Welcome to Binary Brains

"Where Knowledge Knows No Limits"

Our Team Members

Sk. Azmain Zunaeid
(Leader)
ID: 251-15-222

Md. Delower Sarker
ID: 251-15-457

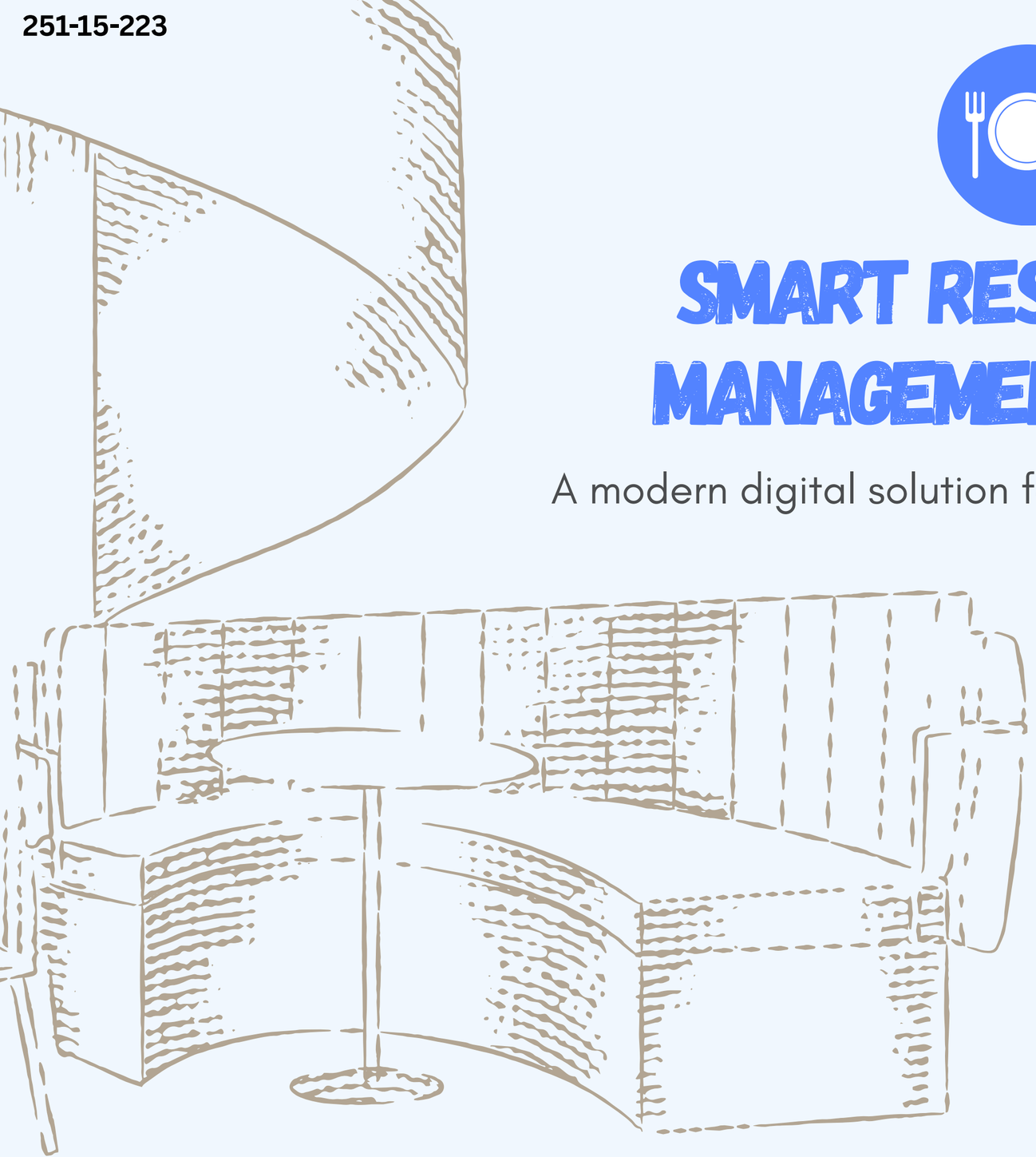
Md. Maruf Mia
ID: 251-15-042

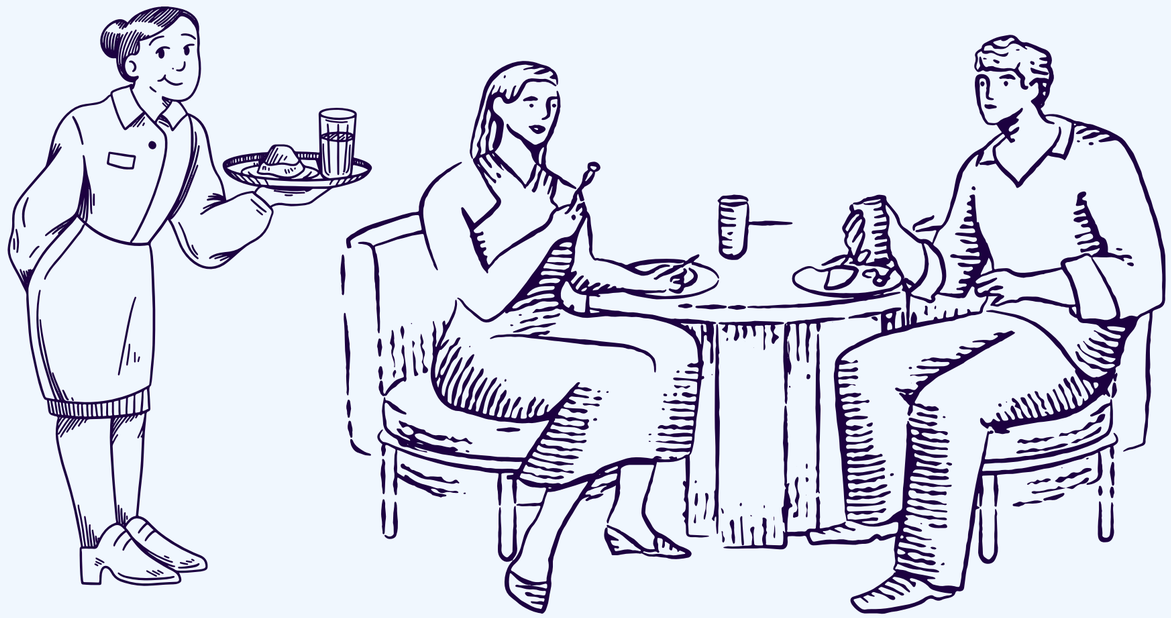
Syeda Zawada Farah
ID: 251-15-223

Maher Afruz Mira
ID: 251-15-017

SMART RESTAURANT MANAGEMENT SYSTEM

A modern digital solution for restaurant automation.





Course Code: CSE124

Data Structure Lab

Department of Computer Science and Engineering

Course Instructor

Mr. Md. Jakaria Zobair

Lecturer

Department of CSE

Daffodil International University

Project Management and Team Work

- **Project Leader:** **Zunaeid -** coordinated project, delegated tasks, managed timeline, verified design, and reviewed final work.
- **Research & Requirements:** **Farah & Mira -** conducted feasibility study, gap analysis, and gathered references.
- **System Implementation:** **Zunaeid & Delower -** coded core functions, integrated modules, debugged, and tested functionality.
- **Testing & QA:** **Maruf -** designed test cases, validated system, ensured robust error handling.
- **Documentation:** **Zunaeid -** compiled project report, wrote methodology, structured content.
- **Presentation:** **Delower -** created slides, simplified complex details, ensured clarity and visual appeal.

Presentation Outline

- Introduction
- Motivation
- Objectives
- Feasibility Study
- Requirement Analysis
- Implementation
- Results and Discussion
- Advantages and Disadvantages
- Conclusion
- QnA

Introduction

- Console-based restaurant management system in C
- Two operational modes: Manager & Customer
- Demonstrates data structure applications
- Real-world problem solving approach

Motivation

- **Pedagogical Motivation**
  Bridges the gap between theoretical and practical knowledge regarding DS

- **Practical Motivation**
  Solving real world problems using fundamental programming concepts

Objectives

- Implement persistent menu management using linked lists with file-based storage
- Develop a queue-based order system following FIFO principles
- Create separate user interfaces for Manager and Customer roles
- Demonstrate data structure concepts through practical, real-world application

