HNDIT2022- Software Development

Higher National Diploma in Information Technology
Week 1

Module Details

Module Code	HNDIT2022	Module Title	Software Development
Credits	03	Lectures	15 Weeks
GPA/NGPA	GPA	Module Type	Compulsory
Semester	2		

Module Aims & Objectives

- This course is aimed at stressing the importance of good design, documentation and usability, emphasizing skills in problem solving and algorithm specification rather than just writing syntactically correct code and introducing a systematic approach to algorithm development which will assist in subsequent modules.

Learning Outcomes

- Distinguish between Systems Software and Application Software
- Be able to develop and understand algorithms
- Develop competence in the techniques of systematic problem analysis, program construction and documentation
- Gain an understanding of the basic concepts of good user-interface design
- Design algorithmic solutions for simple case studies

Assessment Weight

Type	Activity	Weight
Continuous Assessment	Online Quizzes and Assignment	20%
	Mini Project Presentation	20%
End of Semester Examination	Question Paper	60%

Reference

- *Internet Resources*
  - *Advanced Technological Institute library website*
- *Text Books*
  - *Bhargava A.Y., Grokking Algorithms, Manning Publications, 2015978-1617292231*

HNDIT2022- Software Development

Week 1: Fundamental Concepts of the Programming Process

Topics Covered

- What is a program?
- What is Computer Programming?
- Software Applications
- Fundamental Concepts of Programming
- Concepts of an Algorithm
- Development and semi-formal specification of algorithms, based on simplified computer model

What is a program?

- A computer program is a sequence of instructions written using a Computer Programming Language to perform a specified task by the computer.
- A computer program is also called a **computer software**, which can range from two lines to millions of lines of instructions.
- Computer program instructions are also called program source code and **computer programming** is also called **program coding**.

What is Computer Programming?

- Computer programming is the process of **designing** and **writing computer programs**.
- A programming language is a **computer language** that is used by **programmers (developers) to communicate with computers**. It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a specific task.

Software Applications

1. System software: such as compilers, editors, file management utilities
2. Application software**:** stand-alone programs for specific needs.
3. Engineering/scientific software: Characterized by "number crunching"algorithms. such as automotive stress analysis, molecular biology, orbital dynamics etc.

4. Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
5. Product-line software focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
6. WebApps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.
7. AI software uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition game playing

System Software	Application Software
System Software maintains the system resources and gives the path for application software to run.	Application software is built for specific tasks.
Low-level languages are used to write the system software.	While high-level languages are used to write the application software.
It is general-purpose software. Software which is designed to control, integrate and manage the individual hardware components and application software is known as system software.	It is a specific purpose software. Set of computer programs installed in the user's system and designed to perform a specific task is known as the application software.
System software runs independently.	Application software is dependent on system software because they need platform for its functioning.
System Software programming is more complex than application software.	Application software programming is simpler in comparison to system software.
Example: System software is an operating system, etc.	Example: Application software is Photoshop, VLC player, etc.

Basic Fundamental Concepts of Programming

- The basic concepts of programming are similar across languages. Some of these concepts include:
  - Variable Declaration
  - Basic Syntax
  - Data Type and Structures
  - Flow Control Structures (Conditionals and loops)
  - Functional Programming
  - Object-Oriented Programming Debugging
  - IDEs and Coding Environments

Variable Declaration

- **Variables** are containers for storing data values, a memory location for a data type. Variables are created using a declaration or keyword that varies across languages.
- Variable names are usually alphanumeric, that is, they contain a-z and 0-9. They can also include special characters like underscore or the dollar sign.
- Variables can hold values of any data type supported by the programming language. This value may change during program execution.

Basic Syntax

- Every programming language has its syntax.
- Syntax refers to the set of rules that define the structure of a language. It is almost impossible to read or understand a programming language without its syntax.

Data Type and Structures

- Data types refer to the classification of data. The most common data types include:
  - String
  - Boolean (true or false)
  - Numbers, which includes integers (whole numbers from 1) and floating-point numbers (decimal-base)
  - Characters (includes single alphabets or numbers)
  - Arrays (a collection of data, usually of the same data type)

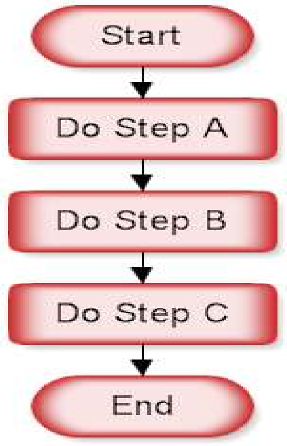
- A **Data Structure** is a collection of data values. These structures include operations that can be applied to that data. Data structures are important in computer programming for organizing, managing, and storing data quickly and efficiently.
- Some common types of data structures include:
  - Stacks
  - Heaps
  - Trees
  - Linked lists
  - Queues
  - Arrays
  - Tables
  - Graphs

Flow Control Structures

- **Flow Control Structures** are the fundamental components of computer programs. They are commands that allow a program to "decide" to take one direction or another.
- There are three basic types of control structures: sequential, selection, and iteration.

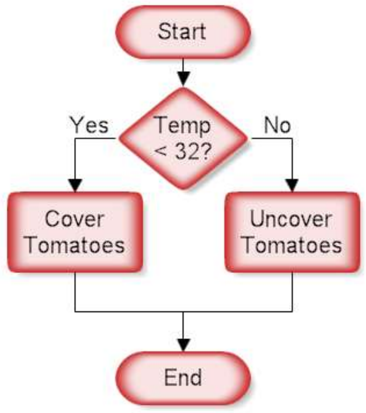
- **Sequential**

- The most basic control flow is **sequential control flow**. It involves the execution of code statements one after the other. A real-world example is following a cooking recipe.

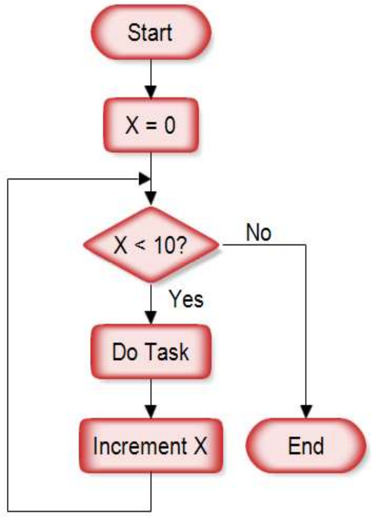


- **Selection (Conditionals)**

  - The basic premise of **selection flow control** is, the computer decides what action to perform based on the result of a test or condition equaling true or false.



- **Iteration (Loops)**
  - A **loop** is a programming structure that allows a statement or block of code to be run repeatedly until a specified condition is no longer true (will return Boolean, true or false).
  - It is one of the most powerful and fundamental programming concepts.



Functional Programming

- **Functions** are containers that take in a set of inputs and return an output. It is not required for a function to return a value. Pure functions will always give the same result for the same set of inputs.
- Functional Programming is a straightforward method of building software that involves using pure functions. This method eliminates the occurrence of data mutation or side effects.

Object Oriented Programming

- Object-Oriented Programming (OOP) is a programming concept that revolves around ‘objects’ and ‘methods’.
- There are four principles of OOP:
  - Inheritance
  - Polymorphism
  - Abstraction
  - Encapsulation

Debugging

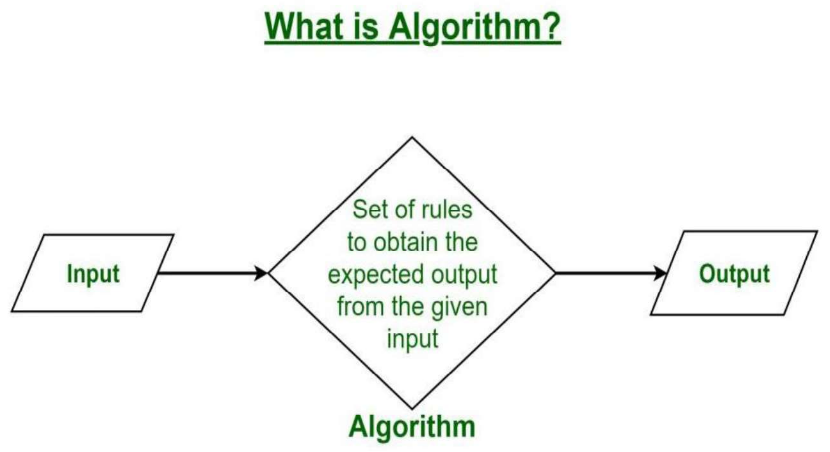
- **Debugging** is a crucial skill. It involves detecting and removing existing and potential errors, defects, or 'loopholes' in one's code.

IDEs and Coding Environments

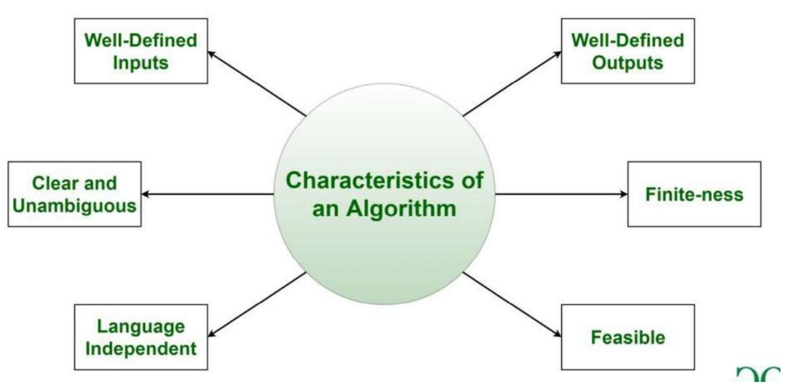
- IDE stands for **Integrated Development Environment** – they are applications programmers use to write code and organize text groups. It increases a programmer's efficiency and productivity, and has added features like code completion, code compilation, debugging, syntax highlighting, etc.
- Some common examples of IDE's are:
  - Visual Studio code
  - IntelliJ IDEA
  - NetBeans
  - Eclipse

What is an Algorithm?

- An algorithm is a set of commands that must be followed for a computer to perform calculations or other problem-solving operations.
- According to its formal definition, an algorithm is a finite set of instructions carried out in a specific order to perform a particular task.
- It is not the entire program or code; it is simple logic to a problem represented as an informal description in the form of a flowchart or pseudocode.


What is Algorithm?
Input
Set of rules to obtain the expected output from the given input
Output
Algorithm

Characteristics of an algorithm



- **Well-defined Inputs:** An algorithm requires some input values. An algorithm can be given a value other than 0 as input.
- **Well-defined Outputs:** At the end of an algorithm, you will have one or more outcomes.
- **Unambiguity:** A perfect algorithm is defined as unambiguous, which means that its instructions should be clear and straightforward.

- **Finiteness:** An algorithm must be finite. Finiteness in this context means that the algorithm should have a limited number of instructions, i.e., the instructions should be countable.
- **Effectiveness:** Because each instruction in an algorithm affects the overall process, it should be adequate.
- **Language independence:** An algorithm must be language-independent, which means that its instructions can be implemented in any language and produce the same results.

Examples

- Problem: Create an algorithm that multiplies two numbers and displays the output.
- Method 1

```
Step 1 − Start
Step 2 − declare three integers x, y & z
Step 3 − define values of x & y
Step 4 − multiply values of x & y
Step 5 − store result of step 4 to z
Step 6 − print z
Step 7 − Stop
```

- Method 2

```
Step 1 – Start mul
Step 2 – get values of x & y
Step 3 – z ← x * y
Step 4 – display z
Step 5 – Stop
```

- In algorithm design and analysis, the second method is typically used to describe an algorithm. It allows the analyst to analyze the algorithm while ignoring all unwanted definitions easily. They can see which operations are being used and how the process is progressing.
- It is optional to write step numbers.

Activity

- Write the algorithms to following scenarios.
  - ➢Following a recipe
  - ➢Finding a book in the library
  - ➢Find the largest number among three numbers
  - ➢Drive to your college from the nearest town

Thank you..

HNDIT2022- Software Development

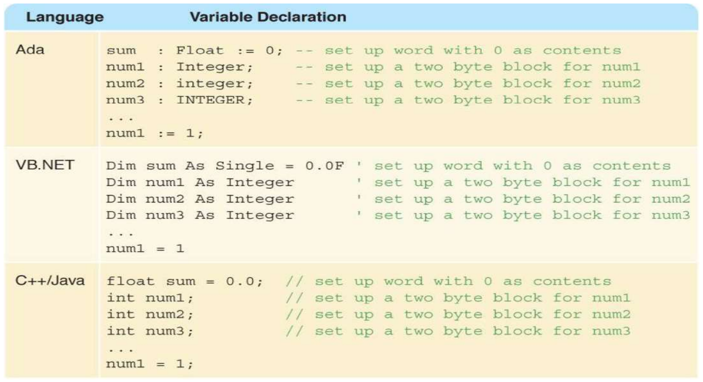
Week 1: Fundamental Concepts of the Programming Process – Part 2

Programming

- We have seen various examples of programming languages.
- C was an example of a procedural language
- Imperative or procedural model
  - Program executes a sequence of instructions to accomplish a task
  - FORTRAN, COBOL, BASIC, C, Pascal, Ada, and C++

Other languages

- Let's look back at other models for programming languages:



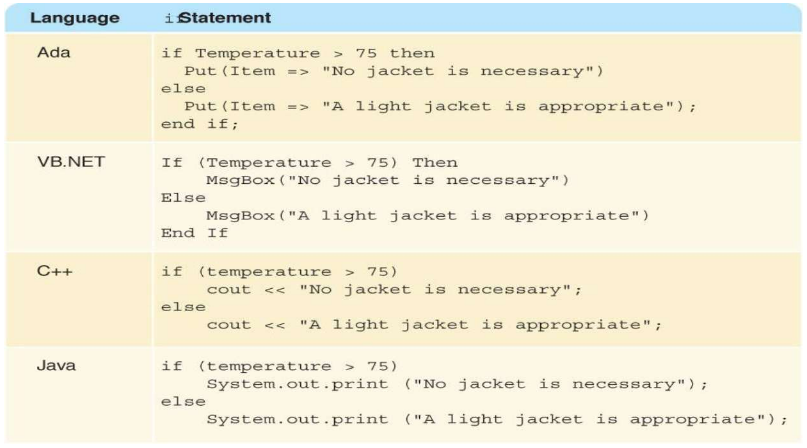
Language	Variable Declaration

Ada	`sum   : Float := 0; -- set up word with 0 as contents`
	`num1  : Integer;     -- set up a two byte block for num1`
	`num2  : integer;     -- set up a two byte block for num2`
	`num3  : INTEGER;     -- set up a two byte block for num3`
	`...`
	`num1  := 1;`

VB.NET	`Dim sum As Single = 0.0F ' set up word with 0 as contents`
	`Dim num1 As Integer       ' set up a two byte block for num1`
	`Dim num2 As Integer       ' set up a two byte block for num2`
	`Dim num3 As Integer       ' set up a two byte block for num3`
	`...`
	`num1 = 1`

C++/Java	`float sum = 0.0;   // set up word with 0 as contents`
	`int num1;          // set up a two byte block for num1`
	`int num2;          // set up a two byte block for num2`
	`int num3;          // set up a two byte block for num3`
	`...`
	`num1 = 1;`

If statements



Language	i Statement

Ada	`if Temperature > 75 then`
	`  Put(Item => "No jacket is necessary")`
	`else`
	`  Put(Item => "A light jacket is appropriate");`
	`end if;`

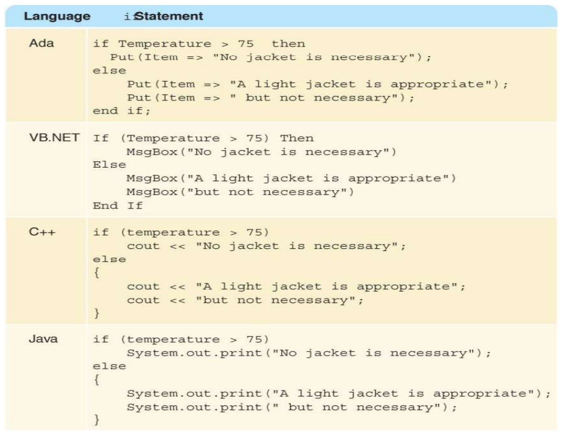
VB.NET	`If (Temperature > 75) Then`
	`    MsgBox("No jacket is necessary")`
	`Else`
	`    MsgBox("A light jacket is appropriate")`
	`End If`

C++	`if (temperature > 75)`
	`    cout << "No jacket is necessary";`
	`else`
	`    cout << "A light jacket is appropriate";`

Java	`if (temperature > 75)`
	`    System.out.print ("No jacket is necessary");`
	`else`
	`    System.out.print ("A light jacket is appropriate");`

Blocks of code

- How did we do if or while statements in C with multiple lines of code?



Language	i Statement

Ada	`if Temperature > 75 then`
	`  Put(Item => "No jacket is necessary");`
	`else`
	`    Put(Item => "A light jacket is appropriate");`
	`    Put(Item => " but not necessary");`
	`end if;`

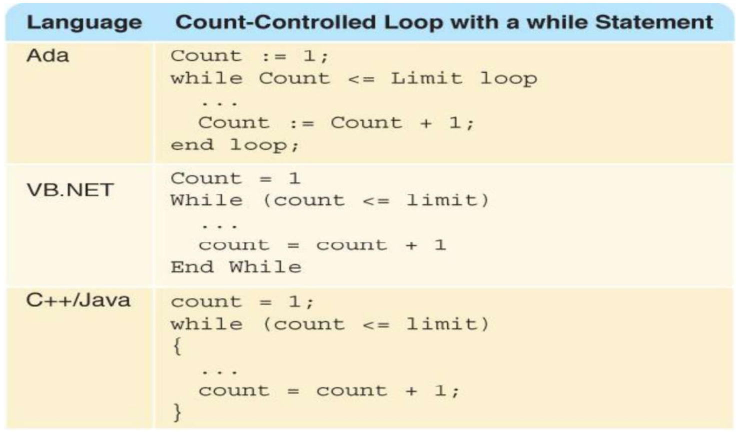
VB.NET	`If (Temperature > 75) Then`
	`    MsgBox("No jacket is necessary")`
	`Else`
	`    MsgBox("A light jacket is appropriate")`
	`    MsgBox("but not necessary")`
	`End If`

C++	`if (temperature > 75)`
	`    cout << "No jacket is necessary";`
	`else`
	`{`
	`    cout << "A light jacket is appropriate";`
	`    cout << "but not necessary";`
	`}`

Java	`if (temperature > 75)`
	`    System.out.print("No jacket is necessary");`
	`else`
	`{`
	`    System.out.print("A light jacket is appropriate");`
	`    System.out.print(" but not necessary");`
	`}`

Loops

- How did we do while loops in C?



