Accredited

A LEVEL

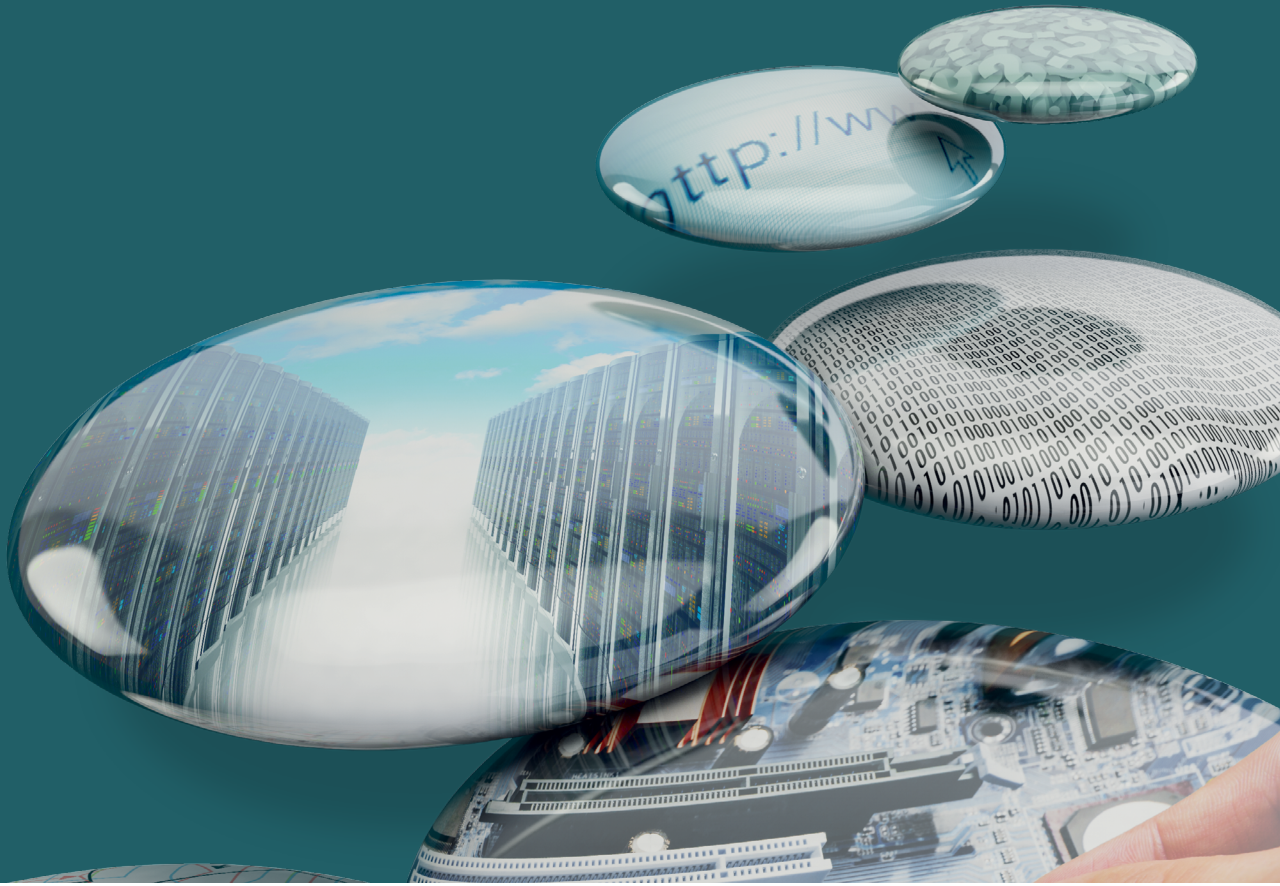
Delivery Guide

H446

COMPUTER SCIENCE

Theme: Programming techniques

September 2015



CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org

OCR
Oxford Cambridge and RSA

Oxford Cambridge and RSA Examinations is a Company Limited by Guarantee. Registered in England. Registered company number 3484466.

