に zeta4 関数の機能, および引数, および返り値を示す.zeta4 関数では第一引数で受け取った多倍長変数 a に計算結果 (円周率) を代入する. 第二引数の loop は式 (18) における b のことである. 第三引数の keta は式 (18) の α のことである. 第四引数の disp は途中経過を何桁おきに表示するかを格納する変数である.

表 22: zeta4 関数の概要

機能	円周率を計算する.
引数	struct NUMBER *a,int loop,int keta,int disp
返り値の型	int
返り値	成功 (0), 失敗 (-1)

リスト 21 に zeta4 関数のソースコードを示す

リスト 21: zeta4 関数のソースコード

```
1  //zeta(4)から円周率を計算
2  //結果をaに代入
3  // int loop 反復回数
4  // int keta r/n^4のrの値
5  // int disp 途中経過の表示回数
6  int zeta4(struct NUMBER *a,int loop,int keta,int disp){
7
8      if(loop<=0){
9          printf("loopが不正な値です.\n");
10         return -1;
11     }
12
13     if(keta>KETA*(int)log10(RADIX) || keta<=0){
14         printf("ketaが不正な値です.\n");
15         return -1;
16     }
17     if(disp>loop || disp<=0){
18         printf("loopが不正な値です.\n");
19         return -1;
20     }
21
22     int i,r;
23     struct NUMBER n,b,temp,temp2,tenE;
24
25     clearByZero(&b);
26     setInt(&tenE,1);
27     for(i=0;i<keta/(int)log10(RADIX);i++){
28         r = mulBy10000(&tenE,&temp);
29         copyNumber(&temp,&tenE);
30     }
31
32     for(i=0;i<keta%(int)log10(RADIX);i++){
33         r = mulBy10(&tenE,&temp);
34         copyNumber(&temp,&tenE);
35     }
36
37     i=1;
38     gettimeofday(&tv,NULL);
39     tstart =(double)tv.tv_sec + (double)tv.tv_usec *1.e-6;
```

```
40 |
41 |     printf("多倍長桁数KETA = %d\n",KETA);
42 |     printf("loop回数 = %d\n",loop);
43 |     printf("keta = %d\n",keta);
44 |     printf("途中経過表示 %d 回毎\n\n",disp);
45 |     printf("----- start loop -----\n");
46 |     while(1){
47 |         //iをセット
48 |         setInt(&n,i);
49 |         //i^4を計算
50 |         r = multiple(&n,&n,&temp);
51 |         if(r==-1){
52 |             printf("overflow in n^2\n");
53 |             return -1;
54 |         }
55 |         multiple(&temp,&temp,&n);
56 |         if(r==-1){
57 |             printf("overflow in n^4\n");
58 |             return -1;
59 |         }
60 |
61 |         divide_2(&tenE,&n,&temp,&temp2);
62 |         add(&b,&temp,&temp2);
63 |         copyNumber(&temp2,&b);
64 |
65 |         if(i%disp==0){
66 |             printf("pi^4/90[i = %d] = ",i);
67 |             dispNumberZeroSuppress(&b);
68 |             printf("\n\n");
69 |             gettimeofday(&tv,NULL);
70 |             tend =(double)tv.tv_sec + (double)tv.tv_usec *1.e-6;
71 |             printf("所要時間 = %f秒\n\n",tend-tstart);
72 |         }
73 |         if(i==loop){
74 |             break;
75 |         }
76 |         i++;
77 |     }
78 |     printf("----- finish loop -----\n");
79 |     printf("pi^4/90 = ");
80 |     dispNumberZeroSuppress(&b);
81 |     printf("\n\n");
82 |
83 |     setInt(&temp2,90);
84 |     multiple(&b,&temp2,&temp);
85 |     printf("pi^4 = ");
86 |     dispNumberZeroSuppress(&temp);
87 |     printf("\n\n");
88 |
89 |     //平方根を取るために桁調整
90 |     copyNumber(&temp,&temp2);
91 |     r = getKeta(&temp2)%(int)log10(RADIX);
92 |     if(r!=4){
93 |         for(i=0;i<r; i++){
94 |             mulBy10(&temp2,&temp);
95 |             copyNumber(&temp,&temp2);
96 |         }
97 |     }
98 |
99 |     // 平方根をとってpi^2を計算
100|     //printf("平方根1回目直前桁数 : %d\n",getKeta(&temp2));
101|     printf("----- calculate pi^2 -----\n");
102|     sqrt_newton(&temp2,&temp);
103|     printf("pi^2 = ");
104|     dispNumberZeroSuppress(&temp);
105|     printf("\n\n");
106|
107|     //平方根を取るために桁調整
108|     copyNumber(&temp,&temp2);
109|     r = getKeta(&temp2)%2;
110|     if(r==0){
111|         mulBy10(&temp2,&temp);
112|         copyNumber(&temp,&temp2);
113|     }
114|
115|     // 平方根をとってpiを計算
116|     //printf("平方根2回目直前桁数 : %d\n",getKeta(&temp2));
117|     printf("----- calculate pi -----\n");
118|     sqrt_newton(&temp,&temp2);
119|     /*printf("pi = ");
120|     dispNumberZeroSuppress(&temp2);
121|     printf("\n");*/
122|
123|     gettimeofday(&tv,NULL);
124|     tend =(double)tv.tv_sec + (double)tv.tv_usec *1.e-6;
125|     printf("所要時間 = %f秒\n\n",tend-tstart);
126|
```

```
127     copyNumber(&temp2,a);
128     return 0;
129 }
```

zeta4 関数の処理内容を次に示す.zeta4 関数はこれまでに作成した関数を下請けとして円周率を計算する関数である.

