Joaquim && Fernando && André

A10201 && A11199 && A11303

Curso

LESI-PL

Relatório

Algoritmos e estruturas de dados II



Índice

[Identificação 3](#_Toc420702796)

[Introdução 3](#_Toc420702797)

[Estruturas de Dados 4](#_Toc420702798)

[SuperDume 5](#_Toc420702799)

[Cartão 5](#_Toc420702800)

[Cliente 5](#_Toc420702801)

[Artigo 6](#_Toc420702802)

[Compra 6](#_Toc420702803)

[Operações/Opções 9](#_Toc420702804)

[Ficheiro 9](#_Toc420702805)

[Artigos 9](#_Toc420702806)

[SuperDume 9](#_Toc420702807)

[Clientes 10](#_Toc420702808)

[Compra 10](#_Toc420702809)

[Estrutura 17](#_Toc420702810)

[Bibliotecas 18](#_Toc420702811)

Identificação

Este relatório foi realizado no âmbito da disciplina de Algoritmos e Estruturas de Dados II, por Joaquim Cardoso, a10201, por Fernando Correia,a11199 e André Vilaça, a11303. O trabalho escolhido foi: Gestão de Clientes e Artigos do SuperDume.

Introdução

A linguagem de programação C é descrita como sendo constituída através de uma sequência lógica de passos que devem ser executados para resolver o problema que lhe é proposto. Este programa é estruturado usando funções e posteriormente o código texto é traduzido, ou seja, é compilado para a linguagem máquina criando assim um ficheiro executável.

Sendo assim, o objetivo principal deste trabalho é elaborar uma aplicação utilizando a linguagem de programação C, de forma a poder responder às necessidades da gestão de um supermercado. Com este trabalho compreendemos e praticamos os temas abordados nas aulas desta disciplina.

Estruturas de Dados

Neste trabalho vai ser usada a Estrutura Abstrata de Dados “Lista Ligada”, “Struct” e “Tabela de Hash” com armazenamento sequencial usando memória dinâmica.

Uma estrutura de dados é um conjunto de tipos de dados, composta por tipos não estruturados básicos, tipos estruturas ou uma mistura de ambos, e um conjunto de operações definidas sobre esses mesmos tipos de dados. De uma forma mais simples, podemos dizer que uma estrutura de dados é composta por 3 partes:

* Um conjunto de operações;
* Uma estrutura de armazenamento que especifica o tipo de dados e variáveis a utilizar;
* Um conjunto de algoritmos para cada operação.

Para que o resultado final seja alcançado, cada algoritmo procura e modifica a estrutura.

Numa lista ligada, os elementos podem ser definidos como um tipo de dados simples (por exemplo, inteiro, double, float ou carácter) ou como um tipo de dados compostos (por exemplo, struct ou file).

Uma lista pode ser vista como um array, mas sem ter que definir o tamanho, pois cada elemento vai apontar para o próximo.

Uma struct pode ser vista como um aglomerado de tipo de dados.

Uma tabela de hash pode ser vista como um array bidimensional, onde o primeiro campo é a nossa chave e o segundo campo é onde guardamos uma struct ou uma lista ligada.

SuperDume

Neste trabalho cada supermercado é definido por um tipo de dados Superdume composto pela seguinte informação:

* Tabela de Hash de artigos;
* Tabela de Hash de clientes;
* Localização do supermercado;

Cartão

Cada Cartão é definido por um tipo de dados Cartoes, que é uma lista ligada, e é composto pela seguinte informação:

* Compras (Lista ligada);
* Pontos (Inteiro);
* Cliente (Struct);
* Numero Emissão (Inteiro);
* Quantidade de Compras (Inteiro);

Cliente

Cada Cliente é definido por um tipo de dados Cliente, que é uma stuct, e é composto pela seguinte informação:

