CS 132, Winter 2022

Lecture 1: Introduction; Basic C++ Programs; `strings`

Thank you to Marty Stepp and Stuart Reges for parts of these slides

Important Information

- Course website:
  - http://allisonobourn.com/edmonds/132
- Instructor email:
  - allison.obourn@edmonds.edu

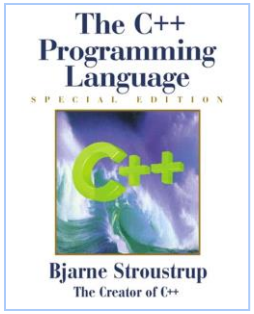
Logistics

- General course setup – same as CS& 131:
  - Outside of class graded work:
    - small problems
    - projects
    - quizzes
    - midterm and final exams
  - In class graded work:
    - labs
  - Help available:
    - Discord - the official message board
    - Office hours
    - Study centers on campus
  - Differences this quarter:
    - **Group** final project

Why switch to C++?

- What was extra difficult or annoying in C?
  - Pointers
  - Strings
  - Arrays not knowing their own length
  - Having to specify a type when you print something
  - Very few built in functions and structures
  - No way to attach functions to structs

What is C++ ?



- **C++**: A programming language developed in 1983 by Bjarne Stroustrup.
  - built on the C language and adds things C is missing
  - still allows for a lot of control to optimize high speed/efficiency
  - one of the world's most widely used languages today
  - continues to be improved over time (latest version: C++11)

- C++ syntax has many similarities with Java and C

  - similar data types (`int`, `double`, `char`, `void`)
  - similar operators (`+`, `-`, `*`, `/`, `%`), keywords
  - use of `{ }` braces for scope
  - comes equipped with a large standard library for you to use

Your First C++ Program

- Everything that works in C works in C++
  - Put the below code (the first code you probably wrote in C) into a file with a `.cpp` extension and run it.

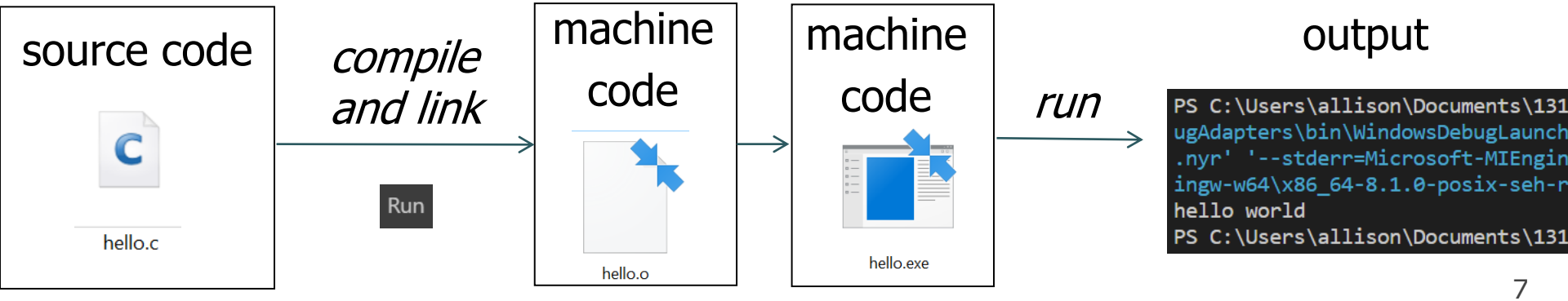
```
#include <stdio.h>
int main() {
   printf("Hello, world!\n\n");
   printf("This program produces\n");
   printf("four lines of output\n");
}
```

  - You have now technically written a C++ program!

Compiling/running a program

1. *Write* it.
2. *Compile* and link it.
3. *Run* (execute) it.

- The only difference is we should save our code in a `.cpp` file instead of a `.c` file.



