# C＋＋言語のミニマル・サブセットとオブジェクト指向性

## Minimal Subset of C++ and Object-Oriented Programming

小　林　健一郎

Ken-ichiro KOBAYASHI

（平成8年4月5日受理）

初学者がプログラミング言語を習得する過程をみると，その言語に関する知識が一定量を越えたかどうかで以後の学習の成否が決まってくるように思われる．この知識の一定量を本論分ではミニマル・サブセットと呼ぶことにする．これは，以後の自習に最低限必要な知識であると同時に，その言語によって何らかの意味あるプログラムをつくるための基本でもある．BASICやC言語についてのミニマル・サブセットは多くの教科書で取り上げられているが，C＋＋言語についてはあまり例がないように思われる．これはC＋＋言語がC言語の拡張であり，C＋＋言語の学習者は通常すでにC言語によるプログラミング経験を十分につんでいると考えられていることによる．実際，ミニマル・サブセットが必要なのは初学者である．経験豊富なプログラマは自分の目的意識に応じてマニュアルを読むことが可能であり，特にミニマル・サブセットを提示される必要は無いからである．

著者は「C＋＋言語を学ぶ前にC言語を習得すべし」という考え方に疑問を持つ．もちろん，本格的なプログラマを目指す場合は当然C言語を習得すべきであるし，C＋＋言語はその後でよいかもしれない．しかし，多様なニーズのある現代においてC＋＋言語からのプログラミング入門の道を用意しておくことは教育現場においても重要なことではないだろうか．C＋＋言語から入門し必要に応じてC言語を学ぶということも当然可能である．以上より，本論文ではC言語を知らない初学者がC＋＋言語を習得する場合のミニマル・サブセットを論じたい．

C＋＋言語で特に重要な側面はいうまでもなくオブジェクト指向

のサポートである．C＋＋言語のミニマル・サブセットを考える場合，当然オブジェクト指向性が重要となるはずであるが，ミニマル・サブセットそのものの考察が少ない中で，当然「初学者のためのオブジェクト指向」もあまり論じられていなかったように思う．本論文では特にこの点を議論したい．

## 1．はじめに

初学者が言語を習得する過程で，ある段階から知識が飛躍的に増加する現象が見られる．これはその言語に関する知識が一定量を超えたため，以後独自に文献を調べたり「プログラム実験」が出来るようになるからである．ここでその知識の一定量を「ミニマル・サブセット」と呼ぶことにする．もちろん，論理的な意味での厳密なミニマルではない．初学者がとりあえず一人立ちできるだけの最低限の基礎知識という意味である．

BASIC や（DOS 用）C 言語のミニマル・サブセットは，多少の差はあるにせよ，いろいろな書物で十分考察されているが，C＋＋言語についてはほとんどなされていないようである．本論文ではC言語を知らない初学者がC＋＋言語を学ぶ場合のミニマル・サブセットを考察する．

パーソナル・コンピュータ，いわゆるパソコンのパワーアップが注目を浴びるようになってすでに久しい．パソコンは特殊な研究者や趣味人の玩具からビジネス社会に必須な道具へと変貌し，大学等の研究室でもかつての大型計算機やミニ・コンピュータに取って代わる場合すら出てきている．

パソコンのパワーアップは，また，使いやすい OS の登場を可能にし，さらなる普及が進んでいる．今後考えられる OS は，マイクロソフト社の Windows 等，GUI 環境を駆使したウインドウ型 OS であろう．そのため，もし，一般的なプログラミングを考えるのであれば，好むと好まざるとに関わらずウィンドウ（ズ）・プログラミング（以下マイクロソフト社の Windows を主に考えウインドウズ・プログラミングと書く）が必要である．

しかし，使いやすさを求めた OS は，逆にプログラミングに著しい困難をもたらしているのも事実である．いかに簡単にウィンドウズ・プログラミングを行うかは，重要な問題である．

ウィンドウズ・プログラミングに使われる代表的な言語としては，まずC言語が挙げられる．これは WinMain 関数を DOS や UNIX プログラムの main 関数の代わりに使うものである．この方法はウィンドウズ・プログラミングを理解する上でおそらく最も簡便な道を示していると考えられるが，ソースコードの記述が必ずしも簡単ではない．それ故，ソフトハウス等の現場ではマイクロソフト社の MFC（Microsoft　Foundation　Class）やボーランド社の OWL（Object Windows Library）を使ったC＋＋言語によるプログラミングが使われるようになっている．（マイクロソフト社の Visual BASIC やボーランド社の Delphi（Object Pascal）も最近では注目されている．）

本論文は言語や商品の比較を目的とするものではないが，現在，C＋＋プログラミングが極めて重要視されていることは認めて議論したい．この場合考えられるのは，主に，MFC や OWL によるプログラミングである．これらのプログラミングは一般には非常に難しいと考えられている．

MFC プログラミングや OWL プログラミングが難しいとされる理由はふたつある．ひとつは「Windows やコントロールの発するメッセージを処理する」というウィンドウズ・プログラミングの考え方が難しいということであり，もうひとつはＣ＋＋言語のオブジェクト指向性が難しいということである．（ウィンドウズ・プログラミングの考え方は本論文では扱わない．）

