[Concepts] Paper 3

CHAPTER 1: DATA REPRESENTATION

1.1 USER-DEFINED DATA TYPES

1.1.1 Introduction to User-Defined Data Types

Definition: User-defined data types are data types designed by the programmer. When object-oriented programming is not being used, a programmer may choose to utilize user-defined data types for a large program as their use can reduce errors and make the program more understandable.

Why Use User-Defined Data Types:

- Reduce errors in the program
- Make code more understandable
- Less restriction than built-in types
- Allows for inevitable user definition

1.1.2 Non-Composite User-Defined Data Types

Enumerated Data Type: A non-composite user-defined data type that is a list of possible data values. The values defined have an implied order of values to allow comparisons.

<PSEUDOCODE>

```
TYPE Season = (Summer, Winter, Autumn, Spring)

DECLARE ThisSeason : Season
DECLARE NextSeason : Season
ThisSeason ← Autumn
NextSeason ← ThisSeason + 1 // NextSeason is set to Spring
```

Characteristics:

- Values are countable and finite
- Allow comparisons (value2 > value1)

- NOT string values (don't use quotes)

Pointer Data Type: Used to reference a memory location. It may be used to construct dynamically varying data structures. The pointer definition has to relate to the type of the variable being pointed to.

<PSEUDOCODE>

```
TYPE IntegerPointer = ^INTEGER

DECLARE IPointer : IntegerPointer
DECLARE MyInt1 : INTEGER
DECLARE MyInt2 : INTEGER

// Store address of MyInt2 in IPointer
IPointer ← @MyInt2

// Access data at address pointed to by IPointer
IPointer^ ← 33

// Store address of MyInt1 in SecondPointer
```

Pointer Notation:

- `@identifier` - returns the address of identifier
- `Pointer^` - accesses the data stored at the address (dereferencing)

1.1.3 Composite User-Defined Data Types

Record Data Type: A data type that contains a fixed number of components that can be of different types. Allows the programmer to collect values with other data types together when these form a coherent whole.

<PSEUDOCODE>

```
TYPE TEmployeeRecord
 DECLARE FirstName : STRING
 DECLARE LastName : STRING
 DECLARE Salary : REAL
 DECLARE Position : STRING
ENDTYPE

DECLARE Employee1 : TEmployeeRecord
```

Set Data Type: Allows a program to create sets and to apply mathematical operations defined in set theory. All elements in the set should be unique.

<PSEUDOCODE>

```
TYPE Days = SET OF STRING
DEFINE Today(Monday, Tuesday, Wednesday, Thursday, Friday) : Days
```

Operations on Sets:

- Union
- Difference
- Intersection
- Include an element
- Exclude an element
- Check whether an element is in a set

Class (in OOP): In object-oriented programming, a program defines the classes to be used. Then, for each class, the objects must be defined. A Class includes variables and methods (functions or procedures that an object can run).

1.2 FILE ORGANISATION AND ACCESS

1.2.1 Types of Files

Text Files:

- Contains data stored according to a defined character code (ASCII or Unicode)
- Can be created using a text editor
- Data appears as readable characters

Binary Files:

- Designed for storing data to be used by a computer program
- Stores data in its internal representation
- Created using specific programs
- Structure: File → Records → Fields → Values

1.2.2 Methods of File Organisation

Serial Files:

- Records have no defined order
- Stored one after another in the order they were added
- New records added at the end of the file
- No end of record character (must have defined format)

Advantages:

- Simple task
- Low cost

Disadvantages:

- Difficult to access specific records
- Must read all preceding records
- Cannot support modern high-speed requirements

File Access: Sequential access only

Uses:

- Batch processing
- Backing up data on magnetic tape
- Bank transactions

Sequential Files:

- Records ordered using a key field
- Key field values must be unique and ordered
- More efficient than serial files due to data integrity
- New records must be added in correct position

File Access Methods:

1. **Sequential Access:** Read key field values until required value found
2. **Direct Access:** Use index to look up address of record location

Random Files:

- Records stored randomly
- Accessed directly using hashing algorithm
- Uses hashing on record's key field to calculate address
- Well suited for magnetic and optical disks

Advantages:

- Quick retrieval of records
- Records may vary in size

1.2.3 Hashing Algorithms

Definition: Takes the key field as input and outputs a value for the record's position relative to the file's start.

