Level 6 Advanced Diploma in Programming (602)
163 Credits

Unit: Advanced C++ Programming **Exam Paper No:** 4	**Guided Learning Hours:** 260 **Number of Credits:** 26
Prerequisites: : Programming experience in C++ for at least six months.	**Corequisites:** A pass or higher in Diploma in Programming or equivalence.
Aim: Topics will focus on ANSI C++ implementation of data structures, use of the Standard Template Library and graphical programming using the classes. Learners will learn programming techniques in Standard C++ for large-scale, complex, or high-performance software. Encapsulation, automatic memory management, exceptions, generic programming with templates and function objects, standard library algorithms and containers. Using single and multiple inheritance and polymorphism for code reuse and extensibility, basic design idioms, patterns, and notation will be covered.	
Required Materials: Recommended Learning Resources.	**Supplementary Materials:** Lecture notes and tutor extra reading recommendations.
Special Requirements: This is a hands-on unit, hence practical use of computers is essential. Requires intensive lab work outside of class time.	
Intended Learning Outcomes:	**Assessment Criteria:**
1. The implementation of a class and the notion of iterator classes that walk through the elements of container classes.	1.1 Explain how to specify const (***constant***) objects and const member functions 1.2 Describe the purpose of friend functions and friend classes 1.3 Describe the use of the ***this*** pointer 1.4 Demonstrate how to create and destroy objects dynamically 1.5 Describe how to use static data members and member functions 1.6 Describe the concept of a container class 1.7 Describe the notion of iterator classes that walk through the elements of container classes.
2. Overloading and how to create array, string and date classes that demonstrate operator overloading.	2.1 Describe how to redefine (overload) operators to work with new Abstract Data Types (ADTs) 2.2 Demonstrate how to convert objects from one class to another class 2.3 Describe when to, and when not to, overload operators
3. How software re-use reduces program development time, the use of constructors and destructors in inheritance hierarchies.	3.1 Describe how to create classes by inheriting from existing classes 3.2 Describe how inheritance promotes software reusability 3.3 Describe the notions of base classes and derived classes 3.4 Describe the protected member-access modifier
4. Polymorphism, how it is implemented and polymorphism in object oriented programming.	4.1 Describe the concept of polymorphism 4.2 Describe how to declare and use virtual functions to effect polymorphism 4.3 Distinguish between abstract and concrete classes

	4.4	Demonstrate how to declare pure virtual functions to create abstract classes
	4.5	Define how polymorphism makes systems extensible and maintainable
	4.6	Describe how C++ implements virtual functions and dynamic binding "under the hood."
	4.7	Identify how to use Run-Time Type Information (RTTI) and operators **typeid** and **dynamic_cast**.
5. Analysing templates, how programmers can use them and the relationships among templates, friends, inheritance and static members.	5.1	Demonstrate how to use function templates to create a group of related (overloaded) functions
	5.2	Distinguish between function templates and function-template specialisations
	5.3	Describe how to use class templates to create a group of related types
	5.4	Distinguish between class templates and class-template specialisations
	5.5	Describe how to overload function templates
6. Input/output streams, low-level and high-level input/output capabilities and demonstrating unformatted and formatted input/output.	6.1	Describe how to use C++ object-oriented stream input/output
	6.2	Identify how to format input and output
	6.3	Describe the stream-I/O class hierarchy
	6.4	Describe how to input/output objects of programmer-defined types
	6.5	Identify how to use stream manipulators
7. Exception handling and how C++ exceptions provides a powerful mechanism for cleanly handling errors separately from the rest of the program's design.	7.1	Demonstrate **try**, **throw** and **catch** to detect, indicate and handle exceptions, respectively process uncaught and unexpected exceptions
	7.2	Describe how to handle new failures
	7.3	Demonstrate how to use **auto_ptr** to prevent memory leaks
	7.4	Describe the standard exception hierarchy.
8. Understand web programming with Common Gateway Interface (CGI).	8.1	Describe HTTP and web server functions
	8.2	Explain the process of accessing web server
	8.3	Describe CGI
	8.4	Demonstrate HTTP transaction
	8.5	Design simple CGI script
	8.6	Use XHTML forms to sent input
	8.7	Describe cookies
9. Understand implementation of abstract data type/ data structures collections; linked lists, stacks, queues and trees.	9.1	Compare and contrast fixed-size vs dynamic data structures
	9.2	Define dynamic memory allocation
	9.3	Create linked list program
	9.4	Create stack program
	9.5	Create queue program
	9.6	Demonstrate tree implementation
	10.1	Define and declare structures

10. Understand bits, character, string and structure data sets.	10.2 Be able to initialise structures 10.3 Pass structure to functions 10.4 Use **typedef** keyword 10.5 Describe bitwise/bit field operators 10.6 Implement character-handling library
Methods of Evaluation: A 3-hour essay written paper with 5 questions, each carrying 20 marks. Candidates are required to answer all questions. Candidates also undertake project/coursework in Advanced C++ Programming with a weighting of 100%.	

Recommended Learning Resources: Advanced C++ Programming

Text Books	• Advanced C++ Programming Styles and Idioms by James O. Coplien. • Algorithms in C++: Fundamentals, Data Structures, Sorting, Searching Pts. 1-4 by Robert Sedgewick. ISBN-10: 0201350882 • The C++ Programming Language by Bjarne Stroustrup. • The C++ Standard Library: A Tutorial and Reference by Nicolai M. Josuttis.
Study Manuals 	BCE produced study packs
CD ROM 	Power-point slides
Software 	C++ Programming Language