9608/42/PRE/O/N/20

Last update: Anuj Verma, 00:03 12/10/2020

These are the files that constute the solution to the pre-release material for Computer Science component 9608/42 of the October/November 2020 examination series.

Filename	Type	Purpose
9608_w20_PM_42	`.pdf`	The pre-release material file released by CAIE.
Planning	`.md`	This is the markdown text file that this PDF was created from.
Planning	`.pdf`	You are currently reading this file. It describes the solution used in answering the pre-release material and houses all material apart from code (such as identifier tables and structured English).
Main Python notebook	`.ipynb`	The Jupyter Notebook in which the Python code was originally written.
Main Python notebook	`.pdf`	The PDF version of the Jupyter Notebook (for the viewer whose system doesn't have Jupyter).
Python Programs	`.py`	The Python 3.8 file that contains all executable code (for the viewer whose system doesn't have Jupyter).
Assembly code	`.docx`	The assembly code done in *Word* to leverage the tables from the question paper.
TASK_1_1	`.png`	The low-level program as required by TASK 1.1.
TASK_1_3	`.png`	The low-level program as required by TASK 1.3.
TASK_1_5	`.png`	The low-level program as required by TASK 1.5.
TASK_3_2	`.png`	The linked list as required by TASK 3.2.
TASK_3_3	`.png`	The linked list as required by TASK 3.3.

TASK 1 – Low-level programming

> Low-level programming is a type of programming language that uses op codes and operands to create instructions.
>
> The table (in the question paper) shows part of the instruction set for a processor that has one general purpose register, the Accumulator (ACC), and an Index Register (IX).

TASK 1.1

> Write assembly language program code that allows a user to input 5 characters. The characters are not stored.

The program is given in the table below, and is in the attached *Word* document.

Label	Op code	Operand	Comment
	LDM	#0	// initialise COUNT to 0
	STO	COUNT	
LOOP:	IN		// input character
	LDD	COUNT	// increment COUNT
	INC	ACC	
	STO	COUNT	
	CMP	MAX	// is COUNT = MAX ?
	JPN	LOOP	// jump to LOOP if FALSE
	END		// end program
MAX:	5		
COUNT:			

TASK 1.2

> Discuss the purpose of the Index Register and how it can be used to access consecutive memory locations.

The Index Register `IX` is used to modify oprands (such as adresses) in low-level programming; we can use it as a counter. If we access the memory location using indexed addressing, we would access the location `n` places after the specified address if the value stored in `IX` is `n`. Consider the example below to access ten consecutive locations:

```
        LDR #0      // Initialize the index register to zero.

LOOP:   LDX 0xFFF6  // If we try to access a memory loaction using indexed
                    // addresssing now, it would be at 0xFFF + IX.

        INC IX      // Increment the contents of the index register.
                    // This has the effect of moving to the next memory location.

        CMP #9      // Compare the value of the counter (IX) with the end
condition.
        JPN LOOP    // Loop back to LOOP if the end condition is not yet reached.

        END         // End the program if the loop has ended.
```

TASK 1.3

> Write assembly language program code that adds the values stored in four consecutive memory locations starting at `NUMBER` using the Index Register.
>
> Store the final total value in memory location `TOTAL`.

The program is given in the image of the table below, and is in the attached *Word* document.

Label	Op code	Operand	Comment
	LDR	#0	// initialise Index Register to 0
LOOP:	LDX	NUMBER	// load value from NUMBER + contents of Index Register
	ADD	TOTAL	// add value to TOTAL
	STO	TOTAL	
	INC	IX	// increment Index Register
	LDD	COUNT	// increment COUNT
	INC	ACC	
	STO	COUNT	
	CMP	MAX	// is COUNT = MAX ?
	JPN	LOOP	// jump to LOOP if FALSE
	END		// end program
MAX:	4		
COUNT:	0		
NUMBER:	23		
	17		
	38		
	13		
TOTAL:	0		

TASK 1.4

> The assembly language instruction set given has the op code `STX`. Discuss the purpose of this op code.

`STX` copies the contents of `ACC` to the address `<address>` + the value from `IX`. The example below would try to store `#48` in memory location `0xFFF6 + #10` (which is `0xFFF`).

```
LDM     #48
LDR     #10
STX     0xFFF6
```

TASK 1.5

> Amend your solution to **TASK 1.1** to allow the program to store each of the characters input into separate, consecutive memory locations starting at the memory locations labelled `CHARACTER`.

The program is given in the table below, and is in the attached *Word* document.