1. $\frac{1}{n^4}$ の和を多倍長変数 b に代入するので b をゼロクリアしておく.
2. 10^{α} を作業用多倍長変数 tenE に代入する.
3. i=1(int 型) を用意する.
4. setInt 関数を用いて作業用多倍長変数 n に i を代入する.
5. n^2 を作業用多倍長変数 temp に代入する.
6. $temp^2$ を作業用多倍長変数 n に代入する. これで n^4 を計算できた.
7. $tenE \div n$ を計算し作業用多倍長変数 temp2 に代入する.
8. $i \div disp$ の剰余が 0 であれば途中経過を表示する.
9. i と loop 回数が等しければ手順 11 に飛ぶ.
10. i をインクリメントして手順 4 に戻る.
11. temp2 を 90 倍して temp に代入する. これで π^4 を計算できた.
12. 平方根を取るために桁調整を行う.
13. sqrt_newton 関数を用いて π^2 を計算する.
14. 平方根を取るために桁調整を行う.
15. sqrt_newton 関数を用いて π を計算する.

2.22 fileNumberZeroSuppress 関数

円周率の計算結果をファイルに書き込む関数として fileNumberZeroSuppress 関数を作成する. 表 23 に fileNumberZeroSuppress 関数の機能, および引数, および返り値を示す. fileNumberZeroSuppress 関数は dispNumberZeroSuppress 関数をファイル書き込みに変更したものである. 第一引数の多倍長変数 a をゼロサプレスして calpi.txt に書き込む. 計算によってもとめた円周率を書き込むための関数であり, 符号が負の多倍長変数には対応していない.

表 23: fileNumberZeroSuppress 関数の概要

機能	多倍長変数をファイルに書き込む.
引数	struct NUMBER *a
返り値の型	int
返り値	成功 (0), 失敗 (-1)

リスト 22 に fileNumberZeroSuppress 関数のソースコードを示す. fileNumberZeroSuppress 関数の内部では最初に, ローカル変数として書き込みを行うファイル名を文字列として定義する. 次に fopen 関数を用いて書き込みモードでファイルを開く. 何らかの理由でファイルが開けなかった場合は, 処理を終了する. ファイルを開くと,a の値をゼロサプレスしてファイルに書き込みを行う.

リスト 22: fileNumberZeroSuppress 関数のソースコード

```
1 //aをファイルに書き込み
2 int fileNumberZeroSuppress(struct NUMBER *a){
3     char *fname = "calpi.txt";
4     FILE *fp;
5     int i;
6     int j=1;
```

```
    if((fp = fopen(fname,"w"))==NULL){
        printf("ファイルを開けませんでした\n");
        return -1;
    }else{
        printf("\nresult write in %s ...",fname);
        if(isZero(a)==0){
        printf("0");
    }
    for(i=KETA-1; i>=0; i--){
        if(j){
            if(a->n[i] != 0){
                fprintf(fp,"%d",a->n[i]);
                j=0;
            }
        }else{
        fprintf(fp,"%04d",a->n[i]);
        }
    }
    printf("complete\n");
    fclose(fp);
    }
    return 0;
}
```

2.23 calcheck 関数

計算によって求めた円周率と真の円周率の値を比較し, 一致した桁数を表示する関数として calcheck 関数を作成する. 表 24 に calcheck 関数の機能, および引数, および返り値を示す.calcheck 関数は計算によって求めた円周率と真の値を比較し, 小数点以下の一致桁数を表示する関数である.

表 24: calcheck 関数の概要

機能	計算による円周率と真の値を比較する.
引数	void
返り値の型	int
返り値	成功 (0), 失敗 (-1)

リスト 23calcheck 関数の処理内容を説明する. 事前に計算によって求めた円周率を fileNumberZeroSuppress 関数を用いて calpi.txt に書き込む. さらに,Wolfarm Alpha[1] から円周率 1000 桁をコピーして pi.txt に書き込む. リスト 24 に pi.txt の冒頭 100 文字を示す. ただしどちらのファイルについても浮動小数点を削除しておく.calpi.txt と pi.txt を値が一致しなくなるまで 1 文字ずつ読み込み, 小数点以下の一致桁を表示する.

