UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

ARTHUR HENRIQUE BANDO UEDA RA:129406
GUSTAVO GLATZ ALVES RA:128592
DIOGO FELIPE SOARES DA SILVA RA:124771

Controle de Estoque de Supermercado em Assembly PARTE 2

Maringá
2025

Introdução

O presente trabalho tem como objetivo desenvolver um CRUD de um sistema de controle de estoque de um Supermercado. Esse sistema será responsável por executar operações básicas sobre os produtos existentes nesse Supermercado. As operações que foram implementadas do trabalho foram as seguintes:

- Inserção de produto
- Remoção de produto
- Atualização de produto
- Consulta de produto
- Consulta financeira
- Gravação de registros
- Recuperação de registros

A linguagem que foi utilizada para a realização deste trabalho foi Assembly GAS com sintaxe AT&T. É uma ferramenta poderosa para programação de baixo nível, otimização e desenvolvimento de sistemas mais robustos. A ordem de dos operandos distingue-se de outras sintaxes, como a da Intel, pois é da forma: "*origem, destino"*. As instruções carregam um sufixo que corresponde ao tamanho do operando, como, por exemplo, ***movl*** para 32 bits. No contexto deste trabalho, a totalidade das operações será de 32 bits.

Abaixo estão os comandos para executar o programa. Note que foi usado o ***gcc*** para fazer a linkagem e compilação do código objeto.

1) Compilação: ***as -32 -g trabalho2.s -o trabalho2.o***
2) Linkagem: ***gcc -m32 trabalho2.o -g***
3) Execução: ***./a.out***

A seguir será documentado as mudanças feitas do trabalho original, conforme requisitado pela especificação do trabalho. As modificações feitas foram:

Remoção de produto por nome ou por estar fora da validade

Quando o usuário seleciona a opção de remover um produto do menu principal (opção 2), um novo menu será mostrado para ele, sendo possível escolher a opção de remover por nome do produto ou remover todos os produtos que estão com a validade vencida.

Como a remoção por nome já foi explicada no relatório anterior, vamos explicar a nova funcionalidade de remover por validade.

As funções criadas para atender esse requisito foram:

get_current_date: Função responsável por obter a data atual (dia, mês e ano) e armazená-las nas variáveis: current_day, current_month e current_year, respectivamente. Para fazer esse implementação foi necessário fazer o uso de funções da biblioteca ***libc***, sendo elas: ***time(NULL)*** e ***localtime(&timestamp)***.

remove_product_by_date: Essa é a função responsável por fazer as comparações necessárias para remover o produto caso ele esteja vencido. Ela faz isso, primeiramente, comparando o valor do ano de validade do produto com o ano atual. Abaixo está um esquema de como essa comparação ocorre:

- (Ano de Validade) < (Ano Atual): Produto vencido e, portanto, é removido da lista ligada;
- (Ano de Validade) > (Ano Atual): Produto não vencido, isto é, o produto ainda está bom para uso ou consumo. Assim, apenas avançamos para o próximo nó;
- (Ano de Validade) = (Ano Atual): Se os anos forem iguais, comparamos o Mês de Validade com o Mês Atual na mesma lógica.

Aqui está o trecho de código que mostra essa comparação:

```
        movl    EXPIRATION_YEAR(%edi), %eax
        cmpl    current_year, %eax
        jl      remove_node_by_date
        jg      node_not_expired

        # same year, compare month
        movl    EXPIRATION_MONTH(%edi), %eax
        cmpl    current_month, %eax
        jl      remove_node_by_date
        jg      node_not_expired

        # same month, compare day
        movl    EXPIRATION_DAY(%edi), %eax
        cmpl    current_day, %eax
        jl      remove_node_by_date

        movl    %edi, %esi
        movl    NEXT_REF(%edi), %edi
        jmp     start_remove_by_date_loop

    node_not_expired:
        # product not expired. Update node
        movl    %edi, %esi
        movl    NEXT_REF(%edi), %edi
        jmp     start_remove_by_date_loop
```

remove_product_by_name: É a mesma função do relatório anterior. Ela só foi separada para melhorar a delegação de tarefas no trabalho e, também, sua legibilidade.

