SCOTTISH QUALIFICATIONS AUTHORITY

Higher National Unit Specification

General information for centres

Unit title: Software Development: Abstract Data Structures

Unit code: D76L 35

Unit purpose: This Unit is designed to enable candidates to become familiar with the data structures and collection structures in common use within current software systems. This knowledge will be supplemented by research, analysis, design and coding of collection structures in order to create applications to meet complex user requirements.

This Unit is suitable for candidates who wish to pursue a career in Software Engineering or who construct applications requiring the use of non-trivial data structures.

On completion of the Unit the candidate should be able to:

1. Investigate data representation and storage in computer systems
2. Utilise searching and sorting techniques in the manipulation of a collection of data items
3. Define interfaces to common collection structures
4. Implement interfaces to common collection structures
5. Create an application that utilises data structures and collection structures

Credit value: 3 HN Credits at SCQF level 8: (24 SCQF credit points at SCQF level 8)

SCQF (the Scottish Credit and Qualifications Framework) brings Scottish qualifications into a single framework of 12 levels ranging from SQA Access 1 to doctorates. The SCQF includes degrees; HNC/Ds; SQA National Qualifications; and SVQs. Each SQA Unit is allocated a number of SCQF credit points at a specific level. 1 SCQF point = 10 hours of learning. HN candidates are normally expected to input a further number of hours, matched to the credit value of the Unit, of non-contact time or candidate-led effort to consolidate and reinforce learning.

Recommended prior knowledge and skills: Access to this Unit will be at the discretion of the Centre, however it is recommended that candidates should have prior experience of appropriate high-level languages and systems development. This may be demonstrated by possession of HN Units such as HN Unit D76X 35: *Software Development: Procedural Programming*, HN Unit D76R 35: *Software Development: Event Driven Programming*, or HN Unit D76V35: *Software Development: Object Oriented Programming*. Alternatively, candidates may have considerable practical work experience and some appreciation of the role of algorithm design, algorithm implementation, program design and program implementation.

Core skills: There may be opportunities to gather evidence towards core skills in this Unit, although there is no automatic certification of core skills or core skills components.

Higher National Unit specification: General information for centres (cont)

Context for delivery: If this Unit is delivered as part of a group award, it is recommended that it should be taught and assessed within the subject area of the group award to which it contributes.

Assessment: Part of Outcome 1 is assessed by a short report that should be given as a research exercise. The remainder of Outcome 1 is assessed by short-response questions under supervised conditions which may also include the assessment requirements for Outcome 2. The remainder of the assessment workload, (ie Outcomes 3, 4 and 5) should be carried out using a mixture of home-based research and observation/oral questioning.

An Assessment Exemplar has been produced to indicate the national standard of achievement required at SCQF level 8.

Higher National Unit specification: statement of standards

Unit title: Software Development: Abstract Data Structures

Unit code: D76L 35

The sections of the Unit stating the Outcomes, knowledge and/or skills, and evidence requirements are mandatory.

Where evidence for Outcomes is assessed on a sample basis, the whole of the content listed in the knowledge and/or skills section must be taught and available for assessment. Candidates should not know in advance the items on which they will be assessed and different items should be sampled on each assessment occasion.

Outcome 1

Investigate data representation and storage in computer systems

Knowledge and/or skills

- Representation of single-valued data types and structured data types
- Representation of complex data types stored externally
- File storage issues relating to data representation

Evidence requirements

Candidates are required to provide evidence in the form of responses to a representative range of questions which sample the following.

At least three from: character, integer, floating point, Boolean, per pixel colour data, date information

At least two from: string, record, 1-dimensional array, 2-dimensional array

This first section of evidence must be supplied within a closed-book environment under supervised conditions. There must be a minimum of three questions per item.

A minimum of 60% correct responses is required.

In addition, a report of between 500 and 800 words is required describing the representation and storage of at least one of the following complex data types: image, sound, document. This report must identify at least three different representations of content, commenting on storage requirements, file standards, quality (where applicable) and application areas.

Candidates must produce this section of evidence as a home exercise, relying on their own research.

