Welcome
to
Level-3 :)

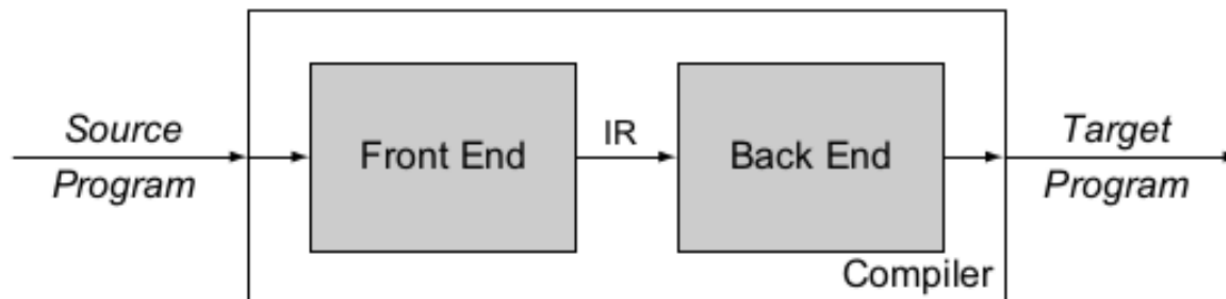# A Typical Day in Level-3

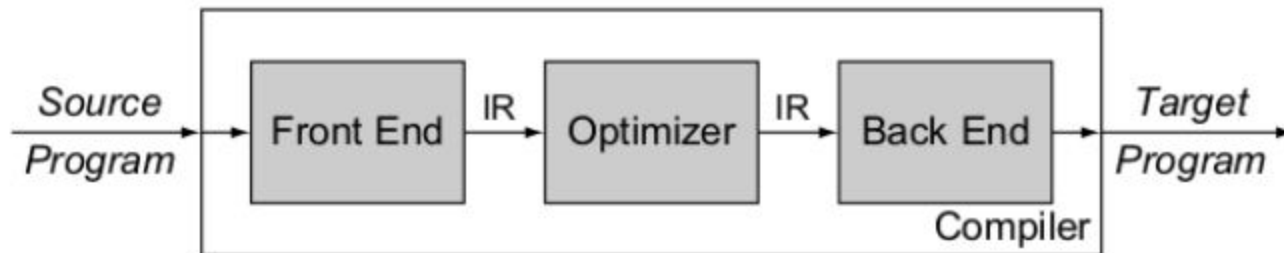# Things to Do

Things NOT to Do

Welcome to CSE 310

Compiler

