Mahidol University
International College

Noodle_Rush Final Project

Kongphop	Kayoonvichien	6680081
Panisa	Laohom	6680091
Printitta	Tangpongsirikul	6680152
Theresa	Rujipatanakul	6680211
Wipavee	Buttayothee	6680655

Mahidol University International College

ECGI 211 Advanced Computer Programming

Asst. Prof. Dr. Mingmana Sivaraksa

12 July 2024

Background

Abstract: Noodle Rush is a game based on Zero Hunger and Responsible Consumption and Production, SDG 2 and 12 respectively. A player will run a noodle shop every day for 7 days, creating custom bowls based on customer orders. Earn revenue from sales, factoring in costs for any incorrect ingredients, then calculate and provide accurate change to the customer. Eventually, after a week of collecting revenue, the player will contribute to charity to provide food for people in need. Noodle Rush have an educational goal

Related SDGs:

- SDG 2: Zero Hunger
    - Support this goal by using revenue to help combat hunger by providing food for underprivileged people.
- SDG 12: Responsible Consumption and Production
    - Ensure no food waste by deducting the money score when choosing the wrong ingredients.

Game idea

1. Firstly, the player has to input a name that contains at least one alphabet and does not already exist. Then, they choose whether they want to skip the tutorial or not.
2. A customer's order will be displayed for a limited time, which varies each day, for the player to memorize.
    a. The order is randomly customized by the program.
3. The player must choose ingredients according to the order. If a player answers incorrectly, the money will be deducted by 5 for every mistake they make.
4. After cooking is done, the player must calculate a change to the customer.
    a. The program determines the customer's payment by adding a number randomized from 1 to 100 to the order's price.
    b. If the input change is less than the actual change, the final income will be deducted by half.
    c. If the input change is larger than the actual change, the player will lose money according to the calculation.
    d. If the input is correct, the customer will tip 30% of the price.
5. Steps 2-4 repeat until the queue on that day is empty.
6. At the end of each day, the program will save data to filename.txt

    a. The player can stop playing on any day and the scoreboard will show the amount of days the player has played and the total income they have gained.

7. The game continues for 7 days, and the number of orders will increase while the time will decrease as the day passes by.

Code Explanation

Github repository: https://github.com/Java12361/Noodle_Rush/tree/main

Main.cpp

```cpp
int main(){
    srand(static_cast<unsigned int>(time(0)));

    LinkedList list;
    string current_name;
    system("clear");
    prologue();
    check_and_write_person(list, current_name);

    queue q;
    order bowl;
    int day=1,time=8;
    int i;
    int a,b,c,d;

    cout << "\nDo you want to skip the tutorial?" <<endl;
    if(doYesNo()==2) tutorial();
    system("clear");
```

```cpp
    while(day<=7){
        cout<<"   /)__/)"<<endl;
        cout<<"  (๑^ᴗ^๑)っ"<<endl;
        cout<<"|¯¯ u¯ ¯ |"<<endl;
        cout<<"|  Day "<<day<<"  |"<<endl;
        cout<<"|_________|"<<endl;


        for(i=1; i<=2+day; i++)
        {
            a = rand()%3+1;
            b = rand()%3+1;
            c = rand()%4+1;
            d = rand()%2+1;
            q.enqueue(order(a,b,c,d,time));
        }
        q.do_order();
        pressEnterToContinue();

        day++;
        increaseday(list, current_name);
        time--;
        increasemoney(list, current_name, money);
        list.sort_by_money();
        write_list_to_file(list, "filename.txt");
    }
    endGame();

    system("clear");
    print_current_player(list, current_name);
    print_scoreboard(list, "filename.txt");
}
```

1. Random different seeds every time by `srand(static_cast<unsigned int>(time(0)));` from include <ctime>.

2. Declare classes, including LinkedList, queue, and order classes. Declare variables.

3. The player needs to enter a unique name with at least one character. If the criteria are not met, the player will be prompted to enter a name until it does.

4. Skip the tutorial option by using the function doYesNo() from "game.h" and call tutorial()

from "tutorial.h".

5. The game will play for 7 days, so we use a while loop `while(day<=7)`.

6. We random a, b, c, and d, which indicate the ingredients, with different seeds every time making the output (order) different. This will be used to randomly customize orders in the next step.

