## MQF 633 C++ FOR FINANCIAL ENGINEERING

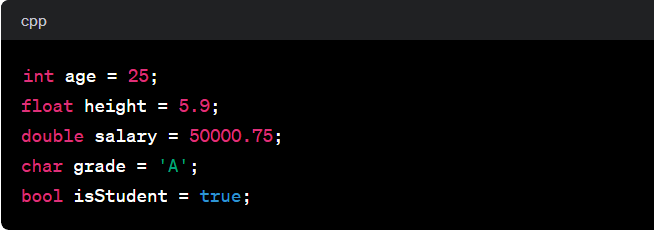
# C++ Data Types, Variable and Basic Syntax

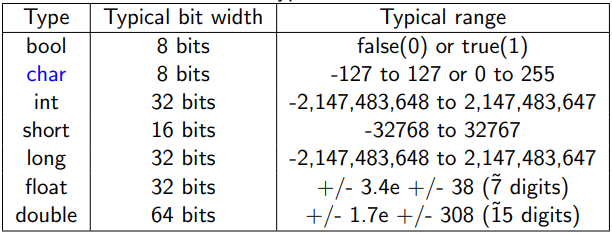
## Part I: Understanding C++ Data Types & Variable

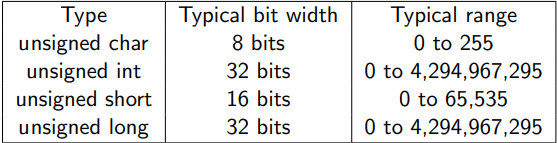
### Overview of Data Types

In C++, data types are used to define the type of data that a variable can hold. It helps in allocating memory and performing operations on variables. C++ supports a variety of data types.

* Fundamental Data Types
* int: Integer type
* float: Floating-point type
* double: Double-precision floating-point type
* char: Character type
* bool: Boolean type
* long: 32bit integer type
* short: 16bit integer type
* unsigned int: represent non-negative integer values

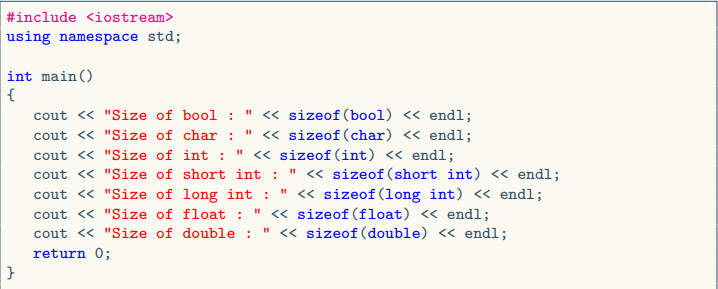






Note: 1 byte = 8 bits

Example: Actual sizeof() Data Types sizeof.cpp, refering to data\_type.cpp.



Some additional notes:

In C++, long and int are different data types, and they have different sizes. Both are used to represent integer values, but their sizes can vary depending on the compiler and the platform. Here are the typical sizes for these data types on a 32-bit and 64-bit system:

On a 32-bit system:

* int: 4 bytes
* long: 4 bytes

On a 64-bit system:

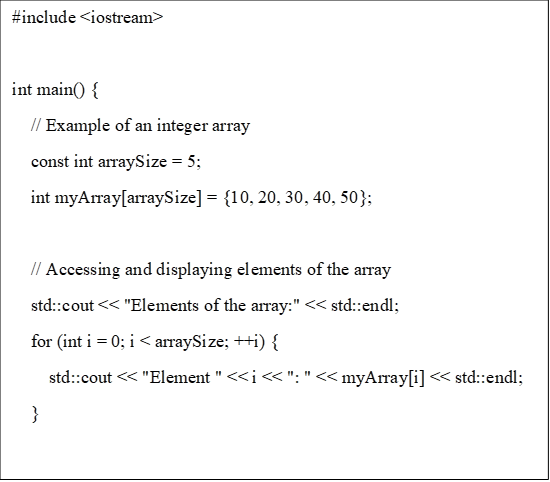
* int: 4 bytes
* long: 8 bytes

In C++, the sizes of these data types are implementation-dependent, meaning they can vary from one compiler to another and from one system to another. The C++ standard only specifies the minimum size of each data type.