Language	Count-Controlled Loop with a while Statement

Ada	`Count := 1;`
	`while Count <= Limit loop`
	`  ...`
	`  Count := Count + 1;`
	`end loop;`

VB.NET	`Count = 1`
	`While (count <= limit)`
	`  ...`
	`  count = count + 1`
	`End While`

C++/Java	`count = 1;`
	`while (count <= limit)`
	`{`
	`  ...`
	`  count = count + 1;`
	`}`

Write a program to count the length of a message

```
#include <stdio.h>
main(void)
{
        char ch;
        int length = 0;

        //Prompt the user for a message

        //Read the first character of the message

        //while loop to count how the message is

        //print length of message

        return 0;
}
```

- But remember, even comments are different in various languages!
- For example, in python, comments are put after % sign instead of //
- We need a way to describe a program which is independent of a specific language.

Algorithms

A set of **unambiguous** instructions for solving a problem or sub problem in a **finite** amount of **time** using a finite amount of ***data***

Why must instructions be unambiguous?

Why must time and data be finite?

Pseudocode

A way of expressing algorithms that uses a mixture of *English phrases* and *indention* to make the steps in the solution explicit

There are no grammar rules in pseudocode

Pseudocode is not case sensitive

Pseudocode for Complete Computer Solution

```
Write "Enter the new base"
Read newBase
Write "Enter the number to be converted"
Read decimalNumber
Set quotient to 1
While (quotient is not zero)
        Set quotient to decimalNumber DIV newBase
        Set remainder to decimalNumber REM newBase
        Make the remainder the next digit to the left in the answer
        Set decimalNumber to quotient
Write "The answer is "
Write answer
```

Pseudocode Functionality

Variables

Names of places to store values

quotient, decimalNumber, newBase

Assignment

Storing the value of an expression into a variable

```
Set quotient to 64
quotient <-- 64
quotient <-- 6 * 10 + 4
```

Pseudocode Functionality

Output

Printing a value on an output device

Write, Print

Input

Getting values from the outside word and storing them into variables

Get, Read

Pseudocode Functionality

Selection

Making a choice to execute or skip a statement (or group of statements)

```
Read number
If (number < 0)
    Write number + " is less than zero."
```

or

```
Write "Enter a positive number."
Read number
If (number < 0)
    Write number + " is less than zero."
    Write "You didn't follow instructions."
```

Pseudocode Functionality

Selection

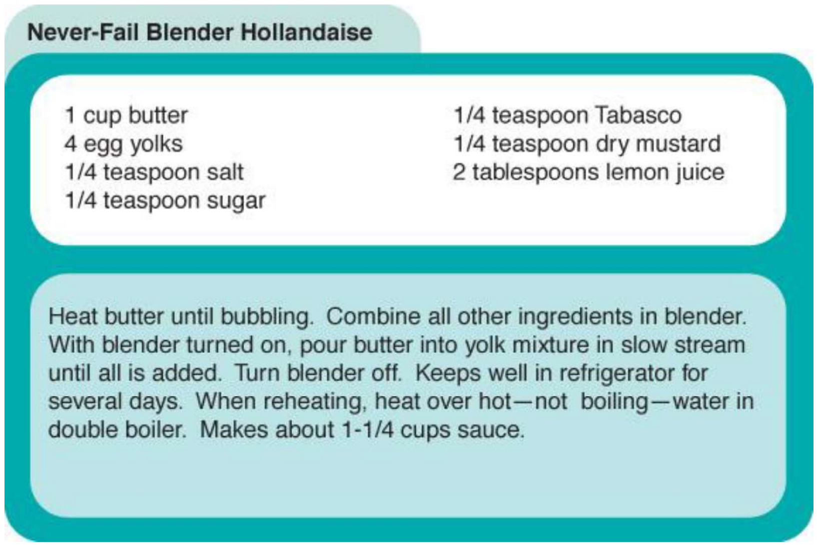
Choose to execute one statement (or group of statements) or another statement (or group of statements)

```
If ( age < 12 )
    Write "Pay children's rate"
    Write "You get a free box of popcorn"
else If ( age < 65 )
    Write "Pay regular rate"
else
    Write "Pay senior citizens rate"
```

Pseudocode Example

```
Write "How many pairs of values are to be entered?"
Read numberOfPairs
Set numberRead to 0
While (numberRead < numberOfPairs)
    Write "Enter two values separated by a blank; press return"
    Read number1
    Read number2
    If (number1 < number2)
        Print number1 + " " + number2
    Else
        Print number2 + " " number1
    Increment numberRead
```

Following an Algorithm



Never-Fail Blender Hollandaise

1 cup butter	1/4 teaspoon Tabasco
4 egg yolks	1/4 teaspoon dry mustard
1/4 teaspoon salt	2 tablespoons lemon juice
1/4 teaspoon sugar	

Heat butter until bubbling. Combine all other ingredients in blender. With blender turned on, pour butter into yolk mixture in slow stream until all is added. Turn blender off. Keeps well in refrigerator for several days. When reheating, heat over hot—not boiling—water in double boiler. Makes about 1-1/4 cups sauce.

Developing an Algorithm

Two methodologies used to develop computer solutions to a problem

- Top-down design focuses on the **tasks** to be done
- Object-oriented design focuses on the **data** involved in the solution

Top-Down Design

Top-Down Design

Problem-solving technique in which the problem is divided into sub problems; the process is applied to each sub problem

Modules

Self-contained collection of steps, that solve a problem or sub problem

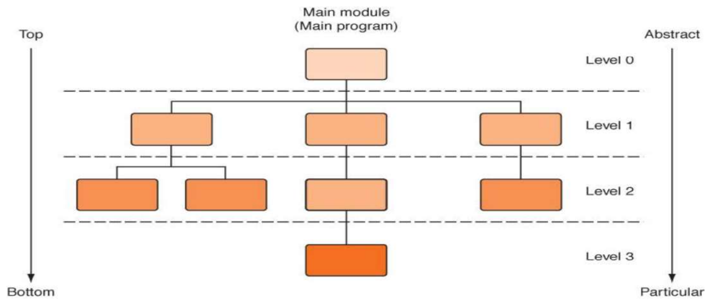
Abstract Step

An algorithmic step containing unspecified details

Concrete Step

An algorithm step in which all details are specified

Top-Down Design

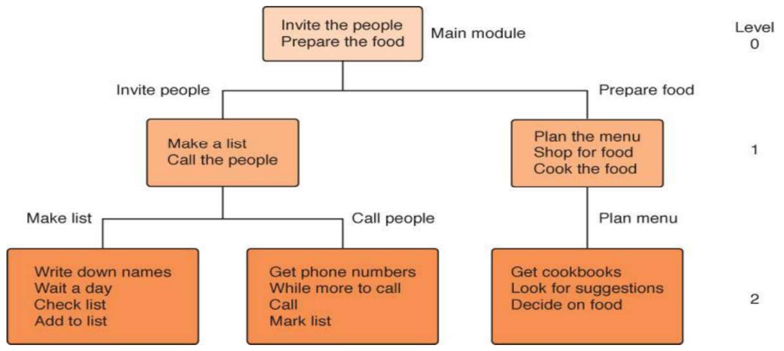


Process continues for as many levels as it takes to make every step concrete

Name of (sub)problem at one level becomes a module at next lower level

A General Example

Planning a large party



Object-Oriented Design

Object-oriented Design

A problem-solving methodology that produces a solution to a problem in terms of self-contained entities called *objects*

Object

A thing or entity that makes sense within the context of the problem

For example, a *student*, a *car*, *time*, *date*

Object-Oriented Design

An analogy: You and your friend fix dinner

Objects: you, friend, dinner

Class: you and friend are people

People have name, eye color, …

People can shop, cook, …

Instance of a class: you and friend are instances of class People, you each have your own name and eye color, you each can shop and cook

You collaborate to fix dinner

Object-Oriented Design

Class (or object class)

A description of a *group* of similar objects

Object (instance of a class)

A concrete example of the class

Classes contain fields that represent the

properties (name, eye color) and
behaviors (responsibilities) (shop, cook) of the class

Method

A named algorithm that defines behavior (shop, cook)

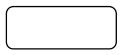
Flowcharts

- Flowcharts is a graph used to depict or show a step by step solution using **symbols** which represent a task.
- The symbols used consist of geometrical shapes that are connected by **flow lines**.
- It is an alternative to pseudocoding; whereas a pseudocode description is verbal, a flowchart is graphical in nature.

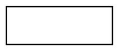
A Flowchart

- shows logic of an algorithm
- emphasizes individual steps and their interconnections
- e.g. control flow from one action to the next

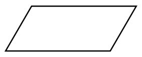
Flowchart Symbols



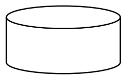
Terminal symbol - indicates the beginning and end points of an algorithm.



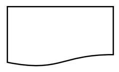
Process symbol - shows an instruction other than input, output or selection.



Input-output symbol - shows an input or an output operation.

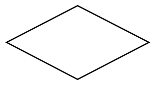


Disk storage I/O symbol - indicates input from or output to disk storage.

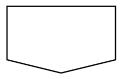


Printer output symbol - shows hardcopy printer output.

Flowchart Symbols cont…

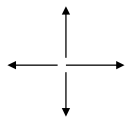


Selection symbol - shows a selection process for two-way selection.



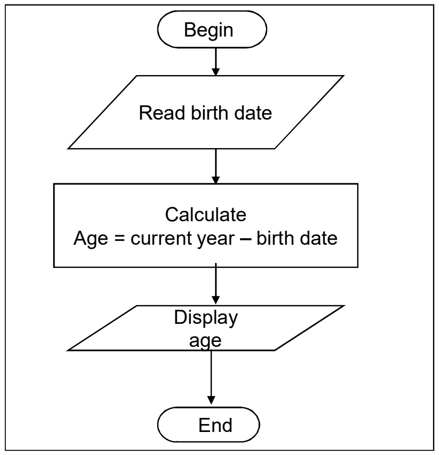
Off-page connector - provides continuation of a logical path on another page.

On-page connector - provides continuation of logical path at another point in the same page.

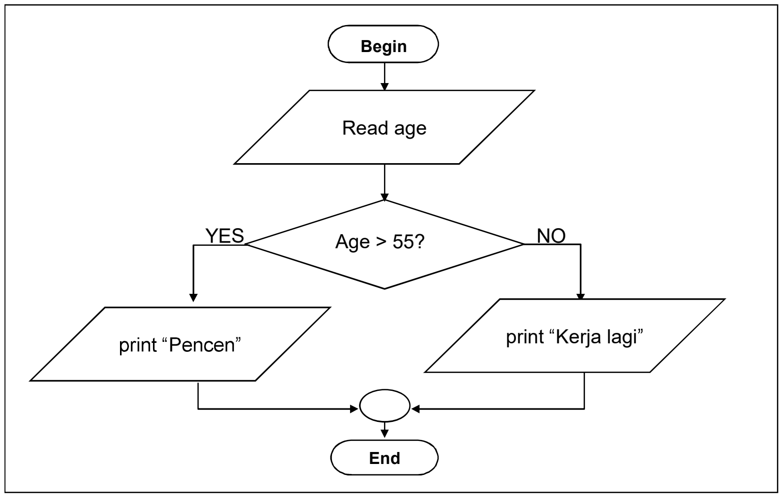


Flow lines - indicate the logical sequence of execution steps in the algorithm.

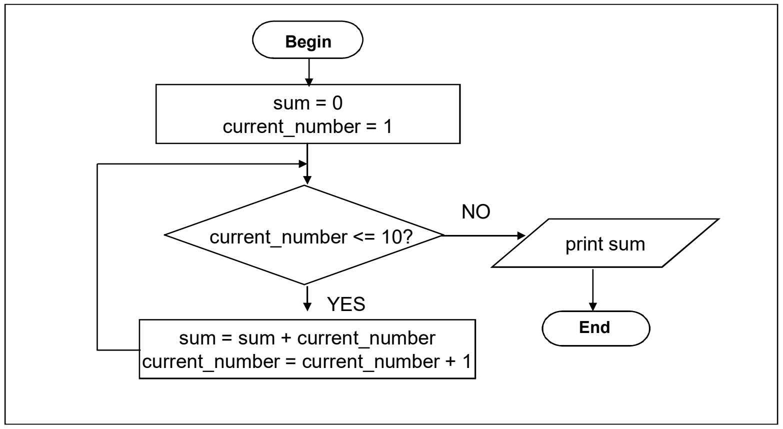
Flowchart – example 1



Flowchart – example 2



Flowchart – Example 3



Example 4

- Write an algorithm and draw a flowchart that will calculate the roots of a quadratic equation $ax^2 + bx + c = 0$
- Hint:$\Delta=(b^2 - 4ac)$,If $\Delta>=0$ then there are roots, otherwise no roots
- **d** = sqrt $(b^2 - 4ac)$, and the roots are: ***x1*** = (–*b* + *d*)/2*a* and ***x2*** = (–*b* – *d*)/2*a*

Example 5

- Write an algorithm and draw a flowchart to convert the length in feet to centimeter.

Pseudocode:

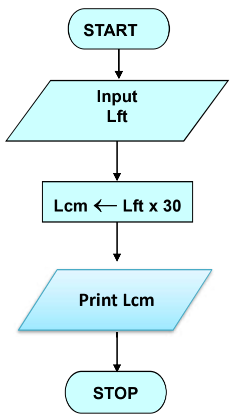
- *Input the length in feet (Lft)*
- *Calculate the length in cm (Lcm) by multiplying LFT with 30*
- *Print length in cm (LCM)*

Example 5

Algorithm

- Step 1: Input Lft
- Step 2: Lcm ← Lft x 30
- Step 3: Print Lcm

Flowchart



Example 6

Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.

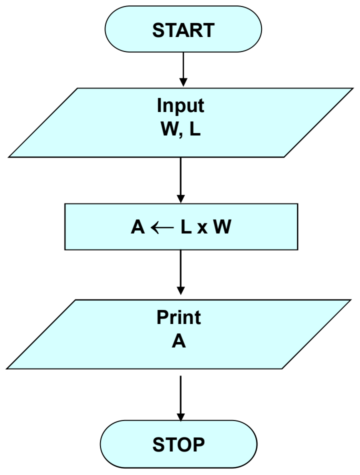
Pseudocode

- *Input the width (W) and Length (L) of a rectangle*
- *Calculate the area (A) by multiplying L with W*
- *Print A*

Example 6

Algorithm

- Step 1: Input W,L
- Step 2: A ← L x W
- Step 3: Print A



Exercises: Algorithm & Flowchart

1.) Create an algorithm and a flowchart that will accept/read two numbers and then display the bigger number.

Exercises: Algorithm & Flowchart

2.) Create an algorithm and a flowchart that will compute the area of a circle.

Exercises: Algorithm & Flowchart

3.) Create an algorithm and a flowchart that will compute the sum of two numbers. If the sum is below or equal to twenty, two numbers will be entered again. If the sum is above 20, it will display the sum.

Lab Activity: Algorithm & Flowchart

4) Create an algorithm and a flowchart that will output the largest number among the three numbers.

Self Study

1. Create an algorithm and a flowchart that will output for g.c.d.
2. Create an algorithm and a flowchart that will output the factorial of a given number.
3. Create an algorithm and a flowchart that will output the Fibonacci series up to a given number.
4. Create an algorithm and a flowchart that will output all the prime numbers between 2 numbers.

Thank you..

HNDIT2022- Software Development

Week 3: Phase-specific issues of software development – Part 1

Development Techniques

- Modular Programming
- Defensive Programming
- Recursion

Modular Programming

- A module is defined as a part of a software program that contains one or more routines. When we merge one or more modules, it makes up a program.
- Modular programming is defined as a software design technique that focuses on separating the program functionality into independent, interchangeable methods/modules. Each of them contains everything needed to execute only one aspect of functionality.
- **Examples of modular programming languages -** All the object-oriented programming languages like C++, Java, etc., are modular programming languages.

Defensive Programming

- Defensive programming is the creation of code for **computer software** designed to avoid problematic issues before they arise and make the product more stable.
- Testing is one of the most important aspects of defensive programming.
- There are two main categories of defensive programming:
  - Secure Programming
  - Offensive Programming

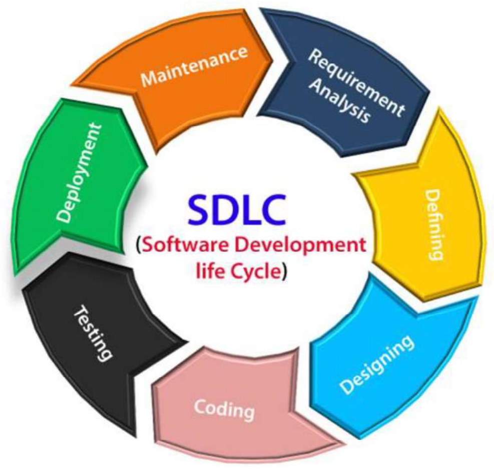
Recursion

- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.
- Using a recursive algorithm, certain problems can be solved quite easily.
- A recursive function solves a particular problem by calling a copy of itself and solving smaller subproblems of the original problems.

Software Development Life Cycle (SDLC)

- A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle.
- A life cycle model represents all the methods required to make a software product transit through its life cycle stages.
- It also captures the structure in which these methods are to be undertaken.

The stages of SDLC



- **Stage1: Planning and requirement analysis**
  - Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.
  - Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product. Before creating a product, a core understanding or knowledge of the product is very necessary.

- **Stage2: Defining Requirements**
  - Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.
  - This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

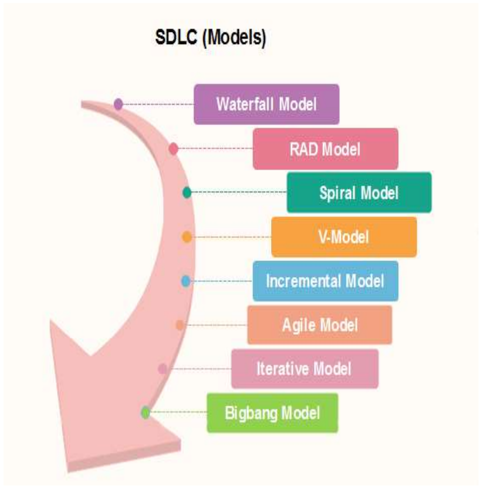
- **Stage3: Designing the Software**
  - The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.
- **Stage4: Developing the project**
  - In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

- **Stage5: Testing**
  - After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage.
  - During this stage, unit testing, integration testing, system testing, acceptance testing are done.
- **Stage6: Deployment**
  - Once the software is certified, and no bugs or errors are stated, then it is deployed.
  - Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.
  - After the software is deployed, then its maintenance begins.

- **Stage7: Maintenance**
  - Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.
  - This procedure where the care is taken for the developed product is known as maintenance.

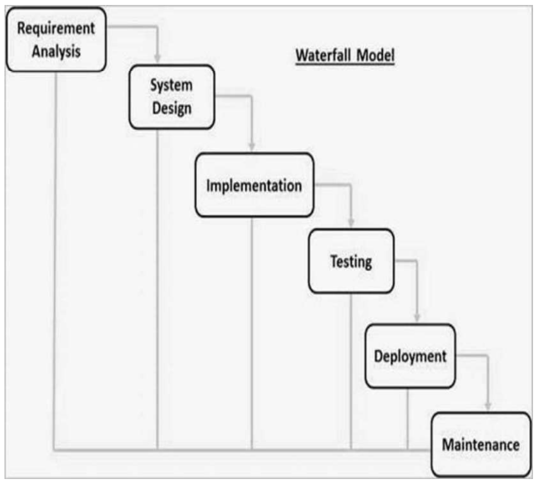
SDLC Models

- Software Development life cycle (SDLC) is a spiritual model used in project management that defines the stages include in an information system development project, from an initial feasibility study to the maintenance of the completed application.