さて Visual Ｃ＋＋等を使ったＣ＋＋言語によるウィンドウズ・プログラミングを習得する標準的なコースはどうなるであろうか．現在考えられている最も丁寧なコースは

1．Ｃ言語による DOS プログラミングの習得
2．Ｃ＋＋言語による DOS プログラミングの習得
3．Ｃ言語による Windows プログラミングの習得
4．Ｃ＋＋言語による Windows プログラミング
（Visual Ｃ＋＋，Borland　Ｃ＋＋など）の習得

となるだろう．ここでステップ 3 については別の機会に論じたい．本論文での論点は「ステップ 4 のみを目的とするならばステップ 1 は省略可能である」ということである．

Ｃ＋＋言語がＣ言語の拡張であるということ，また多くのプログラマが実際にＣ言語からＣ＋＋言語へと進んだということから，この道筋は当然のように考えられている．しかし，このコースはあまりに長い．Ｃ＋＋言語が目的であるならば，Ｃ＋＋言語からはじめてよいのではないだろうか．

例えば，有名な Hello プログラムについて，Ｃ言語では

```
/* program 1.c */
#include <stdio.h>
void main()
{
    printf("Hello World!¥n");
}
```

が，Ｃ＋＋言語では

```
//program 2.cpp
#include <iostream.h>

void main()
{
    cout<<"Hello World!¥n"
}
```

となると教えられる．説明はここで stdio.h が iostream.h にかわり，printf 関数を使うところが cout になるなどとされる．

まず，Ｃ＋＋言語を学びたい初学者は，単純に，今後ほとんど使われることのない printf 関数（や scanf 関数など）の使い方をなぜ学ばねばならなかったのだろうかと疑問に思うであろう．

また，coutについての説明は通常不十分で単純にprintf関数の替わりだろう位に考えてしまう．もちろん，coutはオブジェクトであり，これをはじめから説明するのはほとんど不可能でもあるが，いずれにせよ，C言語を学んだ経験はここでは全く生かされないのである．

実際，プログラミングの初学者を考えた場合，オブジェクト指向プログラミングの理解が最も大きな問題である．C＋＋言語はオブジェクト指向プログラミングをサポートする言語であり，C言語はそれをサポートしない．その意味ではC＋＋言語とC言語は全く別の言語と考えるべきなのである．この点を多くのプログラマも誤解しているのではないだろうか．C＋＋言語がC言語の拡張であるという事実は，C＋＋言語がC言語の文法を踏襲しているということであるが，それぞれのプログラミング思想あるいは哲学は全く異なるものなのである．

言語を学ぶということとその言語のサポートするプログラミング思想を学ぶということは一応別のことであるが，初学者にとっては分かちがたいものでもある．C言語を学ぶということは，(無意識的にせよ)構造化プログラミングの方法を身につけるということになるが，オブジェクト指向プログラミングを学ぶことにはならない．このため，C＋＋言語を学ぶ際にC言語の（わずかな）経験がむしろ障害になるのではないだろうか．もちろん，経験あるプログラマはこの点を意識的もしくは無意識的に処理できてしまうのであるが，そのため，いよいよC＋＋言語は専門家にしか扱えないということになってしまう．

プログラミングは技術であり，一般のプログラマが特定の思想や哲学を議論することに意味があるとは思えないが，C＋＋言語をはじめから学ぶという場合には，このオブジェクト指向プログラミングが自然に身に付くようなコースを考えるべきである．

なお，ここでいくつかコメントしておきたい．まず，本論文で議論していることは技術（もしくは技術のための思想）であって思想そのもの（もしくは思想のための思想）ではない．C言語の習得は構造化プログラミングの習得につながると書いたが，C言語を使うプログラマが構造化プログラミングの思想家になるという意味ではない．批判家は「C言語でプログラムを普通に書くと自然に構造化されてくるのであって，多くのプログラマは意図的に構造化プログラミングをしているのではない」と主張するが，その主張は正しいと思われる．同様にC＋＋言語でプログラムを普通に書くとオブジェクト指向型のプログラムになると考えられる．この論文でいうC＋＋言語のミニマル・サブセットは，初学者をオブジェクト指向の思想家にすることが目的なのではなく，C＋＋言語で普通にプログラミングができるようになる教程を構成することが目的なのである．オブジェクト指向はその過程で必然的に現れる構造なのである．

また，本論文でいうミニマル・サブセットの意味である．ここではVisual C＋＋等によるウィンドウズ・プログラミングを最終目標とする初学者を対象に考えている．DOSプログラミングで有用なプログラムを書くためのミニマル・サブセットは意外に大きなものとなる．本論文で取り上げるミニマル・サブセットはウィンドウズ・プログラミングに至るコースで必要なミニマル・サブセットであり，上記のDOS用ミニマル・サブセットよりは小さいものである．

本論文の教程ではアルゴリズム等は全く論じない．

## 2．C＋＋言語のミニマル・サブセット

前節で述べたように，本論文でいうミニマル・サブセットはC＋＋言語の仕様を出来る限りコンパクトにまとめようというものではない．初学者が独学しはじめる前に必要な基礎知識を