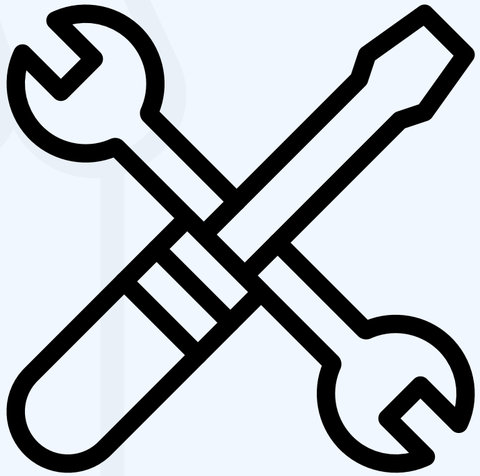
Feasibility Study

- Previous Works
- Technical Feasibility
- Economic Feasibility
- Resource Feasibility
- Operational Feasibility

Requirement Analysis

Hardware Requirements

- **Processor:** 1 GHz or higher
- **Memory:** 1 GB RAM minimum
- **Storage:** 10 MB available space
- **Input:** Standard keyboard
- **Output:** Console/terminal display



Requirement Analysis

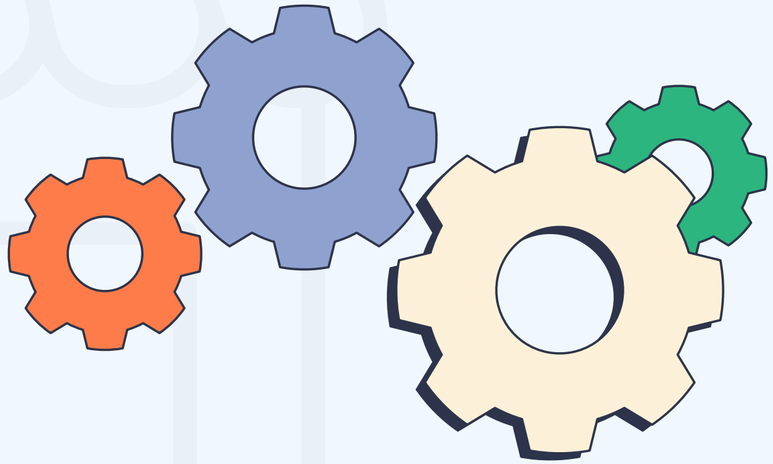
Software Requirements

- **Operating System:** Windows/Linux/macOS
- **Compiler:** GCC, Clang, or any ANSI C compliant compiler
- **Dependencies:** Standard C libraries only

Requirement Analysis

Functional

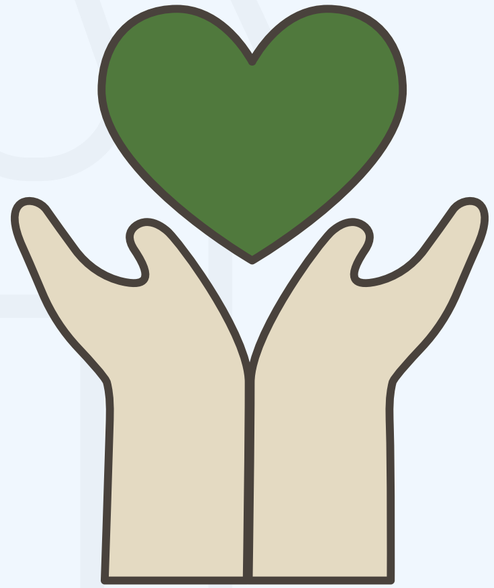
- Authentication System
- Menu Management
- Order Processing
- Persistence
- Order Queue
- Input Validation



Requirement Analysis

Non-Functional

- Performance
- Reliability
- Usability
- Maintainability
- Memory Safety



Requirement Analysis

Stakeholder’s Requirements

- The importance of quick menu updates during peak hours
- Need for clear table-wise order tracking
- Priority of FIFO order serving during busy periods
- Requirement for simple, error-resistant interfaces for staff