Waterfall Model

- The waterfall model is a continuous software development model in which development is seen as flowing steadily downwards (like a waterfall) through the steps of requirements analysis, design, implementation, testing (validation), integration, and maintenance.



RAD Model

- RAD or Rapid Application Development process is an adoption of the waterfall model; it targets developing software in a short period. The RAD model is based on the concept that a better system can be developed in lesser time by using focus groups to gather system requirements.
  - Business Modeling
  - Data Modeling
  - Process Modeling
  - Application Generation
  - Testing and Turnover

Agile Model

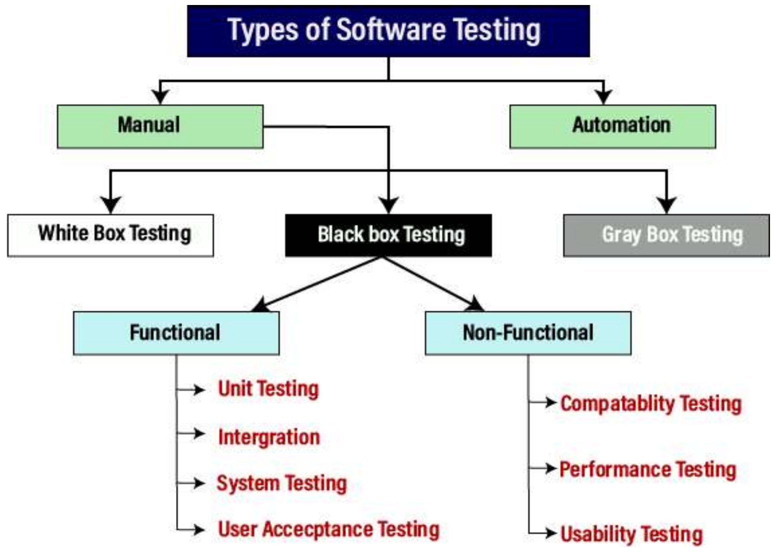
- Agile methodology is a practice which promotes continues interaction of development and testing during the SDLC process of any project.
- In the Agile method, the entire project is divided into small incremental builds.
- All of these builds are provided in iterations, and each iteration lasts from one to three weeks.
- Any agile software phase is characterized in a manner that addresses several key assumptions about the bulk of software projects:
- It is difficult to think in advance which software requirements will persist and which will change. It is equally difficult to predict how user priorities will change as the project proceeds.

Prototype Model

- The prototyping model starts with the requirements gathering. The developer and the user meet and define the purpose of the software, identify the needs, etc.
- A '**quick design**' is then created. This design focuses on those aspects of the software that will be visible to the user. It then leads to the development of a prototype. The customer then checks the prototype, and any modifications or changes that are needed are made to the prototype.
- Looping takes place in this step, and better versions of the prototype are created. These are continuously shown to the user so that any new changes can be updated in the prototype. This process continue until the customer is satisfied with the system. Once a user is satisfied, the prototype is converted to the actual system with all considerations for quality and security.

Software Testing

- Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.
- Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing. So, without testing software cannot be deployed to the end user.



Manual Testing

- The process of checking the functionality of an application as per the customer needs without taking any help of automation tools is known as manual testing.
- Manual testing can be further divided into three types of testing, which are as follows:
  - **White box testing**
  - **Black box testing**
  - **Gray box testing**

- White-box testing
  - The white box testing is done by Developer, where they check every line of a code before giving it to the Test Engineer. Since the code is visible for the Developer during the testing, that's why it is also known as White box testing.
- Black box testing
  - The black box testing is done by the Test Engineer, where they can check the functionality of an application or the software according to the customer /client's needs. In this, the code is not visible while performing the testing; that's why it is known as black-box testing.

- Gray Box testing
  - Gray box testing is a combination of white box and Black box testing. It can be performed by a person who knew both coding and testing. And if the single person performs white box, as well as black-box testing for the application, is known as Gray box testing.

Automation Testing

- Automation testing is a process of converting any manual test cases into the test scripts with the help of automation tools, or any programming language is known as automation testing.
- With the help of automation testing, we can enhance the speed of our test execution because here, we do not require any human efforts. We need to write a test script and execute those scripts.

Dry Run Testing

- **Dry run testing** is a static test and should be performed by the developer to mitigate the effects of a failure of the product - meaning before the end user gets the product and discovers it doesn't do what it says it will.
- In dry run testing, no hardware is used, but it is assumed that the programmer who is testing the code is aware of what each line of code is supposed to do and gives him or her the opportunity to make corrections to the code before it becomes an issue for the actual software.
- Basically, a dry run test consists of programmers manually reading their code line by line to find errors and fix them.

Test Cases

- A **Test Case** is a set of actions executed to verify a particular feature or functionality of your software application. A Test Case contains test steps, test data, pre-condition, post-condition developed for specific test scenario to verify any requirement.
- The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

Test Scenario Vs Test Case

- Test scenarios are rather vague and cover a wide range of possibilities. Testing is all about being very specific.
- Example 1
  - Test Scenario: Check Login Functionality there many possible test cases are:
    - Test Case 1: Check results on entering valid User Id & Password
    - Test Case 2: Check results on entering Invalid User ID & Password
    - Test Case 3: Check response when a User ID is Empty & Login Button is pressed, and many more

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
TU01	Check Customer Login with valid Data	1.Go to site http://demo.guru99.com 2.Enter UserId 3.Enter Password 4.Click Submit	Userid = guru99 Password = pass99	User should Login into an application	As Expected	Pass
TU02	Check Customer Login with invalid Data	1.Go to site http://demo.guru99.com 2.Enter UserId 3.Enter Password 4.Click Submit	Userid = guru99 Password = glass99	User should not Login into an application	As Expected	Pass

- Example 2
  - Let us say that we need to check an input field that can accept maximum of 10 characters.
  - While developing the test cases for the above scenario, the test cases are documented the following way. In the below example, the first case is a pass scenario while the second case is a FAIL.

Scenario	Test Step	Expected Result	Actual Outcome
Verify that the input field that can accept maximum of 10 characters	Login to application and key in 10 characters	Application should be able to accept all 10 characters.	Application accepts all 10 characters.
Verify that the input field that can accept maximum of 11 characters	Login to application and key in 11 characters	Application should NOT accept all 11 characters.	Application accepts all 10 characters.

Debugging

- It is a systematic process of spotting and fixing the number of bugs, or defects, in a piece of software so that the software is behaving as expected.
- Debugging is harder for complex systems in particular when various subsystems are tightly coupled as changes in one system or interface may cause bugs to emerge in another.
- Debugging is a developer activity and effective debugging is very important before testing begins to increase the quality of the system.
- Debugging will not give confidence that the system meets its requirements completely but testing gives confidence.

Thank you..

HNDIT2022- Software Development

Week 4: Phase-specific issues of software development – Part 2

Documentation Requirements

- General requirements of all software documentation
  - Should provide for communication among team members
  - Should act as an information repository to be used by maintenance engineers
  - Should provide enough information to management to allow them to perform all program management related activities
  - Should describe to users how to operate and administer the system

Documentation Requirements

- In all software projects some amount of documentation should be created prior to any code being written
  - Design docs, etc.
- Documentation should continue after the code has been completed
  - User's manuals, etc.
- The two main types of documentation created are *Process* and *Product* documents

Process Documentation

- Used to record and track the development process
  - Planning documentation
  - Cost, Schedule, Funding tracking
  - Schedules
  - Standards
  - Etc.
- This documentation is created to allow for successful management of a software product

Process Documentation

- Has a relatively short lifespan
- Only important to internal development process
  - Except in cases where the customer requires a view into this data
- Some items, such as papers that describe design decisions should be extracted and moved into the *product* documentation category when they become implemented

Product Documentation

- Describes the **delivered** product
- Must evolve with the development of the software product
- Two main categories:
  - System Documentation
  - User Documentation

Product Documentation

- System Documentation
  - Describes how the system works, but not how to operate it
- Examples:
  - Requirements Spec
  - Architectural Design
  - Detailed Design
  - Commented Source Code
    - Including output such as JavaDoc
  - Test Plans
    - Including test cases
  - V&V plan and results
  - List of Known Bugs

Product Documentation

- User Documentation has two main types
  - End User
  - System Administrator
- In some cases these are the same people
  - The target audience must be well understood!

Product Documentation

- There are five important areas that should be documented for a formal release of a software application
  - These do not necessarily each have to have their own document, but the topics should be covered thoroughly

1. Functional Description of the Software
2. Installation Instructions
3. Introductory Manual
4. Reference Manual
5. System Administrator’s Guide

Document Quality

- Providing thorough and professional documentation is important for any size product development team
- The problem is that many software professionals lack the writing skills to create professional level documents

Document Structure

- All documents for a given product should have a similar structure
  - A good reason for *product standards*
- The IEEE Standard for User Documentation lists such a structure
  - It is a superset of what most documents need
- The authors “best practices” are:

1. Put a cover page on all documents
2. Divide documents into chapters with sections and subsections
3. Add an index if there is lots of reference information
4. Add a glossary to define ambiguous terms

Online Documentation

- Either internal to the application or Web based
- Requires rethinking the presentation style since normal “paper” documentation approaches do not carry over well
- Should act as a supplement to paper documentation
- Biggest benefit of Web docs are that they are always current

Document Preparation

- Covers the entire process of creating and formatting a document for publication
  - Author recommends using specialized (and separate) tools for creating and preparing documents
    - This is only important for user documentation
- It is often important to have a professional writer or document publisher evaluate documents before publication to ensure they look good and will carry over to paper well

Document Storage

- Author Recommends (in 2001)
  - File System to store the actual documents
  - Database to store references to the files with metadata to allow searching and referencing
- Today it is probably better to use a content management systems
  - CVS or Subversion
    - Free and Open Source
    - Easy to setup and maintain

What is Software Quality?

- Simplistically, quality is an attribute of software that implies the software meets its specification
- This definition is too simple for ensuring quality in software systems
  - Software specifications are often incomplete or ambiguous
  - Some quality attributes are difficult to specify
  - Tension exists between some quality attributes, e.g. efficiency vs. reliability

Software Quality Attributes

- Safety
- Security
- Reliability
- Resilience
- Robustness
- Understandability
- Testability
- Adaptability
- Modularity
- Complexity
- Portability
- Usability
- Reusability
- Efficiency
- Learnability

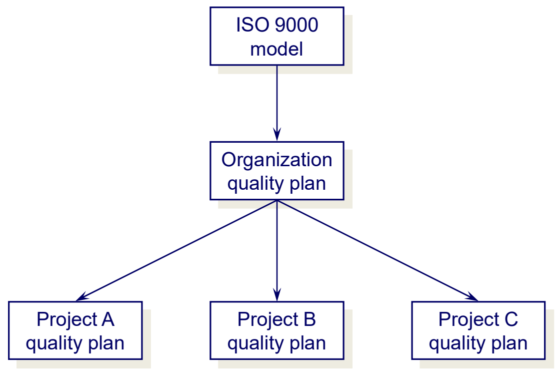
Software Quality

- Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software
  - Software requirements are the foundation from which quality is measured.
    - Lack of conformance to requirements is lack of quality.
  - Specified standards define a set of development criteria that guide the manner in which software is engineered.
    - If the criteria are not met, lack of quality will almost surely result.
  - There is a set of implicit requirements that often goes unmentioned.
    - If software conforms to its explicit requirements but fails to meet its implicit requirements, software quality is suspect.

Software Quality Assurance

- To ensure quality in a software product, an organization must have a three-prong approach to quality management:
  - Organization-wide policies, procedures and standards must be established.
  - Project-specific policies, procedures and standards must be tailored from the organization-wide templates.
  - Quality must be controlled; that is, the organization must ensure that the appropriate procedures are followed for each project
- Standards exist to help an organization draft an appropriate software quality assurance plan.
  - ISO 9000-3
  - ANSI/IEEE standards
- External entities can be contracted to verify that an organization is standard-compliant.

A Software Quality Plan



SQA Activities

- Applying technical methods
  - To help the analyst achieve a high quality specification and a high quality design
- Conducting formal technical reviews
  - A stylized meeting conducted by technical staff with the sole purpose of uncovering quality problems
- Testing Software
  - A series of test case design methods that help ensure effective error detection
- Enforcing standards
- Controlling change
  - Applied during software development and maintenance
- Measurement
  - Track software quality and asses the ability of methodological and procedural changes to improve software quality
- Record keeping and reporting
  - Provide procedures for the collection and dissemination of SQA information

Advantages of SQA

- Software will have fewer latent defects, resulting in reduced effort and time spent during testing and maintenance
- Higher reliability will result in greater customer satisfaction
- Maintenance costs can be reduced
- Overall life cycle cost of software is reduced

Disadvantages of SQA

- It is difficult to institute in small organizations, where available resources to perform necessary activities are not available
- It represents cultural change - and change is never easy
- It requires the expenditure of dollars that would not otherwise be explicitly budgeted to software engineering or QA

Software and Security

- Security is about *regulating access to assets*
  - E.g., information or functionality
- Software provides *functionality*
  - E.g., on-line exam results
- This functionality comes with certain *risks*
  - E.g., what are risks of on-line exam results?
    - Privacy (score leakage); Modification
- Software security is about *managing these risks*

Software and Security

- Security is always a secondary concern
  - Primary goal of software is to provide functionalities or services
  - Managing associated risks is a derived/secondary concern
- There is often a trade-off/conflict between
  - security
  - functionality & convenience
- Security achievement is hard to evaluate when nothing bad happens

Starting Point for Ensuring Security

- Any discussion of security should start with an inventory of
  - the stakeholders (owners, companies…)
  - their assets (data, service, customer info…)
  - the threats to these assets (erase, steal…)
  - Attackers
    - employees, clients, script kiddies, criminals
- Any discussion of security without understanding these issues is meaningless

Security Concepts

- Security is about imposing countermeasures to reduce risks to assets to acceptable levels
  - "Perfect security" is not necessary and costly
- A security policy is a specification of what security requirements/goals the countermeasures are intended to achieve
  - secure against what and from whom ?
- Security mechanisms to enforce the policy
  - What actions we should take under an attack?

Security Objectives: CIA

- Confidentiality (or secrecy)
  - unauthorized users cannot read information
- Integrity
  - unauthorized users cannot alter information
- Availability
  - authorized users can always access information
- Non-repudiation for accountability
  - authorized users cannot deny actions
- Others
  - Privacy, anonymity…

Security Goals

- The well-known trio
  - confidentiality, integrity, avaliability (CIA)
- There are more "concrete" goals
  - traceability and auditing (forensics)
  - monitoring (real-time auditing)
  - multi-level security
  - privacy & anonymity
  - ...
- and meta-property
  - assurance – that the goals are met
    - "information assurance"

Cloud Computing

- Cloud computing is the delivery of different services through the Internet. These resources include tools and applications like data storage, servers, databases, networking, and software.
- Rather than keeping files on a proprietary hard drive or local storage device, cloud-based storage makes it possible to save them to a remote database. As long as an electronic device has access to the web, it has access to the data and the software programs to run it.
- Cloud computing is a popular option for people and businesses for a number of reasons including cost savings, increased productivity, speed and efficiency, performance, and security.

Types of Cloud Computing

- There are four main types of cloud computing:
  - private clouds
  - public clouds
  - hybrid clouds
  - Multi clouds

Private Clouds

- Private clouds are loosely defined as cloud environments solely dedicated to a single end user or group, where the environment usually runs behind that user or group's firewall.
- All clouds become private clouds when the underlying IT infrastructure is dedicated to a single customer with completely isolated access.
- But private clouds no longer have to be sourced from on-premise IT infrastructure. Organizations are now building private clouds on rented, vendor-owned data centers located off-premises, which makes any location and ownership rules obsolete.

Public Clouds

- Public clouds are cloud environments typically created from IT infrastructure not owned by the end user. Some of the largest public cloud providers include Alibaba Cloud, Amazon Web Services (AWS), Google Cloud, IBM Cloud, and Microsoft Azure.
- Traditional public clouds always ran off-premises, but today's public cloud providers have started offering cloud services on clients' on premise data centers. This has made location and ownership distinctions obsolete.

Hybrid Clouds

- A hybrid cloud is a seemingly single IT environment created from multiple environments connected through local area networks (LANs), wide area networks (WANs), virtual private networks (VPNs), and/or APIs.
- The characteristics of hybrid clouds are complex and the requirements can differ, depending on whom you ask. For example, a hybrid cloud may need to include:
  - At least one private cloud and at least one public cloud
  - Two or more private clouds
  - Two or more public clouds
  - A bare-metal or virtual environment connected to at least one public cloud or private cloud

Multi Clouds

- Multi clouds are a cloud approach made up of more than 1 cloud service, from more than 1 cloud vendor—public or private.
- All hybrid clouds are multi clouds, but not all multi clouds are hybrid clouds.
- Multi clouds become hybrid clouds when multiple clouds are connected by some form of integration or orchestration.

Cloud Computing Services

- **Software-as-a-service (SaaS)** involves the licensure of a software application to customers. Licenses are typically provided through a pay-as-you-go model or on-demand. This type of system can be found in Microsoft Office's 365.

- **Infrastructure-as-a-service (IaaS)** involves a method for delivering everything from operating systems to servers and storage through IP-based connectivity as part of an on-demand service. Clients can avoid the need to purchase software or servers, and instead procure these resources in an outsourced, on-demand service. Popular examples of the IaaS system include IBM Cloud and Microsoft Azure.

- **Platform-as-a-service (PaaS)** is considered the most complex of the three layers of cloud-based computing. PaaS shares some similarities with SaaS, the primary difference being that instead of delivering software online, it is actually a platform for creating software that is delivered via the Internet. This model includes platforms like Salesforce.com and Heroku.

Thank you..

HNDIT2022- Software Development

Week 5: Data Types & Subprograms

Program Structure

Virtually all structured programs share a similar overall pattern:

- Statements to establish the start of the program
- Variable declaration
- Program statements (blocks of code)

Variable Declaration

- Variables are place holders for data a program might use or manipulate.
- Variables are given names so that we can assign values to them and refer to them later to read the values.
- Variables typically store values of a given *type*.

Data Types

- Integer – to store integer or “whole” numbers
- Real – to store real or fractional numbers (also called float to indicate a floating point number)
- Character – A single character such as a letter of the alphabet or punctuation.
- String – A collection of characters

Numeric Data Types and Non-numeric Data Types

- Numeric Data Types
  - Numeric data types are types of data that consist of numbers, which can be computed mathematically with various standard operators such as add, minus, multiply, divide and more.
  - Examples of numeric data types are examination marks, height, weight, the number of students in a class, share values, price of goods, monthly bills, fees and others.
  - eg_:- Integer, Double, Float, Long

- Non- numeric Data Types
  - Nonnumeric data types are data that cannot be manipulated mathematically using standard arithmetic operators.
  - The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type.
  - Eg:- String, Boolean, Character

There are eight primitive data types in Java:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Elementary/Fundamental Data Types

- A **Fundamental Datatype** is one which is a concrete form of data type, and it is introduced by the programming language itself.
- Therefore, a fundamental datatype has its own fundamental characteristics and properties defined in the language.
- It also has some fundamental methods to perform operations over the data.
- In case of fundamental datatypes, the only concern is the type and nature of the data.
- Some common fundamental datatypes include **int, char, float, void,** etc.

