

Numerical Conversions Strings in C++

CS 16: Solving Problems with Computers I
Lecture #8

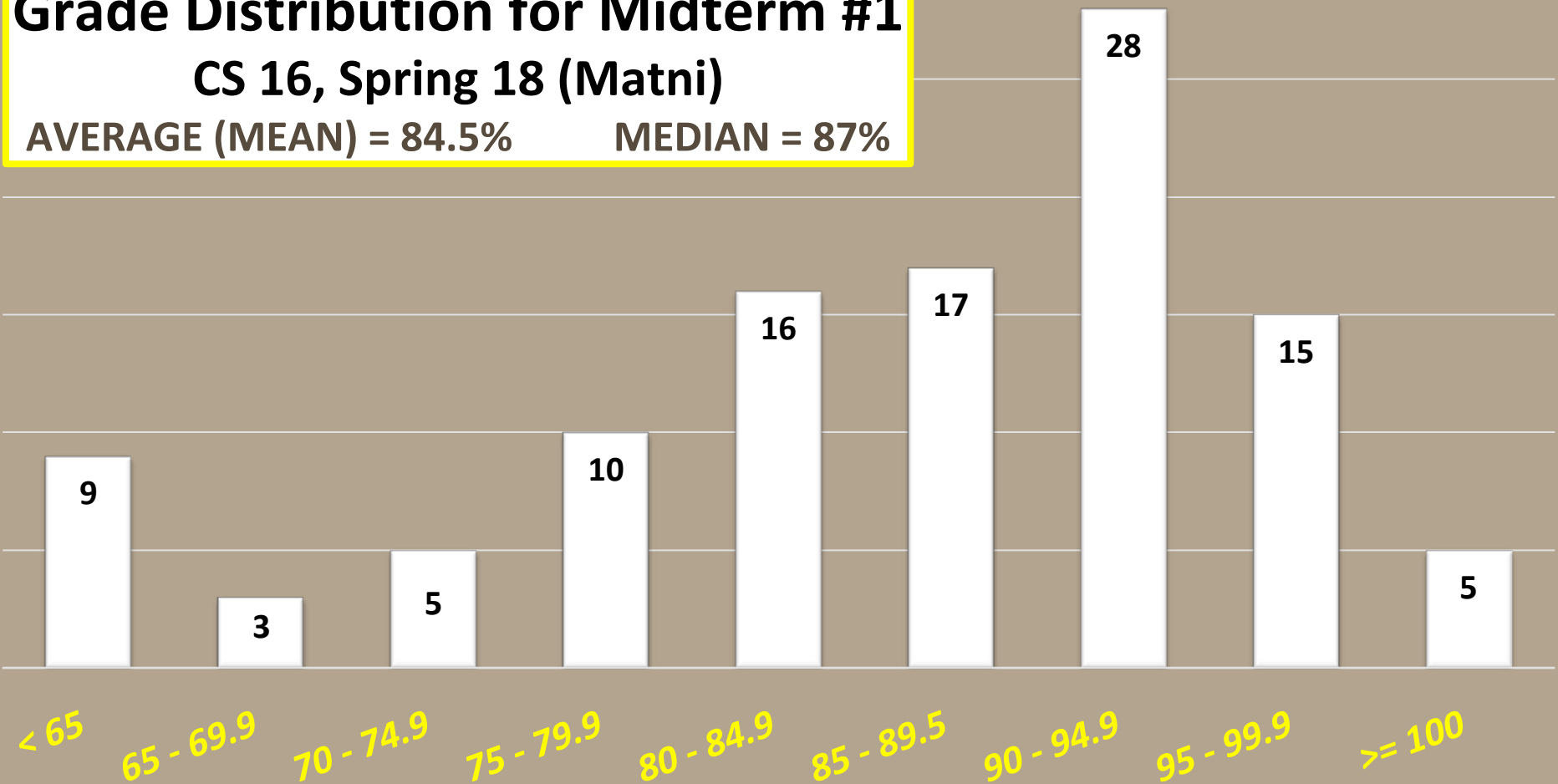
Ziad Matni
Dept. of Computer Science, UCSB

Grade Distribution for Midterm #1

CS 16, Spring 18 (Matni)

AVERAGE (MEAN) = 84.5%

MEDIAN = 87%



Lecture Outline

- Numerical Conversions
 - Binary, Octal, Hexadecimal
- Strings
 - **C Strings** vs. **C++ Strings**