- Convert a source program to a target program
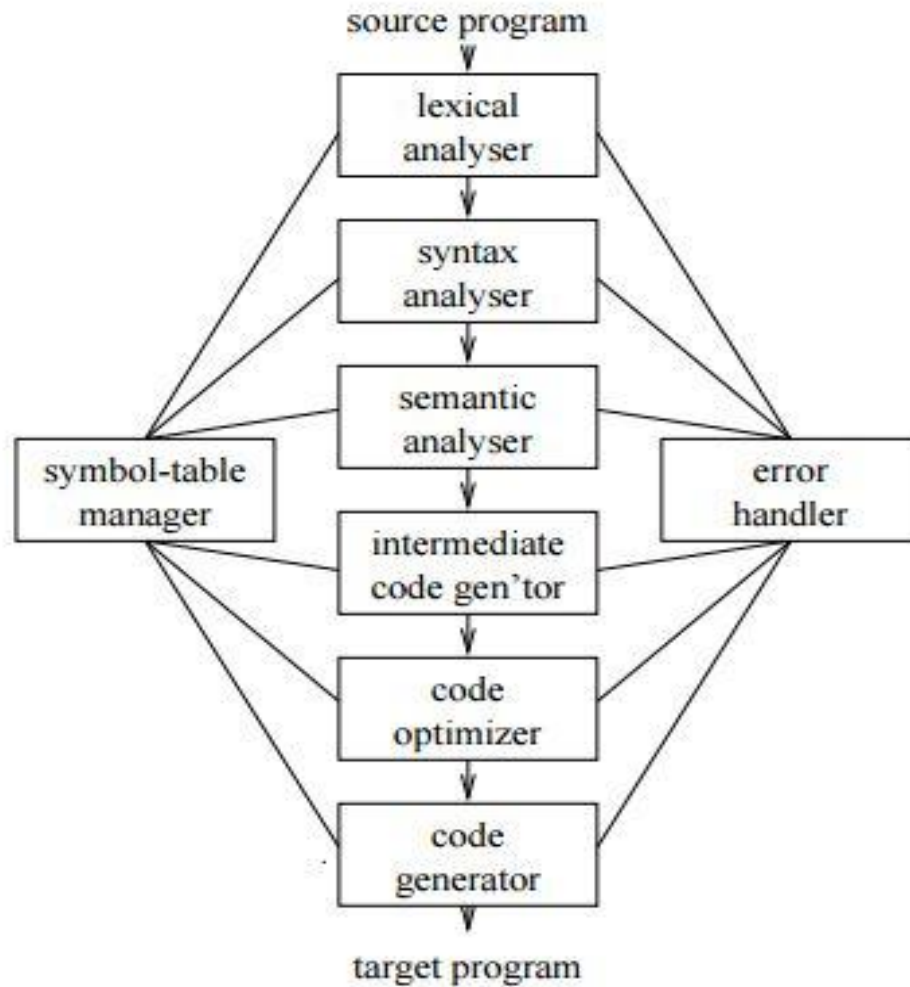- The compilation process usually divided into several phases
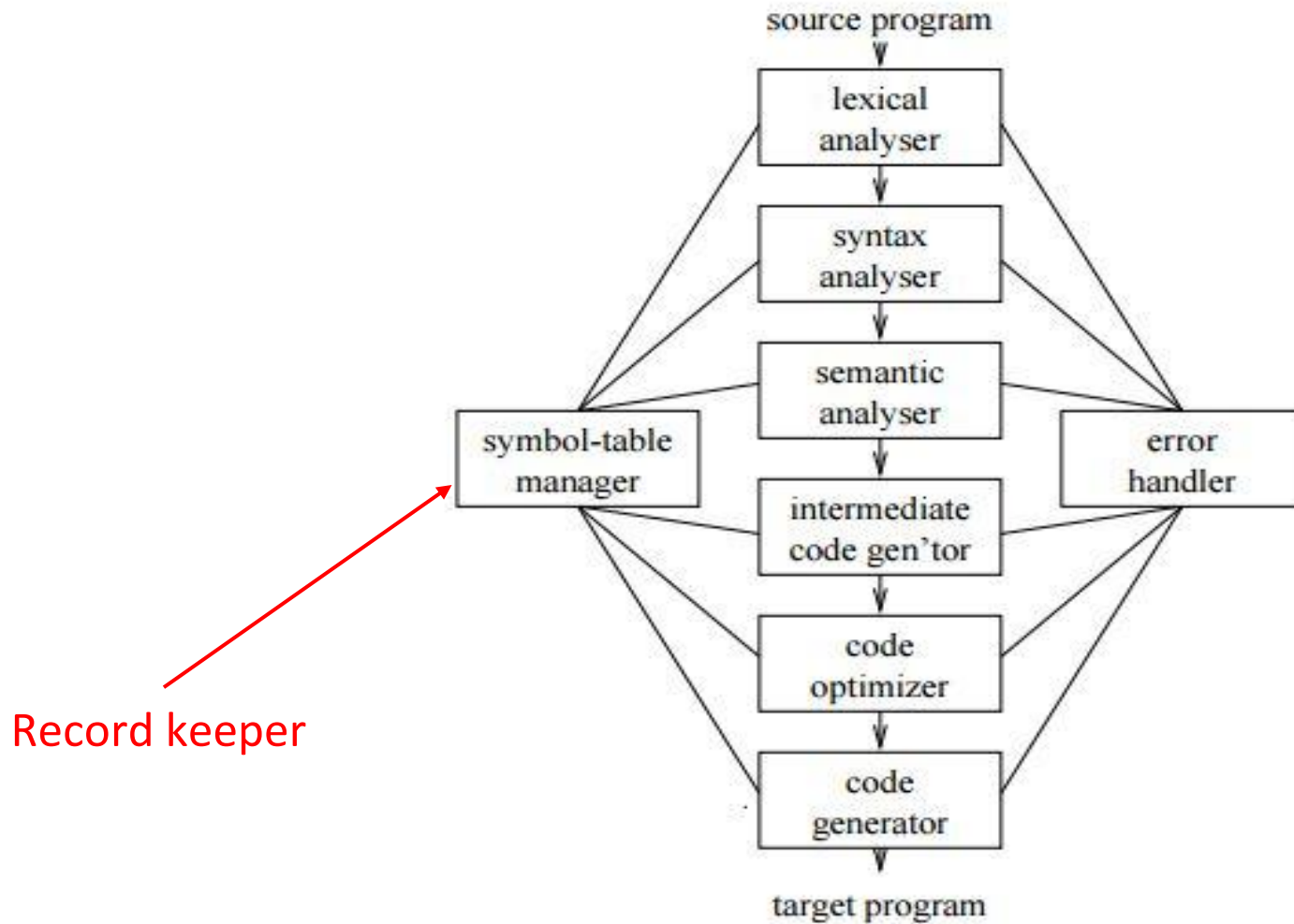
Compiler

