CS620T: Compilers - Principles and Implementation

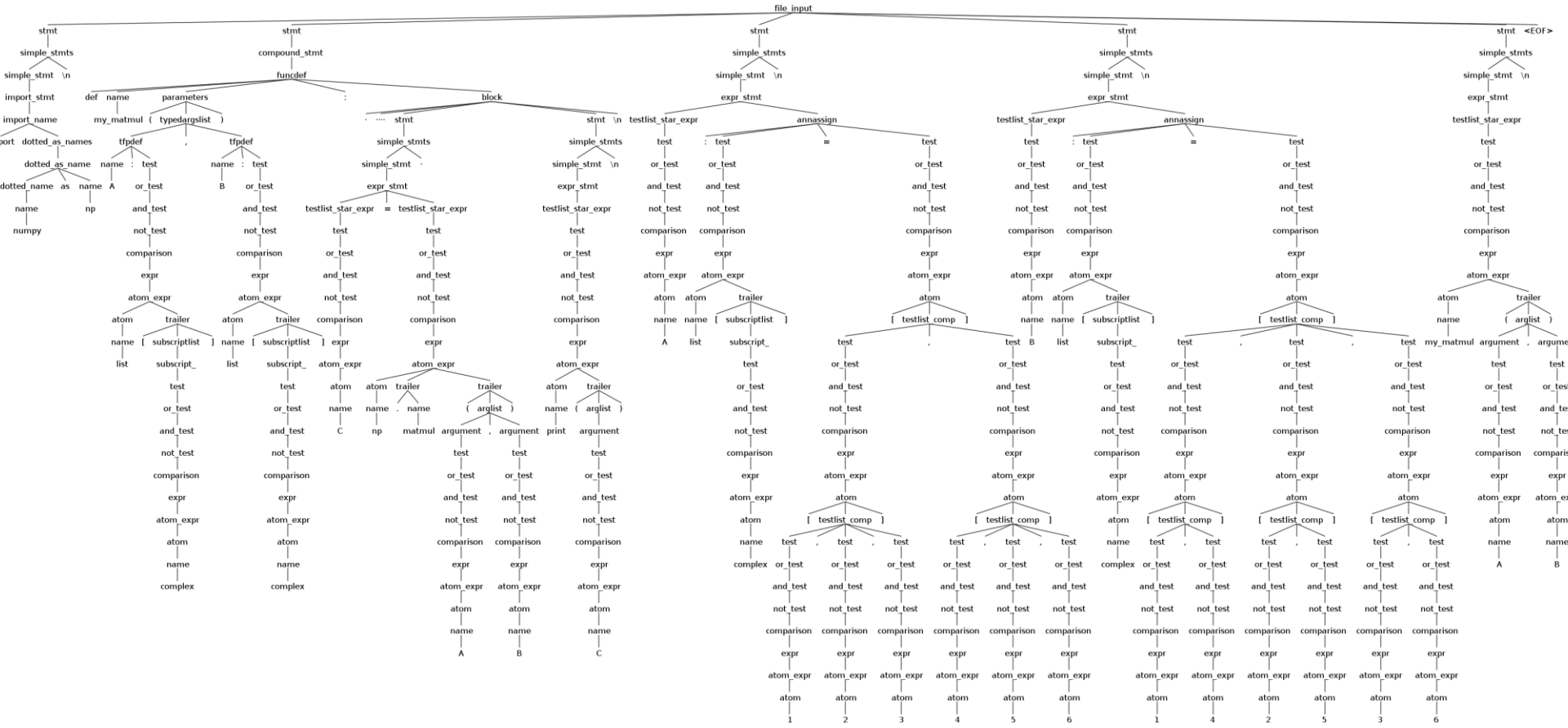
Autumn 2026

Week 5: Semantic processing (contd.) : AST construction, Symbol Table construction, and Intermediate Representation

