MA122 - Computer Programming and Apllications

Indian Institute of Space Science and Technology

January 25, 2017

Lecture 6

A glimpse of Operator Overloading

type `int` /type `int`

```
9 / 5
```

operator performs *int* division

type `long` /type `long`

```
9L / 5L
```

operator performs *long* division

type `double` /type `double`

```
9.0 / 5.0
```

operator performs *double* division

type `float` /type `float`

```
9.0f / 5.0f
```

operator performs *float* division

The Modulus Operator


```
1  //uses % operator to convert lbs to stone
2  #include <iostream>
3  int main()
4  {
5    using namespace std;
6    const int Lbs_per_stn = 14;
7    int lbs;
8    cout << "Enter your weight in pounds: ";
9    cin >> lbs;
10
11   int stone = lbs / Lbs_per_stn;   // whole stone
12   int pounds = lbs % Lbs_per_stn;  // remainder in
         pounds
13
14   cout << lbs << " pounds are " << stone
15   << " stone, " << pounds << " pound(s).\n";
16
17 return 0; }
```

Lecture 6

Potential problems

Table 3.3 **Potential Numeric Conversion Problems**

Conversion Type	Potential Problems
Bigger floating-point type to smaller floating-point type, such as `double` to `float`	Loss of precision (significant figures); value might be out of range for target type, in which case result is undefined.
Floating-point type to integer type	Loss of fractional part; original value might be out of range for target type, in which case result is undefined.
Bigger integer type to smaller integer type, such as `long` to `short`	Original value might be out of range for target type; typically just the low-order bytes are copied.

Type changes on Initialization


```
1  // init.cpp -- type changes on initialization
2  #include <iostream>
3  int main()
4  {
5    using namespace std;
6  //  cout.setf(ios_base::fixed, ios_base::floatfield);
7
8    float tree = 3;    // int converted to float
9    int guess(3.9832); // double converted to int
10   int debt = 7.2E12; // result not defined in C++
11
12   cout << "tree = " << tree << endl;
13   cout << "guess = " << guess << endl;
14   cout << "debt = " << debt << endl;
15
16   return 0;
17 }
```

Initialization Conversions when {} are used (C++11)

- *list-initialization*: doesn't permit narrowing, which is when the type of the variable may not be able to represent the assigned value.

```
1  #include <iostream>
2  int main()
3  {
4    const int code = 66;
5    int x = 66;
6    char c1 {31325}; // narrowing, not allowed
7    char c2 = {66}; // allowed because char can hold 66
8    char c3 {code}; // ditto
9
10   char c4 = {x};  // not allowed, x is not constant
11   x = 31325;
12   char c5 = x; // allowed (not a list-initialization)
13
14   return 0; }
```

Lecture 6

Forcing type changes


```
1  // typecast.cpp -- forcing type changes
2  #include <iostream>
3  int main()
4  {
5  using namespace std;
6  int auks, bats, coots;
7  // the following statement adds the values as double,
8  // then converts the result to int
9
10 auks = 19.99 + 11.99;
11 // these statements add values as int
12
13 bats = (int) 19.99 + (int) 11.99; // old C syntax
14
15 coots = int (19.99) + int (11.99); // new C++ syntax
```

Forcing type changes


```
1
2
3  cout << "auks = " << auks << ", bats = " << bats;
4  cout << ", coots = " << coots << endl;
5
6  char ch = 'Z';
7  cout << "The code for " <<ch << " is "; // print as
       char
8
9  cout << int(ch) << endl;                  // print as
       int
10 cout << "Yes, the code is ";
11
12
13
14 return 0;
15 }
```