Aromal S Kunnel

22BAI10288

ASSIGNMENT-V

Branch/Semester	**B.Tech/Fall semester**	**Session**	**2024-2025**
Name of Faculty	**Dr. Jitendra P S Mathur**	**Subject**	**Object Oriented Programming With C++**
Module	**5**	**Sub Code**	**CSE-2001**
Last date of Submission	**15.08.2024**		**Through Google classroom**

S.No	Questions	CO Attainment
1	Explain the role of seekg(), seekp(), tellg(), tellp(), function in the process of random access in a file.	CO.5
2	Differentiate between opening a file with constructor function and opening a file with open ( ) function.	CO.5
3	Write a C++ program that reads a file and writes to another file after converting every character into upper case letter.	CO.5
4	Explain in brief various functions required for random access file operations. Write a C++ program to update the contents of a file by accessing the contents randomly.	CO.5

A1) Understanding Random Access with seekg(), seekp(), tellg(), and tellp()

In C++, random access to files involves manipulating file pointers to read or write data at specific locations within a file. The functions `seekg()`, `seekp()`, `tellg()`, and `tellp()` are integral to this process.

Get Pointer and Put Pointer

Before diving into the functions, it's essential to understand that C++ maintains two separate pointers for a file stream:

- **Get pointer:** Used for reading operations.
- **Put pointer:** Used for writing operations.

The Functions

seekg() and seekp()

These functions are used to set the position of the respective pointer within a file.

- **Syntax:**
  C++
  ```
  file_object.seekg(offset, reference_point);
  file_object.seekp(offset, reference_point);
  ```
- **Parameters:**
  - `offset`: The number of bytes to move the pointer relative to the reference point.
  - `reference_point`: Specifies the starting point for the offset calculation.
    - `ios::beg`: Beginning of the file
    - `ios::cur`: Current position of the pointer
    - `ios::end`: End of the file
- **Example:**
  C++
  ```
  fstream file("data.txt", ios::in | ios::out);
  ```

```
file.seekg(10, ios::beg); // Move get pointer 10 bytes from the
beginning
file.seekp(20, ios::end); // Move put pointer 20 bytes from the end
```

tellg() and tellp()

These functions return the current position of the respective pointer within a file.

- **Syntax:**
  C++
  ```
  file_object.tellg();
  file_object.tellp();
  ```
- **Return value:** The current position of the pointer as a long integer representing the number of bytes from the beginning of the file.
- **Example:**
  C++
  ```
  fstream file("data.txt", ios::in);
  long current_position = file.tellg(); // Get the current position of
  the get pointer
  ```

Random Access Process

To perform random access in a file, you typically follow these steps:

1. **Open the file:** Use `fstream` or other file stream objects to open the file in appropriate mode (e.g., `ios::in`, `ios::out`, `ios::binary`).
2. **Calculate the offset:** Determine the position where you want to read or write data.
3. **Use `seekg()` or `seekp()`:** Move the appropriate pointer to the desired position.
4. **Read or write data:** Use `read()` or `write()` functions to perform the operation.
5. **Use `tellg()` or `tellp()` (optional):** Check the current position of the pointer if needed.

Example

C++

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    fstream file("data.bin", ios::in | ios::out | ios::binary);

    // Write some data
    int data = 42;
    file.seekp(0, ios::beg); // Move put pointer to the beginning
    file.write((char*)&data, sizeof(int));