7. Orders are constructed using a constructor in “order.h” with randomized ingredients from the previous step and put into the function enqueue() from "queue.h" inside the for loop to create a waiting line. Number of orders increases each day.

8. Call function do_order() from "queue.h" to let the player make orders to serve, do the cashier, and dequeue an order once it is finished.

9. Before the end of the day (before the code in while loop repeats) we call function increaseday() and increasemoney() from "increase_file.h" to write data on filename.txt.

10. Before writing the data in, we sort players by money on “filename.txt” using sort_by_money() from "linkedlist_person.h". Then, update the file by write_list_to_file() function from "file.h".

11. Call function endGame() to display player results after 7 days.

12. Lastly, call function print_current_player() and print_scoreboard() from "file.h" to display the leaderboard

game.h

```
void timer(int sec)
{
    cout << "\n\n\t\t\t\t\t";
    cout << "You have " << sec << " sec to remember!"<<endl;
    for(int i = 0; i < sec; i++)
    {
        cout << "\t\t\t\t\t\t";
        for(int j = 0; j <= i; j++) {
            cout << "[]";
        }
        fflush(stdin);
        sleep(1);
        cout << "\n\t\t\t\t\t\t" << "..." << (i + 1)*100/sec<< "%..." << flush;
        cout << "\033[F"; // ANSI escape code to move cursor up one line
    }
    sleep(1);
    system("clear");
}
```

void timer(int sec): Use a for loop to count down the inserted time in seconds and display the progress.

person.h

```h
#ifndef PERSON_H
#define PERSON_H

#include <iostream>
#include <string>

using namespace std;

class person {
private:
    string name;
    int day;
    double money;

public:
    person(string, int, double);
    person(const person& other);
    string get_name() const;
    int get_day() const;
    double get_money() const;
    void set_day(int new_day);
    void set_money(double new_money);
};
```

Constructors:

- person(string, int, double): Initializes a person with a name, day, and money.
- person(const person& other): Copy constructor to initialize a person from another person object.

Getter Methods:

- string get_name() const: Returns the person's name.
- int get_day() const: Returns the person's day.
- double get_money() const: Returns the person's money.

Setter Methods:

- void set_day(int new_day): Sets a new value for the day.
- void set_money(double new_money): Sets a new value for the money.

node_person.h

```
#ifndef NODE_PERSON_H
#define NODE_PERSON_H

#include "person.h"

using namespace std;

class nodeP {
public:
    person data;
    nodeP* next;

    nodeP(const person& p) : data(p), next(nullptr) {}
};

#endif // NODE_PERSON_H
```

Sets up node with player data to be used in linkedlist_person.h, representing a player in the list

linkedlist_person.h

```
#ifndef LINKEDLIST_PERSON_H
#define LINKEDLIST_PERSON_H

#include "node_person.h"

using namespace std;

typedef nodeP* NodePtr;

class LinkedList {
private:
    NodePtr head;
    NodePtr tail;

public:
    LinkedList();
    ~LinkedList();
    void append(person p);
    bool exists(const string& name) const;
    void print_list() const;
    bool increment_day(const string& name);
    bool increment_money(const string& name, double amount);
    NodePtr get_head() const;
    void sort_by_money();
};
```

The linkedlist_person.h file supports file.h by managing the list of players. It includes functions to add new players, check for existing players, update player progress, and load data from a file.

```
void LinkedList::append(person p) {
    NodePtr newNode = new nodeP(p);
    if (head == nullptr) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        tail = newNode;
    }
}
```

append function use for adding a new player to the end of list and creates a new node and sets its position as the new tail. This function will be called using in create_and_person in file.h.

```
bool LinkedList::exists(const string& name) const {
    NodePtr current = head;
    while (current != nullptr) {
        if (current->data.get_name() == name) {
            return true;
        }
        current = current->next;
    }
    return false;
}
```

exists function checks if a given name already exists in the list. If the entered name is the same as one in the list, it will return true. If the name is unique (not the same as any in the list), it will return false. It will scan through the list using a while loop until it finds nothing left. This function will be called using in uniquename in file.h.

```
bool LinkedList::increment_day(const string& name) {
    NodePtr current = head;
    while (current != nullptr) {
        if (current->data.get_name() == name) {
            current->data.set_day(current->data.get_day() + 1);
            return true;
        }
        current = current->next;
    }
    return false;
}
```

