Homework 7

This homework covers the following topics from class:

- Function palooza part 4, OOP palooza, part 1
    - Polymorphism (Subtype, Ad-hoc, Parametric - Generics vs. Templates)
    - OOP Intro, OOP History

Each problem below has a time associated with it. This is the amount of time that we expect a thoroughly-prepared student would take to solve this problem on an exam. We understand, however, that as you learn these concepts, these questions may take more time to solve, and we don't want to overwhelm you with homework. For that reason, only **required** questions need to be completed when you submit this homework. That said, the exams will cover all materials covered (a) in class, (b) from class projects, (c) in the required homework problems, and (d) in the optional homework problems, so at a minimum, please review the optional problems and/or use them as exam prep materials.

You must turn in a PDF file with your answers via Gradescope - you may include both typed and hand-written solutions, so long as they are legible and make sense to our TAs. Make sure to clearly label each answer with the problem number you're solving so our TAs can more easily evaluate your work.

For homework #7, you must complete the following **required** problems:

- FUNC6: Generics, Templates (25 min)
- FUNC7: Generics, Templates, Duck Typing (10 min)
- FUNC8: Parametric Polymorphism (5 min)
- FUNC9: Duck Typing in Statically Typed Languages (5 min)
- FUNC10: Templates, Generics (15 min)

For homework #7, you may optionally complete the following additional problems:

N/A

FUNC6: Generics, Templates (25 min)

For this problem, you are to write two versions of a container class called *Kontainer* that holds up to 100 elements, and allows the user to add a new element as well as find/return the minimum element, regardless of the data type of the elements (e.g., integers, floating-point numbers, strings, etc.). Don't worry about the user overflowing the container with items. You will be implementing this class using templates in C++ *and* generics in a statically-typed language of your choice (e.g., Java, Swift, Kotlin, Go, Haskell, etc.).

Part A: (10 min.) Show your implementation of the Kontainer class using C++ templates:

Your answer:

Part B: (10 min.) Show your implementation of the Kontainer class using generics in a statically-typed language of your choice - feel free to use any online resources you like if you choose a different language for your Your answer:

Hint: If you haven't programmed in another statically-typed language other than C++ before, there are tons of online examples and online compilers which make this easy! Here are links to a few online compilers: Kotlin, Java, Swift.

Your answer:

Part C: (5 min.) Now try creating two Kontainers of type double and type String with your generic class. Add some values to each Kontainer.

Show your code here:

Your answer:

FUNC7: Generics, Templates, Duck Typing (10 min)

(10 min.) Discuss the benefits and drawbacks of (a) the template approach, (b) the generics approach, and (c) the duck typing approach for implementing generic functions/classes. Consider such things as: performance, compilation time, clarity of errors, code size, and type-checking (when does it occur - at runtime or compile-time). If you were designing a new language and had to pick one approach, which one would you choose and why?

Your answer:

FUNC8: Parametric Polymorphism (5 min)

(5 min.) Explain why dynamically-typed languages can't support *parametric polymorphism*.

Your answer:

FUNC9: Duck Typing in Statically Typed Languages (5 min)

(5 min.) Explain how we can accomplish something like duck-typing in statically-typed languages that use templates. How is this similar to duck typing in dynamically typed languages, and how is it different (e.g., are there any pros/cons of either approach)?

Your answer:

FUNC10: Templates, Generics (15 min)

Consider the following C++ container class which can hold pointers to many types of values:

```
class Holder {
private:
  static const int MAX = 10;
  void *arr_[MAX]; //  array of void pointers to objs/values
  int count_;
public:
  Holder() {
   count_ = 0;
  }
  void add(void *ptr) {
   if (count_ >= MAX) return;
   arr_[count_++] = ptr;
  }
  void* get(int pos) const {
   if (pos >= count_) return nullptr;
   return arr_[pos];
  }
};

int main() {
 Holder h;
 string s = "hi";
 int i = 5;
 h.add(&s);
 h.add(&i);

 // get the values from the container
 std::string* ps = (std::string *)h.get(0);
 cout << *ps << std::endl;  // prints: hi
 int* pi = (int *)h.get(1);
 cout << *pi << std::endl;  // prints: 5
}
```

This class does not use C++ templates, yet by using void pointers (void *) which are a generic type of pointer that can legally point to any type of value, it allows us to store a variety of (pointers to) values in our container object. This is how we used to do things before C++ added templates.

Part A: (5 min.) Discuss the pros and cons of the approach shown above vs. C++ templates.

Your answer:

Part B: (5 min.) How is this approach similar or different to generics in languages like Java?

Your answer:

Part C (5 min): What change might we make to the above program to make it work more like a bounded generic class?

Your answer: