情報工学コース卒業論文

# ごみ集めを意識しない利用が可能なライブラリの設計法

2000年2月

西田 直樹

# 概要

本研究では、ユーザがごみ集めを意識せずに利用が可能なライブラリの設計法を提案し、それに基づいて、可変長整数のライブラリを作成した。そして、そのライブラリを用いたプログラミング、項書換え系（TRS）を入力としてそれと同等の動作をする C 言語プログラムを生成する Cdimple への組込みを行い、評価を行った。

まず最初に、プログラミングにおけるごみ集めの問題を取り上げ、その問題をユーザが意識しないで利用できるライブラリの設計法を明らかにする。ごみ集めには、大きく分けて、ポインタ変数を更新する際の旧データとライブラリ関数の合成の際に使われる中間データの処理がある。設計法としては、まず、資源を 1 箇所からしか参照しない場合について述べる。データ構造の設計に始まり、メモリ割当・ごみ集め関数の作成、最も重要である代入の処理、そしてライブラリ関数の作成の順で設計法を記述していく。そのあとに複数箇所から参照を許す場合に拡張した。

次に、提案した設計法に基づいて C 言語で可変長整数ライブラリを作成した。データ構造は、b-進数の各桁ごとに 1 つのセルに入れた双方向リストを基本として構成し、int 型整数や文字列の可変長整数への変換、可変長整数の表示、比較演算、四則演算のライブラリを作成した。

最後に、実際に可変長整数のライブラリを使って指定された桁数の $e$ の値を求めるプログラムの作成、ならびにこのライブラリの Cdimple への導入を行った。その結果、ごみが蓄積されないことが確認された。また、通常のごみ集めの場合と比較した手間の評価を行い、本手法が十分有効であることを明らかにした。

# 目 次

# 第1章　はじめに

リストなどの可変長の資源を用いるプログラムにおいて、もっとも厄介な問題の一つは「ごみ集め」である。「ごみ集め」とは不要になったデータの記憶領域を回収することである。C 言語などの通常の言語ではごみの回収をプログラマがプログラム中に指示しなければならず、指示の誤りは時として重大なエラーを起こす。また、不要になった資源がきちんと回収されていないと、無駄にメモリを消費してそのプログラムの動作の限界が狭まってしまうことにもつながる。簡単なテストプログラムにおいてはあまり問題が起こらない可能性があるが、実用プログラムの場合には、実際にはポインタ変数を使用するだけでもごみ集めの問題が生じることになる。

次のような場合を考えてみよう。ここで、x, y はポインタ変数とし、関数 f, g はポインタを返すものとする。

```
x = f( 1 );
y = g( x, 0 );
x = g( x, f( y ) );
 ...
```

このとき、f(1), f(y) の値を保持するセルがどこに行ったかを考えると、これは指定するポインタがなくなり回収不能なごみになってしまっている。これらをごみとして残さないようにするには、一時変数を導入してポインタ変数に他の番地が代入される前に解放してやらなければいけない。そのためには、プログラムの行数は増え、デバッグも困難になる。特に、関数の合成の際の中間データの処理は厄介である。関数の合成は頻繁に行なわれるにもかかわらず、その中間データの処理は、意外に忘れられがちである。さらに、このようなプログラミングにおけるごみ集めを一時変数を使って毎回記述するとなると、かなりの手間になる。

そこで、本論文では、ユーザ[1]が「ごみ集め」を意識せずに利用が可能なライブラリの設計法を提案する。そして、提案した設計法に基づいて C 言語で可変

[1]本論文で用いる「ユーザ」とは本論文で提案した設計法に基づいて作成されたライブラリを利用してプログラミングを行う人を指す。

長整数ライブラリを作成し、そのライブラリを用いたプログラムの例、Cdimple への組込みを行った上で、提案した設計法に対する評価を行う。

第 2 章では、ごみ集めにおける問題を明示し、それを解消する設計法を明らかにする。第 3 章では、提案した設計法に基づいて作成した可変長整数ライブラリについて述べる。そのあと、第 4 章で、可変長整数ライブラリを用いたプログラムの例を示し、提案した設計法がごみ集めとその記述に対して有効であることを明らかにする。

# 第2章　ライブラリの設計法

## 2.1　ごみ集めに関する問題点

実際にC言語による複素数を扱うプログラム（付録A）について考える。ここでは、問題点をわかりやすくするために可変長の資源を用いていない例を取り上げた。

今、次のような処理を行ったとする。

```
1: CPLX *x, y;
2: x = make( 1, 2 );            /* x : 1+i2 */
3: y = make( 3, 4 );            /* y : 3+i4 */
4: y = add( y, negate( x ) );   /* y : 2+i2 */
5: ...
```

このとき、5行目以降で参照される可能性があるのは変数xが保持するデータ（ $1+i2$ ）と変数yが保持するデータ（ $2+i2$ ）であり、データ $3+i4$ については、それを指すポインタ変数がなくなるため、これはごみとして残る。さらに、4行目の関数addの引数であるnegate(x)の戻り値であるデータ $-1-i2$ も、関数addで利用された後は、どこからも参照されずにごみとなっている。

このように回収不能なごみが残ってしまった原因は、4行目でyの値を更新する前に旧データ（ $3+i4$ ）を解放しなかったこと、4行目の手前で関数addの引数の値（negate(x)）を一時変数に格納し、4行目の後、一時変数の指すデータ（ $-1-i2$ ）を解放しなかったことである。また、4行目の関数addの引数にyが使われるために、yにも一時変数を用意しなければならない。

そこで、ごみを残さないようにするためには次のように記述しなければならない。ここで関数freeは引数の指すデータのメモリを解放する標準ライブラリ関数である。

```
1: CPLX *x, y, p, q;
2: x = make( 1, 2 );          /* x :  1+i2 */
3: y = make( 3, 4 );          /* y :  3+i4 */
```

```
4: p = negate( x );          /* p : -1-i2 */
5: q = y;                    /* q :  3+i4 */
6: y = add( q, p );          /* y :  2+i2 */
7: free( p );
8: free( q );
9: ...
```

このようにするとごみは残らなくなるが、最初のプログラムに比べて記述する量は格段に増える。また、ユーザは絶えずごみ集めを意識していなければならず、ごみの回収を忘れていても気付かない恐れもある。さらに、完成したプログラムがごみを残していないかを確認することさえ困難である。

## 2.2 設計の方針

ここでユーザが行わなければならないごみ集めをまとめてみる。1つ目はポインタ変数を更新する際の旧データの処理（解放）である。前の例では $3+i4$ の処理にあたる。2つ目はライブラリ関数の合成の際に使われる中間データの処理である。前の例では $-1-i2$ と代入される y のデータを一時変数に格納し、引数として参照した後に解放した一連の処理にあたる。大きく分けて、この２つのデータの処理をプログラマが指定しなければならない。

しかし、これら２つのデータの処理をユーザが考える必要がなくなれば、このライブラリを利用したプログラミングが断然楽になる。そこで本節では、これら２種類のごみ集めの問題をユーザが意識せずに利用が可能なライブラリの設計の方針を明らかにする。

本節の設計の方針、次節の具体的な設計法では、可変長の資源は複数箇所から参照されない[1]という制限のもとに議論を進める。その後、1つのデータを複数箇所から参照することを許す場合における設計法への拡張について簡単に述べる。

### 2.2.1 ポインタ変数を更新する際の旧データの処理

一般に、ポインタ変数の値を更新するには、

1. 旧データの領域へのポインタを一時変数に確保する[2]。

[1] 具体的には、1つのデータを指すポインタを持つ変数は1つしかないということ。

[2] これは、新しいデータを生成する際に更新前の旧データが参照される可能性があるためであ

2. 更新される新しいデータ領域へのポインタに更新する。
3. 一時変数に格納してある旧データの領域を解放する。

という一連の操作が行われる。そこで、この一連の操作を通常の代入と同等の手間で記述するための機能を用意する。代入の操作の記述が長くなる可能性が高いことと、後に述べるいくつかの問題があるため、上記の一連の操作は代入関数を用意して、マクロでポインタ変数がすでにデータ領域へのポインタを保持しているかを判定し、まだならば領域を確保してから代入関数を呼び出すように代入文を定義する。その判定を行うために、ポインタ変数を宣言したときに初期化を行うよう制約を設ける。

### 2.2.2 ライブラリ関数の合成の際の中間データの処理

ライブラリ関数に与えられる引数は、ごみ集めの観点から見ると、2 種類のものが考えられる。1 つは変数にポインタが格納されているデータであり、もう 1 つは他の関数の戻り値であって変数にポインタが格納されていないデータである。後者のデータは呼ばれたライブラリ中でごみ集めをしない限り、回収は不可能である。しかし、前者のデータはライブラリ終了時にはごみではないため、ここで処理をするわけにはいかない。

そこで、各ライブラリ関数は、結果を返す前に、引数で他のライブラリ関数の戻り値であるものだけを解放するようにしなければならない。それには、引数のデータが “変数” であるのか “ライブラリ関数の戻り値” であるのかという情報が必要になる。具体的には、その情報をデータ構造に追加してやればよい。それによって各ライブラリ関数は、結果を返す前に引数のデータが “変数” か他の “ライブラリ関数の戻り値” かを判断して、“戻り値” ならば解放するようにする。

ただし、これだけでは問題を引き起こす場合がある。それは、ライブラリ関数の中で他のライブラリ関数を呼び出す場合である。もし、引数が “関数の戻り値” であるならば、呼び出した関数の中でそのデータが解放されてしまう可能性があり、そうした場合に、その関数から戻ってきた後で、参照しようとしたデータがすでに解放されてしまっているという問題が起こる。これを解決するために、各ライブラリ関数の中では、引数が “変数” か他の “ライブラリ関数の戻り値” のどちらであるかの情報は一時的に保持しておき、その関数の中では “変数” として

---

る。

振舞うようにすればよい。具体的には、関数の冒頭で引数の情報[3]を一時変数に保持した後、引数の情報を “変数” にし、関数の最後で保持していた引数の情報を戻して “関数の戻り値” であれば解放する。また、各ライブラリ関数は戻り値のデータの情報を “関数の戻り値” に設定する。

これにより、前節で述べた代入関数において、代入されるデータの情報を “変数” に設定し直す作業が 1 つ増えることになる。その作業とは、代入が完了した後、つまり変数が新しいデータへのポインタに更新された後に、指定するデータの情報を “変数” に変更することである。

## 2.3　ライブラリの設計法

本節では、具体的な設計法について述べる。最初に、ライブラリ関数を作成する前に行われるデータ構造・ユーティリティ関数の設計について記述し、それから、本設計法において重要な位置を占める代入関数の設計を明らかにした後、ライブラリ関数やユーザ作成関数の設計について述べる。また、扱いやすさの理由から可変長の資源に対してはポインタを使用して操作を行うこととする。

### 2.3.1　データ構造の設計

まずは、可変長の資源のデータ構造を設計することから始める。この時に、そのデータ領域を指すポインタが “変数” であるのか “関数の戻り値” であるのかの情報を格納するタグ[4]を付随しておく。これ以降、“変数” であるものは REF[5] **タイプ**、“関数の戻り値” であるものは NONREF[6] **タイプ**、またそれらの情報を**参照情報**と呼ぶこととする。扱う可変長の資源のデータ構造の特徴によっては、参照情報のタグは先頭のセル、つまりポインタ変数が直接指定するセル（データ構造）にのみ付けておけばよい。

複素数のライブラリの例では、次のようになる。

```
typedef struct complex{
    int real;
```

[3] “変数” か他の “ライブラリ関数の戻り値” のどちらであるかの情報。
[4] 具体的には、整数型のデータを 1 つ設ける。
[5] Refered
[6] Non-refered

```
    int img;
    int state;
} CPLX;

#define REF    0
#define NONREF 1
```

### 2.3.2 メモリ割当・ごみ集めを行う関数

次に作成するのは、メモリ割当・ごみ集め（メモリ解放）を行う関数である。これらの作業は、わざわざ関数を作成して行わなくともよいと思われるかもしれないが、そうすることにはいくつかのメリットがある。単純に組込みのメモリ割当関数・メモリ解放関数を直接使ってもよいが、データ構造が複雑な場合[7]にはいちいちそれらすべてを正確に処理するようにプログラミングを行わなければならないし、メモリ割当は毎回新しくメモリを割当て、メモリ解放では毎回解放されることになる。それでもごみを残さないようにできるが、従来のごみ集め[8]はできない。さらに、ユーザがメモリ割当・メモリ解放を行うのが難しいことになる。それは、ユーザは詳しいデータ構造などは知らない可能性があるためである。

このような理由により、メモリ割当・ごみ集め（メモリ解放）を行う関数をライブラリに用意しておくべきである。また、この 2 つの関数があることにより、後からごみ集めの方法を変更することが容易になる[9]。

メモリ割当関数は、データ構築用の関数と兼ねてもよい。引数にどのようなデータを構築するかの情報を取り、初期化の代わりにそれらの値を入れてやる。そうすることにより、わざわざデータを構築するための関数を用意する必要がなくなる。しかし、データ構造が複雑な場合[10]はデータを構築するための関数を用意したり、それぞれのライブラリ関数の中でデータの中身を直接操作して、データを構築することになる。実装方法は、関数として作成するか、内部で変数を宣言しないならばマクロで定義してもよい。

リストなどデータが連結するような構造になる場合は、ごみ集め関数は複数用意する必要がある。これは、最上位のデータは、この後に述べる代入の操作など

[7] 例えば、データがリストになっている場合など。
[8] 不用になったセルをリストにしておき、新しく割当てる時はそのリストから取ってくる方法。
[9] 基本的に、これら 2 つの関数内を変更するだけでよい。
[10] 引数の情報だけでは構築できないようなもの。データが可変長のリストになる場合などが考えられる。

において、特別に扱われるためである。そのため、少なくともごみ集め関数は、最上位のデータのみを処理するものと、最上位のデータに連結するデータ群を処理するものが必要になる。そして、ごみ集めを行う際は、これらを適切に使い分けなければならない。また、引数のポインタ変数の値が初期値（例えばNULL）ならば、何もしないようにする。

また、処理（ごみ集め）されたデータへのポインタ変数は初期値に設定しておいたほうがよい。これは、プログラミングの際誤ってポインタ変数にすでに処理されたデータへのポインタが格納されたままでその変数を使用しようとした場合、エラーが生じるためである。ごみ集め関数が必ず適切な場所で使用されているなら必要ないことだが、実際にはプログラミングのミスがあり得るので、この操作を加えておくほうが作成が楽になる。実装方法は、引数で指定されたメモリを処理[11]する関数を用意して、マクロで実際に使用するごみ集めのコマンドを定義し、その中で作成したごみ集め関数を呼び出し、その後に、引数のポインタ変数を初期値にする。このとき、引数がNONREFタイプならば、マクロの中で2回呼び出され計算されてしまうので、ごみ集め関数の戻り値として、引数の参照情報を返し、REFならば初期値に設定するようにする。ただし、これは参照情報を持つデータ型の処理を行うごみ集め関数に対してのみ考慮しておけばよい。

複素数の例では、次のようにごみ集めの関数を追加することになる。また、メモリ割当の関数は既存の関数makeを修正する。この場合、データ領域は連結していないので、ごみ集め関数は1つでよい。

```
CPLX *make( int x, int y ){
    CPLX *z;
    z = (CPLX *) malloc( sizeof( CPLX ) );
    z->real = x;
    z->img = y;
    z->state = NONREF;
    return( z );
}

#define free_cplx( x )  ({ \
    if( free_cplx0( (x) ) == REF ) \
        (x) = NULL; \
})

int free_cplx0( CPLX *x ){
```

[11]解放やごみ集め。

```
    int state;
    state = x->state;
    if( x != NULL )
        free( x );
    return( state );
}
```

### 2.3.3 データ構築関数

データ構築に関しては、メモリ割当のところでも述べたように、構造が簡単なものに関してはメモリ割当関数と兼ねることができる。しかし、状況に応じてデータのリストの個数などが変わるような複雑なデータ構造の場合には、データ構築関数は別に用意しなければならない。しかし、この関数はデータ構造の特長によって異なるため一般的な設計法を提案することは難しい。そのため、データを構築する場合は、そのデータ構造に合わせて必要な関数を用意することとする。また、それぞれのライブラリ関数内で、データの一部を直接走査する場合には適切に処理するように注意する。そして、なるべくユーザにはデータの中身を直接触る必要がないようにしなければならない。

### 2.3.4 代入定義文・代入関数

次に最も重要になるであろう代入を行う関数を作成する。最も重要であるという理由は、代入の操作には2つのごみ集めの処理と順序が要求されるからである。2つのごみ集めとは、代入されるポインタ変数がすでに指定している旧データの処理、代入するデータの処理である。また、ごみ集めの順序とは、いつ旧データを処理するかということである。さらに、代入されるポインタ変数がすでにメモリ割当されているか、代入するポインタ変数が初期値でないかも考慮しなければならない。

ここで、使用する言語における関数の引数の “**値呼び**（call by value）”、“**参照呼び**（call by reference）” の問題ついて考える。なぜかというと、“値呼び” の場合ポインタ変数がすでにメモリ割当されていなければ、うまく代入が行えないためである[12]。C 言語などがそうである。それに対して、“参照呼び” の場合は問

[12] この場合、関数の中でメモリを割当てても、関数から出たときにポインタ変数の指定するアドレスがその関数を呼び出す前の元の値に戻り、そのポインタ変数が指定するデータは解放され

題ないが、後で述べるいくつかの問題があるために、最終的には “値呼び”、“参照呼び” のどちらの場合も同じ設計法を取ることになる。そのため、ここでは関数の引数は “値呼び” されるものとして議論を進める。

それでは、具体的な作成方法を述べる。

ユーザによるライブラリ使用時やライブラリ作成時に使用する代入コマンドはマクロで定義する。これを本設計法では、**代入定義文**と呼ぶこととする。C 言語で言えば、「手続き型引数付きマクロ」にあたる。また、代入定義文の中で使われる実際に代入を行う関数を**代入関数**と呼ぶ。

