HW2 Pointer Analysis

Introduction

This project aims to develop a pointer analysis data flow analyzer for C program. Your program should be capable of handling scalar variables, and consideration should be given to two levels of pointers. We will again utilize LLVM as the IR form for this project. You are required to implement the algorithm based on this paper (https://drive.google.com/file/d/1TxgFsX3DryJQlaI9mU118e_k6XPC6XcG/view), and calculate the following information for given C programs:

- Reaching Definition and Flow Equations
  - TREF: The set of trees of referenced variables
  - TGEN: The set of trees of generated variables
  - TDEF: The set of reaching definitions
  - TEQUIV: The set of equivalent trees
- Data Dependences:
  - flow dependence
  - output dependence

Input program

- no loop, or other complex constructs, only straight lines of assignment statements

- RHS of the assignment statements may have complex expression
- up to two level of pointer **
- An example of the input program:

```
1  void foo() {
2         int a, b, c, d, e, x, y, f, g, h, i;
3         int *p;
4  S1:    a  = b + c;
5  S2:    p  = &y;
6  S3:    d  = b + c;
7  S4:    f  = a + d + x + y;
8  S5:    g  = a + d + x + y;
9  S6:    *p = i + 2;
10 S7:    h  = a + d + x + y;
11 S8:    f  = y + 1;
12 }
```

Output format

- Print IN/Out sets and the data dependence (flow/output) for each statement:

```
Sa:--------------------
TREF: {}
TGEN: {}
DEP: {}
TDEF: {}
TEQUIV: {}
```

- TREF,TGEN:
  - {var, ...}
  - e.g. {a, b}
- DEP:

```
{
    var: Sa--->Sb (flow)
    var: Sa-O->Sb (output)
}
```

- TDEF:
  - {(var, stmt), ...}
  - e.g. TDEF:{(a, S1), (p, S2), (d, S3)}

- TEQUIV:
  - {(var, var), ...}
  - TEQUIV:{(*pp, p), (**pp, y), (*p, y)}
- An example of the analysis output

```
S1:--------------------
TREF: {b, c}
TGEN: {a}
DEP: {}
TDEF: {(a, S1)}
TEQUIV: {}

S2:--------------------
TREF: {}
TGEN: {p}
DEP:{}
TDEF:{(a, S1), (p, S2)}
TEQUIV:{(*p, y)}

S3:--------------------
TREF: {b, c}
TGEN: {d}
DEP:{}
TDEF:{(a, S1), (p, S2), (d, S3)}
TEQUIV:{(*p, y)}

S4:--------------------
TREF: {a, d, x, y, *p}
TGEN: {f}
DEP:{
    a: S1--->S4
    d: S3--->S4
}
TDEF:{(a, S1), (p, S2), (d, S3), (f, S4)}
TEQUIV:{(*p, y)}
...
```

Suggested Implementation Steps:

1. Refer to the slide in class "cs5404_Basic_chapter3.2.pdf".
2. Based on A TECHNIQUE FOR ANALYZING POINTER AND STRUCTURE REFERENCES IN PARALLEL RESTRUCTURING COMPILERS (https://drive.google.com/file/d/1TxgFsX3DryJQlaI9mU118e_k6XPC6XcG/view)
3. Implement p.212 to p.216 in the paper and test “icpp.c” to see if steps outputs are same with the paper.

4. Test 4 testcases (https://drive.google.com/drive/folders/16OVWZEG02gAASSBuTAM2OS4Q7mlEvVP1) and print the output of the pass analysis.

Testcase Result (output with [] can be ignored)

- 4 testcases (https://drive.google.com/drive/folders/16OVWZEG02gAASSBuTAM2OS4Q7mlEvVP1)
- icpp

```
S1:--------------------
TREF: {}
TGEN: {p}
DEP:{}
TDEF:{(p, S1)}
TEQUIV:{(*p, x)}

S2:--------------------
TREF: {}
TGEN: {pp}
DEP:{}
TDEF:{(p, S1), (pp, S2)}
TEQUIV:{(*p, x), (*pp, p), (**pp, x)}

S3:--------------------
TREF: {pp}
TGEN: {*pp, p}
DEP:{
    pp: S2--->S3
    p: S1-O->S3
}
TDEF:{(p, S3), (pp, S2), (*pp, S3)}
TEQUIV:{(*pp, p), (**pp, y), (*p, y)}

S4:--------------------
TREF: {p, [*pp]}
TGEN: {*p, y}
DEP:{
    p: S3--->S4
    [*pp: S3--->S4]
}
TDEF:{(p, S3), (pp, S2), (*pp, S3), (*p, S4), (y, S4)}
TEQUIV:{(*pp, p), (**pp, y), (*p, y)}
```

