Lecture 20

Intermediate Code Generation

Awanish Pandey

Department of Computer Science and Engineering
Indian Institute of Technology
Roorkee

March 25, 2025

Symbol table

- Compiler uses symbol table to keep track of scope and binding information about names

Symbol table

- Compiler uses symbol table to keep track of scope and binding information about names
- Symbol table must have mechanism to:

Symbol table

- Compiler uses symbol table to keep track of scope and binding information about names
- Symbol table must have mechanism to:
  - add new entries

Symbol table

- Compiler uses symbol table to keep track of scope and binding information about names
- Symbol table must have mechanism to:
  - add new entries
  - find existing information efficiently

Symbol table

- Compiler uses symbol table to keep track of scope and binding information about names
- Symbol table must have mechanism to:
  - add new entries
  - find existing information efficiently
- Two common mechanisms:

Symbol table

- Compiler uses symbol table to keep track of scope and binding information about names
- Symbol table must have mechanism to:
  - add new entries
  - find existing information efficiently
- Two common mechanisms:
  - linear lists, simple to implement, poor performance

Symbol table

- Compiler uses symbol table to keep track of scope and binding information about names
- Symbol table must have mechanism to:
  - add new entries
  - find existing information efficiently
- Two common mechanisms:
  - linear lists, simple to implement, poor performance
  - hash tables, good performance, need good hash function

Symbol table

- Compiler uses symbol table to keep track of scope and binding information about names
- Symbol table must have mechanism to:
  - add new entries
  - find existing information efficiently
- Two common mechanisms:
  - linear lists, simple to implement, poor performance
  - hash tables, good performance, need good hash function
- Compiler should be able to grow symbol table dynamically

Symbol table

- Compiler uses symbol table to keep track of scope and binding information about names
- Symbol table must have mechanism to:
  - add new entries
  - find existing information efficiently
- Two common mechanisms:
  - linear lists, simple to implement, poor performance
  - hash tables, good performance, need good hash function
- Compiler should be able to grow symbol table dynamically
- If size is fixed, it must be large enough for the largest program

Symbol Attributes and Symbol Table Entries

- Symbols have associated attributes

Symbol Attributes and Symbol Table Entries

- Symbols have associated attributes
- Typical attributes are name, type, scope, size, addressing mode etc

Symbol Attributes and Symbol Table Entries

- Symbols have associated attributes
- Typical attributes are name, type, scope, size, addressing mode etc
- A symbol table entry collects together attributes such that they can be easily set and retrieved

Symbol Attributes and Symbol Table Entries

- Symbols have associated attributes
- Typical attributes are name, type, scope, size, addressing mode etc
- A symbol table entry collects together attributes such that they can be easily set and retrieved
- Example of typical names in symbol table

Name	Type
name	string
size	int
type	enum

Symbol Attributes and Symbol Table Entries

- Symbols have associated attributes
- Typical attributes are name, type, scope, size, addressing mode etc
- A symbol table entry collects together attributes such that they can be easily set and retrieved
- Example of typical names in symbol table

Name	Type
name	string
size	int
type	enum

- A major consideration in designing a symbol table is that insertion and retrieval should be as fast as possible

Symbol Table Entries

- Each entry for a declaration of a name

Symbol Table Entries

- Each entry for a declaration of a name
- Format need not be uniform because information depends upon the usage of the name

Symbol Table Entries

- Each entry for a declaration of a name
- Format need not be uniform because information depends upon the usage of the name
- Each entry is a record consisting of consecutive words

Symbol Table Entries

- Each entry for a declaration of a name
- Format need not be uniform because information depends upon the usage of the name
- Each entry is a record consisting of consecutive words
- To keep records uniform some entries may be outside the symbol table

Symbol Table Entries

- Each entry for a declaration of a name
- Format need not be uniform because information depends upon the usage of the name
- Each entry is a record consisting of consecutive words
- To keep records uniform some entries may be outside the symbol table
- Information is entered into symbol table at various times

Symbol Table Entries

- Each entry for a declaration of a name
- Format need not be uniform because information depends upon the usage of the name
- Each entry is a record consisting of consecutive words
- To keep records uniform some entries may be outside the symbol table
- Information is entered into symbol table at various times
  - Keywords are entered initially

Symbol Table Entries

- Each entry for a declaration of a name
- Format need not be uniform because information depends upon the usage of the name
- Each entry is a record consisting of consecutive words
- To keep records uniform some entries may be outside the symbol table
- Information is entered into symbol table at various times
  - Keywords are entered initially
  - Identifier lexemes are entered by lexical analyzer