リスト 23: calcheck 関数のソースコード

```
//pi.txtとcalpi.txtを比較
int calcheck(void){
    char *fname1 = "pi.txt";
    char *fname2 = "calpi.txt";
    FILE *fp1;
    FILE *fp2;
    int ch1;
    int ch2;
    int i=0;

    if((fp1 = fopen(fname1,"r"))==NULL){
        printf("ファイル %s を開けませんでした\n",fname1);
        return -1;
    }else{
        if((fp2 = fopen(fname2,"r"))==NULL){
        printf("ファイル %s を開けませんでした\n",fname2);
        return -1;
        }else{
            while (( ch2 = fgetc(fp2)) != EOF ) {
                ch1 = fgetc(fp1);
            //putchar(ch2);
            //putchar(ch1);
            if(ch1!=ch2){
                break;
            }
            i++;
```

```
    }
    printf("\n小数点以下 %d 桁一致\n",i-1);
    fclose(fp2);
    fclose(fp1);
    }
    }
    return 0;
}
```

リスト 24: pi.txt の内容

```
31415926535897932384626433832795028841971693993751
05820974944592307816406286208998628034825342117067
```

2.24 メイン関数

メイン関数を作成する. リスト 25 にメイン関数のソースコードを示す. メイン関数では, 最初に多倍長変数 pi を定義する.zeta4 関数に pi を渡して円周率を計算する. zeta4 関数で計算に失敗した場合はメイン関数を終了する. zeta4 関数が失敗していなければ,pi には円周率の計算結果が入っているから,dispNumberZeroSuppress 関数に pi を渡してファイルに計算結果を書き込む. 最後に計算結果の精度の確認を行う.calcheck 関数を用いて小数点以下の一致桁数を確認してメイン関数を終了する.

リスト 25: メイン関数のソースコード

```
#include<stdio.h>
#include "mulprec.h"

#define calpi

int main(void){
    #ifdef calpi
    struct NUMBER pi;
    int loop = 1000; //ループ回数
    int keta = 130;
    int disp =100; //disp回おきに途中経過を表示
    int r;
    r = zeta4(&pi,loop,keta,disp);
    if(r==0){
        printf("\n----- result -----\n");
        printf("pi = ");
        dispNumberZeroSuppress(&pi);
        printf("\n");
    //結果をファイルに書き込み
    fileNumberZeroSuppress(&pi);
    //検算
    printf("\n----- 検算 -----\n");
    calcheck();
    }else{
        printf("zata4関数を異常終了しました\n");
    }
    #endif

    return 0;
}
```

3 実行結果

本章では短時間で計算を行ったときの実行結果, および長時間で計算を行ったときの実行結果について述べる.

3.1 短時間で計算を行ったときの実行結果

作成した関数が正常に動作しているか確認するために, 次の条件で実行を行った.

- 桁数 KETA= 150

- ループ回数 loop= 1000
- 割り算を行うために 10 倍する回数 keta= 130
- 途中経過の表示回数 disp= 100

実行結果を図 1 に示す. 図 1 の result と書いてある上の所要時間 296.7 秒から処理時間はおおよそ 5 分であることが読み取れる. また,result の項目から円周率の計算結果が $pi = 3.1415...$ であることが読み取れる. さらに, 検算の項目から検算の結果小数点以下 9 桁が真の円周率の値と一致していることが分かる.

```
----- finish loop -----
pi^4/90 = + 108 2323 2333 7830 4524 8495 0369 6318 9461 8055 0862 0707 0756 9130 2668 9495 6199
 1483 2747 3395 8134 1395 8748 2419 5701 3794 2907 0402 0965 4105 2653 0717 4125

pi^4 = + 9740 9091 0040 4740 7236 4553 3266 8705 1562 4957 7586 3636 8122 1724 0205 4605 7923 3
494 7260 5623 2072 5628 7341 7761 3124 1486 1633 6188 6886 9473 8776 4567 1250

----- calculate pi^2 -----
pi^2 = + 98 6960 4399 5718 1907 0897 7146 9263 0347 9789 2719 7314 1090 7194 3785 7992 4244

----- calculate pi -----
所 要 時 間  = 296.771857秒


----- result -----
pi = + 31 4159 2653 3482 6931 8061 7810 6291 9860

result write in calpi.txt ...complete

----- 検 算  -----

小 数 点 以 下  9 桁 一 致
```

図 1: ループ回数を少なくして実行した結果

3.2 長時間で計算を行ったときの実行結果

短時間で計算を実行した結果, 円周率の計算プログラムが正常に動くことが確かめられたから, 次の条件で実行を行った.

- 桁数 KETA= 150
- ループ回数 loop= 500000
- 割り算を行うために 10 倍する回数 keta= 120
- 途中経過の表示回数 disp= 1000

実行結果を図 2 に示す. ループ回数を 500000 回にしたとき, 処理に 85874.9 秒かかったことが読み取れる. 時間に直すと 23 時間 51 分 14.9 秒である. また, 円周率の計算結果は小数点以下 17 桁一致していることが読み取れる.

```
----- finish loop -----
pi^4/90 = + 1 0823 2323 3711 1381 8884 9345 0298 6383 4569 4414 3895 1918 7267 93
 5198 4386 9034 2838 3955 3773 5446 0501 1025 2804 9116 5076 8781 3207 8986 2701

pi^4 = + 97 4090 9103 4002 4369 9644 1052 6877 4511 1249 7295 0567 2685 4114 5146
859 4821 3085 5455 5983 9619 0144 5099 2275 2442 0485 6919 0318 8710 8764 3090

----- calculate pi^2 -----
pi^2 = + 98 6960 4401 0893 5860 6675 9854 3837 3095 6280 0854 1083 8416 5769 8150
587

----- calculate pi -----
所要時間 = 85874.980295秒


----- result -----
pi = + 3141 5926 5358 9793 2365 2755 7122 5559

result write in calpi.txt ...complete

----- 検算 -----

小数点以下 17 桁一致
```

図 2: ループ回数を増やして実行した結果

4 考察

2 つの実行結果だけでは明らかにデータ不足であるから桁数を固定し, ループ回数を変化させたときの円周率の精度 (小数点以下) および実行時間の変化を測定した. ループ回数以外の条件を次に示す.