If you need a guaranteed size regardless of the platform or compiler, you can use fixed-size integer types provided by the <cstdint> header, such as int32\_t and int64\_t for 32-bit and 64-bit signed integers, respectively.

### Derived Data Types

Array example: Collection of similar data types, int, double, char ... etc, referring to array.cpp



Explanation

* We declare an integer array named myArray with a size of 5. The elements of the array are initialized using the curly braces {}.
* We use a for loop to iterate through the elements of the array and display them.
* We modify the value of the element at index 2 (third element) to 35. We display the modified array.
* This is a basic example, and arrays in C++ can be used for various purposes, such as storing collections of data, implementing algorithms, and more. Keep in mind that C++ arrays have a fixed size, and their size is specified at the time of declaration. If you need a dynamic-sized array, you might want to consider using a container like std::vector from the C++ Standard Library.

### Pointer: Stores memory address of another variable

In C++, a pointer is a variable that holds the memory address of another variable. It allows indirect access to the value stored in a variable by referencing its memory address. Here's a brief definition and an example of using pointers in C++: A pointer is a variable that stores the memory address of another variable. It is declared using the data type of the variable it points to, followed by an asterisk (\*).

Example of pointer type: referring to pointer.cpp

Explanation

* We declare an integer variable named number and initialize it with the value 42.
* We declare a pointer to an integer (int \*pointerToNumber) and initialize it with the address of the number variable using the address-of operator &.
* We display the value of number and its memory address using std::cout.
* We display the memory address stored in the pointer pointerToNumber.
* We access the value of number through the pointer using the dereference operator \* and display it.
* This example demonstrates the basic usage of pointers, including declaration, initialization, accessing memory addresses, and dereferencing to access the value stored at a memory location.

### Reference: Another name for an existing variable

In C++, a reference is an alias or an alternative name for an existing variable. It provides a way to access the value of a variable indirectly through its reference. Here's a definition and an example of using references in C++:

Reference Definition: A reference is an alias or an alternative name for an existing variable. It is declared using the & symbol after the data type.

Example: refer to reference.cpp

### Pointer VS Reference

Pointer:

* Useful for dynamic memory allocation (e.g., with new and delete operators). It can be used to implement data structures and algorithms.

Reference:

* Convenient for passing variables by reference to functions.
* Often used to create aliases for existing variables in a clear and concise manner.

In summary, both pointers and references offer indirect access to variables, but pointers provide more flexibility in terms of reassignment and nullability, while references offer simplicity and are commonly used in function parameter passing. The choice between pointers and references depends on the specific requirements of the task at hand.

### Difference

Pointers need to be explicitly initialized with the memory address of a variable, References must be initialized when declared and cannot be changed to refer to a different variable after initialization.

int \*ptr = &variable;

int \*ptr;

ptr = &variable;

int &ref = variable;

Pointers can be reassigned to point to different memory locations. References, once initialized, cannot be reassigned to refer to another variable.

int \*ptr = &variable;

ptr = &variable2;

int &ref = variable;

Pointers can be set to nullptr to represent a null or undefined state. References must always be initialized and cannot be null.

int \*ptr = nullptr;

// Error: references must be initialized

int &ref;

Pointer arithmetic operation.

int \*ptr = &variable;

ptr ++;

## User-Defined Data Types

In C++, user-defined types allow programmers to create their own data types, providing a way to organize and structure data in a more meaningful and modular way. The primary user-defined types in C++ include:

* Struct
* Class
* Enumeration: Defines a set of named integer constants
* Typedefs:
* Unions
* Template

### Struct

Struct is a user-defined data type that groups together variables (called members) under a single name. It is similar to class, but by default, the members of struct have public access, whereas in a class they are private.