Label	Op code	Operand	Comment
	LDR	#0	// initialize Index Register to 0
LOOP:	IN		// input value
	STX	CHARACTER	// store the value
	INC	IX	// increment the index register
	CMP	MAX	// check for end of loop condition
	JPN	LOOP	// if the loop has not ended, go to LOOP
	END		// if the loop has ended, end program
MAX:	5		
CHARACTER:			

TASK 2 – Declarative programming

> A knowledge base contains information about students in a class, the colours they like and the colours they do not like. A declarative programming language is used to query the knowledge base.
>
> Some clauses in the knowledge base are shown.
>
> ```
> person(alice).
> person(taylor).
> person(nadia).
> colour(blue).
> colour(red).
> colour(green).
> colour(yellow).
> likes_colour(alice, yellow).
> likes_colour(alice, blue).
> dislikes_colour(taylor, red).
> dislikes_colour(nadia, green).
> ```

TASK 2.1

> Two new students are joining the class: Mehrdad and Nigel. They need to be added to the knowledge base.
>
> Four further colours: pink, orange, purple and black need to be added to the knowledge base.
>
> Write clauses to add the two new students and the new colours to the knowledge base.

```
person(mehrdad).
person(nigel).
colour(pink).
colour(orange).
colour(pueple).
colour(black).
```

TASK 2.2

> Add a clause that states Nadia likes the colour red.

```
likes_colour(nadia, red).
```

TASK 2.3

> Add a clause that states Mehrdad does not like the colour pink.

```
dislikes_colour(mehrdad, pink).
```

TASK 2.4

> Write a goal to find all the colours that a person likes.

We will consider Alice as an example and list all the colours she likes.

```
likes_colour(alice, Colour)
```

TASK 3

> A linked list is an Abstract Data Type.
>
> A linked list is used to store data in a linear structure.

TASK 3.1

> Discuss what a node and a pointer are in terms of a linked list.

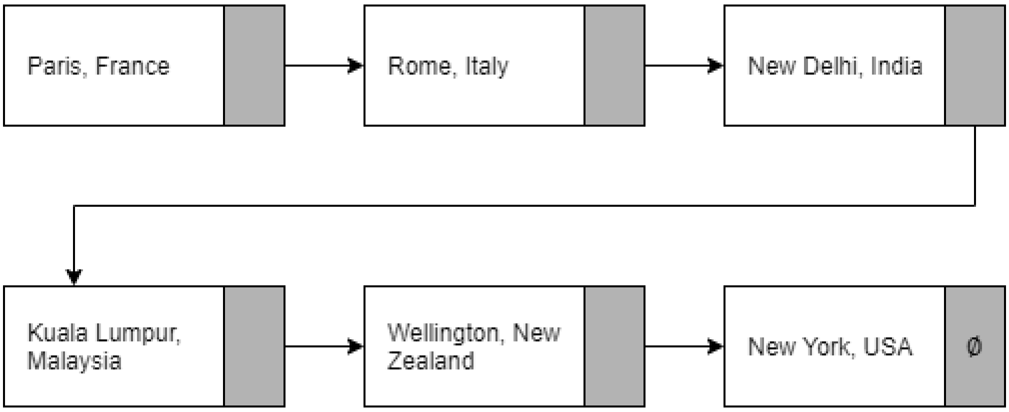
Each element in a linked list is stored in a **node**. Unlike an array where each element simply has an index value and traversing it simply requires incrementing a counter, every element in a linked list leads to the next element directly using a **pointer**.

Consider the linked list of names of students below, which starts at element `[4]` and ends at `[5]`. Each row is a node, and it points to the next element. If the list is traversed using the pointers, we would get this order of elements: Aakash, Anuj, Kalyani, Shiv, Shrey, Sukriti.

Element ID	Element Value	Pointer
0	"Shrey"	5
1	"Kalyani"	3
2	"Anuj"	1
3	"Shiv"	0
4	"Aakash"	2
5	"Sukriti"	-1

