# ENGINEERING PROGRAMMING I

# Chapter 03

2017

# Topics

3.3 Implicit Type Conversion
3.4 Explicit Type Conversion
3.5 Overflow and Underflow
3.6 Named Constants
3.7 Multiple and Combined Assignment
3.8 Formatting Output
3.9 Working with Characters and String Objects
3.10 Using C-Strings
3.11 More Mathematical Library Functions
3.12 Introduction to Files

# Implicit Type Conversion

- Operations are performed between operands of the same type
- If not of the same type, C++ will automatically convert one to be the type of the other
- This can impact the results of calculations

# Hierarchy of Data Types

- Highest

```
long double
double
float
unsigned long
long
unsigned int
int
unsigned short
short
char
```

- Lowest
- Ranked by largest number they can hold

# Type Coercion

- Coercion: automatic conversion of an operand to another data type
- Promotion: converts to a higher type
- Demotion: converts to a lower type

# Coercion Rules

1) **`char`**, **`short`**, **`unsigned short`** are automatically promoted to **`int`**

```
eg) int X = 'a';
    cout << X << endl;  // prints out 97
```

2) When operating on values of different data types, the lower one is promoted to the type of the higher one.

3) When using the = operator, the type of expression on right will be converted to the type of variable on left

# ASCII (American Standard Code for Information Interchange)

# ASCII TABLE

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [ENG OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

# Convert char to int

- ```
  char ch;
  cin >> ch;

  int n;
  if (ch >= '0' && ch <= '9'){
      n = ch - '0';
  }
  ```
  ```

# Explicit Type Conversion

- Also called type casting
- Used for manual data type conversion
- Format

  **static_cast<*type*>(*expression*)**

- Example:

```
cout << static_cast<char>(65); // Displays 'A'
cout << (char) 65;
cout << char(65);
```

# More Type Casting Examples

```
char ch = 'C';
cout << ch << " is stored as "
     << static_cast<int>(ch);

gallons = static_cast<int>(area/500);

avg = static_cast<double>(sum)/count;
```

# Older Type Cast Styles

```
double Volume = 21.58;
int intVol1, intVol2;
intVol1 = (int) Volume; // C-style cast

