aie
SPECIALIST EDUCATORS IN
GAMES, ANIMATION & FILM VFX

2021 Subject & Assessment Guide

Code Design and Data Structures

ICT50120 Diploma of Information Technology (Game Programming)

CUA51015 Diploma of Screen and Media

academy of interactive entertainment
Sydney Melbourne Canberra Adelaide Lafayette Seattle Online

Table of Contents

Code Design and Data Structures

Units of Competency

The units of competency that are covered in this subject are as follows:

ICTPRG547 - Apply advanced programming skills in another language

Assessment processes and competency evidence requirements are described in the *Assessment Criteria* section below. If you have prior or other evidence against competency you should discuss this with your teacher.

Subject Overview

Overall Learning Outcomes

- Apply a knowledge of object-oriented design and modelling
- Demonstrate an understanding of common data structures used in game programming
- Demonstrate an understanding of fundamental programming algorithms
- Apply skills in testing and error handling in game programming

Subject Description

This subject is designed to teach you about the design techniques that underpin modern programming, particularly object-oriented programming (OOP), and techniques commonly used in games programming today.

OOP grew to maturity in the early 1990's and quickly revolutionised how programs were designed and developed. Before that time, the prevailing software design technique was *functional decomposition* (also known as procedural programming) which was all about writing *functions* that implemented the *behaviour* of a program. It was soon realised that this technique was unsuitable for very large software systems and that a better approach was needed.

The new approach became known as *object-orientation*, a term used to describe both the new design methodology and the coding techniques used to implement object-oriented designs in an actual programming language. C++ was developed as an extension of the earlier C language with the express purpose of implementing the new object-oriented principles and techniques.

This subject looks at the design and implementation of common *algorithms* (such as sorting and searching), *data structures* (used to store data efficiently for processing), and systems (including Game State management and Component patterns) used in many games and game engines.

Industry Relevance

Object-oriented design is now the primary design methodology used by almost all modern programming languages in commerce and industry. OOP is vital in the quest to write ever larger and more complex software systems. Skills in OOP design are highly valued and sought after throughout the software industry, including the games industry. Employers are no longer looking only for coders; they are looking for designers / software engineers, i.e. programmers who know how to develop a design and then implement it in effective and maintainable code.

This importance extends even further to the lower-level data structures examined in this course. Once you are actually implementing a software design, it is vitally important to then know which algorithms and data structures to use and when to use them. This translates to understanding of how algorithms and data structures work and their relative efficiencies. A brilliant design is only as good as its actual implementation.

Assumed Knowledge

- An understanding of foundational programming concepts and knowledge of C++ at an introductory level

Subject Textbooks

The following textbooks are highly recommended for this subject:

- Prata, S, ***C++ Primer Plus***, 6th Edition, Addison Wesley (2011)
- Nystrom, R, ***Game Programming Patterns***, 1st Edition, Genever Benning (2014)
    - http://gameprogrammingpatterns.com/

You may also find the following textbooks useful:

- Sherrod, A, ***Data Structures and Algorithms for Game Developers***, 1st Edition, Charles River Media (2007)
- Dalmau, D, ***Core Techniques and Algorithms in Game Programming***, 1st Edition, New Riders Games (2003)
- Sedgewick, R, Wayne, K, ***Algorithms, 4th Edition***, Addison-Wesley Professional (2011), https://algs4.cs.princeton.edu/home/

Assessment Criteria

Assessment Description

Assessment Milestones

Please refer to your Class Schedule for actual dates on your campus

General Description

This assessment requires you to demonstrate your knowledge of various complex data structures and algorithms used within games and simulation industries by implementing and testing each technique.

You are required to complete four assessable exercises throughout this subject in which you will implement the data structures and algorithms listed in tasks 1 through 4 of the table titled *Assessment Tasks and Evidence Descriptions* below. All exercises must be completed as C++ programs.

Descriptions of these assessible exercises can be found in *Appendix 1* to *Appendix* 4, or on the Canvas page for this subject.

In addition to these exercises, you will be designing and writing test cases to debug and validate the correctness of the data structures and algorithms you write.

