Debere Birhan Polytechnic College

MODULE TITLE: **Designing Program Logic**

LO1: Select the program design approach

1.1 Obtaining design documentation and requirement for the program clarification

- **Pseudocode**

In computer science and numerical computation, **pseudocode** is an informal high-level description of the operating principle of a computer program or other **algorithm**.

- An outline of a program, written in a form that can easily be converted into real programming statements.
- It cannot be compiled nor executed, and there is no real formatting or syntax rules. It is simply an important step in producing the final code.

An algorithm is a procedure for solving a problem in terms of the actions to be executed and the order in which those actions are to be executed.

- An algorithm is merely the sequence of steps taken to solve a problem. The steps are normally "sequence," "selection,” "iteration," and a case-type statement.

The "selection" is the "if then else" statement, and the iteration is satisfied by a number of statements, such as the "while," " do," and the "for," while the case-type statement is satisfied by the "switch" statement.

Algorithm vs **Pseudocode**

Both Algorithm and Pseudo code more or less describe the logical sequence of steps that follow in solving a problem **Pseudocode** consists of short readable and formally-styled natural language that used to explain specific tasks within a program's algorithm while an **Algorithm** is a group of instructions or a set of steps applied to solve a particular problem. - A Pseudo code is a method used to define an algorithm.

- An algorithm is written in a natural language while pseudo code can be written in high level programming languages.
- **Pseudcode** cannot be executed on a real computer, but it models and resembles real programming code, and is written at roughly the same level of detail.

Example: The following pseudocode describes an algorithm which will accept two numbers from the keyboard and calculate the sum and product displaying the answer on the monitor screen.

Solution:

```
Use variables sum, product, number1, number2 of type real
Display “Input two numbers”
Accept number1, number2
Sum = number1 + number2
Print “The sum is “, sum
Product = number1 * number2
Print “The Product is “, product
End program
```

- **Flow charts**

A **flowchart** is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting those with arrows.

This diagrammatic representation can give a step-by-step solution to a given problem.

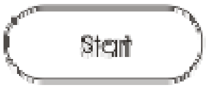
Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

A flow chart can be used to:

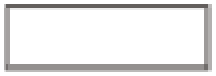
- Define and analyze processes.
- Build a step-by-step picture of the process for analysis, discussion, or communication.
- Define, standardize or find areas for improvement in a process

Most flow charts are made up of three main types of symbol:

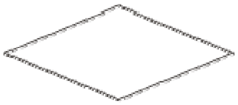
- Elongated circles, which signify the start or end of a process.



- Rectangles, which show instructions or actions.



- Diamonds, which show decisions that must be made

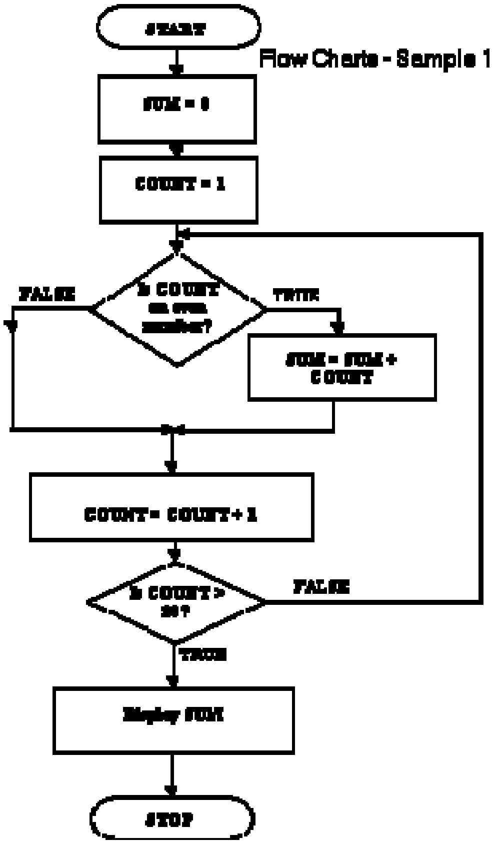


Within each symbol, you can write down what the symbol represents. The above diagrams show the start or finish of the process, the action to be taken, or the decision to be made respectively.
Symbols are connected one to the other by arrows, showing the flow of the process.

Example1: The following pseudocode describes an algorithm which will accept a number from the keyboard and calculate the sum of n numbers and design the corresponding flowchart.

Solution:

```
Start
Sum = 0
Display "Input value n"
Input n
For (I = 1, n, 5)
Input a value
Sum = sum + value
Increment
ENDFOR
Output sum
Stop
```



Flow Charts - Sample 1

Exercise: 2

Design the ***Pseudocode*** that sums all the even numbers between 1 and 20 inclusive and then displays the sum.
It uses a repeat loop and contains a null else within the repeat loop.

```
Start
Sum = 0
count = 1
REPEAT
IF count is even THEN sum = sum + count
count = count + 1
UNTIL count > 20
DISPLAY sum
```

➢ **ERD's**

An **entity-relationship diagram** is a data modeling technique that creates a graphical representation of the entities, and the relationships between entities, within an information system.

Data modeling is the formalization and documentation of existing processes and events that occur during application software design and development.

The three main components of an ERD are:

- The **Entity** is a person, object, place or event for which data is collected.
  Example: if you consider the information system for a business, entities would include not only customers, but also the customer's address, and orders as well. The entity is represented by a rectangle and labelled with a singular noun.
- The **Relationship** is the interaction between the entities. A relationship may be represented by a diamond shape that can be connected by the line to the entities. Verbs are used to label the relationships.
- The ***cardinality*** defines the relationship between the entities in terms of numbers. The three main cardinal relationships are: one-to-one, expressed as 1:1; one-to-many, expressed as 1:M; and many-to-many, expressed as M:N.

The steps involved in creating an ERD are:

- Identify the entities.
- Determine all significant interactions.
- Analyze the nature of the interactions.
- Draw the ERD.

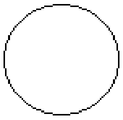
➢ DFD’s

Data flow diagram is the directional movement of data to and from external entities, the process and data stores.
If it flows into a data store, means a write, update, delete, etc. Flows out of data stores, mean read, display (select) types of transaction.

