OCR A Level Computer Science

Your notes

2.4 Types of Programming Language

Contents

Your notes

Programming Paradigms

- Programming paradigms are established **conventions** and **practices** that dictate how computer programs are **structured** and **developed**
- Programming paradigms offer varied methodologies for software construction
- Different paradigms are **suited to different tasks,** e.g. simple web applications can be achieved with light-weight procedural languages, complex enterprise-level software can only be achieved with a complex object-oriented paradigm
- New paradigms arise, and existing ones adapt in response to changes in computing and software challenges

Overview of Programming Paradigms

Paradigm	Description	Examples of Languages	Key Concepts
Procedural	A subtype of imperative, structured around procedure calls.	C, Go, Rust	Procedures, function calls, structured programming
Object-Oriented	Organises code around "objects" (which combine data and functionality) rather than functions.	Java, C#, Swift	Classes, inheritance, polymorphism, encapsulation
Assembly	Low-level mnemonic representation of machine code for a specific computer architecture.	x86-64 Assembly, ARMv8 Assembly	Registers, mnemonics, memory addresses, opcodes

Strengths & Weaknesses of Programming Paradigms

Paradigm	Strengths	Weaknesses
Procedural	▪ Efficient execution of straightforward tasks ▪ A clear flow of control (top to bottom)	▪ Can become unwieldy for large programs ▪ Lack of modularity can lead to code redundancy

	▪ Ease of implementation for algorithms ▪ Strong emphasis on step-by-step procedure execution	▪ Not ideal for applications with complex states or behaviours ▪ Difficulty in managing and scaling the system as it grows
Object-Oriented	▪ Enhances modularity with encapsulation ▪ Enables real-world modelling using objects ▪ Code reuse through inheritance ▪ Polymorphism allows flexibility in interface design	▪ Can lead to unnecessary complexity ▪ Inefficiency due to overhead (e.g., memory for objects) ▪ Not always intuitive for all types of problems ▪ Misuse can lead to overly complex inheritance hierarchies
Assembly	▪ Direct control over hardware ▪ Optimised performance due to low-level operations ▪ A transparent understanding of how the machine operates ▪ Potential for very efficient code	▪ Extremely steep learning curve ▪ Hardware-specific, leading to a lack of portability ▪ Tedious and error-prone due to manual memory management. ▪ Difficult to write, debug, and maintain large programs


Your notes

Procedural Programming


Your notes

Procedural Programming

- Procedural programming follows a step-by-step approach to breaking down tasks into routines and subroutines
- It emphasises modular design, where code is grouped into functions and procedures for reuse and clarity
- Variables hold state and control structures that determine the flow of execution in the program

Variables

- Storing data values that can change

```
x = 10
print(x) # Output: 10
```

Constants

- Storing values that remain unchanged

```
PI = 3.1415
print(PI) # Output: 3.1415
```

Selection

- Decision-making constructs

```
x = 7
if x > 5:
print("Greater") # Output: Greater
else:
print("Smaller")
```

Iteration

- Using loops to repeat actions

```
for i in range(3):
print(i) # Output: 0, 1, 2
```

Sequence

- Executing statements sequentially

```
x = 5
y = x + 10
print(y) # Output: 15
```

Subroutines

- Organising code into reusable parts

```
def greet(name):
return "Hello, " + name

greeting = greet("Alice")
print(greeting) # Output: Hello, Alice
```

String Handling

- Operations on character strings

```
name = "Alice"
upper_name = name.upper()
print(upper_name) # Output: ALICE
```

File Handling

- Reading from and writing to files

```
with open('file.txt', 'w') as file:
file.write("Hello, World!")

with open('file.txt', 'r') as file:
content = file.read()
print(content) # Output: Hello, World!
```

Boolean Operators

- Logical operations

```
x = 7
y = 5
is_valid = x > 5 and y < 10
print(is_valid) # Output: True
```

Arithmetic Operators

- Basic mathematical operations

```
x = 5
y = 3
```


Your notes

```
sum_value = x + y
product = x * y
print(sum_value, product) # Output: 8, 15
```