- Convert a source program to a target program
- The compilation process usually divided into several phases

Compiler

Compiler

source program

| lexical analyser |
| syntax analyser |
| semantic analyser |
| intermediate code gen'tor |
| code optimizer |
| code generator |

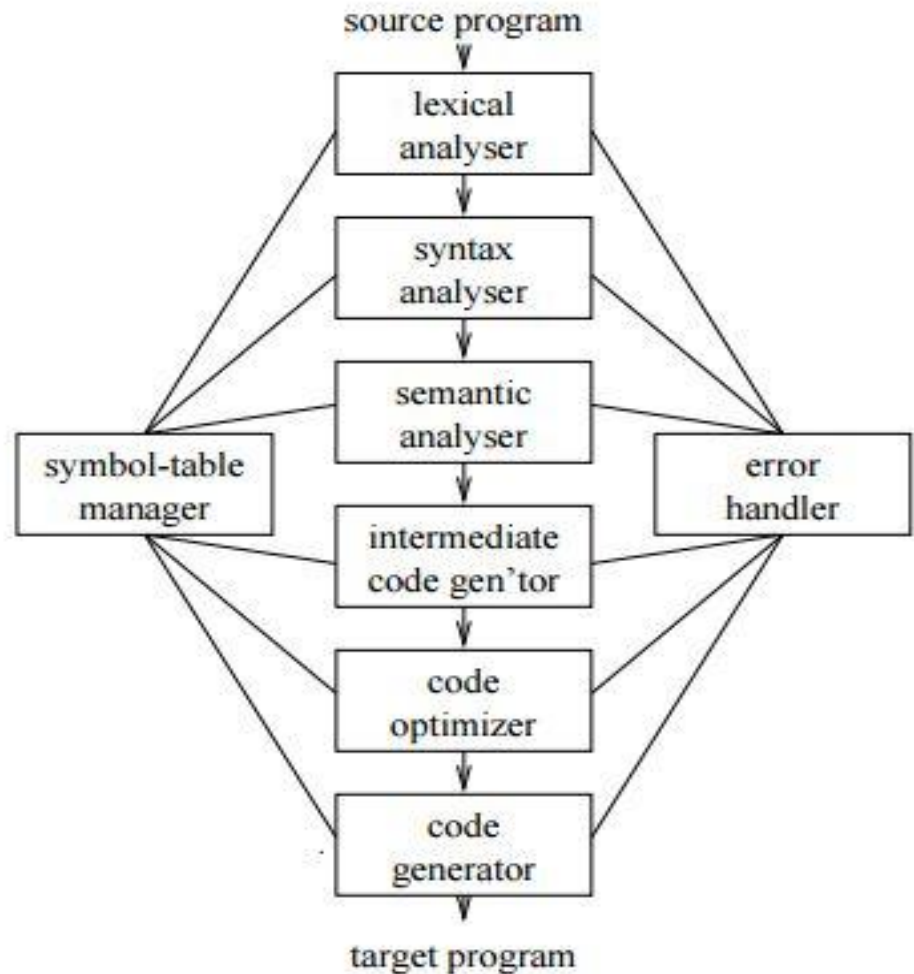symbol-table manager

error handler

target program

Record keeper

Compiler

- **Lexical Analyzer** takes the source program as input and converts it into a stream of tokens
- To be used by the syntax analyzer later on
