

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Departamento de Informática

Unidade Curricular: Estrutura de Dados

Relatório Relativo ao Projeto Final

Projeto de Estrutura de Dados: Supermarket Simulation

Realizado por: Benno Vasconcellos – 25202

João Falcão – 25003

Pedro Tavares – 25207

Viseu, 2023

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Departamento de Informática

Relatório relativo ao Projeto Final

Curso de Licenciatura em Engenharia Informática

Unidade Curricular de Estrutura de Dados

Projeto de Estrutura de Dados: Supermarket Simulation

Ano Letivo 2022/23

Viseu, 2023

# Resumo

O projeto "Supermarket Simulation" em linguagem C busca reproduzir fielmente a realidade de um supermercado, incluindo o fluxo de clientes, funcionários e produtos, além do armazenamento de dados relevantes. Foram realizadas pesquisas aprofundadas para identificar as tecnologias adequadas, seguidas por pré-simulações, fluxogramas e pseudocódigos para o planejamento eficiente do projeto. A implementação utiliza threads, uma hash-map e outras estruturas de dados para simular o ambiente de forma realista e otimizar o desempenho. Funções genéricas, makefiles e o GitHub foram utilizados para modularidade, automação e colaboração. A ferramenta Valgrind foi empregue para garantir a qualidade do código e evitar erros de memória. As Bibliotecas externas complementaram o

projeto.

# Abstract

The "Supermarket Simulation" project in C language seeks to faithfully reproduce the reality of a supermarket, including the flow of customers, employees, and products, in addition to storing relevant data. In-depth research was carried out to identify adapted technologies, followed by pre-simulations, flowcharts, and pseudocodes for efficient project planning. The implementation uses threads, a hash-map, and other data structures to realistically simulate the environment and optimize performance. Generic functions, makefiles, and GitHub were used for modularity, automation, and collaboration. The Valgrind tool was employed to ensure code quality and avoid memory errors. External libraries complement the project.

# PALAVRAS-CHAVE

Hash-map

Supermercado

Listas

Estrutura de Dados

Simulação

Linguagem C

Programação em C

Projeto

Pbplots

Colorful-printf

GitHub

Valgrind

Thread

Lucidchart

# KEY WORDS

Hash-map

Supermarket

Lists

Data structure

Simulation

C language

C Programing

Project

Pbplots

Colorful-printf

GitHub

Valgrind

Thread

**ÍNDICE**

[Resumo 2](#_Toc136555068)

[Abstract 4](#_Toc136555069)

[PALAVRAS-CHAVE 6](#_Toc136555070)

[KEY WORDS 8](#_Toc136555071)

[1. Introdução 1](#_Toc136555072)

[2. Tecnologias 3](#_Toc136555073)

[2.1. GitHub 3](#_Toc136555074)

[2.2. Lucidchart 4](#_Toc136555075)

[2.3. Valgrind 4](#_Toc136555076)

[2.4. Colorful-printf 5](#_Toc136555077)

[2.5. PbPlots 6](#_Toc136555078)

[2.6. Makefile 7](#_Toc136555079)

[3. Planejamento do projeto 8](#_Toc136555080)

[3.1. Organização de alguns dados 8](#_Toc136555081)

[3.2. Batente 9](#_Toc136555082)

[3.3. Estrutura inicial 10](#_Toc136555083)

[3.4. Estrutura Thread Global 11](#_Toc136555084)

[4. Estrutura Dados 12](#_Toc136555085)

[4.1. Hashmap 12](#_Toc136555086)

[4.2. Listas ligadas 13](#_Toc136555087)

[4.3. Vetores 13](#_Toc136555088)

[5. Análise aprofundada da implementação do projeto 14](#_Toc136555089)

[5.1. Ficheiro ImportExport.c 14](#_Toc136555090)

[5.2. Ficheiro ListaLigadaFuncoes.c 15](#_Toc136555091)

[5.3. Ficheiro Historico.c 17](#_Toc136555092)

[5.3.1. Histórico de Transações 17](#_Toc136555093)

[5.3.2. Histórico de Dados Estatísticos 18](#_Toc136555094)

[5.3.1. Como são feitos os cálculos e a gestão dos dados dos históricos 19](#_Toc136555095)

[5.3.2. Estrutura principal dos históricos 20](#_Toc136555096)

[6. Conclusões 21](#_Toc136555097)

[7. Referências 22](#_Toc136555098)

**ÍNDICE DE FIGURAS**

[Figura 1 - GitHub logo 3](#_Toc136555061)