まとめたものという意味であり，網羅的なまとめの対極にあるものである．その性質からして，多くの教員・指導者からのご批判を仰ぎ，よりよいものに作り替えるべきものであるが，そのたたき台を以下に提示するものである．なお，「初学者が興味を引くような教え方」は別の問題であり，ここでは議論しない．初学者に実際に教える場合は以下の項目の順番を適宜変更し，また反復する必要もあると思われる．また，この論文の読者にはC言語の知識を仮定するので，C言語との共通項目は題名を挙げるにとどめる．もちろん，初学者に教える場合「C言語ではこうだが，C＋＋言語ではこうなる」式の教え方は，ここでは無益と考える．

2－1　cout と cin
オブジェクトの説明はしない．単に書き出し，読み込みの道具として説明する．

program 2.cpp

```
//program 3.cpp
# include <iostream.h>

void main()
{
    int x;
    cout<<"整数を入力してください．¥n";              //CRT への出力
    cin>>x;                                          //キーボードからの入力
    cout<<"あなたの入力した整数は"<<x<<"です．¥n";  //CRT への出力

    char name[20];                                   //宣言は文中でも可
    cout<<"次に名前を入力してください．¥n";
    cin>>name;
    cout<<"あなたの名前は"<<name<<"です¥n";
}
```

注：//はコメントを表す．

C言語からC＋＋言語に移行しようとする場合には，この段階で足踏みするプログラマが多いと聞く．それはそのプログラマがオブジェクト指向に馴染めないためであろう．オブジェクト指向であれ何であれプログラミングそのものに馴染みのない初学者には，なるべく早く2－3に進ませ，必要に応じて2－1，2－2を繰り返すべきであるだろう．

2－2　データ型（クラスを除く）と式，標準関数，基本演算子
数値型，文字型，配列，ポインタ．
加減乗除，インクリメント，デクリメント．
if 文，while 文，for 文．

標準的な関数と基本的な演算子
すべてC言語と同じ．サンプル・プログラムは cin，cout で簡単に作れる．

2－3　クラスとオブジェクト

クラスはプログラマが独自に定義する型であり，通常，データと関数がひとまとめにされている．

```
//program 4.cpp
class Company
{
private:                        //実は省略化
    int sikin;                  //資金
    int syainsu;                //社員数
public:
    void SetSikin(int n);       //資金を設定する関数
    int GetSikin();             //資金を返す関数
    void SetSyainsu(int n);     //社員数を設定する
    int GetSyainsu();           //社員数を返す関数
    void Show();                //会社の現状を示す
};


//class Company の関数 GetSyainsu の定義．
//Company の関数であるので Company::をつける．以下同様．
int Company::GetSikin(){ return sikin; }
void Company::SetSikin(int n){ sikin = n;}
int Company::GetSyainsu(){ return syainsu; }
void Company::SetSyainsu(int n){ syainsu = n;}

void Company::Show()
{
    cout<<"資金"<<GetSikin()<<"¥n";
    cout<<"社員数"<<GetSyainsu()<<"¥n";
}


void main()
{
    Company Yours;      //Company 型のオブジェクト Yours を宣言
```

```
    int x;
    cout<<"会社が出来ました．その資金を入力してください．¥n";
    cin>>x;
    Yours.SetSikin(x);
    cout<<"その社員数を入力してください．¥n";
    Yours.SetSyainsu(20);
    Yours.Show();
}
```

はじめに資金を入力させ，その後でその資金と社員数を出力するだけのプログラムである．

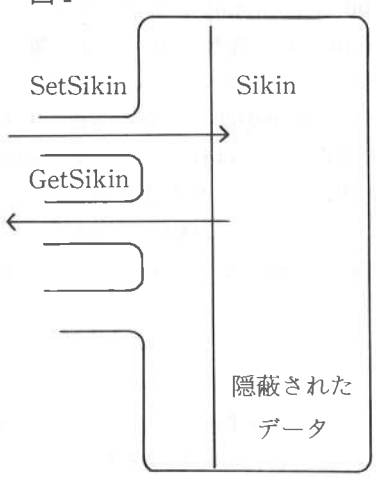
上記の例を見ながらC＋＋言語（およびそのプログラミング思想）についていくつかのことを確認しておきたい．

はじめにCompanyというクラスを宣言している．このクラスは型を表しており「実体」ではない．「実体」はmain関数の中で宣言されているYoursである．このYoursがオブジェクトと呼ばれるものの例である．もちろん同一のクラスから複数のオブジェクトを生成できるがこのサンプルではひとつのみとした．

クラスは構造体のように複数のデータをまとめて保持できるものであるが，簡単のためsikinとsyainsuのみをデータとしてある．sikinはprivate：の後に宣言されているため，プログラム（つまりmain関数）の中で直接呼び出すことが出来ない．正しくは直接呼び出せないようにprivate宣言したのである．直接呼び出すかわりにクラスCompany内で定義したGetSikin関数で値が得られるようにしてあるのである．これらの関数の前にあるpublic:がプログラム中での呼び出しを許可する宣言である．これらの関数を上記のmain関数内で呼び出すには，Yours.GetSikin()などとすればよい．