The same but different

- While everything that works in C works in C++, generally it is better to write C++ code the C++ way.
  - To get full credit in this class you will need to use the new C++ syntax we will be learning
  - We will go over the C++ version of some things we learned in CS& 131.
    - If we don't cover something, assume it works the same as in C.
- Note: not all C++ code will work in C

Things that stay the same

```
int x = 42 + 7 * -5;                    // variables, types
double pi = 3.14159;
char c = 'Q';                           /* two comment styles */
bool b = true;

for (int i = 0; i < 10; i++) {          // for loops
    if (i % 2 == 0) {                   // if statements
        x += i;
    }
}

while (x > 0 && c == 'Q' || b) {        // while loops, logic
    x = x / 2;
    if (x == 42) { return 0; }
}

foo(x, 17, c);                          // function call
bar("this is a string");                // string literals
```

Almost the same: math functions

- To use: `#include <cmath>`

Function name	Description (returns)
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`floor(`***value***`)`	rounds down
`log10(`***value***`)`	logarithm, base 10
`max(`***value1, value2***`)`	larger of two values
`min(`***value1, value2***`)`	smaller of two values
`pow(`***base, exp***`)`	*base* to the *exp* power
`round(`***value***`)`	nearest whole number
`sqrt(`***value***`)`	square root

`sin(`***value***`)`
`cos(`***value***`)`
`tan(`***value***`)`
	sine/cosine/tangent of an angle in radians

Hello World the C++ way

```
/*
 * hello.cpp
 * This program prints a welcome message
 * to the user.
 */
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, world!" << endl << endl;
    cout << "This program produces" << endl;
    cout << "four lines of output" << endl;
}
```

Include/Using

- We will often use different libraries in C++ than C
  - For example:

```
#include <stdio.h>     // the C I/O library
#include <iostream>    // the C++ I/O library
```

- `using namespace name;`
  - Brings symbols from a library's "name space" into the global scope of your program so you can refer to them more easily.
  - Many C++ standard library features are in namespace `std`, so for this class, always use `std` unless told otherwise.

Namespaces

- Namespaces group together their contents and put a wrapper around them so that their names don't conflict with names in other code.

- By default, your code is in an anonymous namespace

  - To access things in another namespace write: **namespace**::**functionName**

```
std::cout << "only necessary if we are in a different namespace";
```

- There is disagreement over whether you should write code `using namespace std;`
  - Bad because it can cause name conflicts
  - Good because it makes your code easier to read, simpler looking.
  - You can also just use individual things: `using std::cout;`

Console output: `cout`

- `cout << expression << expression ...`

- `endl`
  - A variable that means "end of line"
  - Same as `"\n"`, but more compatible with all operating systems

```
// good
cout << "You are " << age << " years old!" << endl;

// bad
cout << "You are " << age << " years old!\n";
```

To use: `#include <iostream>`

Console input: `cin`

- Reads input from console and stores it in the given variable.

```
cin >> variable
```

- Example:

```
int age;
cout << "Please type your age: ";    // prompt
cin >> age;
cout << "Wow, you are " << age << " years old!";
```

- Reads a word at a time; hard to get an entire line

To use: `#include <iostream>`

Console input: `getline`

- Reads a **whole line of input** from console and stores it in the given variable.

```
getline(cin, variable);
```

- Example:

```
string name;
cout << "Please type your full name: ";    // prompt
getline(cin, name);
cout << "Welcome, " << name << "!";
```

To use: `#include <iostream>`

Strings

```
#include <string>
...
string s = "hello";
```

- C++ has a string type!
  - Many useful built-in string member functions and operators
  - Beware: strings are *mutable* (can be changed) in C++.
    - This is not the case in many other languages you may have used that have a string type.
  - Beware: There are two types of strings in C++. :-/
    - `char*` / `char[]` strings still exist

Exercise

- Write a program that prompts the user for the year they were born and name and prints out their age.