Symbol Table Entries

- Each entry for a declaration of a name
- Format need not be uniform because information depends upon the usage of the name
- Each entry is a record consisting of consecutive words
- To keep records uniform some entries may be outside the symbol table
- Information is entered into symbol table at various times
  - Keywords are entered initially
  - Identifier lexemes are entered by lexical analyzer
- Attribute values are filled in as information is available

Data Structures

- List data structure

Data Structures

- List data structure
  - simplest to implement

Data Structures

- List data structure
  - simplest to implement
  - uses a single array to store names and information

Data Structures

- List data structure
  - simplest to implement
  - uses a single array to store names and information
  - search for a name is linear

Data Structures

- List data structure
  - simplest to implement
  - uses a single array to store names and information
  - search for a name is linear
  - entry and lookup are independent operations

Data Structures

- List data structure
  - simplest to implement
  - uses a single array to store names and information
  - search for a name is linear
  - entry and lookup are independent operations
  - cost of entry and search operations are very high and lot of time goes into bookkeeping.

Data Structures

- List data structure
  - simplest to implement
  - uses a single array to store names and information
  - search for a name is linear
  - entry and lookup are independent operations
  - cost of entry and search operations are very high and lot of time goes into bookkeeping.
- Hash table

Data Structures

- List data structure
  - simplest to implement
  - uses a single array to store names and information
  - search for a name is linear
  - entry and lookup are independent operations
  - cost of entry and search operations are very high and lot of time goes into bookkeeping.
- Hash table
  - The advantages are obvious

Representing Scope Information

- Entries are declarations of names.

Representing Scope Information

- Entries are declarations of names.
- When a lookup is done, an entry for an appropriate declaration must be returned.

Representing Scope Information

- Entries are declarations of names.
- When a lookup is done, an entry for an appropriate declaration must be returned.
- Scope rules determine which entry is appropriate

Representing Scope Information

- Entries are declarations of names.
- When a lookup is done, an entry for an appropriate declaration must be returned.
- Scope rules determine which entry is appropriate
- Maintain a separate table for each scope

Representing Scope Information

- Entries are declarations of names.
- When a lookup is done, an entry for an appropriate declaration must be returned.
- Scope rules determine which entry is appropriate
- Maintain a separate table for each scope
- Information about non-local is found by scanning the symbol table for the enclosing procedures

Representing Scope Information

- Entries are declarations of names.
- When a lookup is done, an entry for an appropriate declaration must be returned.
- Scope rules determine which entry is appropriate
- Maintain a separate table for each scope
- Information about non-local is found by scanning the symbol table for the enclosing procedures
- Names are entered in the symbol table in the order they occur

Representing Scope Information

- Entries are declarations of names.
- When a lookup is done, an entry for an appropriate declaration must be returned.
- Scope rules determine which entry is appropriate
- Maintain a separate table for each scope
- Information about non-local is found by scanning the symbol table for the enclosing procedures
- Names are entered in the symbol table in the order they occur
- Most closely nested rule can be created in terms of the following operations:

Representing Scope Information

- Entries are declarations of names.
- When a lookup is done, an entry for an appropriate declaration must be returned.
- Scope rules determine which entry is appropriate
- Maintain a separate table for each scope
- Information about non-local is found by scanning the symbol table for the enclosing procedures
- Names are entered in the symbol table in the order they occur
- Most closely nested rule can be created in terms of the following operations:
  - lookup: find the most recently created entry
  - insert: make a new entry
  - delete: remove the most recently created entry

Declarations

- For each name create symbol table entry with information like type and relative address
- $P \rightarrow \{offset = 0\} \quad D$
- $D \rightarrow D; D$

Declarations

- For each name create symbol table entry with information like type and relative address
- $P \rightarrow \{offset = 0\} \quad D$
- $D \rightarrow D; D$
- $D \rightarrow id : T$

Declarations

- For each name create symbol table entry with information like type and relative address
- $P \rightarrow \{offset = 0\} \quad D$
- $D \rightarrow D; D$
- $D \rightarrow id : T \quad enter(id.name, T.type, offset);$
  $offset = offset + T.width$

Declarations

- For each name create symbol table entry with information like type and relative address
- $P \rightarrow \{offset = 0\} \quad D$
- $D \rightarrow D; D$
- $D \rightarrow id : T \quad enter(id.name, T.type, offset);$
  $offset = offset + T.width$
- $T \rightarrow integer$

Declarations

- For each name create symbol table entry with information like type and relative address
- $P \rightarrow \{offset = 0\} \quad D$
- $D \rightarrow D; D$
- $D \rightarrow id : T \quad enter(id.name, T.type, offset);$
  $offset = offset + T.width$
- $T \rightarrow integer \quad T.type = integer; T.width = 4$
- $T \rightarrow real$

