CSCE 121

Introduction to Program Design & Concepts

The string class and data representation

Dr. Tim McGuire

Grateful acknowledgment to Dr. Philip Ritchey and Dr. Michael Moore for some of the material on which these slides are based.

The C++ string Class

The C++ `string` Class

- Special data type supports working with strings
- `#include <string>`
- Can define `string` variables in programs:

```
string firstName, lastName;
```

- Can receive values with assignment operator:

```
firstName = "Rock";
lastName = "GoodAg";
```

- Can be displayed via `cout`

```
cout << firstName << " " << lastName;
```

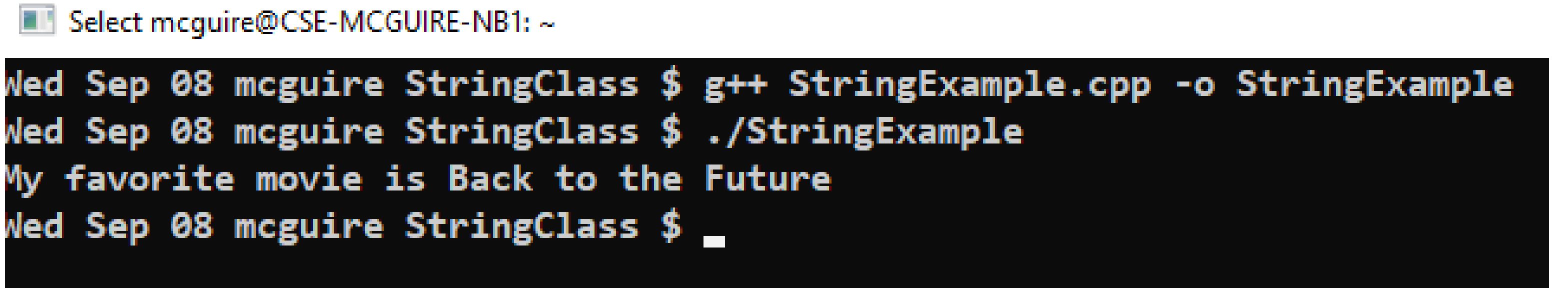
The `string` class in a Program

```
// This program demonstrates the string class.
#include <iostream>
#include <string> // Required for the string class.
using std::cout;

int main()
{
   string movieTitle;

   movieTitle = “Back to the Future";
   cout << "My favorite movie is " << movieTitle << endl;
   return 0;
}
```

Program Output:



```
Wed Sep 08 mcguire StringClass $ g++ StringExample.cpp -o StringExample
Wed Sep 08 mcguire StringClass $ ./StringExample
My favorite movie is Back to the Future
Wed Sep 08 mcguire StringClass $ _
```

Working with **`string`** Objects

- Using **`cin`** with the >> operator to input strings can cause problems:
- It passes over and ignores any leading *whitespace characters (spaces, tabs, or line breaks)*
- To work around this problem, you can use a C++ function named **`getline`**.

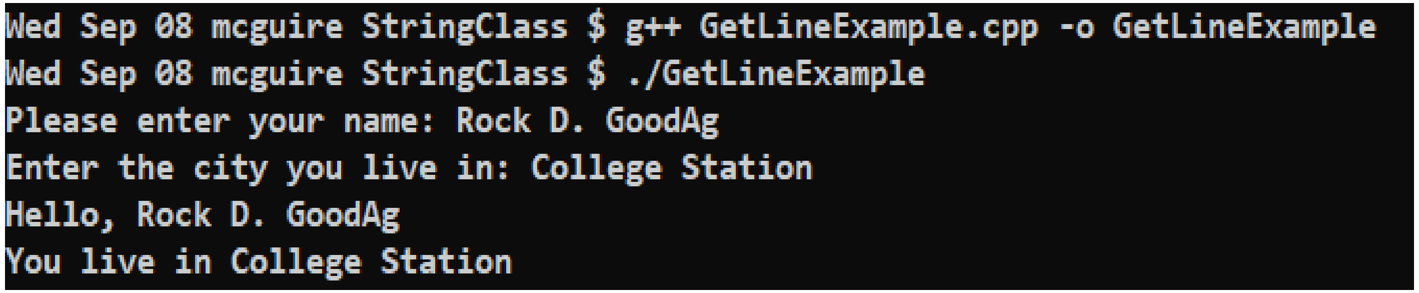
Using **getline** in a program

```
// This program demonstrates using the getline function
// to read character data into a string object.
#include <iostream>
#include <string>
using namespace std;

int main()
{
        string name;
        string city;

        cout << "Please enter your name: ";
        getline(cin, name);
        cout << "Enter the city you live in: ";
        getline(cin, city);

        cout << "Hello, " << name << endl;
        cout << "You live in " << city << endl;
        return 0;
}
```



```
Wed Sep 08 mcguire StringClass $ g++ GetLineExample.cpp -o GetLineExample
Wed Sep 08 mcguire StringClass $ ./GetLineExample
Please enter your name: Rock D. GoodAg
Enter the city you live in: College Station
Hello, Rock D. GoodAg
You live in College Station
```

