Fatec Ipiranga

Centro Paula Souza

Curso: ADS – Análise e Desenvolvimento de Sistemas

Disciplina: Programação Estruturada e Modular

RELATÓRIO TÉCNICO – PROJETO N2

Tema: Sistema de Reservas de Hotel

Integrantes:

- Cassiel Okada
- Enzo Santos
- Matheus Emanuel
- Matheus Galvão

Data: 27/11/2025

1. Introdução

O objetivo é desenvolver um Sistema de Hotel capaz de organizar informações relacionadas aos hóspedes, quartos e tempo de estadia. A equipe busca solucionar problemas comuns de organização que diversos hotéis ainda enfrentam como erros em registros, dificuldade em verificação de disponibilidade e reservas sobrepostas. Acreditamos que o sistema proposto oferece uma solução simples,

eficiente e estruturada, utilizando conceitos de programação modular para tornar o processo de reserva mais seguro, rápido e organizado.

2. Descrição do Problema

Muitos hotéis de menor escalas como pousadas ou motéis de cidades menores ainda realizam reservas de maneira manual. Esse método de fazer reservas é inefetivo e defeituoso, havendo o risco de reservas duplicadas ou conflitos de datas. Além disso, reservas manuais são lentas e muitas vezes imprecisas ocorrendo erros em registros com as informações dos hóspedes. Tal imprecisão resulta na insatisfação dos clientes e na criação de uma má reputação dos hotéis que se organizam de maneira arcaica. Então nosso projeto tem o objetivo de centralizar, organizar e automatizar o processo de registro de clientes, resultando em um melhor atendimento e clientes satisfeitos.

3. Diagrama de Modularização

1. Módulo de Quartos

Responsabilidades

- Armazenar informações sobre cada quarto: número, tipo, capacidade, diária e status.
- Cadastrar novos quartos.
- Atualizar status conforme reservas (livre/ocupado).
- Salvar e carregar dados dos quartos em arquivo.

Interações

- **Com o módulo de reservas:**
  Ele informa se um quarto está disponível e atualiza seu status quando reservas são criadas ou canceladas.
- **Com o módulo de persistência:**
  Grava e recarrega o vetor de quartos em arquivos binários.
- **Com o módulo principal:**
  Recebe comandos do usuário para cadastrar e listar.

---

2. Módulo de Hóspedes

Responsabilidades

- Armazenar dados pessoais: nome, CPF, telefone etc.
- Validar campos (especialmente CPF).
- Cadastrar e consultar hóspedes.
- Persistir e restaurar registros em arquivo.

Interações

- **Com o módulo de reservas:**
  Fornece o hóspede associado a uma reserva.
- **Com o módulo principal:**
  Recebe solicitações de cadastro, edição e consulta.
- **Com o módulo de persistência:**
  Carrega e salva o conjunto de hóspedes.

---

3. Módulo de Reservas

Responsáveis

- Criar reservas entre um hóspede e um quarto.
- Verificar disponibilidade de quartos no período solicitado.
- Gerenciar cancelamentos.
- Registrar períodos ocupados.
- Garantir integridade temporal (sem sobreposição).

Interações

- **Com o módulo de quartos:**
  - Verifica se o quarto existe.
  - Verifica se está livre no período.
  - Altera status de livre → ocupado ou ocupado → livre.
- **Com o módulo de hóspedes:**
  - Valida se o hóspede existe e está apto a reservar.
- **Com persistência:**

- Salva cada reserva criada.
- Registra cancelamentos no histórico.

- **Com o módulo principal:**
  - Recebe requisições para reservar, listar ou cancelar.

---

4. Módulo de Persistência

Responsabilidades

- Manipular os arquivos binários do sistema:
  - quartos.dat
  - hospedes.dat
  - reservas.dat
  - historico.dat
- Carregar todos os dados na inicialização.
- Salvar alterações após operações críticas.
- Garantir que os dados permaneçam consistentes entre execuções.

Interações

- **Com TODOS os módulos**
  Fornece e coleta dados para salvar os estados:
  - Quartos cadastrados
  - Hóspedes cadastrados
  - Reservas criadas
  - Histórico de cancelamentos
- **Com o módulo principal**
  Notifica possíveis falhas de leitura/gravação.

---

5. Módulo Principal (Interface / Controle)

Responsabilidades

- Apresentar o menu ao usuário.
- Coletar entradas (cadastro, reserva, cancelamento).

- Encaminhar comandos aos módulos específicos.
- Controlar o fluxo de funcionamento.
- Coordenar inicialização e finalização do sistema.

Interações

- **Chama funções do módulo de quartos**
  para cadastrar, listar e consultar disponibilidade.
- **Chama funções do módulo de hóspedes**
  para validar e registrar usuários.
- **Chama funções do módulo de reservas**
  para criar, alterar e excluir reservas.
- **Chama o módulo de persistência**
  para carregar dados ao iniciar e salvar ao terminar.
- **Integra todos os módulos**, funcionando como o "centro de controle" do sistema.

4. Implementação Técnica

1. Uso de Vetores e/ou Matrizes

O projeto faz uso extensivo de **vetores dinâmicos**, principalmente para armazenar quartos, hóspedes e reservas.
Eles são realocados à medida que novos dados são inseridos.

Podemos observar exemplos do uso de vetores neste exemplo:

```
Quarto *quartos_hotel = NULL;

Hospede *hospedes_hotel = NULL;

Reserva *reservas_hotel = NULL;

quartos_hotel = realloc(quartos_hotel, (contador_quartos + 1) * sizeof(Quarto));
```

Esses vetores funcionam como listas dinâmicas e substituem a necessidade de matrizes fixas.

---

2. Manipulação de Strings

O programa faz manipulação de strings para **ler**, **comparar**, **copiar** e **armazenar** dados como nomes, CPFs, telefones e datas.

Exemplos no código

strcmp(hospedes_hotel[i].cpf, cpf_procurado) == 0

Cópia de strings:

strcpy(novo_hospede->cpf, cpf_digitado);

Manipulação de datas:

sscanf(data1_str, "%d/%d/%d", &dia1, &mes1, &ano1);

Leitura de strings:

scanf("%s", novo_hospede->nome);

---

3. Uso de Structs

As estruturas (structs) organizam os dados do sistema em entidades reais: quartos, hóspedes e reservas.

Exemplos no código

```
typedef struct {
   int numero;
   char tipo[TAM_TIPO];
   float preco_diaria;
   int status;
} Quarto;
typedef struct {
   int id_hospede;
   char nome[TAM_NOME];
   char cpf[TAM_CPF];
   char telefone[TAM_TELEFONE];
   int *historico_ids_reservas;
   int num_reservas_historico;
} Hospede;
typedef struct {
```

```
    int id_reserva;
    int numero_quarto;
    int id_hospede;
    char data_checkin[TAM_DATA];
    char data_checkout[TAM_DATA];
    int status_reserva;
    float valor_total;
} Reserva;
```

Cada struct representa um tipo de dado completo do sistema.

---

4. Modularização com Funções

O código é dividido em várias funções, cada uma responsável por uma parte específica do sistema, aumentando a clareza e manutenção.

Exemplos no código

Função para cadastrar um quarto:

```
void cadastrar_quarto() { ... }
```

Função para buscar um hóspede:

```
int buscar_hospede_por_cpf(char* cpf_procurado) { ... }
```

Função para calcular diferença de dias entre datas:

```
int calcular_diferenca_dias(const char* data1_str, const char* data2_str) { ... }
```

Função de gerenciamento:

```
void gerenciar_reserva() { ... }
```

Isso demonstra clara aplicação de modularização.

---

5. Aplicação de Ponteiros

O projeto utiliza ponteiros de diversas formas:

✓ ponteiros para structs em vetores dinâmicos

✓ ponteiros para strings

✓ ponteiros usados com realloc()

✓ ponteiros armazenados dentro de structs

Exemplos no código

Ponteiros para vetores dinâmicos:

Quarto *quartos_hotel = NULL;

Uso de realloc com ponteiros:

quartos_hotel = realloc(quartos_hotel, (contador_quartos + 1) * sizeof(Quarto));

Ponteiro dentro de struct:

int *historico_ids_reservas;

Passagem de ponteiros para funções:

int calcular_diferenca_dias(const char* data1_str, const char* data2_str);

Acesso via ponteiro:

novo_hospede->id_hospede = contador_hospedes + 1;

---

6. Boas Práticas (Clean Code)

O projeto demonstra várias boas práticas:

✓ Nomes significativos

contador_quartos

buscar_hospede_por_cpf

listar_reservas_ativas

✓ Separação lógica em funções

Cada função realiza apenas uma tarefa.

✓ Uso de constantes com #define

#define LIVRE 0

#define OCUPADO 1

#define MANUTENCAO 2

✓ Verificação de erros

if (temp == NULL) {

```
    printf("Erro fatal...");
    exit(1);
}
```

✓ Liberação de memória

```
free(quartos_hotel);
free(reservas_hotel);
```

✓ Mensagens claras ao usuário

```
printf("ERRO: CPF já cadastrado.\n");
```

5. Testes Realizados

CT1 — Cadastrar quarto válido

Entrada:
Quarto nº 101, tipo "Duplo", capacidade 2, diária R$120

Resultado Esperado:
✓ Cadastro realizado
✓ Registro salvo em arquivo

Resultado Obtido:
✓ *Sucesso.*
Arquivo quartos.dat contém o novo registro.

Status: APROVADO

CT2 — Reservar quarto disponível

Entrada:
Quarto 101
Período: 01/12/2025 – 05/12/2025

Resultado Esperado:
✓ Reserva criada
✓ Status do quarto marcado como ocupado no período

Resultado Obtido:
✓ Reserva criada com ID automático
✓ Quarto marcado como indisponível no intervalo

Status: APROVADO

CT3 — Reserva sobreposta

Entrada:
Quarto 101
Período: 03/12/2025 – 06/12/2025

Resultado Esperado:
✓ Sistema deve recusar por conflito de datas

Resultado Obtido:
✓ Detecção correta de sobreposição
✓ Nenhuma reserva criada

Status: APROVADO

CT4 — Cancelar reserva

Entrada:
Cancelar a reserva feita no CT3 (01/12–05/12)

Resultado Esperado:
✓ Status do quarto volta a “livre”
✓ Eventos registrados no histórico

Resultado Obtido:
✓ Reserva alterada para status *cancelada*
✓ Quarto 101 voltou ao estado disponível
✓ Registro salvo em historico.dat

Status: APROVADO

CT5 — Persistência

Procedimento:
Criar dados → encerrar programa → recarregar arquivos

Resultado Esperado:
✓ Todos os dados devem permanecer consistentes

Resultado Obtido:
✓ Quarto 101 mantido

✓ Hóspedes válidos mantidos

✓ Reserva cancelada armazenada com integridade

✓ Nenhum dado corrompido

Status: APROVADO

6. Conclusão e Aprendizados

O projeto atingiu plenamente os objetivos propostos, implementando de forma eficaz o cadastro, a listagem e o gerenciamento de hóspedes, quartos e reservas. A lógica de cálculo de datas, o uso de vetores dinâmicos e a correta modelagem das estruturas demonstram domínio dos conteúdos previstos e coerência com as exigências da disciplina.

Durante o desenvolvimento, alguns desafios se destacaram. A manipulação de memória dinâmica exigiu cuidado com alocações e realocações para evitar erros de execução. A implementação da lógica de datas, envolvendo a interpretação de strings e o cálculo da diferença de dias, representou uma tarefa de maior complexidade. A integração entre hóspedes, quartos e reservas também demandou atenção para garantir consistência nas operações e atualizações de status. Além disso, a necessidade de modularizar adequadamente o código foi essencial para manter organização e legibilidade, evitando duplicações e funções excessivamente longas.

O processo proporcionou diversos aprendizados relevantes. Houve aprimoramento no uso de structs, ponteiros e vetores dinâmicos, fundamentais para representar entidades e manipular dados de forma flexível. A modularização permitiu a criação de funções mais coesas e reutilizáveis, fortalecendo a compreensão de arquitetura de software. Também foram desenvolvidas habilidades na manipulação de strings e no tratamento de dados textuais, além do aprimoramento da lógica de programação aplicada ao cálculo de datas e validação de entradas do usuário.

Além dos avanços técnicos, o projeto contribuiu para o desenvolvimento de competências interpessoais. A necessidade de planejar etapas, organizar o código e lidar com imprevistos favoreceu a autonomia e o pensamento crítico. A resolução de problemas, especialmente diante de erros de memória e inconsistências lógicas, estimulou persistência e capacidade analítica. Por fim, a documentação e explicação das funcionalidades exigiram clareza na comunicação técnica, fortalecendo a habilidade de relatar e justificar decisões de implementação.

