ソフトウェア工学 86－6
（１９９２．７．２１）

Ｃ言語用抽象データ型マクロ部品

浅見 秀雄
ＮＴＴソフトウェア研究所

データの宣言と操作を一体化して実現する抽象データ型の概念は、ソフトウェア部品化の手段として有効である。しかし、広く使われているプログラミング言語の多くはそのための言語機能を持っていないことから、一般的に利用されるには至っていない。本稿では、抽象データ型が言語機能として提供されていないＣ言語で、抽象データ型の概念に基づくソフトウェア部品を実現する実用的な方法として、プリプロセッサを利用する方法を提案する。

Abstract Data Types Macro Library for the Programming Language C

Hideo ASAMI
NTT Software Laboratories
Nippon Telegraph and Telephone Corporation
1-9-1 Konan, Minato-ku, Tokyo, 108 Japan

A concept of abstract data types, which defines data and operations in one body, is effective for realizing software parts. However, it is not used widely, because most programming languages don't support it. In this paper we propose a method to realize software parts based on the concept of abstract data types using the preprocessor of the programming language C.

1．はじめに

抽象データ型の概念に基づくプログラム設計では、データとそれに対する操作が一体として定義される[1]。これによる効果は次のとおりである。

①データの宣言とデータに対する操作の実現部分が局所化されるので、その実現方法の正しさの検証が容易になる。

②抽象データ型を利用するプログラムでは、詳細なデータ構造の宣言や、詳細な操作アルゴリズムを書く必要がないので、プログラミングが省力化され、誤りが減る。

③プログラムコードの再利用が促進される。

C言語は、抽象データ型をサポートするための言語機能を備えてはいない。しかし、そのプリプロセッサは、抽象データ型の概念に基づくマクロ部品を実現するために活用できる。

本稿では、C言語のプリプロセッサを利用したマクロ部品の実現について述べる。

2．抽象データ型の考え方

例として、シリアル通信の受信データを格納するキュー（先入れ先出しバッファ）にリングバッファを用いる場合を考える。リングバッファは、配列内のデータ要素格納位置がサイクリックに変化するようにしたものである。すなわち、「容量に限りはあるが終端のないバッファ」というデータ型として見せかけられる（図1）。

このようなリングバッファを取り扱う一般的なプログラミング方法は、図2に示すとおりである。

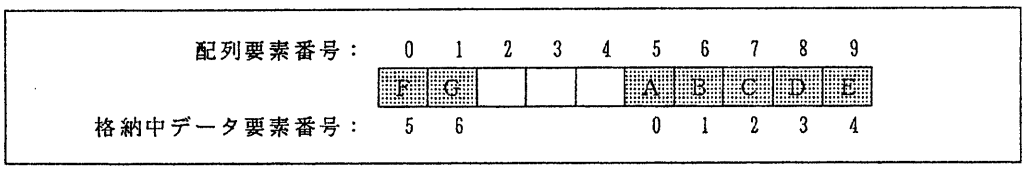

図1　リングバッファの構成

```
定義、宣言：
    #define CAPACITY 1000                    /*バッファ容量*/
    typedef struct { int head;               /*先頭データ要素の配列要素番号*/
                     int last;               /*最新データ要素番号（空のとき-1）*/
                     char place[CAPACITY];   /*データ要素格納域*/
                   } QUEUE;
    static QUEUE buf = {0,-1};
データ要素を入れる：
    if (buf.last<CAPACITY-1)            /*バッファに空きがあることを確認する*/
    { buf.place[(buf.head+(++buf.last))%CAPACITY]=data; }
先頭のデータ要素を取り出す：
    if (buf.last>=0)                    /*データ要素が入っていることを確認する*/
    { data=buf.place[buf.head]; --buf.last; buf.head=(++buf.head)%CAPACITY; }
```