Working with Characters and **string** Objects

- Mixing cin >> and getline() can be tricky, because cin >> leaves the newline in the input, while getline() does not skip leading whitespace.
- To skip over unneeded characters that are still in the keyboard buffer, use cin.ignore():

```
cin.ignore();          // skip next char
cin.ignore(10, '\n'); // skip the next 10 char. or until a '\n'
```

`string` Member Functions and Operators

- To find the length of a string:

```
string state = "Texas";
int size = state.length();
```

- To concatenate (join) multiple strings:

```
greeting2 = greeting1 + name1;
greeting1 = greeting1 + name2;
```

Or using the **+=** combined assignment operator:

```
greeting1 += name2;
```

Comparing `string` Objects

- Strings are compared using their ASCII values

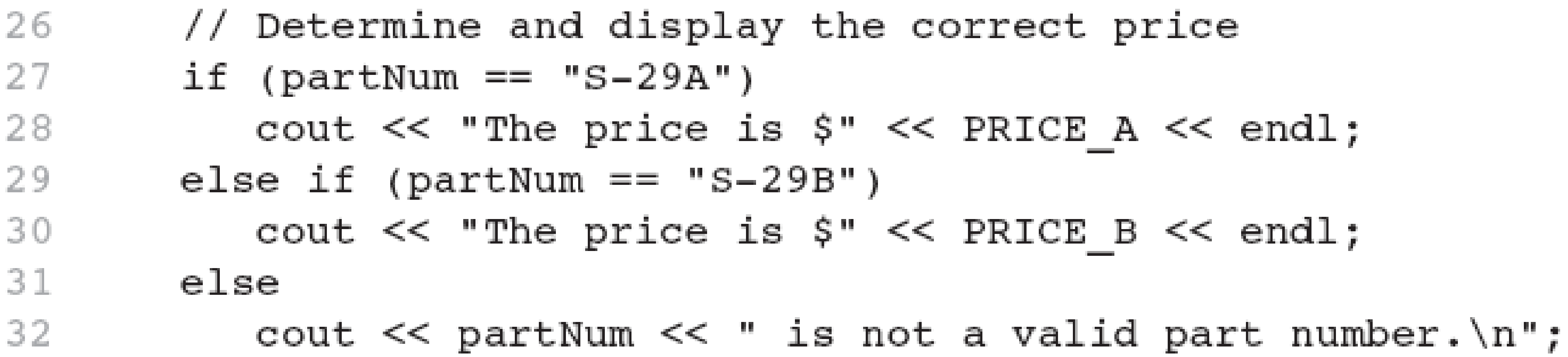
```
string name1 = "Mary";
string name2 = "Mark";

name1 > name2   // true
name1 <= name2 // false
name1 != name2  // true

name1 < "Mary Jane" // true
```

The characters in each string must match before they are equal

Relational Operators Compare Strings



```
// Determine and display the correct price
if (partNum == "S-29A")
   cout << "The price is $" << PRICE_A << endl;
else if (partNum == "S-29B")
   cout << "The price is $" << PRICE_B << endl;
else
   cout << partNum << " is not a valid part number.\n";
```

See StringCompare.cpp for full working code

`string` Operators

OPERATOR	MEANING
`>>`	extracts characters from stream up to whitespace, insert into string
`<<`	inserts string into stream
`=`	assigns string on right to string object on left
`+=`	appends string on right to end of contents on left
`+`	concatenates two strings
`[]`	references character in string using array notation (It is safer to us the at() method, however)

`>, >=, <,`
`<=, ==, !=`
	relational operators for string comparison. Return **`true`** or **`false`**

`string` Operators Example

```
string word1, phrase;
string word2 = " Dog";
cin >> word1; // user enters "Hot Tamale"
              // word1 has "Hot"
phrase = word1 + word2; // phrase has "Hot Dog"
phrase += " on a bun";
for (int i = 0; i < phrase.length(); i++)
     cout << phrase.at(i); // displays "Hot Dog on a bun"
```