7. Referências

https://www.studocu.com/pt-br/document/universidade-do-estado-do-amazonas/engenharia-da-computacao/sistema-de-gerenciamento-de-reservas-para-hospedagem-tcc-utfpr/123669145

https://sistemafacilityhotel.com.br/como-que-funciona-um-sistema-de-reservas-de-hotel/#:~:text=O%20sistema%20de%20reservas%20de,garantir%20transa%C3%A7%C3%B5es%20seguras%20e%20eficientes.

https://ric.cps.sp.gov.br/bitstream/123456789/3284/1/20182S_SOUSAMichelRodrigode_OD0577.pdf

Apêndices

Apêndice A: Código fonte

```
#include <stdio.h>              // Entrada/saída padrão (printf, scanf)
#include <stdlib.h>             // Alocação de memória, exit, realloc, free
#include <string.h>             // Operações com strings (strcpy, strcmp)

#define TAM_TIPO 20             // Tamanho máximo do campo 'tipo' do quarto
#define LIVRE 0                 // Constante para status LIVRE do quarto
#define OCUPADO 1               // Constante para status OCUPADO do quarto
#define MANUTENCAO 2            // Constante para status MANUTENCAO do quarto
```

```
#define TAM_NOME 50                // Tamanho máximo do campo 'nome' do hóspede
#define TAM_CPF 15                 // Tamanho máximo do campo 'cpf' (string)
#define TAM_TELEFONE 20            // Tamanho máximo do campo 'telefone'
#define TAM_DATA 11                // Tamanho para data "DD/MM/AAAA" + '\0' (10 + 1)
#define ATIVA 0                    // Constante para reserva ATIVA
#define CONCLUIDA 1                // Constante para reserva CONCLUIDA
#define CANCELADA 2                // Constante para reserva CANCELADA

typedef struct {                   // Estrutura que representa um quarto
  int numero;                      // Número do quarto
  char tipo[TAM_TIPO];             // Tipo do quarto (ex: "Standard")
  float preco_diaria;              // Preço da diária
  int status;                      // Status atual (LIVRE/OCUPADO/MANUTENCAO)
} Quarto;

typedef struct {                   // Estrutura que representa um hóspede
  int id_hospede;                  // Identificador único do hóspede
  char nome[TAM_NOME];             // Nome do hóspede
  char cpf[TAM_CPF];               // CPF do hóspede (string)
  char telefone[TAM_TELEFONE];     // Telefone do hóspede
```

```
   int *historico_ids_reservas;       // Vetor dinâmico de IDs de reservas no histórico
   int num_reservas_historico;        // Quantidade de IDs armazenados no histórico
} Hospede;

typedef struct {                  // Estrutura que representa uma reserva
   int id_reserva;                // ID único da reserva
   int numero_quarto;             // Número do quarto reservado
   int id_hospede;                // ID do hóspede que fez a reserva
   char data_checkin[TAM_DATA];       // Data de check-in (DD/MM/AAAA)
   char data_checkout[TAM_DATA];      // Data de check-out (DD/MM/AAAA)
   int status_reserva;            // Status da reserva (ATIVA/CONCLUIDA/CANCELADA)
   float valor_total;             // Valor total da reserva
} Reserva;

Quarto *quartos_hotel = NULL;         // Ponteiro para o array dinâmico de quartos (inicialmente vazio)
int contador_quartos = 0;             // Contador do número de quartos cadastrados

Hospede * hospedes_hotel = NULL;      // Ponteiro para o array dinâmico de hóspedes
int contador_hospedes = 0;            // Contador do número de hóspedes cadastrados
```

```
Reserva *reservas_hotel = NULL;        // Ponteiro para o array dinâmico de reservas
int contador_reservas = 0;             // Contador do número de reservas cadastradas



//CALCULOS PARA DIARIA
int eh_bissexto(int ano);              // Protótipo: verifica se ano é bissexto
long contar_dias(int dia, int mes, int ano); // Protótipo: converte data em dias absolutos
int calcular_diferenca_dias(const char* data1_str, const char* data2_str); // Protótipo

int eh_bissexto(int ano) {             // Implementação: verifica ano bissexto
   return (ano % 4 == 0 && ano % 100 != 0) || (ano % 400 == 0); // Regra do calendário gregoriano
}

long contar_dias(int dia, int mes, int ano) { // Converte uma data em um número total de dias
   int dias_por_mes[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; // Dias por mês (índice 1..12)
   long n_dias = 0;                    // Acumulador de dias

   for (int i = 0; i < ano; i++) {     // Soma dias dos anos completos anteriores
```

```
    n_dias += 365;              // 365 dias por ano comum
    if (eh_bissexto(i)) {       // Se ano anterior foi bissexto, soma 1 dia
      n_dias++;
    }
  }

  for (int i = 1; i < mes; i++) {   // Soma dias dos meses completos do ano atual
    n_dias += dias_por_mes[i];
  }
  if (mes > 2 && eh_bissexto(ano)) { // Se já passou fevereiro e o ano atual é bissexto, soma 1
    n_dias++;
  }

  n_dias += dia;                // Soma os dias do mês corrente
  return n_dias;                // Retorna total de dias
}

int calcular_diferenca_dias(const char* data1_str, const char* data2_str) { // Diferença absoluta entre duas datas
  int dia1, mes1, ano1, dia2, mes2, ano2; // Variáveis para armazenar componentes das datas

  sscanf(data1_str, "%d/%d/%d", &dia1, &mes1, &ano1); // Extrai dia/mes/ano da string data1
```

```
   sscanf(data2_str, "%d/%d/%d", &dia2, &mes2, &ano2); // Extrai dia/mes/ano da string data2


   long n_dias1 = contar_dias(dia1, mes1, ano1); // Converte data1 para dias absolutos
   long n_dias2 = contar_dias(dia2, mes2, ano2); // Converte data2 para dias absolutos


   // labs() retorna valor absoluto de long; diferença de dias positiva
   return labs(n_dias2 - n_dias1);
}


void realocar_quartos() {              // Realoca (ou aloca inicialmente) o array de quartos para +1 elemento
   Quarto *temp = (Quarto *)realloc(quartos_hotel, (contador_quartos + 1) * sizeof(Quarto)); // realloc para (n+1)


   if (temp == NULL) {                 // Verifica se a realocação falhou
      printf("Erro fatal: Nao foi possivel alocar memoria para quarto!\n");
      if (quartos_hotel != NULL) free(quartos_hotel); // Libera memória anterior, se houver
      exit(1);                         // Encerra o programa devido ao erro crítico
   }
   quartos_hotel = temp;               // Atualiza ponteiro global para o novo bloco
}
```

```
int verificar_quarto_existe(int numero_procurado) { // Verifica se já existe quarto com certo número
  for (int i = 0; i < contador_quartos; i++) { // Percorre todos os quartos cadastrados
    if (quartos_hotel[i].numero == numero_procurado) { // Se número bate, retorna 1 (existe)
      return 1;
    }
  }
  return 0;                    // Retorna 0 se não encontrar
}

void cadastrar_quarto(){            // Função que cadastra um novo quarto
  realocar_quartos();             // Garante espaço para um novo quarto

  Quarto *novo_quarto = &quartos_hotel[contador_quartos]; // Ponteiro para o novo elemento
  int escolha_status;             // Variável para guardar status escolhido
  int numero_digitado;             // Variável para número do quarto digitado pelo usuário

  printf("\nCadastro de Quarto\n");    // Mensagem inicial

  do {                        // Loop para garantir número único
    printf("Digite o numero do quarto: ");
    scanf("%d", &numero_digitado);    // Lê número do quarto
```

```
    if (verificar_quarto_existe(numero_digitado)) { // Se já existir, avisa
        printf("ERRO: O quarto %d ja existe. Digite um numero unico.\n", numero_digitado);
    }
  } while (verificar_quarto_existe(numero_digitado)); // Repetir enquanto número existir
  novo_quarto->numero = numero_digitado; // Armazena número no novo quarto

  printf("Digite o tipo do quarto (ex: Standard, Deluxe): ");
  scanf("%s", novo_quarto->tipo);      // Lê tipo (palavra sem espaços) e armazena

  printf("Digite o preco da diaria (ex: 150.00): ");
  scanf("%f", &novo_quarto->preco_diaria); // Lê preço da diária e armazena

  do {                          // Loop para validar entrada do status
    printf("\nDigite o numero de acordo com o status do quarto:\n");
    printf("0 - LIVRE\n1 - OCUPADO\n2 - MANUTENCAO\n");
    printf("Escolha: ");
    scanf("%d", &escolha_status);    // Lê escolha do usuário
    if (escolha_status < 0 || escolha_status > 2) { // Verifica validade
        printf("\nOpcao invalida. Digite novamente.\n");
    }
  } while (escolha_status < 0 || escolha_status > 2);
```

```
   novo_quarto->status = escolha_status;  // Atribui status ao novo
quarto


   contador_quartos++;                // Incrementa contador global de
quartos
   printf("\nQuarto %d cadastrado com sucesso!\n", novo_quarto-
>numero); // Confirma cadastro
}


void listar_quartos() {                // Função para listar todos os quartos
cadastrados
   printf("\n--- Lista de Quartos Cadastrados (%d no total) ---\n",
contador_quartos); // Cabeçalho
   if (contador_quartos == 0) {        // Se nenhum quarto cadastrado
      printf("Nenhum quarto cadastrado ainda.\n");
      return;
   }


   printf("Numero | Tipo        | Diaria   | Status\n"); // Títulos das colunas
   printf("-----------------------------------------------\n");
   for (int i = 0; i < contador_quartos; i++) { // Percorre cada quarto
      // Converte status numérico para string legível
      const char* status_str = (quartos_hotel[i].status == LIVRE) ? "Livre"
:
                        (quartos_hotel[i].status == OCUPADO) ? "Ocupado" :
                        (quartos_hotel[i].status == MANUTENCAO) ?
"Manutencao" : "Desconhecido";
```

```
      printf("%-6d | %-13s | R$%-6.2f | %s\n",
          quartos_hotel[i].numero,    // Exibe número do quarto
          quartos_hotel[i].tipo,      // Exibe tipo
          quartos_hotel[i].preco_diaria, // Exibe preço formatado
          status_str);              // Exibe status em texto
    }
}

void realocar_hospedes() {              // Realoca (ou aloca) o array de hóspedes para +1
    Hospede *temp = (Hospede *)realloc(hospedes_hotel, (contador_hospedes + 1) * sizeof(Hospede));
    if (temp == NULL) {                 // Checa falha de alocação
        printf("Erro fatal: Nao foi possivel alocar memoria para hospede!\n");
        if (hospedes_hotel != NULL) free(hospedes_hotel); // Libera memória anterior, se houver
        exit(1);                        // Finaliza o programa
    }
    hospedes_hotel = temp;              // Atualiza ponteiro global
}

int verificar_hospede_existe(char* cpf_procurado) { // Verifica se CPF já está cadastrado
    for (int i = 0; i < contador_hospedes; i++) {
```

```
    if (strcmp(hospedes_hotel[i].cpf, cpf_procurado) == 0) { // Comparação de strings
        return 1;                // Retorna 1 se encontrar
    }
  }
  return 0;                      // Retorna 0 caso contrário
}

int buscar_hospede_por_cpf(char* cpf_procurado) { // Retorna índice do hóspede pelo CPF
  for (int i = 0; i < contador_hospedes; i++) {
    if (strcmp(hospedes_hotel[i].cpf, cpf_procurado) == 0) { // Igualdade de CPF?
        return i;                // Retorna índice no array
    }
  }
  return -1;                     // Retorna -1 se não encontrar
}

void cadastrar_hospede(){              // Função para cadastrar novo hóspede
  realocar_hospedes();               // Garante espaço para +1 hóspede
  Hospede *novo_hospede = &hospedes_hotel[contador_hospedes]; // Ponteiro para novo elemento
  char cpf_digitado[TAM_CPF];        // Buffer local para CPF digitado

  printf("\nCadastro de Hospede\n");    // Cabeçalho
```

```
  do {                        // Loop para garantir CPF único
    printf("Digite o CPF (apenas numeros): ");
    scanf("%s", cpf_digitado);       // Lê CPF como string

    if (verificar_hospede_existe(cpf_digitado)) { // Se CPF já existe, avisa
        printf("ERRO: O CPF %s ja esta cadastrado.\n", cpf_digitado);
    }
  } while (verificar_hospede_existe(cpf_digitado)); // Repetir enquanto CPF existir
  strcpy(novo_hospede->cpf, cpf_digitado); // Copia CPF para estrutura do hóspede

  printf("Digite o nome: ");
  scanf("%s", novo_hospede->nome);      // Lê nome (sem espaços) e armazena

  printf("Digite o telefone: ");
  scanf("%s", novo_hospede->telefone);   // Lê telefone e armazena

  novo_hospede->id_hospede = contador_hospedes + 1; // Atribui ID sequencial
  novo_hospede->historico_ids_reservas = NULL; // Inicializa histórico vazio
  novo_hospede->num_reservas_historico = 0;   // Inicializa contagem do histórico
```

```
    contador_hospedes++;              // Incrementa contador global de hóspedes
    printf("\nHospede %s cadastrado com sucesso! ID: %d\n", novo_hospede->nome, novo_hospede->id_hospede); // Confirma
}

void listar_hospedes() {              // Lista todos os hóspedes cadastrados
    printf("\n--- Lista de Hospedes Cadastrados (%d no total) ---\n", contador_hospedes); // Cabeçalho
    if (contador_hospedes == 0) {       // Se não há hóspedes
        printf("Nenhum hospede cadastrado ainda.\n");
        return;
    }
    printf("ID   | Nome        | CPF          | Telefone      | Historico Reservas\n"); // Títulos
    printf("-------------------------------------------------------------------------\n");
    for (int i = 0; i < contador_hospedes; i++) { // Percorre o array de hóspedes
        printf("%-4d | %-13s | %-14s | %-14s | Total: %d\n",
            hospedes_hotel[i].id_hospede,     // Exibe ID do hóspede
            hospedes_hotel[i].nome,           // Exibe nome
            hospedes_hotel[i].cpf,            // Exibe CPF
            hospedes_hotel[i].telefone,       // Exibe telefone
            hospedes_hotel[i].num_reservas_historico); // Exibe número de reservas no histórico
```

```
    }
}
void atualizar_status_quarto(int numero_quarto, int novo_status) { // Atualiza status do quarto pelo número
    for (int i = 0; i < contador_quartos; i++) { // Percorre quartos
        if (quartos_hotel[i].numero == numero_quarto) { // Se encontra o quarto desejado
            quartos_hotel[i].status = novo_status; // Atualiza o status
            return;                    // Sai da função
        }
    }
}

int buscar_quarto_por_numero(int num_procurado) { // Retorna índice do quarto dado o número
    for (int i = 0; i < contador_quartos; i++) { // Percorre array de quartos
        if (quartos_hotel[i].numero == num_procurado) { // Comparação do número
            return i;                  // Retorna índice se encontrado
        }
    }
    return -1;                         // Retorna -1 se não encontrado
}

void realocar_reservas() {             // Realoca (ou aloca) array de reservas para +1
```

```
   Reserva *temp = (Reserva *)realloc(reservas_hotel,
(contador_reservas + 1) * sizeof(Reserva));
   if (temp == NULL) {                // Verifica falha de alocação
      printf("Erro fatal: Nao foi possivel alocar memoria para
reserva!\n");
      if (reservas_hotel != NULL) free(reservas_hotel); // Libera bloco
anterior, se houver
      exit(1);                        // Finaliza por erro crítico
   }
   reservas_hotel = temp;             // Atualiza ponteiro global
}
void realizar_reserva() {             // Função para criar uma nova reserva
   char cpf_busca[TAM_CPF];           // Buffer para CPF informado
   int indice_hospede;                // Índice do hóspede no array
   int numero_quarto_escolhido;       // Número do quarto escolhido
   char checkin_str[TAM_DATA];        // String do check-in
   char checkout_str[TAM_DATA];       // String do check-out
   int indice_quarto;                 // Índice do quarto no array
   int validacao = 0;                 // Flag de validação do quarto
   int verificar_datas = 0;           // Flag para validação das datas
   float preco_diaria_quarto;         // Preço da diária do quarto
selecionado
   int dias_estadia;                  // Dias calculados entre checkin e
checkout

   printf("\nREALIZAR RESERVA\n");
```

```
  if(contador_quartos ==0){            // Se não há quartos cadastrados, aborta operação
    printf("Nenhum quarto cadastrado no sistema\n");
    return;
  }

