# Assignment Description

This program is a restaurant simulation program. It simulates the happenings of a restaurant over one day, with customers who walk up to the counter, order food, and then wait for their order. A waiter will take their order, then a cook will take the food, cook it, then a waiter will deliver the food.

The user has the ability to hire a varying amount of staff. The user can choose whether they would like to hire more waiters or more cooks, and see what happens. The user can also set the menu, with what the items are named, how much they cost, and how long it takes to make them (in minutes).

Customers and employees are given random names, with employees having random working speeds and customers having random orders.

The simulation is updated every 5 minutes in game, with all durations being rounded to the nearest multiple of 5.

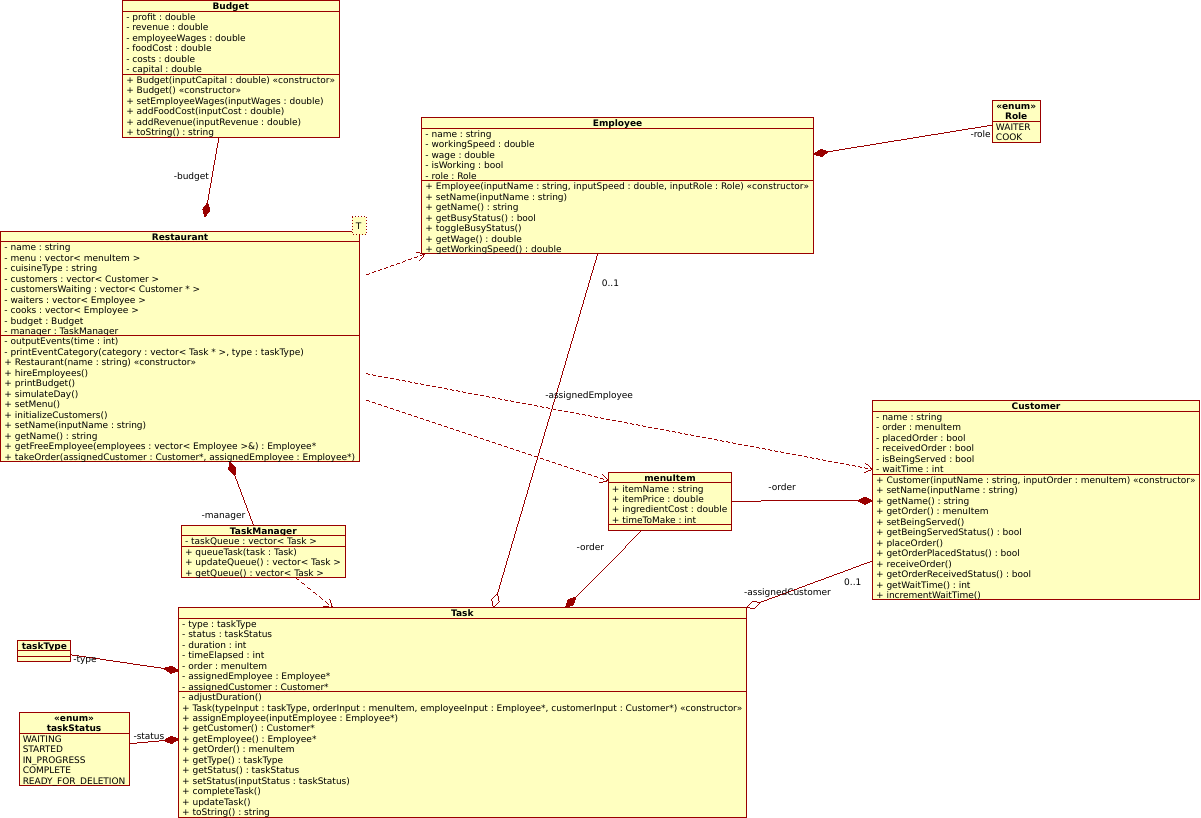
# 1 Readme Documentation

N/A

# 2 Flowchart Screen Shots

N/A

# 3 UML and Use Case Diagrams



UML Class Diagram generated with Umbrello UML Modeller

# 4 Source Code of All files (.h, .cpp)

# Main.cpp

// Main.cpp (54 lines)

#include *<iostream>*

#include *<vector>*

#include *<string>*

#include *<chrono>*

#include *<thread>*

#include *<ctime>*

#include *"Restaurant.h"*

**using** **namespace** **std**;

*/\**

*Project Name: Restaurant Simulator*

*Author: Wesley Hixon*

*Date Last Updated: 12/13/2024*

*\*/*

int main(){

cout << "Welcome to the Restaurant Simulator!" << endl;

cout << "This program will allow you to live out your dreams of owning a restaurant - without the hard parts!" << endl;

cout << endl << " ---Setup--- " << endl;

cout << endl << "1. Choose a name for your restaurant: ";

*// Get restaurant name*

string restaurantName = getStringInput();

cout << endl << "Your restaurant is called **\"**" << restaurantName << "**\"**"<< endl;

Restaurant myRestaurant(restaurantName);

*// Create menu*

myRestaurant.setMenu();

*// Seed rand number generator*

srand(time(0));

myRestaurant.hireEmployees();

this\_thread::sleep\_for(chrono::seconds(2));

*// Create vector of customers*

myRestaurant.initializeCustomers();

myRestaurant.printBudget();

this\_thread::sleep\_for(chrono::seconds(3));

*// Simulate a day*

myRestaurant.simulateDay();

myRestaurant.printBudget();

