# Trabalho 1 – WhatsApl (Parte 1)

Desenvolva uma aplicação em C++ para gestão de uma aplicação de *messaging*. A aplicação deve permitir gerir as conversações entre utilizadores ou grupos de utilizadores, segundo determinados temas. Não se pretende implementar uma aplicação de *messaging*, apenas gerir a informação aí presente.

Para cada utilizador, interessa registar o login, nome, *email*, telemóvel e data de adesão à aplicação. Cada grupo tem um título, data de criação, utilizador moderador e conjunto de utilizadores membros do grupo, com data de adesão ao grupo. Cada conversação inclui uma lista de mensagens trocadas, por ordem cronológica. Cada mensagem pode ser de diversos tipos (texto, imagem, vídeo, …), e é etiquetada por uma data/hora, um emissor e um destinatário (que poderá ser um grupo).

Na gestão de um grupo, o seu moderador pode aceitar/rejeitar pedidos de adesão de utilizadores ao grupo, e bloquear membros (sendo que estes deixam de receber e de poder enviar mensagens destinadas ao grupo). Membros bloqueados podem ser desbloqueados, mas a ocorrência deve ficar registada.

Alguns requisitos obrigatórios são (pode e deve incluir outros que considere relevantes):

- Manutenção de utilizadores
- Manutenção de grupos
- Manutenção de conversações
- Associar utilizadores a grupos
- Associar mensagens a conversações
- Associar conversações a utilizadores e a grupos
- … e outras associações que julgue necessárias

A aplicação deve permitir registar e gerir utilizadores, grupos, conversações e mensagens.

Sugere-se como entidades a implementar: Utilizador, Grupo, Conversa, Mensagem, entre outras que julgue necessárias (considerar hierarquia de classes).

Usando ***estruturas de dados lineares***, deve implementar:

- Manutenção do conjunto de utilizadores
- Manutenção do conjunto de grupos
- Manutenção do conjunto de conversações
- Listagens várias (totais ou parciais com critérios a definir pelo utilizador, com possível ordenação) de, por exemplo:
  - utilizadores
  - grupos
  - conversações
  - mensagens de uma conversação
  - mensagens recebidas/enviadas por um utilizador num grupo (tendo em conta bloqueios)
  - …

*Notas:*

1. Por "manutenção de dados" entende-se as operações básicas CRUD (*Create, Read, Update, Delete*)
2. O trabalho deve respeitar o que está descrito em "Notas relativas à implementação" no texto relativo aos "Enunciados e instruções" da parte 1 do trabalho prático

# Trabalho 2 – Faturação de IVA (Parte 1)

Desenvolva uma aplicação em C++ para registo de faturas dos contribuintes. A aplicação deve permitir gerir as faturas emitidas por diferentes entidades e associadas a diferentes consumidores.

Para cada contribuinte, interessa registar o número de identificação fiscal (NIF), nome, *email* e morada fiscal. Os contribuintes podem ser sujeitos passivos de IVA (e portanto emissores de faturas), para o que terão um ramo de atividade associado (de acordo com a Classificação de Atividades Económicas – CAE). Cada fatura emitida tem um número único, e deve incluir uma data, NIF do emissor, NIF do consumidor, montante e regime de IVA.

Até ao final de cada mês, cada sujeito passivo de IVA deve entregar uma declaração de IVA referente ao mês anterior onde este indica o total da faturação mensal, e fazer o pagamento de IVA correspondente. A declaração só será aceite se o seu montante corresponder ao total das faturas registadas para o mês em causa. No caso de o montante ultrapassar o valor das faturas introduzidas, deve ser indicado o valor em falta correspondente a faturas que deverão ser introduzidas. No caso de o montante declarado ser inferior ao valor das faturas introduzidas, o sistema deve produzir uma listagem de todas as faturas referentes ao mês em causa. Em cada momento deve ser possível saber que declarações de IVA estão em falta. Deve ser ainda possível saber, no final do ano, qual o montante de IVA pago por cada sujeito passivo de IVA.

