クラスモジュールの一構成法

恐神正博*・西田富士夫*

Construction of Class Modules

Masahiro Osogami and Fujio Nishida

Recently object oriented programming techniques have been presented and used in various fields. When constructing or revising class methods, however, it seems that these techniques cannot be used to their full potential by using customized library function modules or other class modules, therefore object oriented programming techniques tend to be constructed in their own class worlds. This paper presents an approach to constructing new class methods by using library generic functions which have head comment sentences written in Japanese.

1. まえがき

１９８０年代の中頃からオブジェクト指向の設計法やプログラミングの手法が提唱され実用化が進んでいる．これは，従来人手により構築されてきたシステムを機械化するには，システムのオブジェクトモデルを設計制作のあらゆる段階で直接的にデータ処理システムに書き換えて具体化するのが簡明で効率的であるとの考えによる．設計制作の上流過程における要求定義や概略設計の段階では，この考え方は当然であるが，設計段階においては個々のオブジェクトやクラスにとらわれない汎用的な手法を適用する方が得策である．すなわち，諸科学における数学的手法の適用や製造業における半製品的機械部品材料のカスタマイズと同じく，汎用的なモジュールを作っておき，これを目的のオブジェクトに合わせてカスタマイズして製作する方が全体的にみて効率的であろう．一般に汎用的な部品モジュールを用いて，カスタマイズシステムを構築する方法はソフトウェア工学の分野ではまだ十分な実用化に至っておらず，オブジェクト指向の名の下にプログラミングは各クラスごとに行われることも多いようである．以下，本稿では，筆者らが用いてきたプログラム作成の自動化の手法に基づいてオブジェクト指向プログラミングにおけるクラスモジュールの作成を自動化する一つの手法を述べる．

2．主な問題点と解決方法の概要

前節で述べたオブジェクト指向言語，例えばC++における設計やコード化の問題点を具体的に述べれば，クラスのメンバー関数の内容がその呼び出し表現の引数部などに表現されていないことである．(1) はC++における関数の宣言の形を示す．

ft　fs(tv*1*,...,tv*n*);　(1)

ここにftは関数の戻り値のタイプ，fsは関数記号，tv*i*はクラスのデータメンバ以外の*i*番目の仮引数のタイプと名前の対を表す．他方 (2) はデータメンバを仮引数td*1*,…,td*m*で

---

*経営工学科

おきかえた関数の宣言の形を示す.

ft　　fs(td*1*,...,td*m*,tv*1*,...,tv*n*);　　(2)

ただし，引数の順序はライブラリで慣用的に用いられている順序に従うが，ここでは簡単のためデータメンバに対する仮引数と，それ以外のものとに分けて表している.

(1) と (2) の形式は同じであるが，プログラム設計上での意義は大きく異なっている. すなわち (1) の形式では，関数定義の本体に書かれている詳細記述やコメント文を見ない限り，(1) の関数呼び出し形がどのような処理を呼び出しているのかわからない. これはこの関数において引数の主要部であるデータメンバが脱落しているからであり，どのようなデータメンバがどういう役割で処理に関係するのか分からないためである. (2) の形なら多くの場合ライブラリなどに収納された汎用的関数であるので，引数の名前と位置により，どのような処理がなされるのか，少なくともコメント文を参照すれば理解することができる. というのも，ライブラリはユーザ数に大小があるにせよ，共通に使用され解釈されるという普遍性をもつからである. ところが (1) の場合，たとえコメント文をつけるとしても，主要引数を脱落させたため，その適用範囲はある特定のクラスに限定されてしまう. 3節以降に述べるように，この関数呼び出しコード表現に到るトップダウン的な設計文書が表示されていれば事情は異なるが，オブジェクト概略設計とプログラム設計との距離は現状では大きくかけ離れている.