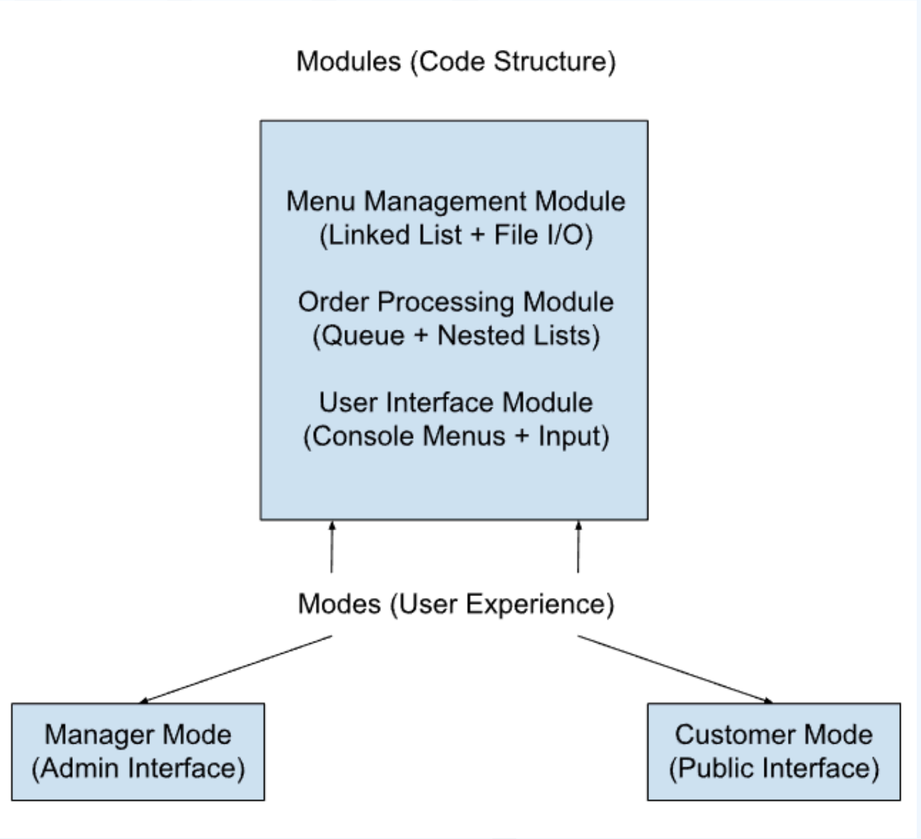
Implementation

System Architecture

The system architecture comprises three core modules using C:

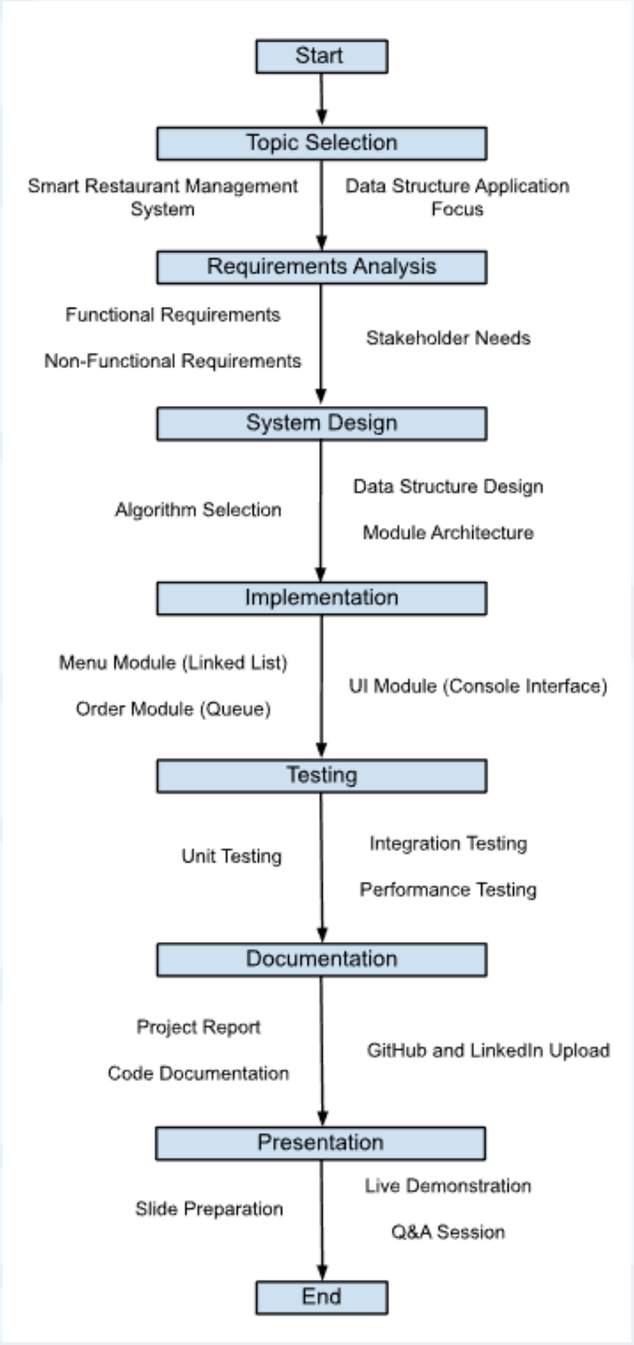
- **Menu Management Module**
- **Order Processing Module**
- **User Interface Module**

Tools: C Language, Linked Lists, Queues.