Evidence Specifications

This is the specific evidence you must prepare for and present by your assessment milestone to demonstrate you have competency in the above knowledge and skills. The evidence must conform to all the specific requirements listed in the table below. You may present additional, or other evidence of competency, but this should be as a result of individual negotiation with your teacher.

Your Roles and Responsibilities as a Candidate

- Understand and feel comfortable with the assessment process.
- Know what evidence you must provide to demonstrate competency.
- Take an active part in the assessment process.
- Collect all competency evidence for presentation when required.

This table defines the individual requirements for each part of the assessment criteria. Listed here are the cumulative requirements for all assessment items. The evidence requirements for specific assessment items can be seen by referring to the table listed for that assessment item in the following sections.

Assessment and Competency Requirements
1. Implement a Double-Linked List Evidence that includes: • Successful creation of a project which implements and demonstrates a double-linked list

○ The created program must demonstrate a custom sorting algorithm • Project submitted as an executable binary file that can be run external to an IDE • Source code and project files also submitted for review
2. Perform Unit Testing and Debugging Evidence that includes: • Design and creation of tests using a testing framework • Use of tests to debug and verity the implementation of one or more data structures • Test report documenting the tests conducted and the test results
3. Implement a Binary Tree Evidence that includes: • Successful creation of a graphical application which implements and demonstrates a binary tree ○ The created program must maintain an ordered tree ○ The created program demonstrates insertion and removal of nodes ○ The created program allows the user to search for a value in the tree ○ The created program uses a third-party library • Project submitted as an executable binary file that can be run external to an IDE • Source code and project files also submitted for review
4. Implement a Hash Function Evidence that includes: • Successful creation of a project which implements and demonstrates hashing techniques • Creation of a custom hashing function • Project submitted as an executable binary file that can be run external to an IDE • Source code and project files also submitted for review
5. Inter-Process Communication Exercise Evidence that includes: • Completion of the inter-process communication exercise • Project submitted as an executable binary file that can be run external to an IDE • Source code and project files also submitted for review
6. Version Control Used Evidence that includes: • A version control repository created and filled with project source code, project files and resources. Version Control requirements include: ○ Creation of a version control repository for project(s) before development begins ○ Observation of successful use of the repository throughout development ○ Submitted screenshot or commit logs showing commits made throughout development of project(s), demonstrating consistent use of the Version Control software chosen
7. Application Handover Evidence that includes: • Visual Studio solutions and projects that compile without errors ○ All temporary and built executable files in the obj and bin folder have been removed

- A “readme” or client document explaining how to compile, run and operate the program, for each application made
- All submitted material archived in a single compressed file (zip, rar, or 7z)

Assessment Instructions for Candidate

METHOD OF ASSESSMENT

Assessment is a cumulative process which takes place throughout a subject. A ‘competent’ or ‘not yet competent’ decision is generally made at the end of a subject. Your assessment will be conducted by an official AIE qualified assessor. This may be someone other than your teacher. The evidence you must prepare and present is described.

above in this assessment criteria document. This evidence has been mapped to the units of competency listed at the beginning of this document. Assessments will be conducted on a specific milestone recorded above in this assessment guide document.

