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

Revision #1
Created 2026-03-16 12:17:28 UTC by Samuel Lee
Updated 2026-03-16 12:17:44 UTC by Samuel Lee