A project report on

**ONLINE FOOD ORDERING SYSTEM**

**Electronics and communication Engineering**

**By**

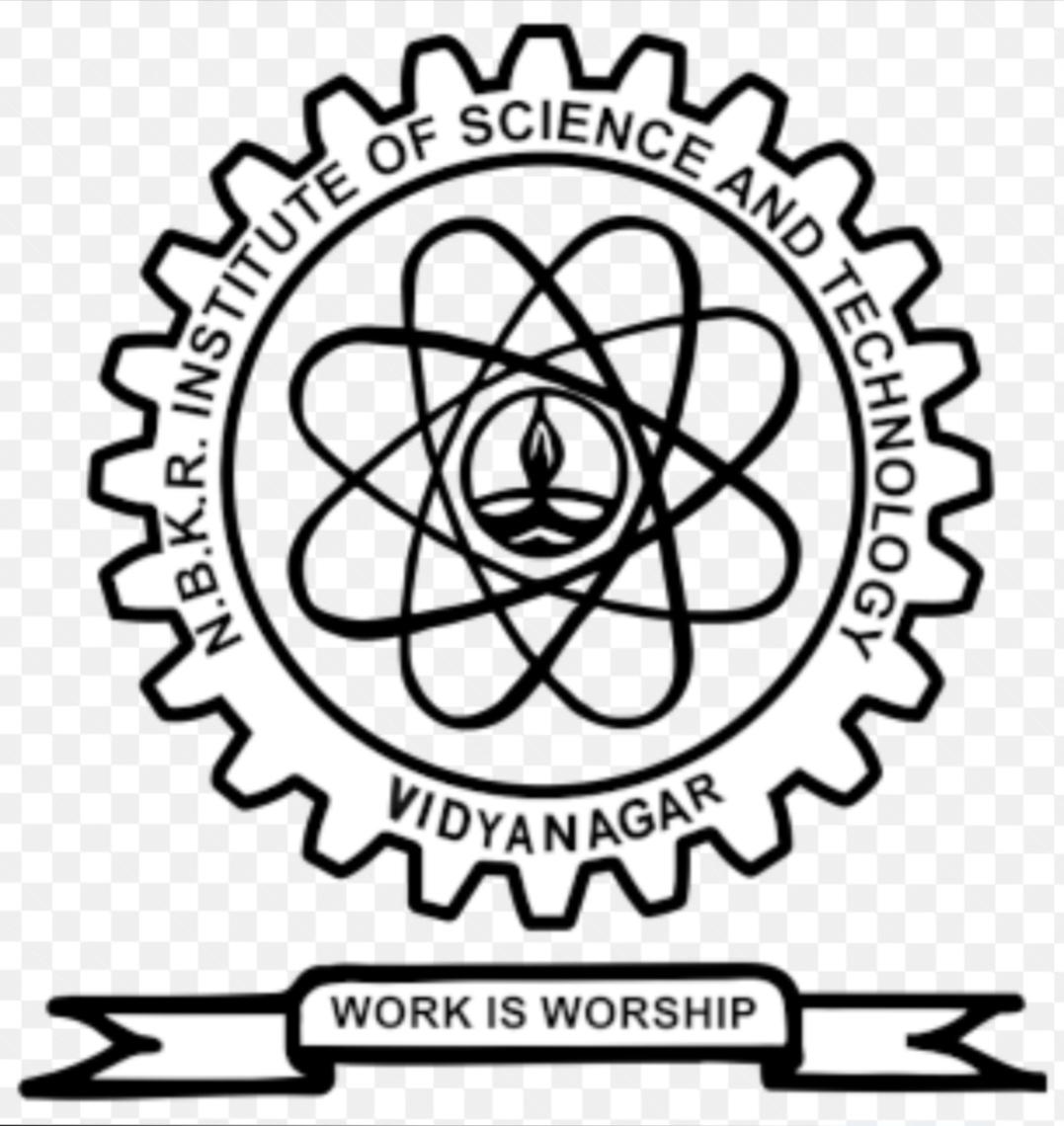
**P.spandana -- 24KB1A04R4**

**K.Geethika -- 24KB1A04G9**

**MM.Nahila baig -- 24KB1A04L9**

**Under the Esteemed supervision of**

**A.V.V SATHISH(TRAINER)**

****

**Department of Electictonics and communication Engineering**

**N.B.K.R INSTITUTE OF SCIENCE AND TECHNOLOGY**

**( An Autonomous Institution)**

**(Affiliated to JNTUA,Accredited By NAAC with ‘A’ Grade)**

**Vidya nagar , Tirupati (dist)**

**DECLARATION**

We hereby declare that the project entitled **“ONLINE FOOD ORDERING SYSTEM ”** is a genuine project. This work has been submitted to the **N.B.K.R INSTITUTEOFSCIENCEANDTECHNOLOGY,** vidyanagar, permanently

Affiliated to JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY,

ANATHAPUR partial fulfillment of the B.Tech degree. We further declare that

This project work has not been submitted in full or part for the award of any degree

Of this or any other educational institutions.

**BY**

**PARICHERLA SPANDANA (24KB1A04R4)**

**KONDURU GEETHIKA (24KB1A04G9 )**

**MIRJA MOHAMMED NAHILA BIAG ( 24KB1A04L9 )**

**ACKNOWLEDGEMENT**

It is with immense pleasure that we would like to express our Gratitude our project guide **A.V.V SATHISH(trainer)**, who has guided us a lot And encouraged us in every step of the project work, her valuable moral support And guidance throughout the project helped us to a greater extent.

Our deepest thanks to our HOD. , Associate Professor for Inspiring us all the way and for arranging all the facilities and resources needed for Our project.

We wish to thank **A.V.V SATHISH SIR**, trainer in bytexl for his support and suggestions during our project work.

We are thankful to our College Management for providing all the facilities In time to us for completion of our project.

Not to forget, Faculty, Lab Technicians, non-teaching staff and our Friends who have directly or indirectly helped and supported us in completing our Project in time.

**INDEX**

**Concept**

**Chapter 1 INTRODUCTION:**

* 1. **History about online food ordering system**
  2. **Techniques**
  3. **Working**
  4. **Overview and objective**

**Chapter 2 REQUIREMENTS ANALYSIS**

* 1. **Introduction of requirements analysis**
  2. **Hardware and software requirements**

**Chapter 3 SYSTEM DESIGN:**

* 1. **Introduction to system design**
  2. **Data dictionary**
  3. **Logical Data base design**
  4. **UML diagrams**

**Chapter 4 CODE IMPLEMENTATION:**

* 1. **Introduction**
  2. **Selected Software**
  3. **Sample code**

**Chapter 5 SIMULATION RESULT:**

* 1. **Introduction**
  2. **Test cases**

**Chapter 6 ADVANTAGES AND APPLICATIONS:**

**Chapter 7 CONCLUSION:**

**CHAPTER --1**

**INTRODUCTION**

**Based on the C program you’ve shared (a simple Online Food Ordering System implemented using linked lists), here’s a tailored introduction of the project area suitable for inclusion in your project documentation:**

**Introduction to the Project Area**

**The project falls under the domain of Software Development and E-Commerce Systems, with specific relevance to online food service automation. This project aims to simulate the basic operations of an online food ordering platform using the C programming language. By utilizing data structures such as linked lists, the system manages menu items and customer orders (cart items) efficiently in memory.**

**The core objective of this project is to provide a simple command-line interface that mimics real-world food ordering systems. The functionalities include viewing a menu, adding items to a cart, viewing the cart, and placing an order to generate a bill. It emphasizes fundamental programming concepts such as dynamic memory allocation, structs, and user interaction through a console.**

