# 卒 業 研 究 報 告

題　　目

## VisualC++6.0 によるゲーム作成

---

指 導 教 員

原 央 教授

---

報 告 者

堀場 敦

---

平成　13 年　2 月　22 日

高知工科大学 電子・光システム工学科

# 目次

# １．ＶｉｓｕａｌＣ＋＋とは何か

## １－１．ＶｉｓｕａｌＣ＋＋とは

ＶｉｓｕａｌＣ＋＋は単なるＣコンパイラではなく、Ｗｉｎｄｏｗｓアプリケーションを作成するための統合環境であると言える。

日本では、Ｗｉｎｄｏｗｓ９５の流行とともに、グラフィック画面をベースとする操作環境が一般のものとして定着した。いわゆるＧＵＩ（Ｇｒａｐｈｉｃａｌ　Ｕｓｅｒ　Ｉｎｔｅｒｆａｃｅ）である。このＧＵＩは、操作が見た目でわかりやすく、ユーザーとしては、非常に取っ付き易いものであると言える。

しかし、そのＧＵＩの登場によって、アプリケーションの作成はとても面倒で手間のかかるものとなってしまった。例えば、画面上に文字を表示したい場合でも、「まず表示するためのウィンドウを作り、それを表示する位置を決め・・」と何十行ものプログラムを別途に必要としていた。

ＶｉｓｕａｌＣ＋＋が登場する以前では、ＧＵＩに対応したアプリケーションを作成するのは困難であり、その負担は全てプログラマにかかっていた。

しかし、ＶｉｓｕａｌＣ＋＋の登場により、プログラマは煩わしい作業から解放された。先ほどの例のように、文字を表示するプログラムをＶｉｓｕａｌＣ＋＋で書こうとする。すると、その命令文は一行で済んでしまうのである。残りのプログラムは、全てＶｉｓｕａｌＣ＋＋が面倒を見てくれる。

すなわち、ＶｉｓｕａｌＣ＋＋はＷｉｎｄｏｗｓアプリケーション作成時に生じる面倒な処理を一手に引き受けてくれるものであり、プログラマは、本来の目的のみに集中して作業できるようになる。

## １－２．ＶｉｓｕａｌＣ＋＋の特徴

ＶｉｓｕａｌＣ＋＋は、プログラミング言語＋Ｗｉｎｄｏｗｓ流インターフェイス開発ツールという形を取っている。つまり、単なるプログラミングだけではなく、プログラムの組み立ての効率化を追求したものとなっているのである。

作成されたプログラムはＣプログラムと違って、メッセージ＋メッセージ関数（メッセージハンドラ）という形で構成される。

メッセージとは、何かの情報や要求を相手に伝えるための構造体やパラメータの集まりのことを言う。何かの動作（操作）が発生すると、ＯＳとアプリケーション間、ウィンドウ間などで、メッセージをやり取りする。ＶｉｓｕａｌＢａｓｉｃではこれを「イベント」と呼ぶ。実行中のプログラムの状態が、ユーザーのマウス操作やキー操作などで変化する

と、ＯＳは対応する相手にメッセージを送る。たとえば、ユーザーがマウスをクリックしたなら、「マウスをクリックした」というメッセージを送ることになる。

メッセージが送られたとき、そのメッセージを受け取り、対応した処理をするための関数をメッセージ関数という。

このことから、ＶｉｓｕａｌＣ＋＋プログラムの構造は、Ｃ言語のような手続き型ではなく、プログラムはプログラマの記述した道筋どおりに走るものではないということが言えるのである。

さらに詳しく構造を説明すると、ＶｉｓｕａｌＣ＋＋プログラムは

・メッセージがこないかいつも見張っている（メッセージテーブルを見ている）

・メッセージが来たら、指定されたメッセージ関数を実行する

という方法で実行される。メッセージテーブルとは、あるメッセージが発行されたときに、どのメッセージ関数を起動するかを指定してくれるものである。

たとえば、ユーザーがマウスをクリックしたら、対応したメッセージが発行される。すると、ＶｉｓｕａｌＣ＋＋プログラムは、メッセージテーブルを見て、マウスクリックに対応する関数があれば、それを起動するというわけである。ＶｉｓｕａｌＣ＋＋プログラムの実行時間のほとんどは「メッセージテーブルをループ監視している」ということになる。そして、ＶｉｓｕａｌＣ＋＋でプログラムを作ることは、おおざっぱに言えば、

・メッセージとメッセージ関数の対応を指定する

・メッセージ関数を記述する

ということの繰り返しとなる。

以下にＶｉｓｕａｌＣ＋＋の特徴を上げる。

＊プログラムの骨格（スケルトン）を作成してくれる機能がある（ＡｐｐＷｉｚａｒｄ）

＊クラスの追加や管理をしてくれる機能がある（ＣｌａｓｓＷｉｚａｒｄ）

＊単一窓（ＳＤＩ）、多重窓（ＭＤＩ）、ダイアログベースのアプリケーションの原型が簡単に作成できる。

＊作成した骨格には、基本的にメニューとツールバーが最初から用意されている。

＊ＭＦＣ（ＭｉｃｒｏｓｏｆｔＦｏｕｎｄａｔｉｏｎＣｌａｓｓ）というクラスライブラリでＷｉｎｄｏｗｓ処理を行う。

＊プログラムのドキュメントとその処理部分を別管理する（ドキュメントクラス）

＊固定的なデータはリソースとしてプログラム本体と分割管理する。

＊デバッグ機能が充実している。

＊Ｗｉｎｄｏｗｓ上で運用するための有用な機能が豊富に用意されている。

# ２．ＶｉｓｕａｌＣ＋＋６．０を使ったゲーム作成。

ＶｉｓｕａｌＣ＋＋を使って、トランプゲーム“ハーツ”を作成する。
ハーツとはどういうゲームであるか、またそれのプログラムを組む手順はどうなっているかを説明する。

## ２－１．ゲームの解説

### トランプゲーム“ハーツ”のルールと、それを再現するプログラムの手順

**ハーツのルール**

ハーツの目的
ハーツの目的は、自分の得点をなるべく低くおさえ、ゲームの終了時点の得点を全プレーヤー中最低にすることである。

ハーツの手順
ほかのプレーヤーに渡すカードを 3 枚選ぶ。ただし、4 回に 1 回、カードを交換しないゲームがある。
ゲームを始めるのは、クラブの 2 を持っているプレーヤーである。
時計回りに 1 人ずつ、カードを場に出していく。通常は、台札と同じスートのカードを出す必要があるが、同じスートのカードを持っていない場合は、どのカードを出してもかまわない。
台札と同じスートで最高のカードを出したプレーヤーが、場に出されたカード（トリックと言う）を取る。トリックを取ったプレーヤーが次の回の台札を出すことになる。

用語
スート：トランプの端についている、ハートやスペードなどのマークのこと。
台札：ゲームの一巡で、一番初めに出されたカードのこと。
トリック：場に出されたカードのこと。

ハーツの得点
取ったハート 1 枚につき 1 点、スペードのクイーンには 13 点が加算されます。
いずれかのプレーヤーの得点が 100 点になるか、または親がゲームを終了するまでゲーム

