Chapter 2

C++ Basics

1. Solutions to the Practice Programs and Programming Projects:

Practice Program 1. Metric - English units Conversion

A metric ton is 35,273.92 ounces. Write a C++ program to read the weight of a box of cereal in ounces then output this weight in metric tons, along with the number of boxes to yield a metric ton of cereal.

Design: To convert 14 ounces (of cereal) to metric tons, we use the 'ratio of units' to tell us whether to divide or multiply:

$$14 \text{ ounces} * \frac{1}{35,273} * \frac{\text{metric tons}}{\text{ounces}} = 0.000397 \text{ metric tons}$$

The use of units will simplify the determination of whether to divide or to multiply in making a conversion. Notice that ounces/ounce becomes unit-less, so that we are left with metric ton units. The number of ounces will be very, very much larger than the number of metric tons. It is then reasonable to divide the number of ounces by the number of ounces in a metric ton to get the number of metric tons.

Now let metricTonsPerBox be the weight of the cereal box in metric tons. Let ouncesPerBox the be the weight of the cereal box in ounces. Then in C++ the formula becomes:

```
const double ouncesPerMetric_ton = 35272.92;

metricTonsPerBox = ouncesPerBox / ouncesPerMetricTon;
```

This is metric tons PER BOX, whence the number of BOX(es) PER metric ton should be the reciprocal:

```
     boxesPerMetricTon = 1 / metricTonsPerBox;
```

Once this analysis is made, the code proceeds quickly:

```
//Purpose: To convert cereal box weight from ounces to
// metric tons to compute number of boxes to make up a
// metric ton of cereal.

#include <iostream>
using namespace std;

const double ouncesPerMetricTon = 35272.92;

int main()
{
    double ouncesPerBox, metricTonsPerbox,
           boxesPerMetricTon;
    char ans = 'y';
    while( 'y' == ans || 'Y' == ans )
    {
        cout << “enter the weight in ounces of your”
             << “favorite cereal:”
        <<endl; cin >> ouncesPerBox;
        metricTonsPerbox =
                 ouncesPerBox / ouncesPerMetricTon;
```

```
          boxesPerMetricTon = 1 / metricTonsPerbox;
           cout << "metric tons per box = "
                << metricTonsPerbox << endl;
           cout << "boxes to yield a metric ton = "
                << boxesPerMetricTon << endl;
           cout << " Y or y continues, any other character ”
               << “terminates." <<
          endl; cin >> ans;
     }
     return 0;
}
```

A sample run follows:

```
enter the weight in ounces of your favorite cereal:
14
metric tons per box = 0.000396905
boxes to yield a metric ton = 2519.49
Y or y continues, any other characters terminates.
y
enter the weight in ounces of your favorite cereal:
20
metric tons per box = 0.000567007
boxes to yield a metric ton = 1763.65
Y or y continues, any other characters terminates.
n
```

Practice Program 2 : Babylonian Algorithm

```
//**********************************************************************
// Chapter 2 Practice Program 2
//
```

```
// This program computes the square root of a number n
// using the Babylonian algorithm.
//
*********************************************************************

#include <iostream>

using namespace std;

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

int main()
{
  double guess;
  double previousguess;
  double n;
  double r;

  // Input number to compute the square root of
  cout << "Enter number to compute the square root of." <<
  endl; cin >> n;

  // Initial guess, although note this doesn't work for the number
  1 previousguess = n;
  guess = n /2;

  // Repeat until guess is within 1% of the previous guess
  while (((previousguess - guess) / previousguess) > 0.01)
  {
      previousguess = guess;
      r = n / guess;
      guess = (guess + r) / 2;
  }

  cout << "The estimate of the square root of " << n << " is "
       << guess <<

  endl; return 0;

}
```

Practice Program 3 : Treadmill Speed

```
// *******************************************************************
// Ch2Proj13.cpp
//
// This program inputs a speed in MPH and converts it to
// Minutes and Seconds per mile, as might be output on a treadmill.
//
*********************************************************************

#include <iostream>

using namespace std;
```

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

int main()
{
  double milesPerHour;
  double hoursPerMile;
  double minutesPerMile;
  double secondsPace;
  int minutesPace;

