**900220-000-00-KM-02, Principles of Programming with C++, NQF Level 4, Credits 6**

**SUMMATIVE ASSESSMENT MEMO**

**Module Two (2)**

|  |  |
| --- | --- |
| **Module Code** | 900220-000-00-KM-02 |
| **NQF Level** | 4 |
| **Credits** | 6 |
| **Skills Programme ID Number** | SP- 230374 |
| **Curriculum Title** | C++ Programmer |
| **Curriculum Code** | 900220-000-00-00 |

|  |  |
| --- | --- |
| **Name** |  |
| **Contact Address** |  |
| **Telephone (H)** |  |
| **Telephone (W)** |  |
| **Facsimile** |  |
| **Cellular** |  |
| **E-mail** |  |

**Note to the learner**

This Learner Guide provides a comprehensive overview of the module. It is designed to improve the skills and knowledge of learners, and thus enabling them to effectively and efficiently complete specific tasks.

**Purpose of the Module**

The main focus of the learning in this knowledge module is to build an understanding of the principles of programming with C++ programming language

The learning will enable learners to demonstrate an understanding of:

* KM-02-KT01:Variables5%
* KM-02-KT02: C++ Strings3%
* KM-02-KT03:Operators3%
* KM-02-KT04: Conditions in C++3%
* KM-02-KT05: Switch statements in C++3%
* KM-02-KT06: Arrays in C++6%
* KM-02-KT07: Loops in C++3%
* KM-02-KT08: References and enumerations in C++3%
* KM-02-KT09: Exception handling in C++3%
* KM-02-KT10: Dynamic Arrays in C++ 3%
* KM-02-KT11: Pointers in C++ 15%
* KM-02-KT12: C++ Char data types 5%
* KM-02-KT13: File handling in C++: Basic input/output 15%
* KM-02-KT14: C++ structure (Struct)3%
* KM-02-KT15: C++ class and object 3%
* KM-02-KT16: C++ operator overloading 3%
* KM-02-KT17: Std: list in C++ 3%
* KM-02-KT18: C++ Functions 5%
* KM-02-KT19: Date/time in C++ 3%
* KM-02-KT20: Debugging in C++ 10%

**Provider Accreditation Requirements for the Knowledge Module**

**Physical Requirements:**

* The provider must have lesson plans and structured learning material or provide learners with access to structured learning material that addresses all the topics in all the knowledge modules as well as the applied knowledge in the application.
* QCTO/ MICT SETA requirements

**Human Resource Requirements:**

Lecturer/learner ratio of 1:20 (Maximum)

* Qualification of lecturer (SME): o NQF 5 qualified in industry recognised qualifications with 1-year experience in the IT industry o Cybersecurity vendor certification
* Assessors and moderators: accredited by the MICT SETA

**Legal Requirements:**

* Legal (product) licences to use the software for learning and training
* OHS compliance certificate

**Venue, Date and Time:**

Consult your facilitator should there be any changes to the venue, date and/or time.Refer to your timetable.

**Assessments**

**Integrated Formative Assessment:** The skills development provider will use the curriculum to guide them on the stipulated internal assessment criteria and weighting. They will also apply the scope of practical skills and applied knowledge as stipulated by the internal assessment criteria. This formative assessment leads to entrance into the integrated external summative assessment.

**Integrated Summative Assessment**: An external integrated summative assessment conducted through the relevant QCTO Assessment Quality Partner is required to issue this qualification. The external integrated summative assessment will focus on the exit level outcomes and associated assessment criteria.

**Skills Programme Purpose**

The need for this skills programme was identified after realising the importance and future impact of the 4IR on the economy of South Africa and its competitiveness. The Minister of Communications then gazetted the Presidential Commission on the Fourth Industrial Revolution (PC4IR) on 9 April 2019. In March 2020 this Commission delivered a report with wide ranging recommendations for Human Capital Development that will drive the 4IR forward.

This report clearly indicated the speed at which companies will have to invest in big data analysis, web-enabled market investment and the use of cloud computing and machine learning. Programming skills and being competent in the use of programming languages such as C++ Language are central to these initiatives.

The development of this C++ Programmer Skills Programme is also in support of the drivers for economic recovery as stated in the Economic Reconstruction and Recovery Plan (ERRP) and the subsequent Economic Reconstruction and Recovery Skills Strategy.

**Skills Programme Purpose**

A C++ Programmer will be able to Implement solutions to solve real life problems in an efficient manner applying a knowledge and understanding of the principles of programming with C++ and applicable tools.