そこで，C++ では関数の定義本体のプログラムコードに直接，データ変数を取り込んでいる. データメンバは既に宣言した変数であり，小さいプログラムであれば，ユーザにとってもこの方が理解しやすい. しかし階層化して下位の処理構造が上位に現れないように隠して形式化することがプログラムの簡明性と再利用の見地からより重要であり，これまで何度も提唱されてきた所である.

また，(1) の関数の定義本体のデータ変数を仮引数に置き換え，(2) の形の関数呼び出し形をとることにすればライブラリの汎用的関数を利用することができる. ライブラリモジュールの利用は，効率性や信頼性の向上といった実用性の他に，ソフトウェア科学や工学の学問体系の上からも重要である.

C++ は現在オブジェクト指向の代表的な言語として広く利用されているが，他の言語と同様に以上のような短所をもっている. ここではこの欠点を補い，ライブラリの C 言語で書かれた汎用関数を利用してクラスのメンバ関数を作成する簡単な手法を述べる.

一つ目の方法はデータメンバを d*1*,⋯,d*m* とするとき (1) を

```
fs(tv1,⋯,tvn){
    fs(d1,⋯,dm,tv1,⋯,tvn);
}
```　　(3)

のように (2) を用いて定義し変数の二重定義を避ける方法である. (2) はライブラリの汎用的関数に属し，その関数定義は仮引数 td*1*,⋯,td*m*,tv*1*,⋯,tv*n* を用いて表されるものとする. (3) の形は二重に同じ様な表現を書くという煩わしさはあるが，直接 fs(d*1*,⋯,d*m*,tv*1*,⋯,tv*n*) の定義本体のプログラムコードを書くよりも優れている.

二つ目の方法は，利用したいライブラリの汎用関数の呼び出し表現 (2) からクラスのデータメンバに対応する仮引数を除去して (1) の関数呼び出し形を作る方法である. また，その関数定義本体にはクラスのデータメンバに対応する仮引数をデータメンバに置き換えて作成すればよい. 作成したメンバ関数は現用の C++ 言語の形式と同じとなる.

3． MAPPにおけるクラスの構築の概要

前節では，Cのライブラリモジュールを出発点として与え，これを利用してC++のクラスのメンバ関数を作成する手法について述べた．しかしC言語のライブラリモジュールは現在のところ日本語文との相互変換ツールや引数指定の機能変換ツールなどを補充しないと検索やカスタマイズが不便なように思われる．

この節では記号列の処理などがより容易なPROLOGで作成したMAPPの手法を用いて，クラスの概略設計からライブラリを用いて，クラスのメンバ関数のコードなどを自動的に生成する手法の概略を述べる．概略設計においては，クラスは互いに関係の深い主な処理対象を集めてデータメンバを作り，また，このデータメンバのひとまとまりをいくつかのメンバー関数単位の処理に分ける．

概略設計で選定した，データメンバのデータベースへの入力には問題となる点はない．例えば，データ名とタイプ名を組にしてリスト形式で一括して次のようなPROLOGの規則節により読み込み，述語data_memberの引数部に記憶する．ただし，クラス名は前もって述語class_nameの引数部に指定してあるものとする．

```
data_member_input :-
   write(' メンバの名前，タイプを次の形で入力してください． '),nl,
   write('[[NAME1,TYPE1],…,[NAMEn,TYPEn]]'),nl,
   read(DATA_MEMBER),class_name(CLASS_NAME),
   assert(data_member(CLASS_NAME,DATA_MEMBER)).                    (4)
```

このようにして作成したdata_member述語の例を次式に示す．

```
data_member(lend,[[doc*,ptr],[int, size]]).                        (5)
```

ここにlendはある図書館の貸し出し処理システムプログラムのクラス名であり，続いてこのクラスで処理されるデータのタイプと名前の対が示される．
doc*はこのシステムで定義している図書の構造体配列のタグ名でタイプを表し，ptrはこれを指すポインタ変数の名前で構造体配列を間接的に表す．また[int,size]はこの配列の大きさsizeが整数型であることを表す．

