Week# 4
Type Conversion, Bitwise, Logical, and Relational Operators

Dr Taimur Ahmed

Department of IT & CS, PAF-IAST

Lecture# 9
Type Conversion, Overflow & Underflow, Type Casting

Type Conversion

- Operations perform **between operands of same type**
- C++ **may convert** one data type to another during an operation, if both types are not same
- This can **affect** the expected result

Type Conversion - Types

❑ Implicit Type Conversion

- ➢ Also known as **automatic type conversion**
- ➢ Complier converts data types on its own without any external instructions

❑ Explicit Type Conversion

- ➢ Also known as **Type Casting**
- ➢ Programmer **enforces** the type conversion

Implicit Type Conversion

Implicit Type Conversion

- Example – assigning `int` to `bool`

```
bool n = 20;
```

Program Output:

```
1
```

```
1  /* Lecture 9: Type Converstion
2      Implicit Type Conversion
3  */
4
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     bool n = 20;
11
12     cout<< n;
13
14     return 0;
15 }
```

Implicit Type Conversion

- Example – assigning `bool` to `int`

```
int m = n; // n is bool
```

Program Output:

```
1
```

```
1  /* Lecture 9: Type Converstion
2      Implicit Type Conversion
3  */
4
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     bool n = 20;
11
12     int m = n;
13
14     cout<< m;
15
16     return 0;
17 }
```

Implicit Type Conversion

- Example – assigning `float` to `int`

```
int m = 4.55;
```

Program Output:

```
4
```

Note that fractional part is truncated

```
1   /* Lecture 9: Type Converstion
2       Implicit Type Conversion
3   */
4
5   #include <iostream>
6   using namespace std;
7
8   int main()
9   {
10      int m = 4.55;
11
12      cout<< m;
13
14      return 0;
15  }
```

Implicit Type Conversion

- Example – assigning `int` to `float`

```
double f = m; //m is int
```

Program Output:

```
4
```

Note that fractional part is truncated, though `f` is `float`

```
1   /* Lecture 9: Type Converstion
2       Implicit Type Conversion
3   */
4
5   #include <iostream>
6   using namespace std;
7
8   int main()
9   {
10      int m = 4.55;
11
12      double f = m;
13
14      cout<< f;
15
16      return 0;
17  }
```

Implicit Type Conversion

- Example – assigning `int` to `char`

```
unsigned char c = -1;
```

Program Output:

Note that output of this program is blank character with integer value of 255 so you will not see any character on screen. **Binary of $(-1)_{10}$ = $(1111\ 1111)_2$ = $(255)_{10}$**

```
1   /* Lecture 9: Type Converstion
2      Implicit Type Conversion
3   */
4
5   #include <iostream>
6   using namespace std;
7
8   int main()
9   {
10      unsigned char c = -1;
11
12      cout<< c;
13
14      return 0;
15  }
```

Hierarchy of Types

- **Data types** are categorized in following hierarchical order with respect to the **largest number** they can hold

Highest

```
long double
double
float
unsigned long
long
unsigned int
int
```

Lowest

Type Coercion

- **Type Coercion** defines the process of **automatic conversion** of an operand to another data type
- **Promotion** is process when an operand of **lower data type** is converted to **higher data type**
- **Demotion** is process when an operand of **higher data type** is converted to **lower data type**

Coercion Rules

- `char, short, unsigned short` **automatically promoted** to `int`
- When operating on values of **different data types**, the lower one is promoted to the type of the higher one
- When using the `=` operator, the type of **expression on right will be converted to type of variable on left**

Explicit Type Conversion

Explicit Type Conversion

- Explicit type conversion is also known as **Type Casting**
- Data type of a variable is **manually** converted by the programmer
- Useful for floating point multiplication/division using `int` variables

```
double m;
int y1,y2,x1,x2;
m = static_cast<double>(y2-y1)/(x2-x1);
```

- Useful to see `int` value of a `char` variable

```
char ch = 'C';
cout << ch << " is "
     << static_cast<int>(ch); //prints 67
```

Explicit Type Conversion

```
1   // Lecture 9: Type Casting
2   #include <iostream>
3   using namespace std;
4
5   int main()
6   {
7       int books, months;
8       double perMonth;
9
10      cout << "How many books you plan to read? \t";
11      cin >> books;
12      cout << "How many months will it take? \t";
13      cin >> months;
14      perMonth = static_cast<double>(books)/months; //books is int
15      cout << "That is \t" <<perMonth << "\t books per month.";
16
17      return 0;
18  }
```

Program output

```
How many books you plan to read? 30
How many months will it take? 7
That is  4.28571 books per month.
```

Type Cast Expressions

- **C-Style cast**
  - Syntax of type casting in C – data type name in ( )

```
cout << ch << " is "  << (int)ch;
```

- **Pre-standard C++ cast**
  - Syntax of type casting in earlier versions of C++ compilers – value in ( )

```
cout << ch << " is "  << int(ch);
```

- **Standard C++** supports both syntax but `static_cast` is recommended

```
static_cast <data type> (variable/value to be changed)
```

Overflow and Underflow

Overflow and Underflow

- **Overflow** happens when a very large value is assigned to a variable which is not expected to hold it

Data Type	Byte	Size
short int	2	singed: -32,768 to 32,767 unsigned: 0 to 65,535

```
Program Output:

-32768
```

```
1   // Lecture 9: Overflow and Underflow
2   // Example of Overflow
3
4   #include <iostream>
5   using namespace std;
6
7   int main()
8   {
9       short int j = 32767;
10      j+=1;
11      cout<< j;
12
13      return 0;
14  }
```

Overflow and Underflow

- **Underflow** occurs when a too small value is assigned to a variable which is lower than its minimum limit

Data Type	Byte	Size
short int	2	singed: -32,768 to 32,767 unsigned: 0 to 65,535

```
Program Output:

32767
```

```
1   //Lecture 9: Overflow and Underflow
2   // Example of Underflow
3
4   #include<iostream>
5   using namespace std;
6
7   int main()
8    {
9       short int i = -32768; //minimum value
10      i -= 1;                // = -32768 - 1
11      cout << i;
12
13      return 0;
14  }
```

Overflow and Underflow

- When Overflow or Underflow occurs during program execution then
  - Compiler may display **warning/error message** **but not in all compilers**
  - **Stops** the program
  - Program can continue with **incorrect value**