
Education and Information Technologies 7:1, 55–66, 2002.

Difficulties in Learning and Teaching Programming—Views of Students and Tutors

IAIN MILNE AND GLENN ROWE
Department of Applied Computing, University of Dundee, Dundee, DD1 4HN.
E-mail: imilne@computing.dundee.ac.uk; growe@computing.dundee.ac.uk

We have conducted a web-based questionnaire on the various concepts and topics of object-oriented programming that students on introductory courses found most difficult to cope with.

A statistical analysis of our results shows that those topics that rely on a clear understanding of pointers and memory-related concepts (such as copy constructors and virtual functions) prove to be the most difficult. In other words, we believe these concepts are only hard because of the student's inability to comprehend what is happening to their program in memory, as they are incapable of creating a clear mental model of its execution.

These results would suggest that a clearer approach to teaching these topics would be beneficial to students. We are currently working on a visualization-based approach to address these issues.

Keywords: object-orientation; programming; novices; programming difficulties; software visualization.

Introduction

Although there have been empirical studies of programmers and programmer comprehension with regards to procedural and object-oriented languages, little work (if any) has been applied to discovering the individual traits of a particular language that cause the most difficulty for novices.

Studies into whether choice of programming language affects program comprehension are well documented, and have shown that different notations facilitate the understanding of different kinds of information found in programs (Wiedenbeck, 1999a; Wiedenbeck, 1999b; Good *et al.*, 1997).

Other studies (Burkhardt *et al.*, 1997; Blackwell, 1996; Turner, 2001) have conducted research into the types of mental models formed by both novice and expert programmers, and how such models affect their understanding of the problem and its solution.

This paper investigates object-oriented (OO) languages—C++ in particular—and queries both students and tutors of the subject on the individual concepts of the language which they struggle to learn and consequently teach.

The lack of these types of studies may be explained by current lecturers of programming simply relying on experience and intuition to inform them of what is difficult and what is not. However with the results presented here we can provide statistical evidence that supports these arguments.

A web-based questionnaire has been completed by 66 respondents, made up of 2nd year programming students from Dundee, and lecturers and teachers of programming from around the UK, with the purpose of determining the most common stumbling blocks encountered by students enrolled in a first course in object-oriented programming.

Method

First year computing students at Dundee complete one full year of procedural programming using C. Second year students then take this knowledge further by completing another full year of programming, this time in C++. In both courses, they are introduced to all the main programming concepts for that language, and may also learn additional data structures and algorithms. Recently, the second year has also been used to give them a basic introduction to graphical user interface programming using Microsoft's Visual C++ 'MFC' API (Application Programming Interface). The students were given the questionnaire after their course was complete.

Just over a third of the overall responses were collected internally from Dundee. For the other responses collected, the questionnaire was offered to subscribers of the Learning and Teaching Support Network (LTSN) *ltsn-ics-computing* electronic mailing list. (LTSN is a UK higher-education network, aiming to promote high quality learning and teaching through the development and transfer of good practices in its subject disciplines). The majority of this list's recipients are actively involved in teaching university level students computing.

The questionnaire was split into three sections:

The first part collected general information about the respondent, such as what operating system and programming languages they've had experience with.

The second part formed the bulk of the questionnaire, presenting a series of twenty-eight questions that asked the respondent to grade programming concepts according to how difficult they were to learn, using a simple scale of 1 to 7. An answer of 1 meant they thought the topic was very easy to learn, and an answer of 7 meant very hard. The topics were presented as a single set, with the more general (procedural) questions forming the first half, and OO questions the latter half, including several that are exclusive to C++ (the main language of interest). The topics selected cover the majority of programming concepts in C and C++, and are those most likely to be encountered by novice programmers tackling an OO language for the first time.

Finally, the third part was provided to give more detailed feedback if the respondent wished to, and also gave them the chance to provide an email address if they were willing to be contacted in the future about the project. Valuable information was also gained from the personal comments received.

A full copy of the questionnaire as it was presented can be found in Appendix I.