代入の処理手順は次のようになる。

1. 代入されるポインタ変数がすでに使用されているか[13]を判定[14]し、まだ使用されていなければメモリを割当てる。
2. 代入関数を呼び出し、代入が失敗[15]ならば代入されるポインタ変数の指定するデータ領域を解放し初期値に設定する[16]。代入関数は、引数は代入定義文と同じものとし、次のように処理を行うように作成する。
   (a) 代入するポインタ変数の値が初期値ならば、代入失敗として指定されたフラグ（例えば”FALSE”）を返す。
   (b) データが連結している構造ならば、旧データに連結しているデータのみを解放する[17]。
   (c) 代入されるデータが NONREF タイプならば、組込みの代入で代入するデータのトップのセル[18]を代入されるポインタ変数の指定するデータ領域（セル）に代入（複写）[19]し、参照情報を REF タイプに設定、代入したデータのトップのセルのみ回収する。REF タイプならば、代入されるポインタ変数の指定するデータ領域を基にして、代入するデータと同じ内容のデー

---

てしまったであろう旧データになり、さらに代入するデータがどこからも指定されずごみになる。

[13]すでにメモリ割当されてデータを指定するポインタの値を持っているかどうかということ。

[14]ポインタが初期値（例えば NULL）かどうかで判定する。

[15]代入関数が呼び出されたとき、代入されるポインタ変数には必ずデータ領域へのポインタが割り当てられている。そのため代入するポインタ変数の値が初期値であるならば代入はできないので、代入の操作は失敗することになる。

[16]代入関数の戻り値で判定する。

[17]ポインタ変数が直接指定しているデータ領域は新しいデータ領域の値に書換えられるため、解放してはいけない。

[18]引数のポインタが指定しているデータ。

[19]ポインタを代入するのではなく、データ領域自身を代入する。これは、“値呼び” では、変数が指定するポインタの値を更新することはできないためである。

タ構造をもう 1 つ構築する。

(d) 代入が成功したので、指定されたフラグ（例えば”TRUE”）を返す。

ここで、ポインタ変数にすでにメモリが割当てられているかを判断するために、ポインタ変数が初期値であるかどうかを用いている。そのために、ポインタ変数を宣言する際に初期化しておくという制約が必要になる。このことを作成したライブラリの使用書に記述しておかなければならない。また、メモリが割当てられているかの判断に初期値を用いた理由は、宣言した際にポインタ変数の値は不定であるために偶然にも使用済みのポインタの値と重複し、どのような判定をするにしろ判断できなくなるという可能性があるためである。そのため、判定方法はいろいろあり得るだろうが、初期値を用いる判定が最も確実な方法と言えるだろう。

また、わざわざ代入関数を作成することには理由がある。もし代入定義文のみで作成した場合に、注意が必要な点がある。まず 1 つは、宣言した変数の名称である。代入定義文はマクロで定義するために、実際には使用した場所に定義した内容が置き換えられる。その時、宣言した変数と同じ名称の変数の有効範囲が、その代入定義文が使用される場所に重なっているとエラーが生じる。これを解決するには、代入定義文の中で使用する変数は他では使われないようなオリジナリティのあるものにすることが考えられるが、変数の名称の重複によるエラーが生じる可能性は残ってしまうことになる。このエラーの可能性を完全に無くすためには、代入定義文の中で変数を宣言しなければよい。そうするために代入関数を用意するのである。また、他に代入の機能を持つ関数がある場合、その代入関数内で、内部で代入関数を呼び出していない代入定義文を使用するとうまく代入が行われない。これは、代入定義文ではポインタ間、代入関数ではポインタの指定するデータ間で代入を行なっている違いがあるためである。そのため、代入関数の中では、内部で代入関数を使用していない代入定義文を使用してはならない。これは、必ず内部で代入関数を使用するように統一することで解決できる。しかし、一時変数の名称の問題があるために必ず代入関数を使用することになるので、本設計法に基づいて作成する限りは、これは問題にならない。

以上のことから、代入定義文は必ず代入関数とセットで作成し、内部では代入関数を使用するようするように設計すべきであると言えることになる。

また、代入するポインタ変数の値が初期値か判定する作業は代入定義文中で行なってもよさそうに思われるかもしれないが、そうした場合、代入するポインタ

変数が “関数の戻り値” の場合、初期値かどうかの判定と代入関数の引数の 2 箇所で使用されることになり、1 回余分に計算されてそのごみが残る。代入定義文中では、前述したように変数を宣言して使用することはできないし、代入するポインタ変数は 1 つしか記述してはいけない。そのため、代入関数の中で判定を行なうようにしたのである。

また、すでにメモリを指定しているポインタ変数に初期値を代入することはできないので、代入する値が初期値の場合、代入関数は代入を失敗したとするのは理にかなっている。

もし、使用する言語が “参照呼び” でも、前述したような問題があるため、マクロによる定義は使わず関数として作成したほうがよい。

複素数の例では、次の代入定義文 assign をヘッダーファイルに、代入関数 assign0 をライブラリ関数ファイルに追加する。

```
#define FALSE 0
#define TRUE  1

#define assign( x, y ) ({ \
    if( (x) == NULL ) \
        (x) = make( 0, 0 ); \
    if( assign0( (x), (y) ) == FALSE ){ \
        free_cplx( (x) ); \
        (x) = NULL; \
    } \
})

int assign0( CPLX *x, CPLX *y ){
    if( y == NULL )
        return( FALSE );
    if( y->state == REF ){
        *x = *make( y->real, y->img );
    }else{
        *x = *y;
        x->state = REF;
        free_cplx( y );
    }
    return( TRUE );
}
```

### 2.3.5 ライブラリ関数の作成

その後、必要に応じてライブラリ関数を作成していく。その際に、次の点を考慮して作成を進める。14 ページに記載した複素関数ライブラリを用いて作成したユーザ関数 sub と比較しながら読んでもらうと理解しやすいだろう。コメントの番号が次のそれぞれの考慮点に対応している。

1. 引数に作成したデータ構造へのポインタ変数がある場合、関数の冒頭で一時変数に参照情報を保持し、引数のポインタ変数の参照情報は REF タイプにする。[20]そして、関数の最後で、引数のポインタ変数の指定するデータの参照情報を一時変数から戻し、NONREF タイプのデータは解放する。
2. 関数の戻り値が作成したデータ構造型のデータであるならば、戻り値のデータを NONREF タイプに設定する。
3. 関数内で宣言した局所ポインタ変数は、不要であることが明らかであるものを関数の最後ですべて解放しておく。この時、局所ポインタ変数がこの後使用するデータ（例えば、戻り値）へのポインタと重複していないことに注意しなければならない。

また、これらのことは、ユーザがライブラリを用いてユーザ関数を作成する場合にも考慮されなければならない。そのため、これらの点もライブラリの使用書に「関数を作成する際の注意点」として記述しておく。

複素数のライブラリの例では、関数 add は次のように修正する。関数 add では、内部で”CPLX *”型の引数をとる関数を使用しないために参照情報の保持は行わなかった。

```
CPLX *add( CPLX *x, CPLX *y ){
    CPLX *z = NULL;
    z = make( x->real + y->real, x->img + y->img );
    if( x->state == NONREF )
        free_cplx( x );
    if( y->state == NONREF )
        free_cplx( y );
    z->state = NONREF;
    return( z );
}
```

[20] これは、関数内部で別の関数を呼び出した場合に呼び出した関数内でまだ参照される可能性のある引数であったポインタ変数の指定するデータが解放されるのを防ぐためである。

さらに、複素数ライブラリを用いて新しく減算を行う関数subを作成すると次のようになる。ここでは、前述の考慮点がわかりやすいようあえて局所変数を宣言して一時変数として利用した。

```
CPLX *sub( CPLX *x, CPLX *y ){
    CPLX *z = NULL, *neg = NULL;
    int state1,state2;                      /*  1.  */

    state1 = x->state;                      /*  1.  */
    x->state = REF;                         /*  1.  */
    state2 = y->state;                      /*  1.  */
    y->state = REF   ;                      /*  1.  */

    assign( neg, negate( y ) );
    assign( z, add( x, neg ) );

    x->state = state1;
    y->state = state2;
    if( x->state == NONREF )                /*  1.  */
        free_cplx( x );                     /*  1.  */
    if( y->state == NONREF )                /*  1.  */
        free_cplx( y );                     /*  1.  */

    z->state = NONREF;                      /*  2.  */

    free_cplx( neg );                           /*  3.  */

    return( z );
}
```

### 2.3.6　ユーザ作成関数内でのごみ集め

ここまで述べてきた設計法の中で、まだ回収されていないごみが存在する。それは、ユーザが作成した関数の中のポインタ変数の指すデータ領域である。しかし、このごみはライブラリが自動的に回収することは難しい。そこで、ユーザは、自分で作成した関数では、関数の最後でその関数の冒頭で宣言したポインタ変数のうち、もはや参照しないことが明らかなもの[21]を解放するように指示しておく必要がある。

[21]おそらく関数の戻り値以外のものが該当する。

また、このことはライブラリ関数の作成の際にも述べたことで、不要になった領域をきちんと回収するように設計しなければならない。

### 2.3.7 ライブラリの使用における制約

ここでは、ライブラリの使用における制約をまとめる。この制約をユーザにわかるようライブラリの使用書に必ず記述しておかなければならない。

- ポインタ変数を宣言する際に、初期化する。
- ユーザが作成する関数における引数にライブラリが用意したデータ型を取る場合、その関数内では一時変数にその参照情報を保持し、引数はREFタイプに設定し、関数の最後で参照情報を戻し、NONREFタイプの引数は解放する。
- ユーザが作成する関数における戻り値がライブラリが用意したデータ型であるときは、戻り値の参照情報をNONREFタイプに設定する。
- 関数を作成したときは、その関数の最後で、宣言した局所ポインタ変数の指定するデータで明らかに不要のもの（おそらく戻り値以外のものにあたる）を解放しておく。

## 2.4 複数箇所からの参照を許可する場合への拡張

それではここで、前節で提案した設計法を、データを複数箇所から参照することを許可する場合への設計法に拡張する。この拡張は容易であることが、この後の文章を読んでもらうと理解してもらえるだろう。

### 2.4.1 参照カウンタの導入

データ構造においては、前節の設計法で使用した参照情報の代わりに、そのデータ領域が何箇所から参照されているかという情報（**参照カウンタ**）を追加する。参照カウンタは、そのデータを指定するポインタが格納されている箇所の個数を表す。実装としては、参照情報のタグの名前を変更してやればよい。参照カ

ウンタの処理は、代入のときは、代入して新しいデータのカウンタを1増やしてから旧データのカウンタを1減らし、関数の引数として参照されるときは、その関数の冒頭で1増やし最後に1減らす。また、関数の戻り値に関しては何もしない。これは関数の戻り値を指定するポインタがまだ存在しないか、もしくは増えていないとして考えるからである。参照カウンタを1減らす関数を用意し、カウンタが0になったものは回収する。この方法では代入用の関数は、前述の参照情報の場合より作業が簡単なる。

### 2.4.2　ごみ集め関数の変更

参照カウンタを用いる場合には、ごみ集め関数の代わりに参照カウンタを操作する関数を作成する。この関数の中では、引数の指定するデータのカウンタを1減らし、0になった場合に、そのデータ領域を処理（解放）する。このとき、引数のポインタ変数の値が初期値（例えば “NULL”）ならば、何もしない。

また、処理されたデータへのポインタ変数を初期値にする。理由は、この後で変更される代入関数において、代入されるポインタ変数がすでにデータを指しているかを判定できるようにするためである。もし、初期値に設定し直されなければ、代入の際にもうすでに回収されたデータを再度回収しようとしてエラーが起こるからである。この関数は参照情報の場合と同様に、マクロで定義する[22]。しかし、実際には、一度ポインタを保持した変数または格納箇所は、別のポインタが代入されるか、もしくは意図的に初期値（例えばNULL）に設定されるので、カウンタ操作関数では強いてポインタ変数を初期値に戻す必要はない。そのため、関数で作成してもよいが、万が一エラーになる状況があるかもしれないのでカウンタが0になったデータを指定するポインタは初期値に設定し直すためマクロと関数で作成する。ごみ集め関数からの変更点としては、マクロで定義したコマンドは名前のみ変更し、関数の内部をカウンタ操作用に変更すればよい。

### 2.4.3　代入定義文・代入関数の変更

代入定義文は、次のように変更する。

[22]（“値呼び” の言語では、）関数で設計すると、参照カウンタが0になってデータが回収された時に、引数のポインタ変数を再び初期値に設定できない。

1. 組込みの代入（＝）を使用して、代入されるポインタ変数に代入関数の戻り値を代入する。
2. 新しいポインタが指定するデータの参照カウンタを 1 増やす。

また、代入定義文で使用する代入関数は次のように作成する。

- 引数は、代入定義文と同じで、代入されるポインタ変数と代入するポインタとする。
- 戻り値は代入するポインタの値。
- 代入されるポインタ変数の指定するデータの参照カウンタを 1 減らす。

このように作成するのは、代入されるポインタ変数の指定する旧データのカウンタを 1 減らすためである。代入されるポインタ変数の指定する旧データは、代入するポインタの指定するデータを求める際に参照される可能性があるので、先にカウンタを減らしてはいけない。先の一時変数の名称の問題があるので、代入定義文中で一時変数に保持するわけにもいかない。そのため、わざわざ関数を用意することとした。

### 2.4.4 ライブラリ関数の作成における変更

ライブラリ関数作成の際考慮すべき点は、次のように変わる。

- 引数に作成したデータ構造へのポインタ変数がある場合、最初にすべての引数の参照カウンタを 1 増やし、最後に関数を出る前に引数の参照カウンタを 1 減らす。
- 関数内で宣言した局所ポインタ変数は、戻り値になるもの以外すべて、関数の最後で参照カウンタを 1 減らす。
- 代入定義文以外で、構築されたデータについては、データとして使用できるようになった時点で参照カウンタを 1 増やす（1 に設定する）。ただし、戻り値になるもののカウンタは操作しない。

戻り値の参照カウンタがおかしくならないように、引数のカウンタの増減以外は参照カウンタをむやみに操作しないように注意しなければならない。これはユーザがライブラリを用いて関数を作成する際にも言える。

表 2.1: 参照情報を用いた場合と参照カウンタを用いた場合の対応箇所

| 位置 | 参照情報の場合 | 参照カウンタの場合 |
|---|---|---|
| データ構造の定義部分 | タグを追加 | タグを追加 |
| 初期値 | REF タイプ | 0 |
| 関数の冒頭 | 引数のタイプを保持 | 引数の参照カウンタを1増やす |
| 関数の最後 | 引数で NONREF タイプのものを解放 | 引数の参照カウンタを1減らす |
| 関数の戻り値 | NONREF タイプに設定 | 何もしない |
| ごみ集め | 関数を用意し、メモリを処理する | 関数を用意し、カウンタが 0 になったものを処理する |
| 局所変数 | 戻り値以外解放 | 戻り値以外カウンタを 1 減らす |

また、データの構造を操作する関数では、戻り値の参照ウカウンタ操作により一層の注意をする。

### 2.4.5 参照情報と参照カウンタにおける相互関係

ここまでの設計法において、参照情報と参照カウンタを用いる場合の両方に、非常に密接した相互関係があることがわかる。データ構造においてはともに1つのタグを追加しているので、その名前が異なる違いしかない。また、それぞれの関数内で操作を記述する箇所も同じである。表 2.1 に、その対応箇所をまとめる。これらの箇所を変更するだけで、容易に参照情報を用いる場合と参照カウンタを用いる場合を変更することができる。

### 2.4.6 ライブラリの使用における制約

ここでは、ライブラリの使用における制約をまとめる。この制約をユーザにわかるようライブラリの仕様書に必ず記述しておかなければならない。

- ポインタ変数を宣言する際に、初期化する。
- ユーザが作成する関数における引数にライブラリが用意したデータ型を取る場合、関数の最初で、引数のカウンタを 1 増やし、最後にカウンタを 1 減らす。
- ユーザが作成する関数における戻り値がライブラリが用意したデータ型であるときは、戻り値のカウンタは操作しない。
- 関数を作成したときは、その関数の最後で、宣言した局所ポインタ変数の指定するデータで明らかに不要のもの（おそらく戻り値以外のものにあたる）のカウンタを 1 減らす。

### 2.4.7 設計法の選択

では、参照情報と参照カウンタのどちらの設計法を選べばよいか。それは可変長の資源の性質による。例えば、前述の複素数のライブラリの場合は、参照情報を持たせるほうがよい。理由は、可変長の資源である複素数のデータが数値であるためにめまぐるしく値が変わリ、また、別の変数として同じ値のものはあまり使わないだろうからである。定数などプログラム中で最後までよく使われるものは、あらかじめ作っておけばよい。そうすれば、参照カウンタを用いる場合のメリットであるメモリの節約をカバーできる。しかし、どちらを選択するかはライブラリの設計者が、可変長の資源の性質とその処理方法をもとに判断することとする。

## 2.5 設計法のまとめ

ここでは、提案した設計法を簡単にまとめる。

- 参照情報を用いる場合
  1. データ構造にタグを付ける。
  2. メモリ割当関数・データ構築関数を作成する。
  3. ごみ集めコマンドをデータ構造ごとにマクロで定義し、その中でごみ集め関数を呼ぶ。

4. 代入定義文・代入関数を作成する。
5. それぞれライブラリ関数を作成する。
6. ライブラリの使用書を作成する。

- 参照カウンタを用いる場合
  1. データ構造にタグを付ける。
  2. メモリ割当関数・データ構築関数を作成する。
  3. 参照カウンタ操作コマンドをマクロで定義し、その中で参照カウンタ操作関数を呼ぶ。
  4. 代入定義文・代入関数を作成する。
  5. それぞれライブラリ関数を作成する。
  6. ライブラリの使用書を作成する。

なお、それぞれの段階における詳しい作成方法は、各記載箇所を参照する。

以上が具体的な設計法である。これに基づいて第3章で実際にライブラリを作成し、第4章で、作成したライブラリを用いてプログラミングを行い、設計法について評価をする。

# 第3章　可変長整数のライブラリ

C 言語では、組み込みの整数型（int 型）では最大で $2^{32}$ 通りの範囲内でしか処理することができない。例えば、ある整数 n の階乗を求めようとした時、符号無しの場合で 13! ですでにオーバーフローを起こす。これではあまり実用的とは言えない。この範囲外での整数を処理するために、任意の大きさの整数計算を行う関数のパッケージ、**可変長整数**（variable length integer）のライブラリを第 2 章で提案した設計法に基づいて作成する。なお、可変長整数のデータ構造・演算関数のアルゴリズム・内部関数内の処理方法は、参考文献 [1]4.2 節を参考にした。