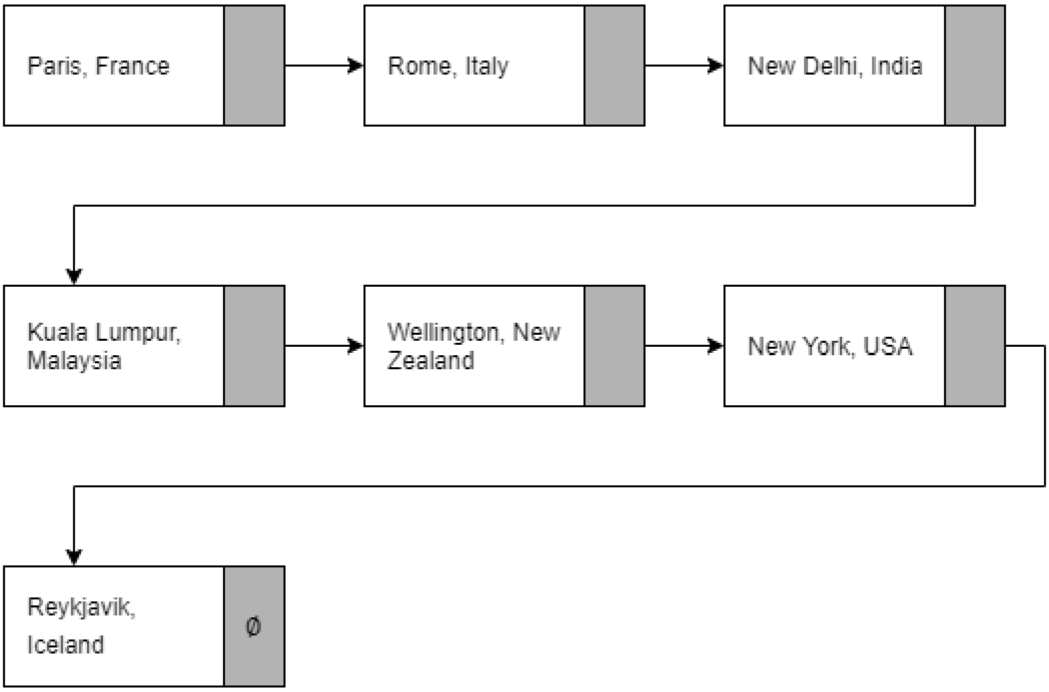
TASK 3.2

> A company has a list of destinations that are visited as part of a round the world holiday.
>
> The destinations are:
>
> - Paris, France
> - Rome, Italy
> - New Delhi, India
> - Kuala Lumpur, Malaysia
> - Wellington, New Zealand
> - New York, USA
>
> The destinations are stored in a linked list in the order shown.
>
> Draw a diagram to represent the data as a linked list.
>
> Use the symbol Ø to represent the null pointer.



TASK 3.3

> A further destination is added after New York; this destination is Reykjavik, Iceland.
>
> Add the new destination to the diagram of your linked list.



TASK 3.4

> Discuss how a node would be removed from the linked list.

A linked list can be setup using two 1-D arrays: `LinkedList` for the data, and `LinkedListPointers` for pointers. An independent subroutine to delete an item `ItemToDelete` (passed as a parameter) from a linked list would be considered.

Firs the routine would traverse the linked list to find `ItemToDelete`; if it could not be found, an error message would be output and the routine would terminate. Once `ItemToDelete` is found (say at index `n`), `LinkedList[n]` is set to a null value. The pointer of the previous element is set to the element after the one at position `n`.

TASK 3.5

> Write **program code** to declare the linked list, using an array.

Identifier	Data Type	Purpose
`insert()`	`PROCEDURE`	A subroutine to insert a new item at the end of the linked list (unless it is full).
tempPointer (`insert()` scope)	`INTEGER`	A temporary holding for the start pointer while a new item is inserted.
`traverse()`	`PROCEDURE`	A subroutine to traverse the linked list and print out its elements.
startPointer	`INTEGER`	A pointer to the first element of the linked list.
heapStartPointer	`INTEGER`	A pointer to the next free location in the linked list.
nullPointer	`INTEGER`	Constant for a termanating pointer.
Destinations	`ARRAY[0:9] OF STRING`	Data stored in the linked list.
DestinationsPointers	`ARRAY[0:9] OF INTEGER`	Linked list pointers.
Destination	`STRING`	A `for` loop element used while inserting items to the linked list.

TASK 3.6

> Extend your **program code** by writing a subroutine that adds a new destination to the end of your linked list.

We already implemented this routine and used it to initiaize the linked list. But a copy along with the identifier table is given here.

Identifier	Data Type	Purpose
`insert()`	`PROCEDURE`	A subroutine to insert a new item at the end of the linked list (unless it is full).
tempPointer (`insert()` scope)	`INTEGER`	A temporary holding for the start pointer while a new item is inserted.

TASK 3.7

> Extend your **program code** by writing a subroutine to delete the destination node entered by the user from the linked list.

Identifier	Data Type	Purpose
`delete()`	`PROCEDURE`	Delete the given element from the linked list.
index (`delete()` scope)	`INTEGER`	The pointer to the element to be deleted.
oldIndex (`delete()` scope)	`INTEGER`	Pointer to the next element.
tempPointer (`delete()` scope)	`INTEGER`	A temporary holding for the start pointer while a new item is inserted.

TASK 3.8

> Discuss other linked list operations that could be implemented. Write **program code** to implement the operation(s) you discuss.

We already wrote a routine to traverse the linked list and print out elements, but we can write two additional routines:

- `find()` to find the given element in the linked list.
- `update()` to change the value of an element.

Identifier	Data Type	Purpose
`find()`	`FUNCTION`	Fuction returns the index of the item passed as the argument, or `-1` if it could not be found.
index (`find()` scope)	`INTEGER`	The temporary variable for the index of the element if it was found.
itemToFind (`find()` scope)	`STRING`	The value of the item to be searched for, passed as a parameter.
`update()`	`PROCEDURE`	Procedure replaces `itemToUpdate` with `newItem` if `itemToUpdate` was found, or throws an error message otherwise.
index (`update()` scope)	`INTEGER`	The temporary variable for the index of the element if it was found.
itemToUpdate (`update()` scope)	`STRING`	The value of the item to be searched for, passed as a parameter.
newItem (`update()` scope)	`STRING`	The new value of `itemToUpdate`, passed as a parameter.