RTB | RWANDA TVET BOARD

RQF LEVEL 4

GENFC401

COMPUTER SYSTEM AND ARCHITECTURE

Fundamentals Of C++ Programming

TRAINER'S MANUAL

October, 2024

FUNDAMENTALS OF C++ PROGRAMMING

KOICA
Korea International Cooperation Agency

TQUM
TVET Quality Management Project

2024

AUTHOR'S NOTE PAGE (COPYRIGHT)

Copies available from:

- HQs: Rwanda TVET Board-RTB
- Web: www.rtb.gov.rw
- **KIGALI-RWANDA**

Original published version: October, 2024

ACKNOWLEDGEMENTS

The publisher would like to thank the following for their assistance in the elaboration of this training manual:

Rwanda TVET Board (RTB) extends its appreciation to all parties who contributed to the development of the trainer's and trainee's manuals for the TVET Certificate IV in Computer System And Architecture , specifically for the module **" GENFC401: FUNDAMENTALS OF C++ PROGRAMMING."**

We extend our gratitude to KOICA Rwanda for its contribution to the development of these training manuals and for its ongoing support of the TVET system in Rwanda

We extend our gratitude to the TQUM Project for its financial and technical support in the development of these training manuals.

We would also like to acknowledge the valuable contributions of all TVET trainers and industry practitioners in the development of this training manual.

The management of Rwanda TVET Board extends its appreciation to both its staff and the staff of the TQUM Project for their efforts in coordinating these activities.

COORDINATION TEAM

RWAMASIRABO Aimable

MARIA Bernadette M. Ramos

MUTIJIMA Asher Emmanuel

Production Team

Authoring and Review

HAKIZIMANA Evariste

MUSABYIMANA Xavier

Validation

NDABAKURANYE Pierre Claver

NISHIMIRWE Liliane

UWAMAHORO Bonaventure

Conception, Adaptation and Editorial works

HATEGEKIMANA Olivier

GANZA Jean Francois Regis

HARELIMANA Wilson

NZABIRINDA Aimable

DUKUZIMANA Therese

NIYONKURU Sylvestre

BYUKUSENGE Protais

Formatting, Graphics, Illustrations, and infographics

YEONWOO Choe

SUA Lim

SAEM Lee

SOYEON Kim

WONYEONG Jeong

MANISHIMWE Marc

Financial and Technical support

KOICA through TQUM Project

This training manual was developed:

Under Rwanda TVET Board (RTB) guiding policies and directives

Under Financial and Technical support of

TABLE OF CONTENT

AUTHOR'S NOTE PAGE (COPYRIGHT)---iii
ACKNOWLEDGEMENTS---iv
TABLE OF CONTENT ---vii
ACRONYMS---ix
INTRODUCTION ---1
MODULE CODE AND TITLE: GENFC401-FUNDAMENTALS OF C++ PROGRAMMING---2
Learning Outcome 1: Apply Basic C++ Concepts.---3
Key Competencies for Learning Outcome 1: Apply basic C++ Concepts 4
Indicative content 1.1: Preparation of Development Environment 6
Indicative content 1.2: Apply variables....11
Indicative content 1.3: Apply operators 17
Indicative content 1.4: Apply control structure....22
Indicative content 1.5: Apply function....29
Indicative content 1.6: Apply arrays 36
Learning outcome 1 end assessment....41
Further information to the trainer....63
Learning Outcome 2: Apply OOP Concepts ---65
Key Competencies for Learning Outcome 2: APPLY OOP CONCEPTS....66
Indicative content 2.1: Apply Class and Object....68
Indicative content 2.2: Apply Inheritance and Polymorphism 75
Indicative content 2.3: Apply Namespaces....81
Indicative content 2.4: Apply Error and Exception handling 86
Learning outcome 2 end assessment....93
Further information to the trainer....103
Learning Outcome 3: Perform CPU Optimization---105
Key Competencies for Learning Outcome 3: Perform CPU Optimization 106
Indicative content 3.1: Apply pointers....108
Indicative content 3.2: Apply file handling 114
Indicative content 3.3: Apply multithreading and concurrency 121

Indicative content 3.4: Apply inline assembly 130
Learning outcome 3 end assessment.....137
Further information to the trainer.....148

ACRONYMS

CLI: Command-Line Interface.

CPU: Central Processing Unit

CSS: Cascading StyleSheet

CSV: Comma-Separated Values

Dev-C++: Developer-C++.

ECU: Engine Control Unit

GUI: Graphical User Interface.

HTML: Hypertext MarkUP Langauge

I/O: Input/Output

IDE: Integrated Development Environment.

Ios: Input/Output Stream.

JSON: Javascript Object Notation

LNN: LongNamespaceName

MinGW: Minimalist GNU for Windows1**.**

MSVC: Microsoft Visual C++

MySQL: My Structured Query Language.

OOP: Object-Oriented Programming

PHP: Hypertext Pre-processor

RAM: Random Access Memory

RQF: Rwanda Qualification Framework

RTB: Rwanda TVET Board

std: Standard

TQUMProject: TVET Quality Management Project.

XML: Extensible Mark Up Language

INTRODUCTION

This trainer's manual encompasses all methodologies necessary to guide you to properly deliver the module titled" **Fundamentals of C++ Programming** ". Students undertaking this module shall be exposed to practical activities that will develop and nurture their competences. The writing process of this training manual embraced Competency-Based Training (CBT) philosophy by providing enough practical opportunities reflecting real life situations.

The trainer's manual is subdivided into Learning Outcomes, each learning outcome has got various topics, you will start guiding a self-assessment exercise to help students rate themselves on their level of skills, knowledge and attitudes about the unit. The trainer's manual will give you the information about the objectives, learning hours, didactic materials, proposed methodologies and crosscutting issues.

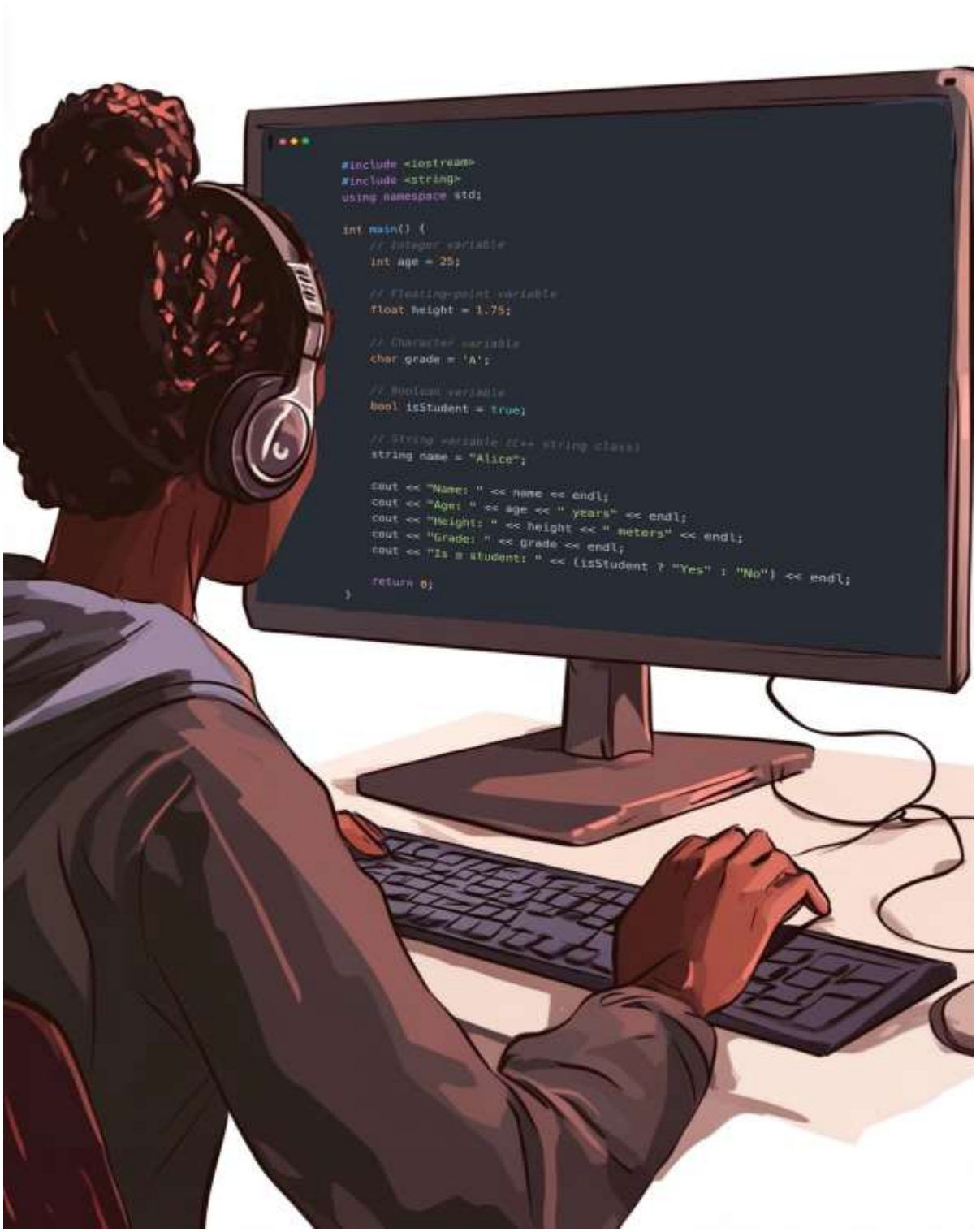
A discovery activity is followed to help students discover what they already know about the unit.

This manual will give you tips, methodologies and techniques about how to facilitate students to undertake different activities as proposed in their trainee's manuals. The activities in this training manual are prepared such that they give opportunities to students to work individually and in groups. After going through all activities, you shall help students to undertake progressive assessments known as formative and finally facilitate them to do their self-reflection to identify your strengths, weaknesses and areas for improvements. Remind them to read the point to remember section which provides the overall key points and takeaways of the unit.

MODULE CODE AND TITLE: GENFC401-FUNDAMENTALS OF C++ PROGRAMMING

Learning Outcome 1: Apply basic C++ concepts
Learning Outcome 2: Apply OOP concepts
Learning Outcome 3: Perform CPU optimization

Learning Outcome 1: Apply Basic C++ Concepts.



Indicative contents

1.1. Preparation of Development Environment
1.2. Apply variables.
1.3. Apply of Operators
1.4. Apply control structure
1.5. Apply function.
1.6. Apply Array

Key Competencies for Learning Outcome 1: Apply basic C++ Concepts

Knowledge	Skills	Attitudes
• Description of C++ development environment. • Description of C++ programming Tools • Description of IDE menus and Icons • Description of variable • Description of operator • Description of C++ control structures • Description of C++ function • Descriptions of C++ arrays	• Installing C++ programming tools (IDE, text editor) • Testing the Development environment • Executing C++ programs • Applying Variable • Applying operators in program • Applying control structures in C++ • Applying C++ function • Applying arrays in C++	• Being self-motivated • Being critical thinker • Being detailed oriented • Being skilful • Being creative

Duration: 20 hrs

Learning outcome 1 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe properly C++ environment based on programming standards.
2. Describe properly C++ programming Tools in line based on their use
3. Install correctly C++ programming tools
4. Test efficiently the installed C++ Development environment.
5. Describe properly IDE menus and Icons
6. Execute correctly C++ program based on C++ standards.
7. Describe properly variable based on C++ programming standards
8. Apply correctly variable in C++ program
9. Describe properly operator and its types based on C++ programming rules and semantics defined.
10. Apply correctly operator in C ++ program
11. Describe correctly the control structures based on C++ rules and syntax defined
12. Apply properly control structures based on C++ rules and syntax defined.
13. Describe correctly C++ function based on task to be done.
14. Applying properly function in C++ programs
15. Describe properly arrays in C++ programming language
16. Apply correctly arrays based on C++ program

Resources

Equipment	Tools	Materials
• Computer • Storage device	• Integrated Development Environment (IDE)	• Internet

Advance Preparation:

Before delivering this learning outcome, you are recommended to:

- Prepare learning place (Classroom/computer lab)
- Avail tutorials of IDE installation

Indicative content 1.1: Preparation of Development Environment

Duration: 2hrs

Theoretical Activity 1.1.1: Description of C++ Programming Language

Notes to the trainer:

Trainer may use small group to describe C++ language.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to answer the following questions:

I. What do you understand by:
   a) C++ programming language
   b) Debugging in C++

II. Describe the following:
   a) C++ file extensions
   b) Features of C+ programming language
   c) Applications of C++ programming language

Step 2: Ask trainees to write their findings on paper/flipcharts

Step 3: Ask trainees to present their findings.

Step 4: Provides expert view

Step 5: Ask trainees to read the key **readings 1.1.1** in trainee's manual

Points to Remember

Description of C++ development environment

- **C++** is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.
- There is a term **Debugging** which is the process of identifying errors and correct them
- There are two most common extensions which are **.cc and .cpp** both store C++ codes.

- C++ programming language has many features including Object-Oriented programming, machine independent, simple to learn as well as multi-threading.
- **C**++ gets its applications in development of operating system, browser, embedded system, distributed system and libraries
- Development tools for C++ are : IDE ,text editors ,translator and debugger

Theoretical Activity 1.1.2: Description of C++ programming Tools

Notes to the trainer:

- Trainer may use small group to describe C++ IDE.
- Trainer may use video to describe C++ IDE

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to answer the following questions:

I. What do you understand by:
   a) Text editor
   b) Interpreter
   c) Debugger

II. Describe the debugging process

III. Differentiate compiler from interpreter.

Step 2: Ask trainees to write their findings on paper/flipcharts

Step 3: Ask trainees to present their findings.

Step 4: Provide expert view

Step 5: Ask trainees to read the key **readings 1.1.2** in trainee's manual.

Points to Remember

- C++ programming tools encompass a variety of software and utilities that assist developers in writing, debugging, and optimizing C++ code. Key tools include:
  - ✓ IDEs like Visual **Studio**, **CLion**, and **Code**::**Blocks**, Dev C++ for development and debugging.
  - ✓ Compilers such as **GCC**, Clang, and **MSVC** to translate code into executable programs.
  - ✓ Version Control tools like **Git** and **SVN** for tracking code changes.
  - ✓ Debuggers like **GDB** and **LLDB** for debugging code.

Practical activity1.1.3: Installing C++ programming tools

Notes to the trainer :

- This activity should take place in computer lab
- Avail computers and Dev-C++ setup

Key steps:

While delivering this activity, pass through the following steps:

Step1: Introduce activity and ask trainees to perform the following task
As computer technician, you are requested to install Dev-C++ in your computer

Step2: Demonstrate to trainees how to install Dev-C++ set up by explaining step by step.

Step3: Ask trainees to run installed Dev-C++ setup and check whether it is well installed.

Step4: Ask trainees to read key **readings 1.1.3.** in trainee's manual

Points to Remember

- **Main steps to install and run Dev C++ on windows system:**
  - Have your Dev C++.
  - Run the Installer
  - Choose Installation Language
  - Accept License Agreement
  - Choose Installation Location
  - Select Components
  - Create Start Menu Folder
  - Installation Process
  - Finish Installation
  - Setup the environment variable
  - Test the installed ID

Practical Activity 1.1.4: Executing C++ program

Notes to the trainer:

- Avail computers having installed Dev-C++

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to perform the following task ,
As Technician in computer system and architecture run a C++ program to display text "**Hello world**" to ensure your development environment is working correctly.

Step 2: Demonstrate to trainees the process of executing program in C++ by explaining step by step.

Step 3: Ask trainees to execute a C++ program by following demonstrated steps and monitor the activity.

Step 4: Ask trainees to inspect outputs of the program to check whether it is the same as the one expected.

Step 5: Ask trainees to read **Key readings 1.1.4.** in trainee's manual

Points to Remember

- The steps followed to execute a C++ program are:
  1. Start the IDE(Dev-C++)
  2. Write program codes
  3. Compile and debug the program
  4. Execute/test the program
  5. Check the output

Application of learning .1.1

XYZ Co Ltd, a software development Company, has a college project that requires integration of C++ programs. As Technician, you are asked to provide help to XYZ Co Ltd by installing Dev C++ in their computers.

Checklist

SN	Criteria	Indicator	Score	
			Yes	No
1	1.Dev-C++ is well installed	1.1 Dev C++ Installer is started		
		1.2 Installation Language is selected		
		1.3 License Agreement is selected		
		1.4 Installation Location is chosen		
		1.5 Start Menu Folder for Dev C++ is created		
		1.6 Dev C++ is launched		

Indicative content 1.2: Apply variables

Duration: 3 hrs

Theoretical Activity 1.2.1: Description of variable

Notes to the trainer:

- Trainer may use small group to describe C++ variable.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to answer the following questions:

1. Describe the following terms:
    a. Variable
    b. Variable name/Identifier
    c. Reserved words /keyword

Step 2: Ask trainees to write their findings on paper/flipcharts

Step 3: Ask trainees to present their findings to whole class

Step 4: Provide expert view

Step 5: Ask trainees to read the key **readings 1.2.1** for more clarification.

Points to Remember

Description of variable

- A variable in C++ is a named storage location in memory that holds a value, which can change during program execution. It consists of:
    - ✓ **Name/Identifier:** A unique name that follows specific naming rules.
    - ✓ **Memory Location:** Memory is allocated based on the data type.
    - ✓ **Value:** Variables can hold and change values during the program's execution.
    - ✓ **Scope:** Defines where the variable can be accessed (local or global).

Theoretical Activity 1.2.2: Description of Data Types

Notes to the trainer:

- Trainer may use small group to describe C++ data type.

Key steps:

While delivering this activity, pass through the following steps:

Step1: Introduce the activity and ask trainees to answer the following questions:

1. What do you understand by data type in C++?
2. Discuss different categories of data types in C++
3. Outline range of data values for each data type in C++

Step2: Ask trainees to write their findings on paper/flipcharts

Step3: Ask trainees to present their findings.

Step4: Provide expert view

Step5: Ask trainees to read the key **readings 1.2.2** for more clarification.

Points to Remember

In C++, data types define the type of data a variable can hold, its size, and the operations allowed. They are classified into three groups:

- **Built-in Data Types:** Predefined by C++, these include:
  - ✓ **Integer types:** int, short, long, long long, and unsigned for various integer sizes.
  - ✓ **Floating-point types:** float, double, and long double for numbers with decimals.
  - ✓ **Character type:** char for single characters.
  - ✓ **Boolean type:** bool for true or false values.
  - ✓ **Void type:** void for no data, often used in functions without a return value.
- **Derived Data Types:** Built from fundamental types, such as arrays, pointers, functions, and references.
- **User-defined Data Types:** Custom types like struct, class, union, and enum, allowing more complex data structures.
- Type modifiers like signed, unsigned, short, and long alter the properties of these types. These data types form the foundation of variable behaviour and memory management in C++.

Theoretical Activity 1.2.3: Description of variable declaration

Notes to the trainer:

- Trainer may use small group to describe declaration of variable in C++

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to answer the following questions:

1) What do you understand by

a. Variable declaration?

b. Variable initialization?

2) Identify the syntax to declare variables in C++

3) Outline various ways of initialization in C++

4) Discuss scope of variable in C++

Step 2: Ask trainees to write their findings on paper/flipcharts

Step 3: Ask trainees to present their findings to the whole class

Step 4: Provide expert view

Step 5: Ask trainees to read the key readings 1.2.3 for more clarification.

Points to Remember

Description of variable declaration

- Variable declaration in C++, refers to the process of reserving memory space to hold a value. It tells the compiler about the existence of the variable and involves specifying a variable's name and data type, allowing the compiler to allocate memory.
- The basic to declare a variable in C++ is: **data_type variable_name**;
- Common data types that involve in declaring variable include int, float, double, char, and bool.
- You can also initialize a variable at the time of declaration, e.g., int age = 30;.
- Multiple variables of the same type can be declared in one line, like int x, y, z;.
- Variable scope determines where it can be accessed: local variables are limited to the function they are declared in, while global variables can be accessed throughout the program.
- Overall, variable declaration is crucial for type safety, memory management, and code clarity in C++.