`string` Member Functions

- Categories:
  - assignment: `assign, copy, data`
  - modification: `append, clear, erase, insert, replace, swap`
  - space management: `capacity, empty, length, resize, size`
  - substrings: `find, substr`
  - comparison: `compare`
- And several more

Member Function length

- The string class member function length returns the number of characters in the string object:
  - Example:

```
int n = stringVar.length( );
```

Member Function at

- at is an alternative to using [ ]'s to access characters in a string.
  - at checks for valid index values
  - Example:

```
string str("Mary");
cout << str[6] << endl;
cout << str.at(6) << endl;
str[2] = 'X';
str.at(2) = 'X';
```

Equivalent

Equivalent

string class to numbers

- C++ has functions to convert a string class object to a number

```
int i;
double d;
string s;
i = stoi("35");  // Converts the string "35" to an integer 35
d = stod("2.5"); // Converts the string "2.5" to the double 2.5
```

- C++ has functions to convert a string class object to a number

```
string s = to_string(1.2*2);  // “2.4” stored in s
```

Finding in a string

find()	**find**(*item*) returns index of first item occurrence, else returns string::npos (a constant defined in the string library). *Item* may be char, string variable, string literal (or char array). **find**(*item*, *indx*) starts at index *indx*.

```
// userText is "Help me!"
userText.find('p') // Returns 3
userText.find('e') // Returns 1 (first occurrence of e only)
userText.find('z') // Returns string::npos
userText.find("me") // Returns 5
userText.find('e', 2) // Returns 6 (starts at index 2)
```

Finding in a string

find()	**find**(*item*) returns index of first item occurrence, else returns string::npos (a constant defined in the string library). *Item* may be char, string variable, string literal (or char array). **find**(*item*, *indx*) starts at index *indx*.

Getting a substring

substr()	***substr***(index, length) returns substring starting at *index* and having *length* characters.

```
// userText is "http://google.com"
userText.substr(0, 7)                    // Returns "http://"
userText.substr(13, 4)                   // Returns ".com"
userText.substr(userText.size() - 4, 4)  // Last 4: ".com"
```

Getting a substring

substr()	***substr***(index, length) returns substring starting at *index* and having *length* characters.

```
// userText is "Help me!"
userText.find('p') // Returns 3
userText.find('e') // Returns 1 (first occurrence of e only)
userText.find('z') // Returns string::npos
userText.find("me") // Returns 5
userText.find('e', 2) // Returns 6 (starts at index 2)
```

Numeric Data Types

Positional Number Systems

- Decimal (base 10) is an example
    - e.g.,
        - 435 means
            - 400 + 30 + 5
            - 4 x 10^2 + 3 x 10^1 + 5 x 10^0
- Example of a non-positional system: Roman numerals
    - inconvenient for humans
    - unusable for computers
- This concept applies to other bases as well

Binary & Decimal Numbers

- Base 2 -- natural for computers
  - 0 represents OFF, 1 represents ON
- Base 10 -- natural for humans
  - decimal system uses 10 symbols, 0 - 9
- binary system uses 2 symbols, 0 & 1
  - **0**, **1**, **10**, **11**, **100**, **101**, **110**, **111**, **1000**, **1001**, etc.
  - 1101_2 may be written as
    - $1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 8 + 4 + 0 + 1 = 13_{10}$

Everything is Bits

- In memory everything is a 1 or a 0.
- **Datatype** indicates how those bits are interpreted.

01000011 00101101 00110111 00000000

- 32 bit binary
- Integer (2’s Complement): 1127036672
- Unsigned Integer: 1127036672
- Float: 173.21484375
- Characters (ASCII): C-7

11000001 00111100 01100101 00000000

- 32 bit binary
- Integer (2’s Complement): -1053006592
- Unsigned Integer: 3241960704
- Float: -11.774658203125
- Characters (ASCII): Á<e

- For signed numbers, the leftmost bit indicates the sign
  - 1: negative
  - 0: positive

Integers

- Raw numbers
- Different size variables
    - C++ datatypes
        - char
        - short
        - int
        - long
    - Number of bytes depends on system and compiler
- Signed numbers frequently (but not always) represented with 2’s complement. See optional slides on Integers.

