計算機プログラミング'95 ＃7

久野　靖*

1995.10.19

1　課題について

守田さんから「課題について心配である」との感想を頂きましたので、課題についてまとめておきます。どれか 1 つだけ選択すればよいのでしたね。

課題 1 画面パッケージを利用して「何か役に立つ」プログラムを設計し製作せよ。

課題 2 画面パッケージを利用したメニューやフォームのサブルーチンにならって、汎用的で役に立ちそうな画面を作り出すルーチンを 2 つ以上設計し実現せよ。

課題 3 画面パッケージのメニューやフォームルーチンを利用して「誰でも簡単に階層メニューのプログラムが作れる」ような仕掛けを設計し、実装せよ。

課題 4 丸や四角を出すプログラムを改良してもっと図形を増やし、簡単な「お絵描きプログラム」にしてみよ。

今後もまだ授業の内容に沿って増えると思いますが、時間が心配ならばこれらの中から選んで設計をはじめても構いません。レポートを書くときは、設計方針や考えたこと、実現上の苦労、実現した結果分かったこと (考察) なども記すこと。リスティングだけではレポートとして認めません。レポートの〆切は 2 学期終了から 2 週間くらい後の予定です。

2　C++言語入門

2.1　復習: 抽象データ型

先週までに取り上げてきたように、抽象データ型 (Abstract Data Type、ADT) とは

- 外部から保護されたデータと、それをアクセスする関数群という点ではパッケージと同様。
- ただし、普通パッケージでは実体が 1 つだが、抽象データ型では任意個数の実体 (インスタンス) を生成できる。
- 利点としては、インタフェースを変更しなければ内部実現は自由に変化させられ、プログラム部品間の独立性が高まること。

しかし、C 言語で ADT をやるには「ある一定の規則に従ってプログラムを構成する」わけで、その規則が言語によって強制されていない以上様々な問題が派生する (具体的には何か?)

*筑波大学大学院経営システム科学専攻

そこで、「よりよい言語」を使用するのがまっとうなアプローチ。ここではC++言語を使用する。C++はオブジェクト指向言語(って何?)と言われるが、オブジェクト指向の利点の1つがADTなので、これはこれでよい(それ以外の利点については次回以降で取り上げる)。

C++では「クラス」によってADTが作成できる。その構文は次の通り。

```
class クラス名 {
private:
  変数定義…
public:
  関数宣言…
};              ←この「;」を忘れないように!!!
```

private:部分はモジュール内のデータ(実体変数)で外部からは隠蔽される(厳密にはモジュール内ローカル関数もあってよい)。public:部分は外部インタフェースをなす関数(メンバ関数)の宣言(厳密にはデータを公開してもよいがADTの主旨からいうと薦められない)。

メンバ関数の定義は

```
[型指定] クラス名::関数名(引数部…) { 文… }
```

つまり普通の関数定義とおなじ。そして、メンバ関数の中ではクラス宣言で定義した変数がローカル変数として使える。

特例として、クラス名と同じ名前の関数があり、これを「コンストラクタ」という。コンストラクタはクラスの実体が生成される時に呼ばれ、実体変数を初期設定するのがおもな役割。クラスを生成するには、

```
new クラス名(引数…)
```

により記憶場所を割り当てクラス実体へのポインタを得る。引数はコンストラクタ関数の引数。そしてメンバ関数を呼び出すには

```
p->メンバ関数名(引数…)
```