を繰り返します。
1 回のゲームでハートのカードすべてとスペードのクイーンを取ると ("シュート ザ ムーン" といいます)、自分には得点が加算されず、ほかのプレーヤー全員に 26 点ずつ加算させることができます。

**ゲームを再現するために必要なプログラム手順**

１．５２枚のトランプをシャッフルする。
２．プレイヤーとコンピュータ合わせて４人でゲームをする。そのため、１テーブルに５２の４分割である１３枚を渡し、左からクラブ、ダイヤ、スペード、ハートの順にまとめ、数字の小さい順に並べて描画する。
３．ゲームの前準備として、ほかのプレイヤーとカードを３枚交換する。
４．プレイヤー以外の３人がクラブの２を持っている場合、中央の台にそれを出すようにする。
５．その後、時計回りにカードを規定の条件にしたがって出す。
６．プレイヤーの順番になったら、ボタンが押されるまでプログラムを停止する。
７．プレイヤーがカードを出し、台に４枚カードが出たら、その中で条件に合った最も大きなカードを選び、それを出した者の得点版に、既定の条件に従い、加点する。
８．前の一巡でトリックを取ったものが台札を出す。以下カードがなくなるまで繰り返す。
９．全て出し終わったところで、そのゲームの最終得点の一覧を表記し、順位を出す。
１０．　１に戻り、２ゲーム目に移行する。
１１．　８ゲーム目が終了するか、誰か一人が１００点を越えるまでゲームを続けるようにする。上の条件を満たした場合、最終順位を表記して終了する。

## ２－２．プログラムの解説

**これから、順にゲームを再現するプログラムの流れを解説していく。**

**１．５２枚のトランプをシャッフルする。**

```
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
cou = 0; fm = 0; dan1 = 0; dan2 = 0; dan3 = 0; dan4 = 0; syu = 0;
iti = 100; sut = 3; ra = 13;
for(i=0; i<=3; i++){
        ten[i] = 0; jyu[i] = 0; cha[i] = 0;
}
for(i=0; i<=12; i++){
        sel[i] = -1; swi[i] = 0; tch[i+1] = 0;
}
for(i=0; i<=51; i++){
        toric[i] = 0;
}
sprintf(sa,"%d",0);
m_ten1.SetWindowText(sa);
sprintf(sa,"%d",0);
m_ten2.SetWindowText(sa);
sprintf(sa,"%d",0);
m_ten3.SetWindowText(sa);
sprintf(sa,"%d",0);
m_ten4.SetWindowText(sa);
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
```

**プログラム．１－１**

**まず、ゲームに使用する変数の初期化を行う。ここで行っている初期化は、１ゲーム毎に初期化せねばならない変数である。各変数の説明は、それが使われたときのプログラムで説明する。ｆｉｒｓｔ関数は、１ゲーム毎の最初の処理部分なので、変数の初期化はここで行うことにする。**

```
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
int syaf;
for(i=0; i<=51; i++){
        card[i]=i+1;
}
for(i=0; i<=51; i++){
        ran();
        syaf = randam;
        sab = card[i]; card[i] = card[syaf]; card[syaf] = sab;
}
sort();
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
```

**プログラム．１－２**

**配列ｃａｒｄは５２枚のカードの情報を扱うためのものである。それぞれのカードには数字が与えられている。それを利用して、配列ｃａｒｄの中に順番にカード番号を入れていく。次に、その順番に並んだ番号をランダムに並べ替える。**
**ランダム関数を使うと、変数ｒａｎｄａｍには０から５１のどれかの値が入力されることになる。ｆｏｒ文を使って、ｉの値のときの配列ｃａｒｄの中身を、ｒａｎｄａｍの値を与えられた変数ｓｙａｆの値のときの配列ｃａｒｄの中身と入れ替える。これを複数回繰り返して、中身をランダムになるようシャッフルする。そしてｓｏｒｔ関数を通して、手持ちの１３枚のカードをスート順に並べ直す。**

```
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
k = 0;
for(i=0; i<=12; i++){
        min = card[i];
        for(g=i; g<=12; g++){
                if (min >= card[g]){
                        min = card[g]; k = g;
                }
        }
        sab = card[i]; card[i] = min; card[k] = sab;
}
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
```

**プログラム．１－３**

上はｓｏｒｔ関数の中身である。配列ｃａｒｄの中は、１－２でシャッフルされており、スートの並び方もバラバラになっている。最初にカードに番号を与えていたとき、スート毎に与えていたことをここで利用する。（クラブの２は１、クラブの３は２・・クラブのＡは１３、ダイヤの２は１４・・）つまり、数の順に並べていけば、スートも自動的にそろうということである。

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
game++;
if(game == 1 || game == 5){ mes="カード３枚を選んでください。左側に渡します。"; }
if(game == 2 || game == 6){ mes="カード３枚を選んでください。向かいに渡します。"; }
if(game == 3 || game == 7){ mes="カード３枚を選んでください。右側に渡します。"; }
if(game == 4 || game == 8){ mes="ゲームを開始します。"; }
sprintf(sa,"%s",mes);
m_mess.SetWindowText(sa);
dispflg=1; Invalidate();
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

プログラム．１－４

プログラム１－２の続きである。
変数ｇａｍｅは、今が何ゲーム目かをあらわすためのものである。初期値は０のため、このｇａｍｅ＋＋でｇａｍｅは１になり、これ以降の命令を１ゲーム目の処理として行うことになる。画面の一番下のメッセージボックスに、プレイヤーへの忠告メッセージを入力したら、描画処理用変数ｄｉｓｐｆｌｇに値を与える。ｄｉｓｐｆｌｇの値が０の時、条件は偽となり、値がある時、条件が真になる。そして、Ｉｎｖａｌｉｄａｔｅという命令で、ＯｎＤｒａw関数に飛ぶことになる。

２．１つのテーブルにカードを１３枚ずつ渡し、スート順に並べて描画する。

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
if(dispflg){
        for(i=0; i<=12; i++){
                if(sel[i] != i){
                        CDC myDC; CBitmap myBMP;
                        CDC* pDC=m_pict.GetDC();
```

```
                    myBMP.LoadBitmap(card[i]);
                    myDC.CreateCompatibleDC(pDC);
                    CBitmap* oldBMP=myDC.SelectObject(&myBMP);
                    pDC->BitBlt(i*26+6,6,i*26+81,86,&myDC,0,0,SRCCOPY);
                    myDC.SelectObject(oldBMP);
                }
        }
                    :
                    :
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．２－１**

**上は描画処理を行うプログラムである。ｉｆ内のｄｈｉｓｐｆｌｇは１－２で説明したとおり値は真となっているいので、条件を満たしている。**
**５行目のｍｙＢＭＰＬｏａｄＢｉｔｍａｐの括弧内に入っているのが、画面に描画される内容である。ｆｏｒ文を利用して、一枚ずつカードを描画していくことになる。８行目のＢｉｔＢｌｔ関数で表示位置を操作して、左から一枚ずつ半分重なるようにして表示していく。ｃａｒｄの配列番号０～１２の中身を表示し終えたら、１３枚のカードが画面に現れる。同じような処理を、あと３回行って、全員分の手札を用意するわけだが、ここではその部分は省略してある。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
if(aro == 3){
        CDC myDC; CBitmap myBMP;
        CDC* pDC=m_picmain.GetDC();
        myBMP.LoadBitmap(ga1);
        myDC.CreateCompatibleDC(pDC);
        CBitmap* oldBMP=myDC.SelectObject(&myBMP);
        pDC->BitBlt(65,90,140,170,&myDC,0,0,SRCCOPY);
        myDC.SelectObject(oldBMP);
        byo = 1;
}
if(aro == 2 || byo == 1){
        CDC myDC; CBitmap myBMP;
```