Alguns requisitos obrigatórios são (pode e deve incluir outros que considere relevantes):

- Manutenção de contribuintes e de sujeitos passivos de IVA
- Manutenção de faturas
- Manutenção de declarações de IVA
- Associar faturas a contribuintes (emissor e consumidor)
- Associar declarações de IVA a contribuintes
- … e outras associações que julgue necessárias

A aplicação deve permitir registar e gerir contribuintes, faturas e declarações de IVA.

Sugere-se como entidades a implementar: Contribuinte, PassivoIVA, Fatura, DeclIVA, entre outras que julgue necessárias (considerar hierarquia de classes).

Usando ***estruturas de dados lineares***, deve implementar:

- Manutenção do conjunto de contribuintes e de sujeitos passivos de IVA
- Manutenção do conjunto de faturas
- Manutenção do conjunto de declarações de IVA
- Listagens várias (totais ou parciais com critérios a definir pelo utilizador, com possível ordenação) de, por exemplo:
  - contribuintes e sujeitos passivos de IVA
  - faturas
  - declarações de IVA (por sujeito passivo, por ramo de atividade, …)
  - montantes de IVA por sujeito passivo ou por contribuinte
  - …

*Notas:*

1. Por "manutenção de dados" entende-se as operações básicas CRUD (*Create, Read, Update, Delete*)
2. O trabalho deve respeitar o que está descrito em "Notas relativas à implementação" no texto relativo aos "Enunciados e instruções" da parte 1 do trabalho prático

# Trabalho 3 – OLZ (Parte 1)

Desenvolva uma aplicação em C++ para gestão de um *site* de compra e venda de artigos usados. A aplicação deve permitir gerir utilizadores, anúncios de produtos e concretização de compras/vendas.

Qualquer pessoa se pode registar no *site* OLZ como utilizador, fornecendo o seu endereço de *email* (que funciona como *login*), nome, número de contacto (telemóvel) e localização (freguesia, concelho e distrito). Cada utilizador pode anunciar produtos para compra/venda. No anúncio, o anunciante especifica um título, uma categoria do produto, uma descrição e zero ou mais imagens. O anunciante especifica ainda quais dos seus dados pessoais quer que sejam visíveis em pesquisas no *site*: *email*, nome e/ou telemóvel. Ao anúncio fica associado um identificador único e uma data de criação. Um anúncio de venda inclui ainda um estado do produto ("novo", "usando como novo", "funcional" ou "para peças"), o preço pretendido e uma indicação da possibilidade de negociação do preço. Um anúncio de compra pode incluir uma referência a um anúncio de venda do mesmo anunciante, alertando os visitantes do *site* para o facto de o anunciante aceitar uma troca.

Qualquer pessoa (registada ou não) pode pesquisar anúncios no *site*, segundo diferentes critérios: por categoria, por localização do anunciante, por palavras-chave e/ou por preço aproximado. Cada anúncio (de compra ou de venda) tem associado um número de visualizações, isto é, o número de vezes que alguém "clicou" no anúncio depois de este aparecer numa listagem. O contacto entre a pessoa interessada em vender/comprar e o anunciante pode ser efetuado pelo *site* (em particular nos casos em que o anunciante optou por não tornar públicos os seus dados pessoais), deixando uma mensagem e contacto, dados que serão enviados por *email* ao anunciante. Este tipo de contactos deve ficar associado ao anúncio em causa. Concretizado o negócio, pede-se ao anunciante que indique o montante da venda/compra e a data, deixando o anúncio de aparecer em pesquisas. O anunciante pode também optar por remover o anúncio.

Alguns requisitos obrigatórios são (pode e deve incluir outros que considere relevantes):

- Manutenção de utilizadores
- Manutenção de anúncios
- Manutenção de contactos
- Associar anúncios a utilizadores
- … e outras associações que julgue necessárias

A aplicação deve permitir registar e gerir utilizadores, anúncios e contactos.

