```
Install-Package Microsoft.Net.Compilers
Install-Package Microsoft.CodeAnalysis
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis.CSharp.Syntax;
namespace ConsoleApp2
{
  public class ClassCollector : CSharpSyntaxWalker
  {
    public ClassCollector(){
      this.Classes = new List<ClassDeclarationSyntax>();
    }
    public IList<ClassDeclarationSyntax>Classes{get;}
    public override void VisitClassDeclaration(
       ClassDeclarationSyntax node)
    {
      if (node.Identifier.ToString().StartsWith("A") ||
         node.Identifier.ToString().StartsWith("P"))
      {
        this.Classes.Add(node);
      }
    }
  }
  class Program
  {
    static void Main(string[] args)
    {
      var syntaxTree = CSharpSyntaxTree.ParseText(
        @"using␣System;
using␣System.Collections;
using␣System.Linq;
using␣System.Text;
namespace␣HelloWorldApplication
{
class␣Program
{
static␣void␣Main(string[]␣args)
{
var␣i=0;
var␣d=5.5;
```

```
Console.WriteLine(""Hello␣World"");
}
}
␣␣class␣AA1␣␣{␣␣␣␣private␣int␣i;␣␣}
␣␣class␣AA2␣␣{␣␣␣␣private␣int␣i;␣␣}
}");
       var listOfSyntaxes = new List<SyntaxTree> {
          syntaxTree };
       var root = syntaxTree.GetRoot() as
          CompilationUnitSyntax;
       CSharpCompilation compilation = CSharpCompilation.
          Create("test", listOfSyntaxes);
       var namespaceSyntax = root.Members.OfType<
          NamespaceDeclarationSyntax>().First();
       var programClassSyntax = namespaceSyntax.Members.
          OfType<ClassDeclarationSyntax>().First();
       var mainMethodSyntax = programClassSyntax.Members.
          OfType<MethodDeclarationSyntax>().First();
       var variableDeclarations = root.DescendantNodes().
          OfType<LocalDeclarationStatementSyntax>();
       var semanticModel = compilation.GetSemanticModel(
          syntaxTree);
       var x = semanticModel.LookupSymbols(1);
       foreach (var item in variableDeclarations)
       {
         Console.Write("DECLARATION:␣");
         Console.WriteLine(item.Declaration);
         Console.Write("TYPE:␣");
         Console.WriteLine(item.Declaration.Type);
         var symbolInfo = semanticModel.GetSymbolInfo(
            item.Declaration.Type);
       }
       Console.WriteLine(mainMethodSyntax.ToString());
       var classCollector = new ClassCollector();
       classCollector.Visit(root);
       Console.WriteLine($"classes␣␣with␣'A'␣is␣{
          classCollector.Classes.Count}");
       Console.ReadKey();
    }
  } }
```