```
            CDC* pDC=m_picmain.GetDC();
            myBMP.LoadBitmap(ga2);
            myDC.CreateCompatibleDC(pDC);
            CBitmap* oldBMP=myDC.SelectObject(&myBMP);
            pDC->BitBlt(105,50,180,130,&myDC,0,0,SRCCOPY);
            myDC.SelectObject(oldBMP);
            byo = 2;
}
if(aro == 1 || byo == 2){
            CDC myDC; CBitmap myBMP;
            CDC* pDC=m_picmain.GetDC();
            myBMP.LoadBitmap(ga3);
            myDC.CreateCompatibleDC(pDC);
            CBitmap* oldBMP=myDC.SelectObject(&myBMP);
            pDC->BitBlt(145,90,220,170,&myDC,0,0,SRCCOPY);
            myDC.SelectObject(oldBMP);
}
byo = 0;
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．２－２**

**ゲーム実行中に、ゲームとは関係無いウィンドウを開き、またゲーム画面に戻ってきたとする。すると、ＶｉｓｕａｌＣ＋＋は、自動的にＯｎｐａｉｎｔ関数を実行し、画像表示をクリアしてしまうのである。上のプログラムは、それを防ぐためのものである。**
**自動的にＯｎｐａｉｎｔ関数が実行されると言うことは、その中にあるプログラム２－１、２－２が実行されると言うことでもある。よって、２－１が実行されるのだから、それぞれの手持ちのカードは自動的に表示される。問題は、ゲーム中に、カードを台の上に出していた場合である。**
**変数ａｒｏは、後述するがゲームの一巡の何番目にカードを出したかを調べる変数である。画面が復帰したとき、この変数ａｒｏの数によって、何枚表示されていたかがわかるようになっている。ａｒｏが３ならば、画面には３枚表示されており、１ならば、１枚だけ表示されていたことになる。よって、上から順に条件文にあてはめ、元の画像を表示していくことになる。**

**３．他のプレイヤーとカードを３枚交換する。**

```
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
if(fm == 0){
        if(swi[1] == 0 && cou < 3){
                rev[0] = 1; swi[1]++; cou++;
        }
        else if(swi[1] == 1){ rev[0] = 0; swi[1] = 0; cou--; }
        dispflg2 = 1; pict();
}
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
```