**Although it’s a basic simulation, the project introduces important concepts that form the foundation for more complex applications in food tech, including order management systems, inventory tracking, and customer service platforms. It can also serve as a stepping stone for extending to GUI-based applications or web/mobile platforms in the future.**

**1.Dynamic Memory Allocation**

**In the program, malloc() is used to allocate memory for new MenuItem and CartItem nodes at runtime. This allows the program to handle an unknown number of items without pre-defining the size of the menu or cart:**

**MenuItem\* newItem = (MenuItem\*)malloc(sizeof(MenuItem)); CartItem\* newCartItem = (CartItem\*)malloc(sizeof(CartItem));**

**This dynamic memory allocation provides flexibility and efficient memory usage, especially when the number of items varies.**

**2.Structs (Structures)**

**Structs are used to define custom data types for the menu and cart items:**

**Typedef struct MenuItem { int id; char name[50]; float price; struct MenuItem\* next; } MenuItem;**

**This allows the grouping of related data (like id, name, price, and next pointer) into a single unit, making it easy to manage and extend complex data in a structured way.**

**3.User Interaction Through Console**

**The program interacts with the user via the command-line interface using printf() to display menus and options, and scanf() to read user input:**

**Printf(“Enter choice: “); scanf(“%d”, &choice);**

**This provides a simple, text-based way for users to:**

**• View the menu**

**• Add items to the cart**

**• View the cart**

**• Place an order**

**The use of loops and switch statements enables continuous interaction until the user exits.**

**The program you provided is console-based, meaning it interacts with users through text input/output in a terminal. However, a GUI-based (Graphical User Interface) application is the visual version of this system, where users interact through buttons, menus, forms, and dialog boxes instead of typing commands.**

**GUI-Based Version of This Program**

**If the same Online Food Ordering System were made as a GUI application, it might include:**

**Key Components:**

**• Menu Display Window:**

**• A list or grid showing all food items with names, prices, and “Add to Cart” buttons.**

**• Optionally, images of the food.**

**• Cart Interface:**

**• A separate section or window showing added items, their quantities, and total price.**

**• Buttons to increase/decrease quantity or remove items.**

**• Checkout Window:**

**• Displays a detailed bill.**

**• Includes fields for user details (name, address).**

**• A “Place Order” button to complete the transaction.**

**Navigation:**

**• Menus or tabs for “Home,” “Menu,” “Cart,” and “Order History.”**

**Technologies to Implement GUI:**

**• C with GTK or WinAPI: For native desktop GUIs.**

**• C++ with Qt: Easier for rich UIs.**

**• Python with Tkinter/PyQt: Beginner-friendly and suitable for prototypes.**

**• Java (Swing/JavaFX): For cross-platform desktop apps.**

**• Web-based GUI: Using HTML/CSS/JavaScript for a browser interface.**

**Summary:**

**A GUI-based version of your program provides a more user-friendly and visually appealing interface that’s intuitive, faster to use, and better suited for real users than the console version. It enhances usability by minimizing typing and relying on visual cues and clicks.**

**Advantages of GUI**

**• User-Friendly**

**• GUI allows users to interact through icons, buttons, and menus, making it easy even for non-technical users.**

**• Visual Appeal**

**• Attractive layouts, colors, and images improve the user experience and make applications more engaging.**

**• Intuitive Navigation**

**• GUI elements guide users naturally through the application, reducing the learning curve.**

**• Error Reduction**

**• Users are less likely to make input errors since options are visually displayed (e.g., dropdowns, buttons).**

**• Faster Interaction**

**• Tasks like selecting items or submitting forms are quicker with mouse clicks or taps rather than typing commands.**

**• Multitasking Support**

**• GUIs can allow multiple windows and tabs, enabling users to work on more than one task at a time.**

**• Accessibility Features**

**• GUI systems can support screen readers, zoom, and voice control, making applications more accessible.**

**• Real-Time Feedback**

**• GUI provides instant visual feedback (like loading bars or error messages), improving user satisfaction.**

**In short, GUI enhances usability, accessibility, and efficiency, making software more practical for everyday use compared to traditional command-line interfaces.**

**1.1 HISTORY ABOUT ONLINE FOOD ORDERING SYSTEM:**

**The concept of online food ordering dates back to the late 1990s and early 2000s when internet access became more widespread. Early systems allowed users to view a restaurant’s menu on a website and place orders via email or basic web forms. As technology evolved, more sophisticated platforms emerged, offering real-time order tracking, digital payments, and mobile apps.**

**Your C program represents a simplified, early-stage simulation of such systems. It mimics the core workflow that early online food systems used:**

**• Displaying a menu – like the original static web pages.**

**• Adding to a cart – simulates order customization.**

**• Viewing the cart and placing orders – represents the checkout and billing phase.**

**• Memory-based management – similar to how data was temporarily stored before being written to databases.**

**Although real-world systems now use databases, APIs, and front-end frameworks, your program demonstrates the foundational logic that powers modern food ordering apps.**

**Modern Evolution from This Concept**

**• 2000s: Basic web portals (like Grubhub, Just Eat) started.**

**• 2010s: Mobile apps became dominant (e.g., Uber Eats, Swiggy, Zomato).**

**• 2020s onward: AI, GPS tracking, and user personalization became standard.**

**In essence, your C program reflects the logical backbone of early online food systems—simple, efficient, and user-directed, just without a GUI or database backend.**

**1.2 Techniques Involved in the Program**

1. **Linked Lists**

**• Used to manage both the menu and cart dynamically.**

**• Each item (menu or cart) is a node linked to the next, allowing flexible addition/removal of items without fixed size limits.**

1. **Dynamic Memory Allocation**

**• malloc() is used to allocate memory at runtime for each new item added to the menu or cart.**

**• This technique enables the program to handle an arbitrary number of items efficiently.**

1. **Structures (struct)**

**• struct MenuItem and struct CartItem are used to define custom data types to store item details like ID, name, price, and quantity.**

**• Structs help encapsulate related data in a clean and manageable format.**

1. **String Handling**

**• strcpy() is used to copy item names into struct fields.**

**• Basic string manipulation is used to manage and display item names.**

1. **Console Input/Output**

**• printf() and scanf() handle user interaction.**

**• Menus and prompts are displayed in the console for a simple command-line interface.**

1. **Conditional and Looping Statements**

**• if, while, and switch-case structures are used for control flow.**

**• These help manage logic like adding items, viewing the cart, or exiting the program.**

1. **Function Decomposition**

**• The program is broken into separate functions for tasks like addMenuItem(), addToCart(), viewCart(), and placeorder().**

**• This improves readability, reusability, and modularity of the code.**

**These techniques together form the foundation for simple but effective software development, especially in C-based system simulations.**

**1.3 Working of the Program**

**The program is a console-based simulation of an online food ordering system using C language. It uses linked lists to store menu and cart items dynamically.**

1. **Menu Setup (Initialization)**

**• At the start of the program (main() function), a predefined set of menu items is added using the addMenuItem() function.**

