RELATÓRIO TÉCNICO – PROJETO U2

1. Introdução

Este relatório apresenta a evolução do projeto desenvolvido inicialmente na Unidade 1, onde foi criado um sistema simples de supermercado em C, permitindo ao usuário visualizar produtos, adicionar itens ao carrinho e finalizar a compra.

Na Unidade 2, o objetivo foi expandir esse projeto, incorporando novos tópicos vistos na disciplina. Além de aumentar a complexidade do código, a U2 buscou aproximar o aluno de problemas reais de gerenciamento de dados, estruturação de programas maiores e uso eficiente da memória.

As novas funcionalidades implementadas incluem: cadastro dinâmico de produtos personalizados pelo usuário, busca por nome usando manipulação de strings, geração de relatório de compras em forma de matriz e operações feitas com ponteiros para reduzir cópias desnecessárias.

2. Metodologia

O desenvolvimento foi realizado em **VSCode**, usando MinGW como compilador e o padrão **C99**. A abordagem utilizada foi incremental: primeiro a estrutura original da U1 foi revisada, depois os novos requisitos foram implementados de forma modular.

A organização do código priorizou: Funções separadas por responsabilidades, uso de ponteiros para passagem de dados por referência, criação de funções auxiliares para evitar repetição, comentários explicativos escritos de forma clara.

A cada função criada, foi feitos testes individuais para garantir que não houvesse erros de lógica ou problemas de memória. Ao final, o projeto passou por uma revisão geral para remover redundâncias e reforçar a segurança do código, especialmente em relação à alocação dinâmica.

3. Análise do Código

Strings

As strings foram usadas principalmente para: Armazenar nomes de produtos cadastrados pelo usuário, realizar comparações durante a busca de produtos, exibir informações formatadas, funções utilizadas da string.h.

Além disso, houve manipulação caractere a caractere ao normalizar nomes durante a pesquisa, convertendo letras maiúsculas/minúsculas.

Exemplo:

```
if (strcmp(busca, produtos[i].nome) == 0) {

    printf("Produto encontrado.\n");

}
```

Estruturas de Repetição Aninhadas

A principal aplicação dos laços aninhados foi na criação e exibição de uma **matriz de relatório**, onde cada linha representa um produto e cada coluna apresenta: ID, nome, quantidade, preço, total.

Exemplo:

```
for (int i = 0; i < qtd; i++) {
    for (int j = 0; j < 5; j++) {
        printf("%s\t", relatorio[i][j]);
    }
    printf("\n");
}
```

Matrizes

Foi criada uma matriz de strings para gerar relatórios completos da compra. Cada célula contém um texto formatado (como “3 unidades” ou “R$ 5.99”).

Uso típico:

```
char relatorio[maxProdutos][5][50];
```

Operações implementadas:

- Preenchimento bidimensional
- Impressão formatada
- Cálculo de totais por linha

O uso de matrizes aumentou a organização visual dos dados e demonstrou o uso dos laços aninhados.

Ponteiros

Ponteiros foram aplicados em vários momentos, principalmente em:

- Funções que recebem vetores
- Funções que alteram diretamente variáveis da main
- Manipulação de memória alocada dinamicamente

Exemplo de passagem por referência:

```
void adicionarProduto(Produto *p, int *qtd) {
   (*qtd)++;
}
```

Alocação Dinâmica

O projeto agora permite cadastrar produtos personalizados pelo usuário. Para isso, foi utilizado:

- malloc() para alocar o vetor de produtos
- malloc() adicional para alocar cada nome de produto
- Verificação de erros de alocação
- free() ao final do programa para liberar toda a memória

Exemplo:

```
produtos = malloc(qtd * sizeof(Produto));if (!produtos) {
   printf("Erro ao alocar memória.\n");
   exit(1);
```

}

Prevenção de memory leaks

Todas as alocações possuem seus free() correspondentes.
Cada string alocada para nomes também é liberada manualmente.

Dificuldades e Soluções

A maior dificuldade desta etapa foi trabalhar com ponteiros e memória dinâmica. Problemas como:

- esquecer de liberar memória
- confundir ponteiros simples com ponteiros para vetor
- acessar posições inválidas

Foram resolvidos com testes constantes e reorganização do código em funções menores. Outra dificuldade relevante foi trabalhar com strings dinâmicas, que exigem cuidado em relação ao tamanho, ao uso de strcpy() e ao tratamento de quebra de linha.

Além disso, a implementação de matrizes exigiu relembrar percorrimento bidimensional e garantir que os dados não saíssem dos limites.

Mesmo com esses desafios, o processo trouxe uma compreensão muito maior sobre como programas mais complexos são estruturados.

Conclusão

A evolução do projeto da U1 para a U2 permitiu consolidar os fundamentos da linguagem C e avançar para estruturas mais complexas.

O código resultante está mais flexível, organizado e seguro, além de mais próximo de um sistema real. A experiência reforçou a importância de modularidade e gerenciamento de memória.

O aprendizado adquirido nas duas primeiras unidades me formou uma base sólida para continuar evoluindo o projeto.