## 3.1　実装

実装は、提案した設計法に基づき、次のように行った。

### 3.1.1　データ構造

データ構造は、可変長整数を b-進数[1]をして表現し、各桁ごとに 1 つのセルに入れた双方向リストを基本とした。双方向リストにした理由は、比較演算・四則演算において最上位の桁から走査していく場合と最下位の桁から走査していく場合があるので走査の効率をよくするためである。そのために最上位と最下位の桁のセルへのポインタが必要になるため、セルのリストの上位に、セルの個数・最上位の桁へのポインタ・最下位の桁へのポインタ、そして、参照情報を持つセルを追加して、可変長整数 VINT 型とした。また、負の数は補数表現を用いた。データ構造としては、トップのセルの構造体（VINT 型）と各桁のセルの構造体（VINT_CELL 型）の 2 つを用意した。

今回の可変長整数のライブラリでは、整数は常に値が変化し同じ値のものを複

[1] ここで b は組込み演算で処理できる最大の整数の平方根より小さいものとする。なぜなら、積を計算する際に各桁の積を求めた時点でオーバーフローが生じるのを防ぐためである。

数使用することは少ないだろうと思われるので最初に考案した参照情報を用いる設計法を選択した。

### 3.1.2　メモリ操作関数

まず最初に、メモリ割当・解放を行う関数を作成した。メモリ割当（get_vint, get_vcell）ではそれぞれの型ごとに組込みの動的メモリ割当関数 malloc を呼び、その後初期化を行うようにした。メモリ解放（free_vint, free_vcell）は、マクロで定義し、実際の解放は関数を呼び出し、引数の指定する可変長整数が REF タイプであったならポインタを NULL に設定するようにした。メモリを解放する関数では、メモリを初期化してから組込みの割当メモリ解放関数 free を呼び、引数の指定する可変長整数の参照情報を返す。さらに VINT 型のセルが持つ VINT_CELL 型のセルのリストを解放する関数 clear_vint を作成した。また、これらの関数は引数が NULL の場合は何もしないようにしてある。ここではセルの再利用によるごみ集めは行わなかったが、メモリ操作関数を変更するだけで簡単にそのようなごみ集めを行うように変更できる。

### 3.1.3　vassign の定義

次に、代入定義文 vassign と代入関数 assign を作成した。代入定義文は手続き型引数付きマクロで定義した。処理手順は、設計法をそのまま実装した。

### 3.1.4　ライブラリ関数の作成

最後に、可変長整数を処理するライブラリ関数を作成した。

まずは変換・表示関数から作成した。変換関数 itovi は int 型整数をまた変換関数 atovi は文字列をそれぞれ可変長整数に変換し、表示関数 put_vint は、可変長整数を表示する関数である。これらの関数は、可変長整数を各桁（セル）ごとに処理するようにした。

その後、比較演算・四則演算を行う関数を、それぞれのアルゴリズムに従い、設計法に基づき作成した。詳しいアルゴリズムに関しては参考文献 [1] の 4.2 節を参照してもらいたい。また、必要に応じてセル操作関数も作成していった。こ

れらの関数も設計法に基づいて作成してある。除算のアルゴリズムに関しては、かなりの工夫を凝らしたので、次節で詳しく述べる。

## 3.2 除算アルゴリズム

除算に関しては、正整数同士のアルゴリズムは参考文献 [1]4.2.4 を参考にしたが、負の数に関しては独自に工夫した。また、この除算アルゴリズムは計算の速さが被除数と除数の b-進数における桁数の差に依存しているのが特徴である。そのため複雑ではあるが、単純に減算を繰り返すアルゴリズム[2]よりは格段に速い。このアルゴリズムは、人間が机上で割算を手計算（筆算）で行なう手法とほぼ同様である。手計算では商の各桁を 2, 3 回予測して求めるが、このアルゴリズムでも各桁の値を 3 つの連続した数までに絞り込める点などが、非常に手計算の手法に類似している。

また、除算では、商も剰余も 1 つのアルゴリズムで同時に求められるので、同じ被除数・除数の商と剰余を求める場合に、同じ計算を 2 回行うのは効率が悪いので、一度に商と剰余の両方を求めて代入する関数 vqrm を特別に用意した。この関数は、代入定義文 vassign において、代入されるポインタ変数を 2 つに増やした代入定義文 vqrm の中で、商と剰余を求めて 2 つの変数に同時に代入を行う代入関数 qrm を呼び出すように作成した。また、商や剰余どちらか 1 つを求める関数 vdiv, vmod は、関数内で商、剰余を格納するポインタ変数両方を宣言し、vqrm を呼び出して計算を行い、値を返さない方を解放するように作成した。例えば、vdiv では vqrm を呼び、商 qs と剰余 rs 両方を求めておいて、剰余 rs は解放し、商 qs へのポインタを返す。これだと、除算アルゴリズムを実行する関数は qrm の 1 つのみでよく、vdiv, vmod は簡単に作成できる。

次節では、参考文献 [1]4.2.4 の「正÷正」の計算の考え方を簡単にまとめて説明する。詳細については文献を参照してもらいたい。

[2] このアルゴリズムの計算の速さは被除数と除数の値の大きさの違い、つまり商に依存しているので、2 つの数の組合せによって速さに大きな違いが出る。可変長整数においては、この違いは計算の速さに大きな影響を及ぼす。

### 3.2.1 「正÷正」の計算方法

除算にはある種の当て推量が必要なので、最も難しい算術計算である。商と余りを求める通常の除算のアルゴリズムは計算段階を繰り返して行なうものである。その段階ごとに商の b-数[3]が 1 桁求められ、次の桁のための余りが計算される。被除数を xs（ $[x_m, \cdots, x_1]$ ）、除数を ys（ $[y_n, \cdots, y_1]$ ）、商を qs、余りを rs とする。ここで、 $[a_k, \cdots, a_1]$ は k 桁の可変長整数を表していて、$0 \leq a_i < b$ であり、$= a_k b^{k-1} + a_{k-1} b^{k-2} + \cdots + a_1$ である。

このとき 1 つの条件が付く。除数の最上位桁 $y_n$ が $\lceil b/2 \rceil$ 以上であることである。これについての詳細は参考文献 [2] で解説されている。この条件を満たすため、あらかじめ被除数、除数に d （ $= \lceil b/(y_n + 1) \rceil$ ）を掛けて条件に合うように補正しておき、最後に余りを d で割ればよい。

商 qs の初期値は [0] で、余り rs の初期値は被除数の先頭から n 個の数を取ってきたもの $[x_m, \cdots, x_{m-n+1}]$ にする。 $m < n$ の場合は rs は xs をそのままのものとする。余り rs にあとに述べる dstep を適用し、その桁の商と余りを求める。商は qs の末尾に追加し、余りは次の桁の dstep に使う。

dstep の処理、つまり各桁の処理は、次の 3 つの場合に分ける必要がある。各桁における被除数[4]xs' $[x'_k, \cdots, x'_1]$ （長さ k）の長さが除数 ys の長さより短いか、または等しいか、またはそれより長いかという 3 つの場合である。実際、前の桁から得られる余り rs の長さは除数の長さ n を超えることはなく、その桁の被除数 xs' は xs の次に計算される桁を 1 つ追加したものであるから、常に $k \leq n + 1$ が成り立つ。そこで、dstep を次のように場合分けする。

- $k < n$ のとき astep
- $k = n$ のとき bstep
- $k = n + 1$ のとき cstep

astep: このときは簡単である。桁数が小さいので、その桁における被除数 xs' は明らかに除数 ys より小さい。よって、この桁の余り rs は xs' そのままになり、商は 0 になる。

---

[3] b-進数の各桁をこう呼ぶ。

[4] このとき xs' は前の桁の余り rs の末尾に、次に計算される桁の被除数の桁の数を追加したものになる。

bstep: 先の条件 $y_n \geq \lceil b/2 \rceil$ より、桁数が同じときは商は 1 を越えないことになる。したがって、被除数 xs' が除数 ys より小さいときは商は 0 で余り rs は xs' そのまま、xs' が ys 以上のときは商は 1 で rs は xs' − ys となる。

cstep: この場合も参考文献 [2]（4.3.1）の結果を利用する。この桁の商を q とすると、 $q' - 2 \leq q \leq q'$ になる。ここで、$q' = \min(x'_k b + x_{k-1}, b - 1)$ である。つまり、商 q は商の推定値 q' よりもたかだか 2 だけしか大きくなることはないというのである。よって、ここではまず $q' - 2$ を求め仮の商 q とし、余り rs を $xs' - q \cdot ys$ として、rs がまだ ys より大きければ仮の商を 1 増やし、余りから ys をひく。それでもまだ rs が ys より大きければ再び仮の商を 1 増やし、余りから ys をひく。これを商、余りとする。

最後に余りを d で割るが、これは上のアルゴリズムの dstep を組込みの演算で行なう関数を用意することにより簡単に実装できる。

### 3.2.2 負の数の計算

では負の数を含む場合はどうするかを考える。「負÷負」の場合は前処理の d を掛ける段階で、「正÷正」になるようにしてやればよい。「負÷正」と「正÷負」の場合についてアルゴリズムを適用してみたところ、「正÷負」はアルゴリズムどおりでは計算できなかったが、「負÷正」は最初の桁の符号をうまく処理してやればほぼアルゴリズムどおりに計算できることがわかった。「負÷正」が計算できれば「正÷負」も計算できることになる。

10-進数として「-1784 ÷ 62」を考える。-1784 を補数で表すと [-1,8,2,1,6] になる。最上位は符号なので桁数には数えない。そこで、余り rs の初期値は最上位の符号とそのあとの除数 ys（[6,2]）の桁数と同じ 2 桁分である [-1,8,2] にする。最上位の符号は桁数に入れずに考えるので、bstep になる。ここでは当然 rs は ys より小さいので商は [0,0]、余り rs は [-1,8,2]。次の桁の被除数は [-1,8,2,1] になり cstep を行なう。このとき推定値 q' は $[-1, 8, 2]/6$ から-3 になり仮の商は-5 となる。余りは [1,3,1] になり、ys より大きいので商を-4 にする。それでも余りは [6,9] で ys より大きい。よってこの桁の商は-3 で余りは [7] になる。次の桁の被除数は [7,6] になる。ここからは「正÷正」のときと同じように求められ、商は 1、余りは [1,4] になる。したがって、アルゴリズムによる答えは商が [0,-3,1]、余りが [1,4] となる。これを正規表現に直すと商は [-1,8,9]、余りは [1,4] である。これらの数

はそれぞれ-29、14 を表し、計算結果が正しいことがわかる。

以上のように「負÷正」のときは最上位の符号を桁数に数えないで「正÷正」のアルゴリズムを適用すればよい。

また、補正のために掛ける d は、除数が負のときは $\lceil b/(b-y_{n-1}) \rceil$ としてやればよい。例えば「892 ÷ (-31)」では d = -2 となり上の例と同じになって計算できる。

実装の際には、符号フラグを用意し、被除数が正の時は 0、負の時には 1 としておき、桁数の比較のときに除数の桁数側に足す。そして、cstep を行なった時点でフラグを正にする。被除数が負のときは最初に商が 0 以外になるのは cstep であり、それ以降はフラグは正のままでよいからである。また、推定値 q' を求めるときも負の場合を考慮する。

これにより、参考文献 [1] の「正÷正」のアルゴリズムを拡張して、負の数にも適用できるようになった。

## 3.3　チェック関数

ライブラリを作成するにあたって、デバッグの際に、関数の引数としてデータ構造が正しく構築されているか、ごみがきちんと回収されているかなどの情報を得るために、それらを調べる関数 check_vint を用意した。

この関数の機能は、すべての使用（メモリ割当）されている可変長のポインタの値の一覧を管理・表示、指定された可変長整数のセル構造の整合性をチェックすることである。引数にどの操作を実行するかの情報を指定して使用する。

この関数は独自に可変長整数のトップ（VINT 型）のセルへのポインタの値のリストを持っている。そして、メモリ割当関数 get_vint の中ではその新しいポインタをリストに追加するように呼び出され、メモリ解放関数 free_vint の中では解放されるポインタをリストから削除するように呼び出される。また、各関数の冒頭で、引数が正しい構造になっているかをチェックし、より正確な位置でバグを発見できるように後発的なエラー[5]が出ないようにした。チェックの方法は、簡単なものは、そのポインタがすでに使用されているかをリストを走査して調べる、

[5]例えば、ある関数が呼び出される前にすでに引数になるデータの構造がおかしくなっているのに、その関数の引数として使われたために、その関数が処理しようとしてエラーを起こすような場合。この関数自体にバグはないので、どこがおかしいか発見しにくくなる。

さらに詳しいものとしては、セルの個数や連結を 1 つ 1 つ調べて構造の整合性を調べるようにした。

しかし、この関数を使うと実行速度が遅くなる[6]ため、デバッグ用にコンパイルしたときのみ使用するようにした。

この関数は、ごみが回収されているかを調べるのに非常に有効であるので、第 4 章のライブラリの評価の際に利用する。

## 3.4 動作テスト

ここでは、作成したライブラリが実際にごみを残していないかをテスト用 main 関数（付録 B test.c）を使って調べた結果を記述する。ごみが残っているかどうかは、チェック関数の使用ポインタ表示機能を用いた。

```
% vint
Atovi : xs1 = 1784
Atovi0: xs2 = 62
Veq  : 1784 == 62 -> FALSE
Vleq : 1784 <= 62 -> FALSE
Vless: 1784 <  62 -> FALSE
Vadd : 1784 +  62 = 1846
Vsub : 1784 -  62 = 1722
Vmul : 1784 *  62 = 110608
Vdiv : 1784 /  62 = 28 Vmod : 48
Vqrm : 1784 /  62 = 28   ...  48
Using Vint(4): --  PTR --
            | 134684720 (    1 cells ) |
            | 134684752 (    1 cells ) |
            | 134684800 (    1 cells ) |
            | 134684848 (    1 cells ) |
Continue?(y/n) y
Atovi : xs1 = -1784
Atovi0: xs2 = 62
Veq  : -1784 == 62 -> FALSE
Vleq : -1784 <= 62 -> TRUE
Vless: -1784 <  62 -> TRUE
Vadd : -1784 +  62 = -1722
Vsub : -1784 -  62 = -1846
Vmul : -1784 *  62 = -110608
Vdiv : -1784 /  62 = -29 Vmod : 14
Vqrm : -1784 /  62 = -29   ...  14
Using Vint(4): --  PTR --
            | 134684720 (    2 cells ) |
            | 134684752 (    1 cells ) |
```

[6]約 3 倍以上の違いが出ることが計測された。第 4 章の $e$ を求めるプログラム vexp.c の 1,000 桁の実行結果を見ると、通常は 3 秒 54（表 4.1）であるのに対し、デバッグ時には 13 秒 28（34 ページ）となっている。

```
            | 134684800 (    2 cells ) |
            | 134684848 (    1 cells ) |
Continue?(y/n) y
Atovi : xs1 = 1234567890
Atovi0: xs2 = 234567890
Veq  : 1234567890 == 234567890 -> FALSE
Vleq : 1234567890 <= 234567890 -> FALSE
Vless: 1234567890 <  234567890 -> FALSE
Vadd : 1234567890 +  234567890 = 1469135780
Vsub : 1234567890 -  234567890 = 1000000000
Vmul : 1234567890 *  234567890 = 289589985019052100
Vdiv : 1234567890 /  234567890 = 5 Vmod : 61728440
Vqrm : 1234567890 /  234567890 = 5   ...  61728440
Using Vint(4): --  PTR --
            | 134684720 (    3 cells ) |
            | 134684752 (    3 cells ) |
            | 134684800 (    1 cells ) |
            | 134684848 (    2 cells ) |
Continue?(y/n) n
%
```

この結果から、1回の動作が終わった時点で絶えず4つの可変長整数が使用されていていることがわかる。この4つの変数は、計算（比較・四則演算）される2数と、結果を格納する2数[7]であり、これらはmain関数内で宣言されていてごみ集めはしていないので当然絶えず使用されていることになる。このことから、作成した可変長整数はきちんとごみ集めができていることがわかる。

## 3.5　ライブラリの利用方法

ライブラリの使用書に記述する内容を次にまとめる。

- 可変長整数は “VINT *” 型で宣言して使用する。宣言する際に、必ず初期化（ = NULL ）しておく。
- 代入はvassignを使用する。xにyを代入するときは「vassign( x, y );」。
- 可変長整数を作成する際は、変数に変換関数itovi, atoviを代入する。int型整数を変換するときはitovi、文字列を変換するときはatoviを使用する。
- 比較演算・四則演算関数は次の通り。ただし、x, y, q, rは可変長整数、nはint型整数。

[7]除算では商と余りがあるため2つ用意した。

  - veq( x, y ) : $x = y$ ならば TRUE を、そうでなければ FALSE を返す。
  - vleq( x, y ) : $x \leq y$ ならば TRUE を、そうでなければ FALSE を返す。
  - vless( x, y ) : $x < y$ ならば TRUE を、そうでなければ FALSE を返す。
  - vadd( x, y ) : x, y の和を返す。( $x + y$ )
  - vsub( x, y ) : x, y の差を返す。( $x - y$ )
  - vmul( x, y ) : x, y の積を返す。( $x * y$ )
  - vdiv( x, y ) : x, y の商を返す。( $x/y$ )
  - vmod( x, y ) : x, y の剰余を返す。( $x\%y$ )
  - vqrm( q, r, x, y ) : x/y の商、剰余を q, r に代入する。( $x = qy + r$ )
  - bmul( x, n ) : x, n の積を返す。( $x * n$ )
  - bdiv( x, n ) : x, n の商を返す。( $x/n$ )
  - bmod( x, n ) : x, n の剰余を返す。( $x\%n$ )

- 変数を解放するときはまず、clear_vint で各桁のセルを解放し、そのあとで free_vint で変数を解放する。例えば、x を解放するときは、

```
 ...
clear_vint( x );
free_vint( x );
 ...
```