ASSESSMENT CONDITIONS

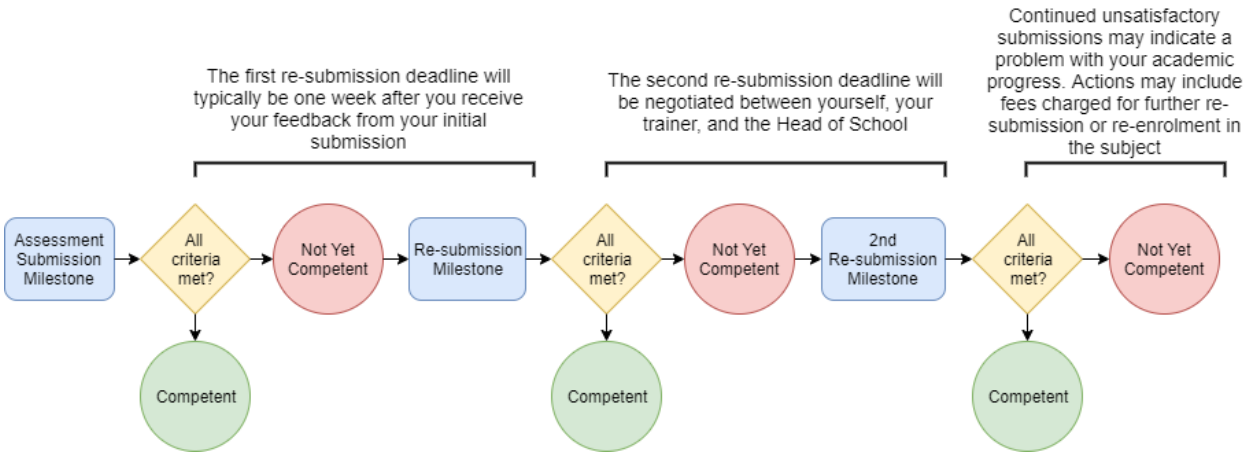
Formative assessment takes place as your teacher observes the development of your work throughout the subject and, although the assessor is likely to be aware of the evidence you are submitting, it is your responsibility to be prepared for the interview where a competency judgement is made (summative assessment). Forgetting something, or making a small mistake at the time of the milestone assessment, can be corrected. However, the assessor may choose to assess other candidates who are better prepared and return to you if time permits.

Upon completion of the assessment you will be issued with feedback and a record of the summative assessment and acknowledge that you have received the result. If you are absent for the nominated assessment milestone (without prior agreement or a sufficiently documented reason) you will be assessed as not yet competent.

GRADING

The assessment you are undertaking will be graded as either *competent* or *not yet competent*.

REASSESSMENT PROCESS



If you are assessed as being not yet competent you will receive clear, written and oral feedback on what you will need to do to achieve competence. Failing to submit an assessment will result in you being assessed as not yet competent. You will be given a reassessment milestone no more than one (1) week later to prepare your evidence. If you are unsuccessful after your reassessment, you may be asked to attend a meeting with your Head of School to discuss your progress or any support you may need and further opportunities to gain competency.

REASONABLE ADJUSTMENTS

We recognise the need to make reasonable adjustments within our assessment and learning environments to meet your individual needs. If you need to speak confidentially to someone about your individual needs, please contact your teacher.

FURTHER INFORMATION

For further information about assessment and support at AIE, please refer to the assessment and course progress sections of your student handbook.

Software

Core

Microsoft Visual Studio

Microsoft's Visual Studio is the recommended IDE for this subject. Other IDEs may be employed if desired as the content of this subject is designed to be cross-platform and IDE agnostic, however we cannot guarantee that all subject material will operate as intended on other IDEs and platforms.

- https://www.visualstudio.com/

GitKraken

GitKraken is a leading Git GUI client for Windows, Mac and Linux, used to create and maintain version control repositories. It helps developers become more productive with Git, and provides an integrated conflict editor, built-in code editor and task tracking. Other version control clients may be employed if desired.

- https://www.gitkraken.com/

7zip

7-Zip is a free and open-source file archiver, a utility used to place groups of files within compressed containers known as "archives". This utility program will be necessary to package your assessment files for submission.

- https://www.7-zip.org/download.html

Suggested

TortoiseGit

TortoiseGit is a Git revision control client, implemented as a Windows shell extension, allowing you to see the status of your Git repository and perform basic Git commands directly in your Windows file system.

- https://tortoisegit.org/

WinMerge

WinMerge is a free software tool for data comparison and merging of text-like files. It is useful for determining what has changed between versions, and then merging changes between versions. It is an essential tool when working with version control systems to resolve conflicts.

- https://winmerge.org/?lang=en

Appendix 1

Assessable Exercise 1: Double-Linked List

Overview:

These exercises form part of your assessment for this subject.

For this exercise you must create a double-linked list class and write a program that demonstrates its use. You must also write a sorting algorithm that will sort your list.

Upon completion of this exercises, upload your completed program (including source code, solution and project files, and executable build) to the assessment submission section on Canvas (https://aie.instructure.com).