Higher National Unit specification: statement of standards (cont)

Assessment guidelines

Questions may be short-response questions (eg multiple choice; one or two sentences).

Examples of short-response questions asked can range from 'What is the range of numbers in a 16-bit, signed integer?' through to 'What data type is best used to represent … ? Questions may also include items such as 'State two advantages of using <x> rather than <y> and, for the simple structures. 'What is the memory address of the <x th> character in a Unicode string beginning at 00A0h?' or 'How many bytes of storage are required for an <x> by <y> table containing ASCII character data?'

The emphasis here is on the storage and use of simple data types and structures, providing the underpinning knowledge to make informed decisions throughout the remainder of the Unit.

Outcome 2

Utilise searching and sorting techniques in the manipulation of a collection of data items

Knowledge and/or skills

- Determining a collation sequence for structured data
- Using common sorting techniques to order data in a collection
- Using common searching techniques to add/retrieve/delete data from a collection

Evidence requirements

There are two discrete types of evidence required, one for searching and one for sorting. Within each of these, the candidate must define and utilise at least one simple and one complex collation from the following:

Simple: numeric, lexical, character set

Complex: multiple-attribute, hash code

Describe, using an example of at least eight items, the sequence of actions leading from an unsorted to a sorted collection within a list. The example must be executed twice using a shell sort and an insertion sort. The description must include details on the number of comparisons and data movements made to achieve the sorted collection.

Describe, using an example of at least eight items, the retrieval and deletion of two items from a sorted list. The example must be executed twice, comparing a linear search and a binary search.

Assessment guidelines

Each of the collation requirements can be used in different parts of the assessment. For example, a numeric sequence created for the sort, a hash code for the search. An example of

a multiple-attribute comparison would be where personal data was to be sorted by ‘sex’ and ‘age’.

Higher National Unit specification: statement of standards (cont)

The assessment can define the specific algorithms used for the sort methods. There are too many variations on these techniques to require the candidate to provide one or other during the assessment.

The reason for choosing the shell and insert sorts is to highlight the differences between a collection that is already ‘in place’ and a collection that grows by addition. This difference must be made clear to candidates. There is no necessity, therefore, to compare the results obtained.

Outcome 3

Define interfaces to common collection structures

Knowledge and/or skills

- Access restrictions and node visibility
- Search and traversal algorithms
- Add/delete algorithms
- Recursion techniques

Evidence requirements

Definitions must be in the form of pre- and post-conditions along with Structured English algorithms. The definitions must include features that allow the size of the collection to be restricted, if desired, by user requirements.

This must be accompanied by a brief description of approximately 50 words identifying access restrictions and/or node visibility.

The following structures must be sampled:

- stack/queue/DEQ
- ordered list/unordered list
- set
- binary tree/B-Tree
- graph

Four structures must be defined. At least three of these must be significantly different as identified in the list above. For example, if a queue and DEQ are assessed, it is not possible to include a stack or both list types in the same assessment. One of the list types is permitted in this situation.

Minor errors and omissions are permitted. However, significant logical errors are not.

Higher National Unit specification: statement of standards (cont)

Assessment guidelines

Specification and use of recursion-based algorithms is strongly encouraged, where applicable.

In order to minimise the assessment workload, Centres are permitted to use completion exercises for up to three of the structures. Therefore, at least one structure must be fully defined by the candidate. Completion exercises must include at least four algorithms and at least six pre- post-conditions.

The collection structure must be designed to hold any data type. Where comparison is necessary and/or ordering is required, the inclusion of a user-defined 'Compare' function can be assumed. In all cases, the existence of a 'createNode' process to hold the data type can be assumed.

As an example, a stack would be expected to provide the following interface processes: 'create' 'push', 'pop', isEmpty' and, depending on user requirements, 'peek', 'contains' and 'empty'/'destroy'. A set would utilise 'create', 'contains', 'union', 'intersection', 'difference', 'add', 'delete', 'isEmpty'.

There should be no reference to 'IsFull' functions in the public interface definitions of an abstract data structure. Of course, there will be private functions required in an implementation of these structures.

As far as possible, algorithims should be defined in terms of other interface processes. For example, the procedure to 'empty' a stack would be implemented as "While not isEmpty do pop".