  となる。

- 関数を作成するときは次の操作を必ず行なうように作成する。

  - 引数に可変長整数を取る場合、一時変数（state）をその個数分宣言し、関数の最初で一時変数にそれぞれの引数の参照情報を代入する。そのあと、引数の参照情報を REF にする。例えば、引数が x, y のときは、

```
int state1, state2;
 ...
state1 = x->state;
state2 = y->state;
x->state = REF;
y->state = REF;
 ...
```

とする。そして、関数が値を返す直前に、一時変数の参照情報を引数に戻し、NONREF のものを解放する。先の例では、

```
 ...
x->state = state1;
y->state = state2;
if( x->state == NONREF ){
    clear_vint( x );
    free_vint( x );
}
if( y->state == NONREF ){
    clear_vint( y );
    free_vint( y );
}
return;
```

となる。

– 戻りが可変長整数の場合、値を返す直前に参照情報を NONREF にする。z を戻り値とするような場合は、

```
 ...
z->state = NONREF;
return( z );
```

とする。

– 関数内で可変長整数を局所的に宣言した場合は、値を返す前に、戻り値以外のものを解放する。

## 3.6　参照カウンタを用いた場合への変更

設計法で述べたように、参照情報を用いる場合と参照カウンタを用いる場合の変更は簡単にできる。ここでは、参照情報を用いて作成した可変長整数ライブラリを、参照カウンタを用いて作成した場合の変更（相違）点を述べる。

参照情報を用いた現在のライブラリの中で変更すればよい点は、

- VINT 型の構造体の中の参照情報タグ state の名前
- 各関数の冒頭のタイプを保持する部分
- 各関数の最後のタイプを戻して NONREF タイプを解放する部分

- メモリ解放関数 free_vint
- 各関数の最後の戻り値を NONREF タイプにする部分
- 代入関数 vassign

である。詳しい変更については省略する。

# 第4章　評価

ここでは、作成した可変長整数ライブラリを用いたプログラミングの例とライブラリを実用した例を挙げて、本設計法が有効であることを示す。

## 4.1　可変長整数ライブラリを用いたプログラムの例

まずは、可変長整数ライブラリを用いて指定された桁数の $e$ の値を求めるプログラム（付録C vexp.c）を作成した。

このプログラムでは、ポインタ変数の宣言の際に初期化を行うこと、代入・演算を関数で書くこと、作成した関数の最後で不要なメモリを解放すること以外は、int 型整数で作成した場合とほとんど内容は変わらない。基本的に、作成した関数の最後で不要なメモリを解放する行と、インクルードファイルの指定の行が増えただけである。このことより、ライブラリを使用するユーザにとってのプログラミングの手間はさほど変化しないことがわかる。さらに、ユーザは自分で作成した関数の最後以外ではごみ集めを意識する必要がない。

### 4.1.1　ごみ集めの状況

では、肝心のごみ集めはきちんと行われているのであろうか。100 桁、1,000 桁の場合の実行結果を次に示す。

```
% time vexp
KETA = 100
e = 2.718281828459045235360287471352662497757247093699959574
966967627724076630353547594571382178525166423 8
End.
Using Vint(6): --  PTR --
            | 134619184 (    1 cells ) |
            | 134619216 (    1 cells ) |
            | 134619264 (    1 cells ) |
            | 134619312 (   26 cells ) |
            | 134619360 (    1 cells ) |
            | 134619392 (   25 cells ) |
```

```
0.121u 0.000s 0:00.14 85.7%     40+255k 0+0io 2pf+0w

% time vexp 1000
KETA = 1000
e = 2.71828182845904523536028747135266249775724709369995957
49669676277240766303535475945713821785251664274274663919320
03059921817413596629043572900334295260595630738132328627943
49076323382988075319525101901157383418793070215408914993488
41675092447614606680822648001684774118537423454424371075390
77744992069551702761838606261331384583000752044933826560297
60673711320070932870912744374704723069697720931014169283681
90255151086574637721112523897844250569536967707854499699679
46864454905987931636889230098793127736178215424999229576351
48220826989519366803318252886939849646510582093923982948879
33203625094431173012381970684161403970198376793206832823764
64804295311802328782509819455815301756717361332069811250996
18188159304169035159888851934580727386673858942287922849989
20868058257492796104841984443634632449684875602336248270419
78623209002160990235304369941849146314093431738143640546253
15209618369088870701676839642437814059271456354906130310720
85103837505101157477041718986106873969655212671546889570350
116
End.
Using Vint(6): --  PTR --
            | 134619184 (    1 cells ) |
            | 134619216 (    1 cells ) |
            | 134619264 (    1 cells ) |
            | 134619312 (  251 cells ) |
            | 134619360 (    1 cells ) |
            | 134620080 (  250 cells ) |
7.440u 0.000s 0:13.28 56.0%     40+295k 0+0io 0pf+0w

%
```

付録C vexp.c を見ると、可変長整数（VINT）型のポインタ変数は、大域変数として3つ、main 関数内で3つの合計6つが宣言されている。これらは解放しないので、check_vint 関数の表示機能による使用中の可変長整数の個数は6つで等しくなるので正しい。また、100万桁を指定した場合のものが400時間以上計算を続けている。もし少しでもごみを残しているならば、たちまちメモリを使い果たして止まるはずである。

これらの結果から、設計法に基づいて設計したライブラリを使用したプログラムでもきちんとごみ集めができていることが明らかになった。

### 4.1.2 プログラミングにおける手間

では次に、本設計法に基づいてライブラリを作成した際の手間について、実際に本設計法に基づいてごみ集めを行わないライブラリとプログラムを用意して比

較した結果について述べる。

まずは、ライブラリ作成時の手間から比較する。本設計法に基づかないライブラリは、すでに作成したライブラリから本設計法に基づいている部分を修正することによって作成した。具体的には、一時変数を導入して関数の合成を行わない、代入コマンドは使用せず、代入の箇所ごとに旧データの解放、ポインタの更新を行うように記述した。単純にこれらの箇所を変更すればよいだけの作業ではあるが、かなり手動によるごみ集めを記述するのは困難であり、ライブラリの修正だけで、2 日を要した。さらに、デバッグが困難であり、プログラムの行数も、設計法に基づいた場合に追加される分を考慮しても明らかに長くなった。もともと完成していたライブラリを修正するだけでこれだけの手間がかかったので、設計法に基づいて作成した場合、かなりプログラミングが楽になることが言えるだろう。また、設計法に基づいてライブラリを作成しているときは、ごみ集めなどに対するデバッグはほとんどする必要がなく、計算結果の違いによる計算手順に対するデバッグがほとんどであった。本設計法に基づかない場合には、さらにごみ集めや代入に対するデバッグが増えるわけである。

次に、そのライブラリを用いたプログラムの作成の手間を比較する。付録 C の設計法に基づいて作成されたライブラリを用いたプログラム vexp.c を見てもらいたい。この場合、変数の宣言の初期化と、関数の引数の参照タイプの処理、関数の最後の局所変数の解放以外は、演算をライブラリ関数で書く必要があるものの、ほぼ int 型整数で作成した場合と同等の行数で記述できている。これは、明らかに設計法に基づくライブラリを使用したプログラミングが簡単に行えることを表していると言ってもよいだろう。さらに、ユーザは前述の箇所以外ではごみ集めを意識する必要がない。しかも、これらの操作は使用書にあらかじめ記載されていることなので、実際にはユーザはまったくごみ集めを意識しなくてもすむ可能性もある。逆に、設計法に基づいていないライブラリを使用したプログラム vexp-cmp.c を見ると、明らかに記述が困難であったことがわかってもらえるだろう。こちらでは、ユーザが絶えずごみ集めを意識していなければならず、さらにプログラムが正常に作動するまでかなりのデバッグを要した。ようやく動いた時点でごみ集めをチェックしたところ、次のような実行結果になった。ただし、使用ポインタの一覧は一部省略した。

```
% time vexp
KETA = 100
e = 2.71828182845904523536028747135266249775724709369995957
```

```
4966967627724076630353547594571382178525166423 8
End.
Using Vint(216): --  PTR --
            | 106544 (    1 cells ) |
            | 106592 (    1 cells ) |
            | 106640 (    1 cells ) |
            | 106800 (   26 cells ) |
            | 106688 (    0 cells ) |

               ...

            | 115920 (    0 cells ) |
            | 113712 (    0 cells ) |
            | 115984 (    1 cells ) |
            | 106736 (    1 cells ) |
            | 106768 (   25 cells ) |
0.281u 0.007s 0:00.29 96.5%     37+273k 0+0io 0pf+0w

%
```

これは明らかにごみの回収し残しがあることを示している。実際、ライブラリの修正と並行しながらこのプログラムを作成したわけだが、それでもこれだけのごみが残ってしまっているのである。もしかしたら、ユーザはごみの残っていることに気付かない可能性も考えられるので、このごみの量は致命的である。

以上のことから、本設計法に基づいて作成されたライブラリが、いかに使用しやすいか、どれだけごみ集めに有効であるかが明らかになった。

### 4.1.3 実行速度の比較

最後に、本設計法に基づいたライブラリと基づかないライブラリ、それぞれを使用した場合における実行速度を比較する。なぜこのような比較を行うかというと、いかに本設計法が、それに基づくライブラリのプログラミング、そのライブラリを使用したプログラミングを容易にし、ごみ集めに有効であったとしても、肝心のプログラムの実行速度があまりにも遅くなったのでは元も子もないからである。

実際に両方を実行した結果を表 4.1 にまとめた。実行においては同一マシンにより同一条件下で計測した。また、 $e$ を求める計算手順も同じとした。この結果から考えて、本設計法に基づいて設計した方が、実行時間が短くなることがわかる。これは、一時変数の処理[1]の時間のためである。このことから、本設計法に

[1]複製したり、解放したりする作業が増える。それぞれの処理時間は短くても、一時変数を多用しなければならないので全体としては遅れが生じる。

表 4.1: 実行速度の比較

| 桁数 | vexp.c | vexp-cmp.c |
|---|---|---|
| 100 | 0 秒 04 | 0 秒 04 |
| 1,000 | 3 秒 54 | 3 秒 69 |
| 10,000 | 6 分 05 秒 50 | 6 分 27 秒 17 |

基づいてライブラリを作成することにより、処理が効率をよくし、それに伴い実行速度が速くなるが明らかになった。

### 4.1.4 設計法に基づくライブラリの利点

ここまでの評価結果から、設計法に基づいて作成されたライブラリを使用してプログラミングを行なう際の利点を次にまとめる。

- 意識しないでごみ集めをきちんと行なえる。
- プログラムの行数が減り、記述しやすくなる。
- 処理が効率化され、実行速度が速くなる。

また、これらに伴い得られる利点もいくつか出てくるだろう。

以上のことから、本設計法が非常に有効な手法であると評価できる。

## 4.2 Cdimple への組込み

次に、可変長整数ライブラリを、項書換え系[2]（TRS）を入力としてそれと同等の動作をする C 言語プログラムを生成する Cdimple [3]に組込んだ。

組込みは、int 型整数の格納場所に可変長整数のポインタを格納するように項を構成するように変更し、整数を扱う箇所[4]をライブラリ関数を用いて修正した。また、整数項を解放したときに可変長整数も解放するように追加した。

[2]参考文献 [3] を参照。
[3]参考文献 [4] を参照。
[4]具体的には、読取り・出力（表示）・演算を行う箇所

組込みの際、Cdimple がより効率的に資源を使えるように書換え規則中の定数を検索し、定数項として用意することによりメモリを節約するように工夫した。

ここでも、ライブラリの使用法に基づき、独立してごみを集めるライブラリを用いていることで、容易にごみを残さないように組込みを行うことができた。このことから、本設計法が実用的にも有効であることを示している。

また、Cdimple への組込におけるプログラムなどの掲載は省略した。

Cdimple に整数 n の階乗を求める TRS を入力して生成された C プログラムを実行した結果を次に記す。

- int 型整数を使用している Cimple-3.3 における結果

```
% intfact
Fact(13)
Fact(13)
1932053504
14 reductions / 0.0001 usec = 112000 red/sec
%
```

- 可変長整数を使用している cdimple-3.4b における結果

```
% intfact
Fact(13)
Fact(13)
6227020800
28+40 reductions / 0.0007 usec = 42042 red/sec
%
```

この結果を見ると、int 型整数を使用している場合では、13 の階乗でオーバーフローを起こして正しい計算の結果を得られないが、可変長整数の場合では、オーバーフローは起こらず正しい計算結果が得られる。しかし、可変長整数ライブラリ関数の演算時間が遅くなるため、1 秒間における書換え回数は少なくなる。

# 第5章　おわりに

## 5.1　まとめ

本研究では、可変長の資源を用いてプログラミングを行なう際のごみ集めの問題に対して、ユーザがごみ集めを意識せずに利用が可能なライブラリの設計法を提案した。そして、その設計法に基づき実際に可変長整数のライブラリを作成し、そのライブラリを用いてプログラミングを行って評価したことにより、提案した設計法が有効であることを明らかにした。

## 5.2　今後の課題

本論文では、「ユーザがごみ集めを意識しない利用が可能なライブラリ」を設計することを目的としていたが、実際にはユーザは、少なくともユーザ関数の作成における局所ポインタ変数の処理により、ごみ集めを意識しなければならない。つまり、完全にごみ集めを意識しないですむようなライブラリの設計法を明らかにすることはできなかった。今後の課題としては、本論文で提案した設計法をより有効にするために、ユーザ関数の作成における局所ポインタ変数に対するごみ集めの問題を解決することが挙げられる。

また、本研究では主にC言語を対象として設計・実装を行なったが、C++への適用を検討していくことは今後の課題である。

# 謝辞

日頃ご指導賜る坂部俊樹教授、酒井正彦助教授、そして坂部研究室の皆様に感謝致します。

# 参考文献

[1] R. バード・P. ワドラー共著, 武市正人訳: ”Function Programming 関数プログラミング”, 近代科学社

[2] Donald E. Knuth: ”The Art of Computer Programming, Volume 2/Seminumerical Argorithms”, Addison-Wesley, Reading, Mass., 1969

[3] 二木厚吉, 外山芳人: ”項書き換え型計算モデルとその応用”, 情報処理 Vol.24 No.2, Feb.1983

[4] 酒井正彦, 坂部俊樹, 稲垣康善: ”抽象データ型の代数的仕様の直接実現系 Cdimple”, コンピュータソフトウェア Vol.4 No.4(1987) pp.16-27, 1987

# 付 録A　複素数のライブラリ

- ヘッダーファイル complex.h

```
typedef struct complex{
    int real;
    int img;
} CPLX;

extern CPLX *make();
extern CPLX *add();
extern CPLX *negate();
extern void disp();
```

- ライブラリ関数ファイル complex.c

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

CPLX *make( int x, int y ){
    CPLX *z;
    z = (CPLX *) malloc( sizeof( CPLX ) );
    z->real = x;
    z->img = y;
    return( z );
}

CPLX *add( CPLX *x, CPLX *y ){
    return( make( x->real + y->real, x->img + y->img ) );
}

CPLX *negate( CPLX *x ){
    return( make( -x->real, -x->img ) );
}

void disp( CPLX *x ){
    if( x->real != 0 )
        printf("%d",x->real);
    if( x->img > 0 ){
        printf("+i%d",x->img);
    }else if( x->img < 0 ){
        printf("-i%d",-x->img);
    }
}
```

# 付 録B　可変長整数ライブラリ

- ヘッダーファイル vint.h

```
/*================================================================
                        vint.h
                1999 N.Nishida ver 3.5
================================================================*/
#ifndef _vint_h
#define _vint_h

#define BIGIT 10000             /* b-digit              */
#define B_LEN 4                 /* B_LEN = log("BIGIT") */
/* #define STR_MAX 1000000 */

#define TRUE 1
#define FALSE 0

/*** sign of VINT ***/
#define PLUS 0
#define MINUS 1

/*** type of VINT ***/
#define REF 0
#define NONREF 1
#define CONS 2

/*** opt of check_vint ***/
#define CHECK 0
#define DETAIL 1
#define VASGN 2
#define ADD 3
#define DEL 4
#define DISP 5

/*** structure of a cell, VINT has ***/
typedef struct vint_cell{
    int bigit;                  /* 0 <= x < BIGIT */
    struct vint_cell *upper;
    struct vint_cell *lower;
} VINT_CELL;

/*** structure of VINT ***/
typedef struct vint{
    long int cell_len;
    int state;
    struct vint_cell *top;
    struct vint_cell *bottom;
} VINT;

/*** structure of a list of using VINT's pointer ***/
typedef struct using_vint{
    struct using_vint *next;
    struct vint *vint;
} USING_VINT;

/****** for public use ******/
/*== vassign ==*/
#ifdef DEBUG
#ifdef DEPTH
#define vassign( xs1, xs2 ) ({ \
            int loop1; \
```

```
            depth ++; \
            for( loop1 = 0 ; loop1 < depth ; loop1 ++ ) \
                fprintf(stderr," "); \
            fprintf(stderr,"vassign\n"); \
            if( (xs1) != NULL && check_vint( (xs1), VASGN ) == FALSE ){ \
                fprintf(stderr,"Vassign: \"xs1\" needs \"VINT *xs1=NULL;\".\n"); \
                exit(1); \
            } \
            if( (xs1) == NULL ){ \
                (xs1) = get_vint(); \
            } \
            if( assign( (xs1), (xs2) ) == FALSE ){ \
                clear_vint( (xs1) ); \
                free_vint( (xs1) ); \
                (xs1) = NULL; \
            } \
            depth --; \
        })
#else
#define vassign( xs1, xs2 ) ({ \
            int loop1; \
            depth ++; \
            if( (xs1) != NULL && check_vint( (xs1), VASGN ) == FALSE ){ \
                fprintf(stderr,"Vassign: \"xs1\" needs \"VINT *xs1=NULL;\".\n"); \
                exit(1); \
            } \
            if( (xs1) == NULL ){ \
                (xs1) = get_vint(); \
            } \
            if( assign( (xs1), (xs2) ) == FALSE ){ \
                clear_vint( (xs1) ); \
                free_vint( (xs1) ); \
                (xs1) = NULL; \
            } \
            depth --; \
        })
#endif
#else
#define vassign( xs1, xs2 ) ({ \
            if( (xs1) == NULL ){ \
                (xs1) = get_vint(); \
            } \
            if( assign( (xs1), (xs2) ) == FALSE ){ \
                clear_vint( (xs1) ); \
                free_vint( (xs1) ); \
                (xs1) = NULL; \
            } \
        })
#endif