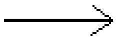
- DFD is an excellent communication tool for analysts to model processes and functional requirements.
- It represents the flows of data between different processes in a business.

DFDs only involve four symbols. They are:

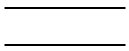
- Process
- Data flow/Data Object/
- Data Store
- External entity



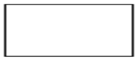
Process
Transform of incoming data flow(s) to outgoing flow(s).



Data Flow
Movement of data in the system.



Data Store
Data repositories for data that are not moving. It may be as simple as a buffer or a queue.



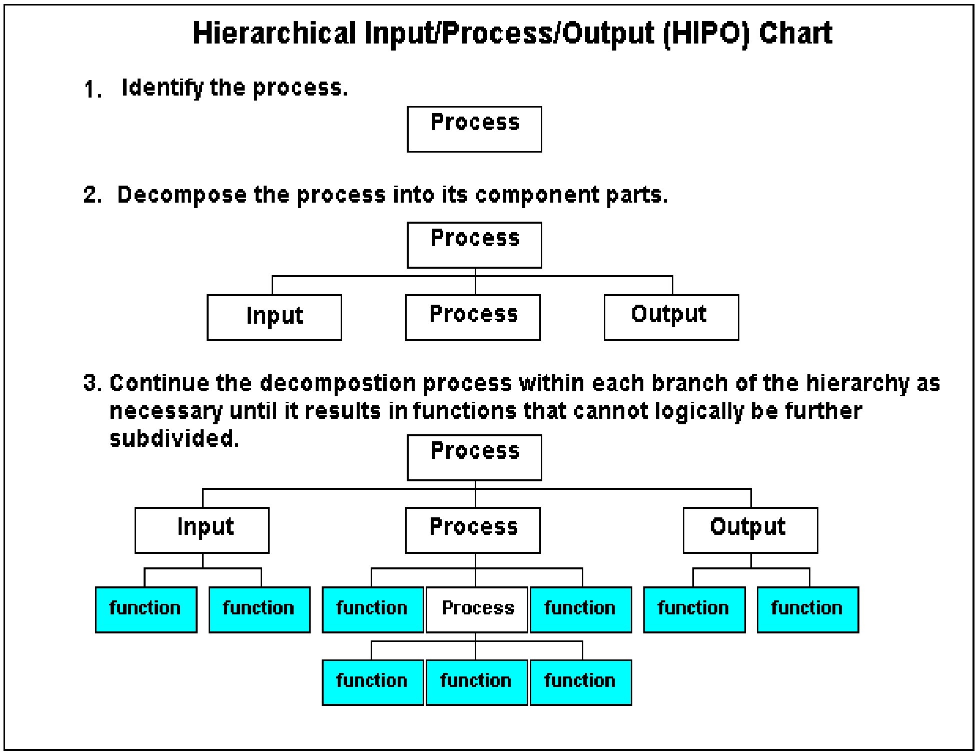
External Entity
Sources of destinations outside the specified system boundary.

➢ HIPO Charts

The HIPO (Hierarchy plus Input-Process-Output) acts as a hierarchical chart for the function performed by the system.

The **HIPO chart** is a tool used to analyze a problem and visualize a solution using the **top down design** approach. Starting at the global (macro) level, the chart is decomposed repeatedly at ever-greater levels of detail until the **logical building blocks (functions)** are identified.

A HIPO model consists of a hierarchy chart that graphically represents the program's control structure and a set of IPO (Input-Process-Output) charts that describe the inputs to, the outputs from, and the functions (or processes) performed by each module on the hierarchy chart.
The general model is illustrated below:


Hierarchical Input/Process/Output (HIPO) Chart
1. Identify the process.
Process
2. Decompose the process into its component parts.
Process
Input
Process
Output
3. Continue the decompostion process within each branch of the hierarchy as necessary until it results in functions that cannot logically be further subdivided.
Process
Input
Process
Output
function
function
function
Process
function
function
function
function
function
function

➢ Data Structure

In computer science, a **data structure** is a particular way(specialized format) of storing and organizing data in a computer so that it can be used efficiently.

- It provides a means to manage huge amounts of data efficiently, such as large databases.
- It includes the array, the file, the record, the table, the tree, and so on.

A data structure is a group of data elements grouped together under one name. These data elements, known as *members*, can have different types and different lengths.

➢ RAD

RAD (Rapid Application Development) is a more advanced program design tool (a programming system) that enables programmers to quickly build working programs.
In general, RAD systems provide a number of tools to help build graphical user interfaces that would normally take a large development effort.
RAD is the process of quickly placing controls on a form—like you just saw done with Visual Basic.
Example: When you place controls on a form, the Visual Basic system handles all the programming needed for that control. You don't ever have to write anything to make a command button act like a command button should. Your only goal is to determine how much command buttons your program needs and where they are to go.

➢ Case Tools

CASE tool (**C**omputer **A**ided **S**oftware **E**ngineering tool) is software that can be used to mean any computer-based tool for software planning, development, and evolution.

- CASE tool is software that assists with software development.

The main purpose of using a CASE tool is to produce error-free, easy to maintain program code.

- It is a category of software that provides a development environment for programming teams.
- It is the use of a computer-assisted method to organize and control the development of software, especially on large, complex projects involving many software components and people.

CASE systems provide tools to automate, manage and simplify the development process.

These can include tools for:

- Summarizing initial requirements
- Developing flow diagrams
- Scheduling development tasks
- Preparing documentation
- Controlling software versions
- Developing program code

➢ Prototyping

Prototyping is the process of quickly putting together a working model (a prototype) in order to test various aspects of a design, illustrate ideas or features and gather early user feedback.

Prototyping is often treated as an integral part of the system design process, where it is believed to reduce project risk and cost.

- It can also be a method used by designers to acquire feedback from users about future designs.

The goal of prototyping is to support requirements determination to develop concrete specifications for the ultimate (final) system.

- Prototyping is most useful in the following circumstances/situation
    - User requirements are not clear or well understood
    - Only one or a few users involved
    - Possible designs are complex
    - Communication problems have existed in the past, between users and analysts

➢ Modular programming

Modular programming (also called "top-down design" and "stepwise refinement") is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable **modules**, such that each contains everything necessary to execute only one aspect of the desired functionality.

