Object Oriented Programming through C++

UNIT-4

Handling Data Files (Sequential and Random), Opening and Closing of Files

Introduction:

Data files are an essential part of many software applications as they allow for the storage and retrieval of data. In this topic, we will discuss handling data files, specifically sequential and random access files, and the process of opening and closing files.

Sequential Access Files:

1. Sequential access files store data records one after another in a linear fashion. To read or write data in a sequential file, you need to start at the beginning and go through the records sequentially until you reach the desired position.

- Opening a Sequential File:
    - To open a sequential file for writing, use the file open mode "w" or "a" (for appending).
    - To open a sequential file for reading, use the file open mode "r".
- Writing to a Sequential File:
    - Open the file in write mode using the appropriate file open mode.
    - Use the file write operations (e.g., fwrite) to write data to the file.
    - Close the file after writing.
- Reading from a Sequential File:
    - Open the file in read mode using the appropriate file open mode.
    - Use the file read operations (e.g., fread) to read data from the file.

- Close the file after reading.

Random Access Files:

2. Random access files allow for direct access to any record in the file. Each record has a unique identifier or key that allows for random retrieval of specific records without traversing through the entire file.

- Opening a Random Access File:
    - To open a random access file, use the file open mode "r+", "w+", or "a+".
    - The "r+" mode allows both reading and writing, while "w+" and "a+" modes create a new file if it doesn't exist.
- Writing to a Random Access File:
    - Open the file in write or append mode using the appropriate file open mode.
    - Use file positioning functions (e.g., fseek) to move the file pointer to the desired record.
    - Use the file write operations (e.g., fwrite) to write data to the file.
    - Close the file after writing.
- Reading from a Random Access File:
    - Open the file in read mode using the appropriate file open mode.
    - Use file positioning functions (e.g., fseek) to move the file pointer to the desired record.
    - Use the file read operations (e.g., fread) to read data from the file.
    - Close the file after reading.

3. Opening and Closing Files:
    - Opening a file:
        - Specify the file path and name, and the desired file open mode.
        - Check if the file was opened successfully to ensure proper error handling.
    - Closing a file:

- After reading or writing data to a file, it is essential to close the file.
- Closing a file frees system resources and ensures data integrity.

Handling data files involves understanding sequential and random access files and knowing how to open and close files correctly. Sequential access files are read or written in a linear fashion, while random access files allow direct access to specific records. Opening and closing files properly is crucial for efficient data handling and resource management in software applications.

Generic Programming Using Templates - Need & Importance of Templates, Function Template and Class Template

Introduction:

Generic programming using templates is a powerful feature in programming languages that allows the creation of reusable code for different data types. Templates enable the writing of generic algorithms and data structures that can work with various types without the need for duplication or code modification.

1. Need and Importance of Templates:
    - Code Reusability: Templates allow developers to write code once and use it with different data types, avoiding code duplication.
    - Type Safety: Templates provide type checking at compile-time, ensuring that operations are performed on appropriate data types.
    - Increased Productivity: By using templates, developers can focus on writing generic algorithms or data structures, leading to more efficient and productive coding.
2. Function Templates:
    - Function templates define generic functions that can work with multiple data types.

- Syntax: `template<typename T> returnType functionName(parameters) { // Code implementation }`
- T is a placeholder representing the generic data type.
- Example:

```
template<typename T> T maximum(T a, T b) { return (a > b) ? a : b;
```

3. Class Templates:
    - Class templates allow the creation of generic classes that can work with different data types.
    - Syntax:

```
template<typename T> class ClassName { // Class definition };
```

    - T is a placeholder representing the generic data type.
    - Example:


```
template<typename T> class Stack { private: T data[SIZE]; int top; public: // Member function declarations and definitions };
```

4. Template Specialization:
    - Template specialization allows defining specific implementations for certain data types.
    - Specializations can be used to handle specific cases differently or optimize performance.
    - Syntax:

```
template<> returnType functionName<specificType>(parameters) { // Specialized implementation }
```

    - Example:

```
template<> double maximum<double>(double a, double b) { return (a > b) ? a : b; }
```

Templates in generic programming provide a flexible and efficient way to write reusable code that can work with multiple data types. Function templates allow generic functions, while class templates enable the creation of generic classes. Template specialization allows customization for specific data types, enhancing the versatility of generic programming.

Exception Handling – Need of Exception Handling, Throw, Try, Catch Block

Introduction:

Exception handling is a mechanism in programming that deals with exceptional or erroneous situations during the execution of a program. It allows developers to gracefully handle errors, recover from unexpected conditions, and ensure the stability and reliability of the software.

1. Need of Exception Handling:
    - Error Handling: Exception handling provides a structured approach to handle errors and exceptional conditions that may occur during program execution.
    - Program Flow Control: By catching and handling exceptions, developers can control the flow of the program and take appropriate actions.
    - Robustness: Exception handling enhances the robustness of the software by preventing crashes or unexpected termination due to unhandled errors.
2. Throw Statement:
    - The throw statement is used to raise an exception explicitly.
    - Syntax:
    - arduino
    - Copy code
    - throw exceptionObject;
    - The exceptionObject can be of any data type, including built-in types or custom-defined exception classes.
3. Try-Catch Block:

- The try-catch block is used to catch and handle exceptions.
- Syntax:

```
try { // Code that may throw an exception } catch (exceptionType1 ex1) { // Code to handle exceptionType1 } catch (exceptionType2 ex2) { // Code to handle exceptionType2 } catch (...) { // Code to handle any other exception }
```

- The try block encloses the code that may throw an exception.
- The catch blocks specify the exception types to catch and the corresponding code to handle those exceptions.
- The ellipsis (...) catch block is used as a catch-all for any other unhandled exceptions.

4. Exception Class Hierarchy:
    - Exception handling often involves defining custom exception classes to represent specific types of errors.
    - Custom exception classes can be derived from the standard exception class or its derived classes, like runtime_error or logic_error.
5. Handling Exceptions:
    - In the catch block, various actions can be taken to handle exceptions, such as displaying error messages, logging, or performing specific recovery operations.
    - The catch block can also rethrow the exception using the throw statement to propagate it to an outer level of exception handling

Exception handling is a crucial aspect of software development, providing a structured way to handle errors and exceptional conditions. The throw statement raises exceptions, and the try-catch block catches and handles them appropriately. Exception handling enhances program stability, error recovery, and overall software robustness.