Sugere-se como entidades a implementar: Utilizador, Anuncio, Contacto, entre outras que julgue necessárias (considerar hierarquia de classes).

Usando ***estruturas de dados lineares***, deve implementar:

- Manutenção do conjunto de utilizadores
- Manutenção do conjunto de anúncios
- Manutenção do conjunto de contactos
- Listagens várias (totais ou parciais com critérios a definir pelo utilizador, com possível ordenação) de, por exemplo:
  - utilizadores
  - anúncios (por utilizador, por tipo, por categoria, …)
  - negócios concretizados (por utilizador, por tipo, por categoria, …)
  - …

*Notas:*

1. Por "manutenção de dados" entende-se as operações básicas CRUD (*Create, Read, Update, Delete*)
2. O trabalho deve respeitar o que está descrito em "Notas relativas à implementação" no texto relativo aos "Enunciados e instruções" da parte 1 do trabalho prático.

# Trabalho 4 – Empresa de Transporte de Mercadorias (Parte 1)

Desenvolva uma aplicação em C++ para gestão de pedidos de transporte de mercadorias por parte de uma empresa. A empresa dispõe de um número fixo de camiões de diferentes tipos, para transporte de mercadorias específicas.

A empresa possui camiões para transporte de: mercadoria que necessita de congelação, produtos considerados perigosos, animais, mercadoria sem requisitos especiais. Os camiões possuem uma capacidade máxima de transporte. O preço de transporte é diferente para cada um dos tipos de camião e depende da quantidade a transportar. Mas outros fatores influenciam também o preço do transporte. No caso de um camião com capacidade de congelação, o preço varia de acordo com a temperatura de congelação exigida (definir gamas). No caso de um camião de transporte de produtos perigosos, o preço varia de acordo com o nível de perigo da mercadoria (inflamável, tóxica, …).

Os clientes são identificados por um nome e NIF e podem requisitar serviços de transporte de mercadorias. Um serviço de transporte deve especificar um ponto de origem e destino, que pode ser traduzido no tempo de ocupação do camião que efetua o transporte (tempo desde que sai até que regressa à sede da empresa). Um serviço de transporte requerido por um cliente pode obrigar ao uso de múltiplos camiões por parte da empresa, dependendo da capacidade destes. Considere que um serviço de transporte inclui o transporte de um único tipo de produtos. A escolha de qual camião atende um determinado serviço é efetuada de modo a equilibrar a ocupação de todos os camiões. O sistema deve também permitir a indicação de conclusão de um serviço (quando o camião regressa à sede da empresa).

Alguns requisitos obrigatórios são (pode e deve incluir outros que considere relevantes):

- Manutenção de camiões
- Manutenção de clientes
- Manutenção de serviços de transporte e cálculo de preços
- Associar clientes a serviços de transporte
- Associar serviços de transporte a camiões
- … e outras associações que julgue necessárias

A aplicação deve permitir registar e gerir camiões, clientes e serviços de transporte.

Sugere-se como entidades a implementar: Empresa, Camiao, Servico, Cliente, entre outras que julgue necessárias (considerar hierarquia de classes).

Usando ***estruturas de dados lineares***, deve implementar:

- Manutenção do conjunto de clientes
- Manutenção do conjunto de camiões
- Manutenção do conjunto de serviços de transporte
- Listagens várias (totais ou parciais com critérios a definir pelo utilizador, com possível ordenação) de, por exemplo:
  - clientes
  - camiões
  - serviços de transporte em curso/terminados
  - serviços de transporte requeridos por determinados clientes
  - …

*Notas:*

1. Por "manutenção de dados" entende-se as operações básicas CRUD (*Create, Read, Update, Delete*)
2. O trabalho deve respeitar o que está descrito em "Notas relativas à implementação" no texto relativo aos "Enunciados e instruções" da parte 1 do trabalho prático

# Trabalho 5 – Condomínio (Parte 1)

Desenvolva uma aplicação em C++ para gestão de um condomínio. O condomínio inclui habitações do tipo vivenda e apartamento.