Practical Activity 1.2.4: Declaring and initializing variable

Notes to the trainer

- Avail computers having installed Dev-C++

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to read the task below:
Write a C++ program that stores and display information about student marks in C++ programming fundamentals

Step 2: Demonstrate to trainees the process of executing program in C++ by explaining step by step.

Step 3: Ask trainees execute a C++ program by following demonstrated steps and monitor the activity.

Step 4: Ask trainees to inspect outputs of the program to check whether it is the same as the one expected.

Step 5: Ask trainees to read **Key readings 1.1.4.** in trainee's manual

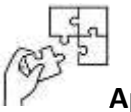
Points to Remember

To perform the task of writing a C++ program that stores and display information about student marks in C++ programming fundamentals

- **Create a New C++ File**:
  - Start a new file and save it with a .cpp extension.
- **Include the Necessary Header**: for example #include <iostream>
  - **Define the main Function**: here the int main() {} is dedfined
- **Declare Variables**: declare necessary variables to be used
  - **Initialize Variables** (optional): Assign initial values to your variables. For example grade = 'A';
- **Output the Variables**: write instructions to display in the console output

```
std::cout << "Age: " << age << std::endl;
std::cout << "Salary: " << salary << std::endl;
std::cout << "Grade: " << grade << std::endl;
```

- **End the main Function**: Return 0 to indicate successful execution, that ends the main task: return 0; }
- **Compile and Run**: Compile your code and run the program to see the output.
- **Observe the Results**: Check the output to confirm that the variables are declared and initialized correctly.



Application of learning 1.2:

INEZA deposited **FRW 200000** in a Micro-finance company at an interest rate of 20% per annum. At the end of each year, the interest earned is added to the deposit and the new amount becomes the deposit for that year. Write a C++ program that would track the growth of deposits over a period of seven years.

Solution:

```
#include <iostream>

#include <cmath>
```

```
int main() {
double principal = 200000.0; // Initial deposit
double rate = 0.20; // Interest rate (20%)
int years;
std::cout << "Enter the number of years: ";
std::cin >> years;
double amount = principal * pow(1 + rate, years);
std::cout << "After " << years << " years, the amount will be: " << amount << " FRW" <<
std::endl;
return 0;
```

Check List

SN	Criteria	Indicator	Score	
			Yes	No
1	1.Variables are properly defined	1.1 Principal amount is initiated		
		1.2 Interest rate is initiated		
		1.3 Number of years is initiated		
2	2.Input are Handled correctly	2.1 Number of years is entered		
3	3.Calculation is correctly performed	3.1 Calculation of compound interest is done.		
4	4.Output is properly obtained	4.1 Expected amount is displayed		
5	5.Code are properly Structured	5.1 Necessary headers are included		
		5.2 Indentation and comments are used		
6	6.Compilation and Testing is well performed	6.1 Syntax errors are checked		
		6.2 Program is running		

Indicative content 1.3: Apply operators

Duration: 3 hrs

Theoretical Activity 1.3.1: Description of Operators

Notes to the trainer:

- Trainer may use small group for describing development environment of C++.
- The use of video and picture as didactic materials are required

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions

1) What do you understand by the terms:
    a. Operator
    b. Operator overloading
    c. Precedence
    d. Associativity
2) Discuss about the types of operators

Step 2: Ask trainees to write the findings on paper /flipchart

Step 3: Ask trainees to present their findings on the whole class

Step 4: Provide expert view

Step 5: Ask trainees to read the **Key readings 1.3.1.** in trainee's manual

Points to Remember

Description of Operators

- **Operator** is a symbol that is applied on the operand in order to perform an operation.

The following are different types of operators:

- **Arithmetic Operators**: These perform basic mathematical operations. Examples include + (addition), - (subtraction), * (multiplication), / (division), and % (modulus).

- **Relational Operators**: These compare two values and return a boolean result. Examples are == (equal to), != (not equal to), > (greater than), < (less than), >= (greater than or equal to), and <= (less than or equal to).
- **Logical Operators**: Used to perform logical operations. Examples include && (logical AND), || (logical OR), and ! (logical NOT).
- **Bitwise Operators**: These perform operations on the binary representations of numbers. Examples are & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), ~ (bitwise NOT), << (left shift), and >> (right shift).
- **Assignment Operators**: Used to assign values to variables. The basic assignment operator is =, but there are compound assignment operators like +=, -=, *=, /=, and %=.
- **Ternary Operator**: This is a shorthand for an if-else statement. It uses the syntax condition ? expression1 : expression2.
- **Operator Precedence and Associativity**: Operators have a defined precedence which determines the order in which operations are performed. For example, multiplication and division have higher precedence than addition and subtraction. Associativity determines the order of operations when operators have the same precedence, typically left-to-right or right-to-left**.**

Practical Activity 1.3.2: Applying Operators

Notes to the trainer

- Avail computers having installed Dev-C++

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to do the task below:
As a computer technician, Create a simple calculator program that performs basic arithmetic operations (addition, subtraction, multiplication, and division) based on user input

Step 2: Demonstrate to trainees the process to create and execute C++ program for simple calculator by explaining step by step.

Step 3: Ask trainees to create and execute following demonstrated steps and monitor the activity.

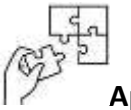
Step 4: Ask trainees to inspect outputs of the program to check whether it is the same as the one expected.

Step 5: Ask trainees to read steps for creating and executing C++ program from **Key readings 1.3.2** in trainee's manual

Points to Remember

To create and run a C++ program that simulates a simple calculator, you have to:

1. Create a new C++ file with a .cpp extension (e.g., calculator.cpp).
2. Include the iostream library at the top of the file.
3. Define the main function to start the program.
4. Declare variables for operands, result, and operation type.
5. Prompt the user for input (two numbers and an operator).
6. Perform the selected arithmetic operation based on user input.
7. Compile and run the program, testing various operations for functionality.



Application of learning.1.3:

As computer technician write a program that will ask a user to enter two numbers and calculate its sum , product, difference, quotient ,remainder and tells the user if both numbers are equal and if the sum is positive and non-zero.

Solution:

```
#include <iostream>
using namespace std;
int main() {

int num1, num2, sum, difference, product, quotient, remainder;

bool areEqual, isSumPositive;

// Asking user to input two numbers

cout << "Enter two integers: ";

cin >> num1 >> num2;
```

```
// Using arithmetic operators

sum = num1 + num2;

difference = num1 - num2;

product = num1 * num2;

quotient = num1 / num2;

remainder = num1 % num2;

// Using relational operators

areEqual = (num1 == num2);

// Using logical operators

isSumPositive = (sum > 0) && (sum != 0);

// Displaying the results

cout << "The sum of " << num1 << " and " << num2 << " is " << sum << endl;

cout << "The difference of " << num1 << " and " << num2 << " is " << difference << endl;

cout << "The product of " << num1 << " and " << num2 << " is " << product << endl;

cout << "The quotient of " << num1 << " and " << num2 << " is " << quotient << endl;

cout << "The remainder of " << num1 << " and " << num2 << " is " << remainder << endl;

cout << "Are the two numbers equal? " << (areEqual ? "Yes" : "No") << endl;

cout << "Is the sum positive and non-zero? " << (isSumPositive ? "Yes" : "No") << endl;

return 0;

}
```

Check List:

SN	Criteria	Indicator	Score	
			Yes	No
1	1.Variable are properly declared	1.1 First number is created		
		1.2 Second number is created		
		1.3 Difference is created		
		1.4 Product is created		
		1.5 Quotient is created		
		1.6 Remainder is created		
2	2.Operator are properly used	2.1 Sum is calculated		
		2.2 Product is calculated		
		2.3 Difference is calculated		
		2.4 Quotient is calculated		
		2.4 Remainder is calculated		
3	3.Input are Handled correctly	3.1 First and second number are entered		
		3.2 Store the input in a variables		
4	4.Output is properly calculated	4.1 Sum is displayed		
		4.2 Product is displayed		
		4.3 Remainder is displayed		
		4.4 Quotient is displayed		
		4.5 Difference is displayed		
		4.6 Two numbers are equal		
5	5.Code are properly Structured	5.1 Necessary headers files are included		
		5.6 Console i/o is used		
6	6.Compilation and Testing is properly implemented	6.1 Syntax errors are checked		
		6.2 Different inputs are used to check the correctness		

Indicative content 1.4: Apply control structure

Duration: 4 hrs

Theoretical Activity 1.4.1: Description of control structure

Notes to the trainer:

- Trainer may use small groups to describe the control structures in C++.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to answer the following questions

1. Define the following terms
    a. Control Structures
    b. Iteration Logic (Repetitive Flow)
    c. Selection Logic (Conditional Flow)
    d. Conditional Statement
    e. Switch Case Statement
    f. Jump statement
2. Differentiate four types of jump statements
3. Describe the basic control structure key concepts

Step 2: Ask trainees to write notes on paper /flipchart

Step 3: Ask trainees to present their findings to the whole class

Step 4: Present expert view

Step 5: Ask trainees to read the Key **readings 1.4.1.** for more clarification

Points to Remember

Description of control structure

- **Proper Syntax**: Ensure you use the correct syntax for each control structure. For example, always use curly braces {} to define the scope of if, else, for, while, and do-while blocks, even if they contain only one statement. This helps avoid errors and improves readability.
- **Condition Evaluation**: Be mindful of the conditions you write. Ensure they are logical and correctly evaluate to true or false. For example, use == for comparison and = for assignment.

- **Avoid Deep Nesting**: Try to avoid deeply nested control structures as they can make your code hard to read and maintain. Consider breaking complex logic into functions.
- **Use `switch` for Multiple Conditions**: When you have multiple conditions based on the same variable, consider using a switch statement instead of multiple if-else statements. This can make your code cleaner and more efficient.
- **Break and Continue**: Use break to exit loops early and continue to skip the current iteration and proceed to the next one. These can help control the flow within loops effectively.
- **Default Case in `switch`**: Always include a default case in a switch statement to handle unexpected values. This ensures your program can handle all possible inputs gracefully.
- **Error Handling**: Implement error handling within your control structures to manage unexpected inputs or conditions. This can prevent your program from crashing and provide useful feedback to users.
- **Logical Operators**: Use logical operators (&&, ||, !) correctly to combine multiple conditions. Ensure you understand the precedence and associativity of these operators to avoid logical errors.
- **Exit Points**: Be aware of where your control structures exit. For example, in nested loops, a break statement will only exit the innermost loop

Practical Activity 1.4.2: Applying control structure

Notes to the trainer

- This activity should take place in computer lab
- Avail computers having installed Dev-C++

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to do the task below

In athletics, runners are awarded medals depending on position as follows: position 1: gold; position 2: silver and position 3: bronze. The rest of the runners are not awarded any medal but receives appreciation message saying "Thank you for your participation". Using **nested if...else** and **switch statements**, write a C++ program that

determines the medal to be awarded to runners depending on time each athlete touches the finish line.

Step 2: Demonstrate to trainees how to create and execute C++ program for awarding runners by explaining step by step.

Step 3: Ask trainees to create and execute the program by following demonstrated steps and monitor the activity.

Step 4: Ask trainees to inspect outputs of the program to check whether it is the same as the one expected.

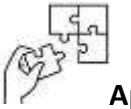
Step 5: Ask trainees to read **Key readings 1.4.2.** in trainee's manual

Points to Remember

This is how to impliment control structure using ,Runners awarding program:

To write and execute a C++ program to award the first runner, you follow these steps:

1) Create a New C++ File**: Open your IDE and create a new project or file.**
2) Write the C++ Code**: write the necessary code, which includes necessary headers, user input for the number of runners and their finish times, sorting of times, and logic to determine medal awards using nested if...else statements (with an optional switch statement).**
3) Compile and Run the Program**: After writing the code, compile it and run the program.**
4) Observe the Output**: The program will display the medal awarded (gold, silver, bronze) or a thank-you message for each runner based on their finishing position.**



Application of learning 1.4.

SANUKI is a cooperative BANK that needs to include technology in their daily working in order to help clients to make transaction ,As CSA Technician Create a simple ATM simulation program that allows a user to: Check their balance ,Deposit money ,Withdraw money and Exit the program, where the minimum balance for a client to withdraw should be 1000 FRW.

Solution:

```
#include <iostream>

using namespace std;

class ATM {

private:

double balance;

public:

ATM(double initialBalance) : balance(initialBalance) {}

void checkBalance() {

cout << "Your balance is: " << balance << " FRW" << endl;

}

void deposit(double amount) {

balance += amount;

cout << "Deposit successful. New balance: " << balance << " FRW" << endl;

}

void withdraw(double amount) {

if (balance - amount >= 1000) {

balance -= amount;

cout << "Withdrawal successful. New balance: " << balance << " FRW" << endl;

} else {

cout << "Insufficient funds. Minimum balance of 1000 FRW must be maintained." << endl;

} }

void runATM() {
```

```
int choice;

double amount;

while (true) {

cout << "ATM Menu:\n";

cout << "1. Check Balance\n";

cout << "2. Deposit Money\n";

cout << "3. Withdraw Money\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

checkBalance();

break;

case 2:

cout << "Enter amount to deposit: ";

cin >> amount;

deposit(amount);

break;

case 3:

cout << "Enter amount to withdraw: ";

cin >> amount;

withdraw(amount);
```

```
break;

case 4:

cout << "Thank you for using the ATM. Goodbye!\n";

return;

default:

cout << "Invalid choice. Please try again.\n";

}

}

}

};

int main() {

 ATM atm(1000);  // Initial balance

atm.runATM();

return 0;

}
```

Checklist

SN	Criteria	Indicator	Score	
			Yes	No
1	Codes are properly structured	1.1. Necessary headers are included		
2	Main Function is properly used	2.1. namespace std to avoid prefixing std:: is used		
		2.2. The main function is defined		
		2.3. Variables like (balance, choice, amount) are initialized		
3	User Interface is clearly designed	3.1. The ATM menu are displayed		
		3.2. The user choice is prompted		
		3.3. Invalid choices are handled		
4	Functionality is properly implemented	4.1. Balance is Checked		
		4.2. Money is deposited		
		4.3. Withdraw money with sufficient balance is checked		
5	Loop and Control Structures are effectively used	5.1. Loops are used		
		5.2. Switch statement for menu choices is used		
6	Input and Output are properly handled	6.1. cin for input and cout for output are used		
		6.2. Ensure proper input validation is ensured		
7	Edge Cases are clearly used	7.1. Insufficient funds during withdrawal is handled		
		7.2. The program exits is clean		
8	Code is clearly readable	8.1. Meaningful variable names are used		
		8.2. Comments to explain the code are added		
		8.3. The code are readable		

Indicative content 1.5: Apply function

Duration: 4 hrs

Theoretical Activity 1.5.1: Description of function

Notes to the trainer:

- Trainer may avail computer

Key steps:

While delivering this activity, pass through the following steps:

Step1: Ask trainees to answer the questions that follow

i. Describe the following terms

a. Function
b. Function defintion
c. Function declaration
d. Function call

ii. List Benefits of using Function in C++

Step 1: Ask trainees to write their findings on paper /flipcharts
Step 2: Ask trainees to present their findings
Step 3: Provide expert view
Step 4: Ask trainees to read the key readings 1.5.1 for more clarification

Points to Remember

Description of function

- **Function Declaration and Definition**:
  - **Declaration**: Also known as a function prototype, it tells the compiler about the function's name, return type, and parameters. It is usually placed before the main function or in a header file.
  - **Definition**: Contains the actual body of the function. It can be placed after the main function or in a separate source file.

- **Return Type**:
    - Every function must specify a return type. If a function does not return a value, use void.
- **Parameters and Arguments**:
    - Functions can take parameters (also called arguments) to pass data. Parameters are specified in the function declaration and definition.
- **Function Call**:
    - To use a function, you call it by its name and pass the required arguments.
- **Pass by Value vs. Pass by Reference**:
    - **Pass by Value**: The function receives a copy of the argument. Changes to the parameter do not affect the original argument.
    - **Pass by Reference**: The function receives a reference to the argument. Changes to the parameter affect the original argument.
- **Default Arguments**:
    - You can provide default values for parameters. If no argument is passed, the default value is used.
- **Function Overloading**:
    - C++ allows multiple functions with the same name but different parameters (number or type). This is called function overloading.
- **Inline Functions**:
    - Use the inline keyword to suggest that the compiler replace the function call with the function code to reduce the overhead of function calls.
- **Recursion**:
    - A function can call itself. This is known as recursion. Ensure there is a base case to terminate the recursive calls.
- **Scope and Lifetime**:
    - Variables declared inside a function are local to that function and are destroyed when the function exits. Use global variables sparingly to avoid side effects.

Theoretical Activity 1.5.2: Description of types of functions

Notes to the trainer:

- Trainer may use small group for describing development environment of C++.
- The use of video and tutorials as didactic materials.

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to answer the following questions .

1) Describe the following:
    a. Built-in function
    b. User defined function
    c. Inline function
    d. Lambda function

Step 2: Ask trainees to write notes of their findings on paper/flipchart

Step 3: Ask trainees to present their findings

Step 4: Provide expert view

Step 5: Ask trainees to read the key readings 1.5.2 in trainee's manuals.

Points to Remember

Description of types of functions

- C++ functions can be found in various types such as
    - Library function; those which are predefined in the system language. Ex: sqrt(), pow()…
    - User defined functions; those functions that are defined by the user. Ex: AddTwo(), PrintString(),…
- A user defined function may be classified as:
    - Inline function
    - Lambda function
- For a function to be implemented it must:
    - Be declared
    - Be defined
    - Be called
- To call a user defined function two ways are provided:

- Call by value
- Call by reference

Practical Activity 1.5.3: Applying function in C++

Notes to the trainer

- This activity should take place in computer lab
- Avail computers having installed Dev-C++

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to perform the following task :
As a firmware development technician you are tasked to write and run C++ program that takes two integer numbers ,find their sum ,and calculate one number raised to the power to the other using **pow()** function

Step 2: Demonstrate to trainees how to apply function in C++

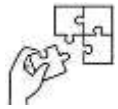
Step 3: Ask trainees to compile and run C++ program by following demonstrated steps and monitor the activity.

Step 4: Ask trainees to inspect outputs of the program to check whether it is the same as the one expected.

Step 5: Ask trainees to read Key readings **1.5.3.** in trainee's manuals

Points to Remember

- To run a program that implements C++ function follow these steps
  - Start the IDE(Dev-C++)
  - Write codes
  - Test the written program codes



Application of learning 1.5.

XYZ SACCO is creating a desktop banking application to help customers manage their savings accounts offline. The program enables users to deposit funds, withdraw money (with insuffi cient funds checks), check account balances, and calculate interest over time at a fixed rate. The task is to create a C++ program simulating this banking system.