[Figura 2 - Lucidchart logo 4](file:///C:\Users\benno\Downloads\1-06.docx#_Toc136555062)

[Figura 3 - Valgrind logo 4](file:///C:\Users\benno\Downloads\1-06.docx#_Toc136555063)

[Figura 4 - Imagem retirada do Readme.md do website 5](file:///C:\Users\benno\Downloads\1-06.docx#_Toc136555064)

[Figura 5 - Imagem retirada do Readme.md 6](#_Toc136555065)

[Figura 6 - imagem representativa de um HashMap 12](file:///C:\Users\benno\Downloads\1-06.docx#_Toc136555066)

[Figura 7 - imagem representativa de uma lista ligada 13](file:///C:\Users\benno\Downloads\1-06.docx#_Toc136555067)

# Introdução

O projeto "Supermarket Simulation" em linguagem C tem como objetivo principal reproduzir fielmente a realidade de um supermercado, abrangendo o fluxo de clientes, funcionários e produtos, bem como o armazenamento de dados relevantes. Para atingir esse objetivo, foram realizadas pesquisas aprofundadas para identificar as tecnologias mais adequadas ao desenvolvimento do projeto. Além disso, foram conduzidas pré-simulações, nas quais foram elaborados fluxogramas e pseudocódigos, visando planejar e estruturar o projeto de forma eficiente.

Após essa fase de análise e planejamento, chegamos à conclusão sobre o formato ideal para a implementação do projeto. Nesse sentido, optamos por utilizar “threads” que operam em paralelo, visando simular o ambiente de um supermercado de forma mais realista. Além disso, adotamos o uso de uma “hash-map” e outras estruturas de dados apropriadas para otimizar o desempenho e a eficiência do sistema. Também desenvolvemos funções genéricas, que podem ser reutilizadas em diferentes partes do código, buscando promover a modularidade e facilitar a manutenção do projeto.

Com o intuito de otimizar o processo de compilação, utilizamos “makefiles”, os quais permitem automatizar tarefas repetitivas e simplificar o processo de compilação e execução do programa. Além disso, empregamos a plataforma GitHub para o controle de versão e o trabalho em equipe, possibilitando uma colaboração mais eficiente entre os membros do projeto. Para garantir a qualidade e a integridade do código, fizemos uso da ferramenta “Valgrind”, disponível no ambiente Linux, a qual nos auxiliou na análise da utilização de memória e na deteção de possíveis vazamentos de memória ou erros.

Durante o desenvolvimento, também fizemos o download de bibliotecas externas (com licença MIT) disponíveis no GitHub, as quais complementaram as funcionalidades do projeto e nos permitiram aproveitar soluções já implementadas por outros desenvolvedores.

Com base em tudo que já foi dito, o relatório será dividido em cinco partes, de modo a fornecer uma explicação mais detalhada e uma perspetiva completa do projeto.

A primeira parte tem como objetivo introduzir o leitor a todo o contexto do projeto, abordando as tecnologias utilizadas, tais como pbPlots, colorful\_printf, Valgrind, GitHub, entre outras.

A segunda parte abordará o planejamento do projeto, descrevendo em detalhes as pesquisas realizadas, os experimentos conduzidos, bem como a elaboração dos fluxogramas e pseudocódigos.

A próxima parte será dedicada à explicação das estruturas de dados utilizadas no projeto, fornecendo informações sobre a sua implementação e funcionalidades.

A quarta parte consistirá na explicação detalhada dos desafios enfrentados durante o desenvolvimento do projeto e das soluções adotadas para superá-los. Também serão abordadas partes específicas do código, com uma análise mais aprofundada.

Por fim, a quinta parte será a conclusão final, na qual serão apresentadas as considerações finais sobre o projeto, incluindo os aprendizados adquiridos, as contribuições realizadas e as perspetivas futuras.

Com essa estrutura organizada, buscamos proporcionar uma compreensão abrangente e detalhada do projeto "Supermarket Simulation", destacando os processos e resultados alcançados ao longo do seu desenvolvimento.

# Tecnologias

Neste capítulo, foram abordadas em detalhes as ferramentas e tecnologias utilizadas ao longo do desenvolvimento do projeto "Supermarket Simulation". Ao explorarmos essas tecnologias em profundidade, podemos compreender melhor como elas foram aplicadas e de que forma contribuíram para a criação de uma simulação de supermercado mais realista e eficiente.

