CIS-17C Project 2 Writeup - Nathan James Pamintuan

Checkoff Sheet

1. Recursions - lines 91-97 - used to calculate recipe prep time
2. Recursive Sort - lines 1667 - quicksorting orders
3. Hashing - lines 332-477 - hashing guests
4. Trees - lines 82-218 - used to find recipe dependencies
5. Graphs - 635-660 - initialization of food delivery map

Introduction

For this project 2, I extended my current restaurant simulator with features that had the new topics, like hashing and trees in mind. Some background features only changed and/or rewritten, leaving no effect on the gameplay. I spent about two weeks working on this, between other commitments. This game has around 2000 lines of code.

Approach to development

Augmenting my project 1 to utilize the newer topics in class, like recursion and trees, took some time to plan out. Eventually I got some ideas. Since recipes in food usually have intermediary steps, I used recursion to keep track of recipes and ingredients. Handling guests was a little clunky, so hashing was implemented to help support the guest logic. Recipes now have ingredients that have some dependencies, and a tree is now used to keep track of them. Food delivery is now a part of the simulator, with graphs being used to model delivering across a region.

Game Rules

The game itself is quite rudimentary, since usage of the STL library was a much higher priority than making an actually fun restaurant game. The menu adds no real gameplay value, but it does demonstrate some STL concepts like forward iterators and sorting. The kitchen is similar, in that it mostly exists to show how the STL library is used. The restaurant operates with a bank balance; money can be earned by serving dishes from the kitchen. There is only a certain amount of time each day to seat and serve customers, and whenever the day is over, operating costs are charged to the restaurant.

====

The game hasn’t changed much from the original final version in the new project 2.There are now more ways to serve customers via order management and delivery.

Description of Code

The game code is organized into several different classes, with each class managing a different aspect of the restaurant simulation. The core of the game resides within the ResSys class, which manages critical data like time and restaurant state. This critical data is shared between other functions through friend class designation. The SubMenu class is an abstract base class which all submenus are built off of via polymorphism. The MenMgr class handles all the features pertaining to the "Manage Menu" menu. The GuestMgr class handles all the features related to guest management, like seating and waitlists. This code is organized such that its running is entirely dependent on these classes. Only two lines of code are inside main().

ResSys Class

Main Purpose: Main restaurant class that manages the entire simulation

Method	**Description**
ResSys()	Constructor that initializes the restaurant
initRest()	Initializes/resets restaurant state
detInsp()	Determines which customers are inspectors
addWait()	Adds a party to the waitlist
popTbl(int, bool)	Populates tables with customers
depopTbl(bool)	Removes customers from tables
chgDaily()	Charges daily expenses
endDay()	Processes end of day activities
earnFromOrder(const string&)	Processes earnings from food orders
improveRating()	Increases restaurant rating
isGameEnded()	Checks if game is over
advTime(bool)	Advances game time

getFormattedTime()	Returns formatted time string
runMain()	Main game loop

SubMenu Class

Main Purpose: Abstract base class for all menu interfaces

Method	**Description**
SubMenu(ResSys&)	Constructor taking reference to restaurant system
virtual ~SubMenu()	Virtual destructor
virtual void run()	Main menu loop for submenus
virtual string getTitle() = 0	Pure virtual method to get menu title
virtual void dispCont() = 0	Pure virtual method to display content
virtual bool handleCh(int)	Virtual method to handle choices
virtual void dispHead(const string&)	Display header
virtual void dispFoot()	Display footer

MenMgr Class

Main Purpose: Menu management submenu

Method	**Description**
MenMgr(ResSys&)	Constructor
string getTitle() override	Returns "Menu Management"
void dispCont() override	Displays menu items and options
bool handleCh(int) override	Handles menu sorting, shuffling, and statistics

GuestMgr Class

Main Purpose: Guest and seating management submenu

Method	**Description**
GuestMgr(ResSys&)	Constructor
string getTitle() override	Returns "Guest & Seating Management"
void dispCont() override	Displays table status and waitlist
bool handleCh(int) override	Handles seating guests and managing tables

KitchOps Class

Main Purpose: Kitchen operations submenu

Method	**Description**
KitchOps(ResSys&)	Constructor
string getTitle() override	Returns "Kitchen Operations"
void dispCont() override	Displays pantry stock and order history
bool handleCh(int) override	Handles serving orders and managing ingredients

Utility Functions (Non-Class)

Function	**Purpose**
clrScrn()	Clears the console screen (platform-dependent)
alert(pair<string, bool>&)	Displays alert message if flag is true
alert(string)	Sets system message for display
main()	Program entry point, creates and runs the restaurant system

Sample Input/Output

```
[== Bistro Ahri Restaurant Management System ==]
Day: 1 | Time: 1:18 / 8:00
Bank Balance: $1000.00 | Rating: 3.3 stars
Tables: 21 occupied / 21 total
Waitlist Size: 0
Orders Served (for undo): 0
=========[ Main Options ]===========
1) Manage Menu
2) Guest Management
3) Kitchen Operations
4) End Current Day
-----------------------------------
0) Exit System
===================================
> 0
```

```
--- [ Guest & Seating Management ] ---

Seating Chart (Tables: 10/16):
--------------------------
Table  1: Colin, Carlos, Barbara, Tom
Table  2: (Empty)
Table  3: Gustavo, Tyrone, Li
Table  4: Beckett, Jordan, Raymond
Table  5: Whitney, Manuel, John
Table  6: Rosario
Table  7: (Empty)
Table  8: Kofi, Walter
Table  9: Zoe, Mariana, Jade, Clint
Table 10: (Empty)
Table 11: (Empty)
Table 12: (Empty)
Table 13: Yael, Erik, Christine, Aaron
Table 14: (Empty)
Table 15: Keith, Chiara, Mia, Kennedy
Table 16: Ted
--------------------------
Waitlist (0 parties): (Empty)
--------------------------
1) Seat Next Party (if space)
2) Add Party to Waitlist (Manual)
3) Wait for tables to free
4) Seat from Waitlist (if space)

--------------------------
0) Return to Main Menu
!! Table 10 has been freed. !!
>
```

