Experiences in Teaching Modern C++ to Beginners

Ryan E. Dougherty

ryan.dougherty@asu.edu (@RyanDoughertyAZ)

School of Computing, Informatics, and Decision Systems Engineering

Arizona State University

http://www.public.asu.edu/~redoughe/

About Me

- Ph.D. Candidate in CS at ASU, through August 2019
- Instructor of record for 8 classes at Arizona State University
  - 2 are CSE100: Principles of Programming with C++
- Primarily using C++ for performance
- Solving problems in automatically generating test suites for large-scale software systems for interaction testing.

About CSE100

- First course in programming C++
- Primarily taught introductory concepts from (at most) C++98
- 3 hrs/week of lecture, and 1 hr/week of lab
- Goal: bring the class more into the “modern” spotlight
- Taught Fall 2018 and Spring 2019
- Some facts about the two offerings:
  - 155 students at the end of both classes
  - 51 freshmen, 46 sophomores, 34 juniors, 22 seniors, 2 other
  - 30 biomedical engineers, 63 electrical engineers, 0 computer science!

CSE100 - Existing Syllabus

- Data Types
- (Basic) Library Functions
- Control Structure Flow
- User-Created Functions
- Scope
- Built-in Arrays
- Classes
- Searching/Sorting

CSE100 - My Syllabus

- Data Types
- auto
- (Basic) Library Functions
- Control Structure Flow
- User-Created Functions
- Scope
- Built-in Arrays (kinda)
- Range-based for
- vector, map, …
- Classes
- Searching/Sorting
- Pointers (including smart ptrs)
- Copy/Move/etc. Constructors, Destructor, custom operators, and rvalue refs
- Lambdas
- Inheritance
- Polymorphism
- Exceptions
- Templates + STL

CSE100 Workload

- Fall 2018:
  - 4 tests + final + quizzes
  - 8 assignments
  - 14 labs
- Spring 2019:
  - 2 tests + final
  - 8 assignments
  - 13 labs
  - 1 writing assignment

Some Best Practices Taught

- Not returning by const value, or by reference (most of the time)
- Avoid pitfalls (new/delete, avoid unnecessary copying, etc.)
- Use STL built-ins whenever possible
- Indentation + curly brackets
- (Almost) always auto
- Understanding *what* techniques to use to solve a problem, instead of how to code a solution.
- Use compiler flags

Lecture Techniques

- Q+A style
- Online videos
- Reviews for tests
- Practice handouts

A Lecture Example

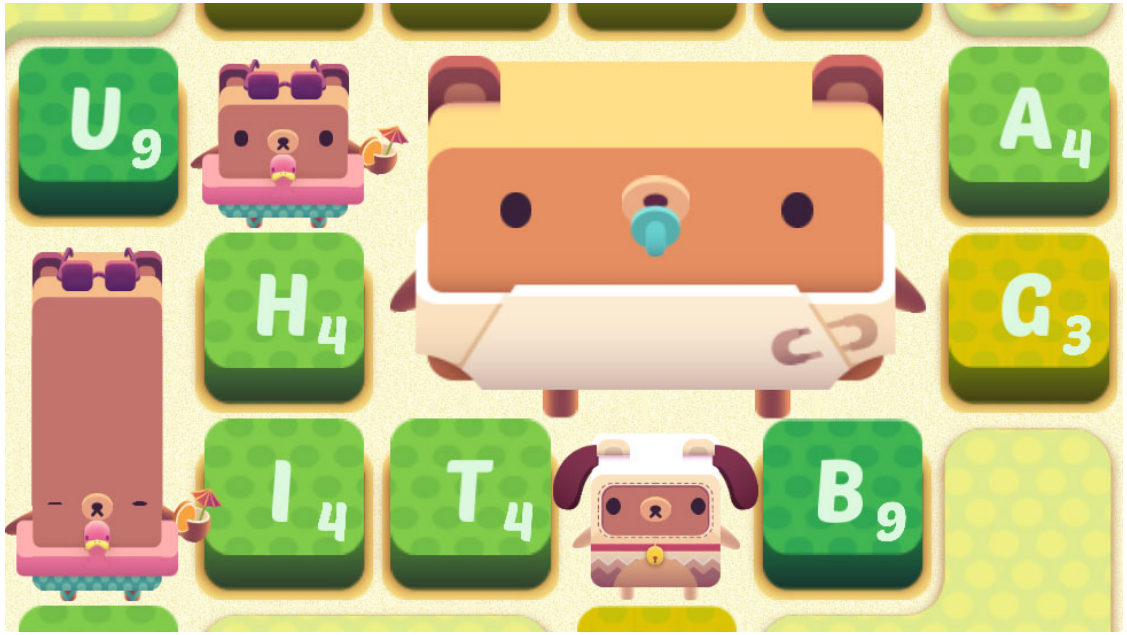
```
#include <iostream>

int main() {
    int n;
    std::cin >> n;
    int arr[n];
}
```

Assignments

- Goal: make assignments not reiterating concepts, but rather present a problem *that requires those concepts* (or at least is easier with them)
- Problems of a mathematical nature
  - Checking whether a solution is correct (Latin squares)
  - Inferring information about data (computing statistics from a list of values)
  - (Dis)proving a conjecture (related to sums of powers of integers)
- Problems that have real-world consequences
  - Minimize a given C++ file and the corresponding executable, using any technique necessary.
  - Determine which of three sorting (bubble, selection, std::sort) algorithms and searching (binary, linear) algorithms is faster for different std::vector contents

Assignments (cont.)

- Problems that are fun and interesting.
  - Write an Alphabear solver. (Image: https://bit.ly/2VCy1Ic)
  - Program an In-and-Out menu
  - Detect collision between a player and an obstacle
- (Spring 2019) Improve on an existing assignment using a “new” feature of C++.



Written Paper (Spring 2019)

- Directions: watch two videos from previous C++ conferences (2017 or 2018), summarize them, and give insight into how to approach problems.
  - One must be “technical”, and the other must be “applied”
  - Tries to emphasize formal writing about an unfamiliar subject, and to reflect on how the talks improve their programming.

What’s the Result?

Some Insights

- Know your stuff. **Really** know your stuff.
- Keep the programming environment consistent.
- Enforce indentation and curly brackets.
- Do many examples, from start to finish.
- Self-restraint is important for yourself, as well as for the students.
- Understand what people have done before you, and assume that they are smart.
- Code examples are important.
- Teaching Assistants and graders can make or break your class.

Some Insights (cont.)

- Don’t teach bitwise operations to beginners.
- Lecture videos are extremely useful.
- 1-on-1 help in office hours helped several students immensely.
- Include more “short answer” questions *on tests only*, rather than fewer ”solve this new problem” questions.
  - However, putting the latter on assignments instead was received well.
- Students’ abilities to approach new material *increases* throughout the semester.
- Interest from the community is mirrored in students (constexpr).
- Use an IDE whenever possible (save + compile + run inconsistency).

Conclusion

- Teaching introductory programming will always be a challenge, but we are getting closer to bridging the existing gaps.

- Special thanks to Zachary Cohen, Conner Durkee, Rex Marsh, Francisco Mata, Lucas Selby, Adin Warner, and Charles Wolfe for suggestions and feedback.