intVol2 = int (Volume); //Prestandard
                        // C++ style cast
```

C-style cast uses prefix notation

Prestandard C++ cast uses functional notation

`static_cast` is the current standard

# Overflow and Underflow

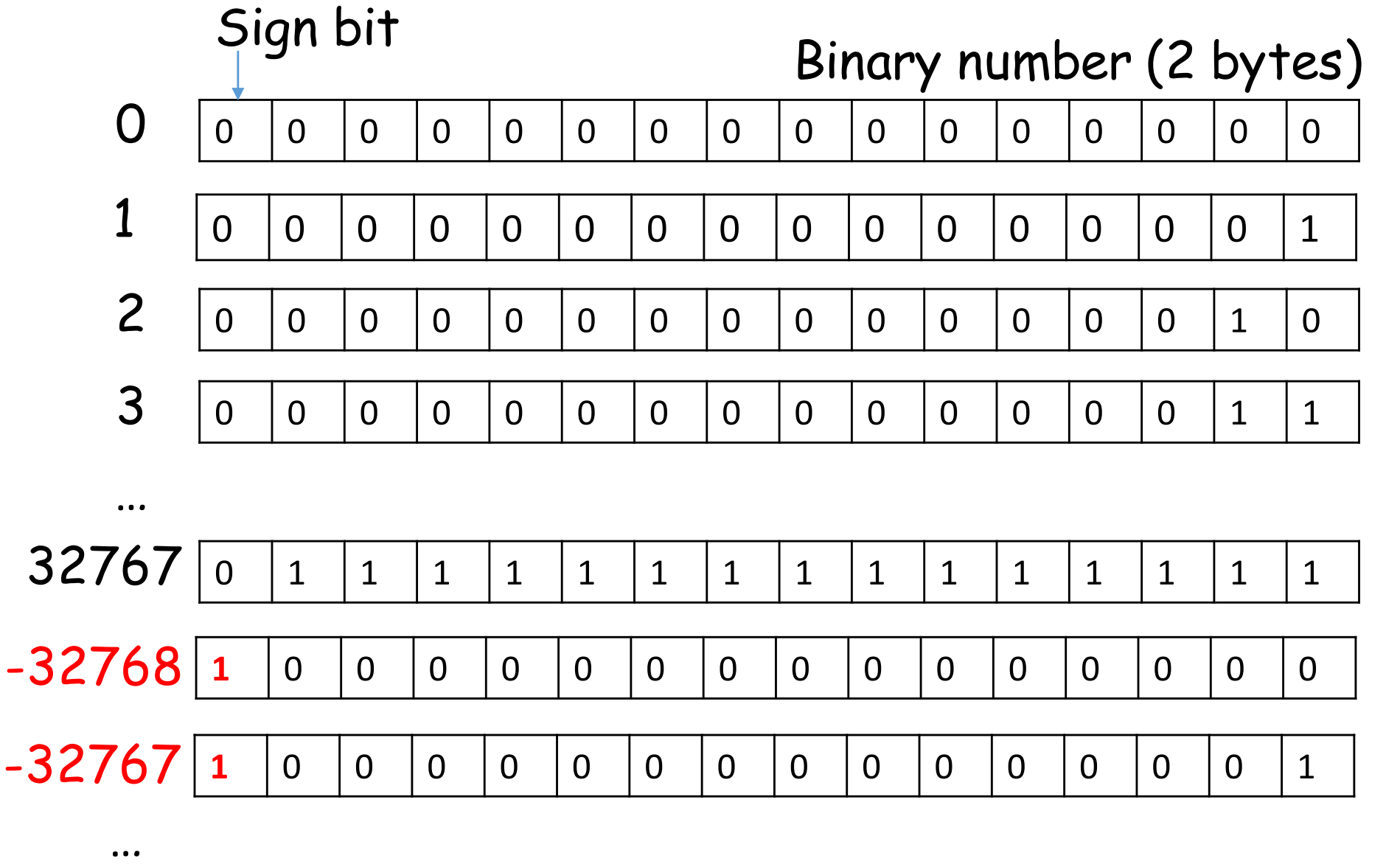
- Occurs when assigning a value that is too large (overflow) or too small (underflow) to be held in a variable
- The variable contains a value that is 'wrapped around' the set of possible values

# Overflow Example

```
// Create a short int initialized to
// the largest value it can hold
short int num = 32767;

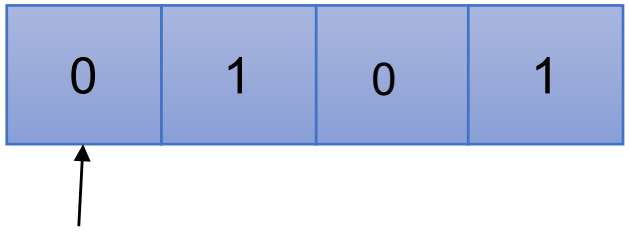
cout << num;      // Displays 32767
num = num + 1;
cout << num;      // Displays -32768
```

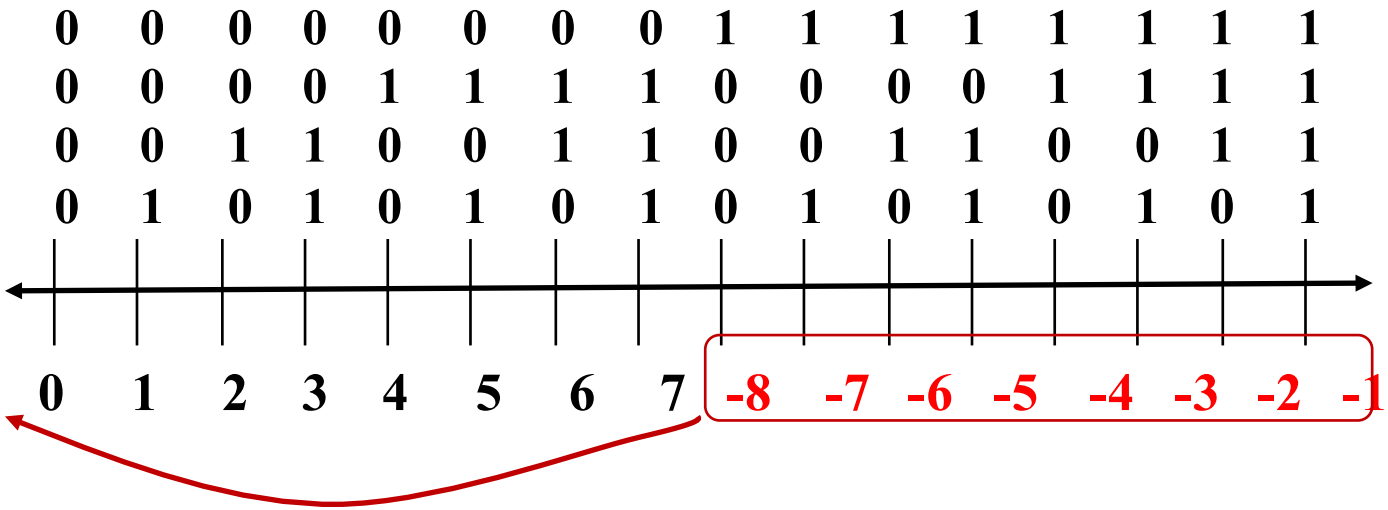
# Overflow


Sign bit
Binary number (2 bytes)
0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
...
32767
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
-32768
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-32767
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
...

# Signed Number (Two's Complement)