**プログラム．３－１**

**ボタン１をクリックしたときの処理。**
**準備段階かどうかを判定する変数ｆｍの値を条件にする。ｆｍが０の時、初期段階であるとして、ゲームの最初の段階である、『３枚を選んで特定の相手に渡す』という処理を行う命令になる。**
**配列ｓｗｉ、ｒｅｖ、ｉｎｔ変数ｃｏｕの初期値はすべて０である。これをふまえて、３枚選んだ時点で決定ボタンを押した時、３枚交換すると言う処理を行うようにする。そうでないときは決定ボタンを押しても反応はない。また、取り消したい場合もあると考え、もう一度同じボタンをクリックすれば、選択を取り消せるようにする。**
**配列ｒｅｖの値が１のとき、選択したカードを反転表示させる。そのため、一度ボタンを押された時はｒｅｖの値を１とし、もう一度押された場合は０に戻すこととする。**
**配列ｓｗｉはボタンが押されたのが何回目かを判定するための変数である。ｓｗｉが０のとき、カードが選択されていない状態と言うことなので、ｒｅｖを１とし、ｓｗｉの値を１とする。ｓｗｉの値が１だったときは、ｒｅｖ、ｓｗｉの値を０とする。**
**変数ｃｏｕは交換するカードが何枚選択されたかを判定する変数である。ｒｅｖが１になった時ｃｏｕを＋１、ｒｅｖが０になったときｃｏｕを－１する。ｃｏｕが３の状態の時に、決定ボタンを押せばカード交換プログラムが開始される。**
**ｄｉｓｐｆｌｇという描画の指定用変数の２番目の値を１にして、その中の内容を実行する。**

```
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
if(dispflg2){
        for(i=0; i<=12; i++){
                CDC myDC; CBitmap myBMP;
                CDC* pDC=m_pict.GetDC();
```

```
                myBMP.LoadBitmap(card[i]);
                myDC.CreateCompatibleDC(pDC);
                CBitmap* oldBMP=myDC.SelectObject(&myBMP);
        if(rev[i]==1){pDC->BitBlt(i*26+6,6,i*26+81,86,&myDC,0,0,NOTSRCCOPY); }
                else{ pDC->BitBlt(i*26+6,6,i*26+81,86,&myDC,0,0,SRCCOPY); }
                myDC.SelectObject(oldBMP);
        }
        dispflg2 = 0;
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．３－２**

**ｄｉｓｐｆｌｇ２＝１のとき、上のプログラムが実行される。これは、カードを選択されたかどうかがわかるように表示する内容のものである。**
**ここで、８行目、ＢｉｔＢｌｔ関数の最後のＮＯＴＳＲＣＣＯＰＹというのは、反転表示をするという命令である。**
**ｒｅｖが１の時反転表示をし、そうでないときはそのままの内容を表示する。**
**最後にｄｉｓｐｆｌｇ２の値を０にし、後でｐｉｃｔ関数に飛んだ時に、間違ってこの命令が実行されることのないようにする。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
if(cou == 3){
        k = 0;
        if(game == 1 || game == 5){
                koukan();
                for(i=0; i<=12; i++){
                        if(rev[i] == 1){
                                trade = kou[k]+13;
                                tra = card[i];
                                card[i] = card[trade];
                                card[trade] = tra;
                                k++;
                        }
                }
        }
                        :
```

```
                    :
        cou = 9;
        sort();
        for(i=0; i<=12; i++){
                for(j=0; j<=2; j++){
                        if(card[i] == reva[j]){ rev[i] = 1; }
                        else{ rev[i] = 0; }
                }
        }
        dispflg3 = 1;
        pict();
        mes="ゲームを開始します。";
        sprintf(sa,"%s",mes,);
        m_mess.SetWindowText(sa);
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．３－３**

**ｃｏｕ＝３の状態で、決定ボタンを押すと、上のプログラムが実行される。最初に何ゲーム目であるかを調べ、それによってどの相手とカードを交換するかを決定する。交換した後、カードを順に並べ替える。**
**（ここでｋｏｕｋａｎ関数に飛んで処理することになる。詳しい説明は２－３．コンピュータの思考ルーチンで行う。）**
**表示関数ｄｉｓｐｆｌｇの３番目に値を加え実行可能にし、下の注釈文ボックスにゲーム開始の宣言文を書きこむ。**
**もう一度決定ボタンを押すと、交換処理は終了し、ゲームの本番へと移行する。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
for(i=0; i<=12; i++){
        min = card[i];
        for(g=i; g<=12; g++){
                if (min >= card[g]){
                        min = card[g]; k = g;
                }
        }
        sab = card[i]; card[i] = min; card[k] = sab;
```

```
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．３－４**

**カードを数字順に並べ替えるプログラムである。カードは、弱いカード順に数字番号を与えてあるので、ｆｏｒ文と配列を利用して簡単に入れ替えを行える。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
if(cou == 10){
        for(i=0; i<=12; i++){
                rev[i] = 0;
        }
        for(i=0; i<=51; i++){
                if(card[i] == 1){ iti = i; }
        }
        dispflg3 = 1;
        pict();
}
if(cou == 9){ cou++; }
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．３－５**

**プログラム３－３のすぐ後につながるプログラムで、Ｏｎｋｅｔｔｅｉ関数の一部である。３－３で、ｃｏｕ＝９の条件を満たしているため、ｃｏｕは１０となる。そのため、もう一度決定ボタンを押すと、ｃｏｕ＝１０の条件が満たされているため、上の内容が実行される。**
**配列ｒｅｖの値を全て初期化する。その後、配列ｃａｒｄの中からクラブの２が格納されている箱の番号を探し出す。（クラブの２にあてられた認識番号は１である）見つけた各のう番号を変数ｉｔｉに代入する。**
**描画用変数ｄｉｓｐｆｌｇの３番目に値を与え実行可能にし、ゲーム本番に移行する。**

**４．プレイヤー以外の３人がクラブの２を持っている場合、中央の台にそれを出させる。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
if(iti/13 == 1){
```

```
        CWnd* myPICT=GetDlgItem(IDC_PICT2);
        CClientDC myDC(myPICT);
        CRect myRECT;
        myPICT->GetClientRect(myRECT);
        myDC.FillSolidRect(myRECT, RGB(0,150,0));
        for(i=0; i<=12; i++){
                if(i != iti-13){
                        CDC myDC; CBitmap myBMP;
                        CDC* pDC=m_pict2.GetDC();
                        myBMP.LoadBitmap(card[i+13]);
                        myDC.CreateCompatibleDC(pDC);
                        CBitmap* oldBMP=myDC.SelectObject(&myBMP);
                        pDC->BitBlt(5,i*20+6,80,i*20+86,&myDC,0,0,SRCCOPY);
                        myDC.SelectObject(oldBMP);
                }
        }
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．４－１**

**ｄｉｓｐｆｌｇ３の内容の一部。プログラム３－３で、ｉｔｉの値が０～５１の間の一つに定まっている。そのｉｔｉを１３で割り、その商が０～３のどれになるかで、誰がクラブの２を持っているのかを判断する。**
**例として、ｉｔｉ＝１８の時、１３で割った商は１となり、ＣＯＭＰ１がクラブ２を持っているということがわかる。そのとき、上のプログラムの条件を満たしており、内容を実行することになる。**
**ＣＷｎｄからの４行は、ＣＯＭＰ１の台を台の色で塗りつぶすと言う命令である。一度何も表示されていないように見える状態にした後で、その上に最描画することとする。一度選ばれたはずのカードが消えずに台の上に残るのを防ぐための処置である。**
**そして、塗りつぶした後の再描画の時に、中央台に出すカードは取り除いておかなければならない。そこで、配列ｃａｒｄのクラブの２が格納されている番号とｉｔｉが一緒の時、そのカードの描画はしないという条件文を挿入する。それ以外の場合は、問題なく描画処理させる。これにより、あたかもカードが抜き出されたかのように見せることができる。**
**そして、中央台にクラブの２を表示する。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
CDC myDC; CBitmap myBMP;
CDC* pDC=m_picmain.GetDC();
myBMP.LoadBitmap(card[iti]);
myDC.CreateCompatibleDC(pDC);
CBitmap* oldBMP=myDC.SelectObject(&myBMP);
fm = -1;
if(iti/13 == 0){  fm = 1;  mes="クラブの２を出してください。";
sprintf(sa,"%s",mes); m_mess.SetWindowText(sa); }
if(iti/13 == 1){ pDC->BitBlt(95,106,170,186,&myDC,0,0,SRCCOPY); jyu[1] = 1; kei[1] =
card[iti]; ban[1] = 2; }
if(iti/13 == 2){ pDC->BitBlt(135,66,210,146,&myDC,0,0,SRCCOPY); jyu[2] = 1; kei[1] =
card[iti]; ban[1] = 3; }
if(iti/13 == 3){ pDC->BitBlt(175,106,250,186,&myDC,0,0,SRCCOPY); jyu[3] = 1; kei[1]
= card[iti]; ban[1] = 4; }
myDC.SelectObject(oldBMP);
if(iti/13 != 0){ card[iti] = -1; aro = 1; }
dispflg3 = 0; dispflg5 = 1;
und = aro;
fir = (kei[und]-1)/13;
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．４－２**

**プログラム４－１の時と同様に、ｉｔｉを１３で割ってその商を見てクラブの２を持っているのが誰かを見極める。それぞれ出す位置が定まっているからである。ＢｉｔＢｌｔ関数で表示位置を決定する。またそのときに、これ位後のゲームを進めるための配列ｊｙｕ、ｋｅｉ、ｂａｎ、変数ａｒｏ、ｕｎｄ、ｆｉｒの値を決定しておく。また、配列ｃａｒｄのｉｔｉ番目の内容を－１に置き換え、今後使用されることのないようにする。ｄｈｉｓｐｆｌｇ３の値を初期化し、ｄｉｓｐｆｌｇ５に値を与え、次のゲーム処理へ移行させる。**

**５．その後、時計回りにカードを規定の条件にしたがって出していく。**

**もし、クラブの２を出したのがＣＯＭＰ１だった場合、カードを出す順は時計回りにＣＯＭＰ２、ＣＯＭＰ３、ＰＬＡＹＥＲの順になる。そのため、クラブの２が出されたら、次の人がカードを出すという処理をせねばならない。**