    // Read the data back
    file.seekg(0, ios::beg); // Move get pointer to the beginning
    int read_data;
    file.read((char*)&read_data, sizeof(int));
    cout << read_data << endl; // Output: 42

    file.close();
    return 0;
}
```

Key points to remember:

- `seekg()` and `tellg()` are used for input operations.
- `seekp()` and `tellp()` are used for output operations.
- The reference points `ios::beg`, `ios::cur`, and `ios::end` can be used with both `seekg()` and `seekp()`.
- For binary files, it's crucial to use `ios::binary` mode when opening the file.

By effectively using these functions, you can efficiently manipulate file data at specific locations, enabling random access capabilities in your C++ programs.

A2) Opening a File in C++: Constructor vs. open() Function

Constructor-Based Opening

- **Syntax:**
  C++
  ```cpp
  ifstream infile("filename.txt");
  ofstream outfile("output.txt");
  fstream file("data.txt", ios::in | ios::out);
  ```

- **Characteristics:**
  - File name is mandatory during object creation.
  - Provides basic opening modes (in, out, in | out, etc.).
  - Suitable for simple scenarios with a single file.

open() Function-Based Opening

- **Syntax:**
  C++
  ```
  ifstream infile;
  infile.open("filename.txt");
  ofstream outfile;
  outfile.open("output.txt", ios::app);
  ```
- **Characteristics:**
  - File name can be provided later after object creation.
  - Offers more flexibility in specifying opening modes (e.g., app, binary, etc.).
  - Useful for handling multiple files or dynamic file names.

Key Differences

Feature	Constructor	open() function
File name	Mandatory	Optional
Opening modes	Limited	More flexible
Use cases	Simple, single file	Multiple files, dynamic file names

When to Use Which

- **Use constructor:**
  - When you know the file name beforehand and require simple opening modes.

    - For a straightforward approach with a single file.
- **Use open() function:**
    - When you need to determine the file name at runtime or dynamically.
    - For complex file operations requiring specific opening modes.
    - When handling multiple files within the same program.

Example:

C++

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    // Constructor-based opening
    ifstream infile("input.txt");
    if (!infile) {
        cerr << "Error opening input file" << endl;
        return 1;
    }

    // open() function-based opening
    ofstream outfile;
    string outputFileName = "output.txt";
    outfile.open(outputFileName, ios::app);
    if (!outfile) {
        cerr << "Error opening output file" << endl;
        return 1;
    }

    // ... file operations ...

    infile.close();
    outfile.close();
    return 0;
}
```

In conclusion, both methods serve the purpose of opening files in C++, but the choice between them depends on the specific requirements of your program.

Would you like to see an example of using these methods in a specific scenario?

A3) Code:

```
#include <iostream>
#include <fstream>
#include <cctype>

using namespace std;

int main() {
    ifstream inputFile("input.txt");
    ofstream outputFile("output.txt");

    if (!inputFile) {
        cerr << "Error opening input file" << endl;
        return 1;
    }

    if (!outputFile) {
        cerr << "Error creating output file" << endl;
        return 1;
    }

    char ch;
    while (inputFile.get(ch)) {
        outputFile << static_cast<char>(toupper(ch));
    }

    inputFile.close();
    outputFile.close();

    cout << "File conversion complete!" << endl;

    return 0;
}
```

A4) Functions for Random Access File Operations:

- **open():** Opens a file for input, output, or both modes.
- **close():** Closes an open file.
- **seekg():** Sets the position of the get pointer within a file.
- **seekp():** Sets the position of the put pointer within a file.
- **tellg():** Returns the current position of the get pointer.
- **tellp():** Returns the current position of the put pointer.
- **read():** Reads a specified number of characters from the file.

- **write():** Writes a specified number of characters to the file.

C++ Program to Update File Contents Randomly:

C++

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    fstream file("data.txt", ios::in | ios::out | ios::binary);

    if (!file) {
        cerr << "Error opening file" << endl;
        return 1;
    }

    // Update the content at position 10
    int position = 10;
    char new_data = 'X';

    // Move the file pointer to the desired position
    file.seekp(position);

    // Write the new data
    file.write(&new_data, 1);

    file.close();

    cout << "File updated successfully!" << endl;

    return 0;
}
```

Explanation:

1. Include necessary headers.
2. Open the file in input, output, and binary mode.
3. Check if the file is opened successfully.
4. Specify the position to update and the new data.
5. Use `seekp()` to move the put pointer to the desired position.
6. Use `write()` to write the new data.
7. Close the file.
8. Print a success message.