/*== vqrm ==*/
#ifdef DEBUG
#ifdef DEPTH
#define vqrm( qs, rs, xs1, xs2 ) ({ \
            int loop2; \
            depth ++; \
            for( loop2 = 0 ; loop2 < depth ; loop2 ++ ) \
                fprintf(stderr," "); \
            fprintf(stderr,"vqrm\n"); \
            if( (qs) == NULL ){ \
                (qs) = get_vint(); \
            }else if( check_vint( (qs), VASGN ) == FALSE ){ \
                fprintf(stderr,"Vqrm: \"qs\" needs \"VINT *qs=NULL;\".\n"); \
                exit(1); \
            } \
            if( (rs) == NULL ){ \
                (rs) = get_vint(); \
            }else if( check_vint( (rs), VASGN ) == FALSE ){ \
                fprintf(stderr,"Vqrm: \"rs\" needs \"VINT *rs=NULL;\".\n"); \
                exit(1); \
            } \
            qrm( (qs), (rs), (xs1), (xs2) ); \
            depth --; \
        })
#else
```

```
#define vqrm( qs, rs, xs1, xs2 ) ({ \
            int loop2; \
            depth ++; \
            if( (qs) == NULL ){ \
                (qs) = get_vint(); \
            }else if( check_vint( (qs), VASGN ) == FALSE ){ \
                fprintf(stderr,"Vqrm: \"qs\" needs \"VINT *qs=NULL;\".\n"); \
                exit(1); \
            } \
            if( (rs) == NULL ){ \
                (rs) = get_vint(); \
            }else if( check_vint( (rs), VASGN ) == FALSE ){ \
                fprintf(stderr,"Vqrm: \"rs\" needs \"VINT *rs=NULL;\".\n"); \
                exit(1); \
            } \
            qrm( (qs), (rs), (xs1), (xs2) ); \
            depth --; \
        })
#endif
#else
#define vqrm( qs, rs, xs1, xs2 ) ({ \
            if( (qs) == NULL ) \
                (qs) = get_vint(); \
            if( (rs) == NULL ) \
                (rs) = get_vint(); \
            qrm( (qs), (rs), (xs1), (xs2) ); \
        })
#endif

extern int assign();            /*  assign xs1 to xs2                           */
extern VINT *itovi();           /*  convert integer to vint                     */
extern VINT *atovi0();          /*  convert string to vint in tail to head order  */
extern VINT *atovi();           /*  convert string to vint in head to tail order  */
extern VINT *vadd();            /*  calculate the sum of xs1 and xs2            */
extern VINT *vsub();            /*  substract xs2 from xs1                      */
extern VINT *vmul();            /*  calculate the product of xs1 by xs2         */
extern VINT *vdiv();            /*  calculate the quotient, xs1 divide by xs2   */
extern VINT *vmod();            /*  calculate the residue, xs1 divide byxs2     */
extern void qrm();              /*  calculate the quotient(xsq)
                                        and the residue(xsr), xs1 divide byxs2  */

extern int veq();               /*  compare xs1 xs2 (=)                         */
extern int vleq();              /*  compare xs1 xs2 (<=)                        */
extern int vless(); /*  compare xs1 xs2 (<)                         */

extern void put_vint();         /*  show xs to stdout                           */
extern void fput_vint();        /*  show xs to stdout                           */
extern void put_vint2();        /*  show xs to stdout for debug                 */

/****** cell handling functions (for internal use) ******/
extern VINT *get_vint();        /*  get new vint                                */
extern VINT_CELL *get_vcell();  /*  get new cll for vint                        */
extern void init_vint();        /*  initialize vint                             */
extern void init_vcell();       /*  initialize cell for vint                    */
extern void clear_vint();       /*  free vcells and reset vint                  */

/*== free_vint ==*/             /*  free vint, and set NULL                     */
#define free_vint( xs ) ({ \
            if( free_vint0( (xs) ) == REF ) \
                (xs) = NULL; \
        })
extern int free_vint0();        /*  free vint, but leave vcells unfreed         */
/*== free_vint ==*/             /*  free vcells, and set NULL                   */
#define free_vcell( ys ) ({ \
            free_vcell0( (ys) ); \
            (ys) = NULL; \
        })
extern void free_vcell0();      /*  free vcells                                 */

/****** sub-routines for public use functions (for internal use) ******/
extern void strep();            /*  remove redundant 0s on the top of xs        */
extern void copy_vcell();       /*  copy vcells valued n to the top of xs       */
extern void align();            /*  eqalize the number of vcells of xs1 and xs2 */
extern void norm();             /*  noramalize xs                               */
extern void negate();           /*  negate xs                                   */
```

```
extern int divide();            /*  calculate the quotient, x divide by y        */
extern int mod();               /*  calculate the residue, x divide by y         */
extern VINT *bmul();            /*  calculate the quotient, xs divide by x       */
extern VINT *bdiv();            /*  calcukate the residue, xs divide by x        */
extern void disp_vcell();       /*  show vcell to stdout                         */
extern int check_vint();        /*  check a VINT's variable                      */

#ifdef DEBUG
extern int depth;
#endif

#endif
```

- ライブラリ関数ファイル vint.c

```
/*================================================================
                            vint.c
                    1999 N.Nishida ver 3.5
================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vint.h"

#ifdef DEBUG
int depth = 0;
#endif

/**************************************************************
                    for public use
**************************************************************/

/*-------------------------------------------------------------
                    assign
-------------------------------------------------------------*/
int assign( xs1, xs2 )
VINT *xs1, *xs2;
{
    VINT_CELL *ys1 = NULL, *ys2 = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"assign\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs1, CHECK ) == FALSE ){
    /*--initialize--*/
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Assign: Argument \"xs1\" needs \"get_vint\"\n");
        exit(1);
    }else
#endif
    if( xs1->state != REF ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Assign: Argument \"xs1\" must be a variable.\n");
        exit(1);
    }
#ifdef DEBUG
    else if( check_vint( xs2, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Assign: Argument \"xs2\" is not proper.\n");
        exit(1);
    }
#endif
```

```
    clear_vint( xs1 );

    if( xs2 == NULL ){
#ifdef DEBUG
        depth --;
#endif
        return( FALSE );
    }else if( xs2->state == NONREF ){
/*--xs2 is a function result--*/
        *xs1 = *xs2;
        xs1->state = REF;
        free_vint( xs2 );
    }else{
/*--otherwise--*/
        copy_vcell( 0, xs2->cell_len, xs1 );
        ys1 = xs1->bottom;
        ys2 = xs2->bottom;
    /*--copy vcells--*/
        while( ys1 != NULL ){
            ys1->bigit = ys2->bigit;
            ys1 = ys1->upper;
            ys2 = ys2->upper;
        }
    }
#ifdef DEBUG
    depth --;
#endif
    return( TRUE );
}

/*---------------------------------------------------------------------
                        itovi
---------------------------------------------------------------------*/
VINT *itovi( x )
int x;
{
    VINT *xs = NULL;
    VINT_CELL *ys = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"itovi\n");
#endif
#endif

/*--create new vint--*/
    xs = get_vint();
/*--get new cell--*/
    ys = get_vcell();
    xs->top = ys;
    xs->bottom = ys;
    xs->cell_len = 1;
    ys->bigit = x;
/*--normalize--*/
    norm( xs );
    xs->state = NONREF;
#ifdef DEBUG
    depth --;
#endif
    return( xs );
}

/*---------------------------------------------------------------------
                        atovi0
---------------------------------------------------------------------*/
VINT *atovi0( string )
char *string;
{
    int i, str_len, sign;
    char bigit[B_LEN+1];
    VINT *xs = NULL;
```

```
    VINT_CELL *ys = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"atovi0\n");
#endif
#endif

    xs = get_vint();
    str_len = strlen(string);
/*--error--*/
    if( str_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Atovi: String has no character.\n");
        free_vint( xs );
        exit(1);
    }
/*--create new cell--*/
    ys = get_vcell();
    xs->bottom = ys;
    xs->top = ys;
    xs->cell_len ++;
    sign = PLUS;
/*--negative case--*/
    if( *string == '-' ){
        string ++;
        str_len --;
    /*--error--*/
        if( str_len == 0 ){
            fprintf(stderr,"\n----Error----\n");
            fprintf(stderr,"Atovi: String has \"-\" only.\n");
            clear_vint( xs );
            free_vint( xs );
#ifdef DEBUG
            depth --;
#endif
            return( NULL );
        }
    /*--flag--*/
        sign = MINUS;
    }
    bigit[B_LEN] = '\0';
/*--string -> integer--*/
    while( str_len > B_LEN ){
        for( i = 0 ; i < B_LEN ; i ++ ){
            bigit[i] = *( string + str_len - B_LEN + i );
        /*--error--*/
            if( ! isdigit( bigit[i] ) ){
                fprintf(stderr,"\n----Error----\n");
                fprintf(stderr,"Atovi: String is not number.\n");
                clear_vint( xs );
                free_vint( xs );
                exit(1);
            }
        }
    /*--add new vcell--*/
        str_len = str_len - B_LEN;
        ys->bigit = atoi( bigit );
        ys->upper = get_vcell();
        ys->upper->lower = ys;
        ys = ys->upper;
        xs->cell_len ++;
        xs->top = ys;
    }
    for( i = 0 ; i < str_len ; i ++ ){
        bigit[i] = *( string + i );
    /*--error--*/
        if( ! isdigit( bigit[i] ) ){
            fprintf(stderr,"\n----Error----\n");
            fprintf(stderr,"Atovi: String is not number.\n");
            clear_vint( xs );
```

```
            free_vint( xs );
            exit(1);
        }
    }
    bigit[str_len] = '\0';
    ys->bigit = atoi( bigit );
/*--negative case--*/
    if( sign == MINUS ){
        negate( xs );
    }
/*--strep--*/
    strep( xs );
    xs->state = NONREF;
#ifdef DEBUG
    depth --;
#endif
    return( xs );
}

/*---------------------------------------------------------------------
                        atovi
---------------------------------------------------------------------*/
VINT *atovi( string )
char *string;
{
    int i, str_len, dif, sign;
    char bigit[B_LEN+1];
    VINT *xs = NULL;
    VINT_CELL *ys=NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"atovi\n");
#endif
#endif

    xs = get_vint();
    str_len = strlen( string );
/*--error--*/
    if( str_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Atovi: String has no character.\n");
        free_vint( xs );
        exit(1);
    }
    ys = get_vcell();
    xs->top = ys;
    xs->bottom = ys;
    xs->cell_len ++;
    sign = PLUS;
/*--negative case--*/
    if( *string == '-' ){
        string ++;
        str_len --;
    /*--error--*/
        if( str_len == 0 ){
            fprintf(stderr,"\n----Error----\n");
            fprintf(stderr,"Atovi: String has \"-\" only.\n");
            clear_vint( xs );
            free_vint( xs );
            exit(1);
        }
    /*--flag--*/
        sign = MINUS;
    }
    i = 0;
    bigit[B_LEN] = '\0';
/*--string -> integer--*/
    while( *( string + 1 ) != '\0' ){
        bigit[i] = *string ;
    /*--error--*/
```

```
        if( ! isdigit( bigit[i] ) ){
            fprintf(stderr,"\n----Error----\n");
            fprintf(stderr,"Atovi: String is not number.\n");
            clear_vint( xs );
            free_vint( xs );
            exit(1);
        }
        i ++;
        string ++;
    /*--add new vcell--*/
        if( i == B_LEN ){
            ys->bigit = atoi( bigit );
            ys->lower = get_vcell();
            ys->lower->upper = ys;
            ys = ys->lower;
            xs->cell_len ++;
            xs->bottom = ys;
            i = 0;
        }
    }
    bigit[i] = *string ;
/*--error--*/
    if( ! isdigit( bigit[i] ) ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Atovi: String is not number.\n");
        clear_vint( xs );
        free_vint( xs );
        exit(1);
    }
    i ++;
    dif = 1;
    while( i < B_LEN ){
        bigit[i] = '0';
        i ++;
        dif *= 10;
    }
    ys->bigit = atoi(bigit);
/*--arrange--*/
    if( dif > 1 && !( xs->bottom->bigit == 0 && xs->cell_len == 1 ) ){
        vassign( xs, vmul( xs, itovi( BIGIT / dif ) ) );
        xs->bottom = xs->bottom->upper;
        free_vcell( xs->bottom->lower );
        xs->cell_len --;
        xs->bottom->lower = NULL;
    }
/*--negative case--*/
    if( sign == MINUS ){
        negate( xs );
    }
/*--strep--*/
    strep( xs );
    xs->state = NONREF;
#ifdef DEBUG
    depth --;
#endif
    return( xs );
}

/*-------------------------------------------------------------------
                        vadd
-------------------------------------------------------------------*/
VINT *vadd( xs1, xs2 )
VINT *xs1, *xs2;
{
    int x1, x2;
    VINT *xs3 = NULL;
    VINT_CELL *ys1 = NULL, *ys2 = NULL, *ys3 = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"vadd\n");
```

```
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs1, CHECK ) == FALSE || check_vint( xs2, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vadd: Argument \"xs1\" or \"xs2\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs1->cell_len * xs2->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vadd: Argument \"xs1\" or \"xs2\" has no value.\n");
        exit(1);
    }

    vassign( xs3, itovi( 0 ) );
/*--#xs1 < #xs2--*/
    if( xs1->cell_len < xs2->cell_len ){
        copy_vcell( 0, xs2->cell_len - 1, xs3 );
    }else{
/*--otherwise--*/
        copy_vcell( 0, xs1->cell_len - 1, xs3 );
    }

    ys1 = xs1->bottom;
    ys2 = xs2->bottom;
    ys3 = xs3->bottom;
/*--add xs1'vcell xs2'vcell--*/
    while( ys1 != NULL || ys2 != NULL){
        if( ys1 != NULL ){
            x1 = ys1->bigit;
            ys1 = ys1->upper;
        }else{
            x1 = 0;
        }
        if( ys2 != NULL ){
            x2 = ys2->bigit;
            ys2 = ys2->upper;
        }else{
            x2 = 0;
        }
        ys3->bigit = x1 + x2 ;
        ys3 = ys3->upper;
    }

/*--normalize--*/
    norm(xs3);

/*--clean--*/
    if( xs1->state == NONREF ){
        clear_vint( xs1 );
        free_vint( xs1 );
    }
    if( xs2->state == NONREF ){
        clear_vint( xs2 );
        free_vint( xs2 );
    }
    xs3->state = NONREF;
#ifdef DEBUG
    depth --;
#endif
    return( xs3 );
}

/*---------------------------------------------------------------------
                        vsub
---------------------------------------------------------------------*/
VINT *vsub( xs1, xs2 )
VINT *xs1, *xs2;
{
    int x1, x2;
    VINT *xs3 = NULL;
    VINT_CELL *ys1 = NULL, *ys2 = NULL, *ys3 = NULL;
```

```
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"vsub\n");
#endif
#endif

/*
#ifdef DEBUG
    printf("vsub(");
    put_vint( xs1 );
    printf(", ");
    put_vint( xs2 );
    printf(")\n");
    fflush(stderr);
#endif
*/

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs1, CHECK ) == FALSE || check_vint( xs2, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vsub: Argument \"xs1\" or \"xs2\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs1->cell_len * xs2->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vsub: Argument \"xs1\" or \"xs2\" has no value.\n");
        exit(1);
    }

    vassign( xs3, itovi( 0 ) );
/*--#xs1 < #xs2--*/
    if( xs1->cell_len < xs2->cell_len ){
        copy_vcell( 0, xs2->cell_len - 1, xs3 );
    }else{
/*--otherwise--*/
        copy_vcell( 0, xs1->cell_len - 1, xs3 );
    }

    ys1 = xs1->bottom;
    ys2 = xs2->bottom;
    ys3 = xs3->bottom;
/*--add xs1'vcell xs2'vcell--*/
    while( ys1 != NULL || ys2 != NULL){
        if( ys1 != NULL ){
            x1 = ys1->bigit;
            ys1 = ys1->upper;
        }else{
            x1 = 0;
        }
        if( ys2 != NULL ){
            x2 = ys2->bigit;
            ys2 = ys2->upper;
        }else{
            x2 = 0;
        }
        ys3->bigit = x1 - x2 ;
        ys3 = ys3->upper;
    }

/*--normalize--*/
    norm(xs3);

/*--clean--*/
    if( xs1->state == NONREF ){
        clear_vint( xs1 );
        free_vint( xs1 );
    }
    if( xs2->state == NONREF ){
```

```
        clear_vint( xs2 );
        free_vint( xs2 );
    }
    xs3->state = NONREF;
#ifdef DEBUG
    depth --;
#endif
    return( xs3 );
}

/*---------------------------------------------------------------------
                        vmul
---------------------------------------------------------------------*/
VINT *vmul( xs1, xs2 )
VINT *xs1, *xs2;
{
    VINT *xs3 = NULL;
    VINT_CELL *ys0 = NULL, *ys1 = NULL, *ys2 = NULL, *ys3 = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"vmul\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs1, CHECK ) == FALSE || check_vint( xs2, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vmul: Argument \"xs1\" or \"xs2\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs1->cell_len * xs2->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vmul: Argument \"xs1\" or \"xs2\" has no value.\n");
        exit(1);
    }
/*--create new vint--*/
    vassign( xs3, itovi( 0 ) );
    copy_vcell( 0, xs1->cell_len + xs2->cell_len, xs3 );
    ys0 = xs3->bottom;
    ys2 = xs2->bottom;
/*--mul xs1 xs2--*/
    while( ys2 != NULL ){
        ys3 = ys0;
        ys1 = xs1->bottom;
    /*--add xs3 (mul x xs2)--*/
        while( ys1 != NULL ){
            ys3->bigit = ys1->bigit * ys2->bigit + ys3->bigit;
            ys1 = ys1->upper;
            ys3 = ys3->upper;
        }
        ys0 = ys0->upper;
        ys2 = ys2->upper;
    }
/*--normalize--*/
    norm( xs3 );
/*--clean--*/
    if( xs1->state == NONREF ){
        clear_vint( xs1 );
        free_vint( xs1 );
    }
    if( xs2->state == NONREF ){
        clear_vint( xs2 );
        free_vint( xs2 );
    }
    xs3->state = NONREF;
#ifdef DEBUG
    depth --;
#endif
```