Registered office: 1 Hills Road
Cambridge
CB1 2EU

OCR is an exempt charity.

CONTENTS

Introduction

Delivery guides are designed to represent a body of knowledge about teaching a particular topic and contain:

- Content: A clear outline of the content covered by the delivery guide;
- Thinking Conceptually: Expert guidance on the key concepts involved, common difficulties students may have, approaches to teaching that can help students understand these concepts and how this topic links conceptually to other areas of the subject;
- Thinking Contextually: A range of suggested teaching activities using a variety of themes so that different activities can be selected which best suit particular classes, learning styles or teaching approaches.

If you have any feedback on this Delivery Guide or suggestions for other resources you would like OCR to develop, please email resourcesfeedback@ocr.org.uk.

KEY

	Click to view associated resources within this document.
Click here	Click to view external resources.
AS Level only	AS Level content only.

Curriculum Content

Content (from A-level)

Software programs exist to solve problems. So, programming is a highly structured way of problem solving. Students should develop the ability to:

Break the problem down into smaller manageable chunks and understand how these interact

Plan the program before coding it using pseudocode and flowcharts

Visualise the flow of data and understand its types, as well as the process of user interaction with the program

Be aware of the facilities of the chosen language and how well they match the problem at hand

Create efficient code without logical or syntax errors

Be aware of the facilities of the chosen IDE to debug and construct interfaces

2.2.1 Programming techniques

a) Programming constructs: sequence, iteration, branching.

b) Recursion, how it can be used and compares to an iterative approach.

c) Global and local variables.

d) Modularity, functions and procedures, parameter passing by value and by reference.

e) Use of an IDE to develop/debug a program.

f) Use of object-oriented techniques.

Curriculum Content

a) Programming constructs: sequence, iteration, branching.

How to Think Like a Computer Scientist

Learning with Python: Interactive Edition 2.0 (the Runestone Interactive Project at Luther College, led by Brad Miller and David Ranum, based on the original work by: Jeffrey Elkner, Allen B. Downey and Chris Meyers) – http://openbookproject.net/thinkcs/python/english3e/

http://interactivepython.org/runestone/static/thinkcspy/toc.html and specifically:

http://interactivepython.org/runestone/static/thinkcspy/Selection/BooleanValuesandBooleanExpressions.html

http://interactivepython.org/runestone/static/thinkcspy/PythonTurtle/TheforLoop.html

http://openbookproject.net/thinkcs/python/english3e/iteration.html

Interactive online textbook that uses Python to illustrate programming concepts. It has interactive code but is also available as a PDF. The original book is available on the Amazon and direct from publisher (http://www.greenteapress.com/thinkpython/).

Controlling Program Flow in Plain English (John G. McGuinn): Explaining these concepts using everyday actions of making coffee – http://www.tutorials4u.com/c/t05.htm

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-00-introduction-to-computer-science-and-programming-fall-2008/video-lectures/lecture-2/ (MIT Electrical Engineering and Computer Science » Introduction to Computer Science and Programming » Video Lectures » 2: Branching, Conditionals, and Iteration)

This is a good overview but can be a bit dry (but surprisingly accessible).

Curriculum Content

b) Recursion, how it can be used and compares to an iterative approach.

http://openbookproject.net/thinkcs/python/english3e/recursion.html

http://fractalfoundation.org/OFC/OFC-11-1.html

c) Global and local variables.

d) Modularity, functions and procedures, parameter passing by value and by reference.

http://openbookproject.net/thinkcs/python/english3e/functions.html

e) Use of an IDE to develop/debug a program.

http://openbookproject.net/thinkcs/python/english3e/app_a.html

f) Use of object-oriented techniques.

http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_I.html

Thinking Conceptually

Common misconceptions or difficulties students may have

Conceptual links to other areas of the specification – useful ways to approach this topic to set students up for topics later in the course.