// Defining a struct

struct Person {

string name;

int age;

float height;

};

Strut example: refer to struct.cpp

Explanation

* We define a struct named Person with three data members: name (string), age (integer), and height (double).
* In the main function, we create an instance of the Person struct named person1.
* We assign values to the data members of person1.
* We use the struct members to display the information about the person.
* Structs are useful for grouping related data members together, creating user-defined data types, and organizing data in a more meaningful way. They provide a convenient way to represent and work with complex data structures.

### Class

a user-defined data type that allows you to encapsulate both data (variables) and functions (methods) into a single entity. It is one of the foundational features of Object-Oriented Programming (OOP) in C++.

class ClassName {

// Access specifiers: public, private, or protected

public:

// Member variables

DataType memberVariable;

// Member functions (methods)

void memberFunction() {

// Function body

}

};

Accessor type

* Public: Members declared as public are accessible from anywhere in the program where the object of the class is visible.
* Private: Members declared as private are accessible only within the class in which they are defined.
* Protected: Members declared as protected are accessible within the class and its derived (sub) classes but not from outside the class hierarchy. Often used in inheritance scenarios to allow derived classes to use or modify base class members.

Example: refer to class.cppExplanation:

* We define a class named Student with three public data members: name (string), age (integer), and GPA (double).
* Inside the class, we declare a public member function displayInfo() to display the information about the student using the class members.
* In the main function, we create an instance of the Student class named student1.
* We assign values to the data members of student1.
* We call the displayInfo() member function to display the information about the student.

### Enum

In C++, an Enum (short for enumerator) is a user-defined data type that assigns names to a set of integral constant values. This makes the code more readable and easier to maintain. Enumerations are particularly useful when you have a fixed set of related constants.

enum EnumName {

Constant1,

Constant2,

Constant3

};

Example of enum refer to enum.cpp

Explanation

* It can be created by using enum or enum class
* The type of enum data field is similar as integer

### Typedefs and Using

Typedef is a keyword used to give an existing type a new name (alias). It has been part of C++ since its inception.

typedef existing\_type new\_type\_name;

typedef unsigned int uint;

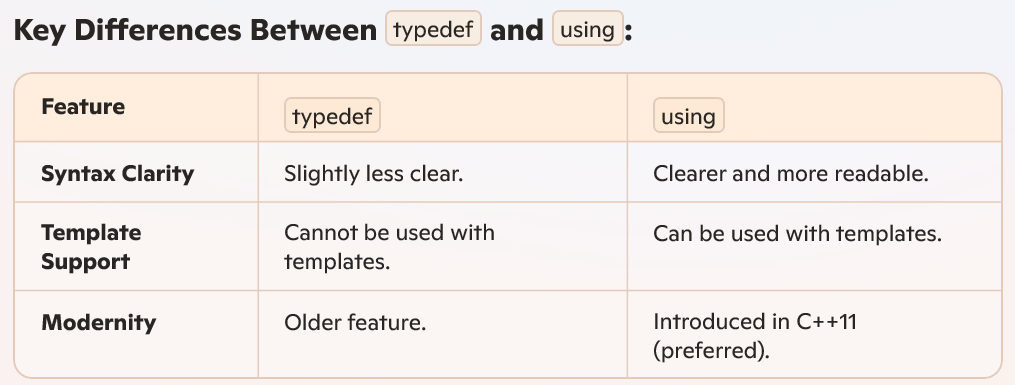
uint age = 25; // Equivalent to: unsigned int age = 25;

**Using** is a modern alternative to typedef that serves the same purpose but is more versatile and intuitive. It can also be used with templates, where typedef cannot. Using is only introduced after C++ 11.

using new\_type\_name = existing\_type;

using uint = unsigned int;

uint age = 25; // Equivalent to: unsigned int age = 25;

Example of typedef refer to typedef\_using.cpp

Explanation

In this example, typedef double Distance; creates an alias Distance for the double data type. The rest of the program uses Distance to declare variables, making the code more readable and abstracting away the specific data type. Note that starting from C++11, the using keyword is often preferred over typedef for creating type.

