Chapter 5 Object Oriented System Design

SUBASH MANANDHAR

Introduction

- Focus on the objects handled by the system, rather than algorithms.
- Programs are designed and implemented as collections of objects, not as collections of procedures.

Principles of object oriented programming

- **Object**:
  - Is basic unit of OOP.
  - Is a component of a program that knows how to perform certain actions and how to interact with other elements of the program.
  - Contains some data and defines a set of operations on that data that can be invoked by other parts of program.
  - E.g. consider Symbol Table as an object used by assembler.
    - Here, set of operations or methods are like Insert_Symbol and Lookup_Symbol.
    - Its data would be contents of hash table used to store symbols and their addresses.

Principles of object oriented programming

- **Class**:
  - Is a blueprint or template or set of instructions to build a specific type of object.
  - Defines the instance variables and methods of an object.
  - An instance is a specific object from specific class.
  - Many objects can be created from same class
  - E.g. for an assembler to translate programs for different versions of machine, class could be Opcode_Table.
    - For this class, object could be created to define instruction set for machine.

Principles of object oriented programming

- **Encapsulation**:
  - Means that the internal representation of an object is generally hidden from view outside of objects definition.
  - Is the hiding of data implementation by restricting access to accessors and mutators.
- **Abstraction**:
  - Is a model, a view or some other focused representation for an actual item.
  - Is the implementation of an object that contains same essential properties and actions we can find in the original object we are representing.

Principles of object oriented programming

- **Inheritance**:
  - Is a way to reuse code of existing objects or to establish a subtype from an existing object.
  - The relationship of classes through inheritance gives rise to hierarchy.
  - **Subclass**:
    - Is a modular, derivative class that inherits one or more properties from another class.
  - **Superclass**:
    - Establishes a common interface and foundation functionality, which specialized subclass can inherit modify and supplement.

Principles of object oriented programming

- **Polymorphism**:
  - Means one name, many forms.
  - Manifests itself by having multiple methods all with same name, but slightly different functionality.

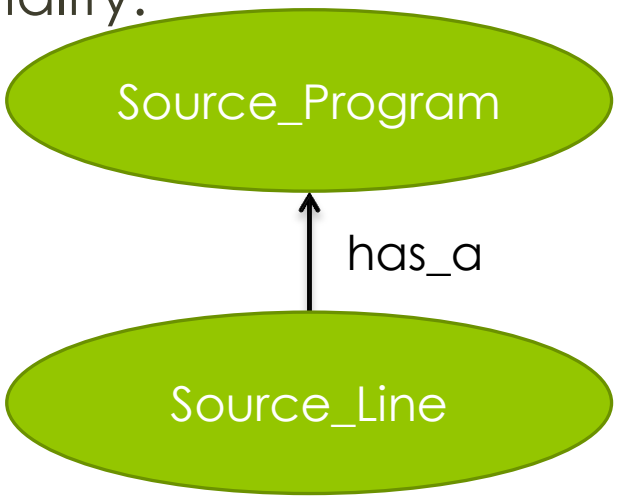


Fig 1: has a relationship

Principles of object oriented programming

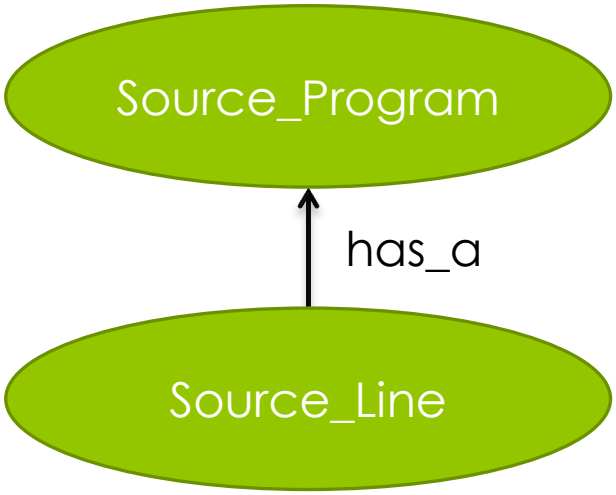


Fig 1: has a relationship

Principles of object oriented programming

- Hash_Table is a base class.
- Other two are subclass.
- If insert_item and search_item are method of base class then other subclasses automatically contains definition of methods.