Declarations

- For each name create symbol table entry with information like type and relative address
- $P \rightarrow \{offset = 0\} \quad D$
- $D \rightarrow D; D$
- $D \rightarrow id : T \quad enter(id.name, T.type, offset);$
  $offset = offset + T.width$
- $T \rightarrow integer \quad T.type = integer; T.width = 4$
- $T \rightarrow real \quad T.type = real; T.width = 8$
- $T \rightarrow array[num]$ of T_1

Declarations

- For each name create symbol table entry with information like type and relative address
- $P \rightarrow \{offset = 0\} \quad D$
- $D \rightarrow D; D$
- $D \rightarrow id : T \quad enter(id.name, T.type, offset);$
  $offset = offset + T.width$
- $T \rightarrow integer \quad T.type = integer; T.width = 4$
- $T \rightarrow real \quad T.type = real; T.width = 8$
- $T \rightarrow array[num]$ of T_1
  $T.type = array(num.val, T_1.type)$
  $T.width = num.val \times T_1.width$
- $T \rightarrow *T_1$

Declarations

- For each name create symbol table entry with information like type and relative address
- $P \rightarrow \{offset = 0\} \quad D$
- $D \rightarrow D; D$
- $D \rightarrow id : T \quad enter(id.name, T.type, offset);$
  $offset = offset + T.width$

width is required to maintain offset

- $T \rightarrow integer \quad T.type = integer; T.width = 4$
- $T \rightarrow real \quad T.type = real; T.width = 8$
- $T \rightarrow array[num]$ of T_1
  $T.type = array(num.val, T_1.type)$
  $T.width = num.val \times T_1.width$
- $T \rightarrow *T_1 \; T.type = pointer(T_1.type)$
  $T.width = 4$

Keeping track of local information

- When a nested procedure is seen, processing of declaration in enclosing procedure is temporarily suspended

Keeping track of local information

- When a nested procedure is seen, processing of declaration in enclosing procedure is temporarily suspended
- A new symbol table is created when procedure declaration $D \rightarrow proc\ id; D_1; S$ is seen

Keeping track of local information

- When a nested procedure is seen, processing of declaration in enclosing procedure is temporarily suspended
- A new symbol table is created when procedure declaration $D \rightarrow proc\ id; D_1; S$ is seen
- Entries for D_1 are created in the new symbol table

Keeping track of local information

- When a nested procedure is seen, processing of declaration in enclosing procedure is temporarily suspended
- A new symbol table is created when procedure declaration $D \rightarrow proc\ id; D_1; S$ is seen
- Entries for D_1 are created in the new symbol table
- The name represented by `id` is local to the enclosing procedure

nested function signature will be present in the parent function's symbol table.

Creating Symbol Table

- **mktable(previous):** create a new symbol table and return a pointer to the new table. The argument previous points to the enclosing procedure.

Creating Symbol Table

- **mktable(previous):** create a new symbol table and return a pointer to the new table. The argument previous points to the enclosing procedure.
- **enter (table, name, type, offset):** Create a new entry

Creating Symbol Table

- **mktable(previous):** create a new symbol table and return a pointer to the new table. The argument previous points to the enclosing procedure.
- **enter (table, name, type, offset):** Create a new entry
- **addwidth (table, width):** records cumulative width of all the entries in a table

Creating Symbol Table

- **mktable(previous):** create a new symbol table and return a pointer to the new table. The argument previous points to the enclosing procedure.
- **enter (table, name, type, offset):** Create a new entry
- **addwidth (table, width):** records cumulative width of all the entries in a table
- **enterproc (table, name, newtable):** creates a new entry for procedure name. newtable points to the symbol table of the new procedure

Creating Symbol Table

- **mktable(previous):** create a new symbol table and return a pointer to the new table. The argument previous points to the enclosing procedure.
- **enter (table, name, type, offset):** Create a new entry
- **addwidth (table, width):** records cumulative width of all the entries in a table
- **enterproc (table, name, newtable):** creates a new entry for procedure name. newtable points to the symbol table of the new procedure
- $P \to D$
- $P \to \quad \{t = mktable(nil);$
  $push(t, tblptr);$
  $push(0, offset)\} \quad D$

Creating Symbol Table

- **mktable(previous):** create a new symbol table and return a pointer to the new table. The argument previous points to the enclosing procedure.
- **enter (table, name, type, offset):** Create a new entry
- **addwidth (table, width):** records cumulative width of all the entries in a table
- **enterproc (table, name, newtable):** creates a new entry for procedure name. newtable points to the symbol table of the new procedure
- $P \to D$
- $P \to \quad \{t = mktable(nil);$
  $push(t, tblptr);$
  $push(0, offset)\} \quad D$
  $\{addwidth(top(tblptr), top(offset));$
  $pop(tblptr);$
  $pop(offset)\}$

tblptr is a stack that maintains table pointers. Top will be the current symbol table pointer.

offset is a stack that contains the corresponding offset.

