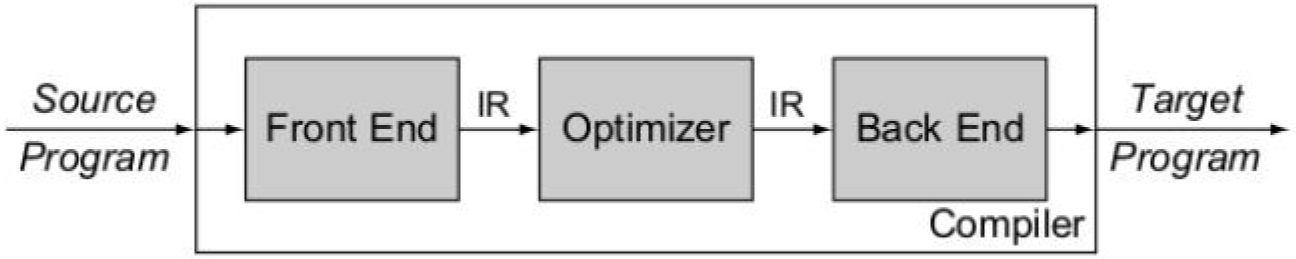
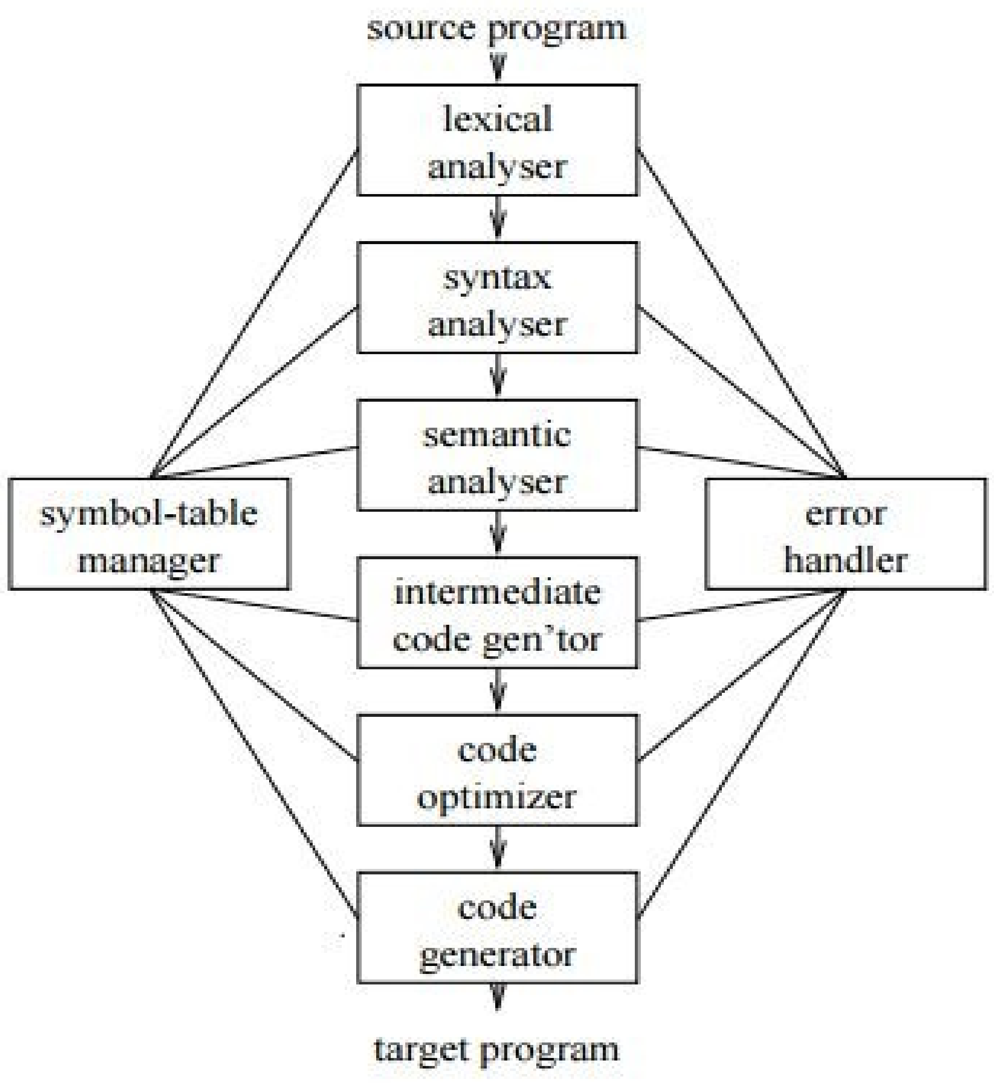
Welcome
to
CSE 310