SOLUTION

```
#include <iostream>
#include <iomanip>
#include <math.h>
using namespace std;
class SavingsAccount {
private:
double balance;
const double interestRate;
public:
SavingsAccount(double initialBalance, double rate)
: balance(initialBalance), interestRate(rate) {}

void deposit(double amount) {
balance += amount;
cout << "You've deposited: " << amount << endl;
displayBalance();
}
void withdraw(double amount) {
if (amount <= balance) {
balance -= amount;
cout << "You've withdrawn: " << amount << endl;
} else {
cout << "Insufficient funds. Withdrawal failed." << endl;
}
displayBalance();
}

void displayBalance() const {
cout << "Current balance: " << fixed << setprecision(2) << balance << endl;
}
```

```
void calculateInterest(int years) const {
double futureBalance=balance * pow((1 + interestRate), years);
cout << "Balance after " << years << " years at " << interestRate * 100 << "% interest rate: "
<< fixed << setprecision(2) << futureBalance << endl;
}
};

int main() {
double initialBalance, rate;
int choice, years;
double amount;

cout << "Welcome to XYZ SACCO Banking System!" << endl;
cout << "Enter initial balance: ";
cin >> initialBalance;
cout << "Enter annual interest rate (e.g., 0.05 for 5%): ";
cin >> rate;

SavingsAccount account(initialBalance, rate);

while (true) {
cout << "\nMenu: \n1. Deposit \n2. Withdraw \n3. Check Balance \n4. Calculate Interest
\n5. Exit" << endl;
cout << "Enter your choice: ";
cin >> choice;

switch (choice) {
case 1:
cout << "Enter amount to deposit: ";
cin >> amount;
account.deposit(amount);
break;
case 2:
cout << "Enter amount to withdraw: ";
cin >> amount;
account.withdraw(amount);
break;
case 3:
account.displayBalance();
break;
case 4:
```

```
cout << "Enter number of years: ";
cin >> years;
account.calculateInterest(years);
break;
case 5:
cout << "Thank you for using XYZ SACCO Banking System. Goodbye!" << endl;
return 0;
default:
cout << "Invalid choice. Please try again." << endl;
}
}
return 0;
}
```

Check List:

SN	Criteria	Indicator	Score	
			Yes	No
1	1.Tools are well used	1.1 Computer is started		
		1.2 Dev C++ opened		
2	2.C++ program is correctly written	2.1 Header files are include		
		2.2 Functions are defined		
		2.3 Main function is defined		
		2.4 Functions are called		
3	3.Output is well produced	3.1 Inputs are accepted		
		3.2 Outputs are displayed		
		3.3 Error message is displayed		

Indicative content 1.6: Apply arrays

Duration: 4 hrs

Theoretical Activity 1.6.1: Descriptions of array

Notes to the trainer:

- Trainer may use small group for describing Arrays in C++
- The use of video and tutorials as didactic materials.

Key steps:

While delivering this activity, pass through the following steps:

Step1: Introduce the activity and ask trainees to perform the task below:

1. Describe the following :
   - Array in C++
   - Array declaration
   - Array initialization
   - Access array elements in C++
2. Identify types of array

Step2: Ask trainees to write their findings on paper /flipchart

Step3: Ask trainees to present their findings

Step4: Provide expert view

Step5: Ask trainees to read the **key readings 1.6.1** in trainee's manuals

Points to Remember

DESCRIPTION OF AN ARRAY

- An array is a collection of more than one item with similar data types stored in contiguous memory location.
- Array is basically made of array element, array index, array name, array memory address as the basic components that allow to store, access, and manipulate its data efficiently
- Before using array you must first declare it and can be initialized through various method such as individual item initialization or declaration time initialization
- Index is an array component that serves for its data access and manipulation.
- To classify array refer to its size and the way it is declared. Hence we can find:

- Single dimension arrays
- Multi dimensions array (2D, 3D arrays)
- Static arrays
- Dynamic arrays

Practical Activity 1.6.2: Applying Arrays

Notes to the trainer

- This activity should take place in computer lab
- Avail computers having installed Dev-C++

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to do the task below :
Develop a C++ program to manage student grades using both single and multi-dimensional arrays. This program will store students' names and grades, calculat e the class average, and display the grades in a tabular format.

Step 2: Demonstrate to trainees the process of applying arrays in C++ by explaining step by step.

Step 3: Ask trainees to compile and run C++ program by following demonstrated steps and monitor the activity.

Step 4: Ask trainees to inspect outputs of the program to check whether it is the same as the one expected.

Step 5: Ask trainees to read Key readings **1.6.2.** in trainee's manual

Points to Remember

- The following are steps involved to run a program that applies arrays in C++
  - Start the IDE(Dev-C++)
  - Write codes
  - Test the written program codes

Application of learning 1.6.:

At XYZ TSS, a C++ trainer for a class of 25 trainees needs to automate the management of trainees' test grades. The task involves creating a C++ program that allows entering marks, calculating the average grade, identifying the highest and lowest grades, and displaying all grades in tabular format.

SOLUTION

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
class GradeManager {
private:
vector<int> grades;
public:
void enterGrades() {
int grade;
for (int i = 0; i < 25; ++i) {
cout << "Enter grade for trainee " << i + 1 << ": ";
cin >> grade;
grades.push_back(grade);
}
}
void calculateAverage() {
double sum = 0;
for (int grade : grades) {
sum += grade;
```

```
}
double average = sum / grades.size();
cout << "Average grade of the class: " << average << endl;
}
void findHighestLowest() {
int highest = *max_element(grades.begin(), grades.end());
int lowest = *min_element(grades.begin(), grades.end());
cout << "Highest grade: " << highest << endl;
cout << "Lowest grade: " << lowest << endl;
}
void displayGrades() {
cout << "All grades:" << endl;
for (int i = 0; i < grades.size(); ++i) {
cout << "Trainee " << i + 1 << ": " << grades[i] << endl;
}
}
};
int main() {
GradeManager gm;
gm.enterGrades();
gm.calculateAverage();
gm.findHighestLowest();
return 0;
}
```

Check list:

SN	Criteria	Indicator	Score	
			Yes	No
1	1. Tools are well used	1.1 Computer is started		
		1.2 Dev C++ opened		
2	2. C++ program is correctly written	2.1 Header files are include		
		2.2 Array is created and initialized		
		2.3 Main function is defined		
		2.4 Array elements are accessed		
		2.5 Average, high grade, low grade and all marks are displayed		
		2.6 Error message is displayed		

Learning outcome 1 end assessment

Theoretical assessment

I. Choose the correct answer about apply C++ concepts

1. Who invented C++?

   a) Dennis Ritchie
   b) Ken Thompson
   c) Brian Kernighan
   d) Bjarne Stroustrup
   Answer: d) Bjarne Stroustrup

2. Which of the following is used for comments in C++?

   a) /* comment */
   b) // comment */
   c) // comment
   d) Both // comment and /* comment */
   Answer: d) Both // comment and /* comment */

3. Which of the following is a correct identifier in C++? a) VAR_1234
   b) $var_name
   c) 7VARNAME
   d) 7var_name
   Answer: a) VAR_1234

4. Which of the following is not a type of constructor in C++?

   a) Default constructor
   b) Parameterized constructor
   c) Copy constructor
   d) Friend constructor
   Answer: d) Friend constructor

5. What is the correct syntax for including a user-defined header file in C++?

   a) #include [userdefined]
   b) #include "userdefined"
   c) #include <userdefined.h>
   d) #include <userdefined>
   Answer: b) #include "userdefined"

6. What is the output of the following C++ code?

```
#include <iostream>
using namespace std;
int main() {
   int a = 5;
  int b = 10;
 cout << a + b;
return 0;
        }
```

   a) 5
   b) 10
   c) 15
   d) Compilation error
   Answer: c) 15

7. Which of the following is used to define a constant in C++? a) #define
   b) const
   c) static
   d) Both #define and const
   Answer: d) Both #define and const

8. What is the default access specifier for members of a class in C++? a) Public
   b) Private
   c) Protected
   d) None of the above
   Answer: b) Private

9. Which of the following is the correct syntax to declare a pointer in C++?

   a) int *ptr;
   b) int ptr*;
   c) int* ptr;
   d) Both a and c
   Answer: d) Both a and c

10. What is the purpose of the return statement in C++? a) To exit a loop
    b) To exit a function and return a value
    c) To define a constant
    d) To allocate memory
    Answer: b) To exit a function and return a value

II. Choose the correct answer about setting up a development environment for C++ along with their answers:

1. Which of the following is an essential component for setting up a C++ development environment?

   a) Text Editor
   b) Compiler
   c) Debugger
   d) All of the above
   Answer: d) All of the above

2. Which IDE is commonly used for C++ development on Windows?

3. a) Xcode
   b) Visual Studio
   c) Eclipse
   d) Code::Blocks
   Answer: b) Visual Studio

4. What is the purpose of a C++ compiler?

   a) To edit the source code
   b) To convert source code into machine code
   c) To debug the program
   d) To run the program directly
   Answer: b) To convert source code into machine code

5. What is the file extension for C++ source files?

   a) .c
   b) .cpp
   c) .java
   d) .py
   Answer: b) .cpp

6. Which of the following is an online IDE for C++?

   a) Geany
   b) ide.geeksforgeeks.org
   c) Notepad++
   d) Sublime Text
   Answer: b) ide.geeksforgeeks.org

7. What is the role of a debugger in a C++ development environment?

   a) To write code
   b) To compile code
   c) To find and fix errors in the code
   d) To execute code
   Answer: c) To find and fix errors in the code

8. Which command is used to compile a C++ program using the GCC compiler?

   a) gcc program.cpp
   b) g++ program.cpp
   c) compile program.cpp
   d) run program.cpp
   Answer: b) g++ program.cpp

9. What is the purpose of setting the PATH environment variable for a compiler?

   a) To specify the location of source files
   b) To specify the location of the compiler executable
   c) To specify the output directory for compiled files
   d) To specify the location of libraries
   Answer: b) To specify the location of the compiler executable

10. Which of the following is a cross-platform IDE for C++ development?

    a) Visual Studio
    b) Xcode
    c) Code::Blocks
    d) Notepad++
    Answer: c) Code::Blocks

III.Choose the correct answer about about applying variables in C++

1. Which of the following is the correct way to declare an integer variable in C++?

   a) int var;
   b) integer var;
   c) int: var;
   d) var int;
   Answer: a) int var;

2. What is the default value of a local variable in C++? a) 0
   b) NULL
   c) Undefined
   d) 1
   Answer: c) Undefined

3. Which of the following is a valid variable name in C++? a) 1variable
   b) variable_1
   c) variable-1
   d) variable 1
   Answer: b) variable_1
4. What will be the output of the following C++ code?

```
#include <iostream>
using namespace std;
int main() {
int a = 5;
int b = 2;
int c = a + b;
cout << c;
return 0;
}
```

   a) 5
   b) 2
   c) 7
   d) Compilation error
   Answer: c) 7
5. Which of the following is used to declare a constant variable in C++? a) const
   b) constant
   c) final
   d) static
   Answer: a) const
6. What is the size of an int variable in C++?

   a) 2 bytes
   b) 4 bytes
   c) 8 bytes
   d) Depends on the system/compiler
   Answer: d) Depends on the system/compiler

7. Which of the following is the correct syntax to initialize a variable at the time of declaration? a) int var = 10;
   b) int var : 10;
   c) int var == 10;
   d) int var <- 10;
   Answer: a) int var = 10;
8. What will be the output of the following C++ code?

```
#include <iostream>
using namespace std;
int main() {
  int x = 10;
int y = 20;
x = y;
 cout << x;
 return 0;
}
```

   a) 10
   b) 20
   c) 30
   d) Compilation error
   Answer: b) 20
9. Which of the following is not a valid variable declaration in C++? a) int var1;
   b) float var2;
   c) double var3;
   d) char var4 = 'A';
   Answer: d) char var4 = 'A';
10. What is the scope of a variable declared inside a function in C++? a) Global
    b) Local
    c) Static
    d) Dynamic
    Answer: b) Local

IV. Choose the correct answer about applying operators in C++

1. Which of the following is the assignment operator in C++?

   a) ==
   b) =
   c) ===
   d) :=
   Answer: b) =

2. Which operator is used for checking equality? a) :=
   b) =
   c) ==
   d) !=
   Answer: c) ==

3. What is the result of 5 % 2?

   a) 2.5
   b) 2
   c) 1
   d) 0
   Answer: c) 1

4. Which operator increases the value of a variable by 1?

   a) ++
   b) --
   c) +=
   d) *=
   Answer: a) ++

5. What is the purpose of the != operator?

   a) Multiplication
   b) Modulus
   c) Not equal to
   d) Division
   Answer: c) Not equal to

6. What does the && operator represent? a) Bitwise AND
   b) Logical AND
   c) Bitwise OR
   d) Logical OR
   Answer: b) Logical AND

7. Which operator has the highest precedence? a) +
   b) /
   c) %
   d) ()
   Answer: d) ()
8. What is the result of 10 | 5? a) 15
   b) 5
   c) 2
   d) 10
   Answer: a) 15
9. Which operator is used to allocate dynamic memory in C++? a) malloc
   b) new
   c) alloc
   d) create
   Answer: b) new
10. What does the ^ operator do in C++?

    a) Exponentiation
    b) Logical XOR
    c) Bitwise XOR
    d) None of the above
    Answer: c) Bitwise XOR

V. Choose the correct answer about applying control structures in C++

1. Which control structure is used for executing a block of statements repeatedly based on a condition?
   a) if
   b) switch
   c) loop
   d) goto
   Answer: c) loop
2. Which keyword is used to test a condition in C++?

   a) test
   b) switch
   c) decide
   d) if
   Answer: d) if

3. Which of the following is NOT a loop in C++?

   a) for
   b) while
   c) do
   d) check
   Answer: d) check

4. How many times is the body of a do-while loop guaranteed to execute?

   a) 0
   b) 1
   c) Until the condition is true
   d) Infinitely
   Answer: b) 1

5. What will be the output of the following code snippet?

```
int x = 5;
if (x == 5)
   cout << "Yes";
else
  cout << "No";
```

   a) No
   b) Yes
   c) Error
   d) None of the above
   Answer: b) Yes

6. Which control structure allows you to choose between multiple alternatives?

   a) if-then
   b) for
   c) switch-case
   d) while
   Answer: c) switch-case

7. What is the purpose of the break statement in loops?

   a) To start the loop
   b) To exit the loop immediately
   c) To skip one iteration
   d) None of the above
   Answer: b) To exit the loop immediately

8. Which loop is best suited for iterating over arrays when you know the number of iterations in advance?

   a) if
   b) do-while
   c) while
   d) for
   Answer: d) for

9. How can you execute a block of code irrespective of whether a condition in an if statement is true or false?

   a) else
   b) elseif
   c) then
   d) finally
   Answer: a) else

10. Which of the following statements will run indefinitely?

    a) for(; ;) { }
    b) while(1) { }
    c) do { } while(1);
    d) All of the above
    Answer: d) All of the above

VI. Choose the correct answer about applying functions in C++

1. Which of the following is the default return value of functions in C++ if no return type is specified?

   a) int
   b) char
   c) float
   d) void
   Answer: a) int

2. What happens to a function defined inside a class without any complex operations (like looping, a large number of lines, etc.)?

   a) It becomes a virtual function of the class
   b) It becomes a default calling function of the class
   c) It becomes an inline function of the class
   d) The program gives an error
   Answer: c) It becomes an inline function of the class

3. What is an inline function?

   a) A function that is expanded at each call during execution
   b) A function that is called during compile time
   c) A function that is not checked for syntax errors
   d) A function that is not checked for semantic analysis
   Answer: a) A function that is expanded at each call during execution

4. When are inline functions expanded?

   a) Compile-time
   b) Run-time
   c) Never expanded
   d) End of the program
   Answer: a) Compile-time

5. In which of the following cases may inline functions not work?

   i) If the function has static variables.
   ii) If the function has global and register variables.
   iii) If the function contains loops
   iv) If the function is recursive
   a) i, iv
   b) iii, iv
   c) ii, iii, iv
   d) i, iii, iv
   Answer: d) i, iii, iv

6. When do we define the default values for a function?

   a) When a function is defined
   b) When a function is declared
   c) When the scope of the function is over
   d) When a function is called
   Answer: b) When a function is declared

7. Where should default parameters appear in a function prototype?

   a) To the rightmost side of the parameter list
   b) To the leftmost side of the parameter list
   c) Anywhere inside the parameter list
   d) Middle of the parameter list
   Answer: a) To the rightmost side of the parameter list

8. If an argument from the parameter list of a function is defined as constant, then:

   a) It can be modified inside the function
   b) It cannot be modified inside the function
   c) Error occurs
   d) Segmentation fault
   Answer: b) It cannot be modified inside the function

9. Which of the following features is used in function overloading and functions with default arguments?

   a) Encapsulation
   b) Polymorphism
   c) Abstraction
   d) Modularity
   Answer: b) Polymorphism

10. What will be the output of the following C++ code?

```
#include <iostream>

using namespace std;

int fun(int x = 0, int y = 0, int z) {
return (x + y + z);
}

int main() {

cout << fun(10);
return 0;
}
```

   a) 10
   b) 0
   c) Error
   d) Segmentation fault
   Answer: c) Error

VII. Choose the correct answer about applying arrays in C++

1. Which of the following correctly declares an array in C++?

   a) int array[10];
   b) int array;
   c) array{10};
   d) array array[10];
   Answer: a) int array[10];

2. What is the index number of the last element of an array with 9 elements?

   a) 9
   b) 8
   c) 0
   d) Programmer-defined
   Answer: b) 8

3. What is the correct definition of an array?

   a) An array is a series of elements of the same type in contiguous memory locations
   b) An array is a series of elements of different types in contiguous memory locations
   c) An array is a series of elements of the same type placed in non-contiguous memory locations
   d) An array is an element of different types
   Answer: a) An array is a series of elements of the same type in contiguous memory locations

4. Which of the following accesses the seventh element stored in an array? a) array[6];
   b) array[7];
   c) array(7);
   d) array;
   Answer: a) array[6];

5. Which of the following gives the memory address of the first element in an array?

   a) array[0];
   b) array[1];
   c) array(2);
   d) array;
   Answer: a) array[0];

6. What will be the output of the following C++ code?

```
#include <iostream>

using namespace std;

int main() {

  int array[] = {0, 2, 4, 6, 7, 5, 3};

 int n, result = 0;

 for (n = 0; n < 7; n++) {

 result += array[n];

}

cout << result;
```

```
return 0;
}
```

a) 25
b) 26
c) 27
d) 21

Answer: b) 27

7. How do you pass an array to a function in C++?

   a) By copying
   b) By value
   c) By pointer
   d) By name

   Answer: c) By pointer

8. What will be the output of the following C++ code?