による。なお、メンバ関数やコンストラクタ以外に、Cと同様の普通の関数もあってよい(mainなどがそう)。

2.2 ディスプレイ型のADT

では早速、前回の「円が出るだけ」のプログラムで、「ディスプレイ」関係のデータをADTにしてみる。

```
/* t71.c --- Display ADT in C++ */

#include <X11/Xlib.h>  ←ヘッダファイルはCと兼用

class Disp {   ←ディスプレイのADT
private:
  Display *dsp1;          ←Xの Display
  Screen *scr1;           ←Xの Screen
  Window root1;           ←Xのルート窓
  unsigned long fg1, bg1; ←白と黒
  GC gc1, rgc1;           ←2つの GC
public:
  Disp::Disp();           ←コンストラクタ
  Display *dsp();         ←以下は各種の値を返す
```

```
  Screen *scr();
  Window root();
  unsigned long fg();
  unsigned long bg();
  GC gc();
  GC rgc();
};

Disp::Disp() {        ← Disp は最初に 1 個だけ作るつもり。
  dsp1 = XOpenDisplay(NULL);  ←各種初期設定
  scr1 = DefaultScreenOfDisplay(dsp1);
  root1 = DefaultRootWindow(dsp1);
  fg1 = BlackPixelOfScreen(scr1);
  bg1 = WhitePixelOfScreen(scr1);
  gc1 = DefaultGC(dsp1, 0);
  XGCValues gcv; gcv.function = GXinvert; gcv.line_width = 3;
  rgc1 = XCreateGC(dsp1, root1, GCFunction|GCLineWidth, &gcv);
}

Display *Disp::dsp() { return dsp1; }  ←アクセス関数群
Screen *Disp::scr() { return scr1; }
Window Disp::root() { return root1; }
unsigned long Disp::fg() { return fg1; }
unsigned long Disp::bg() { return bg1; }
GC Disp::gc() { return gc1; }
GC Disp::rgc() { return rgc1; }

int draw(Disp *d, Window w, int x, int y);  ← C++では宣言必須

main() {
  Disp *d = new Disp();  ←ここで Disp を作る
  Window mw = XCreateSimpleWindow(d->dsp(), d->root(),100,100,400,200,2,
                                  d->fg(), d->bg());
  XSelectInput(d->dsp(), mw, ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(d->dsp(), mw);
  while(1) {
    XEvent ev;
    XNextEvent(d->dsp(), &ev);
    if(ev.type == KeyPress)
      exit(0);
    else if(ev.type == Expose)
      draw(d, mw, 20, 20);
    else if(ev.type == ButtonPress)
      draw(d, mw, ev.xbutton.x, ev.xbutton.y);
  }
}

draw(Disp *d, Window w, int x, int y) {
  XFillArc(d->dsp(), w, d->gc(), x-20, y-20, 40, 40, 0, 360*64);
}
```

いかがですか? Disp という ADT ができただけで、main はあんまり変わっていない。それでもとにかく、「X の各種資源をアクセスするには Disp のインスタンスのアクセス関数で取ってくる」という統一ができただけで結構「美しい」と思うがどうでしょう。

2.3 Window の ADT

ADT は複数の実態が作れるが、上の Disp は 1 個だけしか作らないのであまりらしくなかった。そこで次に「窓」を ADT にして、その後複数の窓を開くことを考えてみよう。Disp クラスはそのままにして、その後を Win クラスと main の組み合わせに置き換える。まず class 定義から。

```
class Win {
private:
  Disp *dsp1;     ←Disp は内部にとっておく
  Window win1;    ←窓の本体
  int exit1;      ←窓を閉じてしまったかどうか?
public:
  Win(Disp *d, int x, int y, int w, int h);  ←コンストラクタ
  void destroy();           ←窓を閉じる
  int handle(XEvent *ev);  ←イベントの処理
private:
  void draw(int x, int y); ←ローカル関数。丸を描く。
};
```

まず窓を作るのはこれまで通り。

```
Win::Win(Disp *d, int x, int y, int w, int h) {
  exit1 = False;
  dsp1 = d;
  win1 = XCreateSimpleWindow(d->dsp(), d->root(),x,y,w,h,2,d->fg(), d->bg());
  XSelectInput(d->dsp(), win1, ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(d->dsp(), win1);
}
```

消すときは XDestroyWindow を使う。また、消したという旗を立てる。

```
void Win::destroy() {
  XDestroyWindow(dsp1->dsp(), win1); exit1 = True;
}
```

おもしろいのは handle。これは、イベントを渡されて「処理しようとする」。ここでイベントが自分の受け持ち範囲なら処理した上で True を返すが、受け持ちでないなら何もせず False を返す。

```
int Win::handle(XEvent *ev) {
  if(exit1 || ev->xany.window != win1) return False;
  if(ev->type == KeyPress)
    exit(0);
  else if(ev->type == Expose)
    draw(20, 20);
  else if(ev->type == ButtonPress)
    draw(ev->xbutton.x, ev->xbutton.y);
  return True;
}
```

draw は前と同じだが、Disp や Window は中で持っているので渡さなくてよい。

```
  void Win::draw(int x, int y) {
    XFillArc(dsp1->dsp(), win1, dsp1->gc(), x-20, y-20, 40, 40, 0, 360*64
);
  }
```

では main。

```
  main() {
    Disp *d = new Disp();
    Win *w1 = new Win(d, 100, 100, 400, 200);
    Win *w2 = new Win(d, 110, 110, 400, 200);
    while(1) {
      XEvent ev;
      XNextEvent(d->dsp(), &ev);
      if(w1->handle(&ev))
        ;
      else if(w2->handle(&ev))
        ;
      else
        ;
    }
  }
```

おもしろいでしょう?

