Chapter 3

MORE FLOW OF CONTROL

1. Solutions for and Remarks on Selected Practice Programs and Programming Projects

In order to preserve flexibility of the text, the author has not dealt with classes in this chapter at all. To help those who really want to do things with classes, some of the Programming Projects will be carried out using one or several classes.

Practice Program 1: Rock Scissors Paper

Here each of two players types in one of strings "Rock", "Scissors", or "Paper". After the second player has typed in the string, the winner of this game is determined and announced.

Rules:

Rock breaks Scissors

Scissors cuts Paper

Paper covers Rock

If both players give the same answer, then there is no winner.

A nice touch would be to keep and report total wins for each player as play proceeds.

To find the classes, we note first that there are 2 identical players. These players record the character entered, and keep track of the wins. There is a constructor that sets the total wins to 0. There is a play() member function that prompts for and gets from the keyboard returns one of the characters 'R', 'P', or 'S' for rock, paper and scissors. There is an accessor member function for ch and an incrementor member function for accumulated wins so that a stand alone function `int` win(ch, ch); can determine who wins and increment the accumulated wins variable.

(A better solution would make the `wins` function a `friend` of class `player` so that wins can have access to the each player's private data to determine who wins and update the accumulated wins without the accessor function `ch()` and `incrementWins()` function.)

Note that I have left a bit of debugging code in the program.

```
class  player
{
public function members
  constructors()
```

```
  play();      // prompts for and gets this player's move
  char ch();   // accessor
  int accumulatedWins();  // accessor
  incrWins();  // increments win count
private data
  character typed in
  accumulatedWins
};

int wins(player user1, player user2);
//player1's character is compared to player2's character using accessor
//  functions.
//  wins returns
//  0 if there no winner
//  1 if player 1 wins,
//  2 if player 2 wins,
//calls the appropriate incrWins() to update the wins
```

Note that once the class is defined and the auxiliary function written, the code "almost writes itself."

```
//Rock Paper Scissors game
//class/object solution
#include <iostream>
#include <cctype>     //for int toupper(int);
using namespace std;

class Player
{
public:
  Player();
  void play();      //prompts for and gets this player's move
  char Ch();        //accessor
  int AccumulatedWins(); //accessor
  void IncrWins();  //increments win count
private:
  char ch;         //character typed in
  int totalWins;  //total of wins
};
```

```
Player::Player():totalWins(0)//intializer sets Player::totalWins to 0
{
  //The body is deliberately empty. This is the preferred C++ idiom.
  //This function could be written omitting the initializer, and
  //putting this line in the body:
  //totalWins = 0;
}
void Player::play()
{
  cout << "Please enter either R)ock, P)aper, or S)cissors."
       << endl;
  cin >> ch;
  ch = toupper(ch);  //no cast needed, ch is declared to be char
}

char Player::Ch()
{
  return ch;
}
int Player::AccumulatedWins()
{
  return totalWins;
}
void Player::IncrWins()
{
  totalWins++;
}

int wins(Player& user1, Player& user2);
// player1's character is compared to player2's character
// using accessor functions.
// The function wins returns
//  0 if there no winner
//  1 if player 1 wins,
//  2 if player 2 wins,
//calls the appropriate IncrWins() to update the wins
```

```
int wins(Player& user1, Player& user2 )  // this must change user1
                       // and user2
{
  if( ( 'R' == user1.Ch() && 'S' == user2.Ch() )||
      ( 'P' == user1.Ch() && 'R' == user2.Ch() )||
      ( 'S' == user1.Ch() && 'P' == user2.Ch() )  )
  {
    // user 1 wins
    user1.IncrWins();
    return 1;
  }
  else if( ( 'R' == user2.Ch() && 'S' == user1.Ch() )
       || ( 'P' == user2.Ch() && 'R' == user1.Ch() )
       || ( 'S' == user2.Ch() && 'P' == user1.Ch() ) )
  {
    // user2 wins
    user2.IncrWins();
    return 2;
  }
  else
    // no winner
    return 0;
}

int main()
{
  Player player1;
  Player player2;
  char ans = 'Y';
  while ('Y' == ans)
  {
    player1.play();
    player2.play();
    switch( wins(player1, player2) )
    {
    case 0:
```

```
      cout << "No winner. " << endl
           << "Totals to this move: " << endl
          << "Player 1: " << player1.AccumulatedWins()
          << endl
          << "Player 2: " << player2.AccumulatedWins()
          << endl
          << "Play again? Y/y continues other quits";
       cin >> ans;
       cout << "Thanks " << endl;
       break;
    case 1:
       cout << "Player 1 wins." << endl
            << "Totals to this move: " << endl
            << "Player 1: " << player1.AccumulatedWins()
            << endl
            << "Player 2: " << player2.AccumulatedWins()
            << endl
            << "Play Again? Y/y continues, other quits. ";
      cin >> ans;
      cout << "Thanks " << endl;
      break;
    case 2:
      cout << "Player 2 wins." << endl
           << "Totals to this move: " << endl
           << "Player 1: " << player1.AccumulatedWins()
           << endl
           << "Player 2: " << player2.AccumulatedWins()
           << endl
           << "Play Again? Y/y continues, other quits.";
      cin >> ans;
      cout << "Thanks " << endl;
      break;
    }
  ans = toupper(ans);
  }
  return 0;
}
```

A typical run for the class/object solution follows:

```
Please enter either R)ock, P)aper, or S)cissors.
R
Please enter either R)ock, P)aper, or S)cissors.
S
inside IncrWins()
1
Player 1 wins.
Totals to this move:
Player 1: 1
Player 2: 0
Play Again? Y/y continues, other quits. y
Thanks
Please enter either R)ock, P)aper, or S)cissors.
S
Please enter either R)ock, P)aper, or S)cissors.
P
inside IncrWins()
2
Player 1 wins.
Totals to this move:
Player 1: 2
Player 2: 0
Play Again? Y/y continues, other quits. y
Thanks
Please enter either R)ock, P)aper, or S)cissors.
P
Please enter either R)ock, P)aper, or S)cissors.
S
inside IncrWins()
1
Player 2 wins.
Totals to this move:
Player 1: 2
Player 2: 1
```

```
Play Again? Y/y continues, other quits.y
Thanks
Please enter either R)ock, P)aper, or S)cissors.
P
Please enter either R)ock, P)aper, or S)cissors.
P
No winner.
Totals to this move:
Player 1: 2
Player 2: 1
Play again? Y/y continues other quits q
Thanks
```

Practice Program 1. Rock Scissors Paper -- Conventional solution:

Here each of two players types in one of strings "Rock", "Scissors", or "Paper". After the second player has typed in the string, the winner of this game is determined and announced.

Rules:

Rock breaks Scissors

Scissors cuts Paper

Paper covers Rock

If both players give the same answer, then there is no winner.

A nice touch would be to keep and report total wins for each player as play proceeds.

This is a conventional solution to the problem.

```
//conventional solution to the Rock Scissors Paper game
#include <iostream>  // stream io
#include <cctype>    // for int toupper(int)
using namespace std;

int wins(char ch1, char ch2)
{
  if( ( 'R' == ch1 && 'S' == ch2 )||
      ( 'P' == ch1 && 'R' == ch2 )||
```

```
      ( 'S' == ch1 && 'P' == ch2 ) )
    return 1; // player 1 wins
  else if(( 'R' == ch2 && 'S' == ch1 )||
          ( 'P' == ch2 && 'R' == ch1 )||
          ( 'S' == ch2 && 'P' == ch1 ) )
    return 2; // player 2 wins
  else
    return 0;  // no winner
}

char play()
{
  char ch;
  cout << "Please enter either R)ock, P)aper, or S)cissors." << endl;
  cin >> ch;
  return toupper(ch); // no cast needed, char is return type
}

int main()
{
  int wins_1 = 0;
  int wins_2 = 0;
  char ch1;
  char ch2;
  char ans = 'Y';
  while ('Y' == ans)
  {
    ch1 = play();
    ch2 = play();

    switch( wins(ch1, ch2) )
    {
    case 0:
    cout << "No winner. " << endl
         << "Totals to this move: " << endl
         << "Player 1: " << wins_1  << endl
         << "Player 2: " << wins_2  << endl
```

```
         << "Play again? Y/y continues other quits";
      cin >> ans;
      cout  << "Thanks " << endl;
      break;
    case 1:
      wins_1++;
      cout << "Player 1 wins." << endl
           << "Totals to this move: " << endl
           << "Player 1: " << wins_1  << endl
           << "Player 2: " << wins_2  << endl
           << "Play Again? Y/y continues, other quits. ";
      cin >> ans;
      cout << "Thanks " << endl;
      break;
    case 2:
      wins_2++;
      cout << "Player 2 wins." << endl
           << "Totals to this move: " << endl
           << "Player 1: " << wins_1  << endl
           << "Player 2: " << wins_2  << endl
           << "Play Again? Y/y continues, other quits.";
      cin >> ans;
      cout << "Thanks " << endl;
      break;
    }
  ans = toupper(ans);
  }
  return 0;
}
```

A typical run follows for the conventional solution is essentially the same as for the 'class' based solution.

Practice Problem 2: Credit Account problem. -- class/object solution

Compute interest due, total amount due, and minimum payment for a revolving credit account.

```
Inputs: account balance
```

```
Process: Computes interest
    interest according to:
      1.5% on 1st $1000
      1.0% on all over $1000
  adds this to account balance get total due.
  Computes minimum payment according to:
  if amount due < $10,
    minimum payment is total due
  else
    minimum payment is 10% of total due

outputs: Interest due, total amount due, minimum payment

The class is the account.
the public interface has functions:
  constructors
    default constructor
    constructor with parameter balance,
    computes interest, total due, minimum pay
  interestDue() reports interest
  totalDue()  reports total amount due
  minPayDue() reports minimum payment due
private data
  balance
  total due
  interest due
  minimum payment

Class-object Implementation:

//Interest
//class-object solution
//Purpose: Compute interest due, total amount due, and minimum payment
//for a revolving credit account.
//
//Inputs: account balance
//
```

```
//Process: Computes interest
//    interest according to:
//      1.5% on 1st $1000 of balance
//      1.0% on all over $1000 of balance
//  adds this to account balance get total due.
//  Computes minimum payment according to:
//    if amount due < $10,
//      minimum payment is total due
//    else
//      minimum payment is 10% of total due
//
//outputs: Interest due, total amount due, minimum payment
#include <iostream>
using namespace std;

const double INTEREST_RATE1 = 0.015;
const double INTEREST_RATE2 = 0.01;
const double MINPAY_FRACTION = 0.10;

class CreditAccount
{
public:
  CreditAccount();  // sets everything to 0
  CreditAccount( double balance );// computes everything needed
  double interestDue();
  double totalDue();
  double minPayDue();
private:
  double balance;
  double interestDue;
  double totalDue;
  double minPay;
};
double CreditAccount::interestDue()
{
  return interestDue;
}
```

```
double CreditAccount::totalDue()
{
  return totalDue;
}
double CreditAccount::minPayDue()
{
  return minPay;
}
CreditAccount::CreditAccount()
  :balance(0),totalDue(0), interestDue(0), minPay(0)
{
  //Body deliberately left empty.
  //This is a C++ idiom. See this IRM Chapter 10.
  // This is covered in Appendix 7 of the text
}
CreditAccount::CreditAccount( double balance )
{
  if (balance <= 1000 )
    interestDue = balance*INTEREST_RATE1;
  else if ( balance > 1000 )
    interestDue = 1000*INTEREST_RATE1 + (balance - 1000)*INTEREST_RATE2;
  totalDue = balance + interestDue;
  if (totalDue <= 10 )
    minPay = totalDue;
  else
    minPay = totalDue * MINPAY_FRACTION;
}

int main()
{
  cout.setf(ios::showpoint);
  cout.setf(ios::fixed);
  cout.precision(2);
  char ans;
  double balance;
  do
  {
```

```
    cout << "Enter the account balance, please: " ;
    cin >> balance;
    CreditAccount account( balance );
    cout << "Interest due: " << account.interestDue() << endl
         << "Total due: " << account.totalDue() << endl
         << "Minimum Payment: " << account.minPayDue() << endl;
    cout << "Y/y repeats. any other quits" << endl;
    cin >> ans;
  }
  while('y' ==ans || 'Y' == ans );
}
```