```
0000 = 0
0001 = 1
0010 = 2
0011 = 3
0100 = 4
0101 = 5
0110 = 6
0111 = 7
1000 = -8
1001 = -7
1010 = -6
1011 = -5
1100 = -4
1101 = -3
1110 = -2
1111 = -1
```




2's complement can be obtained by taking the 1's complement and adding one to the least significant bit.

# Named Constants

- Also called constant variables
- Variables whose content cannot be changed during program execution
- Used for representing constant values with descriptive names

```
const double TAX_RATE = 0.0675;
const int NUM_STATES = 50;
```

- Often named in uppercase letters

# const vs. #define

**#define**

- C-style of naming constants

```
#define NUM_STATES 50
```


- Interpreted by pre-processor rather than compiler
- Does not occupy a memory location like a constant variable defined with **const**
- Instead, causes a text substitution to occur. In above example, every occurrence in program of **NUM_STATES** will be replaced by **50**

# Multiple and Combined Assignment

- The assignment operator (=) can be used more than 1 time in an expression

```
x = y = z = 5;
```

- Associates right to left

```
x = (y = (z = 5));
```

Done 3rd

Done 2nd

Done 1st

# Combined Assignment

- Applies an arithmetic operation to a variable and assigns the result as the new value of that variable
- Operators: `+=` `-=` `*=` `/=` `%=`
- Example:
- `sum += amt;` is short for `sum = sum + amt;`

# More Examples

```
x += 5;   means   x = x + 5;
x -= 5;   means   x = x - 5;
x *= 5;   means   x = x * 5;
x /= 5;   means   x = x / 5;
x %= 5;   means   x = x % 5;
```

The right hand side is evaluated before the combined assignment operation is done.

```
x *= a + b;  means  x = x * (a + b);
```

# Formatting Output

- Can control how output displays for numeric and string data
  - size
  - position
  - number of digits
- Requires **`iomanip`** header file

# Stream Manipulators

- Used to control features of an output field
- Some affect just the next value displayed
  - `setw(x)` : Print in a field at least **x** spaces wide. Use more spaces if specified field width is not big enough.

# Stream Manipulators

- Some affect values until changed again
  - `fixed`: Use decimal notation (not E-notation) for floating-point values.
  - `setprecision(x)`:
    - When used with `fixed`, print floating-point value using `x` digits after the decimal.
    - Without `fixed`, print floating-point value using `x` significant digits.
  - `showpoint`: Always print decimal for floating-point values.
  - `left, right`: left-, right justification of value

# Manipulator Examples

```
const float e = 2.718;
float price = 18.0;
cout << setw(8) << e << endl;
cout << left << setw(8) << e
     << endl;
cout << setprecision(2);
cout << e << endl;
cout << fixed << e << endl;
cout << setw(6) << price;
```

Displays