Example output:

```
What year were you born? 2013
What is your name? Merlin the Cat
Merlin the Cat is 8 years old!
```

Mixing input reading

- The following program prompts for a word, reads it and then prints output. Why does it skip prompting for a line?

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

int main() {
     string word;
     string line;
     cout << "enter a word" << endl;
     cin >> word;
     cout << "enter a line" << endl;
     getline(cin,line);
     cout << "your word is " << word << endl;
     cout << "your line is " << line << endl;
     return 0;
}
```

Mixing input reading

- When you read using `cin >>`, the `"\n"` you typed is still in the buffer.
  - Don't do this! Just read single tokens or lines, not both.
  - If you must use both: call `cin.ignore()` between reading with `cin >>` and `getline`. This will flush the buffer.

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

int main() {
	string word;
	string word;
	cout << "enter a word" << endl;
	cin >> word;
	cin.ignore();
	cout << "enter a line" << endl;
	getline(cin,line);
	cout << "your word is " << word << endl;
	cout << "your line is " << line << endl;
	return 0;
}
```

String and char

```
#include <string>
string s = "Hi 131!";
```

index	0	1	2	3	4	5	6
character	'H'	'i'	' '	'1'	'3'	'1'	'!'

- Characters are values of type `char`, with 0-based indexes.
- Individual characters can be accessed using `[index]` or `at`:

```
char c1 = s[5];                 // '1'
char c2 = s.at(1);              // 'i'
```

- Characters have **ASCII** encodings just like in C:

```
cout << (int) s[0] << endl;    // 72
```

Operators

- **Concatenate** using + or += :

```
string s1 = "All";
s1 += "ison";                          // "Allison"
```

- **Compare** using relational operators (ASCII ABC ordering):

```
string s2 = "Percival";              // == != < <= > >=
if (s1 > s2 && s2 != "Merlin") {     // false
    ...
}
```

- Strings are **mutable** and can be changed (!):

```
s1.append(" Obourn")                 // "Allison Obourn"
s1.erase(3, 2);                      // "Allon Obourn"
s1[3] = '!';                         // "All!n Obourn"
```

String functions

- `#include <string>`

Member function name	What it does
***s*.append(*str*)**	Adds `str` onto the end of `s`
***s*.erase(*index*, *length*)**	Removes `length` number of characters starting at `index`
***s*.find(*str*)** ***s*.rfind(*str*)**	Returns the starting position of `str` in `s` (first occurrence for `find`, last for `rfind`). Returns `string::npos` otherwise.
***s*.insert(*index*, *str*)**	Inserts `str` into `s` starting at index
***s*.length() or *s*.size()**	Returns the number of characters in `s`
***s*.replace(*index*, *len*, *str*)**	Replaces `length` characters starting with the character at `index` with `str`.
***s*.substr(*start*, *length*) or *s*.substr(*start*)**	Returns a substring of `s`. Does not alter `s`.

String examples

```
string name = "Donald Knuth";
if(name.find("Knu") != string::npos ||
   name.substr(0, 3) == "Mr.") {
   name.erase(7, 5);                  // "Donald"
}


string word = "computer";
string newWord = "";
for(int i = 0; i < word.length(); i++) {
    newWord += word[i] + word[i];
}
newWord.insert(3, "---"); // "cco---ommppuutteerr"
```

Naming

- There is no single standard naming convention in C++
- The two standard naming conventions:
  - **camelCase** – the first word starts with a lowercase letter, all following words start with a capital
  - **underscores_between_words** – all words in lowercase with underscores separating them
- Pick one of these and stick with it for each assignment

String exercise

- Write a function `equalsIgnoreCase` that accepts two strings as parameters and returns `true` if they are the same ignoring case, `false` otherwise.
  - For example:

    `equalsIgnoreCase("computer", "computep")` should return `false`.