A typical run for the class/object solution follows:

```
Enter the account balance, please: 9.00
Interest due: 0.14
Total due: 9.13
Minimum Payment: 9.13
Y/y repeats. any other quits
y
Enter the account balance, please: 9.85
Interest due: 0.15
Total due: 10.00
Minimum Payment: 10.00
Y/y repeats. any other quits
y
Enter the account balance, please: 985.22
Interest due: 14.78
Total due: 1000.00
Minimum Payment: 100.00
Y/y repeats. any other quits
y
Enter the account balance, please: 1000
Interest due: 15.00
Total due: 1015.00
Minimum Payment: 101.50
Y/y repeats. any other quits
y
```

```
Enter the account balance, please: 2000
Interest due: 25.00
Total due: 2025.00
Minimum Payment: 202.50
Y/y repeats. any other quits
q
```

Practice Problem 2: Credit Account problem -- Conventional solution

Compute interest due, total amount due, and minimum payment for a revolving credit account.

```
Inputs: account balance

Process: Computes interest
    interest according to:
      1.5% on 1st $1000
      1.0% on all over $1000
  adds this to account balance get total due.
  Computes minimum payment according to:
    if amount due < $10,
      minimum payment is total due
    else
      minimum payment is 10% of total due

outputs: Interest due, total amount due, minimum payment

data
  balance
  total due
  interest due
  minimum payment

//conventional solution
//
//Purpose: Compute interest due, total amount due, and minimum payment
//for a revolving credit account.
//
//Inputs: account balance
```

```
//
//Process: Computes interest
//    interest according to:
//      1.5% on 1st $1000 of balance
//      1.0% on all over $1000 of balance
//  adds this to account balance get total due.
//  Computes minimum payment according to:
//    if amount due < $10,
//      minimum payment is total due
//    else
//  minimum payment is 10% of total due
//
//Outputs: Interest due, total amount due, minimum payment

#include <iostream>
using namespace std;

const double INTEREST_RATE1 = 0.015;
const double INTEREST_RATE2 = 0.01;
const double MINPAY_FRACTION = 0.10;

double interestDue(double balance)
{
  if (balance <= 1000 )
    return balance*INTEREST_RATE1;
  return 1000*INTEREST_RATE1 + (balance - 1000)*INTEREST_RATE2;
}

double minPay(double totalDue)
{

  if (totalDue <= 10 )
    return totalDue;
  return totalDue * MINPAY_FRACTION;
}

int main()
```

```
{
  double balance;
  double interestDue;
  double totalDue;
  double minPay;
  cout.setf(ios::showpoint);
  cout.setf(ios::fixed);
  cout.precision(2);
  char ans;

  do
  {
    cout << "Enter the account balance, please: " ;
    cin >> balance;
    interestDue = interestDue(balance);
    totalDue = balance + interestDue;
    minPay = minPay(totalDue);
    cout << "Interest due: " << interestDue << endl
         << "Total due: " << totalDue << endl
         << "Minimum Payment: " << minPay << endl;
    cout << "Y/y repeats. any other quits" << endl;
    cin >> ans;
  }
  while ( 'y' ==ans || 'Y' == ans );
}
```

A typical run for the conventional solution does not differ significantly from the 'class' based solution.

Practice Program 3: Astrology Class/Object solution.

Notes:

I have not done a conventional solution. All the bits and pieces necessary to do a conventional solution are present here. I also have not done the enhancements suggested. I have, however, included comments that suggest how to carry out these enhancements.

I have used slightly modified new_line and display_line code from the text. Encourage students not to reinvent the wheel, that is, to use bits and pieces code from the book and other sources as long as copyrights are not violated.

I used a library book on astrology for my reference. A newspaper works as well.

Planning:

Input: user's birthday. Month 1-12, day 1-31

Process: table lookup of signs and horoscopes, with repeat at user's option

Output: Sign of the Zodiac and horoscope for that birthday.

Hint: user a newspaper horoscope for names, dates and a horoscope for each sign.

Enhancement: If birthday is within 2 days of the adjacent sign, announce that the birthdate is on a "cusp" and output the horoscope for the adjacent sign also.

Comments and suggestions. Program will have a long multiway branch. Store the horoscopes in a file.

Ask your instructor for any special instructions, such as file name or location if this is a class assignment.

Planning for the solution:

What are the object and classes?

The Astrological Chart would be the `class` if we were doing a full chart. We are only selecting the Sun Sign, but we still let Astro be the `class`.

Zodiac Sign names and dates:

Aries	March 21 - April 19
Taurus	April 20 - May 20
Gemini	May 21 - June 21
Cancer	June 22 - July 22
Leo	July 23 - August 22
Virgo	August 23 - September 22
Libra	September 23 - October 22
Scorpio	October 23 - November 21
Sagittarius	November 22 - December 21
Capricorn	December 21 - January 19
Aquarius	January 19 - February 18

Pices February 19 - March 20

Horoscope file structure.

The initial <cr> is necessary given this code to prevent output from running on from the first horoscope. (We did not make use of the sign number.)

<cr>
sign number
horoscope
#
sign number
horoscope
#

and so on.

```
Pseudocode for the class and what member functions do...

class Astro
{
public:
  constructors
    Astro();
    Astro( Date birthday);
      looks up and sets the sign number,

Enhancement:
      sets iscusp to -1 if birthday is within 2 days
      before the adjacent sign, to -1 if within 2 days
      after adjacent sign and 0 if neither.

  member functions
      void horoscope ();

Enhancement:
      if iscusp == -1, report horoscope before the
        current one and the current one.
```

```
      else if iscusp == 1, report the current horoscope
        and the one following.
      else // iscusp == 0, do the default action:

Unenhanced action:
      Dump the horoscopes from file up to the current one.
      Display current horoscope.

How?    # is sentinel for end of a horoscope. Dump horoscopes
      through (signNumber -1) # symbols, using utility
      functions. Display through next # using utility
      functions.
private:
  Day birthday;
  int signNumber;
  void newHoroscope(istream & inStream);
      // display inStream to next # and discard the #
  void displayHoroscope( istream& sourceFile);
      // read and ignore inStream through the next #
//Enhancement:
//  int isCusp(};
```

Planning done, now to the coding. The contents of "horoscope.dat" are listed with the 'typical run' at the end of the following code.

```
// Requires file "horoscope.dat" having structure
// lines of text
// #
// lines of text
// #
// ...
// lines of text
// #

#include <fstream>  //for stream io
#include <stdlib>   //for exit()
using namespace std;
```

```
// an enum could have certainly been used here

const int aries      = 1;
const int taurus     = 2;
const int gemini     = 3;
const int cancer     = 4;
const int leo        = 5;
const int virgo      = 6;
const int libra      = 7;
const int scorpio     = 8;
const int Sagittarius = 9;
const int capricorn   = 10;
const int aquarius    = 11;
const int pisces      = 12;

//const makes certain no error that changes these constants is made.

struct month //no checking is done... Let the user be warned!
{
  int day; // 1..28, or 29 or 30 or 31, depending on month.
  int month; // 1..12
};

class Astro
{
public:
  Astro();
  Astro( month birthday);
//      looks up and sets the sign number

//Enhancement:  sets iscusp to -1 if birthday is within 2 days
//      before the adjacent sign, to -1 if within 2 days
//      after adjacent sign and 0 if neither.

      void horoscope ();
//Enhancement: if iscusp == -1,
//        dump (signNumber - 2)  horoscopes
```

```
//      else if iscusp == 1
//        dump (signNumber - 1) horoscopes
//      display next two horoscopes.
//      return;
// unenhanced action:  if we get here, iscusp == 0 so we
//      dump (signNumber - 1) horoscopes,
//      display next horoscope.

private:
  month birthday;
  int signNumber;
  void newHoroscope(istream & inStream);
  void displayHoroscope( istream& sourceFile);
//Enhancement:
//  int isCusp;
};

// dumps all from file  through the next #
void Astro::newHoroscope(istream & inStream)
{
  char symbol;
  do
  {
    inStream.get(symbol);
  } while(symbol != '#' );
}

//displays all from file through the next #
void Astro::displayHoroscope( istream& sourceFile)
{
  char next;
  sourceFile.get(next);
  while ( '#' != next )
  {
    cout << next;
    sourceFile.get(next);
  }
```

```
}

void Astro::horoscope()
{
  ifstream infile;
  infile.open( "horoscope.dat");
  int c;
  // cusp == 0 case: dump signNumber - 1 horoscopes.
  for ( int i = 1; i < signNumber; i++)
    newHoroscope( infile );
  displayHoroscope( infile );
  cout << endl;

  //Enhancement, change from //cusp==0 case:
  //so that
  //  if (cusp == -1)
  //  dump thru (signNumber - 2) horoscopes
  //  else if(cusp == 1 )
  //  dump thru (signNumber - 1) hororscopes
  //  from this position,
  //  display two horoscopes
  //  return
  //  this is the cusp = 0 case, as above.
}

Astro::Astro()    // I prefer to use the class initializer list
{           // notation. This follows the text
  signNumber = 0;
  // isCusp = 0;   // for one of the enhancements I did not do
}

Astro::Astro( month birthday)
//int signNumber( month birthday )   // to test this turkey
//      looks up and sets the sign number,
{
  switch( birthday.month )
  {
```

```
  case 1:
     if (birthday.day <= 19) signNumber = capricorn;
     else signNumber = aquarius;

// enhancement code will look like this in all the cases:
//   if ( 17 <= birthday.day && birthday.day < 19 )cusp = -1;
//   else if ( 19 < birthday.day && birthday.day <= 21 )cusp = -1;
//   else cusp = 0;
//
    break;
  case 2:
    if ( birthday.day <= 18 ) signNumber = aquarius;
    else signNumber = pisces;
    break;
  case 3:
    if ( birthday.day <=20 ) signNumber = pisces;
    else signNumber = aries;
    break;
  case 4:
    if ( birthday.day <= 19 ) signNumber = aries;
    else signNumber = taurus;
    break;
  case 5:
    if (birthday.day <= 20 ) signNumber = taurus;
    else signNumber = gemini;
    break;
  case 6:
    if (birthday.day <= 21 ) signNumber = gemini;
    else signNumber = cancer;
    break;
  case 7:
    if (birthday.day <= 22 ) signNumber = cancer;
    else signNumber = leo;
    break;
  case 8:
    if (birthday.day <= 22 ) signNumber = leo;
    else signNumber = virgo;
```

```
    break;
  case 9:
    if (birthday.day <= 22 ) signNumber = virgo;
    else signNumber = libra;
    break;
  case 10:
    if (birthday.day <= 22 ) signNumber = libra;
    else signNumber = scorpio;
    break;
  case 11:
    if (birthday.day <= 21 ) signNumber = scorpio;
    else signNumber = sagittarius;
    break;
  case 12:
    if (birthday.day <= 21 ) signNumber = sagittarius;
    else signNumber = capricorn;
    break;
  default: cout << "no such month as " << birthday.month
           << " Aborting " << endl;
    exit(1);
    break;
  }
}

int main()
{
  month birthday;
  cout << "Any non-digit in the month field will terminate the program."
       << endl;
  cout << "Enter birthday in numeric form: month day, as 12 5 : "
       << endl;
  cin >> birthday.month >> birthday.day;
  do
  {
    cout << birthday.month << " " << birthday.day;
    Astro user(birthday);
    user.horoscope();
```

```
    cout << "Enter birthday in numeric form: month day, as 12 5: " << endl;
    cin >> birthday.month >> birthday.day;
   } while ( cin );
}
```

Test Data and Runs.