Compiler

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

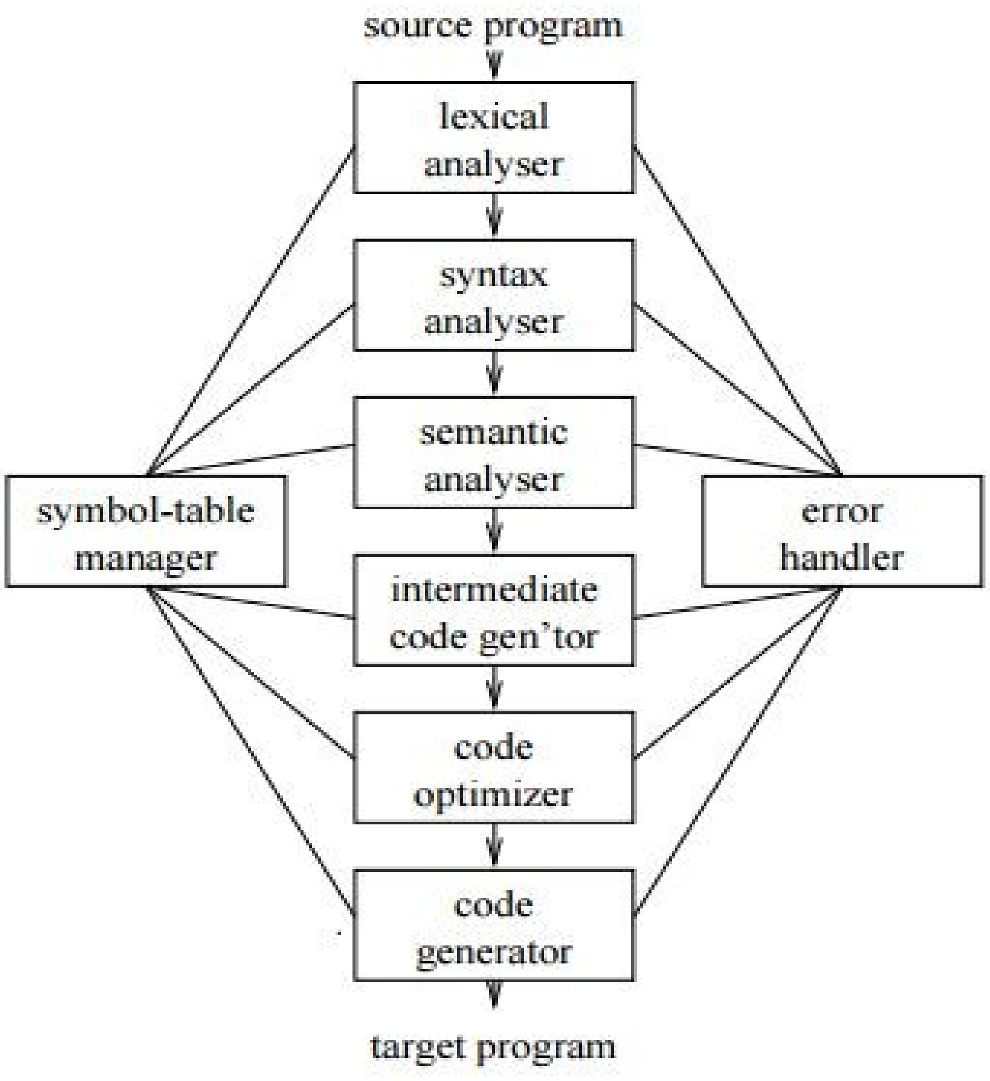


Compiler


source program
lexical analyser
syntax analyser
semantic analyser
intermediate code gen'tor
code optimizer
code generator
target program
symbol-table manager
error handler