## GitHub

O GitHub desempenhou um papel fundamental no desenvolvimento do projeto, fornecendo um ambiente colaborativo e ferramentas de controle de versão. Utilizamos o GitHub para sincronizar e compartilhar o código-fonte entre os membros da equipe, permitindo um trabalho conjunto eficiente. Além disso, a capacidade de rastrear as alterações feitas no código e a possibilidade de reverter para versões anteriores foram vantagens significativas do uso do GitHub.



Figura - GitHub logo

## Lucidchart

A plataforma Lucidchart foi uma ferramenta valiosa para planejar e visualizar a estrutura do projeto. Com o Lucidchart, pudemos criar diagramas que representavam as conexões entre os diferentes dados e as “threads” utilizadas no simulador de supermercado. Essa visualização facilitou a compreensão do fluxo do programa e ajudou a organizar a lógica de execução.



Figura - Lucidchart logo

## Valgrind

O Valgrind foi uma ferramenta essencial para garantir a qualidade e integridade do código. Usamos o Valgrind no ambiente Linux para analisar a utilização de memória e detetar possíveis vazamentos de memória ou erros. Através dessas verificações, pudemos identificar e corrigir problemas relacionados ao uso incorreto da memória e melhorar a eficiência do programa.

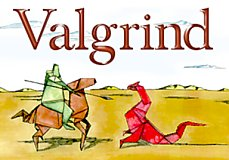


Figura - Valgrind logo

## Colorful-printf

Uma imagem com texto, captura de ecrã, software, Sistema operativo

Descrição gerada automaticamenteNo GitHub, encontramos uma biblioteca chamada colorful-printf (<https://github.com/misc0110/colorful-printf>) que nos ajudou a imprimir mensagens coloridas durante a execução do programa. Com essa biblioteca, conseguimos usar comandos como printc("[red]Olá! [/red]") para exibir mensagens em cores específicas, o que tornou mais fácil visualizar e entender as informações no terminal. Essa funcionalidade foi especialmente útil para destacar informações importantes ou alertas durante a simulação do supermercado. Fizemos algumas alterações nessa biblioteca para adaptá-la ao nosso projeto, permitindo uma integração perfeita e personalizada.

Figura - Imagem retirada do Readme.md do website

## PbPlots

Descobrimos a biblioteca pbPlots no GitHub (<https://github.com/InductiveComputerScience/pbPlots>) e ela foi uma adição valiosa ao nosso projeto. Essa biblioteca em C nos permitiu criar vários tipos de gráficos de forma poderosa. Utilizámos o pbPlots para gerar gráficos que representavam o histórico de dados relevantes da simulação do supermercado, como a média de clientes em loja, a taxa de ocupação das caixas, entre muitos outros. Esses gráficos vão ser uma mais-valia no que toca à análise de desempenho e eficiencia do supermercado como uma empresa.

Uma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamente

Figura - Imagem retirada do Readme.md

## Makefile

O Makefile desempenhou um papel crucial no desenvolvimento do código e, consequentemente, em todo o projeto. Ele permitiu agilizar o processo de compilação por meio da criação de objetos (arquivos .o) que, posteriormente, foram transformados no arquivo final do projeto (arquivoFinal.c). Com uma compilação totalmente automatizada, utilizando listas no Makefile e substituições de extensões, configuramos o Makefile apenas uma vez, o que facilitou todo o processo de compilação ao longo do desenvolvimento do projeto. Vale ressaltar que adaptamos o processo de compilação para ser compatível tanto com o sistema operacional Windows quanto com o Linux, garantindo a portabilidade do projeto. As instruções detalhadas sobre como compilar o código estão disponíveis no arquivo README.md, que oferece duas opções: compilar usando o Makefile ou o GCC.

# Planejamento do projeto

Esta seção do relatório foi cuidadosamente desenvolvida com base em um planejamento detalhado do projeto. Utilizamos o Lucidchart para criar diagramas que nos ajudaram a visualizar e estruturar as ideias de forma clara e organizada. Além disso, promovemos discussões em grupo para obter insights valiosos e aprimorar o planejamento inicial.

Para validar e testar nossas abordagens, realizamos experimentos práticos, implementando exemplos de código relevantes. Isso nos permitiu avaliar a eficácia das soluções propostas e realizar ajustes quando necessário.

Ao longo do processo, dedicamos tempo a pesquisas aprofundadas, com o objetivo de embasar nossas decisões. Essa abordagem nos permitiu tomar decisões informadas e garantir a eficiência do projeto como um todo.