図2　リングバッファを取り扱う一般的なプログラミング方法

図２のようなプログラミングは、めんどうであり、ミスを引き起こしがちである。

抽象データ型の概念を導入すれば、次のことが達成される。

①抽象データ型を別の所で定義しておくことにより、利用者プログラムで抽象データを簡単に宣言できる。

②抽象データに対する操作のアルゴリズムを別の所で定義しておくことにより、利用者プログラムでは、「データ要素を入れる」、「データ要素を取り出す」などの本質的な意味を表現する記述で簡単に操作できる。

Ｃ言語でこれを実現するためには、次章３で説明するプリプロセッサを利用する方法がある。

３．Ｃ言語のプリプロセッサ

プリプロセッサとは、あらかじめマクロとして定義されたテキスト（任意のプログラムコード）を、そのマクロ名が書かれたソースプログラムの中に、コンパイル前に展開するものである。機種やプログラム仕様によって変更されうる定数値や、定まった意味を持つ定数値をマクロ定義するという活用がよく行われている。

また、展開されるテキストの一部をマクロパラメタで指定するようにすることもできる。パラメタ付きのマクロは、主として、式をインライン展開するために使われている。

４．マクロによる抽象データ型の実現方法

パラメタ付きのマクロは、抽象データの宣言文や、抽象データに対する操作の式を展開するために使うことができる。

（１）抽象データの宣言のマクロ化

リングバッファの宣言（図２）は、図３のようにマクロ化できる。こうすれば、マクロパラメタIdで指定するデータ名を変えながら、同じ型の抽象データをいくつでも簡単に宣言できる。また、データ要素の型やバッファの容量も、マクロパラメタで抽象データごとに指定できる。

マクロ定義：

```
#define QUEUE(Id,Type,Capacity)                                    ¥
    static struct { int _head;                                     ¥
                    int _last;                                     ¥
                    int _limit;                                    ¥
                    Type _place[Capacity]; } Id = {0,-1,Capacity}
```

マクロ引用：

```
QUEUE(buf,char,1000);
```

展開結果：

```
static struct { int _head;
                int _last;
                int _limit;
                char _place[1000]; } buf = {0,-1,1000};
```

（注）マクロ定義の行末の「¥」は、マクロ定義が次行へ継続することを示す。
下線は、マクロパラメタによって置き換えられたテキスト部分を示す。

図３　リングバッファの宣言のマクロ化

（2）抽象データに対する操作のマクロ化

リングバッファに対する「データ要素を入れる」、「先頭のデータ要素を取り出す」という操作（図2）は、それぞれ図4、図5のようにマクロ化できる。こうすれば、利用者プログラムでは、関数の呼び出しと同じ記述形式で操作を簡単にプログラミングできる。どの抽象データに対する操作であるかは、マクロパラメタIdで指定する。

展開される式がもたらす型は、マクロパラメタIdで指定された抽象データのデータ要素の型（その抽象データの宣言でマクロパラメタTypeとして指定された型）に適合する。すなわち、抽象データに対する操作には、型が固定される関数呼び出しよりも、式をインライン展開するマクロの方が適している。

```
マクロ定義：
    #define ENQUEUE(Id)                                          ¥
        ( ((Id)._last<(Id)._limit-1)?                            ¥
            (&(Id)._place[((Id)._head+(++(Id)._last))%(Id)._limit]) :  ¥
            (NULL) )
マクロ引用：
    *ENQUEUE(buf) = data;
展開結果：
    *( ((buf)._last<(buf)._limit-1)?
         (&(buf)._place[((buf)._head+(++(buf)._last))%(buf)._limit]) :
         (NULL) ) = data;
```

図4　リングバッファにデータ要素を入れる操作のマクロ化

```
マクロ定義：
    #define DEQUEUE(Id)                                          ¥
        ( ((Id)._last>=0)?                                       ¥
            ( --(Id)._last,                                      ¥
              (Id)._head=(++(Id)._head)%(Id)._limit,             ¥
              &(Id)._place[((Id)._head-1+(Id)._limit)%(Id)._limit] ) :  ¥
            (NULL) )
マクロ引用：
    data = *DEQUEUE(buf);
展開結果：
    data = *( ((buf)._last>=0)?
                ( --(buf)._last,
                  (buf)._head=(++(buf)._head)%(buf)._limit,
                  &(buf)._place[((buf)._head-1+(buf)._limit)%(buf)._limit] ) :
                (NULL) );
```