- Also detects some lexical errors
  - Ill formed number
  - Improper variable declaration
  - Unfinished string/comment etc.

source program
↓
lexical analyser
↓
syntax analyser
↓
semantic analyser
↓
intermediate code gen'tor
↓
code optimizer
↓
code generator
↓
target program

symbol-table manager

error handler

Compiler

- **Syntax analyzer** uses the tokens produced by the lexical analyzer to depict the grammatical structure of the token stream
- Builds implicit syntax tree
- Detects syntax errors
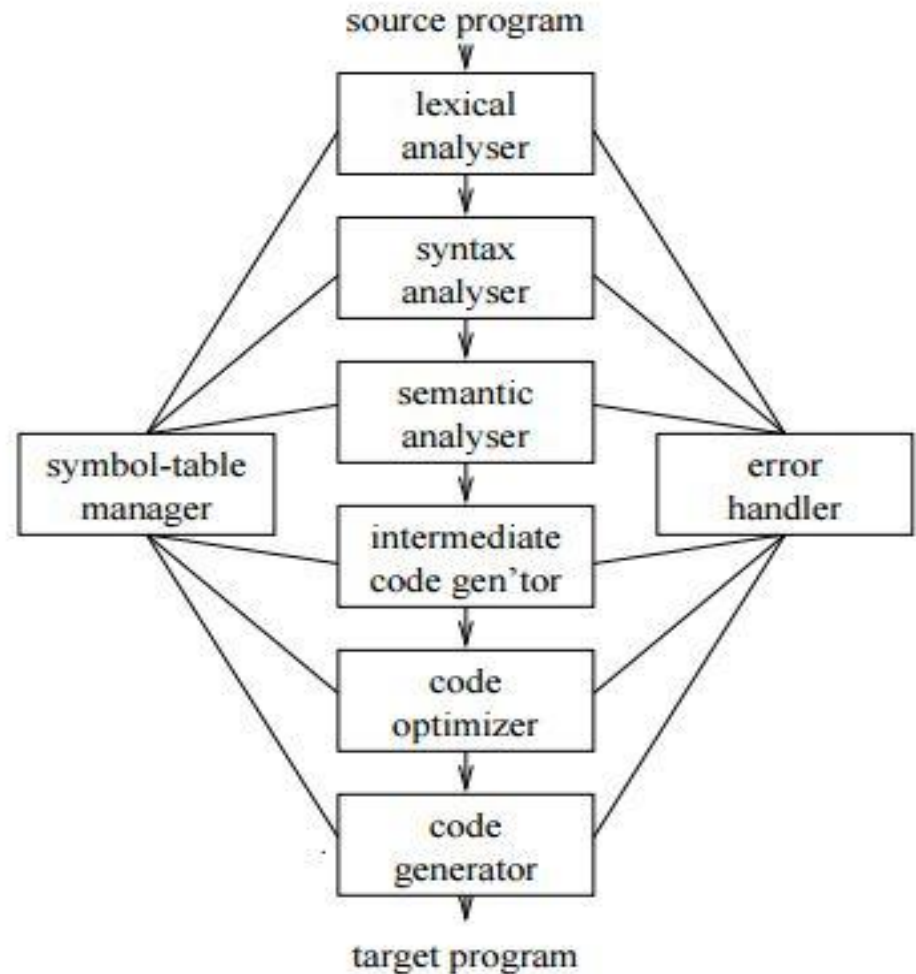
Compiler