Requirements:

You are tasked with creating a Double-Linked List class. The class must support the following operations:

- Inserting and deleting a node at the front of the list
- Inserting and deleting a node at the end of the list
- Inserting and deleting a node at an arbitrary location in the list
- Returning a count of how many nodes are in the list
- Checking if the list is empty
- Returning the first or last node in the list
- Sorting the list

Create a small application to test your linked list class. Your test application need not be a game, but it must allow the user to verify that the double-lined-list class works without inspecting the code (an application containing a graphic user interface, created using a third-partly library like RayLib is recommended).

You may reference this page when writing code to test for memory leaks:
https://msdn.microsoft.com/en-us/library/x98tx3cf.aspx

Submission

You will need to submit the following:

- A Release build of your application that can execute as a stand-alone program
- Your complete Visual Studio project
    - Be sure to remove any temporary build folders (i.e., the Debug and Release folders). Only project files, source code files, and any resource files used should be included in your submission.

Package all files in a single compressed archive file (.zip, .7z, or .rar)

Submission Checklist

This submission checklist is used to assist your assessor in marking your assessment.

A copy of this checklist can be downloaded from https://aie.instructure.com/. You must complete this checklist yourself and submit it with your project.

General

Description	**Y/N**
The project compiles without errors *Programs that don't compile cannot be assessed*	
The program includes a "readme" or document explaining how to compile, execute and operate the program	
The program performs as described in the general description	
The program contains no logical errors	
The code is sufficiently commented and clean	
An attempt has been made to increase the program's efficiency	
Code compiles without no warnings	
Program executes without crashing	
Program has no memory leaks	
A release executable has been made and included in the submission	
Project files and source code are included in the submission	
All files are packaged in a single compressed archive	

Estimate the number of hours taken to complete this assessment task	
How many times have you submitted this assessment task (including this time)?	

Required Features

Complete the following table by providing the class name or file name, along with the line number, to show where you have implemented each feature.

Feature	**Class/File**	**Line Number**
The program implements a double-linked list		
The linked-list class includes functions for inserting a node, as described in the *requirements* section		
The linked-list class includes functions for deleting a node, as described in the *requirements* section		
The linked-list class includes functions for returning the starting and ending node, as described in the *requirements* section		
The linked-list class includes functions for counting the number of nodes, and determining if the list is empty, as described in the *requirements* section		
The program sorts the linked list		
The program demonstrates iterating through (stepping though) the linked list		

Feature	**Y/N**
The application demonstrates the correct functioning of the linked list, ideally using a GUI	

Appendix 2

Assessable Exercise 2: Testing and Debugging

Overview:

This exercises forms part of your assessment for this subject.

For this exercise you must create test cases using a testing framework for one or more of the data structures you have written in previous exercises.

Upon completion of this exercises, upload your completed program (including source code, solution and project files, and executable build) to the assessment submission section on Canvas (https://aie.instructure.com).

Requirements:

For one or more data structures or algorithms you have created in this subject (for example, the double-linked list), design and write several test cases to verify the correctness of your implementation

You are free to use any testing framework you like, but the *Test Explorer* inside *Visual Studio* is recommended.

For more information on using *Test Explorer*, you can consult the Microsoft documentation https://docs.microsoft.com/en-us/visualstudio/test/writing-unit-tests-for-c-cpp?view=vs-2019

Upon completion of your testing, create a document that describes the tests performed, and the test results.

Submission

You will need to submit the following:

- A Visual Studio solution containing your test cases, and data structure or algorithm being tested. Ensure that your assessor can easily re-run your test cases when assessing your submission.
    - Be sure to remove any temporary build folders (i.e., the Debug and Release folders). Only project files, source code files, and any resource files used should be included in your submission.
- A document, in MS Word or PDF format, that describes your test cases and the test results.

Package all files in a single compressed archive file (.zip, .7z, or .rar)

Submission Checklist

This submission checklist is used to assist your assessor in marking your assessment.
A copy of this checklist can be downloaded from https://aie.instructure.com/. You must complete this checklist yourself and submit it with your project.

General

Description	Y/N
All projects compile without errors *Programs that don't compile cannot be assessed*	
If *Visual Studio Text Explorer* has not been used, the program includes a "readme" or document explaining how to compile and execute the test cases	
The project performs as described in the general description	
The project contains no logical errors	
The code is sufficiently commented and clean	
At least one (but preferably several) test cases have been written, and return successful results	
Code compiles without no warnings	
Test cases execute without crashing	
Project files and source code are included in the submission	
All files are packaged in a single compressed archive	

Estimate the number of hours taken to complete this assessment task	
How many times have you submitted this assessment task (including this time)?	