Results

A summary of the results is presented in Tables 1 and 2. Table 1 details responses from LTSN and Table 2 details the student responses from Dundee. Each table lists the

Table 1. Results from LTSN respondents

Topic	Code	Respondents	Average score
Pointers	PTR	37	6.054
Virtual functions	VIR	32	5.969
Dynamic allocation of memory (with malloc)	M_A	34	5.882
Polymorphism	POL	39	5.615
Recursion	REC	38	5.553
Copy constructors	CPY	32	5.438
Other data structures (trees, linked-lists)	DST	37	5.378
Templates	TEM	36	5.194
Dynamic allocation of memory (with new)	M_N	37	5.162
Operator overloading	OOV	38	5.158
Function over-riding (in inheritance)	FOR	40	5.050
Function overloading/default arguments	FOV	40	4.950
Passing by reference/passing by value	REF	40	4.850
Inheritance	INH	40	4.825
Classes and objects	CLA	40	4.600
Encapsulation	ENC	39	4.564
Casting	CAS	39	4.462
Constructors/destructors	CON	38	4.342
Arrays	ARR	39	4.308
Scope of variables	VSC	39	4.308
Structs	STU	35	4.229
Input/output and file handling	IN_O	39	4.205
String handling	STR	38	4.000
Looping operations (for, while, etc)	LOO	39	3.436
Operators and precedence	O&P	39	2.974
Conditional operations (if, else, etc)	CDO	39	2.949
Basic function calling/program flow	BFC	38	2.895
Variable/function declarations	VFD	38	2.895

programming topics queried, along with the average score (out of 7) that they received. They are presented in "hardest" to "easiest" order. (Note: the reason for topics listed in bold text is given later in the discussion. Each topic also has a 'code' associated with it, which can be used when referring to the graphs).

Please refer to the tables when needed.

Not counting the students from Dundee, the respondents had previous experience with the following languages listed on the questionnaire:

- 95% had used Java
- 90% had used C++
- 82.5% had used C
- 57.5% had used Fortran
- 42.5% had used Smalltalk

Table 2. Results from student respondents

Topic	Code	Respondents	Average score
Copy constructors	CPY	26	4.654
Operator overloading	OOV	26	4.538
Templates	TEM	24	4.250
Dynamic allocation of memory (with malloc)	M_A	26	4.192
Pointers	PTR	26	4.154
Other data structures (trees, linked-lists)	DST	26	4.154
Recursion	REC	26	4.115
Casting	CAS	18	4.111
Function overloading/default arguments	FOV	26	3.962
Virtual functions	VIR	26	3.923
Dynamic allocation of memory (with new)	M_N	26	3.846
Function over-riding (in inheritance)	FOR	26	3.808
Polymorphism	POL	25	3.800
Constructors/destructors	CON	26	3.769
Input/output and file handling	IN_O	26	3.769
Passing by reference/passing by value	REF	26	3.692
Inheritance	INH	26	3.423
String handling	STR	26	3.423
Structs	STU	25	3.400
Encapsulation	ENC	26	3.385
Arrays	ARR	26	3.192
Classes and objects	CLA	26	3.000
Scope of variables	VSC	25	2.560
Looping operations (for, while, etc)	LOO	26	2.346
Conditional operations (if, else, etc)	CDO	26	2.154
Operators and precedence	O&P	25	2.040
Basic function calling/program flow	BFC	26	1.885
Variable/function declarations	VFD	26	1.885

Discussion

From a first glance over these results it becomes apparent that the students have given each topic an 'easier' score than its equivalent score in the LSTN table. Obviously, it is the students who primarily encounter the various problems listed, and one would think that they can therefore communicate their difficulties more clearly than any expert, yet we believe that in this case, students can tend to answer such questionnaires with the wrong answers. That is, they may *believe* they understand a topic such as copy constructors, but upon detailed examination or one-to-one querying from a tutor it turns out that they are often wrong in their belief.

Upon closer examination of the results however, it is clear that there is a common theme to the top six ranked concepts in each table—bar 'recursion' and 'templates', they all rely heavily on the programmer having a clear understanding of pointers and memory (these are the topics listed in bold text within the tables. In fact, in both tables, these topics appear towards the top).

Our reasoning behind classing these 'bold' topics as pointer/memory topics can be explained as follows:

Pointers and dynamic allocation of memory are obviously self-explanatory, yet there are a surprising number of more complex object-oriented concepts that can never fully be comprehended without the student first mastering pointers, and hence realising what their program is doing in memory. For example, passing parameters to a function becomes far more complicated once pointers are introduced and passing by reference is to be used.