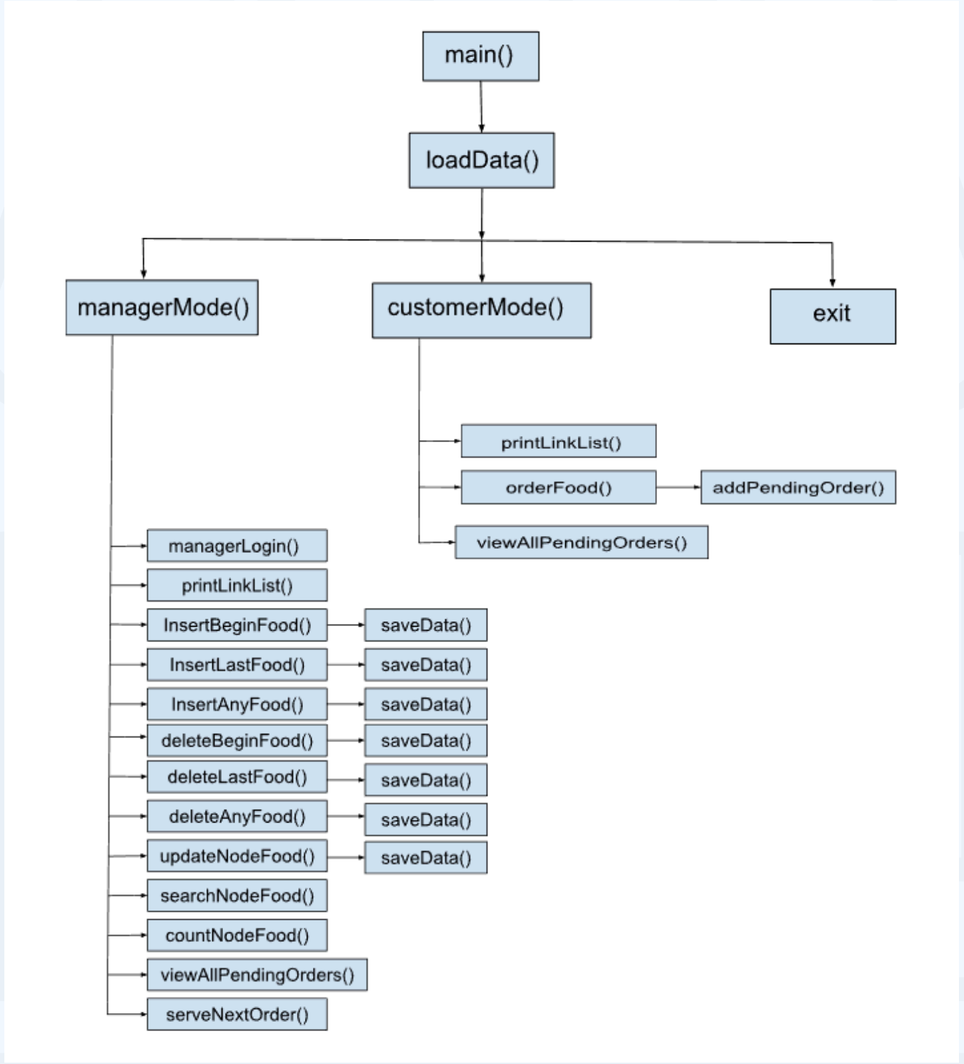
Implementation

Work Flow



Implementation

Function Flow



InsertAnyFood() Function

Main Steps:

- Check if list is empty. If empty → stop.
- Ask for Target ID and search for it. If not found → stop.
- Create a new node for the food.
- Validate inputs:
  - ID must be unique.
  - Name must be unique.
- Insert the node:
  - Set newnode → next to target's next.
  - Set target → next to newnode.
- Save the updated list using saveData().