  // Input miles per hour
  cout << "Enter speed in miles per hour:" <<
  endl; cin >> milesPerHour;

  // Compute inverse, which is hours per
  mile hoursPerMile = 1.0 / milesPerHour;
  // Convert to minutes per mile which is 60 seconds/hour *
  hoursPerMile minutesPerMile = 60 * hoursPerMile;

  // Extract minutes by converting to an integer, while
  // truncates any value after the decimal point
  minutesPace = static_cast<int>(minutesPerMile);

  // Seconds is the remaining number of minutes * 60

  secondsPace = (minutesPerMile - minutesPace) * 60;

  cout << milesPerHour << " miles per hour is a pace of " << minutesPace
       << " minutes and " << secondsPace << " seconds. " <<
endl;

  return 0;
}
```

Practice Program 4 : MadLibs

```
// ********************************************************************
// Chapter 2 Practice Program 4
//
// This program plays a simple game of "Mad Libs".
//
********************************************************************

#include <iostream>

using namespace std;

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

int main()
{
  string instructorName;
  string yourName;
  string food;
  int num;
  string adjective;
  string color;
  string animal;

  cout << "Welcome to Mad Libs! Enter your name: " << endl;
  cin >> yourName;
  cout << "Enter your instructor's first or last name." <<
  endl; cin >> instructorName;
  cout << "Enter a food." << endl;
  cin >> food;
  cout << "Enter a number between 100 and 120." <<
  endl; cin >> num;
  cout << "Enter an adjective." << endl;
  cin >> adjective;
  cout << "Enter a color." << endl;
  cin >> color;
  cout << "Enter an animal." << endl;
  cin >> animal;

  cout << endl;
  cout << "Dear Instructor " << instructorName << "," <<
  endl; cout << endl;
  cout << "I am sorry that I am unable to turn in my homework at
this time."
      << endl;
  cout << "First, I ate a rotten " << food << " which made me turn "
      << color << " and " << endl;
  cout << "extremely ill. I came down with a fever of " << num << "." <<
endl;
  cout << "Next, my " << adjective << " pet " << animal << " must have "
<<
      "smelled the remains " << endl;
  cout << "of the " << food << " on my homework, because he ate it. I am
" <<
      "currently " << endl;
  cout << "rewriting my homework and hope you will accept it late."
<< endl;
  cout << endl;
  cout << "Sincerely," << endl;
  cout << yourName << endl;

  return 0;
}
```

Practice Problem 5 : Volume of a Sphere

```
//*****************************************************************
* // Chapter 2 Practice Problem 5
```

```
//
// Re-write a program using the style described in the chapter for
// indentation, add comments, and use appropriately named constants.
//**********************************************************************
*
// File Name: volume.cpp
// Author:
// Email Address:
// Project Number: 2.16
// Description: Computes the volume of a sphere given the radius
// Last Changed: October 6, 2007
#include <iostream>
using namespace std;
int main()
{
    const double PI = 3.1415;
    double radius, volume;
    // Prompt the user to enter a radius
    cout << "Enter radius of a sphere." <<
    endl; cin >> radius;
    // Compute and print the volume
    volume = (4.0 / 3.0) * PI * radius * radius *
    radius; cout << " The volume is " << volume << endl;
    return 0;
}
```

Programming Project 1. Lethal Dose

Certain artificial sweeteners are poisonous at some dosage level. It is desired to know how much soda a dieter can drink without dying. The problem statement gives no information about how to scale the amount of toxicity from the dimensions of the experimental mouse to the dimensions of the dieter. Hence the student must supply this necessary assumption as basis for the calculation.

This solution supposes the lethal dose is directly proportional to the weight of the subject, hence

$$\texttt{lethalDoseDieter} = \texttt{lethalDoseMouse} * \frac{\texttt{weightOfDieter}}{\texttt{weightOfMouse}}$$

This program accepts weight of a lethal dose for a mouse, the weight of the mouse, and the weight of the dieter, and calculates the amount of sweetener that will just kill the dieter, based on the lethal dose for a mouse in the lab. If the student has problems with grams and pounds, a pound is 454 grams.

It is interesting that the result probably wanted is a *safe* number of cans, while all the data can provide is the minimum lethal number! Some students will probably realize this, but my experience is that most will not. I just weighed a can of diet pop and subtracted the weight of an empty can. The result is about 350 grams. The label claims 355 ml, which weighs very nearly 355 grams. To get the lethal number of cans from the number of grams of sweetener, you need the number of grams of sweetener in a can of pop, and the concentration of sweetener, which the problem assumes 0.1% , that is a conversion factor of 0.001.

gramsSweetenerPerCan = 350 * 0.001 = 0.35

grams/can cans = lethalDoseDieter / (0.35 grams / can)