Pseudocode

```
// Clear screen utility
clrScrn:
        if Windows
        run "cls"
        else
        run "clear"

// Alert system
alert(msg, isActive):
        if isActive
        print "!! " + msg + " !!"
        reset msg and isActive

alert(msg):
        store msg in system message
        mark system message as active

// Base class for submenu screens
SubMenu:
        has a link to the restaurant system

        run:
        keep looping until user exits:
        show screen header with title
        show screen content
        print newline
        show footer
        if system message active
                display it
        print ">" and get user input

        if input invalid:
                clear buffer
                alert "Invalid input. Please enter a number."
                continue

        if user chose 0:
                break
        else:
```

```
            if choice not handled:
            alert "Invalid menu option."

        getTitle:
        to be defined by subclass

        dispCont:
        to be defined by subclass

        handleCh(choice):
        return false by default

        dispHead(title):
        clear screen
        print formatted title

        dispFoot:
        print footer line
        print option 0 to go back

// Menu Manager screen
MenMgr:
        inherits SubMenu

        getTitle:
        return "Menu Management"

        dispCont:
        print "Current Menu:"
        if menu is empty:
        print "(Menu is empty)"
        else:
        for each item in menu:
            print index + ") " + name + " $" + price
        print options:
        1 - sort by price
        2 - sort by name
        3 - shuffle
        4 - show cheapest and priciest

        handleCh(choice):
        if game is over and choice isn't 0:
        alert "Game Over! Cannot manage menu."
        return true
```

```
        if choice is 1:
        sort menu by price
        alert "Sorted by price."
        else if choice is 2:
        sort menu by name
        alert "Sorted by name."
        else if choice is 3:
        shuffle the menu
        alert "Menu shuffled."
        else if choice is 4:
        if menu is empty:
                alert "Menu is empty."
        else:
                find cheapest and most expensive items
                alert with names and prices
        else:
        return false
        return true

// Guest Manager screen
GuestMgr:
        inherits SubMenu

        getTitle:
        return "Guest & Seating Management"

        dispCont:
        show seating chart and waitlist status
        for each table:
        if empty:
                print "(Empty)"
        else:
                print guest names

        print options:
        1 - seat next party
        2 - add party manually
        3 - simulate guests leaving
        4 - seat from waitlist

        handleCh(choice):
        if game is over and choice != 0:
        alert "Game Over! Cannot manage guests."
```

```
        return true

        if choice is 1:
        if space available:
                alert "Seating 1 party..."
                populate table
                advance time
        else:
                alert "Restaurant full."
        else if choice is 2:
        add to waitlist
        else if choice is 3:
        alert "Waiting for guests to leave..."
        simulate depopulation
        advance time
        else if choice is 4:
        if waitlist empty:
                alert "Waitlist is empty."
        else if no space:
                alert "No space to seat from waitlist."
        else:
                remove party from waitlist
                alert "Seating party..."
                populate table
        else:
        return false
        return true

// Kitchen Operations screen
KitchOps:
        inherits SubMenu

        getTitle:
        return "Kitchen Operations"

        dispCont:
        print pantry stock
        if pantry is empty:
        print "(Pantry is empty)"
        else:
        list ingredients and quantities

        print last order in history (if any)
        print options:
```

1 - serve random order
2 - undo last order
3 - check ingredient stock
4 - add ingredient

handleCh(choice):
if game is over and choice not 0 or 2:
alert "Game Over! Cannot operate kitchen."
return true

if choice is 1:
if no customers:
    alert "No customers!"
else if menu is empty:
    alert "No items to serve!"
else:
    pick random item
    check if ingredients exist
    if yes:
    use ingredients
    log order
    earn money
    remove a table
    else:
    alert "Not enough ingredients!"
else if choice is 2:
if no order history:
    alert "Nothing to undo."
else:
    remove last order
    alert "Undid last action."
else if choice is 3:
ask user for ingredient name
show stock or "not found"
else if choice is 4:
ask user for name and quantity
if input invalid:
    alert "Invalid quantity."
else:
    update pantry
    alert with new total
else:
return false
return true

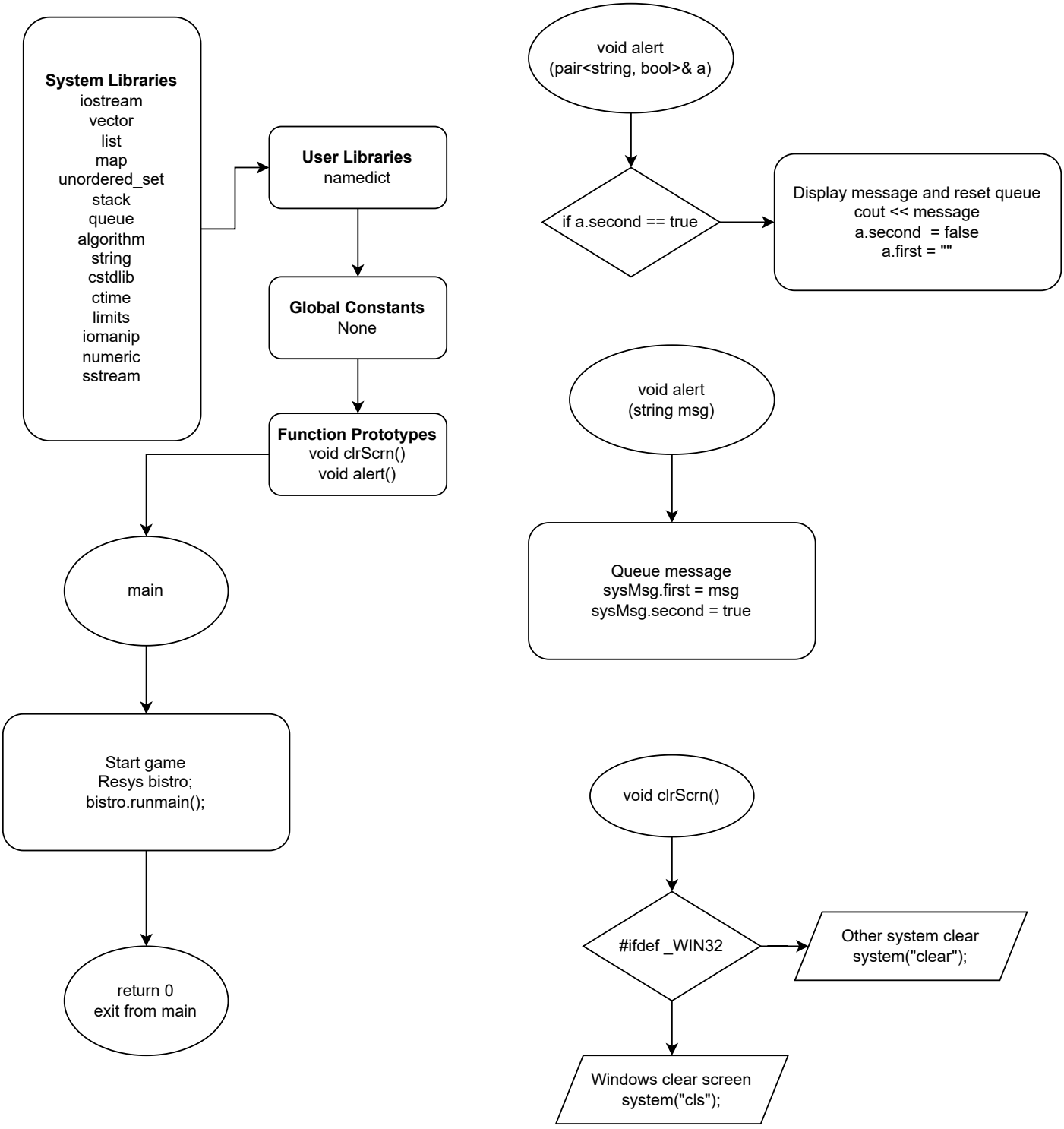
```
// Restaurant System Core
Restaurant:
        has menu, pantry, seating chart, waitlist, etc.

        constructor:
        set up initial values
        seed random
        add starting menu items
        shuffle name dictionary
        set starting cash and rating
        call initializeRestaurant

        initializeRestaurant:
        reset day/hour, seating, and pantry
        build empty tables
        pick inspector guests
        populate tables with some parties

        determineInspectors:
        pick ~10% of name indices randomly
        mark them as inspectors

        addToWaitlist:
        create party name
        add to queue
        alert "Party added to waitlist"

        populateTable(numParties, initialization = false):
        for each party:
        if restaurant is full:
                if not init:
                addToWaitlist
                break
        find empty table
        create random party with names
        mark inspector if detected
        add guests to table

        depopulateTables(showMessages):
        pick random number of tables to clear
        for each:
        if inspector detected:
                alert
```

```
clear guests from table
reduce occupied count
try seating from waitlist after

chargeDailyExpenses:
subtract random daily cost
alert with cost
if money <= 0:
alert "Bankrupt!"
mark game as ended

endDay:
clear all tables
alert how many left
charge expenses

earnFromOrder(dish):
get price
apply rating bonus
add to cash
alert earnings

improveRating:
increase rating by small step
cap at 5.0

isGameEnded:
return whether game is over

advanceTime(random = true):
increase time (optionally random)
if past max time:
alert "Closing time"
end day and start next

getFormattedTime:
return hour and minute as text

runMain:
create instances of menu, guest, and kitchen managers
loop:
show restaurant status (day, time, cash, etc.)
if game over:
        show GAME OVER message
```