```
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
if(aro == 4){ aro = 0; point(); }
if(jyu[0] == 1){
        comp(); Sleep(500);
        CWnd* myPICT=GetDlgItem(IDC_PICT2);
        CClientDC mDC(myPICT);
        CRect myRECT;
        myPICT->GetClientRect(myRECT);
        mDC.FillSolidRect(myRECT, RGB(0,150,0));
        for(i=0; i<=12; i++){
                if(!(i == cpu || card[i+13] == -1)){
                        CDC myDC; CBitmap myBMP;
                        CDC* pDC=m_pict2.GetDC();
                        myBMP.LoadBitmap(card[i+13]);
                        myDC.CreateCompatibleDC(pDC);
                        CBitmap* oldBMP=myDC.SelectObject(&myBMP);
                        pDC->BitBlt(5,i*20+6,80,i*20+86,&myDC,0,0,SRCCOPY);
                        myDC.SelectObject(oldBMP);
                }
        }
        CDC myDC; CBitmap myBMP;
        CDC* pDC=m_picmain.GetDC();
        myBMP.LoadBitmap(card[cpu+13]);
        myDC.CreateCompatibleDC(pDC);
        CBitmap* oldBMP=myDC.SelectObject(&myBMP);
        pDC->BitBlt(95,106,170,186,&myDC,0,0,SRCCOPY);
        myDC.SelectObject(oldBMP);
        jyu[0] = 0; jyu[1] = 1; aro++; und = aro;
        kei[und] = card[cpu+13];
        ban[und] = 2;
        if(aro == 1){ fir = (kei[und]-1)/13; }
        card[cpu+13] = -1;
}
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
```

**プログラム．５－１**

変数ａｒｏは、台に何枚カードが出されているかを判断する変数である。ａｒｏ＝４のとき、全員が１枚ずつ出したときだから、得点算出の関数ｐｏｉｎｔ（）へ飛ぶことになる。
配列ｊｙｕは、時計順に処理を行うために用意されている。
配列、ｋｅｉ、ｂａｎは後のｐｏｉｎｔ関数で使用することになる。
また、ここで変数ｆｉｒの存在が重要になっている。ｆｉｒは、一巡の最初に出されたカードのスートが何であったかを記録しておく変数である。ｐｏｉｎｔ関数を初め、コンピュータの思考ルーチンであるｃｏｍｐ関数などでも使用する。
台に出してしまった配列ｃａｒｄには－１をいれておく。

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
（ｃｏｍｐ関数は、２－３．コンピュータの思考ルーチンで詳しく解説する）
＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

６．プレイヤーの順番になったら、ボタンが押されるまでプログラムを停止する。

プログラム５の処理を行い、時計回り順なのでＣＯＭＰ３がカードを出したところでプレイヤーの順番が回ってくる。ここで、プログラムはユーザーの操作待ち状態に入る。
１～１３のボタンのどれかを押し、カードを台に出すことになる。

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
if(fm < 0){
        if(tch[1] == 0){
                sel[0] = 0;
                if(aro != 0){ suto(); }
                else{ dispflg4 = 1; tch[1] = 1; pict(); }
        }
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

プログラム．６－１

最初のカード交換時に、ｆｍ＝－１となっており、ＯｎＢｕｔｔｏｎ関数はゲーム本番用のプログラムが使用できるようになっている。
配列ｓｅｌには、配列番号と同じ番号を与える。描画処理ｄｈｉｓｐｆｌｇ４で使用する。
変数ａｒｏの値が０でないとき、スートを比べて正しいかどうか調べる関数に飛ぶことに

なる。ａｒｏ＝０ならば、４人の中で最初に出すということだから、スートを気にする必要はなく、ｄｈｉｓｐｆｌｇ４に値を与えて描画処理に移ることができる。

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
int aru;    aru = 0;
for(i=0; i<=12; i++){
        if((card[i]-1)/13 == fir){
                aru = 1;
        }
}
for(i=0; i<=12; i++){
if(sel[i] == i){
        if(aru == 0){
                dispflg4 = 1; tch[i+1] = 1; pict();
        }
        else{
                if((card[i]-1)/13 == fir){
                        dispflg4 = 1; tch[i+1] = 1; pict();
                }
                else{
                        if(fir == 0){ mes = "クラブのカードを出してください。"; }
                        if(fir == 1){ mes = "ダイヤのカードを出してください。"; }
                        if(fir == 2){ mes = "スペードのカードを出してください。"; }
                        if(fir == 3){ mes = "ハートのカードを出してください。"; }
                        sprintf(sa,"%s",mes);
                        m_mess.SetWindowText(sa);
                        sel[i] = -1;
                }
        }
}
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．６－２**

上はプレイヤーが最初にカードを出す場合でなかったとき、押したボタンの場所のカードが、スートの条件を満たしているかどうかを判断するプログラムである。

**まず、手持ちの札全てとｆｉｒを見比べる。もし、ｆｉｒと同じスートが手札にないのならば、条件を気にすることなくどんなカードを出してもかまわないことになる。**
**ｆｉｒと同じスートを持つものが手札に合ったとき、条件を満たすようなカードを出していたならば、問題なく描画処理に移行する。満たしていなかった場合は、メッセージボックスに、どのスートのカードを出せばいいのかを表示する。そして、ボタンが押されたという状態をキャンセルするために、配列ｓｅｌを－１の状態に戻しておく。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
if(dispflg4){
        CWnd* myPICT=GetDlgItem(IDC_PICT);
        CClientDC myDC(myPICT);
        CRect myRECT;
        myPICT->GetClientRect(myRECT);
        myDC.FillSolidRect(myRECT, RGB(0,150,0));
        for(i=0; i<=12; i++){
                if(sel[i] != i){
                        CDC myDC; CBitmap myBMP;
                        CDC* pDC=m_pict.GetDC();
                        myBMP.LoadBitmap(card[i]);
                        myDC.CreateCompatibleDC(pDC);
                        CBitmap* oldBMP=myDC.SelectObject(&myBMP);
                        pDC->BitBlt(i*26+6,6,i*26+81,86,&myDC,0,0,SRCCOPY);
                        myDC.SelectObject(oldBMP);
                }
        }
        aro++;    und = aro;
        for(i=0; i<=12; i++){
                if(sel[i] == i){
                        CDC myDC; CBitmap myBMP;
                        CDC* pDC=m_picmain.GetDC();
                        myBMP.LoadBitmap(card[i]);
                        myDC.CreateCompatibleDC(pDC);
                        CBitmap* oldBMP=myDC.SelectObject(&myBMP);
                        pDC->BitBlt(135,146,210,226,&myDC,0,0,SRCCOPY);
                        myDC.SelectObject(oldBMP);
                        kei[und] = card[i];
```

```
                    card[i] = -1;
                }
            }
            ban[und] = 1;
            if(aro == 1){ fir = (kei[und]-1)/13; }
            jyu[0] = 1;    dispflg4 = 0;    dispflg5 = 1;
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．６－３**

**プログラム４－１の時と同様に、まず塗りつぶしの処理が行われる。配列ｓｅｌの値を条件に表示しないカードや、台に出すカードを選抜する。後はプログラム５－１と同様、ａｒｏ＝１ならばｆｉｒを設定し、ｐｏｉｎｔ用にｋｅｉ、ｂａｎを設定しておく。そして、ＣＯＭＰ１に順番を映すために配列** jyu[0] **に１を与え、ｄｉｓｐｆｌｇ５に値を与える。**

**７．台に４枚カードが出揃ったら、条件に従い点数を加える。**

**台に４枚のカードがそろったとき、ａｒｏ＝４となり、ｐｏｉｎｔ関数に飛ぶことになる。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
Sleep(1500);
CWnd* myPICT=GetDlgItem(IDC_PICMAIN);
CClientDC myDC(myPICT);
CRect myRECT;
myPICT->GetClientRect(myRECT);
myDC.FillSolidRect(myRECT, RGB(0,150,0));
max = kei[1];
if(fir == (kei[2]-1)/13){
        if(max < kei[2]){ max = kei[2]; }
}
if(fir == (kei[3]-1)/13){
        if(max < kei[3]){ max = kei[3]; }
}
if(fir == (kei[4]-1)/13){
        if(max < kei[4]){ max = kei[4]; }
```

```
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．７－１**

