15.12.2022
Programming Language 1

Hotel Reservation

Final task for programming language course

Prabesh Sharma

Introduction

Welcome to my attempt at the final programming task. In my hotel reservation app, a user can select between 4 different features, giving the user comprehensive control over booking a hotel room.

A user can book, check, and cancel their reservations. A feature to quit the program is also available in any point of the time. Additionally, instructions on how the parts of the program work is stated to help user navigate the program.

I am hoping for a full 5 grades from this assignment as I have given very high attention to detail trying to cover bugs and quality of life features for the user.

Features in detail

- *[Reserve a room]*
  - *Selects a room from a list of available ones and books it for the user, while asking them name, preferred room type, and the period of stay*
  - *Selects the amount of discount randomly (10%, 20% or 30%) and calculates the cost accordingly.*
  - *Finally displays all information regarding the booking to the user.*
- *[Your reservation]*
  - *Asks for either room id given to the user after booking a room, or user's name to again display all the rooms booked by the user with the name.*
- *[Cancel reservation]*
  - *As the name suggests, cancels the reservations of user or of specific room ID.*
  - *If multiple reservations are found from a same user asks which one from the list order to cancel.*
- *[Quit application]*
  - *Simply quits the application.*

Source code

Hotel - task.cpp

```
#include < iostream >
#include < iomanip >
#include < string >

#include "Constants.h"
#include "Utility.h"


struct reservation
{
  std:: string r_name = std:: string();

  bool is_set = false;
  bool is_booked = false;
  bool is_double = false;

  int r_id = -1;
  int nights = 0;

  double current_cost = 0.0;
};


int fetch_type()
{
  std:: cout << "\nSelect room type!\n";
  std:: cout << "1. [ Single room ]\n";
  std:: cout << "2. [ Double room ]\n";
  std:: cout << "3. [ Go to start ]\n\n";

  return validated_input<int>(1, 3);
}

int fetch_room(const reservation * reserves, const bool want_doubles)
  {
    constexpr int index = 0;

for (int i = 0; i < MAX_ROOMS; i++)
{
  if (want_doubles) {
    if (reserves[i].is_double && !reserves[i].is_booked) {
      return i;
    }
  }
  else {
    if (!reserves[i].is_double && !reserves[i].is_booked) {
      return i;
    }
  }
}

return index;
}

void fetch_prices(reservation & room)
{
  const int cost = room.is_double ? PRICE_D * room.nights : PRICE_S *
room.nights;
```

```
  const int discount = discount_amount();

  std:: cout << "Accumulated cost: [" << cost << " EUR]\n";
  if (discount > 0)
    std:: cout << "Price discount: [-" << static_cast<double>(cost) *
discount * 0.01 << " EUR. (" << discount << "%)]\n\n";
  else
    std:: cout << "Price discount: [No discounts!]\n\n";

  room.current_cost = cost - (static_cast<double>(cost) * discount * 0.01);
}

#pragma region Logs

void log_choices()
{
  std:: cout << "What's on your mind?\n";
  std:: cout << "1. [ Reserve a room ]\n";
  std:: cout << "2. [ Your reservation ]\n";
  std:: cout << "3. [ Cancel reservation ]\n";
  std:: cout << "4. [ Quit application ]\n\n";
}


void log_reserved(reservation & current_room, const bool foreign = false)
  {
    const char* type = current_room.is_double ? "Double" : "Single";

if (!foreign) std:: cout << "\nYour room has been booked " <<
current_room.r_name << "\n";

std:: cout << "--------------------------------\n";
std:: cout << "Name: [" << current_room.r_name << "]\n";
std:: cout << "Room type: [" << type << "]\n";
std:: cout << "Nights: [" << current_room.nights << "]\n";
std:: cout << "ID: [" << current_room.r_id << "]\n\n";


if (!foreign) fetch_prices(current_room);
std:: cout << "Total cost: [" << current_room.current_cost << " EUR]\n";
std:: cout << "--------------------------------\n\n";
}


void log_all(const reservation * rooms)
{
  std:: cout << "\n";
  bool found = false;

  for (int i = 0; i < MAX_ROOMS; i++)
  {
    if (!rooms[i].is_booked) continue;

    std:: string teller = rooms[i].r_name.empty() ? "Unknown" :
rooms[i].r_name;

    if (rooms[i].is_double) {
      std:: cout << i << ". [ Double ] Booked by: " << teller << ". [" <<
rooms[i].r_id << "]\n";
      found = true;
    }
    else {
```

```
      std:: cout << i << ". [ Single ] Booked by: " << teller << ". [" <<
rooms[i].r_id << "]\n";
      found = true;
    }
  }

  const char* teller = found ? "\n" : "[ No booked rooms found! ]\n\n";
  std:: cout << teller;
}

#pragma endregion


void book_room(reservation & room, int & room_am)
{
  if (room.is_booked || room_am == 0) {
    std:: cout << "\n[ Current room type is unavailable! ]\n\n";
    return;
  }