Your notes

Full example

- This script greets the user, asks for two numbers, and multiplies them if they are both greater than 10
- It gives the user three attempts to provide suitable numbers and asks if they want to continue after each attempt. Finally, it writes the greeting and the last multiplication result to a file

```
# Constants
MAX_ATTEMPTS = 3
FILENAME = 'output.txt'

# Subroutine to greet a user
def greet(name):
    return "Hello, " + name

# Subroutine to multiply two numbers
def multiply(x, y):
    return x * y

# Main program
def main():
    name = input("Please enter your name: ")
    print(greet(name))

    # Iteration to allow multiple attempts
    for attempt in range(MAX_ATTEMPTS):
        x = int(input("Enter the first number: "))
        y = int(input("Enter the second number: "))

        # Selection
        if x > 10 and y > 10:
            result = multiply(x, y)
            print(f"The product of {x} and {y} is {result}")
        else:
            print("Both numbers should be greater than 10.")

        # Asking user if they want to continue
        continue_choice = input("Do you want to continue? (y/n): ")
        if continue_choice.lower() != 'y':
            break

    # File Handling
    with open(FILENAME, 'w') as file:
        file.write(greet(name) + "\n")
        file.write(f"Last multiplication result: {result}")
```

```
    print(f"Results have been saved to {FILENAME}")

# Sequence: Calling the main program
if __name__ == "__main__":
    main()
```

Your notes

WORKED EXAMPLE

You are working for a library and need to develop a program that calculates the total late fees for overdue books. Provide pseudocode that includes a function to calculate the fee for each book.

How to answer this question:

- Notice that the same operation needs to take place against multiple items. This suggests iteration could be used
- A function is required to calculate the fee for each book. Simple names for functions make them clear to understand
- Think of some data structures to hold the bits of data for this scenario
    - Many numbers representing each book's days overdue could be stored in an array
    - The total late fee could be stored in a variable
- Use indentation to show which code is inside a code block e.g. function, if statement, for statement
- **Only** include code comments where you think it's necessary to explain
- The example below contains comments for your understanding

Answer:

Answer that gets full marks:

```
const DAILY_CHARGE = 1 // Many functions can use this const if they need it

function calculateFee(days_overdue)
 IF days_overdue > 0 THEN
  RETURN days_overdue * DAILY_CHARGE // £1 per day
 ELSE
  RETURN 0
 ENDIF
END function

function calculateTotalFee(books)
 var total_fee = 0
 FOR each days_overdue IN books // days_overdue is an identifier that represents each item in books
  total_fee = total_fee + calculateFee(days_overdue) // adding the result of the function to the total_fee variable
 ENDFOR
 RETURN total_fee // returning the variable back to caller
END function

var books = [7, 3, 0, 10] // Array of numbers representing each book's overdue days
var total_fee = calculateTotalFee(books)
```

```
PRINT "Total Late Fee:", total_fee
```

- This solution contains a function that accepts an array of books and a function that will calculate the fee for a single book
- This is well-designed because two smaller functions are better than 1 larger function