練習 1 これを改造して、窓を 5 個開くようにしてみよ (できるだけ配列を利用する)。

練習 2 それぞれの窓で、丸が復活できるように改造せよ。

2.4 簡単なクラスを自作する

上のプログラムではどれかの窓でキーを押したとたんにすべての窓が終わってしまう。これではつまらないので、キーを押したらその窓だけ終わるようにしよう。そのため、「個数を数える」次のようなクラスを設計した。

```
Counter *c = new Counter(5);  ←初期値 5 のカウンタを作る。
c->add(-1);                   ←カウンタに値を足し込む。
while(c->value() > 0) ...     ←カウンタの現在値を参照。
```

練習 3 Counter のクラス定義を書け。

練習 4 Counter のメンバ関数定義を書け。

練習 5 Counter を使うように先のプログラムを直せ。具体的には

- main の先頭で窓の個数を初期値としたカウンタを用意。
- Win の実体変数とコンストラクタの引数にカウンタを追加。
- Win でキーが押されたら窓を消してカウンタを-1 する。
- main の無限ループを「カウンタが 0 以上ならループ」に変更。

2.5 表示対象物もオブジェクトに

次に、表示対象 (この場合は丸) もオブジェクトにする。

```
Circle *c = new Circle(d, w, r, x, y);  ←コンストラクタ
c->draw();  ←描く
```

カウンタよりずっと簡単でしょう?

練習 6 `Circle` クラス (定義、メンバ関数) を作成せよ。

練習 7 `Win` の実現を `Circle` を使うように変更せよ。

2.6 ここまでで完成したプログラムの全リスト

```
/* t74.c --- make circle an object */

#include <X11/Xlib.h>

class Counter {
private:
  int value1;
public:
  Counter(int n);
  int add(int n);
  int value();
};

Counter::Counter(int n) { value1 = n; }
int Counter::add(int n) { return value1 += n; }
int Counter::value() { return value1; }

class Disp {
private:
  Display *dsp1;
  Screen *scr1;
  Window root1;
  unsigned long fg1, bg1;
  GC gc1, rgc1;
public:
  Disp::Disp();
  Display *dsp();
  Screen *scr();
  Window root();
  unsigned long fg();
  unsigned long bg();
  GC gc();
  GC rgc();
};

Disp::Disp() {
  dsp1 = XOpenDisplay(NULL);
  scr1 = DefaultScreenOfDisplay(dsp1);
  root1 = DefaultRootWindow(dsp1);
  fg1 = BlackPixelOfScreen(scr1);
  bg1 = WhitePixelOfScreen(scr1);
  gc1 = DefaultGC(dsp1, 0);
  XGCValues gcv; gcv.function = GXinvert; gcv.line_width = 3;
  rgc1 = XCreateGC(dsp1, root1, GCFunction|GCLineWidth, &gcv);
}

Display *Disp::dsp() { return dsp1; }
Screen *Disp::scr() { return scr1; }
Window Disp::root() { return root1; }
unsigned long Disp::fg() { return fg1; }
unsigned long Disp::bg() { return bg1; }
GC Disp::gc() { return gc1; }
GC Disp::rgc() { return rgc1; }

class Circle {
private:
  Disp *dsp1;
```

```
  Window win1;
  int radius1, xpos1, ypos1;
public:
  Circle(Disp *d, Window w, int r, int x, int y);
  void draw();
};

Circle::Circle(Disp *d, Window w, int r, int x, int y) {
  dsp1 = d; win1 = w; radius1 = r; xpos1 = x; ypos1 = y;
}

void Circle::draw() {
  XFillArc(dsp1->dsp(), win1, dsp1->gc(),
           xpos1-radius1, ypos1-radius1, radius1*2, radius1*2, 0, 360*64);
}

class Win {
private:
  Disp *dsp1;
  Window win1;
  Counter *cnt1;
  int exit1;
  Circle *objs1[100];
  int objuse1;
public:
  Win(Disp *d, int x, int y, int w, int h, Counter *c);
  void destroy();
  int handle(XEvent *ev);
private:
  void drawall();
};

Win::Win(Disp *d, int x, int y, int w, int h, Counter *c) {
  exit1 = False;
  cnt1 = c;
  dsp1 = d;
  objuse1 = 0;
  win1 = XCreateSimpleWindow(d->dsp(), d->root(),x,y,w,h,2,d->fg(), d->bg());
  XSelectInput(d->dsp(), win1, ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(d->dsp(), win1);
}

void Win::destroy() {
  XDestroyWindow(dsp1->dsp(), win1); exit1 = True;
}

int Win::handle(XEvent *ev) {
  if(exit1 || ev->xany.window != win1) return False;
  if(ev->type == KeyPress) {
    cnt1->add(-1); destroy(); }
  else if(ev->type == Expose)
    drawall();
  else if(ev->type == ButtonPress) {
    objs1[objuse1] = new Circle(dsp1, win1, 20, ev->xbutton.x, ev->xbutton.y);
    objs1[objuse1]->draw();
    ++objuse1; }
  return True;
}
```

```
void Win::drawall() {
  int i;
  for(i = 0; i < objuse1; ++i) objs1[i]->draw();
}

main() {
  Counter *c = new Counter(5);
  Disp *d = new Disp();
  Win *a[5];
  int i;
  for(i = 0; i < 5; ++i) a[i] = new Win(d, 100, 100, 400, 200, c);
  while(c->value() > 0) {
    XEvent ev;
    XNextEvent(d->dsp(), &ev);
    for(i = 0; i < 5; ++i)
      if(a[i]->handle(&ev)) break;
  }
}
```