Copy constructors and virtual functions not only both exist within OO programming, but do so due to the specific use of pointers within classes and their instances. Polymorphism allows the creation of objects that point to a base class but call derived class functions, resulting in the need for virtual functions, whereas copy constructors are used when a class containing pointers to memory is passed to a function by value.

The need for copy constructors in C++ is similar to the need for the overloaded assignment operator—that is, whenever a class containing pointers to memory is used in an assignment, the programmer must provide the overloaded assignment operator. Omitting copy constructors or overloaded assignment operators when designing classes can lead to run-time results that prove awkward for novices to debug, as they are unaware of the potential memory-related problems their program is generating.

The 'other data structures' topic also received a high average score. Although there are many simple data structures, the majority taught at the academic level we are dealing with here again involve a heavy use (and hence understanding) of pointers. For example, the various tree and list structures are often programmed using pointers to reference their nodes, whereas in the case of hash tables, the programmer is required to think carefully about how data will be stored and accessed in memory as the data structure they are dealing with can be non-linear.

Pointers as a 'concept' are usually easy enough to explain; it is their actual implementation and subsequent behaviour in the running program that causes the problems for most students. This is true of many of the more difficult programming concepts. As an example, most students have little trouble comprehending what recursion is *for*, but the actual process of programming it causes many problems due to its non-linear structure and execution. Simply viewing recursive source code and trying to follow *what* happens without any suitable visualization of the execution is a difficult task for most students, hence recursion's high ranking.

Templates—although not ranked as hard as some of the other OO topics by the LTSN group—ends up near the top of the students' table. We believe this to be true due its inclusion towards the end of the Dundee 2nd year course, hence allowing the students very little time to properly understand it.

Some of the concepts listed here will be classed as difficult in one language, yet relatively easy in another. For example, with just the basic C++ language available (that is, not using the newer standard template library (STL) classes), all string-based operations have to be at the low level, using char arrays or pointers to arrays. Therefore, for anything other than the very simplest of programs, a good understanding of how pointers work with arrays is required to actually do anything useful with strings, as we may have to know how much memory we have allocated for it, or keep in mind whether the string terminates with a null

character or not. However, these operations become much easier when encapsulated into an actual class, such as within the C++'s Standard Library 'string' class, Java's 'String' class, or even MFC's 'CString' class. But as with any higher-level API, it could be argued that to truly understand it, a thorough knowledge of the lower level constructs it is built upon is required.

As another example, several of the respondents state that the teaching of pointers in C++ comes much more easily after the students have had experience using Java. Pointers are not explicit in Java; instead, every object (other than primitive types) is always accessed by reference. This means the students become accustomed to the 'pointer' concept without worrying about the syntactical difficulties of pointer programming in C/C++ as they no longer have to deal with source code that can be misleading and difficult to read due to the languages' use of the '*' and '&' operators.

Classes and objects are key to the whole object-oriented paradigm, yet they are ranked as one of the least difficult OO concepts presented in the questionnaire. Students *do* struggle with this topic however, especially if they are learning an OO language after experience with procedural programming.

There is some disagreement in the academic community over how OO languages should be taught (Culwin, 1997). On the one hand, there are those that believe that OO languages should only be approached once the student has a thorough grounding in procedural programming, as this way they tend to have less problems with understanding basic program flow. This opinion was expressed by several of the respondents who are now teaching an OO language (such as Java) to students as their first language, and found that although they may grasp concepts such as classes and objects more quickly, and with little difficulty, they continually struggle with the simple conditional and looping constructs.

Statistical Analysis

We will now apply some statistical analysis in order to quantify the discussion in the previous section. We shall begin by considering the question of the significance of the difference between the various scores shown in Tables 1 and 2. We can measure the probability that the data for two different topics, such as pointers and recursion, are drawn from different distributions. In other words, we can determine the probability that the respondents, on average, really did think that one topic was more or less difficult than another.

The appropriate test for this measurement is the t-test. We will use the t-test to test the difference between the means of two independent scores, therefore a 2 sample t-test (Howell, 1992) is appropriate. However, because we have more than two groups, we must take them in pairs and apply t-tests to each pair individually. As we are trying to determine the probability of the hardest ranked topics actually being harder than the lowest ranked ones, for each table we shall test the hardest against three other topics, located at the top, middle and bottom of the table.