（初学者は即座にこのプログラムの意義に疑問を感じるところであるが，実際にどのように教えるかは前述のようにここでは議論しない．ただ，クラスCompanyを使って例えば付録のprogapp1.cppなどを書くことができることは指摘しておきたい．C＋＋言語はこのように一度設計したクラスが別のプログラム内で使われることを想定している．以下を参照のこと．）

図1




オブジェクト指向型のプログラミングになれていないプログラマには，わざわざsikinをprivateにして，GetSikinやSetSikinを定義する上記のサンプルは冗長に感じられるかもしれない．しかし，実際，これがオブジェクト指向プログラミングのスタート点である．よく見かけるものであるが，以下の図がわかりやすい．

この図が示すようにsikinはわざと「外界」からアクセス出来ないように作ったのである．これを「データの隠蔽」と言う．このような構造をよしとするのはC＋＋言

語がいずれにせよ極めて大規模なプログラミングを予期して作られていることによる．

大規模なプログラムは一般に複数のプログラマによって作られる．指揮者を除けば，プログラマは自分の担当箇所のみに集中したいと考えるはずであるし，常に全体の細部を考えることは不可能である．

また，一人でプログラムを作る場合でも，製作に要する時間の長さを考えると，他のプログラマと協力してプログラミングしているのと同じことになる．(俗に半年前に自分が書いたプログラムは他人が書いたプログラムと同じであると言われる．)

この問題をうまく解決するのが，プログラムの部品化である．プログラムを出来るだけ単純な部品に分け，それぞれを構築し，その後に持ち寄って大きなプログラムを完成させるという方法である．いわゆる構造化プログラミングはこの方法を具体化したものである．

非オブジェクト指向の構造化プログラミングで書けないほどの大規模なプログラムがオブジェクト指向プログラミングで書けるかということについて疑問視する人もいるようであるが，少なくとも同程度の規模のプログラムは可能である．

オブジェクト指向プログラミングもプログラムを分割するという発想は同じであるが，非オブジェクト指向型と違い，オブジェクト指向プログラミングでは，データのみならずデータを扱う関数をひとまとめにし（C＋＋言語ではそれがクラスと呼ばれる），さらにデータを（なるべく）隠蔽する．複数のプログラマはこのような約束のもとに分担したクラスを設計し持ち寄りひとつのプログラムとするのである．

データを隠蔽する理由は，他のプログラマがそのデータに自由にアクセス出来ないようにするためである．program 4.cpp のような小規模プログラムであれば，このような制限は煩わしいだけであるが，大規模なプログラムの保守・管理には極めて有効である．

そもそも program 4.cpp の main 関数はクラス Company の動作チェックのためにあるだけで，ここで重要なのは Company である．また，実際の大規模プログラムは他のプログラマが作った別のクラスとあわせてより複雑で意味のある main 関数を持つはずであることは指摘しておいてよいだろう．

そのような大規模プログラムでクラス内のデータ構造が変わった場合にも，そのデータが直接アクセスされていなければクラスの内部以外は何も変更しなくてよいのである．program 4.cpp の Company の構造が複雑になり，例えば「資金」が jikosikin と kariire の合計になり，sikin という変数そのものは不要になったとする．もちろん残しておくことも可能だが，無駄である．この変数をなくした場合でも SetSikin(int n), GetSikin()の内部を変更するだけでプログラムの他の部分は全く変更されないのである．(この例は単純なものである．例えば，kariire の設定がしたければそのための関数を増やす必要はある．また，実際の条件に応じて継承などを使い洗練された方法を考えることもできるが，ここでは議論しない．)

また，ここでは紹介しなかったが，クラスを使うプログラミングでは，コンストラクタと呼ばれる関数を使うことでデータの有効性を常にチェックできる利点もある．

以上をまとめると

オブジェクト指向プログラミングは，一見した冗長性は覚悟した上で，大規模なプログラミングを安全に行える機構を提供している

ということになるだろう．

ところで，初学者がこのような大規模プログラムを目指した言語をそもそも学ぶ必要があるのかという疑問も考えられる．初学者は必然的に大規模なプログラムなど書かないものであるし，まず，簡単なプログラミング言語を覚え，必要に応じてC＋＋言語へ移行すべきであるという意見もでるであろう．しかし，著者は以下の事実も考えるべきであると主張する．

そもそも，C＋＋言語の良い点は，規則さえ学べば他人の作ったクラスを容易に使うことが出来るということである．それ故，初学者がプロのプログラマが作ったクラスを利用することもできる．自身はわずかにコードを書きたすだけで大規模プログラムをつくることが出来るのである．

その典型的な例がウィンドウズ・プログラムである．前節で述べたようにウィンドウズ・プログラムは複雑であり，一般に（少なくとも初学者にとっては）大規模にならざるを得ない．例えば，典型的なHelloプログラムのソース・コードは80行位である．

このようにウィンドウズ・プログラムが長くなるのは，アプリケーションのウィンドウやコントロールを記述しなければならないからである．しかし，大多数のアプリケーション・プログラムはほぼ同じ構造を持っており，通常のプログラムで使われる共通部分がまとめてあれば便利である．