Derived Data Types

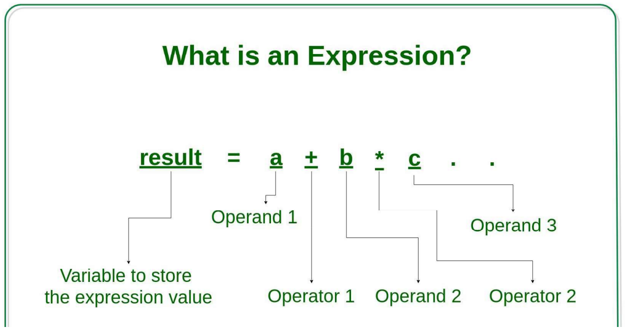
- **Derived Datatypes** are composed of fundamental datatypes; they are derived from the fundamental data types.
- Therefore, they have some additional characteristics and properties other than that of fundamental data types.
- Derived data types are defined by the user because the programming language does not have built-in definition of the derived data types.
- Programmers can modify or redefine the derived datatypes. Some common examples of derived datatypes include **arrays, structures, pointers,** etc.

Difference between Fundamental Data Types And Derived Data Types

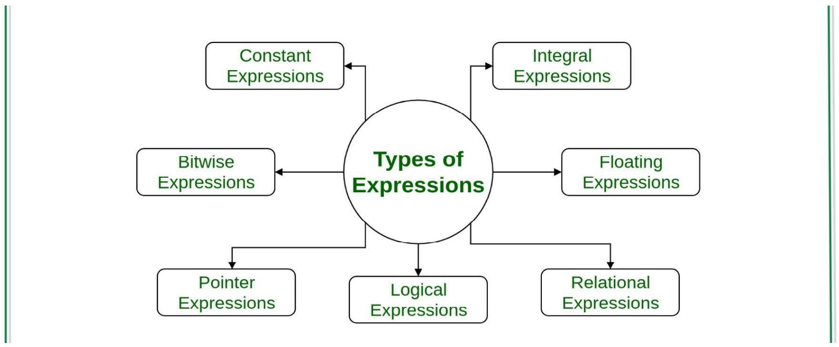
Fundamental Datatypes	Derived Data Types
Fundamental datatypes are also known as primitive datatypes.	Derived datatypes are composed of fundamental datatypes.
Some fundamental datatypes include **int, char, float, void,** etc.	Derived datatypes include arrays, structures, pointers, etc.
Integer or Character datatypes are classified as **int, char, signed int, signed char, unsigned int, unsigned char.**	Pointers are used to store address of some other variables.
Integers are used to store integer type data, not the floating point number.	Arrays are used to store homogeneous data.
Floats are used to store decimal numbers. The variations **include float, double, long double.**	Structures are group of some primitive datatypes like int, **float, double,** etc.
Voids is used where no return values are specified.	Unions are like structures, but all the members of a union share the same memory location.

Expressions

- An expression is a combination of operators, constants and variables.
- An expression may consist of one or more operands, and zero or more operators to produce a value.



Types of Expressions



- ***Constant expressions:*** *Constant Expressions consists of only constant values. A constant value is one that doesn't change.*

Examples:

5, 10 + 5 / 6.0, 'x'

- ***Integral expressions:*** *Integral Expressions are those which produce integer results after implementing all the automatic and explicit type conversions.*

Examples:

*x, x * y, x + int(5.0)*

where x and y are integer variables.

- ***Floating expressions:*** *Float Expressions are which produce floating point results after implementing all the automatic and explicit type conversions.*

Examples:

x + y, 10.75

where x and y are floating point variables.

- **Relational expressions:** Relational Expressions yield results of type bool which takes a value true or false. When arithmetic expressions are used on either side of a relational operator, they will be evaluated first and then the results compared. Relational expressions are also known as Boolean expressions.

Examples:

x <= y, x + y > 2

- **Logical expressions:** Logical Expressions combine two or more relational expressions and produces bool type results.

Examples:

x > y && x == 10, x == 10 || y == 5

- **Pointer expressions:** Pointer Expressions produce address values.

Examples:

&x, ptr, ptr++

where x is a variable and ptr is a pointer.

- **Bitwise expressions:** Bitwise Expressions are used to manipulate data at bit level. They are basically used for testing or shifting bits. Examples:

x << 3

shifts three bit position to left

y >> 1

shifts one bit position to right.

Shift operators are often used for multiplication and division by powers of two.

- **Note: An expression may also use combinations of the above expressions. Such expressions are known as compound expressions.**

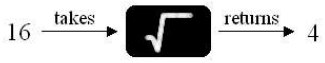
Subprogram

- A subprogram is a piece of code that can be used over and over again.
- If you were coding a calculator, you could make a subprogram that only does adding. Every time you need to do adding, you can use the same subprogram.
- The final construct for directing the flow of control in a program is the subprogram.
- Subprograms are a way of grouping statements that provide a single logical operation.
- Actually, subprograms come in two varieties: procedures and functions.

```
def getName ():
    name = str(input("Enter first name >>>"))
    return namedef getAge():
    age = int(input("Enter your age >>>"))
    return age
def welcome_name (name):
    print("Hello", name)
def welcome_age (age):
    print("You are", age, "years old")
def start():
    first_name = getName()
    welcome_name (first_name)
    ageCheck = getAge()
    welcome_age (ageCheck)
start()
#Add a subprogram that asks for last name (and displays it back)
#OR
#Add a subprogram that asks for a username (and displays it back)
#OR
#Add a subprogram that asks for a password (and displays it back)
```

Functions

- Functions are specialized subprograms that produce a single output as the result of their execution.
- You are probably already familiar with many functions from your study of mathematics. Operations like sine, cosine, exponential, square root, and logarithm are all functions that require a number as input and produce a number as output.
- The diagram below shows the square root function as a black box that takes 16 as an input parameter and returns 4 as the output:



Procedures

- Procedures are like small versions of a program that perform a single logical task.
- A useful example of a procedure is the swap operation. Suppose we had a program that sorted an array of letters.
- A common operation for such a program is to swap the letters stored in two of the array cells so that they are put in alphabetical order. To put the entire array in sorted order, our sort program will need to make many swaps. In the example to the right, we can see that at least three more swaps will be needed to sort the array. In order to actually perform a swap, we will need to include the following three operations in our program.

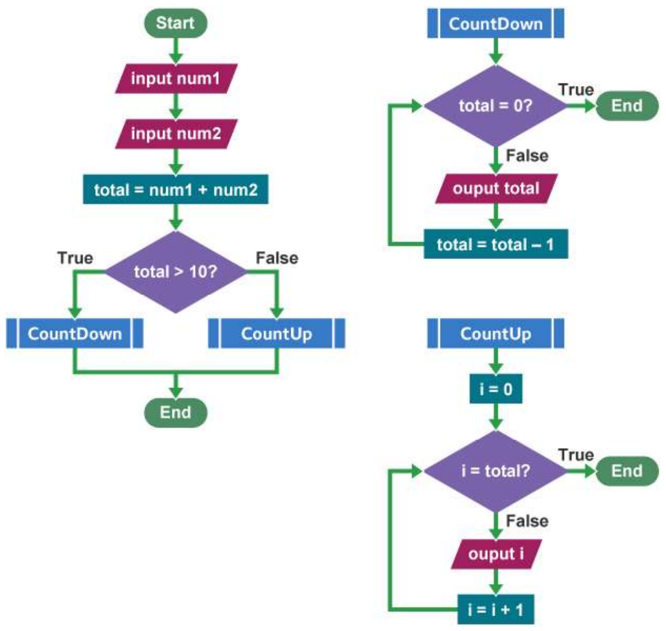
```
temp := variable1
variable1 := variable2
variable2 := temp
```

Rather than writing the three statements for the swap operation every time, we can use a subprogram to group them together into a logical unit. Then whenever we need to swap the values of two variables, we can call the same swap procedure to do the work.

Subroutines

- Subroutines are small blocks of code in a modular program designed to perform a particular task. Since a subroutine is in itself a small program, it can contain any of the sequence , selection and iteration constructs.
- A function is used when a value is returned to the calling routine, while a subroutine is used when a desired task is needed, but no value is returned.

- In the following example, the **algorithm** asks the user to enter two numbers. It then adds them together and, if the total is over 10, it runs the 'CountDown' subroutine. If the total is not over 10, it will run the 'CountUp' subroutine.



Thank you..

HNDIT2022-Software Development

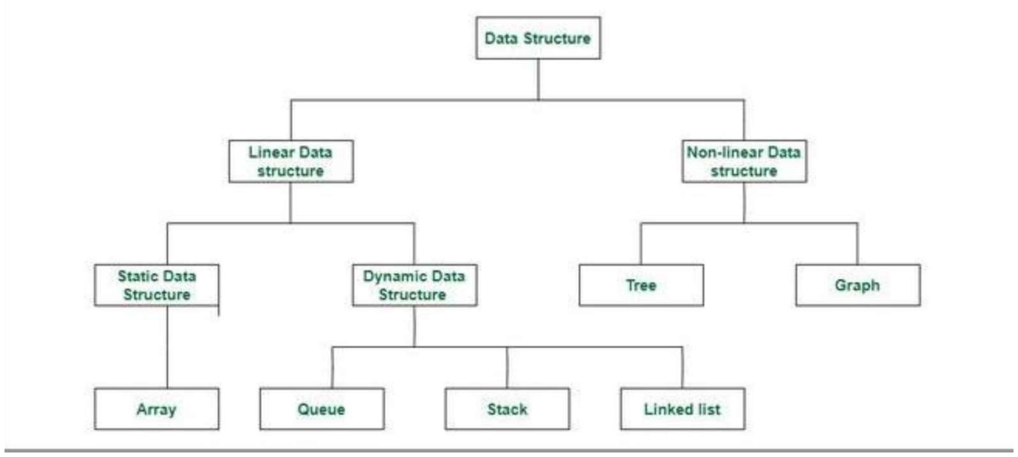
Week 6: Data Structures & Data abstraction

What is a Data Structure?

A data structure is a storage that is used to store and organize data so that it can be used efficiently.

- A data structure is not only used for organizing the data.
- It is also used for processing, retrieving, and storing data.
- Data types are often confused as a type of data structures, but it is not precisely correct even though they are referred to as Abstract Data Types. Data types represent the nature of the data while data structures are just a collection of similar or different data types in one.

Classification of Data Structure



Linear Data Structure

- Data structure in which data elements are arranged sequentially or linearly, where each element is attached to its previous and next adjacent elements, is called a linear data structure.
  Examples of linear data structures are array, stack, queue, linked list, etc.

- **Static data structure:** Static data structure has a fixed memory size. It is easier to access the elements in a static data structure.
  An example of this data structure is an array.
- **Dynamic data structure:** In dynamic data structure, the size is not fixed. It can be randomly updated during the runtime which may be considered efficient concerning the memory (space) complexity of the code.
  Examples of this data structure are queue, stack, etc.

Non-linear Data Structure

- Data structures where data elements are not placed sequentially or linearly are called non-linear data structures. In a non-linear data structure, we can't traverse all the elements in a single run only.
  Examples of non-linear data structures are trees and graphs.

Major Operations

- **Searching:** We can search for any element in a data structure.
- **Sorting:** We can sort the elements of a data structure either in an ascending or descending order.
- **Insertion:** We can also insert the new element in a data structure.
- **Updating:** We can also update the element, i.e., we can replace the element with another element.
- **Deletion:** We can also perform the delete operation to remove the element from the data structure.

Advantages of Data Structures

The following are the advantages of a data structure:

- **Efficiency:** If the choice of a data structure for implementing a particular ADT is proper, it makes the program very efficient in terms of time and space.
- **Reusability:** The data structure provides reusability means that multiple client programs can use the data structure.
- **Abstraction:** The data structure specified by an ADT also provides the level of abstraction. The client cannot see the internal working of the data structure, so it does not have to worry about the implementation part. The client can only see the interface.

Abstract Data Type

- An abstract data type is an abstraction of a data structure that provides only the interface to which the data structure must adhere. The interface does not give any specific details about something should be implemented or in what programming language.

Collections

- A collection is a concept applicable to **abstract data types**, and does not prescribe a specific implementation as a concrete **data structure**, though often there is a conventional choice.

- Examples of collections include **lists**, **sets**, **multisets**, **trees** and **graphs**.

Many collections define a particular linear ordering, with access to one or both ends. The actual data structure implementing such a collection need not be linear—for example, a priority queue is often implemented as a **heap**, which is a kind of tree. Important linear collections include:

- **lists;**
- **stacks;**
- **queues;**
- **priority queues;**
- **double-ended queues;**
- **double-ended priority queues.**

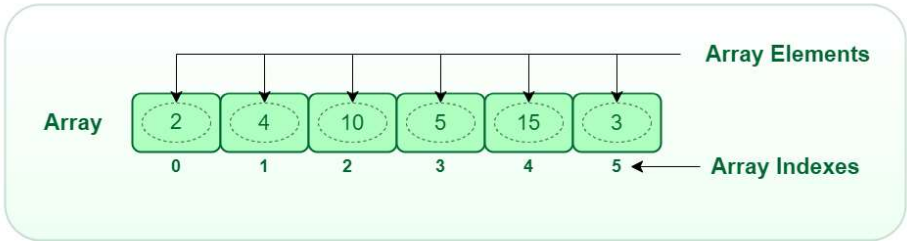
Arrays

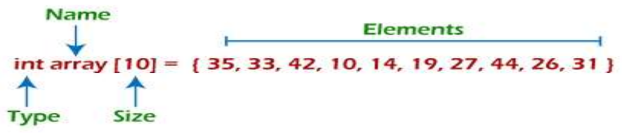
- Arrays are defined as the collection of similar types of data items stored at contiguous memory locations.
- It is one of the simplest data structures where each data element can be randomly accessed by using its index number.

Properties of Array

- Each element in an array is of the same data type and carries the same size that is 4 bytes.
- Elements in the array are stored at contiguous memory locations from which the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Representation of an Array





Basic Operations in the Arrays

- **Traverse** − print all the array elements one by one.
- **Insertion** − Adds an element at the given index.
- **Deletion** − Deletes an element at the given index.
- **Search** − Searches an element using the given index or by the value.
- **Update** − Updates an element at the given index.
- **Display** − Displays the contents of the array.

Insertion Operation in Arrays

- Algorithm

```
1.Start
2. Create an Array of a desired datatype and size.
3. Initialize a variable 'i' as 0.
4. Enter the element at ith index of the array.
5. Increment i by 1.
6. Repeat Steps 4 & 5 until the end of the array.
7. Stop
```

Deletion Operation

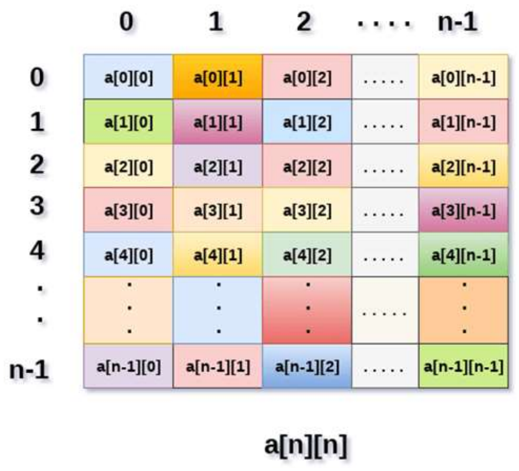
- Algorithm-

```
1. Start
2. Set J = K
3. Repeat steps 4 and 5 while J < N
4. Set LA[J] = LA[J + 1]
5. Set J = J+1
6. Set N = N-1
7. Stop
```

2D Array

- 2D array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns.

int arr[max_rows][max_columns];



	0	1	2	....	n-1
0	a[0][0]	a[0][1]	a[0][2]	.....	a[0][n-1]
1	a[1][0]	a[1][1]	a[1][2]	.....	a[1][n-1]
2	a[2][0]	a[2][1]	a[2][2]	.....	a[2][n-1]
3	a[3][0]	a[3][1]	a[3][2]	.....	a[3][n-1]
4	a[4][0]	a[4][1]	a[4][2]	.....	a[4][n-1]
.	.	.	.	.....	.
.	.	.	.		.
n-1	a[n-1][0]	a[n-1][1]	a[n-1][2]	.....	a[n-1][n-1]

a[n][n]

Advantages of Array

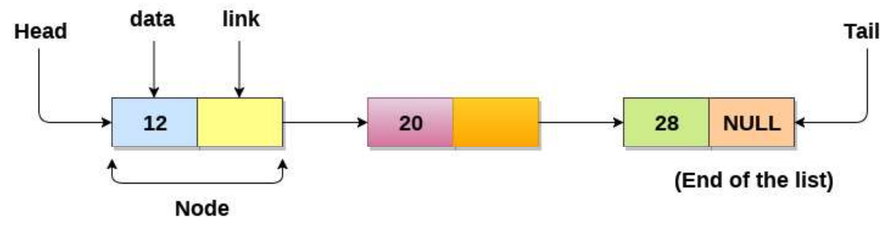
- Array provides the single name for the group of variables of the same type. Therefore, it is easy to remember the name of all the elements of an array.
- Traversing an array is a very simple process; we just need to increment the base address of the array in order to visit each element one by one.
- Any element in the array can be directly accessed by using the index.

Disadvantages of Array

- Array is homogenous. It means that the elements with similar data type can be stored in it.
- In array, there is static memory allocation that is size of an array cannot be altered.
- There will be wastage of memory if we store less number of elements than the declared size.

Linked Lists

- Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.



As per the above illustration, following are the important points to be considered.
Linked List contains a link element called first (head).
Each link carries a data field(s) and a link field called next.
Each link is linked with its next link using its next link.
Last link carries a link as null to mark the end of the list.

Uses of Linked List

- The list is not required to be contiguously present in the memory. The node can reside any where in the memory and linked together to make a list. This achieves optimized utilization of space.
- List size is limited to the memory size and doesn't need to be declared in advance.
- Empty node can not be present in the linked list.
- We can store values of primitive types or objects in the singly linked list.

Why use linked list over array?

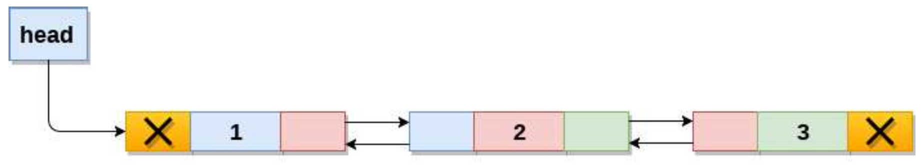
Till now, we were using array data structure to organize the group of elements that are to be stored individually in the memory. However, Array has several advantages and disadvantages which must be known in order to decide the data structure which will be used throughout the program.