increment_day function function increases the day count for a specified player in the list. It uses a while loop to check each player's name until it finds the correct name and increments their day count. This function will be called using in increaseday in file.h.

```cpp
bool LinkedList::increment_money(const string& name, double amount) {
    NodePtr current = head;
    while (current != nullptr) {
        if (current->data.get_name() == name) {
            current->data.set_money(amount);
            return true;
        }
        current = current->next;
    }
    return false;
}
```

increment_money function updates the money count for a specified player in the list. It uses a while loop to check each player's name until it finds the correct name and replaces their money count. This function will be called in order to increase money in file.h.

```cpp
void LinkedList::sort_by_money() {
    if (!head || !head->next) return;

    NodePtr current = head;
    while (current != nullptr) {
        NodePtr maxNode = current;
        NodePtr next = current->next;

        while (next != nullptr) {
            if (next->data.get_money() > maxNode->data.get_money()) {
                maxNode = next;
            }
            next = next->next;
        }

        if (maxNode != current) {
            person temp = current->data;
            current->data = maxNode->data;
            maxNode->data = temp;
        }

        current = current->next;
    }
}

#endif // LINKEDLIST_PERSON_H
```

sort_by_money function sorts the list of players in descending order based on their money. It uses the selection sort algorithm to set the order. This function will be called using in main.cpp.

file.h

```
#ifndef FILE_H
#define FILE_H

#include <iostream>
#include <fstream>
#include <iomanip>
#include <cctype>
#include <algorithm>
#include <sys/stat.h>
#include "person.h"
#include "linkedlist_person.h"

using namespace std;

bool contains_letter(const string& name);
bool uniquename(const LinkedList& list, const string& name);
string to_uppercase(const string& str);
void check_and_write_person(LinkedList& list, string& current_name);
void write_list_to_file(const LinkedList& list, const string& filename);
void print_current_player(const LinkedList& list, const string& current_name);
void print_scoreboard(const LinkedList& list, const string& filename);
```

The file.h file contains functions for file operations, ensuring that player data is correctly saved and loaded. It checks for the existence of the data file, adds new players, and writes the updated list of players to the file. It requires #include <fstream> for using file-related operations and functions.

```
void check_and_write_person(LinkedList& list, string& current_name) {
    string name;
    int date = 0;
    double money = 0;

    struct stat buffer;
    bool fileExists = (stat("filename.txt", &buffer) == 0);

    if (!fileExists) {
        ofstream MyFile("filename.txt");
        MyFile << left << setw(10) << "Name" << setw(10) << "Day" << right << setw(4) << "Money" << endl;
        MyFile.close();
    }

    ifstream checkFile("filename.txt");
    if (!checkFile.is_open()) {
        cerr << "Error opening file!" << endl;
        return;
    }

    bool isEmpty = checkFile.peek() == ifstream::traits_type::eof();
    string file_name;
    int file_day;
    double file_money;

    if (!isEmpty) {
        checkFile.ignore(numeric_limits<streamsize>::max(), '\n'); // Skip header
        while (checkFile >> file_name >> file_day >> file_money) {
            person temp(to_uppercase(file_name), file_day, file_money);
            list.append(temp);
        }
    }
    checkFile.close();
```

Check_and_write_person function uses the stat function, which requires #include <sys/stat.h>, and sets a variable named buffer to check the existence of a file. If stat does not return 0, it indicates that there is no file named "filename.txt". The function then uses this information to create the file and write the header if the file does not exist. Afterward, it uses a function called Checkfile to check and open the file. Checkfile function reads all players' information from filename.txt, skips the header line, and converts the data into a list for checking names against the entered name.

```
    while (true) {
        cout << "Enter your name: ";
        cin >> name;
        string upper_name = to_uppercase(name);

        if (!contains_letter(name)) {
            cout << "Name must contain at least one letter. Please enter a valid name." << endl;
        } else if (list.exists(upper_name)) {
            cout << "Name '" << upper_name << "' already exists. Please enter a different name." << endl;
            cout << "Existing Names: ";
            list.print_list();
        } else {
            current_name = upper_name;
            break;
        }
    }

    person p(current_name, date, money);
    list.append(p);

    ofstream MyFile("filename.txt", ios::app);
    if (!MyFile.is_open()) {
        cerr << "Error opening file!" << endl;
        return;
    }

    MyFile << left << setw(10) << p.get_name() << " " << setw(9) << p.get_day() << right << setw(5) << p.get_money() << endl;
    MyFile.close();
}
```

