CIÊNCIA DA COMPUTAÇÃO

Programação para Interfaceamento de Hardware e Software (9792)

ATIVIDADE 1-2

Professor: Ronaldo Augusto de Lara Gonçalves

Discentes

RA	NOME
130099	GUSTAVO HENRIQUE TRASSI GANAZA
129182	YOSHIYUKI FUGIE

Maringá

2025

Conteúdo

1 Visão Geral do Sistema

Este relatório detalha a implementação do sistema de gerenciamento de produtos em Assembly 32 bits. O sistema utiliza uma lista encadeada para armazenar os produtos, com operações para adicionar, buscar, remover, atualizar, gerar relatórios e realizar consultas financeiras. O código é organizado em funções que manipulam a lista e interagem com o usuário via terminal.

A entrega inclui um arquivo makefile para compilar o código-fonte. O makefile utiliza *gcc -m32* para gerar código 32 bits, portanto é necessário ter instalado o pacote *gcc-multilib* no sistema (em distribuições baseadas no Debian/Ubuntu: *sudo apt install gcc-multilib*). Para compilar o código, execute o comando *make* no terminal. Para rodar o programa, execute *./supermercado*.

2 Estrutura de Dados

A estrutura de um produto é definida com os seguintes campos:

- **next**: ponteiro para o próximo nó (4 bytes)
- **tipo**: inteiro (4 bytes)
- **quantidade**: inteiro (4 bytes)
- **valor_compra**: inteiro (em centavos, 4 bytes)
- **valor_venda**: inteiro (em centavos, 4 bytes)
- **nome**: string (50 bytes)
- **lote**: string (20 bytes)
- **dia**: inteiro (4 bytes)
- **mês**: inteiro (4 bytes)
- **ano**: inteiro (4 bytes)
- **fornecedor**: string (50 bytes)

O tamanho total da estrutura é 152 bytes, sendo 4 bytes para o ponteiro e 148 bytes para os dados.

A lista é acessada através de um ponteiro global `head` que aponta para o primeiro nó.

3 Registradores principais

EAX Acumulador / retorno de função / aritmética.

EBX Preservado por chamadas (*callee-saved*), guarda ponteiros de nó, handle de arquivo, buffer, etc.

ECX Contador em loops, ou divisor em `divl`.

EDX Resto de divisão / auxiliares em formatação de moeda.

ESI, EDI Índices / preservados, usados para percorrer listas ou passar handles.

EBP, ESP Gerenciam o frame de pilha de cada função.

4 Adição de Produto

Função: `add_product_interactive` Esta função interage com o usuário para coletar os dados do produto, aloca memória para o novo nó e o insere na lista de forma ordenada pelo nome. **Registradores principais:**

- **EAX**: usado para armazenar o endereço do novo nó alocado.
- **EBX**: mantém o endereço do novo nó durante a coleta de dados.

Fluxo:

1. Aloca memória para o novo produto (152 bytes) usando `malloc`.
2. Inicializa o ponteiro `next` do novo nó como NULL.
3. Lê os campos do produto (nome, lote, tipo, dia, mês, ano, fornecedor, quantidade, valor de compra, valor de venda) usando funções auxiliares (`read_string_with_prompt`, `printf`, `scanf`).
4. Chama `insert_sorted` para inserir o nó na lista mantendo a ordem alfabética pelo nome.

Função `insert_sorted`:

- **EBX**: contém o endereço do novo nó.
- **EDI**: ponteiro atual na lista (começa em `head`).
- **ESI**: ponteiro anterior (para inserção).
- Compara o nome do novo nó com os nomes dos nós existentes usando `strcmp`.
- Insere o nó na posição correta:
  - Se a lista estiver vazia, `head` aponta para o novo nó.
  - Se o novo nó deve ser o primeiro, atualiza `head` e o ponteiro `next` do novo nó.
  - Caso contrário, insere no meio ou no final.

5 Busca de Produto

Função: `search_product_interactive`

Esta função lê um nome do usuário e percorre a lista imprimindo todos os produtos com nomes que coincidem.

Registradores principais:

- **EBX**: ponteiro para o nó atual durante a busca.
- **EAX**: usado para chamadas de função e comparações.

Fluxo:

1. Lê o nome a ser buscado.
2. Chama `search_product` com o nome.
3. Em `search_product`:
   (a) Inicia no `head`.
   (b) Para cada nó, compara o nome com o nome buscado usando `strcmp`.
   (c) Se coincidir, imprime o produto usando `print_product`.
   (d) Avança para o próximo nó.

6 Remoção de Produto

Função: `remove_product_interactive`

Remove um produto específico baseado no nome e lote.

Registradores principais:

- **EBX**: ponteiro para o nó atual.
- **ESI**: ponteiro para o nó anterior (para ajustar os ponteiros).

