Topic 5

1. Variables
2. Arithmetic
3. Input and output
4. Problem solving: first do it by hand
5. Strings
6. Chapter summary

Strings

- Strings are sequences of characters:

```
"Hello world"
```

- Include the string header, so you can create variables to hold strings:

```
#include <iostream>
#include <string>
using namespace std;
...
string name = "Harry";
         // literal string "Harry" stored
```

String Initializations

- String variables are automatically initialized to the empty string if you don't initialize them:

```
string response;
      // literal string "" stored
```

- `""` is called the empty or null string.

Concatenation of Strings

Use the + operator to *concatenate* strings;
that is, put them together to yield a longer string.

```
string fname = "Harry";
string lname = "Morgan";
string name = fname + lname; //need a space!
cout << name << endl;
name = fname + " " + lname; //got a space
cout << name << endl;
```

The output will be

```
HarryMorgan
Harry Morgan
```

Common Error – Concatenation of literal strings

```
string greeting = "Hello, " + " World!";
                  // will not compile
```

Literal strings cannot be concatenated. And it's pointless anyway, just do:

```
string greeting = "Hello World!";
```

String Input

You can read a string from the console:

```
cout << "Please enter your name: ";
string name;
cin >> name;
```

When a string is read with the **>>** operator, only one word is placed into the **`string`** variable.

For example, suppose the user types

```
Harry Morgan
```

as the response to the prompt.

Only the string "Harry" is placed into the variable name.

String Input

You can use another input string to read the second word:

```
cout << "Please enter your name: ";
string fname, lname;
cin >> fname >> lname;
//fname gets Harry, lname gets Morgan
```

String Functions

- The **`length`** *member function* yields the number of characters in a string.
- Unlike the **`sqrt`** or **`pow`** function, the **`length`** function is *invoked* with the *dot notation*:

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

`substr` Function

- Once you have a string, you can extract substrings by using the **`substr`** member function.
- **`s.substr(start, length)`**
  returns a **`string`** that is made from the characters in the **`string s`**, starting at character **`start`**, and containing **`length`** characters. (**`start`** and **`length`** are integers)
  - NOTE: the first character has an index of 0, not 1.

```
string greeting = "Hello, World!";
string sub = greeting.substr(0, 2);
   // sub contains "He"
```

Another Example of the `substr` Function

```
string greeting = "Hello, World!";
string w = greeting.substr(7, 5);
   // w contains "World" (not the !)
```

- `"World"` is 5 characters long but…
- Why is 7 the position of the `"W"` in `"World"`?
- Why is the `"W"` not @ 8?
- *Because the first character has an index of 0, not 1.*

String Data Representation & Character Positions

```
H  e  l  l  o  ,     W  o  r  l  d  !
0  1  2  3  4  5  6  7  8  9  10 11 12
```

- In most computer languages, the starting position 0 means "start at the beginning."

- The first position in a `string` is labeled 0, the second 1, and so on. And don't forget to count the space character after the comma—but the quotation marks are ***not*** stored.

The position number of the last character is always one less than the length of the ***`string`***.

String Character Positions

```
 H  e  l  l  o  ,     W  o  r  l  d  !
 0  1  2  3  4  5  6  7  8  9 10 11 12
```

```
string greeting = "Hello, World!";
string w = greeting.substr(7);
   // w contains "World!"
```

If you do not specify how many characters to the substr() function, you get all the rest.

String Operations Examples: Table 8

Statement	Result	Comment
`string str = "C";` `str = str + "++";`	str is set to "C++"	When applied to strings,+ denotes concatenation.
`string str = "C" + "++";`	**Error**	**Error:** You cannot concatenate two string literals.

`cout << "Enter name: ";`
`cin >> name;`
(*User input:* `Harry Morgan`)
	name contains "Harry"	The >> operator places the next word into the string variable.

`cout << "Enter name: ";`
`cin >> name >> last_name;`
(*User input:* `Harry Morgan`)
	name contains "Harry", last_name contains "Morgan"	Use multiple >> operators to read more than one word.