- Array contains following limitations:
- The size of array must be known in advance before using it in the program.
- Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
- All the elements in the array need to be contiguously stored in the memory. Inserting any element in the array needs shifting of all its predecessors.

Linked list is the data structure which can overcome all the limitations of an array. Using linked list is useful because,

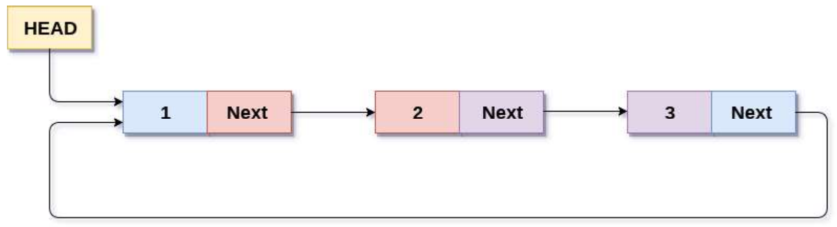
- It allocates the memory dynamically. All the nodes of linked list are non-contiguously stored in the memory and linked together with the help of pointers.
- Sizing is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and limited to the available memory space.

Doubly Linked List



Doubly Linked List

Circular Singly Linked List



Circular Singly Linked List

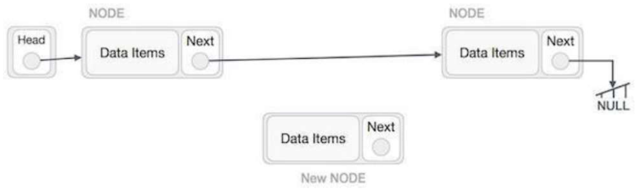
Basic Operations in the Linked Lists

The basic operations in the linked lists are insertion, deletion, searching, display, and deleting an element at a given key. These operations are performed on Singly Linked Lists as given below −

- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Display** − Displays the complete list.
- **Search** − Searches an element using the given key.
- **Delete** − Deletes an element using the given key.

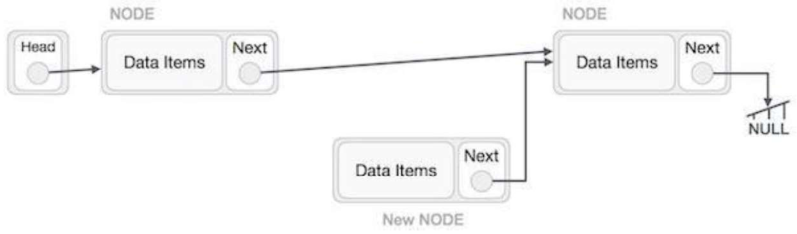
Insertion Operation

- Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.



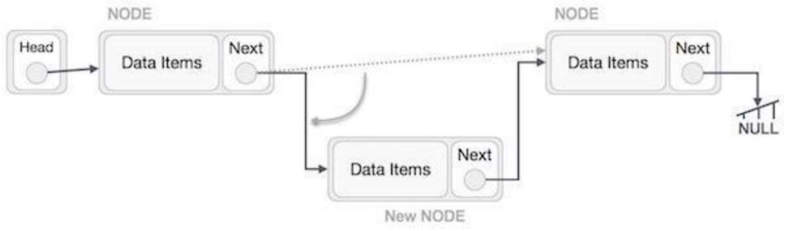
- Imagine that we are inserting a node B (NewNode), between A (LeftNode) and C (RightNode). Then point B.next to C –

NewNode.next −> RightNode;



- Now, the next node at the left should point to the new node.

LeftNode.next −> NewNode;



Insertion at Beginning

1. START
2. Create a node to store the data
3. Check if the list is empty
4. If the list is empty, add the data to the node and assign the head pointer to it.
5. If the list is not empty, add the data to a node and link to the current head. Assign the head to the newly added node.
6. END

Insertion at Ending

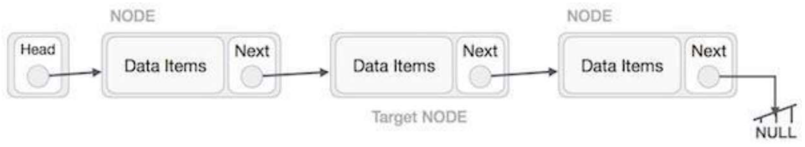
1. START
2. Create a new node and assign the data
3. Find the last node
4. Point the last node to new node
5. END

Insertion at a Given Position

1. START
2. Create a new node and assign data to it
3. Iterate until the node at position is found
4. Point first to new first node
5. END

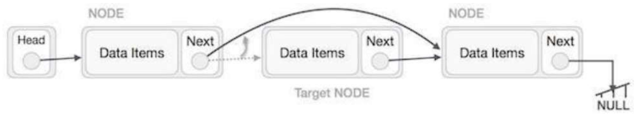
Deletion Operation

- Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.



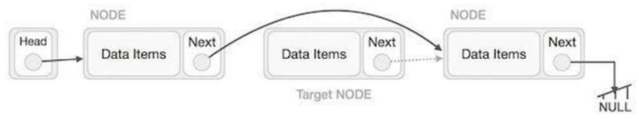
- The left (previous) node of the target node now should point to the next node of the target node –

LeftNode.next −> TargetNode.next;

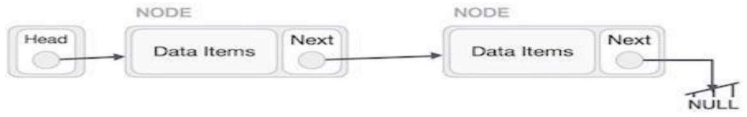


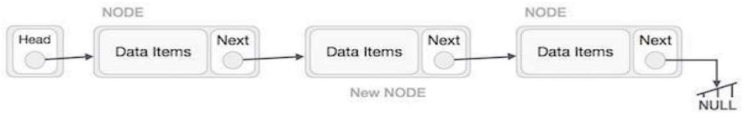
This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

TargetNode.next –> NULL;



We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.





Similar steps should be taken if the node is being inserted at the beginning of the list. While inserting it at the end, the second last node of the list should point to the new node and the new node will point to NULL.

Deletion at Beginning

In this deletion operation of the linked, we are deleting an element from the beginning of the list. For this, we point the head to the second node.

Algorithm-

```
1. START
2. Assign the head pointer to the next node in the list
3. END
```

Deletion at Ending

In this deletion operation of the linked, we are deleting an element from the ending of the list.

Algorithm-

```
1. START
2. Iterate until you find the second last element in the list.
3. Assign NULL to the second last element in the list.
4. END
```

Deletion at a Given Position

In this deletion operation of the linked, we are deleting an element at any position of the list.

Algorithm-

```
    1. START
    2. Iterate until find the current node at position in the list
    3. Assign the adjacent node of current node in the list to its previous node.
    4. END
```

Tuples

- Tuples are used to store multiple items in a single variable.
- Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.
- A tuple is a collection which is ordered and **unchangeable**.
- Tuples are written with round brackets.

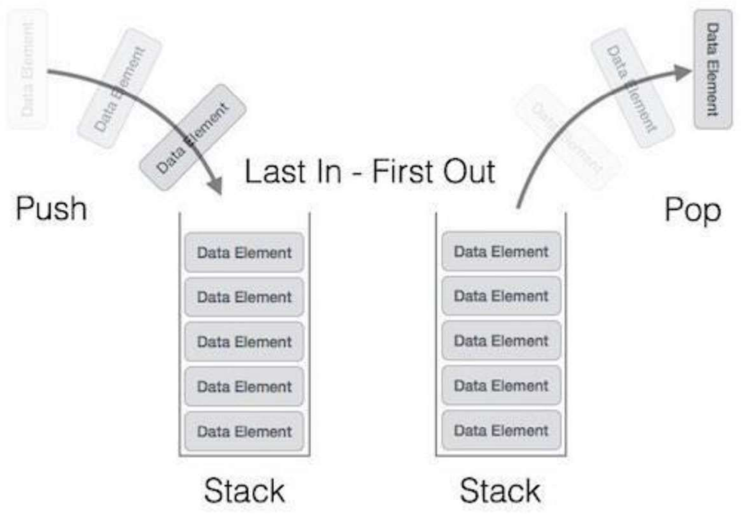
```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Stack

A stack is an Abstract Data Type (ADT), that is popularly used in most programming languages. It is named stack because it has the similar operations as the real-world stacks, for example – a pack of cards or a pile of plates, etc. The stack follows the LIFO (Last in - First out) structure where the last element inserted would be the first element deleted.

Stack Representation

- A Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.
- A stack can be implemented by means of Array, Structure, Pointer, and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.



Basic Operations on Stacks

- Stack operations usually are performed for initialization, usage and, de-initialization of the stack ADT.
- The most fundamental operations in the stack ADT include: push(), pop(), peek(), isFull(), isEmpty(). These are all built-in operations to carry out data manipulation and to check the status of the stack.
- Stack uses pointers that always point to the topmost element within the stack, hence called as the **top** pointer.

Insertion: push()

- push() is an operation that inserts elements into the stack. The following is an algorithm that describes the push() operation in a simpler way.

Algorithm-

```
1 − Checks if the stack is full.
2 − If the stack is full, produces an error and exit.
3 − If the stack is not full, increments top to point next empty space.
4 − Adds data element to the stack location, where top is pointing.
5 − Returns success.
```

Deletion: pop()

pop() is a data manipulation operation which removes elements from the stack. The following pseudo code describes the pop() operation in a simpler way.

Algorithm-

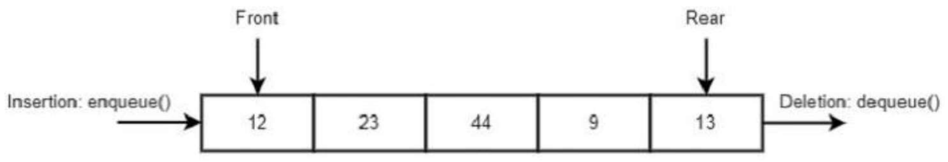
```
1 − Checks if the stack is empty.
2 − If the stack is empty, produces an error and exit.
3 − If the stack is not empty, accesses the data element at which top is pointing.
4 − Decreases the value of top by 1.
5 − Returns success.
```

Queue

- Queue, like Stack, is also an abstract data structure. The thing that makes queue different from stack is that a queue is open at both its ends. Hence, it follows FIFO (First-In-First-Out) structure, i.e. the data item inserted first will also be accessed first. The data is inserted into the queue through one end and deleted from it using the other end.

A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops.

Representation of Queues



Queue: FIFO Operation

Basic Operations

- The most fundamental operations in the queue ADT include: enqueue(), dequeue(), peek(), isFull(), isEmpty(). These are all built-in operations to carry out data manipulation and to check the status of the queue.
- Queue uses two pointers − **front** and **rear**. The front pointer accesses the data from the front end (helping in enqueueing) while the rear pointer accesses data from the rear end (helping in dequeuing).

Insertion Operation: enqueue()

- The enqueue() is a data manipulation operation that is used to insert elements into the stack. The following algorithm describes the enqueue() operation in a simpler way.

Algorithm

```
1 – START
2 – Check if the queue is full.
3 – If the queue is full, produce overflow error and exit.
4 – If the queue is not full, increment rear pointer to point the next empty space.
5 – Add data element to the queue location, where the rear is pointing.
6 – return success.
7 – END
```

Deletion Operation: dequeue()

The dequeue() is a data manipulation operation that is used to remove elements from the stack. The following algorithm describes the dequeue() operation in a simpler way.

Algorithm-

```
1 – START
2 – Check if the queue is empty.
3 – If the queue is empty, produce underflow error and exit.
4 – If the queue is not empty, access the data where front is pointing.
5 – Increment front pointer to point to the next available data element.
6 – Return success.
7 – END
```

Data Abstraction

- Data abstraction is the reduction of a particular body of data to a simplified representation of the whole. Abstraction, in general, is the process of removing characteristics from something to reduce it to a set of essential elements.

Thank you..

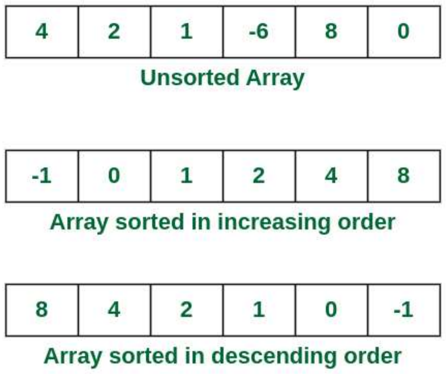
HNDIT2022- Software Development

Week 7: Sorting & Searching Algorithms

What is a Sorting Algorithm?

A Sorting Algorithm is used to rearrange a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.

Sorting Example…



4	2	1	-6	8	0

Unsorted Array

-1	0	1	2	4	8

Array sorted in increasing order

8	4	2	1	0	-1

Array sorted in descending order

Sorting Algorithms

- Selection Sort
- Bubble Sort
- Insertion Sort
- Merge Sort
- Quick Sort
- Radix Sort
- Heap Sort..etc.

Selection Sort

- **Selection sort** is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.
- The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted portion.
- This process is repeated for the remaining unsorted portion of the list until the entire list is sorted..

The algorithm maintains two subarrays in a given array.

- *The subarray which already sorted.*
- *The remaining subarray was unsorted.*

In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the beginning of the sorted subarray.

After every iteration sorted subarray size increase by one and the unsorted subarray size decrease by one.

How does selection sort work?

Lets consider the following array as an example: ***arr[] = {64, 25, 12, 22, 11}***

First pass:

For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where 64 is stored presently, after traversing whole array it is clear that 11 is the lowest value.

64	25	12	22	11

Thus, replace 64 with 11. After one iteration 11, which happens to be the least value in the array, tends to appear in the first position of the sorted list.

11	25	12	22	64

Second Pass:

For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.

11 25 12 22 64

After traversing, we found that 12 is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.

11 12 25 22 64

Third Pass:

Now, for third place, where 25 is present again traverse the rest of the array and find the third least value present in the array.

11 12 25 22 64

While traversing, 22 came out to be the third least value and it should appear at the third place in the array, thus swap 22 with element present at third position.

11 12 22 25 64

Fourth pass:

Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array

As 25 is the 4th lowest value hence, it will place at the fourth position.

11 12 22 25 64

Fifth Pass:

At last the largest value present in the array automatically get placed at the last position in the array

The resulted array is the sorted array.

11 12 22 25 64

Algorithm:

Step 1 − Set MIN to location 0

Step 2 − Search the minimum element in the list

Step 3 − Swap with value at location MIN

Step 4 − Increment MIN to point to next element

Step 5 − Repeat until list is sorted

```
procedure selection sort
  list  : array of items
  n     : size of list

  for i = 1 to n - 1
  /* set current element as minimum*/
    min = i

    /* check the element to be minimum */
    for j = i+1 to n
      if list[j] < list[min] then
        min = j;
      end if
    end for

    /* swap the minimum element with the current element*/
    if indexMin != i  then
      swap list[min] and list[i]
    end if
  end for
end procedure
```

Bubble Sort

- **Bubble Sort** is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.
- This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

How does Bubble Sort Work?

Input: *arr[] = {6, 3, 0, 5}*

First Pass:

- *Bubble sort starts with very first two elements, comparing them to check which one is greater.*
  - *(6 3 0 5) –> (**3 6** 0 5), Here, algorithm compares the first two elements, and swaps since 6 > 3.*
  - *(3 **6 0** 5) –> (3 **0 6** 5), Swap since 6 > 0*
  - *(3 0 **6 5**) –> (3 0 **5 6**), Swap since 6 > 5*

Second Pass:

- *Now, during second iteration it should look like this:*
  - *(**3 0** 5 6) –> (**0 3** 5 6), Swap since 3 > 0*
  - *(0 **3 5** 6) –> (0 **3 5** 6), No change as 5 > 3*

Third Pass:

- *Now, the array is already sorted, but our algorithm does not know if it is completed.*
- *The algorithm needs one **whole** pass without **any** swap to know it is sorted.*
  - *(**0 3** 5 6) –> (**0 3** 5 6), No change as 3 > 0*
- *Array is now sorted and no more pass will happen.*

Follow the below steps to solve the problem:

- Run a nested for loop to traverse the input array using two variables **i** and **j**, such that 0 ≤ i < n-1 and 0 ≤ j < n-i-1
- If **arr[j]** is greater than **arr[j+1]** then swap these adjacent elements, else move on
- Print the sorted array

Pseudocode-

```
procedure bubbleSort( list : array of items )

 loop = list.count;

 for i = 0 to loop-1 do:
   swapped = false

   for j = 0 to loop-1 do:

     /* compare the adjacent elements */
     if list[j] > list[j+1] then
       /* swap them */
       swap( list[j], list[j+1] )
       swapped = true
     end if

   end for
```

```
/*if no number was swapped that means
   array is sorted now, break the loop.*/

   if(not swapped) then
     break
   end if

 end for

end procedure return list
```

Insertion Sort

- **Insertion sort** is a simple sorting algorithm that works similar to the way you sort playing cards in your hands.
- The array is virtually split into a sorted and an unsorted part.
- Values from the unsorted part are picked and placed at the correct position in the sorted part.

Characteristics of Insertion Sort

- This algorithm is one of the simplest algorithm with simple implementation
- Basically, Insertion sort is efficient for small data values
- Insertion sort is adaptive in nature, i.e. it is appropriate for data sets which are already partially sorted.

How does insertion sort work?

Consider an example: arr[]:

{12, 11, 13, 5, 6}

First Pass:

Initially, the first two elements of the array are compared in insertion sort.

12	11	13	5	6

Here, 12 is greater than 11 hence they are not in the ascending order and 12 is not at its correct position. Thus, swap 11 and 12.

So, for now 11 is stored in a sorted sub-array.

11	12	13	5	6

Second Pass:

Now, move to the next two elements and compare them

11 12 13 5 6

Here, 13 is greater than 12, thus both elements seems to be in ascending order, hence, no swapping will occur. 12 also stored in a sorted sub-array along with 11

Third Pass:

Now, two elements are present in the sorted sub-array which are 11 and 12

Moving forward to the next two elements which are 13 and 5

11 12 13 5 6

Both 5 and 13 are not present at their correct place so swap them

11 12 5 13 6

- After swapping, elements 12 and 5 are not sorted, thus swap again
- 11 5 12 13 6
- Here, again 11 and 5 are not sorted, hence swap again
- 5 11 12 13 6
- Here, 5 is at its correct position

Fourth Pass:

Now, the elements which are present in the sorted sub-array are 5, 11 and 12

Moving to the next two elements 13 and 6

5 11 12 13 6

Clearly, they are not sorted, thus perform swap between both

5 11 12 6 13

Now, 6 is smaller than 12, hence, swap again

5 11 6 12 13

- Here, also swapping makes 11 and 6 unsorted hence, swap again
- 5 6 11 12 13
- Finally, the array is completely sorted.

Insertion Sort Algorithm

The simple steps of achieving the insertion sort are listed as follows -

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step2 - Pick the next element, and store it separately in a key.

Step3 - Now, compare the key with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.

Step 5 - Insert the value.

Step 6 - Repeat until the array is sorted.

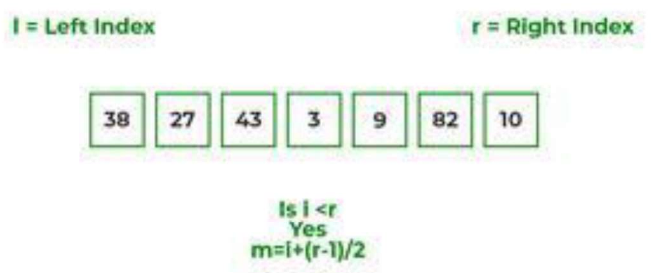
Pseudo Code for Insertion Sort

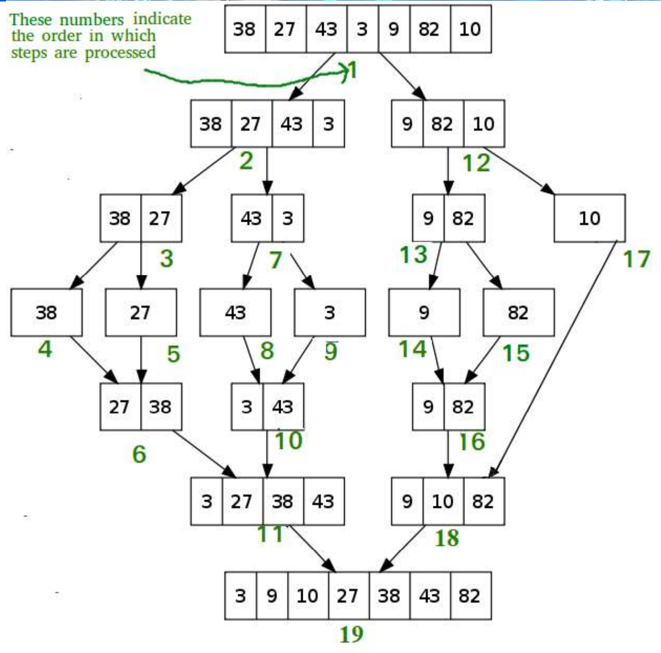
```
procedure insertionSort(arr):
   for i = 1 to n-1
      key = arr[i]
      j = i-1
      while j >= 0 and arr[j] > key
         swap arr[j+1] with arr[j]
         j = j - 1
      end while
   end for
end function
```

Merge Sort

- ***Merge sort** is defined as a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.*

How it works?





Algorithm

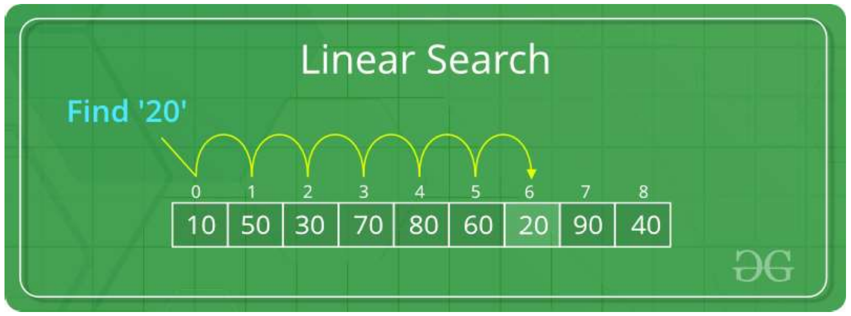
```
step 1: start
step 2: declare array and left, right, mid variable
step 3: perform merge function.
    if left > right
        return
    mid= (left+right)/2
    mergesort(array, left, mid)
    mergesort(array, mid+1, right)
    merge(array, left, mid, right)
step 4: Stop
```

Searching Algorithms

Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored.

Sequential Search

In this, the list or array is traversed sequentially and every element is checked. For example: Linear Search



Linear search Algorithm

Linear Search (Array A, Value x)

```
Step 1: Set i to 1
Step 2: if i > n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit
```

Linear search Pseudocode

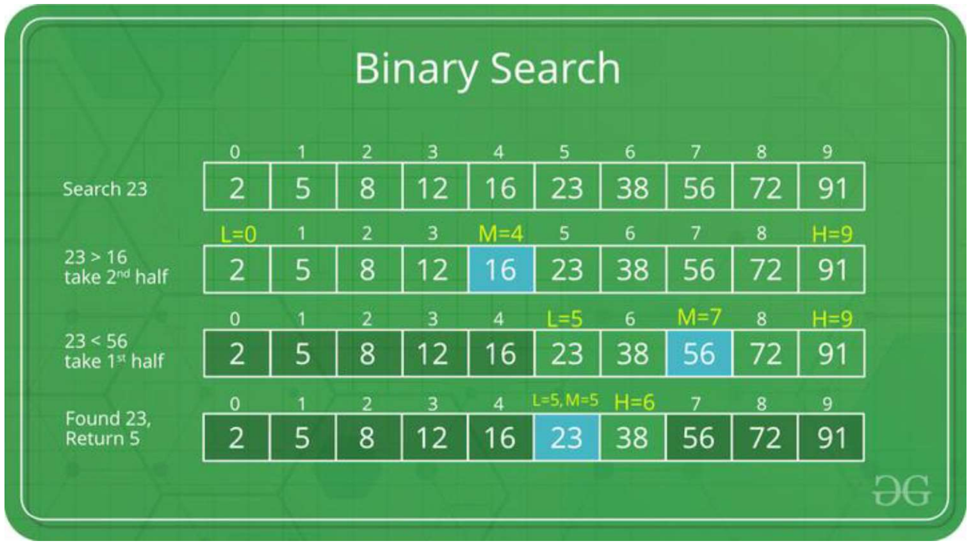
```
procedure linear_search (list, value)

  for each item in the list
    if match item == value
      return the item's location
    end if
  end for

end procedure
```

Interval Search

These algorithms are specifically designed for searching in sorted data-structures.

These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half.

For Example: Binary Search



Binary Search Algorithm

Binary_Search(a, lower_bound, upper_bound, val) // 'a' is the given array, 'lower_bound' is the index of the first array element, 'upper_bound' is the index of the last array element, 'val' is the value to search

```
Step 1: set beg = lower_bound, end = upper_bound, pos = - 1
Step 2: repeat steps 3 and 4 while beg <=end
Step 3: set mid = (beg + end)/2
Step 4: if a[mid] = val
set pos = mid
print pos
go to step 6
```

```
else if a[mid] > val
set end = mid - 1
else
set beg = mid + 1
[end of if]
[end of loop]
Step 5: if pos = -1
print "value is not present in the array"
[end of if]
Step 6: exit
```

Binary Search Pseudocode

```
Procedure binary_search
  A ← sorted array
  n ← size of array
  x ← value to be searched

  Set lowerBound = 1
  Set upperBound = n

  while x not found
    if upperBound < lowerBound
      EXIT: x does not exists.

    set midPoint = lowerBound + ( upperBound - lowerBound ) / 2
```

```
if A[midPoint] < x
      set lowerBound = midPoint + 1

   if A[midPoint] > x
      set upperBound = midPoint - 1

   if A[midPoint] = x
      EXIT: x found at location midPoint
 end while
```

Thank you..

HNDIT2022-Software Development

Week 8 & 9: File Handling

File Handling

- Many programming languages allow the use of files, in Fortran, ALGOLS0 and a few other programming languages file handling is not developed.
- In most of languages including COBOL, PL/i, RPG, C, C++, C#, Python and Java are a more or less complete set of instructions is provided for file handling.

Why File Handling required?

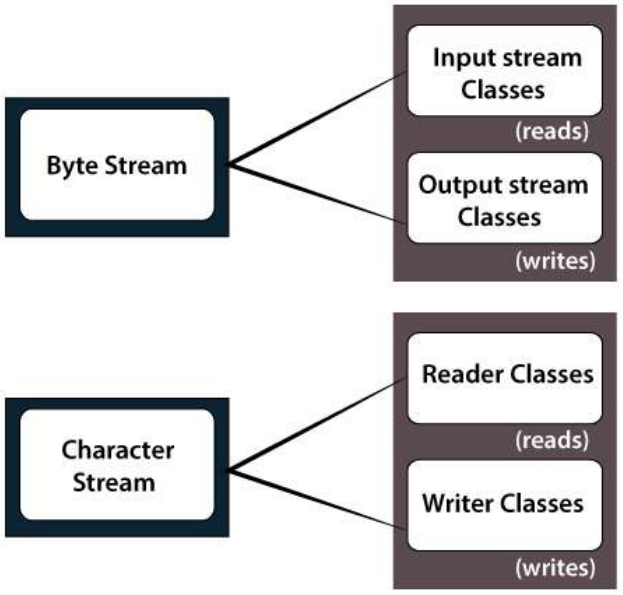
- File Handling is an integral part of any programming language as file handling enables us to store the output of any particular program in a file and allows us to perform certain operations on it.
- In simple words, file handling means reading and writing data to a file.

File Handling in Java

- In Java, a **File** is an abstract data type. A named location used to store related information is known as a **File**.
- There are several **File Operations** like **creating a new File, getting information about File, writing into a File, reading from a File** and **deleting a File**.

Stream

- A series of data is referred to as **a stream**.
- In Java, **Stream** is classified into two types, i.e., **Byte Stream** and **Character Stream**.
- This concept is used to perform I/O operations on a file.



Brief classification of I/O streams

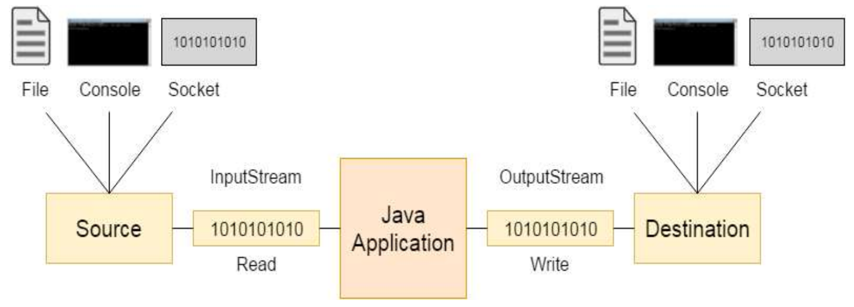
Byte Stream

- **Byte Stream** is mainly involved with byte data.
- A file handling process with a byte stream is a process in which an input is provided and executed with the byte data.
- The byte stream is again subdivided into two types which are as follows:
  - **Byte Input Stream:** Used to read byte data from different devices.
  - **Byte Output Stream:** Used to write byte data to different devices.

Character Stream

- **Character Stream** is mainly involved with character data.
- A file handling process with a character stream is a process in which an input is provided and executed with the character data.
- There are several subclasses of the OutputStream class which are as follows:
  - ByteArrayOutputStream
  - FileOutputStream
  - StringBufferOutputStream
  - ObjectOutputStream
  - DataOutputStream
  - PrintStream

Input Stream vs Output Stream



Input Stream

- The Java InputStream class is the superclass of all input streams. The input stream is used to read data from numerous input devices like the keyboard, network, etc.
- InputStream is an abstract class, and because of this, it is not useful by itself. However, its subclasses are used to read data.
- There are several subclasses of the InputStream class, which are as follows:
  - AudioInputStream
  - ByteArrayInputStream
  - FileInputStream
  - FilterInputStream
  - StringBufferInputStream
  - ObjectInputStream

Methods of Input Stream

S No.	Method	Description
1	read()	Reads one byte of data from the input stream.
2	read(byte[] array)()	Reads byte from the stream and stores that byte in the specified array.
3	mark()	It marks the position in the input stream until the data has been read.
4	available()	Returns the number of bytes available in the input stream.
5	markSupported()	It checks if the mark() method and the reset() method is supported in the stream.
6	reset()	Returns the control to the point where the mark was set inside the stream.
7	skips()	Skips and removes a particular number of bytes from the input stream.
8	close()	Closes the input stream.

Output Stream

- The output stream is used to write data to numerous output devices like the monitor, file, etc. OutputStream is an abstract superclass that represents an output stream. OutputStream is an abstract class and because of this, it is not useful by itself. However, its subclasses are used to write data.
- There are several subclasses of the OutputStream class which are as follows:
  - ByteArrayOutputStream
  - FileOutputStream
  - StringBufferOutputStream
  - ObjectOutputStream
  - DataOutputStream
  - PrintStream

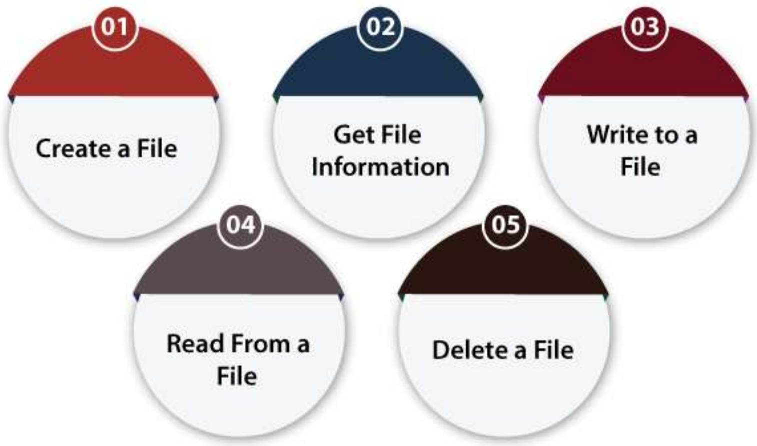
Methods of Output Stream

S. No.	Method	Description
1.	write()	Writes the specified byte to the output stream.
2.	write(byte[] array)	Writes the bytes which are inside a specific array to the output stream.
3.	close()	Closes the output stream.
4.	flush()	Forces to write all the data present in an output stream to the destination.

Java File Class Methods

S.No.	Method	Return Type	Description
1.	canRead()	Boolean	The **canRead()** method is used to check whether we can read the data of the file or not.
2.	createNewFile()	Boolean	The **createNewFile()** method is used to create a new empty file.
3.	canWrite()	Boolean	The **canWrite()** method is used to check whether we can write the data into the file or not.
4.	exists()	Boolean	The **exists()** method is used to check whether the specified file is present or not.
5.	delete()	Boolean	The **delete()** method is used to delete a file.
6.	getName()	String	The **getName()** method is used to find the file name.
7.	getAbsolutePath()	String	The **getAbsolutePath()** method is used to get the absolute pathname of the file.
8.	length()	Long	The **length()** method is used to get the size of the file in bytes.
9.	list()	String[]	The **list()** method is used to get an array of the files available in the directory.
10.	mkdir()	Boolean	The **mkdir()** method is used for creating a new directory.

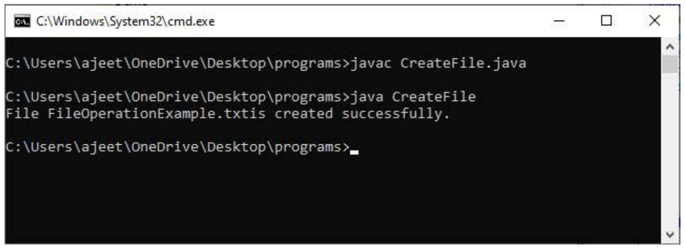
File Operations in Java



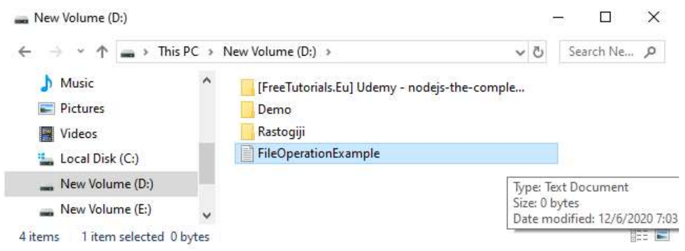
Create a file

- **Create a File** operation is performed to create a new file.
- We use the **createNewFile()** method of file.
- The **createNewFile()** method returns true when it successfully creates a new file and returns false when the file already exists.

```java
// Importing File class
import java.io.File;
// Importing the IOException class for handling errors
import java.io.IOException;
 class CreateFile {
         public static void main(String args[]) {
         try {
               // Creating an object of a file
               File f0 = new File("D:FileOperationExample.txt");
               if (f0.createNewFile()) {
                       System.out.println("File " + f0.getName() + " is created successfully.");
               } else {
                       System.out.println("File is already exist in the directory.");
               }
             } catch (IOException exception) {
                     System.out.println("An unexpected error is occurred.");
                     exception.printStackTrace();
             }
       }
}
```



```
C:\Users\ajeet\OneDrive\Desktop\programs>javac CreateFile.java

C:\Users\ajeet\OneDrive\Desktop\programs>java CreateFile
File FileOperationExample.txtis created successfully.

C:\Users\ajeet\OneDrive\Desktop\programs>
```



New Volume (D:)

This PC › New Volume (D:) ›

[FreeTutorials.Eu] Udemy - nodejs-the-comple...
Demo
Rastogiji
FileOperationExample

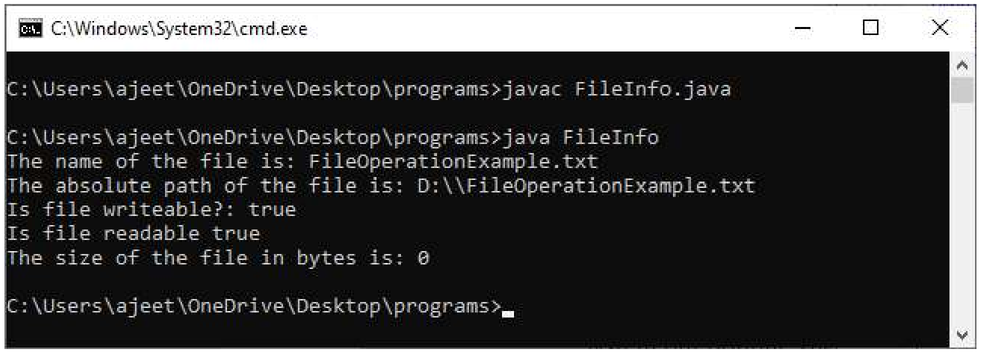
Type: Text Document
Size: 0 bytes
Date modified: 12/6/2020 7:03

4 items 1 item selected 0 bytes

Get File Information

- The operation is performed to get the file information.
- We use several methods to get the information about the file like name, absolute path, is readable, is writable and length.

