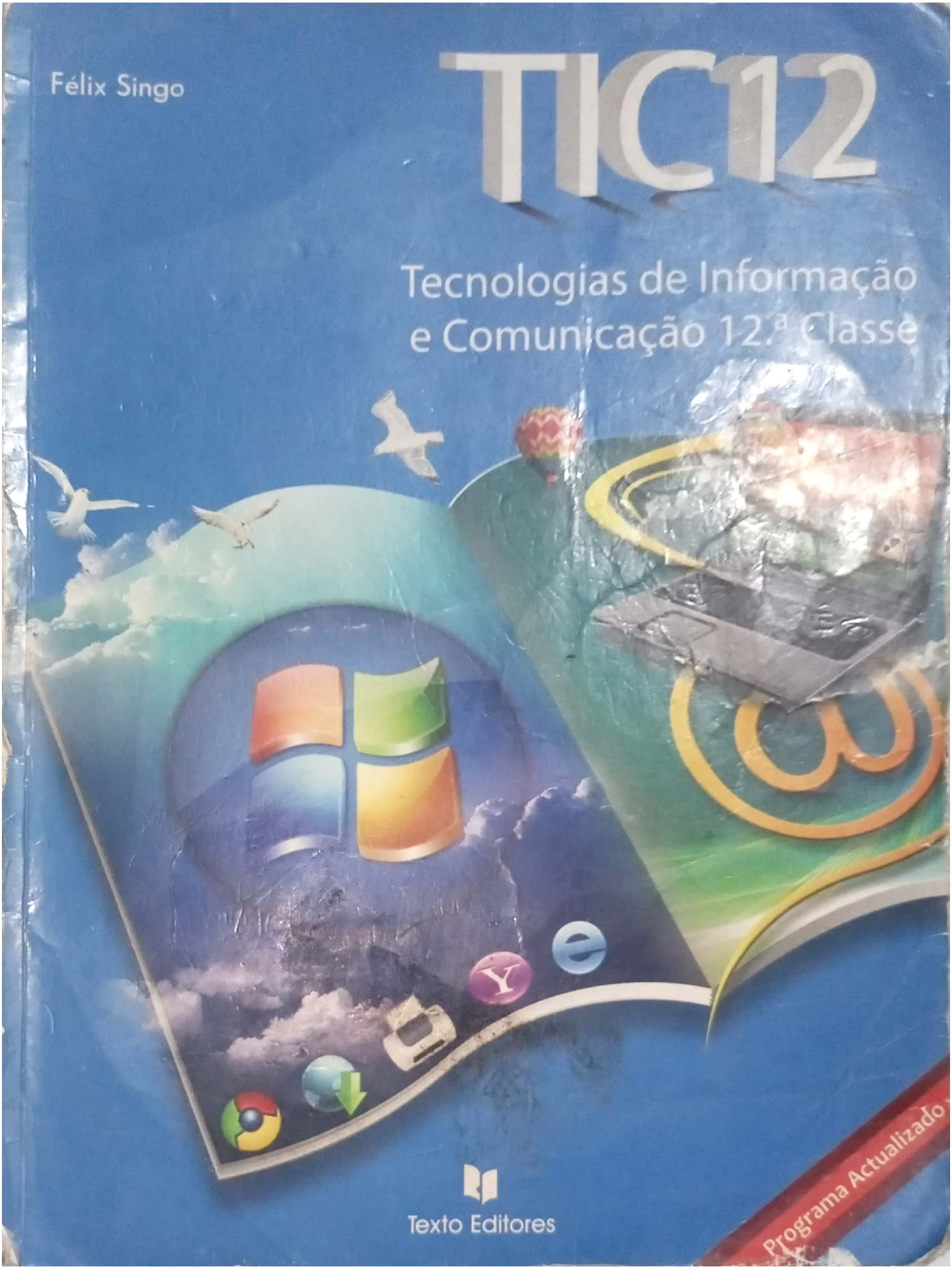
Félix Singo

TIC12

Tecnologias de Informação e Comunicação 12.ª Classe



Programa Actualizado

Texto Editores

f i c h a t é c n i c a

título	**TIC12 • Tecnologias de Informação e Comunicação 12.ª Classe**
autor	**Félix Singo**
coordenação	**Stella Morgadinho**
editor	**Texto Editores, Lda. – Moçambique**
capa e aberturas	**Darlene Mavale**
ilustrações	**Darlene Mavale**
arranjo gráfico	**Darlene Mavale**
paginação	**Darlene Mavale**
pré-impressão	**Texto Editores, Lda. – Moçambique**
impressão e acabamentos	**Texto Editores, Lda.**

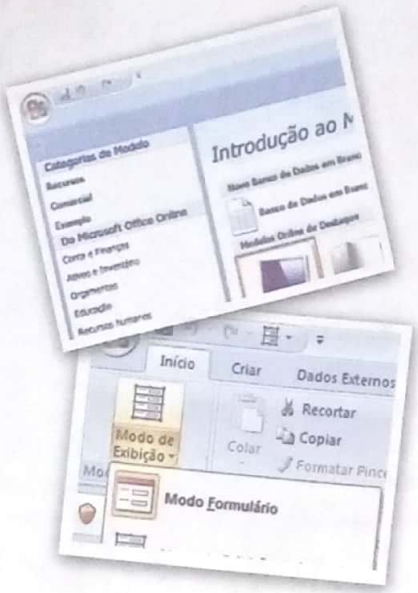
Texto Editores

Avenida Julius Nyerere, 46 • Bairro Polana • Cimento B • Maputo • Moçambique
Tels. (+258) 21 49 86 48 • 21 49 90 71 Fax: 21 49 86 48
E-mail: info@me.co.mz

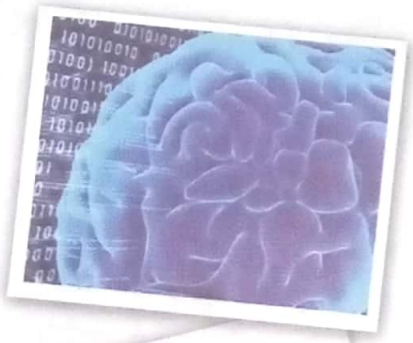
MAPUTO, Março de 2014 • 2.ª EDIÇÃO • 1.ª TIRAGEM • REGISTADO NO INLD SOB O NÚMERO: 5889/RLINLD/08

Índice

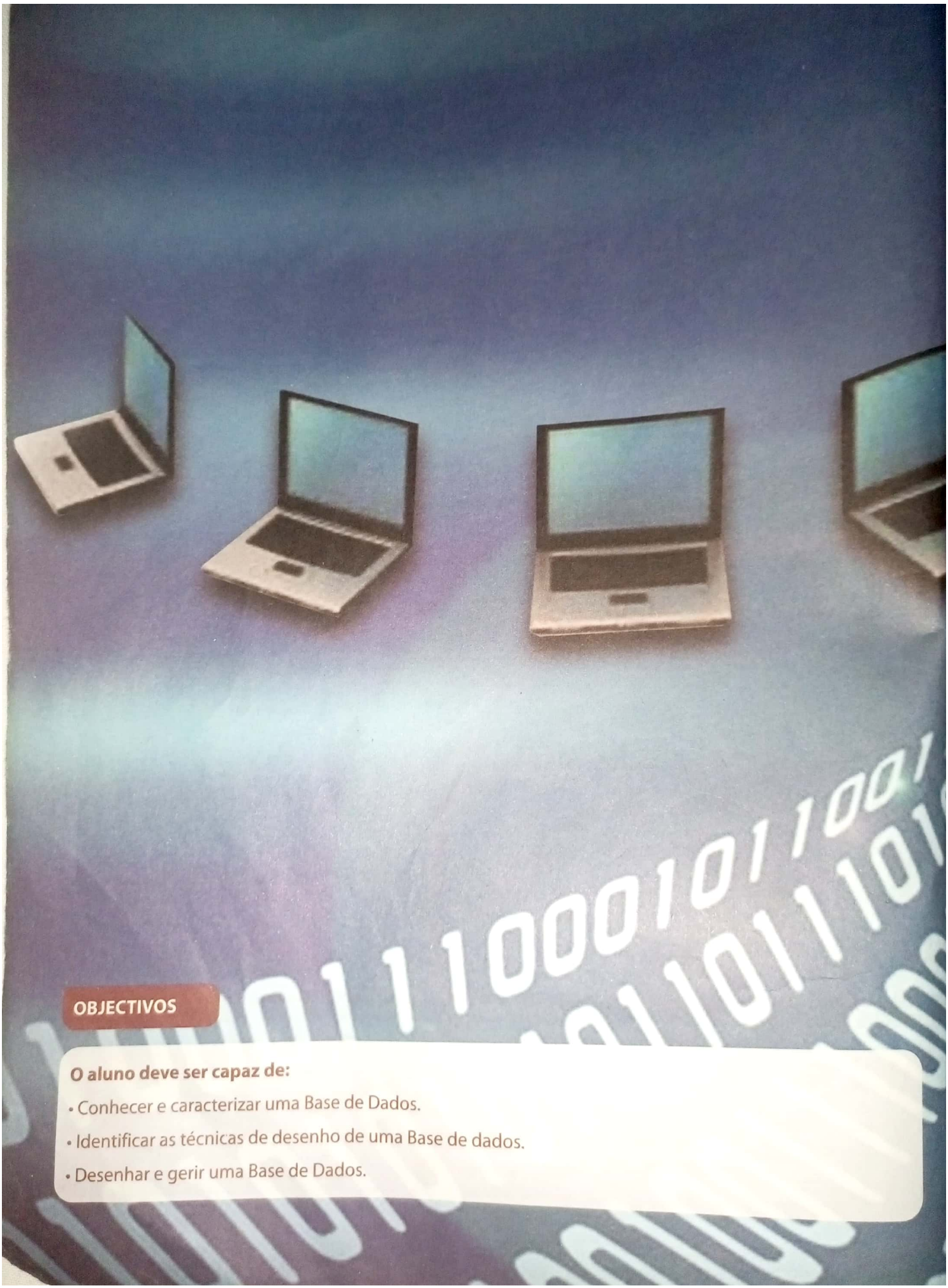
Unidade 1: Introdução às Bases de Dados



Unidade 2: Introdução à programação e segurança do computador



Unidade 3: Trabalho de Projecto



OBJECTIVOS

O aluno deve ser capaz de:

- Conhecer e caracterizar uma Base de Dados.
- Identificar as técnicas de desenho de uma Base de dados.
- Desenhar e gerir uma Base de Dados.

UNIDADE 1

Introdução às Bases de Dados

CONTEÚDOS

Conceitos básicos

- Conceito de Bases de Dados

Introdução à Base de Dados

- Formas de organização de uma Base de Dados
- Principais utilizações de uma Base de Dados

Criação e gestão de Bases de Dados

- Sistemas de gestão de Bases de Dados
- Modelos de Bases de Dados

Págs. 4 a 41

Introdução

Tal como muitas outras tecnologias informáticas, os fundamentos de bases de dados relacionais têm as suas raízes na empresa IBM, nas décadas de 60 e 70, do século passado, através de pesquisas então levadas a cabo com o intuito de automatizar actividades buróticas, isto é, de escritório. Foi durante um período da história na qual as empresas, na luta pela optimização dos seus recursos, descobriram que era muito dispendioso empregar um elevado número de pessoas apenas para fazerem trabalhos rotineiros como armazenar e organizar arquivos.

Algumas empresas perceberam que era altura e que valia a pena fazer mais algum investimento na pesquisa de um meio mais barato e mais eficiente para realizar aquelas actividades (quase) automaticamente.

Durante aquele período foram conduzidas muitas pesquisas nesse sentido, cujos resultados **espelham** os diferentes modelos de bases de dados que hoje conhecemos, dentre os quais podemos citar os modelos hierárquicos, os modelos de rede, os modelos relacionais (amplamente usados) e os modelos orientados para objectos, bem como muita outra tecnologia utilizada hoje em dia.

Em 1970 um pesquisador da IBM – Ted Codd – publicou o primeiro artigo sobre **Bases de Dados Relacionais**. Esse artigo tratava do uso de cálculo e álgebra relacional para permitir que os usuários não técnicos armazenassem e recuperassem grandes quantidades de informação. Codd visionava um sistema onde o usuário seria capaz de aceder às informações através de comandos escritos em língua inglesa, onde as informações estariam armazenadas em tabelas. Era a visão daquilo que hoje se designa **Base de Dados Relacionais (*data base*)**.

O que é uma Base de Dados (*Data Base*)?

Uma **Base de Dados** (BD) é um conjunto de dados interrelacionados e armazenados em algum dispositivo. Por exemplo, pode ser uma lista de material existente num armazém, endereços, dados dos empregados, informações sobre clientes ou facturas. Por «dados» podemos entender «factos conhecidos» que podem ser armazenados e que possuem um significado implícito.

Os dados, numa base de dados, estão organizados segundo uma estrutura e interligados, tendo em vista:

- Serem partilhados por programas de diferentes aplicações e em ambientes diferentes.
- Não permitir a redundância ou repetição de informação.
- Manter a sua integridade e protecção.
- A eficácia do sistema.