Uma habitação (vivenda ou apartamento) é identificada por uma morada. Uma vivenda é caracterizada ainda por área habitacional, área exterior, e existência ou não de piscina. Um apartamento é caracterizado ainda por tipologia, área habitacional e piso. A vivenda e o apartamento possuem valores base mensal de condomínio diferentes. O valor do condomínio de uma vivenda varia também com o tamanho da área exterior e a existência de piscina. O valor do condomínio de um apartamento varia de acordo com a tipologia e piso. O sistema deve gerir o pagamento dos valores mensais do condomínio por cada habitação.

Os condóminos são identificados por um nome e NIF e podem possuir várias habitações. O condomínio dispõe ainda de diversos serviços que os condóminos podem requisitar para as suas habitações: limpeza, canalização, pintura,.... O serviço é aceite ou não, dependendo dos recursos disponíveis (suponha que o condomínio dispõe de um número limitado de prestadores de serviço).

O sistema deve permitir a requisição de serviços por parte dos condóminos, bem como a indicação de término do serviço (para libertar o prestador de serviço respetivo)

Alguns requisitos obrigatórios são (pode e deve incluir outros que considere relevantes):

- Manutenção de habitações e respetivos preços de condomínio
- Manutenção de condóminos
- Manutenção de serviços
- Associar condóminos a habitações
- Associar habitações a serviços
- … e outras associações que julgue necessárias

A aplicação deve permitir registar e gerir habitações, condóminos e serviços.

Sugere-se como entidades a implementar: Condominio, Habitacao, Servico, Condomino, entre outras que julgue necessárias (considerar hierarquia de classes).

Usando ***estruturas de dados lineares***, deve implementar:

- Manutenção do conjunto de habitações
- Manutenção do conjunto de condóminos
- Manutenção do conjunto de serviços
- Listagens várias (totais ou parciais com critérios a definir pelo utilizador, com possível ordenação) de, por exemplo:
  - habitações
  - condóminos
  - serviços em curso/terminados
  - serviços requeridos por habitação/condómino
  - valores de condomínio pagos/não pagos
  - …

*Notas:*

1. Por "manutenção de dados" entende-se as operações básicas CRUD (*Create, Read, Update, Delete*)
2. O trabalho deve respeitar o que está descrito em "Notas relativas à implementação" no texto relativo aos "Enunciados e instruções" da parte 1 do trabalho prático

# Trabalho 6 – Gestão de Correspondência (Parte 1)

Desenvolva uma aplicação em C++ para gestão de envio de correspondência. A correspondência pode ser enviada em três modalidades: correio normal, correio verde e encomenda.

Uma estação de correios pretende gerir o envio da correspondência. Qualquer correspondência é identificada por nome de remetente, morada de remetente, nome de destinatário, morada de destinatário e se é nacional ou internacional. A correspondência pode ser de 3 tipos: correio normal, correio verde e encomenda. Correio normal é também identificado por peso e modo envio (avião ou superfície). Correio Verde é também identificado por formato do volume. Encomenda é também identificada por peso e zona (depende da distância remetente/destinatário). O preço do envio da correspondência é calculado de modo diferente para cada tipo: para mais informação consultar www.ctt.pt

Os clientes usuais podem inscrever-se no sistema e passam a ser identificados por um código.

A estação de correios dispõe de um número fixo de carteiros, que devem ser afetos a tipos diferentes de correspondência. A escolha de qual carteiro é responsável por qual correspondência é efetuada de modo a equilibrar a ocupação de todos os carteiros (que tratam do mesmo tipo de correspondência).

O sistema deve permitir aceitar pedidos de envio de correspondência por parte dos clientes, bem como a indicação de receção da correspondência.