## Variables

A variable is a named storage (in memory) that our programs can manipulate, In C++, each variable has a type, the type of the variable determines:

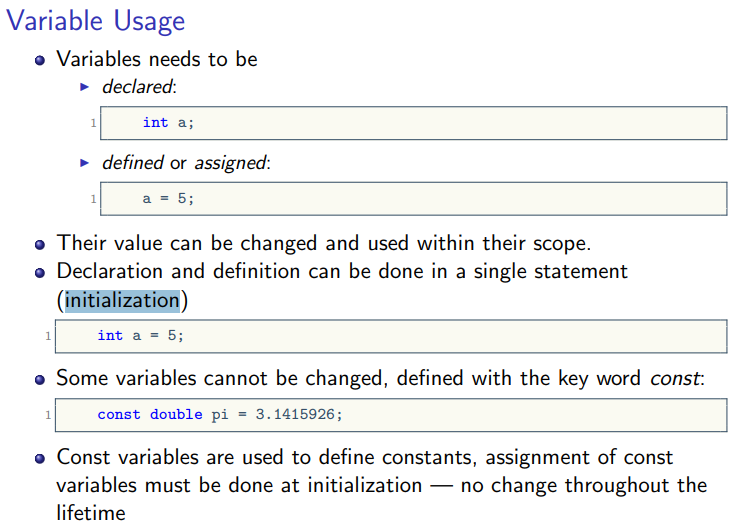
* The size and layout of variable’s memory,
* How this information is interpreted (integer, floating point, etc)
* The range of values that can be stored, and
* The set of operations that can be applied
* The value of the variable is the information stored in this memory location
* Data type of the variable tells the amount of memory space the variable needs and how to interpret the memory

### Variable Name

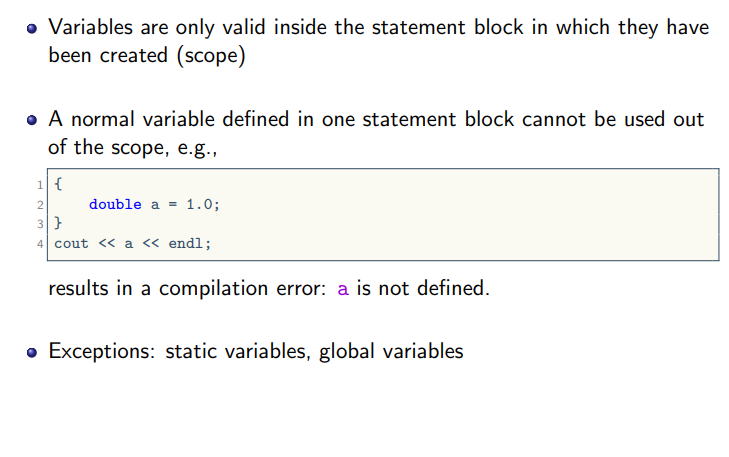
* Variable name cannot start with a digit, e.g., 1a is not a valid variable name — compilation error
* Variable names cannot contain whitespaces or special characters like: !, #, %, etc.
* Naming conventions: rules for choosing the character sequence to be used for identifiers which denote variables, types, functions, etc.
* Multiple-word identifier format: two\_words, TWO\_WORDS, TwoWords, twoWords, etc.
* Good to be consistent for readability and code analysis tools
* Not easy to enforce consistency throughout the whole project