  do {                          // Loop para encontrar hóspede existente
    printf("Digite o CPF do hospede (apenas numeros, ou '0' para cancelar): ");
    scanf("%s", cpf_busca);          // Lê CPF ou '0' para cancelar

    if (strcmp(cpf_busca, "0") == 0) {   // Se usuário optou por cancelar
      printf("Operacao de reserva cancelada.\n");
      return;
    }

    indice_hospede = buscar_hospede_por_cpf(cpf_busca); // Busca hóspede pelo CPF

    if (indice_hospede == -1) {       // Se não encontrado, avisa e repete
      printf("ERRO: Hospede nao cadastrado. Tente novamente ou digite '0' para sair.\n");
    }
  } while (indice_hospede == -1);       // Repete até encontrar hóspede ou cancelar

  do{                           // Loop para selecionar quarto disponível
```

```
    printf("QUARTOS DISPONIVEIS\n");
    listar_quartos();               // Exibe lista de quartos

    printf("\nDIGITE O NUMERO DO QUARTO ESCOLHIDO (ou caso queira cancelar a reserva digite 0):\n");
    scanf("%d", &numero_quarto_escolhido); // Lê número do quarto escolhido

    if(numero_quarto_escolhido == 0){    // Se usuário cancelar a seleção
        printf("RESERVA CANCELADA\n");
        return;
    }

    indice_quarto = buscar_quarto_por_numero(numero_quarto_escolhido); // Busca índice do quarto
    if (indice_quarto == -1) {        // Quarto não existe?
        printf("ERRO: Quarto %d nao existe. Tente novamente.\n", numero_quarto_escolhido);
        validacao = 0;
    }
    else if (quartos_hotel[indice_quarto].status != LIVRE) { // Se não estiver livre
        printf("ERRO: Quarto %d esta ocupado ou em manutencao. Tente novamente.\n", numero_quarto_escolhido);
        validacao = 0;
```

```
    }
    else{
      validacao = 1;              // Quarto válido e livre
    }
  }while (validacao == 0);          // Repete enquanto não for válido

  preco_diaria_quarto = quartos_hotel[indice_quarto].preco_diaria; // Pega preço da diária

  do{                           // Loop para validar datas
    printf("Digite a data de Check-in (DD/MM/AAAA): ");
    scanf("%s", checkin_str);       // Lê check-in
    printf("Digite a data de Check-out (DD/MM/AAAA): ");
    scanf("%s", checkout_str);      // Lê check-out
    dias_estadia = calcular_diferenca_dias(checkin_str, checkout_str); // Calcula diferença de dias
    if (dias_estadia <= 0) {        // Verifica se check-out é posterior a check-in
      printf("ERRO: A data de check-out deve ser posterior a de check-in. Tente novamente\n");
      verificar_datas = 1;
    }else{
      verificar_datas = 0;
    }
  }while(verificar_datas == 1);       // Repete enquanto datas inválidas
```

```
    float valor_total = preco_diaria_quarto * dias_estadia; // Calcula valor total da reserva


    realocar_reservas();                // Realoca array de reservas para adicionar nova
    Reserva *nova_reserva = &reservas_hotel[contador_reservas]; // Ponteiro para a nova reserva
    nova_reserva->id_reserva = contador_reservas + 1; // Atribui ID sequencial
    nova_reserva->numero_quarto = numero_quarto_escolhido; // Guarda número do quarto
    nova_reserva->id_hospede = hospedes_hotel[indice_hospede].id_hospede; // Liga ao hóspede
    strcpy(nova_reserva->data_checkin, checkin_str); // Armazena data check-in
    strcpy(nova_reserva->data_checkout, checkout_str); // Armazena data check-out
    nova_reserva->status_reserva = ATIVA;    // Marca reserva como ATIVA
    nova_reserva->valor_total = valor_total; // Armazena valor total calculado
    atualizar_status_quarto(numero_quarto_escolhido, OCUPADO); // Marca quarto como OCUPADO
    contador_reservas++;                // Incrementa contador de reservas
    printf("\nReserva %d realizada com sucesso para %s no Quarto %d.\n", nova_reserva->id_reserva, hospedes_hotel[indice_hospede].nome, numero_quarto_escolhido); // Mensagem de sucesso
```

```
   printf("Estadia de %d dias. Valor total: R$%.2f\n", dias_estadia, valor_total); // Mostra resumo da reserva
}

void listar_reservas_ativas() {          // Lista reservas que estão com status ATIVA
   printf("\n--- Lista de Reservas Ativas (%d no total) ---\n", contador_reservas); // Cabeçalho (observação: mostra total de reservas, não só ativas)
   if (contador_reservas == 0) { printf("Nenhuma reserva ativa.\n"); return; } // Se não há reservas cadastradas
   printf("ID Res. | Quarto | ID Hosp. | Check-In   | Check-Out  | Valor Total | Status\n");
   printf("-------------------------------------------------------------------------------\n");
   for (int i = 0; i < contador_reservas; i++) { // Percorre todas as reservas
      if (reservas_hotel[i].status_reserva == ATIVA) { // Mostra somente as ATIVAS
         printf("%-7d | %-6d | %-8d | %-10s | %-10s | R$%-8.2f | Ativa\n",
            reservas_hotel[i].id_reserva,      // ID da reserva
            reservas_hotel[i].numero_quarto,   // Número do quarto
            reservas_hotel[i].id_hospede,      // ID do hóspede
            reservas_hotel[i].data_checkin,    // Check-in
            reservas_hotel[i].data_checkout,   // Check-out
            reservas_hotel[i].valor_total);    // Valor total
      }
   }
```

```
}

void adicionar_reserva_ao_historico(int id_hospede, int id_reserva) { // Adiciona ID da reserva ao histórico do hóspede
  int index = -1;                  // Índice local do hóspede

  for (int i = 0; i < contador_hospedes; i++) { // Encontra o hóspede pelo ID
    if (hospedes_hotel[i].id_hospede == id_hospede) {
      index = i;
      break;
  }
}
  if (index == -1) return;            // Se não encontrou hóspede, sai

  Hospede *h = &hospedes_hotel[index];   // Ponteiro para o hóspede encontrado

  h->historico_ids_reservas = realloc(    // Realoca array de IDs no histórico para +1
    h->historico_ids_reservas,
    (h->num_reservas_historico + 1) * sizeof(int)
  );

  h->historico_ids_reservas[h->num_reservas_historico] = id_reserva; // Armazena novo ID
```

```
   h->num_reservas_historico++;       // Incrementa contagem do
histórico
}



void gerenciar_reserva(){              // Função para cancelar ou concluir
reservas ativas
   int id_reserva_alvo;              // ID da reserva alvo informado pelo
usuário
   int quarto_associado;             // Número do quarto associado à
reserva
   int encontrado = 0;               // Flag para controlar busca
   int sub_opcao = 0;                // Escolha do usuário
(cancelar/concluir)
   int novo_status_reserva;          // Novo status a ser aplicado


   printf("\nGERENCIAR RESERVAS ATIVAS\n");
    if (contador_reservas == 0) {       // Se não há reservas no sistema
      printf("Nenhuma reserva ativa no sistema.\n");
      return;
   }
   listar_reservas_ativas();          // Mostra reservas ativas para o
usuário


   do {
      printf("\nDigite o ID da reserva alvo (ou 0 para sair): ");
      scanf("%d", &id_reserva_alvo);    // Lê o ID desejado
```

```
    if (id_reserva_alvo == 0) {       // Se usuário cancelar a operação
      printf("Genciamento cancelado\n");
      return;
    }
    encontrado = 0;                // Reinicia flag de encontrado

    for (int i = 0; i < contador_reservas; i++) { // Percorre reservas cadastradas
      if (reservas_hotel[i].id_reserva == id_reserva_alvo && reservas_hotel[i].status_reserva == ATIVA) { // Encontra reserva ativa
        encontrado = 1;

        printf("\nReserva %d encontrada. Escolha a acao:\n", id_reserva_alvo); // Mostra opções
        printf("1 - CANCELAR reserva (Status: CANCELADA)\n");
        printf("2 - CONCLUIR reserva (Status: CONCLUIDA / Check-out)\n");
        do{
          scanf("%d", &sub_opcao); // Lê escolha do usuário
          if (sub_opcao == 1) {
            novo_status_reserva = CANCELADA; // Define novo status como CANCELADA
            printf("Reserva %d CANCELADA.\n", id_reserva_alvo);
          }else if (sub_opcao == 2) {
            novo_status_reserva = CONCLUIDA; // Define novo status como CONCLUIDA
```

```
                printf("Reserva %d CONCLUIDA (Check-out).\n",
id_reserva_alvo);
            } else {
                printf("Opcao invalida. Digite novamente\n");
            }
        }while(sub_opcao != 1 && sub_opcao != 2);
adicionar_reserva_ao_historico(reservas_hotel[i].id_hospede,
reservas_hotel[i].id_reserva); // Move reserva para histórico do
hóspede
        reservas_hotel[i].status_reserva = novo_status_reserva; //
Atualiza status da reserva
        quarto_associado = reservas_hotel[i].numero_quarto; //
Recupera quarto associado
        atualizar_status_quarto(quarto_associado, LIVRE); // Libera o
quarto
        printf("Quarto %d agora esta LIVRE.\n", quarto_associado); //
Informa liberação
        return;                 // Sai após gerenciar a reserva
      }
    }
    if (encontrado == 0) {          // Se não encontrou reserva válida
      printf("ERRO: ID de reserva invalido ou reserva inativa. Tente
novamente.\n");
    }
  } while (encontrado == 0);        // Repete até encontrar ou sair
}
```

```
void mostrar_historico() {              // Mostra histórico de reservas de um hóspede
   char cpf[TAM_CPF];                   // Buffer para CPF digitado
   printf("Digite o CPF do hospede: ");
   scanf("%s", cpf);                    // Lê CPF

   int index = buscar_hospede_por_cpf(cpf); // Busca índice do hóspede
   if (index == -1) {                   // Se não encontrado
      printf("Hospede nao encontrado.\n");
      return;
   }

   Hospede *h = &hospedes_hotel[index];   // Ponteiro para o hóspede encontrado

   printf("\nHistorico de Reservas de %s (CPF %s)\n",
         h->nome, h->cpf);              // Cabeçalho com nome e CPF

   if (h->num_reservas_historico == 0) {  // Se não há histórico
      printf("Nenhuma reserva concluida/cancelada ainda.\n");
      return;
   }
```

```
  for (int i = 0; i < h->num_reservas_historico; i++) { // Percorre IDs no
histórico
    int id = h->historico_ids_reservas[i]; // Recupera ID da reserva
    printf("- Reserva ID %d\n", id);    // Imprime cada ID
  }
}

int conflito_datas(const char* data_in1, const char* data_out1,
            const char* data_in2, const char* data_out2)
{                          // Verifica se dois intervalos [in1,out1) e [in2,out2)
se sobrepõem
  long in1 = contar_dias(
    atoi(data_in1), atoi(data_in1+3), atoi(data_in1+6)
  );                       // Converte data_in1 para dias (atoi interpreta
substring)
  long out1 = contar_dias(
    atoi(data_out1), atoi(data_out1+3), atoi(data_out1+6)
  );                       // Converte data_out1 para dias
  long in2 = contar_dias(
    atoi(data_in2), atoi(data_in2+3), atoi(data_in2+6)
  );                       // Converte data_in2
  long out2 = contar_dias(
    atoi(data_out2), atoi(data_out2+3), atoi(data_out2+6)
  );                       // Converte data_out2
```

```
   return (in1 < out2 && out1 > in2);    // Retorna true se houver sobreposição
}

int quarto_disponivel_periodo(int numero_quarto, const char* data_in, const char* data_out)
{                                   // Retorna 1 se quarto estiver disponível no período, 0 caso contrário
   for (int i = 0; i < contador_reservas; i++) { // Percorre todas as reservas

      if (reservas_hotel[i].numero_quarto != numero_quarto) { // Se reserva é de outro quarto, ignora
         continue; // não é o quarto procurado
      }

      if (reservas_hotel[i].status_reserva == CANCELADA) { // Reservas canceladas não bloqueiam
         continue; // reservas canceladas não bloqueiam o quarto
      }

      // Se datas conflitam → o quarto NÃO está disponível
      if (conflito_datas(data_in, data_out,
                         reservas_hotel[i].data_checkin,
                         reservas_hotel[i].data_checkout))
      {
         return 0; // Indisponível (conflito encontrado)
      }
```

```
    }

    return 1; // Disponível (nenhum conflito)
}



void listar_quartos_disponiveis_periodo()
{                           // Lista quartos que estão livres para um período
informado
    if (contador_quartos == 0) {        // Se nenhum quarto cadastrado
        printf("Nenhum quarto cadastrado.\n");
        return;
    }

    char data_in[TAM_DATA];             // Buffer para data de check-in
    char data_out[TAM_DATA];            // Buffer para data de check-out

    printf("Digite a data de Check-in (DD/MM/AAAA): ");
    scanf("%s", data_in);               // Lê check-in

    printf("Digite a data de Check-out (DD/MM/AAAA): ");
    scanf("%s", data_out);              // Lê check-out

    printf("\n--- QUARTOS DISPONIVEIS DE %s A %s ---\n", data_in,
data_out); // Cabeçalho mostrando período
```

```
  int encontrou = 0;                // Flag para indicar se encontrou algum quarto

  for (int i = 0; i < contador_quartos; i++) { // Percorre todos os quartos
    if (quarto_disponivel_periodo(quartos_hotel[i].numero, data_in, data_out)) { // Se disponível no período
      printf("Quarto %d (%s) - R$ %.2f / dia\n",
          quartos_hotel[i].numero,  // Número do quarto
          quartos_hotel[i].tipo,    // Tipo do quarto
          quartos_hotel[i].preco_diaria); // Preço por dia

      encontrou = 1;            // Marca que encontrou pelo menos um
    }
  }

  if (!encontrou) {                 // Se não encontrou nenhum quarto disponível
    printf("Nenhum quarto disponível para esse período.\n");
  }
}

int main(){                         // Função principal do programa
  int opcao = 0;                    // Variável para opção do menu

  do{
    printf("\n--- MENU PRINCIPAL ---\n"); // Mostra menu
```

```
    printf("1 - CADASTRAR QUARTO\n");
    printf("2 - CADASTRAR HOSPEDE\n");
    printf("3 - RELIZAR RESERVA\n");
    printf("4 - GERENCIAR RESERVAS\n");
    printf("5 - LISTAR RESERVAS ATIVAS\n");
    printf("6 - LISTAR QUARTOS\n");
    printf("7 - LISTAR HOSPEDES\n");
    printf("8 - HISTORICO DE RESERVAS\n");
    printf("9 - VERIFICAR DISPONIBILIDADE POR PERIODO\n");
    printf("10 - SAIR\n");
    printf("Escolha uma opcao: ");
    scanf("%d", &opcao);            // Lê opção escolhida pelo usuário

    switch(opcao){                  // Roteador de opções do menu
      case 1:
        cadastrar_quarto();         // Chama função de cadastro de quarto
        break;
      case 2:
        cadastrar_hospede();        // Chama função de cadastro de
hóspede
        break;
      case 3:
        realizar_reserva();         // Chama função para realizar reserva
        break;
      case 4:
```

```
            gerenciar_reserva();       // Chama função para gerenciar
reservas
            break;
        case 5:
            listar_reservas_ativas();   // Lista reservas ativas
            break;
        case 6:
            listar_quartos();        // Lista quartos cadastrados
            break;
        case 7:
            listar_hospedes();       // Lista hóspedes cadastrados
            break;
        case 8:
            mostrar_historico();      // Mostra histórico de um hóspede
            break;
        case 9:
            listar_quartos_disponiveis_periodo(); // Verifica
disponibilidade por período
        default:
            printf("Saindo do programa\n"); // Mensagem padrão
(observação: comportamento atual sai mesmo para opções inválidas)
      }
   }while (opcao != 10);              // Repete enquanto opção for diferente
de 10 (SAIR)


   if (quartos_hotel != NULL) {         // Antes de encerrar, libera memória
alocada para quartos
```

```
    free(quartos_hotel);
  }
  if(reservas_hotel != NULL){          // Libera memória das reservas, se
alocada
    free(reservas_hotel);
  }
   if (hospedes_hotel != NULL) {       // Se houver hóspedes alocados
    for(int i = 0; i < contador_hospedes; i++) { // Para cada hóspede,
libera o histórico de IDs se existir
      if(hospedes_hotel[i].historico_ids_reservas != NULL) {
        free(hospedes_hotel[i].historico_ids_reservas);
      }
    }
    free(hospedes_hotel);           // Libera o array de hóspedes
  }

  return 0;                         // Encerra o programa com sucesso
}
```