図5　リングバッファから先頭のデータ要素を取り出す操作のマクロ化

5．抽象データ型マクロの設計の観点

抽象データ型を設計するには、次の観点から考える。

①データ型を利用者にどのような形に見せかける（抽象化する）か。

②抽象データに対するどのような操作を利用者に提供するか。

これに基づいて、宣言用マクロと操作用マクロを作成する。その際、次の方針を定める。

（1）用途別の宣言用マクロ

次の3種の宣言用マクロを抽象データ型ごとに用意する。

①スタティック宣言：「static」と初期値設定を含む宣言文を展開する。

②外部定義宣言（マクロ名：E_…）：「static」を除いた形の宣言文を展開する。

③ダミー宣言（マクロ名：D_…）：「static」と初期値設定を除いた形の宣言文を展開する。

これにより、図6に示す方法で、抽象データの受け渡しもできるようになる。

```
外部定義宣言：
  E_QUEUE(buf,char,1000);
外部参照宣言：
  extern D_QUEUE(buf,char,1000);
関数呼び出し：
  func(&buf)
関数定義：
  func(buf) D_QUEUE(*buf,char,1000);
```

図6　抽象データの受け渡し方法

（2）データ要素へのアクセスの仕様

データ要素にアクセスする操作用マクロは、データ要素の値ではなく、データ要素を指すポインタ値をもたらすように統一する。これにより、データ要素の設定と参照の両方に使える操作用マクロも作れるようになる。また、アクセス誤りという結果をＮＵＬＬポインタ値によって識別できるようになる。

6．抽象データ型マクロライブラリの作成

前章5で述べた観点に基づいて、以下の二つの抽象データ型をサポートするマクロライブラリ「ＡＬＩＢ」（Abstract Data Types Macro Library）を作成した。

（1）リングバッファ

配列をリングバッファに見せかけたデータ型である。配列の添え字の計算が利用者プログラムから隠ぺいされる。

最新のデータ要素を取り出す操作も提供しているので、スタック（先入れ後出しバッファ）としても使える。

リングバッファをサポートするマクロの仕様を表1に示す。

（2）リンクトリスト

データ要素をメモリ域に動的に割り付け、データ要素間をポインタでリンクする形のデータ型である。リンク用ポインタの操作が利用者プログラムから隠ぺいされる。

各データ要素から、2方向（一次方向および二次方向）にリンクを伸ばすことができる（図7）。これにより、線形リスト、二次元のテーブル、二分岐木構造のいずれでも作ることができる。各データ要素には、一次方向、二次方向、および逆方向へのリンク用ポインタが付随する。

リンクトリストをサポートするマクロの仕様を表2に示す。

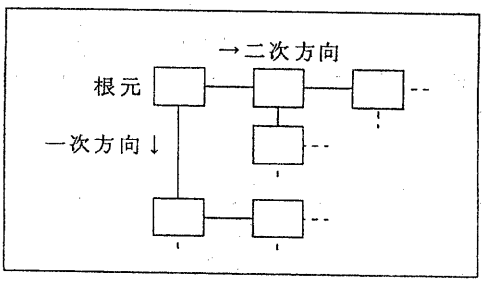

図7　リンクトリストの形状

表1　リングバッファ用マクロの仕様

宣言用マクロ	用途	宣言の形式
QUEUE(Id,Type,Capacity);	スタティック宣言	「static」付き、初期値設定付き
E_QUEUE(Id,Type,Capacity);	外部定義宣言	「static」なし、初期値設定付き
D_QUEUE(Id,Type,Capacity);	ダミー宣言	「static」なし、初期値設定なし
操作用マクロ	**機能**	
ENQUEUE(Id)	データ要素を入れる場所を決めて、その場所を指す。	
DEQUEUE(Id)	先頭のデータ要素を取り出して、それを指す。	
RECALL(Id)	最新のデータ要素を取り出して、それを指す。	
NEW(Id)	最新のデータ要素を指す。	
PEEP(Id,Order)	Order で指定された番号のデータ要素を指す。	
LAST(Id)	最新のデータ要素の番号を示す。	
OCCUPANCIES(Id)	入っているデータ要素の数を示す。	
VACANCIES(Id)	空き容量を示す。	

