TARGET CLEAR

A LEVEL AQA

PRE-RELEASE 2025

Programming Tasks

These questions require you to load the **Skeleton Program** and to make programming changes to it.

Note that any alternative or additional code changes that are deemed appropriate to make must also be evidenced, ensuring that it is clear where in the Skeleton Program those changes have been made.

The objective of this resource is to provide you with a selection of different questions and solutions to those questions. Some questions are more prescriptive than others in how the task should be completed to support a range of learners. Questions which have a similar theme may use different techniques to give students a range of options on how to solve problems. Some Regular Expression solutions use meta characters which are arguably beyond the AQA 7517 specification but make the solution considerably simpler. Students are encouraged to learn these techniques to save coding time in the section D portion of the exam.

Students are recommended to start with a clean copy of the pre-release code before attempting each of the questions in this resource. This will prevent modifications made for one question having an unintended impact on a different question.

Task 1 **Marks:** 3

This question extends the Skeleton Program to allow the user to end the game at any point rather than wait until they are beaten by the **Targets**. Modify the application to allow the user to enter the word "QUIT" to end the game rather than entering an expression. The program should quit and display the final score.

What you need to do

Task 1.1

Update the **PlayGame** method to allow the user to enter the word "QUIT" instead of an expression. Ensure that the code does not decrement the score on that turn.

Test the user input to either play the turn if they enter an expression or quit the game and display the current score.

Task 1.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `y` to start a training game.
- Enter the expression: `8+3-2`
- Show the program correctly identifying the target 9 and awarding the user 1 point.
- When prompted for another expression, enter the word: `QUIT`
- Show the program displaying the "Game over!" message and the final score.

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the modifications to the **PlayGame** method. [2 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

Task 3 **Marks:** 2

The application currently uses list data structures but includes some functionality which would be better suited to stack or queue data structures.

The **UpdateTargets** method uses iteration on a list to interact with the list as if it were a queue. Because the iteration is upperbound by the value of the **MaxNumbersOfTargets** variable, it will always iterate the same number of times. The method therefore has a time complexity of O(1). The code, however, could be simplified to just interact with the start and end of the queue rather than iterating through it. This would make the functionality cleaner.

What you need to do

Task 3.1

Modify the **UpdateTargets** method to simplify how the list is interacted just using start and end indices rather than iterating through it. Ensure that the functionality of the method does not change.

Task 3.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `y` to start a training game.
- Enter the expression: `8+3-2`
- Show the program displaying the updated **Targets** list with the targets moved one index to the left and 119 added to the end of the list.

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the refactoring of the **UpdateTargets** method. [1 mark]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

Task 4 **Marks:** 8

This question extends the functionality of the Skeleton Program to introduce difficulty levels to the game.

Introduce the functionality to have “Easy”, “Medium” and “Hard” levels in the standard random game. Introduce a list of “Large” numbers [25, 50, 75, 100]. The application should populate the **NumbersAllowed** list as follows:

Game Mode	How many “Large” numbers	How many standard random numbers
Standard	0	5
Easy	1	4
Medium	2	3
Hard	4	1

The functionality should produce a brand-new set of NumbersAllowed for each turn even if the user expression does not evaluate to a target.

This functionality is not applicable to a training game.

What you need to do

Task 4.1

Modify the **Main** method to introduce a menu to give the user a choice of the type of standard random game that they would like to play.

Modify the **FillNumbers** method and any other methods required to introduce a difficulty level option in the way described.

Task 4.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Press `enter` to start a standard game.
- Show the program displaying a suitable menu and prompting the user.
- Select a “medium” difficulty game.
- Show the program displaying a **NumbersAllowed** list which contains two values from the list [25, 50, 75, 100].

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the amended **Main** and **FillNumbers** methods and any other methods you have modified when answering this question. [7 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

Task 6 **Marks:** 4

This question extends the Skeleton Program by allowing the user to move the **Targets** list back to the right, but at a cost to their score.

New functionality should be introduced which offers the user the opportunity to move the targets back to the right instead of entering an expression. If they select this option, the **Targets** list should be moved along to the right by one index and the **Score** should be reduced by 2. The list should maintain its standard length. The start of the list should be repopulated with -1.

What you need to do

Task 6.1

Modify the **PlayGame** method to give the user the option to enter an expression or enter "MOVE" to move the **Targets** list.

Create a new method called **MoveTargetsBack** which operates as described.

Task 6.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `y` to start a training game.
- Enter the expression: `512/8+2+2`
- Enter the expression: `8+3-2`
- When prompted to enter another expression, select to move the **Targets** list to the right.
- Show the program displaying the **Targets** list with the order:
  `| | | | |23| |140|82|121|34|45| |75|34|23|119|43|23|119|119|`

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the new method **MoveTargetsBack** and the amended **PlayGame** method. [3 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

Task 7 — Marks: 7

This question extends the Skeleton Program to allow the user to use correctly identified targets in future expressions.

For example, in a standard random game with a **Targets** list and a **NumbersAllowed** list of:

```
|  |  |  |  |  |43|46|24|42|3|15|27|35|8|28|5|4|3|35|36|

Numbers available: 1   9   6   8   2
```

If the user enters the expression `9+6`, this correctly identifies the target number 15. The user should then be given the option to either add the number 15 or just 1 or just 5 into the **NumbersAllowed** list to be used in a later turn. The **NumbersAllowed** list can increase to more than five values. This functionality can add a 0 to the **NumbersAllowed** list if the identified target is a single digit or is divisible by 10.

Introduce new functionality, so that when a target is correctly identified, give the user the option to add the target or part of it into their **NumbersAllowed** list. The program should display the target and its component digits to the user to allow them to choose. If they select to add the target (or one of its component digits) it should be added to the **NumbersAllowed** list.

What you need to do

Task 7.1

Modify the **PlayGame** and **CheckIfUserInputEvaluationIsATarget** methods to display when a target has been correctly identified and prompt the user to choose if they would like to use it to add into the **NumbersAllowed** list.

Create a new method called **SelectValueFromTarget** which displays the correctly identified target and its component digits and invites the user to select which part they would like to add to the **NumbersAllowed** list.

Task 7.2

Modify the **FillNumbers** method to add the selected target from the user to the **NumbersAllowed** list whilst maintaining its functionality at the start of the game of initially populating the **NumbersAllowed** list.

Task 7.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Press `enter` to start a standard random game.
- Enter an expression which correctly identifies a target greater than 9.
- Show the program prompting the user to choose if they would like to use the target in their **NumbersAllowed** list. Select this option.
- Show the program prompting the user which digit in the target they would like to use. The program should show the target and both digits individually. Select one of the options available.
- Show the program correctly adding the selected option to the **NumbersAllowed** list.

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the amended **CheckIfUserInputEvaluationIsATarget** and **PlayGame** methods. [2 marks]
- Your PROGRAM SOURCE CODE showing the new **SelectValueFromTarget** method. [2 marks]
- Your PROGRAM SOURCE CODE showing the amended **FillNumbers** method. [2 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

Task 10 **Marks:** 11

This question extends the Skeleton Program by introducing some object orientation to allow the user to undo previous moves.

Introduce new functionality to allow the user to undo their previous moves. If there are undo moves available, the program should inform the user how many are available and ask them if they would like to undo their last move. If they select this option, the program should undo the last move.

What you need to do

Task 10.1

Create a new class called **UndoState** which stores the copies of the **NumbersAllowed** list, **Targets** list and **Score**. Create suitable accessor methods to be able to access these properties.

Modify the **PlayGame** method to prompt the user appropriately and introduce a suitable data structure to store **UndoState** objects to fulfil the requirements described.

Task 10.2

Create a new method **AddToUndo** which instantiates a copy of the current game state and stores it accordingly to allow undo functionality.

Create a new method **UndoLastTurn** which restores the game to the previous game state, ensuring that the **NumbersAllowed** list, **Targets** list and **Score** are all updated.

Task 10.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `y` to start a training game.
- Enter the expression: `512/8+2+2`
- When prompted, do not undo the last turn.
- Enter the expression: `8+3-2`
- Show the program displaying that 2 undos are available.
- When prompted undo the last move.
- Show the program displaying the **Targets** list:

```
| | | | |23|9|140|82|121|34|45| |75|34|23|119|43|23|119|119|
```

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the new class **UndoState** with a suitable constructor and accessor methods. [4 marks]
- Your PROGRAM SOURCE CODE showing the amended **PlayGame** method with a suitable prompt as described. [2 marks]
- Your PROGRAM SOURCE CODE showing the new **AddToUndo** and **UndoLastTurn** methods. [4 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

Task 11 **Marks:** 12

This question introduces a new "Help Mode" feature to the game by suggesting targets which have been identified as possible, through brute force.

Introduce a "Help Mode" to the game. Create a new method which brute forces different combinations of operators and operands from the **NumbersAllowed** list. Where the brute force approach finds a possible expression, which evaluates to one of the values in the **Targets** list, the program should highlight that target with * * around it. To reduce processing, the brute forcing should **only use expressions involving two or three operands, and only up to five possible targets need to be identified in each turn**. If five possible targets are not found within 30 attempts, the program should stop looking and simply display what it has found. The program **does not** need to display the actual expressions to the user. The brute forcing approach **does not** need to test every possible combination of the values in the **NumbersAllowed** list.

What you need to do

Task 11.1

Modify the **PlayGame** method to give the user the option to have up to five target suggestions identified. Use a suitable data structure to store these values.

Create a new method called **GetRandomSuggestions** which iterates through the **NumbersAllowed** list trying combinations of values from the list with the four mathematical operators used in this game to operate as described.

Task 11.2

Modify the **DisplayState** and **DisplayTargets** methods to operate as described.

Task 11.3

Test that the changes you have made work:

- Run the Skeleton Program.
- Press `enter` to start a standard random game.
- When prompted, choose to receive target suggestions.
- Show the program displaying the **Targets** list with suggested targets highlighted.
- Enter a suitable expression which evaluates to one of the highlighted targets.
- Show the program correctly removing the identified target and updating the score appropriately.

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the new method **GetRandomSuggestions** and the amended **PlayGame** method. [8 marks]
- Your PROGRAM SOURCE CODE showing the amended **DisplayState** and **DisplayTargets** methods. [3 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

Task 12 **Marks:** 7

The program currently does not give a reason if an expression entered by the user does not evaluate to a target in the **Targets** list.

Introduce functionality to tell the user what their expression evaluated to and how many targets were found as a result. Inform the user if they have used any values which are not in the **NumbersAllowed** list.

What you need to do

Task 12.1

Modify the **CheckNumbersUsedAreAllInNumbersAllowed** method to tell the user if they have used any values not present in the **NumbersAllowed** list and which ones they are.

Modify the **CheckIfUserInputEvaluationIsATarget** method to tell the user what their expression evaluated to and how many times it was found in the **Targets** list.

Task 12.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `y` to start a training game.
- Enter the expression: `8+100`
- Show the program displaying a suitable error highlighting that 100 is not valid.
- Enter the expression: `8+3-2`
- Show the program displaying that 9 is a valid target and was found once in the **Targets** list.

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the amended **CheckNumbersUsedAreAllInNumbersAllowed** and **CheckIfUserInputEvaluationIsATarget** methods. [5 marks]
- SCREEN CAPTURE(S) showing the required tests. [2 marks]

Task 13 Marks: 7

This question extends the functionality of the Skeleton Program to clear the **Targets** list more quickly by removing all the targets between two duplicate, correctly identified, targets.

Introduce new functionality so that if an expression evaluates to **pairs of** targets, all of the numbers between those targets are also removed from the **Targets** list, collapsing the gap formed and achieving 2 points per target removed. The **Targets** list should then be backfilled accordingly depending on whether the user is playing a training game or a standard random game.

This question does NOT need to consider the scenario of more than two duplicate targets.

What you need to do

Task 13.1

Modify the **CheckIfUserInputEvaluationIsATarget** method to remove all targets from the **Targets** list if they are between two matching targets identified by the user. The targets themselves should also be removed to close the gap in the list. Award the user 2 points for each target removed.

Modify the **UpdateTargets** method and any other methods required to correctly repopulate the **Targets** list to the default number of targets.

Task 13.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `y` to start a training game.
- Enter the expression: `512/8/2+2`
- Show the program displaying the backfilled **Targets** list:
  `| | | | |23|9|140|82|121|23|119|43|23|119|119|119|119|119|119|119|`

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the amended **CheckIfUserInputEvaluationIsATarget** and **UpdateTargets** methods together with updates to any other required methods. [6 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

Task 15 **Marks:** 10

This question extends the functionality of the Skeleton Program by allowing the user to “reload” a previous game state by entering a character string which represents a game.