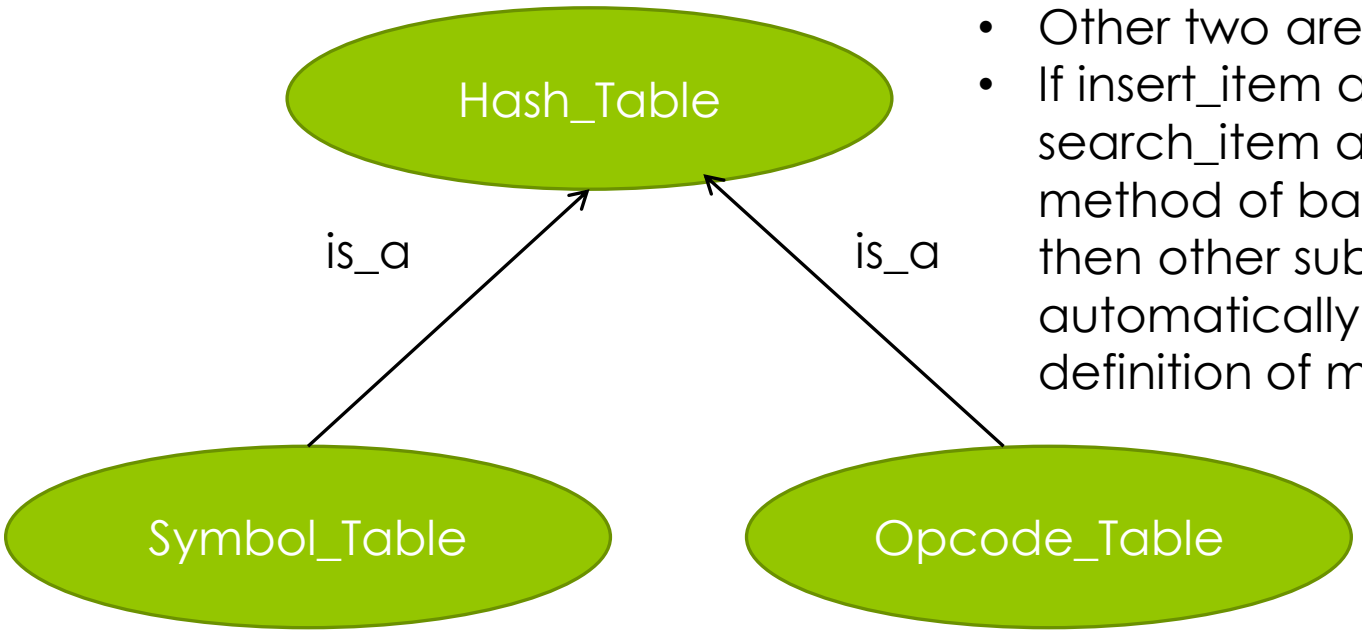


Fig 2: is a relationship or inheritance

Principles of object oriented programming

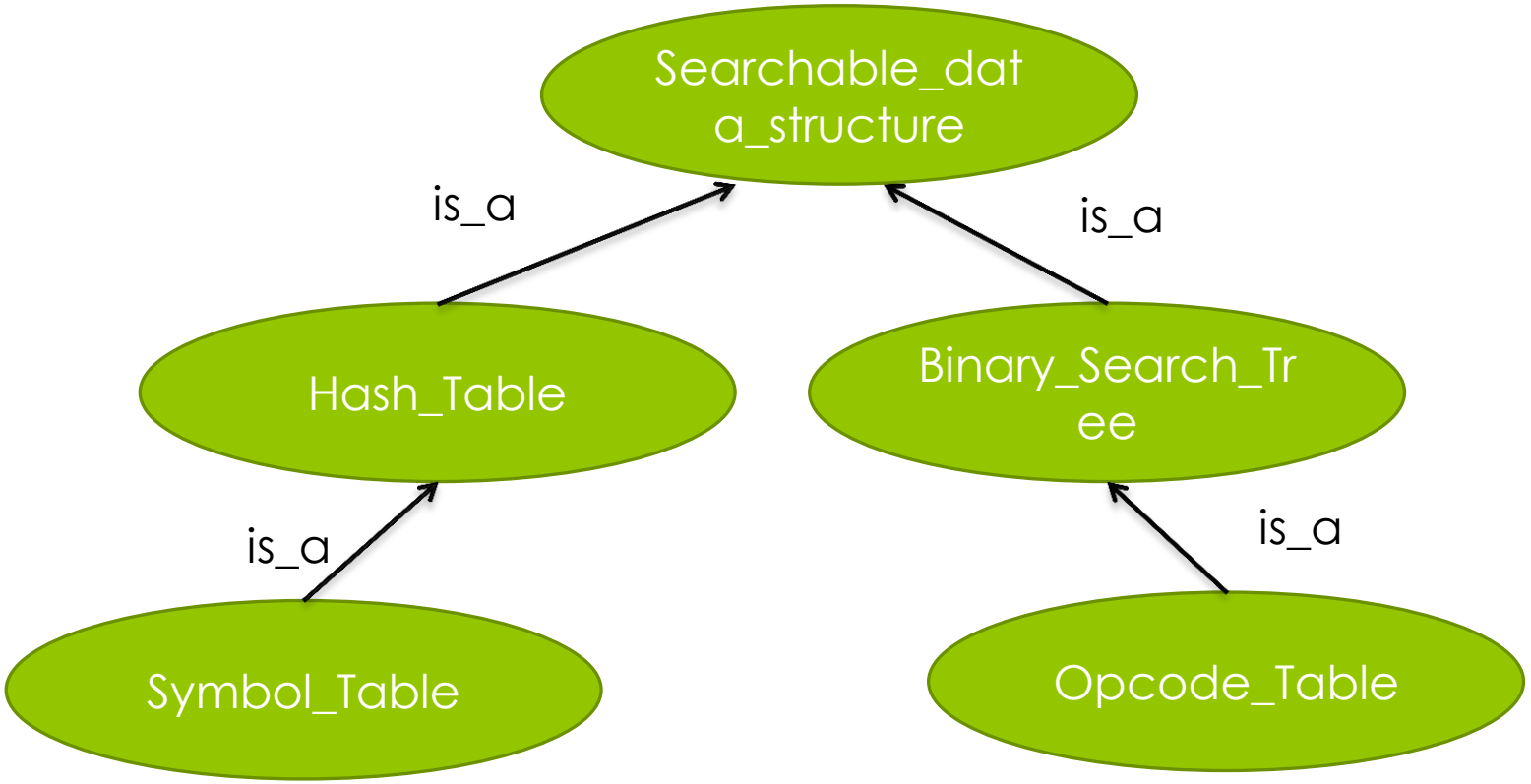


Fig 3: Polymorphism

Principles of object oriented programming

- **Polymorphism**:
- Here,
  - Superclass Searchable_data_structure defines two methods Insert_Item and Search_For_Item.
  - Hash_table and Binary_search_tree are subclasses. So inherits above methods.
  - Implementation of the methods are different in those subclasses. But names of methods and way of invocation are same.
  - If Search_for_item method is invoked as instance of Symbol_table, it will result in a retrieval from Hash_table.
  - If same method is invoked on an instance of Opcode_table, it will result in a Binary_search_tree.
  - This shows polymorphism.

Object Oriented Design of an Assembler

- According to Booch, two different development processes: a)micro b)macro
- Booch's Macro process represents overall activities of development on a long range scale
  - Establish the requirements for the software. (Conceptualization)
  - Develop an overall model of system's behavior. (Analysis)
  - Create an architecture for the implementation. (Design)
  - Develop the implementation through successive refinements. (Evolution)
  - Manage the continued evolution of a delivered system. (Maintenance)
- This Booch's Macro process repeats itself after each release of software.
- Similar to waterfall model.

Object Oriented Design of an Assembler

- Booch's Micro process represents daily activities of system developer
  - Identify the classes and objects of the system.
  - Establish the behavior and other attributed of the classes and objects.
  - Analyze the relationship among the classes and objects.
  - Specify the implementation of classes and objects.
- These activities may be repeated as needed with increasing level of details.

