Part 3: Getting Started in C#

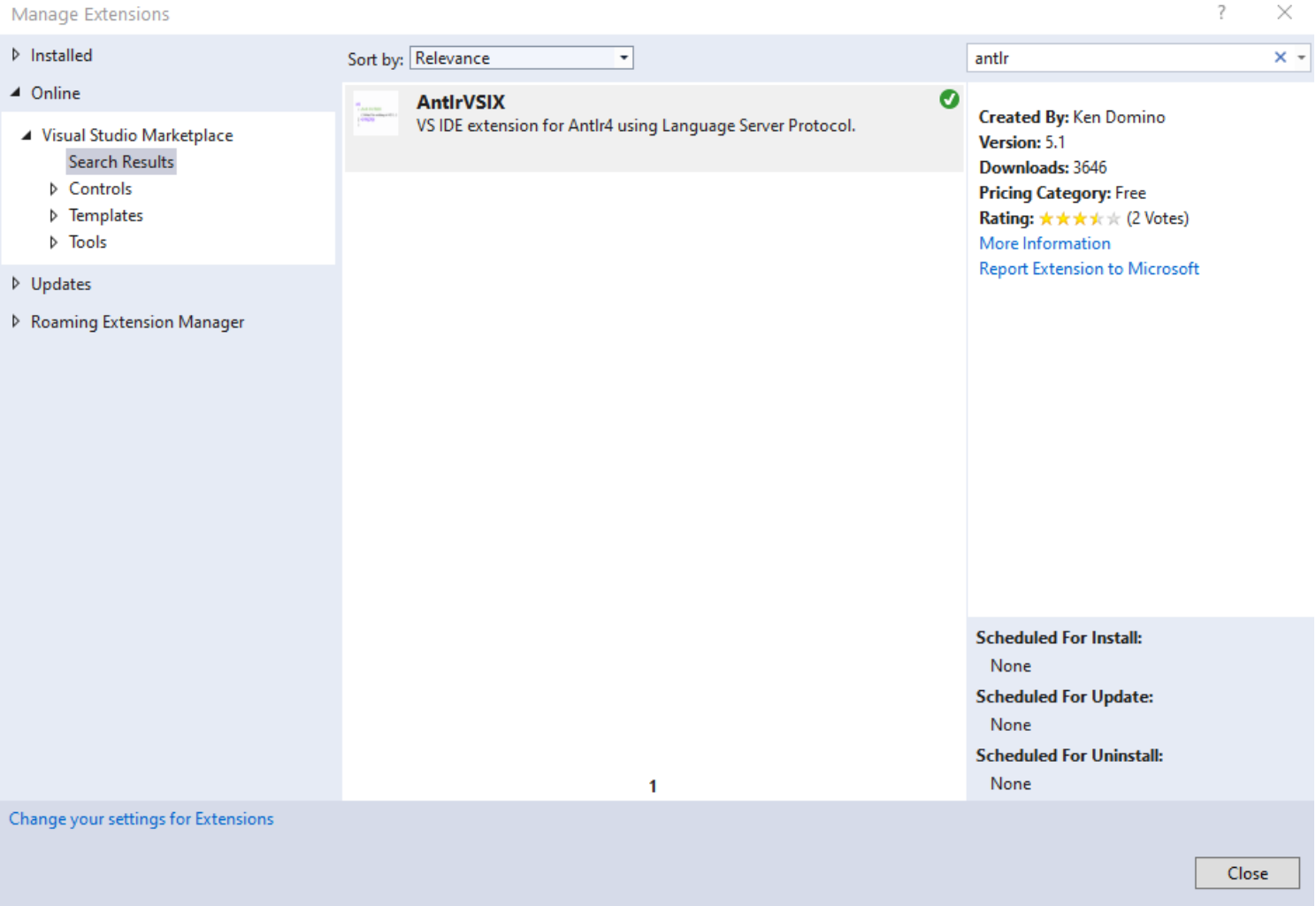
By:
Morteza Zakeri
PhD Student
Iran University of Science and Technology
Winter 2020

Agenda

- ANTLR in Visual Studio 2019 and C#
- Back to the Grammar
- Implement Your First Visitor
- Listeners vs Visitors

