GCST

1.2.4
TYPES OF PROGRAMMING LANGUAGE
TOPIC WISE EXAM QUESTIONS
ANSWERS

A-LEVEL OCR

4	(a)	(i)	• Assembly language uses mnemonics • HLL uses English-like words • Assembly language uses an assembler to convert to machine code • HLL uses a translator (compiler/interpreter) to convert to machine code • Assembly language is a one-to-one conversion to machine code • HLL may produce multiple lines of machine code per line of code // one-to-many • Assembly language requires more knowledge of the processor // allows direct control of the processor • HLL provides more abstraction // requires less knowledge of the processor • Assembly language is likely to be specific to the processor type used // is machine dependent • HLL is portable // can be used for multiple processor types // programmer can pick from a number of HLLs and paradigms // is machine independent	4	Mark in pairs. Allow examples (e.g. JMP, print) for MP1 and 2
5	(a)		• 9	1	

5	(b)		• **input and store/use** a value from user
			• call **doCheck** function with value input from user **and save/use returned value**
			• open and close text file in write/append mode, if given
			• write value returned to text file
			• Write value input to text file
				5	Example code:
					`num = input("enter a number")`
					`value=doCheck(num)`
					`txtfile = openWrite("storedvalues.txt")`
					`txtfile.writeLine(num)`
					`txtfile.writeLine(value)`
					`txtfile.close()`
					
					MP2 - **doCheck** is case sensitive
					
					MP3 - need speech marks around file name

(c)	(i)	• 10 • 60 • 200	3	1 mark per number

	(ii)	• Loads a value into the accumulator
		• Establishes a zero value (by use of DAT / SUB)
		• Stores a zero value into total
		• Program stops
			4	Example 1
				`LDA zero`
				`STA total`
				`HLT`
				`zero DAT 0`
				
				Example 2
				`LDA total`
				`SUB total`
				`STA total`
				`HLT`
				
				BP1 can be given for any value being loaded into the accumulator e.g. INP
				
				If candidate writes LDA donation/total (case sensitive) they can get BP2 as they've used the labels from the question
				
				BP3 - total is case sensitive as given in the question
				
				BP4 - must not be given if the zero value will be attempted to be fetched e.g. HLT is placed after DAT

(g)	(i)	• Class definition with identifier video
		• name, number of views and star rating attributes defined…
		• …As private
		• Constructor method definition inside class definition…
		• …that accepts only one parameter
		• …Name attribute set to parameter passed in
		• Views set to 0 and rating set to 3 **either** when initialised **or** in constructor.
			7	Accept implementations in high-level languages (e.g. __ for private, class name used for constructor, no need for end of class definition in Python)
				
				BP1 - allow empty brackets. Do not allow anything in the brackets
				BP5 - ignore self if included as parameter
				
				`class video`
				`  private name`
				`  private views`
				`  private starrating`
				
				`  public procedure new(newname)`
				`  name = NewName`
				`  views = 0`
				`  starrating = 3`
				`  end procedure`
				`end class`

	(ii)	• Method definition that is public
		• View attribute incremented by one
			2	`public procedure updateviews()`
				`  views = views + 1`
				`end procedure`
				