表2　リンクトリスト用マクロの仕様

宣言用マクロ	用途	宣言の形式
LIST(Id,Type,Frame);	スタティック宣言	「static」付き、初期値設定付き
E_LIST(Id,Type,Frame);	外部定義宣言	「static」なし、初期値設定付き
D_LIST(Id,Type,Frame);	ダミー宣言	「static」なし、初期値設定なし
操作用マクロ	**機能**	
APPEND_PRIMARY(Id)	データ要素を割り付けて、一次方向にリンクする。	
APPEND_SECONDARY(Id)	データ要素を割り付けて、二次方向にリンクする。	
INSERT(Id)	データ要素を割り付けて、逆方向にリンクする。	
REMOVE(Id)	着目中のデータ要素を除去する。	
CURRENT(Id)	着目中のデータ要素を指す。	
PRIMARY(Id)	リンクを一次方向にたどる。	
SECONDARY(Id)	リンクを二次方向にたどる。	
BACKWARD(Id)	リンクを逆方向にたどる。	
RESTORE(Id)	着目点をリンクの根元に戻す。	
IS_PRIMARY(Id)	一次方向にリンクされたデータ要素かどうかを示す。	
IS_SECONDARY(Id)	二次方向にリンクされたデータ要素かどうかを示す。	
CONTENTS(Id)	割り付けられているデータ要素の数を示す。	

7．マクロライブラリ作成上の工夫点

前章6で説明したマクロライブラリを作成するにあたって、次の工夫を行った。

（1）マクロパラメタ名

マクロパラメタ名は、先頭が大文字、以降が小文字である名前とした（例：Id）。これにより、マクロ定義を見るときに、テキストの置き換えが起こる部分を容易に見分けることができるようになる。

（2）内部データ名

抽象データ型の内部データ名は、すべて下線（_）で始まる名前とした（例：_place）。これにより、利用者がふつうに用いる名前とまぎれにくくなるので、「利用者プログラム

で内部データに直接アクセスしない」という規範が守られているかどうかの検証が容易になる。

（３）メモリ割り付け単位の型名

リンクトリストを宣言するマクロは、図８に示すように定義されている。

マクロパラメタFrame は、リンク用ポインタを含むメモリ割り付け単位に与える型名である。これは、リンクトリストを複数宣言したときに「型定義の重複」という文法エラーにならないようにするために、その型名を利用者に与えてもらうようにしたものである。

抽象データ型の実現のための“都合”を利用者プログラムに及ぼすのは望ましくないことであるが、言語の文法に適合させるためにはやむをえない方策である。

実際のところ、利用者にとっては、重複しない適当な名前を指定すればよいだけのことなので、不便さを生じることはない。

```
#define LIST(Id,Type,Frame)                                          ¥
    static struct { struct Frame { struct Frame *_backward,          ¥
                                          *_forward[2];              ¥
                                 Type _cell; } *_root,               ¥
                                              *_current;             ¥
                int _contents; } Id = {NULL,NULL,0}
```

図８　リンクトリストの宣言のマクロ定義

（４）補助関数

リンクトリストでデータ要素を割り付けるときと除去するとき、リンク用ポインタの張り替えが必要である。この処理は、一つの式で書くには複雑すぎる。そこで、割り付けと除去の際に必要な処理を補助関数として分離した。