Gostaríamos de informar que as imagens do Lucidchart, que auxiliaram na representação visual de nosso planejamento, serão enviadas em anexo para análise.

## Organização de alguns dados

As caixas, por exemplo, possuem as seguintes variáveis: um ID para identificação única, um indicador de status para verificar se a caixa está aberta ou fechada, uma variável para lidar com situações de fechamento por urgência, o tempo total de espera na fila, uma referência ao funcionário responsável pela caixa, um mutexLock para garantir a exclusão mútua ao acessar os dados, uma variável para indicar se a thread da caixa está aberta ou não, e uma lista de pessoas que representa a fila de clientes a serem atendidos pela caixa.

Os funcionários, por sua vez, possuem dados como um ID para identificação única, um nome associado ao funcionário, um atraso médio que determina o tempo de atraso na execução de suas atividades, uma variável para indicar se o funcionário está ativo ou não e um número total de vendas realizadas, que registra quantos clientes foram atendidos por aquele funcionário.

Os clientes possuem informações como um ID para identificação única, um nome associado ao cliente, uma data de nascimento para identificação, um saldo no cartão cliente, que representa o saldo disponível para compras, estimativas de tempo para a compra, para o atendimento na caixa e para o tempo de espera na fila, um atraso médio que determina o tempo de atraso na execução de suas atividades, um tempo de brinde que indica o período em que o cliente recebe algum benefício especial, e uma lista de produtos que representa os itens adquiridos pelo cliente.

Por fim, os produtos são representados com informações como um ID único, a quantidade de produtos repetidos disponíveis, uma variável para indicar se o produto foi oferecido ao cliente ou não, os tempos de compra e de atendimento na caixa, o nome do produto e o preço associado a ele.

Essa organização de dados permite que o projeto "Supermarket Simulation" simule com precisão as interações entre caixas, funcionários, clientes e produtos, fornecendo uma experiência de simulação mais realista e funcional.

## Batente

No desenvolvimento do projeto, implementamos uma abordagem eficiente para controlar a presença de clientes e funcionários na loja. Essa funcionalidade foi de grande importância, pois precisávamos ter uma maneira rápida de identificar quais clientes estavam na loja e quais não estavam. Para isso, utilizamos uma estrutura especial chamada "batente" dentro de nosso array de clientes e funcionários, que desempenhou um papel significativo em nossa implementação.

Dividimos o array em duas partes: ativos e inativos. Sempre que um cliente/funcionário se tornava ativo, ele era movido para a posição onde estava o primeiro cliente inativo. Da mesma forma, o primeiro cliente inativo era movido para a posição onde estava o cliente que passou a ser ativo. Essa abordagem nos permitiu ter um acesso rápido aos clientes ativos e inativos.

Ao remover um cliente ativo, realizávamos a seguinte operação: movíamos esse cliente para a última posição dos clientes ativos e, em seguida, pegávamos o último cliente ativo e o movíamos para a posição onde estava o cliente que passaria a ser inativo. Essa operação garantia a integridade da ordem dos clientes ativos, mantendo-os organizados e evitando espaços vazios no array.

Quando precisávamos gerar um novo cliente, podíamos pesquisar no vetor dos clientes inativos, que continha apenas os clientes que não estavam na loja. Isso nos proporcionou uma forma eficiente de encontrar o próximo cliente disponível para entrar na loja.

Essa abordagem do "batente" dentro do array de clientes foi fundamental para otimizar o gerenciamento e controle dos clientes presentes na loja. Com essa estratégia, pudemos obter de forma rápida e eficiente a lista dos clientes ativos e identificar os próximos clientes disponíveis para entrar na loja, proporcionando uma melhor experiência de simulação no projeto.

## Estrutura inicial

Ao planejar a estrutura inicial do projeto, tivemos de considerar cuidadosamente o fato de estarmos trabalhando com threads, o que poderia resultar em problemas de sincronização e perda de dados, entre outros problemas. Portanto, nos esforçamos ao máximo para mitigar esses problemas, garantindo que os dados não fossem dependentes uns dos outros.

No esquema inicial desenvolvido no Lucidchart, projetamos uma thread global que só funcionaria se a loja estivesse aberta. Essa thread seria responsável por permitir a entrada dos clientes na loja e guiá-los pelo processo de compra dos produtos, considerando o tempo de compra de cada um. Após a conclusão das compras, os clientes seriam direcionados para uma fila única, similar a alguns supermercados, onde a thread global realizaria as validações necessárias para escolher a melhor caixa para cada cliente. Essa seleção ocorreria apenas se todas as caixas estivessem com um tempo de atendimento superior ao estipulado pelas definições. Caso contrário, os clientes permaneceriam na fila única, aguardando atendimento.