- **Semantic analyzer** uses the syntax tree and the information in the symbol table to check the source program for semantic consistency with the language definition
- Check semantic errors
    - Type checking
    - Variable declared as void
    - Undeclared variable
    - Error in no./type of function argument during call
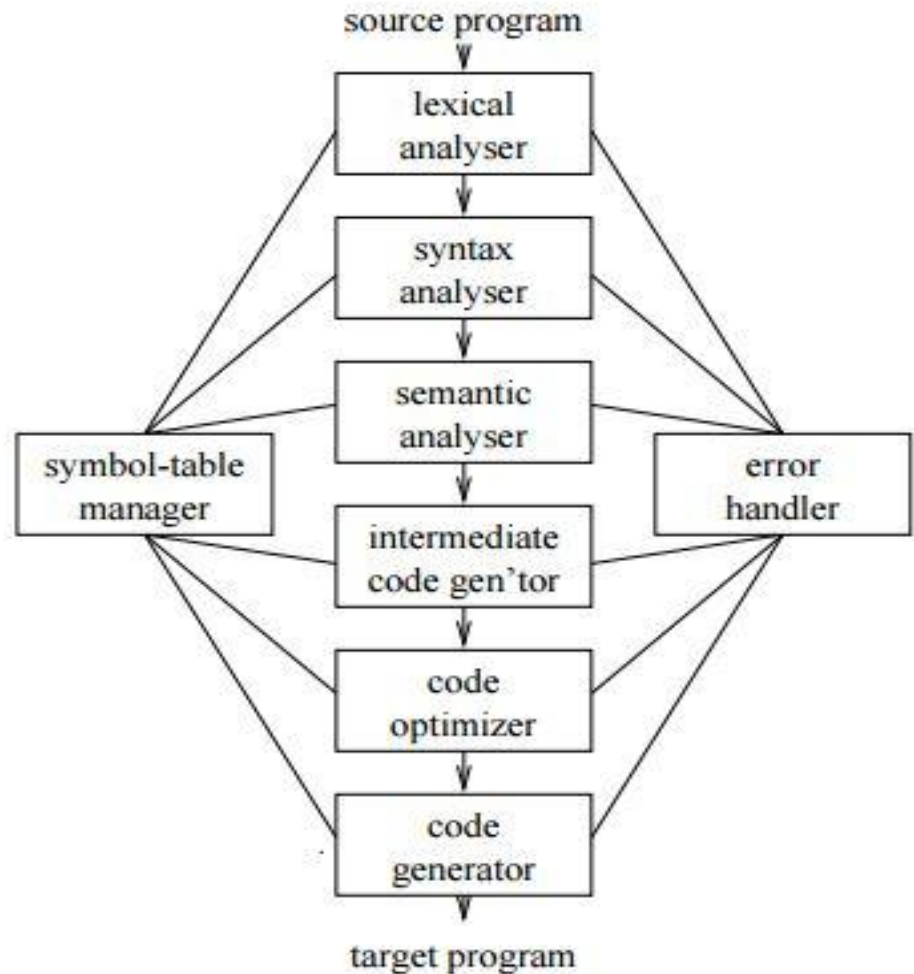
What will we do in this course?

- Construct and manage symbol table
- Perform lexical analysis using flex
- Perform syntax analysis, semantic analysis, and intermediate code generation
- Some code optimization too
- So... We are going to build a COMPILER!

Some Info

- Linux Platform

- No Plagiarism

Symbol Table

- A table storing information of occurrence of various entities in the source program
- Function names, return type, no. parameters; variable name, type etc.
- Information are:
  - Symbol Name
  - Type
  - Scope

