Component 1 | 1.2.4| Types of programming language

Specification & learning objectives

	A Level	Specification point description
	1.2.4a	The need for and characteristics of a variety of programming paradigms
	1.2.4b	Procedural languages
	1.2.4c	Assembly language (including following and writing simple programs with the Little Man Computer Instruction set)
	1.2.4d	Modes of addressing memory (immediate, direct, indirect and indexed)
	1.2.4e	Object-oriented languages with an understanding of classes, objects, methods, attributes, inheritance, encapsulation and polymorphism

Resources

PG Online textbook page ref: 64-73
Hodder textbook page ref: 84-96
CraignDave videos for SLR 7

Component 1 | 1.2.4| Types of programming language

Key question: What do we mean by the term programming paradigm?

There are many different programming languages available. It is useful to be able to group programming languages, so that they can be discussed and compared.
A common way of grouping programming languages is by their general approach to solving problems. These approaches, or 'paradigms', are often specialised.

- Low level languages
- Procedural languages
- Object-orientated languages
- Functional languages
- Declarative languages

Key question: What are the features of procedural languages?

One of the most common programming paradigms is the idea of 'procedural' languages. Procedural languages group sets of instructions together into subroutines or functions. Sequence, selection, and iteration are key components of writing code in procedural languages.
This is a very powerful and versatile approach to programming. Examples are:

Procedural languages are **imperative**. Programs are given a list of explicit instructions, called an algorithm, telling them exactly *how* to carry out a task.

Procedural languages are **sequential**. This means that the order of instructions is important.

Examples of Procedural languages:
C, Pascal, FORTRAN, COBOL

```
y = 1224;
for (x = 0; x < 100; x++) {
   z = y + x;
}
```

Variable y is being set
Enters a loop
So the procedural language is precisely defining what the computer should be doing step by step.

Key question: What are the features of procedural languages?

Pros

- Excellent for general purpose programming
- Many books and references available on well-tried and tested coding algorithms - no need to re-invent the wheel.
- Good level of control without having to know precise target CPU details - unlike low level languages
- Portable source code - use a different compiler to target a different CPU

Cons

- As there are so many procedural languages, a programmer tends to have to specialise in a particular language in order to get work.
- For example : A COBOL specialist has a different clientele to a 'C' specialist.
- Need to be very precise and knowledgeable about programming instructions, and so a fully de-bugged working program takes more time to put together compared to fourth generation languages such as Simulink.
- Not as efficient as hand-crafted source code written in a low level language
- Poor at handling fuzzy conditions as found in Artificial Intelligence applications - unlike declarative languages such as PROLOG.

Key question: What are the features of assembly language?

- Low level languages are tied to specific CPU families. Each CPU family requires its own low level language. Low level languages are almost (but not quite) machine code.
- 'Assembly language' is an example of a low level programming language.
- CPU chip makers create assembly languages themselves. They teach programmers how to best use their assembly language when writing code for their family of CPUs.
- Some features of Low Level languages include:

•'Mnemonics' are used as programming code such as MOV or ADD.

•Many different memory modes can be used.

•Labels are used as reference points to allow the code to jump from one part to another.

•They are CPU specific, making direct use of internal registers.

Annotated example of an assembly program.

```
.MODEL SMALL;
.STACK;
.CODE;
mov ah,1h; moves the value 1h to register ah
mov cx,07h;moves the value 07h to register cx
int 10h;
```

Pros.

- Low level languages are excellent for close control of the CPU, for example many device drivers are coded in assembly language.
- They can be very efficient. Well-optimised code written in a low level language can be made to run very quickly compared to other programming paradigms.

Cons

- They are difficult to use as the programming commands can be quite obscure
- A good assembly language programmer needs to know a lot of detail about the internal structure of the CPU - e.g.its registers and memory management methods
- Low level languages produce the least portable source code.

Component 1 | 1.2.4| Types of programming language

Key question: What are immediate, direct, indirect, indexed and relative memory addressing?

Intermediate Addressing

- Sometimes called intermediate operand
- The operand of the instruction is the actual value to be used e.g. ADD 20 adds the value 20 not address 20
- It removes the I/O operation of loading from memory, so it is a very efficient mode of addressing.
- Can only use constant values

Indirect Addressing

- The operand is the address in memory that contains the real address in memory.
- E.g. Add 20 would go to address 20 and find that it might have number 4000 in it which is the real address of the data.
- Useful as it allows larger addresses than normal to be used
- However, indirect Addressing uses many CPU cycles making it inefficient in some use cases.