Counting Numbers in Different Bases

- We “normally” count in 10s
 - Base 10: **decimal** numbers
 - Number symbols are 0 thru 9
- Computers count in 2s
 - Base 2: **binary** numbers
 - Number symbols are 0 and 1
 - Represented with **1 bit** ($2^1 = 2$)
- Other convenient bases in computer architecture:
 - Base 8: **octal** numbers
 - Number symbols are 0 thru 7
 - Represented with **3 bits** ($2^3 = 8$)
 - Base 16: **hexadecimal** numbers
 - Number symbols are 0 thru F
 - A = 10, B = 11, C = 12, D = 13, E = 14, F = 15
 - Represented with **4 bits** ($2^4 = 16$)
 - **Why are 4 bit representations convenient???**

Positional Notation in Decimal

a.k.a.: How I Learned Numbers in 3rd Grade...

642 is: 6 hundreds, 4 tens, and 2 units

It's a number in base 10 (aka decimal)

We can write it in positional notation:

$$\begin{array}{rclcl} 6 \times 10^2 & = & 6 \times 100 & = & 600 \\ + 4 \times 10^1 & = & 4 \times 10 & = & + 40 \\ + 2 \times 10^0 & = & 2 \times 1 & = & + 2 \end{array} \quad = 642 \text{ in base 10}$$

Positional Notation

Anything → DEC

What if “642” is expressed in the base of 13?

$$\begin{aligned} 6 \times 13^2 &= 6 \times 169 = 1014 \\ + 4 \times 13^1 &= 4 \times 13 = 52 \\ + 2 \times 13^0 &= 2 \times 1 = 2 \\ &= 1068 \text{ in base 10} \end{aligned}$$

So, “642” in base 13 is equivalent to
“1068” in base 10

BUT WHO COUNTS IN BASE 13???!?!?



Maybe, aliens with
13 fingers???

COMPUTERS ARE DIGITAL (Binary) MACHINES

THEY ARE DESIGNED
TO COUNT IN...

2

Positional Notation in Binary

11011 in base 2 *positional notation* is:

$$\begin{aligned}1 \times 2^4 &= 1 \times 16 = 16 \\+ 1 \times 2^3 &= 1 \times 8 = 8 \\+ 0 \times 2^2 &= 0 \times 4 = 0 \\+ 1 \times 2^1 &= 1 \times 2 = 2 \\+ 1 \times 2^0 &= 1 \times 1 = 1\end{aligned}$$

So, **1011** in base 2 is $16 + 8 + 0 + 2 + 1 = 27$ in base 10

Converting Binary to Decimal

*Q: What is the decimal equivalent of the binary number **1101100**?*

A: Look for the position of the digits in the number.

This one has 7 digits, therefore positions 0 thru 6

1	1	0	1	1	0	0
64	32	16	8	4	2	1
2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$\begin{aligned} & \mathbf{1} \times 2^6 = \mathbf{1} \times 64 = 64 \\ & + \mathbf{1} \times 2^5 = \mathbf{1} \times 32 = 32 \\ & + \mathbf{0} \times 2^4 = \mathbf{0} \times 16 = 0 \\ & + \mathbf{1} \times 2^3 = \mathbf{1} \times 8 = 8 \\ & + \mathbf{1} \times 2^2 = \mathbf{1} \times 4 = 4 \\ & + \mathbf{0} \times 2^1 = \mathbf{0} \times 2 = 0 \\ & + \mathbf{0} \times 2^0 = \mathbf{0} \times 1 = 0 \\ & = \mathbf{108} \text{ in base 10} \end{aligned}$$

Other Relevant Bases

- In Computer Science/Engineering, other binary-related numerical bases are used too.
- OCTAL: Base 8 (note that 8 is 2^3)
 - Uses the symbols: 0, 1, 2, 3, 4, 5, 6, 7
- HEXADECIMAL: Base 16 (note that 16 is 2^4)
 - Uses the symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Converting Binary to Octal and Hexadecimal

(or any base that's a power of 2)

- Binary is 1 bit
- Octal is 3 bits ($2^3 = 8$) *octal is base 8*
- Hexadecimal is 4 bits ($2^4 = 16$) *hex is base 16*
- Use the “**group the bits**” technique
 - Always start from the *least significant digit*
 - Group every 3 bits together for bin \rightarrow oct
 - Group every 4 bits together for bin \rightarrow hex

Converting Binary to Octal and Hexadecimal

- Take the example: **10100110**

...to octal:

10	100	110
----	-----	-----

2 4 6

246 in octal

...to hexadecimal:

1010	0110
------	------

10 6

A6 in hexadecimal

Decimal
symbols

0
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F

Octal
symbols

Hex.
symbols

Converting Decimal to Other Bases

Algorithm for converting number in base 10 to other bases

While (the **quotient** is not zero)

1. Divide the decimal number by the **new base**
2. Make the **remainder** the next digit to the **left** in the answer
3. Replace the original decimal number with the **quotient**
4. Repeat until your quotient is zero

EXAMPLE:

Convert the decimal (base 10) number **79** into hexadecimal (base 16)

$$79 / 16 = 4 \text{ R } 15 \quad (15 \text{ in hex is the symbol "F"})$$

$$4 / 16 = 0 \text{ R } 4$$

The answer is: **4F**

Converting Decimal into Binary

Convert 54 (base 10) into binary and hex:

- $54 / 2 = 27 \text{ R } 0$
- $27 / 2 = 13 \text{ R } 1$
- $13 / 2 = 6 \text{ R } 1$
- $6 / 2 = 3 \text{ R } 0$
- $3 / 2 = 1 \text{ R } 1$
- $1 / 2 = 0 \text{ R } 1$

Sanity check:

110110

$= 2 + 4 + 16 + 32$

$= 54$

54 (decimal) = 110110 (binary)
= 36 (hex)

What is a String?

- Characters connected together in a sequence

P	i	k	a	c	h	u
H	i		M	o	m	!

Strings in C/C++

- Recall: C++ is based on C
- Originally (in C), strings were defined as an “array of characters”
 - Called C-Strings and are “legacy” data types in C++
 - Came with the library **<cstring>**
 - Contains lots of built-in functions that go with C-Strings
- In C++, we got a new library: **<string>**
- Made improvements over the old “C-String”
 - Library contains another collection of functions that work with Strings, but not C-Strings!

Why Do We Care About C-Strings??

- Their use STILL comes up in C++
 - Recall: command line arguments...
- Recall that command-line arguments, specifically `argv[x]` are defined as: **`char* []`**
- That's a classic definition of a C-String
 - So if we want to use these `argv[x]`, we'll have to treat them in a C-String fashion...

C strings vs. C++ strings

- Strings in C++ and Strings in C
 - C++ is meant to be backwards compatible with C
 - C has one way of dealing with strings, **while C++ has another**
- C++'s use is much easier and safer with memory allocation
 - This is what you've learned so far with `<string>`
 - Let's briefly review the other (older) way with C-strings...

What's a C++ Programmer to Do?!

- A C-string
 - An array of characters terminated by the **null character** `'\0'`
 - The null character has an ASCII code of **0**.
 - Library for dealing with these types: `<cstring>`
- A C++ string *object*
 - An instance of a “class” data type – used a “black box”
 - Library for dealing with these types: `<string>`

The C-String

- An array of characters that terminates in the null character
 - This terminates the actual string, but not the array necessarily
- Example : a C-string stores “Hi Mom!” in a character array of size 10
 - The characters of the word “Hi Mom!” will be in positions with indices 0 to 6
 - There will be a null character at index 7, and the locations with indices 8 to 9 will contain some unknown value.
 - But we don’t care about positions 8 and 9!
 - The null character says “STOP HERE!”

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
H	i		M	o	m	!	\0	??	??

C-strings

- To declare a C-string variable, include `<cstring>` and use this syntax:

```
char Array_name[ Maximum_C_String_Size + 1 ];
```

– The “+ 1” reserves the additional character needed by `'\0'`

- With C-Strings, you **cannot** do these:

```
myString = "Hello!"           //assignment
```

```
if (myString == "Jimbo")...   //comparison
```

– Instead use `strncpy()` and `strcmp()` from the `<cstring>` library

- But you CAN use `=` and `==` with C++ Strings
 - And so much **more** useful things! 😊

The Standard C++ **string** Class

- The strings we know and love...
- The string class allows the programmer to treat strings as a basic data type
 - No need to deal with the implementations of C-strings
- The string class is defined in the **<string>** library
- We will discuss many different ***member functions*** that are extremely useful to use
 - Like **.length()**, **.erase()**, **.substr()**, **.find()**, *etc...*

Declaring a String in C++

- You have to include the correct library module with:

```
#include <string>
```

- Declare them (and initialize them) with:

```
string MyString=""; // Note the use of double-quotes!
```

- Since strings are made up of characters, you can index individual characters in strings (starting at position 0):