**return** 0;

}

# Restaurant.h

// Restaurant.h (90 lines)

#pragma once

#include *<iostream>*

#include *<string>*

#include *<vector>*

#include *<random>*

#include *"People.h"*

#include *"Budget.h"*

#include *"Task.h"*

**using** **namespace** **std**;

**template** <**typename** **T**>

T getNumInput(double lowerBound, double upperBound){

string userInput;

T num;

bool valid = false;

**while**(!valid){

**try**{

cin >> num;

**if**(num < lowerBound || num > upperBound) **throw**(runtime\_error("Out of range"));

valid = true;

}**catch**(**const** exception& e){

cerr << "Please enter a valid input. Try again." << endl;

cin.clear();

cin.ignore(10000, '\n');

}

}

**return** num;

}

double generateRandomDouble(double lowerBound, double upperBound);

string getStringInput();

**class** **Restaurant**{

**private**:

string name;

vector<menuItem> menu;

string cuisineType;

vector<Customer> customers;

vector<Customer\*> customersWaiting;

vector<Employee> waiters;

vector<Employee> cooks;

Budget budget;

TaskManager manager;

*// Outputs events at a certain time in minutes*

void outputEvents(int time);

void printEventCategory(vector<Task\*> category, taskType type);

**public**:

*// Constructor*

Restaurant(string name);

*// Hires waiters and cooks*

void hireEmployees();

void printBudget();

*// Starts simulation of a 5 hour day*

void simulateDay();

*// Sets menu*

void setMenu();

*// Creates 5 new customers with random names and orders*

*// Then, adds them to array*

void initializeCustomers();

*// Name operations*

void setName(string inputName);

string getName();

*// Returns a free employee from employees vector*

*// If no free employees, returns nullptr*

Employee\* getFreeEmployee(vector<Employee>& employees);

*// Takes a new order from customer*

void takeOrder(Customer\* assignedCustomer, Employee\* assignedEmployee);

};

# Restaurant.cpp

// Restaurant.cpp (384 lines)

#include *<iostream>*

#include *<iomanip>*

#include *<thread>*

#include *<chrono>*

#include *"Restaurant.h"*

#include *"Task.h"*

**using** **namespace** **std**;

string getStringInput(){

string userInput;

bool valid = false;

**while**(!valid){

*// Clear leftover newline character*

**if**(cin.peek() == '\n'){

cin.ignore();

}

getline(cin, userInput);

**if**(!cin){

cerr << "Please enter a valid input." << endl;

cin.clear();

cin.ignore(10000, '\n');

}**else**{

valid = true;

}

}

**return** userInput;

}

double generateRandomDouble(double lowerBound, double upperBound){

random\_device rd;

mt19937 gen(rd());

uniform\_real\_distribution<double> dis(lowerBound, upperBound);

**return** dis(gen);

}

vector<string> randomNames = {

"Olivia", "Liam", "Emma", "Noah", "Ava", "William", "Sophia", "James", "Isabella",

"Benjamin", "Mia", "Lucas", "Charlotte", "Henry", "Amelia", "Alexander", "Evelyn",

"Michael", "Harper", "Ethan", "Luna", "Daniel", "Ella", "Matthew", "Scarlett",

"David", "Victoria", "Joseph", "Chloe", "Samuel", "Grace", "Jackson", "Zoey",

"Sebastian", "Lily", "Aiden", "Penelope", "Caleb", "Hannah", "Mason", "Aria",

"Luke", "Audrey", "Nathan", "Nora", "Logan", "Riley", "Dylan", "Violet", "Owen"

};

Restaurant::Restaurant(string inputName): name(inputName){};

*// Hires waiters and cooks*

void Restaurant::hireEmployees(){

*// Calculate number of employees from initial capital and wages*

int numEmployees = 5;

int maxCooks = numEmployees - 1;

*// Get number of cooks*

cout << endl << " ---Hiring Employees--- " << endl;

cout << "How many cooks would you like to hire? (Maximum " << maxCooks << "): ";

int numCooks = getNumInput<int>(1, maxCooks);

int maxWaiters = numEmployees - numCooks;

*// Get number of waiters*

cout << "How many waiters would you like to hire? (Maximum " + to\_string(maxWaiters) + "): ";

int numServers = getNumInput<int>(1, maxWaiters);

cout << endl << " ---Generating Employees--- " << endl;

cout << "Cooks: " << numCooks << endl;

cout << "Waiters: " << numServers << endl;

*// Generate cooks*

**for**(int i = 0; i < numCooks; i++){

int randomNameIndex = rand() % 50;

string newName = randomNames[randomNameIndex];

double randomSpeed = generateRandomDouble(0.5, 1.5);

Employee newCook(newName, randomSpeed, COOK);

cooks.push\_back(newCook);

}

*// Generate waiters*

**for**(int i = 0; i < numServers; i++){

int randomNameIndex = rand() % 50;

string newName = randomNames[randomNameIndex];

double randomSpeed = generateRandomDouble(0.5, 1.5);

Employee newWaiter(newName, randomSpeed, WAITER);

waiters.push\_back(newWaiter);

}

*// Employees make 10 bucks an hour*

int hourlyWage = 10;

double wages = (numCooks + numServers) \* hourlyWage;

budget.setEmployeeWages(wages);

};

*// Creates 15 new customers with random names and orders*

*// Then, adds them to vector*

void Restaurant::initializeCustomers(){

int numCustomers = 15;