Apêndice B: Prints do funcionamento do programa:

```
--- MENU PRINCIPAL ---
1 - CADASTRAR QUARTO
2 - CADASTRAR HOSPEDE
3 - RELIZAR RESERVA
4 - GERENCIAR RESERVAS
5 - LISTAR RESERVAS ATIVAS
6 - LISTAR QUARTOS
7 - LISTAR HOSPEDES
8 - HISTORICO DE RESERVAS
9 - VERIFICAR DISPONIBILIDADE POR PERIODO
10 - SAIR
Escolha uma opcao: 1

Cadastro de Quarto
Digite o numero do quarto: 1
Digite o tipo do quarto (ex: Standard, Deluxe): Standard
Digite o preco da diaria (ex: 150.00): 120

Digite o numero de acordo com o status do quarto:
0 - LIVRE
1 - OCUPADO
2 - MANUTENCAO
Escolha: 0

Quarto 1 cadastrado com sucesso!
```

```
--- MENU PRINCIPAL ---
1 - CADASTRAR QUARTO
2 - CADASTRAR HOSPEDE
3 - RELIZAR RESERVA
4 - GERENCIAR RESERVAS
5 - LISTAR RESERVAS ATIVAS
6 - LISTAR QUARTOS
7 - LISTAR HOSPEDES
8 - HISTORICO DE RESERVAS
9 - VERIFICAR DISPONIBILIDADE POR PERIODO
10 - SAIR
Escolha uma opcao: 1

Cadastro de Quarto
Digite o numero do quarto: 2
Digite o tipo do quarto (ex: Standard, Deluxe): Standard
Digite o preco da diaria (ex: 150.00): 120

Digite o numero de acordo com o status do quarto:
0 - LIVRE
1 - OCUPADO
2 - MANUTENCAO
Escolha: 0

Quarto 2 cadastrado com sucesso!
```

```
--- MENU PRINCIPAL ---
1 - CADASTRAR QUARTO
2 - CADASTRAR HOSPEDE
3 - RELIZAR RESERVA
4 - GERENCIAR RESERVAS
5 - LISTAR RESERVAS ATIVAS
6 - LISTAR QUARTOS
7 - LISTAR HOSPEDES
8 - HISTORICO DE RESERVAS
9 - VERIFICAR DISPONIBILIDADE POR PERIODO
10 - SAIR
Escolha uma opcao: 2

Cadastro de Hospede
Digite o CPF (apenas numeros): 12312312312
Digite o nome: João
Digite o telefone: 11 98764-0682

Hospede João cadastrado com sucesso! ID: 1
```

```
Quarto 2 cadastrado com sucesso!

--- MENU PRINCIPAL ---
1 - CADASTRAR QUARTO
2 - CADASTRAR HOSPEDE
3 - RELIZAR RESERVA
4 - GERENCIAR RESERVAS
5 - LISTAR RESERVAS ATIVAS
6 - LISTAR QUARTOS
7 - LISTAR HOSPEDES
8 - HISTORICO DE RESERVAS
9 - VERIFICAR DISPONIBILIDADE POR PERIODO
10 - SAIR
Escolha uma opcao: 1

Cadastro de Quarto
Digite o numero do quarto: 3
Digite o tipo do quarto (ex: Standard, Deluxe): Luxo
Digite o preco da diaria (ex: 150.00): 300

Digite o numero de acordo com o status do quarto:
0 - LIVRE
1 - OCUPADO
2 - MANUTENCAO
Escolha: 0

Quarto 3 cadastrado com sucesso!
```

```
--- MENU PRINCIPAL ---
1 - CADASTRAR QUARTO
2 - CADASTRAR HOSPEDE
3 - RELIZAR RESERVA
4 - GERENCIAR RESERVAS
5 - LISTAR RESERVAS ATIVAS
6 - LISTAR QUARTOS
7 - LISTAR HOSPEDES
8 - HISTORICO DE RESERVAS
9 - VERIFICAR DISPONIBILIDADE POR PERIODO
10 - SAIR
Escolha uma opcao: 3

REALIZAR RESERVA
Digite o CPF do hospede (apenas numeros, ou '0' para cancelar): 12312312312
QUARTOS DISPONIVEIS

--- Lista de Quartos Cadastrados (3 no total) ---
Numero | Tipo          | Diaria   | Status
-----------------------------------------------
1      | Standard      | R$120.00 | Livre
2      | Standard      | R$120.00 | Livre
3      | Luxo          | R$300.00 | Livre

DIGITE O NUMERO DO QUARTO ESCOLHIDO (ou caso queira cancelar a reserva digite 0):
3
Digite a data de Check-in (DD/MM/AAAA): 12/12/2012
Digite a data de Check-out (DD/MM/AAAA): 13/12/2012

Reserva 1 realizada com sucesso para João no Quarto 3.
Estadia de 1 dias. Valor total: R$300.00
```

```
--- MENU PRINCIPAL ---
1 - CADASTRAR QUARTO
2 - CADASTRAR HOSPEDE
3 - RELIZAR RESERVA
4 - GERENCIAR RESERVAS
5 - LISTAR RESERVAS ATIVAS
6 - LISTAR QUARTOS
7 - LISTAR HOSPEDES
8 - HISTORICO DE RESERVAS
9 - VERIFICAR DISPONIBILIDADE POR PERIODO
10 - SAIR
Escolha uma opcao: 5

--- Lista de Reservas Ativas (1 no total) ---
ID Res. | Quarto | ID Hosp. | Check-In   | Check-Out  | Valor Total | Status
-------------------------------------------------------------------------------
1       | 3      | 1        | 12/12/2012 | 13/12/2012 | R$300.00    | Ativa
```

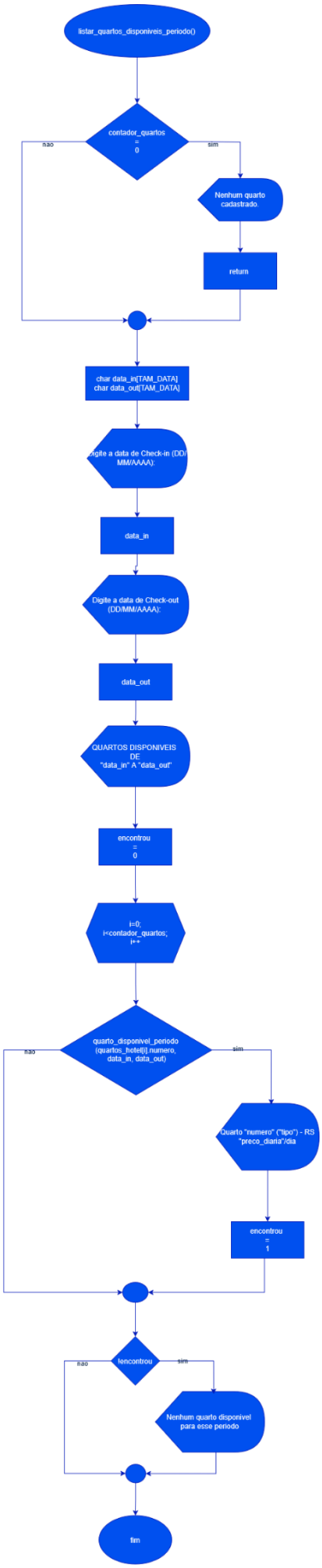
```
--- MENU PRINCIPAL ---
1 - CADASTRAR QUARTO
2 - CADASTRAR HOSPEDE
3 - RELIZAR RESERVA
4 - GERENCIAR RESERVAS
5 - LISTAR RESERVAS ATIVAS
6 - LISTAR QUARTOS
7 - LISTAR HOSPEDES
8 - HISTORICO DE RESERVAS
9 - VERIFICAR DISPONIBILIDADE POR PERIODO
10 - SAIR
Escolha uma opcao: 7

--- Lista de Hospedes Cadastrados (1 no total) ---
ID   | Nome          | CPF           | Telefone      | Historico Reservas
-------------------------------------------------------------------------
1    | João          | 12312312312   | 11            | Total: 0
```

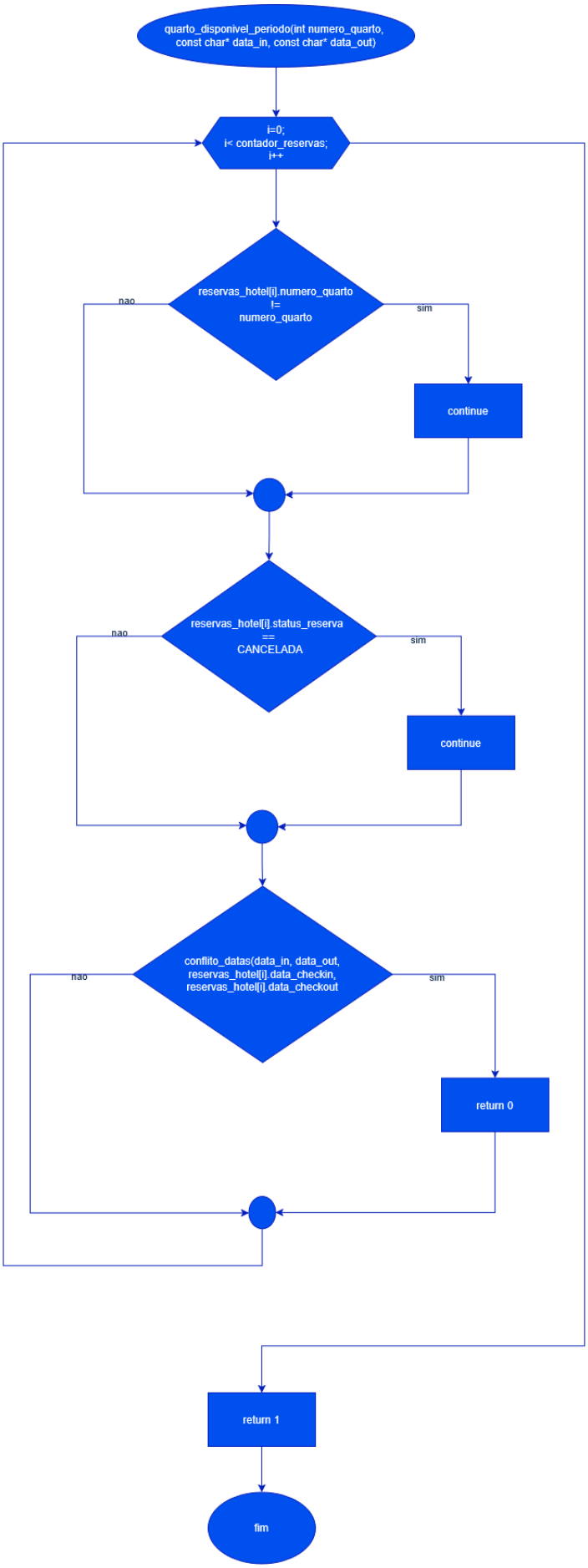
```
--- MENU PRINCIPAL ---
1 - CADASTRAR QUARTO
2 - CADASTRAR HOSPEDE
3 - RELIZAR RESERVA
4 - GERENCIAR RESERVAS
5 - LISTAR RESERVAS ATIVAS
6 - LISTAR QUARTOS
7 - LISTAR HOSPEDES
8 - HISTORICO DE RESERVAS
9 - VERIFICAR DISPONIBILIDADE POR PERIODO
10 - SAIR
Escolha uma opcao: 6

--- Lista de Quartos Cadastrados (3 no total) ---
Numero | Tipo          | Diaria   | Status
-----------------------------------------------
1      | Standard      | R$120.00 | Livre
2      | Standard      | R$120.00 | Livre
3      | Luxo          | R$300.00 | Ocupado
```