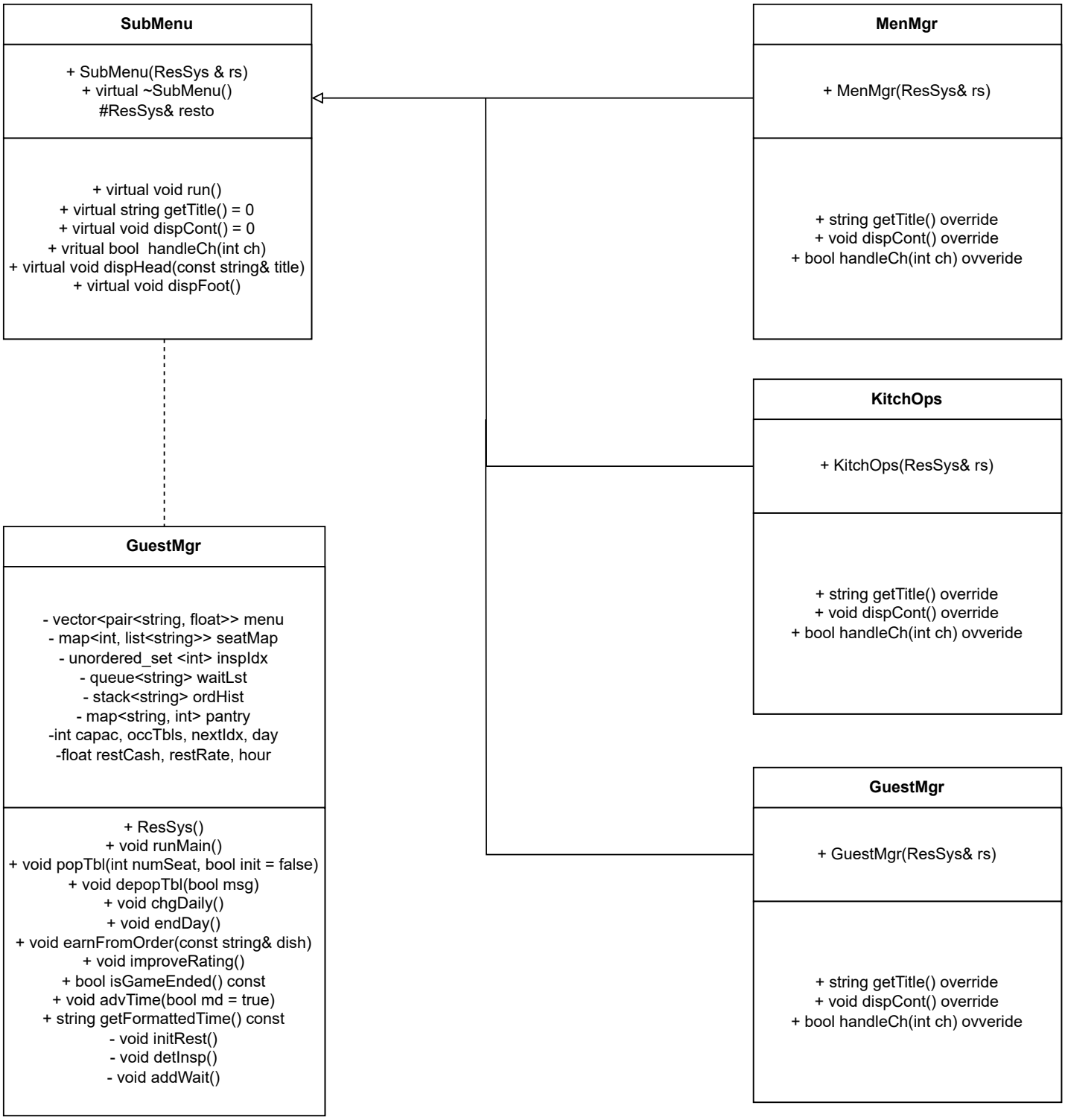
```
        print main options
        get user input
        handle choice:
            1 -> menu
            2 -> guests
            3 -> kitchen
            4 -> end day
            9 -> restart if game ended
            0 -> exit
            else -> alert invalid option

// Program entry point
main:
        create restaurant
        start main game loop
        return
```