				View attribute must have the same name as part i

5	(a)	(i)	• 40	1	CAO
		(ii)	• 70	1	CAO
		(iii)	• 300	1	CAO

(b)		• 2, 2 • 9,0 • 4,0 • 0,3	4 AO2.2	Code finds integer division and remainder.
(c)		• Input two numbers into two separate variables / other suitable data structure • Correctly calculate integer division • Correctly calculate remainder • Print out both	4 AO3.2	Can be completed either by using MOD / DIV or by using repeated subtraction as in LMC example
(d)		**Mark Band 3–High Level (9-12 marks)** The candidate demonstrates a thorough knowledge and understanding of different modes of addressing. The material is generally accurate and detailed. The candidate is able to apply their knowledge and understanding directly and consistently to the context provided. Evidence/examples will be explicitly relevant to the explanation. The candidate provides a thorough discussion which is well balanced. Evaluative comments are consistently relevant and well-considered. There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated. **Mark Band 2-Mid Level (5-8 marks)** The candidate demonstrates reasonable knowledge and understanding of different modes of addressing; the material is generally accurate but at times underdeveloped. The candidate is able to apply their knowledge and understanding directly to the context provided although one or two opportunities are missed. Evidence/examples are for the most part implicitly relevant to the explanation. The candidate provides a sound discussion, the majority of which is focused. Evaluative comments are for the most part appropriate, although one or two opportunities for development are missed. There is a line of reasoning presented with some	12 AO1.1 (2), AO1.2 (2), AO2.1 (3), AO3.3. (5)	AO1 Immediate addressing is where the operand holds the actual data to be used; no address referenced Direct addressing is where the operand holds the address that holds the data to be used. Indirect addressing is where the operand holds an address which is where the data to be used is stored Indexed addressing is where the operand holds an address which is offset using the Index Register to find the true address of the data to be used AO2 Immediate addressing; operand of 27 would refer to the value 27 Direct addressing; operand of 27 would tell the processor to fetch the data held at address 27. Indirect addressing; operand of 27 would tell the processor to fetch the data held at address 27, which itself would then be used an address to fetch data from. Indexed addressing; operand of 27 would be added to the value of the Index register and this would then be used as the address to fetch data from. AO3 Immediate addressing allows simple access to data with no fetch required, but limited by the data size of the operand. Direct addressing allows data to be fetched from memory. Data can potentially be larger in size that with immediate addressing but address range limited by size of operand. Indirect addressing allows a larger range of addresses to be accessed as address fetched. However, multiple fetches required to access data. Indexed addressing allows the Index register to be manipulated to access data stored sequentially e.g. in an array.
(c)	(i)	• Object – instantiated from class • Method – action object performs / link to procedure/functions • Attribute – value held by object / link to variable	3 AO1.1	

			• Appropriate get methods for name **and** attendance that return a value and have no parameter
			• Appropriate set methods for name **and** attendance that take a parameter
			• …that restricts attendance to be 0 to 100.
				AO3.2	Example answer
					`class Customer`
					`  private name`
					`  private attendance`
					
					`  public function getName()`
					`  return name`
					`  end function`
					
					`  public function getAttendance()`
					`  return attendance`
					`  end function`
					
					`  public procedure setName(newName)`
					`  name = newName`
					`  end procedure`
					
					`  public procedure setAttendance(newAttend)`
					`  if newAttend >=0 and newAttend <=100 then`
					`  attendance = newAttend`
					`  end if`
					`  end procedure`
					`end class`

2020

2	(a)	(i)	• Constructor method definition (.e.g `new`)
			• itemname, price passed in as parameters (must use different identifier names to the ones in the question)
			• …and assigned to attributes
			• `discount` attribute assigned to 0.
				4
				AO3.2
					Example answer
					`public procedure new(nItemName, nPrice)`
					`  itemname = nItemname`
					`  price = nPrice`
					`  discount = 0`
					`endprocedure`
					
					Look out for alternative notation
					e.g.
					
					this.itemname = itemName

		(ii)	• 1 mark for creating object with identifier `mushypeas =`
			• 1 mark for creating object as type `ItemForSale`
			• mark for parameters passed in as needed
				3
				AO3.2
					Example answers
					`mushypeas=new ItemForSale("mushy peas", 0.89)`
					
					`ItemForSale mushypeas = ItemForSale("mushy peas",0.89);`
					
					`mushypeas=ItemForSale(("mushy peas",0.89);`
					Do not penalise for use of self parameter as used by languages such as Python.
					Must be correct case and spelling

		(iii)	• method definition for `calculatePrice()`
			• applies percentage discount
			• …returns calculated value
				3
				AO3.2
					Percentage discount must be applied correctly.
					