**• These items are stored in a linked list (MenuItem struct).**

1. **User Menu Options**

**The user is repeatedly presented with a menu:**

1. **View Menu 2. Add to Cart 3. View Cart 4. Place Order 5. Exit**

**The user inputs a choice, and the program responds accordingly.**

1. **Option-wise Working**

**• View Menu (Option 1):**

**• Calls displayMenu(), which traverses the menuHead linked list and prints all available food items.**

**• Add to Cart (Option 2):**

**• Asks the user for an Item ID and quantity.**

**• Searches the menu list for the item.**

**• If found:**

**• Checks if the item already exists in the cart.**

**• If yes, it updates the quantity.**

**• If not, it creates a new CartItem node and adds it to the cart list.**

**• View Cart (Option 3):**

**• Traverses the cartHead list and displays item names, quantities, and total prices.**

**• Place Order (Option 4):**

**• Displays all cart items and computes the total bill.**

**• Frees memory for all cart items after the order is placed (cart is emptied).**

**• Thanks the user.**

**• Exit (Option 5):**

**• Ends the program with a goodbye message.**

**Underlying Mechanisms**

**• Linked lists allow dynamic addition/removal of menu/cart items.**

**• Dynamic memory allocation is used to allocate memory for each item node.**

**• User input/output is handled through the console using scanf() and printf().**

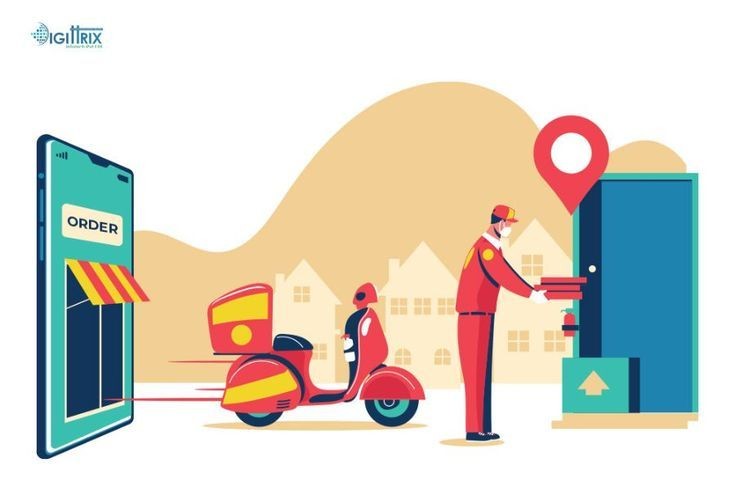
**This structure helps simulate the key features of an online food ordering system in a simplified, text-based format.**

**1.4 Overview**

**The Online Food Ordering System is a console-based application developed in the C programming language. It allows users to browse a predefined food menu, add items to a cart, view the cart, and place an order. The system is implemented using linked lists and dynamic memory allocation to manage the menu and cart items efficiently. It demonstrates core programming concepts such as structures, function modularity, and user interaction via console input/output.**

**This program simulates the basic working of real-world online food ordering platforms like Zomato or Swiggy, but in a simplified form suitable for learning and demonstration purposes.**

**1.4 Objectives**

**• To simulate an online food ordering workflow using fundamental C programming.**

**• To implement and manage menu and cart operations using linked lists for dynamic data handling.**

**• To demonstrate the use of structures (struct) for organizing complex data like food items and orders.**

**• To provide a basic user interface through the console for interacting with the system.**

**• To reinforce the use of dynamic memory allocation (malloc, free) for efficient memory management.**

**• To build a modular program where each functionality (viewing menu, adding to cart, placing order) is separated into functions for clarity and reusability.**

**CHAPTER—2**

**REQUIREMENTS ANALYSIS**

**The C program you provided is a simple online food ordering system using linked lists. Here’s a summary of the tools (concepts and techniques) used in it:**

1. **Structures (struct)**

**MenuItem and CartItem are structures representing menu items and items in the user’s cart.**

1. **Linked Lists**

**Dynamic data structures are used to manage the menu and cart as singly linked lists.**

**Pointers (next) link items in both lists.**

1. **Dynamic Memory Allocation**

**Malloc() is used to allocate memory for new items in the menu and cart.**

1. **String Handling**

**Strcpy() is used to copy names of items between structures.**

1. **Standard I/O**

**Printf() and scanf() handle user interaction and output.**

1. **Control Structures**

**If, while, and switch-case are used for logic control and menu selection.**

1. **Loops**

**Do-while is used for the main menu loop.**

**While loops traverse linked lists.**

1. **Functions**

**Modular code design with functions like addMenuItem(), displayMenu(), addToCart(), etc., each handling a specific task.**

**2.1 INTRODUCTION FOR REQUIREMENTS ANALYSIS**

**The purpose of this system is to simulate a basic online food ordering experience through a command-line interface. Users should be able to view a list of food items (menu), add items to a shopping cart, review their selections, and place an order which generates a final bill.**

1. **Functional Requirements**

These requirements describe the specific behaviors and functions the system must perform:

**Menu Management**

The system should maintain a predefined list of menu items, each with:

A unique ID (integer)

A name (string)

A price (floating-point)

Users should be able to view the entire menu at any time.

**Add to Cart**

Users can add items from the menu to their cart by entering the item’s ID and the desired quantity.

If an item is already present in the cart, its quantity should be updated rather than creating a duplicate entry.

**View Cart**

The system should allow users to view all items currently in the cart.

Each cart item should display:

Item name

Quantity selected

Total cost (price \* quantity)

**Place Order (Checkout)**

When the user chooses to place an order, the system should:

Display a detailed bill of all cart items.

Show the total amount due.

Thank the user for the order.

Clear the cart after checkout.

**Exit Program**

The system should provide a way for the user to exit the application safely.

Non-Functional Requirements

These define the performance and quality aspects of the application:

Usability

The program should have a simple, text-based interface suitable for beginner-level users.

Options and instructions should be clearly printed for the user to follow.

**2.Efficiency**

The application should respond quickly to inputs and operations should be efficient.

Linked lists are used for fast and flexible data handling without fixed size limits. Maintainability

The code is divided into modular functions (e.g., addToCart(), displayMenu(), placeOrder()) to ease future maintenance and enhancement.

Technical Constraints

**Language**

The system is implemented using C language with the standard I/O (stdio.h), string (string.h), and dynamic memory (stdlib.h) libraries.

**Platform**

The system runs in a terminal/console environment without graphical components.

**Data Persistence**

The application uses dynamic memory (linked lists) during runtime, but does not store data permanently. Once the program exits, all data (menu/cart) is lost.

* 1. **HARDWARE AND SOFTWARE REQUIREMENTS.**

This section outlines the minimum and recommended hardware and software configurations needed to successfully compile and run the Online Food Ordering System, which is implemented in the C programming language and operates through a command-line interface (CLI).

1. **Hardware Requirements**

The application is lightweight and does not require high-end hardware. It can be run on almost any standard computer system. Below is a detailed breakdown:

* 1. **Processor**

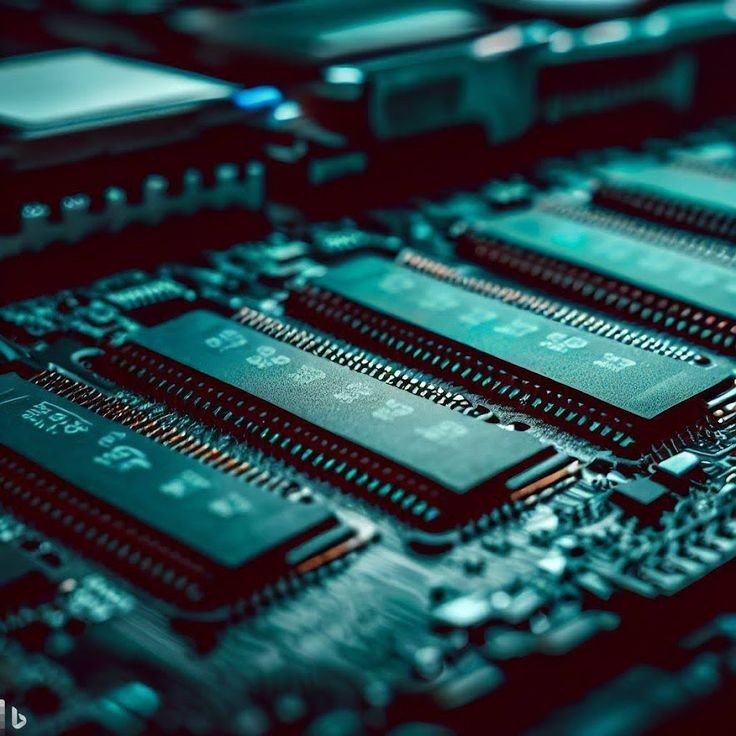
****

Minimum Requirement: An Intel Pentium IV or equivalent processor.