```
//Ch2 Programming Project 1
//Input: lethal dose of sweetener for a lab mouse, weights
//  of  mouse and dieter, and concentration of sweetener in a
//    soda.
//Output: lethal dose of soda in number of cans.
//Assumption: lethal dose proportional to weight of subject
//    Concentration of sweetener in the soda is 1/10 percent
#include <iostream>
using namespace std;
```

```
const double concentration = .001; // 1/10 of 1 percent
const double canWeight = 350;
const double gramsSweetnerPerCan = canWeight *
                concentration; //units of grams/can
int main()
{
    double lethalDoseMouse, lethalDoseDieter,
           weightMouse, weightDieter; //units: grams
    double cans;
    char ans;
    do
    {
        cout << "Enter the weight of the mouse in grams"
             << endl;
        cin >> weightMouse;
        cout << "Enter the lethal dose for the mouse in‟
             << ”grams " << endl;
        cin >> lethalDoseMouse;
        cout << "Enter the desired weight of the dieter in”
             <<‟ grams " << endl;
        cin >> weightDieter;
        lethalDoseDieter =
        lethalDoseMouse * weightDieter/weightMouse;
        cout << "For these parameters:\nmouse weight: "
             << weightMouse
             << " grams " << endl
             << "lethal dose for the mouse: "
             << lethalDoseMouse
             << "grams" << endl
             << "Dieter weight: " << weightDieter
             << " grams " << endl
             << "The lethal dose in grams of sweetener is: "
             << lethalDoseDieter << endl;
        cans = lethalDoseDieter / gramsSweetnerPerCan;
        cout << "Lethal number of cans of pop: "
             << cans << endl;
        cout << "Y or y continues, any other character quits"
             << endl;
        cin >> ans;
    } while ( 'y' == ans || 'Y' == ans );
    return 0;
}
```

```
A typical run follows:
17:23:09:~/AW$ a.out
Enter the weight of the mouse in grams
15
Enter the lethal dose for the mouse in grams
100
Enter the desired weight of the dieter, in grams
45400
For these parameters:
mouse weight: 15 grams
lethal dose for the mouse: 100 grams
Dieter weight: 45400 grams
The lethal dose in grams of sweetener is: 302667
Lethal number of cans of pop: 864762
Y or y continues, any other character quits

y
```

```
Enter the weight of the mouse in grams
30
Enter the lethal dose for the mouse in grams
100
Enter the desired weight of the dieter, in grams
45400
For these parameters:
mouse weight: 30 grams
lethal dose for the mouse: 100 grams
Dieter weight: 45400 grams
The lethal dose in grams of sweetener is: 151333
Lethal number of cans of pop: 432381
Y or y continues, any other character quits
q
17:23:56:~/AW$
```

Programming Project 2. Pay Increase

The workers have won a 7.6% pay increase, effective 6 months retroactively. This program is to accept the previous annual salary, then outputs the retroactive pay due the employee, the new annual salary, and the new monthly salary. Allow user to repeat as desired. The appropriate formulae are:

```
    const double INCREASE = 0.076;
    newSalary = salary * (1 +
    INCREASE); monthly = salary / 12;
    retroactive = (salary – oldSalary)/2;
```

The code follows:

