2E−2

複合オブジェクトのためのデータ構造の考察

宇田川佳久（三菱電機㈱）

1．はじめに

オブジェクト指向データベース（ＯＯＤＢ）と従来のデータベースとの顕著な違いの一つに、ＯＯＤＢがオブジェクト識別子を含む複雑なデータ構造を扱うことができることが上げられる。一方、多くのＯＯＤＢはプログラミング言語Ｃ＋＋を親言語とし、Ｃ＋＋のデータ構造を永続するデータとして管理する機能を提供している。

本文では、Ｃ＋＋がポインタを含むデータ構造を簡潔に表現できること、データベースで必須となる不定個のデータを扱えること、さらに、データ構造を組み合わせてより複雑なデータ構造を構成することができることを確認したことを報告する。

2．データベースとデータ構造

データベースは、データの共同利用を目的としている。したがって、管理するデータの個数をあらかじめ特定することができず、記憶容量が許す限りのデータを蓄えられるものでなければならない。また、多種類のアプリケーションに対応しなければならないから、データ構造も特定することはできず、基本データ型を組み合わせアプリケーションに適したデータ構造を定義できるものでなければならない。以上の考察より、データベースのためのデータ構造の条件として、少なくとも

(1) 特定できない個数のデータを管理できること
(2) データ構造を組み合わせて、より複雑なデータ構造を構成することができること

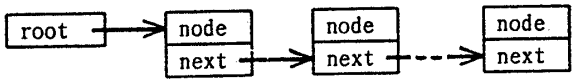
が含まれなければならないと考えられる。

(1) の機能は、データベースの必須機能と言えるもので、３大データモデル（階層、ネットワーク、リレーショナル）のすべてが、この機能を備えている。プログラミングでは、動的メモリ管理機能に対応するものであり、主記憶に限れば、Ｃ言語の malloc 関数やＣ＋＋言語が提供する演算子 newと delete による実現方法が考えられる。

(2) の機能に関して、３大データモデルはかなりの制約を加えている。利用者に許されているのは、基本データ型を並べてレコードを作ることしかなかった。あとは、指定されたレコードの不定個の集まりをＤＢＭＳが管理するというものであった。しかし、ここ数年、利用者が定義したデータ型を使って新しいデータ型を定義することが、複雑化するアプリケーションに対応するための必須機能と考えられるようになり、ＯＯＤＢ宣言[1]では、ＯＯＤＢが満たすべき最初の条件として複合オブジェクトの必要性が述べられている。

3．不定個データの管理

Ｃ＋＋は、不定個数のデータを管理するために動的メモリ管理機能を提供している。図１はこの機能を使ったプログラムで、



のようなリストを処理するものである。

struct Lelem はリスト要素を宣言するもので、整数を記憶する node と Lelemへのポインタ next から構成されている。本文５章でも述べるが、整数 node の部分を書き換えることにより、例えば“整数の集まり（バッグ）のリスト”といったより複雑なデータ構造を表現することができる。class List では、リストの根元を示す変数 root 、Listのコンストラクタとデストラクタ、それにリストへ要素を追加するメンバ関数 inserL(int)、リストの要素をプリントする printL() が宣言されている。続いて、関数 inserL(int), printL()のインプリメンテーションが書かれている。

```
 1  #include  <iostream.h>
 2  struct  Lelem
 3  {  int    node;
 4     Lelem* next; };
 5  class  List
 6  {     Lelem* root;
 7     public:
 8        List() { root= 0;}
 9        ~List(){ delete root;}
10        void   inserL(int);
11        void   printL();
12  };
13  void  List::inserL(int ee)
14  {  Lelem*  pt;
15     if( (pt=new Lelem) != 0 )
16       { pt->node= ee;
17         pt->next= root;
18         root= pt; }
19     else
20       delete pt;
21  }
22  void  List::printL()
23  {  Lelem*  pt;
24     pt = root;
25     if( pt == 0 )
26        cout << "---Nil ¥n";
27     else
28        do{ cout << pt->node << "¥n";
29            pt= pt->next ;
30          } while( pt!=0 );
31  }
```

