UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,
Informatiche e Matematiche

10. LAB 3: Loops e Dominator Trees

Compilatori – Middle end [I215-014]

Corso di Laurea in INFORMATICA
(D.M.270/04) [16-215]
Anno accademico 2024/2025

Prof. Andrea Marongiu
andrea.marongiu@unimore.it

Copyright note

Credits

- Gibbons, Carnegie Mellon University, “Optimizing Compilers”
- Pekhimenko, University of Toronto, “Compiler Optimization”

La classe LLVM LoopInfo

- `#include “llvm/Analysis/LoopInfo.h”`
- È la classe LLVM che individua i natural loops in un CFG
- Si può recuperare un oggetto `LoopInfo` a partire da un FunctionPass:

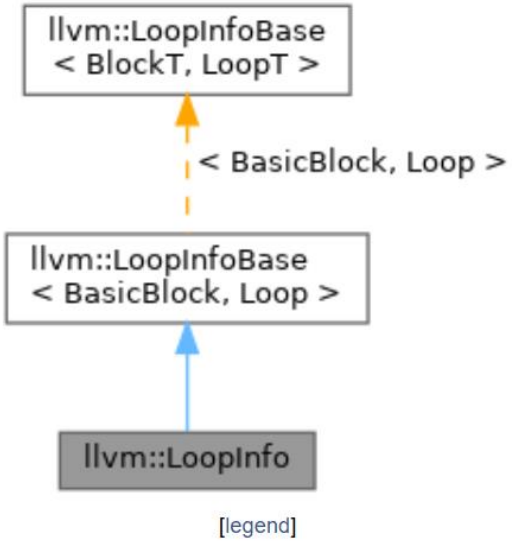
```
PreservedAnalyses run(Function &F,
FunctionAnalysisManager &AM) {

  LoopInfo &LI = AM.getResult<LoopAnalysis>(F);
  ...
}
```

La classe LLVM LoopInfo

- La classe `LoopInfo` contiene (o eredita da LoopInfoBase) vari metodi utili
  - Studiate la documentazione
  - https://llvm.org/doxygen/classllvm_1_1LoopInfo.html

- In particolare proviamo a capire:
  - Come determinare se il CFG non contiene loop
  - Come capire se un basic block del CFG è l'header di un loop
  - Come recuperare l'handle al loop che contiene un dato basic block



La classe LLVM LoopInfo

- Proviamo inoltre a giocare con gli iteratori della classe, che ci permettono di scorrere i loop stessi

```
for (LoopInfo::iterator L = LI.begin(); L != LI.end(); ++L)
{
    Loop *LL = *L; // 'LL' è l'handle al loop corrente
    ...
}
```

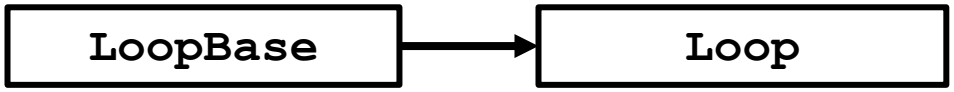
- O, in forma più compatta:

```
for (auto &L: LI)  // 'L' è l'handle al loop
```

I *loop* in LLVM

- Una volta recuperato un handle all'oggetto `Loop &L` possiamo usarne i metodi per analizzare il loop

- La classe `Loop` eredita dalla classe `LoopBase`, che contiene diversi metodi utili per analizzare il loop
  - Studiare la documentazione
  - https://llvm.org/doxygen/classllvm_1_1Loop.html



GERARCHIA DELLE CLASSI LLVM

I *loop* in LLVM

- In particolare cerchiamo di capire come:
- Verificare che il loop sia in forma normale
  - https://llvm.org/docs/LoopTerminology.html
- Recuperare blocchi significativi del loop
  - Preheader
  - Header
  - Tutti i blocchi
- Scorrere i basic blocks che compongono un loop

```
for (Loop::block_iterator BI = L->block_begin(); \
                          BI != L->block_end();
                          ++BI)
```

Esercitazione 1 (Loops)