The most important technique is the ability to break down a complex task into simple sub-tasks and write (or recycle) self-contained code in the form of functions and procedures (and classes/objects at the higher end of ability). The code that isn't made modular becomes quickly overwhelming and leads to endless debugging rather than program improvement, despite having an added benefit of learners becoming familiar with the debugging tools of their IDE.

It is advisable to introduce functions and procedures as early as possible, before branching and iteration. Left too late, modular programming tends to put learners under stress as it requires them to relearn the skills they have just become comfortable with. Teaching learners that we define our routines in one place and trigger them in another place in a program is great for focusing them and reduces the number of blocks they need to track at any one time. While it might be easier to introduce procedures before functions, it is important that learners get a chance to use both, so they can later use the most appropriate ones for the task at hand.

The use of functions requires more planning and confidence in coding and will not come easy to all learners. Object-oriented programming takes modularity to a new level and introduces another layer of variables – object variables. It is important that learners understand that every task can be done either through procedures with global/local variables, or functions with local variables only and parameter passing between the functions, or with objects and it is advisable to ask pupils to produce multiple versions of the same program using all three paradigms.

Sequence, iteration and branching are best introduced through the various validation routines and any mistakes, especially logical mistakes, become obvious quite soon, simplifying the debugging. Nested branching (an if statement inside another if statement) is not trivial but the process of validating input data requires it, e.g., the type check often needs to be performed before range check and needs planning and sequencing of nested branching.

Non-conditional Iteration is best illustrated through lists/arrays, especially 2d arrays. Being able to generate a times table or read a CSV file are the rights of passage, while conditional Iteration lends itself to validation and user interface duties.

Understanding recursion requires confident knowledge of functions, parameter passing, selection and iteration. In other words, it brings together almost all of the concepts of this section. While recursion can always be replaced by simple iteration, conceptually, it requires more meticulous program planning and hence is a good discriminator in assessment and is also needed for efficient sorting and searching. Sorting a nested list based on one of the sub-columns

Activity 1: Branching

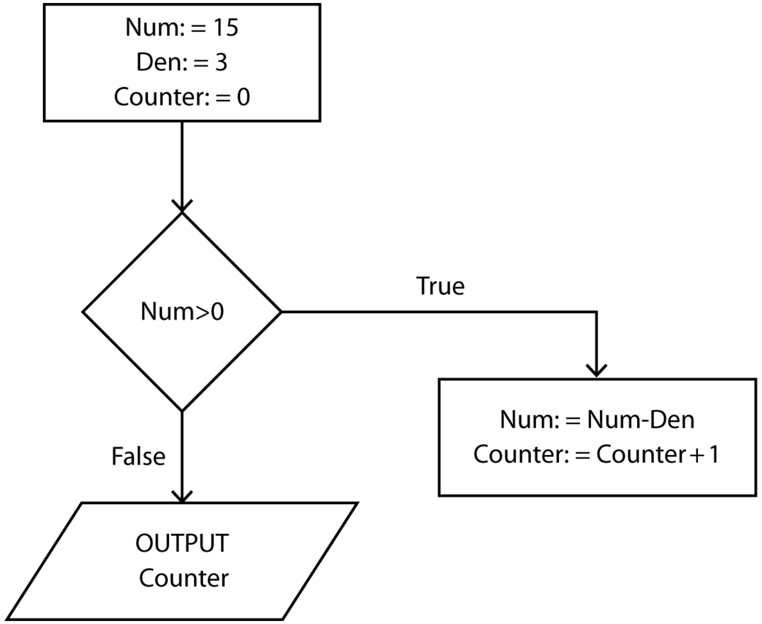
Little Man Computer makes heavy use of branching under three conditions: positive accumulator, zero accumulator and always. Branching effectively replaces iteration.

The explanation can be found here: http://www.gcsecomputing.org.uk/lmc/branching.html.