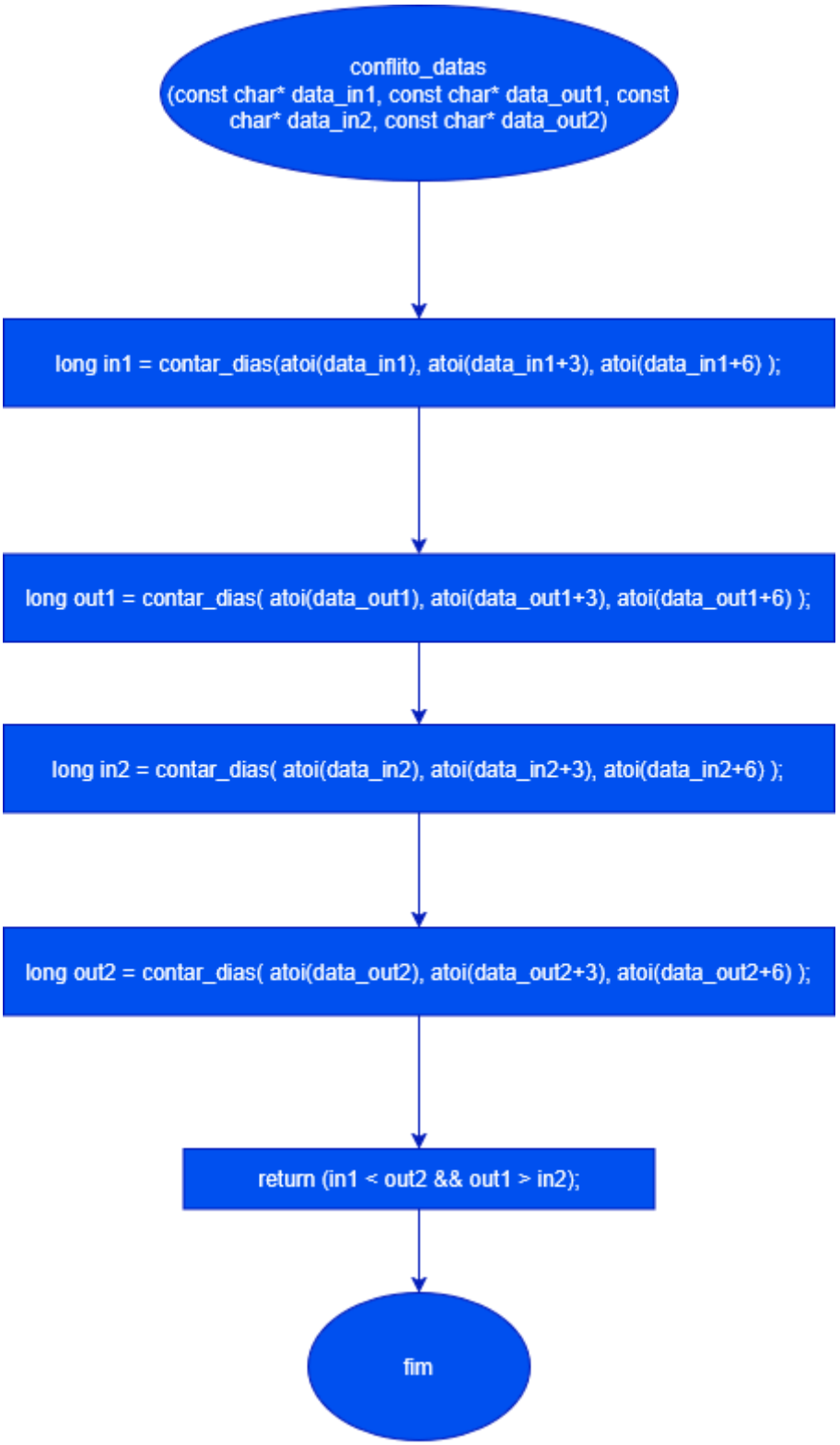
```
--- MENU PRINCIPAL ---
1 - CADASTRAR QUARTO
2 - CADASTRAR HOSPEDE
3 - RELIZAR RESERVA
4 - GERENCIAR RESERVAS
5 - LISTAR RESERVAS ATIVAS
6 - LISTAR QUARTOS
7 - LISTAR HOSPEDES
8 - HISTORICO DE RESERVAS
9 - VERIFICAR DISPONIBILIDADE POR PERIODO
10 - SAIR
Escolha uma opcao: 8
Digite o CPF do hospede: 9
Hospede nao encontrado.
```

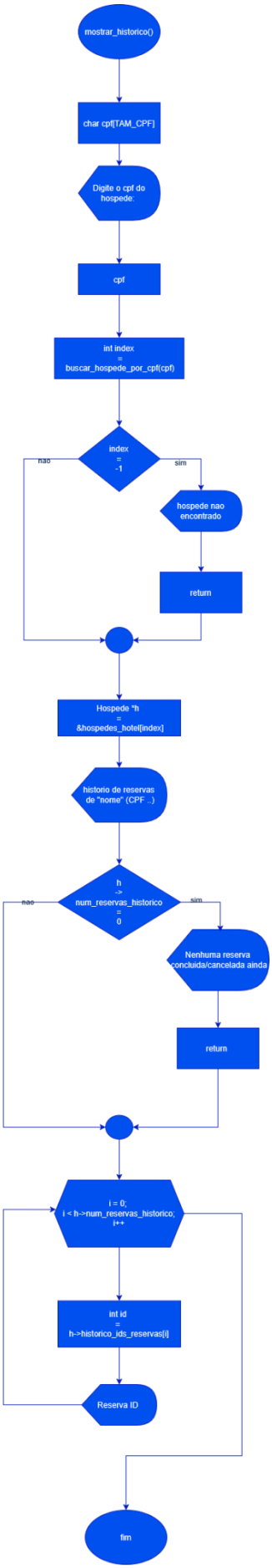
```
--- MENU PRINCIPAL ---
1 - CADASTRAR QUARTO
2 - CADASTRAR HOSPEDE
3 - RELIZAR RESERVA
4 - GERENCIAR RESERVAS
5 - LISTAR RESERVAS ATIVAS
6 - LISTAR QUARTOS
7 - LISTAR HOSPEDES
8 - HISTORICO DE RESERVAS
9 - VERIFICAR DISPONIBILIDADE POR PERIODO
10 - SAIR
Escolha uma opcao: 10
Saindo do programa
```

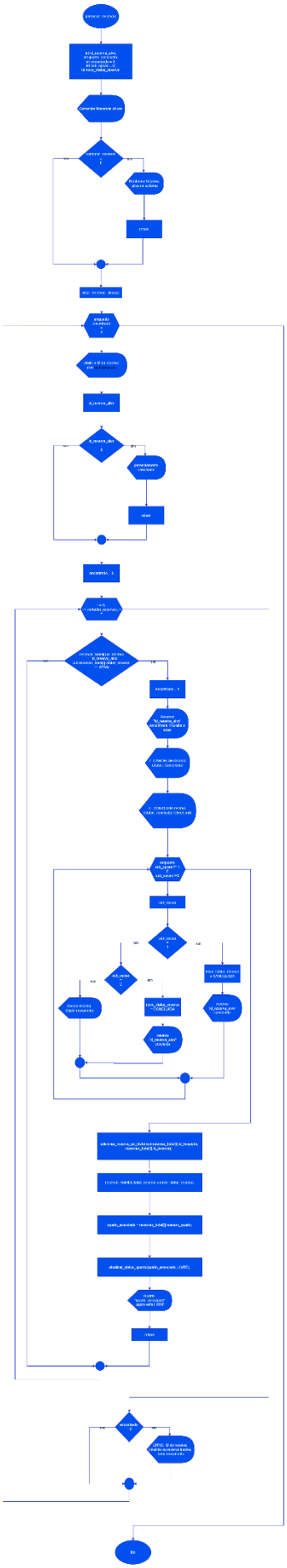
Apêndice C: Diagrama de Blocos:

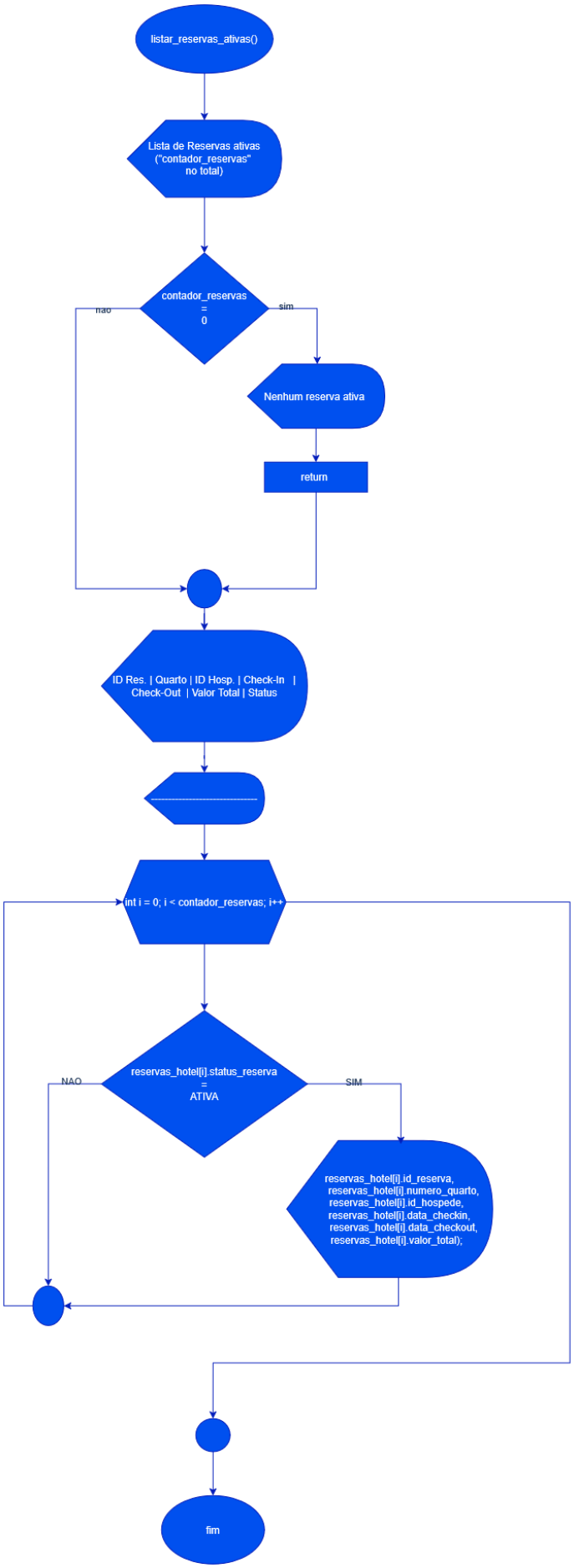

listar_quartos_disponiveis_periodo()
contador_quartos == 0
nao
sim
Nenhum quarto cadastrado
return
char data_in[TAM_DATA]
char data_out[TAM_DATA]
Digite a data de Check-in (DD/MM/AAAA):
data_in
Digite a data de Check-out (DD/MM/AAAA):
data_out
QUARTOS DISPONIVEIS DE "data_in" A "data_out"
encontrou = 0
i=0; i<contador_quartos; i++
quarto_disponivel_periodo (quartos_hotel[i].numero, data_in, data_out)
nao
sim
Quarto "numero" ("tipo") - R$ "preco_diaria"/dia
encontrou = 1
!encontrou
nao
sim
Nenhum quarto disponivel para esse periodo
fim

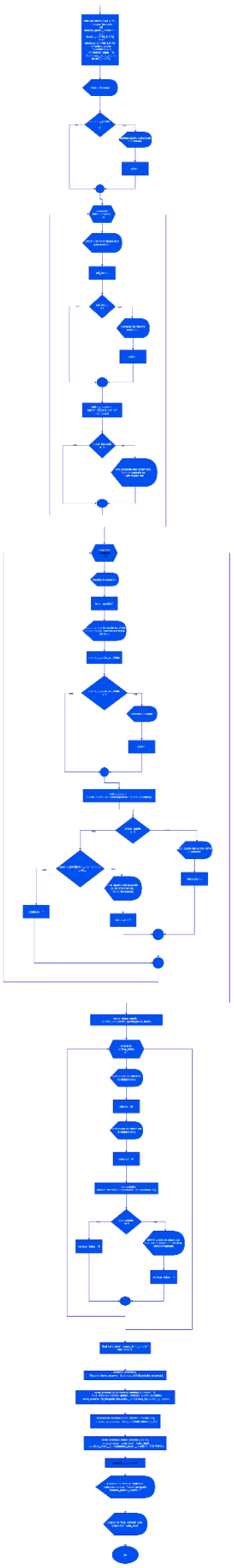

quarto_disponivel_periodo(int numero_quarto, const char* data_in, const char* data_out)
i=0; i< contador_reservas; i++
reservas_hotel[i].numero_quarto != numero_quarto
nao
sim
continue
reservas_hotel[i].status_reserva == CANCELADA
nao
sim
continue
conflito_datas(data_in, data_out, reservas_hotel[i].data_checkin, reservas_hotel[i].data_checkout
nao
sim
return 0
return 1
fim

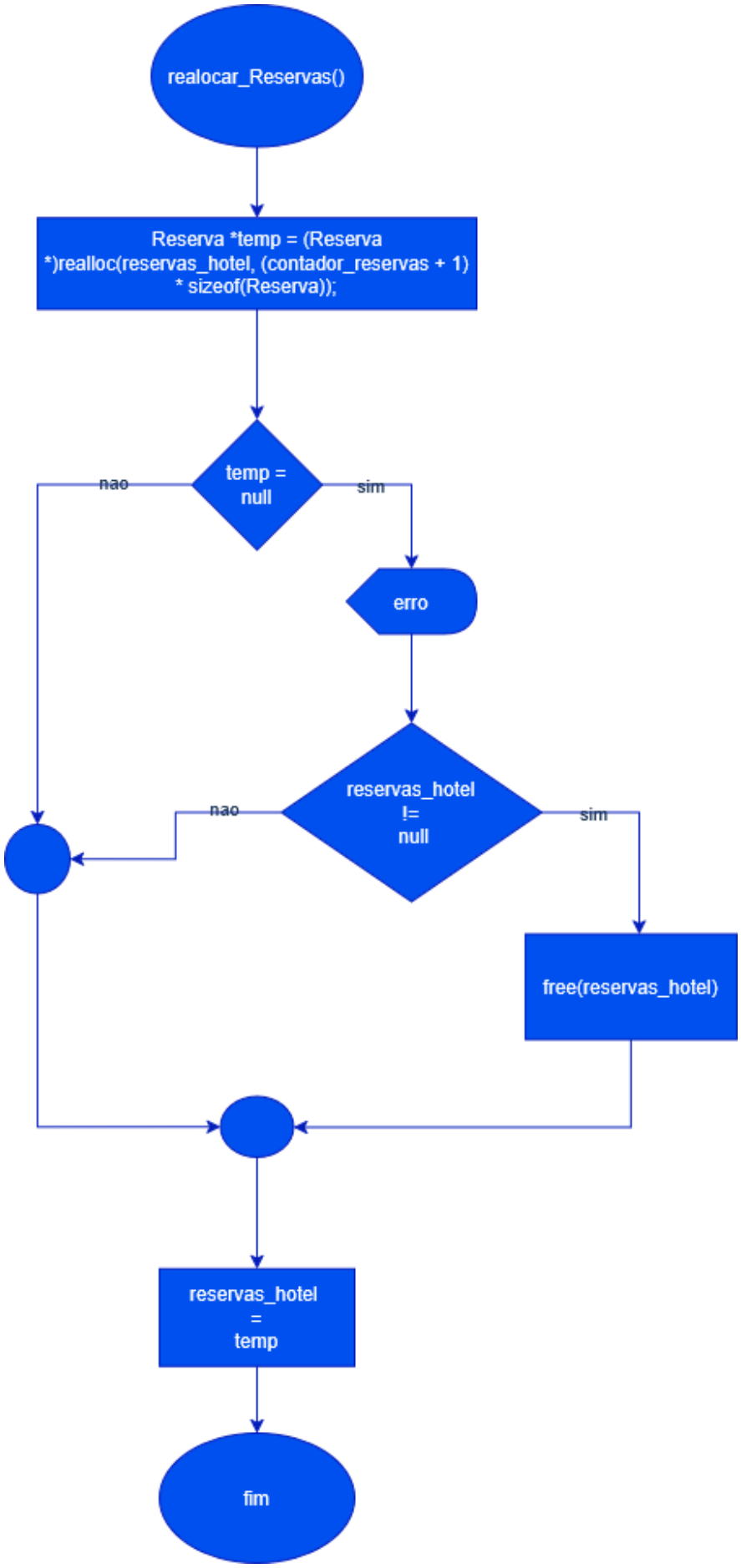

conflito_datas
(const char* data_in1, const char* data_out1, const char* data_in2, const char* data_out2)
long in1 = contar_dias(atoi(data_in1), atoi(data_in1+3), atoi(data_in1+6));
long out1 = contar_dias(atoi(data_out1), atoi(data_out1+3), atoi(data_out1+6));
long in2 = contar_dias(atoi(data_in2), atoi(data_in2+3), atoi(data_in2+6));
long out2 = contar_dias(atoi(data_out2), atoi(data_out2+3), atoi(data_out2+6));
return (in1 < out2 && out1 > in2);
fim

