Using Pointers in the Projects

CS236 - Discrete Structures
Brett Decker and Michael Goodrich
Fall 2020

C++ and Pointers

Many students ask us whether or not they *must* use pointers in the projects for CS236. The answer is no, you do not *have* to use pointers. But there are many reasons to use pointers.

Pros

Pointers are an essential part of the C and C++ programming languages. The use of pointers is the main reason that software engineers use these languages. It is also the main reason that software engineers avoid these languages. So why should you consider using pointers?

Reduce Copying

First, because pointers will reduce copying of objects which can lead to frustrating bugs. When you create an object memory must be allocated. It is either allocated on the stack, e.g. when you have the statement `std::string name = "John";`, or on the heap when you use the `new` keyword, e.g. `std::string* name = new("Smith");`. Note that when you allocate memory on the heap you only get a pointer (the address) to the memory, not the actual object. Objects on the stack do *not* persist outside of the scope (think braces) in which they were created. Thus, for example, you will lose an object created on the stack in a function once you exit the function. If you return the object, you are actually making a copy of the object and returning that. If you pass an object as a parameter to a function, it is *passed by value* which means a copy is sent to the function. The function can now only modify the copy, which means you will have to return the modified copy to keep any of the changes. This is poor coding style and gets really messy and confusing fast (even the explanation is confusing). This often results in bugs and code that is not understood. These error usually start appearing in Project 3 and continue into Projects 4 and 5. The copying continues every time you push or index into a `std::vector` or other collection. If you use a pointer, there is only one object. You simply pass the pointer to functions so they can directly modify the object. If you create a new object in a function on the heap, you do need to store the address, otherwise it will not be accessible outside the function. The usual way to do this is to add the pointer to a collection, such as a vector. When you push into a `std::vector` it just stores the address (note that this usually saves memory because the vector doesn't have to store the entire object), no copy is made. When you access an element in the vector, it gives you the address of the object. With pointers, there is no need to worry about extra copies of the original object.

Inheritance

Second, pointers are key to inheritance in C++. Inheritance is not possible with abstract classes if pointers are not used. Let's look at a concrete example from Project 1. The Parallel and Max approach uses an abstract class `Automaton` as the base class for all other specialized automaton classes, e.g. `MatcherAutomaton`. If you try to create a `std::vector<Automaton>` it will fail compilation, because the compiler is not allowed to create an object of an abstract class. An abstract class can only be used as a type, which is why `std::vector<Automaton*>` is allowed – this vector states it stores pointers of type `Automaton`. If you try to make the `Automaton` class concrete, the compiler will allow you to create the vector of `Automaton` objects, but this will result in a loss of information when you push an object of `MatcherAutomaton` into the vector.

Future

Finally, if you want to claim competence of C or C++ on your resume, you need to understand and be able to use pointers. You also need to understand how the stack and the heap work and the difference between pass-by-reference and pass-by-value. Understanding these topics – and being comfortable with them – will be an expectation of employers and colleagues if you put C++ on your resume.

Cons

There are some cons to using pointers, but we'll try to give some help in each subsection.

Lack of Familiarity

This is by far the most intimidating and biggest obstacle to tackle. If you don't feel comfortable with pointers already, it can be hard to want to try to learn them for the large project in this course.

Tracking the Heap

It is often discouraging or intimidating to students to hear that they will have to clean up the memory on the heap if they use pointers. Memory leaks are hard to debug, but they are quite avoidable. What you need to remember is that when you have a pointer variable you should not assign it a new value (address) unless the current value is stored somewhere else. In the projects in the course you will often have a `std::vector` or other collection. As long as you don't remove a pointer from your vector it will persist for the entirety of your program. It is often easiest to store pointers after creation immediately into the desired vector. This could be done in two ways:

```
MatcherAutomaton* comma = new MatcherAutomaton(",", TokenType.COMMA);
machines.push_back(comma);
```

or in one line

```
    machines.push_back(new MatcherAutomaton(",", TokenType.COMMA));
```

You only need to clean up memory (delete your pointers) when you are done with the objects. For the projects in this course, you often will just do this at the end of your program. It is as simple as iterating through your vectors and deleting each pointer right before your main function returns. You will also want to clear out the collection (as good coding practice). Here are two possible ways:

```
    for (unsigned int i = 0; i < machines.size(); i++) {
        delete machines[i];
    }
    machines.clear();
```

or

```
    for (Automaton* machine : machines) {
        delete machine;
    }
    machines.clear();
```

References

C++ also makes use of references, which are different than pointers. References also eliminate the unnecessary copying of objects when passed to functions. You will want to pass vector objects by reference in your functions. For more information about references and pointers, see this resources Geeks for Geeks and cplusplus.

Using References and Pointers

Here are some usages of references and pointers:

```
int main(int argc, char** argv) {
     std::vector<Token*> allTokens;

     functionModifiesVectorAsPointer(&allTokens);
     functionModifiesVectorAsReference(allTokens);
     functionOnlyReadsVector(allTokens);
}

void functionModifiesVectorAsPointer(const std::vector<Token*>* tokens) {
     tokens->push_back(new Token(...));
}
```

```
void functionModifiesVectorAsReference(std::vector<Token*>& tokens) {
     tokens.push_back(new Token(...));
}

void functionOnlyReadsVector(const std::vector<Token*>& tokens) {
     tokens.at(...);
}
```

Conclusion

There are students that are success in this course that do use pointers and there are also successful students that do not. The opposite is also true. In the end, you need to way the pros and cons to make your decision. We encourage the use of pointers because we believe the pros outweigh the cons. Get help for the instructors and teaching assistants about how to use pointers. Help each other out. This will increase your learning and probability of success.