その「まとめ」の仕組みがクラスで与えられるのである．簡単に言うと大部分のウィンドウズ・プログラミングで共通する部分が，マイクロソフト社の場合はMFC(Microsoft Foudation Class）ライブラリ，ボーランド社ではOWL（Object Window Library）と呼ばれるライブラリ内のクラスに書き込まれているのである．これらのクラスの使い方がわかれば，ウィンドウズ・プログラミングは容易になる．

以上の事実からも，初学者が（トータルで）大規模なプログラムを書くことは十分あり得るし，また，ウィンドウズ・プログラミングを望むのなら，敢えて他の言語で遠回りをする必要はないと考えるのである．

さて以上の説明はすべて教える側の了解事項にすぎない．はじめからすべてを説明されても初学者は戸惑うだけであろう．ミニマル・サブセットに必要なことは，なるべく単純なクラスの構築をさせ，

C＋＋言語のプログラミング＝クラスの設計と使用

ということを体験させることではないだろうか．むしろ初学者の方が，以下のようなサンプルを拒絶しないと思う．

```
//program 5.cpp
# include <iostream.h>
# include <string.h>          //strncpy()を使う
```

```
class Neko
{
    char name[21];          //デフォルトでは private になる
public:
  //コンストラクタ Neko()を宣言する.
  //これはオブジェクトの初期化に使われる特殊なメンバ関数で,
  //クラス名と同じ名前を使う. main()を参照のこと.
  //デフォルトの何もしないコンストラクタ (program 6.cpp 参照)
    Neko(){}
  //名前を設定してオブジェクトを生成するコンストラクタ
    Neko(char* n);
    void SetName(char* n);
    void ShowName();
};

Neko::Neko(char* n)
{
    SetName(n);
}

void Neko::SetName(char* n)
{
    strncpy(name,n,21);
}

void Neko::ShowName()
{
    cout<<"猫の名前は¥n";
    cout<<name<<"です. ¥n¥n";
}

void main()
{
    char name[21];
    //名前を設定するコンストラクタが呼び出される
    Neko Doraneko("ボス");
    //ボスという名の猫が生成された

    Doraneko.ShowName();
```

```
    cout<<"猫の名前を変えてあげましょう．¥n";
    cout<<"好きな名前を入力してください．¥n";
    cin>>name;
    Doraneko.SetName(name);
    cout<<"¥n 名前が変わりました．¥n¥n";

    Doraneko.ShowName();
}
```

2－4　オブジェクトの動的生成と消滅

例えば，飛行機の乗客リストを整理するプログラムを考えたとする．手続き指向のプログラミングでは，乗客リストや乗客のデータが最も効率的な方法で処理されるようなプログラムが設計されるであろうが，一方でそのようなプログラムは効率的であればある程変更しづらいものでもあるだろう．また，実際のシステムとプログラムの内容がかなり異なることは周知の事実である．

一方，オブジェクト指向型のプログラミングでは，実際のシステムがプログラム構成のヒントとなる．プログラマによって異なるプログラムができることは当然だが，「乗客」や「飛行機」のクラス（のオブジェクト）がつくられ，「飛行機の座席に乗客が割り当てられる」という発想は自然である．

オブジェクト指向とはこのように現実をシミュレートする形でプログラミングを進めていくものであった．このようなプログラムではオブジェクトの動的な生成と消滅が重要である．C＋＋言語では生成に new 演算子，消滅に delete 演算子が使われる．オブジェクトの配列を消滅させるときは delete []が使われる．

```
//program 6.cpp
class Neko
{
//略 program 5.cpp を参照のこと
};
//猫のインプリメントも同じ

void main()
{
    int x;
    Neko* cat;      //Neko オブジェクトのポインタ
    char name[21];

    //ユーザの入力を待ってオブジェクトを生成する
    cout<<"猫の数を入力してください．";
    cin>>x;
```

```
    cat = new Neko[x];              //x匹の猫を生成
    //ここでは何もしないコンストラクタが呼び出された
    //このためにこのデフォルトのコンストラクタが必要なのであった

    for(int i = 0;i<x;i++){
        cout<<"猫の名前を入力してください．¥n";
        cin>>name;
        cat[i].SetName(name);
    }

    cout<<"¥n登録された猫の名前を書き出します．¥n";
    for(i = 0;i<x;i++){
        cout<<i+1<<"¥n";
        cat[i].ShowName();
    }
    delete [] cat;      //Nekoオブジェクトの消滅
}
```

2－5　継承と仮想関数

C＋＋言語のオブジェクト指向性は，具体的にはクラスの継承で実現されている．これはあるクラスから別のクラスを派生させるものである．例えばprogram 5.cppでクラスNekoを定義したが，Nekoの中にもいくつかの種類が考えられる．このような場合，Nekoのメンバを含む形で新しいクラスを作ることができる．この新しいクラスを派生クラスという．

ここで例えば月給取りの猫と利子生活者の猫がいたとする．この2種類の猫はともに「名前」が必要であるが，月給取りは「月給」，利子生活者には「貯金」という変数が必要になる．このような猫，SalaryNeko，SisankaNekoは以下のサンプルのように定義される．

```
//neko.h
class Neko
{
    char name[21];
public:
    Neko(){}
    Neko(char* n);
    void SetName(char* n);
    void ShowName();
    //program 5.cppのNekoに次の2行が増えている
    virtual ~Neko(){}       //仮想デストラクタ．仮想関数を使うときには必要．
    virtual long GetSyunyu()= 0;  //純粋仮想関数．後述．
};
```

```
Neko :: Neko(char* n)
{
    SetName(n); //メンバ関数
}

void Neko :: SetName(char* n)
{
    strncpy(name,n,21);
}

void Neko :: ShowName()
{
    cout<<"猫の名前は¥n";
    cout<<name<<"です。¥n";
}

class SalaryNeko : public Neko  //クラス「サラリー猫」
{
    long gekkyu;  //月給。名前は基本クラス Neko にあるのでここでは不要。
public:
    SalaryNeko(char* n, long money);
    long GetSyunyu();
};

//コンストラクタ
//:Neko(n)で基本クラス「Neko」のコンストラクタに n をわたす
SalaryNeko :: SalaryNeko(char* n, long money):Neko(n)  {
    gekkyu = money;
}

long SalaryNeko :: GetSyunyu()
{
    return gekkyu * 12;  //ボーナスなしとした。
}

class SisankaNeko : public Neko  //クラス「資産家猫」の派生
{
    long chokin;  //貯金。名前は基本クラス Neko にあるのでここでは不要。
public:
```

```
    SisankaNeko(char* n, long money);
    long GetSyunyu();
};

SisankaNeko:: SisankaNeko(char* n,long money):Neko(n) // コンストラクタ
//:Neko(n)で基本クラスのコンストラクタにnをわたす
{
    chokin = money;
}

long SisankaNeko::GetSyunyu()
{
    return chokin * 2/100;      //銀行の利率を年2パーセントとした
}
```

クラスNekoのメンバに純粋仮想関数「virtual long GetSyunyu()＝0」を付け加えた．「virtual」が「仮想」，「＝0」が「純粋」を表す．この仮想関数がC＋＋言語（および同系統の言語）の最大の特徴と言えるであろう．

上記の例ではサラリー猫も資産家猫も猫の一種として定義（派生）された．それぞれに収入があり，収入を与えるGetSyunyu関数がある．どちらも「収入」という意味においては同じものであるが計算式は全く異なるものである．

しかし，（純粋）仮想関数「virtual long GetSyunyu()＝0」を基本クラスのメンバとすることで，サラリー猫と資産家猫のGetSyunyu関数を統一的に扱えるようになるのである．まず，それぞれのオブジェクトをポインタで表すときに，それぞれのクラスのポインタ，SalaryNeko *，SisankaNeko *ではなく，基本クラスのポインタNeko *を使う．（C＋＋言語ではこれが可能になっている．）Nekoポインタでオブジェクトのメンバ GetSyunyu を呼び出すと，それぞれの派生クラスのメンバ GetSyunyu が呼び出されるのである．

「純粋」とあるのはクラスNekoそのものでは使われないからである．クラスNekoでも使うようにするには＝0をとって，関数のインプリメントをすればよい．この場合はただの「仮想関数」となる．

なお，仮想関数を使う場合には仮想デストラクタを定義する必要がある．これを置くことでdeleteがこの派生クラスを確実に消滅させるのである．

```
//program 7.cpp
# include "neko.h"
void main()
{
    Neko* neko[2];          //二匹の基本猫のポインタ
    SisankaNeko Rock("ロックフェロー",100000); //資産家猫(名前と貯金)
```

```
    SalaryNeko Tama("たま",12);          //サラリー猫（名前と月給）
    //二匹の猫のポインタを代入
    neko[0]=&Rock;
    neko[1]=&Tama;


    for (int i=0;i<2;i++){
        cout<<"番号"<<i+1<<"¥n";
        neko[i]->ShowName();
        cout<<"収入は"<<neko[i]->GetSyunyu()<<"です．¥n";
        cout<<"¥n";
    }
}
```

2－6　その他

他にも重要な項目があるが，ここでは箇条書きのみとする．
すなわち，インライン関数，const，参照，関数のデフォルト引数，オーバーロード，静的データメンバ，protected，フレンド関数，フレンドクラス，多重継承．
それぞれ上記のコース中で無理なく解説され得ると思われる．

2－7　ストリームについて

C＋＋言語のコンパイラには標準でストリームオブジェクトと呼ばれる入出力の機能がサポートされている．ここでいうストリームオブジェクトは組み込まれたクラスのオブジェクトでバイト列の転送元，転送先として振る舞うものである．

最初の例が cin，cout であるが，これらは iostream クラスの定義済みオブジェクトである．cin は標準入力，cout は標準出力である．他に cerr（上限つきでバッファリングされる標準エラー）と clog（上限無しの標準エラー）がある．

