Lecture 3
Some predefined Functions, Characters, Strings and simple file management

Chapter: 3.2, 3.2.3, 4.11.3 (static_cast), 8.1, 8.2

Objectives

- To be able to use predefined functions.
- To represent characters using the **char** type.
- To encode characters using ASCII code.
- To read a character from the keyboard.
- To represent special characters using the escape sequences.
- To cast a numeric value to a character and cast a character to an integer.
- To compare and test characters.
- To test and convert characters using the C++ character functions.
- To convert a hexadecimal character to a decimal value (**HexDigit2Dec**).
- To represent strings using the **string** type and introduce objects and instance functions.
- To use the subscript operator for accessing and modifying characters in a string.
- To use the **+** operator to concatenate strings.
- To compare strings using the relational operators.
- To read strings from the keyboard.
- To read and write data from/to a file.

Rounding Functions

Function	Description
`ceil(x)`	x is rounded up to its nearest integer. This integer is returned as a double value.
`floor(x)`	x is rounded down to its nearest integer. This integer is returned as a double value.

PreDefined.cpp

The min, max, and abs Functions

- max(**2**, **3**) returns **3**
- max(**2.5**, **3.1**) returns **3.1**
- min(**2.5**, **3.1**) returns **2.5**
- abs(**-2**) returns **2**
- abs(**-2.5**) returns **2.5**

PreDefined.cpp

Character Data Type

char letter = 'A'; (ASCII)

char numChar = '4'; (ASCII)

NOTE: The increment and decrement operators can also be used on char variables to get the next or preceding character. For example, the following statements display character b.

```
char ch = 'a';
cout << ++ch;
```

Read Characters

To read a character from the keyboard, use

```
cout << "Enter a character: ";
char ch;
cin >> ch;
```

Casting between char and Numeric Types

```
int i = 'a';
// Same as int i = static_cast<int>('a');
//old-way:
  int i= (int )('a');
char c = 97;
// Same as char c = static_cast<char>(97);
//old-way:
char c = static_cast<char>(97);
```

Numeric Operators on Characters

The char type is treated as if it is an integer of the byte size. All numeric operators can be applied to char operands. A char operand is automatically cast into a number if the other operand is a number or a character. For example, the following statements

```
int i = '2' + '3'; // (int)'2' is 50 and (int)'3' is 51
cout << "i is " << i << endl; // i is decimal 101
int j = 2 + 'a'; // (int)'a' is 97
cout << "j is " << j  << endl;
cout << j << " is the ASCII code for character " <<
static_cast<char>(j) << endl;
```

Display

i is 101
j is 99
99 is the ASCII code for character **c**

Note

It is worthwhile to note that the ASCII for lowercase letters are consecutive integers starting from the code for 'a', then for 'b', 'c', ..., and 'z'.

The same is true for the uppercase letters.

Furthermore, the ASCII code for 'a' is greater than the code for 'A'. So 'a' - 'A' is the same as 'b' - 'B'.

For a lowercase letter *ch*, its corresponding uppercase letter is static_cast<char>('A' + (ch - 'a')).

Problem: Converting a Lowercase to Uppercase

Write a program that prompts the user to enter a lowercase letter and finds its corresponding uppercase letter.

ToUppercase

Comparing and Testing Characters

Two characters can be compared using the comparison operators just like comparing two numbers. This is done by comparing the ASCII codes of the two characters.

'a' < 'b' is true because the ASCII code for **'a'** (**97**) is less than the ASCII code for **'b'** (**98**).
'a' < 'A' is false because the ASCII code for **'a'** (**97**) is greater than the ASCII code for **'A'** (**65**).
'1' < '8' is true because the ASCII code for **'1'** (**49**) is less than the ASCII code for **'8'** (**56**).

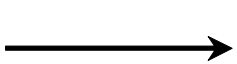
Case Study: Generating Random Characters

Computer programs process numerical data and characters. You have seen many examples that involve numerical data. It is also important to understand characters and how to process them.

Every character has a unique ASCII code between 0 and 127. To generate a random character is to generate a random integer between 0 and 127. You can use the srand(seed) function to set a seed and use rand() to return a random integer. You can also use it to write a simple expression to generate random numbers in any range. For example,

Case Study: Generating Random Characters, cont.

```
rand() % 10
```



Returns a random integer between 0 and 9.

```
50 + rand() % 50
```

Returns a random integer between 50 and 99.

```
a + rand() % b
```

Returns a random number between a and a + b, excluding a + b.

DisplayRandomCharacter

Character Functions

Function	Description
`isdigit(ch)`	Returns true if the specified character is a digit.
`isalpha(ch)`	Returns true if the specified character is a letter.
`isalnum(ch)`	Returns true if the specified character is a letter or digit.
`islower(ch)`	Returns true if the specified character is a lowercase letter.
`isupper(ch)`	Returns true if the specified character is an uppercase letter.
`isspace(ch)`	Returns true if the specified character is a whitespace character.
`tolower(ch)`	Returns the lowercase of the specified character.
`toupper(ch)`	Returns the uppercase of the specified character.