```
    return( xs3 );
}

/*---------------------------------------------------------------------
                        vdiv
---------------------------------------------------------------------*/
VINT *vdiv( xs1, xs2 )
VINT *xs1, *xs2;
{
    int state1, state2;
    VINT *qs = NULL, *rs = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"vdiv\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs1, CHECK ) == FALSE || check_vint( xs2, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vdiv: Argument \"xs1\" or \"xs2\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs1->cell_len * xs2->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vdiv: Argument \"xs1\" or \"xs2\" has no value.\n");
        exit(1);
    }
    state1 = xs1->state;
    xs1->state = REF;
    state2 = xs2->state;
    xs2->state = REF;
    if( veq( xs2, itovi( 0 ) ) ){
        fprintf(stderr,"\n");
        fprintf(stderr,"Vdiv: \"xs1\"/0 is not proper.\n");
#ifdef DEBUG
        depth --;
#endif
        return( NULL );
    }

/*--vqrm--*/
    vqrm( qs, rs, xs1, xs2 );

/*--clean--*/
    xs1->state = state1;
    xs2->state = state2;
    if( xs1->state == NONREF ){
        clear_vint( xs1 );
        free_vint( xs1 );
    }
    if( xs2->state == NONREF ){
        clear_vint( xs2 );
        free_vint( xs2 );
    }
/*  clear_vint( qs );  */
    clear_vint( rs );
    free_vint( rs );
    qs->state = NONREF;
#ifdef DEBUG
    depth --;
#endif
    return( qs );
}

/*---------------------------------------------------------------------
                        vmod
---------------------------------------------------------------------*/
```

```
VINT *vmod( xs1, xs2 )
VINT *xs1, *xs2;
{
    int state1, state2;
    VINT *qs = NULL, *rs = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"vmod\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs1, CHECK ) == FALSE || check_vint( xs2, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vmod: Argument \"xs1\" or \"xs2\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs1->cell_len * xs2->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vmod: Argument \"xs1\" or \"xs2\" has no value.\n");
        exit(1);
    }
    state1 = xs1->state;
    xs1->state = REF;
    state2 = xs2->state;
    xs2->state=REF;
    if( veq( xs2, itovi( 0 ) ) ){
        fprintf(stderr,"\n");
        fprintf(stderr,"Vmod: \"xs1\"/0 is not proper.\n");
#ifdef DEBUG
        depth --;
#endif
        return( NULL );
    }

/*--vqrm--*/
    vqrm( qs, rs, xs1, xs2 );

/*--clean--*/
    xs1->state = state1;
    xs2->state = state2;
    if( xs1->state == NONREF ){
        clear_vint( xs1 );
        free_vint( xs1 );
    }
    if( xs2->state == NONREF ){
        clear_vint( xs2 );
        free_vint( xs2 );
    }
    clear_vint( qs );
/*  clear_vint( rs ); */
    free_vint( qs );
    rs->state = NONREF;
#ifdef DEBUG
    depth --;
#endif
    return( rs );
}

/*-------------------------------------------------------------------
                        qrm
-------------------------------------------------------------------*/
void qrm( xsq, xsr, xs1, xs2 )
VINT *xsq, *xsr, *xs1, *xs2;
{
    int state1, state2, i, q, d, r0, sign;
    VINT *xs10 = NULL, *xs20 = NULL, *qs = NULL, *rs = NULL,
         *rs0 = NULL, *rs1 = NULL, *rs2 = NULL;
```

```
    VINT_CELL *ysr = NULL, *ys1 = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"qrm\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xsq, CHECK ) == FALSE || check_vint( xsr, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vqrm: Argument \"xsq\" or \"xsr\" ");
        fprintf(stderr,"needs \"get_vint\".\n");
        exit(1);
    }else if( check_vint( xs1, CHECK ) == FALSE || check_vint( xs2, CHECK ) == FALSE){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vqrm: Argument \"xs1\" or \"xs2\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs1->cell_len * xs2->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stdout,"Vqrm: Argument \"xs1\" or \"xs2\" has no value.\n");
        exit(1);
    }
    state1 = xs1->state;
    xs1->state = REF;
    state2 = xs2->state;
    xs2->state = REF;
    if( veq( xs2, itovi( 0 ) ) ){
        fprintf(stderr,"\n");
        fprintf(stderr,"Vqrm: \"xs1\"/0 is not proper.\n");
#ifdef DEBUG
        depth --;
#endif
        return;
    }
/*--bmul xs1,bmul xs2--*/
    if( xs2->top->bigit == -1 ){
        d = - divide( BIGIT, BIGIT - xs2->top->lower->bigit );
    }else{
        d = divide( BIGIT, xs2->top->bigit + 1 );
    }
    if( d != 1 ){
        vassign( xs10, bmul( xs1, d ) );
        vassign( xs20, bmul( xs2, d ) );
    }else{
        vassign( xs10, xs1 );
        vassign( xs20, xs2 );
    }
    if( xs10->top->bigit == -1 ){
        sign = MINUS;
    }else{
        sign = PLUS;
    }
/*--take m xs1, drop m xs1--*/
    i = 0;
    rs = get_vint();
    rs->top = get_vcell();
    rs->bottom = rs->top;
    rs->cell_len ++;
    ysr = rs->top;
    ysr->bigit = 0;
    i ++;
    ys1 = xs10->top;
    while( ys1->lower != NULL && i < xs20->cell_len + sign ){
        ysr->bigit = ys1->bigit;
        ysr->lower = get_vcell();
        rs->cell_len ++;
        ysr->lower->upper = ysr;
```

```
        ysr = ysr->lower;
        rs->bottom = ysr;
        ys1 = ys1->lower;
        i ++;
    }
/*--dstep--*/
    vassign( qs, itovi( 0 ) );
    do{
        rs->bottom->bigit = ys1->bigit;
        ys1 = ys1->lower;
        if( rs->cell_len < xs20->cell_len + sign ){
    /*--astep--*/
            qs->bottom->lower = get_vcell();
            qs->cell_len++;
            qs->bottom->lower->upper = qs->bottom;
            qs->bottom = qs->bottom->lower;
            qs->bottom->bigit = 0;
        }else if( rs->cell_len == xs20->cell_len + sign ){
    /*--bstep--*/
            if( vless( rs, xs20 ) == TRUE ){
                qs->bottom->lower = get_vcell();
                qs->cell_len ++;
                qs->bottom->lower->upper = qs->bottom;
                qs->bottom = qs->bottom->lower;
                qs->bottom->bigit = 0;
            }else{
                qs->bottom->lower = get_vcell();
                qs->cell_len ++;
                qs->bottom->lower->upper = qs->bottom;
                qs->bottom = qs->bottom->lower;
                qs->bottom->bigit = 1;
                vassign( rs, vsub( rs, xs20 ) );
                sign = PLUS;
            }
        }else{
    /*--cstep--*/
        /*--guess--*/
            if( sign == PLUS ){
                r0 = rs->top->bigit;
            }else{
                r0 = rs->top->lower->bigit - BIGIT;
            }
            if( r0 >= xs20->top->bigit ){
                q = BIGIT - 1;
            }else if( sign == PLUS ){
                q = divide( rs->top->bigit * BIGIT + rs->top->lower->bigit,
                                                        xs20->top->bigit );
            }else{
                q = divide( rs->top->lower->bigit * BIGIT
                       + rs->top->lower->lower->bigit - BIGIT * BIGIT,
                                                        xs20->top->bigit );
            }
            q -= 2;
            vassign( rs0, vsub( rs, bmul( xs20, q ) ) );
            vassign( rs1, vsub( rs0, xs20 ) );
            vassign( rs2, vsub( rs1, xs20 ) );
            if( vless( rs0, xs20 ) == TRUE ){
        /*--q--*/
                qs->bottom->lower = get_vcell();
                qs->cell_len ++;
                qs->bottom->lower->upper = qs->bottom;
                qs->bottom = qs->bottom->lower;
                qs->bottom->bigit = q;
                vassign( rs, rs0 );
            }else if( vless( rs1, xs20 ) == TRUE ){
        /*--q+1--*/
                qs->bottom->lower = get_vcell();
                qs->cell_len ++;
                qs->bottom->lower->upper = qs->bottom;
                qs->bottom = qs->bottom->lower;
                qs->bottom->bigit = q + 1;
                vassign( rs, rs1 );
            }else{
        /*--q+2--*/
                qs->bottom->lower = get_vcell();
```

```
                qs->cell_len ++;
                qs->bottom->lower->upper = qs->bottom;
                qs->bottom = qs->bottom->lower;
                qs->bottom->bigit = q + 2;
                vassign( rs, rs2 );
            }
            sign = PLUS;
        }
        if( ys1 != NULL ){
            rs->bottom->lower = get_vcell();
            rs->cell_len ++;
            rs->bottom->lower->upper = rs->bottom;
            rs->bottom = rs->bottom->lower;
        }
    }while( ys1 != NULL );
/*--normalize--*/
    norm( qs );
/*--mod arrange--*/
    if( rs->top->bigit == -1 ){
        vassign( rs, vadd( xs20, rs ) );
        vassign( qs, vsub( qs, itovi( 1 ) ) );
    }
    if( d != 1 ){
        vassign( rs, bdiv( rs ,d ) );
    }
    if( rs->top->bigit == -1 ){
        vassign( rs, vsub( rs, xs2 ) );
        vassign( qs, vadd( qs, itovi( 1 ) ) );
    }
/*--clean--*/
    clear_vint( xs10 );
    free_vint( xs10 );
    clear_vint( xs20 );
    free_vint( xs20 );
    if( rs0 != NULL ){
        clear_vint( rs0 );
        free_vint( rs0 );
    }
    if( rs1 != NULL ){
        clear_vint( rs1 );
        free_vint( rs1 );
    }
    if( rs2 != NULL ){
        clear_vint( rs2 );
        free_vint( rs2 );
    }
    xs1->state = state1;
    xs2->state = state2;
    if( xs1->state == NONREF ){
        clear_vint( xs1 );
        free_vint( xs1 );
    }
    if( xs2->state == NONREF ){
        clear_vint( xs2 );
        free_vint( xs2 );
    }
    qs->state = NONREF;
    rs->state = NONREF;
    assign( xsq, qs );
    assign( xsr, rs );
#ifdef DEBUG
    depth --;
#endif
    return;
}

/*----------------------------------------------------------------
                        veq
----------------------------------------------------------------*/
int veq( xs1, xs2 )
VINT *xs1, *xs2;
{
    int state1, state2;
    VINT *xs01 = NULL, *xs02 = NULL;
    VINT_CELL *ys1 = NULL, *ys2 = NULL;
```

```
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"veq\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs1, CHECK ) == FALSE || check_vint( xs2, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Veq: Argument \"xs1\" or \"xs2\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs1->cell_len * xs2->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Veq: Argument \"xs1\" or \"xs2\" has no value.\n");
#ifdef DEBUG
        depth --;
#endif
        return( FALSE );
    }
    state1 = xs1->state;
    xs1->state = REF;
    state2 = xs2->state;
    xs2->state = REF;
/*--copy--*/
    vassign( xs01, xs1 );
    vassign( xs02, xs2 );
/*--align--*/
    align( xs01, xs02 );
    ys1 = xs01->top;
    ys2 = xs02->top;
/*--compare--*/
    while( ys1 != NULL ){
        if( ys1->bigit != ys2->bigit ){
        /*--clean--*/
            clear_vint( xs01 );
            free_vint( xs01 );
            clear_vint( xs02 );
            free_vint( xs02 );
            xs1->state = state1;
            xs2->state = state2;
            if( xs1->state == NONREF ){
                clear_vint( xs1 );
                free_vint( xs1 );
            }
            if( xs2->state == NONREF ){
                clear_vint( xs2 );
                free_vint( xs2 );
            }
#ifdef DEBUG
            depth --;
#endif
            return( FALSE );
        }
        ys1 = ys1->lower;
        ys2 = ys2->lower;
    }
/*--clean--*/
    clear_vint( xs01 );
    free_vint( xs01 );
    clear_vint( xs02 );
    free_vint( xs02 );
    xs1->state = state1;
    xs2->state = state2;
    if( xs1->state == NONREF ){
        clear_vint( xs1 );
        free_vint( xs1 );
    }
```

```
    if( xs2->state == NONREF ){
        clear_vint( xs2 );
        free_vint( xs2 );
    }
#ifdef DEBUG
    depth --;
#endif
    return( TRUE );
}

/*---------------------------------------------------------------------
                        vleq
---------------------------------------------------------------------*/
int vleq( xs1, xs2 )
VINT *xs1, *xs2;
{
    int state1, state2;
    VINT *xs01 = NULL, *xs02 = NULL;
    VINT_CELL *ys1 = NULL, *ys2 = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"vleq\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs1, CHECK ) == FALSE || check_vint( xs2, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vleq: Argument \"xs1\" or \"xs2\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs1->cell_len * xs2->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vleq: Argument \"xs1\" or \"xs2\" has no value.\n");
#ifdef DEBUG
        depth --;
#endif
        return( FALSE );
    }
    state1 = xs1->state;
    xs1->state = REF;
    state2 = xs2->state;
    xs2->state = REF;
/*--copy--*/
    vassign( xs01, xs1 );
    vassign( xs02, xs2 );
/*--align--*/
    align( xs01, xs02 );
    ys1 = xs01->top;
    ys2 = xs02->top;
/*--compare--*/
    while( ys1 != NULL ){
        if( ys1->bigit < ys2->bigit ){
        /*--clean--*/
            clear_vint( xs01 );
            free_vint( xs01 );
            clear_vint( xs02 );
            free_vint( xs02 );
            xs1->state = state1;
            xs2->state = state2;
            if( xs1->state == NONREF ){
                clear_vint( xs1 );
                free_vint( xs1 );
            }
            if( xs2->state == NONREF ){
                clear_vint( xs2 );
                free_vint( xs2 );
            }
```

```
#ifdef DEBUG
            depth --;
#endif
            return( TRUE );
        }else if( ys1->bigit > ys2->bigit ){
        /*--clean--*/
            clear_vint( xs01 );
            free_vint( xs01 );
            clear_vint( xs02 );
            free_vint( xs02 );
            xs1->state = state1;
            xs2->state = state2;
            if(xs1->state == NONREF){
                clear_vint( xs1 );
                free_vint( xs1 );
            }
            if( xs2->state == NONREF ){
                clear_vint( xs2 );
                free_vint( xs2 );
            }
#ifdef DEBUG
            depth --;
#endif
            return( FALSE );
        }
        ys1 = ys1->lower;
        ys2 = ys2->lower;
    }
/*--clean--*/
    clear_vint( xs01 );
    free_vint( xs01 );
    clear_vint( xs02 );
    free_vint( xs02 );
    xs1->state = state1;
    xs2->state = state2;
    if( xs1->state == NONREF ){
        clear_vint( xs1 );
        free_vint( xs1 );
    }
    if( xs2->state == NONREF ){
        clear_vint( xs2 );
        free_vint( xs2 );
    }
#ifdef DEBUG
    depth --;
#endif
    return( TRUE );
}

/*---------------------------------------------------------------------
                        vless
---------------------------------------------------------------------*/
int vless( xs1, xs2 )
VINT *xs1, *xs2;
{
    int state1, state2;
    VINT *xs01 = NULL, *xs02 = NULL;
    VINT_CELL *ys1 = NULL, *ys2 = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"vless\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs1, CHECK ) == FALSE || check_vint( xs2, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vless: Argument \"xs1\" or \"xs2\" is not proper.\n");
        exit(1);
```

```
    }else
#endif
    if( xs1->cell_len * xs2->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Vless: Argument \"xs1\" or \"xs2\" has no value.\n");
#ifdef DEBUG
        depth --;
#endif
        return( FALSE );
    }
    state1 = xs1->state;
    xs1->state  =REF;
    state2 = xs2->state;
    xs2->state = REF;
/*--copy--*/
    vassign( xs01, xs1 );
    vassign( xs02, xs2 );
/*--align--*/
    align( xs01, xs02 );
    ys1 = xs01->top;
    ys2 = xs02->top;
/*--compare--*/
    while( ys1 != NULL ){
        if( ys1->bigit < ys2->bigit ){
        /*--clean--*/
            clear_vint( xs01 );
            free_vint( xs01 );
            clear_vint( xs02 );
            free_vint( xs02 );
            xs1->state = state1;
            xs2->state = state2;
            if( xs1->state == NONREF ){
                clear_vint( xs1 );
                free_vint( xs1 );
            }
            if( xs2->state == NONREF ){
                clear_vint( xs2 );
                free_vint( xs2 );
            }
#ifdef DEBUG
            depth --;
#endif
            return( TRUE );
        }else if( ys1->bigit > ys2->bigit ){
        /*--clean--*/
            clear_vint( xs01 );
            free_vint( xs01 );
            clear_vint( xs02 );
            free_vint( xs02 );
            xs1->state = state1;
            xs2->state = state2;
            if( xs1->state == NONREF ){
                clear_vint( xs1 );
                free_vint( xs1 );
            }
            if( xs2->state == NONREF ){
                clear_vint( xs2 );
                free_vint( xs2 );
            }
#ifdef DEBUG
            depth --;
#endif
            return( FALSE );
        }
        ys1 = ys1->lower;
        ys2 = ys2->lower;
    }
/*--clean--*/
    clear_vint( xs01 );
    free_vint( xs01 );
    clear_vint( xs02 );
    free_vint( xs02 );
    xs1->state = state1;
    xs2->state = state2;
    if( xs1->state == NONREF ){
```

```
        clear_vint( xs1 );
        free_vint( xs1 );
    }
    if( xs2->state == NONREF ){
        clear_vint( xs2 );
        free_vint( xs2 );
    }
#ifdef DEBUG
    depth --;
#endif
    return( FALSE );
}

/*---------------------------------------------------------------------
                        put_vint
---------------------------------------------------------------------*/
void put_vint( xs )
VINT *xs;
{
    int state, b;
    VINT *xs1 = NULL;
    VINT_CELL *ys = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"put_vint\n");
#endif
#endif

    if( xs == NULL ){
       fprintf(stderr,"\nPut_vint: Argument \"xs\" is NULL.\n");
#ifdef DEBUG
       depth --;
#endif
       return;
   }