- Used in almost all phases of a compilation

Offline 1: Symbol Table Management
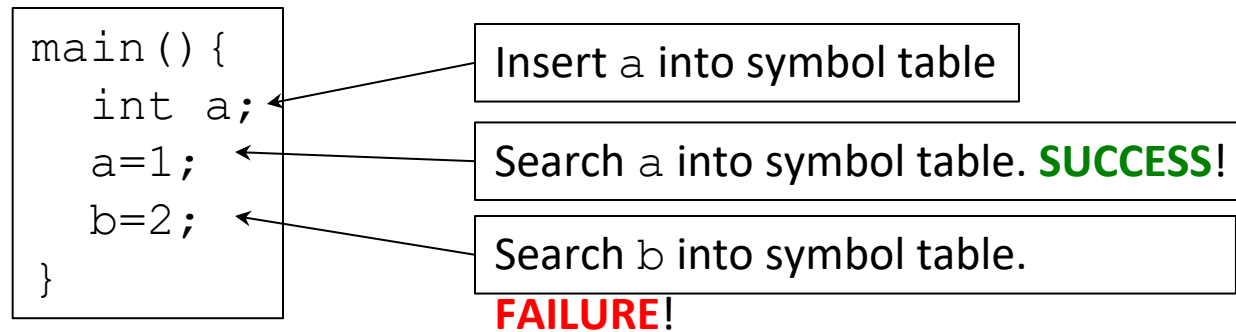
- Implement a simple symbol table

- Hash based (Chaining)

- Each entry is **mainly** a two tuple
  <Symbol Name, Symbol Type>

- Use Symbol Name as key of hash table

Offline 1: Symbol Table Management
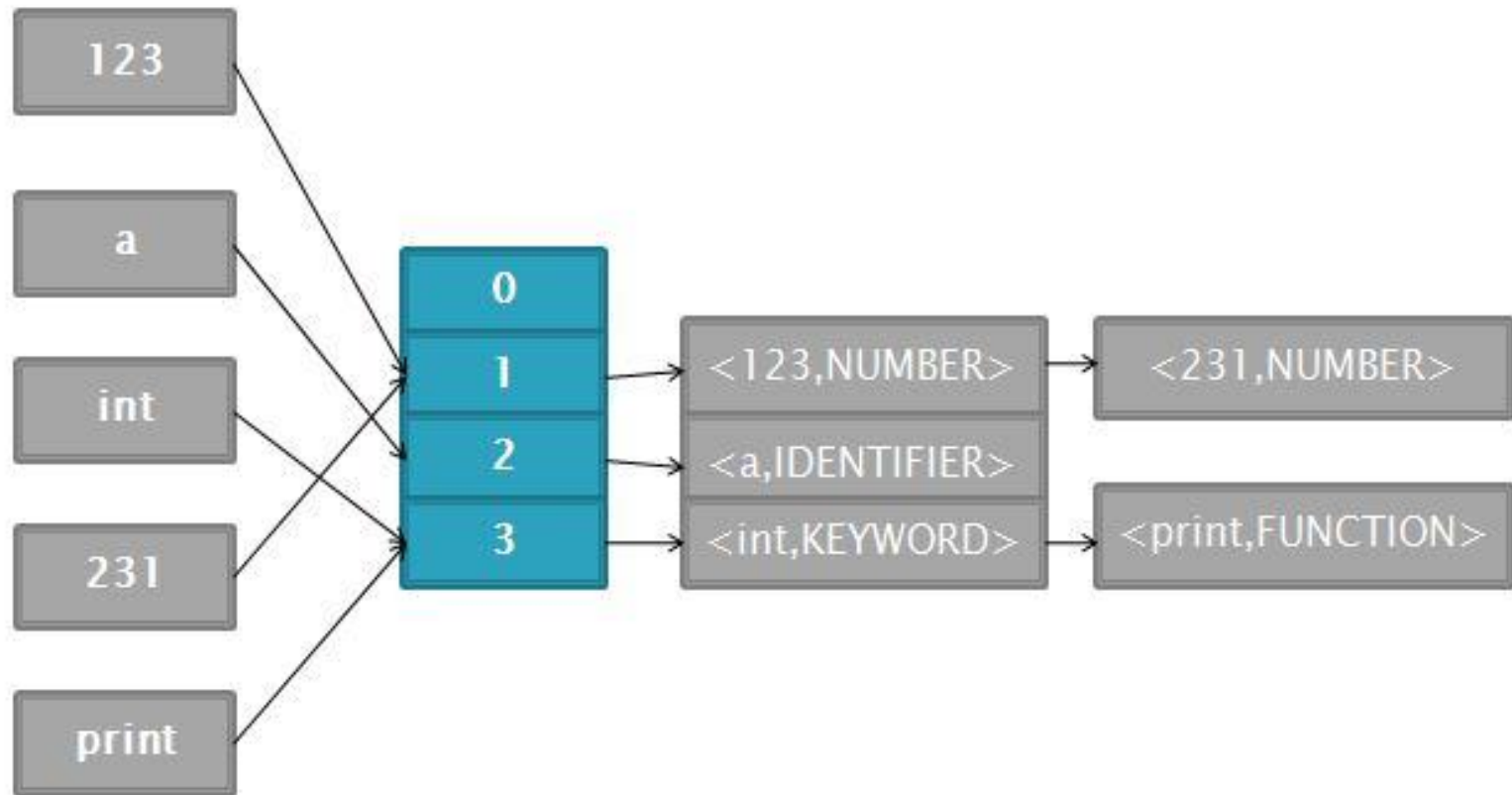
How Symbol Table Helps?