CharacterFunctions

Case Study: Converting a Hexadecimal Digit to a Decimal Value

Write a program that converts a hexadecimal digit into a decimal value.

HexDigit2Dec

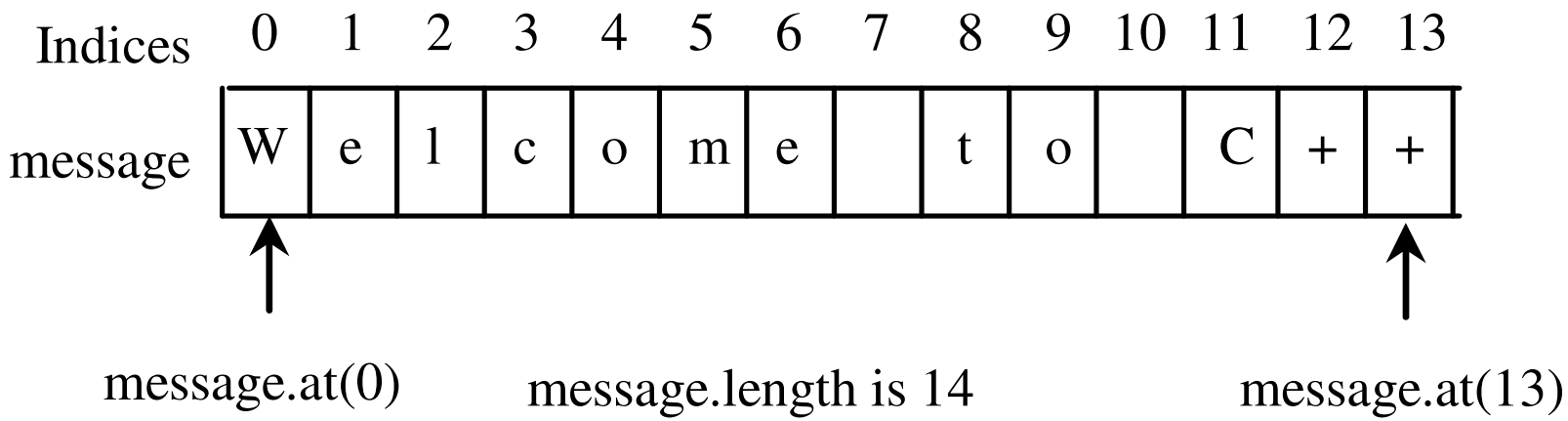
The string Type

string s;

string message = **"Programming is fun"**;

Function	Description
`length()`	Returns the number of characters in this string.
`size()`	Same as length();
`at(index)`	Returns the character at the specified index from this string.

String Subscript Operator



For convenience, C++ provides the subscript operator for accessing the character at a specified index in a string using the syntax **stringName[index]**. You can use this syntax to retrieve and modify the character in a string.

String Subscript Operator

```
string s = "ABCD";
s[0] = 'P';
cout << s[0] << endl;
```

Concatenating Strings

```
string s3 = s1 + s2;

message += " and programming is fun";
```

Comparing Strings

You can use the relational operators **==**, **!=**, **<**, **<=**, **>**, **>=** to compare two strings. This is done by comparing their corresponding characters on by one from left to right. For example,

```
string s1 = "ABC";
string s2 = "ABE";
cout << (s1 == s2) << endl; // Displays 0 (means false)
cout << (s1 != s2) << endl; // Displays 1 (means true)
cout << (s1 > s2) << endl; // Displays 0 (means false)
cout << (s1 >= s2) << endl; // Displays 0 (means false)
cout << (s1 < s2) << endl; // Displays 1 (means true)
cout << (s1 <= s2) << endl; // Displays 1 (means true)
```

Reading Strings

```
1  string city;
2  cout << "Enter a city: ";
3  cin >> city; // Read to array city
4  cout << "You entered " << city << endl;
```

```
1  string city;
2  cout << "Enter a city: ";
3  getline(cin, city, '\n'); // Same as getline(cin, city)
4  cout << "You entered " << city << endl;
```

Example

Write a program that prompts the user to enter two cities and displays them in alphabetical order.

OrderTwoCities

Example

Write a program that prompts the user to enter firstname, lastname and a lucky number and displays them.

MixedInputs

Simple File Output

To write data to a file, first declare a variable of the **ofstream** type:

```
ofstream output;
```

To specify a file, invoke the **open** function from **output** object as follows:

```
output.open("numbers.txt");
```

Optionally, you can create a file output object and open the file in one statement like this:

```
ofstream output("numbers.txt");
```

To write data, use the stream insertion operator (**<<**) in the same way that you send data to the **cout** object. For example,

```
output << 95 << " " << 56 << " " << 34 << endl;
```

SimpleFileOutput

Simple File Input

To read data from a file, first declare a variable of the **ifstream** type:

```
ifstream input;
```

To specify a file, invoke the **open** function from **input** as follows:

```
input.open("numbers.txt");
```

Optionally, you can create a file output object and open the file in one statement like this:

```
ifstream input("numbers.txt");
```

To read data, use the stream extraction operator (**>>**) in the same way that you read data from the **cin** object. For example,

```
input >> score1  >> score2 >> score3;
```

SimpleFileInput