メンバー関数は概略設計において割り当てた機能の概略を参照し，ライブラリから適当な汎用的関数を選んでクラス用にカスタマイズして作る．ライブラリの汎用的関数は，検索しやすいように処理の種別やデータ構造別などに分けている．その見出し文は，処理内容が分かりやすいように処理に関係する主要なデータ名を変数名として含む日本語文表現とする．見出し文は関数の呼び出し表現の引数に格助詞を補ってその役割を明らかにする他，処理情報を補ってその関数の機能の内容を簡明に示すものである．(6)に構造体処理関数の見出しを表示する事実節形式の見出し辞書の例を示す．

```
dict([[head( 構造体 ,struct),
  body(
  [p(1,[A,N],[ 大きさ ,N, の構造体配列 ,A, に kb からデータを読み込む ]),
   p(2,[A,N],[ 大きさ ,N, の構造体配列 ,A, を表示する ]),
        ·····])   ]]).                                             (6)
```

辞書は処理分野を示すhead部とその分野の処理関数の見出し部を集めたbody部からなる．body部の各見出しは先頭の記号pに続きその引数部に見出し番号，ユーザが指定するカスタマイズ変数，見出し文からなる．

ユーザはheadの引数部で検索希望の処理の分野を指定し，bodyの引数部すべての見出しをディスプレイに表示させる．続いてメニュー方式により，希望の処理の見出しを番号の入力により表示させ，カスタマイズ変数に対し適用分野で分かりやすい仮引数名やデータメンバ名を入力する．プログラム生成自動化システムMAPPは，これらの入力データを用いて，見出し文をカスタマイズした後，対応する見出し式を(7)のような変換辞書を参照して見出し式を生成する．

```
conv_dir([jp([ 大きさ ,N, の構造体配列 ,A, に kb からデータを読み込む ],
        [ read_tbl,[A,N]] ),
        proto([void,read_tbl,[[A,A],[int,N]] ]),
        funct(read_tbl(A,N)) ]).                                    (7a)

conv_dir([jp([ 大きさ ,N, の構造体配列 ,A, の中で条件 ,COND, を満たす
                構造体を表示し構造体配列 ,B, に記憶する ],
      retr_str_ar([A,N,COND])]),
      proto([void,retr_str_ar,[[A,A],[int,N],[B,B]] ]),
      funct(retr_str_ar(A,B,N)) ]).                                 (7b)
```

ここに，変換辞書のjp項の引数には見出し文と見出し式の組を収め，proto項にはプログラムにおける関数宣言の形，funct項には関数の呼び出し表現をそれぞれ収納する．このようにして作成された関数の見出し式を積み重ねてクラスのメンバ関数を作成する．

次に図書貸し出しシステムを例にとり，クラスlendにおいて(5)のデータメンバを処理するメンバ関数（メソッド）の集まりを(8)に示す．

```
method_list(lend,[
  [lend,[doc,size]],
  [read_tbl,[doc,n]],
  [fscan_tbl,[fn,doc,n]],
  [print_tbl,[doc,n]],
  [fprint_tbl,[doc,n,doc_fn1]],
  [sort_struct_array,[doc,n,doc_code]],
  [retr,[doc,n,retr_cond,retr_procd]].
  [update,[doc,n,cond,update1]] ]).                                 (8)
```

(5)のデータメンバと(8)のメンバ関数の見出し式から，ライブラリの汎用的関数の情報を参照することによりデータメンバを処理関数でどのように処理するかが明らかになる．これによりクラスの設計が確定し，クラスのコードを生成することができる．

設計表現からプログラム表現に変換するときには，２節で述べたようにデータメンバを表す変数が二重定義にならないようにを留意することが必要がある．クラスの構造は(5)，(8)の設計により明らかであるので，メンバ関数の形式は(1)の形，すなわち，C++で用

いられている形をとる.