Fluxo:

1. Lê o nome e o lote do produto a ser removido.
2. Percorre a lista:
   (a) Compara o nome e o lote do nó atual com os valores lidos.
   (b) Quando encontra, remove o nó:
   - Se for o primeiro nó, atualiza `head`.
   - Se estiver no meio, o nó anterior (`ESI`) aponta para o próximo do nó removido.

(c) Libera a memória do nó com `free`.

7 Atualização de Produto

A função `update_product_interactive`:

1. Lê nome e lote para localizar o nó.
2. Pergunta campo a atualizar (quantidade ou valor de venda).
3. Usa `scanf` + `clear_input_buffer` para ler o inteiro e grava no offset correspondente (8 para quantidade, 16 para venda).
4. Confirma sucesso ou falha via mensagens.

8 Consultas Financeiras

Função: `finance_menu` Apresenta um submenu para o usuário escolher entre:

1. Total gasto em compras.
2. Total estimado de vendas.
3. Lucro total estimado.
4. Capital perdido (produtos vencidos).

Fluxo:

- `total_compra`: Percorre a lista somando `quantidade * valor_compra` para cada produto.
- `total_venda`: Percorre a lista somando `quantidade * valor_venda` para cada produto.
- `lucro_total`: Calcula `total_venda - total_compra`.
- `capital_perdido`:
  1. Pede a data atual ao usuário.
  2. Para cada produto, compara a data de validade com a data atual usando `compare_dates`.
  3. Se a validade for anterior à data atual, soma `quantidade * valor_compra` ao total perdido.
- Os resultados são impressos em formato de moeda (reais.centavos).

Registradores nas funções de soma:

- **EBX**: ponteiro para o nó atual.

- **ESI**: acumulador do total.

9 Persistência em Disco

Gravação (`save_list`)

1. Abre `produtos.bin` em modo `wb`.
2. Para cada nó: copia `dados_size` bytes (offset 4) para um buffer via `memcpy`, chama `fwrite`.
3. Fecha arquivo (`fclose`).

Carregamento (`load_list`)

1. Abre `produtos.bin` em modo `rb`.
2. Enquanto `fread` retornar `dados_size`: aloca nó (`malloc`), copia buffer para `novo+4`, chama `insert_sorted`.
3. Fecha arquivo.

10 Geração de Relatórios

Função: `generate_report`

Gera um relatório em arquivo texto (`relatorio.txt`) com todos os produtos. O usuário pode escolher a ordenação: por nome (padrão), por quantidade ou por data de validade (mais antiga primeiro).

Fluxo:

1. Abre o arquivo para escrita.
2. Pergunta ao usuário o critério de ordenação.
3. Dependendo da escolha:
   - **Ordenação por nome**: percorre a lista normalmente (já está ordenada por nome).
   - **Ordenação por quantidade ou data**:
     (a) Conta o número de nós.
     (b) Aloca um array de ponteiros para nós.
     (c) Preenche o array com os ponteiros dos nós.
     (d) Ordena o array (usando `sort_by_quantity` ou `sort_by_date`).

(e) Percorre o array ordenado, escrevendo cada produto no arquivo.

(f) Libera o array.

4. Para cada produto, escreve todos os campos no arquivo usando `fprintf`.
5. Fecha o arquivo.

Ordenação:

- `sort_by_quantity`: Usa bubble sort no array de ponteiros, comparando o campo `quantidade`.
- `sort_by_date`: Usa bubble sort comparando as datas. A função de comparação `compare_nodes_by_date` extrai dia, mês e ano de cada nó e compara (ano, depois mês, depois dia).

Registradores nas funções de ordenação:

- **ESI**: endereço do array.
- **ECX**: número de elementos.
- **EDI**, **EBX**: índices para loops.
- **EAX**, **ECX**: ponteiros para nós durante a comparação.

11 Limitações e Pontos de Melhoria

Embora funcional, o código ainda apresenta algumas fragilidades:

- **Validação de entrada ausente:** todas as chamadas a `scanf` ou `read_int` assumem que o usuário fornecerá dados válidos. É possível, por exemplo, ler dia, mês ou ano negativos, ou valores fora de faixa.
- **Sem tratamento de erro em tempo de execução:** não há verificação do retorno de `fread`, `fwrite` ou `malloc` além do teste simples de ponteiro nulo. Em caso de falha parcial de I/O, o relatório ou a lista podem ficar corrompidos.
- **Possíveis vazamentos de memória:** em `generate_report`, se a alocação de `node_array` falhar, a função retorna sem liberar memória prévia. Similarmente, em rotinas CRUD, `malloc` bem-sucedido que não é seguido de `insert_sorted` em caso de erro não é sempre liberado.
- **Limite fixo de buffers:** funções como `read_string_with_prompt` usam `fgets(buffer,50)` sem checar se o usuário excedeu 49 caracteres, o que pode resultar em truncamento silencioso.
- **Overflow aritmético:** ao multiplicar quantidade por valor em centavos, não há verificação de estouro de 32 bits em `imull`, podendo levar a resultados incorretos em consultas financeiras.