図１．リストを管理するプログラム

Data Structure for Complex Object

Yosihisa Udagawa

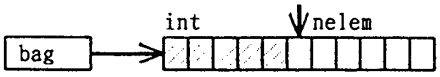
Mitsubishi Electric Corp.

```
 1  #include  <iostream.h>
 2  const  int  nMax=128;
 3  class  iBag
 4  {      int  *bag;
 5         int  nelem;
 6     public:
 7         iBag() { bag=new int[nMax]; nelem=0; }
 8         ~iBag() { delete bag; }
 9         iBag friend Select(iBag, int);
10         void  putBag(int, int);
11         void  print();
12  };
13  void  iBag::putBag(int ee, int ps )
14  {  if( 0<=ps && ps<nMax )
15        { bag[ps]= ee;
16          if(ps>nelem)  nelem=ps;}
17     else
18       cout << "?? Illegal Position ps= " << ps <<"¥n";
19  }
20  void  iBag::print()
21  {  cout << "{ " ;
22     for( int j=0; j<nelem; ++j)
23          cout << bag[j] << " ";
24     cout << "}¥n" ;
25  }
26  iBag  Select(iBag ibp, int val)
27  {  iBag  ans;
28     for( int j=0; j<ibp.nelem; ++j)
29        if( ibp.bag[j] == val )
30           ans.bag[ans.nelem++]= val;
31     return  ans;
32  }
```

図２．バッグを管理するプログラム

４．バッグの実現

図２は、Ｃ＋＋が提供する動的メモリ管理機能を使って整数のバッグを管理するプログラム例である。class iBagは、



のようなデータ構造を表現するためのプライベート変数 bag と nelem、それに５個の関数を宣言している。それぞれの関数の仕様は

iBag()　：バッグの記憶領域の確保と初期化
~iBag()　：バッグの記憶領域の解放
Select(iBag, int)：iBagよりint で指定したデータを選択する
putBag(int, int)　：バッグの指定した所に指定した整数を書き込む
print()　：バッグの内容をプリントする

５．バッグのリストの実現

オブジェクト指向アプローチの目的の一つに、過去に作成したデータまたはプログラムの再利用が上げられている。図３は、図２の iBag の定義をそのまま再利用（継承）し、図１のプログラムを一部修正して“整数のバッグのリスト”を実現したものである。図１と図３を比較すると、図１の int node に関する部分を iBag へのポインタ ibpに書き換えただけであることがわかる。これら一連のプログラムは、Ｃ＋＋でデータ構造の合成が容易に行えることを示す一つの事例となっている。

```
 1  struct  Lelem
 2  {  iBag*  ibp;
 3     Lelem* next; };
 4  class LBag : public iBag
 5  {      Lelem* root;
 6     public:
 7        LBag() { root= 0;}
 8        ~LBag(){ delete root;}
 9        void   inserLB(iBag*);
10        void   printLB();
11  };
12  void  LBag::inserLB(iBag* pb)
13  {  Lelem*  pt;
14     if( (pt=new Lelem) != 0 )
15       { pt->ibp = pb;
16         pt->next= root;
17         root= pt; }
18     else
19       delete pt;
20  }
21  void  LBag::printLB()
22  {  Lelem*  pt;
23     pt = root;
24     if( pt == 0 )
25        cout << "---Nil ¥n";
26     else
27        do{ pt->ibp->iBag::print();
28            pt= pt->next ;
29          } while( pt!=0 );
30  }
31  main()
32  {  LBag  ab;
33     iBag  pB, pC;
34     for( int j=0; j<10; ++j)
35          pB.putBag( (j%3)+100,j);
36       pB.print();
37     ab.inserLB(&pB);
38     for( int k=0; k<7; ++k )
39          pC.putBag( (k%2)+200,k);
40       pC.print();
41     ab.inserLB(&pC);
42     ab.printLB();
43  }
```

図３．バッグのリストを管理するプログラム

なお、図３のプログラムの実行結果は

```
{ 100 101 102 100 101 102 100 101 102 }
{ 200 201 200 201 200 201 }
{ 200 201 200 201 200 201 }
{ 100 101 102 100 101 102 100 101 102 }
```

である。

６．おわりに

本文では、データベースにとって必須であると考えられるデータ構造について考察した。また、Ｃ＋＋によるリスト、バッグを実現するプログラム、それに、この２つのデータ構造を合成してバッグのリストを処理するプログラムを示した。これらのプログラム例は、Ｃ＋＋がデータベースの親言語として優れた特徴を持っていることを示している。

参考文献

(1) Atkinson, K et al : The object-oriented DB manifesto, Proc. DOOD, Kyoto, pp. 40-57, 1989.
(2) Lippman, S. B. : C++ Primer 2nd ed., Addison-Wesley, 1991.