Viewing ParseTree for a Python program using ANTLR

Refer to the README in the github repository

Parse tree for matmul_cmplx.py



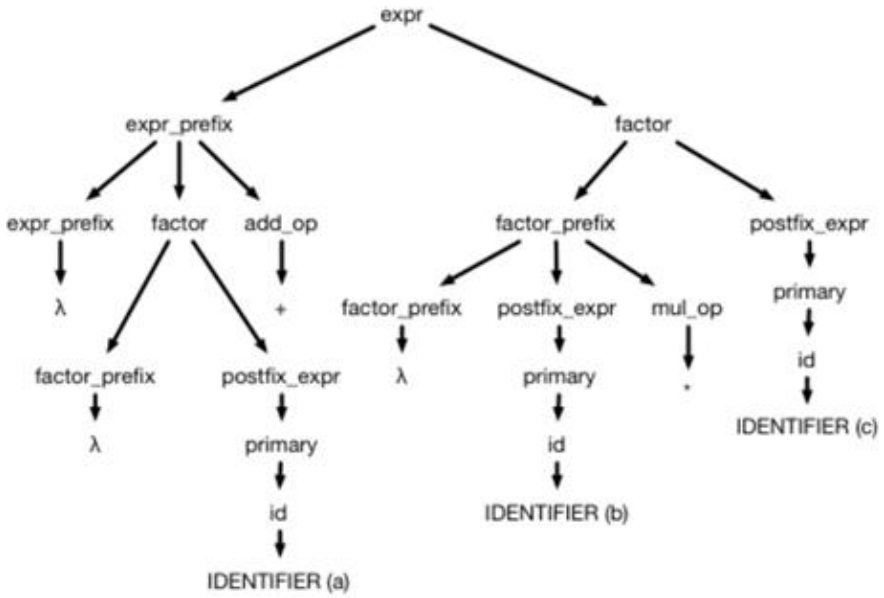
Compare this with the AST that you printed on the terminal (or visualized in the last class)

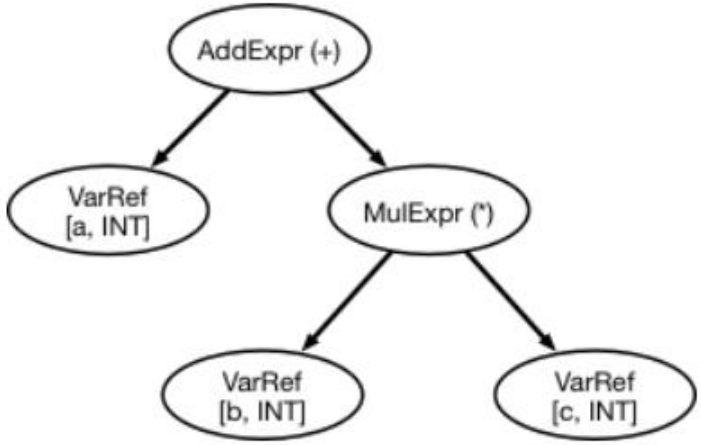
Clang AST

Clang

- C language family *frontend* for LLVM
- One of the responsibilities of the frontend is:
  - Reading a program text and producing an AST (abstract syntax tree), which is a representation of the program.

Parse Tree vs. AST (a*b+c)





- Clang AST – enables you to develop tools
  - for analysing program text

Clang Usage

- As Clang Plugin
  - E.g. `clang -fpass-plugin=mypass.so test.c`

  (a silly use case: replacing + operator in `test.c` with *)
- LibTooling
  - A c++ program (with main function) written to do some analysis of some other program (C, C++)
  - E.g. `bin/myanalyzer test.c`
- LibClang
  - E.g., if We want to write a python program that analyses C++ source code

Clang AST

- Is rich (has source code location info at every node)
- Has types resolved
  - A design choice of compiler construction is that type checking can be done separately from AST construction
- Is huge
  - ~10K .cpp files
  - > 100k lines of code
- Has several classes, objects of which form the AST nodes.
  - The AST nodes are optimized for size.