**for**(int i = 0; i < numCustomers; i++){

*// Gets a random name*

int randomNameIndex = rand() % 50;

string newName = randomNames[randomNameIndex];

*// Gets a random menu item*

int randomMenuItemIndex = rand() % menu.size();

menuItem randomMenuItem = menu[randomMenuItemIndex];

*// create new customer with random name and order*

Customer newCustomer(newName, randomMenuItem);

customers.push\_back(newCustomer);

}

}

void Restaurant::setName(string inputName){name = inputName;}

string Restaurant::getName(){**return** name;}

void Restaurant::printBudget(){

cout << endl << " ---Budget--- " << endl;

cout << budget << endl;

}

*// Simulates day*

*// Day is 5 hours long*

*// Simulation is updated every 5 minutes, time is 1 minute increments*

*// Task completion is rounded to nearest 5 minutes*

void Restaurant::simulateDay(){

cout << endl << " ---Beginning Simulation--- " << endl;

*// 1 - 2 customers walk up to counter (25% chance)*

int orderRate = 25;

int hours = 5;

int minutes = hours \* 60;

**for**(int time = 0; time < minutes; time += 5){

*// First, update queue*

vector<Task> newTasks = manager.updateQueue();

*// Assign employees to new tasks*

**for**(**auto** itr = newTasks.begin(); itr != newTasks.end();){

Employee\* freeEmployee;

**auto** newTask = \*itr;

**if**(newTask.getType() == DELIVER\_ORDER){

freeEmployee = getFreeEmployee(waiters);

}

**else**{

freeEmployee = getFreeEmployee(cooks);

}

*// If no free employees for this task, skip it*

**if**(freeEmployee == **nullptr**){

itr++;

**continue**;

}

newTask.assignEmployee(freeEmployee);

manager.queueTask(newTask);

newTasks.erase(itr);

}

*// 25% chance customer orders*

bool customerOrders = (rand() % 100) < orderRate;

*// Adds customers to the line for ordering*

**if**(customerOrders){

int maxCustomers = 2;

int numCustomers = (rand() % maxCustomers) + 1;

int customersAdded = 0;

**for**(**auto** itr = customers.begin(); itr != customers.end(); itr++){

**if**(customersAdded == numCustomers){

**break**;

}

*// If customer is being served, skip them*

**if**(itr->getBeingServedStatus() == true){

**continue**;

}

*// Convert iterator to pointer*

Customer\* customerPtr = &(\*itr);

*// If they aren't being served, add them to the line of customers waiting to be served*

customersWaiting.push\_back(customerPtr);

customersAdded++;

}

customerOrders = false;

}

*// Take customer orders*

**for**(**auto** itr = customersWaiting.begin(); itr != customersWaiting.end();){

Employee\* freeWaiter = getFreeEmployee(waiters);

Customer\* customer = \*itr;

*// In case there are no free waiters*

**if**(freeWaiter == **nullptr**){

**break**;

}

takeOrder(customer, freeWaiter);

customersWaiting.erase(itr);

}

outputEvents(time);

*// Wait a 1 second between steps*

this\_thread::sleep\_for(chrono::seconds(1));

}

}

void Restaurant::outputEvents(int time){

int hour = time / 60;

int minute = time % 60;

vector<Task\*> takingOrders;

vector<Task\*> cookingOrders;

vector<Task\*> deliveringOrders;

*// Outputting time*

cout << endl << "Time: " << setw(2) << setfill('0') << hour << ":" << setw(2) << setfill('0') << minute << endl;

this\_thread::sleep\_for(chrono::milliseconds(500));

vector<Task> queue = manager.getQueue();

*// Separate the tasks into types*

**for**(**auto** itr = queue.begin(); itr != queue.end(); itr++){

Task\* task = &(\*itr);

**if**(itr->getStatus() == IN\_PROGRESS){

**continue**;

}

**switch**(itr->getType()){

**case** TAKE\_ORDER:

takingOrders.push\_back(task);

**break**;

**case** MAKE\_ORDER:

cookingOrders.push\_back(task);

**break**;

**case** DELIVER\_ORDER:

deliveringOrders.push\_back(task);

**break**;

}

}

*// If no events,*

bool noEvents = takingOrders.empty() && cookingOrders.empty() && deliveringOrders.empty();

**if**(noEvents){

**return**;

}

*// Print each type*

printEventCategory(takingOrders, TAKE\_ORDER);

printEventCategory(cookingOrders, MAKE\_ORDER);

printEventCategory(deliveringOrders, DELIVER\_ORDER);

**return**;

}

void Restaurant::printEventCategory(vector<Task\*> category, taskType type){

string categoryText;

**switch**(type){

**case** TAKE\_ORDER:

categoryText = " - Taking Orders: ";

**break**;

**case** MAKE\_ORDER:

categoryText = " - Making Orders: ";

**break**;

**case** DELIVER\_ORDER:

categoryText = " - Delivering Orders: ";

**break**;

}

**if**(category.empty() == false){

this\_thread::sleep\_for(chrono::milliseconds(200));

cout << endl << categoryText << endl;

}

**for**(**auto** itr = category.begin(); itr != category.end(); itr++){

this\_thread::sleep\_for(chrono::milliseconds(200));

Task\* task = \*itr;

cout << " - " << \*task << endl;