Tasks that the learner will be able to know, do and understand after achievement of the skills programme include:

* Create well-written and readable C++ programs, using a disciplined coding style, including documentation and indentation standards.
* Work collaboratively in a team and execute version control

**Entry Requirements**

Grade 11 with Maths Lit and English.

Access to equipment, internet connectivity and how to work remotely

**EXIT LEVEL OUTCOMES**

**Exit Level Outcomes (ELO) 1**

Describe the basics of C++ Programming

Associated Assessment Criteria

* Fundamentals of the C++ programming language are explained.
* Basic concepts and methods of C++ object-oriented programming and object-oriented design are described.
* The development life cycle as a means of creating C++ applications is described.
* A thorough knowledge of the use of algorithms in problem solving is demonstrated.

**Exit Level Outcomes (ELO) 2**

Programme effectively using C++ frameworks and functionalities

Associated Assessment Criteria

* The use of C++syntax is demonstrated by creating neat and concise coding including application of documentation and indentation standards.
* Well-written and readable C++ programs are created, using a disciplined coding style, including comments and indentation standards.
* Procedural and object oriented concepts and syntax are applied.
* The ability to troubleshoot problems with application development is demonstrated and application is debugged.

**Exit Level Outcomes (ELO) 3**

Work collaboratively in a team using the GitHub platform

Associated Assessment Criteria

* An ability to work with Git and GitHub functionalities is demonstrated.
* The ability to work collaboratively in a team using Git is applied
* Version control is executed using Git functionalities such as repositories, branches, commits and pull requests

**Session 1:** **KM-02-KT01:Variables5%**

1. **IAC0101 State the definitions, functions and features of variables in C++**

**Concept of Variables in C++:**

In C++, a variable is a named storage location that holds a value. It's a fundamental concept in programming that allows you to store and manipulate data within a program. Variables provide a way to give names to data and refer to them throughout the program.

**Definition of Variables in C++:**

In C++, a variable is declared using a combination of a data type and a name. The syntax for declaring a variable is:

cppCopy code

data\_type variable\_name;

**Session 2:** **KM-02-KT02: C++ Strings3%**

1. **IAC0201 Explain the definitions, functions and features of strings in C++**

**1. Variables:**

* **Concept:**
  + A variable is a named storage location in a program that holds a value, which can be changed during the execution of the program.
* **Definition:**
  + Variables are used to store and manipulate data in a program. They have a data type that defines the type of data they can hold.
* **Purpose:**
  + Variables enable the dynamic storage and retrieval of data, allowing programs to work with and process information.

**Session 3:** **KM-02-KT03:Operators3%**

1. **IAC0301 Explain the definitions, functions and features of operators in C++**

**1. Variables:**

* **Concept:**
  + A variable is a symbolic name for a location in memory used to store data in a program.
* **Definition:**
  + Variables are declared with a data type and can hold values that can be changed during program execution.
* **Purpose:**
  + Variables allow the program to store and manipulate data, providing flexibility and dynamic behavior.

**Session 4:** **KM-02-KT04: Conditions in C++3%**

1. **IAC0401 Explain the definitions, functions and features of conditions in C++**

In C++, conditions are used to make decisions in a program. Conditions involve evaluating an expression that results in a boolean value (**true** or **false**). Based on the outcome of this evaluation, the program takes different paths or executes specific blocks of code. Let's explore the concepts, definition, and purpose of conditions in C++:

**Concepts:**

1. **Boolean Expression:**
   * An expression that evaluates to a boolean value.
   * Examples: **a > b**, **x == y**, **isSunny && isWarm**.

**Session 5:** **KM-02-KT05: Switch statements in C++3%**

1. **IAC0501 Explain the definitions, functions and features of switch statements in C++**

In computer programming languages, a switch statement is a type of selection control mechanism used to allow the value of a variable or expression to change the control flow of program execution via search and map.

Switch statements function somewhat similarly to the if statement used in programming languages like C/C++, C#, Visual Basic .NET, Java and exist in most high-level imperative programming languages such as Pascal, Ada, C/C++, C#,: 374–375  Visual Basic .NET, Java,: 157–167  and in many other types of language, using such keywords as switch, case, select or inspect.

**Session 6:** **KM-02-KT06: Arrays in C++6%**

1. **IAC0601 Explain the definitions, functions and features of Arrays in C++**

In computer science, an array is a data structure consisting of a collection of elements (values or variables), of same memory size, each identified by at least one array index or key. An array is stored such that the position of each element can be computed from its index tuple by a mathematical formula. The simplest type of data structure is a linear array, also called one-dimensional array.