Algorithms should be independent of potential implementation issues.

Outcome 4

Implement interfaces to common collection structures

Knowledge and/or skills

- Implementing interface designs using static storage allocation
- Implementing interface designs using dynamic storage allocation

Evidence requirements

Candidates must provide fully documented, working code – including test harnesses – for two different collection types. Here, 'different' has the same meaning as defined in Outcome 3.

Each collection type must be implemented twice, using static and dynamic allocation respectively. The same test harness must be applied to both implementations of the structure.

Higher National Unit specification: statement of standards (cont)

Assessment guidelines

Candidates are not required to create applications that utilise the implemented structures. The assessment is testing their knowledge and understanding of collections and memory allocation concepts.

Therefore, the test harnesses should be simple but comprehensive. For example, the collection structure should be: created/initialised; displayed; elements added; displayed; elements added till bounding conditions are met ('false' bounds should be used for testing dynamic allocation); displayed; elements deleted; displayed; elements deleted until empty; displayed.

Coding must not be plagiarised and assessors should use observation and/or oral questioning to ensure that the candidates understand their own programs.

This may be a 'short' Outcome as far as the description goes but this section of the Unit is likely to incur the greatest overhead in terms of delivery and learning time.

It is strongly recommended that a standard 3GL such as Pascal or C be used to present this section of the Unit.

Outcome 5

Create an application that utilises data structures and collection structures

Knowledge and/or skills

- Defining the base data structure to represent a user requirement
- Determining/defining the programmatic interface for the structure
- Determining/defining the programmatic interface for a collection structure that meets user requirements
- Implementing the interfaces and providing a front-end test suite or completed application

Evidence requirements

Full design documentation is required as defined by in-house styles. The application need not be complete but sufficient coding and testing must be done to provide evidence, along with the design documentation, that the project would be completed according to user requirements if time permitted.

The base structure must be non-trivial.

Higher National Unit specification: statement of standards (cont)

Assessment guidelines

The collection structure may be part of the base structure or external to it. The support notes give examples of non-trivial structures.

There is no requirement to design and code the collection structure(s) from first principles. This has already been done for Outcome 4. Therefore, if the host language already contains collection structures, it is permitted to make use of these. In this case, copies of the relevant API may be included in the documentation (within copyright restrictions).

There is also no requirement to utilise the same host language throughout the Unit; therefore, this assessment can easily be integrated with the requirements of other Units within the framework being studied by the candidate. This also allows the concepts introduced in Outcome 3 and Outcome 4 to be implemented in a third-generation language such as Pascal or C whilst this project-based Outcome may be implemented using an Object-based language such as Java. Indeed, it is strongly recommended that the application project be done within a visual environment rather than a purely text-based environment.

Administrative Information

Unit code: D76L 35

Unit title: Software Development: Abstract Data Structures

Superclass category: CB

Date of publication: June 2001

Source: SQA

Additional copies of this Unit specification can be purchased from the Scottish Qualifications Authority. The cost for each Unit specification is £2.50 (minimum order £5.00). Additional copies of the Assessment Exemplar can also be purchased from the Scottish Qualifications Authority. The cost is £15.00.

Higher National Unit specification: support notes

Unit title: Software Development: Abstract Data Structures

This part of the Unit specification is offered as guidance. The support notes are not mandatory.

While the exact time allocated to this Unit is at the discretion of the Centre, the notional design length is 120 hours.

Guidance on the content and context for this Unit

This Unit is within the SQA frameworks for HNDs in Computing at SCQF level 8. It may be used as a stand-alone Unit within other frameworks but should be restricted to courses at SCQF level 8 or equivalent within the disciplines of Computer Science or Information Systems.

The primary purpose of this Unit is to introduce the candidate to the 'classic' data structures of computer programming with emphasis on the structurers that allow 'collections' of items to be manipulated (no Java-centric reference intended) such as lists, the stack, queues, trees and graphs. For this reason, all but the first Outcome of the Unit is biased towards the identification, definition, construction and manipulation of collections. However, that first Outcome is very diverse and is possibly the only place in the HN frameworks where candidates are required to investigate binary data file structures. As such, its importance cannot be over-emphasised.