Relative Addressing

- Relative addressing means that the next instruction to be carried out is an offset number of locations away, relative to the address of the current instruction.

Direct Addressing

- The address in memory is the operand of the instruction
- E.g. ADD 20 means add the contents of location 20 to the accumulator
- Advantage is that it requires few CPU cycles compared to the later modes
- Disadvantage is that It's inefficient to address many pieces of data to perform the same operation .

Indexed Addressing

- A recursive method where 1 instruction is run many times on multiple data
- This uses an additional CPU register called the index register.
- The index register is added to the first item in the array's address to find the current address
- Bad for small data amounts (lots of CPU cycles!)
- Efficient for arrays and similar.

Key question: What are the features of object orientated languages?

Object Oriented Programming (OOP) languages use templates, called classes, to define items that they are working with. It is easier to think about these using examples.
In video games, you often play a character with a set of attributes, like this:

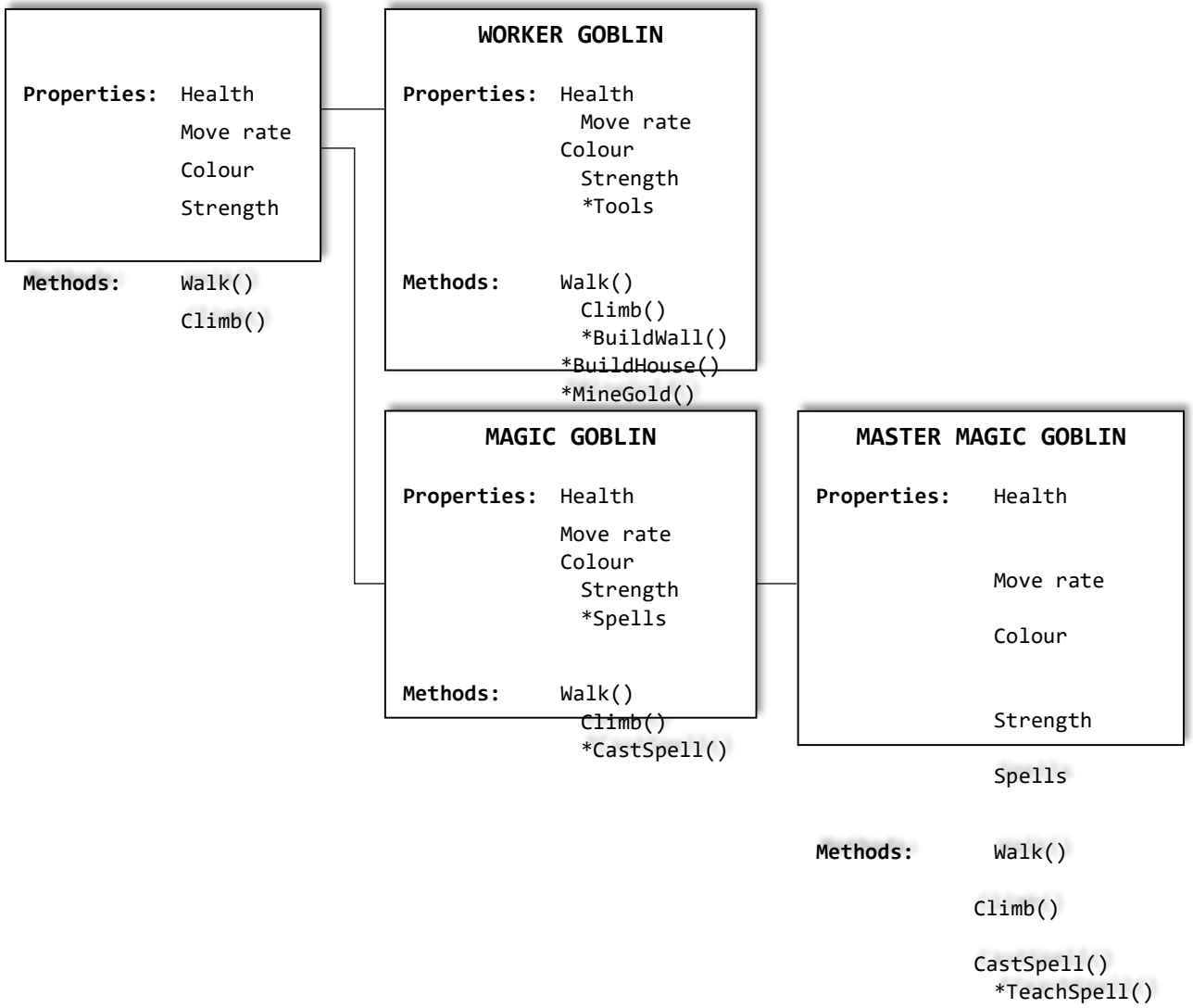
ADVENTURER	
NAME:	JAMES
HEALTH:	20
MANA:	20
STRENGTH:	5
SPEED:	7
INTELLIGENCE:	10
SKILLS:	BLOCK
	HEAL
	FIREBALL