STL Project Flowchart

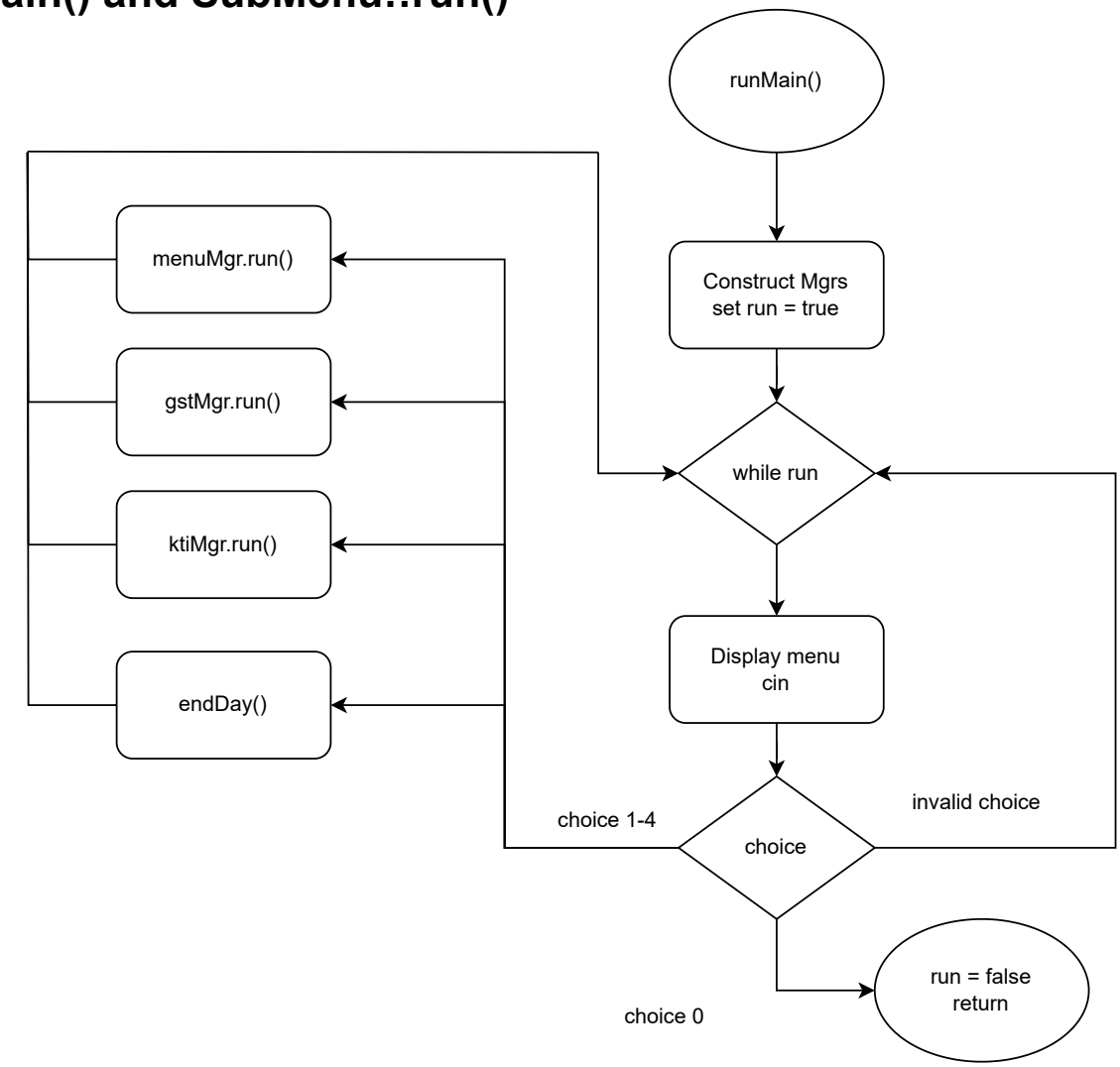
Nathan James Pamintuan - STL Project
CIS17C