- Scrivere un FunctionPass chiamato "LoopPass", abilitabile in opt con flag "-loop-pass", che:
  1. Verifichi se il CFG corrente contiene loop. Se no, ritorni immediatamente
  2. Scorra tutti i basic block (BB) del CFG, e per ciascuno di essi verifichi se è l'header di un loop. In tal caso stampi il BB.
  3. Scorra tutti i loop del CFG e per ciascuno di essi:
     a) Verifichi se è in forma normale
     b) Recuperi l'header del loop, e da lì recuperi l'handle alla funzione che lo contiene. Usando l'handle alla funzione così ottenuto (e NON sfruttando l'handle alla funzione passato dal Pass Manager) stampi il CFG
     c) Stampi tutti i blocchi che compongono il loop

Esercitazione 1 (Loops)

- Per testare il vostro passo potete inizialmente partire dal file Loop.c della precedente esercitazione

- Ricordate che per generare un intermedio LLVM non ottimizzato e privo di load/store occorre svolgere i seguenti passi:
- `clang -S -O0 -emit-llvm -Xclang -disable-O0-optnone Loop.c -o Loop.ll`
- `opt –p mem2reg Loop.ll -So Loop.m2r.ll`

Esercitazione 1 (Loops)

- Successivamente scrivete i vostri file di test, in particolare provando a inserire multipli loop, in sequenza e innestati.

La classe LLVM DominatorTree

- `#include “llvm/IR/Dominators.h”`
- È la classe LLVM che individua il dominator tree di un CFG
- Si può recuperare un oggetto `DominatorTree` a partire da un FunctionPass:

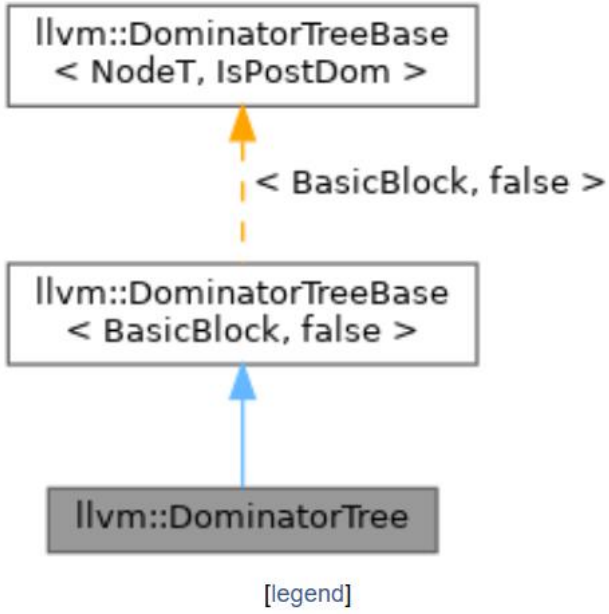
```
PreservedAnalyses run(Function &F,
FunctionAnalysisManager &AM) {

  DominatorTree &DT =
   AM.getResult<DominatorTreeAnalysis>(F);
  ...
}
```

La classe LLVM DominatorTree

- La classe `DominatorTree` contiene (o eredita da DominatorTreeBase) vari metodi utili
  - Studiate la documentazione
  - https://llvm.org/doxygen/classllvm_1_1DominatorTree.html

- In particolare proviamo a capire:
  - Come scorrere i blocchi di un dominator tree (nodo radice, discendenti)
  - Come stabilire le relazioni di dominanza tra basic block e/o istruzioni e/o usi



La classe LLVM DominatorTree

- NOTA: in LLVM è possibile iterare seguendo algoritmi depth-dirst o breadth-first

```
for (auto *DTN : depth_first(DT.getRootNode())) {
for (auto *DTN : breadth_first(DT.getRootNode())) {
```

- Per quest'ultimo vi occorre includere il file

```
#include "llvm/ADT/BreadthFirstIterator.h"
```

Esercitazione 2 (Dominator Tree)

- Modificate il vostro analysis pass perché stampi a schermo il dominance tree di una funzione (qualsiasi, da voi scritta) avente il seguente CFG