Porém, o significado do termo «base de dados» é mais restrito que a definição dada acima. Uma base de dados possui as seguintes propriedades:

- É uma colecção lógica e coerente de dados com um significado inerente.
- Uma disposição desordenada dos dados não pode ser referenciada como uma base de dados.
- É projectada, construída e povoada com dados para um propósito específico.
- Possui um conjunto pré-definido de usuários e aplicações.
- Representa algum aspecto do mundo real, que normalmente é designado «minimundo». Qualquer alteração efectuada no «minimundo» é automaticamente reflectida na base de dados.

Uma base de dados pode ser criada e mantida por um conjunto de aplicações desenvolvidas especialmente para esta tarefa ou por um «Sistema Gerenciador de Banco de Dados (SGBD).»

Um SGBD é, assim, uma colecção de programas que permitem aos usuários criar e manipular uma base de dados. Um SGBD é um sistema de *software* de propósito geral que facilita o processo de definir, construir e manipular bases de dados de diversas aplicações.

A **definição** de uma base de dados envolve a especificação de tipos de dados a serem armazenados na base de dados.

A **construção** de uma base de dados é o processo de armazenar os dados em algum meio que seja controlado pelo SGBD.

A **manipulação** uma base de dados indica a utilização de funções como a de consulta, recuperação de dados específicos, modificação da base de dados para reflectir mudanças no «minimundo» (inserções, actualizações e remoções), e geração de relatórios.

O conjunto formado por uma base de dados incluindo as aplicações que manipulam o mesmo é chamado «**Sistema de Base de Dados**». O quadro seguinte apresenta um esquema genérico de um Sistema de Base de Dados na sua interacção com os seus usuários.

1 - ate 10 Base de Dados
Sistema de Gestão de Base de Dados

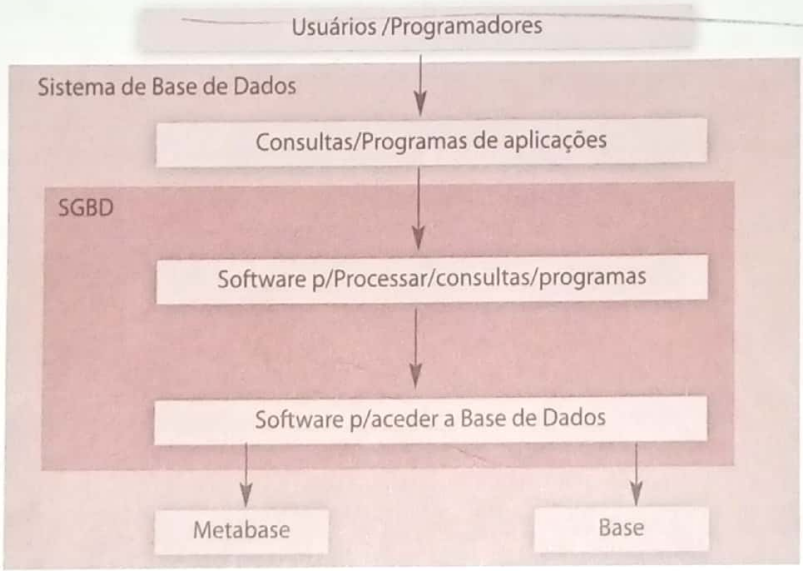


Fig. 1.1 Esquema genérico de um Sistema de Base de Dados

Síntese

Sistema de Base de Dados

É basicamente um sistema informático cujo propósito geral é armazenar informações e permitir ao utilizador buscar e actualizar essas informações quando solicitadas. As informações em questão podem ser qualquer coisa que tenha significado para o indivíduo ou para a organização a que o sistema deve servir.

Ficheiros de dados, registos e campos

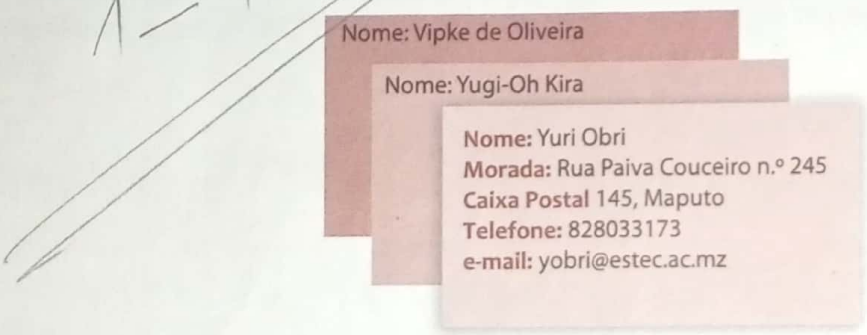
Nos sistemas informáticos a informação é organizada fundamentalmente sob a forma de **ficheiros**. Estes ficheiros podem ser de vários tipos. Sob o ponto de vista que nos interessa, os ficheiros podem corresponder a dois tipos bem distintos de informação:

- **Programas** – que são ficheiros que armazenam instruções, procedimentos ou rotinas para executar em computador.
- **Ficheiros de dados** – que correspondem à informação produzida e manipulada pelos utilizadores, como, por exemplo, documentos de texto, imagens e folhas de cálculo.

Numa base de dados podemos ter um ou mais ficheiros de dados, os quais são manipulados por um ou mais programas de aplicação.

Um ficheiro de dados de uma BD pode conceber-se como arquivo de fichas (ver quadro seguinte), onde cada ficha corresponde a um registo de uma entidade ou objecto (produto, empresa, pessoa,...).

Cada **registo** contém um determinado conjunto de campos de informação, correspondentes aos atributos com que definimos as entidades. A uma entidade podem corresponder vários registos.



Nome	Morada	Caixa Postal	Telefone	e-mail
Yuri Obri	Paiva Couceiro	245	828033173	yobri@gmail.com
Yugi-Oh Kira	Infulene Valley	109	848299320	yugioh@gmail.com
Vipke d'Oliveira	B. Aeroporto	505	828668097	chipate@gmail.com

Fig. 1.2 Organização da informação

Quando se define a estrutura de um registo há que ter em conta:

- A entrada da informação
- A saída de informação
- O processamento da informação
- A flexibilidade, ou seja, a facilidade de adaptação à evolução das necessidades

Os ficheiros de bases de dados que acabámos de referir, em muitos casos, podem representar-se por tabelas, em que as linhas contêm registos e as colunas definem os campos.

Aplicação de Bases de Dados

Já se fez referência aos diversos campos de actividade onde são aplicadas necessariamente as bases de dados, mas, mesmo assim, em jeito de resumo, pode-se dizer que as bases de dados são amplamente usadas em:

- **Bancos:** para informações de clientes, contas, empréstimos e todas as transacções bancárias.
- **Linhas aéreas:** para reservas e informações de horários. As linhas aéreas foram umas das primeiras a usar bases de dados de maneira geograficamente distribuída.
- **Universidades:** para informações de alunos, registos de cursos e notas.
- **Transacções de cartão de crédito:** para compras com cartões de crédito e geração de facturas mensais.
- **Telecomunicações:** para manter registos de chamadas realizadas, gerar cobranças mensais, manter saldos de cartões de chamada pré-pagos e armazenar informações sobre as redes de comunicações.
- **Finanças:** para armazenar informações sobre valores mobiliários, vendas e compras de instrumentos financeiros como acções e títulos; também para armazenar dados de mercado em tempo real a fim de permitir negócios «on-line» por clientes e transacções automatizadas pelas empresas.
- **Vendas:** para informações de clientes, produtos e compras.
- **Revendedores *on-line*:** para dados de vendas descritos aqui, além do acompanhamento de pedidos, de geração de lista de recomendações personalizadas e de manutenção de avaliações de produtos *on-line*.
- **Indústria:** para a gestão da cadeia de suprimento, controlar a produção de itens nas fábricas, e stokes de itens em armazéns e lojas.
- **Recursos humanos:** para informações sobre funcionários, salários, descontos em folhas de pagamento, benefícios e para a geração de contra-cheques.

Síntese

Sistema de Informação

O valor da informação depende da sua actualidade, pois só com base em informações actuais se podem tomar decisões acertadas. Os sistemas devem, pois, fornecer informação actual e conter mecanismos de actualização eficiente da informação.

Conceitos de SGBD – Sistema de Gestão de Bases de Dados

Para criar e gerir bases de dados são necessários programas específicos, normalmente designados SGBD – Sistemas de Gestão de Bases de Dados (ou DBMS – *Database Management Systems*).

Os **SGBD** são programas que permitem criar e manipular bases de dados, em que os dados são estruturados com independência relativamente aos programas de aplicação que os manipulam.

A independência dos dados num SGBD significa que é possível alterar a estrutura dos dados, sem que isso implique, necessariamente, reformular o programa que opera com os dados (o SGBD).

Exemplo de **Sistemas de Gestão de Bases de Dados** comerciais:

- *Microsoft Access*
- *FoxPro*
- *Informix*
- *Oracle*
- *Microsoft SQL Server*
- *PostGreSQL*
- *MySQL*
- *Firebird*, etc.

Sistemas de ficheiros *vs* SGBD

Podemos manter a informação utilizando sistemas de ficheiros ou SGBD's. Mas deve-se ter em conta que a quantidade de informação tem tendência a ser grande e a aumentar com o tempo, tornando-se quase impossível mantê-la em memória, o que requer a utilização de dispositivos de armazenamento de grande capacidade. A utilização que se pretende dar à informação é determinante na decisão da utilização de um SGBD.

Vantagens de SGBD

- Independência dos dados
- Acesso eficiente aos dados
- Redução do tempo de desenvolvimento de aplicações
- Integridade e segurança dos dados
- Administração dos dados
- Acesso concorrente e recuperação de falhas

Quando não se deve utilizar um SGBD

Em algumas situações, o uso de um SGBD pode representar uma carga desnecessária aos custos quando comparado com a abordagem de processamento tradicional de arquivos, como, por exemplo:

- **Alto investimento inicial** na compra de *software* e *hardware* adicionais
- **Generalidade** que um SGBD fornece na definição e processamento de dados.
- **Sobrecarga** na provisão de controlo de segurança, controlo de concorrência, recuperação e integração de funções

Problemas adicionais podem surgir caso os projectistas da base de dados ou os administradores da base de dados não elaborem os projectos correctamente ou se as aplicações não são implementadas de forma apropriada. Se o DBA não administrar a base de dados de forma apropriada, tanto a segurança como a integridade dos sistemas podem ser comprometidas. A sobrecarga causada pelo uso de um SGBD e a má administração justificam a utilização da abordagem de processamento tradicional de arquivos nos seguintes casos:

- A base de dados e as aplicações são simples, bem definidas e não se espera mudanças no projecto.
- A necessidade de processamento em tempo real de certas aplicações, que são terrivelmente prejudicadas pela sobrecarga causada pelo uso de um SGBD.
- Não haverá múltiplo acesso à base de dados.

Visão de dados

Um SGBD é uma **colecção de arquivos** e **programas interrelacionados** que permitem aos utilizadores o acesso a consultas e alterações dos dados. O maior benefício de uma base de dados é o de proporcionar ao utente uma visão abstracta de dados. Isto é, o sistema acaba por ocultar determinados detalhes sobre a forma de armazenamento e manutenção de dados.

Abstracção de dados

Para que se possa usar um sistema, ele precisa de ser eficiente na recuperação de informações. Esta eficiência está relacionada com a forma como foram projectadas as complexas estruturas de representação desses dados na base de dados. Já que muitos dos utilizadores dos sistemas de bases de dados não são treinados em computação, os técnicos em desenvolvimento de sistemas omitem essa complexidade aos utentes por meio dos chamados **níveis de abstracção** (fig. 1.3), de modo a facilitar a interacção dos usuários com o Sistema. Assim, tem-se:

- **Nível físico** ou esquema interno – é o mais baixo nível de abstracção que descreve como os dados estão de facto armazenados – estrutura de dados e ficheiros usados.
- **Nível lógico** ou esquema conceptual – este nível médio de abstracção descreve os dados que estão a ser armazenados na base de dados e quais os interrelacionamentos existentes entre eles. Assim, a base de dados como um todo é descrito em termos de um número relativamente pequeno de estruturas simples. Este nível compreende todos os dados, atributos, relações, restrições sobre os dados, informação semântica dos dados, informação de integridade e segurança.
- **Nível de visão** ou esquema externo – o mais alto nível de abstracção descreve apenas parte da base de dados, portanto, ocupando-se do modo como os dados são vistos pelos utilizadores individuais, é uma descrição personalizada e parcial dos dados. Cada vista externa inclui as entidades, atributos e relações que são relevantes para o utilizador ou departamento em causa. Na verdade, muitos dos usuários da base de dados não precisam de conhecer todas as suas informações. Pelo contrário, os usuários utilizam normalmente apenas parte da base de dados.

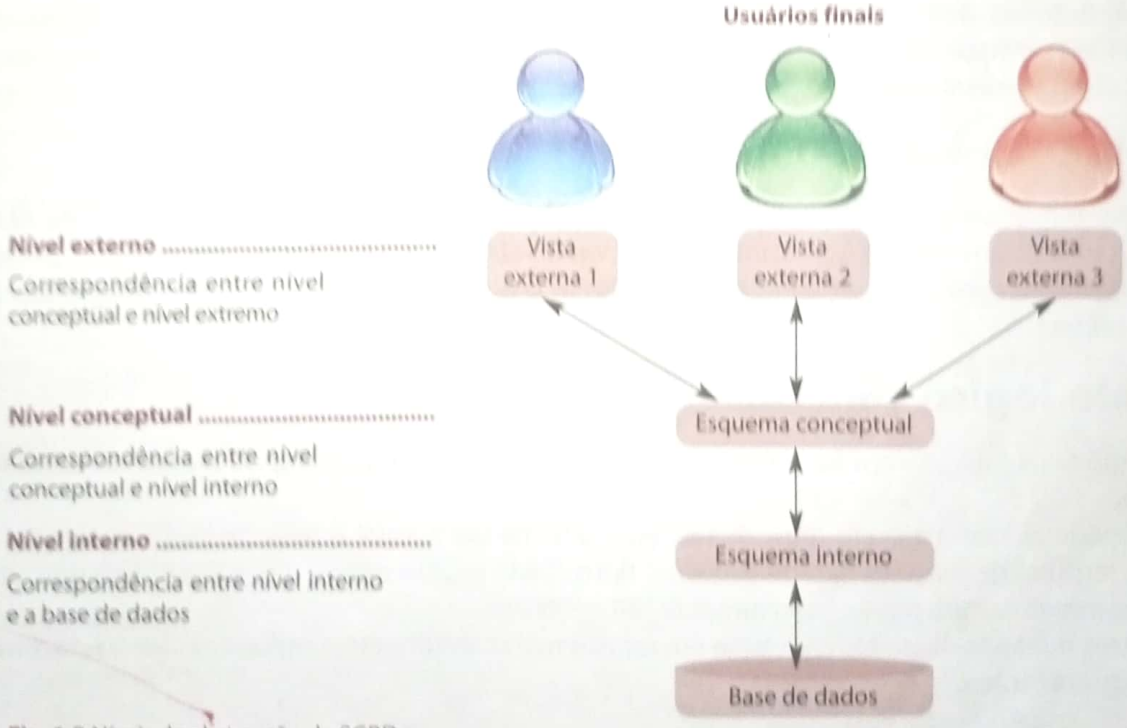


Fig. 1.3 Níveis de abstracção do SGBD

Síntese

Um SGBD é um sistema que permite:

- Armazenar e manipular grandes quantidades de informação.
- Especificar os tipos, as estruturas e as restrições dos dados a serem armazenados.
- Armazenar os dados num meio de armazenamento que é controlado pelo próprio DBMS.
- Manipular os dados através de funções de interrogação, actualização, etc.
- Aceder simultaneamente, por vários utilizadores e/ou programas à Base de Dados.

21 – 30

Modelos de dados

Sob a estrutura da base de dados está o modelo de dados: um conjunto de ferramentas conceituais usadas para a descrição de dados, relacionamentos entre dados, semântica de dados e regras de consistência. Os mais comuns são **modelos lógicos com base em objectos** e **modelos lógicos com base em registos.**

Modelo lógico com base em objectos

Os **modelos lógicos** com base em objectos são usados na descrição de dados no nível lógico e de visões. Existem vários modelos nessa categoria, e outros ainda estão por surgir. Alguns são amplamente conhecidos, como:

- Modelo Entidade-Relacionamento.
- Modelo Orientado para o Objecto.

Modelo Entidade-Relacionamento

O **Modelo** (de dados) **Entidade-Relacionamento** (E/R) tem por base a **percepção do mundo** real como um conjunto de objectos básicos, chamados entidades, e dos relacionamentos entres eles. Uma entidade é uma «coisa» ou um «objecto» do mundo real que pode ser identificado por outros objectos.

Modelo Orientado a Objectos

Como o modelo E/R, o **Modelo Orientado a Objectos** tem por base um **conjunto de objectos.** Um objecto contém valores armazenados em **variáveis instâncias** dentro do objecto. Um objecto também contém conjuntos de códigos que operam o objecto. Tais conjuntos de códigos são chamados **operações.**

Modelo lógico com base em registos

Os modelos lógicos com base em registos são usados **para descrever os dados no nível lógico e de visão.**

Os modelos com base em registos são assim chamados porque a base de dados é estruturada por meio de registos de formato fixo de todos os tipos. Cada registo define um número fixo de campos ou atributos, e cada campo possui normalmente tamanho fixo.

Os três modelos de dados com base em registo mais comummente utilizados são: o **relacional,** o de **rede** e o **hierárquico.**

Modelo relacional

O **modelo relacional** usa um conjunto de tabelas para representar tanto os dados como a relação entre eles, quer dizer, a estrutura fundamental do modelo relacional é a relação (tabela) constituída por um ou mais atributos (campos) que traduzem o tipo de dados a armazenar. Cada instância do esquema (linha) é chamada **tupla** (registo). As tabelas abaixo apresentam um exemplo de base de dados relacional em duas tabelas: uma mostra os clientes do banco e outra as suas contas.

R_Cliente	N_Cliente	M_Cliente	Nr_Conta
12345	Yuri Obri	Matola	101
24576	Vipke d'Oliveira	Maputo	201
12897	Wilherm d'Oliveira	Maputo	201
10784	Nahedade Rissani	Beira	401
10784	Nahedade Rissani	Beira	502

Nr_Conta	Saldo
101	500
201	400
401	300
502	600

Fig. 1.4 Um exemplo de Base de Dados Relacional

Modelo de rede

Os dados no modelo de rede são representados por um conjunto de registos e as relações entre estes registos são representadas por *links* (ligações), as quais podem ser vistas por ponteiros. No modelo em rede, os **registos são organizados em grafos onde aparece um único tipo de associação** (*set*) que define uma relação 1:N entre 2 tipos de registos: **proprietário** e **membro**. O diagrama para representar os conceitos do modelo em redes consiste em dois componentes básicos: **caixas**, que correspondem aos registos e **linhas**, às associações. A figura seguinte ilustra um exemplo de **diagrama do modelo em rede**.