Recommended: Intel Core i3 or newer.

The program performs basic computational tasks like managing a menu and a shopping cart using linked lists. Therefore, any processor capable of running a terminal-based application will suffice.

* 1. **RAM** (Memory)



Minimum Requirement: 512 MB RAM.

Recommended: 1 GB or more.

****The system doesn’t consume much memory, but sufficient RAM is needed for compiling C programs and performing memory allocation (malloc) operations smoothly.

**1.3 storage:**

Minimum Requirement: At least 100 MB of free disk space.

Recommended: 500 MB or more.

The storage is needed for saving the source code, compiler, IDE (if used), and temporary files during compilation. No persistent data storage is required by the program itself.

* 1. **Display**

A standard VGA monitor is sufficient as the program operates in a console window. There are no graphical components.

* 1. **Input Devices**

Mandatory: A keyboard is essential for interacting with the menu system.

Optional: A mouse is not required but may be useful for navigating the IDE.

1. Software Requirements

To develop, compile, and run the program, a basic software environment is needed, with a C compiler and a text editor or integrated development environment (IDE).

* 1. **Operating System**

The program is platform-independent and can run on:

Windows (XP or later)

Linux distributions (such as Ubuntu, Fedora, etc.)

macOS

As the application runs in a terminal or command prompt, no OS-specific graphical environment is necessary.

* 1. **Compiler**

A C compiler is required to compile the source code. Common options include:

GCC (GNU Compiler Collection) – Standard on Linux and available on Windows via MinGW or Cygwin.

Clang – Lightweight and fast, commonly used on macOS.

Turbo C or TCC – Suitable for simple, DOS-style development environments.

* 1. **Text Editor or IDE**

You can write the C code using any basic text editor like Notepad, Notepad++, or use a full-featured IDE for better support:

Code::Blocks

Dev-C++

Turbo C++

Visual Studio Code (with C/C++ extension)

Eclipse CDT

These tools help manage code structure, compile/run the program, and provide features like syntax highlighting and debugging.

* 1. **Terminal or Console**

Since the system runs as a CLI (command-line interface), an appropriate terminal is required:

Windows: Command Prompt or PowerShell.

Linux/macOS: GNOME Terminal, Konsole, or Terminal.app.

1. **Libraries and Dependencies**

The program only uses standard C libraries, ensuring maximum portability:

#include <stdio.h> – Used for input and output functions like printf() and scanf().

#include <stdlib.h> – Required for memory management functions such as malloc() and free().

#include <string.h> – Provides functions like strcpy() for handling strings.

No third-party or external libraries are used, making the program lightweight and easy to compile on any C-supported environment.

1. **Summary**

The Online Food Ordering System is a console-based C application that can be developed and executed in a minimal software environment with modest hardware. Its simplicity ensures compatibility with most systems, and it serves as an excellent educational tool for practicing data structures like linked lists, memory management, and modular programming in C.

**Chapter – 3**

**SYSTEM DESIGN**

**3.1 Introduction to System Design**

The Online Food Ordering System implemented in C is a basic console-based application that simulates the operations of a food ordering platform. The system uses linked lists to dynamically manage a menu and a shopping cart, providing users with options to view the menu, add items to the cart, view the cart, and place an order.

System Components Overview:

**1.Menu Management (Singly Linked List):**

Maintains a list of available food items.

Each MenuItem node contains:

Id: Unique identifier for the item.

Name: Item name.

Price: Item price.

Next: Pointer to the next menu item.

**2.Cart Management (Singly Linked List):**

Tracks the user’s selected items.

Each CartItem node contains:

Id, name, price (copied from the menu item).

Quantity: Number of units ordered.

Next: Pointer to the next cart item.

Key Functional Modules:

**1.addMenuItem()**

Populates the menu list.

Used in main() to create a static list of food items.

**2.displayMenu()**

Prints all available food items to the console.

**3.addToCart()**

Adds items to the cart by matching the id from the menu.

Updates quantity if the item is already in the cart.

**4.viewCart()**

Displays all items currently in the cart with their quantities and total cost per item.

**5.Placeorder()**

Prints the final bill with the total amount.

Frees memory used by cart items to simulate order completion.

User Interaction Flow:

A menu is displayed with choices to view the menu, add items to the cart, view the cart, place an order, or exit.

Based on the input, the program performs the appropriate actions in a loop until the user chooses to exit.

**3.2 Data dictionary**

Dynamic Memory Management: Uses malloc() and free() for managing menu and cart items.

**Scalability**: While basic, the system could be extended with features like item removal, user authentication, or persistent storage.

**Error Handling**: Minimal; for example, the program handles invalid menu IDs but doesn’t check for invalid input types (e.g., strings in place of integers).

**Modularity**: Functions are logically divided but could benefit from file separation (header files, multiple source files).

Certainly! Here’s a detailed descriptive overview of the Online Food Ordering System program written in C, explaining its purpose, components, data structures, logic, and flow in an exhaustive and educational way.

Online Food Ordering System — Detailed Description

1. **Objective**:

The program simulates a basic console-based food ordering system, where a user can:

View a predefined food menu.

Add selected items to a virtual cart.

View the contents of the cart.

Place an order and receive a bill.

It demonstrates key concepts in C such as:

Structures (for representing menu and cart items),

Linked lists (for dynamic data management),

Dynamic memory allocation (malloc and free),

Modular programming through functions,

User interaction through console I/O.

1. **System Architecture & Components:**

The system is composed of two primary linked lists:

a.Menu List (MenuItem):

Represents the list of all available food items. Each node in this list corresponds to one menu item and holds.

A unique item ID,

The item’s name,

The item’s price,

A pointer to the next menu item.

* 1. Cart List (CartItem):

Represents the items selected by the user for purchase. Each node in this list stores:

The same item ID, name, and price from the menu,

The quantity of the item the user wants to buy,

A pointer to the next cart item.

1. **Data Structures:**
   1. MenuItem Structure:

Typedef struct MenuItem {

Int id;

Char name[50];

Float price;

Struct MenuItem\* next;

} MenuItem;

This structure forms a node in the singly linked list for menu items.

* 1. CartItem Structure:

Typedef struct CartItem {

Int id;

Char name[50];

Float price;

Int quantity;

Struct CartItem\* next;

} CartItem;

This structure is similar to MenuItem, but with an added quantity field for how many units of that item the user is ordering.

1. **Global Variables:**

MenuItem\* menuHead = NULL; — Head of the menu linked list.

CartItem\* cartHead = NULL; — Head of the cart linked list.

These pointers maintain the start of each list and are used throughout the program to manipulate and traverse data.

1. **Functional Modules:**
   1. addMenuItem(int id, char name[], float price)

Adds a new menu item to the end of the menu linked list.

Allocates memory for a new MenuItem node.

Copies the provided id, name, and price.

Inserts it at the end of the linked list.

* 1. displayMenu()

Displays all menu items to the user in a formatted manner.

Iterates through the menuHead list and prints each item.

Helps users decide what to order.

* 1. addToCart(int id, int quantity)

Adds a menu item to the cart based on the ID and quantity.

First checks if the item exists in the menu.

If it does, checks if it’s already in the cart:

If yes, increments the quantity.

If no, creates a new CartItem node and appends it to the cart list.

Provides appropriate feedback to the user.