- Classes are templates.
- Classes have a set of attributes.
- Objects within a class share the same set of attributes.
- Objects within a class can have different values for their attributes.

The "adventurer" template, or **class**, has a number of **attributes**. These are qualities that define objects within the class.

In this case, name, health, mana, etc. Each of those attributes has a value. 20 health, or 7 speed, for example.

Annotated example of an object-oriented program.



Typical exam questions

The following assembly code is written for the Little Man Computing instruction set.

```
                INP
                STA     arg1
                INP
                ADD     arg1
                OUT
                INP
                SUB     arg1
                OUT
                HLT
arg1            DAT
```

1. State the output of this program when the inputs are 10, followed by 5 followed by 7. **[1]**

2. Explain what this program is doing. **[4]**

3. Explain, with the aid of a diagram what is meant by the term inheritance. **[4]**

Component 1 | 1.2.4| Types of programming language

Target: Overall grade:

Minimum expectations & learning outcomes

☐	Terms 81-106 from your A Level Key Terminology should be included and formatted.
☐	You must describe the paradigms procedural, assembly and object-oriented.
☐	You must include an annotated example of a procedural program.
☐	You must include an annotated example of an assembly language program written using the LMC Instruction Set.
☐	You must include a comparison of immediate, direct, indirect and indexed addressing and how this relates to RISC & CISC architectures.
☐	You must include some illustrations that explain the main concepts of object orientated programming including class, object, inheritance, public attributes, private attributes, methods, encapsulation and polymorphism.
☐	Answer the exam questions.

Feedback

Breadth	Depth	Presentation	Understanding
☐ All	☐ Analysed	☐ Excellent	☐ Excellent
☐ Most	☐ Explained	☐ Good	☐ Good
☐ Some	☐ Described	☐ Fair	☐ Fair
☐ Few	☐ Identified	☐ Poor	☐ Poor

Comment & action required

Component 1 | 1.2.4| Types of programming language

Reflection & Revision checklist

Confidence	Clarification
☹ 😐 ☺	Candidates need to understand that there are a variety of types of programming paradigms such as procedural, OOP, low-level, and that each has its strengths and weaknesses in specific scenarios, topics or areas.
☹ 😐 ☺	Candidates need to have knowledge and experience of using a procedural programming language for example Python, VB.NET etc.
☹ 😐 ☺	Candidates need to be experienced in using procedural programming features such as (but not limited to) variables, constants, selection, iteration, sequence, subroutines, string handling, file handling, Boolean and arithmetic operators.
☹ 😐 ☺	Candidates need to be able to read, trace, amend and write procedural program code.
☹ 😐 ☺	Candidates need to have an understanding of the purpose and need for assembly language. They need to be familiar with the instructions given in Appendix 5d. They should be able to read, write, trace and amend programs written in the Little Man Computer language.
☹ 😐 ☺	Candidates need an understanding of addressing, which should be integrated with assembly language.
☹ 😐 ☺	Candidates should have experience of using immediate, direct, indirect and indexed addressing in the writing, reading and tracing of programs written in assembly language.
☹ 😐 ☺	Candidates need to understand object-oriented code (as specified in the pseudocode guide). They need to have an understanding of classes, objects, attributes and methods. They need to understand the difference between private and public attributes and methods.
☹ 😐 ☺	Candidates need to understand encapsulation and the use of get and set methods to access private attributes.
☹ 😐 ☺	Candidates need to understand the purpose and principles of inheritance.
☹ 😐 ☺	Candidates need to have an understanding of polymorphism and how it can be used within a program.
☹ 😐 ☺	Candidates need to be able to read, trace, amend and write code that makes use of these object-oriented