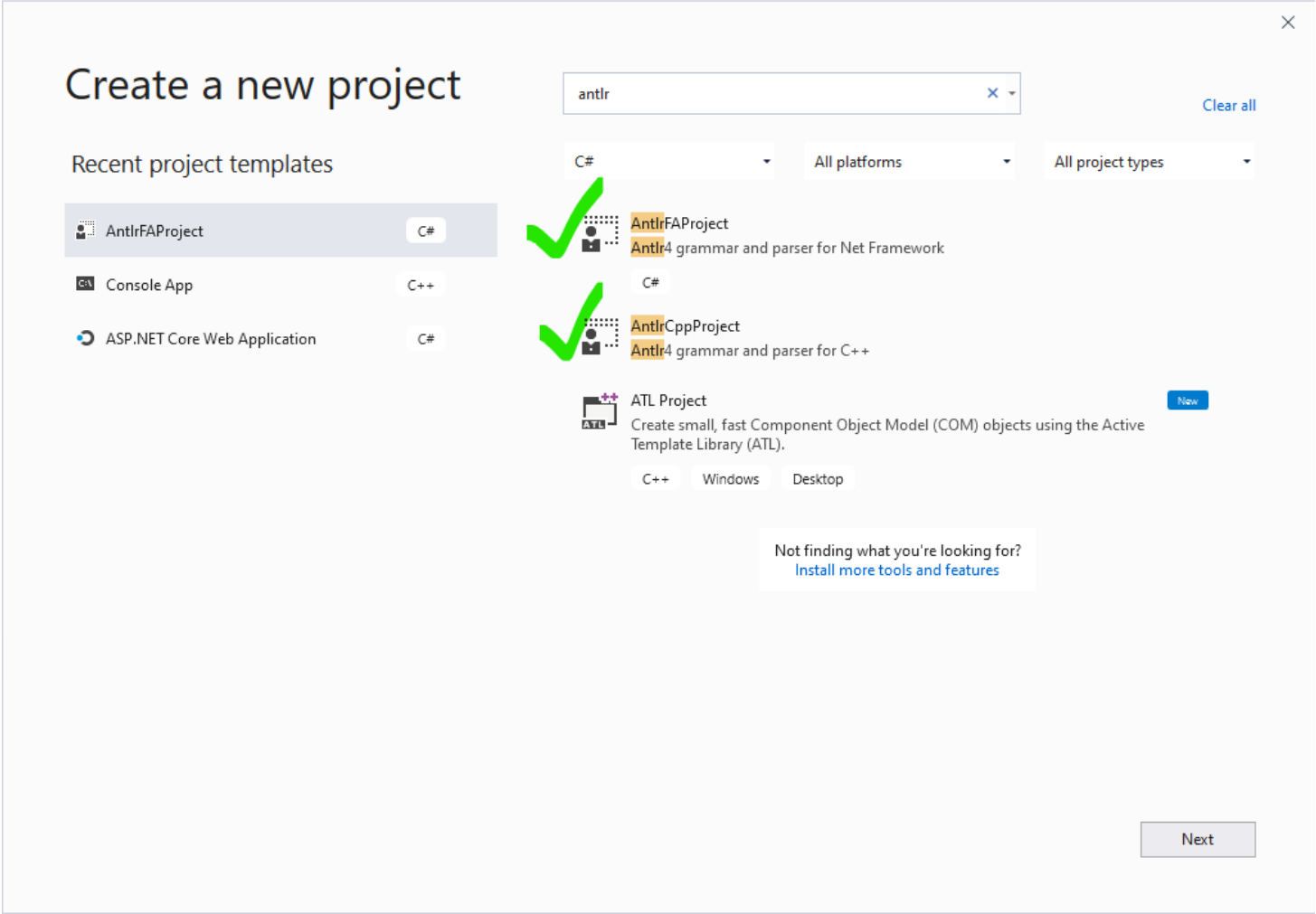
ANTLR in Visual Studio 2019 and C#

- Install AntlrVSIX



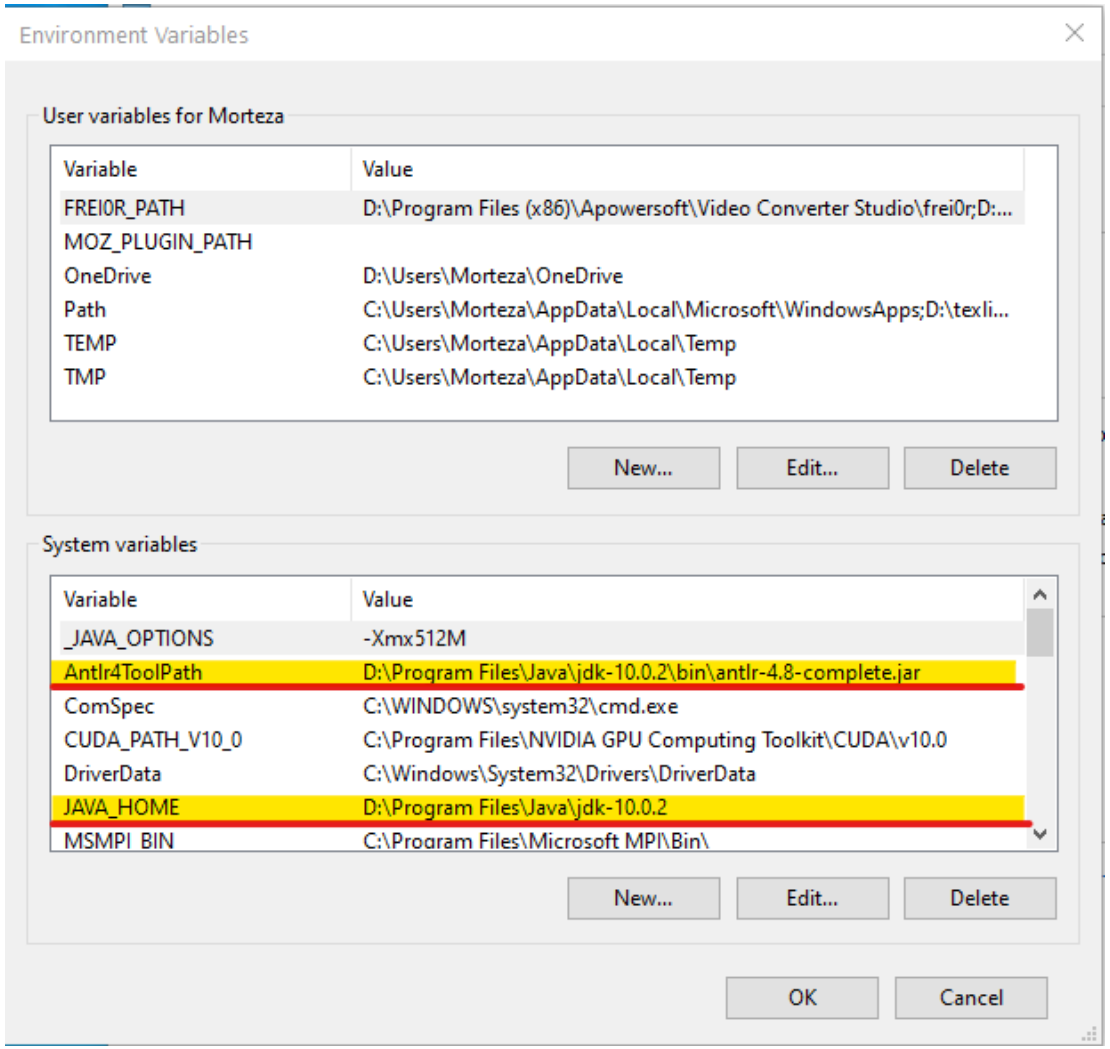
ANTLR in Visual Studio 2019 and C#

- Create a new project
- Use .NET 4.6.x ☺



ANTLR in Visual Studio 2019 and C#

- Download and Install JDK
- Download ANTLR jar file
  - antlr.org/download/antlr-4.8-complete.jar
- Set Environment variables
  - JAVA_HOME
  - Antrl4ToolPath



Antlr4BuildTasks

- The purpose of the package is to integrate the Antlr tool into the building of NET programs that reference Antlr using the Java-based Antlr tool.
- The advantage of this package is that it decouples the Java-based Antlr tool from the package itself.
- Make sure you do not have a version skew between the Java Antlr tool and the runtime versions.

Back to the Grammar

- Expr.g4

```
// Template generated code from Antlr4BuildTasks.Template v 2.1
grammar Expr;
prog:   (expr NEWLINE)* ;
expr:   expr ('*'|'/') expr
    |   expr ('+'|'-') expr
    |   INT
    |   '(' expr ')'
    ;
NEWLINE : [\r\n\t]+;
INT     : [0-9]+ ;
```