As LMC doesn't have a division operator, it has to rely on repeated subtraction and counting how many times one number can be subtracted from another without making the result zero. This is analogous to the main point of division – to find out how many times can we fit one number inside the other.

Task 1: Create a flow chart for dividing 15 into 3 without the use of division.

[Solution: Num is numerator, Den is denominator, Counter will give us the result.]



Resources

Learner Resource 1.1

Click here

Thinking Contextually

<table>
<tr><th>Activity 1: Branching</th><th>Resources</th></tr>
<tr><td>

Task 2: Pretend that your favourite high-level programming language doesn't have division or multiplication. How would you implement this example?

[Solution in Python

```
numerator=15
denominator=3
counter=0
while numerator>0:
    numerator=numerator-denominator
    counter=counter+1
print(counter)
```

]

Output:

```
>>>
5
>>>
```

</td><td></td></tr>
</table>

Activity 1: Branching	Resources
Task 3: Implement this program in LMC.	

[Solution]

loop LDA c

ADD one

STA c

LDA n

SUB d

STA n

BRZ loopend

BRP loop

loopend LDA c

OUT

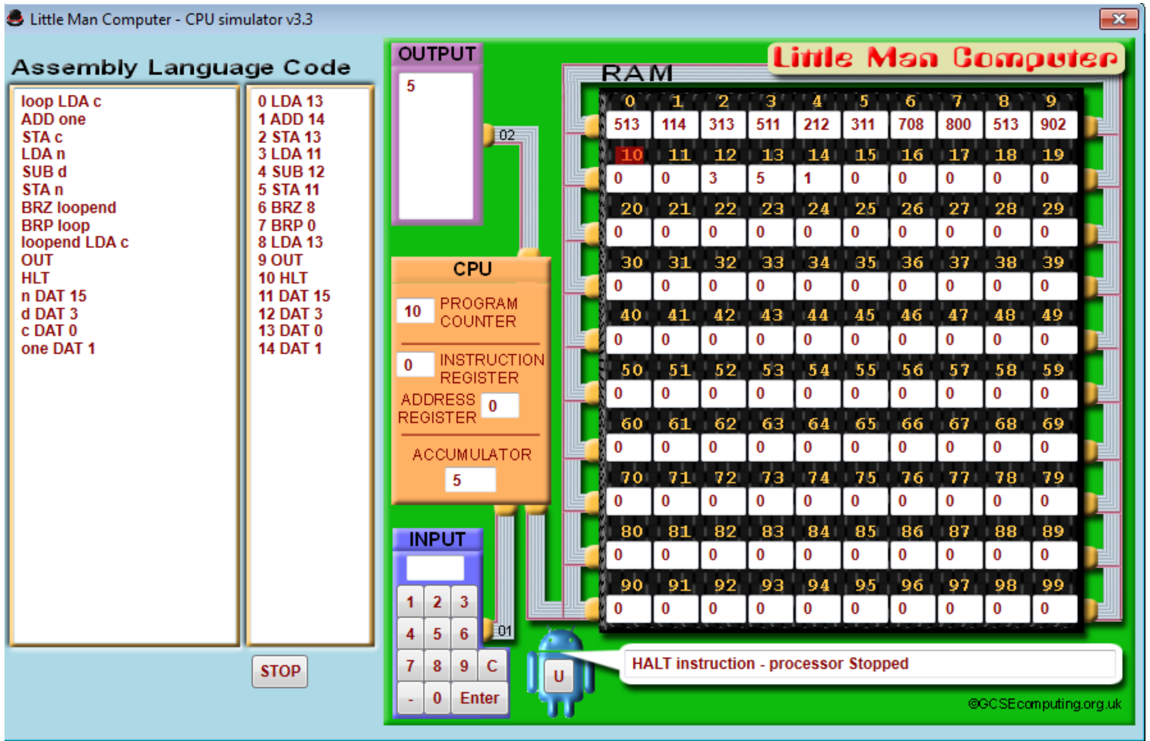
HLT

n DAT 15

d DAT 3

c DAT 0

one DAT 1



[End solution]

Thinking Contextually

Activity 1: Branching

Resources

Task 4: Write out a dry trace table for dividing 15 into 3.