Space restrictions prohibit including a complete horoscope here for each Sun Sign. I have created an abbreviated test file for Horoscopes. (The carriage return at the start of the file is necessary to obtain a carriage return before the Aries horoscope.)

```
$cat horoscope.dat

1 Aries
#
2 Taurus
#
3 Gemini
#
4 Cancer
#
5 Leo
#
6 Virgo
#
7 Libra
#
8 Scorpio
#
9 Sagittarius
#
10 Capricorn
#
11 Aquarius
#
12 Pisces
#
```

A shortened typical run with the test data file following:

```
a.out < data | less

Test data: first entry is the month, second is the day:

cat data
1 19
1 20
1 21
2 18
2 19

 several dates omitted

10 24
11 20
11 21
11 22
12 21
12 22
12 23


Output from this run:

Enter birthday in numeric form: day month, as 12 5:
1 19
10 Capricorn

Enter birthday in numeric form: day month, as 12 5:
1 20
11 Aquarius

Enter birthday in numeric form: day month, as 12 5:
1 21
11 Aquarius
```

Several lines of output have been deleted.

```
Enter birthday in numeric form: day month, as 12 5:
12 21
9 Sagittarius

Enter birthday in numeric form: day month, as 12 5:
12 22
10 Capricorn

Enter birthday in numeric form: day month, as 12 5:
12 23
10 Capricorn

Enter birthday in numeric form: day month, as 12 5:
```

Practice Program 4: Modify Horoscope

```
// Modify Practice Program 3 to request the user's birthday then
// to display the three most compatible astrological signs
// (having the same Element)
//
// Elements are Earth, Air, Fire, and Water,
// Fire signs are Aries, Leo, Sagittarius
// Earth signs are Taurus, Virgo, Capricorn
// Air signs are Gemini, Libra, Aquarius
// Water signs are Cancer, Scorpio, Pisces

//
// Known Bugs:
// Day number is only partially verified. Program aborts if the
// day  number > 31. No further checking is done for day input errors
// such as Feb 29 on non-leap-years or April 31.

// Enhancements suggested from #3 are not implemented. Hints
// on how to do the enhancements are given in strategic places
// in comments.

// Program: file: ch3.3.cc
// Requires file "horoscope.dat" having structure
// lines of text
// #
// lines of text
// #
// ...
```

```
// lines of text
// #

#include <iostream>
#include <fstream>  //for stream io
#include <cstdlib>   //for exit()

using namespace std;
//using std::cout;
//using std::cin;
//using std::endl;


// an enum could have certainly been used here
// enum Signs {aries = 1, Taurus, gemini, cancer, leo, virgo,
// libra, scorpio, sagittarius, capricorn, aquarius, pisces};

const int aries      = 1;
const int taurus     = 2;
const int gemini     = 3;
const int cancer     = 4;
const int leo        = 5;
const int virgo      = 6;
const int libra      = 7;
const int scorpio     = 8;
const int sagittarius = 9;
const int capricorn   = 10;
const int aquarius    = 11;
const int pisces      = 12;

//const makes certain no error that changes these constants is made.

struct month //no checking is done... Let the user be warned!
{
  int day; // 1..28, or 29 or 30 or 31, depending on month.
  int monthName; // 1..12
};


class Astro
{
public:
  Astro();
  Astro(month birthday);
//      looks up and sets the sign number

//Enhancement:  sets iscusp to -1 if birthday is within 2 days
//      before the adjacent sign, to -1 if within 2 days
//      after adjacent sign and 0 if neither.


  void horoscope ();
//Enhancement: if iscusp == -1,
//        dump (signNumber - 2)  horoscopes
//      else if iscusp == 1
```

```
//        dump (signNumber - 1) horoscopes
//      display next two horoscopes.
//      return;
// unenhanced action:  if we get here, iscusp == 0 so we
//      dump (signNumber - 1) horoscopes,
//      display next horoscope.

  void DisplayCompatibleSigns();
  // for the current user's birthday,
  // this displays the three signs having
  //  the same Element

private:
  month birthday;
  int signNumber;
  void newHoroscope(istream & inStream);
  void displayHoroscope( istream& sourceFile);
//Enhancement:
//  int isCusp;
};



// dumps all from file  through the next #
void Astro::newHoroscope(istream & inStream)
{
   char symbol;

   if(!inStream)
   {
     cout << "file not open\n";
     abort();
   }

/*
//DEBUGGING
else
    cout << "file opened OK, continuing\n";

char x[80];
cout << "enter anything followed by <cr> to continue\n";
cin >> x;
//END DEBUGGING

*/

   do
   {
     inStream.get(symbol);
   } while(symbol != '#' );

}


//displays all from file through the next #
```

```
void Astro::displayHoroscope(istream& sourceFile)
{

  cout << "Your Sign number and Horoscope is: ";
  char next;
  sourceFile.get(next);

  while ( '#' != next )
  {
    cout << next;
    sourceFile.get(next);
  }
}

void Astro::horoscope()
{

  ifstream infile;
  infile.open( "horoscope.dat");

  if (!infile)
  {
    cout << "horoscope.dat did not open. Aborting\n";
      abort();
  }

//char c;
/*
infile.get(c);
if (' ' == c)
 cout << "first char in horoscope.dat is a space OK \n";
else
 cout << "first char in horoscope dat is " << c << endl;
*/

  // int c;
  // cusp == 0 case: dump signNumber - 1 horoscopes.

// cout << "signNumber " << signNumber << endl;

  for ( int i = 1; i < signNumber; i++)
    newHoroscope( infile );

  displayHoroscope( infile );

//  cout << endl;

  //Enhancement, change from //cusp==0 case:
  //so that
  //  if (cusp == -1)
  //  dump thru (signNumber - 2) horoscopes
  //  else if(cusp == 1 )
  //  dump thru (signNumber - 1) hororscopes
  //  from this position,
  //  display two horoscopes
```

```
  //  return
  //  this is the cusp = 0 case, as above.
}

Astro::Astro()    // I prefer to use the class initializer list
{           // notation. This follows the text


  cout << "Astro::Astro()\n";
  signNumber = 0;
  // isCusp = 0;   // for one of the enhancements I did not do
}

Astro::Astro( month birthday)
{
  if(birthday.day <=0 || birthday.day > 31)
  {
        cout << "no such day as " << birthday.day << "aborting \n";
        exit(1);
  }
  switch( birthday.monthName )
  {
  case 1:
     if (birthday.day <= 19) signNumber = capricorn;
     else signNumber = aquarius;

// enhancement code will look like this in all the cases:
//   if ( 17 <= birthday.day && birthday.day < 19 )cusp = -1;
//   else if ( 19 < birthday.day && birthday.day <= 21 )cusp = -1;
//   else cusp = 0;
//
    break;
  case 2:
    if ( birthday.day <= 18 ) signNumber = aquarius;
    else signNumber = pisces;
    break;
  case 3:
    if ( birthday.day <=20 ) signNumber = pisces;
    else signNumber = aries;
    break;
  case 4:
    if ( birthday.day <= 19 ) signNumber = aries;
    else signNumber = taurus;
    break;
  case 5:
    if (birthday.day <= 20 ) signNumber = taurus;
    else signNumber = gemini;
    break;
  case 6:
    if (birthday.day <= 21 ) signNumber = gemini;
    else signNumber = cancer;
    break;
  case 7:
    if (birthday.day <= 22 ) signNumber = cancer;
    else signNumber = leo;
```

```
    break;
  case 8:
    if (birthday.day <= 22 ) signNumber = leo;
    else signNumber = virgo;
    break;
  case 9:
    if (birthday.day <= 22 ) signNumber = virgo;
    else signNumber = libra;
    break;
  case 10:
    if (birthday.day <= 22 ) signNumber = libra;
    else signNumber = scorpio;
    break;
  case 11:
    if (birthday.day <= 21 ) signNumber = scorpio;
    else signNumber = sagittarius;
    break;
  case 12:
    if (birthday.day <= 21 ) signNumber = sagittarius;
    else signNumber = capricorn;
    break;

  default: cout << "no such month as " << birthday.monthName
          << " Aborting " << endl;

    exit(1);
    break;
  }
}


// for the current user's birthday,
// this displays the three signs having
// the same Element
void Astro::DisplayCompatibleSigns()
{
  cout << "Compatible Signs are: ";
  switch(signNumber)
  {
    case 1: cout << "Leo, Sagittarius";   // Aries
            break;
    case 2: cout << "Virgo, Capricorn";   // Taurus
            break;
    case 3: cout << "Libra, Aquarius";    // Gemini
            break;
    case 4: cout << "Scorpio, Pices";       // Cancer
            break;
    case 5: cout << "Sagittarius, Aries"; // Leo
            break;
    case 6: cout << "Capricorn, Taurs";    // Virgo
            break;
    case 7: cout << "Aquarius, Gemini";    // Libra
            break;
    case 8: cout << "Pices, Cancer ";      // Scorpio
            break;
```

```
    case 9: cout << "Aries, Leo";          // Sagittarius
            break;
    case 10:cout << "Taurus, Virgo";      // Capricorn
            break;
    case 11:cout << "Gemini, Libra";       // Aquarius
            break;
    case 12:cout << "Cancer, Scorpio";     // Pices
            break;
      default: cout << "Bad signNumber "
                    << signNumber << "aborting\n";
               abort();
  }

  cout << endl;
}



int main()
{
  month birthday;
  cout << "Gives Horoscope and Compatible Signs\n\n"
       << "Any non-digit in the month field terminates the program.\n\n"
       << "Enter birthday in numeric form: month day, as 12 5 : \n";
  cin >> birthday.monthName >> birthday.day;

  do
  {
    cout << "\nYou entered:  "
         << "month " << birthday.monthName << "  day: " << birthday.day
<< endl;
    Astro user(birthday);

    user.horoscope();
      user.DisplayCompatibleSigns();
    cout << endl;
    cout << "Enter birthday in numeric form: month day, as 12 5: " <<
endl;
    cin >> birthday.monthName >> birthday.day;

  } while ( cin );

  return 0;
}

/*
Test Data and Runs.
Space restrictions prohibit including a complete horoscope here for each
Sun Sign. I have created an abbreviated test file for Horoscopes. (The
carriage return at the start of the file is necessary to obtain a
carriage return before an Aries horoscope.)
$type horoscope.dat

1 Aries
#
```

```
2 Taurus
#
3 Gemini
#
4 Cancer
#
5 Leo
#
6 Virgo
#
7 Libra
#
8 Scorpio
#
9 Sagittarius
#
10 Capricorn
#
11 Aquarius
#
12 Pisces
#
```

