Accredited

AS and A LEVEL

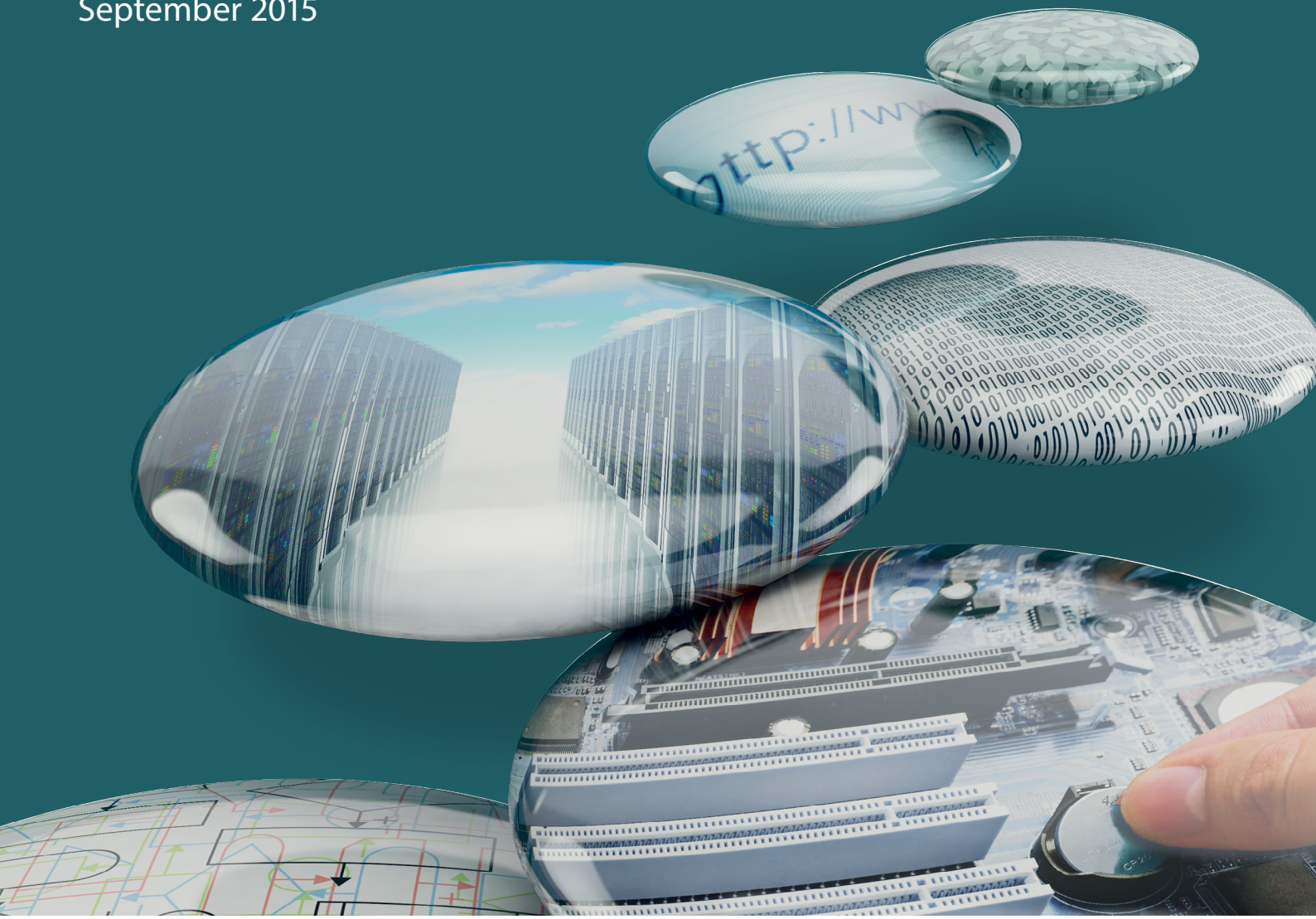
Topic Exploration Pack

H046/H446

COMPUTER SCIENCE

Theme: Types of Programming Language

September 2015



CAMBRIDGE UNIVERSITY PRESS
www.cambridge.org

OCR
Oxford Cambridge and RSA

We will inform centres about any changes to the specification. We will also publish changes on our website. The latest version of our specification will always be the one on our website (www.ocr.org.uk) and this may differ from printed versions.