}

}

*// Creates menu of 1 - 5 items with user input*

void Restaurant::setMenu(){

cout << endl << " ---Menu Creation--- " << endl;

cout << "Now, create 1 - 5 menu items for your menu." << endl;

cout << "How many items would you like to create? ";

int numMenuItems = getNumInput<int>(1,5);

**for**(int i = 0; i < numMenuItems; i++){

cout << endl << "Enter a name for menu item number " << i + 1 << ": ";

string itemName = getStringInput();

cout << "Enter a price for this item between 5 and 20 dollars: ";

double itemCost = getNumInput<double>(5, 20);

cout << "Enter how many minutes (between 10 and 30) it takes to make this item: ";

int timeToMake = getNumInput<int>(10,30);

menuItem newMenuItem;

newMenuItem.itemName = itemName;

newMenuItem.itemPrice = itemCost;

newMenuItem.ingredientCost = itemCost \* .30; *// Ingredient cost is 30% of price*

newMenuItem.timeToMake = timeToMake;

*// Adds menu item to menu*

menu.push\_back(newMenuItem);

}

cout << "Here is your new menu:" << endl;

**for**(**auto** itr = menu.begin(); itr != menu.end(); itr++){

int itemNum = distance(menu.begin(), itr) + 1;

cout << itemNum << ". " << itr->itemName << endl;

cout << " - Cost: " << itr->itemPrice << endl;

cout << " - Ingredient Cost: " << itr->ingredientCost << endl;

cout << " - Time to make: " << itr->timeToMake << " minutes" << endl;

}

}

*// Starts a customer's order*

void Restaurant::takeOrder(Customer\* assignedCustomer, Employee\* assignedEmployee){

menuItem order = assignedCustomer->getOrder();

Task newOrder(TAKE\_ORDER, order, assignedEmployee, assignedCustomer);

budget.addRevenue(order.itemPrice);

budget.addFoodCost(order.ingredientCost);

manager.queueTask(newOrder);

}

*// Returns a free employee from an employee vector*

Employee\* Restaurant::getFreeEmployee(vector<Employee>& employees){

*// Iterates through employees to find a free employee*

**for**(**auto**& employee: employees){

**if**(employee.getBusyStatus() == false){

Employee\* employeePtr = &employee;

**return** employeePtr;

}

}

*// If none found, return nullptr*

**return** **nullptr**;

}

# Task.h

// Task.h (54 lines)

#pragma once

#include *"People.h"*

#include *<vector>*

**enum** **taskType**{

TAKE\_ORDER,

MAKE\_ORDER,

DELIVER\_ORDER

};

**enum** **taskStatus**{

WAITING,

STARTED,

IN\_PROGRESS,

COMPLETE,

READY\_FOR\_DELETION

};

**class** **Task**{

**private**:

taskType type;

taskStatus status = WAITING;

int duration;

int timeElapsed = 0;

menuItem order;

Employee\* assignedEmployee;

Customer\* assignedCustomer = **nullptr**;

void adjustDuration();

**public**:

Task(taskType typeInput, menuItem orderInput, Employee\* employeeInput = **nullptr**, Customer\* customerInput = **nullptr**);

void assignEmployee(Employee\* inputEmployee);

Customer\* getCustomer();

Employee\* getEmployee();

menuItem getOrder();

taskType getType();

taskStatus getStatus();

void setStatus(taskStatus inputStatus);

void completeTask();

void updateTask();

string toString();

**friend** ostream& **operator**<<(ostream& os, Task inputTask);

};

**class** **TaskManager**{

**private**:

vector<Task> taskQueue;

**public**:

void queueTask(Task task);

vector<Task> updateQueue();

vector<Task> getQueue();

};

# Task.cpp

// Task.cpp (228 Lines)

#include *"People.h"*

#include *"Task.h"*

#include *<vector>*

#include *<cmath>*

#include *<iostream>*

**using** **namespace** **std**;

*// Adds task to task queue*

void TaskManager::queueTask(Task task){

taskQueue.push\_back(task);

}

*// Updates task queue and returns a vector of new tasks to be assigned to an employee*

vector<Task> TaskManager::updateQueue(){

vector<Task> newTasks;

*// Iterate through task queue*

**for**(**auto** itr = taskQueue.begin(); itr != taskQueue.end();){

*// Check if task completed and update the task*

itr->updateTask();

taskStatus status = itr->getStatus();

*// Erase*

**if**(status == READY\_FOR\_DELETION){

itr = taskQueue.erase(itr);

**continue**;

}

*// If task not completed, go to next task*

**if**(status != COMPLETE){

itr++;

**continue**;

}

*// If task is completed, start building the next task*

taskType oldTaskType = itr->getType();

taskType newTaskType;

*// Create new task type based on previous*

**switch**(oldTaskType){

**case** taskType::TAKE\_ORDER:

newTaskType = taskType::MAKE\_ORDER;

**break**;

**case** taskType::MAKE\_ORDER:

newTaskType = taskType::DELIVER\_ORDER;

**break**;

}

*// If task still has steps, create a new task which is next in sequence*

*// Otherwise, just mark it for deletion*

**if**(oldTaskType == TAKE\_ORDER || oldTaskType == MAKE\_ORDER){

Customer\* customer = itr->getCustomer();

menuItem order = customer->getOrder();

Task newTask(newTaskType, order, **nullptr**, customer);

newTasks.push\_back(newTask);

itr->setStatus(READY\_FOR\_DELETION);

}

**else** **if**(oldTaskType == DELIVER\_ORDER){

itr->setStatus(READY\_FOR\_DELETION);