Alguns requisitos obrigatórios são (pode e deve incluir outros que considere relevantes):

- Manutenção de correspondências e respetivos preços de envio
- Manutenção de clientes
- Manutenção de carteiros
- Associar clientes a correspondências
- Associar correspondência a carteiros
- … e outras associações que julgue necessárias

A aplicação deve permitir registar e gerir correspondência, clientes e carteiros.

Sugere-se como entidades a implementar: Estacao, Correspondencia, Cliente, Carteiro, entre outras que julgue necessárias (considerar hierarquia de classes).

Usando ***estruturas de dados lineares***, deve implementar:

- Manutenção do conjunto de correspondências
- Manutenção do conjunto de clientes
- Manutenção do conjunto de carteiros
- Listagens várias (totais ou parciais com critérios a definir pelo utilizador, com possível ordenação) de, por exemplo:
  - correspondência enviada/recebida
  - clientes
  - carteiros (e correspondência de que é responsável)
  - correspondência enviada/recebida por cliente e respetivo valor
  - correspondência enviada/recebida por localidade/país
  - …

*Notas:*

1. Por "manutenção de dados" entende-se as operações básicas CRUD (*Create, Read, Update, Delete*)
2. O trabalho deve respeitar o que está descrito em "Notas relativas à implementação" no texto relativo aos "Enunciados e instruções" da parte 1 do trabalho prático

# Trabalho 7 – Campeonatos Polidesportivos (Parte 1)

Desenvolva uma aplicação em C++ para gestão de um campeonato de polidesportivo, como as Olimpíadas, onde equipas podem concorrer em diferentes desportos, como natação, corrida, hipismo, etc. Cada desporto, poderá ter uma ou mais modalidades diferentes, como corrida de 100 m, ou corrida de 400 m com barreiras, no desporto corrida.

Ao se inscreverem, as equipas indicam os desportos em que pretendem participar, com os seus respetivos atletas. Os atletas de uma equipa podem participar em um ou mais desportos, e em várias modalidades, atendendo a que as provas não se sobreponham nos calendários.

Considerando que possam haver provas de desportos diferentes em simultâneo, mas modalidades podem necessitar de uma mesma infraestrutura, modalidades de um desporto poderão não poder ser executadas em simultâneo, como no caso da natação. Cada prova terá um tempo de duração. Cada desporto terá um determinado tipo de pontuação: por exemplo, corrida poderá considerar o tempo, assim como a natação, mas o salto em altura, ou futebol, poderão ter outras pontuações, como a altura, ou o número de golos, respetivamente.

Alguns requisitos obrigatórios são (pode e deve incluir outros que considere relevantes):

- Manutenção de equipas e respectivos atletas
- Manutenção do campeonato
- Manutenção do calendário das provas
- Associar atletas de uma equipa às respectivas modalidades
- Associar provas ao calendário do campeonato
- … e outras associações que julgue necessárias

A aplicação deve permitir registar e gerir equipas, atletas e provas.

Sugere-se como entidades a implementar: Campeonato, Desporto, Modalidade, Equipa, Atleta, Calendário, entre outras que julgue necessárias (considerar hierarquia de classes).

Usando ***estruturas de dados lineares***, deve implementar:

- Manutenção do conjunto de equipas
- Manutenção do conjunto de atletas
- Manutenção do conjunto de provas
- Listagens várias (totais ou parciais com critérios a definir pelo utilizador, com possível ordenação) de, por exemplo:
  - provas realizadas ou por realizar
  - atletas, por equipas, modalidade, desporto
  - colocações (primeiros, segundos, e terceiros lugares)
  - desempenho das equipas, pela colocação dos seus atletas
  - desempenho dos atletas, pelas suas colocações nas diversas provas em que participa
  - …

*Notas:*

1. Por "manutenção de dados" entende-se as operações básicas CRUD (*Create, Read, Update, Delete*)
2. O trabalho deve respeitar o que está descrito em "Notas relativas à implementação" no texto relativo aos "Enunciados e instruções" da parte 1 do trabalho prático

# Trabalho 8 – Oficina Mecânica (Parte 1)

Desenvolva uma aplicação em C++ para gestão de uma oficina mecânica. O sistema deve conter informação sobre veículos e serviços efetuados, clientes e funcionários.