Check_and_write_person function (continue) enters a loop to prompt the user for a name, ensuring it is unique and contains at least one letter. The name is then converted to uppercase and added to the list. The function uses ofstream Myfile("filename.txt", ios::app) to append the player's information to filename.txt.

```
void write_list_to_file(const LinkedList& list, const string& filename) {
    ofstream MyFile(filename);
    if (!MyFile.is_open()) {
        cerr << "Error opening file!" << endl;
        return;
    }

    MyFile << left << setw(10) << "Name" << setw(10) << "Day" << right << setw(4) << "Money" << endl;
    NodePtr current = list.get_head();
    while (current != nullptr) {
        MyFile << left << setw(10) << current->data.get_name() << " " << setw(9) << current->data.get_day() << right << setw(5) << current->data.get_money() << endl;
        current = current->next;
    }
    MyFile.close();
}
```

write_list_to_file function is used to save and write the list of players including all players information (name, day, money) to a file.

order.h

```
#include <iostream>
#include <cstdlib>
#include <stdio.h>
#include <stdlib.h>
#include <iomanip>
#include "stdlib.h"
#include "game.h"
#ifndef order_h
#define order_h

using namespace std;

float money=0, tutorialMoney=0;

class order
{
    private:
        string name;
        float price;
        int noodle; //1. rice n (เส้นเล็ก) 2. wide rice n (เส้นใหญ่) 3. egg n (บะหมี่)
        int meat; //1. pork 2. beef 3. seafood
        int soup; //1. dried 2. clear 3. tom yum 4. nam tok
        int veg; //1. yes 2.no
        int time;
    public:
        order(){};
        order(int,int,int,int, int);
        ~order();
        void print();
        bool select_noodle();
        bool select_meat();
        bool select_soup();
        bool select_veg();
        void make();
        void cashier(int=1);
        string get_name();
};
```

order class contains variables and functions that are necessary to create and serve a bowl of noodles

```cpp
order::order(int choice_n, int choice_m, int choice_s, int choice_v, int t)
{
    //set order
    noodle = choice_n;
    meat = choice_m;
    soup = choice_s;
    veg = choice_v;
    time = t;

    //set name
    string n, m, s, v;
    switch(choice_n)
    {
        case 1: n="Rice Noodles"; break;
        case 2: n= "Wide Rice Noodles"; break;
        case 3: n= "Egg Noodles"; break;
    }
    switch(choice_m)
    {
        case 1: m="Pork"; price=40; break;
        case 2: m= "Beef"; price=50; break;
        case 3: m= "Seafood"; price=60; break;
    }
    switch(choice_s)
    {
        case 1: s="Dried "; break;
        case 2: s= "Clear"; break;
        case 3: s= "Tom Yum"; break;
        case 4: s= "Nam Tok"; break;
    }
    switch(choice_v)
    {
        case 1: v=" & Veggies"; break;
        case 2: v= " & No Veggies"; break;
    }

    if (soup==1) name = s + n + " with " + m + v;
    else name = n + " in " + s + " Soup with " + m + v;
}
```

Order constructor is used for setting the ingredients and price in each order as randomized in main.cpp, also combining every ingredient name together to be a whole menu name.

```
bool order::select_meat()
{
    int x;
    do
    {
        //system("clear");
        fflush(stdin);
        cout<<"Select \033[31mMeat\033[0m Type?"<<endl;
        cout<<" 1. Pork"<<endl<<" 2. Beef"<<endl<<" 3. Seafood"<<endl;
        cin>>x;
        if (cin.fail()){
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout<<"Please insert number between 1-3 "<<endl;
        }
        else
        if(x<1 || x>3) cout<<"Please insert number between 1-3 "<<endl;
        else break;
    } while (x<1 || x>3);

    if (x!=meat)
    {
        cout<<"You chose the wrong one! -5 "<<endl;
        price -=5;
        return false;
    }
    else return true;
}
```

This kind of function is used for making the player select the right ingredient in each order as it is set in an order constructor. There are 4 similar functions including: select_noodle, select_meat, select_soup and select_veg.