```
//Ch2 Programming Project 2
```

```
//Given 6 mos retroactive 7.6% pay increase,
//input salary
//Output new annual and monthly salaries, retroactive pay
#include <iostream>
using namespace std;

const double INCREASE = 0.076;

int main()
{
   double oldSalary, salary, monthly, retroactive;
   char ans;
   cout << "Enter current annual salary." << endl
        << "I'll return new annual salary, monthly ″
        << ‶salary, and retroactive pay." << endl;
   cin >> oldSalary;//old annual salary
   salary = oldSalary*(1+INCREASE);//new annual
   salary monthly = salary/12;
   retroactive = (salary – oldSalary)/2;
   cout << "new annual salary " << salary << endl;
   cout << "new monthly salary " << monthly << endl;
   cout << "retroactive salary due: "
        << retroactive << endl;
   return 0;
}
17:50:12:~/AW$ a.out
Enter current annual salary.
100000
I'll return new annual salary, monthly salary, and

retroactive pay.
```

```
new annual salary 107600 new
monthly salary 8966.67
retroactive salary due: 3800
```

Programming Project 3. Retroactive Salary

```
// Modify program from Programming Project #2 so that it
// calculates retroactive
// salary for a worker for a number of months entered by the user.
//Given a 7.6% pay increase,
//input salary
//input number of months to compute retroactive salary
//Output new annual and monthly salaries, retroactive pay
#include <iostream>

const double INCREASE = 0.076;

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

  double oldSalary, salary, monthly, oldMonthly, retroactive;
  int numberOfMonths; // number of months to pay retroactive
increase
  char ans;
  cout << "Enter current annual salary and a number of months\n"
       << "for which you wish to compute retroactive pay.\n"
       << "I'll return new annual salary, monthly "
       << "salary, and retroactive pay." << endl;

  cin >> oldSalary;//old annual

  salary cin >> numberOfMonths;

  salary = oldSalary * (1+INCREASE); //new annual salary
  oldMonthly = oldSalary/12;
  monthly = salary/12;

  retroactive = (monthly - oldMonthly) * numberOfMonths;
  // retroactive = (salary - oldSalary)/2; // six months
retroactive pay increase.
  cout << "new annual salary " << salary << endl;
  cout << "new monthly salary " << monthly <<
  endl; cout << "retroactive salary due: "
       << retroactive <<
  endl; return 0;
}
/*
Typical run

Enter current annual salary and a number of months
for which you wish to compute retroactive pay.
I'll return new annual salary, monthly salary, and
retroactive pay.
12000
9
new annual salary 12912
new monthly salary 1076
retroactive salary due: 684
Press any key to continue
  */
```

Programming Project 6. Payroll

This problem involves payroll and uses the selection construct. A possible restatement: An hourly employee's regular payRate is `$16.78/`hour for `hoursWorked <= 40 hours.` If `hoursWorked > 40 hours`, then `(hoursWorked -40)` is paid at an overtime premium rate of `1.5 * payRate.` FICA (social security) tax is 6% and Federal income tax is 14%. Union dues of $10/week are withheld. If there are 3 or more covered dependents, $15 more is withheld for dependent health insurance.

a) Write a program that, on a weekly basis, accepts hours worked then outputs gross pay, each withholding amount, and net (take-home) pay.

b) Add 'repeat at user discretion' feature.

I was unpleasantly surprised to find that with early GNU g++ , you cannot use a leading 0 (such as an SSN 034 56 7891) in a sequence of integer inputs. The gnu iostreams library took the integer to be zero and went directly to the next input! You either have to either use an array of char, or 9 char variables to avoid this restriction.

Otherwise, the code is fairly straight forward.

```
//Programming Project 6
//Pay roll problem:
//Inputs: hoursWorked, number of dependents
//Outputs: gross pay, each deduction, net pay
//
//This is the 'repeat at user discretion' version
//Outline:
//In a real payroll program, each of these values would be
//stored in a file after the payroll calculation was printed
//to a report.
//
//regular payRate = $10.78/hour for hoursWorked <= 40
//hours.
//If hoursWorked > 40 hours,
//   overtimePay = (hoursWorked - 40) * 1.5 * PAY_RATE.
//FICA (social security) tax rate is 6%
//Federal income tax rate is 14%.
//Union dues = $10/week .
//If number of dependents >= 3
//    $15 more is withheld for dependent health insurance.
//
#include <iostream>
```