A oficina aceita efetuar serviços sobre diferentes tipos de veículos: automóveis, motorizadas, camiões, autocarros,….

Todos os veículos possuem um funcionário responsável que responde perante o cliente sobre os serviços efetuados no seu veículo. Procure manter uma distribuição equilibrada de veículos a funcionários responsáveis.

A oficina oferece serviços:

- *standard*: revisão, mudança óleo, limpeza… A descrição, preço e duração possuem valores pré-definidos. Pode considerar outros atributos, se necessário.
- não *standard*. A descrição, preço e duração são valores a definir quando da criação do serviço.

Alguns requisitos obrigatórios são (pode e deve incluir outros que considere relevantes):

- Manutenção de clientes
- Manutenção dos veículos
- Manutenção de serviços *standard*
- Manutenção de funcionários
- Associar funcionários a veículos (um funcionário pode ser responsável por mais que um veículo)
- Associar clientes a veículos (um cliente pode possuir mais que um veículo)
- Associar veículos a serviços (um veículo pode ter sido sujeito a mais que um serviço)

A aplicação deve permitir registar e gerir toda a informação relativa a quais serviços foram efetuados pela oficina e quando.

Considere o conjunto de algumas das entidades implementadas: Cliente; Veiculo (entidade no topo da hierarquia de veículos); Funcionário.

Usando ***estruturas de dados lineares***, deve implementar:

- Manutenção do conjunto de clientes
- Manutenção do conjunto de veículos
- Manutenção do conjunto de funcionários
- Manutenção do conjunto de serviços *standard*
- Listagens várias (quer totais, quer parciais com critérios a definir pelo utilizador) de, por exemplo:
  - clientes
  - veículos
  - serviços efetuados, serviços efetuados por veículo, serviços efetuados por cliente
  - …

*Notas:*

1. Por "manutenção de dados" entende-se as operações básicas CRUD (Create, Read, Update, Delete)
2. O trabalho deve respeitar o que está descrito em "Notas relativas à implementação" no texto relativo aos "Enunciados e instruções" da parte 1 do trabalho prático

# Trabalho 9 – Agência de Viagens (Parte 1)

Desenvolva uma aplicação em C++ para a gestão de informação numa agência de viagens. A agência mantém uma carteira de clientes, que podem ser particulares ou comerciais, a quem divulga viagens, a depender do tipo de cliente e, possivelmente, do seu histórico de destinos e preferências. O produto básico da agência é uma viagem, que incluirá um itinerário, com vários troços (origem, destino, tipo de transporte), e opcionalmente alojamento (em hotel, aparthotel, etc). A viagem pode ser adquirida como um pacote (promocional, pré-definido pela agência) ou constituído de acordo com as necessidades do cliente.

O cliente comercial, como habitualmente efetua viagens de negócio e em grupo, é caracterizado pelo nº de viagens efetuado e nº médio de participantes. O preço de uma viagem, para um cliente comercial, pode sofrer um desconto de: 5%, se o nº de viagens > 5 e nº médio de participantes > 10; 10%, se nº viagens >5 e nº médio de participantes > 15.

A aplicação deve registar e gerir toda a informação necessária à identificação das viagens (promocionais ou não) da agência, dos seus destinos, meios de transportes disponíveis e opções de alojamento. Os clientes devem poder aceder, através da aplicação, ao seu histórico de viagens, selecionar pacotes ou compor viagens, a partir da composição de itinerários.

O sistema a implementar deve registar toda a informação administrativa e contabilística da agência de viagens, com os seguintes requisitos base (pode e deve incluir outros que considere relevantes):

- Manutenção dos clientes, particulares e comerciais
- Manutenção dos itinerários e opções de transportes
- Manutenção das opções de alojamento em cada destino
- Manutenção dos pacotes promocionais
- Associar itinerários a transportes e alojamentos
- Associar clientes a viagens
- … e outras associações que julgue necessárias

A aplicação deve permitir registar e gerir clientes, viagens e itinerários.