Example:

```
<TEXT>
If key field is numeric: Divide by suitable large number and use remainder
Position = Key MOD FileSize
```

Handling Collisions: When two different keys produce the same position, use the next available position in the file.

File Access:

- Value in key field submitted to hashing algorithm
- Algorithm provides position in file
- May require short linear search due to collisions

1.3 FLOATING-POINT NUMBERS

1.3.1 Floating-Point Representation

Definition: The approximate representation of a real number using binary digits.

Format:

<TEXT>

```
Number = ±Mantissa × Base^Exponent
```

- **Mantissa:** The non-zero part of the number
- **Exponent:** The power to which the base is raised
- **Base:** 2 (in binary floating-point)

1.3.2 Converting Denary to Floating-Point Binary

Steps:

1. Convert whole number part to binary
2. Add sign bit (0 for positive, 1 for negative)
3. Convert fractional part by multiplying by 2 and recording whole parts
4. Combine parts and adjust exponent
5. Normalise the number

Example: Converting 8.75 to floating-point (8-bit: 4 for mantissa, 4 for exponent)

<TEXT>

```
Step 1: Convert whole number
8 = 1000

Step 2: Convert fractional part
0.75 × 2 = 1.5 → 1
0.5 × 2 = 1.0 → 1
0.0 × 2 = 0 → 0
0.75 = 0.11 (binary)

Step 3: Combine
8.75 = 1000.11

Step 4: Normalise
1000.11 = 0.100011 × 2^4

Step 5: Represent
Mantissa: 10001100 (0.100011 with padding)
Exponent: 0100 (4 in 4-bit two's complement)
```

1.3.3 Normalisation

Purpose:

- Maximise precision using all available bits in mantissa
- Ensure most significant bits are different (0 1 for positive, 1 0 for negative)
- Avoid multiple representations of the same number

Process:

- For positive numbers: Shift left until first bit after binary point is 1
- For negative numbers: Shift left until first two bits are different (10)
- Each shift left decreases exponent by 1

1.3.4 Problems with Floating-Point Numbers

1. **Rounding Errors:**
   - Binary representation of fractions is often approximate
   - Can become significant after multiple calculations
2. **Overflow:**
   - Result too large to represent
   - Occurs when exponent exceeds maximum
3. **Underflow:**
   - Result too small to represent
   - May be rounded to zero
4. **Inability to Store Zero:**
   - Normalised form requires mantissa to be 0.1 or 1.0
   - Zero must be handled as special case

1.3.5 Precision vs Range

Increase	Effect
Mantissa bits	Better precision
Exponent bits	Larger range

Trade-off: Must balance precision and range based on application needs

CHAPTER 2: COMMUNICATION AND INTERNET TECHNOLOGIES

2.1 PROTOCOLS

2.1.1 Introduction to Protocols

Definition: A protocol is a set of rules governing communication between computers. It ensures the computers that communicate understand each other.

Key Terms:

- **MAC Address:** Unique number assigned to each device's networking hardware worldwide
- **IP Address:** Unique number assigned to each node/networking device in a network
- **Port Number:** Software-generated number specifying an application or process communication endpoint

2.1.2 TCP/IP Protocol Suite

Four Layers:

Layer	Purpose
Application	Encodes the data being sent
Transport	Breaks data into packets, adds port numbers
Network/Internet	Adds IP addresses for routing
Link	Adds MAC addresses, handles transmission
Physical	Converts to signals for transmission

Data Flow - Sender Side:

1. **Application Layer:** Encodes data in appropriate format
2. **Transport Layer:** Creates packets with port numbers
3. **Network Layer:** Adds sender and receiver IP addresses
4. **Link Layer:** Formats into frames, adds error checking
5. **Physical Layer:** Converts to signals for transmission

Data Flow - Receiver Side: Reverse of sender side, stripping headers at each layer

2.1.3 Key Protocols

Protocol	Full Name	Purpose
HTTP	Hyper Text Transfer Protocol	Handles transmission of data to/from websites
HTTPS	Hyper Text Transfer Protocol Secure	Secure HTTP with encryption
FTP	File Transfer Protocol	Handles file transmission across networks
SMTP	Simple Mail Transfer Protocol	Sending emails (push protocol)
POP3	Post Office Protocol 3	Receiving emails (pull protocol)
IMAP	Internet Message Access Protocol	Advanced email retrieval
BitTorrent	-	Peer-to-peer file sharing