```
using namespace std;

const double PAY_RATE = 16.78; const
double SS_TAX_RATE = 0.06; const
double FedIRS_RATE = 0.14; const
double STATE_TAX_RATE = 0.05; const
double UNION_DUES = 10.0; const
double OVERTIME_FACTOR = 1.5; const
double HEALTH_INSURANCE = 15.0;

int main()
{
   double hoursWorked, grossPay, overTime, fica,
          incomeTax, stateTax, union_dues, netPay;
   int numberDependents,
   employeeNumber; char ans;
   //set the output to two places, and force .00 for cents
   cout.setf(ios::showpoint);
   cout.setf(ios::fixed);
   cout.precision(2);
   // compute payroll
   do
   {
       cout << "Enter employee SSN (digits only,"
            << " no spaces or dashes) \n";
       cin >> employeeNumber ;
       cout << "Please the enter hours worked and number "
            << "of employees." << endl;
       cin >> hoursWorked ;
       cin >> numberDependents;
```

```
cout << endl;

if (hoursWorked <= 40 )
    grossPay = hoursWorked * PAY_RATE;
else
{
    overTime =
    (hoursWorked - 40) * PAY_RATE * OVERTIME_FACTOR;
    grossPay = 40 * PAY_RATE + overTime;
}


fica = grossPay * SS_TAX_RATE;
incomeTax = grossPay * FedIRS_RATE;
stateTax = grossPay * STATE_TAX_RATE;
netPay =
      grossPay - fica - incomeTax
                          - UNION_DUES - stateTax;

if ( numberDependents >= 3 )
    netPay = netPay - HEALTH_INSURANCE;

//now print report for this employee:
cout << "Employee number: "
     << employeeNumber << endl;
cout << "hours worked: " << hoursWorked << endl;
cout << "regular pay rate: " << PAY_RATE << endl;

if (hoursWorked > 40 )
{
    cout << "overtime hours worked: "
```

```
                 << hoursWorked - 40 << endl;
            cout << "with overtime premium: "
                 << OVERTIME_FACTOR << endl;
        }

        cout << "gross pay: " << grossPay << endl;
        cout << "FICA tax withheld: " << fica << endl;
        cout << "Federal Income Tax withheld: "
             << incomeTax << endl;
        cout << "State Tax withheld: " << stateTax << endl;

        if (numberDependents >= 3 )
            cout << "Health Insurance Premium withheld: "
                 << HEALTH_INSURANCE << endl;

        cout << "Flabbergaster's Union Dues withheld: "
             << UNION_DUES << endl;
        cout << "Net Pay: " << netPay << endl << endl;
        cout << "Compute pay for another employee?”
             << “ Y/y repeats, any other ends" << endl;
        cin >> ans;
    } while( 'y' == ans || 'Y' == ans );
    return 0;
}
```

//A typical run:

```
14:26:48:~/AW $ a.out
Enter employee SSN (digits only, no spaces or dashes)
234567890
Please the enter hours worked and number of employees.
```

```
10
1
Employee number: 234567890
hours worked: 10.00
regular pay rate: 16.78
gross pay: 167.80
FICA tax withheld: 10.07
Federal Income Tax withheld: 23.49
State Tax withheld: 8.39
Flabbergaster's Union Dues withheld: 10.00
Net Pay: 115.85
Compute pay for another employee? Y/y repeats, any other ends
y
Enter employee SSN (digits only, no spaces or dashes)
987654321
Please the enter hours worked and number of employees.
10
3
Employee number: 987654321
hours worked: 10.00
regular pay rate: 16.78
gross pay: 167.80
FICA tax withheld: 10.07
Federal Income Tax withheld: 23.49
State Tax withheld: 8.39
Health Insurance Premium withheld: 35.00
Flabbergaster's Union Dues withheld: 10.00
Net Pay: 80.85
Compute pay for another employee? Y/y repeats, any other ends