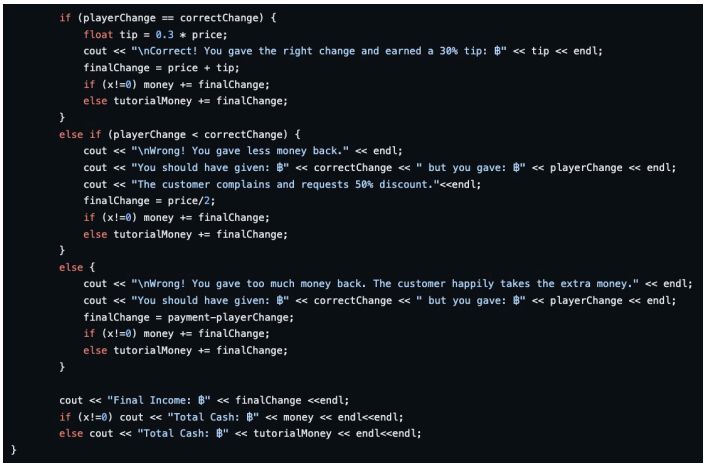
```
void order::make(){
  bool noodle, meat, soup, veg;
  pressEnterToContinue();
  cout<<"\t\t\t_____________________________________________________________"<<endl;
  cout<<"\n\t\t\t  "<<get_name()<<endl;
  cout<<"\t\t\t_____________________________________________________________"<<endl;
  if (time!=0) timer(time);
  noodle = select_noodle();
  meat = select_meat();
  soup = select_soup();
  veg = select_veg();
  cout<<endl<<name<<" is ready to serve!"<<endl;
  pressEnterToContinue();
  cashier();
}
```

Make function: Print the customer's order for a specific time which will be different for each day of selling using timer(int) function in game.h. Call all the select_(ingredient) functions to let the player make a bowl of noodles and call the cashier function (explained on the next page).

```
void order::cashier(int x) //if x==0 -> tutorial
{
        float payment = price + ((rand() % 100 + 1)); // Random payment larger than menu price
        float correctChange = payment - price;
        float playerChange;
        float finalChange;

        cout<<"____________________"<<endl<<endl;
        cout<<"      Cashier"<<endl;
        cout<<"____________________"<<endl<<endl;


        cout << "Menu Price: ฿" << price << endl;
        cout << "Customer pays: ฿" << payment << endl;
        cout << "Calculate the change to give back to the customer: ฿";
        fflush(stdin);
        cin >> playerChange;
        if (cin.fail()){
            cin.clear();
            cin.ignore();
            playerChange=0;
        }
```



```
        if (playerChange == correctChange) {
            float tip = 0.3 * price;
            cout << "\nCorrect! You gave the right change and earned a 30% tip: ฿" << tip << endl;
            finalChange = price + tip;
            if (x!=0) money += finalChange;
            else tutorialMoney += finalChange;
        }
        else if (playerChange < correctChange) {
            cout << "\nWrong! You gave less money back." << endl;
            cout << "You should have given: ฿" << correctChange << " but you gave: ฿" << playerChange << endl;
            cout << "The customer complains and requests 50% discount."<<endl;
            finalChange = price/2;
            if (x!=0) money += finalChange;
            else tutorialMoney += finalChange;
        }
        else {
            cout << "\nWrong! You gave too much money back. The customer happily takes the extra money." << endl;
            cout << "You should have given: ฿" << correctChange << " but you gave: ฿" << playerChange << endl;
            finalChange = payment-playerChange;
            if (x!=0) money += finalChange;
            else tutorialMoney += finalChange;
        }

        cout << "Final Income: ฿" << finalChange <<endl;
        if (x!=0) cout << "Total Cash: ฿" << money << endl<<endl;
        else cout << "Total Cash: ฿" << tutorialMoney << endl<<endl;
}
```

cashier function: Initialize Payment and Correct Change by generating a random payment amount that is larger than the menu price. Then the player will input the change they would give back. Check Player's Change: If the player's change is correct add a 30% tip to the final change. If the player's change is less than the correct amount Apply a 50% discount on the price. If the player's change is more than the correct amount, calculate the extra money given back to the customer. Lastly updates the total cash based on whether it's a tutorial mode (x != 0) or an actual transaction (x == 0).