					Example answer
					`function calculatePrice()`
					`  newPrice = price - (price * discount/100)`
					`  return newPrice`
					`end function`

2	(b)		• `discount` attribute made private • Set method created • …that restricts value to maximum 50%	3 AO2.2	
2	(c)		• Create new class • …inheriting from / subclass of **`ItemsForSale`** • new attribute for stock location • `calculatePrice()` method overridden // new version of `calculatePrice()` implemented in subclass.	4 AO2.2	

6 (a)

- Stores current player
- …and alternates between player on each go
- Allows a number to be input
- …and validates that this is between 1 and 3
- Outputs numbers chosen (e.g. 4, 5, 6)
- Checks if number 15 has been reached
- …and displays winning message
- …and stops

8

AO3.2

Example answer

```
num = 1
turn = "player 1"
while num <= 15
    print(turn + "'s turn")
    choice = input("how many numbers?")
    if choice >= 1 and choice <= 3 then
        for y = 1 to choice
            print(num)
            num = num + 1
        next y
        //swap turn
        if turn == "player 1" then
            turn = "player 2"
        else
            turn = "player 1"
        end if
    end if
endwhile
print(turn + " wins!")
```

7 (a)

- Store value in accumulator at address given
- BRA // BR
- Branch if zero
- Branch if zero or positive
- HLT // COB // END

5

AO1.1

Mnemonic	Instruction
ADD	Add
SUB	Subtract
STA	Store value in accumulator at address given
LDA	Load (to accumulator)
BRA	Branch always
BRZ	Branch if zero
BRP	Branch if zero or positive
INP	Input
OUT	Output
HLT	End program

7 (b)

- Inputs two numbers
- ..stores at least one of them
- Comparison / subtraction to decide which is larger
- Jump / output if num1 larger
- Jump / output if num2 larger or nums equal
- Loops back to start after either output

6

AO3.2

Example answer

```
start INP
      STA x
      INP
      STA y
      SUB x
      BRP  first
      LDA x
      OUT
      BRA start
first  LDA y
      OUT
      BRA start
x     DAT
y     DAT
```

8	(a)		**Mark Band 3–High Level (9-12 marks)** The candidate demonstrates a thorough knowledge and understanding of procedural and object oriented programming. The material is generally accurate and detailed. The candidate is able to apply their knowledge and understanding directly and consistently to the context provided. Evidence/examples will be explicitly relevant to the explanation. The candidate provides a thorough discussion which is well balanced. Evaluative comments are consistently relevant and well-considered. There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated. **Mark Band 2-Mid Level (5-8 marks)** The candidate demonstrates reasonable knowledge and understanding of procedural and object oriented programming; the material is generally accurate but at times underdeveloped. The candidate is able to apply their knowledge and understanding directly to the context provided although one or two opportunities are missed. Evidence/examples are for the most part implicitly relevant to the explanation. The candidate provides a sound discussion, the majority of which is focused. Evaluative comments are for the most part appropriate, although one or two opportunities for development are missed. There is a line of reasoning presented with some structure. The information presented is in the most part relevant and supported by some evidence. **Mark Band 1-Low Level (1-4 marks)** The candidate demonstrates a basic knowledge of procedural or object oriented programming; the material is basic and contains some inaccuracies. The candidate makes a limited attempt to apply acquired knowledge and understanding to the context provided.	12 AO1.1 (2) AO1.2 (2) AO2.1 (23 AO3.3 (5)	AO1 Object oriented programming makes use of classes (templates) from which objects are made. Classes have attributes and methods Classes can be encapsulated by making attributes private and providing public access methods Object oriented programming supports inheritance which allows classes to use attributes and methods of parent classes. Object oriented programming supports polymorphism meaning that class attributes and methods can take on many different forms if required. AO2 OO approach would call on classes per type of player or enemy object Objects made from these classes, so one enemy class may generate many enemy objects, each with different values for their attributes (e.g. speed, energy) Special types of Player or Enemy objects could be instantiated from classes that inherit from the original Player/Enemy classes, but have attributes/methods of their own. Polymorphism would allow for the attributes/methods of Player/Enemy objects to behave differently from normal if required. AO3 OO promotes a modular approach (procedural through use of subroutines, OO through classes). OO is an abstraction of a real world problem, with classes for each type of things to be modelled and objects for each instance of these. OO has advantages in data security in that encapsulation forces developers/users to use methods (with their built in validation) to access/amend data stored in attributes. OO has advantages in efficiency of design where classes can be reused and can inherit from one another. Procedural programming struggles to support this. OO also offers flexibility through polymorphism.