Após a seleção da caixa, cada thread de caixa seria responsável por processar os clientes atribuídos a ela. Com essa estrutura, buscamos minimizar problemas de sincronização entre as threads, permitindo que cada thread de caixa operasse independentemente, processando os clientes de forma eficiente e evitando conflitos de acesso aos dados.

Ao longo do desenvolvimento do projeto, fizemos ajustes e refinamentos nessa estrutura inicial com base em experimentos práticos e pesquisas aprofundadas, visando otimizar a sincronização entre as threads e garantir a integridade dos dados durante a simulação do supermercado.

## Estrutura Thread Global

No desenvolvimento do projeto, foi criada uma thread global responsável por controlar o fluxo de entrada dos clientes no supermercado. Essa thread operava dentro de um Loop que continuava enquanto a loja estivesse aberta e ainda houvesse clientes presentes.

Dentro desse Loop, era verificada uma probabilidade para gerar uma pessoa. Caso essa probabilidade fosse satisfeita, era escolhido aleatoriamente se seria gerado um cliente ou um "guest" (usuário temporário e ocasional, que não regista dados pessoais, apenas informações genéricas e estatísticas). Os clientes eram selecionados a partir de uma estrutura de vetor dinâmico, onde existia uma função de batente que mantinha as pessoas disponíveis no início do vetor e atualizava o número de clientes ativos.

Após essa etapa, as pessoas eram encaminhadas para outra thread responsável por simular o tempo de compra de cada cliente. Essa simulação refletia o tempo que cada pessoa levava para comprar seus produtos antes de se dirigir às caixas.

A lista de pessoas prontas para serem atendidas nas caixas seguia o princípio da "fila única" adotado por supermercados como o Continente. Esse sistema permitia selecionar a caixa adequada com base em critérios como a disponibilidade das caixas e o tempo de atendimento estipulado pelas definições do projeto. Se todas as caixas estivessem fechadas ou se o tempo de atendimento estivesse acima do estipulado, uma nova caixa era aberta.

Essa estrutura do loop com a thread global e as operações de seleção de clientes e caixas permitia simular de forma mais realista o fluxo de entrada, tempo de compra e atendimento dos clientes, evitando conflitos de acesso aos dados e promovendo uma experiência de simulação eficiente.

# Estrutura Dados

Escolher as estruturas de dados adequadas é de extrema importância no desenvolvimento de um projeto, pois elas impactam diretamente na eficiência e desempenho do sistema. Cada estrutura de dados possui características específicas que a tornam mais adequada para determinados tipos de operações, como busca, inserção, remoção e acesso aos dados. Entre as estruturas de dados utilizadas, destacam-se:

## Uma imagem com texto, diagrama, captura de ecrã Descrição gerada automaticamenteHashmap

Um hashmap é uma estrutura de dados que permite armazenar pares de chave-valor, em que cada chave é única e corresponde a um valor específico. No nosso projeto, utilizamos o hashmap para mapear cada cliente com suas informações relevantes, como nome, idade, e lista de produtos comprados, bem como para mapear cada produto com suas informações, como nome, preço e quantidade disponível. A principal vantagem do hashmap é a sua eficiência na busca e recuperação de informações, uma vez que permite acesso rápido aos dados através da chave correspondente.

Figura - imagem representativa de um HashMap

## Listas ligadas

Utilizamos listas ligadas para representar estruturas de dados como filas de clientes nas caixas e prateleiras de produtos. Uma lista ligada é uma estrutura em que cada elemento contém um valor e um ponteiro para o próximo elemento da lista. No caso das filas de clientes, cada nó da lista representa um cliente e seu respetivo tempo de chegada. Para as prateleiras de produtos, cada nó da lista representa um produto e suas informações associadas. A vantagem das listas ligadas é a sua eficiência em adicionar e remover elementos, o que é essencial para a manipulação das filas e prateleiras do supermercado.