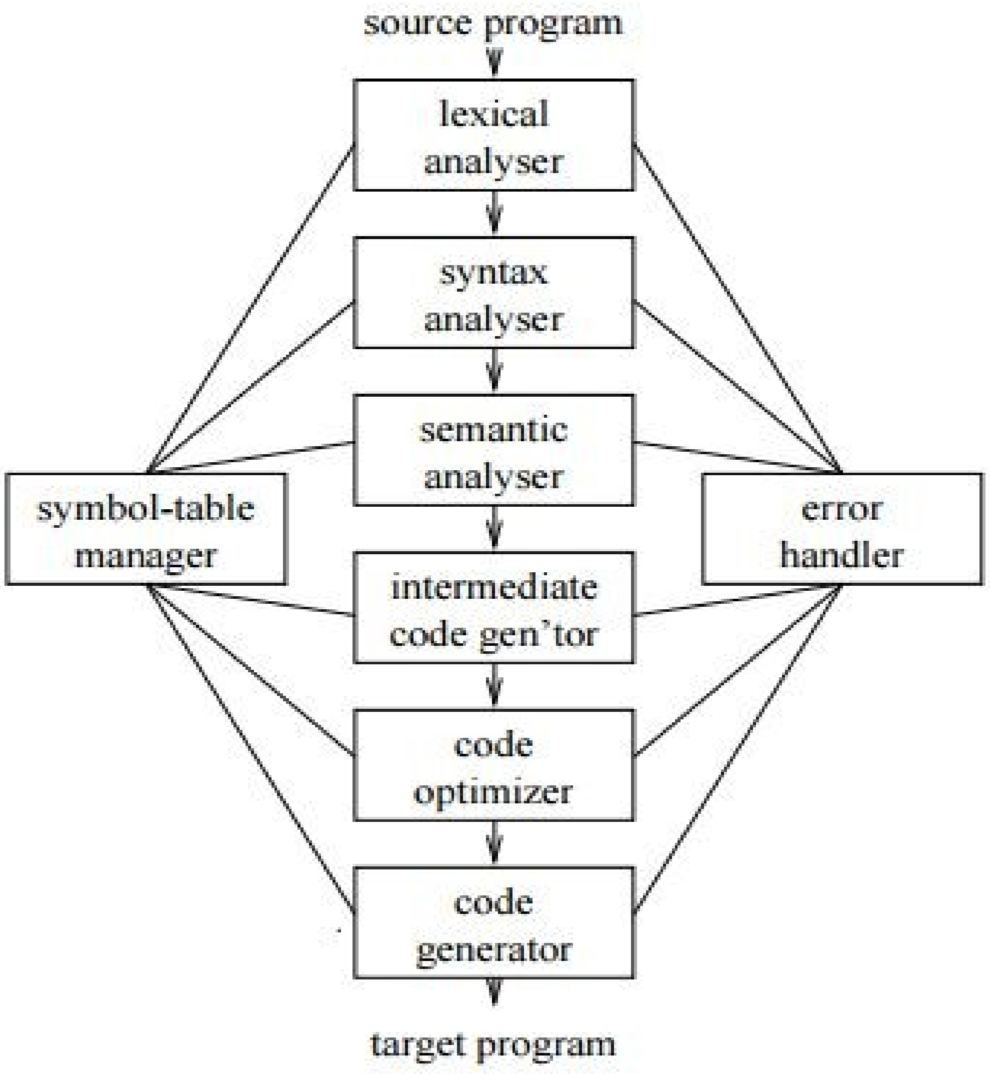
Compiler

- 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.