A module is a separate unit of software or hardware.

- Modular programming means break a large program into smaller independent modules (process of subdividing a computer program into separate sub-programs)

Designing program

So, Modular programming is a solution to the problem of very large programs that are difficult to debug and maintain. By segmenting the program into modules that perform clearly defined functions, you can determine the source of program errors more easily.

In modular programming, similar functions are grouped in the same unit of programming code and separate functions are developed as separate units of code.

Modular programming enables multiple programmers to divide up the work and debug pieces of the program independently.

The benefits of using modular programming include:

- Less code has to be written.
- A single procedure can be developed for reuse, eliminating the need to retype the code many times.
- Programs can be designed more easily because a small team deals with only a small part of the entire code.
- It allows many programmers to collaborate on the same application.
- The code is stored across multiple files.
- Code is short, simple and easy to understand.
- Errors can easily be identified, as they are localized to a subroutine or function.
- The same code can be used in many applications.
- The scoping of variables can easily be controlled.

LO2. Document the program logic or design

2.1 Structuring diagrams of flow and modules

- **Visio**

Visio is a Microsoft tool for drawing diagrams, including database diagrams (ERDs).
It is a diagramming tool that can be used to visually communicate technical as well as non-technical representations of ideas, processes, concepts, structures, layouts, software models, blueprints, etc.

- **Visio** software eliminates the laborious (difficult or lengthy) process of creating diagrams by providing the tools to create complex diagrams in a user friendly manner.

- **Smart Draw**

Smart Draw is a visual processor used to create flowcharts, organization charts, mind maps, project charts, and other visuals.
I.e.:- It is a tool used to help you create visuals for reports, presentations, business functions, and any other reason you may need visuals.
Smart Draw is unique in three key ways:

1. It is automated. Smart Draw is the only software that makes it possible for anyone to create presentation-quality visuals in minutes. Instead of forcing you to draw visuals manually, it draws for you, ensuring a professional result every time.
2. It is comprehensive (complete). Smart Draw is the only software that allows you to create every kind of visual-more than 70 different types-including flowcharts, timelines, org charts, mind maps, floor plans, marketing charts and more.
3. It is integrated. Smart Draw works seamlessly (effortlessly) with Microsoft Office.

Create a drawing in 3 basic steps
There are many kinds of Visio drawings, but you can use the three basic steps to create nearly all of them:

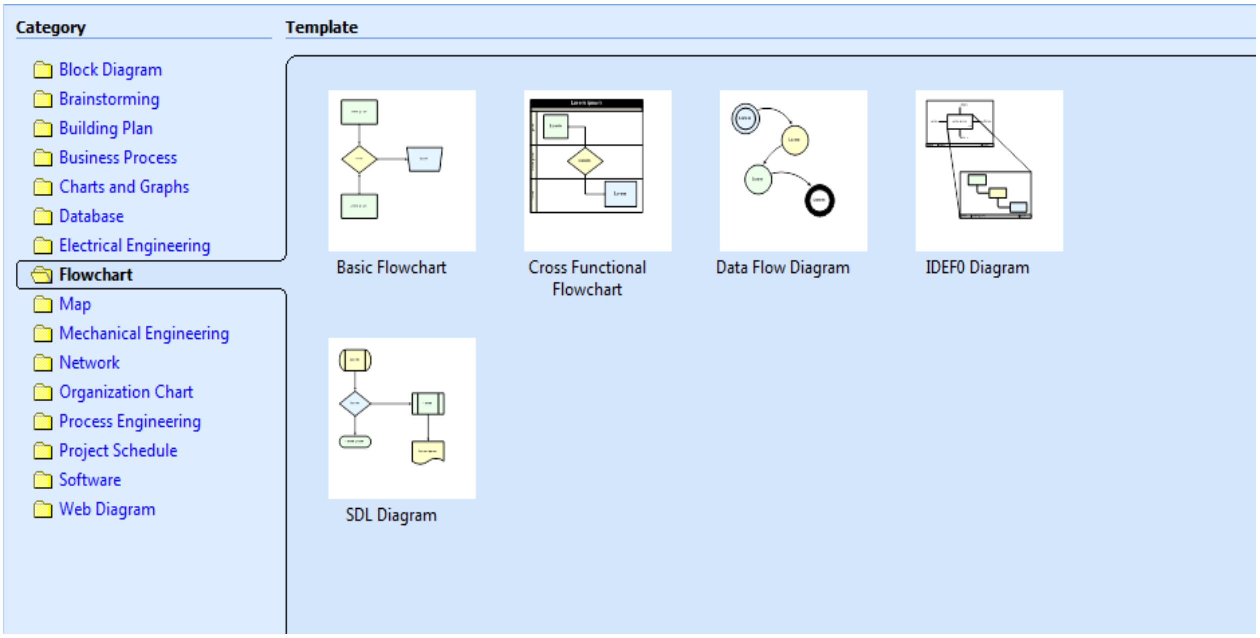
- Choose and open a template.
- Drag and connect shapes.
- Add text to shapes.

The following steps show how to create a basic flowchart.

Step 1: Choose and open a template

- Start Visio 2007.
- In the **Template Categories** list:
  * Click **Flowchart** to Design flowchart.
  * Click **Basic diagram** to Design the ERD for a database.
  * Click **software** to develop data flow diagram, and so on.

Example: To create a flowchart, in the **Flowchart** window, under **Featured Templates**, double-click **Basic Flowchart**.



When you open a template, the shapes you need open with it, in collections called stencils. The stencils that open with the **Basic Flowchart** template are called **Arrow Shapes**, **Backgrounds**, and **Basic Flowchart Shapes**.

What are Visio shapes, stencils, and templates?