Clang AST Classes

- `ASTContext` – one of the first classes that we encounter

  AST nodes point to e.g.:

  Identifier Table - storing identifiers

  Source Manager – managing source code locations

  - organizes information around the AST
  - provides entry point to the AST with the help of `TranslationUnitDecl* getTranslationUnitDecl()`

Clang AST Core Classes

- `Decl`
  - Base class for many other classes e.g. `VarDecl`

  `int x=100;` //int x corresponds to a VarDecl node

  exercise: what type of node represents 100? `IntegerLiteral`

  Other examples: `CXXRecordDecl`
- `Stmt`
  - `CompoundStmt`

    No common base class

    `ReturnStmt -> e.g., return x;`
  - `BinaryOperator ->` e.g., `i > 0, x + 2`
    - Is an expression (`Expr`) and expressions are `Stmts`
- `Type`
  - `PointerType`
  - `ParenType`

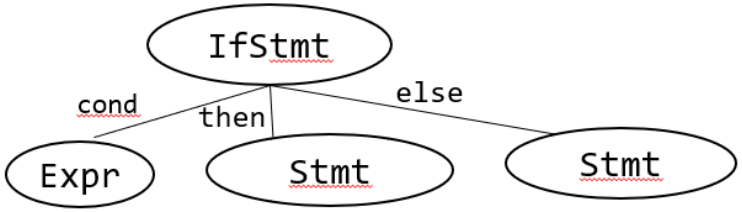
Clang AST Glue Classes

- `DeclContext`
  - Inherited by `Decl`s that contain other `Decl`s
- `TemplateArgument`
  - To represent template arguments
- `NestedNameSpecifier`
- `QualType`
  - For representing type qualifiers e.g. `const, unsigned` etc.

Clang AST Glue Methods

Help in traversing AST

- `IfStmt : getThen(), getElse(), getCond()`



Traverse across parts of AST

- `CXXRecordDecl: getDescribedClassTemplate() method`

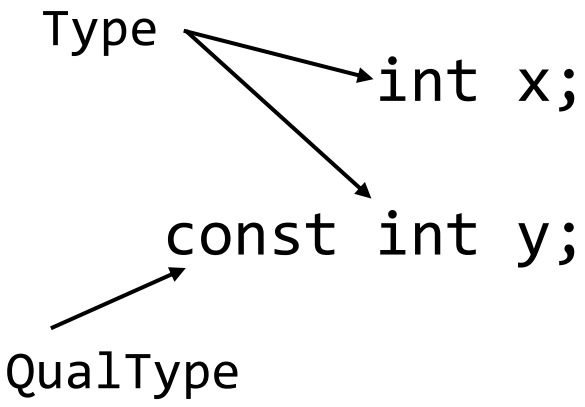
- `Type: getAsCXXRecordDecl() method`

For getting the declaration of a struct/union/class

Detour: declaration vs. definition of function definitions, variable definitions, class/struc definitions in C/C++

Types

- Represent types in AST

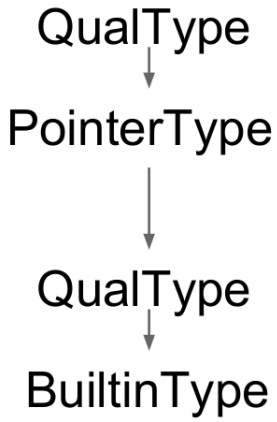



```
int * const * y;
```

Exercise: what is the type of y?

Types can be nested. Qualifier is sandwiched.

```
Constant pointer to a int *
```




```
class PointerType{
   QualType getPointeeType() const;
}
```

Navigating Source

- `SourceLocation` – tells the location of a token

AST Traversal

- Multiple abstractions available:
  - `RecursiveASTVisitor`
  - `ASTMatcher`

  - `Next: RecursiveASTVisitor`

ASTFrontEndAction

```
class FindNamedClassAction : public clang::ASTFrontendAction {
public:
  virtual std::unique_ptr<clang::ASTConsumer> CreateASTConsumer(
    clang::CompilerInstance &Compiler, llvm::StringRef InFile) {
    return
std::make_unique<FindNamedClassConsumer>(&Compiler.getASTContext());
  }
};
```