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Put_vint: Argument \"xs\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs->cell_len == 0 ){
        fprintf(stdout,"NO VALUE");
#ifdef DEBUG
        depth --;
#endif
        return;
    }
    ys = xs->top;
    state = xs->state;
    xs->state = REF;
/*--strep--*/
    while( ys->bigit == 0 && ys->lower != NULL ){
        ys = ys->lower;
    }
/*--print--*/
    if( ys->bigit >= 0 ){
        fprintf(stdout,"%d",ys->bigit);
        ys = ys->lower;
        while( ys != NULL ){
        /*  fprintf(stdout,",");  */
        /*--print 0--*/
            if( ys->upper != NULL ){
                b = BIGIT / 10;
                while( ys->bigit < b && b != 1 ){
                    fprintf(stdout,"0");
                    b /= 10;
```

```
                }
            }
        /*--print bigit--*/
            fprintf(stdout,"%d",ys->bigit);
            ys = ys->lower;
        }
/*--negative case--*/
    }else{
    /*--print "-"--*/
        fprintf(stdout,"-");
    /*--copy--*/
        vassign( xs1, xs );
    /*--negate--*/
        negate( xs1 );
    /*--print--*/
        put_vint( xs1 );
    /*--cean--*/
        clear_vint( xs1 );
        free_vint( xs1 );
    }
/*--clean--*/
    xs->state = state;
    if( xs->state == NONREF ){
        clear_vint( xs );
        free_vint( xs );
    }

#ifdef DEBUG
    depth --;
#endif
    return;
}

/*---------------------------------------------------------------------
                        fput_vint
---------------------------------------------------------------------*/
void fput_vint( xs, fp )
VINT *xs;
FILE *fp;
{
    int state, b;
    VINT *xs1 = NULL;
    VINT_CELL *ys = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"fput_vint\n");
#endif
#endif

    if( xs == NULL ){
       fprintf(stderr,"\nFput_vint: Argument \"xs\" is NULL.\n");
#ifdef DEBUG
       depth --;
#endif
       return;
   }

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Put_vint: Argument \"xs\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs->cell_len == 0 ){
        fprintf(fp,"NO VALUE");
#ifdef DEBUG
        depth --;
#endif
```

```
        return;
    }
    ys = xs->top;
    state = xs->state;
    xs->state = REF;
/*--strep--*/
    while( ys->bigit == 0 && ys->lower != NULL ){
        ys = ys->lower;
    }
/*--print--*/
    if( ys->bigit >= 0 ){
        fprintf(fp,"%d",ys->bigit);
        ys = ys->lower;
        while( ys != NULL ){
        /*  fprintf(fp,","); */
        /*--print 0*--*/
            if( ys->upper != NULL ){
                b = BIGIT / 10;
                while( ys->bigit < b && b != 1 ){
                    fprintf(fp,"0");
                    b /= 10;
                }
            }
        /*--print bigit--*/
            fprintf(fp,"%d",ys->bigit);
            ys = ys->lower;
        }
/*--negative case--*/
    }else{
    /*--print "-"--*/
        fprintf(fp,"-");
    /*--copy--*/
        vassign( xs1, xs );
    /*--negate--*/
        negate( xs1 );
    /*--print--*/
        put_vint( xs1 );
    /*--cean--*/
        clear_vint( xs1 );
        free_vint( xs1 );
    }
/*--clean--*/
    xs->state = state;
    if( xs->state == NONREF ){
        clear_vint( xs );
        free_vint( xs );
    }

#ifdef DEBUG
    depth --;
#endif
    return;
}

/*---------------------------------------------------------------------
                    put_vint2 for debug (do not free)
---------------------------------------------------------------------*/
void put_vint2( xs )
VINT *xs;
{
    int state, b;
    VINT *xs1 = NULL;
    VINT_CELL *ys = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"put_vint2\n");
#endif
#endif

    if( xs == NULL ){
```

```
       fprintf(stderr,"\nPut_vint: Argument \"xs\" is NULL.\n");
#ifdef DEBUG
       depth --;
#endif
       return;
   }

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Put_vint2: Argument \"xs\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs->cell_len == 0 ){
        fprintf(stdout,"NO VALUE");
#ifdef DEBUG
        depth --;
#endif
        return;
    }
    ys = xs->top;
    state = xs->state;
    xs->state = REF;
/*--strep--*/
    while( ys->bigit == 0 && ys->lower != NULL ){
        ys = ys->lower;
    }
/*--print--*/
    if( ys->bigit >= 0 ){
        fprintf(stdout,"%d",ys->bigit);
        ys = ys->lower;
        while( ys != NULL ){
        /*  fprintf(stdout,",");  */
        /*--print 0--*/
            if( ys->upper != NULL ){
                b = BIGIT / 10;
                while( ys->bigit < b && b != 1 ){
                    fprintf(stdout,"0");
                    b /= 10;
                }
            }
        /*--print bigit--*/
            fprintf(stdout,"%d",ys->bigit);
            ys = ys->lower;
        }
/*--negative case--*/
    }else{
    /*--print "-"--*/
        fprintf(stdout,"-");
    /*--copy--*/
        vassign( xs1, xs );
    /*--negate--*/
        negate( xs1 );
    /*--print--*/
        put_vint( xs1 );
    /*--cean--*/
        clear_vint( xs1 );
        free_vint( xs1 );
    }
/*--clean--*/
    xs->state = state;
/*
    if( xs->state == NONREF ){
        clear_vint( xs );
        free_vint( xs );
    }
*/
#ifdef DEBUG
    depth --;
#endif
    return;
}
```

```
/*****************************************************************
            cell handling functions (for internal use)
*****************************************************************/

/*---------------------------------------------------------------
                        get_vint
---------------------------------------------------------------*/
VINT *get_vint()
{
    VINT *xs = NULL;
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"get_vint\n");
#endif
#endif

/*--allocate--*/
    xs = (VINT *) malloc( sizeof( VINT ) );
#ifdef DEBUG
    check_vint( xs, ADD );
#endif
/*--initialize--*/
    xs->cell_len = 0;        /*                    */
    xs->state = REF;          /*  init_vint( xs );  */
    xs->top = NULL;          /*                    */
    xs->bottom = NULL;       /*                    */
#ifdef DEBUG
    depth --;
#endif
    return( xs );
}

/*---------------------------------------------------------------
                        get_vcell
---------------------------------------------------------------*/
VINT_CELL *get_vcell()
{
    VINT_CELL *ys = NULL;
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"get_vcell\n");
#endif
#endif

/*--allocate--*/
    ys = (VINT_CELL *) malloc( sizeof( VINT_CELL ) );
/*--initialize--*/
    ys->bigit = 0;            /*                     */
    ys->upper = NULL;         /*  init_vcell( ys );  */
    ys->lower = NULL;         /*                     */
#ifdef DEBUG
    depth --;
#endif
    return( ys );
}

/*---------------------------------------------------------------
                        init_vint
---------------------------------------------------------------*/
void init_vint( xs )
VINT *xs;
{
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
```

```
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"init_vint\n");
#endif
#endif

/*--initialize--*/
    xs->cell_len = 0;
    xs->state = REF;
    xs->top = NULL;
    xs->bottom = NULL;
#ifdef DEBUG
    depth --;
#endif
    return;
}

/*---------------------------------------------------------------
                        init_vcell
---------------------------------------------------------------*/
void init_vcell( ys )
VINT_CELL *ys;
{
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"init_vcell\n");
#endif
#endif

/*--initialize--*/
    ys->bigit = 0;
    ys->upper = NULL;
    ys->lower  =NULL;
#ifdef DEBUG
    depth --;
#endif
    return;
}

/*---------------------------------------------------------------
                        clear_vint
---------------------------------------------------------------*/
void clear_vint( xs )
VINT *xs;
{
    VINT_CELL *ys = NULL;
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"clear_vint\n");
#endif
#endif

/*--free vcells--*/
    if( xs != NULL ){
#ifdef DEBUG
        if( xs != NULL && check_vint( xs, VASGN ) == TRUE ){
#endif
            while( xs->top != NULL ){
                ys = xs->top;
                xs->top = ys->lower;
                if( xs->top != NULL )
                    xs->top->upper = NULL;
                free_vcell( ys );
            }
#ifdef DEBUG
        }
#endif
```

```
/*--initialize--*/
        xs->cell_len = 0;        /*                    */
        xs->state = REF;          /*  init_vint( xs );  */
        xs->top = NULL;          /*                    */
        xs->bottom = NULL;       /*                    */
    }
#ifdef DEBUG
    depth --;
#endif
    return;
}

/*---------------------------------------------------------------------
                        free_vint0
---------------------------------------------------------------------*/
int free_vint0( xs )
VINT *xs;
{
    int state;
    VINT_CELL *ys = NULL;
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"free_vint\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( xs !=NULL && check_vint( xs, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Free_vint: Argument \"xs\" is not proper.\n");
        exit(1);
    }
#endif
/*--initialize--*/
    if( xs != NULL ){
        state = xs->state;
        xs->cell_len = 0;        /*                    */
        xs->state = REF;          /*  init_vint( xs );  */
        xs->top = NULL;          /*                    */
        xs->bottom = NULL;       /*                    */
#ifdef DEBUG
        check_vint( xs, DEL );
#endif
        free( xs );
#ifdef DEBUG
        depth --;
#endif
        return( state );
    }
#ifdef DEBUG
    depth --;
#endif
    return( REF );
}

/*---------------------------------------------------------------------
                        free_vcell0
---------------------------------------------------------------------*/
void free_vcell0( ys )
VINT_CELL *ys;
{
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"free_vcell\n");
#endif
```

```
#endif

/*--initialize--*/
    if( ys != NULL ){
        ys->bigit = 0;
        ys->upper = NULL;
        ys->lower = NULL;
        free( ys );
    }
#ifdef DEBUG
    depth --;
#endif
    return;
}


/*****************************************************************
    sub-routines for public use functions (for internal use)
*****************************************************************/

/*---------------------------------------------------------------
                        strep
---------------------------------------------------------------*/
void strep( xs )
VINT *xs;
{
    int zero;
    VINT_CELL *ys = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"strep\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Strep: Argument \"xs\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Strep: Argument \"xs\" has no value.\n");
        exit(1);
    }else if( xs->state == NONREF ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Strep: Argument \"xs\" is a function result.\n");
        exit(1);
    }
/*--dropwhile =0--*/
    while( xs->top->bigit == 0 && xs->top->lower != NULL ){
        xs->top = xs->top->lower;
        free_vcell( xs->top->upper );
        xs->top->upper = NULL;
        xs->cell_len --;
    }
/*--negative case--*/
    zero = FALSE;
    while( xs->top->bigit == -1 && xs->cell_len > 2
        && xs->top->lower->bigit == BIGIT - 1
            && ( xs->top->lower->lower->bigit != 0 || zero == FALSE ) ){
        if( xs->top->lower->lower->bigit != 0 ){
            xs->top->lower = xs->top->lower->lower;
            free_vcell( xs->top->lower->upper );
            xs->top->lower->upper = xs->top;
            xs->cell_len --;
        }else{
            zero == TRUE;
```

```
            ys = xs->top->lower->lower;
            while( ys != NULL && zero == TRUE ){
                if( ys->bigit != 0 )
                    zero = FALSE;
                ys = ys->lower;
            }
            if( zero == FALSE ){
                xs->top->lower = xs->top->lower->lower;
                free_vcell( xs->top->lower->upper );
                xs->top->lower->upper = xs->top;
                xs->cell_len --;
            }
        }
    }
#ifdef DEBUG
    depth --;
#endif
    return;
}

/*---------------------------------------------------------------------
                        copy_vcell
---------------------------------------------------------------------*/
void copy_vcell( bigit, n, xs )
int bigit, n;
VINT *xs;
{
    int i;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"copy_vcell\n");
#endif
#endif

    if( n == 0 ){
#ifdef DEBUG
        depth --;
#endif
        return;
    }
/*--error--*/
    if( n < 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Copy_vcell: Argument \"n\" is not proper.\n");
        exit(1);
    }else
#ifdef DEBUG
    if( check_vint( xs, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Copy_vcell: Argument \"xs\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs->state == NONREF ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Copy_vcell: Argument \"xs\" is a function result.\n");
        exit(1);
    }
/*--no vcell--*/
    if( xs->cell_len == 0 ){
        xs->bottom = get_vcell();
        xs->cell_len ++;
        xs->top = xs->bottom;
        xs->bottom->bigit = bigit;
    }else{
/*--otherwise--*/
        xs->top->upper = get_vcell();
        xs->cell_len ++;
        xs->top->upper->lower = xs->top;
        xs->top = xs->top->upper;
```

```
        xs->top->upper = NULL;
        xs->top->bigit = bigit;
    }
    copy_vcell( bigit, n-1, xs );
#ifdef DEBUG
    depth --;
#endif
    return;
}

/*---------------------------------------------------------------------
                        align
---------------------------------------------------------------------*/
void align( xs1, xs2 )
VINT *xs1, *xs2;
{
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"align\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs1, CHECK ) == FALSE || check_vint( xs2, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Align: Argument \"xs1\" or \"xs2\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs1->cell_len * xs2->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Align: Argument \"xs1\" or \"xs2\" has no value.\n");
        exit(1);
    }else if( xs1->state == NONREF || xs2->state == NONREF ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Align: Argument \"xs1\" or \"xs2\" is a function result.\n");
        exit(1);
    }
/*--#xs1 > #xs2--*/
    if( xs1->cell_len > xs2->cell_len ){
        copy_vcell( 0, xs1->cell_len - xs2->cell_len, xs2 );
    }else{
/*--otherwise--*/
        copy_vcell( 0, xs2->cell_len - xs1->cell_len, xs1 );
    }
#ifdef DEBUG
    depth --;
#endif
    return;
}

/*---------------------------------------------------------------------
                        norm
---------------------------------------------------------------------*/
void norm( xs )
VINT *xs;
{
    VINT_CELL *ys = NULL;
    int carry;
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"norm\n");
#endif
#endif

/*--error--*/
```

```
#ifdef DEBUG
    if( check_vint( xs, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Norm: Argument \"xs\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Norm: Argument \"xs\" has no value.\n");
        exit(1);
    }else if( xs->state == NONREF ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Norm: Argument \"xs\" is a function result.\n");
        exit(1);
    }
    ys = xs->bottom;
    carry = 0;
/*--normalize--*/
    do{
        ys->bigit += carry;
        carry = divide( ys->bigit, BIGIT );
        ys->bigit = mod( ys->bigit, BIGIT );
        ys = ys->upper;
    }while( ys != NULL );
    while( ( carry != 0 && carry != -1 ) || ( carry == -1 && xs->top->bigit == 0 ) ){
        ys = get_vcell();
        xs->top->upper = ys;
        ys->lower = xs->top;
        xs->top = ys;
        xs->cell_len ++;
        ys->bigit = mod( carry, BIGIT );
        carry = divide( carry, BIGIT );
    }
/*--negative case--*/
    if( carry == -1 ){
        ys = get_vcell();
        xs->top->upper = ys;
        ys->lower = xs->top;
        xs->top = ys;
        xs->cell_len ++;
        ys->bigit = carry;
    }
/*--strep--*/
    strep( xs );
#ifdef DEBUG
    depth --;
#endif
    return;
}

/*---------------------------------------------------------------------
                        negate
---------------------------------------------------------------------*/
void negate( xs )
VINT *xs;
{
    int carry;
    VINT_CELL *ys = NULL;
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"negate\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Negate: Argument \"xs\" is not proper.\n");
        exit(1);
```

```
    }else
#endif
    if( xs->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Negate: Argument \"xs\" has no value.\n");
        exit(1);
    }else if( xs->state == NONREF ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Negate: Argument \"xs\" is a function result.\n");
        exit(1);
    }
    ys = xs->bottom;
/*--negate--*/
    while( ys != NULL ){
        ys->bigit = - ys->bigit;
        ys = ys->upper;
    }
/*--normalize--*/
    norm( xs );
#ifdef DEBUG
    depth --;
#endif
    return;
}

/*---------------------------------------------------------------------
                        divide
---------------------------------------------------------------------*/
int divide( x, y )
int x, y;
{
    int q, r;
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"divide\n");
#endif
#endif

/*--error--*/
    if( y == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Div: \"div(%d,0)\" is not proper.\n",x);
        exit(1);
    }
    q = x / y;
    r = x % y;
    if( r < 0 ){
        if( y > 0 ){
            q --;
            r += y;
        }else{
            q ++;
            r -= y;
        }
    }
#ifdef DEBUG
    depth --;
#endif
    return( q );
}

/*---------------------------------------------------------------------
                        mod
---------------------------------------------------------------------*/
int mod( x, y )
int x, y;
{
    int q, r;
#ifdef DEBUG
    int i0;
    depth ++;
```

```
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"mod\n");
#endif
#endif

/*--error--*/
    if( y == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Mod: \"div(%d,0)\" is not proper.\n",x);
        exit(1);
    }
    q = x / y;
    r = x % y;
    if( r < 0 ){
        if( y > 0 ){
            q --;
            r += y;
        }else{
            q ++;
            r -= y;
        }
    }
#ifdef DEBUG
    depth --;
#endif
    return( r );
}

/*---------------------------------------------------------------------
                        bmul
---------------------------------------------------------------------*/
VINT *bmul( xs, x )
VINT *xs;
int x;
{
    int state;
    VINT *xs0 = NULL;
    VINT_CELL *ys0 = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"bmul\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Bmul: Argument \"xs\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Bmul: Argument \"xs\" has no value.\n");
        exit(1);
    }
    state = xs->state;
    xs->state = REF;
/*--copy--*/
    vassign( xs0, xs );
    ys0 = xs0->bottom;
/*--mul xs'vcell x--*/
    do{
        ys0->bigit *= x;
        ys0 = ys0->upper;
    }while( ys0 != NULL );
/*--normalize--*/
```

```
    norm( xs0 );
/*--clean--*/
    xs->state = state;
    if( xs->state == NONREF ){
        clear_vint( xs );
        free_vint( xs );
    }
    xs0->state = NONREF;
#ifdef DEBUG
    depth --;
#endif
    return( xs0 );
}