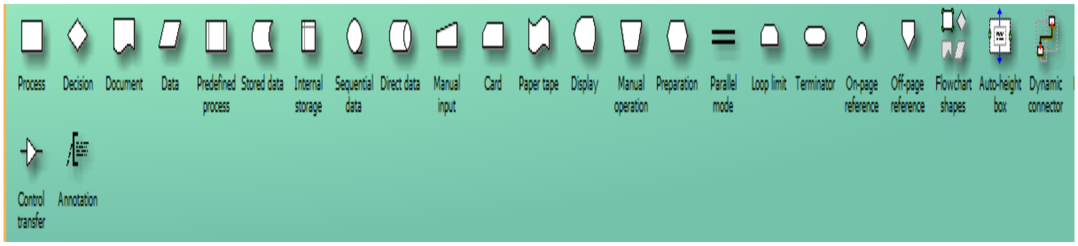
. Visio shapes, stencils, and templates can make Visio much easier to use.

- **Shapes**

Visio shapes are ready-made images that you drag onto your drawing page.

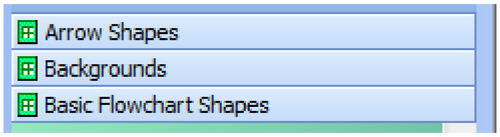
. Visio shapes are the building blocks of your drawing.

. The shapes can be a collection of shapes that you need to create a particular kind of diagram.



- **Stencils**

Visio stencils hold collections of shapes. The shapes in each stencil have something in common. The **Basic Flowchart Shapes** stencil contains common flowchart shapes, and the **Backgrounds** stencil contains a variety of backgrounds. You can even create your own stencil of favorite shapes.

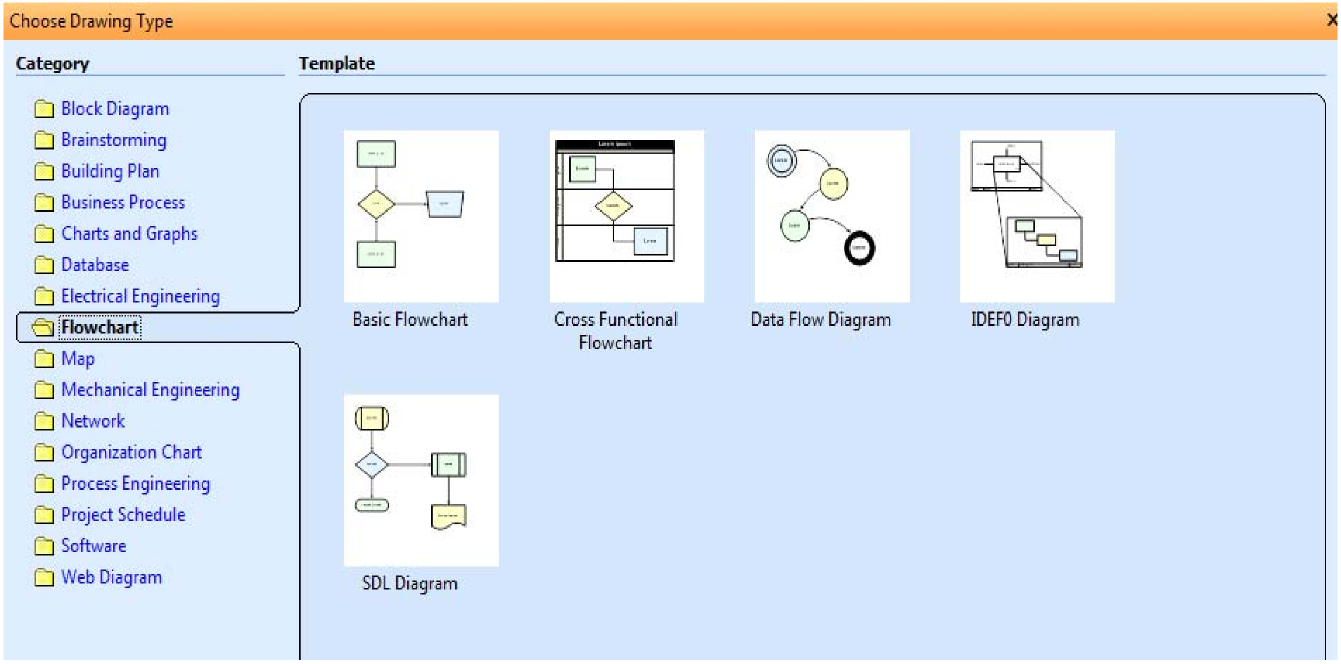


- **Templates**

A Visio template combines a blank drawing page with any combination of the following:

. Stencils full of the shapes that are needed to create a particular kind of drawing.

. Each template opens with the stencils that you need to create a particular kind of drawing.



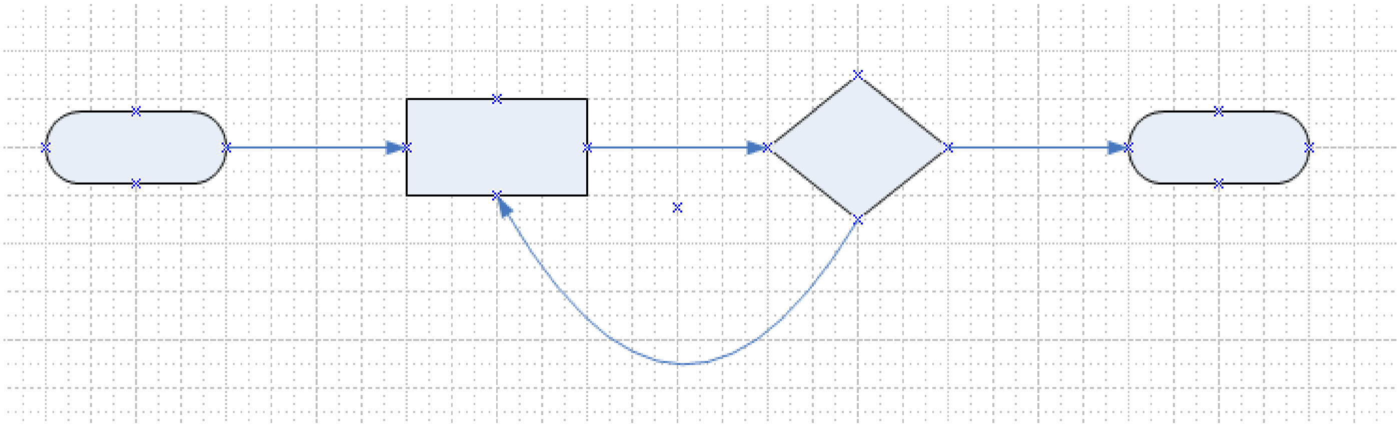
Step 2: Drag and connect shapes

To create your drawing, all you need to do is drag shapes from stencils onto the blank drawing page and connect them to one another.

i. Drag the selected shape from the **Basic Flowchart Shapes** stencil onto the drawing page, and then release the mouse button.

ii. Drag the connector from the **Basic flowchart Shape** stencil onto the drawing page, and then put the connector between the two shapes and connect them.

iii. Continue to build your drawing by repeating steps i-ii.