- Inherit from `ASTFrontendAction` and override `CreateASTConsumer`
  - `ASTFrontendAction` is an interface that allows user-specific actions to happen during compilation.
- `CreateASTConsumer`
  - Consumes the AST produced by the Clang parser

ASTConsumer

```
class FindNamedClassConsumer : public clang::ASTConsumer {
public:
  explicit FindNamedClassConsumer(ASTContext *Context)
    : visitor(Context) {}

  virtual void HandleTranslationUnit(clang::ASTContext &Context) {
    visitor.TraverseDecl(Context.getTranslationUnitDecl());
  }
private:
  FindNamedClassVisitor visitor;
};
```

- Override (as many) methods to take user-specific action on visiting AST nodes.
- `HandleTranslationUnit` called after entire source code is parsed (not while it is being parsed)
  - `ASTContext` class represents AST for the source file.
  - `Visitor.TraverseDecl(Context.getTranslationUnitDecl())` begins visiting nodes of the tree

RecursiveASTVisitor

```
class FindNamedClassVisitor
  : public RecursiveASTVisitor<FindNamedClassVisitor> {
public:
  bool VisitCXXRecordDecl(CXXRecordDecl *Declaration) {
    //for illustration only. Dump shows which nodes already visited.
    Declaration->dump();

    // The return value indicates whether we want the traversal to proceed.
    // Return false to stop the traversal of the AST.
    return true;
  }
};
```

- Don’t call Visit functions directly
- Implement `VisitStmt, VisitDecl, VisitPantry etc.` as per your needs

Pattern Matching and Source Location

```
if (Declaration->getQualifiedNameAsString() == "n::m::C") {
      FullSourceLoc FullLocation = Context->getFullLoc(Declaration-
>getBeginLoc());
      if (FullLocation.isValid())
        llvm::outs() << "Found declaration at "
                     << FullLocation.getSpellingLineNumber() << ":"
                     << FullLocation.getSpellingColumnNumber() << "\n";
}
```

- Get the location manager from ASTContext

Symbol Table

- A *symbol table* maintains
  - Symbolic names
  - Attributes of a name
    - E.g. type, scope, accessibility
- Used to manage declarations of symbols and their correct usage

Symbol Table – Names

For the sample program shown below identify all names (note: this is not a valid micro program)

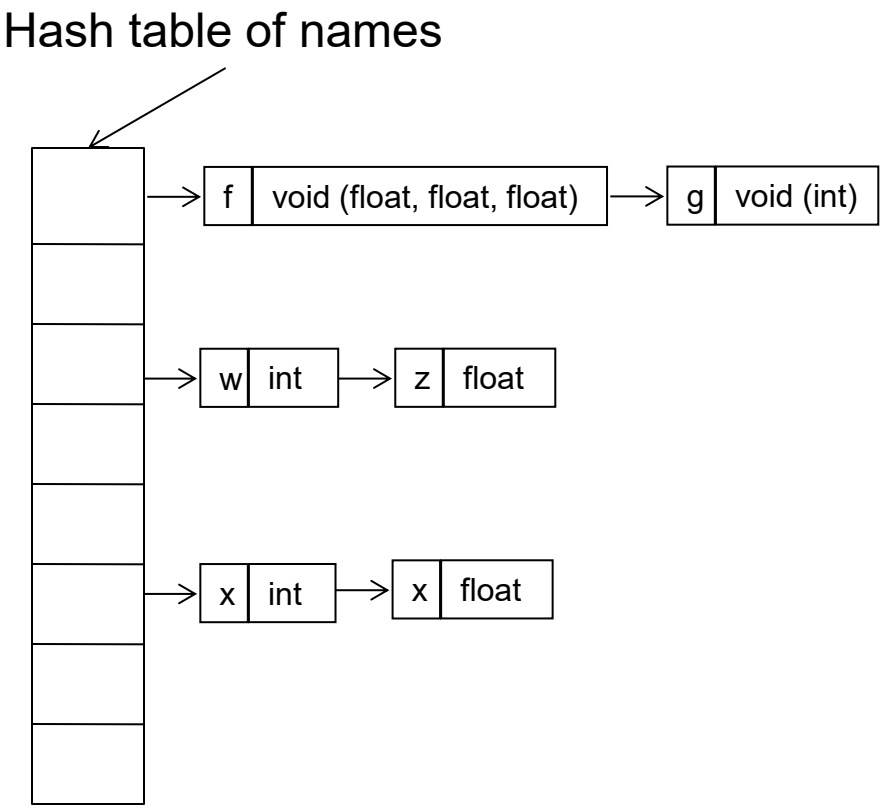
```
PROGRAM scope_test
BEGIN
#global declarations
FUNCTION void f(float, float, float)
FUNCTION void g(int)
{
    INT w, x;
    {
        FLOAT x, z;
        f(x, w, z);
    }
    g(x);
}

END
```