```
#include <iostream>
using namespace std;
void modifyArray(int arr[], int size) {
for (int i = 0; i < size; i++) {
arr[i] += 5;
}
}
int main() {
int arr[3] = {1, 2, 3};
modifyArray(arr, 3);
for (int i = 0; i < 3; i++) {
cout << arr[i] << " ";
}
return 0;
}
```

a) 1 2 3
b) 6 7 8
c) 1 7 3
d) Compilation error

Answer: b) 6 7 8

9. Which of the following are true about multidimensional arrays in C++?

   a) They are arrays of arrays
   b) The syntax to declare a 2D array is type arrayName[rows][columns];
   c) They can have two or more dimensions
   d) All of the above
   Answer: d) All of the above

10. What is the size of the array char arr[] = "geeksforgeeks";?

    a) 12
    b) 13
    c) 15
    d) 14
    Answer: b) 13

VIII. Answer the questions below by filling the missing terms in the blank space

1. The keyword used to define a constant variable in C++ is _____
   Answer: const

2. The _____ loop is guaranteed to execute at least once.
   Answer: do-while

3. The _____ statement is used to exit a loop prematurely.
   Answer: break

4. A function that calls itself is known as a _____ function.

   Answer: recursive

5. The return type of a function that does not return a value is _____.
   Answer: void

6. The _____ statement is used to exit a loop prematurely.

   Answer: break

7. A function that calls itself is known as a _____ function.

   Answer: recursive

8. The return type of a function that does not return a value is _____.

   Answer: void

9. An array is a collection of elements of the same _____ stored in contiguous memory locations.

   Answer: type

10. Question: The index of the first element in a C++ array is _____.
    Answer: 0

11. Question: The correct way to declare an array of 10 integers is _____.
    Answer: int arr[10];
12. To initialize an array of 5 integers with values 1, 2, 3, 4, 5, you write _____.
    Answer: int arr[5] = {1, 2, 3, 4, 5};
13. The syntax to access the third element of an array named arr is _____.
    Answer: arr[2]
14. To change the value of the fifth element of an array arr to 10, you write _____.
    Answer: arr[4] = 10;
15. Question: A two-dimensional array can be declared as _____.
    Answer: int arr[3][4];
16. Question: To access the element in the second row and third column of a 2D array arr, you write _____.
    Answer: arr[1][2]
17. The function _____ can be used to get the size of an array in C++.
    Answer: sizeof
18. Question: To pass an array to a function, you typically pass the _____ of the array.
    Answer: pointer
19. To find the maximum value in an array, you need to iterate through the array using a _____. **Answer: loop**
20. The keyword used to declare a variable in C++ is _____.
    Answer: int, float, double, etc.
21. To initialize a variable x of type int with the value 10, you write _____.
    Answer: int x = 10
22. A variable declared inside a function is called a _____ variable.
    Answer: local
23. A variable declared outside all functions is called a _____ variable.
    Answer: global
24. A variable that can hold a sequence of characters is of type _____.
    Answer: string
25. A variable that can hold a true or false value is of type _____.
    Answer: bool
26. A constant variable must be initialized at the time of _____.
    Answer: declaration
27. The data type used to store a sequence of characters is _____.
    Answer: **string**
28. The data type used to store a single character is _____.
    Answer: **char**
29. The data type used to store a true or false value is _____.
    Answer: **bool**
30. The data type used to store a floating-point number is _____.
    Answer: **float or doubl**

31. The operator used to add two numbers is _____.
    Answer: **+**
32. The operator used to subtract one number from another is _____.
    Answer: -
33. The operator used to multiply two numbers is _____.
    Answer: *
34. The operator used to divide one number by another is _____.
    Answer: /
35. The operator used to find the remainder of a division is _____.
    Answer: %
36. The operator used to check if two values are equal is _____.
    Answer: ==
37. The operator used to check if one value is greater than another is _____.
    Answer: >
38. The operator used to check if one value is less than or equal to another is _____. **Answer**: <=
39. The operator used to perform a logical AND operation is _____.
    Answer: &&
40. The operator used to perform a logical OR operation is _____.
    Answer: ||
41. The operator used to perform a logical NOT operation is _____.
    Answer: !
42. The operator used to assign a value to a variable is _____.
    Answer: =
43. The operator used to add and assign a value to a variable is _____.
    Answer: +
44. The operator used to subtract and assign a value to a variable is _____.
    Answer: -=

IX. In the table below ,Match every C++ terms with its corresponding description by writing the letterof C++ term after the number of its descriptionin the answer column

Answers	**Descriptions**	**C++terms**
1. constant	**1.** The keyword used to declare a constant variable is	Poinnter
2. Pointer	**2.** A variable that stores the address of another variable is called a ___	constant
3. +	**3.** The operator used to add two numbers is _____.	= =

4. = =	**4.** The operator used to check if two values are equal is _____.	Recursive
5. Recursive	**5.** A function that calls itself is known as a _____ function.	+
6. Inline	**6.** The keyword _____ is used to suggest to the compiler to expand the function inline.	0
7. 0	**7.** The index of the first element in a C++ array is _____.	Inline
8. int arr[5] = {1, 2, 3, 4, 5};	**8.** To initialize an array of 5 integers with values 1, 2, 3, 4, 5, you write _____	Local
9. Local	**9.** A variable declared inside a function is called a _____ variable.	int arr[5] ={1, 2, 3, 4, 5};
10. %	**10.** The operator used to find the remainder of a division is _____	
11. Void	**11.** The return type of a function that does not return a value is _____.	%
12. int arr[3][4];	**12.** A two-dimensional array can be declared as _____.	Void

X. Read the following statements and Answer by true if it is correct or false if it is not correct

1. The #include directive is used to include standard libraries in a C++ program.
   Answer: True

2. Variables in C++ must be declared before they are used.
   Answer: True

3. The int data type in C++ can store decimal values.
   Answer: False
4. The = operator is used for comparison in C++.
   Answer: False

5. The if statement is used to execute a block of code only if a specified condition is true.
   Answer: True
6. A for loop in C++ can be used to iterate over a range of values.
   Answer: True
7. Functions in C++ can return multiple values.
   Answer: False
8. Arrays in C++ can hold elements of different data types.
   Answer: False

9. The main function is the entry point of a C++ program.
   Answer: True
10. The && operator is used for logical AND operations in C++.
    Answer: True
11. The || operator is used for logical OR operations in C++.
    Answer: True
12. The ! operator is used for logical NOT operations in C++.
    Answer: True
13. The switch statement can be used to select one of many code blocks to be executed.
    Answer: True
14. The break statement is used to exit a loop or switch statement prematurely.
    Answer: True
15. The continue statement is used to skip the current iteration of a loop and proceed to the next iteration.
    Answer: True
16. The return statement is used to exit a function and return a value to the caller.
    Answer: True

17. The void keyword is used to declare a function that does not return a value.
Answer: True
18. The sizeof operator returns the size of a variable or data type in bytes.
Answer: True
19. The cin object is used to output data to the console.
Answer: False
20. The cout object is used to input data from the console.
Answer: False

Practical assessment

Create a C++ program to calculate a student's average grade from scores in five subjects. Use variables for the student's name and scores, store the scores in an array, and define a function to compute the average. Implement loops for input and total calculation, then display the student's name and average score.

Solution

```
#include <iostream>

#include <string>

using namespace std;

// Function to calculate the average score

double calculateAverage(int scores[], int numSubjects) {

int total = 0;

for (int i = 0; i < numSubjects; i++) {

total += scores[i];

}

return static_cast<double>(total) / numSubjects;

}

int main() {

const int numSubjects = 5; // Number of subjects

string studentName; // Variable for student's name

int scores[numSubjects]; // Array to store scores

// Input student's name

cout << "Enter the student's name: ";
```

```
getline(cin, studentName);
// Input scores for each subject
for (int i = 0; i < numSubjects; i++) {
cout << "Enter score for subject " << (i + 1) << ": ";
cin >> scores[i];
}
// Calculate average score
double average = calculateAverage(scores, numSubjects);
// Display the results
cout << "Student's Name: " << studentName << endl;
cout << "Average Score: " << average << endl;
return 0;
}
```

Check List

SN	Criteria	Indicators	scores	
			Yes	No
1	user inputs properly handled	1.1. Student's name and scores are entered		
		1.2. Input validation is indicated		
2	Variables are appropriately used.	Variables are defined		
3	Arrays are properly implemented	Scores are stored in an array		
4	Functions are effectively used.	CalculateAverage function is used to		
5	Control structures are properly used.	Loops are utilized		
6	Output is clearly displayed and informative	6.1. The student's name and average score are displayed in an understandable format.		

		6.2. The program is compiled and executed		
7	Codes are well structured.	7.1. Indentation, spacing, and naming conventions are applied.		
		7.2. Comments are used		
8	Functional requirements are properly met.	Average grade is calculated and displayed		

Further information to the trainer

Babb, J. (2017). C++: A Beginner's Guide (4th ed.). McGraw-Hill.

Barlow, J., & McGowan, C. (2016). C++ Programming for Beginners. CreateSpace Independent Publishing Platform.

Bjarne Stroustrup. (2014). The C++ Programming Language (4th ed.). Addison-Wesley.

Bjarne Stroustrup. (2019). The C++ Programming Language (5th ed.). Addison-Wesley.

C++ Institute. (2020). C++ Programming Essentials. Retrieved from https://www.cppinstitute.org/

Cplusplus.com - C++ Language Tutorial: This tutorial explains C++ from its basics to more advanced features with practical examples.

Deitel, P. J., & Deitel, H. M. (2016). C++: How to Program (10th ed.). Pearson Education.

Dromey, R. G. (2018). How to Solve It by Computer. Prentice Hall.

Eckel, B. (2010). Thinking in C++ (2nd ed.). Prentice Hall.

Gaddis, T. (2018). Starting Out with C++: From Control Structures through Objects (9th ed.). Pearson.

GeeksforGeeks - C++ Basics: This tutorial covers the fundamental concepts of C++ with examples and explanations.

Gibbons, J. (2015). Data Structures and Algorithms in C++. Cambridge University Press.

Griffiths, D. (2019). C++ Programming: From Problem Analysis to Program Design (5th ed.). Cengage Learning.

Here are some useful links to help you learn and apply basic C++ concepts:

Kochan, S. G. (2014). Programming in C++ (4th ed.). Sams Publishing.

Lambert, K. (2018). Fundamentals of C++ Programming (3rd ed.). Cengage Learning.

LearnCpp.com: A free website dedicated to teaching modern C++ with detailed lessons and best practices.
Lippman, S. B., Lajoie, J. & Moo, B. E. (2013). C++ Primer (5th ed.). Addison-Wesley.

Savitch, W. J. (2016). Absolute C++ (6th ed.). Pearson.

Schildt, H. (2018). C++: The Complete Reference (4th ed.). McGraw-Hill.

Sebesta, R. W. (2019). Concepts of Programming Languages (11th ed.). Pearson Education.

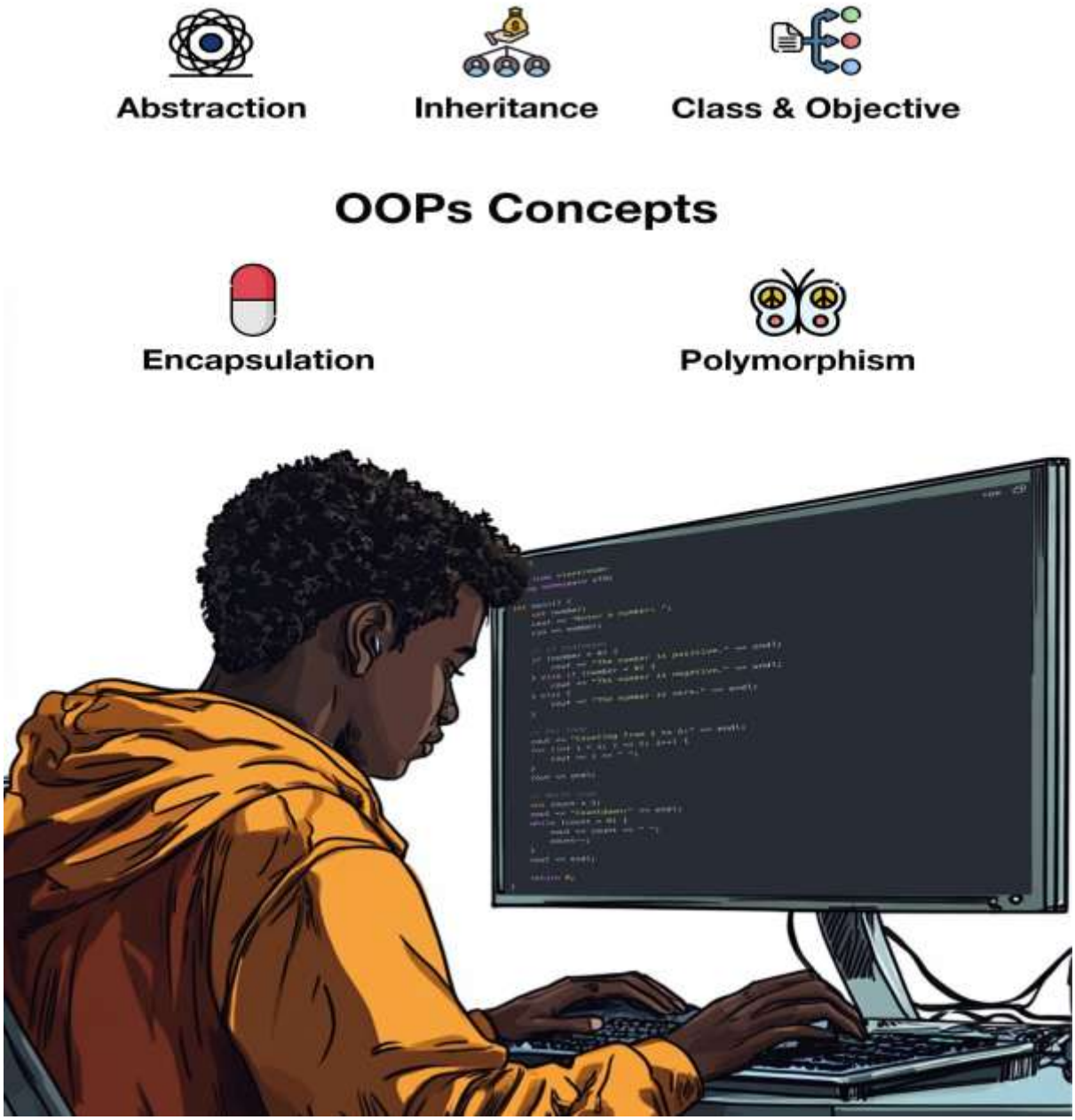
Simplilearn - C++ Basics: This guide provides an easy-to-understand introduction to C++ concepts.

Stroustrup, B. (2013). The C++ Programming Language (4th ed.). Addison-Wesley.
Useful links

W3Schools - C++ Tutorial: A comprehensive guide to learning C++ with interactive examples and exercises.

Walther, C. (2020). C++ Programming: A Practical Approach. Independently published.

West, C. (2018). C++ in 21 Days (5th ed.). Sams Publishing.


Abstraction
Inheritance
Class & Objective
OOPs Concepts
Encapsulation
Polymorphism

Indicative contents

2.1. Apply Class and object
2.2. Apply Inheritance and polymorphism
2.3. Apply Namespaces
2.4. Errors and Exceptions handling

Key Competencies for Learning Outcome 2: APPLY OOP CONCEPTS

Knowledge	Skills	Attitudes
• Description of OOP key terms • Description of Inheritance and Polymorphism • Description of Errors and Exceptions handling • Description of namespaces	• Applying class and object • Implementing Inheritance and polymorphism • Performing of Errors and Exceptions handling • Applying namespace	• Being cooperative • Being Hard working • Being creative and innovative • Being self _confident

Duration: 30hrs

Learning outcome 2 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe properly OOP key terms as used in C++ programming language
2. Apply properly Class and object based on the principles of object-oriented programming.
3. Describe correctly Inheritance and polymorphism based on the rules and principles of object-oriented programming.
4. Apply properly Inheritance and polymorphism based on the rules and principles of object-oriented programming.
5. Describe correctly Namespaces based on the rules and conventions
6. Apply properly Namespaces based on the rules and conventions
7. Describe correctly Errors and Exceptions according to the rules and practices of error handling
8. Apply effectively Errors and Exceptions according to the rules and practices of error handling

Resources

Equipment	Tools	Materials
Computer	• Integrated Development Environment (IDE)	• Internet

Advance Preparation:

Before delivering this learning outcome, you are recommended to:

- Avail didactic materials(flip charts, markers)
- Prepare learning place (Classroom/computer lab)

Indicative content 2.1: Apply Class and Object

Duration: 10 hrs

Theoretical Activity 2.1.1: Description of OOP Key Terms

Notes to the trainer:

- Trainer may avail the computer with installed Dev C++

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to answer the following questions

I. Define the following terms as used in C++ OOP:

a. Class

b. Object

c. Encapsulation

d. Data Abstraction

e. Constructors

f. Destructors

g. Inheritance

h. Polymorphism

i. Function Overloading

j. Function Overriding

II. Describe Class and Object

Step 2: Ask trainees to write their findings on paper/flipchart

Step 3: Ask trainees to present their findings to the whole class

Step 4: Present expert view

Step 5: Ask trainees to read the Key readings 2.1.1. in trainee's manual

Points to Remember

While working with OOP in C++, **h**ere are the main key terms of OOP concepts**:**

- ✓ **Class**: A blueprint for creating objects. It defines a data type by bundling data and methods that work on the data into one single unit.
- ✓ **Object**: An instance of a class. It is created from a class and can use the methods and properties defined in the class.
- ✓ **Encapsulation**: The concept of wrapping data and methods that operate on the data within a single unit, typically a class.
- ✓ **Data Abstraction**: The process of hiding the complex implementation details and showing only the
- ✓ **Constructor**: A special member function of a class that initializes objects of the class. It is called automatically when an object is created.
  - **Characteristics of constructor :**
    - Called automatically when an object is created.
    - Can be overloaded (multiple constructors with different parameters).
    - Cannot be inherited or virtual.
    - No return type, not even void.
- ✓ **Destructor**: A special member function of a class that cleans up when an object is destroyed.
  - **Characteristics of Destructor are**
    - Called automatically when an object goes out of scope or is explicitly deleted.
    - Cannot be overloaded (only one destructor per class).
    - No parameters and no return type.
    - Called in the reverse order of constructors.
- ✓ **Inheritance**: A mechanism where a new class (derived class) inherits properties and behavior (methods) from an existing class (base class). It promotes code reusability.
- ✓ **Polymorphism**: The ability of a function, object, or method to take on many forms. It allows methods to do different things based on the object it is acting upon, typically achieved through method overriding and

- ✓ **Function Overloading**: The ability to create multiple functions with the same name but different parameters. The correct function is called based on the arguments passed.
- ✓ **Function Overriding**: When a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden.
- ✓ **Advantages of Using OOP** (Object-Oriented Programming are the following Modularity**,** Reusability**,** Scalability**,** Maintainability and Flexibility

Practical Activity 2.1.2: APPLYING CLASS AND OBJECT

Notes to the trainer

- This activity should take place in computer lab
- Avail computers having installed Dev-C++

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to do the task below:
Create a BankAccount class to manage basic banking operations. The class should include private data members for account number, account holder's name, and balance. Implement public member functions: deposit() to add funds, withdraw() to deduct funds if sufficient(not less than 1000), and display() to show account details. Define the class and its functions, then create and test objects to ensure all functionalities work correctly.

Step 2: Demonstrate to trainees steps involved to carry out the above task

Step 3: Ask trainees to work out the task following demonstrated steps and monitor the activity.

Step 4: Ask trainees to check the result to ensure it is the same as the one expected

Step 5: Ask trainees to read the key readings **Key readings 2.1.2** in trainee's manual

Points to Remember

The following are the main points to Remember for Creating a BankAccount Class that implements class and object

- **Class Definition**:
  - ✓ Use the class keyword to define the BankAccount class.
  - ✓ Ensure proper encapsulation by declaring data members as private.
- **Data Members**:
  - ✓ Include private members:
    - accountNumber (e.g., int or string)
    - accountHolderName (e.g., string)
    - balance (e.g., double or float)
- **Member Functions**:
  - ✓ **deposit()**:
    - Accept an amount as a parameter.
    - Increase the balance by this amount.
  - ✓ **withdraw()**:
    - Check if the withdrawal amount is less than or equal to the current balance minus 1000.
    - Deduct the amount only if sufficient funds are available; otherwise, display an error message.
  - o **display()**:
    - Print account details: account number, holder's name, and current balance.
- **Input Validation**:
  - ✓ Ensure that the deposit and withdrawal amounts are positive.
  - ✓ Handle potential invalid inputs (e.g., negative amounts).
- **Object Creation and Testing**:
  - ✓ Create instances of the BankAccount class to represent different accounts.

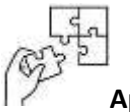
- ✓ Test all member functions to ensure they work as intended, including edge cases for deposits and withdrawals.

- **Code Clarity and Documentation**:

  - ✓ Write clear, well-organized code with meaningful variable names.
  - ✓ Add comments to explain the logic and functionality of each member function.

- **Error Handling**:

  - ✓ Implement appropriate error handling for invalid operations, such as attempting to withdraw an amount greater than the available balance minus 1000.



Application of learning 2.1:

Create a class called Person with private member variables for name, age, and country. Implement member functions to set and get the values of these variables.

SOLUTION:

```
#include <iostream>
#include <string>
class Person {
private:
std::string name;
int age;
std::string country;
public:
// Setter functions
void setName(const std::string &n) {
name = n;
}
void setAge(int a) {
```