- 桁数 KETA= 150
- 割り算を行うために 10 倍する回数 keta= 120
- 途中経過の表示回数 disp= 100

図 3 にループ回数と精度の関係を示す. 図 3 はループ回数 (底が 10 の対数) と小数点以下の円周率の計算の精度を表している. 図 3 から精度はループ回数の対数に比例することが読み取れる. このことから収束が非常に遅いことが分かる. 最小二乗法を用いて円周率を小数点以下 1000 桁求めるために必要なループ回数を計算すると, おおよそ 10^{43000} であることが分かった. これは現実的なループ回数の数値ではない. このことからゼータ関数 $\zeta(s)$ が $s = 4$ のときのアルゴリズムで円周率を 1000 桁求めることは現実的ではないと考えられる.

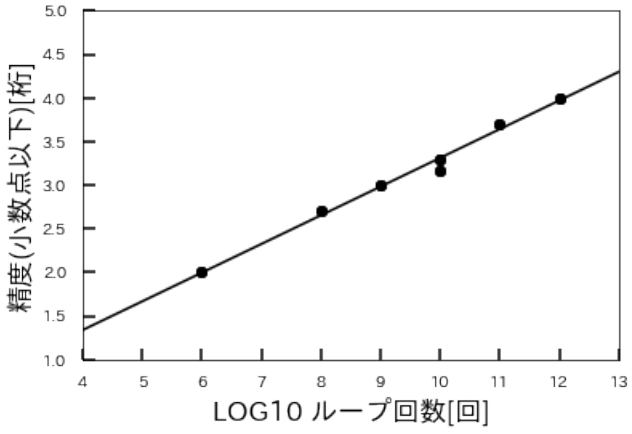

図 3: ループ回数と精度の関係

図 4 にループ回数と時間の関係を示す. 図 4 から, 時間はループ回数に比例することが読み取れる. 最小二乗法を用いてループが 1 増えたときの処理時間を計算すると,5.9 秒でることが分かった. 円周率を 1000 桁求めるためには,10^{43000} 回ループを回す必要があるのでこれに 5.9 秒をかけると現実的な時間ではないことが分かる. このことからもゼータ関数 $\zeta(s)$ が $s = 4$ のときのアルゴリズムで円周率を 1000 桁求めることは現実的ではないと考えられる.

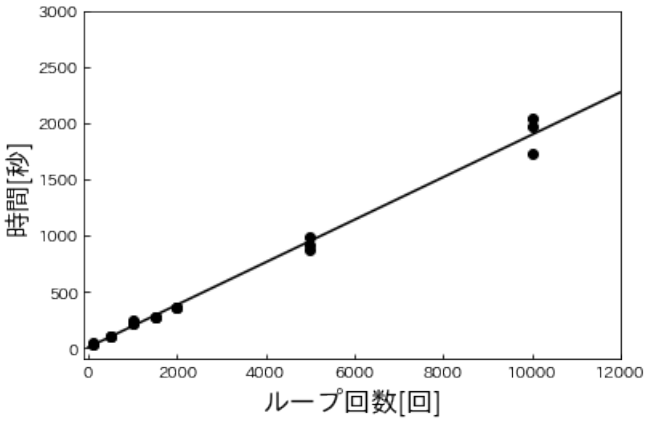

図 4: ループ回数と時間の関係

5 プログラムリスト

mulprec.c のプログラムをリスト 26 に示す. ただし, 授業で作成したプログラムの基数を 10000 に変更している.

リスト 26: mulprec.c のソースコード

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/time.h>
#include<math.h>
#include<limits.h>

#define KETA 150 //桁数
#define RADIX 10000 //基数

struct timeval tv;
double tstart,tend;

struct NUMBER{
    int n[KETA];
    int sign;
};

//aが0かをチェック
//0->return 0 else->return -1
int isZero(struct NUMBER *a){
    int i;
    for(i=0; i<KETA; i++){
        if(a->n[i]!=0){
            return -1;
        }
    }
    return 0;
}

// aの桁数を返す
int getKeta(struct NUMBER *a){
    int i,j,base,temp,acopy;
    base = RADIX/10;
    for(i=KETA-1; i>=0; i--){
        if(a->n[i]!=0){
            //printf("a->n[%d] = %4d\n",i,a->n[i]);
            acopy=a->n[i];
            for(j=0;j<(int)log10(RADIX);j++){
                temp = acopy/base;
                acopy %= base;
                if(temp!=0){
```