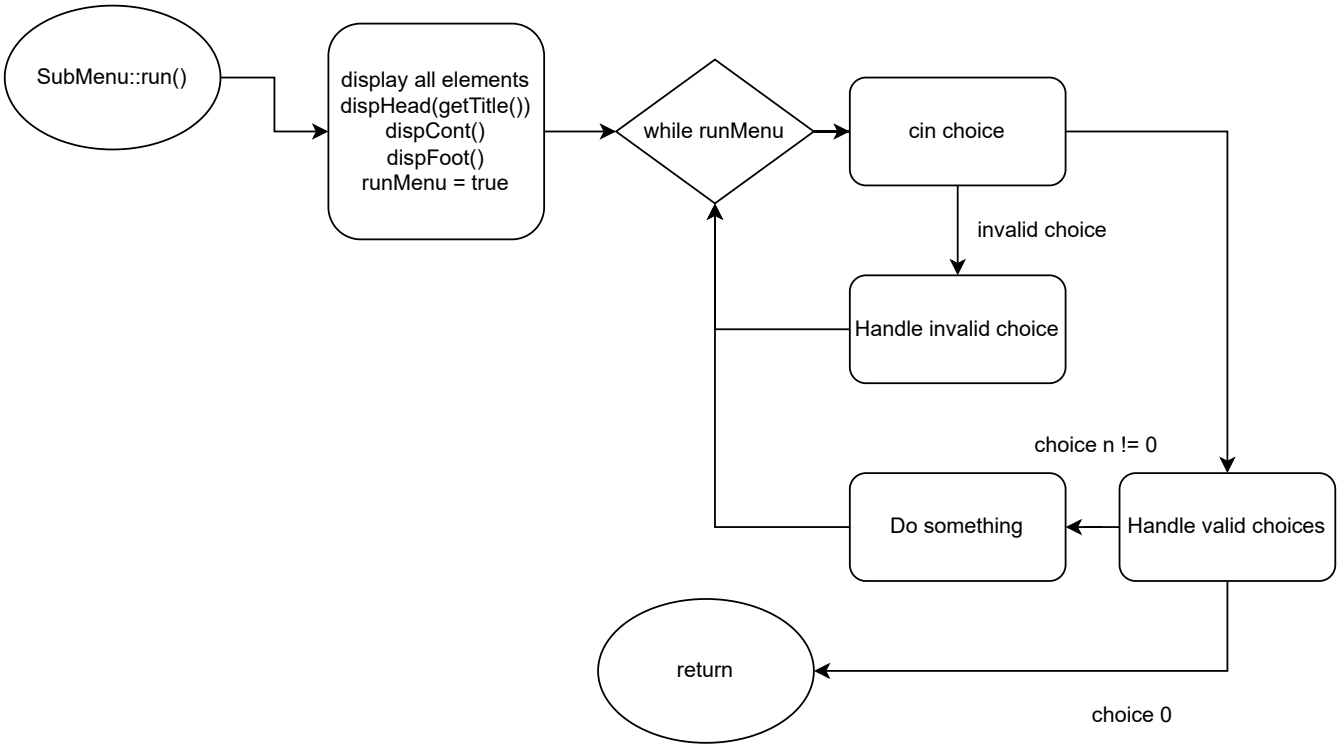


UML Class Diagram

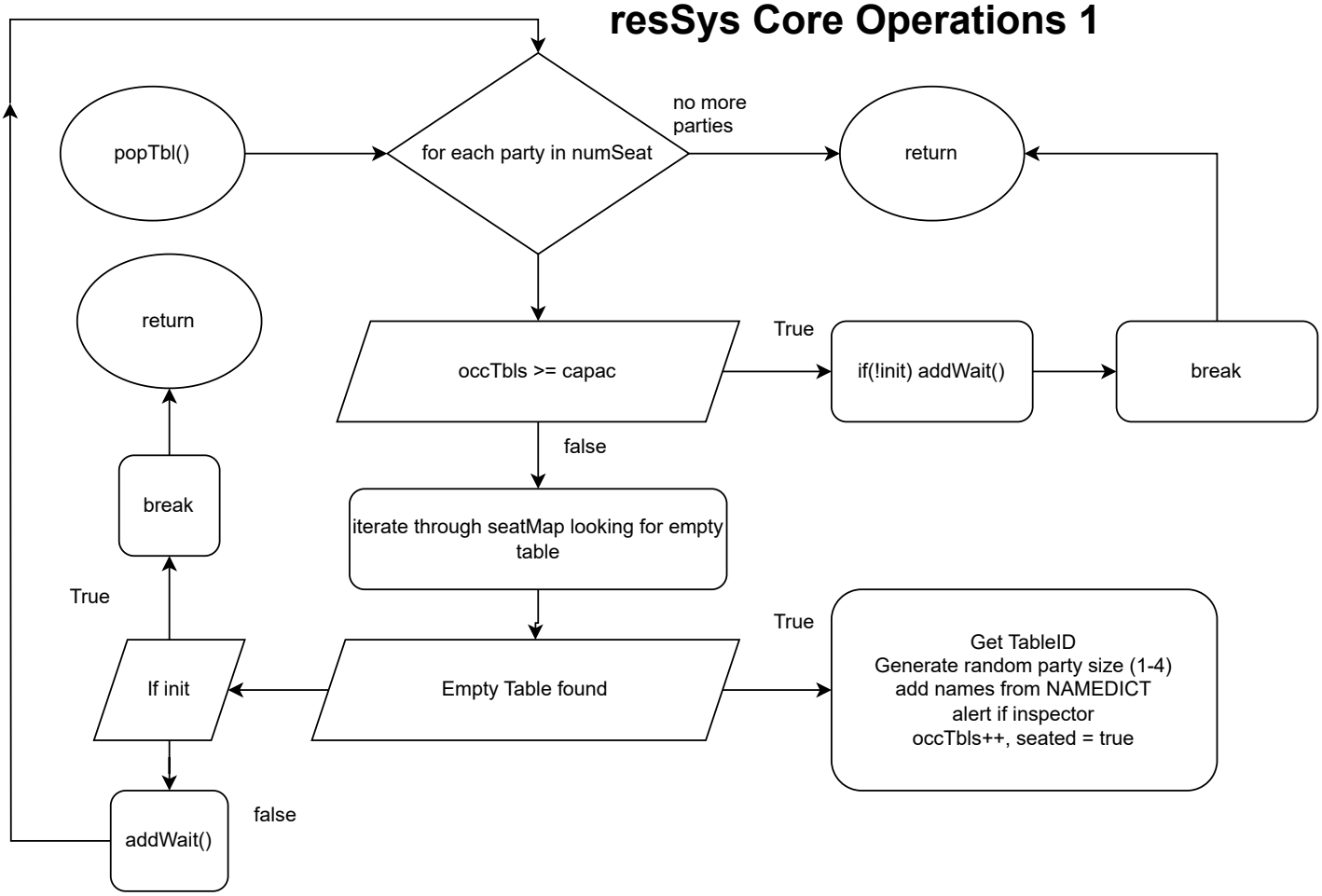


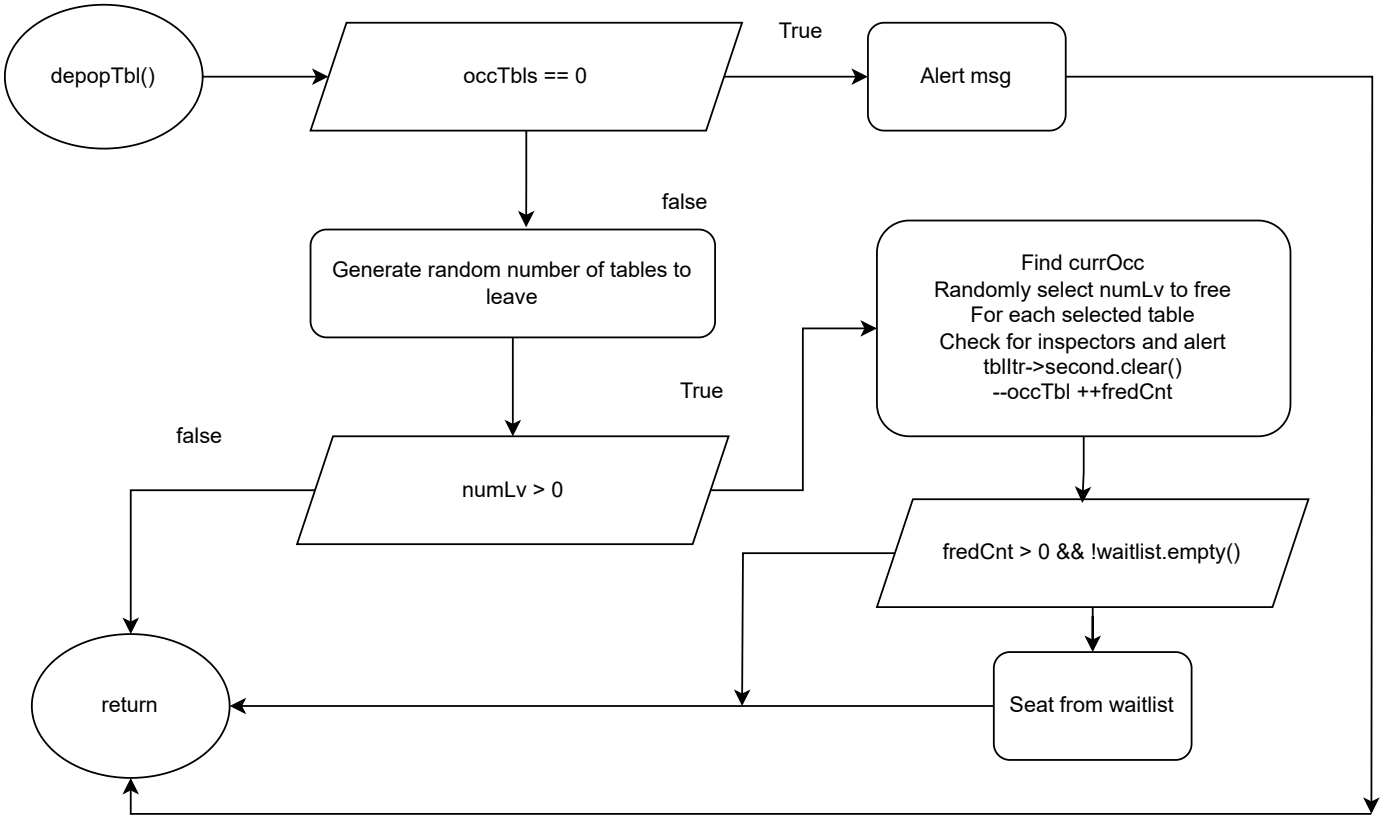
runMain() and SubMenu::run()



