1.1 Introduction

Consider the concept of Temperature. It is used in heat transfer which seeks to understand energy transfer in material bodies as a result of temperature differences. At an atomic level, temperature measures the rate at which atoms of an object vibrate. The faster the vibration, the higher the temperature.

In this chapter we are going to look at certain issues of arithmetic in C++ and we will learn a technique that is very useful in program development and maintenance. We will learn about tracing a program.

1.2 Model

Over the centuries engineers and scientist have used various scales to measure temperature. Some have become standardized. These include the Centigrade or Celsius, Fahrenheit, Rankin, and Kelvin scales.

The standard scales that are currently in use are derived making reference to physical events. These make reference to the temperature at which water has a different phase. Water becomes a solid at 0°C, 32°F, 273.16°K, and 491.69°R, where the C is the Centigrade or Celsius scale, F is the Fahrenheit Scale, K is the Kelvin scale, and R is the Rankin scale. Water boils at 100°C or 212°F. The lowest absolute temperature corresponds to the state where the atoms of an object do not vibrate any longer. These are measured as 0°K and 0°R,
1. Change of Scales

1.3 Method

The scales are such that they vary linearly against each other. Hence with a bit of straightforward algebra, the following familiar equations can be derived.

\[
\begin{align*}
°F &= \frac{9}{5}°C + 32, \\
°K &= °C + 273.16, \\
°R &= °F + 459.69.
\end{align*}
\]

These equations allow the conversion from one scale to the other.

1.4 Implementation

The implementation of these conversion equations is straightforward in a language such as C++.

```cpp
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

/*
  AUTHOR: -- J. C. Diaz
  ORIGINALLY WRITTEN DATE: -- May 24 -1999
  FINAL MODIFICATION DATE: -- Jan 19 -2004
  PURPOSE: To change temperature scale from Centigrade to Fahrenheit, Kelvin, Rankin
  REQUIRED INPUT: Temperature value in Centigrades.
  OUTPUT: The corresponding values in each scale:
    Centigrade, Fahrenheit, Kelvin, Rankin
  VARIABLES: cent, Fahr, Kelv, Rank
*/

int main()
{
    float cent, Fahr, Kelv, Rank;

    cout << "Please type a temperature in Centigrades" << endl;
    cin >> cent;
    // Further code for conversion
    return 0;
}
```
1. Change of Scales

```cpp
Fahr = (9/5) * cent + 32;
Kelv  = cent + 273.16;
Rank  = Fahr + 459.69;

cout << "The Temperature in Centigrade =" << cent << endl;
cout << "The Temperature in Fahrenheit =" << Fahr << endl;
cout << "The Temperature in Kelvin =" << Kelv << endl;
cout << "The Temperature in Rankin =" << Rank << endl;
```

1.4.1 Assessment of Program

This program has been saved and compiled as the executable `scale_change`. The results of running it are below.

```
% scale_change
0
   Centigrade = 0.0
   Fahrenheit = 32.0
   Kelvin     = 273.16
   Rankin     = 491.69
% scale_change
100
   Centigrade = 100.0
   Fahrenheit = 132.0
   Kelvin     = 373.16
   Rankin     = 591.69
% scale_change
-273.16
   Centigrade = -273.16
   Fahrenheit = -241.16
   Kelvin     = 0.0
   Rankin     = 218.53
```

These results clearly are not correct! The reason has to do with how C++ handles its arithmetic operations.

The language C++ allows to combine in one arithmetic expression several operations. However, since the computer will execute only one operation at a time, it becomes important to determine the order in which the operations are performed. This order is dependent on the precedence of the operators.

1.4.2 Operator Precedence

C++ uses the same precedence for the binary arithmetic operations that the same algebraic operations have.

- Multiplication * and division have the next highest order of precedence.
- Addition + and subtraction - have the lowest order of precedence.
1. Change of Scales

Parenthesis ( ) can be used whenever required for clarity.

- $A + B / C + D$ means $A + \frac{B}{C} + D$.
- $(A + B) / (C + D)$ means $\frac{A+B}{C+D}$.