また，ファイルへの入出力をする場合は ifstream，ofstream クラスのオブジェクトが使われる．この場合はもちろん自分でオブジェクトを定義する必要がある．

```
//program 8.cpp
#include <iostream.h>
#include <fstream.h>
void main()
{
    int a;
    ofstream file("a:¥¥file.txt");          //書き込むファイル file.txt を作成

    cout<<"好きな整数を入力してください．";
    cin>>a;
```

```
    file<<a;                              //file.txt への書き込み
}
```

2－8　ポインタの概念をよく理解したものへの補充
　代入演算子，コピーコンストラクタ．

付録

```
//progapp1.cpp
#include <iostream.h>
#include <stdlib.h>     //乱数の発生に必要
#include <conio.h>      //getch()を使う

class Company
{
private:
    int sikin;                  //資金
    int syainsu;                //社員数，このプログラムでは使用しない．
public:
    void SetSikin(int n);       //資金を設定する関数
    int GetSikin();             //資金を返す関数
    void SetSyainsu(int n);     //社員数を設定する
    int GetSyainsu();           //社員数を返す関数
    void Show();                //会社の現状を示す
};


//class Company の関数 GetSyainsu の定義．以下同様．
int Company::GetSikin(){ return sikin; }
void Company::SetSikin(int n){ sikin=n;}
int Company::GetSyainsu(){ return syainsu; }
void Company::SetSyainsu(int n){ syainsu=n;}

void Company::Show()
{
    cout<<"資金"<<GetSikin()<<"¥n";
    cout<<"社員数"<<GetSyainsu()<<"¥n";
}

class Keizaikai                 //二大企業からなる経済界
{
```

```
    Company Mine,Yours;      //私の会社とあなたの会社
public:
    void Hajime();           //事の起こり
    void Shoubu();           //5年間の戦い
};

void Keizaikai:: Hajime()
{
    int x;
    cout<<"二つのライバル会社ができました．¥n";
    cout<<"一つは私の会社でもう一つはあなたの会社です．¥n";
    cout<<"一方の会社の利益は他方の会社の損失で，¥n";
    cout<<"利益・損失はランダムに決まります．¥n";
    cout<<"私の会社の資金は100とします．¥n";
    Mine.SetSikin(100);
    cout<<"次にあなたの会社が出来ます．その資金を決めてください．¥n";
    cout<<"100以下の整数とします．¥n";
    cin>>x;
    Yours.SetSikin(x);
}

void Keizaikai::Shoubu()     //アルゴリズムの効率は考えていないので注意
{
    int i,r,s;
    randomize();                    //Turbo C++
    cout<<"¥n今後5年間の競争をシミュレートします．¥n";
    cout<<"どれかキーを押してください．¥n";
    getch();

    for(i=0;i<5;i++){
        r=rand()%10;  //私の利益
        s=rand()%10;  //あなたの利益
        Mine.SetSikin(Mine.GetSikin()+r-s);     //自分の得は相手の損
        Yours.SetSikin(Yours.GetSikin()+s-r);  //相手の得は自分の損
        cout<<i+1<<"年目:  ";
        cout<<"私の会社のお金  "<<Mine.GetSikin()<<"    ";
        cout<<"あなたの会社のお金  "<<Yours.GetSikin()<<"¥n";
    }
//ここではランダムに業績を決めているので実行してもあまり意味はない．
//しかし，ケースバイケースで業績算出方法を変えればいろいろなシミュレーション
```

```
//ができる．この場合，書き直すのはこの Shoubu 関数のみでよい．
//もちろん，複数の業種の算出方法が確定していれば継承と仮想関数を使える．
}

void main()
{
    Keizaikai Nihon;

    Nihon.Hajime();
    Nihon.Shoubu();
}
```

## 3．オブジェクト指向プログラムの設計思想

手続き指向型の構造化プログラミングでは，まずプログラムの動作を決定し，そのためのアルゴリズムを考え，アルゴリズムの各ステップをトップダウン式に分解し，それをプログラミング言語で記述していくという手法がとられている．

この方法は構造化プログラミングのかつての流行を別にしても，一定の説得力を持つ手法であると考えられる．（著者には，従来の「理科系的発想」の自然な結論に思えるものであり，その有効性を否定することはできない．）

一方，オブジェクト指向プログラミングの発想は全く異なるものである．メモリ上の構造を「もの＝オブジェクト」と見立て，オブジェクト間の相互作用システムを構築するのがプログラムなのである．

従って，プログラムの設計は，「動詞」ではなく「名詞」からはじまると言える．まず，必要な「もの」を考え，これをクラスとして登録する．このクラスの属性と動作を考え，クラス間の関係を記述する．main 関数ではクラスのオブジェクトを生成し，後はそのオブジェクトの動作がプログラムの目的を遂行していくというものである．

また，オブジェクトの構造としては，可能な限りデータが隠蔽されているものが推奨される．これは 2 節で述べたように多人数でのプログラミングを考えた場合有効な規則である．また，データが隠蔽されていなければオブジェクトの独立性が失われ，オブジェクト指向の構造そのものが曖昧になるからである．

このようなプログラミングの手法が，もちろん万能でないことも強調しておきたい．オブジェクト指向が非常にうまく適合するプログラムとしては，例えば，複数の「もの」が相互作用する現実のシステムのシミュレーションがある．しかし，この手法が従来のプログラムより効率的であるとは限らない．