Creating Symbol Table

- $D \rightarrow proc\ id\ D_1; S;$

Creating Symbol Table

- $D \rightarrow proc\ id\ \ D_1; S;$
- $D \rightarrow proc\ id; \{t = mktable(top(tblptr));$
  $push(t, tblptr);$
  $push(0, offset)\} \quad D_1; S$

Creating Symbol Table

- $D \rightarrow proc\ \ id\ \ D_1; S;$
- $D \rightarrow proc\ \ id; \{t = mktable(top(tblptr));$
  $push(t, tblptr);$
  $push(0, offset)\} \quad D_1; S\{t = top(tblptr);$
  $addwidth(t, top(offset)); pop(tblptr); pop(offset);$
  $enterproc(top(tblptr), id.name, t)\}$
- $D \rightarrow id\ :\ T$

procedure signature is added at the last to parent's symbol table.

Creating Symbol Table

- $D \rightarrow proc\ \ id\ \ D_1; S;$
- $D \rightarrow proc\ \ id; \{t = mktable(top(tblptr));$
  $push(t, tblptr);$
  $push(0, offset)\} \quad D_1; S\{t = top(tblptr);$
  $addwidth(t, top(offset)); pop(tblptr); pop(offset);$
  $enterproc(top(tblptr), id.name, t)\}$
- $D \rightarrow id : T \quad \{enter(top(tblptr), id.name, T.type, top(offset));$
  $top(offset) = top(offset) + T.width\}$

Syntax directed translation of expression into 3-address code

- $S \rightarrow id = E$

Syntax directed translation of expression into 3-address code

- $S \rightarrow id = E \quad\quad S.code = E.code ||$
  $gen(id.place = E.place)$

Syntax directed translation of expression into 3-address code

- $S \rightarrow id = E \quad S.code = E.code||$
  $gen(id.place = E.place)$
- $E \rightarrow E_1 + E_2$

Syntax directed translation of expression into 3-address code

- $S \rightarrow id = E \quad S.code = E.code||$
  $gen(id.place = E.place)$
- $E \rightarrow E_1 + E_2 \quad E.place = newtmp()$
  $E.code = E_1.code||E_2.code||$
  $gen(E.place = E_1.place + E_2.place)$
- $E \rightarrow E_1 * E_2$

Syntax directed translation of expression into 3-address code

- $S \rightarrow id = E \quad S.code = E.code||$
  $gen(id.place = E.place)$
- $E \rightarrow E_1 + E_2 \quad E.place = newtmp()$
  $E.code = E_1.code||E_2.code||$
  $gen(E.place = E_1.place + E_2.place)$
- $E \rightarrow E_1 * E_2 \quad E.place = newtmp()$
  $E.code = E_1.code||E_2.code||$
  $gen(E.place = E_1.place * E_2.place)$
- $E \rightarrow -E_1$

Syntax directed translation of expression into 3-address code

- $S \rightarrow id = E \quad S.code = E.code||$
  $gen(id.place = E.place)$
- $E \rightarrow E_1 + E_2 \quad E.place = newtmp()$
  $E.code = E_1.code||E_2.code||$
  $gen(E.place = E_1.place + E_2.place)$
- $E \rightarrow E_1 * E_2 \quad E.place = newtmp()$
  $E.code = E_1.code||E_2.code||$
  $gen(E.place = E_1.place * E_2.place)$
- $E \rightarrow -E_1 \quad E.place = newtmp()$
  $E.code = E_1.code||$
  $gen(E.place = -E_1.place)$

Syntax directed translation of expression into 3-address code

- $E \rightarrow (E_1)$ $E.place = E_1.place$
  $E.code = E_1.code$

Syntax directed translation of expression into 3-address code

- $E \rightarrow (E_1)$ $E.place = E_1.place$
  $E.code = E_1.code$
- $E \rightarrow id$

Syntax directed translation of expression into 3-address code

- $E \rightarrow (E_1)$ $E.place = E_1.place$
  $E.code = E_1.code$
- $E \rightarrow id$ $\quad E.place = id.place$
  $E.code =$ " "

Generate 3AC for $a = b * -c + b * -c$