2019

3	a		**Input** | **Output** 1 | 1 2 | 4 3 | 9 1 per row, max 3	3 AO3.3	
	b		Squares a number / multiplies a number by itself	1 AO3.3	
	e		– Immediate addressing… – …operand is the value to be used. – Indirect Addressing… – …operand is the memory location holding a value representing the memory location to be used. – Indexed Addressing… – …Operand is added to contents of Index Register to get memory location of value needed. (1 mark for naming addressing mode, 1 mark for correct description)	2 AO1.1	

<table>
<tr><td>b</td><td>i</td><td>
<table>
<tr><th>Input</th><th>Green Light</th><th>Red Light</th></tr>
<tr><td>1</td><td>✓</td><td></td></tr>
<tr><td>2</td><td>✓</td><td></td></tr>
<tr><td>3</td><td>✓</td><td></td></tr>
<tr><td>4</td><td>✓</td><td></td></tr>
<tr><td>5</td><td>✓</td><td></td></tr>
<tr><td>6</td><td></td><td>✓</td></tr>
<tr><td>7</td><td></td><td>✓</td></tr>
<tr><td>8</td><td></td><td>✓</td></tr>
<tr><td>9</td><td></td><td>✓</td></tr>
</table>
Rows 1-4 correct 1 Mark
Rows 5-9 correct 1 Mark
</td><td>2
(AO3.3)</td><td>Accept T for a tick. Penalise if blank table elements have content.</td></tr>
<tr><td></td><td>iii</td><td>- Takes in a value from user.
- If value is 5 or less it shows green
- Otherwise it shows Red

(1 Mark per -, max 3)</td><td>3
(AO 3.2)</td><td>Do not credit structured English
Example

<pre>value = input("Enter a Value")
if value <=5 then
   print("GREEN")
else
   print("RED")
endif</pre>
Accept equivalents to <=5 (e.g. <6)
For Green/Red (or 1/0) accept any pseudocode equivalent (GreenLightOn(), Output 1, print(1) Output Green etc.) as long as the logic is correct.</td></tr>
<tr><td></td><td>iv</td><td>Mark Band 3–High Level (7-9 marks)
The candidate demonstrates a thorough knowledge and understanding of assembly code and high level languages. The material is generally accurate and detailed.

The candidate is able to apply their knowledge and understanding directly and consistently to the context provided. Evidence/examples will be explicitly relevant to the explanation.

The candidate provides a thorough discussion which is well balanced. Evaluative comments are consistently relevant and well-considered.

There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated.
Mark Band 2-Mid Level (4-6 marks)
The candidate demonstrates reasonable knowledge and understanding assembly code and high level languages; the material is generally accurate but at times underdeveloped.

The candidate is able to apply their knowledge and understanding directly to the context provided although one or two opportunities are missed.
Evidence/examples are for the most part implicitly relevant to the explanation.

The candidate provides a sound discussion, the majority of which is focused. Evaluative comments are for the most part appropriate, although one or two opportunities for development are missed.

There is a line of reasoning presented with some structure. The information presented is in the most part relevant and supported by some evidence.