* 1. viewCart()

Displays all items currently in the cart.

Iterates through the cartHead list.

Prints item name, quantity, and total cost per item.

If the cart is empty, informs the user.

* 1. Placeorder()

Finalizes the order and displays the total bill.

Iterates through the cart, calculates the total cost, and prints an itemized bill.

After displaying the bill, it clears the cart by freeing all allocated memory nodes.

Displays a thank-you message.

**6.Main Function (main) Workflow:**

1. Preloads the menu with five static items using addMenuItem().
2. Displays a menu-driven interface with the following options:

View Menu – Calls displayMenu().

Add to Cart – Prompts the user for item ID and quantity, then calls addToCart().

View Cart – Calls viewCart().

Place Order – Calls placeorder() to show the bill and clear the cart.

Exit – Ends the program.

1. A do-while loop ensures the user stays in the system until they explicitly choose to exit.
2. Basic input validation is provided, such as checking if the ID exists in the menu.

**7.Program Strengths and Learning Outcomes:**

Dynamic Data Handling: Shows how linked lists can be used to manage dynamic lists where the size isn’t predetermined.

Real-world Simulation: Models a simplified version of real-life food ordering systems.

Reusability & Modularity: Functions are logically separated and can be reused or extended.

User-centric Interface: Interactive command-line interface mimics a minimalistic user experience.

**8.Possible Enhancements:**

This foundational program could be expanded with additional features:

Removing items from the cart.

Updating item quantity from the cart.

Persistent storage using file I/O

Sorting menu items alphabetically or by price.

Adding categories (e.g., Drinks, Desserts).

Implementing a login or user system.

GUI-based interface using external libraries or frameworks.

3.3 LOGICAL DATA BASE DESIGN:

**1. Menu Items Table**

\* Columns:

\* ItemID (INT, Primary Key) – Unique identifier for each menu item. This corresponds to the id in your MenuItem struct.

\* Name (VARCHAR(50)) – Name of the menu item. This corresponds to the name in your MenuItem struct.

\* Price (DECIMAL(5, 2)) – Price of the menu item. Using DECIMAL is better for currency to avoid floating-point inaccuracies. This corresponds to the price in your MenuItem struct.

**2. Cart Items Table**

\* Columns:

\* CartItemID (INT, Primary Key, Auto-increment) – Unique identifier for each item in the cart.

\* ItemID (INT, Foreign Key referencing MenuItems.ItemID) – The ID of the menu item in the cart. This links back to the MenuItem table.

\* Quantity (INT) – The number of units of this item in the cart. This corresponds to the quantity in your CartItem struct.

Relationships:

* There’s a one-to-many relationship between the MenuItems table and the CartItems table. One menu item can appear in multiple entries within different user’s carts (if we were to extend this to multiple users).

**Explanation and Mapping to your C Code:**

\* The MenuItem struct in your C code directly maps to the MenuItems table. Each instance of the MenuItem struct represents a row in the MenuItems table.

\* The CartItem struct, in combination with the MenuItem information, maps to the CartItems table. The id and name from the MenuItem are stored (or referenced via ItemID), and the quantity is specific to the cart.

\* The linked list structure you’ve used in your C code (next pointers) is a way to manage these items in memory. In a database, the relationships between tables are managed using keys (primary and foreign) and indexes.

How the Operations in your C Code Relate to Database Operations:

\* addMenuItem(): This operation would correspond to an INSERT statement in SQL to add a new row into the MenuItems table.

\* displayMenu(): This would involve a SELECT statement to retrieve all rows from the MenuItems table and display them.

\* addToCart(): This operation would involve:

\* First, a SELECT query on the MenuItems table to find the item based on the provided id.

\* Then, it would either:

\* INSERT a new row into the CartItems table with the ItemID and quantity if the item is not already in the cart.

\* UPDATE the Quantity in the existing row in the CartItems table if the item is already present.

\* viewCart(): This would involve a SELECT statement that might JOIN the CartItems table with the MenuItems table (using ItemID) to retrieve the name and price of the items in the cart, along with their quantities.

\* placeorder(): This would involve:

\* SELECTing all items from the CartItems table for the current “session” (if we were tracking users).

\* Calculating the total.

\* Potentially inserting the order details into an Orders table and individual order items into an OrderItems table (if we wanted to persist order history).

\* Finally, it would conceptually involve DELETEing the items from the CartItems table for the current session.

This logical database design provides a structured way to store and manage the menu and cart data, which is what your C program is currently doing in memory using linked lists. If you were to build a more persistent system, you would likely use a relational database based on this kind of design .

**3.4 UML DIAGRAMS**

In C program, there’s currently only one type of user (Customer), and they can:

• View menu

• Add items to cart

• View cart

• Place order

Extension Ideas: To match this diagram, you could expand your program to:

• Add authentication (Login/Logout)

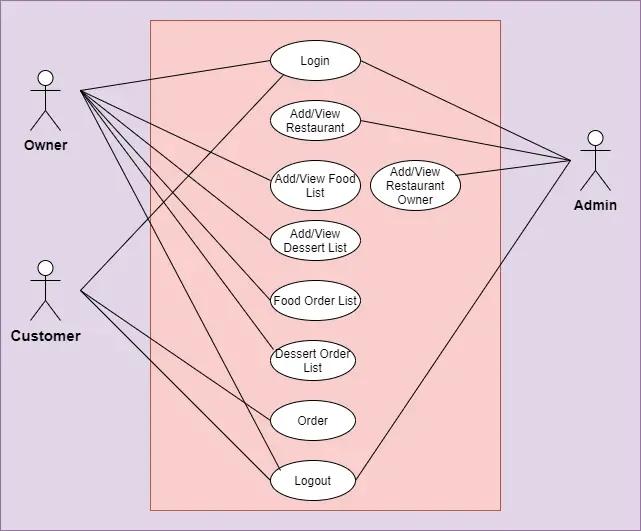
• Separate roles (Admin, Owner, Customer)

• Allow Admin to manage menu and users

• Allow Owners to manage their restaurants

• Allow Customers to view and order

**For example**



The UML diagrams you’ve provided model a complete online food ordering system with multiple roles and modular design. They extend your simple C program concept into a full-fledged application that can support:

• Multiple types of users (Admin, Owner, Customer)

