601.220 Intermediate Programming

Introduction to C++

C versus C++

If learning C is like learning "business English," learning C++ is like learning the rest of English

C versus C++

Sometimes programming in C is the best or only option

- You "inherited" C code
- No C++ compiler is available for system you're targeting
- Your software must work closely with the Linux kernel, or other C-based software

If we started a new project today, especially if it was big or involved many people, we'd probably choose C++

C versus C++

Classes – like Java classes

Templates – like Java generics

Standard Template Library – like `java.util`

More convenient text input & output

C versus C++

C++ is not a "superset" of C; most C programs don't immediately work in C++

Think of C and C++ as closely related but different languages

C versus C++

Many features/concepts from C language are also relevant in C++:

- types: `int`, `char`, `float`, `double`, pointer types
  - C++ adds `bool` (equals either `true` or `false`)
- numeric representations & properties
- operators: assignment, arithmetic, relational, logical
- arrays, pointers, `*` and `&`, pointer arithmetic
- control structures: `if/else`, `switch/case`, `for`, `while`, `do/while`
- pass-by-value (still the default), pass by address
- stack vs. heap, scope & lifetime
- (with minor differences) `struct`

C++

Our favorite tools work just as well with C++:

- `git`
- `make`
- `gdb`
- `valgrind`

C++

```cpp
// hello_world.cpp:
#include <iostream>

using std::cout;
using std::endl;

int main() {
    cout << "Hello world!" << endl;
    return 0;
}
```

```
$ g++ -std=c++11 -pedantic -Wall -Wextra -c hello_world.cpp
$ g++ hello_world.o -o hello_world
$ ./hello_world
Hello world!
```

C++

Programming stages same as for C:

- edit -> preprocess -> compile -> link -> execute

When compiling/linking:

- g++ instead of gcc
- `-std=c++11` instead of `-std=c99`
- `.cpp` instead of `.c`

```
$ g++ -std=c++11 -pedantic -Wall -Wextra -c hello_world.cpp
$ g++ hello_world.o -o hello_world
$ ./hello_world
Hello world!
```

C++

Options we used with `gcc` work with `g++` too

- `-o` to set name of executable
- `-c` to compile to `.o` file
- `-g` to include debug symbols
- `-Wall -Wextra -pedantic` for sensitive warnings & errors

C++: libraries

```
#include <iostream>
```

As with C, C++ library headers are included with < angle brackets >

For C++ headers, *omit the trailing* `.h`

- `<iostream>`, not `<iostream.h>`

```
#include "linked_list.h"
```

User-defined headers use " quotes " and end with `.h` as usual

C++: libraries

Can use familiar C headers: `assert.h`, `math.h`, `ctype.h`, `stdlib.h`, ...

When `#include`'ing, drop `.h` & add `c` at the beginning:

```
// hello_world_2.cpp:
#include <iostream>
#include <cassert>  // dropped .h, added c at beginning

using std::cout;
using std::endl;

int main(int argc, char *argv[]) {
    assert(argc > 1); // our old friend assert
    cout << "Hello " << argv[1] << "!" << endl;
    return 0;
}
```

C++: I/O

```
$ g++ -std=c++11 -pedantic -Wall -Wextra -c hello_world_2.cpp
$ g++ hello_world_2.o -o hello_world_2
$ ./hello_world_2 Everyone
Hello Everyone!
```

Note: `argc` and `argv` work just like in C

C++: I/O

`iostream` is the main C++ library for input and output

```
#include <iostream>
```

C++: I/O

```
using std::cout;
using std::endl;
```

C++ has *namespaces*.

- In C, when two things have the same name, we get errors (from compiler or linker) and confusing situations ("shadowing")
- In C++, items with same name can safely be placed in distinct "namespaces", similar to Java packages / Python modules

C++: namespaces

Most C++ functionality lives in namespace called `std`

If we didn't include:

```
using std::cout;
using std::endl;
```

at the top, then we would have to write the fully qualified name each time:

```
std::cout << "Hello world" << std::endl;
```

C++: namespaces

Do not use `using` in a header file

Doing so affects all the source files that include that header, even indirectly, which can lead to confusing name conflicts

- *Only* use `using` in source `.cpp` files
- This will be enforced in homework grading

Use fully qualified names (e.g. `std::endl`) in headers

C++: namespaces

```
using namespace std;  //too broad!
```

- This is a catch-all way to include everything in the `std` namespace, whether you need it all or not
- Causes confusion due to accidental name conflicts, so we disallow it in this course

C++: I/O - `std::cout` and <<

Text input & output in C++ are simpler than in C, thanks to C++'s stream operators and libraries

(We will not cover binary I/O in C++)