Symbol Table Implementation – High-level Requirements

- Should accommodate:
  - Efficient retrieval of names
  - Frequent insertion and deletion of names
- Should consider *scopes*

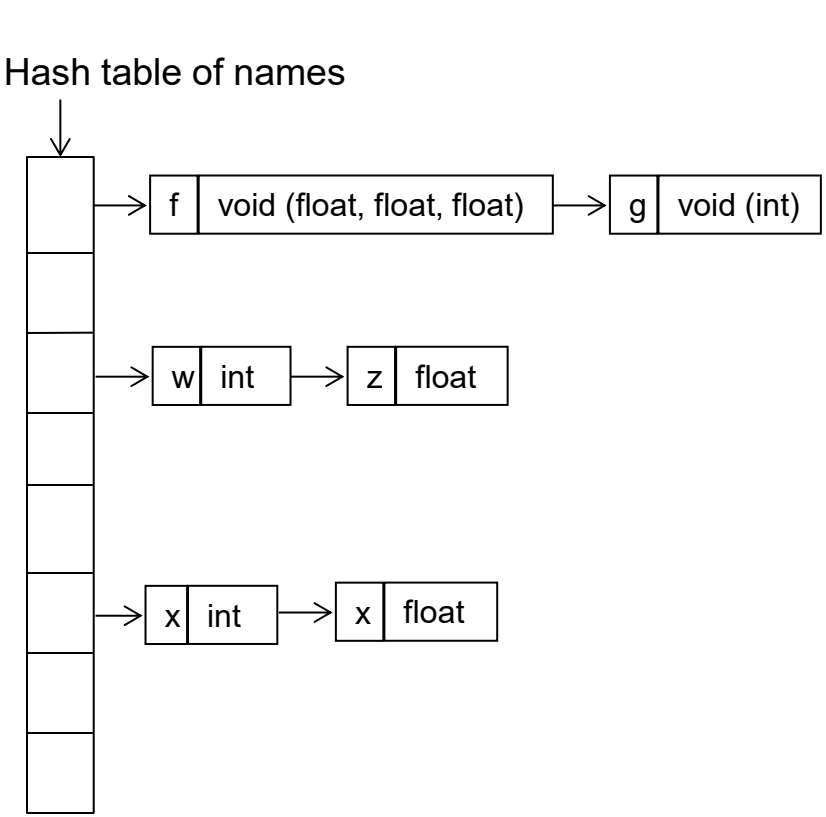
Symbol Table – an implementation

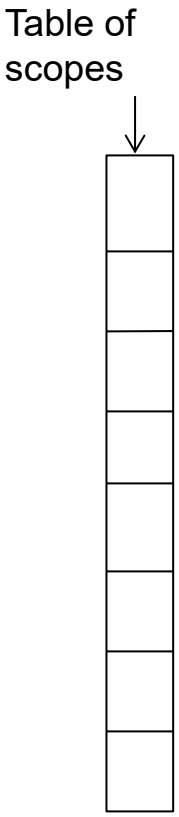



```
PROGRAM scope_test
BEGIN
#global declarations
FUNCTION void f(float, float, float)
FUNCTION void g(int)
{
      INT w, x;
      {
            FLOAT x, z;
            f(x, w, z);
      }
      g(x);
}

END
```