2.7 別の種類の窓をつくる

さて、ここまでは「丸」を描く窓しかなかったが、もっと別の種類の窓を用意してみよう。たとえば、丸と四角を切り替えたりプログラムを終わらせるのに「スイッチ」のように使える窓を考える。

```
class SwWin {
private:
  Disp *dsp1;
  Window win1;
  char **lines1;
  int lnum1, sel1, exit1, width1, height1;
public:
  SwWin(Disp *d, int x, int y, int w, int h, char **l, int n);
  void destroy();
  int handle(XEvent *ev);
  int sel();
  void drawall();
};
```

つまり、作成するときにメニューとして使う「文字列の配列」を用意し、これを表示する窓を作る。そして、マウスでクリックすることで「選択肢」を変更できる。メンバ関数は次の通り。

```
SwWin::SwWin(Disp *d, int x, int y, int w, int h, char **l, int n) {
  exit1 = False; dsp1 = d; lines1 = l;
  lnum1 = n; sel1 = 0; width1 = w; height1 = h;
  win1 = XCreateSimpleWindow(d->dsp(), d->root(),x,y,w,h,2,d->fg(), d->bg());
  XSelectInput(d->dsp(), win1, ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(d->dsp(), win1);
}

void SwWin::destroy() {
```

```
  XDestroyWindow(dsp1->dsp(), win1); exit1 = True;
}

int SwWin::handle(XEvent *ev) {
  if(exit1 || ev->xany.window != win1) return False;
  if(ev->type == KeyPress) {
    if(++sel1 >= lnum1) sel1 = 0; }
  else if(ev->type == Expose)
    drawall();
  else if(ev->type == ButtonPress) {
    sel1 = ev->xbutton.y * lnum1 / height1;
    if(sel1 >= lnum1) sel1 = lnum1 - 1;
    if(sel1 < 0) sel1 = 0;
    drawall(); }
  return True;
}

int SwWin::sel() {
  return sel1;
}

void SwWin::drawall() {
  int i;
  int d = height1 / lnum1;
  XClearWindow(dsp1->dsp(), win1);
  for(i = 0; i < lnum1; ++i) {
    XDrawImageString(dsp1->dsp(), win1, dsp1->gc(), 4, d*(i+1)-5,
                     lines1[i], strlen(lines1[i])); }
  XFillRectangle(dsp1->dsp(), win1, dsp1->rgc(), 0, d*sel1, width1, d);
}
```

これを使って、丸と四角と「終了」を選択する窓を増やして見た。

```
main() {
  static char *menu[] = { "Circle", "Rectangle", "Quit" };
  Counter *c = new Counter(5);
  Disp *d = new Disp();
  Win *a[5];
  int i;
  SwWin *sw = new SwWin(d, 100, 100, 100, 60, menu, 3);
  sw->drawall();
  for(i = 0; i < 5; ++i) a[i] = new Win(d, 100, 100, 400, 200, c);
  while(sw->sel() != 2 && c->value() > 0) {
    XEvent ev;
    XNextEvent(d->dsp(), &ev);
    if(sw->handle(&ev))
      ;
```

```
      else
        for(i = 0; i < 5; ++i)
          if(a[i]->handle(&ev)) break;
    }
  }
```

練習 8 これを直して、四角も打てるようにするにはどうしたらいいか。

課題 5 お絵描きプログラムを C++の ADT を利用して作成してみよ。