```
                //printf("j = %d\n",j);
                return (i+1)*(int)log10(RADIX)-j;
            }
            base/=10;
        }
    }
  }
  return 0;
}

//aに符号をセット
//aが0のときに-をセットしようとするとreturn -1
int setSign(struct NUMBER *a,int s){

  if((isZero(a)==0)&&(s==-1)){
    printf("aは0です\n");
    return -1;
  }

  if(s==1){
    a->sign = 1;
    return 0;
  }else if(s==-1){
    a->sign = -1;
    return 0;
  }else{
    return -1;
  }
}

//aの符号を取得
int getSign(struct NUMBER *a){
  if(a->sign==1){
    return 1;
  }else{
    return -1;
  }
}



//aをゼロクリア
void clearByZero(struct NUMBER *a){
  int i;

  for(i=0; i<KETA;i++){
    a->n[i] = 0;
  }

  setSign(a,1);
}


// aを全桁表示
void dispNumber(struct NUMBER *a){
  int i;

  if(getSign(a)==1){
    printf("+ ");
  }else{
    printf("- ");
  }

  for(i=KETA-1; i>=0; i--){
    printf("%04d ",a->n[i]);
  }
}


// aを全桁表示(ゼロサプレス)
void dispNumberZeroSuppress(struct NUMBER *a){
  int i;
  int j=1;

  if(getSign(a)==1){
    printf("+ ");
  }else{
    printf("- ");
  }

  if(isZero(a)==0){
    printf("0");
  }
  for(i=KETA-1; i>=0; i--){
    if(j){
      if(a->n[i] != 0){
        printf("%d ",a->n[i]);
        j=0;
      }
```

```
131         }else{
132         printf("%04d ",a->n[i]);
133         }
134     }
135 }
136 
137 //下位k桁に乱数を生成
138 void setRnd(struct NUMBER *a,int k){
139     int i;
140     clearByZero(a);
141 
142     for(i=0; i<k; i++){
143         a->n[i] = random()%(RADIX);
144     }
145     if(random()%2==1){
146         setSign(a,1);
147     }else{
148         setSign(a,-1);
149     }
150 }
151 
152 //aをbにコピー
153 void copyNumber(struct NUMBER *a,struct NUMBER *b){
154     *b=*a;
155 }
156 
157 //aの絶対値をbに代入
158 void getAbs(struct NUMBER *a,struct NUMBER *b){
159     copyNumber(a,b);
160     setSign(b,1);
161 }
162 
163 //aを10倍してbに代入
164 int mulBy10(struct NUMBER *a,struct NUMBER *b){
165     int i,temp;
166     int carry =0;
167     setSign(b,getSign(a));
168     if(a->n[KETA-1]/(int)(RADIX/10) != 0){
169         return -1;
170     }
171 
172     b->n[KETA-1] = a->n[KETA-1]*10;
173     for(i=KETA-2; i>=0;i--){
174         temp = a->n[i]*10;
175         carry = temp/RADIX;
176         b->n[i] = temp % RADIX;
177         b->n[i+1]+=carry;
178     }
179     return 0;
180 }
181 
182 //aを10000倍してbに代入
183 int mulBy10000(struct NUMBER *a,struct NUMBER *b){
184     int i;
185     setSign(b,getSign(a));
186     if(a->n[KETA-1]!=0){
187         return -1;
188     }
189 
190     for(i=0; i<KETA-1;i++){
191         b->n[i+1] = a->n[i];
192     }
193     b->n[0]=0;
194     return 0;
195 }
196 
197 //xをaに代入
198 int setInt(struct NUMBER *a,int x){
199     int temp,i;
200     if(KETA<3){
201         printf("KETAが足りません\n");
202         return -1;
203     }
204 
205     clearByZero(a);
206 
207     temp=x;
208     if(x<0){
209         temp*=-1;
210     }
211     for(i=0;i<3;i++){
212         a->n[i]=temp%RADIX;
213         temp-=a->n[i];
214         temp/=RADIX;
215     }
216 
217     if(x<0){
218         setSign(a,-1);
219     }else{
```

```
        setSign(a,1);
    }
    return 0;
}

//多倍長の大小比較 a==b:0 a>b:1 a<b:-1
int numComp(struct NUMBER *a,struct NUMBER *b){
    int i,Sa,Sb;
    Sa = getSign(a);
    Sb = getSign(b);

    // a+ b-
    if((Sa==1)&&(Sb==-1)){
        return 1;
    }

    //a- b+
    if((Sa==-1)&&(Sb==1)){
        return -1;
    }
    //a+ b+
    if((Sa==1)&&(Sb==1)){
        for(i=KETA-1; i>=0; i--){
            if(a->n[i] > b->n[i]){
                return 1;
            }

            if(a->n[i] < b->n[i]){
                return -1;
            }
        }
        return 0;
    }

    if((Sa==-1)&&(Sb==-1)){
        for(i=KETA-1; i>=0; i--){
            if(a->n[i] > b->n[i]){
                return -1;
            }

            if(a->n[i] < b->n[i]){
                return 1;
            }
        }
        return 0;
    }
    return 623;
}

int sub(struct NUMBER *a,struct NUMBER *b,struct NUMBER *c);
// c<- a+b
int add(struct NUMBER *a,struct NUMBER *b,struct NUMBER *c){
    int Sa = getSign(a);
    int Sb = getSign(b);
    struct NUMBER Nd,Ne;
    int d,i;
    int e=0;
    //a+ b+
    if((Sa==1)&&(Sb==1)){
        for(i=0;i<KETA;i++){
            d = a->n[i] + b->n[i] +e;
            c->n[i] = d%RADIX;
            e = (d - c->n[i]) /RADIX;
        }
        if(e==1){
            clearByZero(c);
            return -1;
        }
        setSign(c,1);
        return 0;
        }