When two operators have the same precedence, they assume a left-to-right precedence.

- $A * B / C$ means $(A \times B) / C$.
- $A / B * C$ means $(A / B) \times C$.

Intermediate Results

Since the computer will execute only one operation at a time, the order of precedence determines the order in which the operations are performed. Intermediate results are computed in the appropriate order. Let us consider the formula for the roots of the quadratic equation.

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

is written in C++ as:

```c++
float a, b, c;
float x1;
...
x1 = ( -b + sqrt(b*b -4*a*c))/(2*a);
```

and actually computed as:

```c++
float a, b, c;
float x1;
float temp, temp2;
...
temp = b*b;
temp2= a*c;
temp2= 4*temp2;
temp = temp - temp2;
temp = sqrt(temp);
temp = -b + temp;
temp2= 2*a;
x1 = temp/temp2;
```

Where temp, temp2 are intermediate results.

Assessment of scale_change Program

It is clear by looking at the program `scale_change` above that the order of operation is going to be the desired one. The problem is not with the order of operations. It is with how C++ represents the number system.
1. Change of Scales

1.4.3 Number Representation

C++ allows three types for numeric variables \texttt{int}, \texttt{float}, \texttt{double} and \texttt{complex}. \texttt{Complex} type variables are treated as a pair of \texttt{float} type variables. Most implementations use standard forms. The key is the size of a computer \texttt{word}. A word is generally set by the computer manufacturer to be 32-bit or 64-bit.

The \texttt{int} Data Type

The \texttt{int} number system is modeled by

$$i = s \sum_{k=0}^{q-1} d_k 2^k$$

where $i$ is the integer value
$s$ is the sign (+1 or -1)
$q$ is the number of digits ($q = 32, 64$)
$d_k$ is the k-th digit and it is an integer value of 0 or 1.

The \texttt{float} Data Type

The real number system uses scientific notation to represent a wide range of values. The 32-bit representation is known as single precision, \texttt{float} where the 64-bit representation is known as the double precision representation, \texttt{double}. The IEEE standard is a representation that is used by most manufacturers. The 32-bit representation is modeled by

$$x = s 2^e \sum_{k=1}^{p-1} f_k 2^k$$

where $x$ is the real value
$s$ is the sign (+1 or -1)
$e$ is an integer between some minimum and maximum ($-126 \leq e \leq 127$)
$p$ is the number of of mantissa digits ($p = 24$)
$f_k$ is the k-th digit and it is an integer value of 0 or 1. The IEEE standard does not represent $f_1$ which is assumed to be 1.

The mantissa, including its sign, can be represented in 24 bits. The exponent, including its sign, takes 8 bits, for a total of 32 bits in the single precision representation.
1.4.4 Truncation

Arithmetic operations C++ involving two variables of the same type result in intermediate values of the same type. For instance, when the arithmetic operation is performed using two int variables, the intermediate result is int.

Consider the case of division of an int by an int.

```c
int num1, num2;
num1 = num2 / 2;
```

Mathematically the result of a quotient of two int is not generally an int. The computed result is the int value closest to the quotient and between zero and the quotient inclusively. If num2 = 5 then num1 = 2, where as if num2 = 8 then num1 = 4.

This explains what is wrong with the scale_change program above. The statement

```c
Fahr = (9/5) * cent + 32;
```

has an intermediate calculation of two int values 9/5 which results in 1 rather than 1.8. However, now the result of the quotient 9/5 is an int and it is being multiplied by a float cent to which an int is being added 32. That means that the number system must be able to mix arithmetic types. This requires special rules which have significant implications in programming.

Mixed Type Arithmetic

C++ allows mixed-type arithmetic. There are situations where it is convenient to use mixed-type operations. Mixed-type arithmetic needs to be understood to avoid surprises. Consider

```c
int numinch;
float side;
numinch = side / 2.4;
```

We know that the division of the float variable side and the float constant 2.4 results in a float value. However, the result is to be stored as an int. The float value is truncated. The computer drops the fractional part. In general, when a variable does not have the same type as the result of the operation, the value of the variable is converted to the type of the result and the operation is performed.