Outcome 1

The first Outcome provides an opportunity to investigate the 'basic' data types and structures that will be contained in and manipulated by these collection structures such as the atomic data types (integer, float and char), strings, records, and so on.

As far as possible, provide background on:

- different sizes of integer and their limitations (especially porting across operating systems),
- character data (8-bit, 16-bit and language-specific directionality,
- 'endian' issues (OS again), etc.

For the representation of Date information, it would be useful to introduce, for example, long int, record structure, etc and some discussion on it, eg Y2K, overflow, etc.

In addition, the 'large' data types more often associated with file storage are to be investigated, such as graphic and document files. This allows the Centre to review and enhance the candidate's knowledge regarding storage requirements, file types (by extension, MIME type and 'magic number', ie file header information) and, most importantly, internal structure of data.

Higher National Unit specification: support notes (cont)

Because of the above, this is also an opportunity to introduce structure not specifically covered within the Unit but still relevant to the candidate in their understanding of computer systems and software. Examples here might include:

- mark-up files (such as XML and associated style sheets)
- spreadsheets (ie bi-directional, quadruple-linked lists representing a sparse matrix of 'cells', where a cell is itself a 'union'-type structure)
- executable files (from diverse systems, such as EXE, Class, elf, COFF)
- audio/video/graphic standards (eg 'lossless' vs. compressed image formats such as PNG and jpg)

Above all, this Outcome should allow the candidate to become familiar with at least some of the huge number of diverse structures available to represent different types of data in binary form. It will also be an opportunity to introduce the RFC system of international standards definition as well as proprietary standards.

The assessment requirements for this Outcome are deliberately 'light' to allow lecturers to cover a wide range of material without the need to assess everything. The research-based evidence allows candidates to choose file types from their preferred vocational areas for further study whilst still getting a broader appreciation from classroom delivery/course notes on the diversity of data representation standards.

Outcome 2

This section is concerned with the concepts and techniques of searching and sorting. The major topics here are:

- determine/define a collation sequence with emphasis on character and numeric comparison. (including Unicode, hash algorithms)
- sorting techniques (eg shuffle, radix, quicksort) and timing considerations
- searching concepts (including ordered and un-ordered data sets, timing) and algorithms

The collation sequence part is to highlight the notion that multiple attributes through to whole records, rather than only single attributes, could be used to determine which item comes 'before' or 'after' in the sequence.

The 'searching' section allows a first introduction to lists, trees and, possibly, buckets or other collection-based structures without the extra 'baggage' of a full knowledge of access methods, structural restrictions and the like. However, centres should deliver at least two variations on each of the identified sort methods, ie all data already available in the collection (eg bubble, shell, comb, quicksort) and new data inserted into an ordered collection (eg a list, binary tree, bucket). The use of formative exemplars requiring some research by candidates is strongly encouraged here.

Higher National Unit specification: support notes (cont)

A number of different hash code algorithms should be introduced and compared.

Outcome 3

This section deals with the 'classic' collection structures. The treatment of the structures is at a high level rather than looking at the implementation details and is, therefore concerned with:

- bounded and un-bounded requirements as defined by specific needs of a project/user
- access restrictions/node visibility (eg stack, (DE)Q)
- traversal algorithms (ie search)/retrieve as permitted by the structure). For example, pre-, post-, in order tree traversal; travelling salesman problem
- add/delete operations and algorithms (eg Push/Pop and equivalents; set union/difference; tree balancing)

Definitions should be in the form of pre- and post-conditions along with Structured English. Specification and use of recursion-based algorithms is strongly encouraged and may be required.

Outcome 4

This is the implementation exercise. Here, at least two *different* structures should be coded using both static and dynamic implementation. Since the interface to the structure remains unchanged, the same test programs can/must be run on both versions.

This is the section that is likely to introduce dynamic memory allocation to most candidates. There are issues regarding the host language used here and it is proposed that standard 3GL languages are used to introduce the concepts. Suitable languages are Pascal And C. Object-based extensions to these languages and other Object-based languages such as Java will have their own subsystems for handling memory that may introduce additional complications or complexities in the delivery of this section.