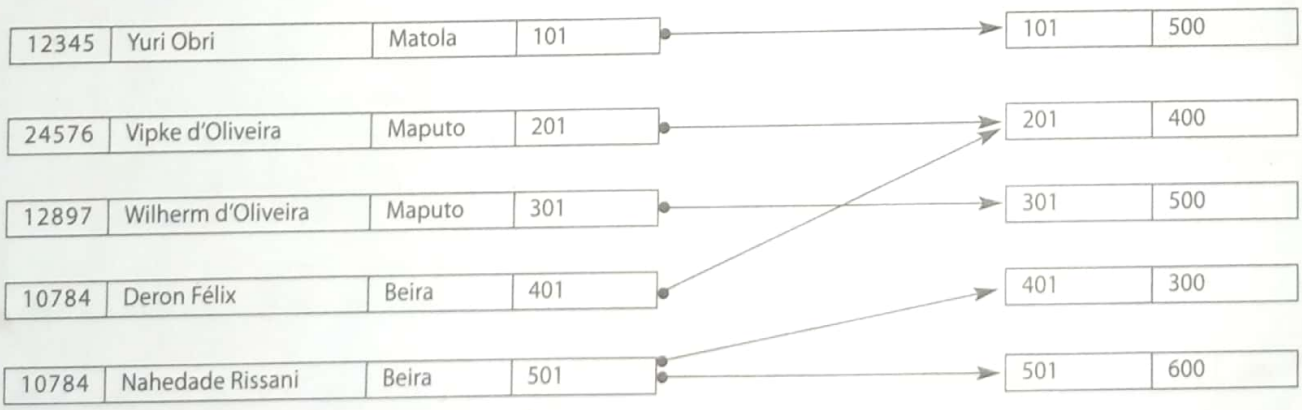


Fig. 1.5 Um exemplo de diagrama do modelo em rede

Modelo hierárquico

O **modelo hierárquico** foi o **primeiro a ser reconhecido como um modelo de dados**. O seu desenvolvimento somente foi possível devido à consolidação dos discos de armazenamento endereçáveis, pois esses discos possibilitaram a exploração da sua estrutura de endereçamento físico para viabilizar a representação hierárquica de informações. Neste modelo de dados, estes são estruturados em hierarquias ou árvores. Os **nós** das hierarquias contêm ocorrências de registos, onde cada registo é uma colecção de campos – **atributos** – cada um contendo apenas uma informação. O registo da hierarquia que precede outros é o **registo-pai**, os outros são chamados **registos-filhos**. Portanto, o modelo hierárquico é similar ao modelo em rede, pois os dados e as suas relações são representados, respectivamente, por registos e *links*. A diferença é que, no modelo hierárquico, os registos estão organizados em árvores e não em grafos arbitrários.

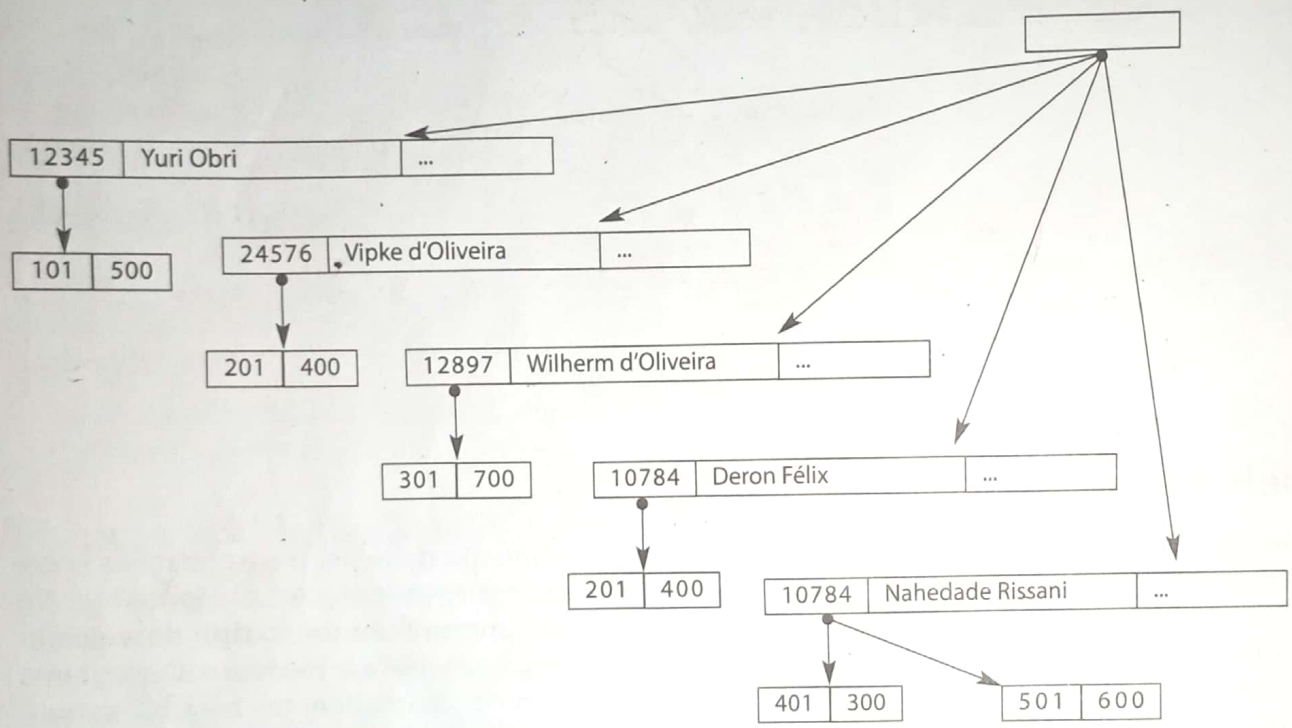


Fig. 1.6 Um exemplo de base de dados em rede.

Síntese

- **Modelos de Dados**
  - Modelo lógico com base em objectos
    - Modelos Entidade Relacionamento
    - Modelos Orientado a Objectos
  - Modelo lógico com base em registos
    - Modelos relacional
    - Modelos de rede
    - Modelos hierárquico

Classificação dos SGBD

Existem diferentes tipos de SGBD, que se classificam de acordo com os seguintes critérios:

- Quanto ao número de máquinas onde a base de dados está armazenada:
  - **Centralizada ou localizada:** quando todos os dados estão numa máquina (ou num disco).
  - **Distribuída** (homogénea ou heterogénea): quando os dados estão distribuídos por diversas máquinas (ou diversos discos).
- Quanto ao número de utilizadores que o sistema é capaz de suportar:
  - **Pessoal** (*single user*) – utilizado em computadores pessoais.
  - **Multiutilizador** – utilizado em estações de trabalho, minicomputadores e máquinas de grande porte.

Arquitectura

A **arquitectura** de um sistema de base de dados é fortemente influenciada pelo sistema básico computacional sobre o qual o sistema de base de dados vai ser executado. Pode-se ter quatro arquitecturas básicas:

Sistemas centralizados

São executados sobre um único sistema computacional (*standalone*) e não interagem com mais nenhum outro sistema.