Appendix 3

Assessable Exercise 3: Binary Tree

Overview:

These exercises form part of your assessment for this subject.

For this exercise you must implement a binary tree and write a program that demonstrates its use. You must also write a sorting algorithm that will sort your list.

Upon completion of this exercises, upload your completed program (including source code, solution and project files, and executable build) to the assessment submission section on Canvas (https://aie.instructure.com).

Create a Binary Tree Class:

The tutorial for the session on *Binary Trees* walks through the creation of a Linked Binary Tree class (in which pointers are used to store the left and right branches).

For this assessable task you may either complete the Linked Binary Tree class that you started in this tutorial, or create an Arrayed Binary Tree (which is also briefly mentioned in this tutorial).

Your Binary Tree class must include functions for inserting a new value, finding a value, and removing a value from the tree.

The Binary Tree class should be contained in its own source files (i.e., it is not appropriate to write your entire program within *main.cpp*)

You must also create a program to visualize and test your binary tree. The program must allow the user to insert and remove values from the tree, and display all values in the tree. Your program must display a graphical user interface and must be implemented using a third-party library, such as the AIEBootstrap or RayLib.

You are free to incorporate other elements or add additional features as you see fit.

Requirements:

You are tasked with creating a Binary Tree. The Binary Tree must support the following operations:

- The binary tree must be ordered, and maintain ordering as nodes are inserted and removed
- Inserting a node into the binary tree
- Removing a node from the binary tree
- Searching for a value

Create a small application to test and demonstrate your Binary Tree implementation.

Your test application need not be a game, but it must allow the user to verify that the binary tree works without inspecting the code. It must be a graphical application, not a command-line only application.

You must implement a graphic user interface, created using a third-partly library such as the AIEBootstrap or RayLib.

You may reference this page when writing code to test for memory leaks: https://msdn.microsoft.com/en-us/library/x98tx3cf.aspx

Submission

You will need to submit the following:

- A Release build of your application that can execute as a stand-alone program
- Your complete Visual Studio project
    - Be sure to remove any temporary build folders (i.e., the Debug and Release folders). Only project files, source code files, and any resource files used should be included in your submission.

Package all files in a single compressed archive file (.zip, .7z, or .rar)

Submission Checklist

This submission checklist is used to assist your assessor in marking your assessment.
A copy of this checklist can be downloaded from https://aie.instructure.com/. You must complete this checklist yourself and submit it with your project.

General

Description	**Y/N**
The project compiles without errors *Programs that don't compile cannot be assessed*	
The program includes a "readme" or document explaining how to compile, execute and operate the program	
The program performs as described in the general description	
The program contains no logical errors	
The code is sufficiently commented and clean	
An attempt has been made to increase the program's efficiency	
Code compiles without no warnings	
Program executes without crashing	
Program has no memory leaks	
A release executable has been made and included in the submission	
Project files and source code are included in the submission	
All files are packaged in a single compressed archive	

Estimate the number of hours taken to complete this assessment task	
How many times have you submitted this assessment task (including this time)?	

Required Features

Complete the following table by providing the class name or file name, along with the line number, to show where you have implemented each feature.

Feature	**Class/File**	**Line Number**
The program implements a binary tree		

Inserting a node into the tree		
Deleting a node from the tree		
Searching for a value in the tree		

Feature	**Y/N**
The application demonstrates the correct functioning of the binary tree	
The program displays a graphic user interface (GUI) for inserting/removing nodes	
A third-party library is used in the creation of the project	

Appendix 4

Assessable Exercise 4: Hashing

Overview:

This exercises forms part of your assessment for this subject.

For this exercise you must create a hash table, plus a small application to test your implementation.

Upon completion of this exercises, upload your completed program(s) (including source code, solution and project files, and executable build) to the assessment submission section on Canvas (https://aie.instructure.com).

Create a Hash Function:

Implement your own HashFunction namespace that contains Hash Functions.