```
age = a;
}
void setCountry(const std::string &c) {
country = c;
}
// Getter functions
std::string getName() const {
return name;
}
int getAge() const {
return age;
}
std::string getCountry() const {
return country;
}
};
int main() {
Person person;
// Set the person's details
person.setName("John Doe");
person.setAge(30);
person.setCountry("USA");
// Get and display the person's details
std::cout << "Name: " << person.getName() << std::endl;
std::cout << "Age: " << person.getAge() << std::endl;
std::cout << "Country: " << person.getCountry() << std::endl;
return 0;
}
```

Checklist:

SN	Criteria	Indicators	Score	
			Yes	No
1	**Class Person is correctly defined**	Confirm class Person is defined		
2	**Private Member Variables are correctly used**	private variables name, age, and country are initiated		
3	**Constructor is properly Implemented**	constructor initializes name, age, and country are initiated		
4	**Set Functions are correctly used**	void setName(string name), void setAge(int age), and void setCountry(string country) are implemented		
5	**Get Functions are correctly used**	string getName(), int getAge(), and string getCountry() are implemented		
6	**member variables are properly accessed**	member variables are accessed through member functions		
7	**Main Function is properly used**	int main() function demonstrates setting and getting values are used		
8	**Program Output is correctly displayed**	Output is displayed		

Indicative content 2.2: Apply Inheritance and Polymorphism

Duration: 8hours

Theoretical Activity 2.2.1: Description of Inheritance and polymorphism

Notes to the trainer:

- Trainer may use small groups to describe inheritance and polymorphism
- Trainer may use didactic materials and videos

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Ask trainees to read the task below

1. Describe the following terms
   a) Inheritance
   b) Polymorphism
2. Discuss the following:
   a) Base class
   b) Access specifier
   c) Derived class
3. Differentiate Function Overriding from function Overloading

Step 2: Ask trainees to write their findings on papers/flipcharts

Step 3: Ask trainees to present their findings to the whole class

Step 4: Present your expert view

Step 5: Ask trainees to take key notes and ask for clarification if any

Step 6: Ask trainees to read the **Key readings 2.2.1**. in trainee's manual

Points to Remember

Here are key points to remember for describing inheritance and polymorphism

- ✓ **Inheritance** refers to a mechanism in object-oriented programming where a new class (derived class) acquires properties and behaviours (methods) from an existing class (base class). It promotes code reusability and establishes a relationship between classes.It can take different forms such as: Single, multiple, hierarchical, multilevel, and hybrid inheritance.

- ✓ **Polymorphism** is the ability of different objects to respond to the same method call in different ways. **It can be Compile-time (Static) Polymorphism**; Achieved through method overloading or operator overloading, orr**Run-time (Dynamic) Polymorphism**; achieved through method overriding, allowing a derived class to provide a specific implementation of a method that is already defined in its base class.

- **Base Class is t**he class from which other classes (derived classes) inherit properties and methods. It contains common attributes and behaviours that can be shared with derived classes.
- **Derived Class is a** class that inherits from one or more base classes. **It c**an access public and protected members of the base class; can also override methods to provide specific functionality.
- **Access Specifier** is keyword that sets the accessibility of classes, methods, and other members. **This can be:**
  - ✓ **Public**: Accessible from any other class.
  - ✓ **Private**: Accessible only within the class itself.
  - ✓ **Protected**: Accessible within the class and by derived classes.
- ✓ **Function Overriding** is a process of redefining a method in a derived class that already exists in the base class with the same name and signature and it is used to achieve dynamic polymorphism; allowing the derived class to provide a specific implementation.
- ✓ **Function Overloading** is creating multiple methods with the same name but different signatures (parameter types or number) within the same class. This allows to achieve compile-time polymorphism; allowing methods to handle different types or numbers of inputs.

Practical Activity 2.2.2: Applying Inheritance and Polymorphism

Notes to the trainer

- This activity should take place in computer lab
- Avail computers having installed Dev-C++

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Ask trainees to read the task related to how Implement Inheritance:
Develop a vehicle management system for a car rental company to manage cars, trucks, and motorcycles. Create a base class Vehicle with common attributes and methods, and implement derived classes Car, Truck, and Motorcycle with specific attributes and overridden displayInfo() methods. Demonstrate polymorphism by creating a list of Vehicle pointers and invoking their methods to show their specific behaviors.

Step 2: Demonstrate steps that involve in performing the above task

Step 3: Ask trainees to perform the task by following demonstrated steps and monitor the activity

Step 4: Ask trainees to check the expected result

Step 5: Ask trainees to tread the Key readings 2.2.2. in trainee's manual

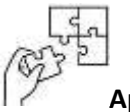
Points to Remember

Implementing inheritance and polymorphism

Here's a list of key points to remember for solving the vehicle management system task:

- **Define the Base Class**:
    - ✓ Name: Vehicle
    - ✓ Common Attributes: make, model, year
    - ✓ Common Methods: start(), stop(), displayInfo()

- **Create Derived Classes**:
    - ✓ **Car**:
        - Additional Attribute: numberOfDoors
        - Override displayInfo() to include door information.
    - ✓ **Truck**:
        - Additional Attribute: payloadCapacity
        - Override displayInfo() to include capacity details.
    - ✓ **Motorcycle**:
        - Additional Attribute: type (e.g., cruiser, sport)
        - Override displayInfo() to include motorcycle type.
- **Implement Polymorphism**:
    - ✓ Use a vector (or list) of pointers to Vehicle to hold different vehicle objects (cars, trucks, motorcycles).
    - ✓ Iterate through the list and call start(), stop(), and displayInfo() methods.
- **Testing**:
    - ✓ Create instances of each derived class and add them to the vehicle list.
    - ✓ Ensure each method works as expected by verifying outputs during iteration.



Application of learning 2.2.

Define a base class Animal with attributes like name and age, along with methods such as speak() and move(). Then, create derived classes Dog and Cat that inherit from Animal, adding specific attributes like breed and color. Override the speak() method in each class to return "bark" for Dog and "meow" for Cat. Finally, instantiate Dog and Cat objects to demonstrate their inherited properties and methods.

Solution:

```
#include <iostream>
using namespace std;
// Base class
class Animal {
public:
virtual void makeSound() {
cout << "The animal makes a sound" << endl;
}};
// Derived class Dog
class Dog : public Animal {
public:
void makeSound() override {
cout << "The dog says: bow wow" << endl;
}};
// Derived class Cat
class Cat : public Animal {
public:
void makeSound() override {
cout << "The cat says: meow" << endl;
}};
// Function to demonstrate polymorphism
void demonstratePolymorphism(Animal* animal) {
animal->makeSound();}
int main() {
Animal* myDog = new Dog();
Animal* myCat = new Cat();
demonstratePolymorphism(myDog);
demonstratePolymorphism(myCat);
delete myDog;
delete myCat;
return 0;
}
```

Checklist:

SN	Criteria	Indicator	Score	
			Yes	No
1	Base Class is correctly Defined	class Animal with attributes name and age are initiated		
2	Base Methods are properly Implemented	void speak() and void move() methods in Animal is are initiated		
3	Derived Class Dog is properly defined	class Dog : public Animal with additional attributes breed and color are used		
4	Derived Class Cat is properly defined	class Cat : public Animal with additional attributes breed and color are created		
5	Override Speak Method in Dog is properly used	void speak() override in Dog returns "bark" are used		
6	Override Speak Method in Cat is correctly used	void speak() override in Cat returns "meow" are used		
7	Dog Object is correctly instantiated	Dog object is created		
8	Cat Object is correctly instantiated	Cat object is created		
9	Inheritance is correctly demonstrated	Dog and Cat objects demonstrate inherited properties and methods are used		
10	Program Output is correctly displayed	overridden speak() methods are displayed		

Indicative content 2.3: Apply Namespaces

Duration: 5 hrs

Theoretical Activity 2.3.1: Description of namespace

Notes to the trainer:

- Trainer may use small groups to describe namespace in C++
- Trainer may use didactic materials and videos

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to answer the following questions

1) What is a namespace in C++
2) What is the correct syntax for defining a namespace in C++?
3) What are the rules for using namespaces in C++?
4) Discuss how the using directive affects the visibility of names within a namespace

Step 2: Ask trainees to write their findings on papers/flipcharts

Step 3: Ask trainees to present their findings

Step 4: Present expert view

Step 5: Ask trainees to read the **Key readings 2.3.1.** in trainee's manual

Points to Remember

The following are points to remember while working with namespace in C++:

- A namespace is a declarative region for organizing code and grouping related identifiers (variables, functions, classes).
- It prevents naming conflicts, allowing the same name to be used in different namespaces without collision.
- **Syntax**: Defined using the namespace keyword followed by the name and enclosed in curly braces, e.g.,
  - ✓ namespace NamespaceName { /* Declarations */ }

- **Rules for Use**:
  - ✓ Access identifiers with the scope resolution operator (::).
  - ✓ Support for nested namespaces for further organization.
  - ✓ Use of using keyword to bring names into the current scope, though it may lead to conflicts.
- **Using Directive**: The using directive (e.g., using namespace NamespaceName;) makes all names in that namespace available without qualification.

Practical Activity 2.3.2: Applying namespace

Notes to the trainer

- This activity should take place in computer lab
- Avail computers having installed Dev-C++

Key steps:

While delivering this activity, pass through the following

Step 1: Introduce activity and highlight the expected results from the following task: Create a namespace called MathOperations that contains two functions: add for summing two integers and subtract for finding their difference. In the main function, prompt the user for two integers and utilize the add and subtract functions to perform calculations and display results.

Step 2: Demonstrate to trainees steps that involve in carrying out the above task.

Step 3: Ask trainees to work out the task following demonstrated steps and monitor the activity.

Step 4: Ask trainees to check the output making sure they are the same as expected results.

Step 5: Ask trainees to read the key readings **2.3.2** in trainee's manual

Points to Remember

Here are the basic steps about creating and using the MathOperations namespace

with add and subtract functions:

- **Namespace Definition:**

The namespace MathOperations is defined using the namespace keyword followed by the namespace name.

- ✓ It is to be noted that, there is no semicolon (;) after the closing brace.
- ✓ To call the namespace-enabled version of either function or variable, prepend the namespace name as follows:
- ✓ namespace_name: :code; // code could be variable , function or class.

- **Function Definitions:**

add(int a, int b): This function takes two integers as parameters and returns their sum.

subtract(int a, int b): This function takes two integers as parameters and returns the result of subtracting the second integer from the first.

- **Encapsulation:**

The namespace encapsulates the add and subtract functions, grouping them logically under MathOperations.

- **Using Namespace Functions:**

To call the functions within the namespace, you need to use the scope resolution operator **::** like **MathOperations::add** and **MathOperations::subtract.**

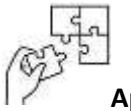
- **Main Function:**

The main function demonstrates how to use the add and subtract functions from the MathOperations namespace.

It includes examples of adding and subtracting two integers and prints the results.

- **Code Structure:**

The code is structured to include the namespace definition, function implementations, and a main function to test the operations.



Application of learning 2.3.

As a Computer Technician ,Create two namespaces NamespaceA and NamespaceB, each containing a function printMessage that prints different messages.

Write a main function to call both functions without causing name conflicts

Solution:

```
#include <iostream>
// Define NamespaceA
namespace NamespaceA {
void printMessage() {
std::cout << "Hello from NamespaceA!" << std::endl;
}}
// Define NamespaceB
namespace NamespaceB {
void printMessage() {
std::cout << "Hello from NamespaceB!" << std::endl;
}  }
int main() {
// Call printMessage from NamespaceA
NamespaceA::printMessage();
// Call printMessage from NamespaceB
NamespaceB::printMessage();
return 0;  }
```

Checklist:

SN	Criteria	Indicators	Score	
			Yes	No
1	Header is properly Includes	#include <iostream> is included		
2	Namespace A is clearly defined	NamespaceA is implementated		
3	Namespace A Function is properly used	Void NamespaceA::printMessage() is printed		
4	Namespace B is clearly defined	NamespaceB implementated		
5	Namespace B Function is is clearly defined	void NamespaceB::printMessage() is printed		
6	Main Function is properly used	int main() function is used		
7	NamespaceA Function Call is correctly implemented	NamespaceA::printMessage(); is called in main()		
8	NamespaceB Function Call is correctly implemented	NamespaceB::printMessage();is called in main()		
9	Program Output is displayed correctly	output: "Hello from NamespaceA!" and "Hello from NamespaceB!" are used		

Indicative content 2.4: Apply Error and Exception handling

Duration: 7hrs

Theoretical Activity 2.4.1: Description of Error and Exception handling

Notes to the trainer:

- Trainer may use small groups to describe namespace in C++
- Trainer may use didactic materials and videos

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to answer the following questions:

1. Define the term bellow:

   a) Errors
   b) Exception

2. Describe the types the following:

   a) Types of errors
   b) Exception handling mechanisms

Step 2: Ask trainees to write their findings on paper/flipchart
Step 3: Ask trainees to present their findings.
Step 4: Provides expert view.
Step 5: Ask trainees to read the key readings 2.4.1 in trainee's manual

Points to Remember

Errors and exceptions handling

While working on errors and handling exceptions in C++ note the following:

- Errors refer to issues that occur during program execution, causing it to behave unexpectedly or crash. They can arise from syntax mistakes, logical flaws, or resource unavailability. Errors can be broadly classified into compile-time and runtime errors.
- An exception is an unexpected event that disrupts the normal flow of a program during execution. Exceptions are objects that encapsulate error conditions and can be thrown and caught to handle these issues gracefully, allowing the program to recover or terminate cleanly.
- **Types of Errors**:

    - ✓ **Syntax Errors**: Mistakes in the code structure that prevent the program from compiling (e.g., missing semicolons).
    - ✓ **Runtime Errors**: Errors that occur during program execution, such as division by zero or accessing invalid memory.
    - ✓ **Logical Errors**: Flaws in the program's logic that lead to incorrect results, despite the program compiling and running without crashing.

- **Exception Handling Mechanisms**:

    - ✓ **Try-Catch Blocks**: Code that may throw exceptions is placed within a try block, and potential exceptions are caught and handled in corresponding catch blocks.
    - ✓ **Throwing Exceptions**: The throw keyword is used to signal an exception when an error condition occurs, transferring control to the nearest catch block.
    - ✓ **Finally Blocks (not native to C++)**: While not present in C++, similar functionality can be achieved through destructors or cleanup code to ensure resources are released regardless of whether an exception occurs.

Practical Activity 2.4.2: Applying Error and Exception handling

Notes to the trainer

- This activity should take place in computer lab
- Avail computers having installed Dev-C++

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Ask trainees to read and perform the task below: Create a C++ program that safely performs division using a function safeDivide, which throws an exception for division by zero. Prompt the user for input in the main function and use a try block to call safeDivide. Handle any exceptions by displaying an error message, or print the division result if successful

Step 2: Demonstrate to the trainees the steps perform the above task

Step 3: Ask trainees to carry out the task following demonstrated steps and monitors the activity.

Step 4: Ask trainees to check the outputs whether they match the expected results.

Step 5: Ask trainees to read the key readings **2.4.2** in trainee's manual

Points to Remember

In order to do the above task, here are key points to remember for carrying out the task of creating a C++ program that safely performs division:

Key Points to Remember

- **Define the Function**:
    - ✓ Create a function named safeDivide(double numerator, double denominator) that performs division.
    - ✓ Check if the denominator is zero. If it is, throw an exception (e.g., std::runtime_error).

- **Implement Exception Handling**:

    - ✓ Use a try block in the main function to call safeDivide().
    - ✓ Surround the division call with exception handling to catch errors.

- **User Input**:

    - ✓ Prompt the user to enter two numbers: the numerator and the denominator.
    - ✓ Use std::cin to read the input values and ensure they are of type double.

- **Error Handling**:

    - ✓ Catch exceptions using a catch block.
    - ✓ Display a user-friendly error message if an exception is caught, indicating that division by zero is not allowed.

- **Output the Result**:

    - ✓ If no exceptions occur, print the result of the division.
    - ✓ Ensure that the output is clear and easy to understand.

- **Testing**:

    - ✓ Test the program with various inputs, including valid divisions, division by zero, and non-numeric inputs to ensure robustness.
    - ✓ Consider edge cases, such as very large or very small numbers.

- **Code Clarity**:

    - ✓ Keep the code well-organized and commented to explain the logic, especially in error handling sections.
    - ✓ Use meaningful variable names for clarity.

- **Compilation and Execution**:

    - ✓ Ensure the program compiles without errors.
    - ✓ Run the program in an environment where user input can be tested interactively.

Application of learning 2.4:

Create a program for an employee management system with a base class Employee that has attributes like name and salary, and a virtual method calculateBonus(). Implement derived classes Manager and Developer, each overriding calculateBonus() for their specific bonus calculations. Include error handling for scenarios such as invalid salary values and incorrect employee types, using exceptions to manage errors effectively. This task highlights practical applications of error and exception handling within an inheritance structure in a real-world context.

Solution :

```
#include <iostream>
#include <stdexcept>
#include <string>
class Employee {
protected:
std::string name;
double salary;
public:
Employee(const std::string& name, double salary) {
if (salary < 0) {
throw std::invalid_argument("Salary cannot be negative.");
}
this->name = name;
this->salary = salary;
}
virtual double calculateBonus() const = 0; // Pure virtual method
virtual ~Employee() = default; // Virtual destructor
};
class Manager : public Employee {
```