- How can this type of Symbol Table help?
  - Detect undeclared variable

```
main(){
    int a;
    a=1;
    b=2;
}
```

Insert `a` into symbol table

Search `a` into symbol table. **SUCCESS**!

Search `b` into symbol table. **FAILURE**!

  - Type checking
    - Add an extra field for each symbol named **datatype**
    - During an assignment operation, check datatype field of RHS and LHS

How Symbol Table Helps?

- ## How can this type of Symbol Table help?
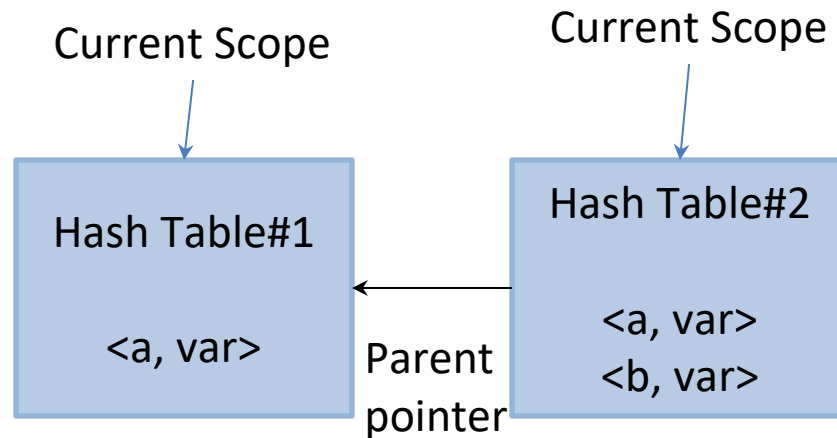 - Scope Management

```
main(){
    int a;
    {
        int a,b;
    }
    b=2;
}
```

- Need to allow duplicate entry in symbol table
- Also delete some entries when a block ends
- How to accommodate this??

Symbol Table for Scope Management

- ## List of Hash Tables

```
main(){
   int a;
   {
       int a,b;
   }
   b=2;
}
```
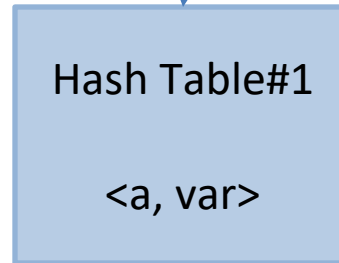
Current Scope

Current Scope

Hash Table#1

<a, var>

Parent
pointer

Hash Table#2

<a, var>
<b, var>

Symbol Table for Scope Management

- List of Hash Tables

```
main(){
    int a;
    {
        int a,b;
    }
    b=2;
}
```