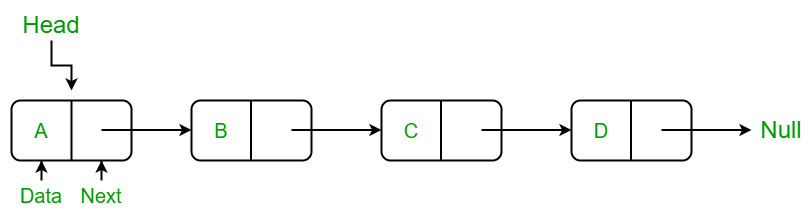


Figura - imagem representativa de uma lista ligada

## Vetores

Empregamos vetores para representar estruturas de dados fixas, como o conjunto de caixas do supermercado e o estoque de produtos. Um vetor é uma estrutura de dados contígua que permite o acesso direto aos elementos com base em seus índices. No caso das caixas do supermercado, cada posição do vetor representa uma caixa e suas informações associadas, como o estado (aberta ou fechada) e o tempo de atendimento. Para o estoque de produtos, cada posição do vetor representa um produto e suas informações, como nome, preço e quantidade disponível. A principal vantagem dos vetores é o acesso rápido aos elementos, uma vez que não é necessário realizar operações de inserção ou remoção.

# Análise aprofundada da implementação do projeto

Nesta etapa do relatório, dedicaremos nossa atenção à explicação detalhada de algumas das funções implementadas nos respetivos arquivos do código do projeto "Supermarket Simulation". Nosso objetivo é fornecer uma compreensão mais abrangente das funcionalidades e do papel de cada função no contexto da simulação do supermercado.

A análise das funções implementadas é essencial para uma compreensão completa do projeto "Supermarket Simulation" e destaca a qualidade do código desenvolvido, bem como o esforço em fornecer uma simulação de supermercado realista e funcional.

## Ficheiro ImportExport.c

O arquivo "ImportExport.c" contém um conjunto de funções para importar e exportar dados relacionados a clientes, funcionários e produtos em um sistema de gerenciamento de uma loja. Essas funções permitem que os dados sejam armazenados em arquivos de texto ou binários, facilitando o armazenamento e recuperação dessas informações.

A função principal, "importarDados", recebe como parâmetros uma função para salvar os dados importados e o tipo de dados a ser importado. Essa função lê os dados de um arquivo de texto correspondente ao tipo de dados especificado, percorre as linhas do arquivo e extrai os valores correspondentes a cada atributo do objeto de dados. Em seguida, ela invoca a função para salvar os dados importados na estrutura de dados apropriada.

Por exemplo, a função "importarClientes" é usada para importar os dados dos clientes. Ela recebe uma linha do arquivo de texto contendo as informações de um cliente, divide essa linha em campos separados por tabulação e armazena cada valor correspondente em um objeto da estrutura de dados "ClienteStruct". Essa estrutura contém campos como "id", "nome", "saldoCartaoCliente" e "dataNascimento". Os valores extraídos do arquivo são atribuídos a esses campos.

Similarmente, as funções "importarFuncionarios" e "importarProdutos" são responsáveis por importar os dados dos funcionários e produtos, respetivamente. Elas seguem a mesma abordagem de dividir as linhas do arquivo em campos e atribuir os valores correspondentes aos campos apropriados nas estruturas de dados correspondentes.

Além disso, o arquivo "ImportExport.c" também fornece as funções "exportarDados" e "exportarOpcoes" para exportar os dados e opções do sistema de volta para os arquivos de texto ou binários. Essas funções percorrem as estruturas de dados e gravam os valores correspondentes nos arquivos especificados.

Em resumo, o arquivo "ImportExport.c" contém um conjunto de funções para importar e exportar dados de clientes, funcionários e produtos em um sistema de gerenciamento de loja. Essas funções facilitam a leitura e gravação dos dados em arquivos de texto ou binários, permitindo o armazenamento e recuperação eficiente das informações do sistema.

## Ficheiro ListaLigadaFuncoes.c

O arquivo "ListaLigadaFuncoes.c" contém um conjunto de funções para lidar com listas ligadas. Essas funções permitem a criação de listas genéricas, adição e remoção de elementos no início, fim ou em um índice específico da lista, pesquisa e remoção de elementos com base em critérios personalizados, destruição de elementos e destruição da lista em si.

A função "criarLista" é responsável por criar e retornar uma nova lista vazia. Ela aloca memória para a estrutura da lista, inicializa seus campos e retorna o ponteiro para a lista recém-criada.

A função "criarElemento" é utilizada para criar um elemento a ser adicionado à lista. Ela recebe um ponteiro genérico para o elemento a ser armazenado e verifica se o elemento é válido. Se o elemento for nulo, a função exibe uma mensagem de erro e retorna nulo. Caso contrário, ela aloca memória para a estrutura do elemento, atribui o valor do elemento recebido ao campo "Info" do elemento e retorna o ponteiro para o elemento criado.