`string greeting = "H & S";` `int n = greeting.length();`	n is set to 5	Each space counts as one character.
`string str = "Sally";` `string str2 = str.substr(1, 3);`	str2 is set to "all"	Extracts the substring of length 3 starting at position 1. (The initial position is 0.)
`string str = "Sally";` `string str2 = str.substr(1);`	str2 is set to "ally"	If you omit the length, all characters from the position until the end are included.
`string a = str.substr(0, 1);`	a is set to the initial letter in str	Extracts the substring of length 1 starting at position 0.
`string b = str.substr(str.length() - 1);`	b is set to the last letter in str	The last letter has position str.length() - 1. We need not specify the length.

String Functions, Complete Program Example

ch02/initials.cpp

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
   cout << "Enter your first name: ";
   string first;
   cin >> first;
   cout << "Enter your significant other's first name: ";
   string second;
   cin >> second;
   string initials = first.substr(0, 1)
      + "&" + second.substr(0, 1);
   cout << initials << endl;

   return 0;
}
```

Representing Characters: Unicode

- Printable characters in a `string` are stored as bits in a computer, just like `int` and `double` variables
- The bit patterns are standardized:
  - ASCII (American Standard Code for Information Interchange) is 7 bits long, specifying $2^7 = 128$ codes:
    - 26 uppercase letters A through Z + 26 lowercase letters a through z
    - 10 digits
    - 32 typographical symbols such as +, -, ', \...
    - 34 control characters such as space, newline, and 32 others for controlling printers and other devices.
  - Unicode, which has replaced ASCII in most cases, is 21 bits
    - superset of ASCII; the first 128 codes match
    - The extra bits allow many more characters ($2^{21} > 2 \times 10^6$), required for worldwide languages
    - About 136,000 characters have been assigned so far
    - UTF-8 is the 8-bit subset of Unicode, and UTF-16 is 16-bit, often used by websites and compilers

Topic 6

1. Variables
2. Arithmetic
3. Input and output
4. Problem solving: first do it by hand
5. Strings
6. Chapter summary

Chapter Summary Part 1

Write variable definitions in C++.

- A variable is a storage location with a name.
- When defining a variable, you MUST specify the type of its values.

  And you should also specify an initial value:

```
int x=0;
```

- Use the **`int`** type for numbers that cannot have a fractional part.
- Use the **`double`** type for floating-point numbers.

Chapter Summary Part 2

- An assignment statement stores a new value in a variable, replacing the previously stored value.
- The assignment operator = does *not* denote mathematical equality.
- You cannot change the value of a variable that is defined as `const`.
- Use comments to add explanations for humans who read your code. The compiler ignores comments.

Chapter Summary Part 3

Use the arithmetic operations in C ++

- Use * for multiplication and / for division.
- The `++` operator adds 1 to a variable; the `--` operator subtracts 1.
- If both arguments of / are integers, the quotient is an int, and the remainder is discarded.
- The `%` operator computes the remainder of an integer division.
- Assigning a floating-point variable to an integer drops the fractional part.
- The C++ <cmath> library defines many math functions such as `sqrt` (square root) and `pow` (raising to a power).

Chapter Summary Part 4

Write programs that read user input and write formatted output.

- Use the **>>** operator to read a value and place it in a variable.

```
int x=0;
cout << "Enter value for x: ";
cin >> x;
```

- You use manipulators to specify how OUTPUT values should be formatted.

```
const double PI=3.14159265;
cout << "Pi =" << setprecision(8) << setw(10) << PI<<
endl;
```

Carry out hand calculations when developing an algorithm, *before* typing your C++ code.

- Pick concrete values for a typical situation to use in a hand calculation, to very algorithm correctness.

Chapter Summary Part 5: Strings

Write programs that process strings.

- Strings are sequences of characters
- Use the **+** operator to concatenate strings; that is, put them together to yield a longer string.

```
string s1= "hello", s2= " world";
string s3 = s1 + s2; //s3 gets "hello world"
```

- The **`length`** member function yields the number of characters in a string. A member function is invoked using the dot notation:

```
int len =s1.length();
```

Use the **`substr`** member function to extract a substring:

```
string s3 = s1.substr(0,2); //s3 gets "he"
```