Test data: first entry is the month, second is the day:

```
> type data.dat
1 19
1 20
1 21
2 18
2 19
3 3
4 15
4 20
5 17
5 23
10 24
11 20
11 21
11 22
12 21
12 22
12 23
```

Command to run with the test data file is:

```
a.out < data | more  // Or use your favorite pager.
```

Output:

```
Gives Horoscope and Compatible Signs

Any non-digit in the month field terminates the program.

Enter birthday in numeric form: month day, as 12 5 :

You entered:  month 1  day: 19
Your Sign number and Horoscope is:
10 Capricorn
Compatible Signs are: Taurus, Virgo

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 1  day: 20
Your Sign number and Horoscope is:
11 Aquarius
Compatible Signs are: Gemini, Libra

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 1  day: 21
Your Sign number and Horoscope is:
11 Aquarius
Compatible Signs are: Gemini, Libra

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 2  day: 18
Your Sign number and Horoscope is:
11 Aquarius
Compatible Signs are: Gemini, Libra

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 2  day: 19
Your Sign number and Horoscope is:
12 Pisces
Compatible Signs are: Cancer, Scorpio

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 3  day: 3
Your Sign number and Horoscope is:
12 Pisces
Compatible Signs are: Cancer, Scorpio

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 4  day: 15
Your Sign number and Horoscope is:
1 Aries
Compatible Signs are: Leo, Sagittarius

Enter birthday in numeric form: month day, as 12 5:
```

```
You entered:  month 4  day: 20
Your Sign number and Horoscope is:
2 Taurus
Compatible Signs are: Virgo, Capricorn

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 5  day: 17
Your Sign number and Horoscope is:
2 Taurus
Compatible Signs are: Virgo, Capricorn

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 5  day: 23
Your Sign number and Horoscope is:
3 Gemini
Compatible Signs are: Libra, Aquarius

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 10  day: 24
Your Sign number and Horoscope is:
8 Scorpio
Compatible Signs are: Pices, Cancer

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 11  day: 20
Your Sign number and Horoscope is:
8 Scorpio
Compatible Signs are: Pices, Cancer

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 11  day: 21
Your Sign number and Horoscope is:
8 Scorpio
Compatible Signs are: Pices, Cancer

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 11  day: 22
Your Sign number and Horoscope is:
9 Sagittarius
Compatible Signs are: Aries, Leo

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 12  day: 21
Your Sign number and Horoscope is:
9 Sagittarius
Compatible Signs are: Aries, Leo

Enter birthday in numeric form: month day, as 12 5:
```

```
You entered:  month 12  day: 22
Your Sign number and Horoscope is:
10 Capricorn
Compatible Signs are: Taurus, Virgo

Enter birthday in numeric form: month day, as 12 5:

You entered:  month 12  day: 23
Your Sign number and Horoscope is:
10 Capricorn
Compatible Signs are: Taurus, Virgo

Enter birthday in numeric form: month day, as 12 5:
x

*/
```

Practice Program 5: Prime Numbers

```
//
*************************************************************************
//
// This program outputs all prime numbers between 3 and 100 using
// a doubly-nested loop.
//
*************************************************************************

#include <iostream>

using namespace std;

// ====================
//     main function
// ====================

int main()
{
  int i,j;
  bool isPrime;

  for (i=3; i<=100; i++)
  {
      // Inner loop checks to see if i is prime.  Assume it is and
      // try to find a number that divides into it to determine if it is
      // not prime.
      isPrime = true;
      for (j=2; j<i-1; j++)
      {
            if ((i % j)==0)
```

```
            {
                  isPrime = false;
            }
      }
      if (isPrime)
      {
            cout << i << " is a prime number." << endl;
      }
  }

  return 0;
}
```

Practice Program 6: Buoyancy

```
// ************************************************************************
//
// Write a program that inputs the weight (in pounds) and radius (in
// feet) of a sphere and outputs whether the sphere will sink or float
// in water
//************************************************************************

// File Name: buoyancy.cpp
// Author:
// Email Address:
// Project Number: 3.15
// Description: Computes the buoyancy of a sphere and determines whether
//              or not it will sink or float in water.
// Last Changed: October 5, 2007

#include <iostream>
using namespace std;

int main()
{
    const double SPECIFIC_WEIGHT_OF_WATER = 62.4;
    const double PI = 3.1415;
    double weight, radius, buoyant_force;

    // Prompt the user for the radius and weight
    cout << "Enter the radius of the sphere, in feet: ";
    cin >> radius;
    cout << "Enter the weight of the sphere, in pounds: ";
    cin >> weight;

    // Compute the buoyant force
    buoyant_force = (4.0 / 3.0) * PI * radius * radius * radius *
                    SPECIFIC_WEIGHT_OF_WATER;

    // Compare the buoyant force to the weight of the object
    if (buoyant_force < weight)
    {
```

```
      cout << "The sphere will sink.";
    }
    else
    {
      cout << "The sphere will float.";
    }

}
```

Practice Program 7 : Same temperature in C and F.

```
// ************************************************************************
// Write a program that finds the temperature that is the same in both
// Celsius and Fahrenheit
//************************************************************************

// File Name: sametemp.cpp
// Author:
// Email Address:
// Project Number: 3.16
// Description: Finds the temperature that is the same in both Celsius and
//              Fahrenheit
// Last Changed: October 9, 2007

#include <iostream>
using namespace std;

int main()
{
    // Declare variables to hold the two temperatures
    int celsius = 100;
    int fahrenheit;

    // Compute the fahrenheit value for each Celsius value,
    // stopping when they are equal.
    do
    {
       celsius--;

       // Note that we have to be careful with integer
       // division. This, for example:
       //
       //     fahrenheit = (9/5) * celsius + 32;
       //
       // will cause us to loop indefinitely, since (9/5) == 1.

       fahrenheit = (9 * celsius)/5 + 32;
    }
    while (celsius != fahrenheit);

    // Output the solution
    cout << celsius << " degrees Celsius is equal to " << fahrenheit
        << " degrees Fahrenheit." << endl;

    return 0;
}
```

Programming Project 1: Cost of a Long Distance Call

A complete solution is not presented for this problem. I am presenting some notes on how to set up a `class` to help solve the problem and part of the code.

A conventional solution should be fairly easy from these notes.

- Pricing rules:
  a) A call *started* between 8:00 AM and 6:00 PM Monday - Friday is billed at $0.40 /minute.
  b) A call *started* before 8:00 AM or after 6:00 PM Monday - Friday is billed at $0.25 /minute.
  c) A call started on Saturday or Sunday is billed at $0.15 /minute. Input is day of the week, time the call starts, and the length of the call in minutes.
- Output is the cost of the call.
- Time: 24hour notation: 13:30 is 1:30 PM.
- Day of week read in as pair of characters:
- Mo Tu We Th Fr Sa Su
- Allow any case what so ever.
- Use an `int` variable for length of call, and assume that the user rounds up.
- Allow repeat at user option.

Enhancement: After debugging completely, add file read capability for a week's phone calls, and write bill to another file. The bill should show each call and the charge for each call and the total due for all calls.

Object Solution:

What are the objects? The calls themselves are the central objects, so we create a `class` Call:

```
#include <iostream>
using namespace std;

class Call
{
public:
  Call(); // initializes every thing to zero.

  Call( int hour, int minute, int length, char d_1, char d_2 );
  // sets the call private data in accord with the parameters
```

```
  // d1 and d2 are first two letters of the day of the week
  // computes the cost according to the above schedule.

  double cost();
  // reports the cost of this call

   //For the proposed enhancement:
   //void addToBill();
  // writes call transaction to the file
private:
  double costOfCall;  // cost of this call.
  int day;  // day the call started
        //internal store is 0..6 where 0 is Su, 1 is Mo etc
  int len;  // length of call in minutes
  int hours;
  int minute; // hours:minute is the time the call started.
};
```

Once this is coded, the main program might be:

```
int main()
{
  char d1, d2;
  int length;
  int hour;
  int minute;
  Call call;
  while( !cin.eof())
  {
  cout << "Enter the day the call was made, first two initials:"
       << endl;
  cin >>  d1 >> d2;
  cout << "Enter the time of the call in the format "
       << "hour minute" << endl;
  cin >> hour >> minute;
  cout << "Enter the length of the call in minutes. " << endl;
  cin >> length;
```

```
  call = Call( hour, minute, length, d1, d2 );
  cout << "Cost of the call at time, " << hour << ":" << minute
       << " length " << length << ", on day " << d1 << d2 << " is "
       << call.cost() << endl;
  }
  return 0;
}
```

Programming Project 4: Blackjack Hand Scoring Program

I have provided three solutions. The first solution is a full blown Class/Object based. The second is more nearly what the instructor might expect from the student. Since Chapter 4, "More Flow of Control" can be covered before Chapter 10, "Defining Classes and ADTs," I have provided a solution without structs in the next section.

Problem:

Player receives 2 - 5 cards

Cards numbered 2 - 10 are scored at the number on the card, 2 - 10.

Face card, Jack, Queen, King, score 10 points.

Ace: `if` (score without Ace + 11 <= 21)

    Ace scores 11

`else`

    Ace scores 1

Player is asked how many cards, response is some integer 2 - 5.

Player is asked for card values. response is 2 - 9, t, j, q, k, a,

where t = 10, j = jack, q = queen, k = king, a= ace, which are

stored entered into a `char` variable.

Output is either an integer 2 - 21 or the word BUSTED.

The classes are Card and Hand.