Mark Band 1-Low Level (1-3 marks)
The candidate demonstrates a basic knowledge assembly code and high level languages; the material is basic and contains some inaccuracies. The</td><td>9

AO1.1
(2)
AO1.2
(2)
AO2.1
(2)
AO3.3
(3)</td><td>AO1
Assembly code uses mnemonics to represent machine code instructions/opcodes.
High level languages use more natural/mathematical notation.
Assembly code consists of simple instructions
As such many more lines of assembly code are required to perform the same task as a few lines of a high level language.
Assembly code is specific to the instruction set of a given processor. High Level languages are not architecture specific.

AO2
Assembly code allows the programmer to choose the exact instructions so they can write code that is highly efficient.
It also allows them to have direct control of how memory is used via addressing modes.
Direct control of hardware.
High level language compilers have optimisers that can also try and do this (and in some cases may outperform a human writing in assembly code).

As high level code is more intuitive and easier to read it is easier to follow, debug and build as part of a team. It can also be written in a much shorter time frame.
The high level code can be recompiled for different architectures.
High level languages come in a variety of paradigms so programmers can choose according to the problem/their preference.

AO3
<u>Assembly language is best suited to situations such as:</u>
-compilers or interpreters don't exist for the target CPU i.e. embedded systems
-highest possible performance is critical
-memory is very limited.
For larger projects which don't fall under the constraints above high level languages are likely to be preferable.</td></tr>
</table>

b	i	- Created Obstacle object called bollard
		- Has put the correct arguments in, in the correct order.
		(1 Mark per -, Max 2)
			2 (AO3.2)	Examples
				`bollard=new Obstacle(false, 7.8, 8)`
				`Obstacle bollard = Obstacle(false, 7.8, 8);`
				`bollard=Obstacle(False, 7.8, 8)`
				
				Do not penalise for use of `self` parameter as used by languages such as Python.

	ii	-The attribute `distance` is private… -…and therefore updated with the method `update distance`	2 (AO3.2)	
	iii	- Reduces the chance of errors/inconsistences - Ensures objects can only be changed in the way intended/ Ensuring changes are consistent with how the object should behave - Protecting data/ Can't be changed accidentally (1 Mark per -, Max 2)	2 (AO1.2)	Read 'securing' as 'protecting'

c		- First line Clear use of inheritance of Obstacle.
		E.g.:
		Person inherits Obstacle /
		Person extends Obstacle /
		Person : Obstacle /
		Person(Obstacle)
		
		In the method
		- Less than 2 metres triggers brake
		- Equal to but not greater than 2 metres triggers brake.
		- Less than or equal to 2 metres triggers horn
		- Less than or equal to 5 metres triggers horn
		(1 Mark per -, Max 5)
			5 (AO3.2)	`class Person inherits Obstacle`
				
				`public procedure updateDistance(givenDistance)`
				`if givenDistance<=5 then`
				`Controls.beepHorn()`
				`if givenDistance<=2 then`
				`Controls.applyBrakes()`
				`endif`
				`endif`
				`distance = givenDistance`
				`endprocedure`
				
				`endclass`
				
				NB a number of ways exist of writing the method – be careful of the logic. Two such correct examples are below.
				`if givenDistance<=5 then`
				`Controls.beepHorn()`
				`endif`
				`if givenDistance<=2 then`
				`Controls.applyBrakes()`
				`endif`
				
				`if givenDistance<=2 then`
				`Controls.beepHorn()`
				`Controls.applyBrakes()`
				`elseif givenDistance<=5 then`
				`Controls.beepHorn()`
				`endif`

	d	Advantages of an automated driver are it is potentially: - safer than a human driver (due to quicker reaction speeds etc.). - cheaper as no wage to cover. - less likely to make mistakes with route. Disadvantages of an automated driver are it is potentially: - May not be able to understand natural speech. - May be limited in terms of the roads on which it can operate. - Vulnerable to hacking. - Only as good as the program running it – a bug in the code could cause catastrophic accidents. - May prioritise safety of pedestrians over that of the passenger. (e.g. may take actions that may put the passenger at risk to save the lives of numerous people outside the car.) - No discussion possible with the driver / no "human presence" to reassure nervous customers. Max 1 advantage and max 1 disadvantage	2 (AO2.2)	