        //a+ b-
    if((Sa==1)&&(Sb==-1)){
        getAbs(b,&Nd);
        sub(a,&Nd,c);
        return 0;
    }

    //a- b+
    if((Sa==-1)&&(Sb==1)){
        getAbs(a,&Nd);
        sub(b,&Nd,c);
        return 0;
    }

    //a- b-
    if((Sa==-1)&&(Sb==-1)){
        getAbs(a,&Nd);
        getAbs(b,&Ne);
```

```
310         add(&Nd,&Ne,c);
311         setSign(c,-1);
312         return 0;
313     }
314     return 623; //不明なエラー検出
315 }
316
317 // c<- a-b
318 int sub(struct NUMBER *a,struct NUMBER *b,struct NUMBER *c){
319     struct NUMBER Nd,Ne;
320     int Sa = getSign(a);
321     int Sb = getSign(b);
322     int i;
323     int h=0;
324     //a+ b+
325     if((Sa==1)&&(Sb==1)){
326     if(numComp(a,b)==1){
327         setSign(c,1);
328         for(i=0;i<KETA;i++){
329             a->n[i]-=h;
330             if(a->n[i]>=b->n[i]){
331                 c->n[i] = a->n[i] - b->n[i];
332                 h=0;
333             }else{
334                 c->n[i] = RADIX + a->n[i] - b->n[i];
335                 h=1;
336             }
337         }
338         return 0;
339     }
340
341     if(numComp(a,b)==-1){
342         for(i=0;i<KETA;i++){
343             b->n[i]-=h;
344             if(b->n[i]>=a->n[i]){
345                 c->n[i] = b->n[i] - a->n[i];
346                 h=0;
347             }else{
348                 c->n[i] = RADIX
349                  + b->n[i] - a->n[i];
350                 h=1;
351             }
352         }
353         setSign(c,-1);
354     }
355
356     if(numComp(a,b)==0){
357         clearByZero(c);
358     }
359     return 0;
360     }
361
362     //a+ b-
363     if((Sa==1)&&(Sb==-1)){
364         getAbs(b,&Nd);
365         add(a,&Nd,c);
366         return 0;
367     }
368
369     //a- b+
370     if((Sa==-1)&&(Sb==1)){
371         getAbs(a,&Nd);
372         add(&Nd,b,c);
373         setSign(c,-1);
374         return 0;
375     }
376
377     //a- b-
378     if((Sa==-1)&&(Sb==-1)){
379         getAbs(b,&Ne);
380         add(a,&Ne,c);
381         return 0;
382     }
383
384     return 623;//不明なエラー検出
385 }
386
387 // b<- a++
388 int increment(struct NUMBER *a,struct NUMBER *b){
389     struct NUMBER one;
390     int r;
391     setInt(&one,1);
392     r = add(a,&one,b);
393     return r;
394 }
395
396 // c<-a*b
397 int multiple(struct NUMBER *a,struct NUMBER *b,struct NUMBER *c){
398     int e,h,i,j;
399     int Sa = getSign(a);
```

```
    int Sb = getSign(b);
    struct NUMBER d,d2,ee;
    clearByZero(c);

    if(Sa==1 && Sb==1){
        for(i=0;i<KETA;i++){
            h=0;
            clearByZero(&d);
            for(j=0;j<KETA;j++){
                e=(a->n[j]* b->n[i])+h;
                d.n[j] = e%RADIX;
                e-=d.n[j];
                h = e/RADIX;
            }
            if(h != 0){
                clearByZero(c);
                return -1;
            }

            for(j=0;j<i*((int)log10(RADIX));j++){
                if(mulBy10(&d,&d2)==-1){
                    clearByZero(c);
                    return -1;
                }
                copyNumber(&d2,&d);
            }

            if(add(c,&d,c)==-1){
                clearByZero(c);
                return -1;
            }
        }
        return 0;
    }

    if(Sa==-1 && Sb==1){
        getAbs(a,&d);
        i = multiple(&d,b,c);
        if(i==-1){
            return -1;
        }
        setSign(c,-1);
        return 0;
    }

    if(Sa==1 && Sb==-1){
        getAbs(b,&d);
        i = multiple(a,&d,c);
        if(i==-1){
            return -1;
        }
        setSign(c,-1);
        return 0;
    }
    if(Sa==-1 && Sb ==-1){
        getAbs(a,&d);
        getAbs(b,&ee);
        i = multiple(&d,&ee,c);
        if(i==-1){
            return -1;
        }
        setSign(c,1);
        return 0;
    }
    return 623;
}