2.1.4 BitTorrent Protocol

Components:

- **Torrent File:** Contains details regarding the tracker
- **Tracker:** Server that keeps track of peers
- **Peers:** Users downloading and uploading simultaneously
- **Swarm:** Network of peers sharing the torrent
- **Seeding:** Uploading file after/complete download
- **Leeching:** Downloading without uploading

How It Works:

1. Peers obtain torrent file (small)
2. Torrent file points to tracker
3. Tracker lists all peers in swarm
4. Peers download chunks from each other
5. Peers upload chunks to other peers

2.2 CIRCUIT SWITCHING AND PACKET SWITCHING

2.2.1 Circuit Switching

Definition: A method of data transfer where a dedicated communication channel is established before transmission begins.

Characteristics:

- Dedicated path between sender and receiver
- Path remains open for entire duration
- Like traditional telephone system

Advantages:

- Guaranteed bandwidth
- Consistent quality
- No delay from routing decisions

Disadvantages:

- Inefficient if data is sporadic
- Connection setup takes time
- Line unavailable if circuit busy

Example Use:

- Traditional landline phones

2.2.2 Packet Switching

Definition: A method of data transfer where the message is broken into parts and sent over optimum routes to reach its destination.

Characteristics:

- Data divided into packets
- Each packet can take different route
- Packets reassembled at destination
- Used in the Internet

Advantages:

- Efficient use of network capacity
- If one route fails, packets can reroute
- Better for bursty data

Disadvantages:

- Packets may arrive out of order
- Variable delays
- More complex infrastructure

2.2.3 Router Function

Definition: A device that connects two or more computer networks and directs incoming packets to their receiver according to network traffic.

Functions:

- Examines packet destination IP address
- Determines best route for each packet
- Manages network traffic congestion
- Connects LAN to WAN

2.3 SSL/TLS

2.3.1 SSL and TLS

SSL (Secure Socket Layer):

- Provides secure communication
- Functions between TCP and application layer

- Creates "socket" for secure connection

TLS (Transport Layer Security):

- Improved version of SSL
- Provides encryption, compression, integrity checking

When to Use:

- Online shopping websites
- Online banking
- Any site requiring secure data transmission

Handshake Process:

1. Client sends request to server
2. Server sends digital certificate (includes public key)
3. Client validates certificate
4. Client generates session key, encrypts with server's public key
5. Server decrypts session key
6. Secure session established

CHAPTER 3: HARDWARE AND VIRTUAL MACHINES

3.1 RISC AND CISC PROCESSORS

3.1.1 RISC (Reduced Instruction Set Computers)

Characteristics:

- Fewer instructions
- Simpler instructions
- Small number of instruction formats
- Single-cycle instructions where possible
- Fixed-length instructions
- Only load/store instructions access memory
- Fewer addressing modes
- Multiple register sets
- Hard-wired control unit
- Pipelining easier

3.1.2 CISC (Complex Instruction Set Computers)

Characteristics:

- More instructions
- Complicated instructions
- Many instruction formats
- Multi-cycle instructions
- Variable-length instructions
- Many types of memory addressing instructions
- More addressing modes
- Fewer registers
- Microprogrammed control unit

- Pipelining difficult

3.1.3 Pipelining

Definition: Instruction-level parallelism where the fetch-decode-execute cycle is separated into several stages.

Five Stages:

1. Instruction Fetch (IF)
2. Instruction Decode (ID)
3. Operand Fetch (OF)
4. Instruction Execution (IE)
5. Result Write Back (WB)

Advantages:

- Multiple instructions processed simultaneously
- Increased throughput
- More efficient use of processor components

Disadvantages:

- Interrupt handling complex
- Pipeline stalls possible

Interrupt Handling in Pipelines:

- When interrupt occurs, must clear pipeline
- Store current state of all instructions in pipeline
- Execute ISR
- Resume interrupted instructions

3.1.4 Parallel Processing Architectures

SISD (Single Instruction Single Data):

- Single processor
- Early computers
- No pipelining

SIMD (Single Instruction Multiple Data):

- Multiple processors
- Array processors
- Same instruction on multiple data points

MISD (Multiple Instruction Single Data):

- Multiple processors
- Same data, different instructions
- Used for sorting

MIMD (Multiple Instruction Multiple Data):

- Modern computers
- Each processor executes different instructions
- Most common parallel architecture