}

itr++;

}

**return** newTasks;

}

vector<Task> TaskManager::getQueue(){**return** taskQueue;}

*// Task constructor, assigns employee and sets duration if employee is input*

Task::Task(taskType typeInput, menuItem orderInput, Employee\* employeeInput, Customer\* customerInput):

type(typeInput),

order(orderInput),

assignedEmployee(employeeInput),

assignedCustomer(customerInput)

{

**if**(assignedEmployee == **nullptr**){

**return**;

}

*// Assign employee and adjust duration*

**else**{

assignEmployee(employeeInput);

}

}

*// Updates time elapsed and returns true if task completed*

void Task::updateTask(){

timeElapsed += 5;

*// Updates status*

**if**(status == STARTED){

status = IN\_PROGRESS;

}

**if**(timeElapsed == duration){

completeTask();

}

}

*// Updates employee and customer status*

void Task::completeTask(){

assignedEmployee->toggleBusyStatus();

status = COMPLETE;

*// Change customer status*

**if**(type == TAKE\_ORDER){

assignedCustomer->placeOrder();

}

**if**(type == DELIVER\_ORDER){

assignedCustomer->receiveOrder();

}

**return**;

}

*// Tasktype*

taskType Task::getType(){**return** type;}

*// Status operations*

taskStatus Task::getStatus(){**return** status;}

void Task::setStatus(taskStatus inputStatus){status = inputStatus;}

*// Order*

menuItem Task::getOrder(){**return** order;}

*// Get employee and customer*

Employee\* Task::getEmployee(){**return** assignedEmployee;}

Customer\* Task::getCustomer(){**return** assignedCustomer;}

*// Assigns employee and adjusts duration*

void Task::assignEmployee(Employee\* inputEmployee){

assignedEmployee = inputEmployee;

assignedEmployee->toggleBusyStatus();

status = STARTED;

adjustDuration();

*// Change the customer's status to being served, if not already*

**if**(assignedCustomer->getBeingServedStatus() == false){

assignedCustomer->setBeingServed();

}

}

void Task::adjustDuration(){

*// Default durations of tasks*

int durationPlaceOrder = 5;

int durationMakeOrder = order.timeToMake; *// Depends on menu item*

int durationDeliverOrder = 10;

*// Getting employee speed and cooking speed*

double employeeSpeed = assignedEmployee->getWorkingSpeed();

double adjustedDuration;

*// Adjusting duration*

**if**(type == TAKE\_ORDER || type == DELIVER\_ORDER){

adjustedDuration = durationPlaceOrder \* employeeSpeed;

}

**else** **if**(type == MAKE\_ORDER){

adjustedDuration = durationMakeOrder \* employeeSpeed;

}

*// This will round the duration to nearest multiple of 5*

int roundedDuration = round(adjustedDuration / 5) \* 5;

duration = roundedDuration;

}

string Task::toString(){

string statusText;

string taskString;

string employeeString;

string taskTag;

employeeString = assignedEmployee->getName();

**switch**(status){

**case** WAITING:

**return** string("Error: No Employee Assigned");

**case** STARTED:

statusText = " has started ";

taskTag = "[ACTION] ";

**break**;

**case** IN\_PROGRESS:

statusText = " is still ";

**break**;

**case** COMPLETE: **case** READY\_FOR\_DELETION:

statusText = " has finished ";

taskTag = "[COMPLETE] ";

**break**;

}

**switch**(type){

**case** TAKE\_ORDER:

employeeString += " (Waiter)";

statusText += "taking ";

**break**;

**case** MAKE\_ORDER:

employeeString += " (Cook)";

statusText += "making ";

**break**;

**case** DELIVER\_ORDER:

employeeString += " (Waiter)";

statusText += "delivering ";

**break**;

}

taskString = taskTag + employeeString + statusText + assignedCustomer->getName() + "'s order of " + order.itemName;

**return** taskString;

}

ostream& **operator**<<(ostream& os, Task inputTask){

os << inputTask.toString();

**return** os;

}

# People.h

// People.h (72 Lines)

#pragma once

#include *<string>*

**using** **namespace** **std**;

**enum** **Role**{

WAITER, COOK

};

**class** **Employee**{

**private**:

string name;

double workingSpeed; *// Speed is double between 0.5 and 1.5*

double wage = 10;

bool isWorking = false;

Role role;

**public**:

Employee(string inputName, double inputSpeed, Role inputRole);

void setName(string inputName);

string getName();

bool getBusyStatus();

void toggleBusyStatus();

double getWage();

double getWorkingSpeed();

};

**struct** **menuItem**{

string itemName;

double itemPrice;

double ingredientCost;

int timeToMake; *// Minutes between 10 and 30*

};

**class** **Customer**{

**private**:

string name;

menuItem order;

bool placedOrder = false;

bool receivedOrder = false;

bool isBeingServed = false;

int waitTime = 0;

**public**:

*// Constructor*

Customer(string inputName, menuItem inputOrder);

*// Name operations*

void setName(string inputName);

string getName();

*// Get menu order*

menuItem getOrder();

*// Being served flag*

void setBeingServed();

bool getBeingServedStatus();

*// Placing orders*

void placeOrder();

bool getOrderPlacedStatus();

*// Receiving orders*

void receiveOrder();

bool getOrderReceivedStatus();

*// Wait time logic*

int getWaitTime();

void incrementWaitTime();

};

// People.cpp (49 lines)