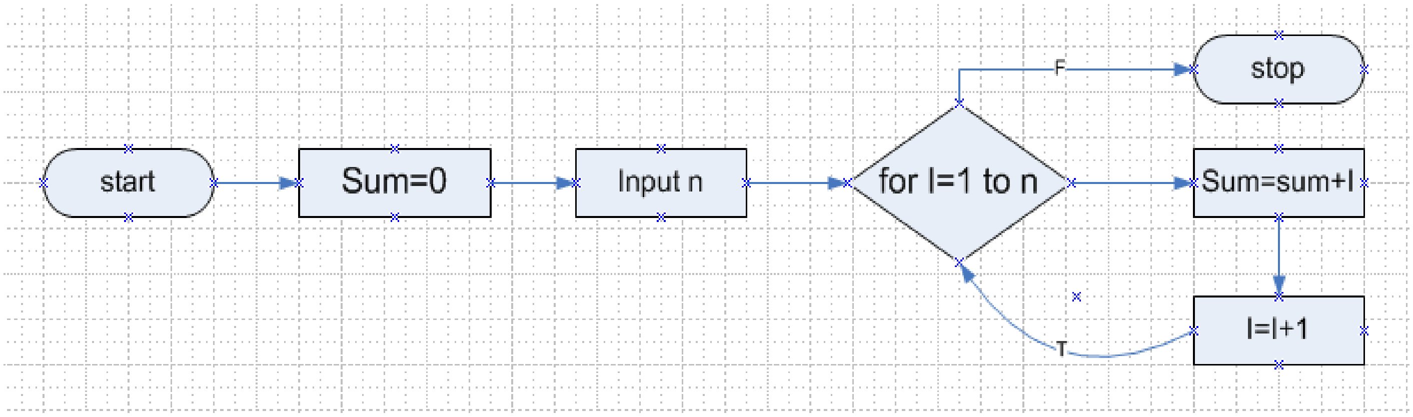


Step 3: Add text directly to a shape

. Double-click the shape.

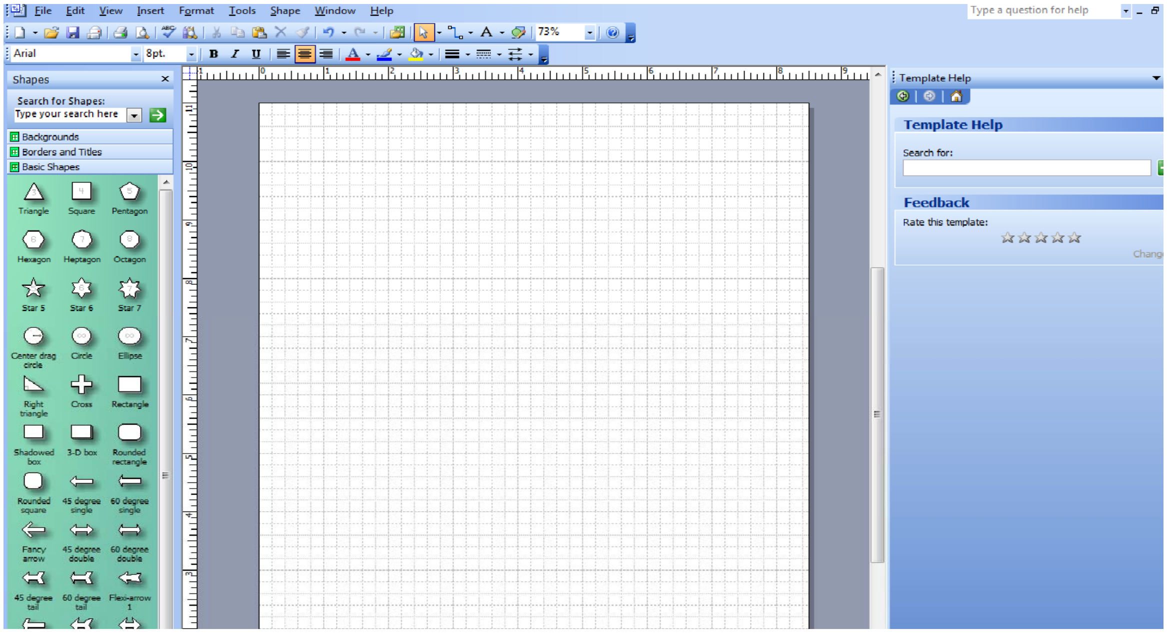
. Start typing.

. When you finish typing, click on a blank area of the drawing page.



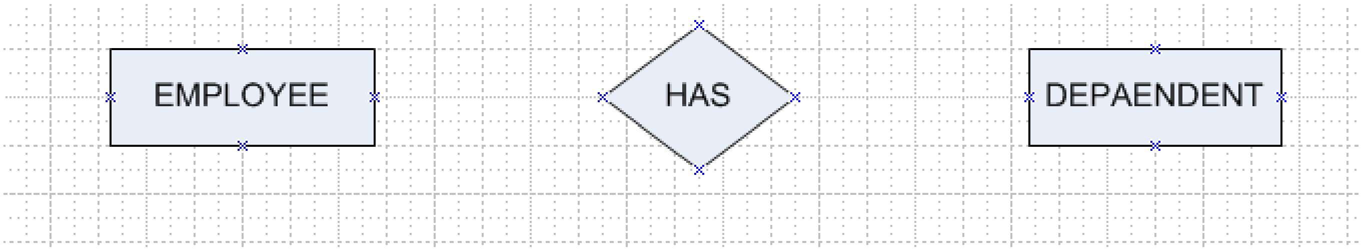
- ## **How to create the ER Diagram of a database using Visio software?**