//割り算高速化
// a/b = c...d
int divide_2(struct NUMBER *a,struct NUMBER *b,struct NUMBER *c,struct NUMBER *d){
    struct NUMBER n,p,q,r;
    int over;
    int Sa = getSign(a);
    int Sb = getSign(b);
    clearByZero(c);
    clearByZero(d);

    if(isZero(b)==0){
        printf("div by 0\n");
        return -1;
    }

    copyNumber(a,&n);

    if(Sa==1 && Sb==1){
        while(1){
            if(numComp(&n,b)==-1){
                break;
            }
            copyNumber(b,&p);
```

```
490             setInt(&q,1);
491             while(1){
492                 over = mulBy10(&p,&r);
493                if(over == -1){
494                     break;
495                 }
496                 if(numComp(&n,&r)==-1){
497                     break;
498                 }
499                 copyNumber(&r,&p);
500                 mulBy10(&q,&r);
501                 copyNumber(&r,&q);
502             }
503             sub(&n,&p,&r);
504             copyNumber(&r,&n);
505             add(c,&q,&r);
506             copyNumber(&r,c);
507 
508         }
509         copyNumber(&n,d);
510         return 0;
511     }
512     if(Sa==1 && Sb==-1){
513         getAbs(b,&p);
514         divide_2(&n,&p,c,d);
515         setSign(c,-1);
516         return 0;
517     }
518 
519     if(Sa==-1 && Sb==1){
520         getAbs(a,&p);
521         divide_2(&p,b,c,d);
522         setSign(c,-1);
523         setSign(d,-1);
524         return 0;
525     }
526 
527     if(Sa==-1 && Sb==-1){
528         getAbs(a,&p);
529         getAbs(b,&q);
530         divide_2(&p,&q,c,d);
531         setSign(d,-1);
532         return 0;
533     }
534     return 623;
535 }
536 
537 // b<- sqrt{a} (Newton-Raphson法)
538 int sqrt_newton(struct NUMBER *a,struct NUMBER *b){
539     struct NUMBER x,temp,temp2,c,d;
540 
541     setInt(&d,2);
542     divide_2(a,&d,&x,&c);
543     if(isZero(&x)==0){
544         copyNumber(a,b);
545         return 0;
546     }
547 
548     if(getSign(&x)==-1){
549         return -1;
550     }
551 
552     copyNumber(&x,&d);
553 
554     while(1){
555         copyNumber(&d,&c);
556         copyNumber(&x,&d);
557 
558         divide_2(a,&d,&temp,&temp2);
559         add(&d,&temp,&temp2);
560         setInt(b,2);
561         divide_2(&temp2,b,&x,&temp);
562 
563         if(numComp(&x,&d)==0){
564             break;
565         }
566         if(numComp(&x,&c)==0){
567             if(numComp(&d,&x)==-1){
568                 copyNumber(&d,&x);
569             }
570         }
571     }
572     copyNumber(&x,b);
573     return 0;
574 }
575 
576 //zeta(4)から円周率を計算
577 //結果をaに代入
578 // int loop 反復回数
```

```
// int keta r/n^4のrの値
// int disp 途中経過の表示回数
int zeta4(struct NUMBER *a,int loop,int keta,int disp){

    if(loop<=0){
        printf("loopが不正な値です.\n");
        return -1;
    }

    if(keta>KETA*(int)log10(RADIX) || keta<=0){
        printf("ketaが不正な値です.\n");
        return -1;
    }
    if(disp>loop || disp<=0){
        printf("loopが不正な値です.\n");
        return -1;
    }

    int i,r;
    struct NUMBER n,b,temp,temp2,tenE;

    clearByZero(&b);
    setInt(&tenE,1);
    for(i=0;i<keta/(int)log10(RADIX);i++){
        r = mulBy10000(&tenE,&temp);
        copyNumber(&temp,&tenE);
    }

    for(i=0;i<keta%(int)log10(RADIX);i++){
        r = mulBy10(&tenE,&temp);
        copyNumber(&temp,&tenE);
    }

    i=1;
    gettimeofday(&tv,NULL);
    tstart =(double)tv.tv_sec + (double)tv.tv_usec *1.e-6;

    printf("多倍長桁数KETA = %d\n",KETA);
    printf("loop回数 = %d\n",loop);
    printf("keta = %d\n",keta);
    printf("途中経過表示 %d 回毎\n\n",disp);
    printf("----- start loop -----\n");
    while(1){
        //iをセット
        setInt(&n,i);
        //i^4を計算
        r = multiple(&n,&n,&temp);
        if(r==-1){
            printf("overflow in n^2\n");
            return -1;
        }
        multiple(&temp,&temp,&n);
        if(r==-1){
            printf("overflow in n^4\n");
            return -1;
        }

        divide_2(&tenE,&n,&temp,&temp2);
        add(&b,&temp,&temp2);
        copyNumber(&temp2,&b);

        if(i%disp==0){
            printf("pi^4/90[i = %d] = ",i);
            dispNumberZeroSuppress(&b);
            printf("\n\n");
            gettimeofday(&tv,NULL);
            tend =(double)tv.tv_sec + (double)tv.tv_usec *1.e-6;
            printf("所要時間 = %f秒\n\n",tend-tstart);
        }
        if(i==loop){
            break;
        }
        i++;
    }
    printf("----- finish loop -----\n");
    printf("pi^4/90 = ");
    dispNumberZeroSuppress(&b);
    printf("\n\n");

    setInt(&temp2,90);
    multiple(&b,&temp2,&temp);
    printf("pi^4 = ");
    dispNumberZeroSuppress(&temp);
    printf("\n\n");

