C++ Week 9

Khaula Molapo

1.

In an e-commerce system, exception handling is very important to keep everything running smoothly even when errors happen. For example, when a customer tries to make a payment, things can go wrong, such as network failure, invalid card details, or the payment gateway being down. Without exception handling, the system could crash or show confusing messages to the user. Instead, the program can catch these errors and show a friendly message like “Payment failed, please try again later.” The system can also log the problem for the developers to fix later. This helps protect user trust and prevents data loss or wrong transactions. Exception handling is also used in other areas, such as adding items to a cart, updating inventory, or processing refunds. By handling exceptions properly, the e-commerce platform becomes more reliable, stable, and user-friendly, which helps both the business and the customers have a smoother experience overall.

2.

Standard exceptions in C++ are built-in error types that help developers handle common problems in their programs. Some of the most used standard exceptions include std::runtime_error, std::invalid_argument, std::out_of_range, and std::overflow_error. These exceptions come from the C++ Standard Library and make it easier to manage errors in a structured way. For example, std::out_of_range can be used when accessing an element outside the size of a container like a vector, while std::invalid_argument helps when a function receives an invalid input. Using these built-in exceptions helps programmers avoid writing the same error-handling code over and over. It also makes debugging easier because the type of error clearly describes what went wrong. Overall, standard exceptions make code safer, more readable, and less likely to crash unexpectedly, which is important for maintaining professional and stable software.

3.

In Visual Studio Code, I created a custom exception in C++ to handle specific problems that standard exceptions don’t fully describe. For example, I made a class called InvalidLoginException that inherits from std::exception. This class shows a custom message like “Invalid username or password” when thrown. Here’s a simple example:

```
#include <iostream>
#include <exception>
using namespace std;

class InvalidLoginException : public exception {
public:
   const char* what() const noexcept override {
      return "Invalid login credentials.";
```

```
    }
};

int main() {
    try {
        throw InvalidLoginException();
    } catch (const exception& e) {
        cout << e.what();
    }
}
```

Creating custom exceptions like this helps make programs more flexible and readable. Instead of using one general error for everything, each situation has its own message, making debugging and user experience much easier and clearer.

4.

An exception-handling function is a special function designed to detect and manage errors in a program without stopping it completely. For example, a function called handleFileError() can catch exceptions that occur when opening or reading a file. It could look like this:

```
void handleFileError() {
    try {
        throw runtime_error("File not found");
    } catch (const exception& e) {
        cout << "Error: " << e.what() << endl;
    }
}
```

The purpose of such a function is to make the program more stable and user-friendly. Instead of crashing when an error happens, the program catches the problem, shows a clear message, and maybe allows the user to try again. This makes it easier for users to continue using the software, and for developers to fix issues quickly. Exception-handling functions improve reliability and overall software quality.