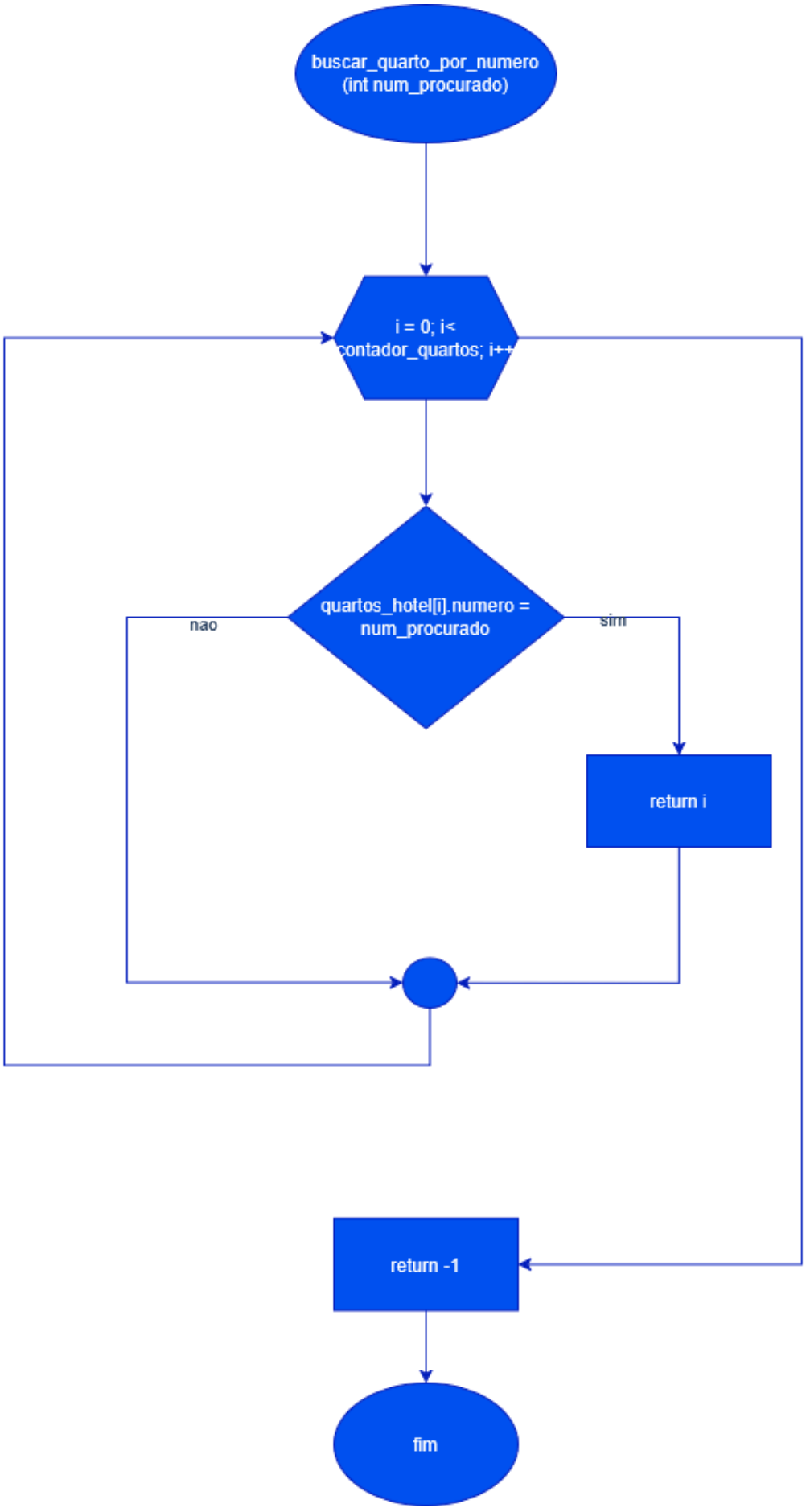

mostrar_historico()
char cpf[TAM_CPF]
Digite o cpf do hospede:
cpf
int index = buscar_hospede_por_cpf(cpf)
index = -1
não
sim
hospede nao encontrado
return
Hospede *h = &hospedes_hotel[index]
historio de reservas de "nome" (CPF ...)
h -> num_reservas_historico = 0
nao
sim
Nenhuma reserva concluida/cancelada ainda
return
i = 0; i < h->num_reservas_historico; i++
int id = h->historico_ids_reservas[i]
Reserva ID
fim

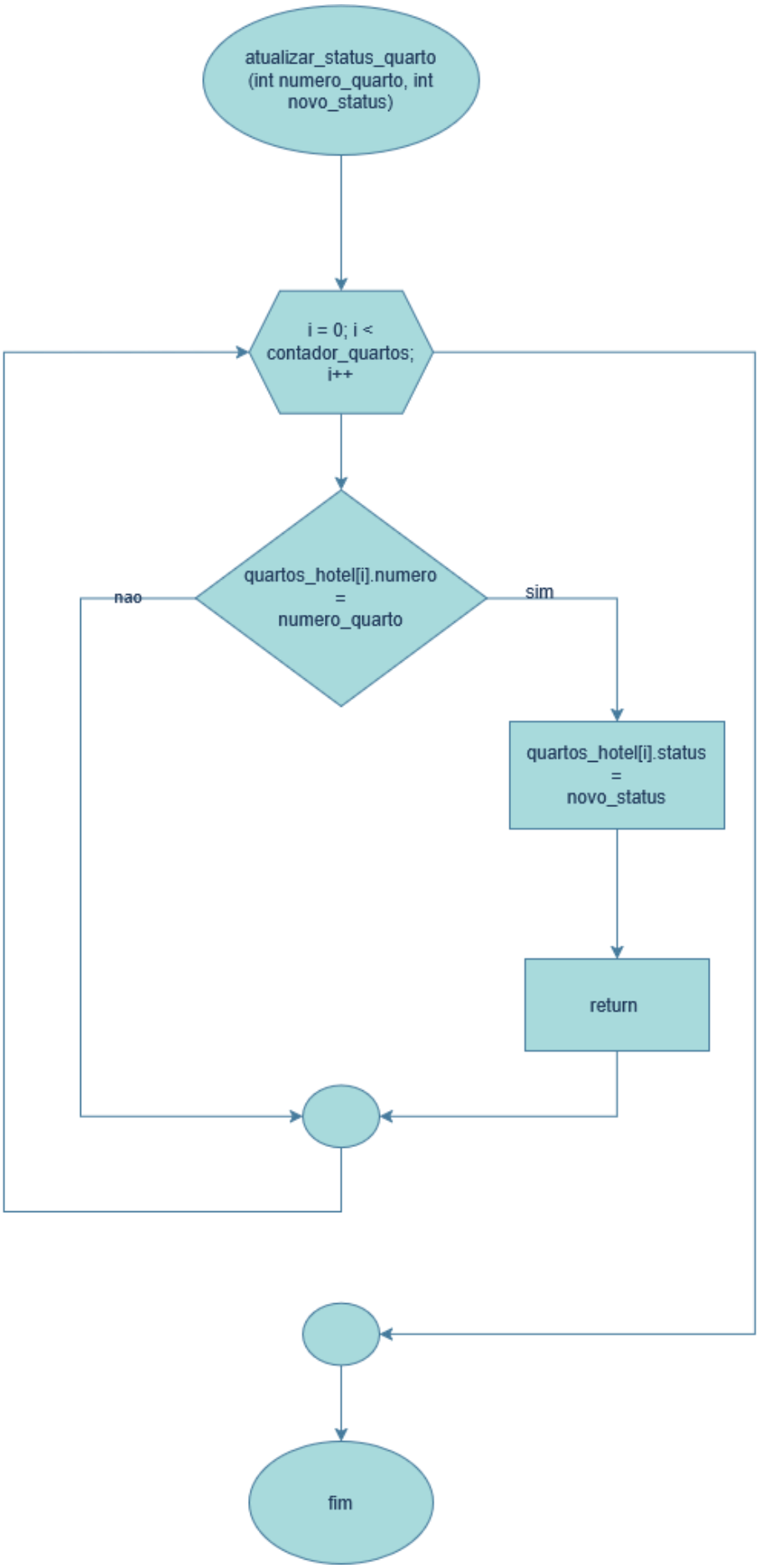


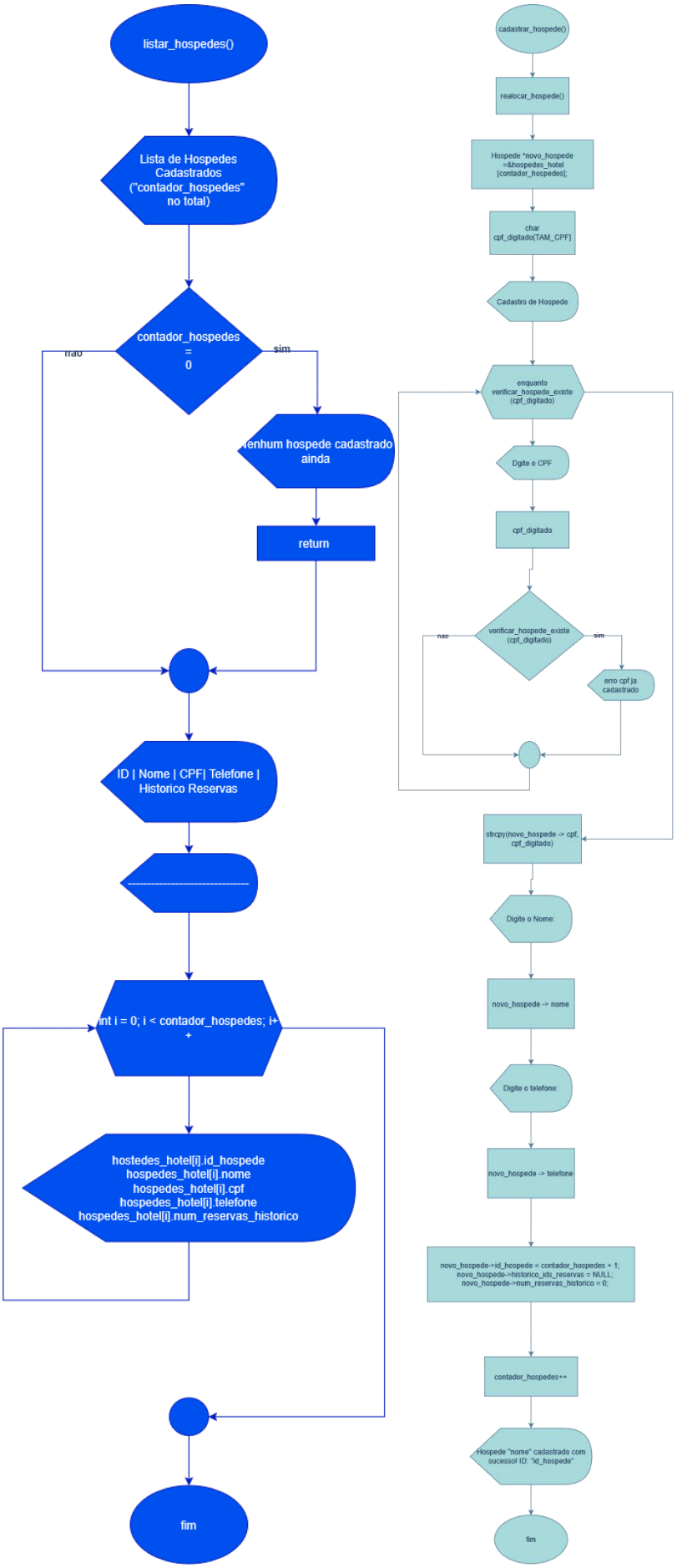

listar_reservas_ativas()
Lista de Reservas ativas ("contador_reservas" no total)
contador_reservas = 0
não
sim
Nenhum reserva ativa
return
ID Res. | Quarto | ID Hosp. | Check-In | Check-Out | Valor Total | Status
int i = 0; i < contador_reservas; i++
reservas_hotel[i].status_reserva = ATIVA
NAO
SIM
reservas_hotel[i].id_reserva, reservas_hotel[i].numero_quarto, reservas_hotel[i].id_hospede, reservas_hotel[i].data_checkin, reservas_hotel[i].data_checkout, reservas_hotel[i].valor_total);
fim