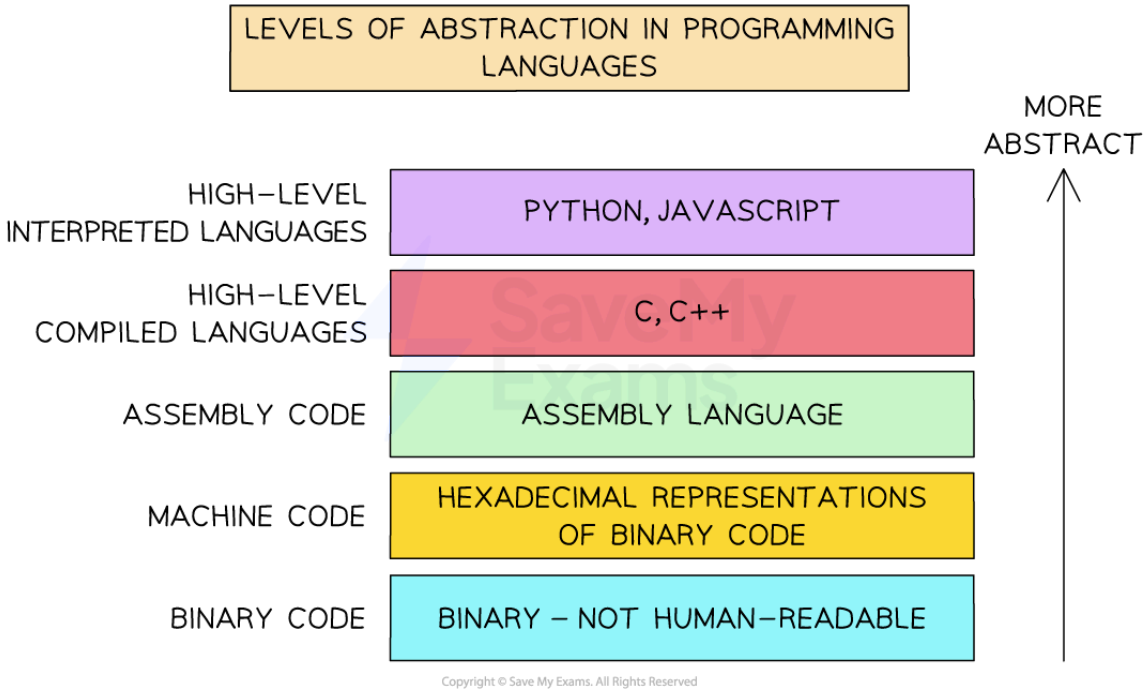

Your notes

Your notes

Assembly Language & Little Man Computer

What is the Purpose of Assembly Language?

- Assembly language sits **between** high-level languages (like Python, Java) and machine code (binary code executed by the computer hardware)
- Allows developers to write more **efficient**, albeit **more complex**, code when compared to high-level languages
- Direct access and manipulation of hardware components, e.g., registers, memory addresses
- Each type of computer CPU has its specific assembly language



Levels of Abstraction of Programming Languages

Little Man Computer

- The Little Man Computer (LMC) is a hypothetical computer model used for understanding the fundamental operations and mechanics of a computer

- The LMC is a simplified version of a computer
- It has key elements like **memory**, a **calculator**, an **accumulator**, and an **instruction** set

Your notes

Little Man Computer Instruction set

- The following mnemonics represent different actions that can take place in an LMC program

Mnemonic	Instruction	Alternative Mnemonic
ADD	Add	
SUB	Subtract	
STA	Store	STO
LDA	Load	LOAD
BRA	Branch always	BR
BRZ	Branch if zero	BZ
BRP	Branch if positive OR zero	BP
INP	Input	IN, INPUT
OUT	Output	
HLT	End program	COB, END
DAT	Data location	

Example 1: Add two numbers

```
INP;    // Input the first number
STA 90;  // Store the first number in memory location 90
INP;    // Input the second number
ADD 90;  // Add the number in memory location 90 to the accumulator
OUT;    // Output the result
HLT;    // End the program
DAT;    // Memory location 90 for storing data
```

Example 2: Find the smallest of three numbers

This program inputs three numbers and determines the smallest of the three, outputting the result.

```
INP                    // Input the first number
STA 91                 // Store the first number in memory location 91
INP                    // Input the second number
STA 92                 // Store the second number in memory location 92
INP                    // Input the third number
STA 93                 // Store the third number in memory location 93

LDA 91                 // Load the first number
SUB 92                 // Subtract the second number
BRP CHECK_THIRD_FROM_FIRST     // If result is positive, then first number > second number

LDA 92                 // Load the second number
SUB 93                 // Subtract the third number
BRP OUTPUT_SECOND          // If result is positive, then second number > third number
LDA 93
OUT                    // Output the third number
HLT

CHECK_THIRD_FROM_FIRST:
  LDA 91
  SUB 93
  BRP OUTPUT_FIRST
  LDA 93
  OUT
  HLT

OUTPUT_FIRST:
  LDA 91
  OUT
  HLT

OUTPUT_SECOND:
  LDA 92
  OUT
  HLT

DAT                    // Memory locations for data storage
DAT
DAT
```

Your notes

WORKED EXAMPLE

A digital thermostat has a CPU that uses the Little Man Computer Instruction Set.

The thermostat allows users to set a desired room temperature. The acceptable range for room temperature settings is between 15 and 25 degrees Celsius, inclusive. If the user sets a

temperature within the range, the code outputs a 1 indicating a valid setting. If the temperature setting is outside of the acceptable range, the code outputs a 0 indicating an invalid setting.

The code is as follows:

```
<>INP
STA tempSetting
LDA minTemp
SUB tempSetting
BRP checkMax
LDA invalid
BRA end

checkMax
    LDA maxTemp
    SUB tempSetting
    BRZ valid
    BRP invalid
    valid LDA valid
    end OUT
HLT

valid DAT 1
invalid DAT 0
minTemp DAT 15
maxTemp DAT 25
tempSetting DAT
```

a) What is the purpose of the label checkMax in the code? Describe its role in determining the validity of the temperature setting.

[2]

b) If a user inputs a temperature setting of 14, what will be the output? Justify your answer.