Massively Parallel Computers:

- Vast amounts of processing power
- Bus structure for multiple processors
- Network infrastructure for multiple hosts
- Used for complex mathematical problems

3.2 BOOLEAN ALGEBRA AND LOGIC CIRCUITS

3.2.1 Boolean Algebra Laws

Identity Law:

```
<TEXT>
A + 0 = A
A · 1 = A
```

Null Law:

```
<TEXT>
A + 1 = 1
A · 0 = 0
```

Idempotent Law:

<TEXT>

A + A = A

A · A = A

Inverse Law:

<TEXT>

A + A' = 1

A · A' = 0

Commutative Law:

<TEXT>

A + B = B + A

A · B = B · A

Associative Law:

<TEXT>

(A + B) + C = A + (B + C)

(A · B) · C = A · (B · C)

Distributive Law:

<TEXT>

A + B·C = (A + B) · (A + C)

A · (B + C) = A·B + A·C

De Morgan's Laws:

<TEXT>

(A + B)' = A' · B'

(A · B)' = A' + B'

3.2.2 Karnaugh Maps (K-Maps)

Purpose: Simplify Boolean expressions and reduce number of gates needed.

Benefits:

- Minimises Boolean expressions
- Minimises logic gates
- More efficient circuits

Methodology:

1. Fill K-map from truth table
2. Group '1's in powers of 2 (1, 2, 4, 8)
3. Wrap around edges allowed
4. Within each group, only constant values remain
5. Write expression from groups

Example K-map (2 variables):

<TEXT>

```
 AB'
 0 1
A 0 0 1
 1 1 1
```

3.2.3 Half Adders and Full Adders

Half Adder:

- Adds two bits
- Outputs sum and carry

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0

A	B	Sum	Carry
1	1	0	1

Sum = A XOR B Carry = A AND B

Full Adder:

- Adds three bits (two operands + carry in)
- Outputs sum and carry out

3.2.4 Flip-Flops

SR Flip-Flop:

- Set-Reset flip-flop
- Stores 1 bit of data
- Two inputs: S (set) and R (reset)

JK Flip-Flop:

- Improvement over SR
- All input combinations valid
- Uses clock pulse for synchronization
- Toggle action when J=K=1

Purpose:

- Store bits of data
- Create memory from flip-flops
- Used in registers and counters

3.3 VIRTUAL MACHINES

3.3.1 Concept of Virtual Machines

Definition: Software that provides an exact copy of hardware. The process interacts with the software interface provided by the OS, which provides this copy. The OS kernel handles interaction

with actual host hardware.

Benefits:

- Multiple OS on single system
- Testing without affecting main system
- Legacy software on newer hardware
- Server consolidation

Drawbacks:

- Performance drop from native OS
- Time and effort for implementation

Examples:

- VMware
- VirtualBox
- Hyper-V

CHAPTER 4: SYSTEM SOFTWARE

4.1 PURPOSES OF AN OPERATING SYSTEM

4.1.1 Resource Optimisation

How OS Maximises Resources:

- CPU scheduling
- Memory management
- I/O optimisation
- File system management

4.1.2 User Interface

Purpose: Hides complexities of hardware from user. Allows users to interact with application programs without knowing hardware details.

Types:

- Command-line interface (CLI)
- Graphical User Interface (GUI)
- Touch interface

4.1.3 Process Management

Process: A program being executed which has an associated Process Control Block (PCB) in memory.

Process States:

- **Ready:** New process arrived, PCB created
- **Running:** Has CPU access
- **Blocked:** Cannot progress until event occurs

PCB Contents:

- Process state
- Program counter
- CPU registers
- Memory management information
- I/O status information

4.1.4 Scheduling Routines

First-Come-First-Served (FCFS):

- Non-preemptive
- FIFO queue
- Simple but may cause long waits

Round Robin:

- Preemptive
- Allocates time slice to each process
- No prioritization
- Fair but may waste CPU on context switching

Priority-Based:

- Most complex
- Priorities re-evaluated on queue change
- Can be preemptive or non-preemptive

Shortest Job First:

- Non-preemptive
- Processes with shortest execution time first
- Optimal for minimizing average waiting time

Shortest Remaining Time:

- Preemptive version of Shortest Job First

4.1.5 Memory Management

Paging:

- Process split into pages
- Memory split into frames
- All pages can be loaded into memory