```
// Import the File class
import java.io.File;
class FileInfo {
   public static void main(String[] args) {
      // Creating file object
      File f0 = new File("D:FileOperationExample.txt");
      if (f0.exists()) {
         // Getting file name
         System.out.println("The name of the file is: " + f0.getName());

         // Getting path of the file
         System.out.println("The absolute path of the file is: " + f0.getAbsolutePath());

         // Checking whether the file is writable or not
         System.out.println("Is file writeable?: " + f0.canWrite());

         // Checking whether the file is readable or not
         System.out.println("Is file readable " + f0.canRead());

         // Getting the length of the file in bytes
         System.out.println("The size of the file in bytes is: " + f0.length());
      } else {
         System.out.println("The file does not exist.");
      }
   }
}
```

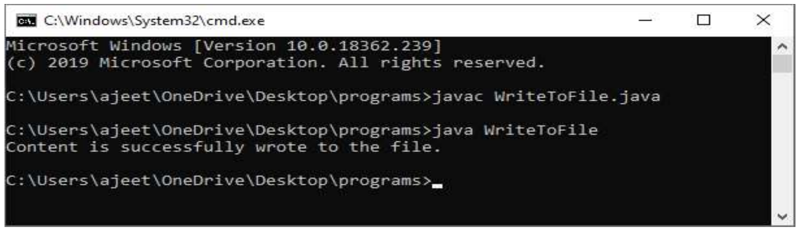


```
C:\Users\ajeet\OneDrive\Desktop\programs>javac FileInfo.java

C:\Users\ajeet\OneDrive\Desktop\programs>java FileInfo
The name of the file is: FileOperationExample.txt
The absolute path of the file is: D:\\FileOperationExample.txt
Is file writeable?: true
Is file readable true
The size of the file in bytes is: 0

C:\Users\ajeet\OneDrive\Desktop\programs>_
```

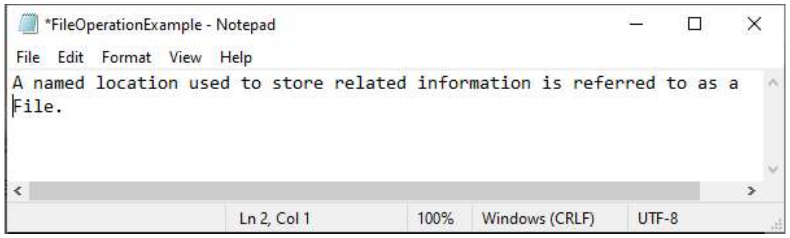
Write a File

- The next operation which we can perform on a file is **"writing into a file"**.
- In order to write data into a file, we will use the **FileWriter** class and its **write()** method together.
- We need to close the stream using the **close()** method to retrieve the allocated resources.

```
// Importing the FileWriter class
import java.io.FileWriter;

// Importing the IOException class for handling errors
import java.io.IOException;

class WriteToFile {
   public static void main(String[] args) {

   try {
      FileWriter fwrite = new FileWriter("D:FileOperationExample.txt");
      // writing the content into the FileOperationExample.txt file
      fwrite.write("A named location used to store related information is referred to as a File.");

      // Closing the stream
      fwrite.close();
      System.out.println("Content is successfully wrote to the file.");
   } catch (IOException e) {
      System.out.println("Unexpected error occurred");
      e.printStackTrace();
      }
   }
}
```



```
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet\OneDrive\Desktop\programs>javac WriteToFile.java

C:\Users\ajeet\OneDrive\Desktop\programs>java WriteToFile
Content is successfully wrote to the file.

C:\Users\ajeet\OneDrive\Desktop\programs>_
```

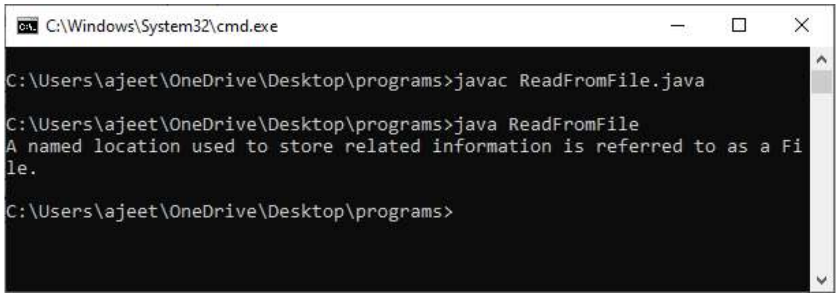


```
A named location used to store related information is referred to as a
File.
```

Read from a file

- The next operation which we can perform on a file is **"read from a file"**.
- In order to write data into a file, we will use the **Scanner** class.
- Here, we need to close the stream using the **close()** method.
- We will create an instance of the Scanner class and use the **hasNextLine()** method **nextLine()** method to get data from the file.

```
// Importing the File class
import java.io.File;
// Importing FileNotFoundException class for handling errors
import java.io.FileNotFoundException;
// Importing the Scanner class for reading text files
import java.util.Scanner;

class ReadFromFile {
    public static void main(String[] args) {
        try {
            // Create f1 object of the file to read data
            File f1 = new File("D:FileOperationExample.txt");
            Scanner dataReader = new Scanner(f1);
            while (dataReader.hasNextLine()) {
                String fileData = dataReader.nextLine();
                System.out.println(fileData);
            }
            dataReader.close();
        } catch (FileNotFoundException exception) {
            System.out.println("Unexcpected error occurred!");
            exception.printStackTrace();
        }
    }
}
```

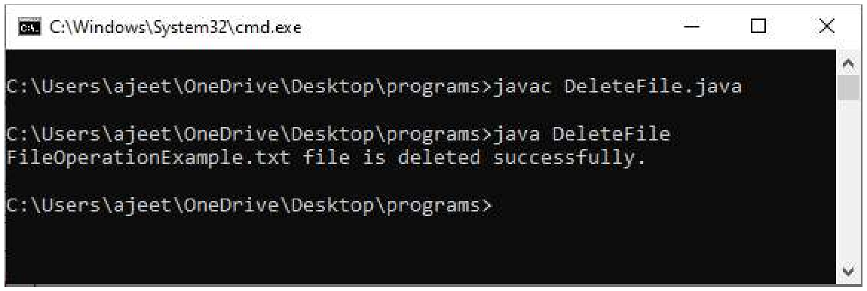


```
C:\Users\ajeet\OneDrive\Desktop\programs>javac ReadFromFile.java

C:\Users\ajeet\OneDrive\Desktop\programs>java ReadFromFile
A named location used to store related information is referred to as a Fi
le.

C:\Users\ajeet\OneDrive\Desktop\programs>
```

Delete a file

- The next operation which we can perform on a file is **"deleting a file"**.
- In order to delete a file, we will use the **delete()** method of the file.
- We don't need to close the stream using the **close()** method because for deleting a file, we neither use the FileWriter class nor the Scanner class.

```
// Importing the File class
import java.io.File;
class DeleteFile {
  public static void main(String[] args) {
    File f0 = new File("D:FileOperationExample.txt");
    if (f0.delete()) {
      System.out.println(f0.getName()+ " file is deleted 
successfully.");
    } else {
      System.out.println("Unexpected error found in deletion of 
the file.");
    }
  }
}
```



```
C:\Users\ajeet\OneDrive\Desktop\programs>javac DeleteFile.java

C:\Users\ajeet\OneDrive\Desktop\programs>java DeleteFile
FileOperationExample.txt file is deleted successfully.