Code (Page 21 - 23)

```
282  void InsertAnyFood() // For inserting in any position of the menu; For inserting after any item of the menu
283  {
284      if(start == NULL)
285      {
286          printf("No items available in the menu.\n");
287          return;
288      }
289
290      char x[20];
291      printf("After which food ID you want to insert: ");
292      scanf("%19s", x);
293
294      struct food *i = start;
295      while(i != NULL && strcmp(i->id, x) != 0) i = i->next;
296      if(i == NULL)
297      {
298          printf("Given ID not found.\n");
299          return;
300      }
301
302      // UPDATED CODE: ID and Name validation loop
303      struct food *newnode = (struct food*)malloc(sizeof(struct food));
304      struct food *temp;
305      int exists;
306
307      // STEP 1: ID Validation Loop
308      while(1)
309      {
310          printf("Enter Food ID: ");
311          scanf("%19s", newnode->id);
312
313          temp = start;
314          exists = 0;
315
```

```
316         // Check if ID exists
317         while(temp != NULL)
318         {
319             if(strcmp(temp->id, newnode->id) == 0)
320             {
321                 exists = 1;
322                 break;
323             }
324             temp = temp->next;
325         }
326 
327         if(exists) printf("Error: This ID already exists! Please enter a unique ID.\n");
328 
329         else  break;
330     }
331 
332     // STEP 2: Name Validation Loop
333     while(1)
334     {
335         printf("Enter Food Name: ");
336         scanf(" %[^\n]", newnode->name);
337 
338         temp = start;
339         exists = 0;
340 
341         // Check if Name exists
342         while(temp != NULL)
343         {
344             if(strcmp(temp->name, newnode->name) == 0)
345             {
346                 exists = 1;
347                 break;
348             }
349             temp = temp->next;
350         }
351 
```

```
352         if(exists)   printf("Error: This Name already exists! Please enter a unique Name.\n");
353
354         else break;
355
356     }
357     //printf("Enter Food ID: ");
358     //scanf("%19s", newnode->id);
359     // printf("Enter Food Name: ");
360     // scanf(" %[^\n]", newnode->name);
361     printf("Enter Food Price: ");
362     scanf("%f", &newnode->price);
363
364     printf("Enter Stock Quantity: "); // NEW ADDED CODE
365     scanf("%d", &newnode->stockCount); // NEW ADDED CODE
366
367     newnode->next = i->next;
368     i->next = newnode;
369
370     printf("Item added after ID %s.\n", x);
371     saveData();
372 }
373
```

deleteAnyFood() Function

Main Steps:

- Check Empty List: If start == NULL, stop.
- **Delete First Node:**
  - If the first node's ID matches → move start to start → next and free the old node.
- **Delete Middle/End Node:**
  - Traverse the list using a pointer.
  - Use peek-ahead (i → next) to find the node before the target.
  - Re-link: i → next = i → next → next
  - Free the removed node.

Memory Handling:

- Frees the deleted node to avoid memory leaks.

Saving Changes:

- Calls saveData() after successful deletion.

Code

```
421  void deleteAnyFood() // For deleting from any position of the menu; For deleting by ID number
422  {
423      if(start == NULL)
424      {
425          printf("No items available in the menu.\n");
426          return;
427      }
428
429      char y[20];
430      printf("Which food ID you want to delete: ");
431      scanf("%19s", y);
432
433      if(strcmp(start->id, y) == 0)
434      {
435          struct food *temp = start;
436          start = start->next;
437          free(temp);
438
439          printf("Item deleted having ID %s.\n", y);
440          saveData();
441          return;
442      }
443
444      struct food *i = start;
445      while(i->next != NULL && strcmp(i->next->id, y) != 0) i = i->next;
446      if(i->next == NULL)
447      {
448          printf("Given ID not found.\n");
449          return;
450      }
451
452      struct food *temp = i->next;
453      i->next = i->next->next;
454      free(temp);
455      printf("Item deleted having ID %s.\n", y);
456      saveData();
457  }
458
```

serveNextOrder() Function

How It Works:

- Uses two lists at the same time:
  - Order List: Contains ordered food IDs.
  - Menu List: Searched to find each item's Name and Price for the bill.
- Each ordered item is matched with the menu to display full details.

Queue Handling:

- Takes the front node of pendingOrders.
- Moves the head pointer forward:
- pendingOrders = pendingOrders ➔ next
- This removes the served order from the queue.