Peculiarity: C++ requires that The `Hand::Hand(int, char, char, char, char);` constructor have base class constructor calls for each `Card class` object in the `class Hand.` These follow a ":"

after the argument list, before the body of the constructor. See the Ellis and Stroustrup, The Annotated C++ Reference Manual, section 12.6.2, page 290, for more details.

```
//Black Jack hand scoring

#include <iostream>
#include <cstdlib>
#include <cctype>
using namespace std;

class Card
{
public:

  Card( char ); // sets card value
  int Value();  // returns integer value of this card
  int cardIsAce();
private:
  int value;
};

int Card::cardIsAce()
{
  if( 11 == Value() )
    return 1;
  return 0;
}

class Hand
{
public:
  // computes value of hand and stores it
  Hand( int numberOfCards, char c1, char c2, char c3, char c4, char c5);

// reports value of hand to screen: 2-21 or busted.
void reportValue();
```

```
private:
  int numberOfCards;
  int value; // if 0, "busted" is reported
  Card card1;
  Card card2;
  Card card3;
  Card card4;
  Card card5;
};

int main()
{
  char v1, v2, v3, v4, v5;
  char ans;
  do
  {
    v1 = v2 = v3 = v4 = v5 = 0;
    int n;
    cout << "Enter a number of cards you have to enter " << endl;
    cin >> n;
    cout << n << endl;
    cout << "Now enter the value of the cards, 2-9, t, j, q, k, a"
         << endl;
    switch (n)
    {
      case 2:
      cin >> v1 >> v2;
      cout << v1 << " " << v2 << endl;
      break;
    case 3:
      cin >> v1 >> v2 >> v3;
      cout << v1 << " " << v2 << " "
           << v3 << endl;
      break;
    case 4:
      cin >> v1 >> v2 >> v3 >> v4;
```

```
      cout << v1 << " " << v2 << " "
           << v3 << " " << v4 << endl;
      break;
    case 5:
      cin >> v1 >> v2 >> v3 >> v4 >> v5;
      cout << v1 << " " << v2 << " " << v3 << " "
           << v4 << " " << v5 << endl;
      break;
    default:
      cout << "wrong number of cards - all assigned 0 "
           << endl;
      break;
    }
    Hand hand( n, v1, v2, v3, v4, v5 );
    hand.reportValue();
    cout << endl;
    cout << "Y/y repeats, any other character quits" << endl;
    cin >> ans;
    cout << endl;
  } while ( 'y' == ans || 'Y' == ans);
  return 0;
}


Card::Value()
{
  return value;
}

Card::Card( char ch )
{
  if ( isdigit(ch) )
  {
    value = ch - '0';
    return;
  }
  else
```

```
    switch( ch )
    {
      case 't':
      case 'j':
      case 'q':
      case 'k':
        value = 10;
        break;
      case 'a':
        value = 11;
        break;
      case 0:
        value = 0;
        break; // there is no card present.
      default:
        cout << ch << " is not a card value! Aborting "
             << endl;
        exit(1);
    }
}

void Hand::reportValue()
{
  cout   << "The value of your hand = " << value;
  if(0==value || value > 21 )
    cout << " = Busted" << endl;
}

Hand::Hand(int n, char ch1, char ch2, char ch4, char ch5, char ch6)
:card1(ch1), card2(ch2), card3(ch4), card4(ch5), card5(ch6)
  // These call a constructor for each of the private members of Hand
{
  numberOfCards = n;

  value = card1.Value() + card2.Value() + card3.Value() +
    card4.Value() + card5.Value();
  if (value <= 21)
```

```
    return;
  else if (card1.cardIsAce())
    value = value - 10; //drop one ace to 1
  if (value <= 21)
    return;
  else if ( card2.cardIsAce())
    value = value - 10;  // drop another ace to 1
  if (value <= 21)
    return;
  else if (card3.cardIsAce())
    value = value - 10;  // drop another ace to 1
  if (value <= 21)
    return;
  else if (card4.cardIsAce())
    value = value - 10;  // drop another ace to 1
  if (value <= 21)
    return;
  else if (card5.cardIsAce())
    value = value - 10;  // drop another ace to 1
  if (value <= 21)
    return;

}
```

A run follows.

```
Here is the data used:
cat blackjack.hands
5
2 3 4 5 6
y
5
4 5 6 7 8
y
4
t a a q 2
y
```

```
3
a t q
y
2
a t
q
```

And a typical run is:

```
Enter a number of cards you have to enter
5
Now enter the value of the cards, 2-9, t, j, q, k, a
2 3 4 5 6
The value of your hand = 20
Y/y repeats, any other character quits

Enter a number of cards you have to enter
5
Now enter the value of the cards, 2-9, t, j, q, k, a
4 5 6 7 8
The value of your hand = 30 = Busted

Y/y repeats, any other character quits

Enter a number of cards you have to enter
4
Now enter the value of the cards, 2-9, t, j, q, k, a
t a a q
The value of your hand = 22 = Busted

Y/y repeats, any other character quits

Enter a number of cards you have to enter
3
Now enter the value of the cards, 2-9, t, j, q, k, a
a t q
```

```
The value of your hand = 21
Y/y repeats, any other character quits

Enter a number of cards you have to enter
2
Now enter the value of the cards, 2-9, t, j, q, k, a
a t
The value of your hand = 21
Y/y repeats, any other character quits
```

Blackjack Hand Scoring: Solution using structs.

Player receives 2 - 5 cards

Cards numbered 2 - 10 are scored at the number on the card, 2 - 10.

Face card, Jack, Queen, King, score 10 points.

Ace: `if` (score without Ace + 11 <= 21)

    Ace scores 11

  `else`

    Ace scores 1

Player is asked how many cards, response is some integer 2 - 5.

Player is asked for card values. response is 2 - 9, t, j, q, k, a,

where t = 10, j = jack, q = queen, k = king, a= ace, which are

stored entered into a `char` variable.

Output is either an integer 2 - 21 or the word BUSTED.

```
//Black Jack hand scoring - using structs

#include <iostream>
#include <cstdlib>
#include <cctype>
using namespace std;

struct Card
{
  char face;
```

```
  int value;

};

struct Hand
{
  int numberOfCards;
  int value; // if 0, "busted" is reported
  Card card1;
  Card card2;
  Card card3;
  Card card4;
  Card card5;
};

void setHandValue( Hand& );
void reportValue( Hand );

// strictly a debugging aid! It was very useful!
void dumpHand (Hand hand)
{

  cout << "current hand has " << hand.numberOfCards
       << " cards. They are: "
       << hand.card1.face << " " << hand.card1.value << " "
       << hand.card2.face << " " << hand.card2.value << " "
       << hand.card3.face << " " << hand.card3.value << " "
       << hand.card4.face << " " << hand.card4.value << " "
       << hand.card5.face << " " << hand.card5.value << endl;
}


int main()
{
  char ans;
  Hand hand;
  int n;
```

```
do
{
  hand.card1.face = hand.card2.face = hand.card3.face
          = hand.card4.face = hand.card5.face = 0;
  hand.card1.value = hand.card2.value = hand.card3.value
           = hand.card4.value = hand.card5.value = 0;
  cout << "Enter a number of cards you have to enter " << endl;
  cin >> n;
  hand.numberOfCards = n;
  cout << "number of card = " << n << endl;
  cout << "Enter the face of the cards, 2-9, t, j, q, k, a"
       << endl;

  switch (n)
  {
  case 2:
    cin >> hand.card1.face >> hand.card2.face;
    cout << hand.card1.face << " "
         << hand.card2.face << endl;
    break;
  case 3:
    cin >> hand.card1.face >> hand.card2.face
        >> hand.card3.face;
    cout << hand.card1.face << " "
         << hand.card2.face << " "
         << hand.card3.face << endl;
    break;
  case 4:
    cin >> hand.card1.face >> hand.card2.face
      >> hand.card3.face >> hand.card4.face;
    cout << hand.card1.face << " "
         << hand.card2.face << " "
         << hand.card3.face << " "
         << hand.card4.face << endl;
    break;
  case 5:
    cin >> hand.card1.face >> hand.card2.face
```

```
          >> hand.card3.face >> hand.card4.face
          >> hand.card5.face;
      cout << hand.card1.face << " "
           << hand.card2.face << " "
           << hand.card3.face << " "
           << hand.card4.face << " "
           << hand.card5.face << endl;
      break;
    default:
      cout << "bad number of cards - all assigned 0 "
           << endl;
      break;
    }
    setHandValue( hand );
    reportValue( hand );
    cout << endl;
    cout << "Y/y repeats, any other character quits" << endl;
    cin >> ans;
    cout << endl;
  } while ( 'y' == ans || 'Y' == ans);
  return 0;
}

void setValueCard ( Card& card )
{
  if ( isdigit (card.face) )
  {
    card.value = card.face - '0';
    return;
   }
  else
  { card.face = tolower(card.face);
    switch( card.face )
    {
      case 't':
      case 'j':
      case 'q':
```

```
      case 'k':
        card.value = 10;
        break;
      case 'a':
        card.value = 11;
        break;
      case 0:
      card.value = 0; break; // there is no card present.
      default:
        cout << card.face
           << " is not a card face! Aborting " << endl;
        exit(1);
    }
  }
}

void reportValue(Hand hand)
{
  cout << "The value of your hand = " << hand.value << endl;
  if(0==hand.value || hand.value > 21 )
    cout << " = Busted" << endl;
}

void setHandValue( Hand& hand)
{
  setValueCard(hand.card1);
  setValueCard(hand.card2);
  setValueCard(hand.card3);
  setValueCard(hand.card4);
  setValueCard(hand.card5);

  hand.value = hand.card1.value + hand.card2.value +
    hand.card3.value + hand.card4.value + hand.card5.value;
  if (hand.value <= 21)
    return;
  else if (hand.card1.face == 'a' )
    hand.value = hand.value - 10; //count one ace as 1
```

```
  if (hand.value <= 21)
    return;
  else if ( hand.card2.face == 'a' )
    hand.value = hand.value - 10; // count another ace as 1
  if (hand.value <= 21)
    return;
  else if (hand.card3.face == 'a' )
    hand.value = hand.value - 10; // count another ace as 1
  if (hand.value <= 21)
    return;
  else if (hand.card4.face == 'a' )
    hand.value = hand.value - 10; // count another ace as 1
  if (hand.value <= 21)
    return;
  else if (hand.card5.face == 'a' )
    hand.value = hand.value - 10; // count another ace as 1
}
```

A run for this solution is essentially the same as for the object based solution, so I am omitting the 'typical run.'