C++: I/O - `std::cout` and <<

```
cout << "Hello world!" << endl;
```

`cout` is our old friend, the standard output stream

- Like `stdout` in C

`endl` is the newline character

- C++ has `'\n'` too, but `endl` is usually preferred

<< is the *insert operator*

- replacing the placeholder syntax of `printf` in C

Comparing it with how it is written in C:

```
printf("Hello world!\n");
```

C++: I/O - std::cout and <<

Insert operator joins all the items to write in a "chain"

Leftmost item in chain is the stream being written to

```
cout << "We have " << inventory << " " << item << "s left,"
     << " costing $" << price << " per unit" << endl;
```

C++: I/O - std::cout and <<

```
// cpp_io_1.cpp:
#include <iostream>
using std::cout;
using std::endl;

int main() {
    int inventory = 44;
    double price = 70.07;
    const char *item = "chainsaw";

    cout << "We have " << inventory << " " << item << "s left,"
         << " costing $" << price << " per unit" << endl;
    return 0;
}
```

```
$ g++ -std=c++11 -pedantic -Wall -Wextra -c cpp_io_1.cpp
$ g++ cpp_io_1.o -o cpp_io_1
$ ./cpp_io_1
We have 44 chainsaws left, costing $70.07 per unit
```

C++: I/O - std::cout and <<

No format specifiers (%d, %s etc)

Instead, items to be printed are arranged in printing order; easier to read and understand

```
int inventory = 44;
double price = 70.70;
const char *item = "chainsaw";

cout << "We have " << inventory << " " << item << "s left,"
     << " costing $" << price << " per unit" << endl;
```

C++: I/O - `std::cout` and <<

An example of C++ I/O but also an example of *operator overloading*

<< usually does bitwise left-shift; but if operand on the left is a C++ stream (`cout`), << is the insert operator

```
cout << "Hello world!" << endl;
```

More on this later

C++: I/O - std::cout and <<

How much of C can we use in C++? Nearly everything.

```
// cpp_io_2.cpp:
#include <cstdio>

int main() {
    int inventory = 44;
    double price = 70.70;
    const char *item = "chainsaw";

    printf("We have %d %ss left costing $%f per unit\n",
           inventory, item, price);
    return 0;
}
```

```
$ g++ -std=c++11 -pedantic -Wall -Wextra -c cpp_io_2.cpp
$ g++ cpp_io_2.o -o cpp_io_2
$ ./cpp_io_2
We have 44 chainsaws left costing $70.700000 per unit
```

C++: I/O - std::cin and >>

How about scanf?

```
// cpp_io_3.cpp:
#include <iostream>
#include <string>  // new header -- not used in C

using std::cout; using std::cin;
using std::endl; using std::string;

int main() {
    cout << "Please enter your first name: ";
    string name;
    cin >> name;  // read user input into string object
    cout << "Hello, " << name << "!" << endl;
    return 0;
}
```

C++: I/O - std::cin and >>

```
$ g++ -std=c++11 -pedantic -Wall -Wextra -c cpp_io_3.cpp
$ g++ cpp_io_3.o -o cpp_io_3
$ echo Ed | ./cpp_io_3
Please enter your first name: Hello, Ed!
```

C++: I/O - `std::cin` and >>

```
cin >> name;
```

Reads one whitespace-delimited token from standard input and places the result in string `name`

>> is the *extraction operator*

C++: I/O - std::cin and >>

```
// smallest_word.cpp:
#include <iostream>
#include <string>
using std::cout; using std::cin;
using std::endl; using std::string;

int main() {
  string word, s_word;
  while(cin >> word) {
    if(s_word.empty() || word < s_word) s_word = word;
  }
  cout << s_word << endl;
  return 0;
}
```

C++: I/O - std::cin and >>

```
$ g++ -std=c++11 -pedantic -Wall -Wextra -c smallest_word.cpp
$ g++ smallest_word.o -o smallest_word
$ echo "the quick brown fox" | ./smallest_word
brown
```

C++: I/O - std::cin and >>

```
while(cin >> word) {
    // ...
}
```

`cin >> word` evaluates to true if the input stream is still in a "good state" (no error, no EOF) after reading the word

C++: I/O - std::cin.get()

```
// uppercase_cpp.cpp:
#include <iostream>
#include <cctype>
using std::cout; using std::cin; using std::endl;

int main() {
    char ch;
    while(cin.get(ch)) { // read single character
        ch = toupper(ch);
        cout << ch; // print single character
    }
    cout << endl;
    return 0;
}
```

C++: I/O - std::cin.get()

```
$ g++ -std=c++11 -pedantic -Wall -Wextra -c uppercase_cpp.cpp
$ g++ uppercase_cpp.o -o uppercase_cpp
$ echo "The Quick Brown Fox" | ./uppercase_cpp
THE QUICK BROWN FOX
```