Current Scope

Hash Table#1

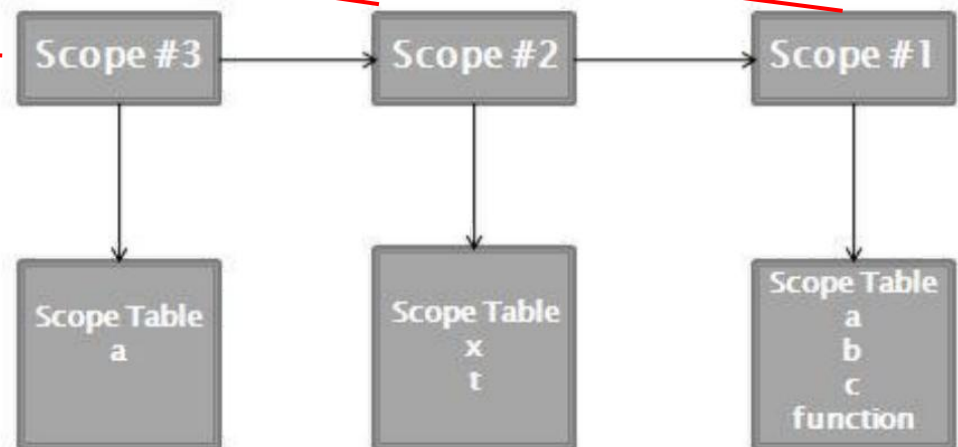<a, var>

Symbol Table for Scope Management

- **Stack of Hash Tables**
- How does accessing work?
  - Search the topmost scope table
  - If fail, search its parent scope table and so on

Symbol Table for Scope Management

- ## Stack of Hash Tables

```
1. int a, b, c;
2. int func(int x) {
3.    int t = 0;
4.    if(x == 1) {
5.        int a = 0;
6. ——→  t = 1;
7.    }
8.    return t;
9. }
10. int main() {
11.    int x = 2;
12.    func(x);
13.    return 0;
14. }
```
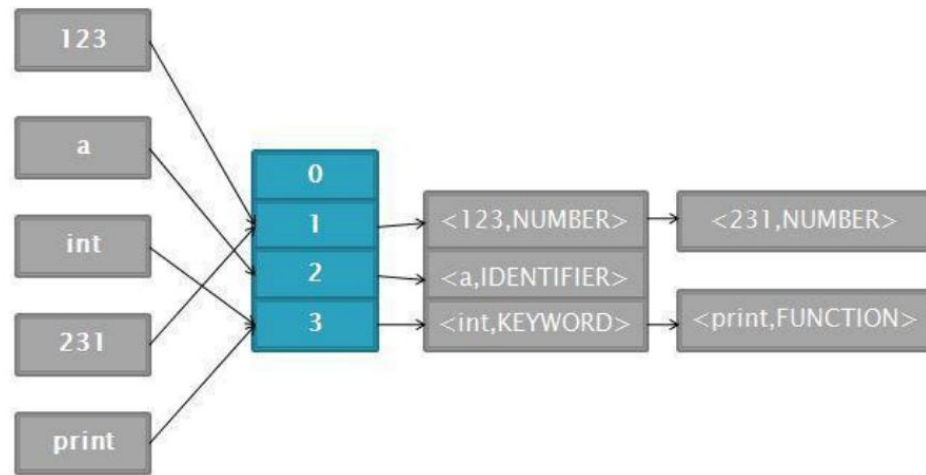
Offline 1: Symbol Table Management

- ## Three Classes
  - ### 1. SymbolInfo
    - Each entry of symbol table is an instance of SymbolInfo (Remember two tuples!!!)
    - Three member vars
      - name, type, pointer to SymbolInfo (separate chaining)

Offline 1: Symbol Table Management

- **Three Classes**

  2. **ScopeTable**

     - This class is the implementation of a hash table.

     - Represents each scope

     - Implement four operations

       » Insert

       » Lookup

       » Delete

       » Print

Offline 1: Symbol Table Management

- **Three Classes**
  3. **SymbolTable**
     - Maintains a list of Scope Tables
     - Implement six operations
       - » Enter Scope
       - » Exit Scope
       - » Insert
       - » Delete
       - » Print All Scope Tables
       - » Print Current Scope Table

No Memory Leak

- use -fsanitize=address flag

Acknowledgement

- Ajmain Yasar Ahmed Sahil