y
```

```
Enter employee SSN (digits only, no spaces or dashes)
123456789
Please the enter hours worked and number of employees.
45
3
Employee number: 123456789
hours worked: 45.00
regular pay rate: 16.78
overtime hours worked: 5.00
with overtime premium: 1.50
gross pay: 797.05
FICA tax withheld: 47.82
Federal Income Tax withheld: 111.59
State Tax withheld: 39.85
Health Insurance Premium withheld: 35.00
Flabbergaster's Union Dues withheld: 10.00
Net Pay: 552.79
Compute pay for another employee? Y/y repeats, any other ends
n
14:28:12:~/AW $
```

Programming Project 8. Installment Loan Time

No down payment, 18 percent / year, payment of $50/month, payment goes first to interest, balance to principal. Write a program that determines the number of months it will take to pay off a $1000 stereo. The following code also outputs the monthly status of the loan.

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

// Chapter 2 Programming Project 8
```

```
int main()
{
    double principal = 1000.;
    double interest, rate = 0.015;

    int months = 0;
    cout << "months\tinterest\tprincipal" << endl;

    while ( principal > 0 )
    {
         months++;
         interest = principal * rate;
         principal = principal - (50 - interest);
         if ( principal > 0 )
             cout << months << "\t" << interest << "\t\t"
                  << principal << endl;
    }

    cout << "number of payments = " << months;
    //undo the interation that drove principal negative:
    principal = principal + (50 - interest); //include
    interest for last month:
    interest = principal * 0.015;
    principal = principal + interest;
    cout << " last months interest = " << interest;
    cout << " last payment = " << principal << endl;
    return 0;
}
Testing is omitted for this problem.
```

Programming Project 9. Separate numbers by sign, compute sums and averages

```
// Programming Problem 9
// Read ten int values output
// sum and average of positive numbers
// sum and average of nonpositive numbers,
// sum and average of all numbers,
//
// Averages are usually floating point numbers.We mulitply
// the numerator of the average computation by 1.0 to make
// the int values convert automatically to double.

#include <iostream>


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

  int value, sum = 0, sumPos = 0, sumNonPos =

  0; int countPos = 0, countNeg = 0;

  cout << "Enter ten numbers, I'll echo your number and compute\n"
         << "the sum and average of positive numbers\n"
       << "the sum and average of nonpositive numbers\n"

         << "the sum and average of all numbers\n\n";
  for(int i =0; i < 10; i++)
  {
        cin >> value;
        cout << "value " << value <<endl;
        sum += value;
        if (value > 0)
        {     sumPos += value;

              countPos++;
        }
        else
        {     sumNonPos += value;

              countNeg++;
        }
  }
  cout << "Sum of Positive numbers is "
         << sumPos   << endl;
  cout << "Average of Positive numbers is "


         << (1.0 * sumPos) / countPos  << endl;
  cout << "Sum of NonPositive numbers is "
         << sumNonPos   << endl;
  cout << "Average of NonPositive numbers is "
```

```
         << (1.0 * sumNonPos) / countNeg << endl;

  cout << "Sum "  << sum << endl;
  cout << "Average is " << (1.0 * sum)/(countPos + countNeg) << endl;

  if((countPos + countNeg)!= 10)
        cout << "Count not 10, error some place\n";

  return 0;
}
```

```
Typical run
Enter ten numbers, I'll echo your number and
compute the sum and average of positive numbers
the sum and average of nonpositive numbers
the sum and average of all numbers
4
value 4
5
value 5
-1
value -1
3
value 3
-4
value -4
-3
value -3
9
value 9
8
value 8
7
value 7
2
value 2
Sum of Positive numbers is 38
Average of Positive numbers is 5.42857
Sum of NonPositive numbers is -8
Average of NonPositive numbers is -2.66667
Sum 30
Average is 3
Press any key to continue
```

Programming Project 11 :Velocity of Sound

```
  //*********************************************************************
  // Chapter 2 Programming Project 11
//
  // This program allows the user to input a starting and an ending
  // temperature.  Within this temperature range the program should output
  // the temperature and the corresponding velocity in one degree
  // increments.
//
  ***********************************************************************
  #include <iostream>
  using namespace std;
  int main()
  {
      double VELOCITY_AT_ZERO = 331.3;
      double INCREASE_PER_DEGREE = 0.61;
      // Declare variables for the start and end temperatures, along with
      // a variable that we'll increment as we compute the temperatures.
      // (Note that we could also just increment the 'start'
      variable. int temp, start, end;
      // Prompt the user to input the time
      cout << "Enter the starting temperature, in degrees Celsius:
      "; cin >> start;
      cout << "Enter the ending temperature, in degrees Celsius:
      "; cin >> end;
      // Set cout such that we only show a single digit after the
      // decimal point
      cout.setf(ios::fixed);
      cout.setf(ios::showpoint);
      cout.precision(1);
      temp = start;
      while (temp <= end)
      {
         cout << "At " << temp << " degrees Celsius the velocity of sound is "
              << (VELOCITY_AT_ZERO + (temp * INCREASE_PER_DEGREE))
              << " m/s\n";
         temp++;
      }
      return 0;
  }
```