[2]

c) If the acceptable range of temperature settings was expanded to include temperatures up to 30 degrees Celsius, what modification would be needed in the code?

[2]

Answer:

Example answer that gets full marks:

a) The label checkMax begins a code segment that checks if the user's desired temperature is within the maximum allowable temperature. It subtracts the user's input from the maximum temperature (maxTemp). If the result is zero (meaning they are equal) or positive (meaning the user's input is less than maxTemp), the user's desired temperature is within the allowable range.

Your notes

b) If a user inputs a temperature setting of 14, the output will be 0 (indicating an invalid setting). This is because when 14 is subtracted from the minimum allowed temperature (minTemp), the result is positive, which then causes the code to skip checking the maximum value and directly output the invalid value, which is 0.

c) To increase the maximum allowable temperature setting to 30 degrees Celsius, modify the maxTemp DAT value. The new line should read:

maxTemp DAT 30

Acceptable answers you could have given instead:

a) Any response mentioning that checkMax it is for checking if the user's input is below or equal to the maximum allowable temperature should be awarded marks.

b) Any answer stating that the output will be 0 because 14 is less than 15, or similar logic, should be awarded marks.

c) Any answer suggesting a change to the maxTemp DAT value to 30 should be awarded marks.

Your notes

Modes of Addressing

Your notes

Modes of Addressing

- **Immediate Addressing**
  - Operand is part of the instruction itself

MOV AX, 1234h // Moves the immediate hex value 1234h to the AX register

- **Direct Addressing**
  - The memory address of the operand is directly specified

MOV AX, [1234h] // Take the value stored in memory location 1234h and move to the AX register

- **Indirect Addressing**
  - A register contains the memory address of the operand
  - If BX contains the value 2000h:

MOV AX, [BX] // Moves the value from memory location 2000h to the AX register

- This **does not** mean "Move the value 2000h into AX"
- Instead, it means, "Look in the memory address 2000h (the value currently stored in the BX register) and move whatever value you find into the AX register."
- When brackets [ ] are around a register in assembly language (like [BX]), it's an instruction to treat the value inside that register as a memory address and to use the data at that memory address for the operation
- **Indexed Addressing**
  - Combines a base address with an index to compute the effective address
  - If BX contains 0050h and SI has a base address 1000h:

MOV AX, [BX + SI] // Move the value at memory location 1050h to AX

- Fetches data from the effective address (because 1000h + 0050h is 1050h) and moves it into the AX register

WORKED EXAMPLE

Consider a basic computer system with the following assembly language instructions and a memory layout starting at address 1000.

1000: MOV AX, 8
1002: ADD AX, [BX]
1004: MOV [0008], AX
1006: MOV CX, [BX+DI]
1008: HLT

Assume the registers have the following values before execution:

AX = 0000
BX = 0003
DI = 0002
CX = 0010

Memory contains:

0000: 0
0001: 0
0002: 0
0003: 5
0004: 0
0005: 7
0006: 7
0007: 9
0008: 0

a) For the instruction at 1002, identify the addressing mode used and explain what it does in the context of this instruction.

[2]

b) After the instruction at 1004 has executed, what will the value at memory address 0008 be? Justify your answer.

[2]

c) What value will be moved into the CX register after the instruction at 1006 executes? Explain the addressing mode used.

[2]

Answer:

Answer that gets full marks:

a) The instruction at 1002 uses **Indirect Addressing**. The instruction ADD AX, [BX] adds the value at the memory address contained in the BX register to the AX register. In this case, since BX is 3, it will add the value at memory address 3 to AX.

b) The value at memory address 0008 will be 13. Before the instruction, AX contains the value 8. After adding 5 (from memory address 3 due to the instruction at 1002), AX will have the value 13. The

Your notes

instruction at 1004 then moves this value to memory address 0008.

c) The value moved into the CX register will be 7. The instruction at 1006 uses **Indexed Addressing**. It accesses memory by combining the address in BX with the offset in DI. Given BX is 3 and DI is 2, the effective address is 3 + 2 = 5, so it fetches the value 7 from address 0005 into CX.

Acceptable responses:

a) Any answer identifying Indirect Addressing and explaining its use in the context of fetching a value from memory for the instruction should be awarded marks.

b) Any answer stating that the value at address 0008 will be 13 due to adding 8 and 5 should be awarded marks.

c) Any response indicating the use of Indexed Addressing and explaining the value fetch from address 5 should be awarded marks.


Your notes