#include *"People.h"*

#include *<iostream>*

#include *<string>*

**using** **namespace** **std**;

*// Customer Constructor*

Customer::Customer(string inputName, menuItem inputOrder): name(inputName), order(inputOrder) {};

*// Name Operations*

void Customer::setName(string inputName){name = inputName;}

string Customer::getName(){**return** name;}

*// Being served operations*

void Customer::setBeingServed(){isBeingServed = true;}

bool Customer::getBeingServedStatus(){**return** isBeingServed;}

*// Increments time waiting by 5 minutes*

int Customer::getWaitTime(){**return** waitTime;}

void Customer::incrementWaitTime(){waitTime += 5;}

*// Placing orders*

bool Customer::getOrderPlacedStatus(){**return** placedOrder;}

void Customer::placeOrder(){placedOrder = true;}

*// Receiving order*

bool Customer::getOrderReceivedStatus(){**return** receivedOrder;}

void Customer::receiveOrder(){receivedOrder = true;}

*// Getting menu order*

menuItem Customer::getOrder(){**return** order;}

*// Employee Constructor*

Employee::Employee(string inputName, double inputSpeed, Role inputRole): name(inputName), workingSpeed(inputSpeed), role(inputRole) {};

*// Name operations*

void Employee::setName(string inputName){name = inputName;}

string Employee::getName(){**return** name;}

*// Busy status operations*

bool Employee::getBusyStatus(){**return** (isWorking);}

void Employee::toggleBusyStatus(){isWorking = !isWorking;}

*// Gets wage*

double Employee::getWage(){**return** wage;}

*// Gets speed*

double Employee::getWorkingSpeed(){**return** workingSpeed;}

# Budget.h

// Budget.h (24 lines)

#pragma once

#include *<iostream>*

**using** **namespace** **std**;

**class** **Budget**{

**private**:

double profit = 0;

double revenue = 0;

double employeeWages = 0;

double foodCost = 0;

double costs = 0;

double capital = 500;

**public**:

Budget(double inputCapital);

Budget();

void setEmployeeWages(double inputWages);

void addFoodCost(double inputCost);

void addRevenue(double inputRevenue);

string toString();

**friend** ostream& **operator**<<(ostream& os, Budget budget);

};

# Budget.cpp

// Budget.cpp (47 lines)

#include *<iostream>*

#include *<iomanip>*

#include *<sstream>*

#include *"Budget.h"*

**using** **namespace** **std**;

Budget::Budget(double inputCapital): capital(inputCapital) {}

Budget::Budget() {}

string Budget::toString(){

stringstream budgetSS;

budgetSS << fixed << showpoint << setprecision(2);

budgetSS << "Starting Capital: " << capital;

budgetSS << "**\n**Revenue: " << revenue;

budgetSS << "**\n**Costs: " << costs;

budgetSS << "**\n**Profits: " << profit;

budgetSS << "**\n**Ending Capital: " << capital + profit;

string budgetString = budgetSS.str();

**return** budgetString;

}

*// Set wages and update costs*

void Budget::setEmployeeWages(double inputWages){

employeeWages = inputWages;

costs = employeeWages + foodCost;

profit = revenue - costs;

}

*// Set food cost and update costs*

void Budget::addFoodCost(double inputCost){

foodCost += inputCost;

costs = employeeWages + foodCost;

profit = revenue - costs;

}

void Budget::addRevenue(double inputRevenue){

revenue += inputRevenue;

profit = revenue - costs;

}

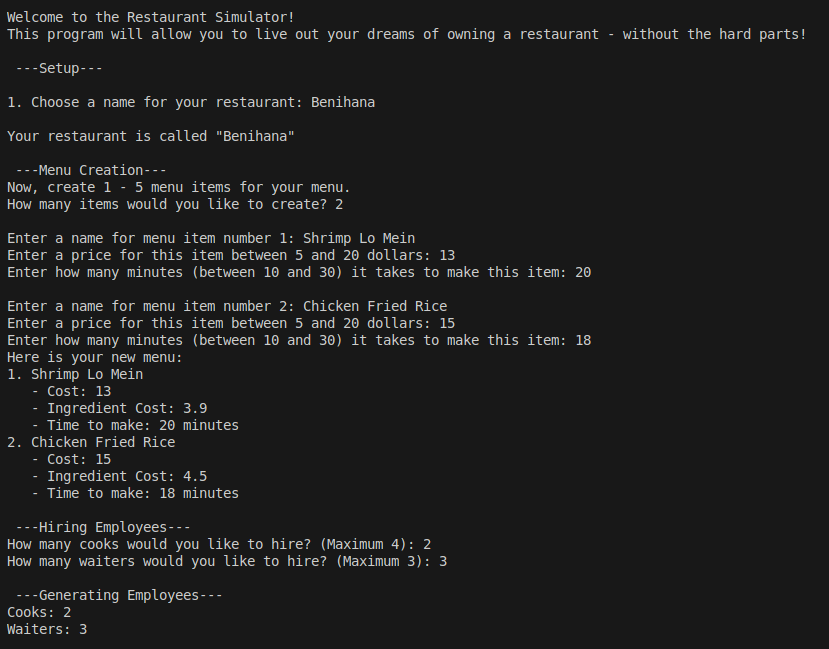
ostream& **operator**<<(ostream& os, Budget budget){

os << setprecision(2) << budget.toString();