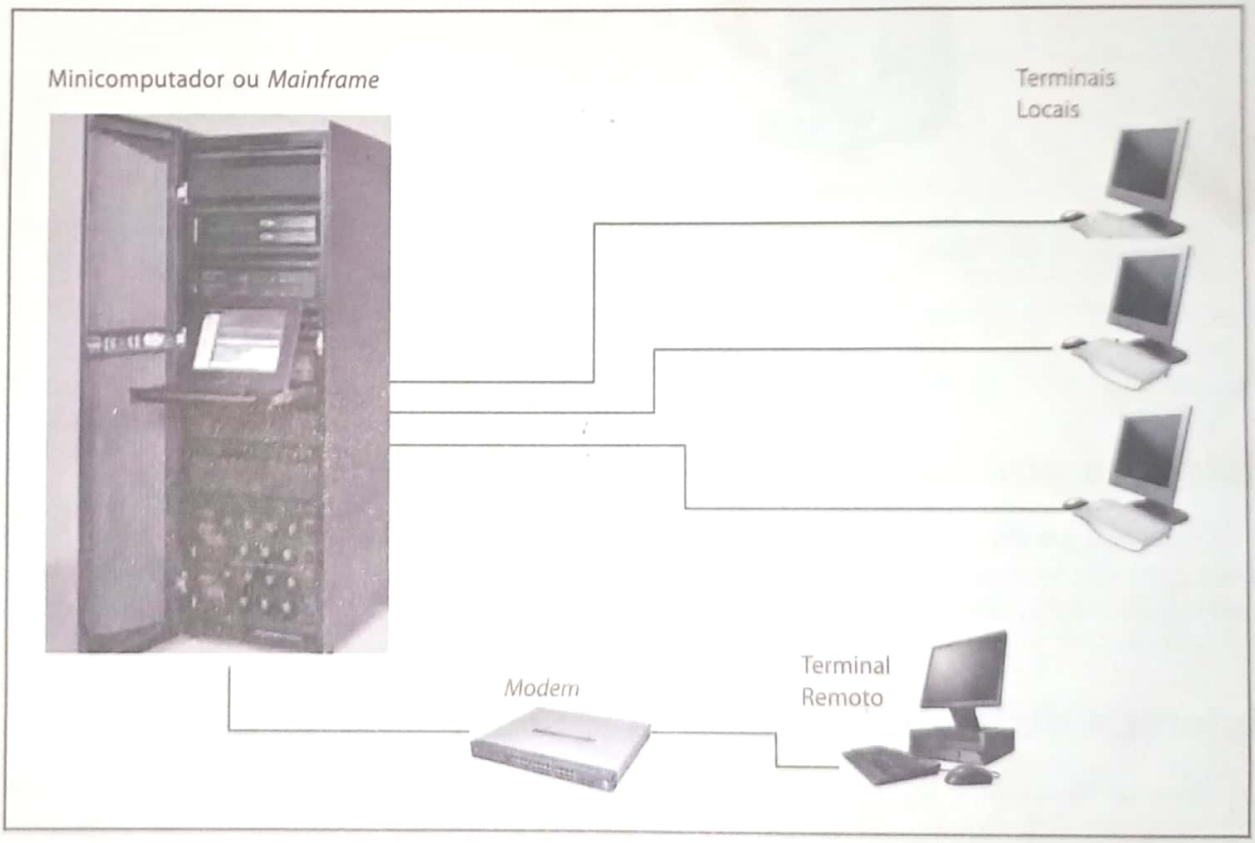


Fig. 1.7 Arquitectura centralizada

Sistemas Cliente-Servidor

O **cliente** (*front-end*) executa as tarefas do aplicativo, ou seja, fornece a *interface* do usuário (tela e processamento de entrada e saída). O **servidor** (*back-end*) executa as consultas no SGBD e retorna os resultados ao cliente.

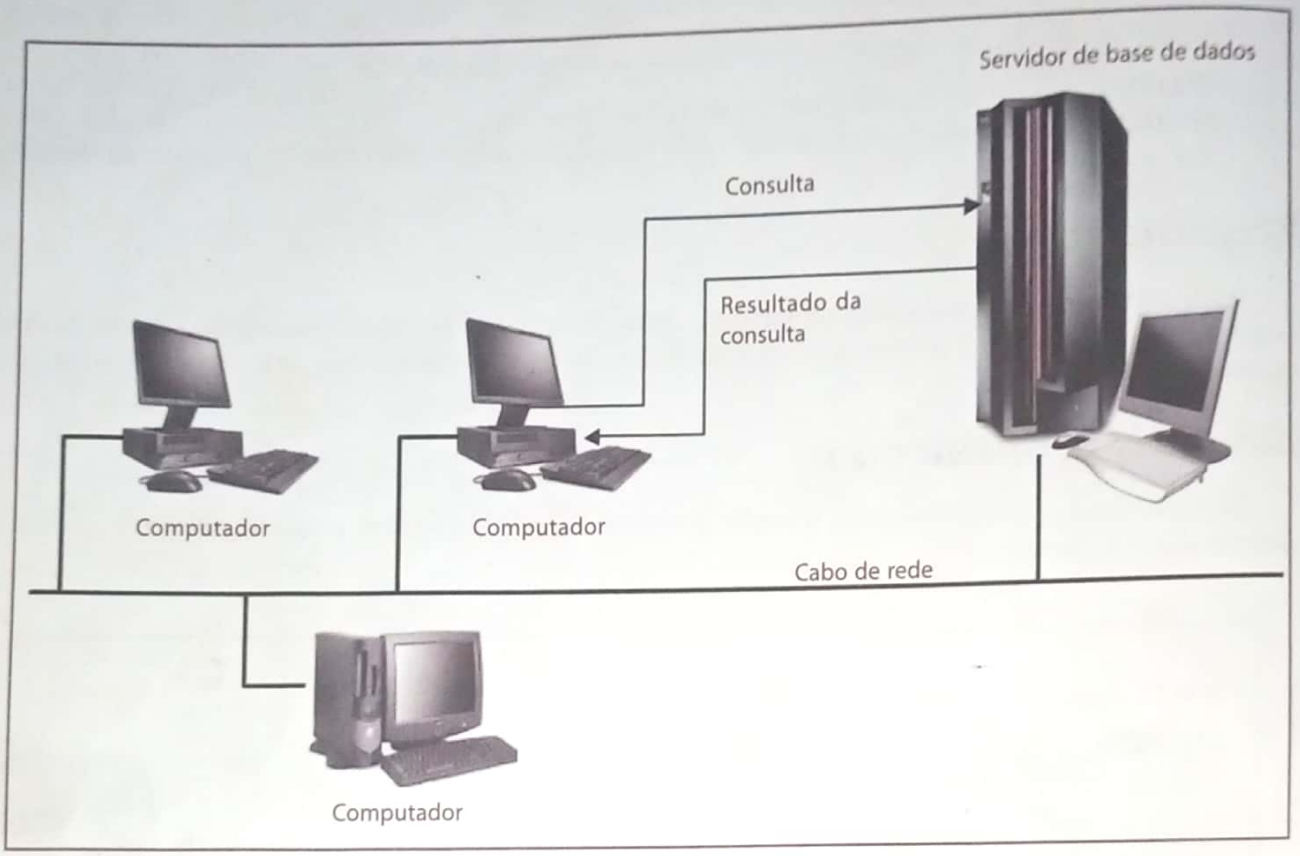


Fig. 1.8 Sistemas Cliente/Servidor

Sistemas paralelos

Consistem em **diversos processadores** e **diversos discos** conectados por redes de alta velocidade ou um computador multiprocessador em que estes (os processadores) são utilizados para o processamento paralelo de uma única transação. Sistemas paralelos têm custo elevado e uma gestão complexa.

