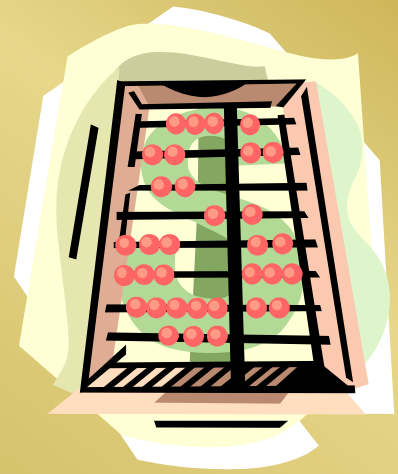
Unit 3 Lesson 4

How Data Types Affect Calculations

Dave Clausen

La Cañada High School



Type Compatibility

- Mixed mode expressions
  - expressions with different data types
    - int, char, double, etc. in the same expression
- Pascal and BASIC would give a Type Mismatch Error Message and quit.
- C+ + does not give any error messages for this.
- Extra care is necessary for mixed mode expressions.

Mixing Data Types

- C++ allows you to mix data types in calculations.
- Many programming languages do not allow the mixing of data types.
- This is because it can lead to errors if you do not understand the proper way to deal with mixed data types and the consequences of mixing them.

Code List 4-1

// share.cpp share.txt

```
#include <iostream>
int main ( )
{
    int number_of_people;          // declare number_of_people as an integer
    float money;                   // declare money as a float
    float share;                   // declare share as a float

    cout << “How many people need a share of the money? ”;
    cin >> number_of_people;
    cout << “How much money is available to share among the people? ”;
    cin >> money;

    share = money / number_of_people;          // number_of_people treated as float for this line of code.
    cout << “Give each person $” << share << ‘\n’;

    return 0;
}
```

Promotion

- In cases of mixed data types, the complier makes adjustments to produce the most accurate answer.
- In the program in Code List 4-1, for example, the integer value is temporarily converted to a float so that the fractional part of the variable “money” can be used in the calculation.
- This is called ***promotion***.
- The variable named number_of_people is not actually changed from type int to type float, the compiler temporarily converts it to type float for just this line of code.

Type Promotion

- Type promotion
  - converting a less inclusive data type into a more inclusive data type (i.e. int to double)
  - When adding an integer to a double, the compiler temporarily “converts” the integer to type double, adds, and gives an answer of type double.
  - int, char, and double are “compatible”

Implicit Type Conversions

int_var = double_var;	Truncates the decimals
double_var = int_var;	adds .0
int_var = char_var;	get ASCII code of char
char_var = int_var;	get the character whose ASCII code is the integer value
Ex.	
whole_num=‘A’ + 1;	66
digit = ‘5’ - ‘0’;	5

Code List 4-2

// losedata.cpp losedata.txt

```
#include <iostream.h>

int main ( )
{
      int answer, i;
      float x;

      i = 3;
      x = 0.5;
      answer = x *  i;   //answer holds truncated result

      cout << answer << '\n';
      return 0;
}
```

Truncation

- In Code List 4-2, the variable “i” is promoted to a float in the calculation statement.
- The answer is assigned to another variable of type int, which results in loosing the fractional part of the answer.
- The floating point number is ***truncated***, which means that the digits after the decimal point are deleted.

Typecasting

- Even though C++ handles the mixing of data types fairly well, unexpected results can occur.
- To give the programmer more control over the results when data types are mixed, C++ allows you to explicitly change one data type to another using operators called ***typecast operators***.
- Using a typecast operator is usually referred to as ***typecasting***.

Type Casts

- Type cast
  - an operation that a programmer can use to convert the data type
- Explicit type conversion
  - the use of an operation by the programmer to convert one type of data into another
- Form of type cast
  - <type name> (<expression>);
  - (<type name>) <expression>;

Type Cast Examples

truncate_pi = int (3.14);

convert_long_int = (long int) integer_var;

answer = double (numerator) / double (denominator);

Type casting can add clarity to your program while reminding you of the data types involved in your calculations.

Code List 4-4

// sharecast.cpp sharecast.txt

```
#include <iostream.h>
#include <conio.h>

int main()
{
  int number_of_people;          // declare number_of_people as an integer
  float money;                   // declare money as a float
  float share;                   // declare share as a float

  cout << "How many people need a share of the money? ";
  cin >> number_of_people;
  cout << "How much money is available to share among the people? ";
  cin >> money;
  share =  money / float (number_of_people);     //typecast number_of_people to float
  cout << "Give each person $" << share << endl;
  getch();
  return 0;
}
```

Overflow

- Overflow is the condition where as value becomes too large for its data type.
- The program in Code List 4-5 shows a simple example of overflow.
- The expression
  j = i + 2000;
  results in a value of 34000, which is too large for the short data type.
- No error message is generated, however the value of “j” becomes unpredictable.

Code List 4-5

// overflow.cpp overflow.txt

```
#include <iostream.h>

int main( )
{
    short i, j;

    i = 32000;
    j = i + 2000;        // The result (34000) overflows the short int type
    cout << j << '\n';
    return 0;
}
```

Integer Overflow

- Stored as binary numbers inside the computer.
- Integers produce exact answers
- Int_Min and Int_Max

  -32,768 and 32,767
- Integer Overflow
  - a number is too large or too small to store
  - no error message
  - unpredictable value

Underflow

- Underflow is similar to overflow.
- Underflow occurs in floating-point numbers when a number is too small for the data type.
- For example, 1.5×10^{-144} is too small to fit in the data type float. The compiler considers this value to be 0 (zero).

Code List 4-6

```
// unflow.cpp          unflow.txt

#include <iostream.h>

int main ( )
{
     float x;

     x = 1.5e-144;

     cout << x << endl;
     return 0;
}
```

Scientific Notation

- Scientific or Exponential Notation allows us to represent very large or very small numbers, for example:

```
mass_of_electron = 9.109e-31;  //kilograms
speed_of_light = 2.997e9; //meters per second
estimated_grains_of_sand_on_earth = 1.0e24;
```

Code List 4-7

```
// floaterr.cpp          floaterr.txt

#include <iostream.h>   // necessary for cout command

int main ( )
{
     float x, y;

     x = 3.9e10 + 500.0;
     y = x – 3.9e10;                  //results vary from compiler to compiler

     cout << y << '\n';
     return 0;
}
```

Floating Point Rounding Errors

- In Code List 4-7, it would stand to reason that adding 500 to a number and then subtracting 500 from that result would give us the original number.
- If our number is very large, and 500 is relatively small, then we will get unpredictable results which is known as Floating Point Rounding Error.
- How this is dealt with varies from compiler to compiler in the handling of such errors.
- The data type float is only accurate to approximately 7 significant digits. In our example, the 5 from 500 gets lost since it is beyond the number of significant digits that the data type float can handle.