**上はｐｏｉｎｔ関数の初めの部分である。ｓｌｅｅｐ関数は待機状態を作る命令である。この場合、約１．５秒の間プログラムは停止し、台に置かれたカードを見ることができるようになっている。待機が終わったら、台の部分を塗りつぶし、次の一巡ができるように準備しておく。**

**次に得点計算を行う。ハーツのルールでは、最も強いカードを出したものがトリックをとらされることになっている。しかし、一巡の最初に出たカードのスートと異なるカードを出していれば、大きさは全く関係ない。これを踏まえて、配列ｋｅｉの１番目をｍａｘに当てる。このｋｅｉは、一巡の一番最初に出されたカードの番号である。次に、ｋｅｉの２番目以降が変数ｆｉｒと同じスートになっているかどうかを判定する。もし、同じスートであった場合、ｍａｘの値と比べ、より大きかった場合、その数が新たにｍａｘとなる。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
ban[0] = 0; hart = 0;
for(i=1; i<=4; i++){
        tor[i] = 0;
        if((kei[i]-1)/13 == 3){
                hart++; tor[i] = kei[i];
        }
        if(kei[i] == 37){ hart = hart+13; tor[i] = kei[i]; }
}
for(i=1; i<=4; i++){
        if(max == kei[i]){
                ban[0] = ban[i];
        }
}
if(ban[0] == 1){
        ten[0] = ten[0] + hart;
        sprintf(sa,"%d",ten[0]);
        m_ten1.SetWindowText(sa);
        for(i=1; i<=4; i++){
                if(tor[i] != 0){
                        toric[dan1] = tor[i]; dan1++;                    }
```

```
                    }
            }
}
syu++;
if(syu == 13){
            if(ten[0] == 26){ ten[0] = 0; ten[1] = 26; ten[2] = 26; ten[3] = 26; }
            if(ten[1] == 26){ ten[0] = 26; ten[1] = 0; ten[2] = 26; ten[3] = 26; }
            if(ten[2] == 26){ ten[0] = 26; ten[1] = 26; ten[2] = 0; ten[3] = 26; }
            if(ten[3] == 26){ ten[0] = 26; ten[1] = 26; ten[2] = 26; ten[3] = 0; }
            syouhai();
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．７－２**

**７－１に続いて、ｐｏｉｎｔ関数の一部である。７－１で誰に加点されるのかが決定された。続いて、何点加えられるのかを決めなくてはならない。**
**変数ｈａｒｔは、台に出されたカードに、ハートのカードが何枚あったかをわかるようにする変数である。配列ｔｏｒは、１ゲームの終了時に取ってしまったトリックを画面に表示するためのものである。また、スペードのクイーンがでた時は、１３点加点されるので、ｋｅｉに３７（スペードのクイーンに与えられた数字）が与えられていたとき、加点する。**
**配列ｂａｎの０は、最終的に加点される対象が誰かを決めるために用意されている。プログラム５－１、６－３でｂａｎには値が与えられており、その値を比べることによって、加点の対象が決定する。**
**追加された得点は、それぞれのエディットボックスに表示される。そのときに、配列ｔｏｒｉｃにそれぞれが取ってしまったトリックがなんであるかを覚えさせる。**
**変数ｓｙｕは、このゲーム内で何回一巡したかを調べるものである。もし、この変数が１３になったとき、全てのカードを出し終えているはずだから、１ゲームの終了の処理へと移ることになる。そのとき、ハーツのルールに従い、“シュートザムーン”が起こったかどうかを判定する。起こっていた場合、点数をルールに従って配分し、ｓｙｏｕｈａｉ関数へ飛ぶ。**

**８．前の一巡でトリックを取った者が台札を出す。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
else{
```

```
        if(fir == 4){ fir = 3; }
        else{ fir++; }
        if(ban[0] == 1){
                jyu[0] = 0; jyu[1] = 0; jyu[2] = 0;
        }
        else{
                for(i=2; i<=4; i++){
                        if(ban[0] == i){ jyu[i-2] = 1; }
                        else{ jyu[i-2] = 0; }
                }
        }
        pict();
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．８－１**

**説明のために分割したｐｏｉｎｔ関数の最後の部分である。**
**変数ｓｙｕが１３でなかった場合、ゲームを続けなければならない。ハーツのルールに従うと、一巡した後に札を最初に出すのは、トリックを取ったものと決められている。そのため、配列ｊｙｕの値を変化することで、誰が最初にカードを出すかを決めることができる。ＰＬＡＹＥＲがトリックを取った場合は配列ｊｙｕを全て０にすることで、相手の処理を飛ばして自分の手番にできる。トリックを取ったのがＰＬＡＹＥＲでなかった場合、その配列番号ｊｙｕを１にすることで、そのトリックを取った者が最初にカードを出すことになる。**
**ｐｉｃｔ関数に飛び、変数ｓｙｕが１３になるまで、プログラム５～７を繰り返す。**

**９．カードを全て出し終えたところで、そのゲームの最終得点の一覧と順位を表示する。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
dispflg6 = 1; Invalidate();
for(i=1; i<=4; i++){
        tot[(game-1)*4+i] = tot[(game-1)*4+i] + ten[i-1];
}
if(game >= 2){
```

```
            tot[(game-1)*4+1] = tot[(game-1)*4+1] + tot[(game-2)*4+1];
            tot[(game-1)*4+2] = tot[(game-1)*4+2] + tot[(game-2)*4+2];
            tot[(game-1)*4+3] = tot[(game-1)*4+3] + tot[(game-2)*4+3];
            tot[(game-1)*4+4] = tot[(game-1)*4+4] + tot[(game-2)*4+4];
}
nu[0] = tot[(game-1)*4+1]; nu[1] = tot[(game-1)*4+2];
nu[2] = tot[(game-1)*4+3]; nu[3] = tot[(game-1)*4+4];
k = 0;
for(i=0; i<=3; i++){
            min = nu[i];
            for(g=i; g<=3; g++){
                        if (min >= nu[g]){
                                    min = nu[g]; k = g;
                        }
            }
            sab = nu[i]; nu[i] = min; nu[k] = sab;
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．９－１**

**ｄｈｉｓｐｆｌｇの６に値を与え、描画処理を行う。この時は、一度画面を初期化してから描画処理を行うため、ｐｉｃｔ関数ではなく、ＩｎｖａｌｉｄａｔｅでＯｎＰａｉｎｔ関数に飛ぶ。ここで、それぞれが取ってしまったトリックを表示する。**
**配列ｔｏｔは総合得点を入れておくためのものである。ゲームが進むたびに加点されるように計算式を作る。ＰＬＡＹＥＲが使用するｔｏｔの配列番号は、１、５、９と４ａ＋１の式であらわされる。同様に、ＣＯＭＰ１は４ａ＋２、ＣＯＮＭＰ２は４ａ＋３となる。**
**次に、導かれたｔｏｔの値を配列ｎｕに順番に入れる。配列ｎｕは最終的にプレイヤーの順位がどうなるったかを導くためのものである。ｎｕの中を昇順に並び替えておく。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
Cjyuni myDL;
if(game >= 1){ myDL.m_g1t1=tot[1]; myDL.m_g1t2=tot[2]; myDL.m_g1t3=tot[3];
                                    :
                                    :
if(game >= 8){ myDL.m_g8t1=tot[29]; myDL.m_g8t2=tot[30]; myDL.m_g8t3=tot[31];
myDL.m_g8t4=tot[32]; }
```

```
if(tot[(game-1)*4+1] == nu[0]){ mes="PLAYER は１位です"; }
if(tot[(game-1)*4+1] == nu[1]){ mes="PLAYER は２位です"; }
if(tot[(game-1)*4+1] == nu[2]){ mes="PLAYER は３位です"; }
if(tot[(game-1)*4+1] == nu[3]){ mes="PLAYER は４位です"; }
myDL.m_jyuni=mes;
myDL.DoModal();
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．９－２**

**１ゲームの得点、ゲームの総合得点、現在の順位を新しいウィンドウを出して、そこに表示する。**

**ＣｊｙｕｎｉＤｌｇ．ｃｐｐは、このプログラムが入っているｈａｒｔＤｌｇ．ｃｐｐとは違う、新しいソースプログラムで、独立したウィンドウを有している。ｍｙＤＬという名前を設定することで、このｈａｒｔＤｌｇから、ＣｊｙｕｎｉＤｌｇのウィンドウに文字を表示できるようになる。ｔｏｔの値をそれぞれの得点版に記入し、左下のエディットボックスに順位を表示する。すでに配列ｎｕは昇順に並べられているため、ｎｕの値が、総合得点といっしょだったとき、順位が判明する。最終的には、ＤｏＭｏｄａｌ関数で、ソース間を越えた処理を行っている。**

**１０．１に戻り、次のゲームに移行する。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
if(game != 8 && nu[3] < 100){
        first();
}
else{
        EndDialog(IDOK);
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．１０－１**

**１１．８ゲーム目が終了するか、誰か一人が１００点を越えるとゲームを終了する。**

**プログラム１０－１でわかるとおり、ゲームが８ゲーム目でなく、一番得点の高い人が１００点を越えていないとき、ｆｉｒｓｔ関数に戻り、次のゲームに移行する。**
**そうでない場合、最終的な順位を表示し、ハーツのプログラムを完全に終了させる。ＯＫボタンを押すとウィンドウが消える。**

## ２－３．コンピュータの思考ルーチン

コンピュータの思考ルーチンを考える。コンピュータには、最も有効と思われる手段を取らせることになる。そのため、ハーツではどのようなカードの出し方が有効かを知っておく必要がある。

**ハーツの有効なカードの出し方**

１．大きな手札はなるべく早めに出すようにする。
２．台札と同じマークのカードを出さなければならないときは、なるべく小さなカードを出しておく。
３．台札と同じマークのカードを出さなくてもよい場合は、１に従い、なるべく大きなカードを出す。

この３つのことを念頭に置き、コンピュータの思考ルーチンを組み立てる。

１．大きな手札はなるべく早めに出すようにする。

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
int ma[2];
ma[0] = 0; ma[1] = 0;
for(j=0; j<=2; j++){
        c = 0; d = 0; s = 0; h = 0;
        for(i=0; i<=12; i++){
                strc2[i] = 0; strd2[i] = 0; strs2[i] = 0; strh2[i] = 0;
        }
        for(i=0; i<=12; i++){
                if(card[i+13] != ma[0] && card[i+13] != ma[1]){
                        if((card[i+13]-1)/13 == 0){
                                strc2[c] = card[i+13];