Programming Project 12: Water Well

```
  //*********************************************************************
  // Chapter 2 Programming Project 12
//
  // Estimate the amount of water in a water well.
  //**********************************************************************
  *
  #include <iostream>
  using namespace std;
  int main()
  {
        const double GALLONS_PER_CUBIC_FOOT =
        7.48; int radiusInches;
        double radiusFeet;
        double depthFeet;
        double volumeCubicFeet;
        double gallons;

        cout << "What is the radius in inches of the well casing?" <<
        endl; cin >> radiusInches;
        cout << "What is the depth of the well in feet?" <<
        endl; cin >> depthFeet;
        // Convert radius to feet
        radiusFeet = radiusInches / 12.0;
        // Compute volume in cubic feet
        volumeCubicFeet = (3.1415 * radiusFeet * radiusFeet) *
        depthFeet; // Convert to gallons
        gallons = volumeCubicFeet * GALLONS_PER_CUBIC_FOOT;

        cout << "Your well contains " << gallons << " gallons of water."
             << endl;
        char ch;
        cin >> ch;
        return 0;
}
```

Programming Project 13 : Calories to Maintain weight.

```
//*********************************************************************
// Chapter 2 Programming Project 13
//
// Compute calories to maintain weight if no exercise
// and then map those calories to candy bars.
//*********************************************************************
//*********************************************************************
#include <iostream>
using namespace std;

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

        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;
      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);
      }

      cout << "You need to eat " << (bmr/CALORIES_PER_CANDYBAR) <<
" candy bars to maintain your weight." << endl;
      char ch;
      cin >> ch;
      return 0;
}
```

Programming Project 14 : Classroom Scores

```
//*********************************************************************
// Chapter 2 Programming Project 14
//
// Calculate an overall percentage from individual grade scores.
//*********************************************************************
//*********************************************************************
#include <iostream>
using namespace std;

int main()
{
      int numExercises;
      int totalScore=0;
      int pointsPossible=0;

      cout << "How many exercises to input?" <<
      endl; cin >> numExercises;
      for (int i = 1; i <= numExercises; i++)
      {
            cout << "Score received for Exercise " << i << " ";
            int score;
            cin >> score;
            totalScore += score;
            cout << "Total points possible for Exercise " << i << " ";
            int points;
            cin >> points;
            pointsPossible += points;
      }
      double percent = ((double) totalScore / pointsPossible) *
      100; cout << "Your total is " << totalScore << " out of " <<
            pointsPossible << ", or " << percent << "%" << endl;

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

2. Outline of topics in the chapter

2.1 Variables and assignments

2.2 Input and Output

2.3 Data Types and Expressions

2.4 Simple Flow of Control

   branching

   looping

2.5 Program Style

3. General Remarks on the chapter:

This chapter is a very brief introduction to the minimum C++ necessary to write simple programs.

Comments in the Student's code:

Self documenting code is a code feature to be striven for. The use of identifier names that have meaning within the context of the problems being solved goes a long way in this direction.

Code that is not self documenting for whatever reasons may be made clearer by appropriate (minimal) comments inserted in the code.

"The most difficult feature of any programming language to master is the comment."

-- a disgruntled maintenance programmer.

"Where the comment and the code disagree, both should be assumed to be in error." --

an experienced maintenance programmer.

With these cautions in mind, the student should place remarks at the top of file containing program components, describing the purpose. Exactly what output is required should be specified, and any conditions on the input to guarantee correct execution should also be specified.

Remarks can clarify difficult points, but remember, if the comment doesn't add to what can be gleaned from the program code itself, the comment is unnecessary, indeed it gets in the way. Good identifier names can reduce the necessity for comments!

iostreams vs stdio

You will have the occasional student who has significant C programming experience. These students will insist on using the older stdio library input-output (for example, `printf`). Discourage this, insist on C++ iostream i/o. The reason is that the iostream library understands and uses the type system of C++. You have to tell stdio functions every type for every output attempted. And if you don't get it right, then the error may not be obvious in the output. Either iostreams knows the type, or the linker will complain that it doesn't have the member functions to handle the type you are using. Then it is easy enough to write stream i/o routines to do the i/o in a manner consistent with the rest of the library.

It is possible to do all i/o using the stream `operators` >> and << : file i/o, device i/o, as well as console i/o. You can overload the `operator>>` and `operator<<`

functions to carry out i/o for any and all objects you define. You can adjust behavior to your taste using manipulators.

Use of `endl` vs `'\n'`

With regard to `cout << "... \n";` versus `cout << "..." << endl;` this writer prefers the use of the endl manipulator in most cases. Using either will get the same result in all correct programs. With early C++ compilers, if the program has a runtime error, then the usage:

```
    cout << "... \n";
```

may fail to give any output. Under the same circumstances

```
    cout << "..." << endl;
```

would give useful information which could help determine where the program failed. The i/o library provided with current compilers are more friendly. These libraries flush the output when a carriage return ('\n') is sent, the same as sending *endl*.

Pitfall: Use of '=' Where '==' is Intended

In the text's section: "Pitfall: Using = in place of ==" the instructor should note that the very best of programmers fall into this error.

Note that the assignment

```
    x = 1;
```

is an *expression that has a value*. The value of this expression is the value transferred to the variable on the left hand side of the assignment. This value could be used by an `if` or `while` as a condition, just as any other expression can. In fact, this is a typical C/C++ idiom. While it is useful, it can be quite confusing, and I do not use it, and I discourage it in beginner classes.

Some compilers warn if an assignment expression is used as the Boolean expression in an `if` statement or within a loop. There is a way to get some further help at the price of a little discipline. If the programmer *consistently* writes the constant in a comparison on the left, as in

```
if ( 12 == x )
...;
```

instead of

```
if ( x == 12 )
...;
```

then the compiler will catch this pitfall of using = instead of ==:

```
if ( 12 = x ) // error: invalid l-value in assignment
...;
```

Otherwise this error is very hard to see, since this *looks* right. (This is one of the warts C++ inherited from C.) There is a similar pitfall in using only one `&` instead of `&&`, or one |, instead of ||.

Another Semicolon Pitfall

Occasionally one of my students will bring code that is similar to this, inquiring what might be the matter:

```
if(x == 12);
  x = 0;
.
```

After this code segment, the variable `x` has the value 0. The hard-to-see intent error is of course *caused by* the semicolon at the end of the `if` line which defines a *null* statement. The `if` control construct controls a single statement, which is the null statement here. The extra semicolon problem can cause infinite loops. See Display 2.11, Syntax of the while-Statement, and the following code segment:

```
x = 10;
while ( x > 0 );
{
  cout << x << endl;
  x--;
}
```

A program having this loop in it hangs, giving no output. As before, the extraneous semicolon after the `while` causes the problem. Note that placing assignments in the control of an `if` or a `while` is C (and C++) idiom. Even so, some compilers warn about this.

How Do I Find What Key Will Stop My Program?

Such code as in the last paragraph must be killed to get back control of the computer. Under UNIX or Unix work-alike systems, there are several system-defined keys that will terminate a program. The one named `kill` is the best one to try first. This is usually `control-c,` but under System V, this may be the `delete` key. If this fails, the current value for a Unix or Linux system may be found by typing the command `stty` at the system prompt. The command `stty - a` may be necessary. (The space between `stty` and the `-` is *required*, and there *must* be no space between the `-` and the `a`. In this last case, you may wish to type `stty -a | more` (`stty - a`, piped into `more`) to get this information a screenful at a time. This gives you more information than you ever wanted about the terminal settings. The values of interest are `intr`, `quit` and `kill`. These are set to `^C, ^\`, and `(^U control C, control-back-slash` and `control U)` on this writer's Debian 3.0 Linux system. With Windows press control-alt-del to bring up the task manager to stop a runaway program. If using an IDE such as Visual Studio you can also stop the program by clicking on the stop icon.