* Nome (Array de Char);
* Cartão de Cidadão (Array de Char);
* Correio eletrónico (Array de Char);
* Morada (Array de Char);
* Número de telefone (Inteiro)
* Número de identificação fiscal (Long)

Artigo

Cada Artigo é definido por um tipo de dados Artigo, que é uma stuct, e é composto pela seguinte informação:

* Descrição do produto (Array de Char);
* Código do artigo (Inteiro);
* Preço (Float);
* Quantidade Vendida (Inteiro);
* Quantidade em Stock (Inteiro);

Compra

Cada Compra é definido por um tipo de dados Compra, que é uma stuct, e é composto pela seguinte informação:

* Lista de artigos (Lista Ligada);
* Número de artigos comprados (Inteiro);

No caso da utilização de armazenamento sequencial com memória dinâmica, todos os elementos da lista são mantidos num “array” (“array” dinâmico), cujo tamanho não é fixo e vai sendo alterado à medida que o programa vai sendo executado, consoante as alterações que forem necessárias. Quando se escreve o programa, apenas se define uma variável do tipo apontador para um elemento do tipo de dados da lista, assim como uma variável para guardar o seu tamanho. Ou seja, no início apenas se declara um ponteiro que receberá o endereço do primeiro elemento da lista, mas que no início não tem elementos. Assim, a memória necessária para armazenar cada elemento do “array” é reservada à medida que um novo elemento é inserido na lista e libertada à medida que os elementos são removidos. No entanto, as restantes operações a efetuar num “array” dinâmico são efetuadas da mesma forma como é feita sobre um “array” estático.

Uma lista é composta por os seguintes elementos:

* Dados;
* Chave (opcional) − campo pelo qual a lista está ordenada;
* Localização (índice) na lista.

Numa lista A, o elemento de índice 0 (primeira posição, A[0]) não tem antecessor, o elemento de índice N-1 (última posição, A[N-1]) não tem sucessor e o elemento da lista com índice k, tem como sucessor o elemento k+1 e como antecessor o elemento k-1.

O conceito de lista e array dinâmico estão relacionados de um certo modo uma vez que uma lista de tamanho variável pode ser implementada usando um array com tamanha variável à medida que os elementos são inseridos ou removidos.

Ao longo do programa acerca deste trabalho encontramos várias vezes a função calloc, que aloca memória dinâmica e inicializa-a a NULL, que é caracterizada pela seguinte estrutura:

void\* calloc (size\_t num, size\_t size)

Sendo que “num” é o número de elementos que iram ser alocados, e “size” é a nova dimensão que se pretende para o mesmo bloco. Esta função retorna um apontador para o bloco de memória redimensionado.

Também foram usados algoritmos de pesquisa tais como:

* Pesquisa sequencial cuja estrutura é a seguinte:

ListaCartao ProcuraCartaoRetornaLista(unsigned int numeroEmissao)

{

int indice = HashCartao(numeroEmissao), encontrado = 0;

ListaCartao aux = NULL ;

ListaCartao anterior = NULL;

if (indiceLocalSuperDume >= 0)

{

aux= superMercadoLocal[indiceLocalSuperDume]->tabelaCartao[indice];

while (aux != NULL && encontrado == 0)

{

if (aux->cartao.numeroEmissao == numeroEmissao)

encontrado = 1;

else

aux = aux->proximo;

}

}

return aux;

}

Operações/Opções

Ficheiro

No menu principal temos as seguintes opções para ficheiro:

* Carregar
* Guardar

Artigos

No submenu dos artigos existe as seguintes opções:

* Inserir artigo
* Remover artigo
* Consultar artigo
* Alterar artigo
* Verificar se existe artigo
* Remover todos artigos
* Repor Stock
* Atualizar Stock

SuperDume

No submenu do SuperDume existe as seguintes opções:

* Insere Superdume
* Remover Superdume

Clientes

No submenu dos clientes existe as seguintes opções:

* Criar cartão
* Remover cartão
* Realizar compra
* Consultar compras de um determinado Cartão
* Consultar valor total de compras de um determinado Cartão
* Consultar pontos de um determinado Cartão
* Consultar todos os clientes

Compra

No submenu da Compra existe as seguintes opções:

* Adicionar artigo
* Remover artigo
* Mostrar carrinho

1. **Adicionar Artigo (Adicionar um artigo á Hash Table)**

A função utilizada é a seguinte:

**Void InsereArtigo(CriaArtigo())**

Este aglomerado de funções tem como objetivo criar um artigo, onde vai conter um conjunto de informações sobre um determinado artigo, e inseri-lo numa lista ligada dentro da tabela de Hash.

1. **Atualizar Stock (Diminuir a quantidade de artigos em stock)**

**Void ActualizarStockSemParametros()**

Esta função tem como objetivo diminuir a quantidade de artigos em stock.

1. **Repor Stock (Aumentar a quantidade de artigos em stock)**

**Void ReporStock()**

Esta função tem como objetivo Aumentar a quantidade de artigos em stock.

1. **Listar Todos os Artigos (Mostra todos os artigos)**

**Void ImprimeListaTodosArtigos()**

Esta função tem como objetivo mostrar ao utilizador todos os artigos que o SuperDume contém.

1. **Eliminar Artigo (Elimina um artigo)**

**Void EliminaArtigo(int)**

Esta função tem como objetivo eliminar um artigo introduzindo o seu código.

1. **Eliminar Todos os Artigos (Elimina todos artigos)**

**Void EliminaTodosArtigo()**

Esta função tem como objetivo eliminar todos artigos.

1. **Procura Artigo (Procura artigo com determinado código)**

**Artigo ProcuraArtigoRetornaArtigo(int)**

Esta função tem como objetivo verificar a existência de um artigo com um determinado código.

1. **Altera Artigo (Altera artigo campos de um artigo com determinado código)**

**Void AlteraArtigo(ProcuraArtigoRetornaArtigo(int))**

Este aglomerado de funções tem como objetivo procurar um artigo com determinado código, e possibilitar ao utilizador a alteração dos dados do artigo.

1. **Insere Cartão (Insere na tabela de Hash um cartão)**

**Void InsereCartao(CriaCartao())**

Este aglomerado de funções tem como objetivo a criação de um cartão com dados do cliente e dados do SuperDume, e também a sua inserção na tabela de Hash.

1. **Elimina Cartão (Elimina um cartão com determinado número de Emissão)**

**Void EliminaCartao(int)**

Esta função tem como objetivo verificar a eliminar um cartão com determinado número de emissão.

1. **Imprime Todos os Cartões (Mostra ao utilizador todos os cartões)**

**Void ImprimeListaTodosCartoes()**

Esta função tem como objetivo mostrar ao utilizador todos os cartões existentes num determinado SuperDume.

1. **Procura Cartão (Verifica a existência de um cartão)**

**Cartao ProcuraCartaoRetornaCartao(CriaCartao())**

Esta função tem como objetivo verificar a existência de determinado cartão com um determinado número de emissão.

1. **Insere Compra (Insere compra em cartão)**

**Void InsereCompra(int, CriaCompra())**

Este aglomerado de funções tem como objetivo a criação de uma compra, e a inserção da cuja num cartão com um determinado número de emissão.

1. **Imprime Compras de Cartão (Mostra todas as compras de um cartão)**

**Void ImprimeListaCartao(int)**

Esta função tem como objetivo verificar a mostrar ao utilizador todas as compras realizadas de determinado cartão com um determinado número de emissão.

1. **Quanto Gastei (Somatório das compras de um cartão)**

**Void QuantoGastei(int)**

Esta função tem como objetivo verificar a informar o utilizador do valor gasto em todas as compras realizadas de determinado cartão com um determinado número de emissão.