  - Open the Visio program
  - File → new → choosing drawing type
  - Double click on **Block Diagram**
  - Click on **Basic Diagram**



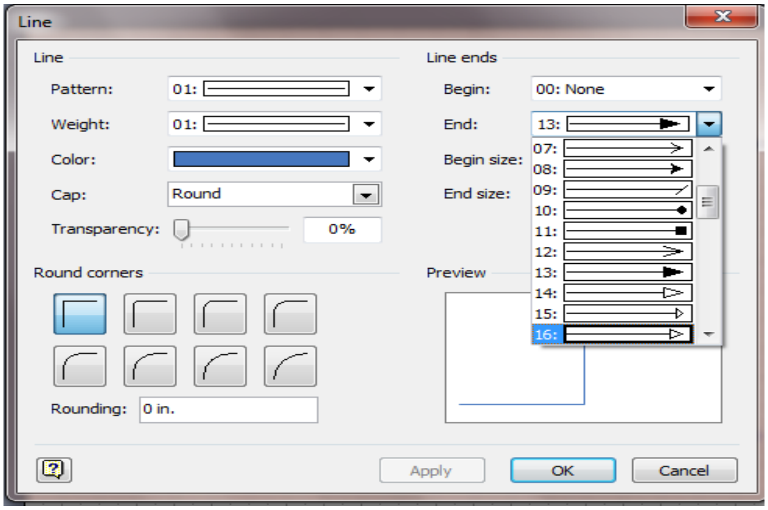
If you need another basic shape like diamond shape , you can click the find shape button on the top tool bar and search for a shape.

Click and drag the rectangular shape for Entities and the diamond shape for Relationships. You can add text to the middle of the shape by double clicking on it. Text can be customized.



To add the relationship lines, use the Dynamic Connector tool. This will automatically lock to a predefined hot spot on the shape, or you may create special hot spot on the shape to which your connect will stick.

To indicate the minimum and maximum cardinality you will need to edit the connector line. Right click on the connector and select Format – Line. You should see a pop up with lots of options. Select Line Ends (and either begin or end depending of which one you are changing). Scroll down the various options and you will see the notation we used in class.



Line

Line

Pattern: 01:

Weight: 01:

Color:

Cap: Round

Transparency: 0%

Round corners

Rounding: 0 in.

Line ends

Begin: 00: None

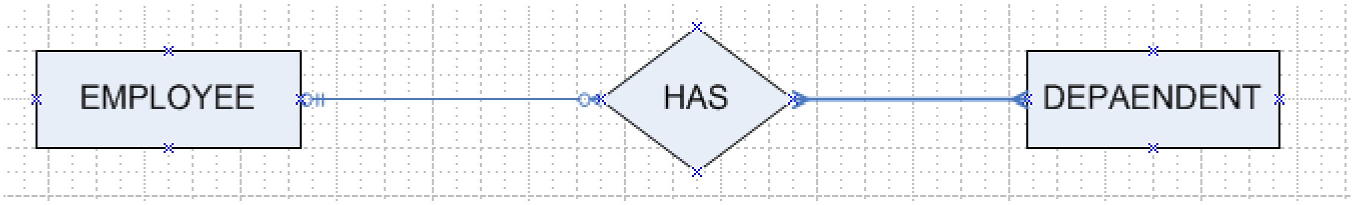
End: 13:

Begin size:

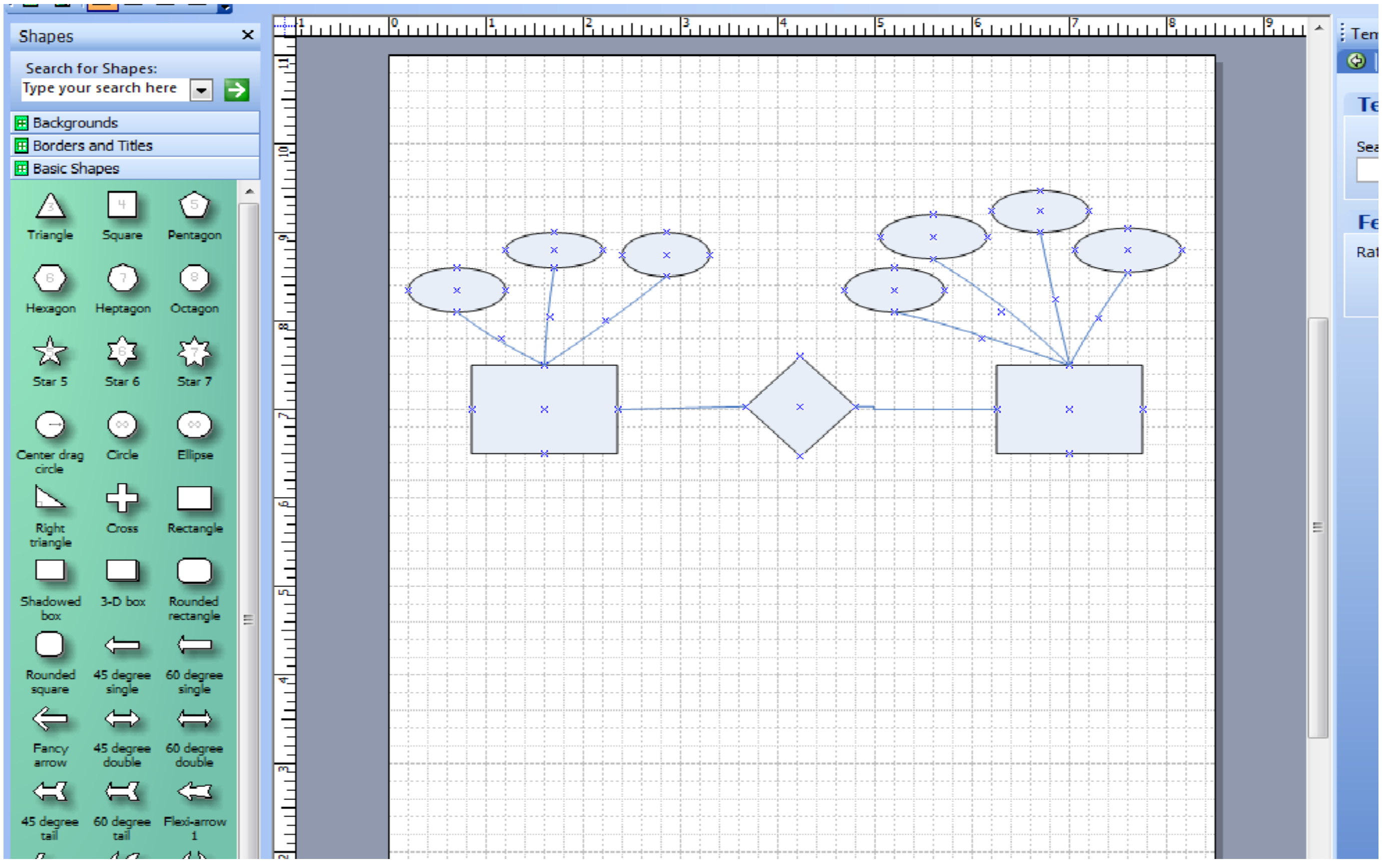
End size:

Preview

To increase the readability you also want to change the **Begin** and the **End Size** of the connector.



To add **attributes**, use the Oval shape and the connector again (line connector this time). You can format the Line around the Ovals to indicate derived attributes etc.



➢ ERD naming, conventions, and design issues

Specify structural constraints on relationships

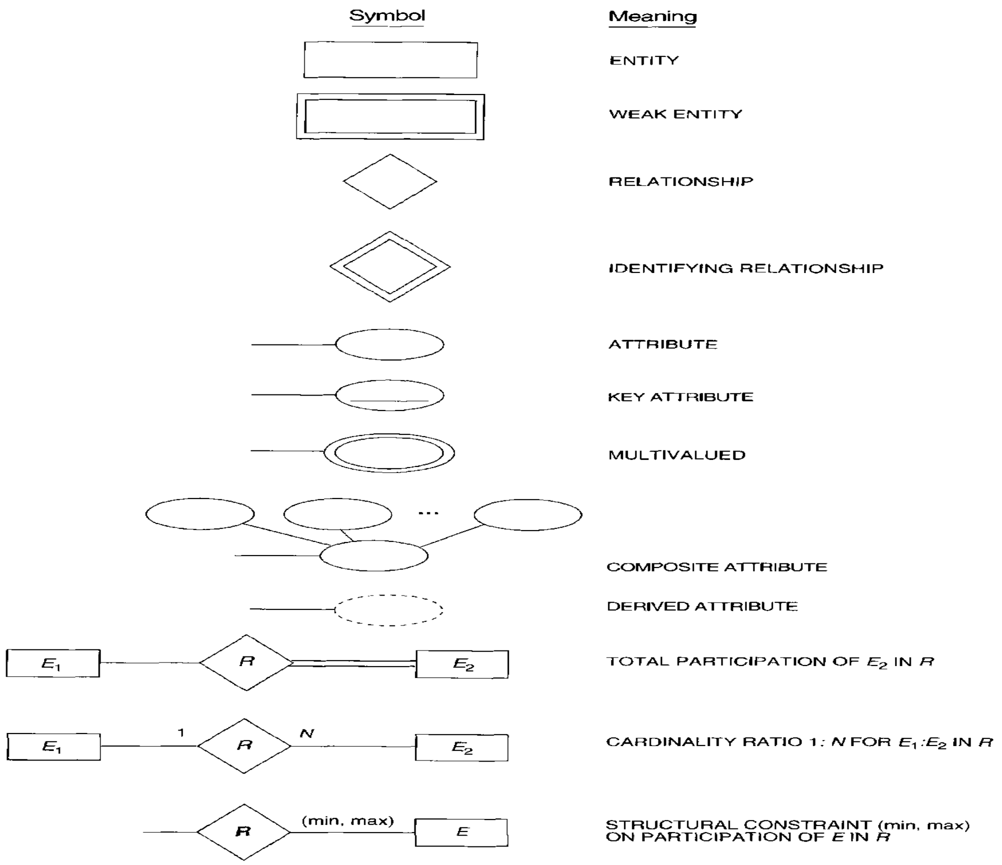
- Replaces cardinality ratio (1:1, 1:N, M:N) and single/double line notation for participation constraints

Entity naming convention

- Entity names should be a noun, singular (written in upper case).
- Where necessary, underscores should join the words.
- Where possible, avoid using the organization's name as part of the table name.
- For physical implementation, Entity Names can have a maximum of 44 characters.
- Names for the entities must be meaningful and must not conflict with other entity names.
- Where abbreviations are used, the full words must be obvious.
- Each entity name should be unique in the database

Attribute Naming convention

- Attributes' name initial character should be upper case followed by lower case.
- Where necessary, underscores should join the words.
- No two attributes of an entity type having the same attribute name, but Attributes of *different entities* can have the same name.



➢ **Notation for UML class diagrams**

What is UML?

The UML (Unified Modeling Language) is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the object of a software system. It captures decisions and understanding about systems that must be constructed.

The UML gives you a standard way to write a system's blueprints, covering conceptual things, such as business processes and system functions, as well as concrete things, such as classes written in a specific programming language, database schemas, and reusable software components."

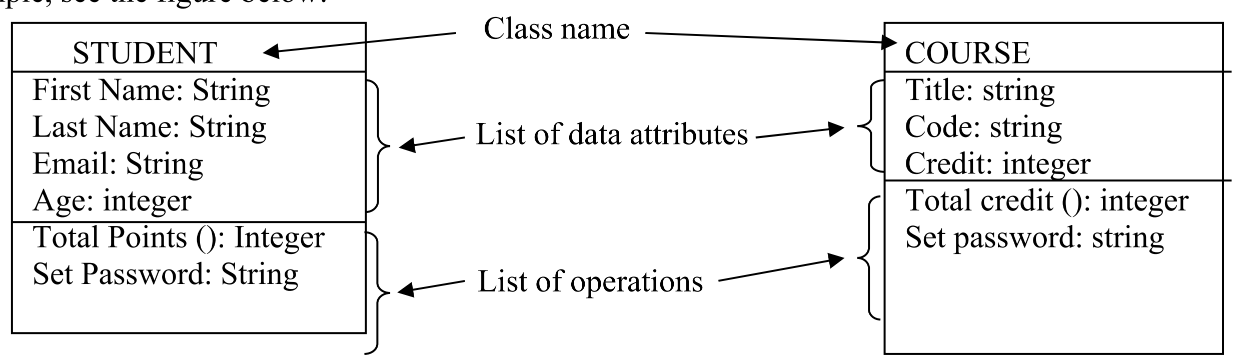
What is class?