Symbol Table – an implementation



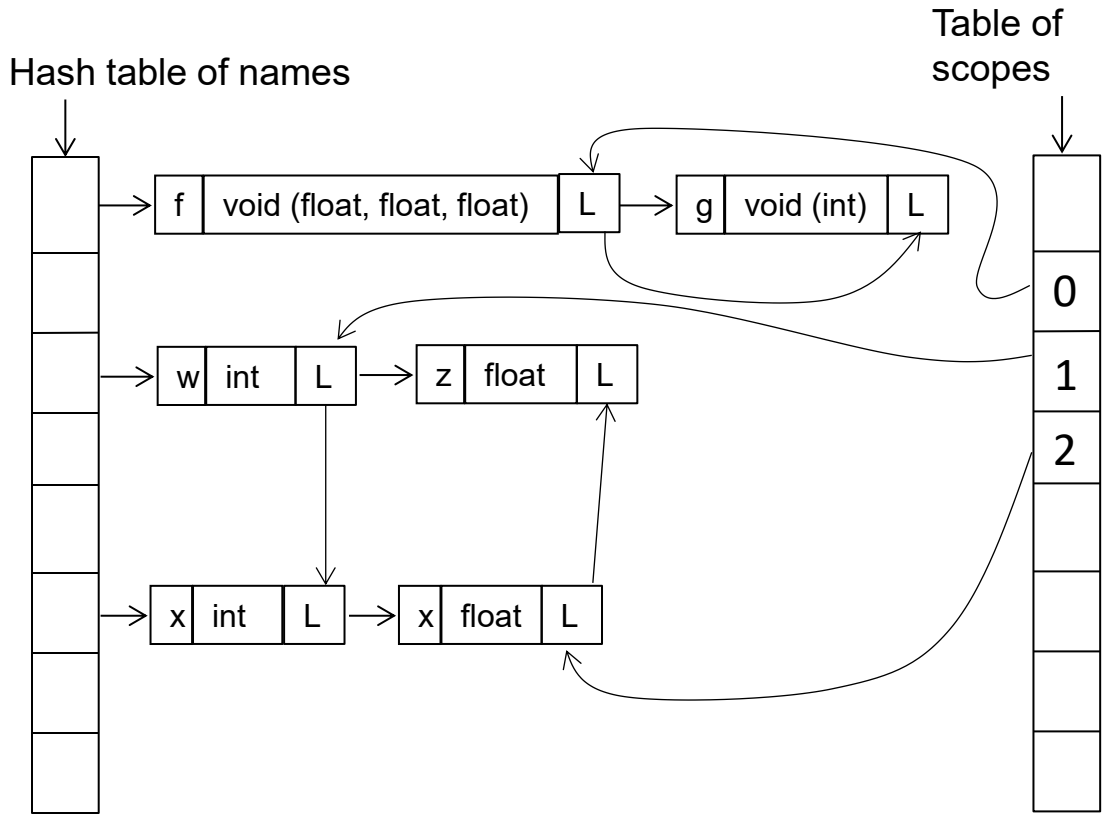



```
PROGRAM scope_test
BEGIN
#global declarations
FUNCTION void f(float, float, float)
FUNCTION void g(int)
{
      INT w, x;
      {
            FLOAT x, z;
            f(x, w, z);
      }
      g(x);
}

END
```

- be aware of current scope
- Be aware of all active scopes
- Chain names by their scope-levels