See <https://en.wikipedia.org/wiki/Naming_convention>

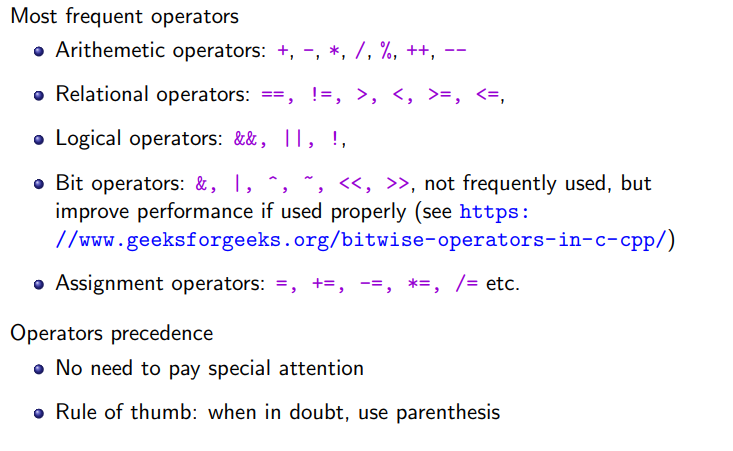
### Define Variable



### Variable Scope



### Operator



Example of operator refers to operator.cpp

### Const variable and const modifier

The const modifier in C++ is used to declare variables, function parameters, return values, or member functions as constant, which means their value or behaviour cannot be modified after they are initialized. It is a crucial feature for ensuring immutability and maintaining the integrity of data.

Key Benefits

* Ensures immutability of data
* Helps in preventing accidental modifications
* Enhances program reliability and readability

1. Constant Variables

Variables declared with const cannot be modified after initialization. They must be initialized when declared.

const int maxValue = 100;

maxValue = 200; // Error: Cannot modify a const variable

1. Function Parameters

If a function parameter is declared as const, it means the function cannot modify the value of that parameter.

void display(const string& name) {

// name = "New Name"; // Error: Cannot modify a const parameter

cout << name << endl;

}

1. Function Return Values

A function can return a const value to ensure the returned value is not modified.

const int getValue() {

return 10;

}

int main() {

const int value = getValue();

// value = 20; // Error: Cannot modify a const return value

}

1. Class Member Functions

A member function can be declared as const, meaning it cannot modify any class member variables.

class Example {

private:

int value;

public:

int getValue() const {

// value = 10; // Error: Cannot modify class members in a const function

return value;

}

};

1. Pointers and const

The const modifier can be applied in several ways with pointers

* Pointer to a const value: The value being pointed to cannot be modified.
* Const pointer to a value: The pointer itself cannot change to point to another address.

const int\* ptr;

int\* const ptr = &x;

const int\* const ptr = &x; // neither point not variable can be changed

rule of thumb, see what is the type immediate after const modifier.

More examples refer to const\_example.cpp

## Part II: C++ Basic Syntax

### Flow control using if-else statement

#include <iostream>

int main() {

// Declare a variable to store the user input

int number;

// Prompt the user to enter a number

std::cout << "Enter an integer: ";

std::cin >> number;

// Check if the number is even or odd

if (number % 2 == 0) {

std::cout << number << " is an even number." << std::endl;

}

else {

std::cout << number << " is an odd number." << std::endl;

}

return 0;

}

### Using a switch statement

#include <iostream>

int main() {

// Declare a variable to store the user input

int dayNumber;

std::cout << "Enter a number (1-7) to represent the day of the week: ";

std::cin >> dayNumber;

// Use a switch statement to determine the day of the week

switch (dayNumber) {

case 1:

std::cout << "Monday" << std::endl;

break;

case 2:

std::cout << "Tuesday" << std::endl;

break;

case 3:

std::cout << "Wednesday" << std::endl;

break;

case 4:

std::cout << "Thursday" << std::endl;

break;

case 5:

std::cout << "Friday" << std::endl;

break;

case 6:

std::cout << "Saturday" << std::endl;

break;

case 7:

std::cout << "Sunday" << std::endl;

break;

default:

std::cout << "Invalid input. Please enter a number between 1 and 7." << std::endl;

}

return 0;

}

### For loop

#include <iostream>

int main() {

int limit, sum = 0;

std::cout << "Enter a positive integer as the limit: ";

std::cin >> limit;

if (limit <= 0) {

std::cout << "Please enter a positive integer." << std::endl;

return 1; // Exit the program with an error code

}

// Use a for loop to calculate the sum of numbers from 1 to the limit

for (int i = 1; i <= limit; ++i) {

sum += i; // Add the current number to the sum

}

// Display the result

std::cout << "The sum of numbers from 1 to " << limit << " is: " << sum << std::endl;

return 0;