    //平方根を取るために桁調整
    copyNumber(&temp,&temp2);
    r = getKeta(&temp2)%(int)log10(RADIX);
```

```
667     if(r!=4){
668         for(i=0;i<r; i++){
669             mulBy10(&temp2,&temp);
670             copyNumber(&temp,&temp2);
671         }
672     }
673 
674     // 平方根をとってpi^2を計算
675     //printf("平方根1回目直前桁数 : %d\n",getKeta(&temp2));
676     printf("----- calculate pi^2 -----\n");
677     sqrt_newton(&temp2,&temp);
678     printf("pi^2 = ");
679     dispNumberZeroSuppress(&temp);
680     printf("\n\n");
681 
682     //平方根を取るために桁調整
683     copyNumber(&temp,&temp2);
684     r = getKeta(&temp2)%2;
685     if(r==0){
686         mulBy10(&temp2,&temp);
687         copyNumber(&temp,&temp2);
688     }
689 
690     // 平方根をとってpiを計算
691     //printf("平方根2回目直前桁数 : %d\n",getKeta(&temp2));
692     printf("----- calculate pi -----\n");
693     sqrt_newton(&temp,&temp2);
694     /*printf("pi = ");
695     dispNumberZeroSuppress(&temp2);
696     printf("\n");*/
697 
698     gettimeofday(&tv,NULL);
699     tend =(double)tv.tv_sec + (double)tv.tv_usec *1.e-6;
700     printf("所要時間 = %f秒\n\n",tend-tstart);
701 
702     copyNumber(&temp2,a);
703     return 0;
704 }
705 
706 //aをファイルに書き込み
707 int fileNumberZeroSuppress(struct NUMBER *a){
708     char *fname = "calpi.txt";
709     FILE *fp;
710     int i;
711     int j=1;
712     if((fp = fopen(fname,"w"))==NULL){
713         printf("ファイルを開けませんでした\n");
714         return -1;
715     }else{
716         printf("\nresult write in %s ...",fname);
717         if(isZero(a)==0){
718         printf("0");
719     }
720     for(i=KETA-1; i>=0; i--){
721         if(j){
722             if(a->n[i] != 0){
723                 fprintf(fp,"%d",a->n[i]);
724                 j=0;
725             }
726         }else{
727         fprintf(fp,"%04d",a->n[i]);
728         }
729     }
730     printf("complete\n");
731     fclose(fp);
732     }
733     return 0;
734 }
735 
736 //pi.txtとcalpi.txtを比較
737 int calcheck(void){
738     char *fname1 = "pi.txt";
739     char *fname2 = "calpi.txt";
740     FILE *fp1;
741     FILE *fp2;
742     int ch1;
743     int ch2;
744     int i=0;
745 
746     if((fp1 = fopen(fname1,"r"))==NULL){
747         printf("ファイル %s を開けませんでした\n",fname1);
748         return -1;
749     }else{
750         if((fp2 = fopen(fname2,"r"))==NULL){
751         printf("ファイル %s を開けませんでした\n",fname2);
752         return -1;
753         }else{
754             while (( ch2 = fgetc(fp2)) != EOF ) {
```

```
755                 ch1 = fgetc(fp1);
756             //putchar(ch2);
757             //putchar(ch1);
758             if(ch1!=ch2){
759                 break;
760             }
761             i++;
762     }
763     printf("\n小数点以下 %d 桁一致\n",i-1);
764     fclose(fp2);
765     fclose(fp1);
766     }
767     }
768     return 0;
769 }
```

main.c のプログラムをリスト 27 に示す.

リスト 27: main.c のソースコード

```
1  #include<stdio.h>
2  #include "mulprec.h"
3
4  #define calpi
5
6  int main(void){
7      #ifdef calpi
8      struct NUMBER pi;
9      int loop = 500; //ループ回数
10     int keta = 130;
11     int disp =100; //disp回おきに途中経過を表示
12     int r;
13     r = zeta4(&pi,loop,keta,disp);
14     if(r==0){
15         printf("\n----- result -----\n");
16         printf("pi = ");
17         dispNumberZeroSuppress(&pi);
18         printf("\n");
19     //結果をファイルに書き込み
20     fileNumberZeroSuppress(&pi);
21     //検算
22     printf("\n----- 検算 -----\n");
23     calcheck();
24     }else{
25         printf("zata4関数を異常終了しました\n");
26     }
27     #endif
28
29     return 0;
30 }
```

参考文献

[1] Wolfarm Alpha,https://www.wolframalpha.com/