Blackjack Hand Score - conventional solution

This is a solution derived from the immediately preceding solution.

```
//Blackjack Hand Score - conventional solution

#include <iostream>
#include <cstdlib>
#include <cctype>
using namespace std;

void setHandValue(   int& handValue,
      char card1Face, char card2Face,
      char card3Face, char card4Face, char card5Face,
      int& card1Value, int& card2Value, int& card3Value,
      int& card4Value, int& card5Value
       );
```

```
void reportValue( int handValue );

// strictly a debugging aid! It was very useful!
void dumpHandValue(int& handValue, int numberOfCards,
      char card1Face, char card2Face,
      char card3Face, char card4Face, char card5Face,
      int& card1Value, int& card2Value, int& card3Value,
      int& card4Value, int& card5Value   )
{
  cout << "current hand has " << numberOfCards
       << " cards. They are: "
       << card1Face << " " << card1Value << " "
       << card2Face << " " << card2Value << " "
       << card3Face << " " << card3Value << " "
       << card4Face << " " << card4Value << " "
       << card5Face << " " << card5Value << endl;
}

int main()
{

  int numberOfCards;
  int handValue; // if 0, "busted" is reported

  char card1Face, card2Face, card3Face, card4Face, card5Face;
  int card1Value, card2Value, card3Value, card4Value, card5Value;

  char ans;

  int n;
  do
  {
    card1Face = card2Face = card3Face
              = card4Face = card5Face = 0;
    card1Value = card2Value = card3Value
               = card4Value = card5Value = 0;
```

```
cout << "Enter a number of cards you have to enter "
     << endl;
cin >> n;
numberOfCards = n;
cout << "number of card = " << n << endl;
cout << "Enter the face of the cards, 2-9, t, j, q, k, a"
     <<  endl;

switch (n)
{
  case 2:
  cin >> card1Face >> card2Face;
  cout << card1Face << " "
       << card2Face << endl;
  break;
case 3:
  cin >> card1Face >> card2Face
      >> card3Face;
  cout << card1Face << " "
       << card2Face << " "
       << card3Face << endl;
  break;
case 4:
  cin >> card1Face >> card2Face
      >> card3Face >> card4Face;
  cout << card1Face << " "
       << card2Face << " "
       << card3Face << " "
       << card4Face << endl;
  break;
case 5:
  cin >> card1Face >> card2Face
      >> card3Face >> card4Face
      >> card5Face;
  cout << card1Face << " "
       << card2Face << " "
       << card3Face << " "
```

```
            << card4Face << " "
            << card5Face << endl;
       break;
     default:
       cout << "bad number of cards - all assigned 0 "
            << endl;
       break;
     }
      setHandValue( handValue,
         card1Face, card2Face, card3Face,
         card4Face, card5Face,
         card1Value, card2Value, card3Value,
         card4Value, card5Value );
     reportValue( handValue );

     cout << endl;
     cout << "Y/y repeats, any other character quits" << endl;
     cin >> ans;
     cout << endl;
  } while ( 'y' == ans || 'Y' == ans);
  return 0;
}

void setValueCard ( char face, int& value )
{
  if ( isdigit (face) )
  {
     value = face - '0';
     return;
   }
  else
  {  face = tolower(face);
     switch( face )
     {
     case 't':
     case 'j':
     case 'q':
```

```
     case 'k':   value = 10;
                 break;
     case 'a': value = 11;
                 break;
     case 0:  value = 0; break; // there is no card present.
     default:  cout << face
                    <<  " is not a card face! Aborting " << endl;
                 exit(1);
    }
  }
}

void reportValue(int handValue)
{
  cout << "The value of your hand = " << handValue << endl;
  if(0==handValue || handValue > 21 )
    cout << " = Busted" << endl;
}

void setHandValue( int& handValue,
      char card1Face, char card2Face,
      char card3Face, char card4Face, char card5Face,
      int& card1Value, int& card2Value, int& card3Value,
      int& card4Value, int& card5Value
       )
{
  setValueCard(card1Face, card1Value);
  setValueCard(card2Face, card2Value);
  setValueCard(card3Face, card3Value);
  setValueCard(card4Face, card4Value);
  setValueCard(card5Face, card5Value);

  handValue = card1Value + card2Value +
    card3Value + card4Value + card5Value;
  if (handValue <= 21)
    return;
  else if (card1Face == 'a' )
```

```
    handValue = handValue - 10; //count one ace as 1
  if (handValue <= 21)
    return;
  else if ( card2Face == 'a' )
    handValue = handValue - 10; // count another ace as 1
  if (handValue <= 21)
    return;
  else if (card3Face == 'a' )
    handValue = handValue - 10; // count another ace as 1
  if (handValue <= 21)
    return;
  else if (card4Face == 'a' )
    handValue = handValue - 10; // count another ace as 1
  if (handValue <= 21)
    return;
  else if (card5Face == 'a' )
    handValue = handValue - 10; // count another ace as 1
  if (handValue <= 21)
    return;
}
```

The run is identical to the struct solution.

Programming Project 6: Fibonacci numbers

There is no real advantage to 'class-ifying' this solution. We do a conventional solution here.

The F_n are defined as follows:

$F_0 = 1;$

$F_1 = 1;$

$F_{n+2} = F_n + F_{n+1};$ $n = 0, 1, 2 \ldots$

This means that once you have two preceding members, each member is the sum of the previous two, and the two that don't have predecessors are 1. The values are `1 1 2 5 8 13 etc.`

The Program:

This program takes as input some initial amount of 'green crud' and some number of days. Then, assuming that green crud can reach reproductive maturity in 5 days. Assume that all the reproduction happens on day 5 of the interval between reproduction. Output is the amount of "green crud" after the number of days entered. Allow arbitrary repeats.

```
//compute Fibonacci numbers iteratively.

#include <iostream>
#include <cctype>
using namespace std;

int NthFib( int n )
{
  int f0, f1, f2 ;
  f0 = f1 = 1;
    if ( 1 == n || 0 == n )
    return 1;
  for ( int i = 1; i < n; i++)
    {
    f2 = f1 + f0;
    f0 = f1; // never need f0 again, make new f1 old f0
    f1 = f2; // new f1 is old f2
  }
  return f2;
}

int main()
{
  int n;
  char ans;
  double amount;
  do
  {
    cout << "Please enter the initial number of pounds of green crud: "
      << endl;
    cin >> amount;
```

```
    cout << amount << endl;
    cout << "Enter the number of days the green crud has to reproduce: "
       << endl;
    cin >> n;
    cout << n << endl;
    cout << "After " << n << " days, there should be "
       << NthFib(n/5)*amount << " pounds of green crud scumming "
       << "the pond." << endl;

    cout << "enter Y/y to continue, anything else quits " << endl;
    cin >> ans; ans = tolower( ans );
  } while ( 'y' == ans );
  return 0;
}
```

The data used follows:

```
10
5
y
10
9
y
```

Several data were removed here for lack of space.

```
10
20
n
```

Output from the program was:

```
Please enter the initial number of pounds of green crud:
10
Enter the number of days the green crud has to reproduce:
5
After 5 days, there should be 10 pounds of green crud scumming the pond.
enter Y/y to continue, anything else quits
Please enter the initial number of pounds of green crud:
10
```

```
Enter the number of days the green crud has to reproduce:
9
After 9 days, there should be 10 pounds of green crud scumming the pond.
enter Y/y to continue, anything else quits
Please enter the initial number of pounds of green crud:
10
```

Several data were omitted for lack of space.

```
Please enter the initial number of pounds of green crud:
10
Enter the number of days the green crud has to reproduce:
20
After 20 days, there should be 50 pounds of green crud scumming the pond.
enter Y/y to continue, anything else quits
```

Programming Project 7 : Exponential Function and Polynomial Approximation

The instructor should not expect this solution from the student without a generous collection of hints. The text author (Prof. Savitch) suggested that instructors should either give a substantial body of hints or accept the straight forward solution and present this as an elegant solution that avoids the round off and truncation error that the obvious solution has.

The first n terms of the power series expansion for e^x are

$$e^x = 1 + x + x^2/2! + x^3/3! + ...+ x^n/n!$$

The problem is to evaluate the partial sums of this for each power of x, 1 through 100 and a user chosen value of x.

input: `x` and `n`, `n` takes on each of the values `1..100`

output: sum of the `n` terms of the above polynomial, and `exp(x)` from the math library.

Notice that the power series through terms in x^4,

$$p(x) = 1 + x + x^2/2! + x^3/3! + x^4/4!$$

may be rewritten:

```
p(x) = 1 + x*(1 + x/2*(1 + x/3*(1 + x/4)))
```

Now let T_4 = 1 + x/4

T_3 = 1 + x/3*(1 + x/4)

T_2 = 1 + x/2*(1 + x/3*(1 + x/4))

and T_1 = 1 + x*(1 + x/2*(1 + x/3*(1 + x/4)))

This suggests the recurrence relation

```
T4 = 1 + x/4;
Ti-1 = 1 + x/(i-1) * Ti;
```

Then a `for` loop gives

```
T = 1 + x/4;
for ( i = 4; i > 0; i--)
{
  TL1 = 1 + x/(i-1)*T;
  T = TL1;
}
```

If you unroll the loop, you get, successively

```
T = 1 + x/4;
TL1 = 1 + x/3 * T = 1 + x/3 * (1+x/4)
T = TL1
TL1 = 1 + x/2 * T = 1 + x/2 * ( 1 + x/3 (1 + x/4))
T = TL1
TL1 = 1 + x * T = 1 + x * ( 1 + x/2 * ( 1 + x/3 * ( 1 + x/4)))
```

Ah ha! We have something! The general technique to sum the series then should be, for each n, and arbitrary x:

```
T = 1 + x/n;
for ( i = n; i > 0; i--)
{
  TL1 = 1 + x/(i-1)*T;
  T = TL1;
}
```

Let's try it!

```
//sum series for exponential

#include <iostream>
#include <cmath>
using namespace std;

double poly( int n, double x)
{
  double TL1;
  double  T = 1 + x/n;

  for (int i = n; i >= 2; i--)
  {
    TL1 = 1 + x/(i-1) * T;
    T = TL1;
  }
  return T;
}

int main()
{
  char ans;
  double x;
  cout << "To try the iterative scheme for McClaurin expansion "
     << "of exp(x)"
     << endl;
  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout.precision(16);
  do
   {
    cout << "Enter a value of x " << endl;
    cin >> x ; cout << "value of x = " << x << endl;
    for ( int i = 1; i <= 100; i++)
      cout << poly(i, x) << endl;
    cout << exp(x) << endl;
    cout << "Another x? Y/y, other quits" << endl;
```

```
    cin >> ans;
  }while( 'y' == ans || 'Y' == ans );
  return 0;
}

/*
Data used for the typical run:
cat data
1
y
2
y
3
y
10
y
20
y
-1
n
```

I did my runs under Linux in this fashion, because it allows me to see the how the individual output numbers change with each iteration.

```
a.out < data | less
```