method_list の引数部から各メンバ関数の見出し式を抽出し，(7) の述語辞書を用いて [関数記号，引数リスト] を検索し，クラス名とともに述語 gen_proto の引数部に与える．これからメンバ関数の宣言部を作る.

```
gen_proto(CLASS_NAME,[FS,NAME_LIST]) :-                          ①
    conv_dir(S),member(jp(J,[FS,NAME_LIST]),S),                  ②
    member(proto([FT,FS,ARG_LIST],S),                            ③
    writelist([FT,' ',FS, '('] ),                                ④
    proto_arg_list(CLASS_NAME,ARG_LIST),write(');'),nl.          ⑤
                                                                 (9)
```

上式は見出し式からこれを含むメンバ関数の proto_type 項を conv_dir 辞書から求め，関数頭部の戻り値の型 FT と関数記号 FS を求める．引数部は (10) により作られる.

```
proto_arg_list(CLASS_NAME,[[HT,HN]|T]) :-                       ①
     data_member(CLASS_NAME,DATA_MEMBER)                         ②
     ((member([HT,HN],DATA_MEMBER);
      member([HT,'*i_ptr'],DATA_MEMBER));                        ③
     (prec(0),write(HT),ra(prec(0),prec(1));write(','),write(HT))),  ④
     proto_arg_list(CLASS_NAME,T).                               ⑤   (10a)
proto_arg_list(CLASS_NAME,[]).                                       (10b)
```

①の引数部にはクラス名とチェックすべき引数が与えられる．MAPP は②のデータメンバを参照して③により引数がデータメンバのメンバーであると，その引数を無視して⑤により次の引数の処理に進む．一方，その引数がデータメンバでなければ，④によりその引数のタイプ部を書き出す．prec(0) は引数がその前にないことを表し prec(1) は引数が前にあることを表す．また，ra(pec(0),prec(1)) は prec(0) をデータベースから消し prec(1) をデータベースに書き出すことを表す．④は何個かの引数 (タイプ) をその間にコンマを挿入して書き出すことを表す.

プログラムコードにおける関数の呼び出し表現とその関数本体は (8) の関数の見出し式とその関数定義に基づいて作る．プログラムコード生成のとき，関数定義の本体部は (8) の関数の見出し式の引数に従ってデータメンバ名と仮引数名を用いて具象化される.

関数見出し部は，(10) の関数宣言部と同様な方法により，(8) のような見出し式からデータメンバとして宣言されている引数を自動的に除去して作られる.

4．汎用的ライブラリ関数の構築

関数の中には処理がよく似ているものがあり，これらは汎用的関数としてライブラリなどにまとめておき使用すると便利である．汎用的関数と類似なものにサブルーチンなどがあり，従来から主に引数を仲介として広く用いられてきている．C++ では近年変数のタイプによらない汎用的関数を定義したり利用できるようになった．この他，筆者らの MAPP の手法により構造体配列（通常の１次元配列や２次元配列を含む．）や，クラスのデータメンバなどを具体的に指定すれば，それを入出力したり，ソートした結果や，それらの平均値，最大値，最小値などを求める関数を自動的に生成することができる．３節で

は(6)における見出し文に，データメンバを指定して(8)の関数見出し式や宣言部を自動的に作成する手法を概説した．構造体メンバの名前やタイプなどを指定することにより，その関数本体部も自動的に作成することができる．本節ではその原理の概要を構造体データの読み込み関数の例について概説する．

```
funct_body(CLASS_NAME,[read_tbl,[NAME,SIZE]]) :-                    ①
include_list_a(CLASS_NAME,[iostream.h',iomanip.h'],                  ②
begin_body,c(for(NAME,SIZE,read_struct(NAME))),end_body.             ③     (11)
```