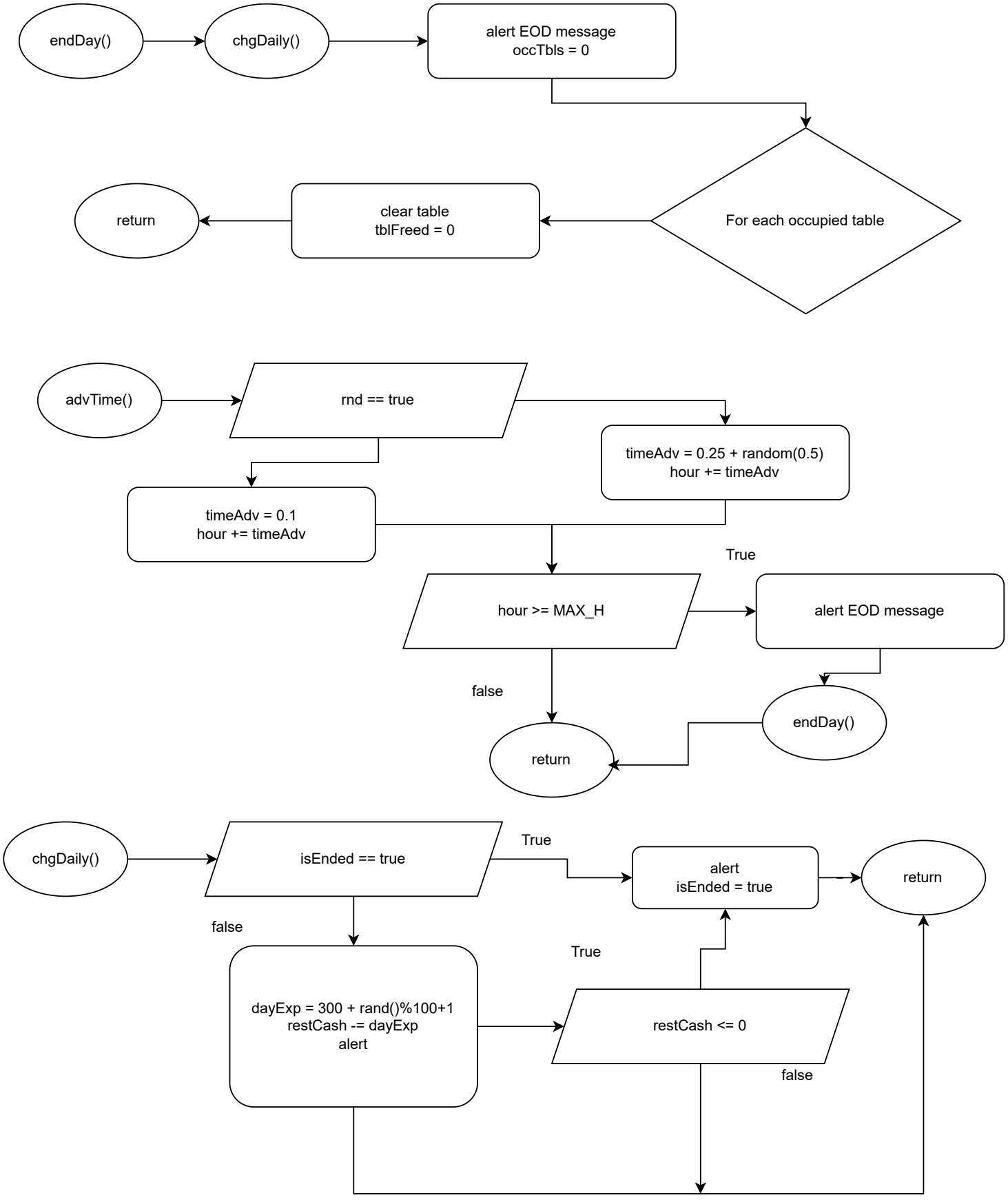


resSys Core Operations 1

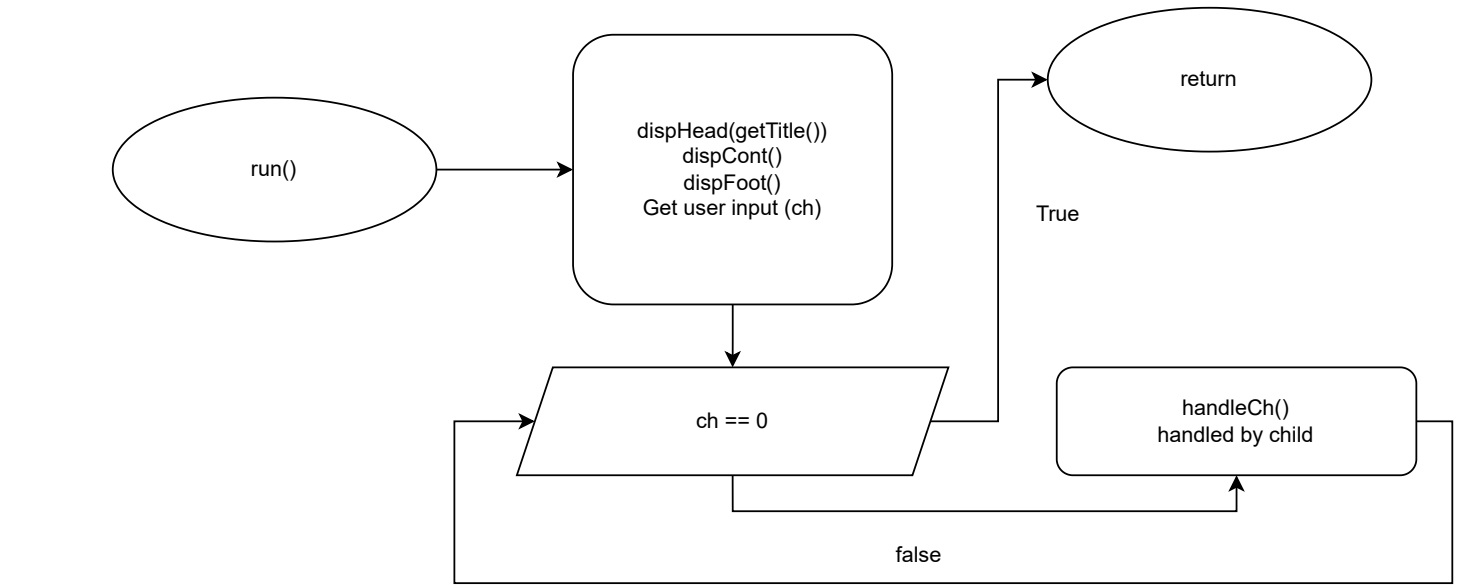


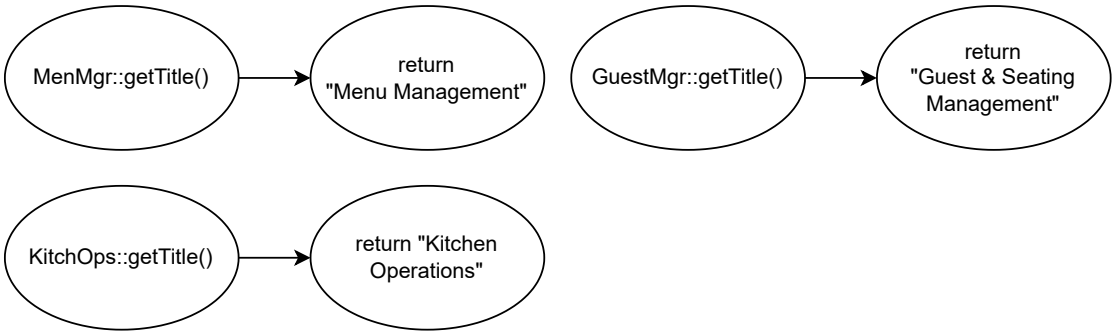