    `equalsIgnoreCase("computer", "COMPuter")` should return `true`.

char and cctype

- #include <cctype>
  - the same functions we used in C
  - notice, no .h in the include

Function name	Description
isalpha(*c*) isalnum(*c*) isdigit(*c*) isspace(*c*) isupper(*c*) ispunct(*c*) islower(*c*)	returns true if the given character is an alphabetic character from a-z or A-Z, a digit from 0-9, an alphanumeric character (a-z, A-Z, or 0-9), an uppercase letter (A-Z), a space character (space, \t, \n, etc.), or a punctuation character (. , ; !), respectively
tolower(*c*) toupper(*c*)	returns lower/uppercase equivalent of a character

```
//     index 01234567890123456789
string s = "Grace Hopper Bot v2.0";
if (isalpha(s[6]) && isnumer(s[18])
        && isspace(s[5]) && ispunct(s[19])) {
    cout << "Grace Hopper Smash!!" << endl;
}
```

Exercise solution

```
string toLowerCase(string s) {
    string newStr = "";
    for(int i = 0; i < s.length(); i++) {
        newStr += tolower(s[i]);
    }
    return newStr;
}

bool equalsIgnoreCase(string s1, string s2) {
    if(s1.length() != s2.length()) {
        return false;
    } else {
        return toLowerCase(s1) == toLowerCase(s2);
    }
}
```

C vs. C++ strings

- C++ has a string type but we can still use `char*`/`char[]`:
  - **C strings** (`char` arrays) and **C++ strings** (`string` objects)

- A string literal such as `"hi there"` is a C string.
  - This means we can't use any methods/behavior shown previously on them.
    - Example: `"cat".length()` will not work

- Converting between the two types:
  - `string("`***`text`***`")` `// C string to C++ string`
  - ***`string`***`.c_str()` `// C++ string to C string`

Beware: literals are still C strings

- `string s = "hi" + "there";  // C-string + C-string`
- `string s = "hi" + '?';  // C-string + char`
- `string s = "hi" + 41;  // C-string + int`
  - C strings can't be concatenated with `+`.
  - C-string `+` `char`/`int` produces garbage, not `"hi?"` or `"hi41"`.

- `string s = "hi";`
  `s += 41;  // "hi)"`
  - Adds character with ASCII value 41, `')'`, doesn't produce `"hi41"`.

- `int n = (int) "42";  // n = 0x7ffdcb08`
  - Bug; sets `n` to the memory address of the C string `"42"` (ack!).

C string bugs fixed

- `string s = string("hi") + "there";`
- `string s = "hi";  // convert to C++ string`
  `s += "there";`
  - These both compile and work properly.

- `string s = "hi";  // C++ string + char`
  `s += '?';  // "hi?"`
  - Works, because of auto-conversion.

- `s += to_string(41);  // "hi?41"`
  `int n = stoi("42");  // 42`
  - Explicit string <-> int conversion.

String exercise

- Write a function `trim` that accepts a string as a parameter and removes the

  - For example, `trim("    computer ")` should return `"computer"`.

Why doesn't this work?

```
void trim(string s) {
    int i = 0;
    while(isspace(s[i])) {
        i++;
    }
    s.erase(0, i);

    i = s.length() - 1;
    while(isspace(s[i])) {
        i--;
    }
    i += 1;
    s.erase(i, s.length() - i);
}
```

- **C++ strings are passed by value** unless you specify otherwise

Pointer solution

```
void trim(string* s) {
    int i = 0;
    while(isspace((*s)[i])) {
        i++;
    }
    (*s).erase(0, i);

    i = (*s).length() - 1;
    while(isspace((*s)[i])) {
        i--;
    }
    i += 1;
    (*s).erase(i, (*s).length() - i);
}
```

- To call this function write:

```
string word = "computer";
trim(&word);
```

lib132.cpp

- In your projects this quarter you can include a file called `lib132.cpp`
  - Include useful functions in this that you might want to reuse:
    - lab question solutions
    - lecture examples
      - `trim`, `toLowerCase` and `equalsIgnoreCase` from today
    - small problem solutions
    - code you write on your own