The output is as follows. I edited the output to delete much boring output

```
To try the iterative scheme for McClaurin expansion of exp(x)
Enter a value of x
value of x = 1.0000000000000000
2.0000000000000000
2.5000000000000000
2.6666666666666665
2.7083333333333335
2.7166666666666668
2.7180555555555554
2.7182539682539684
[snip]
```

```
2.7182818284589945
2.7182818284590424
2.7182818284590451
2.7182818284590451
2.7182818284590451
...
  a lot of entries that are the same
...
2.7182818284590451
2.7182818284590451
2.7182818284590451
Another x? Y/y, other quits
Enter a value of x
value of x = 2.0000000000000000
3.0000000000000000
5.0000000000000000
6.3333333333333339
7.0000000000000000
7.2666666666666666
7.3555555555555552
7.3809523809523805
7.3873015873015868
[snip]
7.3890460157126823
7.3890545668323444
7.3890558823892158
7.3890560989306504
...
  A lot of entries that are the same
...

7.3890560989306504
7.3890560989306504
7.3890560989306504
7.3890560989306504
Another x? Y/y, other quits
Enter a value of x
```

```
value of x = 3.0000000000000000
4.0000000000000000
8.5000000000000000
13.000000000000000
16.375000000000000
18.400000000000021
19.412499999999979
19.846428571428680
20.009151785714284
20.063392857142858
20.079665178571428
20.084103084415581
20.085212560876623
[snip]
20.085536923187668
20.085536923187668
...
  A lot of entries that are the same
...
20.085536923187668
20.085536923187668
20.085536923187668
20.085536923187668
20.085536923187668
20.085536923187668
Another x? Y/y, other quits
Enter a value of x
value of x = 10.000000000000000
11.000000000000000
61.000000000000000
227.66666666666628
644.33333333333257
1477.6666666666515
2866.5555555555657
4850.6825396825388
7330.8412698412703
10086.573192239859054
```

```
12842.3051146384495951
15347.5159531826211605
17435.1916519694314047
19041.0960356515934109
20188.1705954245626344
20952.8869686065518181
21430.8347018452841439
21711.9804272798355669
21868.1724969657007023
21950.3788494319414895
21991.4820256650637020
22011.0549667284540192
22019.9517581209111086
22023.8199282915375079
22025.4316658626339631
22026.0763608910710900
22026.3243205173966999
22026.4161574160316377
[snip]
22026.4657948067033431
22026.4657948067142570
22026.4657948067142570
...
  A lot of entries that are the same
...
22026.4657948067142570
22026.4657948067142570
22026.4657948067142570
22026.4657948067142570
22026.4657948067142570
22026.4657948067178950
Another x? Y/y, other quits
Enter a value of x
value of x = 20.0000000000000000
21.0000000000000000
221.0000000000000000
1554.3333333333334849
```

```
8221.0000000000000000
34887.6666666666715173
123776.5555555555474712
377744.8095238095265813
1012665.4444444443797693
2423600.1887125223875046
5245469.6772486772388220
10376141.4745871406048536
18927261.1368179135024548
32082829.8479421846568584
50876499.4352625608444214
75934725.5516897439956665
107257508.1972236931324005
144107840.7213812768459320

[snip - About a half page of data omitted]

485165195.2878378033638000
485165195.3658750653266907
485165195.3942522406578064
[snip]
485165195.4097895622253418
485165195.4097899794578552
485165195.4097902774810791
485165195.4097902774810791
485165195.4097902774810791
...
[snip]
...
485165195.4097902774810791
485165195.4097902774810791
485165195.4097902774810791
485165195.4097902774810791
485165195.4097902774810791
Another x? Y/y, other quits
Enter a value of x
value of x = -1.0000000000000000
```

```
0.0000000000000000
0.5000000000000000
0.3333333333333333
0.3750000000000000
0.3666666666666667
0.3680555555555556
0.3678571428571429
0.3678819444444444
0.3678791887125221
[snip]
0.3678794411714421
0.3678794411714423
0.3678794411714423
...
  A lot of entries that are the same.
...
0.3678794411714423
0.3678794411714423
0.3678794411714423
Another x? Y/y, other quits
```

Programming Project 8: Approximation PI Using an Alternating Harmonic Series

```
// Pi can be approximated by the series
// pi = 4*(1 - 1/3 + 1/5 - 1/7 + 1/9 ... + ((-1)^n)/(2n+1))
// where the caret ^n means raised to the power n.
//
// The program requires input that sepcifies the number
// of terms to sum.
//
// Technique uses a variable sum of type double inside
// a loop.to sum the general term, where  account is
// taken of the sign change on each term and taking
// truncating int arithmetic.carefuly into account
//
// Note that this is a slowly converging series,
// consequently round-off may "eat" the computation
// before the series can give useful information.
//
// Posible Improvement:
//
// Even values of N produce values than PI and odd values
// produce values over PI. One approach is to average these
// two. One could then place that idea in the computation to
```

```
// get faster, better results.


#include <iostream>

int main()
{
  using std::cout;
  using std::cin;
  using std::endl;

  int N;
  char ans = 'y';
  do
  {
    double sum = 0;
    cout << "Enter the number of terms you want summed.\n";
    cin >> N;
    cout << "You requested that I sum  " << N << " terms\n";

    for (int i = 0; i < N; i++)
      {

        if(i%2)
          sum += -1.0/(2*i+1);
        else
          sum += 1.0/(2*i+1);

      }

    sum *= 4;

    cout << "4 * sum of  " << N << " terms is  \n";
    cout << sum << endl;
    cout << "Y or y continues, anything else quits\n";
    cin >> ans;
  }while( ans == 'y' || ans == 'Y');

    return 0;
}

/*

Typical run

Enter the number of terms you want summed.
10
You requested that I sum  10 terms
4 * sum of  10 terms is
3.04184
Y or y continues, anything else quits
y
Enter the number of terms you want summed.
15
You requested that I sum  15 terms
```

```
4 * sum of  15 terms is
3.20819
Y or y continues, anything else quits
y
Enter the number of terms you want summed.
16
You requested that I sum  16 terms
4 * sum of  16 terms is
3.07915
Y or y continues, anything else quits
y
Enter the number of terms you want summed.
17
You requested that I sum  17 terms
4 * sum of  17 terms is
3.20037
Y or y continues, anything else quits
Enter the number of terms you want summed.
18
You requested that I sum  18 terms
4 * sum of  18 terms is
3.08608
Y or y continues, anything else quits
n
Press any key to continue
 */
```

Programming Project 9 : Monty Hall Game Show Problem

```
// ********************************************************************
// Chapter 3 Programming Project 9
//
// This program simulates the Monty Hall game show problem.
// The three doors are represented by the numbers 0, 1, and 2.
// A random number is selected for the prize, contestant selection,
// and the door to reveal.  We compute a valid door for the host to
reveal
// and for the contestant to switch to.  Then we tally the number of wins
// when switching or staying with the original pick.
//
// We will win with probability 2/3 if we switch, and probability of 1/3
// if we stay.
//
********************************************************************

#include <iostream>
#include <cstdlib>      // Needed for random numbers

using namespace std;

// ====================
//     main function
// ====================
```

```
int main()
{
  int i;
  int numWinsStay = 0;
  int numWinsSwitch = 0;
  int prizeDoor, choiceDoor, switchDoor, revealDoor;

  // Run simulation 10,000 times
  for (i=0; i<10000; i++)
  {
      prizeDoor = rand() % 3;       // Pick door with prize behind it
      choiceDoor = rand() % 3;      // Door of contestant's pick

      // Find a door to reveal that is not the prize and not the
contestant's
      // choice
      revealDoor = 0;
      while ((revealDoor == prizeDoor) || (revealDoor == choiceDoor))
      {
            revealDoor++;
      }

      // Find a door if the contestant switches
      switchDoor = 0;
      while ((switchDoor == choiceDoor) || (switchDoor == revealDoor))
      {
            switchDoor++;
      }
      // See if we would have won and increment counters
      if (choiceDoor == prizeDoor)
      {
            numWinsStay++;
      }
      else if (switchDoor == prizeDoor)
      {
            numWinsSwitch++;
      }
  }

  cout << "If you switch, you will win " << (numWinsSwitch / 100) << "%"
      << " of the time. " << endl;
  cout << "If you stay, you will win " << (numWinsStay / 100) << "%"
      << " of the time. " << endl;

  return 0;
}
```

Programming Project 10 : BMR Calculator

```
// ****************************************************************
// Re-do the BMR calculator from Chapter 2 with an activity level
```

```
// calculation.
//*************************************************************************

#include <iostream>
using namespace std;

int main()
{
	const int CALORIES_PER_CANDYBAR = 230;
	int pounds;
	int feet, inches;
	int age;
	char sex;
	double bmr;
	char activity;

	cout << "Enter your weight in pounds." << endl;
	cin >> pounds;
	cout << "Enter your height in feet and inches (use the format 'feet
inches', e.g. '5 10' for 5 feet and 10 inches)." << endl;
	cin >> feet;
	cin >> inches;
	cout << "Enter your age in years." << endl;
	cin >> age;
	cout << "Enter M for male or F for female." << endl;
	cin >> sex;
	cout << "Are you: " << endl;
	cout << "A. Sedentary" << endl;
	cout << "B. Somewhat Active (Exercise occasionally)" << endl;
	cout << "C. Active (Exercise 3-4 days per week)" << endl;
	cout << "D. Highly Active (Exercise every day)" << endl;
	cin >> activity;

	if (sex == 'M')
	{
		bmr = 66 + (6.3 * pounds) + (12.9 * (feet*12 + inches)) -
			(6.8 * age);
	}
	else
	{
		bmr = 655 + (4.3 * pounds) + (4.7 * (feet*12 + inches)) -
			(4.7 * age);
	}
	// Adjust for activity level
	/*
	a.	Sedentary
	b.	Somewhat active (exercise occasionally)
	c.	Active (exercise 3-4 days per week)
	d.	Highly active (exercise every day)
	If the user answers "Sedentary" then increase the calculated BMR by 20
	percent.
	If the user answers "Somewhat active" then increase the calculated BMR by
	30 percent.
	If the user answers "Active" then increase the calculated BMR by 40
	percent.
	Finally, if the user answers "Highly active" then increase the calculated
	BMR by 50 percent.
	*/
	if (activity == 'A')
		bmr *= 1.2;
```

```
	else if (activity == 'B')
		bmr *= 1.3;
	else if (activity == 'C')
		bmr *= 1.4;
	else if (activity == 'D')
		bmr *= 1.5;
	cout << "You need to eat " << (bmr/CALORIES_PER_CANDYBAR) << " candy bars
to maintain your weight." << endl;

	char ch;
	cin >> ch;
	return 0;
}
```

Programming Project 11: Defective Keypad.

Notes: This program requires searching through number ranges for numbers that don't contain certain digits. This may be worth revisiting in Chapter 4 after discussing functions since that can eliminate a significant amount of duplicate code.

```
// ************************************************************************
// Chapter 3 Programming Project 11
//
// Search for the closest temperature on a defective keypad.
//************************************************************************
#include <iostream>
using namespace std;

int main()
{
	int temp;
	int onesDigit, tensDigit, hundredsDigit;
	bool contains147;

	cout << "Enter the desired temperature." << endl;
	cin >> temp;

	int lowerTemp = temp;
	// Determine if the temperature has a 1, 4, or 7
	onesDigit = lowerTemp % 10;
	tensDigit = (lowerTemp / 10) % 10;
	hundredsDigit = (lowerTemp / 100) % 10;
	if ((onesDigit==1) || (onesDigit==4) || (onesDigit==7) ||
		(tensDigit==1) || (tensDigit==4) || (tensDigit==7) ||
		(hundredsDigit==1) || (hundredsDigit==4) ||
(hundredsDigit==7))
		contains147 = true;
	else
		contains147 = false;
	while (contains147)
	{
		lowerTemp--;
		onesDigit = lowerTemp % 10;
		tensDigit = (lowerTemp / 10) % 10;
```

```
            hundredsDigit = (lowerTemp / 100) % 10;
            if ((onesDigit==1) || (onesDigit==4) || (onesDigit==7) ||
                  (tensDigit==1) || (tensDigit==4) || (tensDigit==7) ||
                  (hundredsDigit==1) || (hundredsDigit==4) ||
(hundredsDigit==7))
                  contains147 = true;
            else
                  contains147 = false;
      }

      int upperTemp = temp;
      // Determine if the temperature has a 1, 4, or 7
      onesDigit = upperTemp % 10;
      tensDigit = (upperTemp / 10) % 10;
      hundredsDigit = (upperTemp / 100) % 10;
      if ((onesDigit==1) || (onesDigit==4) || (onesDigit==7) ||
            (tensDigit==1) || (tensDigit==4) || (tensDigit==7) ||
            (hundredsDigit==1) || (hundredsDigit==4) ||
(hundredsDigit==7))
            contains147 = true;
      else
            contains147 = false;
      while (contains147)
      {
            upperTemp++;
            onesDigit = upperTemp % 10;
            tensDigit = (upperTemp / 10) % 10;
            hundredsDigit = (upperTemp / 100) % 10;
            if ((onesDigit==1) || (onesDigit==4) || (onesDigit==7) ||
                  (tensDigit==1) || (tensDigit==4) || (tensDigit==7) ||
                  (hundredsDigit==1) || (hundredsDigit==4) ||
(hundredsDigit==7))
                  contains147 = true;
            else
                  contains147 = false;
      }

      // See which one is closer to desired temp and output it
      cout << lowerTemp << endl;
      cout << upperTemp << endl;

      char ch;
      cin >> ch;
      return 0;
}
```