Object Oriented Design of an Assembler

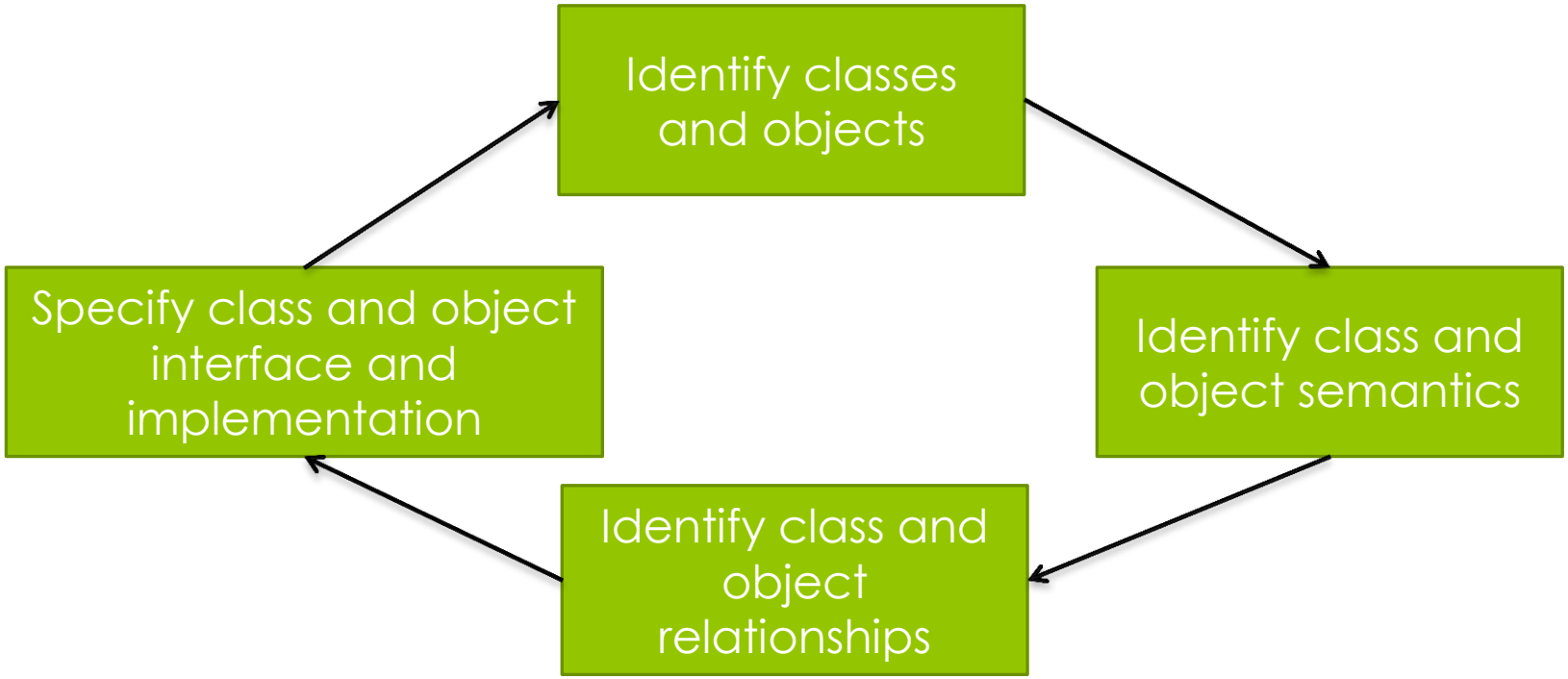


Fig: Booch Micro

Object Oriented Design of an Assembler

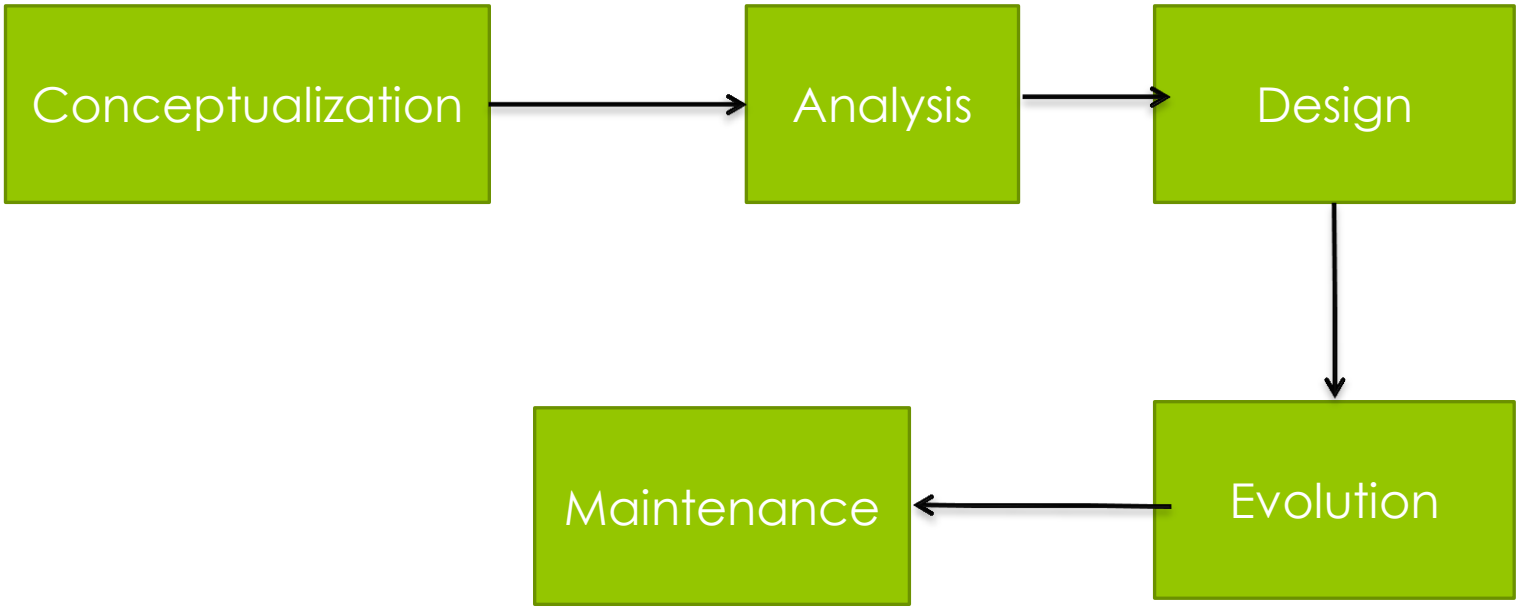


Fig: Booch Macro

Objects identified during design of assembler

- **Source_Program**
  - **Contents**
    - Program
    - Current location counter value, summary of errors
    - One object of class Source_Line for each line of the program
  - **Methods**
    - **Assemble**
      - Translate source program, produce an object program and an assembly listing.

Objects identified during design of assembler

- **Source_Line**
  - **Contents**
    - Line of source program
    - Location counter value, errors
  - **Methods**
    - **Create**
      - Create and initialize new instance of Source
    - **Assign_Location**
      - Assign location counter value to line
      - Return updated location counter value
      - Enter label on line (if any) in symbol table

Objects identified during design of assembler

- **Source_Line**
  - **Methods**
    - **Translate**
      - Translate the instruction or data definition on the line into machine language.
      - Make entries in object program and assembly listing
    - **Record_error**
      - Record error detected.

Objects identified during design of assembler

- **Symbol_Table**
  - **Contents**
    - Labels defined in source program with its location counter value.
  - **Methods**
    - **Enter**
      - Enter a label and location counter value into table.
      - Return error if label is already defined.
    - **Search**
      - Search table for specified label
      - Return location counter value of label or error if label is not defined.

Objects identified during design of assembler

- **Opcode_Table**
  - **Contents**
    - Mnemonic instructions
    - Includes machine instruction format and opcode
  - **Methods**
    - **Search**
      - Search table for specified mnemonic instruction
      - Return information about instruction format and operands required
      - Return error if mnemonic instruction not defined

Objects identified during design of assembler

- **Object_Program**
  - **Contents**
    - Object program after assembly
    - Includes machine language translation of instruction and data definition from object program
    - Includes program length
  - **Methods**
    - **Enter_Text**
      - Enter machine language translation of an instruction or data definition into object program

Objects identified during design of assembler

- **Object_Program**
  - **Methods**
    - **Complete**
      - Enter program length and complete generation of external object program file.
- **Assembly_Listing**
  - **Contents**
    - Listing of lines of source program and corresponding machine language translation
    - Includes errors for each line and summary of errors in program.