Build and Run Your Project

```
var str = new AntlrInputStream(input);
var lexer = new ExprLexer(str);
var tokens = new CommonTokenStream(lexer);
var parser = new ExprParser(tokens);
parser.BuildParseTree = true;
var tree = parser.expr();


System.Console.WriteLine("parse completed.");
System.Console.WriteLine(tokens.OutputTokens());
System.Console.WriteLine("-------");
System.Console.WriteLine(tree.OutputTree(tokens));
System.Console.WriteLine("=======");
```

```
C:\Users\Morteza\source\repos\AntlrConsolePilot2\AntlrConsolePilot2\bin\Debug\AntlrConsolePilot2.exe
parse completed.
Token 0 8 channel DEFAULT_TOKEN_CHANNEL 1
Token 1 3 channel DEFAULT_TOKEN_CHANNEL +
Token 2 8 channel DEFAULT_TOKEN_CHANNEL 2
Token 3 1 channel DEFAULT_TOKEN_CHANNEL *
Token 4 8 channel DEFAULT_TOKEN_CHANNEL 3
Token 5 -1 channel DEFAULT_TOKEN_CHANNEL <EOF>

-------
( expr
  ( expr
    ( DEFAULT_TOKEN_CHANNEL i=0 txt=1 tt=8
  ) )
  ( DEFAULT_TOKEN_CHANNEL i=1 txt=+ tt=3
  )
  ( expr
    ( expr
      ( DEFAULT_TOKEN_CHANNEL i=2 txt=2 tt=8
    ) )
    ( DEFAULT_TOKEN_CHANNEL i=3 txt=* tt=1
    )
    ( expr
      ( DEFAULT_TOKEN_CHANNEL i=4 txt=3 tt=8
) ) ) )

=======
-------
(expr (expr 1) + (expr (expr 2) * (expr 3)))
```

Implement Your First Visitor

```
7   using Antlr4.Runtime.Misc;
8   using Antlr4.Runtime.Tree;
9   using IToken = Antlr4.Runtime.IToken;
10  using ParserRuleContext = Antlr4.Runtime.ParserRuleContext;
11
12  namespace AntlrConsolePilot2
13  {
14      class MyExprVisitor<Result>: ExprBaseVisitor<Result>
15
16      {
17          public override Result VisitTerminal(ITerminalNode node)
18          {
19              Console.WriteLine(node.GetText());
20              return base.VisitTerminal(node);
21          }
22      }
23  }
24
```

```
29          var visitor = new MyExprVisitor<string>();
30          var result  = visitor.Visit(tree);
```

Select C:\Users\Morteza\source\repos\AntlrConsolePilot2\AntlrConsolePilot2\bin\Debug\AntlrConsolePilot2.exe

```
Token 3 1 channel DEFAULT_TOKEN_CHANNEL *
Token 4 8 channel DEFAULT_TOKEN_CHANNEL 3
Token 5 -1 channel DEFAULT_TOKEN_CHANNEL <EOF>

-------
( expr
  ( expr
    ( DEFAULT_TOKEN_CHANNEL i=0 txt=1 tt=8
  ) )
  ( DEFAULT_TOKEN_CHANNEL i=1 txt=+ tt=3
  )
  ( expr
    ( expr
      ( DEFAULT_TOKEN_CHANNEL i=2 txt=2 tt=8
    ) )
    ( DEFAULT_TOKEN_CHANNEL i=3 txt=* tt=1
    )
    ( expr
      ( DEFAULT_TOKEN_CHANNEL i=4 txt=3 tt=8
) ) ) )

-------
1
+
2
*
3
-------
(expr (expr 1) + (expr (expr 2) * (expr 3)))
```

Listener vs Visitor

- **Listener methods** are called automatically by the ANTLR provided walker object, whereas visitor methods must walk their children with explicit visit calls. Forgetting to invoke visit() on a node's children means those subtrees don't get visited.

Listener vs Visitor

- Listener methods can’t return a value, whereas visitor methods can return any custom type. With listener, you will have to use mutable variables to store values, whereas with visitor there is no such need.

Listener vs Visitor

- Listener uses an explicit stack allocated on the heap, whereas visitor uses call stack to manage tree traversals. This might lead to StackOverflow exceptions while using visitor on deeply nested ASTs.
- Read more:
  - https://saumitra.me/blog/antlr4-visitor-vs-listener-pattern/
  - https://jakubdziworski.github.io/java/2016/04/01/antlr_visitor_vs_listener.html

Assignments and Projects

Assignments and Projects

- **Assignment 0: C++ (Python)** lexical and syntax analyzing
- **Assignment 1:** Instrumenting the **C++ (Python)** source codes based on independent execution paths.
- **Assignment 2:** Refactoring **C++ (Python)** source codes based on clean code principles (Robert C. Martin book).
- **Assignment 3:** Extracting class diagram (annotated directed graph) from **C++ (Python)** source code.
- **Assignment 5 (Optional):** Identifying the design patterns in the code.

Assignments and Projects

- **Final Project:** Put all together ☺
  - A comprehensive tool for program analysis!