STEP	ACC	PC	CIR	MAR	MDR	DAT 1	DAT 2	DAT 3	OUT

Activity 1: Branching	Resources
[Solution]	

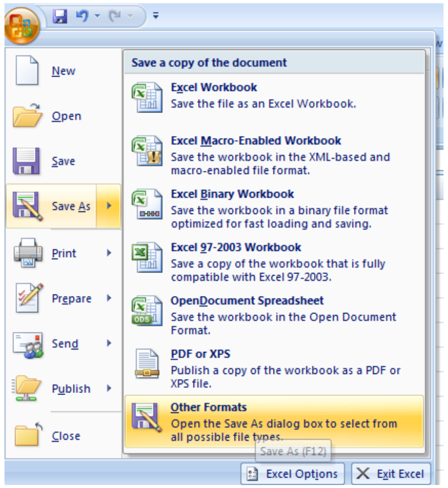
Run LMC in the step mode and copy out the registers' contents into the table.

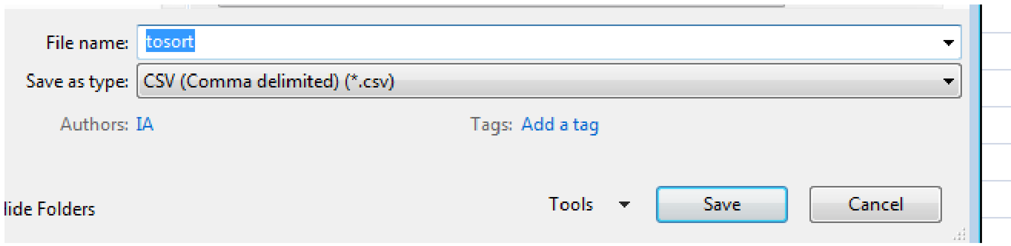
Can be written as:

STEP	ACC	PC	CIR	MAR	MDR	DAT 1	DAT 2	DAT 3	OUT
41	5	10	0	0	0	0	3	1	5

[End solution]

Activity 2: Sorting a nested list	Resources
Nested lists can't be easily sorted by the second column in Python (or most other languages). Task A: Design a solution that can sort a 2-column nested list alphabetically by the second, not first column. [Solution] Step 1:	





Activity 2: Sorting a nested list

Resources

Save in the same folder as your Python program will be.

```
n=[ ] #set up empty list to become the main nested list
f=open("tosort.csv","rt")
contents=f.read()
rows=contents.split("\n")
print(rows)
```

Output:

```
>>>
['John,Bull', 'Peter,Jennings', 'Amira,Kataf', 'Helga,Danson', '']
>>>
```

Need to remove the empty last element:

```
n=[ ] #set up empty list to become the main nested list
f=open("tosort.csv","rt")
contents=f.read( )
rows=contents.split("\n")
rows.remove("")
print(rows)
```

Output:

```
>>>
['John,Bull', 'Peter,Jennings', 'Amira,Kataf', 'Helga,Danson']
>>>
```

Activity 2: Sorting a nested list	Resources
Populate the nested (2d) list n:	

```
n=[] #set up empty list to become the main nested list
f=open("tosort.csv","rt")
contents=f.read()
rows=contents.split("\n")
rows.remove("")
for row in rows:
    n.append(row.split(",")  )
print(n)
```

Output:

```
>>>
[['John', 'Bull'], ['Peter', 'Jennings'], ['Amira', 'Kataf'], ['Helga', 'Danson']]
>>>
```

Activity 2: Sorting a nested list	Resources

Display the list in tabulated form:

```
n=[] #set up empty list to become the main nested list
f=open("tosort.csv","rt")
contents=f.read()
rows=contents.split("\n")
rows.remove("")
for row in rows:
    n.append(row.split(",")   )
print(n)#output in raw form
for each in n: #output in table form separated by tab symbol "\t"
    print("\t".join(each))
```

Output:

```
>>>
[['John', 'Bull'], ['Peter', 'Jennings'], ['Amira', 'Kataf'], ['Helga', 'Danson']]
John        Bull
Peter       Jennings
Amira       Kataf
Helga       Danson
>>>
```

[End solution]

Activity 3: Iterative vs recursive printing out of a list	Resources

Task A: Given a list a=[2,9,6,8,3], develop at least two iterative solutions to print out every item on this list, from the first to the last.