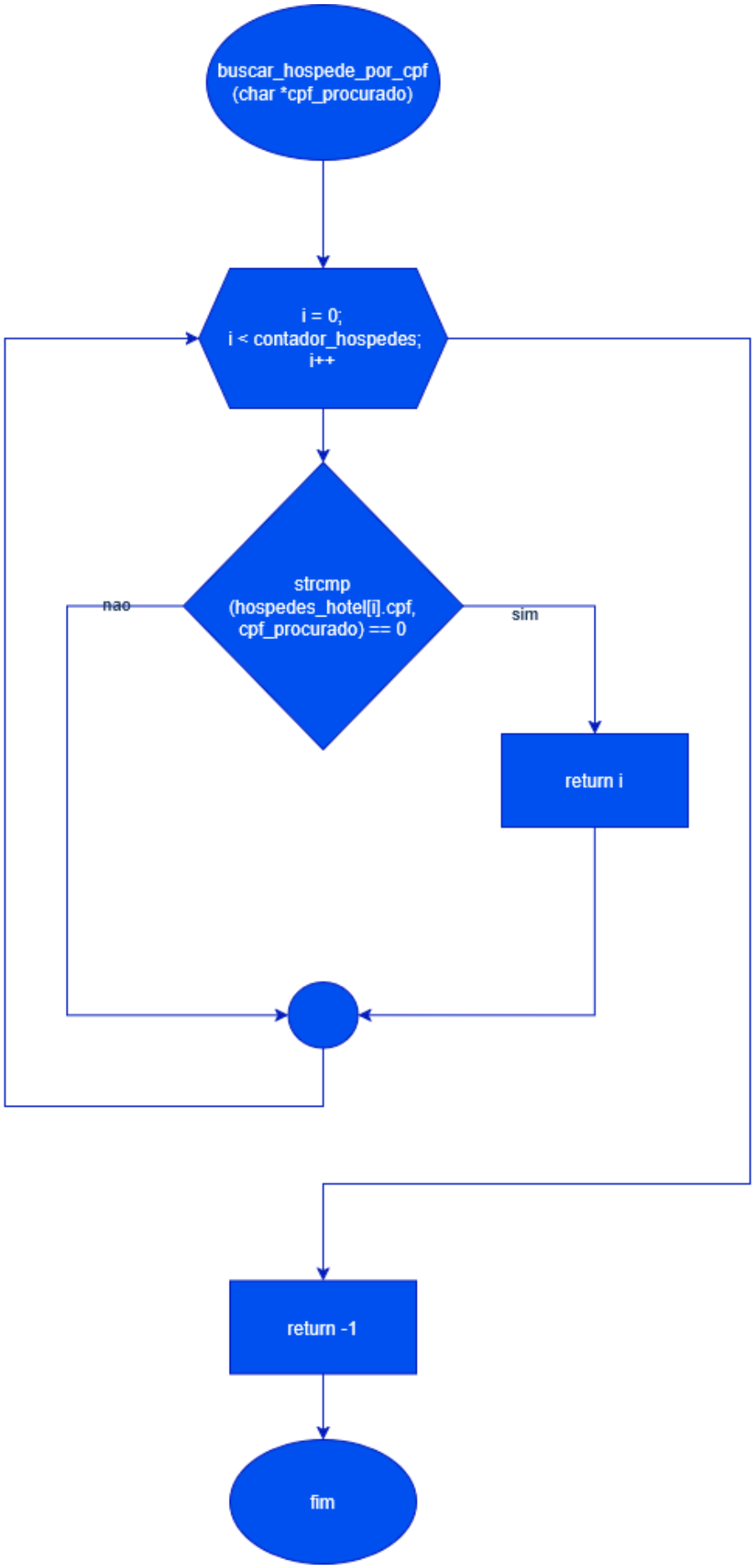


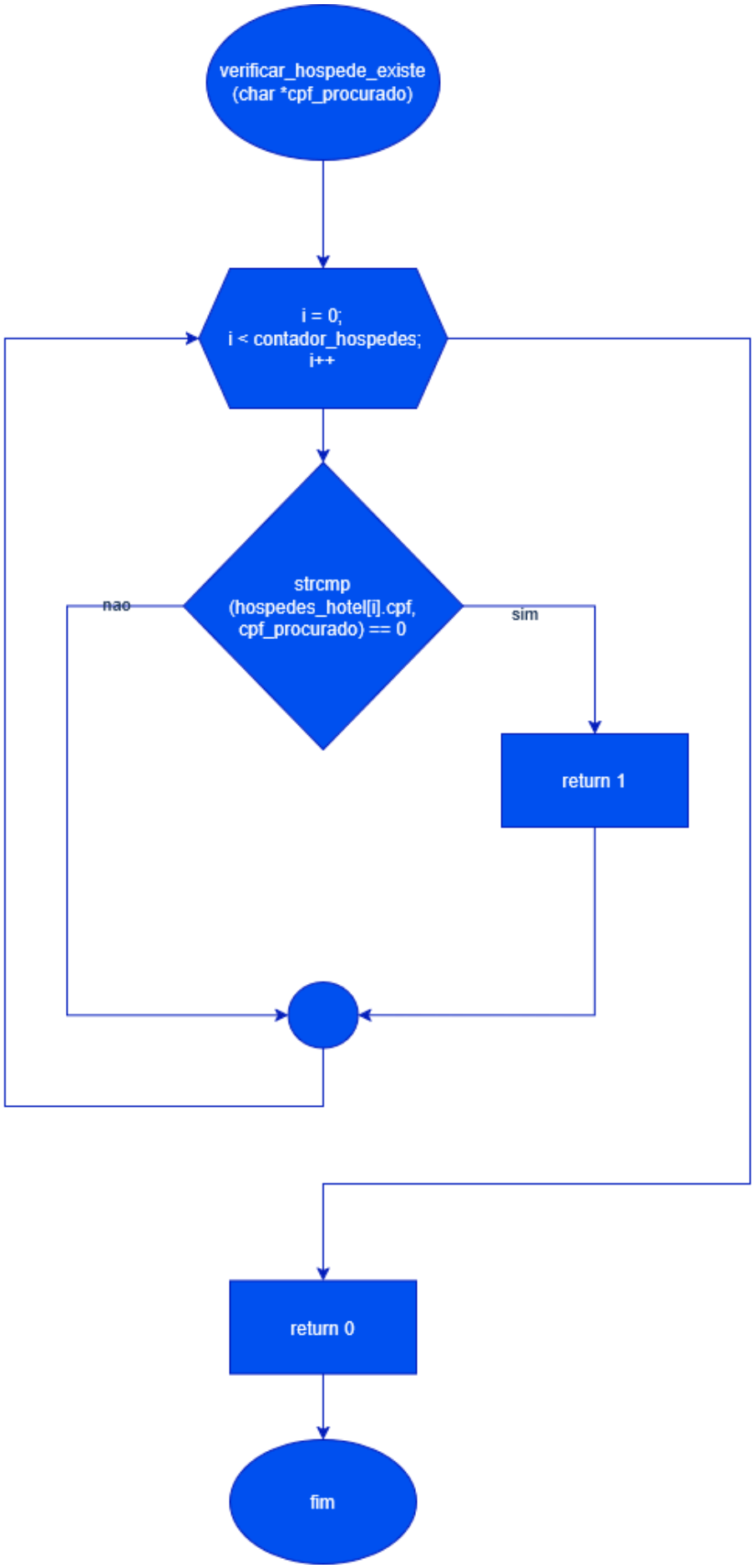

realocar_Reservas()
Reserva *temp = (Reserva *)realloc(reservas_hotel, (contador_reservas + 1) * sizeof(Reserva));
temp = null
nao
sim
erro
reservas_hotel != null
nao
sim
free(reservas_hotel)
reservas_hotel = temp
fim

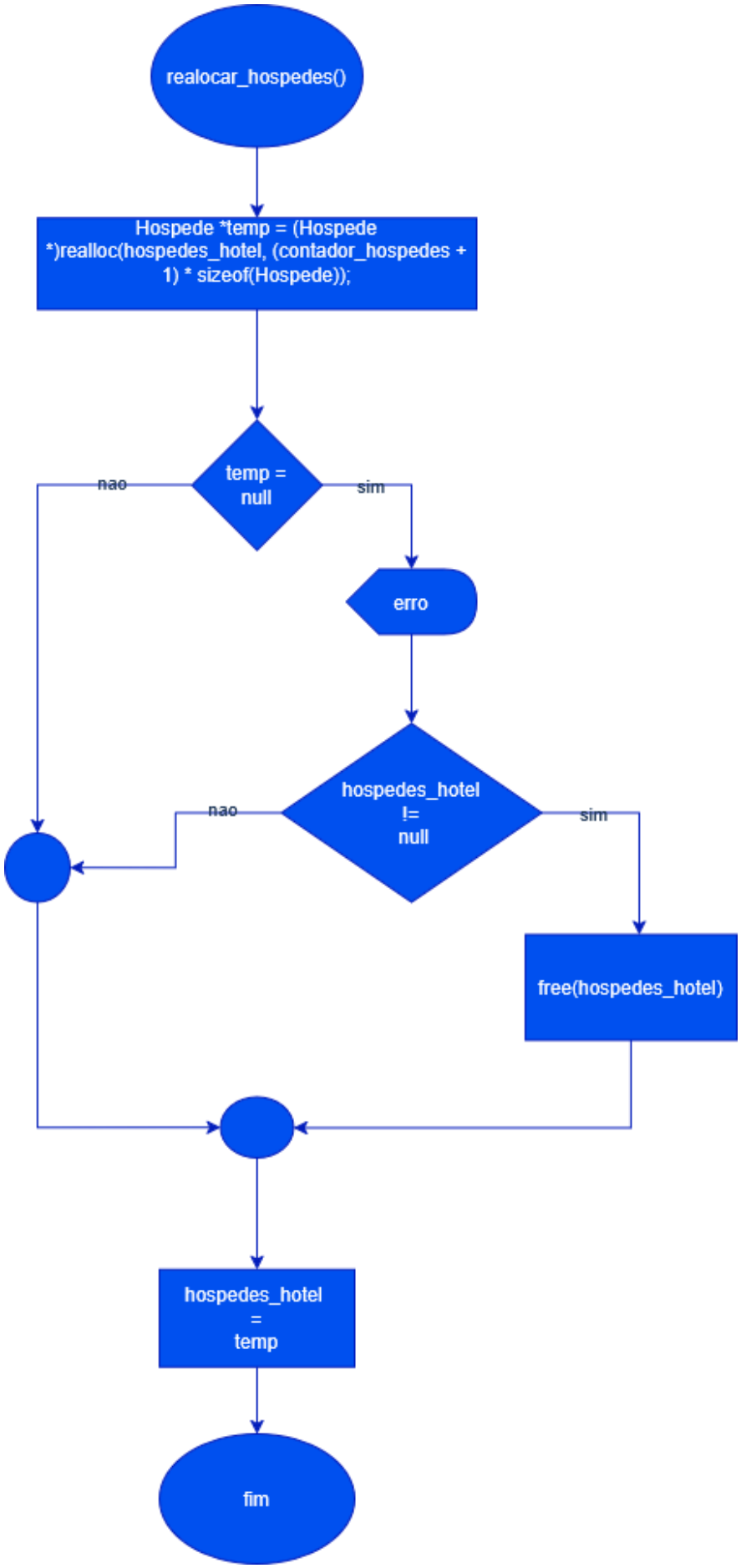

buscar_quarto_por_numero
(int num_procurado)
i = 0; i<
contador_quartos; i++
quartos_hotel[i].numero =
num_procurado
nao
sim
return i
return -1
fim

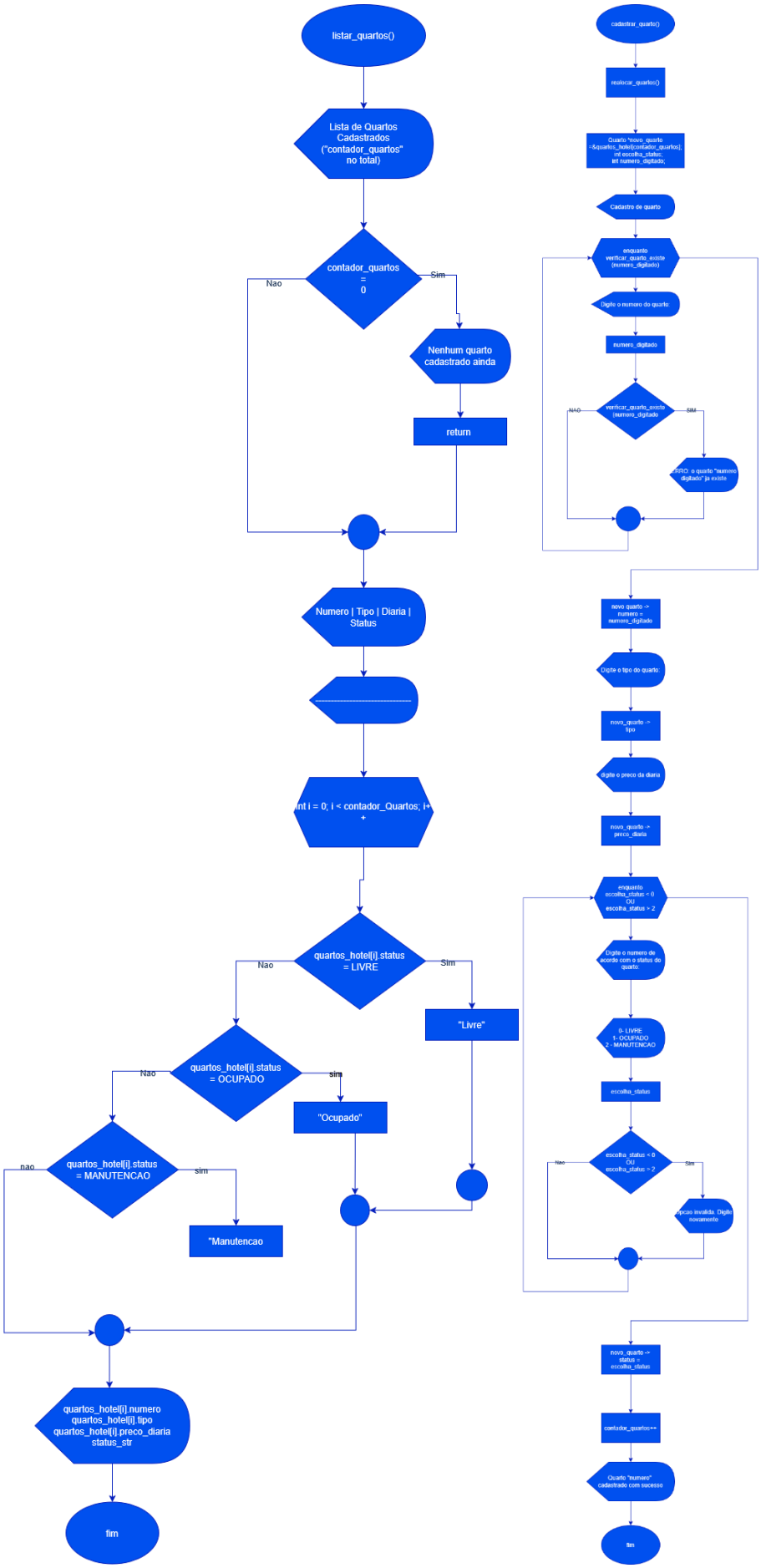

atualizar_status_quarto (int numero_quarto, int novo_status)
i = 0; i < contador_quartos; i++
quartos_hotel[i].numero = numero_quarto
nao
sim
quartos_hotel[i].status = novo_status
return
fim