For Table 1 this will be *pointers* (PTR) against *dynamic allocation of memory (with malloc)* (M_A), *casting* (CAS), and *variable/function declarations* (VFD). As the ordering of topics in Table 2 is slightly different, the test will use *copy constructors* (CPY) against *operator overloading* (OOV), *constructors* (CON), and VFD again.

For each of the three tests, the null and alternative hypotheses are:

H_0: The means of the two scores are equal
H_1: The means of the two scores are not equal

The alternative hypothesis is bi-directional, and thus we will use a two-tailed test. The results of the three tests for each table are presented below:

T-TEST (LTSN)	PROBABILITY
PTR versus VIR	0.746
PTR versus INH	0.000
PTR versus VFD	0.000
T-TEST (STUDENTS)	**PROBABILITY**
CPY versus OOV	0.810
CPY versus CON	0.049
CPY versus VFD	0.000

In the first case, PTR and M_A, the results show an almost 75% probability of the means being equal, which is obviously not grounds for rejecting H_0. However, with PTR and CAS, and PTR and VFD, the results show probabilities of 0.000% in both tests. For these two latter cases, the probability is low enough to conclude that there is a significant difference between the means of the populations from which our observations were drawn.

Similarly, with the student t-tests, the latter two cases return probabilities which are significantly low.

From this, we can say that the respondents to our questionnaire certainly believe that understanding pointers is more difficult than understanding inheritance, which in turn is more difficult than understanding variable and function declarations.

Taking this test further, we can plot out the results of testing these three topics against every other topic in the questionnaire. The chart shown in Figure 1 shows the probability of PTR, INH, and VFD being drawn from the same distribution as the other topics. (Note that the ordering of the x-axis follows that of Table 1, so topics at the top of the table are plotted towards the right of the chart).

As shown, it becomes clear that there's a significant chance of a topic being harder (or easier) than all but its nearest five or six neighbours in the table.

Although not shown here, plotting a similar chart for the student data returns almost identical curves, albeit offset more to the left due to the overall average of the student scores being less.

However, what we have been trying to demonstrate in this paper is that it is pointer and memory topics that cause the most problems. Therefore, we can modify our t-tests so that rather than testing individual topics against each other, we can group together all of the pointer/memory topics and test them against the remaining (non pointer/memory) ones.

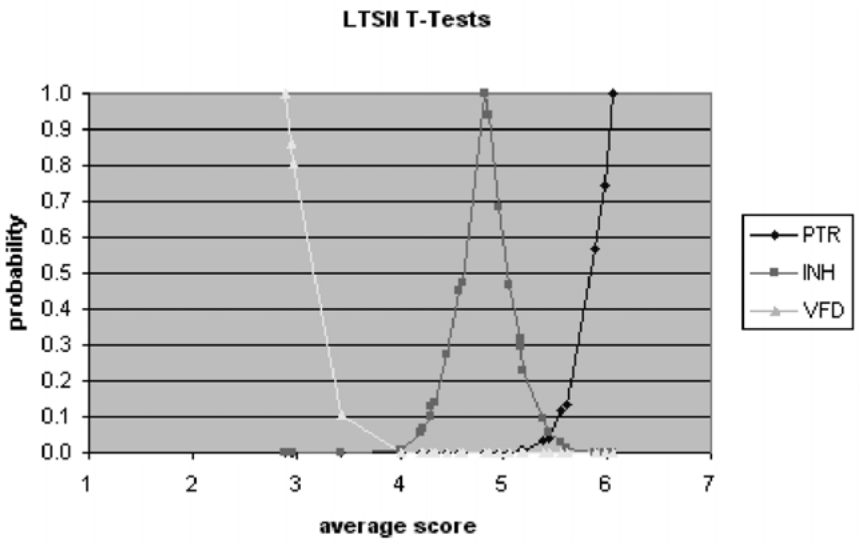


Figure 1. Results of calculating t-tests for the top, middle, and bottom topics (as ranked in Table 1) against every other topic in the questionnaire.

This test was run three times—on all the results combined, on just the student results, and on just the LTSN results. In all three cases, the t-tests returned a probability of the pointer/memory topics being drawn from the same distribution as the other topics of less than 0.001.