Binary Numbers

- Base Ten Numbers (Integers)
  - digits: 0 1 2 3 4 5 6 7 8 9
  - 5401 is $5x10^3 + 4x10^2 + 0x10^1 + 1x10^0$
- Binary numbers are the same
  - digits: 0 1
  - 1011 is $1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$

Converting Binary to Base 10

- $2^3 = 8$
- $2^2 = 4$
- $2^1 = 2$
- $2^0 = 1$

1. $1001_2 = \_\_\_\_\__{10} =$
   - $1x2^3 + 0x2^2 + 0x2^1 + 1x2^0 =$
   - $1x8 + 0x4 + 0x2 + 1x1 =$
   - $8 + 0 + 0 + 1 = 9_{10}$
2. $0110_2 = \_\_\_\_\__{10}$ (Try yourself)
   - $0110_2 = 6_{10}$

Converting Base 10 to Binary

- $388_{10} = \underline{\qquad}_2$
- $388_{10} / 2 = 194_{10}$ Remainder **0**
- $194_{10} / 2 = 97_{10}$ Remainder **0**
- $97_{10} / 2 = 48_{10}$ Remainder **1**
- $48_{10} / 2 = 24_{10}$ Remainder **0**
- $24_{10} / 2 = 12_{10}$ Remainder **0**
- $12_{10} / 2 = 6_{10}$ Remainder **0**
- $6_{10} / 2 = 3_{10}$ Remainder **0**
- $3_{10} / 2 = 1_{10}$ Remainder **1**
- $1_{10} / 2 = 0_{10}$ Remainder **1**

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	0	0	0	1	0	0

Other common number representations

- Octal Numbers
  - digits: 0 1 2 3 4 5 6 7
  - 7820 is $7 \mathrm{x} 8^3 + 8 \mathrm{x} 8^2 + 2 \mathrm{x} 8^1 + 0 \mathrm{x} 8^0$
  - 4112 (base 10)
- Hexadecimal Numbers
  - digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F
  - 2FD6 is $2 \mathrm{x} 16^3 + \mathrm{Fx} 16^2 + \mathrm{Dx} 16^1 + 6 \mathrm{x} 16^0$
  - 12,246 (base 10)

http://www.kaagaard.dk/service/convert.htm

Negative Numbers

- Can we store a negative sign?
- What can we do?
  - Use a bit
  - Negative: leftmost bit set to 1
  - Positive: leftmost bit set to 0
- Most common is two’s complement

“leftmost” – aka “most significant”

Representing Negative Numbers

- Two’s Complement
  - To convert binary number to its negative…
  - flip all the bits
    - change 0 to 1 and 1 to 0
  - add 1
  - if the leftmost bit is 0, the number is 0 or positive
  - if the leftmost bit is 1, the number is negative

Two’s Complement

- What is -9?
  - 9 is 00001001 in 8-bit binary
  - flip the bits → 11110110
  - add 1 → 11110111
- Addition and Subtraction are easy
  - always addition
    - If it is subtraction, first take negative of number being subtracted
    - Then add!

Two’s Complement

- Subtraction
  - 4 - 9 = -5
  - 4 + (-9) = -5 (becomes addition)
  - 00000100 + 11110111 = ?

$$
\begin{array}{cccccccccc}
0 & 0 & 0 & 0 & 1 & 0 & 0 & & & \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & & \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & & \\
\hline
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & = & ?
\end{array}
$$

Negative since it starts with a 1

- Flip bits
  - 11111011 → 00000100
  - Add one → 00000101
  - Which is 5
  - Since its positive two’s complement is 5, the number is -5!

Two’s Complement

- Subtraction
    - 13 - 9 = 4
    - 13 + (-9) = 4 (becomes addition)
    - 00001101 + 11110111 = ?