                                c++;
                        }
                        if((card[i+13]-1)/13 == 1){
                                strd2[d] = card[i+13]-13;
                                d++;
                        }
```

```
                    if((card[i+13]-1)/13 == 2){
                        strs2[s] = card[i+13]-26;
                        s++;
                    }
                    if((card[i+13]-1)/13 == 3){
                        strh2[h] = card[i+13]-39;
                        h++;
                    }
                }
            }
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．Ａ－１**

**上は、３枚交換時の、コンピュータ思考を司る、ｋｏｕｋａｎ関数の一部である。**
**有効なカードの出し方の１番目に、大きな手札はなるべく早めに出すようにする、とある。そのため、手札３枚交換時には、手札で最も大きいカード３枚を出すようなプログラムにする。**
**ｉｎｔ配列ｍａは、最も大きい３枚のカードの数字番号を一時的に保存するための配列である。これを途中で判別する条件文を入れておかないと、３枚とも一番大きな数が同じになってしまう。それを防ぐのが、上から９行目の条件文である。**
**変数ｃ配列ｓｔｒｃ２は、手札のクラブのカードの数字番号を昇順に入れていくために用意したものである。同様に、ｄ、ｓｔｒｄ２はダイヤ、ｓ、ｓｔｒｓ２はスペード、ｈ、ｓｔｒｈ２はハートのカードのために用意されている。**
**１０行目から２５行目のｉｆ文は、手札のカードを見て、そのカードのマークが何であるかを分類し、それを昇順に並べていく命令文である。**
**カードを分類し終わったとき、それぞれのカードの最大値が入った配列は、例えばクラブの場合は、**strc[c-1]**となる。初めに変数ｃ＝０と指定しており、クラブのカードが一枚見つかるたびにｃが１つずつ増えるように設定されている。そのため、このカードが昇順に並べられていることより、最も大きいクラブのカードは、配列ｓｔｒｃのｃ－１番目（最後に余分にプラス１しているため、ここでマイナスしておく必要がある）と言える。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
        max1 = strc2[c-1];
        if (max1 < strd2[d-1]){ max1 = strd2[d-1]; }
        if (max1 < strs2[s-1]){ max1 = strs2[s-1]; }
        if (max1 < strh2[h-1]){ max1 = strh2[h-1]; }
```

```
        for(i=0; i<=12; i++){
                if((card[i+13]-1)/13 == 0){
                        if(card[i+13] == max1){ kou[j] = i; }
                }
                if((card[i+13]-1)/13 == 1){
                        if(card[i+13] == max1+13){ kou[j] = i; }
                }
                if((card[i+13]-1)/13 == 2){
                        if(card[i+13] == max1+26){ kou[j] = i; }
                }
                if((card[i+13]-1)/13 == 3){
                        if(card[i+13] == max1+39){ kou[j] = i; }
                }
        }
        ma[j] = max1;
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．Ａ－２**

**プログラムＡ－１の続きである。**
**それぞれのカードの最大値がでたので、今度はその４枚の中でどれが一番大きいかを調べる。最初にｍａｘ１に、クラブの最大の数である、**strc[c-1]**を入れる。そして、順番に見比べていき、最も大きかった数を最終的にｍａｘ１とするようにしておく。**
**次に、その最大値のカードが、手札の何番目に置いてあるのかを調べる。上のプログラムは、左側のＣＯＭと交換するときのものである。よって** card[i+13]**（ｉ＝０～１２）の配列のどの番号にｍａｘ１のカードが入っているのかを調べておかねばならない。**
**条件文を使い、配列ｃａｒｄの中身とｍａｘ１の値が一致したときの変数ｉが必要な配列番号である。それを、配列ｋｏｕに入れる。このｋｏｕは、プログラム３－３で使うことになる。**
**以上のことを、あと２回繰り返し、最も大きい値のカードを３枚選出する。**

**２．台札と同じマークのカードを出さなければならないときは、なるべく小さなカードを出しておく。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
aru = 0;
```

```
for(i=0; i<=12; i++){
        if((card[i+13]-1)/13 == fir){
                aru = 1;
        }
}
if(aro != 0){
        if(aru == 1){
                for(i=0; i<=12; i++){
                        if(card[i+13] != -1){
                                if((card[i+13]-1)/13 == fir){
                                        cpu = i; i = 20;
                                }
                        }
                }
        }
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．Ｂ－１**

**上は、コンピュータの思考ルーチンである、ｃｏｍｐ関数の一部である。**
**変数ａｒｕは、手札に台札と同じマークのカードがあるかどうかを調べるためのものである。台札のマークは変数ｆｉｒに数字情報として入っているため、それと配列ｃａｒｄの中のマークを照らしあわして行き、同じマークが一枚でもあるならば、変数ａｒｕの値を１にする。**
**変数ａｒｏは、一巡の中で、自分が何番目にカードを出すのかを示す変数である。今回の場合は、最初にカードを出すかどうかを判定するために用意されている。もし、最初にカードを出すのであれば、台札と同じカードを出さなければならないという条件には当てはまらず、好きなカードを出せる。よって、そのためのプログラムは別途に用意しなければならない。**
**変数ａｒｏが０で無かった場合の命令を考える。**
**ここで、変数ａｒｕが１だった場合、台札のマークと同じカードを出さなければならない。有効なカードの出し方２に従い、最も小さいカードを出すようにする。すでに出してしまったカードの値は－１となっているので、それを読みこんでしまわないよう、条件文をつけておく。そして、まだ出していないカードの中で、台札と同じカードを見つけたら、即座にｆｏｒ文を終了し、その結果を変数ｃｐｕに代入する。カードは小さい順に並べられているため、一番最初に見つかったカードこそが、最も小さいカードと言える。**

**３．台札と同じマークのカードを出さなくてもよい場合は、１に従い、なるべく大きなカードを出す。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
if(aru == 0){
        for(j=0; j<=2; j++){
                c = 0; d = 0; s = 0; h = 0;
                for(i=0; i<=12; i++){
                        strc2[i] = 0; strd2[i] = 0; strs2[i] = 0; strh2[i] = 0;
                }
                for(i=0; i<=12; i++){
                        if(card[i+13] != -1){
                                if((card[i+13]-1)/13 == 0){
                                        strc2[c] = card[i+13];
                                        c++;
                                }
                                if((card[i+13]-1)/13 == 1){
                                        strd2[d] = card[i+13]-13;
                                        d++;
                                }
                                if((card[i+13]-1)/13 == 2){
                                        strs2[s] = card[i+13]-26;
                                        s++;
                                }
                                if((card[i+13]-1)/13 == 3){
                                        strh2[h] = card[i+13]-39;
                                        h++;
                                }
                        }
                }
                max1 = strc2[c-1];
                if (max1 < strd2[d-1]){ max1 = strd2[d-1]; }
                if (max1 < strs2[s-1]){ max1 = strs2[s-1]; }
                if (max1 < strh2[h-1]){ max1 = strh2[h-1]; }
                for(i=0; i<=12; i++){
                        if((card[i+13]-1)/13 == 0){
```

```
                    if(card[i+13] == max1){ cpu = i; }
                }
                if((card[i+13]-1)/13 == 1){
                    if(card[i+13] == max1+13){ cpu = i; }
                }
                if((card[i+13]-1)/13 == 2){
                    if(card[i+13] == max1+26){ cpu = i; }
                }
                if((card[i+13]-1)/13 == 3){
                    if(card[i+13] == max1+39){ cpu = i; }
                }
            }
        }
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．Ｂ－２**

**変数ａｒｕが０だった場合の処理である。**
**この場合、台札と同じマークのカードがないのだから、有効なカードの出し方３に基づき、最も大きなカードを出す。**
**内容は、プログラムＡのｋｏｕｋａｎ関数とほぼ同じである。８行目ののｉｆ文を、すでに出してしまったカードは考えないようにするものに変更し、結果はひとつだけでいいのだから、配列ｍａはなくしてある。また、最終的な結果は変数ｃｐｕに入る。**

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