Introduce new functionality in the opening menu of the game to allow the user to enter L to load a previous game state using an encoded string. The user should then be prompted to enter a string of 26 characters. Continue to ask the user for a code until a valid code is entered. The first 20 characters should represent the **Targets** list with the @ symbol representing -1. The next five characters should represent the **NumbersAllowed** list, with the final character representing the **Score**. The functionality can only represent a standard random game because of the **MaxTarget** size of 50.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Subtract 64 from the ASCII equivalent of each character in the encoded game string. For example, the letter D should represent the number 4 in the **Targets** list or **NumbersAllowed** list.

What you need to do

Task 15.1

Create a new **RestoreGame** method to allow the user to “load” a game from a previous state encoded string. The method should give suitable error messages if the string is the incorrect length or contains invalid characters. The portion of the string representing the **NumbersAllowed** list and **Score** cannot contain the @ symbol.

Modify the **Main** and **PlayGame** methods to allow the program to operate as described.

Task 15.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `L` to load a previous game state.
- Enter the string: `game.txt`
- Show the program displaying a suitable error.
- Enter the string: `@@@@@Qf_Hg@kCCN@WJhSCHGJAC`
- Show the program displaying the correct "loaded" game state:

```
| | | | | |17|38|31|8|39| |43|3|3|14| |23|10|40|19|

Numbers available: 3  8  7  10  1

Current score: 3
```

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the new **RestoreGame** method together with the amended **Main** and **PlayGame** methods. [9 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

Task 16 **Marks:** 12

This question extends the mathematical range of the Skeleton Program by introducing parentheses.

Add functionality to the game to allow users to input infix notation using brackets in their expressions to control the order of operations.

You can assume that the user will enter a valid number of brackets in their expression.

What you need to do

Task 16.1

Create a new method **ConvertToRPNWithBrackets** to replace the **ConvertToRPN** method. Use a shunting algorithm to correctly interpret the precedence of an expression which includes brackets. Update the application to replace wherever the **ConvertToRPN** method is called.

Modify the **CheckValidOperator** and **CheckIfUserInputValid** methods to update the Regular Expressions to match with a suitable pattern to allow brackets in an infix expression.

Task 16.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Enter `y` to start a training game.
- Enter the expression: `(8+2)*2+3`
- Show the program displaying an updated **Targets** list with 23 correctly removed:
  `| | | | |9|140|82|121|34|45|68|75|34| |119|43| |119|119|119|`

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the new **ConvertToRPNWithBrackets** method and any additional methods required to operate. [9 marks]
- Your PROGRAM SOURCE CODE showing the amended **CheckValidOperator** and **CheckIfUserInputValid** methods. [2 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

Task 19 **Marks:** 14

This question extends the functionality of the Skeleton Program by introducing an auto solver which tests **all** the possible combinations of **NumbersAllowed** and standard operators in an expression, and offers suggestions to the user for different expressions which could be entered to identify targets.

Create new functionality in the program to offer suggestions for expressions which identify targets. The program should prompt the user if they would like to receive suggestions in each turn. If selected, the program should display a list of expressions, together with what they evaluate to, onto the screen. If, in testing evaluations, multiple expressions identify the same target, only one of those expressions needs to be displayed to the user.

What you need to do

Task 19.1

Modify the **PlayGame** method to prompt the user as described and display a list of identified targets with their associated expressions.

Create a new method **GenerateEvaluations** which calculates possible solutions to targets using the values in the **NumbersAllowed** list, to operate as described.

Task 19.2

Test that the changes you have made work:

- Run the Skeleton Program.
- Press `enter` to start a standard random game.
- Show the program displaying a prompt asking the user if they would like helper suggestions.
- Show the program displaying expressions and associated evaluations which could be used to identify targets in the **Targets** list.

Evidence that you need to provide:

- Your PROGRAM SOURCE CODE showing the amended **PlayGame** method. [6 marks]
- Your PROGRAM SOURCE CODE showing the new **GenerateEvaluations** method and any additionally created methods required. [7 marks]
- SCREEN CAPTURE(S) showing the required tests. [1 mark]