Assignment 0: Build Compiler Front-end

- Put your compiler knowledge in practice!
  - A software engineer with compiler science.
- Download the C++ (Python) grammar from ANTLR GitHub pages.
  - https://github.com/antlr/grammars-v4

Assignment 1: Instrumentation

- **Part 1: Simple instrumenting**
- Input: C++ (Python) source code
- Output: C++ (Python) instrumented source code

Assignment 1: Instrumentation

```
1  #include "cod.hpp"
2  #include <string.h>
3  bool tool_texture(const char *texture, int map_version)
4  {
5      if (tool_textures[map_version])
6      {
7          unsigned int i = 0;
8          while (tool_textures[map_version][i])
9          {
10             if (!strcmp(tool_textures[map_version][i], texture))
11             {
12                 return true;
13             }
14             i++;
15         }
16     }
17     return false;
18 }
19 bool spawn_classname(const char *classname, int map_version)
20 {
21     if (spawn_classnames[map_version])
22     {
23         unsigned int i = 0;
24         while (spawn_classnames[map_version][i])
25         {
26             if (!strcmp(spawn_classnames[map_version][i], classname))
27             {
28                 return true;
29             }
30             i++;
31         }
32     }
33     return false;
34 }
```

```
1  #include "cod.hpp"
2  #include <string.h>
3  bool tool_texture(const char *texture, int map_version)
4  {
5   cout << "p(1)" << endl;
6      if (tool_textures[map_version])
7      {
8  cout << "p(2)" << endl;
9          unsigned int i = 0;
10         while (tool_textures[map_version][i])
11         {
12 cout << "p(3)" << endl;
13             if (!strcmp(tool_textures[map_version][i], texture))
14             {
15 cout << "p(4)" << endl;
16                 return true;
17             }
18             i++;
19         }
20     }
21     return false;
22 }
23 bool spawn_classname(const char *classname, int map_version)
24 {
25  cout << "p(5)" << endl;
26     if (spawn_classnames[map_version])
27     {
28 cout << "p(6)" << endl;
29         unsigned int i = 0;
30         while (spawn_classnames[map_version][i])
31         {
32 cout << "p(7)" << endl;
33             if (!strcmp(spawn_classnames[map_version][i], classname))
34             {
35 cout << "p(8)" << endl;
36                 return true;
37             }
38             i++;
39         }
40     }
41     return false;
42 }
```

Assignment 1: Instrumentation

- Another example (GCD program):

```c
#include <stdio.h>
int main()
{
    int n1, n2, i, gcd;
    printf("Enter two integers: ");
    scanf("%d %d", &n1, &n2);
    for(i=1; i <= n1 && i <= n2; ++i)
    {
        // Checks if i is factor of both integers
        if(n1%i==0 && n2%i==0)
            gcd = i;
    }
    printf("G.C.D of %d and %d is %d", n1, n2, gcd);
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
 cout << "p(1)" << endl;
    int n1, n2, i, gcd;
    printf("Enter two integers: ");
    scanf("%d %d", &n1, &n2);
    for(i=1; i <= n1 && i <= n2; ++i)
    {
cout << "p(2)" << endl;
        // Checks if i is factor of both integers
        if(n1%i==0 && n2%i==0)
            {
cout << "p(3)" << endl;
 gcd=i;
}
    }
    printf("G.C.D of %d and %d is %d", n1, n2, gcd);
    return 0;
}
```

Assignment 1: Instrumentation

- **Problem solving idea**
  - Implement listener for two Non-terminals
  - Rewriting the Input Stream

```
public override void EnterStatement([NotNull]CPP14Parser.StatementContext context)
{
        // put your code here
}

public override void EnterFunctionbody([NotNull] CPP14Parser.FunctionbodyContext
context){
        // put your code here
}
```

Assignment 1: Instrumentation

- **Part 2: Deal with some challenges**
  - Block with Single Statements

```
1  #include <stdio.h>
2  int main()
3  {
4      int n1, n2, i, gcd;
5      printf("Enter two integers: ");
6      scanf("%d %d", &n1, &n2);
7      for(i=1; i <= n1 && i <= n2; ++i)
8      {
9          [redacted] of both integers
10
11
12     }
13     printf("G.C.D of %d and %d is %d", n1, n2, gcd);
14     return 0;
15 }
```

```
1  #include <stdio.h>
2  int main()
3  {
4   cout << "p(1)" << endl;
5      int n1, n2, i, gcd;
6      printf("Enter two integers: ");
7      scanf("%d %d", &n1, &n2);
8      for(i=1; i <= n1 && i <= n2; ++i)
9      {
10 cout << "p(2)" << endl;
11 [redacted]
12
13
14
15
16
17     }
18     printf("G.C.D of %d and %d is %d", n1, n2, gcd);
19     return 0;
20 }
```