The initial construction and testing is to be done in isolation but there should be a holistic approach to evidence gathering across the Unit and that is where the final Outcome (mini-projects) is used. It will also be necessary to implement 'comparison' and 'display' procedures for the chosen data type outside of the collection structure interface.

There may also be an opportunity here to consider the visual representation of the structure. For example, hierarchical data might be represented as a table or a tree, with the option to allow the user to switch views. (Maybe getting a bit Java-centric again since this is effectively trivial in Java). For assessment purposes, assuming a non-visual language is being used, simple text output with minimal formatting is permitted to describe the contents of the structure.

Higher National Unit specification: support notes (cont)

Outcome 5

This will be one or two mini-projects or, where integrated across Units, part of a major project where the candidate is required to:

- define the base structure to represent a user requirement (which may, in itself contain a collection structure or other nested base structures, eg date-of-birth field; set of Units achieved by a candidate in an academic institute)
- determine/define the api for the structure
- determine/define the api for a collection structure that meets user requirements
- implement the api's and provide a (visual) front-end test suite or completed application

An example project could be a game (representation of the playing area/game board, playing pieces and attributes, etc) with associated game tree for 'look-ahead' (algorithms for determining a 'score' at a particular point in time; depth-first/breadth-first look-ahead, etc).

Other examples might include an OS simulator (stacks, queues, lists), spreadsheet or other multi-linked node system, etc.

Note: On the whole, the first four Outcomes provide the basic skills and it would make great sense to see pre-defined objects or structures, available in the host language, used during the project. Although it is intellectually stimulating to create implementations of structures from first principles it is unnecessary to enforce this for more than a couple of classic structures as done in Outcome 4. It would be more useful to allow the candidate to see the fruits of their labours within a complete application.

Guidance on the delivery and assessment of this Unit

For classroom delivery, the following is a guideline only.

Outcome 1: 8-10 hours

Outcome 2: 20-25 hours

Outcome 3: 20-25 hours

Outcome 4: 30-35 hours

Outcome 5: 25-35 hours

Guidance on delivery and assessment is sufficiently provided for in the guidance section on Content and Context above.

Open learning

If this Unit is delivered by open or distance learning methods, additional planning and resources may be required for candidate support, assessment and quality assurance.
A combination of new and traditional authentication tools may have to be devised for assessment and re-assessment purposes. For further information and advice, please see *Assessment and Quality Assurance fort Open and Distance Learning* (SQA, February 2001 – publication code A1030).

Special needs

Oral questioning and the use of scribes may be used for candidates with difficulties in reading or writing.

This Unit specification is intended to ensure that there are no artificial barriers to learning or assessment. Special needs of individual candidates should be taken into account when planning learning experiences, selecting assessment instruments or considering special alternative Outcomes for Units. For example, some candidates may require a longer period for the single assessment or may require that it be split into more than one event. For information on these, please refer to the SQA document *Guidance on Special Assessment and Certification Arrangements for Candidates with Special Needs and Candidates for whom English is an Additional Language* (SQA, 2000).

General information for candidates

Unit title: Software Development: Abstract Data Structures

This Unit is designed to enable you to become familiar with the data structures and collection structures in common use within current software systems. This knowledge will be supplemented by research into and analysis, design and coding of collection structures in order to create applications to meet complex user requirements.

The collection structures include stacks, queues, trees, graphs and sets. You will also learn about the searching and sorting techniques required to utilise these structures to best effect.

You will become familiar with a wide variety of formats used to store data on a file system such as audio, graphics and video, sound and common document formats.

This Unit is suitable for those of you who wish to pursue a career in Software Engineering or who construct applications requiring the use of non-trivial data structures.

On completion of the Unit you will be able to:

1. Investigate data representation and storage in computer systems
2. Utilise searching and sorting techniques in the manipulation of a collection of data items
3. Define interfaces to common collection structures
4. Implement interfaces to common collection structures
5. Create an application that utilises data structures and collection structures

With the exception of a short test under supervised conditions, the majority of the assessment workload will be carried out at home or in tutor-supported class time.