/*---------------------------------------------------------------------
                        bdiv
---------------------------------------------------------------------*/
VINT *bdiv( xs, x )
VINT *xs;
int x;
{
    int r;
    VINT *xs0 = NULL;
    VINT_CELL *ys0 = NULL, *ys = NULL;

#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"bdiv\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Bdiv: Argument \"xs\" is not proper.\n");
        exit(1);
    }else
#endif
    if( xs->cell_len == 0 ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Bdiv: Argument \"xs\" has no value.\n");
        exit(1);
    }else if( x == 0 ){
        fprintf(stderr,"\n");
        fprintf(stderr,"Bdiv: \"xs\"/0 is not proper.\n");
#ifdef DEBUG
        depth --;
#endif
        return( NULL );
    }
/*--divide--*/
    xs0 = get_vint();
    xs0->top = get_vcell();
    xs0->cell_len ++;
    xs0->bottom = xs0->top;
    ys0 = xs0->top;
    ys = xs->top;
    ys0->bigit = divide( ys->bigit, x );
    r = mod( ys->bigit, x );
    ys = ys->lower;
    while( ys != NULL ){
        ys0->lower = get_vcell();
        xs0->cell_len ++;
        ys0->lower->upper = ys0;
        xs0->bottom = ys0->lower;
        ys0 = ys0->lower;
        ys0->bigit = divide( r * BIGIT + ys->bigit, x );
        r = mod( r * BIGIT + ys->bigit, x );
        ys = ys->lower;
    }
```

```
/*--normalize--*/
    norm( xs0 );
/*--clean--*/
    if( xs->state == NONREF ){
        clear_vint( xs );
        free_vint( xs );
    }
    xs0->state = NONREF;
#ifdef DEBUG
    depth --;
#endif
    return( xs0 );
}

/*---------------------------------------------------------------------
                        disp_vcell
---------------------------------------------------------------------*/
void disp_vcell( xs )
VINT *xs;
{
    int state;
    VINT_CELL *ys = NULL;
#ifdef DEBUG
    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"disp_vcell\n");
#endif
#endif

/*--error--*/
#ifdef DEBUG
    if( check_vint( xs, CHECK ) == FALSE ){
        fprintf(stderr,"\n----Error----\n");
        fprintf(stderr,"Disp_vcell: Argument \"xs\" is not proper.\n");
    /*  put_vint2( xs );  */
        fprintf(stderr," (%2d) ",xs->cell_len);
        switch( xs->state ){
            case REF:
                fprintf(stderr,"REF state");
                break;
            case NONREF:
                fprintf(stderr,"NONREF state");
                break;
            case CONS:
                fprintf(stderr,"CONS state");
                break;
            default:
                fprintf(stderr,"Unknown state");
        }
        fprintf(stderr,": (%8d %8d)\n",xs->top,xs->bottom);
        ys = xs->top;
        while( ys != NULL ){
            fprintf(stderr,"%4d (%8d %8d %8d)\n",ys->bigit,ys->upper,ys,ys->lower);
            ys = ys->lower;
        }
        exit(1);        /* return; */
    }
#endif
/*--display--*/
    state = xs->state;
    xs->state = REF;
    put_vint( xs );
    xs->state = state;
    fprintf(stdout," (%2d) ",xs->cell_len);
    switch( xs->state ){
        case REF:
            fprintf(stdout,"REF state");
            break;
        case NONREF:
            fprintf(stdout,"NONREF state");
            break;
        case CONS:
```

```
            fprintf(stdout,"CONS state");
            break;
        default:
            fprintf(stdout,"Unknown state");
    }
    fprintf(stdout,": (%8d %8d)\n",xs->top,xs->bottom);
    ys = xs->top;
    while( ys != NULL ){
        fprintf(stdout,"%4d (%8d %8d %8d)\n",ys->bigit,ys->upper,ys,ys->lower);
        ys = ys->lower;
    }
/*--clean--*/
/*  if( xs->state == NONREF ){
        clear_vint( xs );
        free_vint( xs );
    }  */
#ifdef DEBUG
    depth --;
#endif
    return;
}

#ifdef DEBUG
/*----------------------------------------------------------------
                        check_vint
----------------------------------------------------------------*/
int check_vint( xs, opt )
VINT *xs;
int opt;
{
    int vcell_cnt;
    static int vint_cnt = 0;
    VINT_CELL *ys = NULL;
    static USING_VINT head = {NULL,NULL};
    USING_VINT *ptr = NULL;
    USING_VINT *pre_ptr = NULL;

    int i0;
    depth ++;
#ifdef DEPTH
    for( i0 = 0 ; i0 < depth ; i0 ++ )
        fprintf(stderr," ");
    fprintf(stderr,"check_vint");
#endif

    switch( opt ){

        case CHECK:
#ifdef DEPTH
            if( opt == CHECK )
                fprintf(stderr,"(CHECK)\n");
#endif
        case DETAIL:
#ifdef DEPTH
            if( opt == DETAIL )
                fprintf(stderr,"(DETAIL)\n");
#endif
        case VASGN:
#ifdef DEPTH
            if( opt == VASGN )
                fprintf(stderr,"(VASGN)\n");
#endif
    /*--check--*/
            ptr = head.next;
            while( xs != NULL && ptr != NULL ){
                if( ptr->vint == xs ){
                    if( opt = DETAIL ){
                /*--detail check--*/
                        if( xs->cell_len < 0 ){
                            fprintf(stderr,"\n----Check----\n");
                            fprintf(stderr,"Check_vint: xs->cell_len < 0 .\n");
                            depth --;
                            return( FALSE );
                        }else if( xs->cell_len == 0
                                && xs->top == NULL && xs->bottom == NULL ){
```

```
                        depth --;
                        return( TRUE );
                    }else if( xs->top == NULL ){
                        fprintf(stderr,"\n----Check----\n");
                        fprintf(stderr,"Check_vint: xs->top == NULL .\n");
                        depth --;
                        return( FALSE );
                    }else if( xs->top->upper != NULL ){
                        fprintf(stderr,"\n----Check----\n");
                        fprintf(stderr,"Check_vint: xs->top->upper == NULL .\n");
                        depth --;
                        return( FALSE );
                    }else{
                        ys = xs->top;
                        vcell_cnt = 1;
                        while( vcell_cnt <= xs->cell_len && ys->lower != NULL ){
                            ys = ys->lower;
                            vcell_cnt ++;
                        }
                        if( vcell_cnt != xs->cell_len || ys != xs->bottom ){
                            fprintf(stderr,"\n----Check----\n");
                            fprintf(stderr,"Check_vint: xs->top->lower->...->lower != xs->bottom .\n");
                            depth --;
                            return( FALSE );
                        }else{
                            depth --;
                            return( TRUE );
                        }
                    }
                }
                depth --;
                return( TRUE );
            }else{
                ptr=ptr->next;
            }
        }
        if( opt != VASGN ){
            fprintf(stderr,"\n----Check----\n");
            fprintf(stderr,"Check_vint: Argument \"xs\" is not used.\n");
            fprintf(stderr,"            [ VINT *xs; ] -> [ VINT *xs = NULL; ]\n");
            fprintf(stderr,"            [ VINT xs;  ] ->          X\n");
        }
        depth --;
        return( FALSE );

    case ADD:
#ifdef DEPTH
        fprintf(stderr,"(ADD)\n");
#endif
        if( xs == NULL ){
            depth --;
            return( TRUE );
        }
    /*--add--*/
        ptr = &head;
        while( ptr->next != NULL ){
            if( ptr->vint == xs ){
                depth --;
                return( TRUE );
            }else{
                ptr = ptr->next;
            }
        }
        ptr->next = (USING_VINT *) malloc( sizeof( USING_VINT ) );
        ptr->next->next = NULL;
        ptr->next->vint = xs;
        vint_cnt ++;
        depth --;
        return( TRUE );
    case DEL:
#ifdef DEPTH
        fprintf(stderr,"(DEL)\n");
#endif
        if( xs == NULL ){
            depth --;
```

```
                return( TRUE );
            }
    /*--delete--*/
            ptr = head.next;
            pre_ptr = &head;
            if( ptr == NULL ){
                depth --;
                return( FALSE );
            }
            while( ptr->vint != xs ){
                pre_ptr = ptr;
                ptr = ptr->next;
                if( ptr == NULL ){
                    depth --;
                    return( FALSE );
                }
            }
            pre_ptr->next = ptr->next;
            free( ptr );
            vint_cnt --;
            depth --;
            return( TRUE );
        default:
#ifdef DEPTH
            fprintf(stderr,"(DISP)\n");
#endif
    /*--display--*/
            fprintf(stderr,"Using Vint(%d):",vint_cnt);
            ptr = head.next;
            if( ptr == NULL ){
                fprintf(stderr," --  No Use --\n");
                depth --;
                return( TRUE );
            }else{
                fprintf(stderr," --  PTR --\n");
            }
            while( ptr != NULL ){
                fprintf(stderr,"            | %6d ( %4d cells ) |\n",ptr->vint,ptr->vint->cell_len);
                ptr = ptr->next;
            }
            depth --;
            return( TRUE );
    }

}
#endif
```

- テスト用 main 関数ファイル test.c

```
/*================================================================
                          test.c
================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "vint.h"

#define STR_MAX 1000

/*----------------------------------------------------------------
                      main
----------------------------------------------------------------*/
main()
{
    char string[STR_MAX+1];
    VINT *xs1 = NULL, *xs2 = NULL, *xs3 = NULL, *xs4 = NULL;
    char input[10];

    do{
    /* Input xs1, xs2 */
        printf("Atovi : xs1 = ");
        scanf("%s",string);
```

```
    vassign( xs1, atovi( string ) );
    printf("Atovi0: xs2 = ");
    scanf("%s",string);
    vassign( xs2, atovi0( string ) );
/* Veq */
    printf("Veq  : ");
    put_vint( xs1 );
    printf(" == ");
    put_vint( xs2 );
    printf(" -> ");
    if( veq( xs1, xs2 ) == TRUE ){
        printf("TRUE");
    }else{
        printf("FALSE");
    }
    printf("\n");
/* Vleq */
    printf("Vleq : ");
    put_vint( xs1 );
    printf(" <= ");
    put_vint(xs2);
    printf(" -> ");
    if( vleq( xs1, xs2 ) == TRUE ){
        printf("TRUE");
    }else{
        printf("FALSE");
    }
    printf("\n");
/* Vless */
    printf("Vless: ");
    put_vint( xs1 );
    printf(" <  ");
    put_vint( xs2 );
    printf(" -> ");
    if( vless( xs1, xs2 ) == TRUE ){
        printf("TRUE");
    }else{
        printf("FALSE");
    }
    printf("\n");
/* Vadd */
    printf("Vadd : ");
    put_vint( xs1 );
    printf(" +  ");
    put_vint( xs2 );
    printf(" = ");
    vassign( xs3, vadd( xs1, xs2 ) );
    put_vint( xs3 );
    printf("\n");
/* Vsub */
    printf("Vsub : ");
    put_vint( xs1 );
    printf(" -  ");
    put_vint(xs2);
    printf(" = ");
    vassign( xs3, vsub( xs1, xs2 ) );
    put_vint( xs3 );
    printf("\n");
/* Vmul */
    printf("Vmul : ");
    put_vint( xs1 );
    printf(" *  ");
    put_vint( xs2 );
    printf(" = ");
    vassign( xs3, vmul( xs1, xs2 ) );
    put_vint( xs3 );
    printf("\n");
/* Vdiv, Vmod */
    printf("Vdiv : ");
    put_vint( xs1 );
    printf(" /  ");
    put_vint( xs2 );
    printf(" = ");
    vassign( xs3, vdiv( xs1, xs2 ) );
    put_vint( xs3 );
```

```
        printf(" ");
        printf("Vmod : ");
        vassign( xs3, vmod( xs1, xs2 ) );
        put_vint( xs3 );
        printf("\n");
    /* Vqrm */
        printf("Vqrm : ");
        put_vint( xs1 );
        printf(" /  ");
        put_vint( xs2 );
        printf(" = ");
        vqrm( xs3, xs4, xs1, xs2 );
        put_vint( xs3 );
        printf("   ...  ");
        put_vint( xs4 );
        printf("\n");
    /* Disp using vint */
#ifdef DEBUG
        check_vint( NULL, DISP );
#endif

        do{
            printf("Continue?(y/n) ");
            scanf("%s",input);
        }while( input[0] != 'y' && input[0] != 'n' );
    }while( input[0] == 'y' );
}
```

# 付 録C　可変長整数ライブラリを用いたプログラム例

可変長整数ライブラリを利用して指定された桁数の $e$ の値を求めるプログラムを作成した。

- 設計法に基づいたライブラリを使用した場合のプログラム vexp.c

```
/*=====================================================================
                            vexp.c
=====================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include "vint.h"

#define DEFAULT_KETA 100

VINT *e( VINT *x, int keta );
VINT *v10n( int n );

VINT *zero = NULL, *one = NULL, *ten = NULL;

void main( int arg, char *argv[] )
{
    long int n = DEFAULT_KETA;
    VINT *keta = NULL, *qs = NULL, *rs = NULL;

    if( arg > 1 )
        sscanf( argv[1], "%ld", &n );
    printf("KETA = %ld\n",n);
    vassign( zero, itovi( 0 ) );
    vassign( one, itovi( 1 ) );
    vassign( ten, itovi( 10 ) );
    vassign( keta, v10n( n ) );
    printf("e = ");
    vqrm( qs, rs, e( one, n ), keta );
```

```
    put_vint( qs );
    printf(".");
    put_vint( rs );
    printf("\nEnd.\n");
/* Disp using vint */
#ifdef DEBUG
    check_vint( NULL, DISP );
#endif
#ifdef STOP
    getchar();
#endif
}

VINT *e( VINT *x, int keta )
{
    int state;
    VINT *E_i = NULL, *res_prev = NULL, *res = NULL, *i = NULL;

    state = x->state;
    x->state = REF;

    vassign( E_i, one );
    for( ; keta > 0 ; keta -- )
        vassign( E_i, vmul( E_i, ten ) );

    vassign( res_prev, zero );
    vassign( res, E_i );

    for( vassign( i, one ) ; ! veq( E_i, zero ) ;
                                vassign( i, vadd( i, one ) ) ){
        vassign( res_prev, res );
        vassign( E_i, vdiv( vmul( x, E_i ), i ) );
        vassign( res, vadd( res_prev, E_i ) );
    }

    clear_vint( E_i );
    free_vint( E_i );
    clear_vint( res_prev );
    free_vint( res_prev );
    clear_vint( i );
    free_vint( i );
    x->state = state;
    if( x->state == NONREF ){
        clear_vint( x );
```

```
        free_vint( x );
    }

    res->state = NONREF;
    return( res );
}

VINT *v10n( int n )
{
    int i;
    VINT *keta = NULL;

    vassign( keta, itovi( 1 ) );
    for( i = 0 ; i < n / B_LEN ; i ++ )
        vassign( keta, vmul( keta, itovi( BIGIT ) ) );
    for( i = 0 ; i < n % B_LEN ; i ++ )
        vassign( keta, vmul( keta, ten ) );

    keta->state = NONREF;
    return( keta );
}
```

- 手動でごみ集めを行うライブラリを使用した場合のプログラム vexp-cmp.c

```
/*====================================================================
                            vexp-cmp.c
====================================================================*/
#include <stdio.h>
#include <stdlib.h>
#include "vint-cmp.h"

#define DEFAULT_KETA 100

VINT *e( VINT *x, int keta );
VINT *v10n( int n );

VINT *zero, *one, *ten;

void main( int arg, char *argv[] )
{
    long int n = DEFAULT_KETA;
    VINT *keta, *qs, *rs;
    VINT *p;
```

```
    if( arg > 1 )
        sscanf( argv[1], "%ld", &n );
    printf("KETA = %ld\n",n);
    zero = itovi( 0 );
    one = itovi( 1 );
    ten = itovi( 10 );
    keta = v10n( n );
    printf("e = ");
    p = e( one, n );
    qs = itovi( 0 );
    clear_vint( qs );
    rs = itovi( 0 );
    clear_vint( rs );
    qrm( qs, rs, p, keta );
    clear_vint( p );
    free_vint( p );
    put_vint( qs );
    printf(".");
    put_vint( rs );
    printf("\nEnd.\n");
/* Disp using vint */
#ifdef DEBUG
    check_vint( NULL, DISP );
#endif
#ifdef STOP
    getchar();
#endif
}

VINT *e( VINT *x, int keta )
{
    int state;
    VINT *E_i, *res_prev, *res, *i;
    VINT *p;

    E_i = itovi( 0 );
    clear_vint( E_i );
    copy( E_i, one );
    for( ; keta > 0 ; keta -- ){
        p = E_i;
        E_i = vmul( E_i, ten );
        clear_vint( p );
        free_vint( p );
    }
```

```
    res_prev = itovi( 0 );
    clear_vint( res_prev );
    copy( res_prev, zero );
    res = itovi( 0 );
    clear_vint( res );
    copy( res, E_i );

    i = itovi( 0 );
    clear_vint( i );
    for( copy( i, one ) ; ! veq( E_i, zero ) ; ({ p = i;
                                                  i = vadd( i, one );
                                                  clear_vint( p );
                                                  free_vint( p ); }) ){
        clear_vint( res_prev );
        copy( res_prev, res );
        p = vmul( x, E_i );
        clear_vint( E_i );
        free_vint( E_i );
        E_i = vdiv( p, i );
        clear_vint( p );
        free_vint( p );
        clear_vint( res );
        free_vint( res );
        res = vadd( res_prev, E_i );
    }

    clear_vint( E_i );
    free_vint( E_i );
    clear_vint( res_prev );
    free_vint( res_prev );
    clear_vint( i );
    free_vint( i );

    return( res );
}

VINT *v10n( int n )
{
    int i;
    VINT *keta;
    VINT *p1, *p2;

    keta = itovi( 1 );
```

```
    for( i = 0 ; i < n / B_LEN ; i ++ ){
        p1 = itovi( BIGIT );
        p2 = keta;
        keta = vmul( keta, p1 );
        clear_vint( p1 );
        free_vint( p1 );
        clear_vint( p2 );
        free_vint( p2 );
    }
    for( i = 0 ; i < n % B_LEN ; i ++ ){
        p1 = keta;
        keta = vmul( keta, ten );
        clear_vint( p1 );
        free_vint( p1 );
    }

    return( keta );
}
```