Assignment 1: Instrumentation

- **Problem solving idea**
  - Implement listener for two Non-terminals:

```
public override void EnterStatement([NotNull] CPP14Parser.StatementContext context)
{
    var parentType = context.Parent.GetType();
    if (parentType == typeof(SelectionstatementContext) || parentType == typeof(IterationstatementContext))
    {
        branchNumber++;
        var child = context.children.FirstOrDefault();
        if (child.GetType() == typeof(CompoundstatementContext))
            tokenStreamRewriter.InsertAfter(context.Start, $"\ncout << \"p({branchNumber})\" << endl;");
        else
            tokenStreamRewriter.Replace(context.Start, context.Stop, $"{{\ncout << \"p({branchNumber})\" << endl;\n
    }
}
```

Assignment 1: Instrumentation

- **Part 3: Deal with some challenges**
  - Loops make some problems, e.g.:
    - *gcd.exe < 6 4*

<p(1)p(2)p(3)p(2)p(3)p(2)p(2)end>

< p(1)p(2)p(3)end >

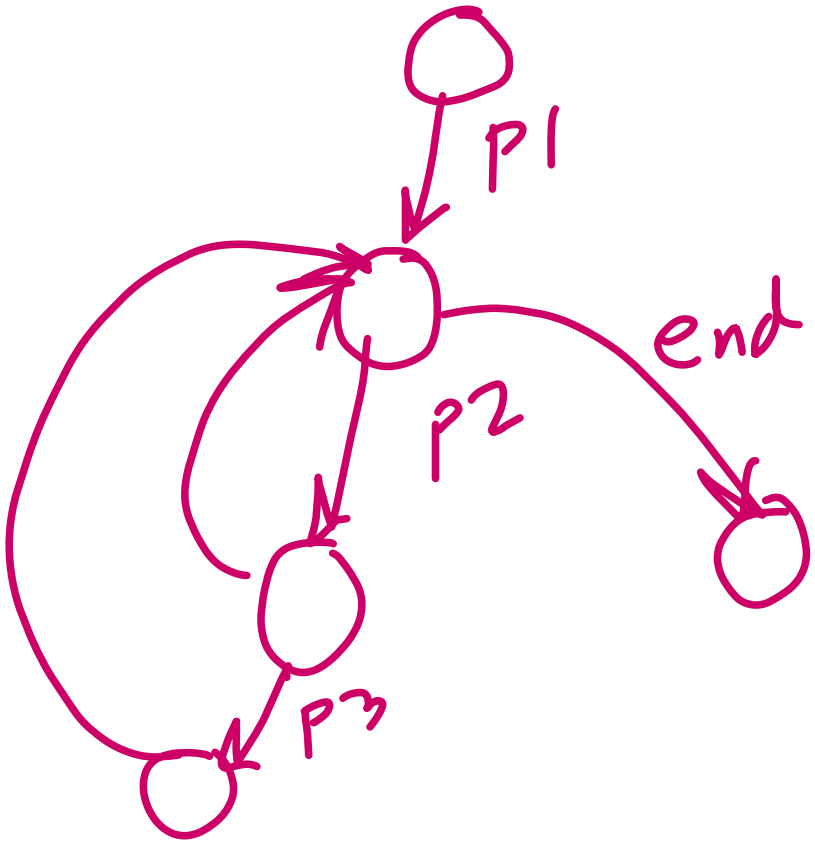
  - Or
    - *gcd.exe < 24 16*

<p(1)p(2)p(3)p(2)p(3)p(2)p(2)p(3)p(2)p(2)p(2)p(2)p(3)p(2)
p(2)p(2)p(2)p(2)p(2)p(2)p(2)end>

```
#include <stdio.h>
int main()
{
 cout << "p(1)" << endl;
    int n1, n2, i, gcd;
    printf("Enter two integers: ");
    scanf("%d %d", &n1, &n2);
    for(i=1; i <= n1 && i <= n2; ++i)
    {
cout << "p(2)" << endl;
        // Checks if i is factor of both integers
        if(n1%i==0 && n2%i==0)
            {
cout << "p(3)" << endl;
 gcd=i;
}
    }
    printf("G.C.D of %d and %d is %d", n1, n2, gcd);
    return 0;
}
```