7			- Initialise all 4 totals variables
			- Checks through all items in the array via suitable loop.
			- Add temperatures <=10 to Band A
			- Adds temperatures >=31 to Band D
			- Correctly assigns temperatures between 11 and 20 inclusive to Band B and those between 21-30 inclusive to Band C
			- Uses else if (or equivalent) for efficiency rather than multiple ifs **OR** uses select/case **OR** any other solution that stops trying to categorise a temperature once its band is found.
			- Displays results in similar format to shown in question.
			(1 per -, max 6)
				6
				
				AO3.2
				(6)
					Example:
					`bandA = 0`
					`bandB = 0`
					`bandC = 0`
					`bandD = 0`
					`for i=0 to temperatures[].length - 1`
					`  if temperatures[i]<=10 then`
					`  bandA = bandA + 1`
					`  elseif temperatures[i]<=20 then`
					`  bandB = bandB + 1`
					`  elseif temperatures[i]<=30 then`
					`  bandC = bandC + 1`
					`  else`
					`  bandD = bandD + 1`
					`  endif`
					`next i`
					`print("Band A: " + bandA)`
					`print("Band B: " + bandB)`
					`print("Band C: " + bandC)`
					`print("Band D: " + bandD)`
					
					Some solutions may use Select/Case.
					
					E.g.
					
					`Select Case temperatures[i]`
					`  Case Is<=10`
					`  bandA=bandA+1`
					
					Look out for alternative methods of iteration such as using iterators

	c		- Asks for a number.
			- repeatedly…
			- until 3 is entered.
			(1 per -, max 3)
				3
				AO3.2
				(3)
					`loop  INP`
					`  SUB num`
					`  BRZ end`
					`  BRA loop`
					`end  HLT`
					`num  DAT 3`
					
					Accept answers that use immediate addressing
					ie., SUB #3

GCST

2017

5	a	i	- Stores the value 10 (1) - In a memory location (1) - Given the label/symbolic address ten (1)	3 (AO1.2)	MP3 Accept identifier
		ii	**Starting value in Accumulator** / **Pass or Fail** 29 / Fail 30 / Pass 31 / Fail 1 Mark per row	3 (AO2.1)	
	b	i	LDA (1) SUB (1) ADD (1) INP (1) (Max 1)	1 (AO1.2)	
		ii	BRA (1) BRP (1) BRZ (1) (Max 1)	1 (AO1.2)	
		iii	20	1 (AO2.1)	
		iv	40	1 (AO2.1)	
		v	Rounds up (the number input)... (1) ...To the nearest multiple of ten (and outputs it) (1)	2 (AO2.2)	Rounds to multiple of ten gets one mark.

AS - Level

4	a		- Where weight is less than 4 and volume is less than 0.3m^3 returns 5... (1)
			- ...And only when both weight<4 and volume is<0.3 (1)
			- Where weight is >= 4 and volume >= 0.3 it returns the larger of weight * 2 if that is larger (than volume*20)... (1)
			- ...and volume * 20 if that is larger (than weight * 2). (1)
				4
				AO3.2
					Example
					`function getCost(weight, volume)`
					`  if weight < 4 and volume < 0.3 then`
					`  return 5 (Accept getCost = 5)`
					`  else`
					`  cost=weight*2`
					`  if cost<volume*20 then`
					`  cost=volume*20`
					`  end if`
					`  return cost (Accept getcost = cost)`
					`  end if`
					`endfunction`

If you found this useful, drop a follow to help me out!

THANK YOU!

GCST

GCSECOMPUTERSCIENCETUTOR.COM