Virtual Memory:

- No need for all pages in memory
- CPU address space larger than physical
- Address resolved by Memory Management Unit (MMU)

Benefits:

- Large programs can run with less physical memory
- More programs can run simultaneously

Disk Thrashing:

- Perpetual loading/unloading of pages
- Occurs when too many processes compete for memory
- Performance severely degraded

4.1.6 OS Structure

User Mode:

- Available for users and applications
- Limited access to system resources

Kernel/Privileged Mode:

- Sole access to parts of memory
- Access to certain system functions
- Only OS components run in this mode

4.2 TRANSLATION SOFTWARE

4.2.1 Compilation Stages

Lexical Analysis:

- Converts sequence of characters to sequence of tokens
- Identifies keywords, identifiers, operators

Syntax Analysis:

- Checks code for grammar mistakes
- Identifies syntax errors

Code Generation:

- Generates intermediate code after syntax analysis

Optimization:

- Improves efficiency of code
- Removes redundant operations

4.2.2 Interpreters vs Compilers

Feature	Compiler	Interpreter
Translation	Entire program before execution	Line-by-line
Output	.exe file	No executable
Execution speed	Faster	Slower
Error reporting	All errors at end	Stops at first error
Debugging	Difficult	Easier

4.2.3 BNF (Backus-Naur Form)

Purpose: Formal mathematical way of defining syntax unambiguously.

Components:

- Set of terminal symbols
- Set of non-terminal symbols
- Set of production rules

Example:

<TEXT>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<integer> ::= <digit> | <integer> <digit>

4.2.4 Reverse Polish Notation (RPN)

Definition: A method of representing expressions without brackets or special punctuation. Uses postfix notation where operator is placed after variables.

Example:

- Infix: A + B
- RPN: A B +

Advantages:

- No need for brackets
- No need for operator precedence
- Simpler for machine to evaluate
- No backtracking needed

Evaluation:

- Use stack
- When operator encountered, pop two operands, apply operator, push result

CHAPTER 5: SECURITY

5.1 ENCRYPTION AND ENCRYPTION PROTOCOLS

5.1.1 Encryption Concepts

Plain Text: Original data before encryption.

Cipher Text: Result of applying encryption algorithm to data.

Encryption: Process of making cipher text from plain text.

Key: Value used by encryption/decryption algorithm.

5.1.2 Symmetric Key Encryption

Definition: Same key used for encryption and decryption.

Process:

1. Sender and receiver share secret key
2. Sender encrypts plain text with key → cipher text
3. Cipher text transmitted
4. Receiver decrypts with same key → plain text

Advantages:

- Fast
- Simple

Disadvantages:

- Key distribution problem
- Multiple keys needed for multiple recipients

Examples:

- AES (Advanced Encryption Standard)

- DES (Data Encryption Standard)

5.1.3 Asymmetric Key Encryption

Definition: Different keys for encryption and decryption.

Public Key:

- Shared with everyone
- Used for encryption
- Used for signature verification

Private Key:

- Kept secret
- Used for decryption
- Used for digital signatures

Sending Private Message:

1. Receiver sends public key to sender
2. Sender encrypts message with public key
3. Only receiver (with private key) can decrypt

Sending Verified Message:

1. Sender encrypts with private key
2. Anyone can decrypt with public key
3. Message verified as from sender

5.1.4 Digital Signatures

Process:

1. Calculate hash of message (digest)
2. Encrypt digest with sender's private key → digital signature
3. Send message + signature
4. Receiver decrypts signature with public key → digest
5. Receiver calculates hash of message
6. If digests match → message authentic

5.1.5 Digital Certificates

Purpose: Verify that public key belongs to claimed entity.

Certificate Contents:

- Entity's public key
- Entity's identity information
- CA's digital signature

Obtaining Certificate:

1. Entity contacts Certification Authority (CA)
2. CA confirms entity's identity
3. CA creates certificate with entity's public key
4. CA signs certificate with its private key
5. Entity posts certificate on website

5.1.6 SSL/TLS Protocol

Purpose: Provide secure communication between client and server.