Memory Cleanup:

- Frees all items inside the order (inner list).
- Frees the main order node at the end.
- Prevents memory leaks.

Code

```c
/* ----------------- Serving Orders - Dequeue (1) - Manager ----------------- */

void serveNextOrder() // For serving pending orders
{
    if(pendingOrders == NULL)
    {
        printf("No pending orders available.\n");
        return;
    }

    struct orderNode *serve = pendingOrders;
    printf("\nServing Table %d: \n", serve->tableNo);
    struct orderItem *it = serve->items;

    while(it != NULL)
    {
        struct food *f = start;
        while(f != NULL && strcmp(f->id, it->foodID) != 0) f = f->next;
        if(f != NULL) printf("  %s x %d (%.2f Tk)\n", f->name, it->qty, f->price * it->qty);
        else printf("  Unknown Food ID %s x %d\n", it->foodID, it->qty);
        it = it->next;
    }

    pendingOrders = pendingOrders->next;

    it = serve->items;
    while(it != NULL)
    {
        struct orderItem *tmp = it;
        it = it->next;
        free(tmp);
    }
    free(serve);
    printf("Order served.\n");
}


```

printLinkList() Function

- **Prints the entire restaurant menu stored in a linked list.**
- Starts from the head pointer start.
- If the list is empty → shows “No items available” and exits.
- Otherwise prints a formatted menu header.
- **Traverses each node and displays:**
  - Food ID
  - Food Name
  - Price
  - Stock Status (quantity or [OUT OF STOCK])
- Continues until the end of the list (i == NULL).
- Prints a closing line after the menu.

Code

```c
/* ################### SINGLY LINKED LIST OPERATIONS (10) ################### */

/* ----------------- Menu Print (1) - Both ----------------- */

void printLinkList() // For printing the menu
{
    printf("\n--------------------- Smart Restaurant Menu ---------------------\n");
    struct food *i = start;
    if(i == NULL)
    {
        printf("\nNo items available.\n");
        printf("\n-----------------------------------------------------------------\n");
        return;
    }
    while(i != NULL)
    {
        // UPDATED CODE: Added Logic to Show Stock or Stock Out
        if(i->stockCount > 0)
        {
            printf("\nID: %-10s | %-20s | Price: %.2f Tk | Stock: %d\n", i->id, i->name, i->price, i->stockCount);
        }
        else
        {
            printf("\nID: %-10s | %-20s | Price: %.2f Tk | *** STOCK OUT ***\n", i->id, i->name, i->price);
        }
        i = i->next;
    }
    printf("\n-----------------------------------------------------------------\n");
}

```

orderFood() Function

- **Handles food ordering for a table.**
- Repeats while the user enters **'y'**.
- Takes Food ID and quantity as input and checks it's positive.
- **For each item:**
  - If found and stock available ➜ add to total, confirm item, and call **addPendingOrder()**.
  - If out of stock or insufficient ➜ show message.
  - If not found ➜ show "Food ID not found".
- At the end, prints total payable and order confirmation.

Code (Page 31 - 33)

```
738  /* ################### MODES (4) ################### */
739
740  /* ----------------- Customer Mode (2) - Customer ----------------- */
741
742  void orderFood(int table) // Placing orders by providing the table number, food id and quantity
743  {
744      char id[25];
745      int qty;
746      char more = 'y';
747      float total = 0.0f;
748
749      while(more == 'y' || more == 'Y')
750      {
751
752          printf("Enter the food ID to order: ");
753          scanf("%24s", id);
754          printf("Enter the quantity: ");
755
756          if(scanf("%d", &qty) != 1)
757          {
758              while(getchar()!='\n');
759              printf("Invalid quantity!\n");
760              continue;
761          }
762
763          if(qty <= 0)
764          {
765              printf("Quantity must be a positive number.\n");
766              continue;
767          }
768
769          struct food *i = start;
770          int found = 0;
771
```

```
772         while(i != NULL)
773         {
774             if(strcmp(i->id, id) == 0)
775             {
776                 // NEW ADDED CODE: Checking Stock Logic Started
777                 if(i->stockCount >= qty)
778                 {
779                     i->stockCount = i->stockCount - qty; // Decreasing stock
780                     saveData(); // Saving Updated Stock to file
781 
782                     total += i->price * qty;
783                     printf("Added %d x %s (%.2f Tk)\n", qty, i->name, i->price * qty);
784                     addPendingOrder(table, id, qty);
785                 }
786                 else if(i->stockCount == 0)
787                 {
788                     printf("Sorry! This item is OUT OF STOCK.\n");
789                 }
790                 else
791                 {
792                     printf("Insufficient Stock! Only %d items available.\n", i->stockCount);
793                 }
794                 // NEW ADDED CODE: Checking Stock Logic Ended
795 
796                 found = 1;
797                 break;
798             }
799             i = i->next;
800         }
801         if(!found) printf("Given ID not found.\n");
802 
803         printf("Order more? (y/n): ");
804         scanf(" %c", &more);
805     }
```

```
806
807         if(total > 0) // UPDATED CODE: Added this check
808         {
809             printf("\nTotal payable: %.2f Tk.\n", total);
810             printf("Your order will be served shortly.\n");
811         }
812     }
813
```