- icpp2

```
S1:--------------------
TREF: {}
TGEN: {p}
DEP:{}
TDEF:{(p, S1)}
TEQUIV:{(*p, x)}

S2:--------------------
TREF: {}
TGEN: {pp}
DEP:{}
TDEF:{(p, S1), (pp, S2)}
TEQUIV:{(*p, x), (*pp, p), (**pp, x)}

S3:--------------------
TREF: {pp}
TGEN: {*pp, p}
DEP:{
    pp: S2--->S3
    p: S1-O->S3
}
TDEF:{(p, S3), (pp, S2), (*pp, S3)}
TEQUIV:{(*pp, p), (**pp, y), (*p, y)}

S4:--------------------
TREF: {p, [*pp]}
TGEN: {*p, y}
DEP:{
    p: S3--->S4
    [*pp: S3--->S4]
}
TDEF:{(p, S3), (pp, S2), (*pp, S3), (*p, S4), (y, S4)}
TEQUIV:{(*pp, p), (**pp, y), (*p, y)}

S5:--------------------
TREF: {*pp, pp, *p, p, y, [**pp]}
TGEN: {**pp, y}
DEP:{
    *pp: S3--->S5
    pp: S2--->S5
    *p: S4--->S5
    p: S3--->S5
    y: S4--->S5
    y: S4-O->S5
}
TDEF:{(p, S3), (pp, S2), (*pp, S3), (*p, S4), (y, S5), (**pp, S5)}
TEQUIV:{(*pp, p), (**pp, y), (*p, y)}
```

- icpp3

```
S1:--------------------
TREF: {}
TGEN: {p}
DEP:{}
TDEF:{(p, S1)}
TEQUIV:{(*p, x)}

S2:--------------------
TREF: {}
TGEN: {pp}
DEP:{}
TDEF:{(p, S1), (pp, S2)}
TEQUIV:{(*p, x), (*pp, p), (**pp, x)}

S3:--------------------
TREF: {pp}
TGEN: {*pp, p}
DEP:{
    pp: S2--->S3
    p: S1-O->S3
}
TDEF:{(p, S3), (pp, S2), (*pp, S3)}
TEQUIV:{(*pp, p), (**pp, y), (*p, y)}

S4:--------------------
TREF: {p, *pp}
TGEN: {*p, y}
DEP:{
    p: S3--->S4
    *pp: S3--->S4
}
TDEF:{(p, S3), (pp, S2), (*pp, S3), (*p, S4), (y, S4)}
TEQUIV:{(*pp, p), (**pp, y), (*p, y)}

S5:--------------------
TREF: {*pp, pp, p}
TGEN: {**pp, y}
DEP:{
    *pp: S3--->S5
    pp: S2--->S5
    p: S3--->S5
    y: S4-O->S5
}
TDEF:{(p, S3), (pp, S2), (*pp, S3), (*p, S4), (y, S5), (**pp, S5)}
TEQUIV:{(*pp, p), (**pp, y), (*p, y)}
```

- foo

```
S1:--------------------
TREF: {b, c}
TGEN: {a}
DEP:{}
TDEF:{(a, S1)}
TEQUIV:{}

S2:--------------------
TREF: {}
TGEN: {p}
DEP:{}
TDEF:{(a, S1), (p, S2)}
TEQUIV:{(*p, y)}

S3:--------------------
TREF: {b, c}
TGEN: {d}
DEP:{}
TDEF:{(a, S1), (p, S2), (d, S3)}
TEQUIV:{(*p, y)}

S4:--------------------
TREF: {a, d, x, y, *p}
TGEN: {f}
DEP:{
    a: S1--->S4
    d: S3--->S4
}
TDEF:{(a, S1), (p, S2), (d, S3), (f, S4)}
TEQUIV:{(*p, y)}

S5:--------------------
TREF: {a, d, x, y, *p}
TGEN: {g}
DEP:{
    a: S1--->S5
    d: S3--->S5
}
TDEF:{(a, S1), (p, S2), (d, S3), (f, S4), (g, S5)}
TEQUIV:{(*p, y)}

S6:--------------------
TREF: {p, i}
TGEN: {*p, y}
DEP:{
    p: S2--->S6
}
TDEF:{(a, S1), (p, S2), (d, S3), (f, S4), (g, S5), (*p, S6), (y, S6)}
TEQUIV:{(*p, y)}

S7:--------------------
TREF: {a, d, x, y, *p}
TGEN: {h}
```

```
DEP:{
    a: S1--->S7
    d: S3--->S7
    y: S6--->S7
    *p: S6--->S7
}
TDEF:{(a, S1), (p, S2), (d, S3), (f, S4), (g, S5), (*p, S6), (y, S6), (h, S7)}
TEQUIV:{(*p, y)}

S8:--------------------
TREF: {y, *p}
TGEN: {f}
DEP:{
    y: S6--->S8
    *p: S6--->S8
    f: S4-O->S8
}
TDEF:{(a, S1), (p, S2), (d, S3), (f, S8), (g, S5), (*p, S6), (y, S6), (h, S7)}
TEQUIV:{(*p, y)}
```