1. **Quantos Ponto (Pontos de um cartão)**

**Int RetornaPontos(int)**

Esta função tem como objetivo verificar a informar o utilizador dos pontos de um determinado cartão com um determinado número de emissão.

1. **Imprime Todos os Cliente (Mostra ao utilizador todos os clientes)**

**Void ImprimeListaTodosClientes()**

Esta função tem como objetivo mostrar ao utilizador todos os clientes existentes num determinado SuperDume.

1. **Lê de ficheiro Artigos (Lê de um ficheiro os artigos)**

**Void LeFicheiroArtigo(char[])**

Esta função tem como objetivo carregar dados dos artigos de um ficheiro para a estrutura lógica de um SuperDume.

1. **Lê de ficheiro Cartões (Lê de um ficheiro os cartões)**

**Void LeFicheiroCartoes(char[])**

Esta função tem como objetivo carregar dados dos cartões de um ficheiro para a estrutura lógica de um SuperDume.

1. **Escreve em ficheiro Cartões (Escreve num ficheiro os cartões)**

**Void EscreveFicheiroArtigo(char[])**

Esta função tem como objetivo recolher todos os artigos existentes no SuperDume e escreve-los em ficheiro que por definição é “artigos.txt”.

1. **Escreve em ficheiro Cartões (Escreve num ficheiro os cartões)**

**Void EscreveFicheiroCartoes(char[])**

Esta função tem como objetivo recolher todos os cartões existentes no SuperDume e escreve-los em ficheiro que por definição é “cartoes.txt”.

1. **Main (Programa)**

A função void main(int argc, char\*\* argv), função principal, contêm o menu principal onde abrange as seguintes opções:

* Guardar e Carregar em ficheiro
* Gerir SuperDume
* Gerir Clientes
* Gerir Artigos
* Sair

Estas opções já foram referidas e explicadas a cima. Para que o relatório não fosse demasiado ”chato” e extenso, muitos métodos usados sem que o utilizador se aperceba dessa utilização não foram demonstrados no presente relatório.

Tais como:

* Alocação de memória
* Inicialização
* Calculo
* Menus e Submenus
* Fórmula de Hash
* Desenhos
* Informação Pedida ao Sistema

Estrutura

No início do programa foi criada uma estrutura para o SUPERDUME onde contêm todas as informações acerca deste.

#define \_CRT\_SECURE\_NO\_WARNINGS

Esta linha de código é usada para prevenir algumas falhas no compilador do Visual Studio. Sem isto o programa obriga a usar funções acabadas em \_s, denominadas como seguras.



É utilizado alguns defines para tornar o código um pouco mais dinâmico. E é utilizada uma macro, apenas para fazer uma operação de troca sem ter que criar uma variável auxiliar.

Bibliotecas

Neste trabalho foram usadas sete bibliotecas:

* **stdio.h**
* **stdlib.h**
* **string.h**
* **windows.h**
* **time.h**
* **locale.h**
* **Header.h**

**Header.h** é a minha biblioteca que importa a estrutura de dados.

**Locale.h** é uma biblioteca da linguagem C que suporta a “escrita” numa determinada língua, símbolos, acentos, etc.

**Time.h** é uma biblioteca da linguagem C que contém funções para obter e manipular informações de datas e tempo.

**Stdio.h** é uma biblioteca padrão da linguagem C. Possui definições relativas às operações de entrada/saída, tais como a leitura de dados a partir do teclado e a exibição de informações no ecrã.

**Stdlib.h** é uma biblioteca da linguagem C e possui funções que envolvem a alocação de memória, controlo de processos, conversões, pesquisa entre outras coisas.

**String.h** fornece funções e definições da biblioteca padrão da linguagem C para a manipulação de arrays de caracteres e regiões de memória.

**Windows.h** é uma biblioteca que pode ser usada em linguagem C, e contém todas as funções no Windows API, e todos os tipos de dados usados pelas funções.

 