```
c = 0; d = 0; s = 0; h = 0;
for(i=0; i<=12; i++){
        strc[i] = 60; strd[i] = 60; strs[i] = 60; strh[i] = 60;
}
else{
        for(i=0; i<=12; i++){
            if(card[i+13] != -1){
                if((card[i+13]-1)/13 == 0){
                    strc[c] = card[i+13];
                    c=1;
                }
```

```
                        if((card[i+13]-1)/13 == 1){
                                strc[d] = card[i+13]-13;
                                d=1;
                        }
                        if((card[i+13]-1)/13 == 2){
                                strs[s] = card[i+13]-26;
                                s=1;
                        }
                        if((card[i+13]-1)/13 == 3){
                                strh[h] = card[i+13]-39;
                                h=1;
                        }
                }
        }
        min = strc[0];
        if (min > strd[0]){ min = strd[0]; }
        if (min > strs[0]){ min = strs[0]; }
        if (min > strh[0]){ min = strh[0]; }
        for(i=0; i<=12; i++){
                if((card[i+13]-1)/13 == 0){
                        if(card[i+13] == min){ cpu = i; }
                }
                if((card[i+13]-1)/13 == 1){
                        if(card[i+13] == min+13){ cpu = i; }
                }
                if((card[i+13]-1)/13 == 2){
                        if(card[i+13] == min+26){ cpu = i; }
                }
                if((card[i+13]-1)/13 == 3){
                        if(card[i+13] == min+39){ cpu = i; }
                }
        }
}
```

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

**プログラム．Ｂ－３**

上は、変数ａｒｏが０だったとき、つまり一番初めにカードを出すことになったときの思考ルーチンである。
有効なカードの出し方２に従い、手札の中で最も小さいカードを出すことにする。考え方は、ほとんどＢ－２と同じで、最も小さな数を選ばなくてはならないのだから、不等号は全て逆にしなければならない。また、配列ｃａｒｄの中の数字情報は昇順に並べられているため、最初に配列ｓｔｒに入ったものが最も小さな数になる。よって、例えばクラブの場合は配列ｓｔｒｃの０番目が最も小さい値となる。
こうして、最も小さいカードを検出し、それが配列ｃａｒｄの何番目であるかを調べ、変数ｃｐｕに値を入れる。

このように、コンピュータは最も有効であると思われる法則に基づき、カードを出していくよう設定されている。

# ４．最後に

**製作結果**

**トランプゲーム“ハーツ”を一通り遊ぶことができる内容のものは完成した。**

**改良点**

**＊コンピュータの思考ルーチンの強化と、３つのＣＯＭそれぞれに性格を付ける。**
**＊トリックを取ったとき、誰が取ったのかを一目瞭然にするため、動きをつける。**
**＊ボタンをグラフィックの影に隠して、ボタンなしのゲーム画面を構成する。**

**などの改良点があげられる。**

**参考文献**

「VisualC++6.0 プログラミング入門」桜田幸嗣／田口景介著　アスキー出版局
「新 VisualC++6.0 入門―ビギナー編―」林晴比古著　ＳＯＦＴＢＡＮＫ
「新 VisualC++6.0 入門―シニア編―」林晴比古著　ＳＯＦＴＢＡＮＫ