Requirements

- LLVM version: 17.0.2 (https://github.com/llvm/llvm-project/releases/tag/llvmorg-17.0.2)
- Deadline: 2024/1/7 23:59
- Please upload the following files to NTHU eeclass:
  - 3 files (hw2.cpp, Makefile, hw2_<student_id>_report.pdf), don't need to compress
  - Source code
    - hw2.cpp (pass source code)
    - Makefile
      - how to build your .ll
      - TA will run "make hw2.so (http://hw2.so)" to build the .so
  - Report
    - hw2_<student_id>_report.pdf
    - content:
      1. How to run your program (e.g. any paths that TA should modify in your Makefile)
      2. Display the output of the released testcases that you can handle
      3. Experiment report – how you implement the pass
- Late submission rules:

- submit before 2024/1/14 23:59 (for HW1 and HW2)
- 20 points deduction (for HW2)

Grading Policy

- 5 testcases:
  - icpp.c
  - icpp2.c
  - icpp3.c
  - foo.c
  - hidden testcase

Rank	Request
D	Report only
C- ~ B-	Cannot pass any testcases
B	Pass 1 testcase
B+	Pass 2 testcases
A-	Pass 3 testcases
A	Pass 4 testcases
A+	Pass 5 testcases

HW2 Pass Template

```
#include "llvm/IR/PassManager.h"
#include "llvm/Passes/PassBuilder.h"
#include "llvm/Passes/PassPlugin.h"

using namespace llvm;

namespace {

struct HW2Pass : public PassInfoMixin<HW2Pass> {
  PreservedAnalyses run(Function &F, FunctionAnalysisManager &FAM);
};

PreservedAnalyses HW2Pass::run(Function &F, FunctionAnalysisManager &FAM) {
  errs() << "[HW2]: " << F.getName() << '\n';
  return PreservedAnalyses::all();
}

} // end anonymous namespace

extern "C" ::llvm::PassPluginLibraryInfo LLVM_ATTRIBUTE_WEAK
llvmGetPassPluginInfo() {
  return {LLVM_PLUGIN_API_VERSION, "HW2Pass", LLVM_VERSION_STRING,
          [](PassBuilder &PB) {
            PB.registerPipelineParsingCallback(
                [](StringRef Name, FunctionPassManager &FPM,
                   ArrayRef<PassBuilder::PipelineElement>) {
                  if (Name == "hw2") {
                    FPM.addPass(HW2Pass());
                    return true;
                  }
                  return false;
                });
          }};
}
```

Makefile Sample

```
LLVM_CONFIG=/path/to/your/llvm_build/bin/llvm-config

CXX=`$(LLVM_CONFIG) --bindir`/clang
CXXFLAGS=`$(LLVM_CONFIG) --cppflags` -fPIC -fno-rtti
LDFLAGS=`$(LLVM_CONFIG) --ldflags`
IRFLAGS=-Xclang -disable-O0-optnone -fno-discard-value-names -S -emit-llvm
OPT=`$(LLVM_CONFIG) --bindir`/opt

.PHONY: all test run clean
all: hw2.so test

test: test1.ll

hw2.so: hw2.cpp
	$(CXX) -shared -o $@ $< $(CXXFLAGS) $(LDFLAGS)

test.ll: test.c
	$(CXX) $(IRFLAGS) -o $@ $<

run: test.ll hw2.so
	$(OPT) -disable-output -load-pass-plugin=./hw2.so -passes=hw2 $<

clean:
	rm -f *.o *.ll *.so
```

Hidden Testcase

```
// hidden.c
void hidden() {
  int a, b, c, x, y, *p1, *p2, **pp;
  p1 = &x;
  p2 = &y;
  pp = &p1; // pp -> p1 -> x
  *p1 = a + y; // x = a + y;
  **pp = *p2 + b; // x = y + b;
  y = a + b;
}
```