**return** os;

}

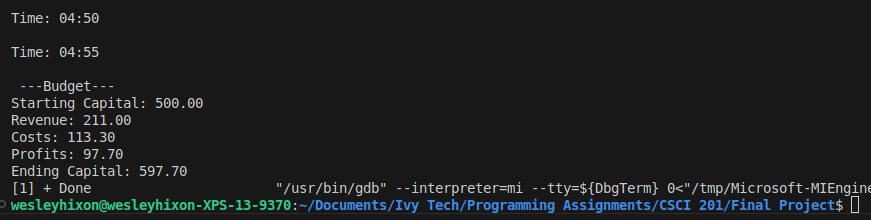
# 5 Three Use Case Screen Shots

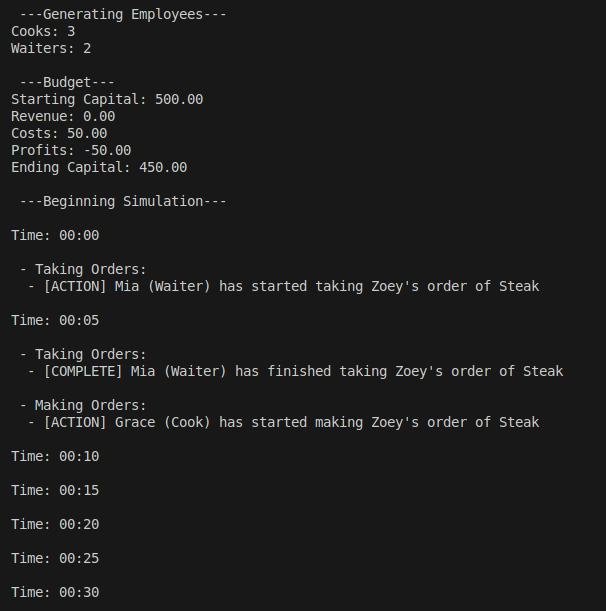
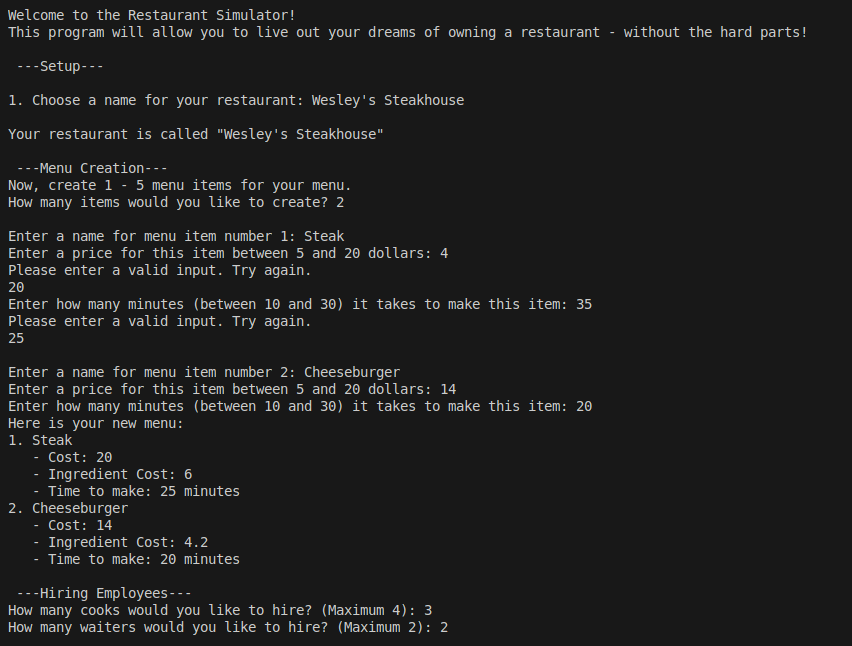