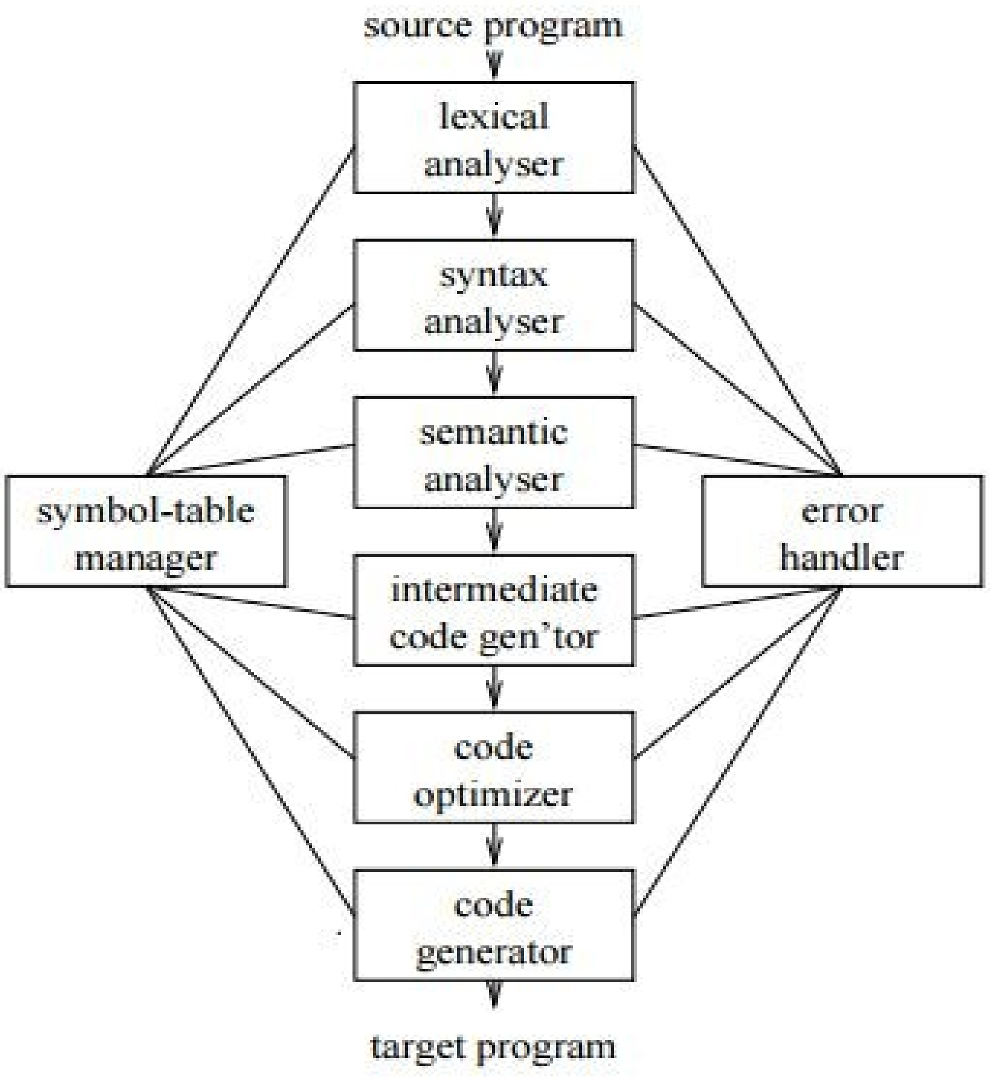
Compiler

- 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

- The 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



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 using bison
- Some code optimization too.
- So... We are going to build a COMPILER!