saveData() Function

```
38  /* ---------------- File Operations (2) - Both ---------------- */
39
40  void saveData() // For saving all the operational data permanently
41  {
42      FILE *fp = fopen("menu.txt", "w");
43      if(fp == NULL)
44          return;
45
46      struct food *i = start;
47      while(i != NULL)
48      {
49          // UPDATED CODE: Added stockCount to save file
50          fprintf(fp, "%s;%s;%.2f;%d\n", i->id, i->name, i->price, i->stockCount);
51          i = i->next;
52      }
53      fclose(fp);
54  }
55
```

addPendingOrder() Function

```
601  /* ################### QUEUE AND NESTED NODE OPERATIONS (3) ################### */
602
603  /* ----------------- Confirming Placed Orders - Enqueue and Nested Node Creation (1) - Customer ----------------- */
604
605  void addPendingOrder(int table, char *foodID, int qty) // Confirming placed orders by providing the table number, food id and quantity
606  {
607      struct orderNode *ptr = pendingOrders;
608      while(ptr != NULL)
609      {
610          if(ptr->tableNo == table) break;
611          ptr = ptr->next;
612      }
613
614      struct orderItem *newIt = (struct orderItem*)malloc(sizeof(struct orderItem));
615      strcpy(newIt->foodID, foodID);
616      newIt->qty = qty;
617      newIt->next = NULL;
618
619      if(ptr != NULL)
620      {
621          if(ptr->items == NULL)
622          {
623              ptr->items = newIt;
624          }
625
626          else
627          {
628              struct orderItem *it = ptr->items;
629              while(it->next != NULL) it = it->next;
630              it->next = newIt;
631          }
632      }
633
```

```
634         else
635         {
636             struct orderNode *newOrd = (struct orderNode*)malloc(sizeof(struct orderNode));
637             newOrd->tableNo = table;
638             newOrd->items = newIt;
639             newOrd->next = NULL;
640 
641             if(pendingOrders == NULL) pendingOrders = newOrd;
642 
643             else
644             {
645                 struct orderNode *t = pendingOrders;
646                 while(t->next != NULL) t = t->next;
647                 t->next = newOrd;
648             }
649         }
650     }
651 
```

Results and Discussion

Results

- **Menu Management Module Testing:** CRUD works correctly with validation, memory safety, and data persistence.
- **Order Processing Module Testing:** Table orders work correctly; queue maintains FIFO; memory freed on serving; bills accurate.
- **User Interface Module Testing:** Manager authentication works; table selection validated; menu intuitive; errors informative.

Results and Discussion

Discussion

- Data Structure Implementation Success
- Memory Management Validation
- Error Handling & Robustness
- Real-World Applicability

Advantages and Disadvantages

Advantages

- Low Resource Usage
- Persistent Storage
- Educational Value
- Efficient Data Handling
- FIFO Order Management
- Modular Code

Disadvantages

- No GUI
- Limited Security
- Linear Search
- No Edit/Cancel
- Not Fully Scalable

Conclusion

Summary

The Smart Restaurant Management System successfully delivers a functional restaurant prototype and educational tool by implementing linked lists for menus and queues for orders in C. Key outcomes include persistent CRUD operations, FIFO order processing, robust error handling, proper memory management, and modular code—effectively bridging data structure theory with practical application.

Future Scope

- Implementation of a full GUI
- Hash tables for faster lookups
- Secure, dynamic authentication
- More dynamic order control features

Thank You

Any Question?