**Session 7:** **KM-02-KT07: Loops in C++3%**

1. **IAC0701 Explain the definitions, functions and features of Arrays in C++**

LOOP is a simple register language that precisely captures the primitive recursive functions.[1] The language is derived from the counter-machine model. Like the counter machines the LOOP language comprises a set of one or more unbounded registers, each of which can hold a single non-negative integer. A few arithmetic instructions (like 'CleaR', 'INCrement', 'DECrement', 'CoPY', ...) operate on the registers. The only control flow instruction is 'LOOP x DO ... END'. It causes the instructions within its scope to be repeated x times. (Changes of the content of register x during the execution of the loop do not affect the number of passes.)

**Session 8:** **KM-02-KT08: References and enumerations in C++3%**

1. **IAC0801 Explain the definitions, functions and features of references and enumerations C++**

In C++, references and enumerations are features that provide additional capabilities and flexibility in programming.

**References:**

A reference in C++ is an alias or alternative name for an existing variable. It allows you to work with the same data using different names. References are declared using the **&** symbol.

**Session 9:** **KM-02-KT09: Exception handling in C++3%**

1. **IAC0901 Explain the definitions, functions and features of exception handling in C++**

Exception handling in C++ allows you to deal with unexpected situations or errors in a more controlled and structured way. It provides a mechanism to detect and handle errors during program execution. The key components of C++ exception handling are **try**, **catch**, and **throw**.

**Try-Catch Blocks:**

The **try** block contains the code that might throw an exception. If an exception occurs, it is thrown out of the **try** block, and the corresponding **catch** block is executed to handle the exception.

**Session 10:** **KM-02-KT10: Dynamic Arrays in C++ 3%**

1. **IAC1001 Ex[lain the definitions, functions and features of dynamic arrays in C++**

In C++, dynamic arrays are created using pointers and the **new** operator to allocate memory on the heap. Dynamic arrays provide flexibility in terms of size, allowing you to allocate and deallocate memory as needed during program execution. Here's a basic overview:

**Dynamic Array Allocation:**

cppCopy code

#include <iostream> int main() { // Allocate a dynamic array of integers int\* dynamicArray = new int[5]; // Initialize the array for (int i = 0; i < 5; ++i) { dynamicArray[i] = i \* 2; } // Access and print array elements for (int i = 0; i < 5; ++i) { std::cout << dynamicArray[i] << " "; } // Deallocate the dynamic array delete[] dynamicArray; return 0; }

**Session 11:** **KM-02-KT11: Pointers in C++ 15%**

1. **IAC1101 Explain the definitions, functions and features of pointers C++**

In C++, pointers are variables that store memory addresses. They play a crucial role in dynamic memory allocation, function arguments, and various low-level operations. Here's an overview of pointers in C++:

**Declaration and Initialization:**

cppCopy code

int main() { // Declaration of a pointer int\* ptr; // Initialization with the address of an existing variable int num = 42; ptr = &num; // Accessing the value using the pointer std::cout << "Value at the memory location pointed to by ptr: " << \*ptr << std::endl; return 0; }

**Session 12:** **KM-02-KT12: C++ Char data types 5%**

* + 1. **IAC1201 Explain the definitions, functions and features of char data types in C++**

In C++, the **char** data type is used to represent single characters. It is a fundamental data type that can store a single 8-bit character. Here are some key aspects of the **char** data type:

**1. Declaration and Initialization:**

cppCopy code

char myChar; // Declaration of a char variable myChar = 'A'; // Initialization with a character literal char anotherChar = 'B';// Declaration and initialization in one line

**2. Character Literals:**

Character literals are enclosed in single quotes, and they represent individual characters. Examples include **'A'**, **'1'**, **'\n'**, etc.

**Session 13:** **KM-02-KT13: File handling in C++: Basic input/output 15%**

1. **IAC1301 Explain the definitions, functions and features of file handling in C++**

File handling in C++ involves performing input and output operations on files. Here's a basic example of file input/output in C++:

**Writing to a File:**

cppCopy code

#include <iostream> #include <fstream> int main() { // Create an ofstream object (output file stream) std::ofstream outputFile("example.txt"); // Check if the file is opened successfully if (outputFile.is\_open()) { // Write data to the file outputFile << "Hello, File Handling in C++!" << std::endl; // Close the file outputFile.close(); } else { // Print an error message if the file couldn't be opened std::cerr << "Error: Unable to open the file for writing." << std::endl; } return 0; }

**Session 14:** **KM-02-KT14: C++ structure (Struct)3%**

1. **IAC1401 Explain the definitions, functions and features of structure (Struct) in C++**

In C++, a structure is a user-defined data type that allows you to group together variables of different data types under a single name. Structures provide a way to organize and represent complex data in a more meaningful way.