(11)はクラス名がCLASS_NAMEのメンバ関数read_tblの本体が②③からなっていることを示す．関数read_tblは大きさがSIZEで名前がNAMEの構造体配列にキーボードからデータを読み込むことを表す．②は関数read_tblに必要なヘッダファイルを一括して記憶するために述語の引数部に取り込み、③はread_tbl処理を繰り返して行うことを示す．

```
c(for(NAME,SIZE,P) :- write_sp,                                      ①
  writelist(['for(', NAME,'*dp=dpt ;','dp<dpt+',SIZE,';dp++){' ]),nl, add_sp,   ②
  proc_list(P),sub_sp,write_sp,write('}'),nl.                        ③     (12)
```

(12)は大きさがSIZEで名前がNAMEの構造体配列に処理Pを施すために，②③のように配列NAMEを指すポインタ変数dpを用いたfor文を出力することを示す．add_sp,sub_spはそれぞれスペースカウンタを1個増減することを表し，write_spはそのときのスペースカウンタが示す個数のスペースを出力しインデントすることを示す．

(11)のread_structは (12)のproc_listの引数に書き換えられ，(13)のc(read_sruct(X))として定義される．

```
c(read_struct(X)) :- obj([name(X),type(struct(TAG))]),               ①
 struct_member(CLASS_NAME,TAG,L),c(read_mem_list(X,L)).              ②     (13)
```

(13)はXがタグ名TAGの構造体であり，Lがその構造体メンバのリストなら，①の頭部を実行するには②の右端の述語(14)を実行すればよいことを示す．

```
c(read_member_list(X,[[HN,HT]|T])) :-                                ①
c(read_mem(struct(dp),[HN,HT])),c(read_mem_list(X,T).                ②     (14)
```

ここにHN,HTは構造体の先頭メンバの名前とタイプである．②から

```
c(read_mem(struct(dp),[HN,HT])) :-
  swritelist(['cout << "dp->',HN,'=in ',HT,'"; ']),nl,
  swritelist(['cin >> dp->',HN,';']),nl.                             (15)
```

によりコード化する．上述の手法はレベル1の任意の構造体メンバのデータ入力プログラムの生成に適用できることが分かる．出力例を5節のread_tbl()の関数定義に示す．この手法はメンバの中に構造体メンバを含む構造体の入力処理に拡張することができる．

処理の中には大まかな処理は同じであっても細部がいろいろ異なり，汎用的関数として定義することが困難なものがある．例えば，検索処理や更新処理の場合である．このような場合には概略の処理の汎用的関数を作り，ユーザはこれを用いて処理の骨組みを作り，

詳細部はひとまとまり毎に一括して処理名として残し，処理名毎に詳細化する方法が考えられる．更新処理を例にとると見出し文として(16)を準備する．

[大きさ ,N, の構造体配列 ,A, から条件 ,COND, を満たす構造体に対し更新処理 ,
UDATE, をする]. (16)

以下，図書館台帳の貸し出し更新処理に例をとる．ユーザが借り出しまたは返却する図書コード req_code が，台帳の図書コード doc_code に一致する検索条件を cond として(17)により検索条件 cond を詳細化する．

```
cond :- cond_c(cond,'dp->code==req_doc').
```
(17)

ここに (17) の右辺の本体部において，第２引数の’’で挟まれた部分は，プログラム中の cond を置き換える C のコード部分である．

このときの更新処理は，貸し出しの場合には貸し出し中のマークを１とし，ユーザコード，返却予定日とともに更新する．また返却の場合には，これらの値をすべて０に更新するものとする．

これらひとまりの処理を update1 として， if_else 見出し式を用い cond(lend_or_retn), procd(lend),procd(retn) に分け，同様に詳細化を進めることができる． (18) における出力例を５節の update() に示す．

```
update1 :- c(if_else(cond(lend_or_retn),procd(lend),procd(retn))).
cond(lend_or_retn) :- cond_c(lend_or_retn,'lend_retn==1');
        .....
```
(18)