Programming Project 13: Holy Digits Batman!

```
// ********************************************************************
// Chapter 3 Programming Project 13
//
// Find the address where the Riddler is planning his next caper.
//********************************************************************
```

```
#include <iostream>
using namespace std;

int main()
{
 int ones, tens, hundreds, thousands;

 // Loop through all possible values and output them when
 // we  meet the conditions in the problem
 for (ones = 0; ones < 10; ones++)
  for (tens = 0; tens < 10; tens++)
   for (hundreds = 0; hundreds < 10; hundreds++)
    for (thousands = 0; thousands < 10; thousands++)
    {
        // Digits different
        if ((ones != tens) && (ones != hundreds) && (ones != thousands)
            &&
            (tens != hundreds) && (tens != thousands) &&
            (hundreds != thousands) &&
            (thousands*3 == tens))  // thousands digits 3 times the tens
        {
          int num = ones + 10*tens + 100*hundreds + 1000*thousands;
          // Check if odd
          if (num % 2 == 1)
         {
               // Check if sum is 27
               if (ones+tens+hundreds+thousands == 27)
               {
                       cout << "The address is " << num << endl;
               }
          }
        }
    }
 return 0;
}
```

2. Outline of Topics in the Chapter

3.1 Using Boolean Expressions

- Evaluating Boolean Expressions
- Functions That Return a Boolean Value
- Enumeration Types (Optional)

3.2 Multiway Branches

- Nested Statements
- Multiway if-else-Statements
- The switch-Statement
- Using switch-Statements for Menus

3. General Comments on the Chapter

Note that Chapter 3 can be covered prior to Chapter 10, in spite of this writers' (I hope not too obvious) bias toward in order traversal. (All puns intended! :)

We need a bit more sophistication in the control constructs we can use. This chapter completes (very nearly) the study of C++ control constructs. The only ones missing are the `goto`, which Boehm and Jacopini proved is unnecessary, and exceptions, which we will treat in Chapter 16.

The ISO C++ Standard provides for a Boolean data type, which it has named `'bool'` with values `false` and `true`. This change breaks any code that defines `bool, true, or false` in lower case.

The `bool` type was implemented with surprising speed. All compilers known to this author implement the `bool` type.

Here is my reading of the ISO/ANSI Standard on the `bool` type.

The C++ Standard on the `bool` data type

A data type is a collection of values with allowable operations. C++ historically has followed the path that C has followed, where Boolean expressions evaluate to `int` values. The C++ Standards committee has adopted the Boolean type. The name of the type is `bool`, and the values are `true` and `false`.

The ANSI/ISO C++ Standard reserves `bool` for the type name and `false` and `true` for Boolean literals. An integer, floating point, or pointer expression can be converted to an r-value of type `bool`. A zero value or

null pointer value is converted to `false`, any other value is converted to `true`. The `bool` value `false` may be converted to 0. The `bool` value `true` may be converted to 1.

The Boolean, or more properly, Boolean operators && (`and`), `||`(`or`) operators take Boolean arguments. The standard provides the alternate keywords `and` and `or` for use instead of `&&` and `||` respectively. The relational operators `(<, >, <=, >=)`, the equality and inequality operators `(==, !=)`, and the unary operator `!` return values of type `bool`, that value being either `false` or `true`.

The `if`, `while`, `for`, or `do-while` condition is any expression (do other possibilities exist?) that expression may be converted to a `bool` type.

3.1 Using Boolean Expressions

C++ Boolean expressions use short-circuit evaluation. Occasionally, I have forgotten and have been bitten by this, so I suggest that this be discussed.

Programming Tip: With early compilers, you may define TRUE and FALSE.

In the past, C++ used integers 1 and 0 as the values of `true` and `false` respectively. For use with these compilers, the text suggests in Appendix 10 you make the definitions:

```
  const int TRUE  = 1;
  const int FALSE = 0;
  typedef int BOOL;
or
  const int true  = 1;
  const int false = 0;
  typedef int bool;
```

Either of these will result in code that works fine with most compilers that do not have not implemented the `bool` type. This will also result in code that is portable to compilers that implement the `bool` type provided you either (1) do not define the lower case versions of Boolean, `true`, or `false`, and `typedef` bool to `int`, or (2) do define the lowercase versions of these but comment out these definitions for use with later compilers. (The text discusses `typedef` in Chapter 9.)

3.2 Multiway Branching

We needed this in the `class Month` in the last chapter. The text discusses nested `if else` statements, and gives an example,

```
if ( count > 0 )
```

```
if ( score > 5 )
  cout << "count > 0  and score > 5 \n";
else
  cout << "count > 0  and score <= 5 \n";
```

This of course works as the indentation suggests. However, the compiler is only interested in the sequence of tokens, or keywords, identifiers, punctuation and white space.

```
if ( count > 0 ) if ( score > 5 ) cout << "count > 0 and score > 5\n";
else cout << "count > 0 and score <= 5 \n";
```

If the code were laid out like this, the compiler would not care. It would compile exactly the same. Consequently, there is a rule for the compiler to decide. The text calls this the dangling `else` problem. It is a problem only for the *human* reader of the code. The language specifies which `if` the lonely `else` matches, and the compiler complies with this rule. The rule is: *an* `else` *matches the nearest* unmatched `if` *preceding the* `else`. Unfortunately, it is the human reader that we must worry about.

The text recommends that you use braces to group the `if-else` combinations to guarantee that what you want is what you get. The importance of using braces to force the `else` to match the `if` you want is well illustrated by the code in Display 3.4 of the text.

Multiway if-else-Statements and The switch-Statement

The `switch` corresponds to the '`case`' statement in Pascal and Modula, and to some extent to the computed `GOTO` in FORTRAN. The code generated by the `switch` construct computes the address of the label to be selected is, then generates a machine code to branch directly to that address. The switch generates far more efficient code than the semantically equivalent nested `if-else` statement..

The syntax of the C-C++ `switch` statement is just a bit ugly. It doesn't allow ranges in the `case` values, but it doesn't care what the order of the `case` values is. The `case` values must be distinct, i.e., no two can be the same. We give the g++ and BCC compiler messages for the error of two `case` constants with the same value at the end of the next code. The commented numbers are line numbers to find the errors easily.

```
/*1*/ //File: case.cc  Purpose: To obtain error messages for
/*2*/ //switch statement with two identical case constants
/*3*/
/*4*/ void switchSameValue()
/*5*/ {
/*6*/   int x = 3, y = 4;
```

```
/*7*/   switch ( x )
/*8*/   {
/*9*/     case 1: y++; break;
/*10*/    case 2: y = y + 2; break;
/*11*/    case 3: y = y + 3; break;
/*12*/    case 1: y = 4; break;
/*13*/    default: y = 0; break;
/*14*/  }
/*15*/ }

g++ -W -Wall --pedantic -c case.cpp
In function `void switchSameValue()':
12: duplicate case value `1'
9: previously used here

Borland C++ 5.4 for Win32
bcc32 -c case.cpp:
12: Duplicate case in function switchSameValue()
12: Expression syntax in function switchSameValue()
```

GNU's g++ tries to mark both places the error could be.

For my students, I add one rule to the author's indenting and `break` usage rules: I insist on a `break;` at the end of the last `case`, whether it is a `default` or just another `case` value. The reason is that since the `case` values are not required to be in any particular order, maintenance programmers will frequently add a `case` to the end of a `switch` statement. The absence of a `break` may be noticed -- *some* of the time. If the absence of a `break` is not noticed, and the `case` is added, when the old last `case` is encountered, the execution will "fall through" to the new `case`, producing interesting results.

To put a `break` there is to protect oneself and the programmer who follows the current programmer in working on this body of code.

3.3 More About C++ Loop Statements

The text identifies the `for` statement as especially for counter controlled loops. This is common practice in C and C++ programs. The three parameters of the `for` statement are identified as Initialization_Action, Boolean_Expression and Update_Action. The language makes no restriction on the use of these expressions.

Any expression can be used in the 'Initialization_Action', the Boolean_Expression', or the Update_Action fields. (The Boolean_Expression must, of course, either return a `bool` value or be able to be converted to type `bool`.) These expressions execute whatever is in the Initialization_Action field, then repeatedly executes the body then the Update_Action, then the Boolean_Expression, until the Boolean_Expression field evaluates to `false`. The loop stops at that point.

While C++ puts few requirements on these fields, the need for discipline suggests that we should not be quite so loose. One way is to use the `for` loop `for` counter controlled loops, as the text suggests.

Here are some idiomatic ways that a `for` loop is used.

```
for( ; while_control; )
{
  // body
}
```

or as `do-while`, using the comma expression:

```
for ( ; body, while_control; )
  ;   //left deliberately empty
```

or as an infinite loop. To escape from this loop, a `break` controlled by an `if` is used:

```
for ( ; : )
{
  // body
}
```

Any of these fields can be empty. If the Boolean expression is empty, the value is take to be `true`. Incidentally, this is *not* true for either the `while` or the `do-while` condition expressions. I mention this generality of the `for` loop because the student will frequently see these idioms.

This `for` statement is semantically almost exactly equivalent to the following `while`:

```
for( initialization; condition; update )
  body;
```

is equivalent to the `while` loop

```
initialization;
while( condition )
{
```

```
    body;
    update;
  }
```

Here is an exception to the exact equivalence of the `for` and a `while`:

When a `while` loop body contains a C++ `continue`; the `continue` statement stops the current iteration of the a `while` loop, returning to the condition. If the condition evaluates to `true` the body is executed again.

As said above, the `while` is equivalent to a `for` loop and it behaves in every respect as the `for` loop except when a `continue` is encountered. If a `continue` is encountered in a `for` statement, the update expression is executed prior to execution of the condition, then if successful, the body is executed again, if not the loop ends.

In spite of the fact that we do not treat the `continue` statement in this course, some students will run into the `continue` statement in idiomatic C and C++ code, so I mention it here for completeness.

You may wish to mention the ranged-for loop with C++11 but it is not useful until arrays are covered in Chapter 7.