```
^^^2.718

2.718^^^

2.7
2.72
^18.00
```

# Working with Characters and String Objects

- `char`: holds a single character
- `string`: holds a sequence of characters
- Both can be used in assignment statements
- Both can be displayed with `cout` and <<

# String Input

Reading in a string object

```
string str;

cin >> str;              // Reads in a string
                         // with no blanks

getline(cin, str);       // Reads in a string
                         // that may contain
                         // blanks
```

# Character Input

Reading in a character

```
char ch;

cin >> ch;    // Reads in any non-blank char

cin.get(ch); // Reads in any char

cin.ignore(); // Skips over next char in
              // the input buffer
```

# String Operators

= Assigns a value to a string

```
string words;
words = "Tasty ";
```

+ Joins two strings together

```
string s1 = "hot", s2 = "dog";
string food = s1 + s2; // food = "hotdog"
```

+= Concatenates a string onto the end of another one

```
words += food; // words now = "Tasty hotdog"
```

# Using C-Strings

- C-string is stored as an array of characters
- Programmer must indicate maximum number of characters at definition

```
const int SIZE = 5;
char temp[SIZE] = "Hot";
```

- NULL character (\0) is placed after final character to mark the end of the string

- Programmer must make sure array is big enough for desired use; `temp` can hold up to 4 characters plus the \0.

| H | o | t | \0 | |
|---|---|---|---|---|

# C-String and Null

- `char temp[3] = "hot"; //what is wrong?`

- A string literal contains a Null at the end

- ```
  char temp[3];
  temp[0] = 'h';
  temp[1] = 'o';
  temp[2] = 't';
  cout << temp << endl;
  ```
  ```

# C-String Input

- Reading in a C-string

```
const int SIZE = 10;
char Cstr[SIZE];
cin >> Cstr;     // Reads in a C-string with no
                 // blanks. Will write past the
                 // end of the array if input string
                 // is too long.
cin.getline(Cstr, 10);
                 // Reads in a C-string that may
                 // contain blanks. Ensures that <= 9
                 // chars are read in.
```

- Can also use **`setw()`** and **`width()`** to control input field widths

# C-String Initialization vs. Assignment

- A C-string can be initialized at the time of its creation, just like a string object

```
const int SIZE = 10;

char month[SIZE] = "April";
```

- However, a C-string cannot later be assigned a value using the = operator; you must use the `strcpy()` function

```
char month[SIZE];
month = "August"              // wrong!
strcpy(month, "August"); //correct
```

# Mathematical Library Functions

- These require cmath header file
- Take double arguments and return a double
- Commonly used functions

| | |
|---|---|
| `abs` | Absolute value |
| `sin` | Sine |
| `cos` | Cosine |
| `tan` | Tangent |
| `sqrt` | Square root |
| `log` | Natural (e) log |

# Introduction to Files

- Can use a file instead of keyboard for program input
- Can use a file instead of monitor screen for program output
- Files are stored on secondary storage media, such as disk
- Files allow data to be retained between program executions

# What is Needed to Use Files

1. Include the **`fstream`** header file
2. Define a file stream object
   - **`ifstream`** for input from a file

     ```
     ifstream inFile;
     ```
   - **`ofstream`** for output to a file

     ```
     ofstream outFile;
     ```

# Open the File

3. Open the file

- Use the **open** member function

```
inFile.open("inventory.dat");
outFile.open("report.txt");
```

- Filename may include drive, path info.
- Output file will be created if necessary; existing output file will be erased first
- Input file must exist for **open** to work

# Use the File

4. Use the file

- Can use output file object and << to send data to a file

```
outFile << "Inventory report";
```

- Can use input file object and >> to copy data from file to variables

```
inFile >> partNum;
inFile >> qtyInStock >> qtyOnOrder;
```

# Close the File

5. Close the file

- Use the `close` member function

```
inFile.close();
outFile.close();
```

- Don't wait for operating system to close files at program end
  - May be limit on number of open files
  - May be buffered output data waiting to be sent to a file that could be lost

# Chapter 3: Expressions and Interactivity

Starting Out with C++
Early Objects
Seventh Edition

by Tony Gaddis, Judy Walters,
and Godfrey Muganda

Addison-Wesley
is an imprint of
PEARSON