Use the following as a starting point:

HashFunction.h:

```
#include <functional>

namespace HashFunction {

    typedef std::function< unsigned int(const char*, unsigned int)> HashFunc;

    // implementation of a basic addition hash
    unsigned int badHash( const char* data, unsigned int length );

    // ADD YOUR FUNCTIONS HERE

    // a helper to access a default hash function
    static HashFunc default = badHash;
}
```

HashFunction.cpp:

```
#include "HashFunction.h"

namespace HashFunction {

    // implementation of a basic addition hash
    unsigned int badHash( const char* data, unsigned int length ) {
        unsigned int hash = 0;

        for (unsigned int i = 0 ; i < length ; ++i)
            hash += data[ i ];

        return hash;
    }
```


Add a Hash Function to your namespace and set the default to your improved method rather than badHash.

To replace the default simply assign HashFunction::default to a function other than badHash.

Write a program that can store the contents of a file (say, a texture) in a hash table, where the key is a hash value derived from the file name. See if you can optimize your hash function based on the files your program will work with.

Submission

You will need to submit the following:

- A Release build of your application that can execute as a stand-alone program
- Your complete Visual Studio project
    - Be sure to remove any temporary build folders (i.e., the Debug and Release folders). Only project files, source code files, and any resource files used should be included in your submission.

Package all files in a single compressed archive file (.zip, .7z, or .rar)

Submission Checklist

This submission checklist is used to assist your assessor in marking your assessment.
A copy of this checklist can be downloaded from https://aie.instructure.com/. You must complete this checklist yourself and submit it with your project.

General

Description	**Y/N**
The project compiles without errors *Programs that don't compile cannot be assessed*	
The program includes a "readme" or document explaining how to compile, execute and operate the program	
The program performs as described in the general description	
The program contains no logical errors	
The code is sufficiently commented and clean	
An attempt has been made to increase the program's efficiency	
Code compiles without no warnings	
Program executes without crashing	
Program has no memory leaks	
A release executable has been made and included in the submission	
Project files and source code are included in the submission	
All files are packaged in a single compressed archive	

Estimate the number of hours taken to complete this assessment task	
How many times have you submitted this assessment task (including this time)?	

Required Features

Complete the following table by providing the class name or file name, along with the line number, to show where you have implemented each feature.

Feature	Class/File	Line Number
The program implements a custom hash function		
The program stores values in a hash table or hash map		

Appendix 5

Assessable Exercise 5: Inter-process Communication

Overview:

This exercises forms part of your assessment for this subject.

For this exercise you must complete the tutorial on inter-process communication. This tutorial explores the inter-process communication technique of *named shared memory*.

Named Shared Memory allows us to create a block of memory within one application and map it for use within other applications, using a string to identify the block. NSM is an Operating System specific feature and is not available on all platforms.

Upon completion of this exercises, upload your completed program(s) (including source code, solution and project files, and executable build) to the assessment submission section on Canvas (https://aie.instructure.com).

Inter-process communication:

Complete the tutorial on this page:
https://aie.instructure.com/courses/39/pages/Interprocess%20Communication

Upon completion of the tutorial, package your solution and submit it to Canvas.

Submission

You will need to submit the following:

- A Release build of your application that can execute as a stand-alone program
- Your complete Visual Studio project
    - Be sure to remove any temporary build folders (i.e., the Debug and Release folders). Only project files, source code files, and any resource files used should be included in your submission.

Package all files in a single compressed archive file (.zip, .7z, or .rar)

Submission Checklist

This submission checklist is used to assist your assessor in marking your assessment.
A copy of this checklist can be downloaded from https://aie.instructure.com/. You must complete this checklist yourself and submit it with your project.

General

Description	**Y/N**
The project compiles without errors *Programs that don't compile cannot be assessed*	
The program includes a "readme" or document explaining how to compile, execute and	

operate the program	
The program performs as described in the general description	
The program contains no logical errors	
The code is sufficiently commented and clean	
An attempt has been made to increase the program's efficiency	
Code compiles without no warnings	
Program executes without crashing	
Program has no memory leaks	
A release executable has been made and included in the submission	
Project files and source code are included in the submission	
All files are packaged in a single compressed archive	

Estimate the number of hours taken to complete this assessment task	
How many times have you submitted this assessment task (including this time)?	