また，C＋＋言語の制作者であるストラウストラップ氏は著書「プログラミング言語C＋＋」で，継承と仮想関数を用いて利用できる型の間の共通性の量が少ない場合，オブジェクト指向プログラミングの機能は必要ないようだと述べている．

## 4. MFC, OWL プログラミングへのコメント

本論文でウィンドウズ・プログラミングを議論する余裕はない．しかし，第1節で述べたようにC＋＋言語を学ぶひとつの理由はMFCもしくはOWLプログラミングの習得であった．ここではMFCプログラミングの概要について述べてみたい．

MFCライブラリはMicrosoft Foudation Classと呼ばれる定義済みのクラスの集合である．プログラマはこのクラスを直接使用するか，派生クラスを作って使用することにより，「ウインドウの枠づくり」のようなルーチンワークから解放される．

例えばアプリケーション内にスタンダードなエディタが必要になることがある．エディタを作ることは困難ではないがかなり手間のかかることである．MFCにはエディタのクラスが定義されており，そのオブジェクトを生成するだけで「メモ帳」エディタを利用することが出来るのである．

MFCの最も基本的なクラスはCObjectで，大部分のMFCはCObjectの派生クラスである．プログラマがこのクラスの派生クラスを作成・使用することもできる．この派生クラスではシリアライズ等の機能を使うことができ，かなり便利である．

CObjectとは独立したクラスにデータ型のCPointなどがある．CPointは画面上の点の座標を格納する構造体にインターフェースがついたものである．このような標準クラスを自分でそのつど定義しなくてもよいのは精神衛生上も歓迎される．

MFCを使うウィンドウズ・プログラミングの有力な道具としてマイクロソフト社のVisual C＋＋があるので，例としてこれを考える．簡単に言うと，Visual C＋＋がアプリケーションクラス，ドキュメントテンプレートクラス，ドキュメントクラス，フレームウィンドウクラス，ビュークラスを自動的に生成し，そのオブジェクト間の相互作用を記述することがプログラミングとなる．アプリケーションオブジェクトはアプリケーション全体の管理，ドキュメントテンプレートオブジェクトはドキュメント，フレームウィンドウ，ビューの各オブジェクトの統合，ドキュメントオブジェクトはデータの管理を行い，フレームウィンドウオブジェクトはビューの外枠，ビューオブジェクトはデータを見せる窓，を意味している．

これらのクラスの一般的な共通部分はMFCで定義されており，プログラマは派生や仮想関数を使うことで，独自のコードを付け加えていくのである．

OWLプログラミングの基本的なアイデアは同じで，こちらはボーランド社が用意したクラスを使うプログラミングである．また，マイクロソフト社に限らず多くのソフト会社が独自のすぐれたプログラミング環境を提示していることも付け加えておく．

MFCおよびOWLの詳細については別の機会に論じたい．

## 5. 結　　語

C＋＋言語のミニマル・サブセットを考察した．ミニマル・サブセットを学習することで，

C言語を経由することなく容易にC＋＋言語を習得できると考えるからである．C言語を経由せずにC＋＋言語を学ぶ理由は，オブジェクト指向プログラミングをより早く学ぶことにある．これによりMFCやOWLなどを使ったウインドウズ・プログラミングにも容易に移行できると考えられる．

最後にC＋＋言語の生みの親であるストラウストラップ氏のコメントを「プログラミング言語C＋＋」（読者への覚え書き）から引用する．

「Cをよく知っていればいるほどCのスタイルでC＋＋を書くことを避けるのが難しく，それによりC＋＋の潜在的な恩恵のいくつかを失うように思われる．

（中略）

最も重要なことだが，プログラムをデータ構造とそのビットを上げ下げする関数の束とみなすことに代わり，プログラムをクラスやオブジェクトとして表現された相互作用する概念の集合と考えるよう試みよ．」

## 謝　辞

高橋恒介教授にはC／C＋＋をはじめ様々な問題を議論していただきました．また，樽松直樹教授，佐野典秀助教授，川口順功助教授，大石義助教授にも日頃から価値ある情報を教えていただいています．小谷内郁宏助教授には情報処理センターにおいていろいろと便宜をはかっていただきました．以上の先生方をはじめとする静岡学園短期大学諸先生方のご助力に感謝いたします．

## 参考文献

B．ストラウストラップ著，斉藤信男，三好博之，追川修一，宇佐見徹訳『プログラミング言語C＋＋』トッパン　1993

S．C．デューハースト，K．T．スターク著，小山裕司訳『C＋＋言語入門』アスキー出版局第2版　1995

マイクロソフト社『C＋＋チュートリアル』MS-C／C＋＋，Visual　C＋＋に付属のマニュアル

マイクロソフト社『C＋＋　Language Reference』MS-C／C＋＋，Visual　C＋＋に付属のマニュアル

マイクロソフト社『クラスライブラリ　リファレンス』Visual　C＋＋に付属のマニュアル

ボーランド社『Object Windows』Turbo C＋＋に付属のマニュアル