Aqui está o menu de opções para a remoção do produto:

```
Escolha uma opção: 2

----LISTA ORDENADA----
Arroz -> Batata -> Camarão ->

-----Remover Produto-----
1) Remover por nome de produto
2) Remover todos os produtos fora da validade
```

Um exemplo de execução é mostrado abaixo, sendo que inicialmente o estoque possui 3 produtos. Considere que “Arroz” e “Camarão” estão vencidos. Dessa forma,

ao escolher a opção de remover todos os produtos fora de validade, a lista resultado deverá mostrar apenas o produto “Batata”.

Lista inicial:

```
----LISTA ORDENADA----
Arroz -> Batata -> Camarão ->
```

Lista final:

```
Foi excluído 2 produto(s) vencido(s) no estoque

----LISTA ORDENADA----
Batata ->
```

Relatório de Registros pode ser ordenado por nome, data de validade e quantidade em estoque

É importante mencionar que o nome da função de Relatório de Registros foi alterada do trabalho 1 para o trabalho 2, sendo o nome inicial ***records_report*** e seu nome atual ***registration_report***. Essa mudança foi simplesmente baseada na tradução do inglês livre.

A funcionalidade de Relatório de Registro agora possui um menu de opção para o usuário escolher como ele deseja visualizar esse relatório. Há a opção de ser ordenado por nome, por data de validade e por quantidade em estoque.

A opção de ordenar por nome já está pronta, visto que os produtos já são inseridos ordenados por ordem alfabética. Assim, foi necessário a implementação dos outros dois casos. Para isso, tem-se as seguintes funções principais de ordenação:

registration_report_by_date: Função responsável preencher vetor auxiliar (chamada para a função ***populate_temp_array***), ordenar vetor auxiliar por data de

validade (chamada para a função ***bubble_sort_by_date***) e exibir os registros já ordenados para o usuário (chamada para a função ***print_report_loop***);

bubble_sort_by_date: Ordenação bubble sort que ocorre no vetor auxiliar (***temp_array***). Esse vetor auxiliar é um vetor somente de ponteiros. Sendo que cada ponteiro aponta para um produto na lista original. Esse tem um espaço de 4000 bytes, isto é, é capaz de armazenar 1000 produtos. Além disso, esse método de ordenação compara dois produtos adjacentes e verifica quem é o maior, trocando os dois de posição de necessário.

registration_report_by_quantity: Função responsável preencher vetor auxiliar (chamada para a função ***populate_temp_array***), ordenar vetor auxiliar por quantidade em estoque (chamada para a função ***bubble_sort_by_quantity***) e exibir os registros já ordenados para o usuário (chamada para a função ***print_report_loop***);

bubble_sort_by_quantity: É a mesma ordenação bubble sort. A diferença é que compara a quantidade em estoque entre os produtos adjacentes e não a data de validade.

Aqui está um exemplo da execução desta nova funcionalidade:

Lista dos produtos:

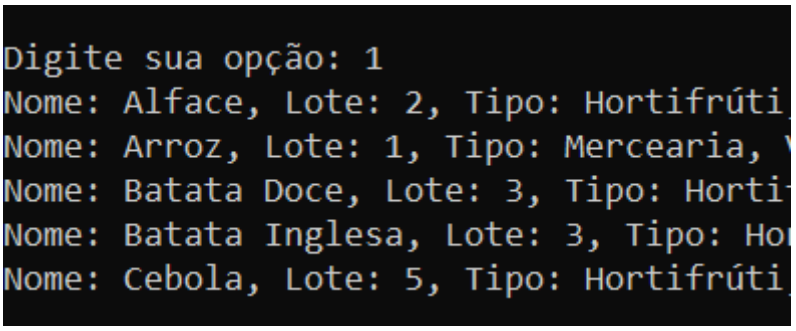
```
----LISTA ORDENADA----
Alface -> Arroz -> Batata Doce -> Batata Inglesa -> Cebola ->
```