```
public:
Manager(const std::string& name, double salary) : Employee(name, salary) {}
double calculateBonus() const override {
return salary * 0.10; // 10% bonus for managers
}
};
class Developer : public Employee {
public:
Developer(const std::string& name, double salary) : Employee(name, salary) {}
double calculateBonus() const override {
return salary * 0.05; // 5% bonus for developers
}
};
int main() {
try {
Employee* emp1 = new Manager("Alice", 80000);
Employee* emp2 = new Developer("Bob", 60000);
std::cout << emp1->calculateBonus() << std::endl; // Output: 8000
std::cout << emp2->calculateBonus() << std::endl; // Output: 3000
delete emp1;
delete emp2;
} catch (const std::invalid_argument& e) {
std::cerr << "Error: " << e.what() << std::endl;
} catch (const std::exception& e) {
std::cerr << "General error: " << e.what() << std::endl;
}
return 0;
}
```

Checklist

SN	Criteria	Indicators	Score	
			Yes	No
1	Headers properly Included	#include <iostream>, #include <stdexcept>, #include <string> are included		
2	Classes are clearly defined	EEmployee, Manager, Developer classes are defined		
3	Constructor Input are properly Validated	if (salary < 0) condition is checked		
4	Pure Virtual Method is properly implemented	virtual double calculateBonus() const = 0 is implemented		
5	Inheritance correctly Established	class Manager : public Employee are validated		
6	Override Methods is clearly used	double calculateBonus(), const override are used		
7	Exceptions correctly handled	presence of try, catch blocks are checked		
8	Dynamic memory is clearly allocated	new Manager, new Developer are Confirmed		
9	Memory is Properly deallocated	delete emp1; delete emp2; statements are Verified		
10	Output is correctly displayed	std::cout << statements for bonus output is used		

Learning outcome 2 end assessment

Theoretical assessment

I. Referring from the contents about Apply OOP concepts in C++ focusing on class, object, inheritance, polymorphism, namespaces, error handling and exception handling, answer the following questions by **True** or **False**:

1. In C++, an object is an instance of a class.
   Answer: True.
2. Inheritance allows one class to acquire the properties and behavior of another class.
   Answer: True.
3. In C++, runtime polymorphism can be achieved using function overloading.
   Answer: False. (Function overloading is a form of compile-time polymorphism; runtime polymorphism is achieved through virtual functions and inheritance.)
4. A class constructor in C++ can have a return type.
   Answer: False. (Constructors do not have return types.)
5. Namespaces in C++ help avoid name conflicts by encapsulating identifiers.
   Answer: True.
6. A virtual function must be redefined in all derived classes to enable polymorphism.
   Answer: False. (Redefining virtual functions is optional in derived classes, but the base class version will be used if not redefined.)
7. When a derived class inherits privately from a base class, all public members of the base class become private in the derived class.
   Answer: True.
8. In C++, exceptions are handled using try, catch, and finally blocks.
   Answer: False. (C++ uses try, catch, and throw blocks. There is no finally in C++, though destructors often serve this purpose.)
9. The standard C++ exception handling mechanism guarantees that every exception will be caught.
   Answer: False. (Exceptions must be explicitly caught by appropriate catch blocks.)
10. If a function is declared inside a namespace, you must use the namespace qualifier to access it unless using a using directive.
    Answer: True.

11. Can derived class access private members of its base class directly.

Answer: false

II. Choose the best answer by circling the letter that fits:

1. Which of the following correctly defines an object in C++?
   a) A method of a class
   b) An instance of a class
   c) A static variable
   d) A friend function
   Answer: b) An instance of a class

2. What is the main purpose of a constructor in C++?
   a) To destroy an object when it goes out of scope
   b) To initialize an object when it is created
   c) To allocate memory for an object
   d) To return values from an object
   Answer: b) To initialize an object when it is created

3. In C++, which type of inheritance allows a derived class to inherit from more than one base class?
   a) Single inheritance
   b) Multiple inheritance
   c) Hierarchical inheritance
   d) Multilevel inheritance
   Answer: b) Multiple inheritance

4. How run-time polymorphism is typically achieved in C++?
   a) Function overloading
   b) Operator overloading
   c) Virtual functions
   d) Friend functions
   Answer: c) Virtual functions

5. What is the primary purpose of using namespaces in C++?
   a) To group constants together
   b) To avoid name conflicts in large programs
   c) To increase runtime performance
   d) To create abstract classes
   Answer: b) To avoid name conflicts in large programs

6. In C++, what happens to the public members of a base class when inherited privately?
   a) They remain public in the derived class
   b) They become protected in the derived class
   c) They become private in the derived class
   d) They are not inherited at all
   Answer: c) They become private in the derived class

7. Which of the following is used to throw an exception in C++?
   a) catch
   b) throw
   c) try
   d) error
   Answer: b) throw

8. Which of the following statements about virtual functions in C++ is correct?
   a) A virtual function cannot be overridden
   b) A virtual function can only be used in abstract classes
   c) A virtual function is defined in a base class and can be overridden in a derived class
   d) A virtual function must return an integer
   Answer: c) A virtual function is defined in a base class and can be overridden in a derived class

9. If you have a function inside a namespace, how can you access it without using the namespace prefix every time?
   a) By declaring the function as static
   b) By using the using directive
   c) By re-declaring the function globally
   d) By using a friend function
   Answer: b) By using the using directive

10. In C++, what happens if an exception is thrown but not caught by any catch block?
    a) The program terminates with an error message
    b) The program continues to run without interruption
    c) The exception is ignored
    d) The exception is automatically handled by the operating system
    Answer: a) The program terminates with an error message

III. Complete sentences that follow using one of the words: **catch,throw, virtual, class,inherit, multiple, initialize,conflicts,private, compile**

a. An object is an instance of a _____.
   Answer: class
b. A constructor is used to _____ an object.
   Answer: initialize
c. Inheritance allows a class to _____ properties from another class.
   Answer: inherit
d. Runtime polymorphism is achieved using _____ functions.
   Answer: virtual
e. Namespaces are used to avoid name _____.
   Answer: conflicts
f. The keyword used to handle exceptions in C++ is _____.
   Answer: catch
g. A class that inherits from more than one base class exhibits _____ inheritance.
   Answer: multiple
h. When a class is inherited privately, public members of the base class become _____ in the derived class.

   Answer: private
i. Function overloading is a form of _____-time polymorphism.
   Answer: compile
j. Exceptions are thrown using the _____ keyword.
   Answer: throw

IV. Answer the following questions:

a. What is the relationship between a class and an object in C++?
   Answer: A class is a blueprint, and an object is an instance of a class.
b. What is the role of a constructor in C++?
   Answer: A constructor is used to initialize objects of a class when they are created.
c. How does inheritance promote code reusability?
   Answer: Inheritance allows a derived class to reuse properties and methods from a base class, reducing code duplication.
d. How is runtime polymorphism achieved in C++?
   Answer: Runtime polymorphism is achieved through virtual functions and inheritance.
e. What is the purpose of using namespaces in C++?
   Answer: Namespaces are used to organize code and prevent naming conflicts between identifiers like variables, classes, and functions.

f. Which keywords are used for exception handling in C++?
Answer: The keywords used are try, catch, and throw.

g. What is multiple inheritance in C++?
Answer: Multiple inheritance is when a class inherits from more than one base class.

h. What happens to the public members of a base class when it is inherited privately?
Answer: The public members become private in the derived class.

i. What is function overloading in C++?
Answer: Function overloading is the ability to define multiple functions with the same name but different parameter lists.

j. What happens if an exception is thrown but not caught in C++?
Answer: The program terminates and may display an error message.

V. Match the following OOP Terms with their corresponding meanings

Answers	OOP Terms	Meaning
……D	1. Class	A. An instance of a class.
……A	2. Object	B. A special member function of a class that initializes objects of the class.
…….F	3. Encapsulation	C. A mechanism where a new class (derived class) inherits properties and behavior (methods) from an existing class (base class).
……G	4. Data Abstraction:	D. A blueprint for creating objects.
……B	5. Constructor:	E. A special member of a class that cleans up when an object is destroyed.

……..E	6. Destructor	F. The concept of wrapping data and methods that operate on the data within a single unit
…….C	7. Inheritance:	G. The process of hiding the complex implementation details and showing only the necessary features of an object
…….K	8. Polymorphism	H. The ability to create multiple functions with the same name but different parameters
…….H	9. Function Overloading	I. When a derived class has a definition for one of the member functions of the base class.
……I	10. Function Overriding	J. Is a member function that does not modify any member variables of the class.
…….J	11. A constant function	K. The ability of a function, object, or method to take on many forms
		L. It define the access level of class members.

Practical assessment

Scenario 1: Library Management System

As a computer technician design a simple Library Management System that will help librarian to manage the following management books, members, and transactions by using OOP principles like classes, inheritance, polymorphism, and encapsulation:

SOLUTION

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
class Book {
private:
string title;
string author;
string ISBN;
int copiesAvailable;

public:
Book(string t, string a, string isbn, int copies) : title(t), author(a), ISBN(isbn),
copiesAvailable(copies) {}

void getDetails() {
cout << "Title: " << title << ", Author: " << author << ", ISBN: " << ISBN << ", Copies Available:
" << copiesAvailable << endl;
}

void updateCopies(int count) {
copiesAvailable += count;
}
};

class Person {
protected:
string name;
string email;
public:
Person(string n, string e) : name(n), email(e) {}

void getDetails() {
cout << "Name: " << name << ", Email: " << email << endl;
}
};
class Member : public Person {
private:
int memberID;
```

```
vector<Book> borrowedBooks;

public:
Member(string n, string e, int id) : Person(n, e), memberID(id) {}

void borrowBook(Book &book) {
borrowedBooks.push_back(book);
book.updateCopies(-1);
}
void returnBook(Book &book) {
for (auto it = borrowedBooks.begin(); it != borrowedBooks.end(); ++it) {
if (it->getISBN() == book.getISBN()) {
borrowedBooks.erase(it);
book.updateCopies(1);
break;
}
}
}
};

class Library {
private:
vector<Book> books;
vector<Member> members;

public:
void addBook(Book book) {
books.push_back(book);
}

void addMember(Member member) {
members.push_back(member);
}

void issueBook(int memberID, string ISBN) {
// Implementation for issuing a book
}

void returnBook(int memberID, string ISBN) {
// Implementation for returning a book
}
```

```
void searchBook(string title) {
for (auto &book : books) {
if (book.getTitle() == title) {
book.getDetails();
return;
}
}
cout << "Book not found!" << endl;
}
};
int main() {
Library library;
Book book1("The Great Gatsby", "F. Scott Fitzgerald", "123456789", 5);
Member member1("John Doe", "john@example.com", 1);
library.addBook(book1);
library.addMember(member1);
library.searchBook("The Great Gatsby");
return 0;
}
```

Checklist

SN	Criteria	Indicator	Score	
			Yes	No
1	Classes are defined properly	1.1 class Book is defined		
		1.2 class Member is defined		
		1.3 class Transaction is defined		
2	Class Members are clearly Implemented	Book		
		2.1. Attributes: title, author, id are used		
		2.2. Methods: Constructor, Getters, Setter are used		
		Member		
		2.3. Attributes: name, memberId are used		
		2.4. Methods: Constructor, displayDetails() are used		
		Transaction		

		2.5.Attributes: transactionId, bookId, memberId, date are used		
		2.6. Methods: Constructor is used		
3	Inheritance is properly demonstrated	3. 1. Create PremiumMember class inheriting from Member is used		
		3. 2. Additional Attribute: rewardPoints are used		
		3. 3. Additional Methods: addPoints(), Override displayDetails() are used		
4	Polymorphism is properly implemented	Use virtual functions to allow displayDetails() to be overridden in PremiumMember are used		
5	Encapsulation effectively applied	5.1. Use private access specifiers for class attributes are used		
		5.2 . Provide public getter and setter methods are used		
6	Main Function is properly declared	6.1. Create instances of Book, Member, and PremiumMember are used		
		6.2. Demonstrate usage of methods and polymorphism is used		
7	Error Handling is properly implemented	Implement error handling for invalid inputs or operations are used		
8	User Interface is properly designed	A simple user interface for interacting with the system is designed		
9	Data Storage is prepared properly	how data will be stored and retrieved (e.g., using files or databases) is considered		

Further information to the trainer

Bjarne, S. (2014). *The C++ programming language* (4th ed.). Addison-Wesley.

Bjarne, S. (2018). *A tour of C++* (2nd ed.). Addison-Wesley.

C++ Standards Committee. (2014). *ISO/IEC 14882:2014 - Programming languages — C++. International Organization for Standardization*.

Deitel, P. J., & Deitel, H. M. (2015). *C++: How to program* (10th ed.). Pearson.

Eckstein, D., & Heller, R. (2007). *C++ programming for the absolute beginner*. Course Technology.

Effective C++ (2019). *Polymorphism in C++: A Tutorial*. Retrieved from https://www.effectivecpp.com/polymorphism-in-c/

Ellis, P., & Stroustrup, B. (2014). *The Annotated C++ Reference Manual*. Addison-Wesley.

Hsu, W. (2018). *Object-oriented programming with C++*. Oxford University Press.

Khosravi, R., & Huang, D. (2017). *Error and exception handling in C++: Best practices*. IEEE Transactions on Software Engineering, 43(5), 1110-1122.

Koenig, A., & Moo, B. E. (2005). *C++ Templates: The Complete Guide*. Addison-Wesley.

Lippman, S. B., Lajoie, J., & Moo, B. E. (2012). *C++ primer* (5th ed.). Addison-Wesley.

Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media.

McKinsey, M. (2020). *Namespaces in C++: A Comprehensive Guide*. Journal of Software Engineering, 12(4), 45-59.

Meyers, S. (2014). *Effective C++: 55 specific ways to improve your programs and designs* (3rd ed.). Addison-Wesley.

Miller, S. (2019). *Understanding Exception Handling in C++*. Retrieved from https://www.geeksforgeeks.org/exception-handling-c/

Roberts, D. (2016). *Understanding C++: An Introduction to Object-Oriented Programming*. Cengage Learning.

Sedgewick, R., & Wayne, K. (2011). *Algorithms in C++* (3rd ed.). Addison-Wesley.

Sia, J. (2021). *C++ Inheritance: A Beginner's Guide*. O'Reilly Media

Stroustrup, B. (2013). *Programming: Principles and practice using C++* (2nd ed.). Addison-Wesley.

Sutherland, I. E. (2009). *C++: An Introduction to Object-Oriented Programming*. Cengage Learning.

Useful links

Understanding Encapsulation, Inheritance, Polymorphism, Abstraction in OOPs - GeeksforGeeks
Inheritance, Polymorphism, and Abstract Classes - Javanotes
Inheritance and Polymorphism in Java - Syntax Savvy
Inheritance and Polymorphism – Programming Fundamentals
Inheritance and Polymorphism in Java - CodingDrills
Namespaces in C++ - GeeksforGeeks
C++ Namespaces - W3Schools
Understanding Namespaces in C++ - Tutorialspoint
Classes and Objects in C++ - GeeksforGeeks
C++ Classes and Objects - W3Schools
Classes and Objects in C++ - Tutorialspoint
Exception Handling in C++ - GeeksforGeeks
C++ Exception Handling - W3Schools
Error Handling in C++ - Tutorialspoint

Indicative contents
3.1 Apply pointers **3.2.Apply file handling** **3.3. Apply multithreading and concurrency** **3.4. Apply inline assembly**

Key Competencies for Learning Outcome 3: Perform CPU Optimization

Knowledge	Skills	Attitudes
• Description of pointer • Declaration of pointer • Initialization of pointer • Description of file • Description of header file • Description of fstream class • Definition of assembly language • Description of inline assembly	• Declaring and Initializing the pointer • Dereferencing pointer • Calling function with pointer • Creating threads • Manage threads • Applying thread synchronization • Implementing thread pool • Applying parallel algorithm • Implementing Parallel Data Processing • Implementing Task-Based Parallelism • Implementing Load Balancing • Applying inline assembly • Using register constraints	• Being self-motivated • Being critical thinker • Being detailed oriented • Being skilful • Being creative

Duration:20 hrs

Learning outcome 3 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe properly pointer in C++ programming language
2. Differentiate correctly the pointer from standard variables
3. Declare properly the pointer inline with C++ language
4. Us correctly a pointer as function argument during its call
5. Implement properly the dynamic memory allocation in C++
6. Describe properly the deadlock and race conditions in multithreading context
7. Implement correctly thread pool during concurrency process
8. Apply correctly parallel algorithm in multithreading context
9. Apply correctly inline assembly using C++ programming language

Resources

Equipment	**Tools**	**Materials**
▪ Computer ▪ Storage device	▪ Integrated Development Environment (IDE)	▪ Online Tutorials ▪ Internet Connection

Advance Preparation:

Before delivering this learning outcome, you are recommended to:

- Prepare learning place (Classroom/computer lab)

Indicative content 3.1: Apply pointers

Duration: 4hrs

Theoretical Activity 3.1.1: Description of pointers

Notes to the trainer:

- Trainer may use small group for describing development environment of C++.
- The use of video as didactic materials is required

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to answer the following questions

1. What do you mean by pointer in C++?
2. Describe the following :
    a) Memory address
    b) Dereferencing a pointer
    c) Memory leak
    d) Smart pointers
3. How do you declare pointer in C++?
4. Discuss the relationship between variables and pointers in C++.

Step 2: Ask trainees to note the essential of their findings on paper/flipchart

Step 3: Ask trainees to present their findings

Step 4: Provide expert view.

Step 5: Ask trainees to read the key readings **3.1.1** in trainee's manual

Points to Remember

Here bellow is a summary of key points that should help you understanding effectively the use of pointers and dynamic memory allocation in your C++ programs.

A pointer as a variable that stores the memory address of another variable has to be declared before being used. To declared using the following **syntax:**

data_type *pointer_name; (e.g., int *p;)

It is common knowledge that to access values stored in the pointer is referred to as dereferencing a pointer. The pointer can point to invalid address, so it is called null pointer.

Initialize a pointer to **nullptr** or valid address to avoid undefined behaviours. You can perform arithmetic operations on pointers, but they are interpreted in the terms of size of data type they point to. This action is referred to as pointer arithmetic.

The primary purpose of dynamic memory allocation is allocating a memory at runtime based on the program needs using the new operator to allocate and the delete to de-allocate it. You can also use the **new[]** and **delete**[] operator to allocate and de-allocate array pointers For memory management; avoid memory leaks by de-allocating memory when it is no longer needed. You can also use smart pointers to automatically manage the memory.

Practical Activity 3.1.2: Applying pointers

Notes to the trainer

- The trainer may use demonstration methods to show to trainees all steps passed through while carrying out the activity

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to perform the following task,
As a firmware development technician you are tasked to Write and Run a C++ Program for managing student grades using dynamic memory and pointer**s**

Step 2: Demonstrate to trainees the process of applying arrays in C++ by explaining step by step.

Step 3: Ask trainees compile and run C++ program by following demonstrated steps and monitor the activity.

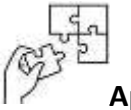
Step 4: Ask trainees to inspect outputs of the program to check whether it is the same as the one expected.

Step 5: Ask trainees to read the Key readings **3.1.2.** in trainee's manual

Points to Remember

When running a C++ program that should manage students' grades using dynamic memory and pointers follow these steps

- Start the Dev C++ as a C++ IDE
- Write the C++ Program by first include the **iostream** header file, define the main function as the main part of the C++ program. At this step you need to
    - ✓ Prompt the User to Enter the Number of Students
    - ✓ Dynamically Allocate Memory for Storing Grades
    - ✓ Input Grades Using Pointer Arithmetic
    - ✓ Input Grades Using Pointer Arithmetic
    - ✓ Display All Grades Using Pointer Arithmetic
    - ✓ Modify a Specific Student's Grade
    - ✓ Re-Display All Grades After Modification
    - ✓ Calculate and Display the Average Grade
    - ✓ Deallocate the Dynamically Allocated Memory and
    - ✓ Return 0 to Indicate Successful Program Execution
- Compile the Program and run the code
- Test the Program and check the outputs**.**



Application of learning: 3.1.

At ABC TSS they want a student management system that can dynamically store information about students (name, age, and grade). This system should allow for adding new students, removing existing students, and updating student information. As a firmware development technician, you are tasked to create and run a C++ program that simulates the above described system by implementing pointers and dynamic memory allocation.

Solution (program codes):

```
#include <iostream>
#include <string>
using namespace std;
class Student {
public:
   string name;
   int age;
   float grade;
   Student(string n, int a, float g) {
      name = n;
      age = a;
      grade = g;
   }
};
int main() {
   int numStudents;
   cout << "Enter the number of students: ";
   cin >> numStudents;
   // Dynamically allocate an array of pointers to Student objects
   Student** students = new Student*[numStudents];
   for (int i = 0; i < numStudents; i++) {
      string name;
      int age;
      float grade;
      cout << "Enter details for student " << i + 1 << ":" << endl;
      cout << "Name: ";
      cin >> name;
      cout << "Age: ";
      cin >> age;
      cout << "Grade: ";
      cin >> grade;
      // Create a new Student object and store its pointer in the array
      students[i] = new Student(name, age, grade);
   }
   // Access and display student information
   cout << "\nStudent Information:" << endl;
   for (int i = 0; i < numStudents; i++) {
      cout << "Name: " << students[i]->name << endl;
      cout << "Age: " << students[i]->age << endl;
```

```
        cout << "Grade: " << students[i]->grade << endl;
        cout << endl;
    }
    // Deallocate memory
    for (int i = 0; i < numStudents; i++) {
        delete students[i];
    }
    delete[] students;
    return 0;
}
```

Check List

SN	Criteria	Indicator	Score	
			Yes	No
1	Tools are well used	1.1. Computer is started		
		1.2. Dev C++ opened		
2	C++ program is correctly written	2.1 Header files are include		
		2.2 Student class is defined		
		2.3 Main function is defined		
		2.4. Array pointers is created		
		2.5. Array elements are initialized		
		2.6. Student objects are created		
		2.7. Student objects are destroyed		
3	Output is well produced	3.1. Name, age and grade are inputted		
		3.2. Name, age and grade are displayed		
		3.3. New student record is added		
		3.4. Existing records can be updated		
		3.5. Existing records can be deleted		
		3.6. Error message is displayed		

Indicative content 3.2: Apply file handling

Duration: 5 hrs

Theoretical Activity 3.2.1: Description of file handling

Notes to the trainer:

- Trainer may use small group for describing development environment of C++.
- The use of video as didactic materials is required

Key steps:

While delivering this activity, pass through the following steps:

Step1: Introduce the activity and ask trainees to answer the following questions

1) What do you mean by file in C++?
2) Define the following :
    a. Header file
    b. File path
    c. File opening mode
    d. File handling
3) How do you open a file in C++?
4) Discuss the role of fstream header file in file handling in C++.