Symbol Table – an implementation

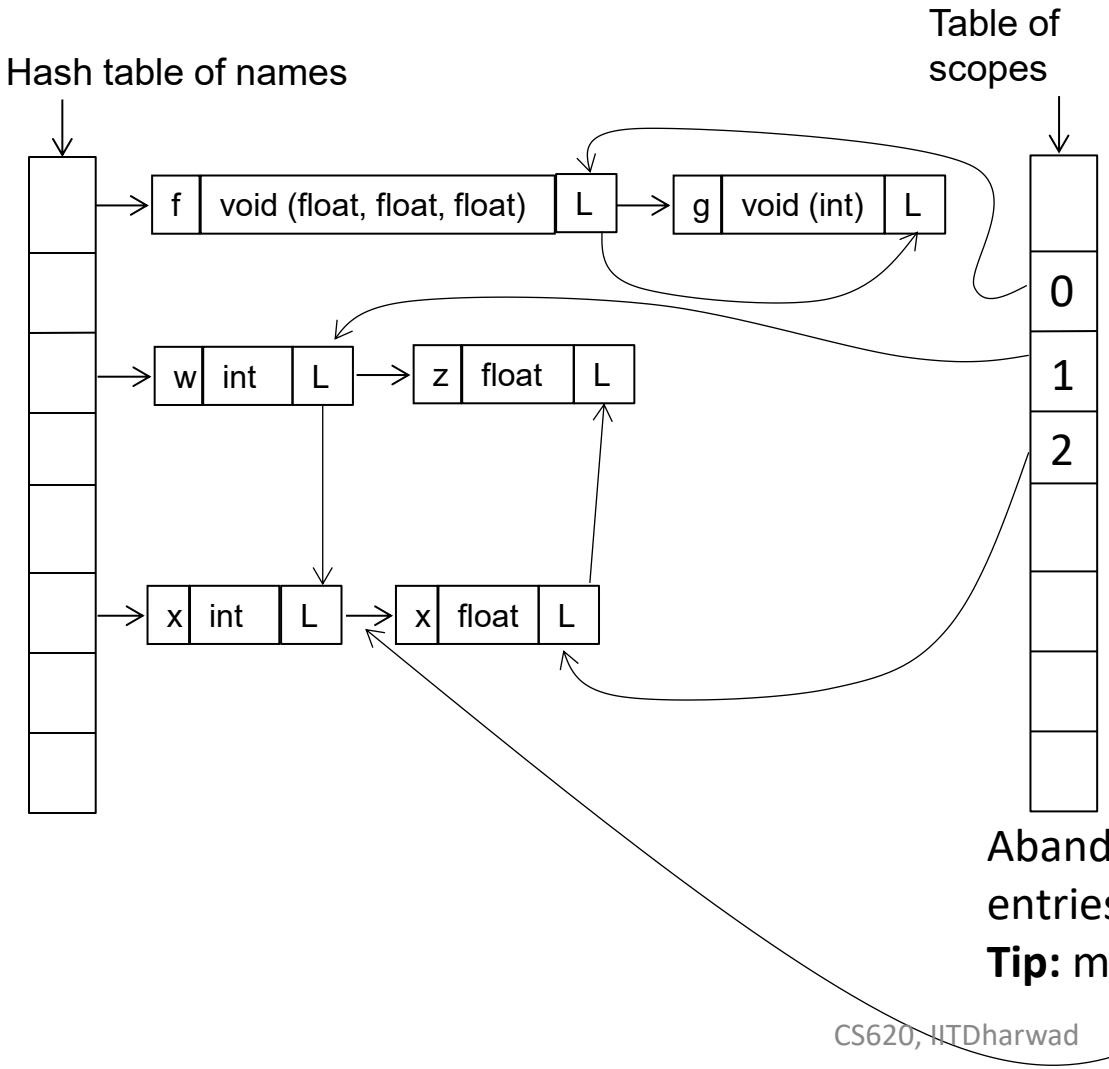



```
PROGRAM scope_test
BEGIN
#global declarations
FUNCTION void f(float, float, float)
FUNCTION void g(int)
{
      INT w, x;
      {
            FLOAT x, z;
            f(x, w, z);
      }
      g(x);
}

END
```