これらは，状況に応じてライブラリ関数を利用したり， C 言語などで直接にコード化する．

５．図書貸し出しシステムの例

実験例として簡単な図書貸し出しシステムを半自動的に構築し，所期の結果を得た．

システムは lend class, usr class, time class の３クラスからなる． lend class は貸し出し台帳に資料の貸し出し状況を記録する．資料の構造体メンバはコード番号，貸し出し中かどうか，返却予定日，利用者コードなどである． user class は利用者のコード，身元，借りている資料のコードと冊数などを処理する． time class は貸出日から返却予定日などを計算し出力する．

メイン部では，貸し出しか返却かの別，資料番号，利用者番号などの利用者からの入力により貸し出し台帳の処理を行う．貸し出しの場合， usr class の借り出し冊数を調べて，ub 冊以内であれば貸し出すこととする．

さて３個のクラスのうち， lend class と usr class はともに構造体配列を処理するクラスでその構造はよく似ており，構造体メンバの入出力，ソート，検索，更新などの処理を行う．，これらは３節４節に述べた手法を用いて， (5), (8) のようなクラスメンバと 構造体メンバから自動的に次のようなクラスコードに展開される．以下は lend class code の例でその主要なメンバ関数を示す．

```
struct doc{
    int code;
    int lend_yn;
    long retn_date;
    long u_code;
};
class lend{
    doc* dpt;
    int n;
    public:
        lend(int);
        void read_tbl();
        void print_tbl();
        void fprint_tbl(char*);
        void fscan_tbl(char*);
        long  retr(int);
        void update(int,int,long,long);
};
lend :: lend(int size){
        n=size;
        dpt= new doc[n];
}
void lend :: read_tbl(){
        for(doc* dp=dpt;dp<dpt+n; dp++){
            cout << " dp->code?= ";
            cin >> dp->code;
            cout << "dp->lend_yn?= ";
            cin >> dp->lend_yn;
            cout << "dp->retn_date?= ";
            cin >> dp->retn_date;
            cout << "dp->u_code ?= ";
            cin >> dp->u_code ;
        }
}
void lend :: print_tbl(){
    for(doc* dp=dpt;dp<dpt+n; dp++){
        cout << setw(6) << dp->code <<
        setw(6) << dp->lend_yn <<
        setw(8) << dp->retn_date <<
        setw(8) << dp->u_code << "¥n" ;
    }
}
void lend :: fscan_tbl(char *fn){
    ifstream fpr (fn);
    for(doc* dp=dpt;dp<dpt+n;dp++){
        fpr  >>  dp->code  >>
        dp->lend_yn >> dp->retn_date >>
        dp->u_code ;
    }
    fpr.close();
}
void lend :: update(int lend_retn,
int req_doc,long requester,long retn_date){
    for(doc* dp=dpt;dp<dpt+n;dp++){
        if(dp->code==req_doc){
            if(lend_retn==1){
                dp->lend_yn=1;
                dp->u_code=requester;
                dp->retn_date=retn_date;
            } else {
                dp->lend_yn=0;
                dp->u_code=0;
                dp->retn_date=0;
            }
        }
    }
}
```

6．あとがき

C++言語などのオブジェクト指向言語はいろいろな長所をもつ言語であるが，メソッドの作成などにおいてタイプ処理を除き，汎用的モジュールを作成したり利用するという方向が損なわれている見解を述べ，簡単な対応策を提案した．続いて本格的な対応策として，見出し文から適用可能な汎用的関数を検索しカスタマイズし，クラスモジュールを半自動的に構築する一つの手法を述べた．処理の概要は似ているが，詳細はいろいろな点で異なる関数群の汎用化は重要であるが困難な問題である．本文に一つのアプローチを述べたが，整備と拡張が必要である．

参考文献

1) B. ｽﾄﾗｳｽﾄﾗｯﾌﾟ著，斉藤信男他訳：プログラミング言語C++，凸版（株）

2) 恐神正博，西田富士夫：構造体汎用モジュールの作成，福井工業大学研究紀要 第26号，pp.311 ～ 318, (1996)

（平成9年12月5日受理）