  std:: cout << "[ Specify the number of nights to stay ]\n\n";
  validated_input(room.nights, 1, 365);

  room.is_booked = true;
  room.is_set = true;
  room.r_id = reserve_id();

  std:: cout << "\n[ Specify the reservee's name ]\n\n";
  validated_input(room.r_name);

  log_reserved(room);

  room_am--;
}


void remove_room(reservation & room)
{
  std:: cout << "\nAre you sure you want to cancel this reservation " <<
room.r_name << "?\n";
  std:: cout << "1. [ Yes! Cancel reservation! ]\n";
  std:: cout << "2. [ No! Take me back! ]\n\n";

  const int choice = validated_input<int>(1, 2);

  switch (choice) {
    case 1:
      std:: cout << "\n[ Reservation for " << room.r_name << " with ID " <<
room.r_id << " is now canceled! ]\n\n";
      room.is_booked = false;
      room.r_name = std:: string();
      room.r_id = -1;
      room.nights = 0;
      room.current_cost = 0.0;
      break;
    case 2:
      std:: cout << "\n";
      return;
    default:
      break;
  }
}
```

```
void release_room(const reservation * reserves, int & singles, int & doubles)
{
  const int ind_d = fetch_room(reserves, true);
  const int ind_s = fetch_room(reserves, false);

  if (!reserves[ind_d].is_booked && reserves[ind_d].is_set) {
    doubles++;
    return;
  }
  if (!reserves[ind_s].is_booked && reserves[ind_s].is_set) {
    singles++;
    return;
  }
}


void manage_reservation(reservation * reserves, const bool cancellation =
false)
  {
    int found_index[MAX_ROOMS];

int id = 0;
int amount = 0;
bool is_str;

const char* teller = cancellation ? "cancel" : "check";

std:: cout << "\n[ To " << teller << ", enter Room ID or Reservee's name
]\n\n";
const std:: string r_info = validated_input(true);

try {
  id = std:: stoi(r_info);
  is_str = false;
}
catch (const std:: exception&)
{
  is_str = true;
}

for (int i = 0; i < MAX_ROOMS; i++)
{
  if (!reserves[i].is_booked) continue;

  if (is_str) {
    if (r_info == reserves[i].r_name) {
      found_index[amount] = i;
      amount++;
    }
  }
  else {
    if (id == reserves[i].r_id) {
      found_index[amount] = i;
      amount++;
    }
  }
}

if (amount == 0) {
  std:: cout << "\n[ " << r_info << " ] not found in the system!\n\n";
```

```
  return;
}

if (!is_str && amount > 1) {
  reservation & reservee = reserves[found_index[0]];
  reservee.r_id = id + 1;

  std:: cout << "\n[ Something went wrong! Reservee " << reservee.r_name <<
"'s id was renewed to " << reservee.r_id << " ]\n\n";


  return;
}

std:: cout << "\nRoom booked by [" << r_info << "]\n";
for (int i = 0; i < amount; i++)
{
  log_reserved(reserves[found_index[i]], true);
}

if (cancellation) {
  int choice = 0;

  if (amount > 1) {
    std:: cout << "\n[ " << amount << " rooms found in your reservations!
Please enter the corresponding number for the list above! ]\n\n";
    validated_input<int>(choice, 1, amount);


    choice -= 1;
  }

  remove_room(reserves[found_index[choice]]);
}
}


void controller(reservation * reserves, int & single_am, int & double_am,
bool & wants_to_quit)
{
  printf("AVAILABLE ROOMS\nsingle: %i, double: %i\n\n", single_am,
double_am);
  log_choices();

  const int current_choice = validated_input<int>(1, 5);

  switch (current_choice) {
    case 1:

      if (single_am + double_am == 0) {
        std:: cout << "\n[ Apologies! All rooms are currently booked! ]\n\n";
        break;
      }

      switch (fetch_type()) {
        case 1:
          std:: cout << "\n";

          book_room(reserves[fetch_room(reserves, false)], single_am);
          break;
        case 2:
          std:: cout << "\n";
```

```
          book_room(reserves[fetch_room(reserves, true)], double_am);
          break;
        case 3:

          return;
        default:
          break;
      }
      break;
    case 2:
      manage_reservation(reserves);
      break;
    case 3:
      manage_reservation(reserves, true);
      release_room(reserves, single_am, double_am);
      break;
    case 4:

      std:: cout << "\nAre you sure you want to quit?\n";
      std:: cout << "1. [ Yes! Get me out! ]\n";
      std:: cout << "2. [ No! It was a mistake! ]\n\n";

      if (validated_input<int>(1, 2) > 1) {
        std:: cout << "\n";
        break;
      }
      else {
        wants_to_quit = true;
        std:: cout << "\n";
        return;
      }

      break;
    case 5:
      log_all(reserves);
      break;
    default:
      break;
  }

}

int main()
{
  bool wants_to_quit = false;
  const int r_division = room_division() * 2;
  int single_room = r_division / 2, double_room = r_division / 2;

  reservation user_reservation[MAX_ROOMS];
  for (int i = 0; i < double_room; i++)
  {
    user_reservation[single_room + i].is_double = true;
  }

  do {
    controller(user_reservation, single_room, double_room, wants_to_quit);
  }
  while (!wants_to_quit);

  return 0;
}
```