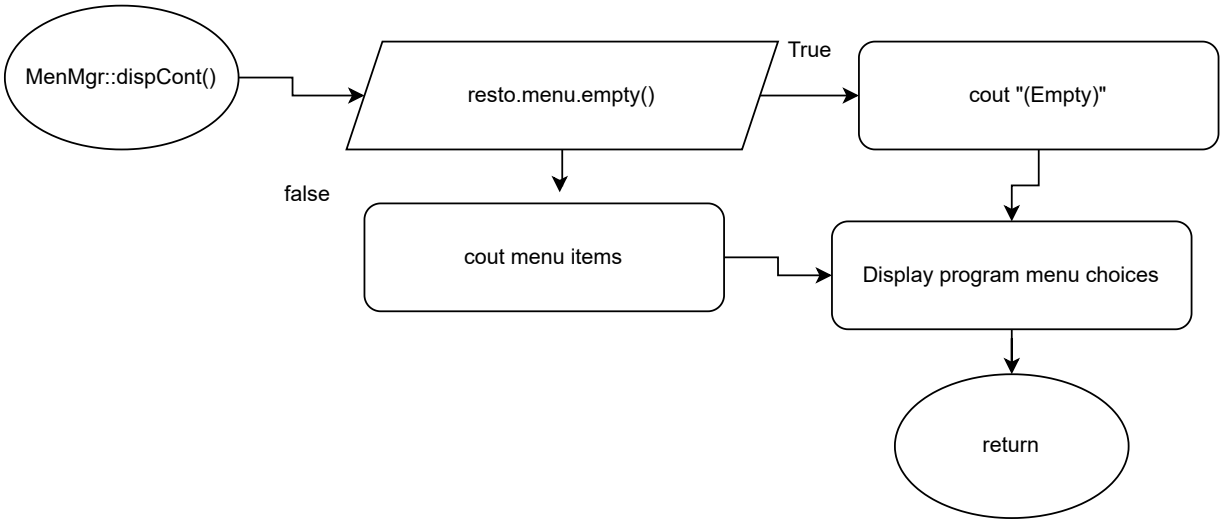
resSys Core Operations 2



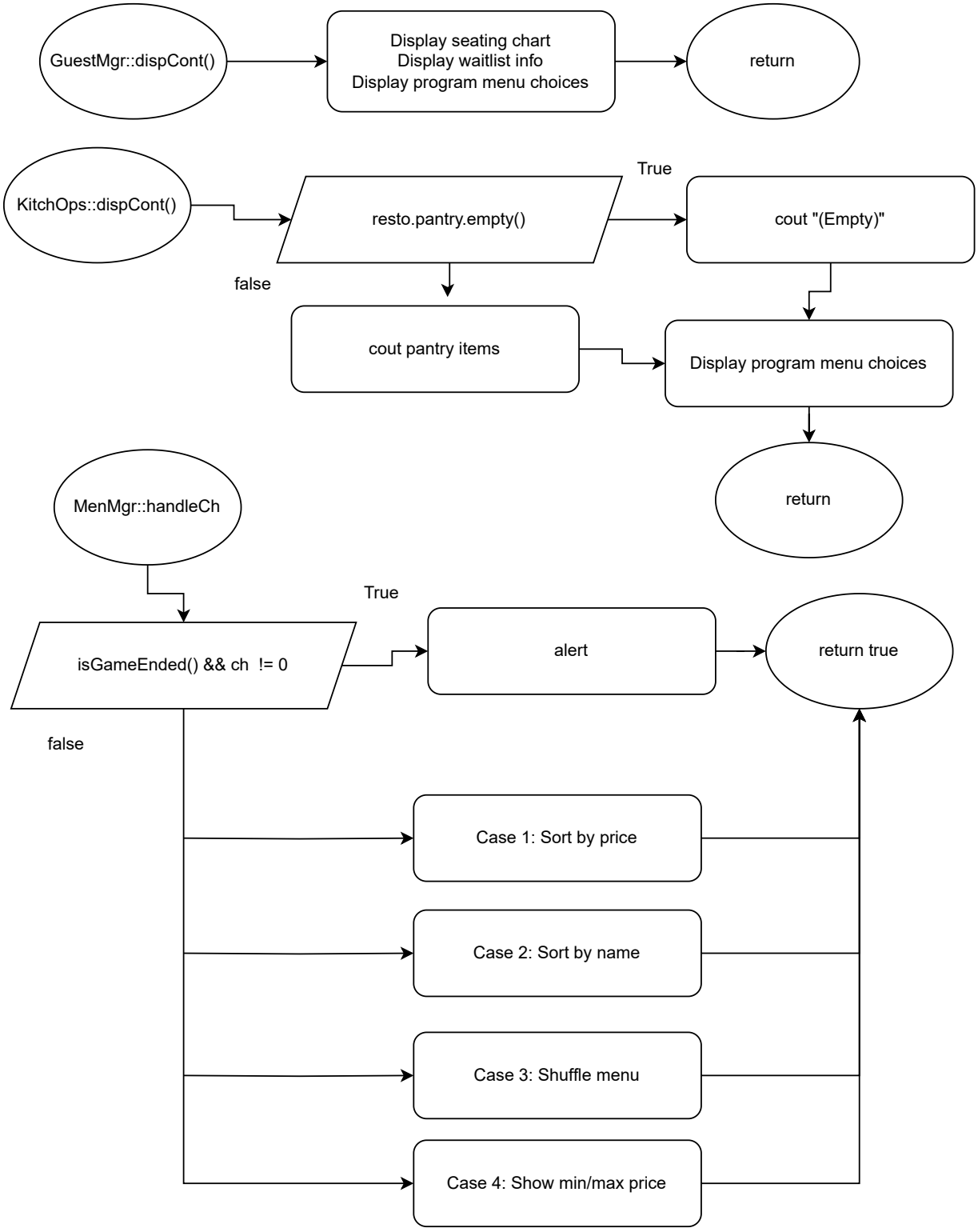