As funções "AddElementoInicio" e "AddElementoFim" são responsáveis por adicionar elementos no início e no fim da lista, respectivamente. Ambas as funções verificam se a lista e o elemento são válidos. Em seguida, realizam as operações necessárias para adicionar o elemento à lista, atualizando os campos "head" e "tail" conforme necessário, e incrementam a quantidade de elementos da lista.

A função "RemElementoUltimo" é utilizada para remover o último elemento da lista. Ela verifica se a lista não é nula e se não está vazia. Em seguida, percorre a lista até o último elemento, atualizando os ponteiros necessários. Ao final, ela remove o elemento da lista, atualiza os campos "head" e "tail" se necessário e decrementa a quantidade de elementos da lista. O elemento removido é retornado.

A função "RemElementoInicio" remove o primeiro elemento da lista. Ela verifica se a lista não é nula e se não está vazia. Em seguida, atualiza o ponteiro "head" para apontar para o próximo elemento da lista, decrementa a quantidade de elementos e retorna o elemento removido.

A função "RemElementoIndex" é responsável por remover um elemento em um índice específico da lista. Ela verifica se a lista não é nula, se não está vazia e se o índice está dentro dos limites da lista. A função percorre a lista até o índice desejado, atualiza os ponteiros necessários e remove o elemento da lista. Os campos "head" e "tail" são atualizados se necessário, e a quantidade de elementos é decrementada. O elemento removido é retornado.

A função "RemElementoPesquisa" remove um elemento com base em uma pesquisa personalizada. Ela verifica se a lista não é nula, se não está vazia e se o elemento de pesquisa não é nulo. A função percorre a lista até encontrar o elemento desejado usando a função "compareInfo" fornecida como argumento. Uma vez encontrado o elemento, os ponteiros são atualizados e o elemento é removido da lista. A quantidade de elementos é decrementada e o elemento removido é retornado.

As funções "destruirElemento" e "destruirLista" são utilizadas para liberar a memória alocada para os elementos e a lista em si, respectivamente. Ambas as funções verificam se o elemento ou a lista não são nulos. A função "destruirElemento" chama a função "destruirInfo" fornecida como argumento para liberar a memória do campo "Info" do elemento antes de liberar o elemento em si. A função "destruirLista" percorre todos os elementos da lista, liberando a memória alocada para cada elemento e, em seguida, libera a memória alocada para a estrutura da lista.

## Ficheiro Historico.c

O histórico tem como objetivo guardar os diferentes dados manipulados e gerados no decorrer da simulação e, mais tarde, apresentá-los ao utilizador do programa.

Existem dois tipos de históricos neste projeto: o **histórico de transações** e o **histórico de dados estatísticos**.

### Histórico de Transações

O histórico de transações é responsável por armazenar e apresentar os diferentes dados manipulados e gerados durante a simulação do programa. Cada cliente atendido gera uma transação, e todos os detalhes dessa transação são registados no histórico.

Como as informações armazenadas neste histórico vão ser de grande utilidade no decorrer da simulação, tínhamos de engenhar uma maneira de organizar estes dados de forma personalizada para que se tornasse mais rápido e fácil de os guardar e ler. Para isso utilizámos a estrutura de dados conhecida como **hashmap**. Um esquema ilustrando a estrutura do histórico de transações pode ser encontrado no arquivo "HistoricoTransacoesEstrutura.pdf" anexado a este relatório.

A estrutura utilizada para armazenar os detalhes das transações é definida da seguinte forma:

A screen shot of a computer program

Description automatically generated with medium confidence

Essa estrutura contém informações como o funcionário responsável, a lista de produtos adquiridos, o tempo estimado na caixa, possíveis atrasos, o movimento do cartão do cliente, o preço total da transação, o valor do produto oferecido, se algum, e a data da transação.

No final de cada dia, os dados são usados para cálculos estatísticos e salvos em formatos como txt e csv e, alguns deles, representados de forma visual utilizando os gráficos da biblioteca PbPlots.

### Histórico de Dados Estatísticos

O histórico de dados estatísticos armazena periodicamente diversas informações relacionadas ao funcionamento do supermercado. Essas informações incluem o número de clientes, o tempo de espera em determinadas caixas em um determinado horário, entre outras. Um esquema que ilustra a estrutura do histórico de dados estatísticos pode ser encontrado no arquivo "HistoricoDadosEstatisticosEstrutura.pdf", anexado a este relatório.

