**Homework:**
(You may spend ~20 hours for this homework)

0. Please prepare a document to explain your assumptions/design-decisions for the following questions, if any.

1. (80 pts) Write down `kcov-branch-identify` using Clang based on the provided template C++ file. `kcov-branch-identify` receives a file name of a single C file and prints the information on the conditional statements at source code level as they are and the total number of branches of the C file.

   See the following output for the attached `example-kcov.c`:

```
$ ./kcov-branch-identify example-kcov.c
function: f2
   If   ID: 0   Line: 4    Col: 2     Filename: ./example-kcov.h
function: f1
   If   ID: 1   Line: 19   Col: 2     Filename: example-kcov.c
function: main
   If   ID: 2   Line: 30    Col: 2     Filename: example-kcov.c
   If   ID: 3   Line: 32    Col: 9     Filename: example-kcov.c
   For  ID: 4   Line: 40    Col: 2     Filename: example-kcov.c
   While ID: 5  Line: 45    Col: 2     Filename: example-kcov.c
   Do   ID: 6   Line: 50    Col: 2     Filename: example-kcov.c
   Case ID: 7   Line: 52    Col: 4     Filename: example-kcov.c
   Case ID: 8   Line: 55    Col: 4     Filename: example-kcov.c
   ?:   ID: 9   Line: 56    Col: 9     Filename: example-kcov.c
   Default ID: 10  Line: 59   Col: 4   Filename: example-kcov.c
   If  ID: 11    Line: 64    Col: 2    Filename: example-kcov.c
   ?:  ID: 12    Line: 64    Col: 7    Filename: example-kcov.c
  ImpDef. ID: 13  Line: 68   Col: 2     Filename: example-kcov.c
   Case    ID: 14  Line: 69   Col: 3   Filename: example-kcov.c
   Case    ID: 15  Line: 72   Col: 3   Filename: example-kcov.c
   Do     ID: 16  Line: 77    Col: 2   Filename: example-kcov.c
   If     ID: 17  Line: 77    Col: 2   Filename: example-kcov.c
Total number of branches: 30
```

   Note 1. We count each `case` as one branch (i.e., considering `switch(){…}` has multiple outgoing edges). Also, we count (implicit) `default` statement as one branch. A line and a column of an implicit `default` is those of corresponding `switch()`.

2. (10 pts) Print out the branches in the attached `grep` source code file (i.e., `grep.c`) by using your `kcov-branch-identify`. Submit the output of your `kcov-branch-identify` on `grep.c`.

   Note 1. If your program fails to find header files of a target program (grep.c), you have to modify `include_paths` (line 137) in the initialization part of the kcov-branch-identify.cpp template file.
   Note 2. You can ignore various Clang warnings.
   Note 3. Your program should print out functions which have no branches.
   Note 4. The total # of branches of the grep C file: > 3000

3. (10 pts) Report the branch coverage of your `kcov-branch-identify` on your own test input files including `grep.c` by using `gcov`. Please submit a test script file `test_script` w/ necessary data files to show the branch coverage of your `kcov-branch-identify`.

4. (80 pts) Write down `kcov` using Clang. You have to submit your `kcov` code.
   A. `kcov` receives a file name of a *preprocessed* single C file `<f>.i` (which is generated from non-preprocessed C file `<f>.c` using the below command) and generates the instrumented version `<f>-cov.c` to measure branch coverage of `<f>.c` through testing.
      i. A preprocessed C file can be obtained by `gcc -E <filename>.c -o <filename>.i`
      ii. If you give a complex C file like `grep.c` **w/o preprocessing** to `kcov` as an input, `kcov` may crash due to the high complexity of handling source code locations.

   B. When `<f>-cov.c` is compiled and executed 1st time, `<f>-cov.c` generates a coverage measurement file `coverage.dat`. After then, `<f>-cov.c` updates `coverage.dat` through testing `<f>-cov.c`. The format of `coverage.dat` is as follows

```
Line# |# of execution   |# of execution   | conditional
      |of then branch |of else branch | expression
1453       0        0            errnum
1474       0        7            size && !result
1484       3        0            ptr
1488       0        0            size && !result
…
6950       0        0            (end = memchr(beg + len, '\n', (buf + size) -
(beg + len))) != 0
6955       0        0            beg > buf && beg[-1] != '\n'
Covered: 581 / Total: 3101 = 18.735892%
```

Note1. If one line has multiple branches (i.e., nested `if` statements), you can print out these branches in separate lines with the same line id
Note2. The # of execution of else branch of `case` should be always 0 (i.e., meaningless)
Note3. For a `switch` statement, your program should print out `case` and (implicit) `default` statements. A conditional expression of `case` statement is a corresponding case value and that of `default` is ``default''

5. (10 pts) Print out the coverage measurement file of the *preprocessed* grep C code with the following test cases (execution commands) where `grep.c` is the grep source code file used for your HW (not preprocessed C file)

```
./grep -n "if" grep.c
./grep -E "[0-9][0-9]+" grep.c
./grep -E "[[:digit:]][[:alpha:]]" grep.c
```

6. (10 pts) Report the branch coverage of your `kcov` on your own test input files including `grep.c` by using `gcov`. Please submit a test script file `test_script` w/ necessary data files to show the branch coverage of your `kcov`.

Appendix. Partial solution code as a hint for kcov (the code below is just for your reference; you do not have to use the code below in your HW if you have your own good idea)

For example, for a conditional statement below,

```
if  (   x> 10                          )  {...} else {...}
```

your `kcov` may insert a probe as follows (where `34` is a unique ID for the `if` statement):

```
if  ((( x> 10) ?
      (mycov_onTrueCondition (34),1):
      (mycov_onFalseCondition(34),0))) {...} else {...}
```

Note. "`,`" operator in C/C++ evaluates its left operand and provides an evaluated value of its right operand.

```
class MyASTVisitor : public RecursiveASTVisitor<MyASTVisitor> {…
bool VisitStmt(Stmt *s) {
  …
  if (isa<IfStmt>(s)) {
              isBranchWithCondition = true;
              type = "If";

              IfStmt *ifStmt = cast<IfStmt>(s);
              condition = ifStmt->getCond();

              conditionEnd = getEndOfConditionParenthesis(condition);
  } else if (isa<WhileStmt>(s)) {
  …

  if (isBranchWithCondition && conditionEnd.isValid()) {
    stringstream probePostfix;

    probePostfix
    << ")?(mycov_onTrueCondition(" << m_branchID << "),1):(mycov_onFalseCondition("
    << m_branchID << "),0))";

    conditionStart = (condition->getBeginLoc());
    m_rewriter.InsertTextAfter(conditionStart, "((");
    m_rewriter.InsertTextAfter(conditionEnd, probePostfix.str());

    m_conditionList[m_branchID] = getConditionString(condition);
    m_branchCount[m_branchID] = 2;
    m_typeList[m_branchID] = type;
    m_lineList[m_branchID] = lineNum;
    m_branchID++;
  }
  return true;
}}

class MyASTConsumer : public ASTConsumer {…
void OnFirstTopLevelDecl(Decl *d) {
  …
  m_rewriter.InsertTextBefore(d->getBeginLoc(),
    "String to represent multiple lines of code that declare data structure (e.g., m_branchID) and define
     function (e.g., mycov_onTrueCondition) to record branch coverage information at runtime"     );
}

void OnLastTopLevelDecl() {
    …
    m_rewriter.InsertTextAfter(m_visitor.getEndOfStmt(m_lastDecl->getEndLoc(),
    clang::tok::semi).getLocWithOffset(1), lastProbe.str());
}}
```