If `MyString = "Hello!"`

Then `MyString[0] = 'H', MyString[1] = 'e', etc...`

String Basics

- Use the **+** operator to *concatenate* 2 strings

```
string str1 = "Hello ", str2 = "world!", str3;  
str3 = str1 + str2;    // str3 will be "Hello world!"
```
- Use the **+=** operator to *append* to a string

```
str1 += "Z";    // str1 will be "Hello Z"
```
- Call out a character in the string based on **position**, using [] braces
 - Recall array indices in C++ start at zero (0)

```
cout << str1[0];    // prints out 'H'  
cout << str2[3];    // prints out 'l'
```


Built-In String Member Functions

- Search functions
 - **find, rfind, find_first_of, find_first_not_of**
- Descriptor functions
 - **length, size**
- Content changers
 - **substr, replace, append, insert, erase**

Search Functions: **find** 1

- You can search for a the ***first occurrence*** of a string in a string with the **.find** function

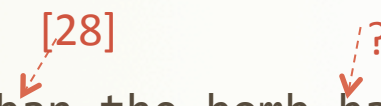
```
string str = "With a banjo on my knee and ban the bomb-ban!";  
int position = str.find("ban");  
cout << position;    // Will display the number 7
```



Search Functions: **find** 2

- You can also search for the ***first occurrence*** of a string in a string, starting at position ***n***, using a slight mod to **.find()**

```
string str = "With a banjo on my knee and ban the bomb-ban!";  
int position = str.find("ban", 12);  
cout << position;    // Will display the number 28
```



The diagram illustrates the search process for the string "ban" starting at index 12. A red dashed arrow points from the index [28] to the first occurrence of "ban" in the string "ban the bomb-ban!". Another red dashed arrow points from a question mark to the second occurrence of "ban" in "bomb-ban!", indicating that the search stops at the first match.

Search Functions: **find** 3


- You can use the **find** function to make sure a substring is **NOT** in the target string using the “no position” value
string::npos is returned if no position exists

```
if (MyStr.find("piano") == string::npos)
    cout << "There is no piano there!"
// This will happen if “piano” is NOT in the string MyStr
```

Search Functions: **rfind**

- You can search for the *last occurrence* of a string in a string with the **.rfind** function

```
string str = "With a banjo on my knee and ban the bomb-ban!";  
int rposition = str.rfind("ban");  
cout << rposition;    // Will display the number 41
```



Search Functions:

find_first_of and **find_first_not_of**

- **find_first_of**
 - Finds 1st occurrence of **any** of the characters included in the specified string
- **find_first_not_of**
 - Finds 1st occurrence of a character that is **not any** of the characters included in the specified string
- Example:

See demo file:
non_numbers.cpp

Descriptor Functions: **length** and **size**

- The **length** function returns the length of the string
- The member function **size** is the same exact thing...
 - So, if `string str1 = "Mama Mia!"`,
then `str1.length() = 9`
and `str1.size() = 9` also

Example – what will this code do?:

```
string name = "Bubba Smith";  
for (int i = name.length(); i > 0; i--)  
    cout << name[i-1];
```

Content Changers: **append**

- Use function **append** to append one string to another

```
string name1 = " Max";  
string name2 = " Powers";  
cout << name1.append(name2); // Displays " Max Powers"
```
- Does the same thing as: **name1 + name2**

Content Changers: **erase**

- Use function **erase** to clear a string to an empty string
- One use is:
name1.erase() -- Does the same thing as: **name1 = ""**
- Another use is:
name1.erase(*start position, how many chars to erase*)
 - Erases only part of the string
 - Example:

```
string s = "Hello!";  
cout << s.erase(2, 2);
```

// Displays "Heo!"

Content Changers: **replace** and **insert**

- Use function **replace** to replace part of a string with another
 - Popular Usage:
`string.replace(start position, # of places after start position to replace, replacement string)`
- Use function **insert** to insert a substring into a string
 - Popular Usage:
`string.insert(start position, insertion string)`

Example:

```
string country = "Back in the USSR";           // length is 16
cout << country.replace(14, 2, "A");           // Displays "Back in the USA"
cout << country.insert(15, "BC");              // Displays "Back in the USABC"
```

Content Changers: **substr**

- Use function **substr** (short for “substring”) to extract and return a substring of the **string** object

– Popular Usage:

`string.substr(start position, # of places after start position)`

Example:

```
string city = “Santa Barbara”;  
cout << city.substr(3, 5)    // Displays “ta Ba”
```

YOUR TO-DOS

- ☐ Prepare Lab4 for Wednesday!
- ☐ Do HW8 by next Thursday
- ☐ Visit Prof's and TAs' office hours if you need help!
- ☐ Run a mile. Or two.

</LECTURE>