Menu de opções:

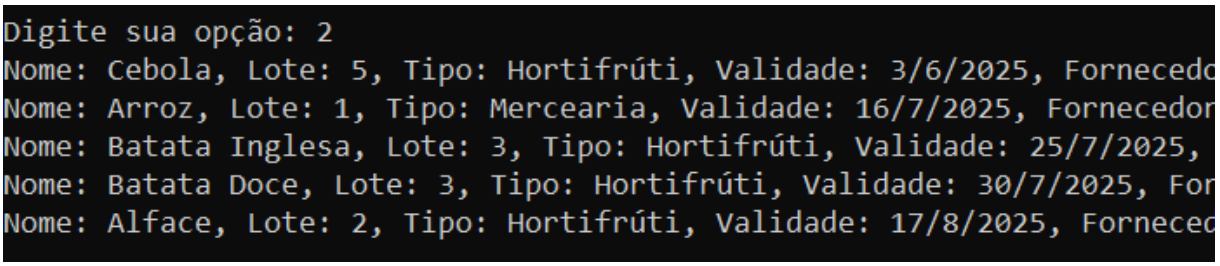
```
-----RELATÓRIO DE REGISTROS-----

Deseja ver o relatório ordenado por:
1) Nome
2) Data de validade
3) Quantidade em estoque
```

Ordenados por nome (opção 1):

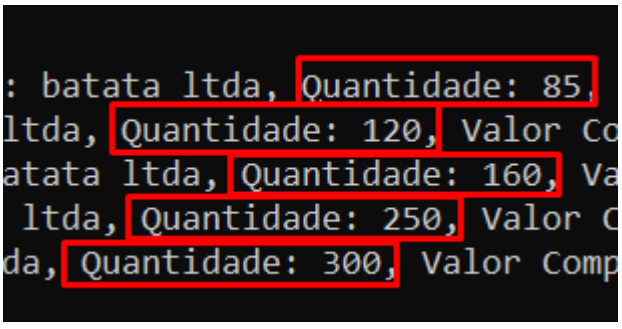


Ordenados por data de validade (opção 2):



Ordenados por Quantidade em estoque (opção 3):

Por falta de espaço não foi possível mostrar os nomes dos produtos na imagem. Entretanto a ordem foi: Batata Inglesa, Cebola, Batata Doce, Alface e Arroz.



Variáveis valor de compra (purchase_price) e valor de venda (sale_price) vão ser tipo float

Para implementar isso foi necessário fazer a modificação na declaração dessas variáveis e em outra variáveis que dependem delas, como:

```
purchase_price:      .float 0.0
sale_price:          .float 0.0
total_purchase:      .float 0.0
total_sale:          .float 0.0
total_profit:        .float 0.0
```

Além disso, foi necessário modificar o especificador de formato em todas as ocorrências de leitura e escrita dessas variáveis para **%f**.

Para fazer o ***printf*** dessas variáveis foi necessário modificá-las de ***float*** para ***double***, visto que o ***printf*** só lê valores ***double***.

Na funcionalidade de consulta financeira, foi necessário modificar as operações para que elas fossem realizadas na FPU, visto que agora os valores são de ponto flutuante.

Aqui está o trecho de código que fez a operação para calcular o lucro total esperado do estoque:

```
# calculate total profit = total_sale - total_purchase
        flds    total_purchase
        flds    total_sale
        fsub    %st(1), %st(0)
        fsts    total_profit
        fstp    %st(0)
        fstp    %st(0)
```

Leitura e Escrita em Arquivo por Chamada de Função (System Calls)

Lembrando que essa funcionalidade já foi implementada na versão 1 deste trabalho. A única diferença é foi necessário fazer pequenas modificações para esses operações suportarem os valores de ponto flutuante ***purchase_price*** e ***sale_price***.