補助関数の呼び出しは、マクロで隠ぺいされている。

８．マクロによる抽象データの実現の制約

Ｃ言語のマクロで抽象データ型を実現するうえでは、次のような制約がある。

（１）抽象データの配列宣言ができない

これは、抽象データ配列のすべての要素に初期値を設定する宣言文をマクロ定義することが、Ｃ言語の文法上できないからである。

この問題の対処策としては、図９に示すように、抽象データの実体とそれを指すポインタ配列を宣言する方法がある（ただし、コンパイル時に「初期値の型が合わない」という警告が出る）。

```
QUEUE(buf_1,char,1000);                                  /*第１のバッファの宣言*/
QUEUE(buf_2,char,1000);                                  /*第２のバッファの宣言*/
static D_QUEUE(*buf[2],char,1000) = {&buf_1,&buf_2};  /*ポインタ配列の宣言*/
*ENQUEUE(*buf[0]) = data;                        /*第１のバッファにデータ要素を入れる*/
*ENQUEUE(*buf[1]) = data;                        /*第２のバッファにデータ要素を入れる*/
```

図９　抽象データを配列として取り扱うための代替手段

（２）繰り返しを要する操作は実現しにくい

これは、Ｃ言語では繰り返しを一つの式で書けないからである。たとえばリンクトリストでリンク上をいっきに複数回移動する操作は、式のインライン展開では実現できない。繰り返しを実現するには、補助関数を作る必要がある。

しかし、補助関数を用いてまで繰り返しを実現するよりも、繰り返しによる定型的なプログラミングテクニックをマニュアルで示す方が得策である。利用者プログラムで繰り返しを記述することは容易にできるからである。

（３）実行効率は犠牲になる

これの主な要因は、データ要素にアクセスする操作用マクロで、誤ったアクセスをしないためのチェックをしていることである。チェックを省略するプログラミングテクニックで実行効率を上げることは、このマクロライブラリを使う限りはできない。

これは、プログラムの生産性、信頼性を高めることの代償と考えるべきである。

９．適用実績

このマクロライブラリは、これまでに、ＵＮＩＸワークステーションで動作する実用プログラム（設計ドキュメント生成プログラム、ネットワークルーティング情報生成プログラム[2]、データベースサービスプログラム）に適用されている。そのほか、パソコンのプログラムにも利用されている。

これまでの適用では、いずれも、二次元のテーブルを取り扱うためにリンクトリスト用マクロが活用されている。これにより、利用者は、テーブルの大きさの制限と煩雑なポインタ操作の両方から解放される。これを利用するプログラミングテクニックの習得にやや時間がかかることはあるが、習得した人には「非常に便利」と重宝がられている。

一方、リングバッファ用マクロはほとんど利用されていない。これは、キューやスタックは線形リスト（一方向のリンクトリスト）でも実現でき、その方が容量に制限がない点で便利だからである。しかし、リングバッファ用マクロは、通信プログラムには有効であろう。メモリ域を提供するＯＳのない通信制御装置のプログラムにも利用でき、また、データ要素の出し入れをリンクトリストよりも高速に処理できるからである。

１０．おわりに

抽象データ型の概念が発表され、その利点が説かれて久しい。しかし、実用プログラムの開発を担う多くのプログラマがそれを活用するには至っていない。新しい方法論を開発現場に浸透させるには、開発現場での従来の方法論をベースとして発展できるような方策が重要である。

本稿で紹介したＣ言語用抽象データ型マクロ部品は、既存の言語機能を利用して抽象データ型の方法論を実践できるための道具である。それが開発現場のプログラマに受け入れられ、実用的であることは、いくつかの実用プログラムへの適用で実証された。

謝辞

このマクロライブラリを実用プログラムに適用し、貴重なご意見をくださった、ＮＴＴ通信ソフトウェア本部の石原勝彦氏、大崎等氏、宇佐美康氏、ＮＴＴ情報システム本部の小川正彦氏に感謝の意を表する。

参考文献

[1] B. H. Liskov, S. Z. Zilles: "Programming with abstract data types", SIGPLAN Notices Vol.9, No.4, pp.50--59 (1974)

[2] 浅見、田中、福山「ルーティング情報の自動生成方式」１９９２年電子情報通信学会春季大会 B-660