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

This Topic Exploration Pack should accompany the OCR resource ‘Types of programming language’ learner activities, which you can download from the OCR website.

This activity offers an opportunity for English skills development.

Types of programming language

An average programmer learns a number of programming languages and paradigms over his/her career. Different projects require the knowledge of different languages, depending on the synergies between the language and the objectives of the project. The hardest language to learn is the first one; then most constructs begin to repeat, while the skills remain the same – debugging, algorithm design, iteration, selection, arrays, interface, etc.

Most languages belong to one of the few paradigms – procedural, functional, low-level or object-oriented. Some languages incorporate more than one paradigm. Most common and basic programming is done with procedural languages, such as Visual Basic, C, JavaScript (some functional languages can be said to be close enough to procedural) but Java and C# are object-oriented, while Python can be anything. It is important for learners to understand the principles of splitting a larger task into smaller self-contained sub-programs (procedures). The trickiest part is parameter passing, especially when more than one variable is being passed at once. The next biggest problem is the scope of variables – local and global and passing data to the wrong variable (or of the wrong type).

Object-oriented languages take the notion of self-contained subs further by relying on mini-templates (classes) which combine procedures (called methods) and class/object variables. Objects are much more self-contained and are easily recycled either inside the same or between multiple programs. Object-oriented programming (OOP) programming is more complex than procedural due to more layers of variables and more cryptic error messages – due to the more complex structure, errors become harder to trace. It is also hard to justify to pupils the use of OOP on shorter programs (most programming examples are short). It needs to be communicated to learners that OOP only really pays off on larger programs with a lot of repetitive codes. While encapsulation is a straightforward concept related to the local variables in procedural languages, inheritance and polymorphism present more of a challenge as they require better planning which pupils often tend not to do.

Assembly programming is in a league of its own. Pupils should be familiar with the foundations of LMC from GCSE Computing but since that task was just one of those available, some pupils might not have experience with it. Assembly programming is very time intensive and requires extensive flowcharting before implementation; otherwise, it is easy to get stuck with errors. Pupils need to understand that LMC is just a learning tool and professional assemblers are much more powerful and still quite often used in projects where hardware is slow and the interface is not demanding. A particular skill of low-level programming is memory addressing modes. While operands look very similar, there is a difference between ‘add 5’ (immediate) and ‘add whatever is stored in register 5’ (direct), ‘add whatever is stored in the location number you will find in location 5’ (indirect), ‘add whatever is stored 5 locations ahead of the current’ (indexed).

To prepare learners for assembler programming, it might be prudent to create a list of programs that will be attempted and do them first in a regular high-level language. It is also useful to have a working knowledge of the operators that LMC uses, as well as the use of labels. Without labels it is impossible to do any branching. The best site for it is http://www.gcsecomputing.org.uk/lmc/index.html, which contains examples and a list of operators. It also wisely recommends the use of labels that don't necessarily do anything functional to mark the place where conventional IF ELSE structure would be placed in a high-level language.

The most confusing aspect of LMC is the constant use of the Accumulator and the limited set of commands – there is no multiplication or division and there is a 'while' loop but no 'for' loop, and the fact that variables are declared at the end of the program. The hardest programs are the ones that use nested loops – not a trivial solution in LMC.

It might also be useful to look back at Basic language (VBA still supports this feature) with its numbered lines and GOTO statements. While these have been banished from most modern programming, it remains the foundation of LMC branching, so perhaps doing a few programs in Basic (it can be found through emulators online and offline) will help with the concept of simple branching as used by assembler.

Version 1

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