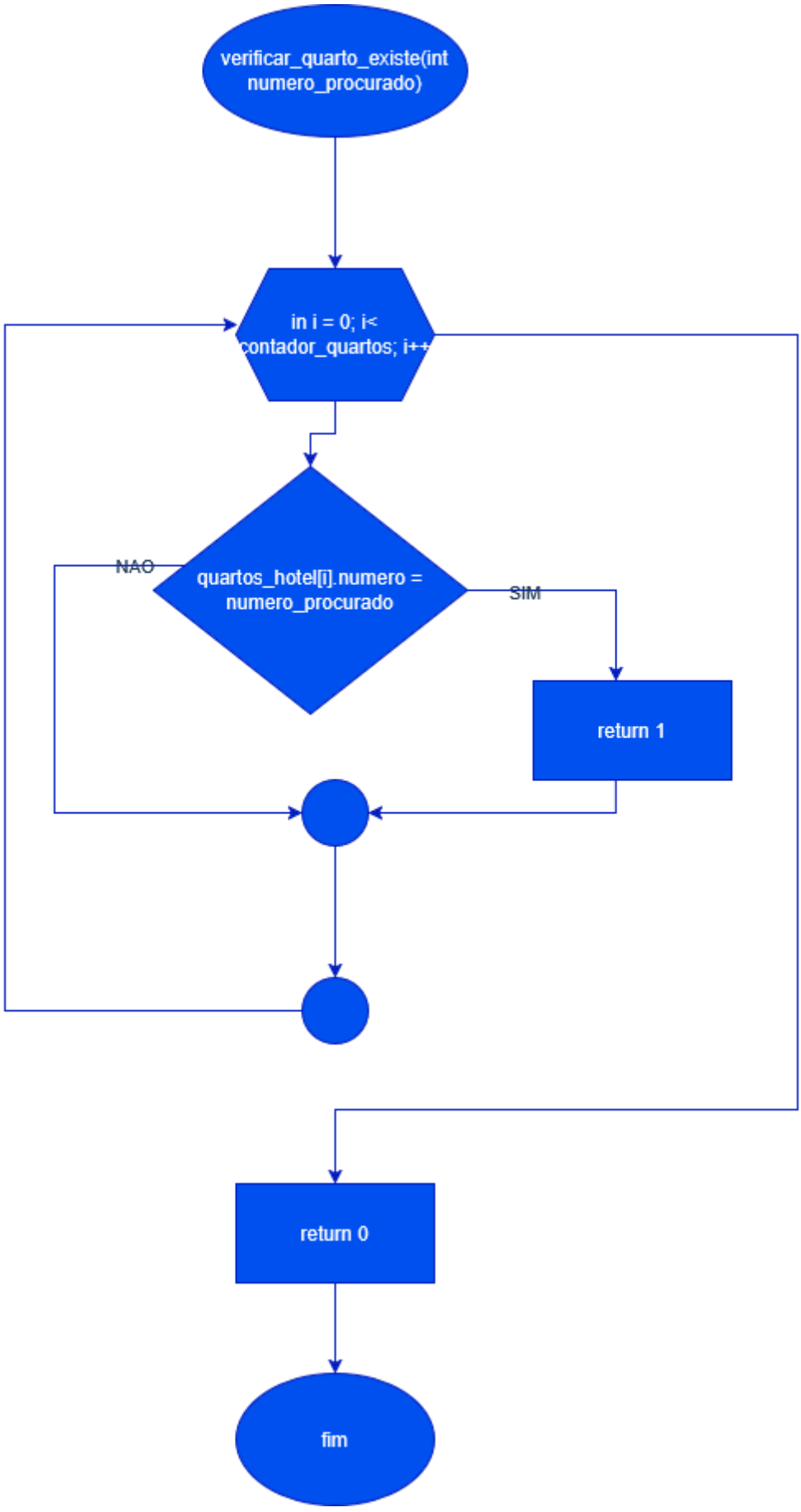

listar_hospedes()
Lista de Hospedes Cadastrados ("contador_hospedes" no total)
contador_hospedes = 0
nao
sim
enhum hospede cadastrado ainda
return
ID | Nome | CPF| Telefone | Historico Reservas
nt i = 0; i < contador_hospedes; i++
hostedes_hotel[i].id_hospede
hospedes_hotel[i].nome
hospedes_hotel[i].cpf
hospedes_hotel[i].telefone
hospedes_hotel[i].num_reservas_historico
fim
cadastrar_hospede()
realocar_hospede()
Hospede *novo_hospede =&hospedes_hotel [contador_hospedes];
char cpf_digitado[TAM_CPF]
Cadastro de Hospede
enquanto verificar_hospede_existe (cpf_digitado)
Dgite o CPF
cpf_digitado
verificar_hospede_existe (cpf_digitado)
nao
sim
erro cpf ja cadastrado
strcpy(novo_hospede -> cpf, cpf_digitado)
Digite o Nome:
novo_hospede -> nome
Digite o telefone:
novo_hospede -> telefone
novo_hospede->id_hospede = contador_hospedes + 1;
novo_hospede->historico_ids_reservas = NULL;
novo_hospede->num_reservas_historico = 0;
contador_hospedes++
Hospede "nome" cadastrado com sucessol ID: "id_hospede"
fim

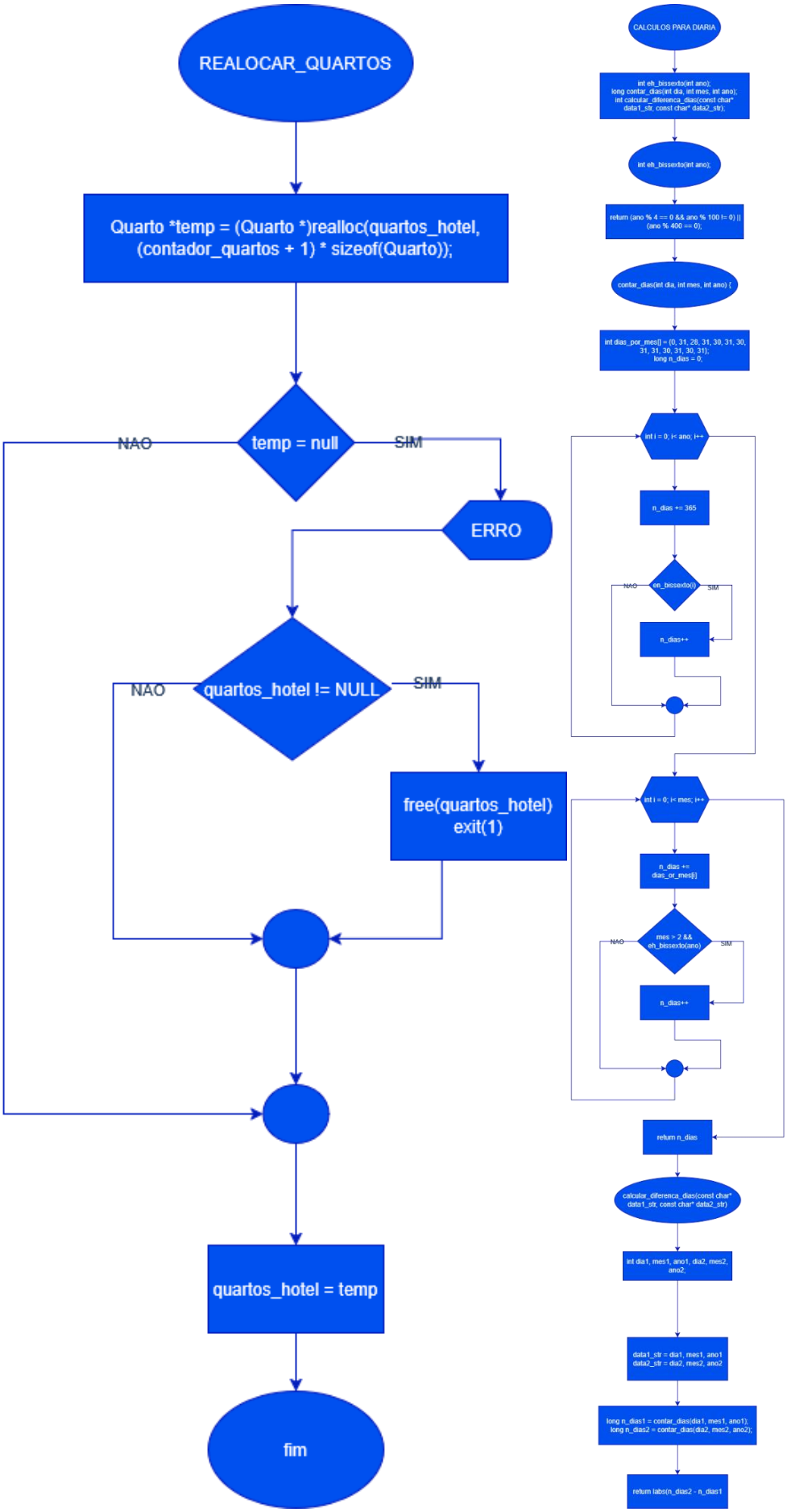

buscar_hospede_por_cpf
(char *cpf_procurado)
i = 0;
i < contador_hospedes;
i++
strcmp
(hospedes_hotel[i].cpf,
cpf_procurado) == 0
nao
sim
return i
return -1
fim


verificar_hospede_existe
(char *cpf_procurado)
i = 0;
i < contador_hospedes;
i++
strcmp
(hospedes_hotel[i].cpf,
cpf_procurado) == 0
nao
sim
return 1
return 0
fim


realocar_hospedes()
Hospede *temp = (Hospede *)realloc(hospedes_hotel, (contador_hospedes + 1) * sizeof(Hospede));
temp = null
nao
sim
erro
hospedes_hotel != null
nao
sim
free(hospedes_hotel)
hospedes_hotel = temp
fim


listar_quartos()
Lista de Quartos Cadastrados ("contador_quartos" no total)
contador_quartos = 0
Nao
Sim
Nenhum quarto cadastrado ainda
return
Numero | Tipo | Diaria | Status
int i = 0; i < contador_Quartos; i++
quartos_hotel[i].status = LIVRE
Nao
Sim
"Livre"
quartos_hotel[i].status = OCUPADO
Nao
sim
"Ocupado"
quartos_hotel[i].status = MANUTENCAO
nao
sim
"Manutencao"
quartos_hotel[i].numero
quartos_hotel[i].tipo
quartos_hotel[i].preco_diaria
status_str
fim
cadastrar_quarto()
realocar_quartos()
Quarto *novo_quarto =&quartos_hotel[contador_quartos];
int escolha_status;
int numero_digitado;
Cadastro de quarto
enquanto verificar_quarto_existe (numero_digitado)
Digite o numero do quarto:
numero_digitado
verificar_quarto_existe (numero_digitado
NAO
SIM
ERRO: o quarto "numero digitado" ja existe
novo_quarto -> numero = numero_digitado
Digite o tipo do quarto:
novo_quarto -> tipo
digite o preco da diaria
novo_quarto -> preco_diaria
enquanto escolha_status < 0 OU escolha_status > 2
Digite o numero de acordo com o status do quarto:
0- LIVRE
1- OCUPADO
2 - MANUTENCAO
escolha_status
escolha_status < 0 OU escolha_status > 2
Nao
Sim
Opcao invalida. Digite novamente
novo_quarto -> status = escolha_status
contador_quartos++
Quarto "numero" cadastrado com sucesso
fim


verificar_quarto_existe(int numero_procurado)
in i = 0; i< contador_quartos; i++
quartos_hotel[i].numero = numero_procurado
NAO
SIM
return 1
return 0
fim


REALOCAR_QUARTOS
Quarto *temp = (Quarto *)realloc(quartos_hotel, (contador_quartos + 1) * sizeof(Quarto));
temp = null
NAO
SIM
ERRO
quartos_hotel != NULL
NAO
SIM
free(quartos_hotel)
exit(1)
quartos_hotel = temp
fim
CALCULOS PARA DIARIA
int eh_bissexto(int ano);
long contar_dias(int dia, int mes, int ano);
int calcular_diferenca_dias(const char* data1_str, const char* data2_str);
int eh_bissexto(int ano);
return (ano % 4 == 0 && ano % 100 != 0) || (ano % 400 == 0);
contar_dias(int dia, int mes, int ano) {
int dias_por_mes[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
long n_dias = 0;
int i = 0; i< ano; i++
n_dias += 365
eh_bissexto()
NAO
SIM
n_dias++
int i = 0; i< mes; i++
n_dias += dias_or_mes[i]
mes > 2 && eh_bissexto(ano)
NAO
SIM
n_dias++
return n_dias
calcular_diferenca_dias(const char* data1_str, const char* data2_str)
int dia1, mes1, ano1, dia2, mes2, ano2;
data1_str = dia1, mes1, ano1
data2_str = dia2, mes2, ano2
long n_dias1 = contar_dias(dia1, mes1, ano1);
long n_dias2 = contar_dias(dia2, mes2, ano2);
return labs(n_dias2 - n_dias1


typedef struct
Hospede
int id_hospede;
char nome[TAM_NOME];
char cpf[TAM_CPF];
char telefone[TAM_TELEFONE];
int *historico_ids_reservas;
int num_reservas_historico;
typedef struct
Quarto
int numero;
char tipo[TAM_TIPO];
float preco_diaria;
int status;
typedef struct
Reservs
int id_reserva;
int numero_quarto;
int id_hospede;
char data_checkin[TAM_DATA];
char data_checkout[TAM_DATA];
int status_reserva;
float valor_total;
VARIÁVEIS GLOBAIS DE
ARMAZENAMENTO
Quarto *quartos_hotel = NULL;
contador_quartos = 0;
Hospede * hospedes_hotel = NULL;
icontador_hospedes = 0;
Reserva *reservas_hotel = NULL;
contador_reservas = 0;