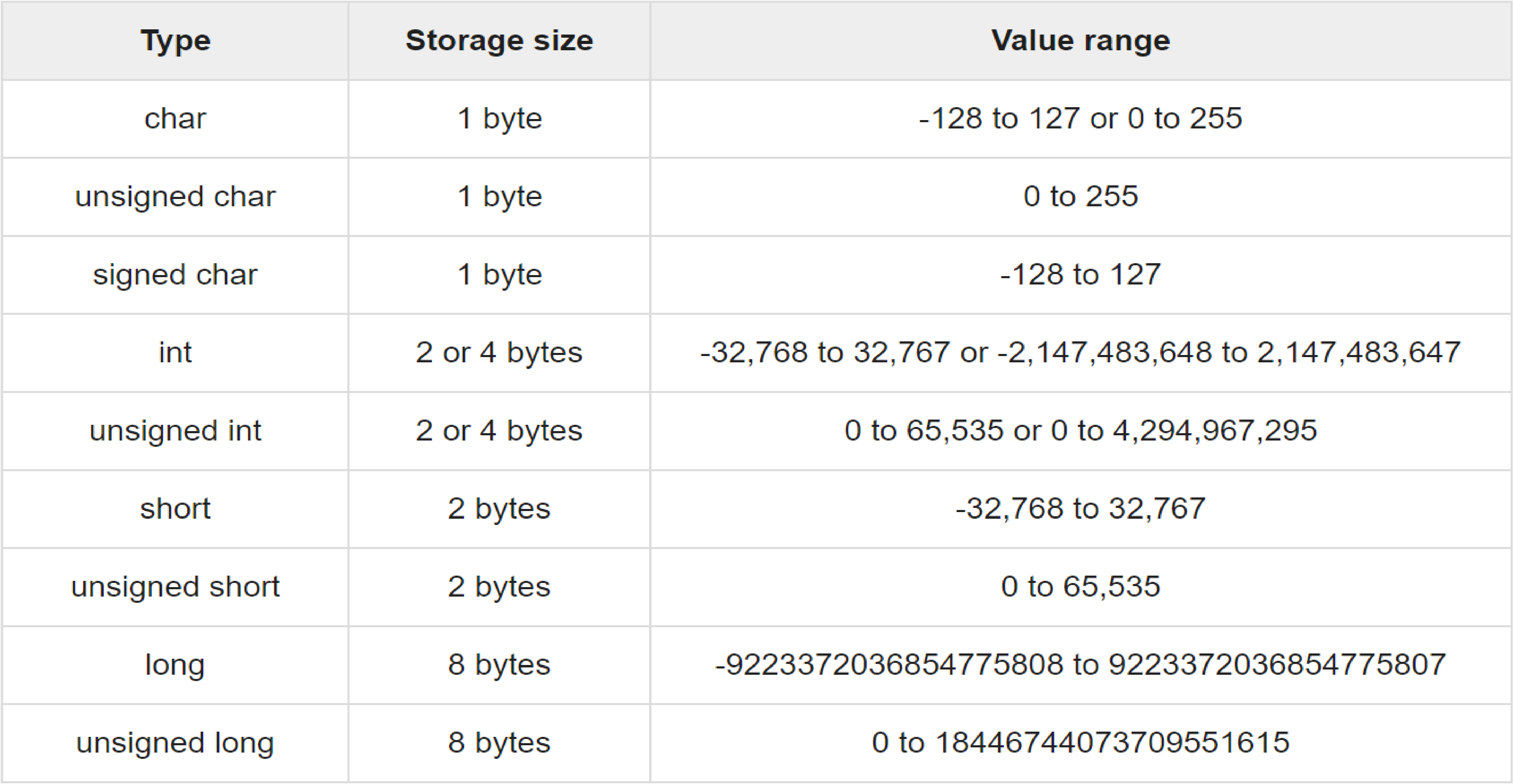
This bit is “lost”

$$\begin{array}{cccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & \\ & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & \\ & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & \\ \hline & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & = \;\; 4 \end{array}$$

But that doesn’t matter since we get the correct answer anyway

Two’s Complement Range

- If we did not use two’s complement and used the left most bit to indicate negative:
  - Negative (1)1111111 to positive (0)1111111
  - -127 to 127
- With two’s complement
  - -128 to 127
  - Gain an extra digit in the range.
    - Where did it come from?



Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

https://www.tutorialspoint.com/cprogramming/c_data_types.htm

Booleans

- Logically
  - True
  - False
- C++
  - Represented by an integer in the background
    - false is 0
    - true is literal 1 by default
      - Any non-zero value is *truthy*
    - sizeof(bool) is implementation-defined, not required to be 1.
      - but is probably 1 (byte).

Characters

- Char
  - Actually a numeric datatype
    - Output converts number to corresponding output character (lookup)
  - Only 256 options
  - http://www.rapidtables.com/code/text/ascii-table.htm

- Unicode (we won’t use but good to know it exists)
  - http://www.utf8-chartable.de/unicode-utf8-table.pl?number=1024&utf8=bin

- C++ 11 supports
  - wchar_t   (wide character)
  - char16_t
  - char32_t

Floating Point Numbers

- Decimal Numbers
  - 3 parts
    - Sign
    - Exponent
    - Mantissa