node.h

```
#ifndef node_h
#define node_h
#include "order.h"

class node
{
private:
    order bowl;
    node *next;

public:
   node(order);
   void print();
   void make_bowl();
   ~node();
    void set_next(node* t) {next=t;}
    node * get_next() {return next;}

};
```

Sets up a node containing ‘order’ called bowl to be used in queue.h. The bowl represents a customer’s order waiting to be made.

```
void node::make_bowl(){
    bowl.make();
}
```

make_bowl function is simply used to access member function make() in the class order. This will be called in do_order() in queue.h.

queue.h

```
#ifndef queue_h
#define queue_h
#include "node.h"
#include "order.h"
#include "game.h"
typedef node* nodePtr;


class queue {
        nodePtr headPtr,tailPtr;
        int size;
public:
    void enqueue(order);
    void dequeue();
    void print_order();
    void do_order();
    int get_size();
    queue();
    ~queue();
};
```

The queue.h file supports the main game by creating a waiting line of orders as a queue. It includes functions to add new orders, make orders, and delete finished orders.

```
void queue::enqueue(order x){
    nodePtr new_node= new node(x);
    if(new_node){
        if(headPtr!=NULL) tailPtr->set_next(new_node); //if(size==0)
        else headPtr=new_node;

        tailPtr=new_node;
        size++;
    }
}

void queue::dequeue(){
  if(headPtr!=NULL){
     nodePtr t=headPtr;
     if(size==1)
     {
        headPtr=NULL;
        tailPtr=NULL;
     }
     else headPtr=headPtr->get_next();

     size--;

     cout<<"You've served ";
     t->print();

     delete t;
  }
  else cout<<"Empty queue"<<endl;
}
```

This follows FIFO or first-in-first-out principle. Enqueue function is used to add new order to the queue at the tail and increase the size of the queue. Dequeue function is used to delete the finished order (at the head) from the queue.

```
void queue::do_order(){
  nodePtr t;
  int n=size;
  for (int i=0; i<n; i++)
  {
    t=headPtr;
    t->make_bowl();
    dequeue();
  }
  cout<<"\nIt's the end of the day"<<endl;
}
```

do_order function is used to make all the orders in a day. In a for loop, it calls the function make_bowl() which is declared in the node to make a bowl of noodles and do the cashier, then it calls the dequeue function. The for loop will loop until it matches the size of the queue that is until the queue is empty.

Work Distribution

<table>
<tr><td>Name</td><td colspan="2">Code</td></tr>
<tr><td>Kongphop (Java) : 6680081</td><td>Sub-game functions & Debug</td><td>game.h</td></tr>
<tr><td>Panisa (Pla) : 6680091</td><td rowspan="2">Save game functions</td><td rowspan="2">file.h, linkedlist_person.h,
node_person.h</td></tr>
<tr><td>Printitta (Mind) : 6680152</td></tr>
<tr><td>Theresa (Tea-Tree) : 6680211</td><td rowspan="2">Main game functions</td><td rowspan="2">order.h, node.h, queue.h</td></tr>
<tr><td>Wipavee (Pang) : 6680655</td></tr>
</table>

Requirements

1. Using 6 classes as these following files:
    1.1. person.h
    1.2. node_person.h
    1.3. linkedlist_person.h
    1.4. order.h
    1.5. node.h
    1.6. queue.h
2. Sorted algorithm is implemented in the sort_by_money function of linkedlist_person.h file, using selection sort.
3. Integrating person.h in linkedlist_person.h and order.h.in queue.

Limitation

Even after all the hardship that we put into this project, there are still some limitations, including:

- Our game is a play-through game, which means the player cannot choose to exit the game midway.
- Even though our program saves data, including the player's name and number of days they have played, it is not possible to resume the game from where the player left off.

Presentation Link

https://www.canva.com/design/DAGJbXYBeuI/xtzVNC6lzKUf9LhRkwrRKg/view?utm_content=DAGJbXYBeuI&utm_campaign=designshare&utm_medium=link&utm_source=editor