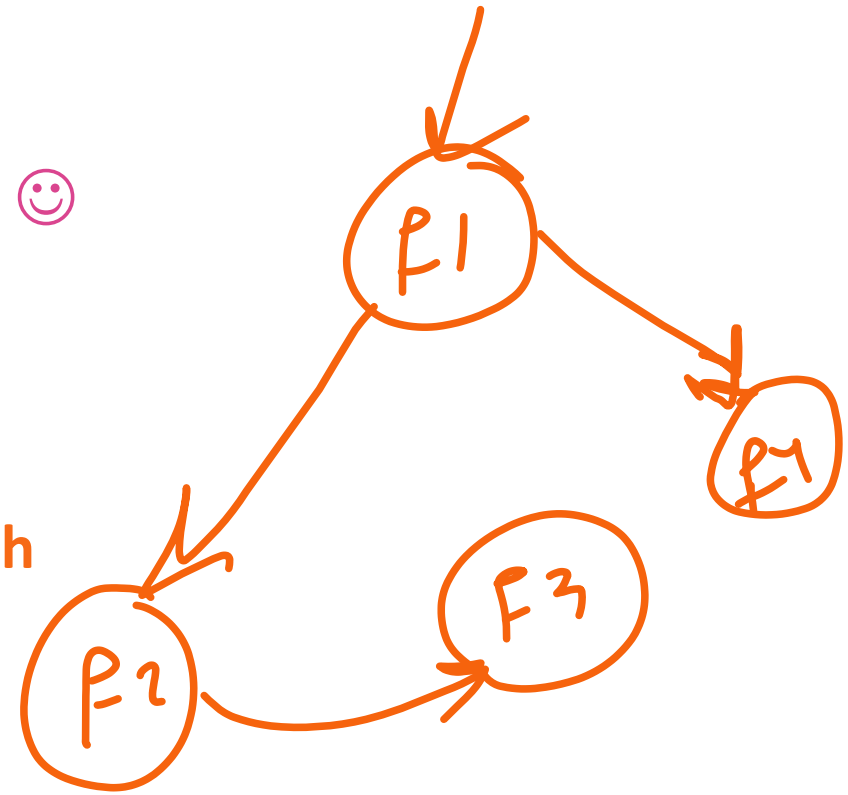
Assignment 1: Instrumentation

- **Tips for a solution (function level):**
  - Reduce each sequence to **independent execution path**
  - Independent execution paths can be computed using **CFG**
  - Use Antlr to extract CGF and Independent execution paths:
    i. <p(1)p(2)p(3)end>
    ii. <p(1)p(2)end>
    iii. <p(1)end>



Assignment 1: Instrumentation

- **Tips for a solution (program level):**
  - We cared about functions, i.e., unit testing ☺
    - Testing at function level
  - What about program? ☹
    - Testing at program level (integration testing)
    - In the level of program we need **call flow graph**
  - Use Antlr to extract call flow graph



Assignment 2: Clean Code

- **Part 1:** Find the clean code violence in names, functions, comments, and formatting, based on the **Clean Code** book by Robert C. Martin.
  - Read chapters 1 – 5
  - Using ANTLR
- Report the result to the programmer

Robert C. Martin

Assignment 2: Clean Code

- **Part 2:** Refactor the founded clean code violence automatically.
  - Again using ANTLR
  - Using NLTK for naming
    - https://www.nltk.org/