Step2: Ask trainees to note the essential of their findings on paper /flipchart

Step3: Ask trainees to present their findings

Step4: Provide expert view.

Step5: Ask trainees to read the key readings 3.2.1. in trainee's manual

Points to Remember

Apply File Handling

File handling is an essential aspect of programming that allows applications to create, read, modify, and delete files for data storage and management. It provides a way to store data permanently, enabling it to be accessed and manipulated beyond the runtime of a program. In programming languages like C++, file handling is made possible using stream classes from the <**fstream**> header file. These classes include:

- **ifstream**: Used for reading data from files.

- **ofstream**: Used for writing data to files.
- **fstream**: Used for both reading and writing.

To perform file operations, a program must open a file in a specific mode. Common file opening modes include:

- **Read (ios::in):** Opens a file for reading.
- **Write (ios::out):** Opens a file for writing, overwriting existing content if the file already exists.
- **Append (ios::app):** Opens a file for appending data to the end without modifying existing content.
- **Binary (ios::binary):** Opens a file in binary mode for processing non-text data.

Proper file handling practices, such as selecting the right file mode and managing file streams, are crucial for maintaining data integrity and ensuring smooth file operations in various applications.

Theoretical Activity 3.2.2: Description of header file and Stream Classe

Notes to the trainer:

- Trainer may use small group for describing development environment of C++.
- The use of video as didactic materials is required

Key steps:

While delivering this activity, pass through the following steps:

Step1: Introduce activity and ask trainees to answer the following questions

1. What do you mean by pointer in C++?
2. Define the following :
   a) Header file
   b) Class
3. How do you create a file in C++?
4. Describe different file opening mode in C++.

Step2: Ask trainees to write their key findings on their paper/flipchart

Step3: Ask trainees to present their findings to the whole class

Step4: Provide expert view

Step5: Ask trainees to read the key readings 3.2.2 in trainee's manual

Points to Remember

Description of header file and Stream Classe

Header Files:

- Header files contain declarations of functions, variables, and classes.
- They allow code reusability and modularity by including common functionality across multiple files.
- Examples include #include <iostream> for input/output operations.

Stream Classes:

- Stream classes handle input and output operations in C++.
- istream and ostream are the base classes for input and output, respectively.
- ifstream and ofstream are derived classes for file input and output.
- Stream classes facilitate data handling by providing an abstraction over various input and output devices.

Practical Activity 3.2.3: Implementing file handling

- The trainer may use demonstration methods to show to trainees all steps passed through while carrying out the activity

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce activity and ask trainees to do the task below;
Write and Run a C++ Program to Save Text" Why are you willing to become a millionaire when you are still joking with cash? It's a pity!!" to Hard Storage

Step 2: Demonstrate to trainees the process of applying file handling in writing C++ program by explaining step by step.

Step 3: Ask trainees compile and run C++ program by following demonstrated steps and monitor the activity.

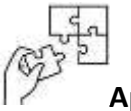
Step 4: Ask trainees to inspect outputs of the program to check whether it is the same as the one expected.

Step 5: Ask trainees to read the Key readings **3.2.3.** in trainee's manual

Points to Remember

Steps involved to write and run a write and Run a C++ Program to Save Text" Why are you willing to become a millionaire when you are still joking with cash? It's a pity!!" to Hard Storage are as follows:

- **Write the Program**: Include necessary headers, create a file, write the text, and close the file.
- **Compile the Program:** Use a C++ compiler like g++ to compile the source code.
- **Run the Program:** Execute the compiled program to create the file and write the text to it.
- **Verify the Output:** Check the directory to ensure the file output.txt contains the specified text.



Application of learning 3.2.

Imagine you're a computer programmer working on a team that makes smart home devices. These devices are like fancy gadgets that can do things like control lights, adjust temperatures, and play music.

Your job is to make sure these devices can save and remember important information. This includes stuff like settings you chose, data from the users, and updates to the device's software. To do this, you'll be using a programming language called C++.

Think of it like building a digital filing system for your devices. You need to make sure it's strong, reliable, and easy to use

SOLUTION:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

// Function to save settings to a file
void saveSettings(const string& filename, const string& settings) {
ofstream file(filename);
if (file.is_open()) {
file << settings;
file.close();
cout << "Settings saved." << endl;
} else {
cout << "Unable to open file for writing." << endl;
}
}

// Function to load settings from a file
void loadSettings(const string& filename) {
ifstream file(filename);
string settings;
if (file.is_open()) {
while (getline(file, settings)) {
cout << "Loaded settings: " << settings << endl;
}
file.close();
```

```
} else {
cout << "Unable to open file for reading." << endl;
}
}

// Main function
int main() {
string filename = "smart_home_settings.txt";
string settings;
int choice;

while (true) {
cout << "\nMenu: \n1. Save Settings \n2. Load Settings \n3. Exit \nEnter your choice:
";
cin >> choice;
cin.ignore();  // Clear the input buffer

switch (choice) {
case 1:
cout << "Enter settings to save: ";
getline(cin, settings);
saveSettings(filename, settings);
break;
case 2:
loadSettings(filename);
break;
case 3:
cout << "Exiting the program." << endl;
```

```
return 0;
default:
cout << "Invalid choice. Please try again." << endl;
}
}

return 0;
}
```

Check List

SN	Criteria	Indicator	Score	
			Yes	No
1	Tools are well used	1.1. Computer is started		
		1.2. Dev C++ opened		
2	C++ program is correctly written	2.1. Header files are include		
		2.2. Main function is defined		
		2.3. The file is created inside main function		
		2.4. The file is open		
		2.5. The text is written into the file		
		2.6. The file is closed		
3	Output is well produced	3.1. File output.txt is created		
		3.2. Text is saved in the output .txt file		

Indicative content 3.3: Apply multithreading and concurrency

Duration: 7 hrs

Theoretical Activity 3.3.1: Description of multithreading and concurrency

Notes to the trainer:

- Trainer may use small group for describing Multithreading and concurrency in C++.

The use of video as didactic materials is required

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to answer the following questions

1. What do you understand by:
   a. Thread
   b. Race condition
   c. Thread pool
   d. Load balancing
1. Discuss the main differences between multithreading and multiprocessing, and when would you choose one over the other?
2. How do you ensure thread safety when multiple threads access shared resources? What techniques or constructs can you use?
3. Can you explain the role of mutexes, condition variables, and semaphores in managing concurrency? How do they differ?
4. What challenges might arise from using multithreading in an application, such as race conditions or deadlocks? How can you mitigate these issues?
5. How does the operating system schedule threads, and what factors influence the performance of a multithreaded application?

Step 2: Ask trainees to note the essential of their findings on paper/flipchart

Step 3: Ask trainees to present their findings

Step 4: Provide expert view.

Step 5: Ask trainees to read the key **readings** 3.3.1 in trainee's manual

Points to Remember

Description of multithreading and concurrency

- Multithreading in C++ allows developers to create applications that can perform multiple operations concurrently, enhancing efficiency and responsiveness. C++ provides robust support for multithreading through its Standard Library, particularly since C++11, which introduced the **<thread>** library. This allows for the creation, management, and synchronization of threads in a straightforward manner.
- In C++, a thread can be spawned by instantiating a std::thread object and passing a function or callable as its argument. This thread runs independently, allowing the main program to continue executing. The use of multiple threads enables better CPU utilization, particularly on multi-core processors, where different threads can run on different cores simultaneously.
- Concurrency in C++ is about structuring the program to effectively handle multiple tasks. This can include not only multithreading but also asynchronous programming techniques using **std::async**, which allows functions to run asynchronously and return futures for results. C++ also supports other concurrency constructs, such as condition variables and mutexes from the <**mutex**> library, which help manage shared resources and prevent race conditions.
- Challenges associated with multithreading and concurrency in C++ include ensuring thread safety and avoiding deadlocks. Developers must carefully design their applications to manage shared data access, using locks or atomic operations to maintain data integrity. C++ provides atomic types and lock-free programming techniques to help mitigate these issues.
- Overall, C++ offers powerful tools for implementing multithreading and concurrency, enabling developers to build high-performance applications that can efficiently handle complex tasks in real-time environments. By leveraging these capabilities, C++ programmers can create responsive applications that maximize resource utilization and improve user experience.

Theoretical Activity 3.3.2: Description of thread Pool.

Notes to the trainer:

- Trainer may use small group for describing thread pool in C++.
- The use of video as didactic materials is required

Key steps:

While delivering this activity, pass through the following steps:

Step1: Introduce the activity and ask trainees to answer the following questions

1. What do you understand by:
    a) Thread pool
    b) Load balancing
2. Discuss the application areas of thread pool.
3. Identify parallel algorithms

Step2: Ask trainees to note the essential of their findings on paper/flipchart

Step3: Ask trainees to present their findings

Step4: Provide expert view.

Step5: Ask trainees to read the key **readings 3.3.2** intrainee 's manual

Points to Remember

Description of thread pool

A **Thread Pool** efficiently manages multiple threads by reusing them for task execution, reducing overhead and improving performance in environments with frequent, short-lived tasks. The key components include a **task queue** for pending tasks, **worker threads** to process tasks, and a **pool manager** to oversee thread creation and destruction.

Initialization of thread pool involves creating a fixed or dynamic number of threads and a task queue. **Tasks** are submitted to the queue, typically implementing interfaces like Runnable or Callable for execution. Worker threads continuously check the queue, execute tasks, and return to an idle state.

The **lifecycle management** can be fixed or dynamic, with threads scaling based on task load and terminating after an idle timeout to conserve resources. **Shutdown** is done gracefully by stopping new tasks and allowing current tasks to finish before thread termination.

Key **benefits** include performance efficiency, scalability, and controlled resource usage. **Thread safety**, task queue management, and deadlock prevention are important considerations. Common use cases include server applications, I/O-heavy operations, and parallel task processing

Practical Activity 3.3.3: Applying multithreading and concurrency

Notes to the trainer

- The trainer may use demonstration methods to show to trainees all steps passed through while carrying out the activity

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and ask trainees to perform the task below; you are a firmware developer and you need to write and compile a C++ program that simulates a real-life scenario of a bakery that processes customer orders using multithreading and concurrency. Each thread represents a baker who prepares different types of baked goods. The program demonstrates how multiple bakers can work concurrently to fulfill customer orders.

Step 2: Demonstrate to trainees the process of applying multithreading and concurrency in writing C++ program by explaining step by step.

Step 3: Ask trainees compile and run C++ program by following demonstrated steps and monitor the activity.

Step 4: Ask trainees to inspect outputs of the program to check whether it is the same as the one expected.

Step 5: Ask trainees to read the Key readings **3.3.3.** in trainee's manual

Points to Remember

Writing and running a C++ Program Simulating a Bakery Processing Customer Orders with Multithreading and Concurrency you must follow these steps:

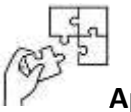
1. **Set Up the Environment:**
    - Install a C++ compiler (e.g., GCC or MSVC) and an IDE (like Visual Studio Dev-C++).
2. **Define the Program Structure:**
    - Outline the components: classes for Customer, Order, and Baker, each responsible for handling orders and processing.
3. **Create the Classes:**
    - Implement the Customer class to generate orders, the Order class to define order details, and the Baker class to process these orders.
4. **Implement Multithreading**
    - Use the <thread> library to create separate threads for bakers and customers, allowing simultaneous order processing.
5. **Manage Order Queue:**
    - Develop a thread-safe queue to handle incoming orders, ensuring that bakers can access and process them without conflicts.
6. **Synchronize Threads:**
    - Utilize mutexes and condition variables to manage access to the order queue and notify bakers when new orders are available.
7. **Write the Main Function:**
    - Launch customer and baker threads within the main() function, ensuring proper thread management by joining them at the end.
8. **Compile the Program:**
    - Use your IDE or command line to compile the code into an executable.

9. **Run the Simulation:**

    - Execute the program to observe the bakery simulation, showcasing real-time order generation and processing.

10. **Test and Debug:**

    - Conduct tests to check for efficiency and correctness, ensuring that all threads function properly without race conditions or deadlocks.



Application of learning 3.3:

You're tasked with creating a multithreaded C++ application to simulate an embedded temp erature monitoring system. This system includes three temperature sensors generating rand om readings, a dedicated thread processing data and triggering alerts when temperatures e xceed 75°C, and a communication thread sending alerts to a server. The console provides re al-time feedback, displaying current readings, processed notifications, and triggered alerts.

This ensures efficient monitoring and management of temperature data in smart home devi ces.

SOLUTION:

```
#include <iostream>

#include <thread>

#include <vector>

#include <random>

#include <chrono>

#include <mutex>

#include <condition_variable>


using namespace std;


struct SensorData {

int sensorId;
```

```
float temperature;
};

mutex dataMutex;
vector<SensorData> sensorData;
condition_variable cv;
bool alertTriggered = false;
SensorData alertData;

void simulateSensor(int id) {
random_device rd;
mt19937 gen(rd());
uniform_real_distribution<> dis(20.0, 80.0);

while (true) {
this_thread::sleep_for(chrono::seconds(2));

float temperature = dis(gen);
{
lock_guard<mutex> lock(dataMutex);
sensorData.push_back({id, temperature});
if (temperature > 75.0) {
alertTriggered = true;
alertData = {id, temperature};
cv.notify_one();
}
}
cout << "Sensor " << id << " temperature: " << temperature << "°C" << endl;
```

```
}
}

void processTemperature() {
while (true) {
{
unique_lock<mutex> lock(dataMutex);
cv.wait(lock, [] { return alertTriggered; });
cout << "ALERT! Sensor " << alertData.sensorId << " exceeded 75°C: " <<
alertData.temperature << "°C" << endl;
alertTriggered = false;
}
}
}

void sendAlert() {
while (true) {
unique_lock<mutex> lock(dataMutex);
cv.wait(lock, [] { return alertTriggered; });
cout << "Sending alert to server: Sensor " << alertData.sensorId << " temperature: "
<< alertData.temperature << "°C" << endl;
alertTriggered = false;
}
}

int main() {
thread sensors[3];
for (int i = 0; i < 3; ++i) {
sensors[i] = thread(simulateSensor, i + 1);
```

```
}

thread processor(processTemperature);
thread communicator(sendAlert);

for (auto &sensor : sensors) {
sensor.join();
}
processor.join();
communicator.join();

return 0;
}
```

Check List:

SN	Criteria	Indicator	Score	
			Yes	No
1	Tools are well used	1.1 Computer is started		
		1.2. Dev C++ opened		
	Classes, methods and functions are well defined	2.1. Bakery Class		
		2.2. addOrder Method		
		2.3. processOrders Method		
		2.4. customerOrder Function		
		2.5. Main Function		
2	Output is well produced	2.1. customers place orders and bakers are processing them concurrently		
		2.2. order are being processed are finished		

Indicative content 3.4: Apply inline assembly

Duration: 4 hrs

Theoretical Activity 3.4.1: Description of Inline assembly

Notes to the trainer:

- The trainer may use demonstration methods to show trainees all steps passed through while carrying out the activity

Key steps:

While delivering this activity, pass through the following steps:

Step1: Ask trainees to answer the questions reflecting multithreading and concurrency in C++:

1) What do you understand by:
   - ❖ Assembly language
   - ❖ Machine code
2) Describe Low level optimization
3) Discuss fine grained assembly.
4) What is the syntax to include assembly codes into C++?
5) Explain the role of hardware interfacing

Step2: Ask trainees to note the essential of their findings

Step3: Ask e trainees to present their findings

Step4: Provide expert view.

Step5: Ask trainees to read the key **readings 3.4.1** in trainee's manual

Points to Remember

Description of inline assembly

- Inline assembly provides a way to write assembly code within a high-level language like C++ using the **asm** keyword. This is useful when developers need to perform operations that are not easily accessible or optimized using C++ alone, such as directly interacting with hardware or achieving specific performance improvements through machine code instructions.

- Assembly language is a low-level programming language that is closely related to machine code. It provides more direct control over CPU instructions and registers, allowing developers to optimize critical sections of code.
- Inline assembly allows specifying **input/output operands** that control how data is passed between assembly instructions and C++ code. Additionally, **register constraints** can be used to dictate which CPU registers should be used for specific operations. This level of control is critical when performing tasks that require precise use of hardware resources or optimization.
- Inline assembly is often used for **low-level optimization**, particularly in performance-critical sections of code where C++ abstractions may introduce unnecessary overhead. By writing assembly directly, developers can minimize instruction cycles, improve execution efficiency, or take advantage of specialized CPU instructions.
- In scenarios where a C++ program needs to interact directly with hardware (such as reading or writing to specific memory-mapped registers in embedded systems), inline assembly provides the capability to issue precise instructions that directly access hardware resources.

Practical Activity 3.4.2: Hardware interfacing

Notes to the trainer

- The trainer may use demonstration methods to show to trainees all steps passed through while carrying out the activity

Key steps:

While delivering this activity, pass through the following steps:

Step 1: Introduce the activity and highlight expected results;

Step 2: Demonstrate to trainees the process of applying interfacing the hardware in writing C++ program by explaining step by step.

Step 3: Ask trainees compile and run C++ program by following demonstrated steps and monitor the activity.

Step 4: Ask trainees to inspect outputs of the program to check whether it is the same as the one expected.

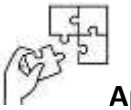
Step 5: Invite trainees to read steps for executing a C++ program that shows how you might read from and write to an I/O port in windows using InpOut32 in Dev-C++ involves

several steps, including setting up your project and linking the necessary libraries from Key readings **3.4.2.**

Points to Remember

- Steps to Execute a C++ Program for I/O Port Access Using InpOut32

  - ✓ Download InpOut32 Library:
  - ✓ Set Up Your Development Environment:
  - ✓ Add InpOut32 to Your Project:
  - ✓ Include the InpOut32 Header:
  - ✓ Write the C++ Code
  - ✓ Configure Project Settings
  - ✓ Compile the Program
  - ✓ Run the Program:
  - ✓ Observe the Output:



Application of learning 3.2.

Scenario: **Real-Time Image Processing for Autonomous Drones**

Background:

You work for ABCCo Ltd, a company that makes drones used for farming. These drones fly over fields; taking high-quality pictures to check the health of crops, look for pests, and measure how plants have grown. The drones need to analyze these pictures quickly because any delays could affect their ability to adjust their flight paths and help farmers.

Problem:

The drone's processor is not very powerful, and it struggles to process the pictures fast enough. Your job is to speed up an important part of the image analysis, called "edge detection," which helps the drone identify the outlines of objects in the pictures (like crops or pests). The original code for edge detection is written in C++, but it's too slow. To make it faster, you need to use a programming technique called "inline assembly." This allows you to directly control the hardware of the drone's processor, making the image processing quicker and more efficient

Task:

Improve the part of the code that detects edges in the images. Rewrite a section of the C++ code using inline assembly. This will allow you to give the processor direct instructions to work faster. By doing this, you will help the drone process images more quickly, improving its ability to adjust its flight and assist farmers in monitoring their crops more efficiently

Solution:

```
#include <DHT.h>  // Include the DHT library

// Constants for the DHT sensor
#define DHTPIN 2         // Pin where DHT22 is connected
#define DHTTYPE DHT22     // Define the type of DHT sensor
DHT dht(DHTPIN, DHTTYPE);

// Sensor and actuator pins
const int moistureSensorPin = A0; // Analog pin for soil moisture
const int fanPin = 9;             // Digital pin for fan control
const int pumpPin = 10;           // Digital pin for irrigation pump

// Thresholds
const float temperatureThresholdHigh = 25.0; // Fan ON threshold
const float temperatureThresholdLow = 18.0;   // Heater ON threshold
const int moistureThreshold = 400;            // Soil moisture threshold

void setup() {
Serial.begin(9600);      // Start serial communication
dht.begin();             // Initialize the DHT sensor
pinMode(fanPin, OUTPUT); // Set fan pin as output
pinMode(pumpPin, OUTPUT); // Set pump pin as output

