No.3(a)

1. Arrays

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

Declaration: data\_type array\_name[array\_size];

Example: int scores[5];

Deployment Steps:

1. Declare the array with a specific data type and size.

2. Initialize the array using curly brackets {} or assign values individually.

3. Access array elements using their index (starting from 0).

4. Modify array elements using their index.

5. Perform array operations like searching, sorting, and it

Applications:

1. Store and process collections of data.

2. Implement algorithms (e.g., sorting, searching).

3. Represent vectors or matrices in mathematical computations.

1. Multi-Dimensional Arrays

Declaration: data\_type array\_ name[row\_size] [column\_size];

Example: int matrix[3][4];

Deployment Steps:

1. Declare the multi-dimensional array with specific row and column sizes.

2. Initialize the array using curly brackets {} or assign values individually.

3. Access array elements using their row and column indices (starting from 0).

4. Modify array elements using their row and column indices.

5. Perform array operations like iterating, searching, and matrix calculations.

Applications:

1. Represent matrices in linear algebra and calculus.

2. Store and process image or game board data.

3. Implement neural networks or machine learning algorithms.

Key Points:

1. Arrays and multi-dimensional arrays can be used to store and manipulate collections of data.

2. Arrays are zero-indexed, meaning the first element is at index 0.

3. Multi-dimensional arrays can be thought of as arrays of arrays.

4. Iterating through arrays and multi-dimensional arrays can be done using nested loops.

Best Practices:

1. Use meaningful variable names.

2. Initialize arrays properly.

3. Check array bounds to avoid errors.

4. Use functions to perform array operations

No 5c

) Single-Line Comments

- Why: Single-line comments are used to provide brief explanations or notes about a specific line or section of code. They help clarify the code's intent and make it easier to understand.

- How: Single-line comments start with // (two forward slashes). Anything following // on the same line is ignored by the compiler.

```

```

// This is a single-line comment

int x = 5; // This is another single-line comment

\* \*\*When:\*\* Use single-line comments:

\* To explain variable declarations or assignments.

\* To clarify complex expressions or logic.

\* To provide context for a specific section of code.

\* To temporarily disable a line of code (during debugging).

\*\*(ii) Multi-Line Comments\*\*

\* \*\*Why:\*\* Multi-line comments are used to provide longer explanations, descriptions, or documentation for code blocks, functions, or classes. They help organize and clarify larger sections of code.

\* \*\*How:\*\* Multi-line comments start with `/\*` and end with `\*/`. Anything between `/\*` and `\*/` is ignored by the compiler.

cpp

/\*

This is a multi-line comment

that spans multiple lines.

\*/

- \*When:\* Use multi-line comments:

- To document functions, classes, or modules.

- To explain complex algorithms or logic.

- To provide licensing or copyright information.

- To temporarily disable large blocks of code (during debugging).

\*Best Practices:\*

1. Use comments consistently throughout your code.

2. Keep comments concise and focused on the relevant code.

3. Avoid unnecessary comments; let the code speak for itself.

4. Update comments when modifying code.

5. Use Doxygen-style comments for documenting functions and classes.

\*Doxygen-Style Comments\*

Doxygen is a popular documentation generator for C++. Use Doxygen-style comments to document functions, classes, and modules:

/\*\*

\* @brief Calculates the area of a rectangle.

\*

\* @param width The width of the rectangle.

\* @param height The height of the rectangle.

\* @return The area of the rectangle.

\*/

int calculateArea(int width, int height) {

return width \* height;

}

```

Single-line Comments

Why to use:

To provide brief explanations or notes about specific lines of code.

To temporarily disable a line of code for debugging purposes.

How to use:

Start the comment with //. Everything following // on that line will be treated as a comment.

int main() {

int a = 5; // Initialize variable a with 5

// std::cout << a; // This line is commented out

return 0;

}

When to use:

When you need to add a quick note or explanation.

When you want to comment out a single line of code during debugging.

Multi-line Comments

Why to use:

To provide detailed explanations or documentation for a block of code.

To comment out multiple lines of code at once.

How to use:

Enclose the comment block with /\* at the beginning and \*/ at the end.

int main() {

/\* This is a multi-line comment.

It can span multiple lines.

Useful for detailed explanations. \*/

int a = 5;

return 0;

}

When to use:

When you need to write longer comments that span multiple lines.