Here's a basic example of a C++ structure:

cppCopy code

#include <iostream> #include <string> // Define a structure named Person struct Person { std::string name; int age; double height; }; int main() { // Declare a variable of type Person Person person1; // Assign values to the members of the structure person1.name = "John Doe"; person1.age = 25; person1.height = 175.5; // Display information about the person std::cout << "Name: " << person1.name << std::endl; std::cout << "Age: " << person1.age << std::endl; std::cout << "Height: " <<

**Session 15:** **KM-02-KT15: C++ class and object 3%**

1. **IAC1501 Explain the definitions, functions and features of class and object in C++**

In C++, a class is a user-defined data type that allows you to encapsulate data and functions into a single unit. An object is an instance of a class, created from the blueprint provided by the class definition. Let's look at a simple example:

cppCopy code

#include <iostream> // Define a class named MyClass class MyClass { public: // Members (data and functions) of the class int myVariable; // Member function to display the value of myVariable void display() { std::cout << "Value of myVariable: " << myVariable << std::endl; } }; int main() { // Declare an object of the MyClass class MyClass myObject; // Access and modify the data member of the object myObject.myVariable = 42; // Call a member function of the object myObject.display(); return 0; }

**Session 16:** **KM-02-KT16: C++ operator overloading 3%**

1. **IAC1601 Explain the definitions, functions and features of operator overloading in C++**

Operator overloading in C++ allows you to define how operators behave when applied to objects of a class. It enables you to extend the functionality of operators for user-defined types. Let's explore the concept of operator overloading with examples:

**Overloading Binary Operators:**

cppCopy code

#include <iostream> // Define a class named Complex for complex numbers class Complex { public: // Data members double real; double imag; // Constructor Complex(double r, double i) : real(r), imag(i) {} // Overload the + operator for adding two complex numbers Complex operator+(const Complex& other) const { return Complex(real + other.real, imag + other.imag); } // Overload the << operator for printing complex numbers friend std::ostream& operator<<(std::ostream& os, const Complex& complex) { os << complex.real << " + " << complex.imag << "i"; return os; } }; int main() { // Create complex numbers Complex c1(2.0, 3.5); Complex c2(1.5, 2.0); // Use the overloaded + operator Complex result = c1 + c2; // Display the result using the overloaded << operator std::cout << "Result: " << result << std::endl; return 0; }

**Session 17:** **KM-02-KT17: Std: list in C++ 3%**

1. **IAC1701 Explain the definitions, functions and features of std::list in C++**

**std::list** is a doubly-linked list implementation in C++ provided by the Standard Template Library (STL). It is part of the C++ Standard Library and is defined in the **<list>** header. Here are some key features and operations associated with **std::list**:

**Features:**

1. **Doubly-Linked List:**
   * Each element in a **std::list** is a node that contains a value and pointers to the previous and next nodes.
   * This allows for efficient insertion and removal of elements at both the beginning and end of the list.

**Session 18:** **KM-02-KT18: C++ Functions 5%**

1. **IAC1801 Explain the definitions, functions and features of functions in C++**

In C++, a function is a block of code that performs a specific task. Functions provide modularity and code reusability by allowing you to divide a program into smaller, manageable pieces. Here's an overview of C++ functions:

**Function Declaration:**

cppCopy code

// Function declaration (prototype) void myFunction(int arg1, double arg2); int main() { // Function call myFunction(42, 3.14); return 0; } // Function definition void myFunction(int arg1, double arg2) { // Function body // Perform tasks using arg1 and arg2 }

**Session 19:** **KM-02-KT19: Date/time in C++ 3%**

1. **IAC1901 Explain the definitions, functions and features of date/time in C++**

In C++, date and time functionalities are provided by the **<ctime>** and **<chrono>** libraries. Here's an overview of how you can work with date and time in C++:

**Using <ctime> for Date and Time:**

1. **Current Date and Time:**
   * To get the current date and time, you can use the **std::time** function along with **std::ctime** for formatting.

cppCopy code

#include <iostream> #include <ctime> int main() { std::time\_t currentTime = std::time(nullptr); std::cout << "Current date and time: " << std::ctime(&currentTime); return 0; }

**Session 20:** **KM-02-KT20: Debugging in C++ 10%**

1. **IAC2001 Describe the application of debugging techniques is understood**

Debugging is an essential skill in programming, and C++ provides several tools and techniques to help identify and fix errors in your code. Here are some common debugging techniques and tools in C++:

**1. Print Statements:**

* Use **std::cout** or **std::cerr** statements to print the values of variables at different points in your code. This can help you trace the flow of your program and identify unexpected behavior.