Os dados estatísticos são coletados em intervalos de tempo regulares, neste caso, a cada 10 minutos e ao final de um dia vamos ter uma matriz 24x6 (24 horas e 6 intervalos de 10 minutos cada) do tipo "DadosInstantaneoStruct" que será responsável por armazenar os dados recolhidos. Esses dados são utilizados posteriormente para cálculos estatísticos e são salvos em arquivos txt, csv e de forma visual utilizando os gráficos da biblioteca PbPlots.

A estrutura utilizada para armazenar os dados instantâneos é definida da seguinte forma:

A picture containing text, businesscard, screenshot, font

Description automatically generated

Essa estrutura contém informações como o tempo de espera e o número de clientes em fila para cada caixa, o número de caixas abertas e o número total de clientes no supermercado.

As estruturas mencionadas são essenciais para o armazenamento, cálculo e apresentação dos dados no final de cada dia.

### Como são feitos os cálculos e a gestão dos dados dos históricos

Como já foi referido anteriormente, ao final de cada dia, ambos os históricos de transaç4oes e de dados estatísticos contêm informações importantes relativas ao funcionamento do supermercado. Estas informações são então usadas para fazer cálculos para obter insights importantes tais como o funcionário que atendeu mais ou menos pessoas, a hora em que existiu um pico de fluxo de clientes na loja, qual o número de clientes na fila de uma determinada caixa a uma determinada hora, em que hora do dia o fluxo de clientes numa determinada caixa foi menor, entre muitos outros.

Para isso, foi criada uma estrutura responsável por armazenar os dados recolhidos de forma organizada e os resultados de cálculos estatísticos realizados com os mesmos. A seguir encontra-se uma imagem do código usado para criar a estrutura responsável por armazenar esses mesmos dados:

A picture containing text, screenshot, display, software

Description automatically generated

Tal como o histórico de transações e o histórico de dados estatísticos, os cálculos realizados são limpos e inicializados ao final de cada dia.

### Estrutura principal dos históricos

Nesta estrutura chamada “HistoricoStruct” juntámos ambos os históricos e as conclusões estatísticas realizadas através dos mesmos num sítio só. Está abaixo uma imagem que demonstra como foi feito o código usado para criar esta mesma estrutura.

A picture containing text, businesscard, screenshot, font

Description automatically generated

Como podemos ver, na linha 3 foi definido o histórico de transações, na linha 5 foi definido o histórico de dados estatísticos e na linha 6 foram definidas as estruturas usadas para armazenar os resultados dos cálculos estatísticos.

A variável “mediaDiaria” é limpa e inicializada a cada dia, enquanto a variável “global” é atualizada, servindo o mesmo propósito que a anterior, mas desta vez para todos os dias em que a simulação já ocorreu.

# Conclusões

Neste projeto "Supermarket Simulation", conseguimos realizar os principais objetivos de reproduzir fielmente a realidade de um supermercado, abrangendo o fluxo de clientes, funcionários e produtos, bem como o armazenamento de dados relevantes. Por meio de pesquisas aprofundadas e pré-simulações, planejamos e estruturamos o projeto de forma eficiente.

Utilizamos "threads" para simular o ambiente de um supermercado de forma mais realista, e empregamos estruturas de dados adequadas, como uma "hash-map", para otimizar o desempenho do sistema. A modularidade e reutilização de código foram promovidas por meio do desenvolvimento de funções genéricas.

Para facilitar o processo de compilação, utilizamos "makefiles" e aproveitamos a plataforma GitHub para o controle de versão e a colaboração entre os membros da equipe. A qualidade do código foi garantida por meio do uso da ferramenta "Valgrind" para análise da utilização de memória.

Durante o desenvolvimento, também aproveitamos bibliotecas externas disponíveis no GitHub, que complementaram as funcionalidades do projeto.

Com base em todo o trabalho realizado, o projeto foi dividido em cinco partes para fornecer uma explicação detalhada e uma visão completa do mesmo.

No geral, o projeto "Supermarket Simulation" foi bem-sucedido em alcançar seus principais objetivos, proporcionando uma simulação realista de um supermercado e demonstrando a aplicação de conceitos avançados de programação em linguagem C.

# Referências

(s.d.). Obtido de ChatGPT: https://chat.openai.com/

*Youtube*. (s.d.). Obtido de Computerphile: https://www.youtube.com/watch?v=7ENFeb-J75k&ab\_channel=Computerphile