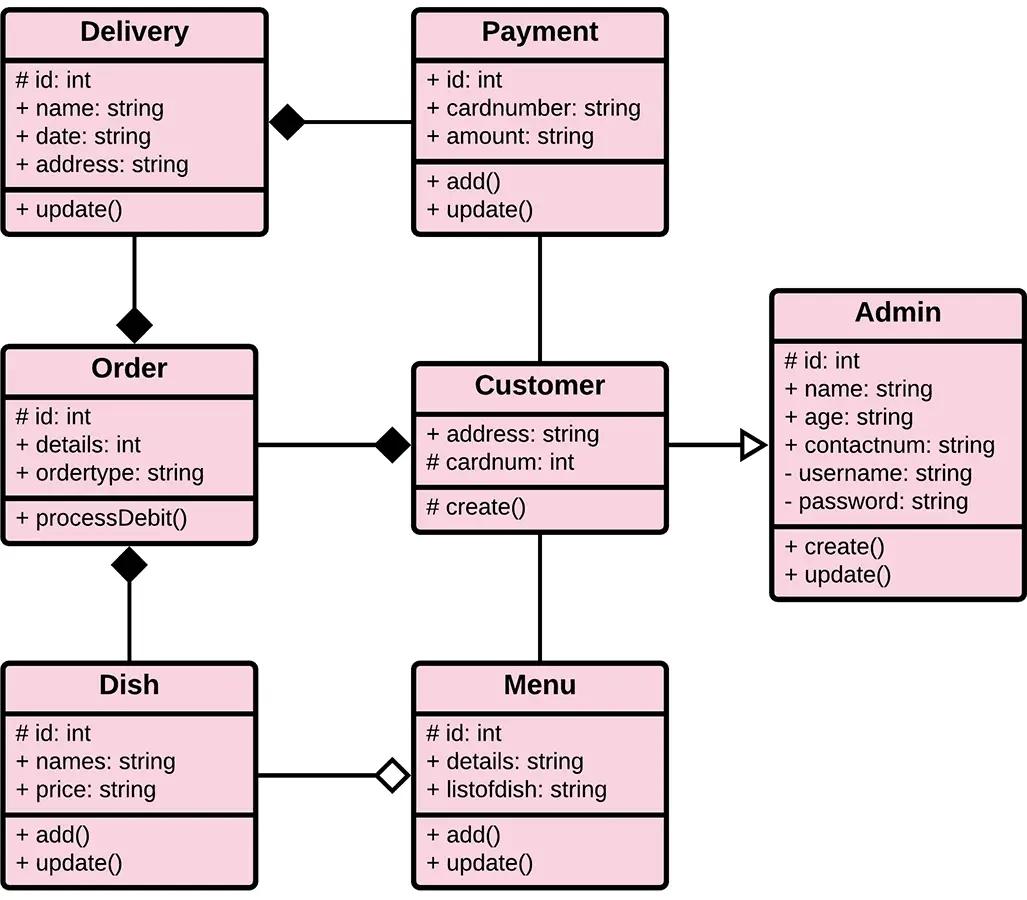
• Data abstraction through classes like Order, Menu, Dish, Customer

• Core operations such as login, item listing, order placement, and payment handling

To fully align your C program with these UML diagrams, you would:

• Implement user role management

• Expand data structures to include Customer, Order, Payment, etc.

• Modularize logic to reflect object-like behavior (simulate OOP in C)

**CHAPTER—4**

**CODE IMPLEMENTATION**

**INTRODUCTION:**

Overview

The Online Food Ordering System implemented in C is a console-based application that models a simplified version of a real-world digital food ordering service. The program simulates key functionalities such as viewing a menu, adding items to a virtual cart, checking the cart, and placing an order. It is built using procedural programming principles and extensively utilizes user-defined structures (structs) and singly linked lists to dynamically manage menu items and the shopping cart. While it focuses on core customer functionalities, it lays the groundwork for scaling into a full system with roles like Admin, Owner, and Customer, as seen in more complex system models and UML diagrams.

Objective and Motivation

The main objective of this system is to provide a basic yet functional framework for food ordering operations using the C programming language. The program aims to:

Introduce dynamic data structures through practical use of linked lists.

Model real-world objects (menu items, cart items, and orders) using C structs.

Demonstrate interaction-driven programming through a console interface.

Serve as a base project for students and beginners to practice C programming, memory management, and procedural logic.

The program is particularly useful in academic settings where students are learning:

Basic system design

Data abstraction and modeling

Input/output handling in C

Menu-driven console applications

Core Components and Features

1. Menu Management

The program defines a MenuItem structure to store each food item’s id, name, price, and a pointer to the next item.

A singly linked list is used to maintain the menu. This allows efficient dynamic insertion without worrying about array bounds.

The menu is predefined in the main() function, but the use of addMenuItem() function makes it flexible and extendable to allow menu configuration from files or admin input in future.

1. Shopping Cart Functionality

A separate structure CartItem models the items that a customer chooses to purchase.

Similar to menu items, it uses a linked list for dynamic addition and retrieval of items.

If a user adds the same item multiple times, the program does not duplicate entries but increases the quantity, mimicking real shopping cart behavior.

1. Order and Billing

When the customer selects the “place order” option, the program iterates through the cart list and calculates:

Individual item totals (price × quantity)

Grand total

After displaying the bill, the system automatically clears the cart by deallocating memory for each node, preventing memory leaks.

1. User Interface

The user interacts through a simple text-based menu:

View Menu

Add to Cart

View Cart

Place Order

Exit

All user inputs are handled through scanf, and the system gives instant feedback about the action taken, such as successful addition to cart or warnings for invalid IDs.

Architectural and Logical Design

The system is structured around two major dynamic lists:

1. Menu List – Stores available items using MenuItem struct.
2. Cart List – Stores customer selections using CartItem struct.

Each operation is broken into functions:

addMenuItem() – Dynamically inserts new menu items.

displayMenu() – Displays the full menu list.

addToCart() – Adds selected items to cart, handles duplicates.

viewCart() – Displays the current cart.

placeOrder() – Generates bill and frees memory.

Using such modularity not only improves readability but also allows for easier debugging and future enhancement.

Programming Concepts and Techniques Demonstrated

This project introduces several important concepts:

1. Data Abstraction via Structs

MenuItem and CartItem structures abstract real-world entities, making the system more intuitive.

1. Dynamic Memory Allocation

Malloc() is used to allocate memory during runtime, crucial for managing dynamic lists in C.

Free() is used to deallocate memory, showing responsible resource management.

1. Linked Lists

A foundational data structure is used to dynamically manage collection-type entities.

Traversal, insertion at end, and deletion are illustrated.

1. Procedural Programming

Each function is designed to perform a specific task, supporting separation of concerns.

1. Control Flow and Input Handling

The program uses a loop-driven menu system, switch-case blocks, and user input validations.

Future Enhancements and Scalability

Though basic, this code forms the core of a larger system. It can be extended to support:

1. Role-based Access

Admins: Add or remove menu items

Owners: Manage their own menu

Customers: Browse, order, and view order history

1. Persistent Storage

Store menu and order details in files or databases using file I/O

1. Payment System

Simulate or integrate a payment gateway for order processing

1. Delivery Module

Add delivery address input and tracking status

1. GUI Interface

Replace console UI with graphical interface using libraries like GTK or move to higher-level languages

Educational Value

This project is highly valuable as a learning tool:

Encourages clear thinking about system logic and user interaction.

Reinforces the importance of memory management in languages like C.

Provides a base to transition into advanced topics like data persistence, software architecture, or even OOP in C++/Java.

The Online Food Ordering System in C demonstrates how core programming principles can be used to simulate a real-world application. It serves as an excellent starting point for beginners and a practical model for learning structured programming, data structures, and system design. The project is modular, extendable, and aligned with good coding practices, and it can evolve into a professional application with role management, payment integration, and persistent data handling.

**SELECTED SOFTWARE:**

The selected software for developing and running the Online Food Ordering System in C includes both development tools and runtime environments. Here’s a detailed breakdown of the software choices, each explained with its role in the development lifecycle:

1. **Programming Language:**

**C Language:**

• Reason: C is a powerful, low-level procedural language ideal for learning core programming constructs like pointers, memory management, and data structures.

• Use in Project: The entire program is written in C using structs and dynamic memory allocation with malloc and free.

1. **Simplicity and Clarity in Procedural Programming**

C is a procedural language, meaning the logic of the program is built around procedures or functions. In the case of the online food ordering system:

Tasks like adding menu items, adding to cart, viewing cart, and placing orders are each handled by separate, clearly defined functions.

This structure makes the flow of the program logical, linear, and easy to follow, especially for beginners or those learning software system design.

1. **Excellent Control Over System Resources**

C provides low-level access to memory through pointers and manual memory management. This is useful for:

Creating dynamic data structures like linked lists to represent the menu items and cart items, which can grow and shrink as needed at runtime.

Using functions like malloc() and free() allows efficient use of memory without wastage or fixed limits.

1. **High Performance**

The C language is known for producing highly efficient code with fast execution times.