C:\Users\ajeet\OneDrive\Desktop\programs>
```

Thank you..

HNDIT2022-
Software
Development

Week 10 & 11: Introduction to concept of user interface design

The User Interface

- System users often judge a system by its interface rather than its functionality
- A poorly designed interface can cause a user to make catastrophic errors
- Poor user interface design is the reason why so many software systems are never used

Graphical user interfaces

- Most users of business systems interact with these systems through graphical interfaces although, in some cases, legacy text-based interfaces are still used

GUI characteristics

Characteristic	**Description**
Windows	Multiple windows allow different information to be displayed simultaneously on the user's screen.
Icons	Icons different types of information. On some systems, icons represent files; on others, icons represent processes.
Menus	Commands are selected from a menu rather than typed in a command language.
Pointing	A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interest in a window.
Graphics	Graphical elements can be mixed with text on the same display.

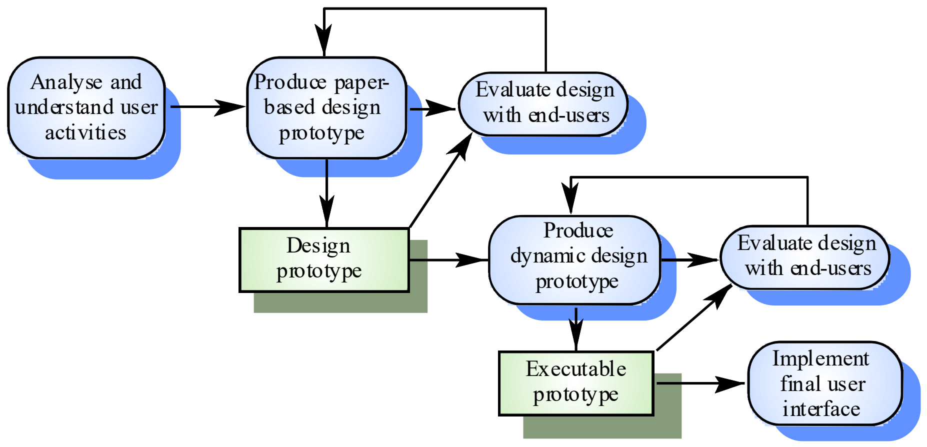
GUI advantages

- They are easy to learn and use.
  - Users without experience can learn to use the system quickly.
- The user may switch quickly from one task to another and can interact with several different applications.
  - Information remains visible in its own window when attention is switched.
- Fast, full-screen interaction is possible with immediate access to anywhere on the screen

User-centred design

- User-centred design is an approach to UI design where the needs of the user are paramount and where the user is involved in the design process
- UI design always involves the development of prototype interfaces

User interface design process



Analyse and understand user activities
Produce paper-based design prototype
Evaluate design with end-users
Design prototype
Produce dynamic design prototype
Evaluate design with end-users
Executable prototype
Implement final user interface

UI design principles

- UI design must take account of the needs, experience and capabilities of the system users
- Designers should be aware of people’s physical and mental limitations (e.g. limited short-term memory) and should recognise that people make mistakes
- UI design principles underlie interface designs although not all principles are applicable to all designs

User interface design principles

Principle	Description
User familiarity	The interface should use terms and concepts which are drawn from the experience of the people who will make most use of the system.
Consistency	The interface should be consistent in that, wherever possible, comparable operations should be activated in the same way.
Minimal surprise	Users should never be surprised by the behaviour of a system.
Recoverability	The interface should include mechanisms to allow users to recover from errors.
User guidance	The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities.
User diversity	The interface should provide appropriate interaction facilities for different types of system user.

Design principles

- User familiarity
  - The interface should be based on user-oriented terms and concepts rather than computer concepts. For example, an office system should use concepts such as letters, documents, folders etc. rather than directories, file identifiers, etc.
- Consistency
  - The system should display an appropriate level of consistency. Commands and menus should have the same format, command punctuation should be similar, etc.
- Minimal surprise
  - If a command operates in a known way, the user should be able to predict the operation of comparable commands

Design principles

- Recoverability
  - The system should provide some resilience to user errors and allow the user to recover from errors. This might include an undo facility, confirmation of destructive actions, 'soft' deletes, etc.
- User guidance
  - Some user guidance such as help systems, on-line manuals, etc. should be supplied
- User diversity
  - Interaction facilities for different types of user should be supported. For example, some users have seeing difficulties and so larger text should be available

Choosing Interface Elements

- **Input Controls**: buttons, text fields, checkboxes, radio buttons, dropdown lists, list boxes, toggles, date field
- **Navigational Components**: breadcrumb, slider, search field, pagination, slider, tags, icons
- **Informational Components**: tooltips, icons, progress bar, notifications, message boxes, modal windows
- **Containers**: accordion

User-system interaction

- Two problems must be addressed in interactive systems design
  - How should information from the user be provided to the computer system?
  - How should information from the computer system be presented to the user?
- User interaction and information presentation may be integrated through a coherent framework such as a user interface metaphor

Interaction styles

- Direct manipulation
- Menu selection
- Form fill-in
- Command language
- Natural language

Interaction style	Main advantages	Main disadvantages	Application examples
Direct manipulation	Fast and intuitive interaction Easy to learn	May be hard to implement Only suitable where there is a visual metaphor for tasks and objects	Video games CAD systems
Menu selection	Avoids user error Little typing required	Slow for experienced users Can become complex if many menu options	Most general-purpose systems
Form fill-in	Simple data entry Easy to learn	Takes up a lot of screen space	Stock control, Personal loan processing
Command language	Powerful and flexible	Hard to learn Poor error management	Operating systems, Library information retrieval systems
Natural language	Accessible to casual users Easily extended	Requires more typing Natural language understanding systems are unreliable	Timetable systems WWW information retrieval systems

Advantages and disadvantages

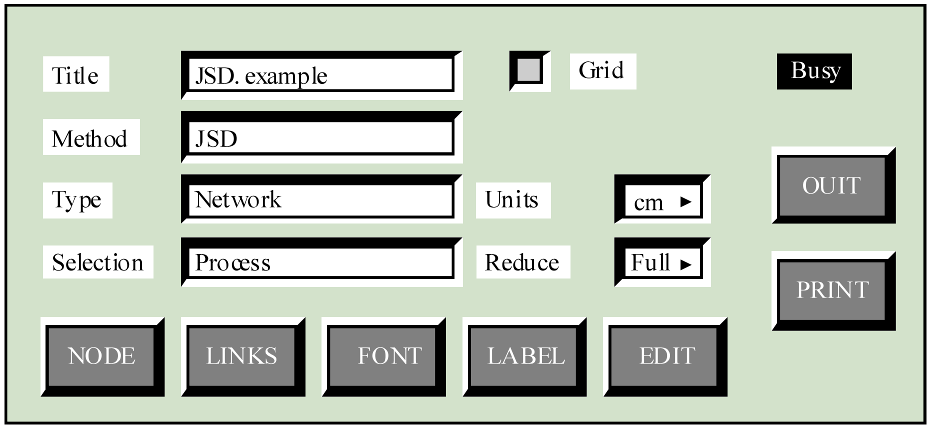
Direct manipulation advantages

- Users feel in control of the computer and are less likely to be intimidated by it
- User learning time is relatively short
- Users get immediate feedback on their actions so mistakes can be quickly detected and corrected

Direct manipulation problems

- The derivation of an appropriate information space model can be very difficult
- Given that users have a large information space, what facilities for navigating around that space should be provided?
- Direct manipulation interfaces can be complex to program and make heavy demands on the computer system

Control panel interface


Title: JSD. example
Method: JSD
Type: Network
Selection: Process
Grid
Busy
Units: cm
Reduce: Full

Menu systems

- Users make a selection from a list of possibilities presented to them by the system
- The selection may be made by pointing and clicking with a mouse, using cursor keys or by typing the name of the selection
- May make use of simple-to-use terminals such as touchscreens

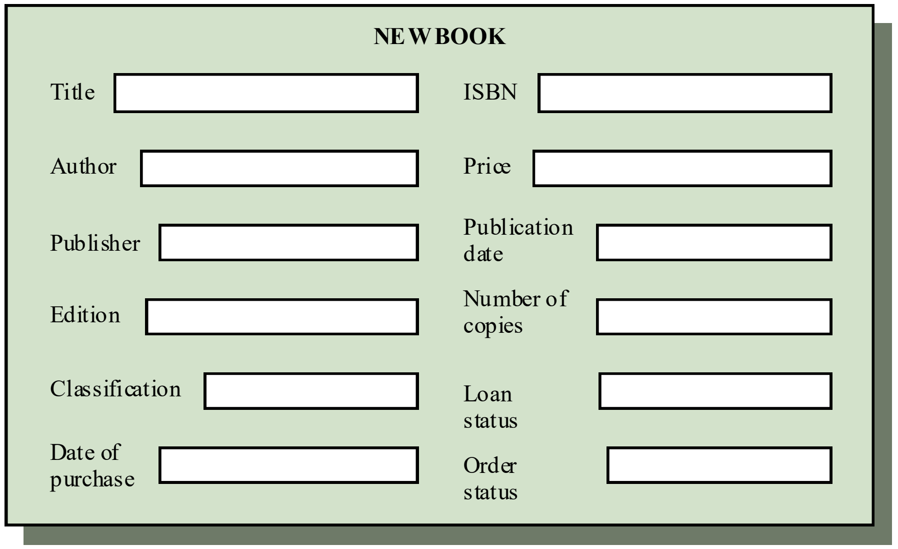
Advantages of menu systems

- Users need not remember command names as they are always presented with a list of valid commands
- Typing effort is minimal
- User errors are trapped by the interface
- Context-dependent help can be provided. The user’s context is indicated by the current menu selection

Problems with menu systems

- Actions which involve logical conjunction (and) or disjunction (or) are awkward to represent
- Menu systems are best suited to presenting a small number of choices. If there are many choices, some menu structuring facility must be used
- Experienced users find menus slower than command language

Form-based interface



NEWBOOK

Title		ISBN	
Author		Price	
Publisher		Publication date	
Edition		Number of copies	
Classification		Loan status	
Date of purchase		Order status	

Command interfaces

- User types commands to give instructions to the system e.g. UNIX
- May be implemented using cheap terminals.
- Easy to process using compiler techniques
- Commands of arbitrary complexity can be created by command combination
- Concise interfaces requiring minimal typing can be created

Problems with command interfaces

- Users have to learn and remember a command
  language. Command interfaces are therefore unsuitable for occasional users
- Users make errors in command. An error detection and recovery system is required
- System interaction is through a keyboard so typing ability is required

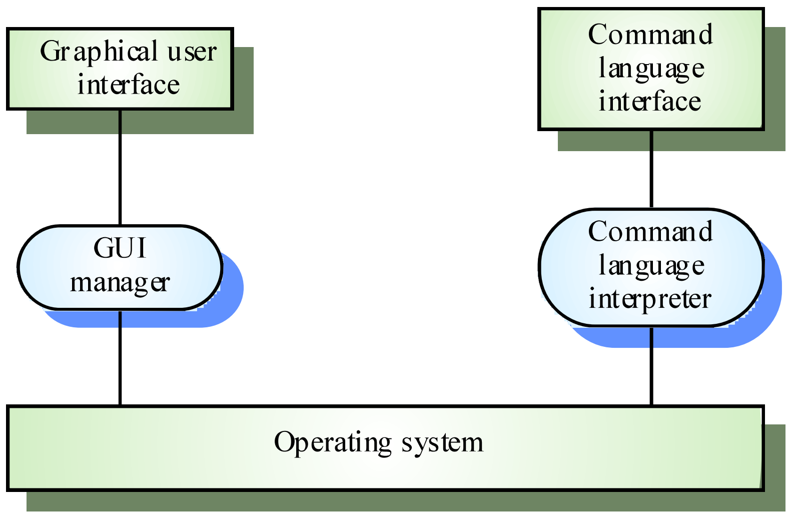
Command languages

- Often preferred by experienced users because they allow for faster interaction with the system
- Not suitable for casual or inexperienced users
- May be provided as an alternative to menu commands (keyboard shortcuts). In some cases, a command language interface and a menu-based interface are supported at the same time

Natural language interfaces

- The user types a command in a natural language. Generally, the vocabulary is limited and these systems are confined to specific application domains (e.g. timetable enquiries)
- NL processing technology is now good enough to make these interfaces effective for casual users but experienced users find that they require too much typing

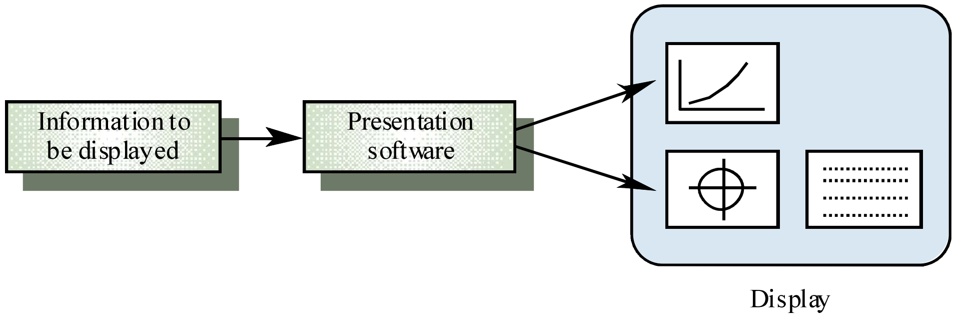
Multiple user interfaces



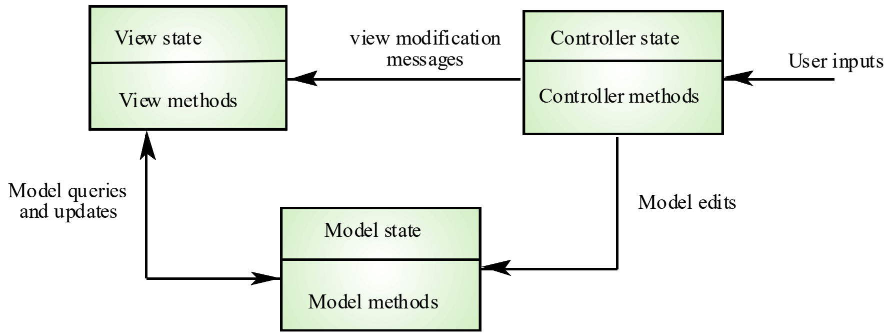
Information presentation

- Information presentation is concerned with presenting system information to system users
- The information may be presented directly (e.g. text in a word processor) or may be transformed in some way for presentation (e.g. in some graphical form)
- The Model-View-Controller approach is a way of supporting multiple presentations of data

Information presentation



Model-view-controller



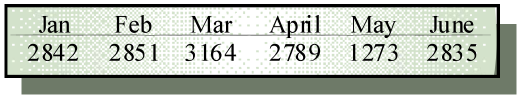
Information presentation

- Static information
  - Initialised at the beginning of a session. It does not change
    during the session
  - May be either numeric or textual
- Dynamic information
  - Changes during a session and the changes must be communicated to the system user
  - May be either numeric or textual

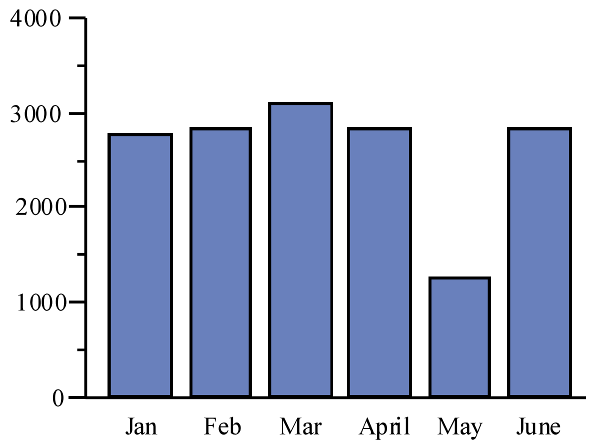
Information display factors

- Is the user interested in precise information or data relationships?
- How quickly do information values change? Must the change be indicated immediately?
- Must the user take some action in response to a change?
- Is there a direct manipulation interface?
- Is the information textual or numeric? Are relative values important?

Alternative information presentations



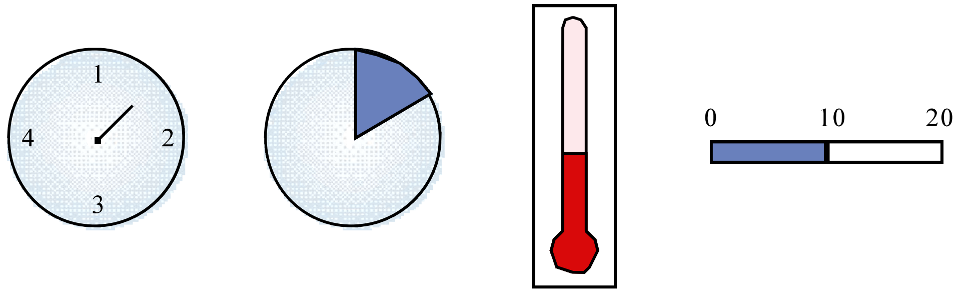
Jan	Feb	Mar	April	May	June
2842	2851	3164	2789	1273	2835



Analogue vs. digital presentation

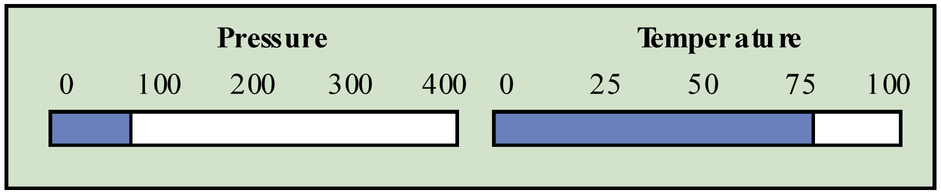
- Digital presentation
  - Compact - takes up little screen space
  - Precise values can be communicated
- Analogue presentation
  - Easier to get an 'at a glance' impression of a value
  - Possible to show relative values
  - Easier to see exceptional data values

Dynamic information display

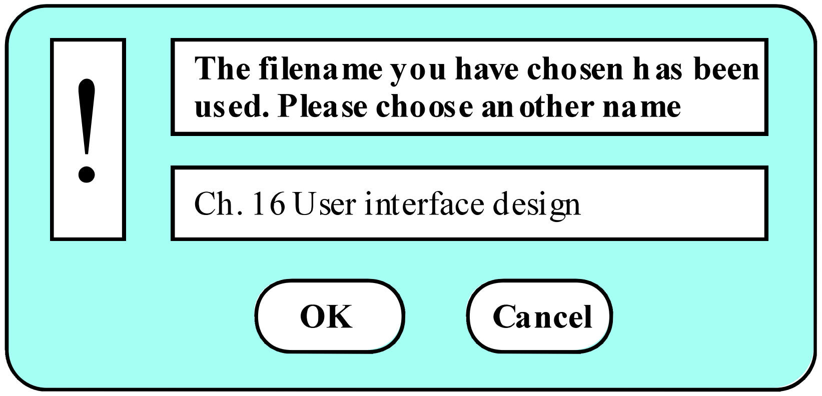


Dial with needle Pie chart Thermometer Horizontal bar

Displaying relative values



Textual highlighting


!

The filename you have chosen has been used. Please choose another name

Ch. 16 User interface design

Data visualisation

- Concerned with techniques for displaying large amounts of information
- Visualisation can reveal relationships between entities and trends in the data
- Possible data visualisations are:
  - Weather information collected from a number of sources
  - The state of a telephone network as a linked set of nodes
  - Chemical plant visualised by showing pressures and temperatures in a linked set of tanks and pipes
  - A model of a molecule displayed in 3 dimensions
  - Web pages displayed as a hyperbolic tree

Colour displays

- Colour adds an extra dimension to an interface and can help the user understand complex information structures
- Can be used to highlight exceptional events
- Common mistakes in the use of colour in interface design include:
  - The use of colour to communicate meaning
  - Over-use of colour in the display

Colour use guidelines

- Don't use too many colours
- Use colour coding to support use tasks
- Allow users to control colour coding
- Design for monochrome then add colour
- Use colour coding consistently
- Avoid colour pairings which clash
- Use colour change to show status change
- Be aware that colour displays are usually lower resolution

User support

- User guidance covers all system facilities to support users including on-line help, error messages, manuals etc.
- The user guidance system should be integrated with the user interface to help users when they need information about the system or when they make some kind of error
- The help and message system should, if possible, be integrated

Best Practices for Designing an Interface

- **Keep the interface simple.** The best interfaces are almost invisible to the user. They avoid unnecessary elements and are clear in the language they use on labels and in messaging.
- **Create consistency and use common UI elements.** By using common elements in your UI, users feel more comfortable and are able to get things done more quickly. It is also important to create patterns in language, layout and design throughout the site to help facilitate efficiency. Once a user learns how to do something, they should be able to transfer that skill to other parts of the site.
- **Be purposeful in page layout.** Consider the spatial relationships between items on the page and structure the page based on importance. Careful placement of items can help draw attention to the most important pieces of information and can aid scanning and readability.

- **Strategically use color and texture.** You can direct attention toward or redirect attention away from items using color, light, contrast, and texture to your advantage.
- **Use typography to create hierarchy and clarity.** Carefully consider how you use typeface. Different sizes, fonts, and arrangement of the text to help increase scanability, legibility and readability.
- **Make sure that the system communicates what's happening.** Always inform your users of location, actions, changes in state, or errors. The use of various UI elements to communicate status and, if necessary, next steps can reduce frustration for your user.
- **Think about the defaults.** By carefully thinking about and anticipating the goals people bring to your site, you can create defaults that reduce the burden on the user. This becomes particularly important when it comes to form design where you might have an opportunity to have some fields pre-chosen or filled out.

Thank you..

HNDIT2022- Software Development

Week 12 & 13: Role and need for System Software

System Software

- Software is a set of instructions, which is designed to perform a defined task, and it tells the computer how to work. It is of mainly two types, namely System software and Application software.
- *System software is a set of computer programs that is designed to manage system resources.*
- It is a collection of such files and utility programs that are responsible for running and smooth functioning of your computer system with other hardware. Moreover, it is solely responsible for running the operating system (OS) and managing the computer device entirely.
- System software acts as a platform for other software to work.

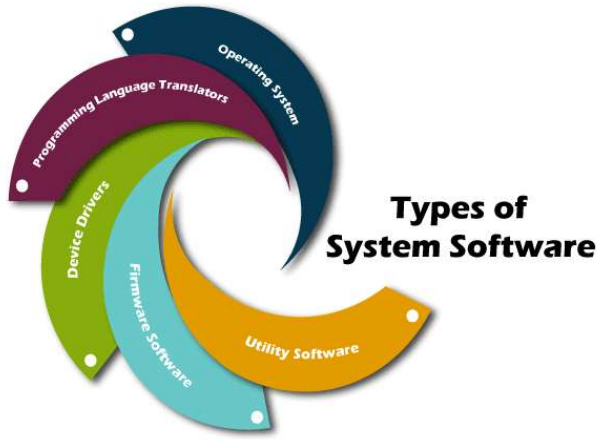
What is System Software?

- *System Software is the most important type of software required to administer the resources of the computer system.*
- System software runs and functions internally with application software and hardware. Moreover, it works as a linking interface between a hardware device and the end-user.
- System software runs in the background and manages all functioning of the computer itself. It is called *Low-Level Software* as it runs at the most basic level of computer and is usually written in a low-level language. As soon as we install the operating system on our device, it gets automatically installed on the same device.
- System software helps to generate the user interface and allows the operating system to interact with the computer hardware.

Features of System Software

- It is very difficult to design system software.
- System software is responsible to directly connect the computer with hardware that enables the computer to run.
- Difficulties in manipulation.
- It is smaller in size.
- System Software is difficult to understand.
- It is usually written in a low-level language.
- It must be as efficient as possible for the smooth functioning of the computer system.

Types of System Software



1. Operating System

- An Operating System is the most basic type of System Software that helps to manage computer hardware and software.
- It is the central part of any computer system which is responsible for the smooth functioning of any computer device.
- An Operating system primarily operates your computer when you start it. If you haven't installed the operating system on your computer, then you will not be able to start your computer. Some most common examples of OS are **macOS, Linux, Android, and Microsoft Windows.**

2. Programming Language Translators

- *Programming translators are the software that converts high-level language into machine language.* A computer can only understand the machine language or binary bits pattern, either 0 or 1.
- A CPU understands this machine language that is not easy to understand by a normal human. Hence, First, the end-user interacts with the computer in a high-level language like Java, Python, C, PHP, and C++, etc., then the translator converts these languages into machine code.
- A CPU or computer processor executes these machine codes into binary. It means any program written in a high-level programming language must be converted into binary codes first. This entire process to convert high-level language into machine code or binary codes are known as compilation.

- Language translators are of mainly two types: Compiler and Interpreter.
- A compiler is also a type of system software used to convert high-level programming languages into executable machine codes or low-level programming languages.
- Similar to a compiler, an Interpreter is also used to perform the same function. But the only difference is that compiler translates the whole program at once while the interpreter converts each line individually.

3. Device Drivers

- *Device Drivers are the types of system software that reduce the troubleshooting issues in your system*. The operating system communicates with hardware components internally. This communication can easily be managed and controlled with the help of device drivers.
- The operating system contains a number of device drivers to drive the hardware components. Most of the device drivers, such as a mouse, keyboards, etc., are already installed in the computer system by the computer manufacturing companies. However, in case of any new device for the operating system, users can install them through the internet also.

4. Firmware Software

- These are the operational software installed on the computer motherboards that help the operating system to identify the Flash, ROM, EPROM, EEPROM, and memory chips.
- However, the primary function of any firmware software is to manage and control all activities of individual devices. Initially, it uses non-volatile chips for installation purposes, but later it gets installed on the flash chips.

5. Utility Software

- Utility software works as an interface between system software and application software.
- Utility software is a third-party tool designed to reduce maintenance issues and detect errors in the computer system. It comes with the operating system in your computer system.
- Here are some specific features of utility software:
  - It helps users to protect against threats and viruses.
  - It helps to reduce disk size such as WinRAR, WinZip.
  - It works as a Windows Disk Management service and helps in a disk partition.
  - It facilitates users to back up the old data and enhance the security of the system.
  - It works as a de-fragmentation of a disk to organize the scattered files on the drive.
  - It helps to recover the lost data.
  - It helps to perform the antivirus and security software to maintain the security of the data in a computer.

Homework

- Write an article on the usages of different types of application software.

Thank you..

HNDIT2022- Software Development

Week 14: Case Studies in Problem Solving

Case Study

- A *case study* describes a programming problem, the process used by an expert to solve the problem, and one or more solutions to the problem.
- Case studies emphasize the decisions encountered by the programmer and the criteria used to choose among alternatives.
- Learners are engaged in the case study through questions that are interspersed in the presentation. Students can "think along" with the experts by making predictions, helping with parts of the solution, or analyzing alternatives. More comprehensive questions ask students to modify the solution, apply the ideas to new problems, detect bugs in related problems, and reflect on their own methods for solving problems.

What are the advantages of case studies?

- They model efficient ways to organize programming knowledge.
- They help students construct techniques and strategies that reduce or postpone the complexity of program design and development.
- They guide students to apply program design skills to large, complex problems and to learn context-dependent design strategies.
- They encourage students to reflect on completed solutions, comparing them to one another.
- They stimulate students to recognize their own strengths and weaknesses (in order to guard against the latter).
- They make large programs accessible to students, and thus give students a better picture of the nature of "real programming".
- They form the basis for assessment of a student's ability to design, understand, analyze, modify, and debug code that is both educational and effective.

Examples

- Example 1: Determining even/odd number
  - A number divisible by 2 is considered an even number, while a number which is not divisible by 2 is considered an odd number. Write pseudo-code to display first N odd/even numbers.
- Example 2: Computing Weekly Wages
  - Gross pay depends on the pay rate and the number of hours worked per week. However, if you work more than 40 hours, you get paid time-and-a-half for all hours worked over 40. Write the pseudo-code to compute gross pay given pay rate and hours worked

Homework

- Write an algorithm on the previous examples.

Thank you..

HNDIT2022-Software Development

Week 15: Algorithm Analysis

Algorithm Analysis

- Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem.
- Most algorithms are designed to work with inputs of arbitrary length. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.
- Usually, the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps, known as **time complexity**, or volume of memory, known as **space complexity**.

Why we need algorithm analysis?

- Analysis of algorithm is the process of analyzing the problem-solving capability of the algorithm in terms of the time and size required (the size of memory for storage while implementation). However, the main concern of analysis of algorithms is the required time or performance. Generally, we perform the following types of analysis –
- **Worst-case** – The maximum number of steps taken on any instance of size **a**.
- **Best-case** – The minimum number of steps taken on any instance of size **a**.
- **Average case** – An average number of steps taken on any instance of size **a**.

Asymptotic Notation and Analysis

- Asymptotic Analysis is *defined as the big idea that handles the above issues in analyzing algorithms.*
- *In Asymptotic Analysis, we* evaluate the performance of an algorithm in terms of input size *(we don't measure the actual running time). We calculate, how the time (or space) taken by an algorithm increases with the input size.*

- Asymptotic notation is a way to describe the running time or space complexity of an algorithm based on the input size. It is commonly used in complexity analysis to describe how an algorithm performs as the size of the input grows. The three most commonly used notations are Big O, Omega, and Theta.
  - Big O notation (O): This notation provides an upper bound on the growth rate of an algorithm’s running time or space usage. It represents the worst-case scenario, i.e., the maximum amount of time or space an algorithm may need to solve a problem. For example, if an algorithm’s running time is O(n), then it means that the running time of the algorithm increases linearly with the input size n or less.
  - Omega notation (Ω): This notation provides a lower bound on the growth rate of an algorithm’s running time or space usage. It represents the best-case scenario, i.e., the minimum amount of time or space an algorithm may need to solve a problem. For example, if an algorithm’s running time is Ω(n), then it means that the running time of the algorithm increases linearly with the input size n or more.
  - Theta notation (Θ): This notation provides both an upper and lower bound on the growth rate of an algorithm’s running time or space usage. It represents the average-case scenario, i.e., the amount of time or space an algorithm typically needs to solve a problem. For example, if an algorithm’s running time is Θ(n), then it means that the running time of the algorithm increases linearly with the input size n.

Example 1: Printing each element of an array

```
for (var i = 0; i < a.length; i++) {
   System.out.println(a[i]);
}
```

Here n=a.length (provided we know that all of the items in the array have a fixed size, which is often the case). The abstract operation of interest here is the print. We print n times. The algorithm has n steps. So we say T(n)=n.

Example 2

```
for (var i = 0; i < a.length; i+=2) {
   System.out.println(a[i]);
}
```

Again we are assuming we live in a world in which n=a.length. And again the abstract operation is the print. We can see that T(n)=n/2. This can also be written T(n)=n/2 .

Example 3: Print the last element of an array

```
if (a.length > 0) {
   return a[a.length - 1];
} else {
   throw new NoSuchElementException();
}
```

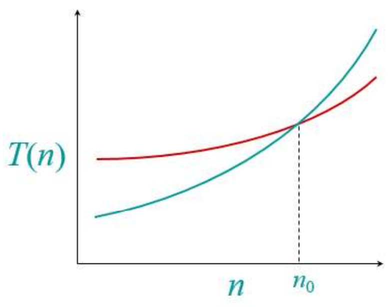
One step, **no matter how big the array is**! Here n=a.length, and T(n)=1. This is the best you can get. So good.

Example 4: Basic Manipulations

- Drop low-order terms; ignore leading constants.
- $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

Asymptotic Performance

- When n gets large enough, a $\Theta(n^2)$ algorithm *always* beats a $\Theta(n^3)$ algorithm.



- Asymptotic analysis is a useful tool to help to structure our thinking toward better algorithm
- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing

Thank you..