[Solution]

```
a=[2,9,6,8,3] #set up a list
print(a) #print the list in its raw form

for i in a: #iterative solution
    print(i)
#or
for i in range(len(a)):
    print(a[i])

>>>
2
9
6
8
3
>>>
```

[End solution]

Activity 3: Iterative vs recursive printing out of a list	Resources
Task B: Given a list a=[2,9,6,8,3], develop at least two iterative solutions to print out every item on this list, from the last to the first.	

[Solution]

```
for i in range(len(a)-1,-1,-1):
    print(a[i])

for x in a[::-1]:
    print(x)
```

[End solution]

Activity 3: Iterative vs recursive printing out of a list	Resources

Task C: Given a list a=[2,9,6,8,3], develop at least one recursive solution to print out every item on this list, from the first to the least and one recursive solution to print out every item from the last to the first.

[Solution]

```
def r(x):
    if x==0: #terminating condition when counting down
        print( a[0])
    else:
        print(a[x])
        r(x-1) #calls itself
r(4)


def r1(x):
    ubound= len(a)-1
    if x==ubound: #terminating condition when counting up to length #of list
        print( a[ubound])
    else:
        print(a[x])
        r1(x+1) #calls itself
r1(0)
```

[End solution]

Learner resource 1.1

Activity 1: Branching

Little Man Computer makes heavy use of branching under 3 conditions: positive accumulator, zero accumulator and always. Branching effectively replaces iteration.

The explanation can be found here: http://www.gcsecomputing.org.uk/lmc/branching.html.

As LMC doesn't have a division operator, it has to rely on repeated subtraction and counting how many times one number can be subtracted from another without making the result zero. This is analogous to the main point of division – to find out how many times we can fit one number inside the other.

Task 1: Create a flow chart for dividing 15 into 3 without the use of division.

Task 2: Pretend that your favourite high-level programming language doesn't have division or multiplication. How would you implement this example?

Task 3: Implement this program in LMC.

Learner resource 1.1

Task 4: Write out a dry trace table for dividing 15 into 3.

STEP	ACC	PC	CIR	MAR	MDR	DAT 1	DAT 2	DAT 3	OUT

Learner resource 1.1

Activity 2: Sorting a nested list.

Nested lists can't be easily sorted by the second column in Python (or most other languages).

Task A: Design a solution that can sort a 2-column nested list alphabetically by the second, not first column.

Activity 3: Iterative vs recursive printing out of a list.

Task A: Given a list a=[2,9,6,8,3], develop at least two iterative solutions to print out every item on this list, from the first to the last.

Task B: Given a list a=[2,9,6,8,3], develop at least two iterative solutions to print out every item on this list, from the last to the first.

Task C: Given a list a=[2,9,6,8,3], develop at least one recursive solution to print out every item on this list, from the first to the least and one recursive solution to print out every item from the last to the first.

We'd like to know your view on the resources we produce. By clicking on the 'Like' or 'Dislike' button you can help us to ensure that our resources work for you. When the email template pops up please add additional comments if you wish and then just click 'Send'. Thank you.

If you do not currently offer this OCR qualification but would like to do so, please complete the Expression of Interest Form which can be found here: www.ocr.org.uk/expression-of-interest

OCR customer contact centre
General qualifications
Telephone 01223 553998
Facsimile 01223 552627
Email general.qualifications@ocr.org.uk