Utility.h

```
#pragma once
#include < cstdlib >
#include < ctime >

  inline int reserve_id()
{
  srand(static_cast<int>(time(nullptr)));
  return rand() % 90000 + 10000;
}

inline int discount_amount()
{
  srand(static_cast<int>(time(nullptr)));
  return rand() % 3 * 10;
}

inline int room_division()
{
  srand((int)(time(nullptr)));
  return rand() % MAX_ROOMS / 4 + MIN_ROOMS;
}

#pragma region Input Validation
template < typename T >
  void validated_input(T & input, int min = 0, int max = 0)
{
  std:: cin >> std:: setw(1) >> input;
  const bool comparable = !(min == max);

  bool in_between = input >= min && input <= max;

  while (!std:: cin.good() || (!in_between && comparable))
  {
    if (std:: cin.good())
    {
      std:: cout << "\n[ Only numbers from " << min << " : " << max << " are
valid! ]\n\n";

      std:: cin >> std:: setw(1) >> input;
      in_between = input >= min && input <= max;
    }
        else
    {
      std:: cin.clear();
      std:: cin.ignore(INT_MAX, '\n');

      std:: cout << "\n[ Only numbers are allowed! ]\n\n";

      std:: cin >> std:: setw(1) >> input;
      in_between = input >= min && input <= max;
    }
  }
```

```
  std:: cin.clear();
  std:: cin.ignore(INT_MAX, '\n');
}


template < typename T >
  T validated_input(const int min = 0, const int max = 0)
    {
      T input;
validated_input<T>(input, min, max);

return input;
}

inline void validated_input(std:: string & input, const bool num_allowed =
false)
  {
    bool success = false;
std:: getline(std:: cin, input);

if (num_allowed) return;

while (!success) {
  try {
    std:: stoi(input);
    std:: cout << "\n[ Name cannot start with a number ]\n\n";
    std:: getline(std:: cin, input);
  }
  catch (const std:: exception&)
  {
    if (input.empty()) {
      std:: cout << "\n[ Please enter your name before you continue ]\n\n";
      std:: cin.clear();
      std:: getline(std:: cin, input);
      continue;
    }

    success = true;
  }
}
}

inline std:: string validated_input(const bool num_allowed = false)
  {
    std:: string input;
validated_input(input, num_allowed);

return input;
}

#pragma endregion

Constants.h
#pragma once
constexpr int MAX_ROOMS = 80;
constexpr int MIN_ROOMS = 20;

constexpr int PRICE_S = 100;
constexpr int PRICE_D = 150;
```