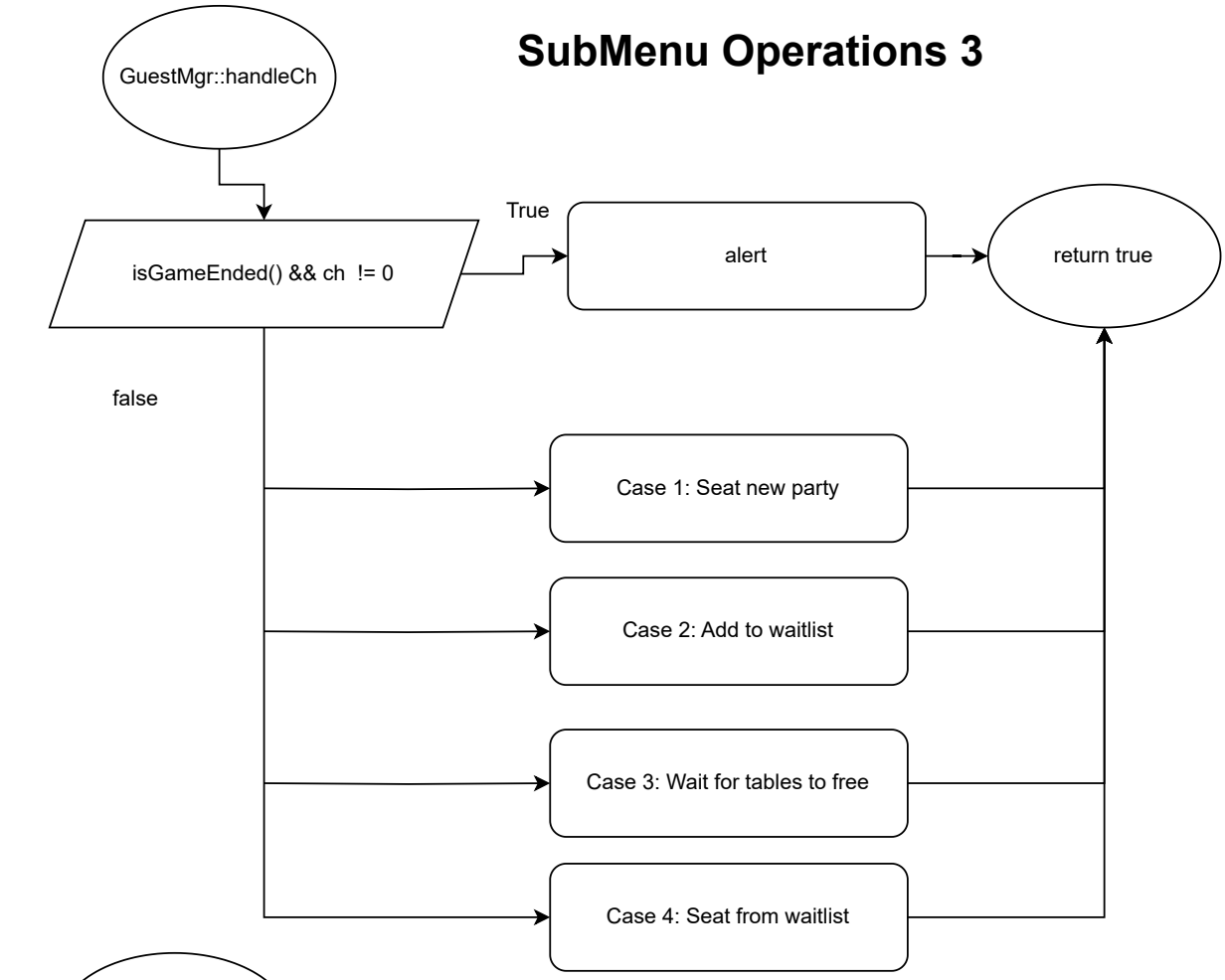
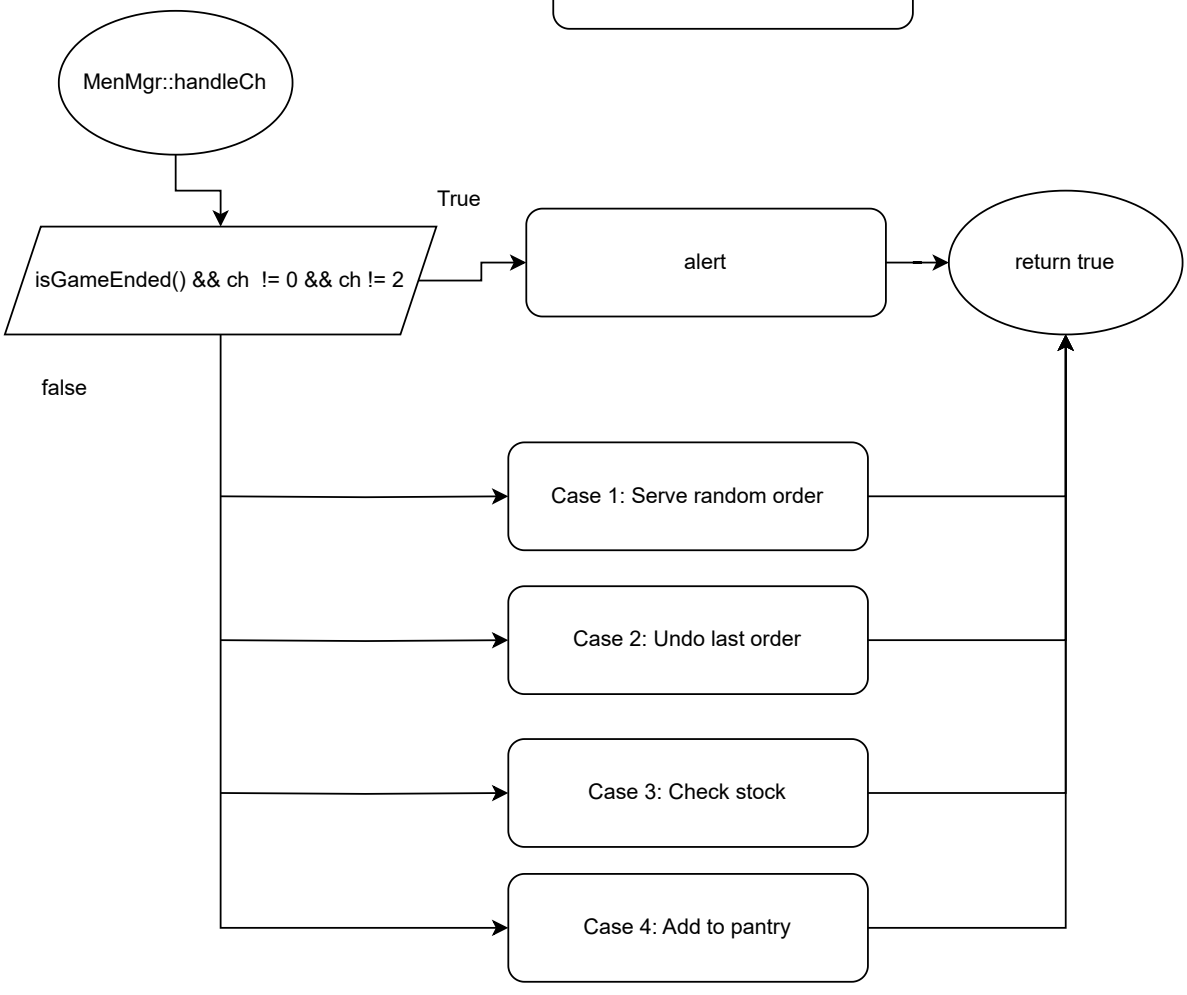
SubMenu Operations 1







SubMenu Operations 2



SubMenu Operations 3