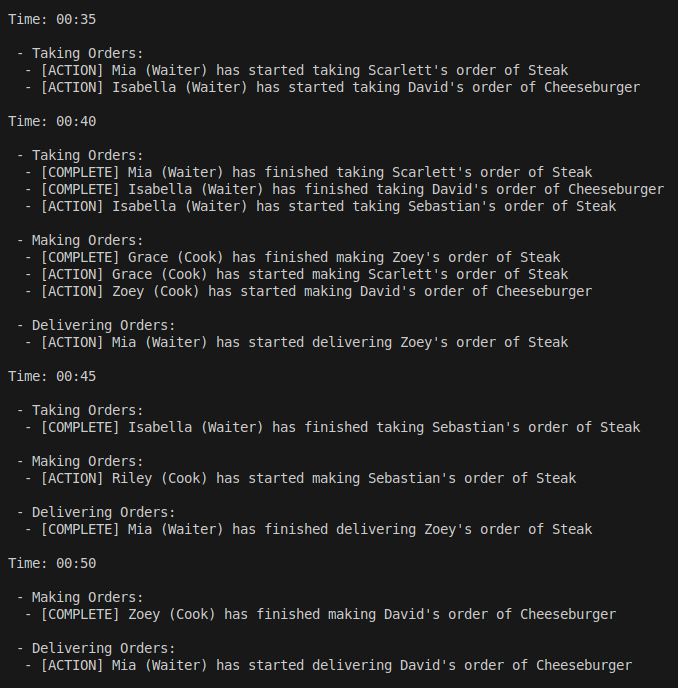




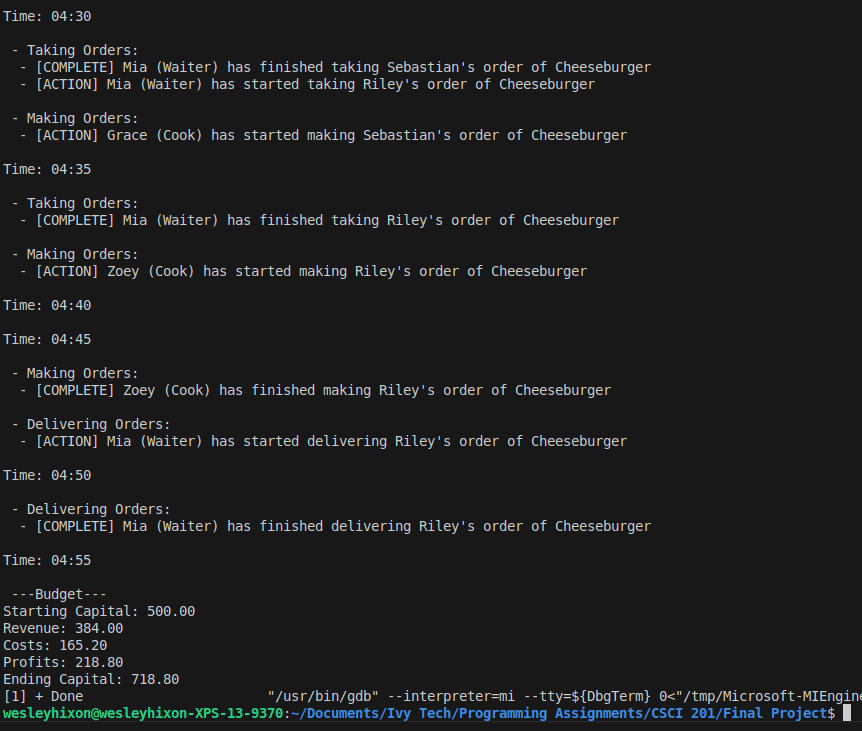
This continues for 5 in game hours

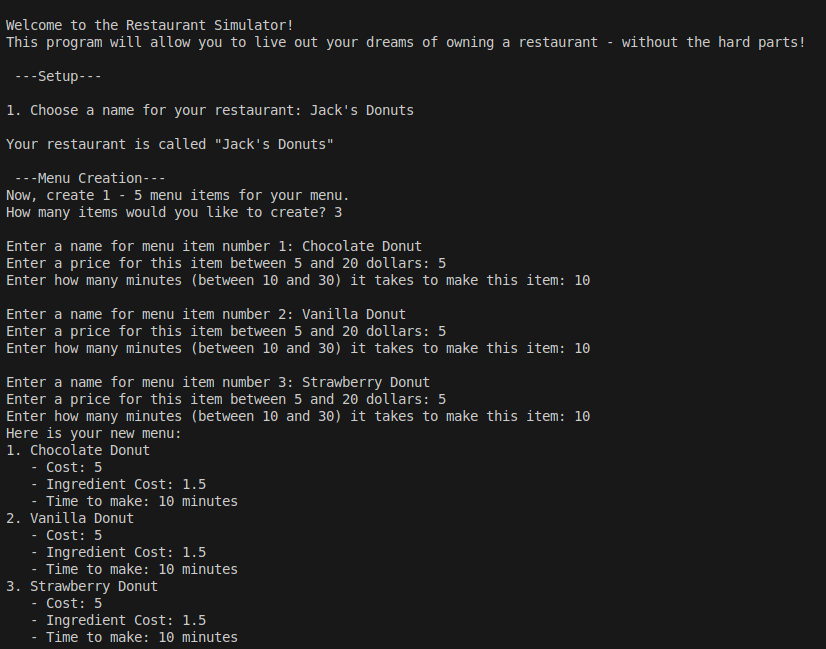


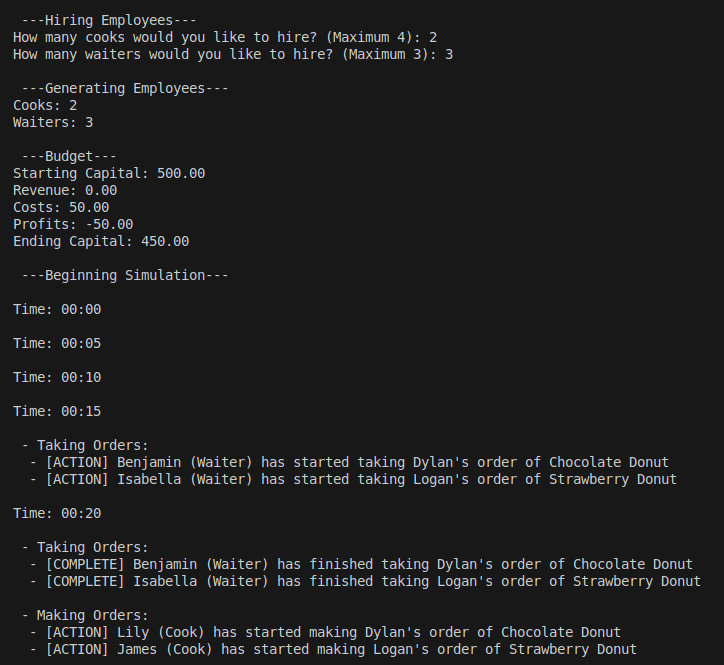




This continues for 5 in game hours







Simulation continues for 5 in game hours