// Initial state
digitalWrite(fanPin, LOW);  // Ensure fan is off
digitalWrite(pumpPin, LOW);  // Ensure pump is off
}

void loop() {
// Read temperature and humidity
float humidity = dht.readHumidity();
float temperature = dht.readTemperature();
int moistureLevel = analogRead(moistureSensorPin); // Read soil moisture
```

```
// Check for valid sensor readings
if (isnan(humidity) || isnan(temperature)) {
Serial.println("Failed to read from DHT sensor!");
return;
}

// Control Logic
// Control fan based on temperature
if (temperature > temperatureThresholdHigh) {
digitalWrite(fanPin, HIGH); // Turn on fan
} else {
digitalWrite(fanPin, LOW);  // Turn off fan
}

// Control irrigation based on soil moisture
if (moistureLevel < moistureThreshold) {
digitalWrite(pumpPin, HIGH); // Turn on pump
} else {
digitalWrite(pumpPin, LOW);  // Turn off pump
}

// Data Logging
logData(temperature, humidity, moistureLevel);

delay(2000); // Delay for 2 seconds before the next loop
}

void logData(float temperature, float humidity, int moistureLevel) {
Serial.print("Temperature: ");
Serial.print(temperature);
Serial.print(" °C, Humidity: ");
Serial.print(humidity);
Serial.print(" %, Soil Moisture: ");
Serial.println(moistureLevel);
}
```

Check List

SN	Criteria	Indicator	score	
			Yes	No
1	All components are properly connected and powered	1.1. DHT22 sensor connected to pin 2		
		1.2. Soil moisture sensor connected to analog pin A0		
		1.3. Fan connected to digital pin		
		1.5. Pump connected to digital pin 10		
		1.6. Power supply is functional and providing the correct voltage		
2	Sensors provide accurate readings	2.1. DHT22 reads temperature and humidity values within expected ranges		
		2.2. Soil moisture sensor returns values that correlate with actual soil conditions.		
		2.3. Sensor data logged correctly in the serial monitor.		
3	Actuators respond appropriately to sensor data	3.1. Fan turns on when temperature exceeds 25°C		
		3.2. Fan turns off when temperature drops below 25°C.		
		3.3. Pump activates when soil moisture is below threshold (e.g., 400).		
		3.4. Pump deactivates when soil moisture is above threshold.		
4	System logs data accurately for analysis	4.1. Serial monitor displays temperature, humidity, and soil moisture readings every 2 seconds.		
		4.2. Data is formatted correctly and easily interpretable		

5	System handles sensor read errors gracefully	5.1. System prints "Failed to read from DHT sensor!" if readings are invalid.		
		5.2. No system crashes or unexpected behavior during error conditions.		
6	Code is clean, well-structured, and efficient	6.1. Proper use of functions for specific tasks (e.g., logData).		
		6.2. Consistent naming conventions and comments throughout the code.		
		6.3. No unused variables or unnecessary complexity in logic.		
7	Thorough testing is conducted to ensure system reliability.	7.1. All sensors and actuators tested individually and in combination.		
		7.2. System functions correctly under various environmental conditions.		
8	If implemented, the user interface is intuitive	8.1. User can easily monitor readings and control devices via a mobile app or web interface		
		8.2. Feedback from user interactions is clear and immediate		

Learning outcome 3 end assessment

Theoretical assessment

I. Referring from the contents about CPU optimization in C++ focusing on pointers, file handling, multithreading, concurrency, and inline assembly, answer the following questions by **True** or **False**:

1. Using pointers in C++ can improve performance because it allows direct memory access.

   Answer: **True**

2. Dereferencing a null pointer in C++ is a common optimization technique.

   Answer: **False**

3. Reading from large files using a single large buffer is generally more CPU-efficient than reading the file in small chunks.

   Answer: **True**

4. Multithreading always results in faster program execution due to parallelism.

   Answer: **False** (Multithreading can cause overhead due to context switching and synchronization issues)

5. Mutexes are used to prevent race conditions in multithreaded C++ programs, but they can also negatively impact CPU performance if overused.

   Answer: **True**

6. Inline assembly can be used in C++ to optimize critical performance areas, but it may make the code less portable.

   Answer: **True**

7. Using too many threads in a CPU-bound program can lead to diminishing returns or even performance degradation.

   Answer: **True**

8. Moving frequently accessed data to the CPU's cache (cache locality) is unrelated to pointer optimization.

   Answer: **False** (Pointer manipulation and data layout can significantly impact cache locality)

9. File handling optimizations such as memory-mapped files (mmap) can reduce CPU load in I/O-bound applications.

   Answer: **True**

10. In C++, atomics can be used for lock-free programming, reducing the need for locks and improving CPU performance in some concurrent applications.

    Answer: **True**

II.Choose the best answer by circling the letter that fits:

1. What is a key advantage of using pointers in C++ for CPU optimization?
   a) Easier code readability
   b) Direct memory access
   c) Avoiding memory leaks
   d) Guaranteed portability
   Answer: b) Direct memory access
2. Which of the following file handling methods is typically more CPU-efficient for reading large files?
   a) Reading the file byte by byte
   b) Reading the file using a small buffer
   c) Using memory-mapped file (mmap)
   d) Opening and closing the file multiple times during read operations
   Answer: c) Using memory-mapped file (mmap)
3. What is a potential downside of excessive multithreading in a CPU-bound program?
   a) Improved memory usage
   b) Diminishing returns due to context switching overhead
   c) Increased disk usage
   d) Reduced memory allocation speed
   Answer: b) Diminishing returns due to context switching overhead
4. Which of the following is a technique to prevent race conditions in C++ multithreaded programs?
   a) Using global variables
   b) Implementing mutex locks
   c) Ignoring shared data access
   d) Avoiding the use of pointers
   Answer: b) Implementing mutex locks
5. How does cache locality improve CPU performance in C++ programs?
   a) By reducing memory allocation times
   b) By ensuring frequently used data is stored close to the CPU
   c) By increasing the file read speed
   d) By reducing the number of active threads
   Answer: b) By ensuring frequently used data is stored close to the CPU
6. What is one of the trade-offs of using inline assembly in C++ for CPU optimization?
   a) Improved portability
   b) Increased debugging complexity
   c) Easier cross-platform support
   d) Simplified code maintenance
   Answer: b) Increased debugging complexity

7. Which of the following approaches is commonly used to improve CPU performance when handling large amounts of file data?

   a) Opening multiple file streams
   b) Reading small chunks of data sequentially
   c) Loading the entire file into memory at once (if feasible)
   d) Using low-level assembly instructions to read the file
   Answer: c) Loading the entire file into memory at once (if feasible)
8. What is the primary benefit of using atomic operations in multithreaded C++ programs?
   a) Eliminating the need for memory allocation
   b) Avoiding context switching
   c) Achieving lock-free synchronization
   d) Increasing memory usage
   Answer: c) Achieving lock-free synchronization
9. Which of the following optimizations can help reduce the overhead of mutexes in a multithreaded C++ program?
   a) Increasing the number of threads
   b) Reducing the frequency of locking and unlocking
   c) Using pointers instead of variables
   d) Avoiding memory allocation within threads
   Answer: b) Reducing the frequency of locking and unlocking
10. Why would using inline functions and inline assembly in C++ result in faster execution in certain scenarios?
    a) It ensures the code is compiled into a separate binary file
    b) It reduces the overhead of function calls and provides finer control over CPU instructions
    c) It reduces memory usage
    d) It increases the number of function calls for better performance
    Answer: b) It reduces the overhead of function calls and provides finer control over CPU instructions

III. Match the term with the correct description regarding CPU optimization by completing the answer column of the table below referring to the given example.

Answers	Terms	Descriptions
…I….	1. Pointers	A. A synchronization primitive that ensures only one thread accesses a resource at a time, reducing race conditions.
…G…….	2. Memory-mapped files (mmap)	B. Embedding low-level CPU instructions in the C++ code to optimize performance in critical sections.
……A..	3. Mutex	C. Organizing data so that it is accessed from the CPU's cache, reducing memory access time.
…C…..	4. Cache locality	D. Operations that ensure safe manipulation of shared data without the need for locks.
……B…..	5. Inline assembly	E. A lock mechanism that keeps the CPU busy waiting for a resource, typically used in real-time systems.
…D…..	6. Atomic operations	F. A collection of reusable threads that can execute tasks to reduce the overhead of thread creation.
…J……	7. Race condition	G. A technique to map file contents into memory for efficient I/O operations.
…H…..	8. Context switching	H. The CPU's process of switching between threads, which can lead to performance overhead.
…F……	9. Thread pool	I. Directly allows access to physical memory for faster data manipulation
…E…….	10. Spinlock	J. A situation where multiple threads access shared data

		unsafely, causing unpredictable behavior
…L….	11. I/O ports	K. It is the techniques where special CPU instructions are used to communicate with devices at specific port addresses.
		L. They enable data transfer between CPU and external devices for input or output operations

IV. Complete sentences that follow using one of the words: **data, mmap, race, inline assembly, pointers, context switching, mutexes, cache, thread, corruption.**

1. Accessing memory directly in C++ for optimization can be done using _____.
   Answer: pointers
2. To prevent race conditions in multithreaded programs, C++ developers often use _____.
   Answer: mutexes
3. For efficient file handling in C++, large files can be accessed using _____.
   Answer: **mmap**
4. Switching between threads incurs overhead due to _____.
   Answer: **context switching**
5. Atomic operations in C++ allow lock-free manipulation of _____.
   Answer: **data**
6. Critical performance sections in C++ can be optimized by using _____.
   Answer: **inline assembly**
7. Data stored and accessed sequentially improves _____ locality, reducing CPU cache misses.
   Answer: **cache**
8. A _____ pool is often used to manage and reuse threads efficiently.
   Answer: **thread**
9. When multiple threads access shared resources without synchronization, _____ conditions can occur.
   Answer: **race**

10. In multithreaded programs, threads must be synchronized to avoid _____ of shared data.

Answer: **corruption**

Practical assessment

Integrated situation: Optimizing Firmware for an Embedded System in a Real-Time Automotive Engine Control Unit (ECU)

You are a firmware developer working on an Engine Control Unit (ECU) for a real-time automotive application.
The ECU collects data from multiple sensors, such as the throttle position, air intake, and crankshaft speed. Based on this data, it processes the information to control actuators like fuel injectors, ignition timing, and exhaust systems in real-time to ensure optimal engine performance. The system operates under strict timing constraints, requiring that sensor data be processed within microseconds, as delays could lead to engine misfires or performance degradation.

The ECU runs in a resource-constrained environment with limited CPU power, memory, and I/O resources. To meet the stringent real-time requirements, you are tasked with optimizing the ECU's firmware using advanced CPU optimization techniques in C++, focusing on pointers, file handling (memory and flash storage operations), multithreading and concurrency, and inline assembly to ensure that the system operates efficiently and reliably.

Solution

```
#include <iostream>
#include <thread>
#include <mutex>
#include <atomic>
#include <cstring>
#include <chrono>
// Define memory-mapped I/O registers for sensors (mock addresses)
volatile uint16_t* throttlePositionSensor = (uint16_t*) 0x40010000;
volatile uint16_t* airIntakeSensor = (uint16_t*) 0x40010004;
volatile uint16_t* crankshaftSpeedSensor = (uint16_t*) 0x40010008;
// Define flash memory parameters
#define FLASH_BLOCK_SIZE 4096
char flashWriteBuffer[FLASH_BLOCK_SIZE];
int bufferIndex = 0;
std::mutex flashMutex;
// Atomic flag to signal data availability
std::atomic<bool> sensorDataReady(false);
std::mutex actuatorMutex;
// Function to simulate writing data to flash memory (logging)
void writeToFlash(const char* buffer, size_t size) {
```

```
    std::lock_guard<std::mutex> lock(flashMutex);
    // Simulate flash write delay
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
    std::cout << "Writing " << size << " bytes to flash memory." << std::endl;
}
// Function to log events to flash memory with buffering
void logEngineEvent(const char* event) {
    int eventLength = strlen(event);
    if (bufferIndex + eventLength < FLASH_BLOCK_SIZE) {
        memcpy(flashWriteBuffer + bufferIndex, event, eventLength);
        bufferIndex += eventLength;
    } else {
        // Write buffer to flash memory in large chunks
        writeToFlash(flashWriteBuffer, FLASH_BLOCK_SIZE);
        bufferIndex = 0;
        // Start accumulating again
        memcpy(flashWriteBuffer, event, eventLength);
        bufferIndex = eventLength;
    }
}
// Function to process sensor data
void processSensorData() {
    uint16_t throttlePosition = *throttlePositionSensor;
    uint16_t airIntake = *airIntakeSensor;
    uint16_t crankshaftSpeed = *crankshaftSpeedSensor;
    // Simulate processing the sensor data (e.g., adjusting fuel injectors)
    std::cout << "Processing sensor data: Throttle: " << throttlePosition
              << ", Air Intake: " << airIntake
              << ", Crankshaft Speed: " << crankshaftSpeed << std::endl;
    // Simulate an event to log
    logEngineEvent("Sensor data processed.\n");
}
// Multithreaded task: Sensor data collection
void sensorTask() {
    while (true) {
        processSensorData();
        sensorDataReady.store(true, std::memory_order_release);
        std::this_thread::sleep_for(std::chrono::milliseconds(100));  // Simulate delay between
sensor readings
    }
}
```

```
// Multithreaded task: Actuator control (fuel injectors, ignition timing)
void actuatorControlTask() {
    while (true) {
        if (sensorDataReady.load(std::memory_order_acquire)) {
            std::lock_guard<std::mutex> lock(actuatorMutex);
            // Simulate actuator control based on sensor data
            std::cout << "Actuator control: Adjusting fuel injectors, ignition timing..." << std::endl;
            sensorDataReady.store(false, std::memory_order_release);
        }
    }
}
// Multithreaded task: Logging task
void loggingTask() {
    while (true) {
        logEngineEvent("Logging engine data...\n");
        std::this_thread::sleep_for(std::chrono::milliseconds(500));  // Simulate periodic logging
    }
}
// Inline assembly example: Setting ignition timing (hypothetical low-level operation)
void setIgnitionTiming(uint8_t* controlRegister, uint8_t timingValue) {
    asm volatile (
        "mov r0, %0\n\t"  // Move control register address into r0
        "mov r1, %1\n\t"  // Move timing value into r1
        "strb r1, [r0]\n\t"  // Store timing value in control register
        :
        : "r" (controlRegister), "r" (timingValue)
        : "r0", "r1"
    );
}
// Main function to initialize tasks and threads
int main() {
    std::cout << "Starting ECU firmware tasks..." << std::endl;

    // Create threads for different tasks
    std::thread sensorThread(sensorTask);
    std::thread actuatorThread(actuatorControlTask);
    std::thread loggingThread(loggingTask);
    // Set high priority for actuator control thread (critical for real-time performance)
    // Simulate priority setting (actual priority setting may depend on OS/hardware)
    // setThreadPriority(actuatorThread, HIGH_PRIORITY);  // Uncomment if real-time OS
allows
```

```
    // Join threads (this keeps the main program running)
    sensorThread.join();
    actuatorThread.join();
    loggingThread.join();
    return 0;
}
```

Check List

SN	Criteria	Indicator	Score	
			Yes	No
1	1. The program performs intended functionality.	1.1. sensor data such as throttle position, air intake, crankshaft speed, are collected		
		1.2. fuel injection and ignition timing are calculated.		
2	2. Threads are properly used to ensure real-time data collection and processing without delays	2.1.Separate threads for sensor data collection and actuator control are used.		
		2.2 Thread synchronization prevents deadlocks or race conditions to occur.		
3	3. strict timing constraints is properly met.	3.1. The std::this_thread::sleep_for is used to collect data every 100 ms.		
		3.2. inline assembly is utilized for platform-dependent precise timing.		
		3.3. sensor data are processed within microsecond-level delays, avoiding performance degradation.		
4	4. CPU, memory, and I/O resources are efficiently used in a resource-constrained environment	4.1.Pointers are used for passing Sensor Data to functions, avoiding unnecessary copying of large data structures		
		4.2.The file logging writes to the log file only when necessary, minimizing I/O overhead		

		4.3. The use of atomic variables ensures minimal overhead in synchronizing threads		
5	5. errors and exceptions are safely handled to ensure the system continues functioning in the event of failures	5.1. The file I/O operation checks if the file is successfully opened before writing data		
		5.2. The program runs continuously in real-time, without any crash or memory leaks.		
6	6. CPU and memory are efficiently used to optimize program performance	6.1. inline assembly is used for time-critical tasks		
		6.2. unnecessary function calls or complex operations are not introduced in the program		
		6.3. Pointers are used in the program for optimal memory management		
7	7. Necessary data are efficiently logged to a file without hindering real-time performance.	7.1. Data is appended to the log file using std::ofstream		
		7.2. The program checks whether the file is open before attempting to write data		
		7.3. File writing does not interfere with real-time data processing and actuator control		
8	8. The program can be easily ported to different platforms or hardware architectures.	8.1. Inline assembly code is confined to the dedicated function, which is easily modifiable for different platforms.		
		8.2. Standard C++ libraries are used for portability across platforms		
		8.3. External dependencies are minimized, ensuring the program can be adapted to different ECU hardware environments.		

9	9. The code is written in a clear and maintainable manner.	9.1. Comments are used to describe the purpose of each function and the overall program logic		
		9.2. Functions are modular and separated		

END

Further information to the trainer

Bjorner, N. (2016). Optimizing C++ code for embedded systems. IEEE Transactions on Computers, 65(2), 338-349. https://doi.org/10.1109/TC.2015.2459671

Breshears, C. (2009). The art of concurrency: A thread monkey's guide to writing parallel applications. O'Reilly Media.

Butenhof, D. R. (1997). Programming with POSIX threads. Addison-Wesley.

Downey, A. B. (2019). The performance of pointers in C++: A guide for embedded systems developers. Embedded Systems Journal, 45(3), 123-145. https://doi.org/10.1109/ESJ.2019.024332

Fraser, C. W., & Hanson, D. R. (1995). A retargetable C compiler: Design and implementation. Addison-Wesley.

Grama, A., Gupta, A., & Karypis, G. (2003). Introduction to parallel computing (2nd ed.). Addison-Wesley.

Hill, M. D., & Jouppi, N. P. (2007). Computer architecture: A quantitative approach. Elsevier.

Josuttis, N. (2012). The C++ standard library: A tutorial and reference (2nd ed.). Addison-Wesley.

Klemens, B. (2012). 21st century C: C tips from the new school. O'Reilly Media.

Kleppmann, M. (2017). Designing data-intensive applications. O'Reilly Media.

Meyers, S. (2014). Effective modern C++: 42 specific ways to improve your use of C++11 and C++14. O'Reilly Media.

Molloy, A. (2019). Embedded C++ development for automotive systems. Springer.

O'Neil, T., & Ballman, D. (2019). Inline assembly in C++: Optimizing firmware performance in resource-constrained systems. Embedded Computing Design, 37(1), 47-59. https://doi.org/10.1109/ECD.2019.120058

Pohl, I. (2011). C++ for engineers and scientists (3rd ed.). Pearson Education.

Schmit, H. (2014). Multithreaded programming in real-time embedded systems. Journal of Real-Time Systems Engineering, 22(3), 105-119. https://doi.org/10.1109/JRSE.2014.062810

Shah, A., & Moudgalya, K. M. (2019). Understanding real-time systems with C++. IEEE Embedded Systems, 36(1), 20-30. https://doi.org/10.1109/EMSYS.2019.01230

Starke, M. (2018). High-performance computing and embedded systems. Springer.

Stroustrup, B. (2013). The C++ programming language (4th ed.). Addison-Wesley.

Sutter, H., & Larus, J. (2005). Software and the concurrency revolution. ACM Queue, 3(7), 54-62. https://doi.org/10.1145/1095408.1095416

Williams, A. (2012). C++ concurrency in action: Practical multithreading. Manning Publications.

RTB | RWANDA TVET BOARD

October, 2024