Some Info

- Linux platform
- No plagiarism

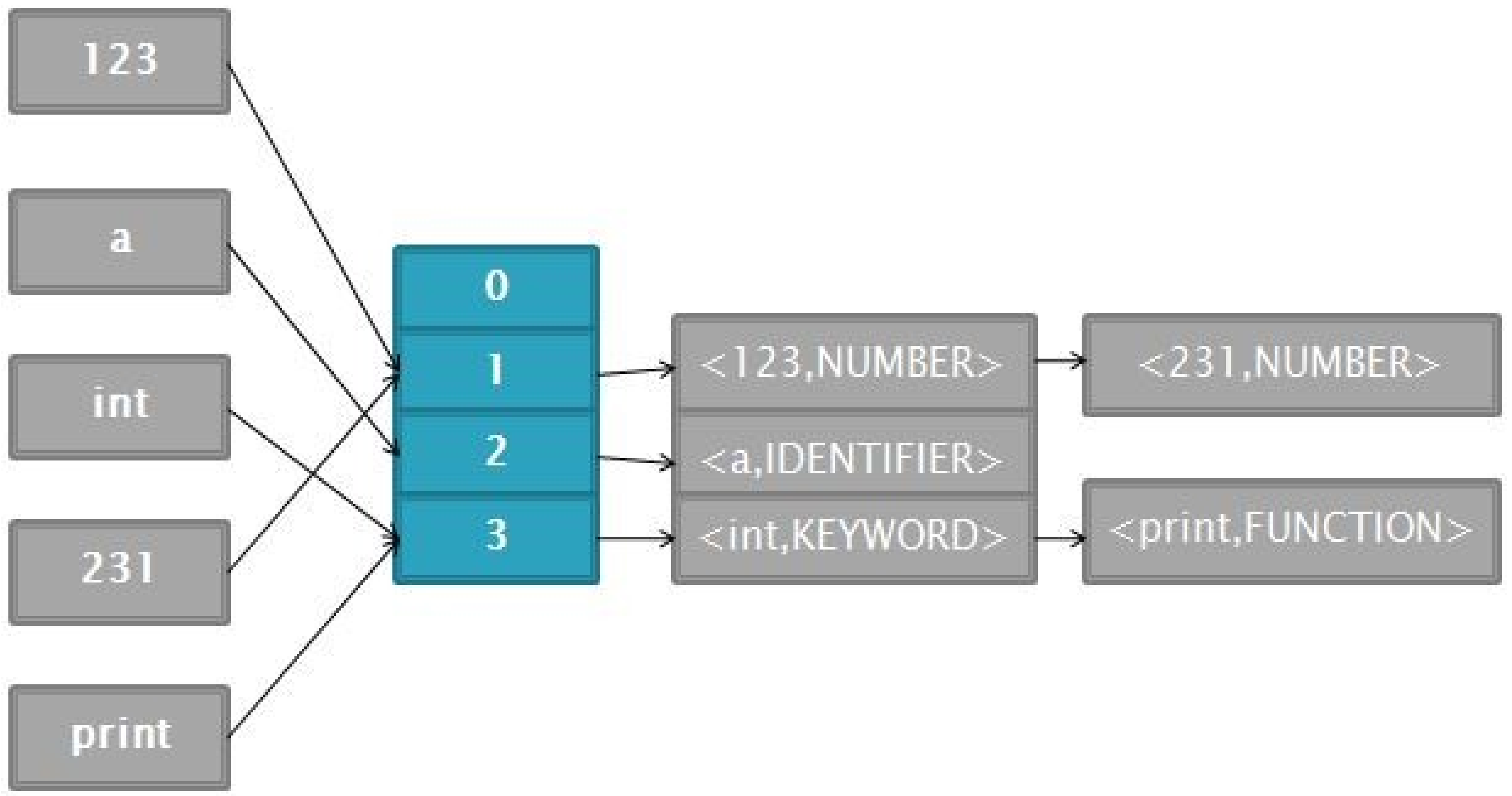
Symbol Table

- 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
  - Value
- Used in almost all phases of a compiler

Offline 1: Symbol Table Management

- Implement a simple symbol table
- Hash based (Chaining)
- Each entry is 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. **SUCCES**

Search b into symbol table. **FAILUR**

  - 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?

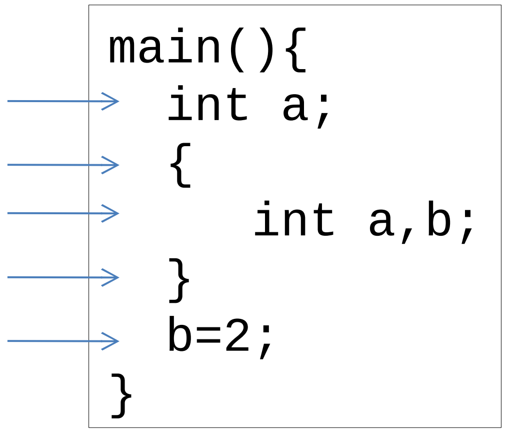
- How can this type of Symbol Table help?
  - Scope Management

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

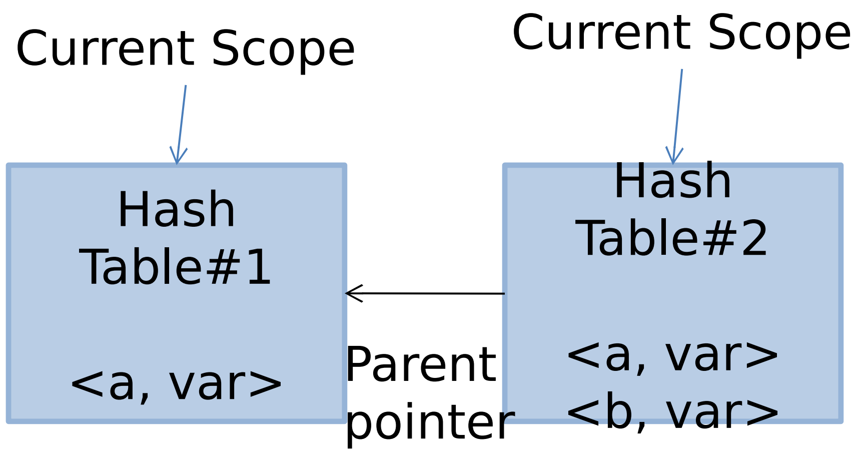
- Need to allow duplicate entry in symbol table
- Also delete some entries when a block exits
- How to accommodate this??

Symbol Table for Scope Management

- List of Hash Tables



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



Offline 1: Symbol Table Management

- Three Classes
  1. SymbolInfo
     - Each entry of symbol table is an instance of SymbolInfo.(Remember two tuples!)

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
     - Maintain a list of ScopeTables
     - Implement four operations
       - » Enter Scope
       - » Exit Scope
       - » Insert
       - » Delete
       - » Print All Tables
       - » Print Current Table