Challenges: The biggest challenge of this project was dealing with structs with long paths and arrays. The key to solving this problem was to continuously loop through the path and get the field of each new struct then set curr_type equal to it. So no matter when the path ended the curr_type was set correctly.

Testing:Explanations written above each test

```
Test1:Checks for an error if an object of an undefined struct is made
TEST(BasicSemanticCheckerTests, MainMakesNewUndefinedStruct) {
 stringstream in(build_string({
       "void main() {S s = new S}",
     }));
 SemanticChecker checker;
 try {
   ASTParser(Lexer(in)).parse().accept(checker);
   FAIL();
 } catch (MyPLException& ex) {
   string msg = ex.what();
   ASSERT_TRUE(msg.starts_with("Static Error:"));
 }
}
Test2:Checks for an error if a primitive type and struct have the same name
TEST(BasicSemanticCheckerTests, VarAndStructWithSameName) {
 stringstream in(build_string({
       "struct S {double val, S s}",
       "void main() {",
       "  int x = 0",
       "  S x = new S",
       "  char y = 'a'",
       "}",
     }));
 SemanticChecker checker;
 try {
   ASTParser(Lexer(in)).parse().accept(checker);
   FAIL();
 } catch (MyPLException& ex) {
   string msg = ex.what();
   ASSERT_TRUE(msg.starts_with("Static Error:"));
 }
}
Test3:Checks for an error if int x is not declared to ints
TEST(BasicSemanticCheckerTests, BadIntAssign) {
 stringstream in(build_string({
       "void main() {",
       "  int x1 = 3.5 + true",
       "}",
     }));
 SemanticChecker checker;
 try {
   ASTParser(Lexer(in)).parse().accept(checker);
   FAIL();
 } catch (MyPLException& ex) {
   string msg = ex.what();
   ASSERT_TRUE(msg.starts_with("Static Error:"));
 }
}
```
Test4:Checks for an error if int y is given not int with a variable
```
TEST(BasicSemanticCheckerTests, VarDeclWithTypeMismatch) {
 stringstream in(build_string({
       "void main() {",
       "  bool x = true",
       "  int y = x + 1",
       "}",
     }));
 SemanticChecker checker;
 try {
   ASTParser(Lexer(in)).parse().accept(checker);
   FAIL();
 } catch (MyPLException& ex) {
   string msg = ex.what();
   ASSERT_TRUE(msg.starts_with("Static Error:"));
 }
}
```
Test5:Checks for an error if a function returns an invalid type from another function
```
TEST(BasicSemanticCheckerTests, ReturnFunctionCallMismatch) {
 stringstream in(build_string({
       "int f() {return 5}",
       "double t() {return f()}",
       "void main() {}",
     }));
 SemanticChecker checker;
 try {
   ASTParser(Lexer(in)).parse().accept(checker);
   FAIL();
 } catch (MyPLException& ex) {
   string msg = ex.what();
   ASSERT_TRUE(msg.starts_with("Static Error:"));
 }
}
Test6:Checks for an error if and if stmt does not have a bool with a function call
TEST(BasicSemanticCheckerTests, NonBoolFunctionCallIfCondition) {
 stringstream in(build_string({
       "int f() {return 5}",
       "void main() {if (f()) {}}",
     }));
 SemanticChecker checker;
 try {
   ASTParser(Lexer(in)).parse().accept(checker);
   FAIL();
 } catch (MyPLException& ex) {
   string msg = ex.what();
   ASSERT_TRUE(msg.starts_with("Static Error:"));
 }
}
Test7:Checks for an error if the VarDecl in a for loop does not have the right type
TEST(BasicSemanticCheckerTests, BoolInVarDeclInForLoop) {
 stringstream in("void main() {for(int i = true; i < 1; i = i + 1) {}}");
 SemanticChecker checker;
 try {
   ASTParser(Lexer(in)).parse().accept(checker);
   FAIL();
 } catch (MyPLException& ex) {
   string msg = ex.what();
   ASSERT_TRUE(msg.starts_with("Static Error:"));
 }
}
Test8:Checks for an error if x is not given a string from a struct array call
TEST(BasicSemanticCheckerTests, StructPathInvalidArrayType) {
 stringstream in(build_string({
       "struct S {array double val, S s}",
       "void main() {",
       "  S s = new S",
       "  s.s.val = new double[10]",
       "  string x = s.s.val[0]",
       "}",
     }));
 SemanticChecker checker;
 try {
   ASTParser(Lexer(in)).parse().accept(checker);
   FAIL();
 } catch(MyPLException& ex) {
   string msg = ex.what();
   ASSERT_TRUE(msg.starts_with("Static Error:"));
 }
}
Test9:Checks for an error if the if stmt does not get a bool from struct
TEST(BasicSemanticCheckerTests, StructPathInvalidTypeInIf) {
 stringstream in(build_string({
       "struct S {double val, S s}",
       "void main() {",
       "  S s = new S",
       "  if(s.s.val) {}",
       "}",
     }));
 SemanticChecker checker;
 try {
   ASTParser(Lexer(in)).parse().accept(checker);
   FAIL();
 } catch(MyPLException& ex) {
   string msg = ex.what();
   ASSERT_TRUE(msg.starts_with("Static Error:"));
 }
}
Test10:Checks for an error if while condition does not get a bool from an array
TEST(BasicSemanticCheckerTests, ArrayCallInvalidTypeInIf) {
 stringstream in(build_string({
       "void main() {",
       "  array double val = new double[10]",
       "  while(v[0]) {}",
       "}",
     }));
 SemanticChecker checker;
 try {
   ASTParser(Lexer(in)).parse().accept(checker);
   FAIL();
 } catch(MyPLException& ex) {
   string msg = ex.what();
   ASSERT_TRUE(msg.starts_with("Static Error:"));
 }
```

```
}
```