Mixed-type expressions may be inaccurate because of truncation if operations between int are embedded in the expression. Consider the calculation of the area of a circle of radius r.

```c
float area, r;
area = (1/2)*3.14159* r * r;
```

The division of 1 by 2 yields the intermediate value of 0!. And the final answer will be wrong. Arithmetic expressions that include them should be avoided.

Conversions of operand's data type and within mixed mode arithmetic expressions and assignments are done implicitly by the compiler. C++ provides also for explicit type conversions through the cast operator. These are operators having the form data-type( expression) where data-type is the desired type into which the result of the expression is to be converted.
float a, b;
int j;
...
j = int(a*b);
b = float(j + 5);

1.4.5 Final Assessment of scale_change Program

The solution consists of modifying the program `scale_change` by simply adding a period for every number that is to be represented as a float type value. The revised program results in the program `scale_change2` below.

```cpp
// file scale_change2.cpp
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

/*
*  AUTHOR: -- J. C. Diaz
*  FINAL MODIFICATION DATE: -- May 24 -1999
*  PURPOSE: To change temperature scale from Centigrade to Fahrenheit, Kelvin, Rankin
*  REQUIRED INPUT: Temperature value in Centigrades.
*  OUTPUT: The corresponding values in each scale:
*          Centigrade, Fahrenheit, Kelvin, Rankin
*  VARIABLES: cent, Fahr, Kelv, Rank
*/

void main()
{
  float cent, Fahr, Kelv, Rank;
  cout << "Please type a temperature in Centigrades" << endl;
  cin >> cent;
  Fahr = (9./5.) * cent + 32;
  Kelv = cent + 273.16;
  Rank = Fahr + 459.69;
  cout << "The Temperature in Centigrade =" << cent << endl;
  cout << "The Temperature in Fahrenheit =" << Fahr << endl;
  cout << "The Temperature in Kelvin =" << Kelv << endl;
  cout << "The Temperature in Rankin =" << Rank << endl;
}
```
1. Change of Scales

The compilation of this program into an executable called scale_change2 gives the following correct results:

% scale_change2
0
Centigrade = 0.0
Fahrenheit = 32.0
Kelvin = 273.16
Rankin = 491.69
% scale_change2
100
Centigrade = 100.0
Fahrenheit = 212.0
Kelvin = 373.16
Rankin = 671.69
% scale_change2
-273.16
Centigrade = -273.16
Fahrenheit = -459.688
Kelvin = 0.0
Rankin = 2.01416E-03

1.5 Assessment

1.5.1 Exercises

Order of Intermediate Operations Exercises

The order of the intermediate operations in a program statement are crucial to the intended result. Consider the sample.cpp program:

// File sample.cpp
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
    int x, z;
    float j, n;
    x = 3 / 5 * 10.0;
    n = 10. / 6 * 2;
    j = 3.0 + 5.0 / 2.0 + 2;
    z = 2 * 2 * 2 / 3.;
    cout << " Value of x: " << x << endl;
    cout << " Value of n: " << n << endl;
    cout << " Value of j: " << j << endl;
    cout << " Value of z: " << z << endl;
}
Consider the calculation of the value of the variable \( x \) in this short code. The compiler reads the expression on the left of the assignment one character at a time. The value 3 is read and stored as an int temporary, the operator \(/\) is read, and its second operand is sought. The value of 5 is read and stored as another int temporary. The operator \(*\) is read. It is determined that the two operators have the same order of precedence, thus, the first operator must be computed to produce an intermediate result to be the left operand for the second operator. Hence, an operation between two int temporaries is completed yielding the value 0 as the first operand for the multiplication operator.

Then the 10.0 and the ; are read. The last one indicates the end of the statement. The other is stored as a float temporary. It is the second operand of the multiplication operator. However, its first operand is an int and the second is a float. Hence, the value of the first operand is cast as a float and the operation is done in float and produces a float.

Now the expression of the right hand side of the assignment operator has produced a value, a float. The target variable on the left side of the assignment is an int. Thus, the compiler casts the resulting float value of the expression as an int to be stored in the target variable.

**Exercises**

Show the order of the operations of the other statements in the sample program.

### Free energy in an ideal Gas

The free energy in an ideal gas depends on the pressure and the temperature of the gas. The free energy \( G \) of a gas at \( P \) atm pressure is given by:

\[
G = G^0 + RT \ln(P).
\]

Where \( G^0 \) is the free energy of the gas at a 1 atm pressure, \( T \) is the temperature, and \( R \) is the gas constant.

Write a main() program that receives as input the pressure, the free energy of the gas at a 1 atm pressure, the temperature, and the gas constant and outputs the free energy. The input is from the standard input, the keyboard. The results are sent to the standard output, the screen.

### Osmotic pressure

The osmotic pressure of a solution is determined by the molarity and the temperature of the solution. The osmotic pressure, \( \Pi \) is given by

\[ \Pi = MRT. \]

Where \( M \) is the molarity of the solute, \( T \) is the temperature, and \( R \) is the gas constant.

Write a main() program that receives as input the molarity of the solute, the temperature, and the gas constant and outputs the osmotic pressure. The input is from the standard input, the keyboard. The results are sent to the standard output, the screen.

### 1.5.2 Pressure Units

A fluid at rest exerts a force perpendicular to any surface with which it is in contact. The fluid on the two sides of a surface within the fluid exerts equal and opposite forces on the surface. Pressure at a
point in a fluid is the ratio of the normal force to a small area around that point to the area. If $F$ is the uniform force on all points on a plane surface with area $A$, then

$$p = \frac{F}{A}$$

Pressure is force per unit area.

The SI units for pressure in $1 \text{ N} / \text{m}^2$ which is given the name of 1 Pascal or 1 Pa where 1 N is 1 Newton = Kg m/ s s. There are however other units of pressure. In meteorology one bar is 100,000 Pa, and one milibar is 100 Pa.

These changes of scales can be represented as equations. Let the pressure be $P$ when measured in Pa, $b$ when measured in bar, and $mb$ when measured in milibar. Just as one Hour is 60 min, time $m$ in minutes is converted to $h$ in hours by DIVIDING $m$ by 60.

$$h = \frac{m}{60}.$$ 

Hence, the conversion between Pa and milibar is given by the equation:

$$mb = \frac{P}{100}.$$ 

The normal atmospheric pressure at sea level 1 atm is the pressure of the earth’s atmosphere at sea level which is 101325 Pa. Let the pressure be at when measured in atm. Hence, the conversion between Pa and atm is given by the equation:

$$atm = \frac{P}{101325}.$$ 

Petroleum engineers use two hybrid systems: Darcy and Field. Pressure unit in the Darcy system 1 atm. Pressure in Field units is psia where Force is in lb ft/(sXs) and area in square-feet. One atmosphere (1 atm) is 14.7 psia. Let the pressure be psi when measured in psia. Hence, the conversion between psia and atm is given by the equation:

$$at = \frac{psi}{14.7}.$$ 

**Pressure Units Lab**

1. Change directory into your cs2503Programs directory.

2. Write a C++ `main()` that reads the pressure of a fluid in Field units, prints the input value is psia, and reports the equivalent pressure in Pa, atm, bar. Save it in a file named `pressureScales.cpp` under your cs2503Programs directory.

3. Modify your program so the input is read from a file named `pressurepsia.txt`.

4. Modify your program so the output is written to a file named `pressureScales.txt`.

5. Once you have completed the program, Hand in `pressurepsia.txt`, `pressureScales.txt`, and `pressureScales.cpp`.

6. Copy the files `pressurepsia.txt`, `pressureScales.txt` to your public_html/cs2503Reports directory.

7. Change directory to your public_html/cs2503Reports directory, and create the report file `pressureScales.html` which explains the lab and includes relative links to the files `pressurepsia.txt`, `pressureScales.txt`.

8. Modify your cs2503.html to place a link to `pressureScales.html`.

10. Clean up your home directory and any of your public areas to make sure that no files containing your program or portion thereof are accessible by any user.