Finally, we can analyse the differences between the student results and the LTSN results. From looking at the average scores given by both groups, it becomes apparent that the students gave 'easier' scores to every single topic. However, this does not account for the ordering differences between the two tables. The t-tests have shown though, that we can only state that "Topic A is more difficult than Topic B" with great confidence so long as the two topics are not very close to each other in the table. Hence, some variation in the ordering is acceptable.

We can show though, that on the whole, students and lecturers agree on the relative difficulty of the various topics. This is shown in Figure 2 where we display a scatter plot of the student scores against the LTSN scores. Plotting a line through the points (using a linear fit, least-squares technique) returns an R^2 value of 0.77.

Conclusion

This paper has presented results from a web-based questionnaire whose purpose was to rank programming concepts in order of difficulty, both from the students' points of view, and those of their lecturers.

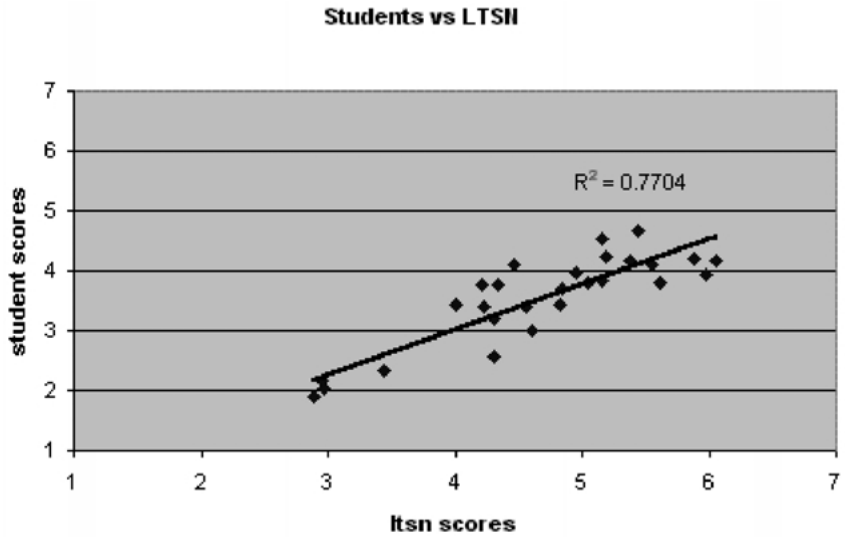


Figure 2. A scatter plot of the student results against the LTSN results, showing a strong correlation between what each group ranked as 'hard'.

We believe that the results show that the most difficult topics are so ranked because of the lack of understanding by the students of what happens in memory as their programs execute. Therefore, the students will struggle in their understanding until they gain a clear mental model of how their program is 'working'—that is, how it is stored in memory, and how the objects in memory relate to one another.

This provides us with the motivation to design a program visualization tool whose primary goal is to aid and enhance the programmer's understanding of what is happening in memory as their program executes.

Program visualization tools aimed specifically at novices have started to appear over the last few years (see (Smith and Webb, 2000; Boroni *et al.*, 1996), and (Rowe, 2000) for examples of prototype systems based on C and Pascal, and (Fernandez, 1998) and (Jerding and Stasko, 1994) for ones covering high-level OO concepts).

What we aim to do now is to take this kind of work further and produce a system capable of visualizing OO programs by building upon this existing work, providing detailed syntactical visualizations in addition to the high-level ones already defined. Development is now under way on a system that can provide these visual aids to the comprehension process.

For those that are interested, the questionnaire itself is still available online at the following URL: http://134.36.34.138/imilne/softvis-web/

The tables of results presented in this paper are also available via the above link, and are updated automatically whenever new questionnaire submissions are made. It is hoped that over the course of time, more comprehensive results can be collected by offering the questionnaire out to as many students and lecturers as possible.

References

Blackwell, A. (1996) Metacognitive Theories of Visual Programming: What do we think we are doing? In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, pp. 33–43.

Boroni, C. M. *et al.* (1996) Dancing with Dynalab. In *Proceedings of the 27th SIGCSE Technical Symposium on CS Education.* Philadelphia, February. pp. 135–139.

Burkhardt, J.-M. *et al.* (1997) Mental Representations Constructed by Experts and Novices in Object-Oriented Program Comprehension. In *Human Computer Interaction: INTERACT'97.* Sydney, July. pp. 339–346.

Culwin, F. (1997) Objects First, Objects Last or Objects At All? In *5th Annual Conference on the Teaching of Computing.* Dublin, August. pp. 56–59.