Uses:

- Online shopping
- Online banking
- HTTPS websites

Process:

1. Client connects to server (port 443)
2. Server sends certificate
3. Client validates certificate
4. Client generates session key
5. Client encrypts session key with server's public key
6. Server decrypts session key
7. Secure session begins

5.2 MALWARE AND RESTRICTION METHODS

5.2.1 Types of Malware

Type	Description	Exploits
Virus	Replicates inside executable files	Executable files
Worm	Runs independently, propagates to networks	Shared networks
Spyware	Collects and transmits information	Background processes
Phishing	Emails requesting confidential info	User trust
Pharming	Bogus website redirects	Website appearance

5.2.2 Restriction Methods

Malware	Restriction Method
Virus	Anti-virus software with daily scans
Worm	Firewall protection
Spyware	Real-time anti-spyware
Phishing	Check sender email address
Pharming	Verify website URL

CHAPTER 6: ARTIFICIAL INTELLIGENCE

6.1 INTRODUCTION TO AI

6.1.1 Definition

Artificial Intelligence: The ability of computers to perform tasks that usually only humans would be able to do (decision-making, speech recognition, etc.).

Machine Learning (ML): A subfield of AI where computers learn to perform tasks without being explicitly programmed. Uses historical training data to produce a model for predictions.

Deep Learning (DL): A subset of ML where computers learn using neural networks similar to human brain function.

6.1.2 Types of Machine Learning

Supervised Learning:

- Trained on labelled data
- Known inputs and outputs
- Model learns from examples
- Used for classification and regression

Unsupervised Learning:

- Trained on unlabelled data
- Finds hidden patterns
- Groups similar data
- Used for clustering

Reinforcement Learning:

- Reward-based system

- Agent learns from actions
- Not given specific instructions
- Learns from trial and error

6.1.3 Classification, Regression, and Clustering

Classification:

- Split data into predefined groups
- Example: Spam/not spam email

Regression:

- Predict value based on explanatory variable
- Used for forecasting
- Examples: Weather, sales, prices

Clustering:

- Split data into groups based on features
- Groups not predefined
- Used for pattern discovery

6.2 ARTIFICIAL NEURAL NETWORKS

6.2.1 Structure

Layers:

1. **Input Layer:** Accepts inputs in various formats
2. **Hidden Layers:** Perform calculations, find patterns
3. **Output Layer:** Provides results

Nodes:

- Connected to nodes in adjacent layers
- Each connection has weight
- Data passes through network

6.2.2 Back Propagation

Definition: "Backward propagation of errors" - standard method for training neural networks.

Process:

1. Input data fed to network
2. Network generates output
3. Output compared to expected result
4. Error calculated
5. Error propagated backwards through network
6. Weights adjusted
7. Process repeated until acceptable error

6.2.3 Graph Theory in AI

Graph Components:

- **Nodes:** Fundamental units (represent objects)
- **Edges:** Lines connecting nodes (represent relationships)

Real-World Uses:

- Social media (users as nodes, friendships as edges)
- Google Maps (locations as nodes, roads as edges)

In AI:

- Represent ANN structure
- Find paths in problem solving
- Algorithms examine graphs

6.3 GRAPH SEARCH ALGORITHMS

6.3.1 Dijkstra's Algorithm

Purpose: Find shortest path between nodes in weighted graph.

How It Works:

1. Start at initial node
2. Calculate distances to all connected nodes
3. Select node with smallest distance
4. Update distances through that node
5. Repeat until destination reached

Limitations:

- No heuristics (searches all directions)
- Inefficient on large networks
- Cannot handle negative weights

6.3.2 A* Algorithm

Purpose: Informed search algorithm using heuristics to find shortest path efficiently.

Formula:

<TEXT>

```
F = G + H
```

- G: Movement cost from start
- H: Heuristic (estimated cost to goal)
- F: Total estimated cost

Advantages over Dijkstra:

- Uses heuristics to guide search
- More efficient
- Finds goal faster

CHAPTER 7: COMPUTATIONAL THINKING AND PROBLEM-SOLVING

7.1 RECURSION

7.1.1 Essential Features

Base Case:

- Stopping condition
- Prevents infinite recursion

Recursive Case:

- Calls itself
- Changes state toward base case

Unwinding:

- When base case reached
- Returns through call stack
- Calculates final result

7.1.2 Advantages and Disadvantages

Advantages	Disadvantages
Simpler, more natural solutions	Less efficient (time and space)
Elegant for tree-like problems	Stack overflow risk
Reduced code complexity	Hard to debug

7.1.3 How Compilers Handle Recursion

Stack Usage:

1. Before call: Save current state (registers, local variables) onto stack
2. Make recursive call
3. During return: Restore state from stack

Unwinding:

- When base case met
- Values popped from stack in reverse order
- Previous call states restored

7.1.4 Examples

Factorial:

```
<PYTHON>
def factorial(n):
 if n <= 1:
 return 1
 else:
 return n * factorial(n-1)
```

Fibonacci:

```
<PYTHON>
def fibonacci(n):
 if n <= 1:
 return n
 else:
 return fibonacci(n-1) + fibonacci(n-2)
```

7.2 ALGORITHM COMPLEXITY

7.2.1 Big O Notation

Notation	Name	Example
O(1)	Constant	Array index access
O(log n)	Logarithmic	Binary search
O(n)	Linear	Linear search
O(n log n)	Linearithmic	Merge sort
$O(n^2)$	Quadratic	Bubble sort

7.2.2 Comparing Algorithms

Time Complexity:

- How execution time grows with input size

Space Complexity:

- How memory usage grows with input size

CHAPTER 8: FURTHER PROGRAMMING

8.1 PROGRAMMING PARADIGMS

8.1.1 Low-Level Programming

Characteristics:

- Close to machine language
- Direct hardware access
- Instructions converted to machine code without compiler
- Non-portable (specific to processor)
- Small memory footprint
- Examples: Assembly, Machine code

Addressing Modes:

- Immediate: Data is value itself
- Direct: Address contains data
- Indirect: Address at address contains data
- Indexed: Address + index register contents

8.1.2 Imperative (Procedural) Programming

Characteristics:

- Sequence of statements changes state
- Explicit control flow
- Variables modified by assignments
- Based on Von Neumann architecture

Examples:

- C, Pascal, BASIC
- Python (also supports other paradigms)

8.1.3 Object-Oriented Programming (OOP)

Key Terms:

- **Object:** Instance of a class
- **Class:** Blueprint for creating objects
- **Property/Attribute:** Data in object
- **Method:** Function in object
- **Inheritance:** Child class inherits from parent
- **Polymorphism:** Same interface, different behaviour
- **Encapsulation:** Hiding internal details
- **Containment:** Object contains other objects

Inheritance Example:

<PYTHON>

```
class Person:
 def __init__(self, name, age):
 self.name = name
 self.age = age

 def get_name(self):
 return self.name

class Student(Person): # Inherits from Person
 def __init__(self, name, age, student_id):
 self.student_id = student_id
```

8.1.4 Declarative Programming

Characteristics:

- States what needs to be done, not how
- Non-procedural
- Control flow implicit
- Backtracking supported

Example:

- Prolog
- SQL

8.2 FILE PROCESSING

8.2.1 File Modes

Mode	Purpose
READ	Read from file
WRITE	Write to file (overwrites)
APPEND	Add to end of file
RANDOM	Direct access

8.2.2 Serial File Operations

Reading:

<PYTHON>

```
file = open("data.txt", "r")
while not EOF(file):
 line = READFILE(file)
 process(line)
CLOSEFILE(file)
```

Writing:

<PYTHON>

```
file = open("data.txt", "w")
WRITEFILE(file, "data")
CLOSEFILE(file)
```

8.2.3 Sequential File Operations

Search:

- Read records sequentially
- Compare key field

- Stop when found or key exceeds target

Add:

- Read from old file
- Write to new file
- Insert new record in correct position

Delete:

- Read from old file
- Write to new file
- Skip record to delete

8.2.4 Random File Operations

Hashing:

```
<TEXT>
Position = Hash(Key) MOD FileSize
```

Direct Access:

```
<PYTHON>
SEEK(file, position)
GETRECORD(file, record)
PUTRECORD(file, record)
```

8.3 EXCEPTION HANDLING

8.3.1 Exceptions

Definition: Runtime errors that cause program to terminate if not handled.

Common Exceptions:

- Division by zero
- File not found

- Invalid input
- Memory overflow

8.3.2 Handling Exceptions

Try-Except Structure:

<PYTHON>

```
try:
 # Code that might cause error
 file = open("data.txt", "r")
 data = file.read()
except FileNotFoundError:
 print("File not found")
except Exception as e:
 print(f"Error: {e}")
finally:
 # Always executes
 if 'file' in locals():
  file.close()
```

Purpose:

- Prevent program crash
- Provide meaningful error messages
- Allow graceful recovery