Objects identified during design of assembler

- **Assembly_Listing**
  - **Methods**
    - **Enter_Line**
      - Enter source line, the corresponding machine language translation and description of errors detected for the line into assembly listing.
    - **Complete**
      - Enter summary of errors and detected and complete the generation of external assembly listing file.

Object Diagram

- Indicates the methods that are invoked by each object.
- E.g. Source_Program object invokes methods create, Assign_Location and translate on the Source_Line objects.
- Object diagram may also indicate the class of each object.
- The invocation may be numbered to indicate the sequence in which they occur and the flows of information they cause.

Object Diagram

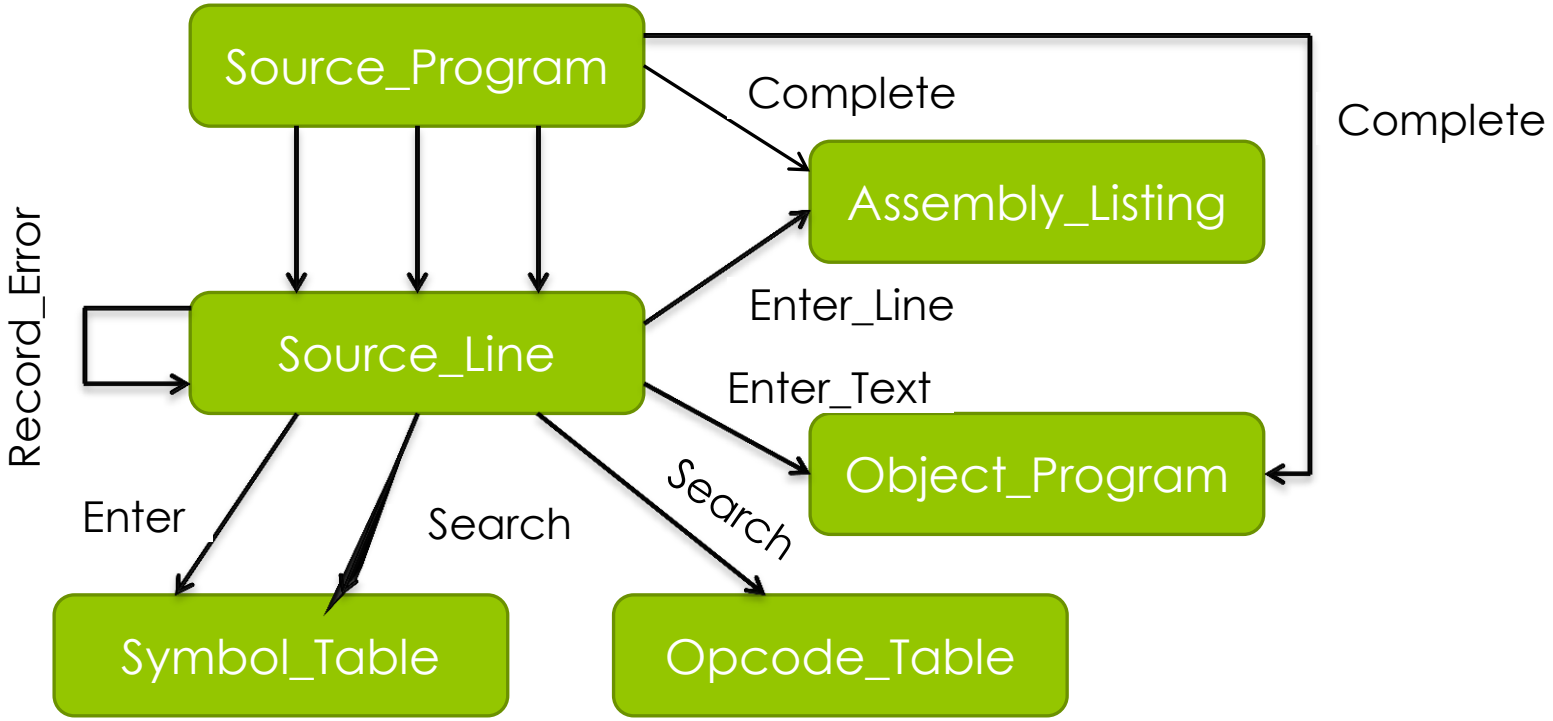
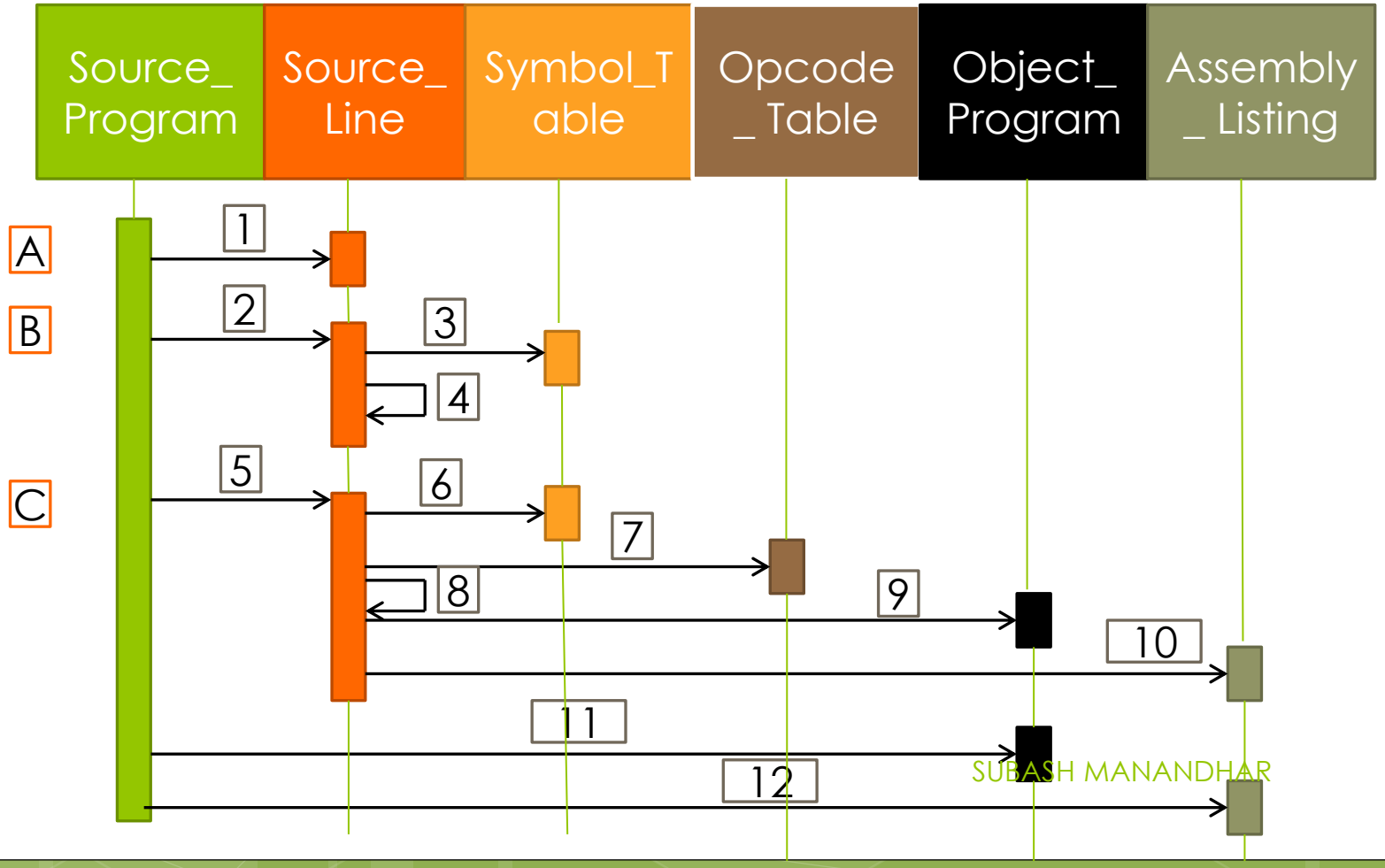
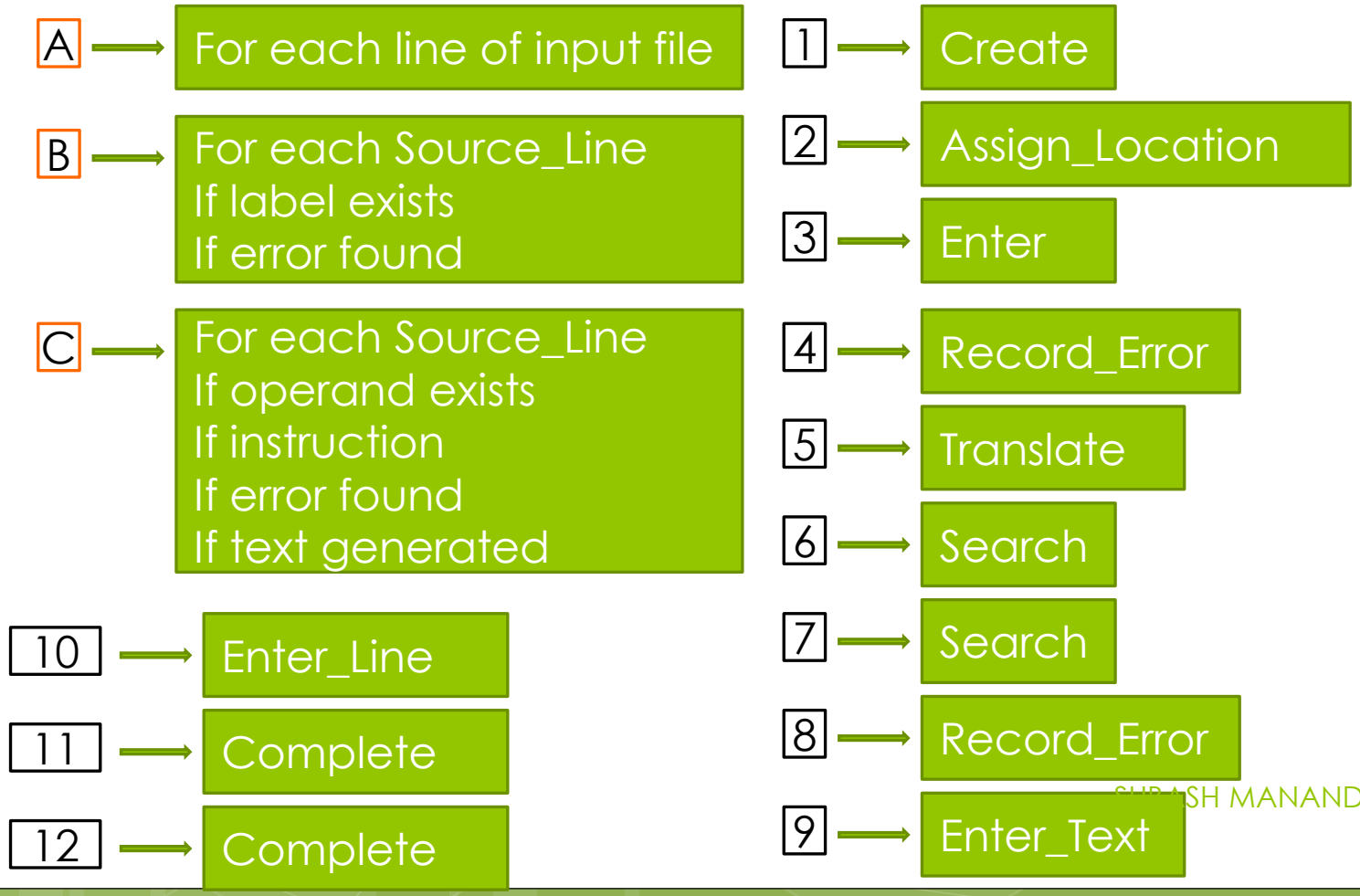


Fig: Object diagram of Assembler

Interaction Diagram



Interaction Diagram



Interaction Diagram

- Makes easy to visualize the sequence of objects invocation and flow of control between objects.
- Here, each object is represented by solid vertical line.
- Invocation of method is shown by horizontal line between one object and another.
- The sequence is indicated by their vertical position in diagram.
- A script is often written at L.H.S. of diagram to describe condition and iteration.
- A narrow vertical box can be used to indicate the time that how of control is focused in each object.