Fernandez, A. *et al.* (1998) *A Learning Environment to Improve Object-Oriented Thinking.* Presented at OOPSLA '98, Vancouver, October.

Good, J. *et al.* (1997) Novices and Program Comprehension: Does Language Make a Difference?" In *Proceedings of 19th Annual Conference of the Cognitive Science Society.* Stanford University, August. pp. 936–937.

Howell, D. C. (1992) *Statistical Methods for Psychology.* Third Edition. PWS-Kent Publishing, Belmont, CA.

Jerding, D. F. and Stasko, J. (1994) Using Visualization to Foster Object-Oriented Programming Understanding. Georgia Institute of Technology Technical Report GIT-GVU-94-33.

Rowe, G. R. (2000) VINCE — An on-line tutorial tool for teaching introductory programming. *British Journal of Educational Technology,* **31**(4), 359–369.

Smith, P. A. and Webb, G. I. (2000) The Efficacy of a Low-Level Program Visualisation Tool for Teaching Programming Concepts to Novice C Programmers. *Journal of Educational Computing Research,* **22**(2), 27–39.

Turner, T. A. and Zachary, L. (2001) Javiva: A Tool for Visualizing and Validating Student-Written Java Programs. In *Proceedings of 32nd SIGCSE Technical Symposium on Computer Science Education.* Charlotte, February. pp. 45-49.

Wiedenbeck, S. (1999a) Novice comprehension of small programs written in the procedural style. *International Journal Human-Computer Studies*, **51**(1), 71–87.

Wiedenbeck, S. *et al.* (1999b) A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, **11**, 255–282.

Appendix 1

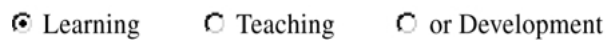
What follows is a copy of the questionnaire as it originally appeared. It is still available at: http://134.36.34.138/imilne/softvis-web/

The main purpose of this questionnaire is to find out what areas of programming you find difficult to cope with, and how an educational software visualization tool could provide ways of dealing with, or overcoming, such difficulties.

All questions are optional, but the more you answer, the easier it will be to process the results.

Basic information

Regarding programming, are you primarily involved with:



For how long, approximately:

0 (years) 0 (months)

And with what experience in the following languages:

C	○ Beginner	○ Competent	○ Expert
C++	○ Beginner	○ Competent	○ Expert
Java	○ Beginner	○ Competent	○ Expert
Smalltalk	○ Beginner	○ Competent	○ Expert
Fortran	○ Beginner	○ Competent	○ Expert

When writing programs, what combination of editor and compiler do you mainly use (for example, "Emacs and gcc" or "Microsoft Visual Studio"). If you use more than one, just list them one after another.

In addition to this development environment, do you use (either separately, or as part of the tool):

◉ No debugger ○ A visual debugger ○ A textual debugger

Some questions on common programming topics and/or problems

The following is a list of concepts from both procedural and object-oriented programming.

Try and rank them according to how difficult you believe each topic was to learn (or teach), where a score of 1 means you thought the topic was very easy, and a score of 7 means you thought it was very difficult.

You may not have heard of all the concepts—if so, just skip those ones and move on.

Variable/function declarations	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Basic function calling / program flow	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Operators and precedence	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Conditional operations (if, else, etc)	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Looping operations (for, while, etc)	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Scope of variables	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Input/output and file handling	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Arrays	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7

Other data structures (trees, linked-lists)	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
String handling	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Structs	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Pointers	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Dynamic allocation of memory (with *malloc*)	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Dynamic allocation of memory (with *new*)	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Recursion	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Classes and objects	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Casting	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Inheritance	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Encapsulation	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Polymorphism	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Passing by reference/passing by value	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Function overloading/default arguments	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Function over-riding (in inheritance)	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Constructors/destructors	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Templates	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Copy constructors	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Operator overloading	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7
Virtual functions	○ 1	○ 2	○ 3	○ 4	○ 5	○ 6	○ 7

And finally

If you would like to give answers as to why you found any of the above topics difficult, then please do so below. Alternatively, if there are any topics or concepts you find particularly difficult which aren't listed, you can mention them here.

If you are willing to participate in an additional one-to-one discussion about these topics at some future date, please enter your email address below. Such a discussion would only last about five or ten minutes.

Submit Answers