}

### While loop

#include <iostream>

int main() {

int number, factorial = 1;

std::cout << "Enter a positive integer: ";

std::cin >> number;

if (number < 0) {

std::cout << "Please enter a positive integer." << std::endl;

return 1; // Exit the program with an error code

}

int i = 1;

while (i <= number) {

factorial \*= i; // Multiply the current number to the factorial

++i; // Increment the loop variable

}

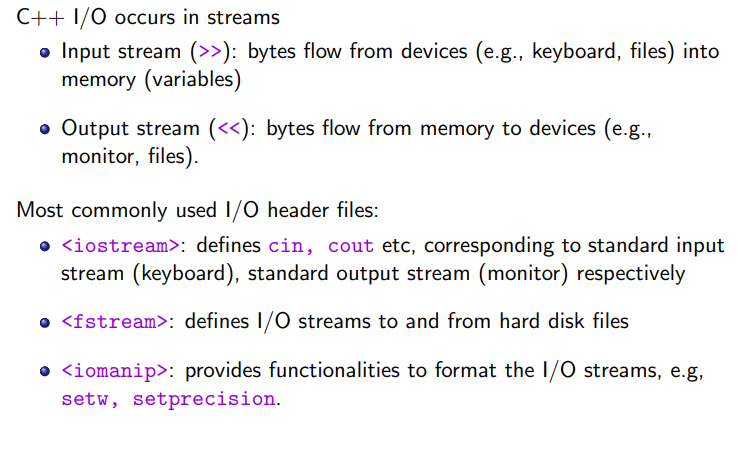
std::cout << "The factorial of " << number << " is: " << factorial << std::endl;

return 0;

}

### Input and Output

In C++, I/O (Input/Output) refers to the process of taking input from the user (keyboard, file, etc.) and providing output to the user (console, file, etc.). This functionality is provided through the iostream library, which contains predefined classes and functions for handling input and output operations.



Input (cin): console input

#include <iostream>

using namespace std;

int number;

cout << "Enter a number: ";

cin >> number;

Output (cout): stands for "console output."

cout << "Hello, World!" << endl;

### C++ I/O Example to Console

#include <iostream>

#include <string>

int main() {

// Output: Displaying text to the console

std::cout << "Enter your name: ";

// Input: Reading user input using std::cin

std::string name;

std::getline(std::cin, name); // store the cin result into a string variable

// Output: Displaying the user's name

std::cout << "Hello, " << name << "!" << std::endl;

// Input: Reading numeric input

std::cout << "Enter your age: ";

int age;

std::cin >> age;

// Output: Displaying the user's age

std::cout << "You are " << age << " years old." << std::endl;

return 0; }

### C++ I/O example to File

#include <iostream>

#include <fstream>

#include <string>

int main() {

// Writing to a file

std::ofstream outputFile("output.txt");

if (outputFile.is\_open()) {

std::cout << "Enter a line to write to the file (type 'exit' to stop):" << std::endl;

std::string inputLine;

while (true) {

std::getline(std::cin, inputLine);

if (inputLine == "exit")

break;

outputFile << inputLine << std::endl;

}

std::cout << "Data written to the file 'output.txt'." << std::endl;

// Close the file

outputFile.close();

}

else {

std::cerr << "Error opening the file for writing." << std::endl;

return 1;

}

// Reading from a file

std::ifstream inputFile("output.txt");

if (inputFile.is\_open()) {

std::cout << "Contents of the file 'output.txt':" << std::endl;

std::string line;

while (std::getline(inputFile, line)) {

std::cout << line << std::endl;

}

// Close the file

inputFile.close();

}

else {

std::cerr << "Error opening the file for reading." << std::endl;

return 1;

}

return 0;

}

### Exception handling

C++ supports several types of error handling mechanisms to address issues that occur during program execution. Here are the main ones:

1. Exception Handling (Using try, catch, and throw)

try {

throw "An error occurred";

}

catch (const char\* msg) {

cout << msg << endl;

}

1. Customize error message: You can use cerr (standard error output) to manually display error messages to the user.

#include <iostream>

using namespace std;

int main() {

cerr << "An error occurred!" << endl;

return 1;

}

More examples of error handling refer to error\_handling.cpp

Explanation

* The divideNumbers function takes two integers as parameters and checks if the denominator is zero. If it is, it throws a std::runtime\_error exception with an error message.
* In the main function, we have a try block where we call divideNumbers(10, 2); which should not throw an exception, and then we have a commented line // divideNumbers(5, 0); which, if uncommented, would result in an exception due to division by zero.
* The catch block catches any exceptions that are thrown within the try block. It prints an error message using std::cerr.
* Note: It's a good practice to catch exceptions by reference (const std::exception& e) to ensure proper handling and avoid object slicing.
* This is a basic example, and in real-world scenarios, you might create your own exception classes for more specific error handling.

### Note: What Happens When a runtime error is Thrown?

If the exception is handled using a try-catch block, the program will continue executing after the block finishes. Example

try {

throw std::runtime\_error("Something went wrong!");

}

catch (const std::runtime\_error& e) {

std::cerr << "Caught exception: " << e.what() << std::endl;

}

std::cout << "Program continues..." << std::endl;

If the exception is not caught, the program will terminate and output an error message, often in the form:

terminate called after throwing an instance of 'std::runtime\_error'

what(): Something went wrong!

The termination occurs because the exception propagates to the top level of the program without finding a matching catch block. This causes the program to invoke the default exception handler, which calls std::terminate().

## Homework assignment

## Number Operations Program

## Objective:

## Write a C++ program that takes a series of numbers as input from the console, performs calculations (e.g., sum and average of all odd number), and writes the results to a file.

Steps:

Step 1: Create the Program Structure: start by including the necessary headers: <iostream> for input/output and <fstream> for file operations

Step 2: Input from the User

* Declare a variable to store the number of inputs (e.g., numCount).
* Use a loop (for loop or while loop) to allow the user to input multiple numbers into an array or a vector.

Step 3: Perform Calculations

* Use if clause to check the value input is a odd number
* If yes, perform the sum

Step 4: Write Results to a File

Step 5: Upload both cpp file and result file into e-learning.

## Appendix

### Quiz

1. Question: What is the size of the int data type in C++ on a 32-bit system?

a) 2 bytes

b) 4 bytes

c) 8 bytes

d) It depends on the compiler

1. Question: Which data type is used to store single-precision floating-point numbers in C++?

a) float

b) double

c) long double

d) float32

1. Question: In C++, what is the maximum value that can be stored in an unsigned short?

a) 32767

b) 65535

c) 2147483647

d) 4294967295

1. Question: How do you declare a character constant in C++?

a) char constant = 'A';

b) const char character = 'A';

c) constant character = 'A';

d) character constant := 'A';

1. Question: Which data type is used to represent true or false values in C++?

a) bool

b) int

c) char

d) double

1. Question: What is the difference between signed and unsigned data types in C++?

a) signed is used for integers, and unsigned is used for floating-point numbers

b) signed can represent both positive and negative values, while unsigned represents only positive values

c) There is no difference; the terms are interchangeable

d) unsigned is used for integers, and signed is used for floating-point numbers

1. Question: In C++, how do you initialize a variable at the time of declaration?

a) initialize int x = 10;

b) int x; x = 10;

c) int x(10);

d) int x = 10;

1. Question: What is the scope of a local variable in C++?

a) It is accessible throughout the entire program

b) It is accessible only within the function or block where it is declared

c) It is accessible in any function within the same file

d) It is accessible in any function within the program

1. Question: What is the correct syntax to declare a pointer variable in C++?

a) int \*ptr;

b) pointer int ptr;

c) ptr \* int;

d) declare pointer int ptr;