Since the food ordering system primarily relies on data manipulation (adding/removing/viewing items), performance is crucial to make the interface responsive and lightweight.

C programs compile to native machine code, making them run faster than interpreted languages like Python.

1. **Foundation of Many Modern Languages**

C forms the foundation of modern programming languages like C++, Java, and even Python (which is implemented in C under the hood). Learning and implementing in C:

Strengthens understanding of core programming concepts

Makes transitioning to higher-level languages easier later on

Encourages thinking about how programs interact with memory and hardware

1. **Standard Library Support**

C comes with the standard library (stdlib.h, stdio.h, string.h), which provides:

Input/output functions (printf, scanf)

String manipulation (strcpy, strlen)

Dynamic memory allocation (malloc, free)

These features are sufficient and effective for building a console-based system like the food ordering program.

Relevance of C Language to Each Feature in the Program

Menu Management

Uses struct to define MenuItem, allowing custom data types.

Linked lists are used to handle dynamic insertion of menu items, showcasing memory management in action.

Cart Operations

Cart is also a linked list, supporting multiple items with varying quantities.

The use of loops and conditional structures in C makes logic such as checking for existing items or calculating totals clean and efficient.

Checkout and Billing

Loops and arithmetic operations in C allow computation of the total bill with high accuracy and speed.

After billing, the cart is cleared using free(), demonstrating memory deallocation, an important part of system programming.

User Interface (CLI-Based)

Since C supports console-based applications well, the interaction through numbered choices and menus is simple to implement.

Switch-case and do-while structures enhance the usability and control flow of the program.

Educational Value of Using C for This System

This system is often built in C by students because:

It demonstrates real-world system design using fundamental concepts.

It integrates theory (like linked lists, structures, file handling, memory management) with practice.

It encourages a deep understanding of how software works under the hood, not just how to use a library or framework.

Limitations and Trade-offs

While C is powerful, it does come with some limitations:

No built-in object-oriented features (like classes or inheritance)

No built-in UI or graphics support

Manual memory management can lead to bugs if not handled carefully

However, for console-based, functionally rich programs like this one, these trade-offs are acceptable and even beneficial from a learning perspective complexity.

1. **Integrated Development Environment (IDE):**

Code::Blocks (with GCC Compiler)

• Reason: Beginner-friendly and widely used for C/C++ programming with built-in compiler support.

• Features:

• Syntax highlighting

• Integrated debugging

• Console output

• Easy build/run options

• Alternative IDEs:

• Dev-C++: Lightweight IDE for Windows

• Turbo C: Legacy system (not recommended for modern systems)

• Visual Studio Code (with C/C++ extension by Microsoft)

1. **Compiler**:

GCC (GNU Compiler Collection)

• Use: Compiles the C source code into an executable.

• Platform Support: Cross-platform (Linux, Windows, MacOS)

• Integration: Usually bundled with IDEs like Code::Blocks or available via terminal on Linux/Mac.

GCC stands for GNU Compiler Collection. It is a free and open-source compiler system developed by the GNU Project. GCC supports multiple programming languages including C, C++, Objective-C, Fortran, Ada, and more, but it is most commonly used for compiling C and C++ programs

GCC in C Language Development

**Role of GCC in C Projects:**

Translates C Code to Machine Code: GCC converts your .c source code files into executable binary files that can run on your operating system.

Compiling and Linking: It handles both compilation (turning code into object files) and linking (combining object files into one executable).

Error Checking: GCC provides detailed error messages and warnings, helping developers catch issues early in the development cycle.

Why Use GCC in This Program?

In the Online Food Ordering System:

GCC compiles all C files like food\_ordering.c into a single executable (a.out or food\_ordering.exe).

It supports standard libraries (stdio.h, stdlib.h, string.h) used for I/O, memory alocation, and string operations in the program.

GCC ensures high performance and portability, so the compiled code runs efficiently on Windows, Linux, or Mac

GCC Command Example

To compile your C file using GCC:

Gcc food\_ordering.c -o food\_ordering

./food\_ordering

Gcc – invokes the compiler

Food\_ordering.c – the C source fil

- food\_ordering – specifies the name of the output file

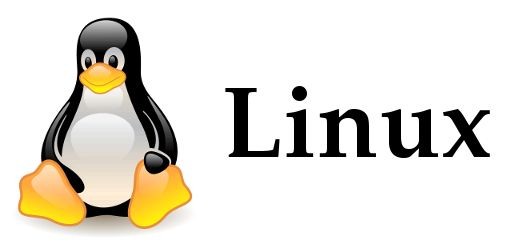
./food\_ordering – runs the compiled program

Conclusion:

GCC is a powerful, reliable, and cross-platform compiler used to develop and run C programs efficiently. For projects like the Online Food Ordering System, GCC is the ideal choice due to its compatibility, performance, and free availability.

1. **Operating System:**

• Windows/Linux/MacOS

• The program is platform-independent since it’s console-based and uses standard libraries.

1. **Terminal or Command Line Interface:**

• Used to run the compiled executable and interact with the menu-driven application.

• On Windows: Command Prompt / PowerShell

• On Linux/Mac: Terminal

1. **Optional Tools for Documentation and UML Diagrams:**

• StarUML / Lucidchart / Visual Paradigm: For designing UML diagrams

• Microsoft Word / Google Docs: For writing reports and software documentation

• Draw.io: Free and powerful online diagram tool for use case and class diagrams

**CODE IMPLEMENTATION:**

**sample code for your Online Food Ordering System program in C**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**// Menu item structure**