- A class is a set of objects that share the same attributes, operations, and relationships. So, a class is similar to an entity type, only operations are added.
- It is the most important building block of any object-oriented system.

A class is symbolized by a rectangle with normally three "compartments" (sections) that contain:

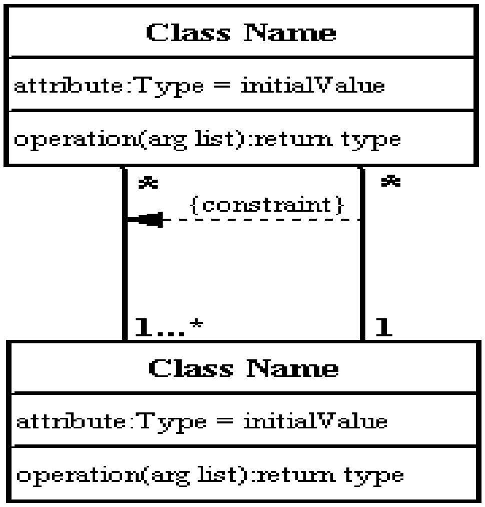
- Top section (class name).
- Middle section (attributes).
- Bottom section (operations that can be applied to individual objects).

For example, see the figure below:



- Relationships in UML are called **“associations”**.
- Cardinalities in UML are called multiplicities.

Place multiplicity notations (association names) above, on, or below the association line near the ends of an association in the form of (min, max). These symbols indicate the number of instances of one class linked to one instance of the other class.


Class Name
attribute:Type = initialValue
operation(arg list):return type
*
{constraint}
*
1...*
1
Class Name
attribute:Type = initialValue
operation(arg list):return type

An operation is a specification of a transformation or query that an object may be called to execute.
It has a name and a list of parameters.

1.2. Enhanced ER diagram and UML modeling

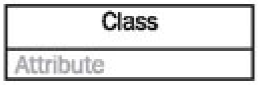
Enhanced ER diagram includes all modeling concepts of the ER model.
In addition, EER diagram includes:

- Subclasses and superclasses
- Specialization and generalization
- Category or union type
- Attribute, relationship, and inheritance

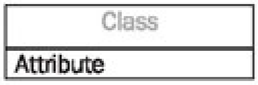
EER diagram is used to model concepts more accurately than the ER diagram

- **Represent specialization/Generalization and inheritance in UML class diagrams**

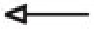
Specialization, generalization and inheritance in UML class diagrams include Basic notation for **superclasse, Subclasses, association and multiplicity.**
An UML class diagram includes the following elements:
Class - A class represents a relevant concept from the domain, a set of persons, objects, or ideas.



Attribute- An attribute of a class represents a characteristic of a class .



Generalization - *Generalization* is the process of extracting shared characteristics from two or more classes, and combining them into a generalized superclass.

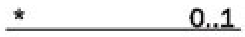


Association - An association represents a relationship between two classes:

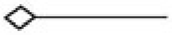
Works_on ▶

An association indicates that objects of one class have a relationship with objects of another class, in which this connection has a specifically defined meaning.

Multiplicity - A multiplicity allows for statements about the number of objects that are involved in an association:



Aggregation - An aggregation is a special case of an association (see above) meaning "consists of":



2.2 Documenting program scope and limits

Scope is an important concept in programming languages – one cannot read or write large programs without properly understanding the concept of scope. The **scope** of a variable in a program is the lines of code in the program where the variable can be accessed.

In other hand, the program SCOPE document is all about **realistic limits, boundaries, and how to achieve** the goals of the program and program limit document is a document which contains the different types of conditions in the program.

2.3 Documenting special routines or procedures

Identifying and revising references

- **Tables**

A **table** is a set of data elements (values) that is organized using a model of vertical columns (which are identified by their name) and horizontal rows, the cell being the unit where a row and column intersect.

A table has a specified number of columns, but can have any number of rows.

In computer programming, a table is a data structure used to organize information, just as it is on paper.

- **Inputs**

Input refers to anything that is given to a computer for processing. It may take various forms such as numbers for calculation, text to be printed, etc. The computer has to be told what to do with the input

It is usually connected with other terms, *e.g.*, input field, input variable, input parameter, input value, input signal, input port, input device and input file (file format).

- **Outputs**

Any information that has been processed and comes from a computer or computer device is considered **output**.

- It refers to the result of data processing such as a printed text.

Once the computer processes the input using the instructions given to it, it produces an output using an output device such as a screen or a printer.

- **Other program functionalities**

The **functionalities of a computer program** are in fact dictated (read out) by a specific and limited purpose.

In this, they are similar to ideas. That is why there may be a number of computer programs offering the same functionalities.

2.1 Applying Templates

A **template** is a tool for enforcing a standard layout and look and feel across multiple pages or within content regions. When you change a template, any pages or regions that are based on that template are automatically changed as well. Templates provide additional standardization controls, depending on the type you use.

- Templates must be used as applicable

LO3. Validate the design

3.1. Checking program flow, states or conditions

Testing aimed at ensuring that a product or system fulfils the defined user needs and specified requirements, under specified operating conditions.

3.1.1 Interfaces and compliance to design documentation requirement

An interface control drawing or interface control document describes the interface or interfaces to a system or subsystem. Ensuring the compliance of individual projects with the enterprise architecture is an essential aspect of architecture governance.

Program flow, states or conditions are checked for interfaces and compliance to design documentation requirements

3.2. Gaining feedback or input

Gaining feedback from appropriate person enables you to revise the designed system again to satisfy your customers. By getting customer feedback, you can make your customers happier.