- Chain names by their scope-levels

Symbol Table – an implementation



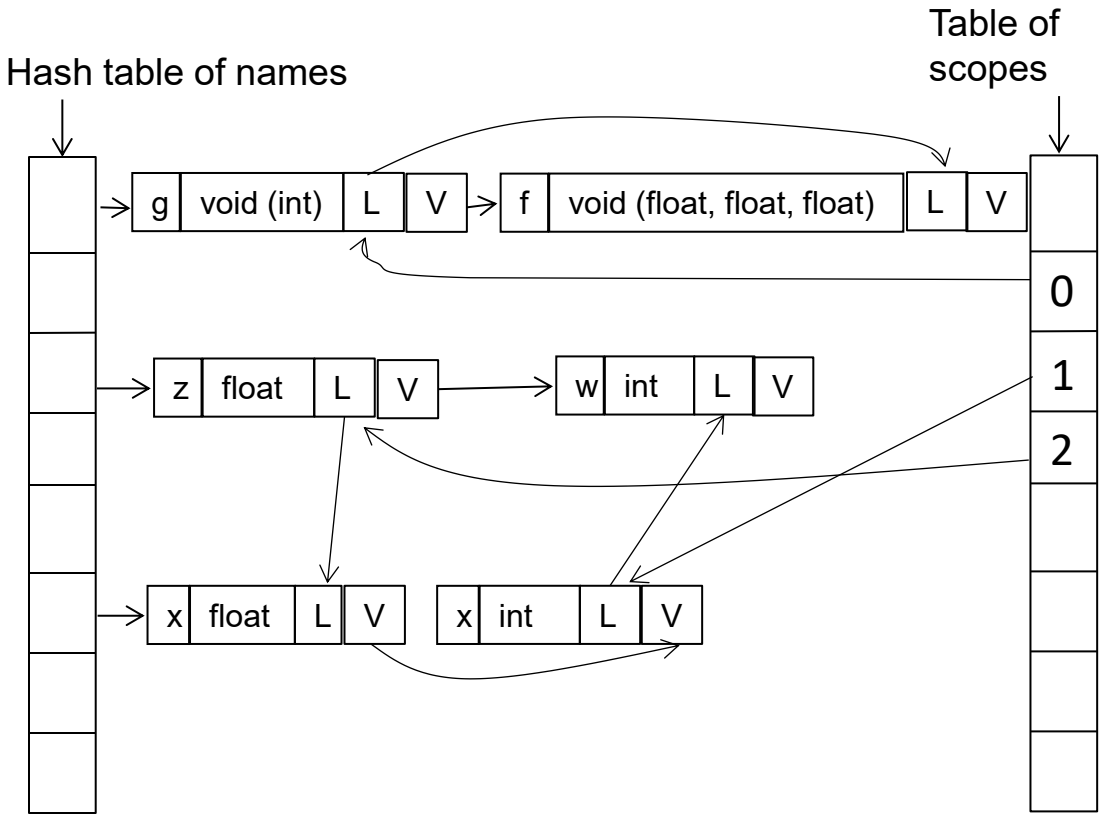

```
PROGRAM scope_test
BEGIN
#global declarations
FUNCTION void f(float, float, float)
FUNCTION void g(int)
{
      INT w, x;
      {
           FLOAT x, z;
           f(x, w, z);
      }
      g(x);
}

END
```

Abandon scope: remove references to all entries that exists at the current level.

Tip: make the hash chain a doubly linked list.

Symbol Table – an implementation




```
PROGRAM scope_test
BEGIN
#global declarations
FUNCTION void f(float, float, float)
FUNCTION void g(int)
{
      INT w, x;
      {
           FLOAT x, z;
           f(x, w, z);
      }
      g(x);
}

END
```

What if I want to access the integer x here?
Tip: maintain an ordered stack for each symbol name appearing in the current scope.

Notice the order of objects: “insert at the front of the list”

Symbol Table – an implementation