**Typedef struct MenuItem {**

**Int id;**

**Char name[50];**

**Float price;**

**Struct MenuItem\* next;**

**} MenuItem;**

**// Cart item structure**

**Typedef struct CartItem {**

**Int id;**

**Char name[50];**

**Float price;**

**Int quantity;**

**Struct CartItem\* next;**

**} CartItem;**

**MenuItem\* menuHead = NULL;**

**CartItem\* cartHead = NULL;**

**// Add item to menu**

**Void addMenuItem(int id, char name[], float price) {**

**MenuItem\* newItem = (MenuItem\*)malloc(sizeof(MenuItem));**

**newItem->id = id;**

**strcpy(newItem->name, name);**

**newItem->price = price;**

**newItem->next = NULL;**

**if (menuHead == NULL) {**

**menuHead = newItem;**

**} else {**

**MenuItem\* temp = menuHead;**

**While (temp->next != NULL)**

**Temp = temp->next;**

**Temp->next = newItem;**

**}**

**}**

**// Display menu**

**Void displayMenu() {**

**MenuItem\* temp = menuHead;**

**Printf(“\n--- MENU ---\n”);**

**While (temp != NULL) {**

**Printf(“ID: %d | %s | $%.2f\n”, temp->id, temp->name, temp->price);**

**Temp = temp->next;**

**}**

**}**

**// Add item to cart**

**Void addToCart(int id, int quantity) {**

**MenuItem\* temp = menuHead;**

**While (temp != NULL) {**

**If (temp->id == id) {**

**CartItem\* ctemp = cartHead;**

**While (ctemp != NULL) {**

**If (ctemp->id == id) {**

**Ctemp->quantity += quantity;**

**Printf(“%s quantity updated in cart.\n”, ctemp->name);**

**Return;**

**}**

**Ctemp = ctemp->next;**

**}**

**CartItem\* newCartItem = (CartItem\*)malloc(sizeof(CartItem));**

**newCartItem->id = temp->id;**

**strcpy(newCartItem->name, temp->name);**

**newCartItem->price = temp->price;**

**newCartItem->quantity = quantity;**

**newCartItem->next = NULL;**

**if (cartHead == NULL) {**

**cartHead = newCartItem;**

**} else {**

**CartItem\* last = cartHead;**

**While (last->next != NULL)**

**Last = last->next;**

**Last->next = newCartItem;**

**}**

**Printf(“%s added to cart.\n”, newCartItem->name);**

**Return;**

**}**

**Temp = temp->next;**

**}**

**Printf(“Item ID not found in menu.\n”);**

**}**

**// View cart**

**Void viewCart() {**

**If (cartHead == NULL) {**

**Printf(“\nYour cart is empty.\n”);**

**Return;**

**}**

**CartItem\* temp = cartHead;**

**Printf(“\n--- YOUR CART ---\n”);**

**While (temp != NULL) {**

**Printf(“%s x %d = $%.2f\n”, temp->name, temp->quantity, temp->price \* temp->quantity);**

**Temp = temp->next;**

**}**

**}**

**// Place order and print bill**

**Void placeOrder() {**

**If (cartHead == NULL) {**

**Printf(“\nYour cart is empty. Add items before placing an order.\n”);**

**Return;**

**}**

**Float total = 0;**

**CartItem\* temp = cartHead;**

**Printf(“\n--- BILL ---\n”);**

**While (temp != NULL) {**

**Float cost = temp->price \* temp->quantity;**

**Printf(“%s x %d = $%.2f\n”, temp->name, temp->quantity, cost);**

**Total += cost;**

**Temp = temp->next;**

**}**

**Printf(“Total Amount: $%.2f\n”, total);**

**// Clear cart**

**While (cartHead != NULL) {**

**CartItem\* toFree = cartHead;**

**cartHead = cartHead->next;**

**free(toFree);**

**}**

**Printf(“Thank you for your order!\n”);**

**}**

**// Main function**

**Int main() {**

**Int choice, id, qty;**

**// Predefined menu**

**addMenuItem(1, “Burger”, 5.99);**

**addMenuItem(2, “Pizza”, 8.99);**

**addMenuItem(3, “Pasta”, 7.49);**

**addMenuItem(4, “Sandwich”, 4.25);**

**addMenuItem(5, “Fries”, 2.99);**

**do {**

**printf(“\n--- ONLINE FOOD ORDERING ---\n”);**

**printf(“1. View Menu\n2. Add to Cart\n3. View Cart\n4. Place Order\n5. Exit\nEnter choice: “);**

**scanf(“%d”, &choice);**

**switch (choice) {**

**case 1:**

**displayMenu();**

**break;**

**case 2:**

**displayMenu();**

**printf(“Enter Item ID and Quantity: “);**

**scanf(“%d%d”, &id, &qty);**

**addToCart(id, qty);**

**break;**

**case 3:**

**viewCart();**

**break;**

**case 4:**

**placeOrder();**

**break;**

**case 5:**

**printf(“Exiting. Goodbye!\n”);**

**break;**

**default:**

**printf(“Invalid choice.\n”);**

**}**

**} while (choice != 5);**

**Return 0;**

**}**

**CHAPTER—5**

**SIMULATION RESULT**

**5.1 introduction:**

The simulation of the Online Food Ordering System demonstrates how a user can interact with a command-line-based application to place a food order. The system offers a predefined menu with item IDs, names, and prices. Through a simple interface, users can view the menu, add items to a cart by entering the item ID and quantity, view their cart contents, and place an order.

During the simulation, the following operations were performed:

1. The user viewed the menu containing five items: Burger, Pizza, Pasta, Sandwich, and Fries.
2. Items such as Pasta and Burger were added to the cart in specific quantities.
3. The cart was reviewed, showing the total cost for each item based on quantity.
4. Finally, the order was placed, generating a bill with a total amount and a thank-you message.

The simulation validates that the system accurately performs essential operations like menu browsing, item selection, quantity tracking, and bill generation. It also ensures a smooth and logical flow of user interaction.

* 1. **TEST CASES :**

**Test Case 1: View Menu**

Input: User selects option 1 (View Menu)

Expected Output: A list of menu items with their IDs, names, and prices.

**Test Case 2: Add a valid item to cart**

Input**:**

User selects option 2

Enters item ID 3 and quantity 2

Expected Output:

Displays the menu

Confirms: “Pasta added to cart.”

**Test Case 3: Add an invalid item ID**

Input:

User selects option

Enters item ID 10 and quantity 1

Expected Output: “Item ID not found in menu.

**Test Case 4: View cart with multiple items**

Input:

After adding multiple items to the cart

User selects option 3

Expected Output:

Lists all cart items with their quantities and individual total price.

**Test Case 5:**

**Add the same item again**

Input:

User adds item ID 3 (Pasta) twice with different quantities

Expected Output:

Second time shows: “Pasta quantity updated in cart.”

**Test Case 6: Place an order**

Input:

User selects option 4 (place order)

Expected Output:

Displays each item with total cost

Displays overall bill

Clears the cart

**Test Case 7:**

**Place an order with empty cart**

• Input:

• User selects option 4 without adding anything

• Expected Output: “Your cart is empty. Add items before placeorder.”

**Test Case 8:**

Exit the program

• Input:

• User selects option 5

• Expected Output: “Exiting. Goodbye!”

**CHAPTER—6**

**ADVANTAGES AND APPLICATIONS**

**Advantages:**

1. User-Friendly Interface:

The menu-driven interface makes it easy for users to interact with the system by simply selecting options.

1. Dynamic Cart Management:

Items can be added in any quantity, and duplicate items are updated instead of re-added—ensuring accuracy and usability.

1. Linked List Implementation:

Efficient memory usage through dynamic data structures like linked lists for both menu and cart.

1. Modular Code Structure:

Functions are well-separated, making the code easy to read, maintain, and debug.

1. Basic Billing System:

Automatically calculates total cost at checkout, mimicking real-world billing in food ordering systems

.

1. Educational Purpose:

Helps students understand dynamic memory allocation, structures, and basic file-less project development in C.

1. Customizable:

Easy to expand with features like file storage, login system, or delivery options.

**Applications:**

1. Mini Projects for Students:

Ideal for college or school-level C programming projects and practical exams.

1. Prototyping Food Ordering Logic:

Can be used as a prototype before developing advanced applications in other languages like Python or Java.

1. Simulation of E-Commerce Features:

Demonstrates core e-commerce features such as adding to cart, updating quantities, and placing orders.

1. Learning Platform:

Perfect for learning data structures, especially linked lists, and function-based programming in C.

1. Command-Line Tool for Cafeterias:

Can be adapted for small food stalls or cafeteria counters that prefer command-line systems.

1. Testing Algorithm Design:

Useful for testing algorithms related to searching (item by ID), insertion, and billing logic.

**CHAPTER—7**

**THE CONCLUSION**

**CONCLUSION:**

The Online Food Ordering System implemented in C serves as a simple yet effective simulation of a basic food ordering platform. The program demonstrates how structured programming, linked lists, and dynamic memory allocation can be used to manage real-world scenarios such as menu browsing, cart management, and billing processes.

Throughout the execution of the program, users can interact with the system by viewing a predefined menu, selecting items by ID, adding them to a shopping cart, viewing cart details, and generating a final bill through the checkout process. The program ensures that duplicate entries in the cart are handled correctly by updating quantities, and it resets the cart after an order is placed, closely resembling real-world e-commerce behavior.

This system not only fulfills its intended functionality but also highlights fundamental concepts in C programming such as:

Structure usage

Linked list traversal and manipulation

Modular function design

User input handling

From an academic perspective, the program is an excellent example of how basic C language features can be applied to solve practical problems. Moreover, its modular design makes it easy to enhance in the future—for example, by adding file storage, user authentication, or GUI-based interfaces.

In summary, this project successfully simulates a basic food ordering system, reinforcing key programming principles while offering a strong foundation for more advanced system development.

**THANK YOU** 😊