Sugere-se como entidades a implementar: Cliente, Viagem, Itinerario, Pacote, entre outras que julgue necessárias (considerar hierarquia de classes).

Usando ***estruturas de dados lineares***, deve implementar:

- Manutenção do conjunto de pacotes promocionais
- Manutenção do conjunto de destinos e opções de alojamento
- Manutenção do conjunto de itinerários
- Manutenção do conjunto de clientes e histórico de viagens
- Listagens várias (totais ou parciais com critérios a definir pelo utilizador, com possível ordenação) de, por exemplo:
  - destinos disponíveis e opções de alojamento por destino
  - clientes (e histórico de viagens com preços)
  - itinerários e pacotes promocionais
  - …

*Notas:*

1. Por "manutenção de dados" entende-se as operações básicas CRUD (*Create, Read, Update, Delete*)
2. O trabalho deve respeitar o que está descrito em "Notas relativas à implementação" no texto relativo aos "Enunciados e instruções" da parte 1 do trabalho prático

# Trabalho 10 – Hipermercado (Parte 1)

Desenvolva uma aplicação em C++ para a gestão de informação relativa a encomenda e respetivos fornecedores num hipermercado. O hipermercado mantém uma carteira de fornecedores, que podem ser fornecedores em nome individual ou fornecedores empresa. Os fornecedores em nome individual vendem todos os seus produtos com base num preço unitário (unidade/litro/kilo/…). Os fornecedores empresa vendem os seus produtos a um preço que depende da quantidade vendida (deve definir patamares), e estipulam, para cada produto, um valor mínimo da quantidade a vender.

O fornecedor, além da informação que o identifica (designação, NIF, morada, …), inclui informação sobre os produtos que possui para venda, stock e preços.

Quando o hipermercado solicita a aquisição de produtos (pedido de encomenda), o sistema deve escolher os fornecedores mais baratos para a compra das quantidades que se pretendem de cada produto do pedido. Pode ser necessário efetuar mais que uma encomenda, se for preferível comprar a fornecedores diferentes (até para o mesmo produto). Uma encomenda pode incluir vários produtos, mas refere-se a um único fornecedor.

O sistema a implementar deve registar toda a informação relativa a encomendas do hipermercado e respetivos fornecedores, com os seguintes requisitos base (pode e deve incluir outros que considere relevantes):

- Manutenção dos fornecedores e respetivos produtos
- Manutenção dos pedidos de encomenda (pedidos do hipermercado)
- Manutenção das encomendas (encomendas a fornecedores)
- Associar pedidos a encomendas
- Associar encomendas a fornecedores
- … e outras associações que julgue necessárias

A aplicação deve permitir registar e gerir pedidos de encomendas, encomendas e fornecedores.

Sugere-se como entidades a implementar: Fornecedor, Produto, PedidoEncomenda, Encomenda, entre outras que julgue necessárias (considerar hierarquia de classes).

Usando ***estruturas de dados lineares***, deve implementar:

- Manutenção do conjunto de fornecedores
- Manutenção do conjunto de produtos de um fornecedor
- Manutenção do conjunto de pedidos de encomenda
- Manutenção do conjunto de encomendas
- Listagens várias (totais ou parciais com critérios a definir pelo utilizador, com possível ordenação) de, por exemplo:
  - encomendas, com informação sobre fornecedor, produtos encomendados e preços
  - fornecedores e respetivos produtos
  - produtos, com informação sobre preços
  - …

*Notas:*

3. Por "manutenção de dados" entende-se as operações básicas CRUD (*Create, Read, Update, Delete*)
4. O trabalho deve respeitar o que está descrito em "Notas relativas à implementação" no texto relativo aos "Enunciados e instruções" da parte 1 do trabalho prático