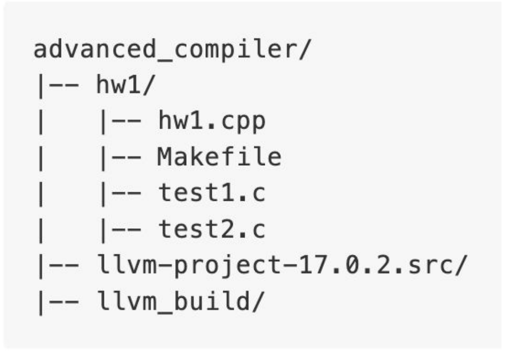
How to run program

1. Directory as Fig. 1 (as the course have provided)
2. enter hw1, perform `make run` for test1, `make run2` for test2
   a. I’ve added a new rule called run2



```
advanced_compiler/
|-- hw1/
|   |-- hw1.cpp
|   |-- Makefile
|   |-- test1.c
|   |-- test2.c
|-- llvm-project-17.0.2.src/
|-- llvm_build/
```

Cases that can handle

1. can only one dimension loop (like ai+b, i is the induction variable)
2. the accessed index should be affine (cannot appear case like A[i*i])
3. case like A[3*(i+2)] is permitted (multiply the constant index)
4. Induction variable should be named “i”
    a. Just being too lazy to parse the Induction variable name.

Experiment report

1. Parse through the IR (.ll file)
2. Some Datas that we need
    a. Loop Bounds:
        i. upperBound at llvm::loop::getHeader(),
        ii. lowerBound at llvm::loop::getPreHeader()
    b. Induction Variable: “i”
    c. All the Read access and the write access
        i. if induction variable is loaded, then start calculating the index
        ii. llvm::gep has the array name (A, B, C, D)
3. use R/W access to find flow/anti dependence, use W/W access to find the output dependence.
    a. Solve the diophantine by solver
    b. Should also consider the order of statements and dependence direction

Bonus

- Mixin pattern
  - Everyone implements each pass and they might have same functions commonly lile getName(), in the past there was a pass manager that inherited a pass manager interface.
  - The virtual property may bring up some overhead (in C++, vtable, vpointer …) in execution, so new design pattern CRTP is applied.
  - Without the function override, we only need to implement the function “run()” then our pass will run under FunctionPassManager under ModulePassManager
  - As there is no containing IR construct for a Module, a manager for passes over modules forms the base case which runs its managed passes in sequence over the single module provided.

ref: PassManager.h, C++ 的靜態多型 : CRTP, WritingAnLLVMNewPMPass.rst (docs)

Bonus

- Utilize ADT
- I use containers from stl, which is also recommended ing llvm docs
    - for saving read/write access I’ve use the std::vector<std::pair<std::string, std::vector<int>>>
    - std::pair<std::string, std::vector<int>> is an access of Read or write
    - .first stores the name of accessed array name
    - .second store the coefficients of i at index 0 and index 1
    - .second store the info of which statement is it at index 2