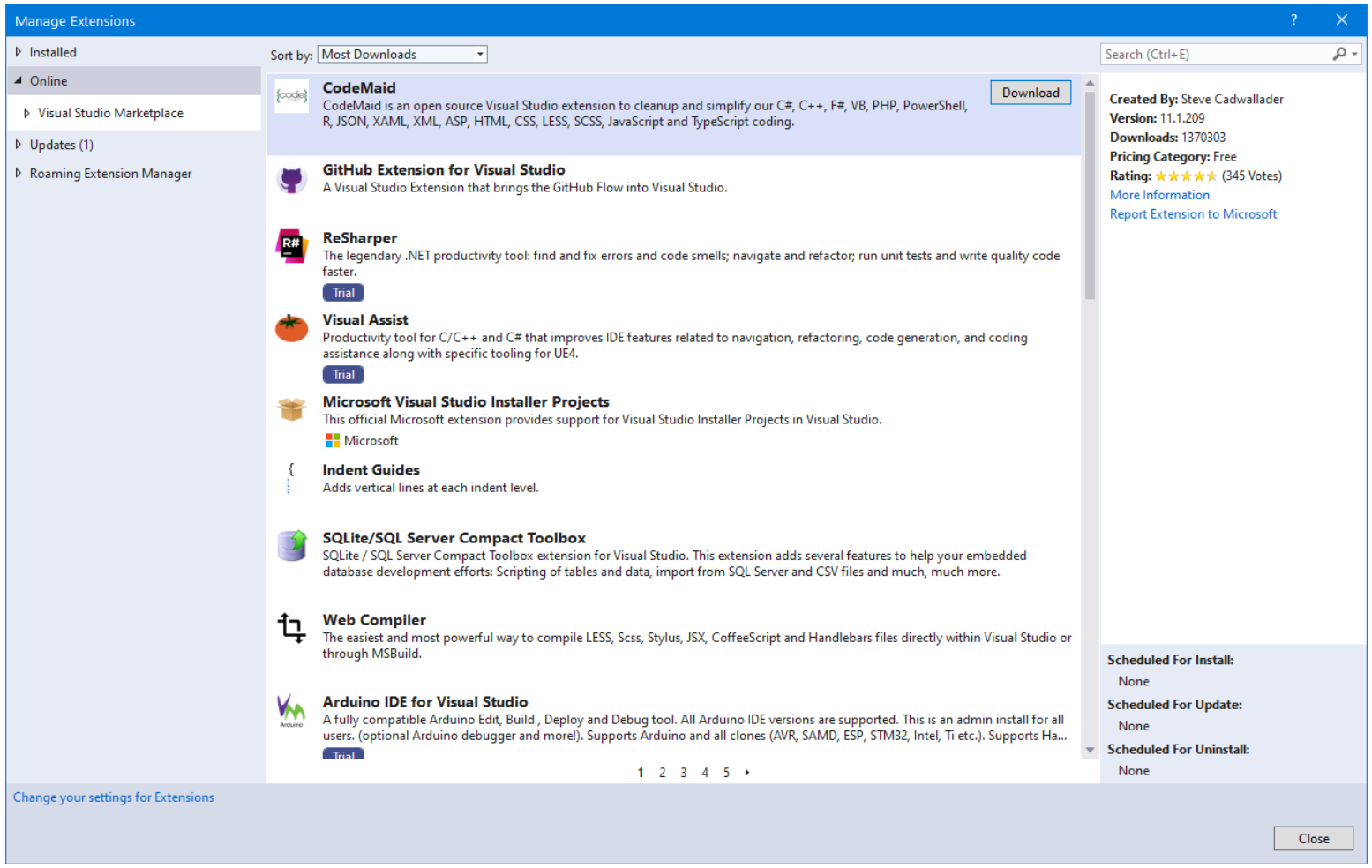
Assignment 3: Class Diagram Extraction

- We will talk about it in the next sessions.

Project

- **Final Project:** Put all assignment together
- Policies
  - Student with odd digit in last of their ID number *(e.g. 9513231117):*
    - **Prepare your tools for Python3**
  - Other students *(e.g. 9513231118):*
    - **Prepare your tools for CPP14**
  - Deadlines:
    - **Assignment 1 and 2 deadlines: 20 Farvardin 1399.**
    - **Assignment 3 deadline will be determined later!**

Commercialize Your Tool



References

1. AntlrVSIX
   - https://github.com/kaby76/AntlrVSIX/blob/master/doc/readme.md
2. Getting Started With ANTLR in C#
   1. https://dzone.com/articles/getting-started-with-antlr-in-c
3. The Definitive ANTLR 4 Reference
   - Terence Parr, The Pragmatic Programmers, LLC; 2012.
4. ANTLR 4 Official Website:
   - http://www.antlr.org/

Thank you for your attention!

- **Do you have any question?**
  - **m-zakeri@live.com**

Appendix

- Extension Methods in C#
- $ in C# Strings

Extension Methods in C#

- Extension methods enable you to "add" methods to existing types
  - without creating a new derived type, recompiling, or otherwise modifying the original type.
  - special kind of static method, but they are called as if they were instance methods on the extended type.
  - The most common extension methods are the LINQ standard query operators that add query functionality to the existing System.Collections.IEnumerable and System.Collections.Generic.IEnumerable<T> types.

Extension Methods in C#

```
namespace ExtensionMethods
{
    public static class MyExtensions
    {
        public static int WordCount(this String str)
        {
            return str.Split(new char[] { ' ', '.', '?' },
                             StringSplitOptions.RemoveEmptyEntries).Length;
        }
    }
}
```

```
string s = "Hello Extension Methods";
int i = s.WordCount();
```

$ in C# Strings

- $ is short-hand for String.Format() and is used with string interpolations, which is a new feature of C# 6.

```
var anInt = 1;
var aBool = true;
var aString = "3";
var formated = string.Format("{0},{1},{2}", anInt, aBool, aString);
```

Now becomes:

```
var anInt = 1;
var aBool = true;
var aString = "3";
var formated = $"{anInt},{aBool},{aString}";
```