Sistemas distribuídos

Nesta arquitectura, a **informação** está **distribuída por diversos servidores.** Cada servidor actua como no sistema cliente-servidor. Porém, as consultas oriundas dos aplicativos são feitas para qualquer servidor indistintamente. Exemplos típicos são as bases de dados em que o volume de informação é

muito grande e, por isso, deve ser distribuída por servidores. A característica básica é a existência de diversos programas aplicativos consultando a rede para aceder aos dados necessários, porém, sem o conhecimento explícito sobre quais são os servidores que dispõem desses dados.

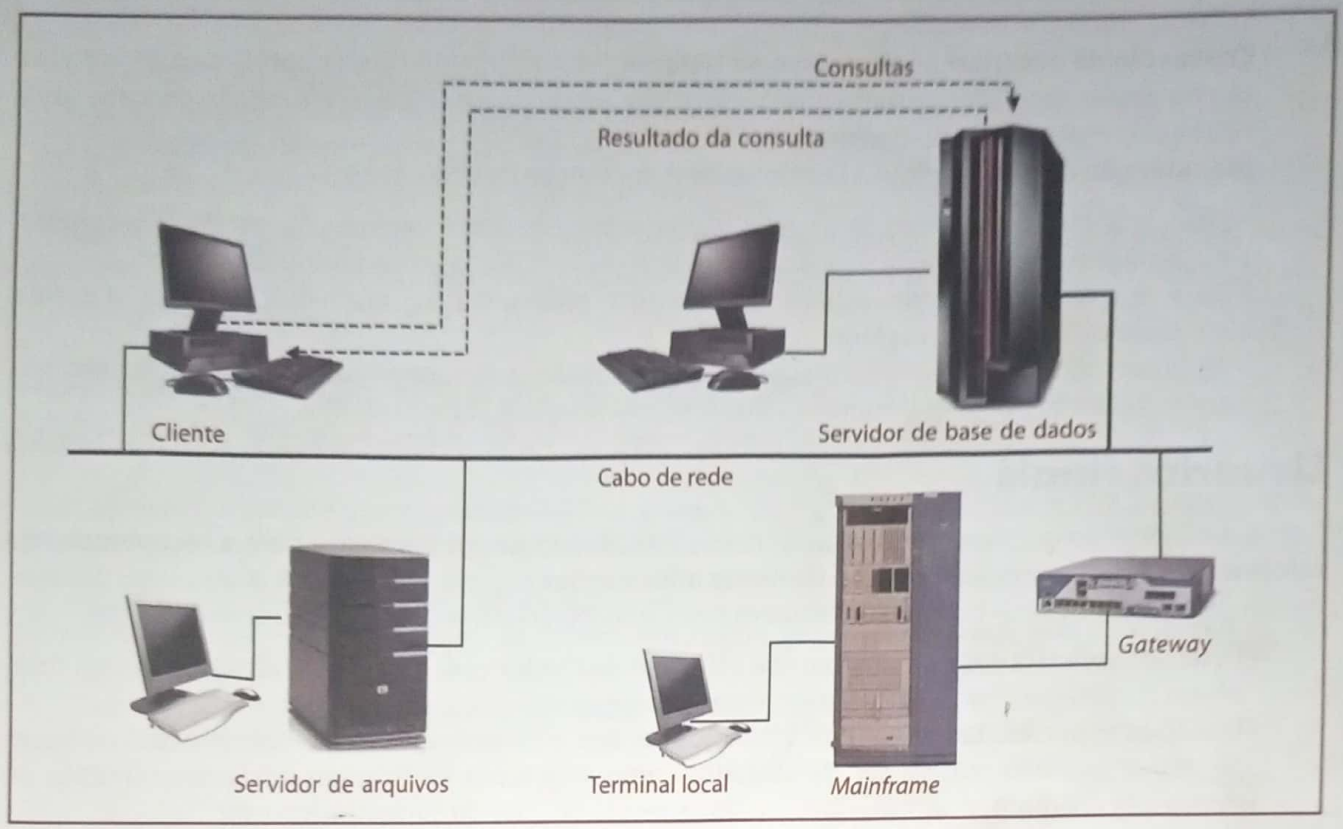


Fig. 1.9 Sistemas distribuídos

Usuários

Como um conjunto de informações pode ser utilizado por um conjunto diferenciado de *usuários*, é importante que estes usuários possam ter «visões» diferentes da base de dados. Uma «visão» é definida como um subconjunto de uma base de dados, formando, deste modo, um conjunto «virtual» de informações.

Para uma grande base de dados, existe, via de regra, um grande número de pessoas envolvidas, desde o projecto, passando pelo uso até à manutenção.

Administrador de Base de Dados (DBA)

Num ambiente de base de dados, o recurso primário é a própria base de dados e o recurso secundário o SGBD e os *softwares* relacionados. A administração destes recursos cabe ao **Administrador da base de dados.** Dentre as **funções** de um DBA destacam-se:

- **Definição do esquema**: o DBA cria o esquema da base de dados original escrevendo um conjunto de definições que são transformadas pelo compilador DDL num conjunto de tabelas armazenadas de modo permanente no dicionário de dados.
- **Definição da estrutura de dados e métodos de acesso**: o DBA cria estrutura de dados e métodos de acesso apropriados escrevendo um conjunto de definições, as quais são traduzidas pelo compilador de armazenamento de dados e pelo compilador de linguagem de definição de dados.

- **Esquema e modificações na organização física**: os programadores realizam relativamente poucas alterações no esquema da base de dados ou na descrição da organização física de armazenamento por meio de um conjunto de definições que serão usadas ou pelo compilador DDL ou pelo compilador de armazenamento de dados e definição de dados, gerando modificações na tabela interna apropriada ao sistema.
- **Concessão de autorização de acesso ao sistema**: o fornecimento de diferentes tipos de autorização no acesso aos dados permite que o administrador de dados regule o acesso dos diversos usuários às diferentes partes do sistema.
- **Manutenção de Rotina**: alguns exemplos de manutenção de rotinas são:
  - Realizar *backups* periódicos da base de dados, sejam em discos externos ou em servidores remotos, para previnir a perda de dados no caso de acidentes, como incêndio, inundação, etc.
  - Garantir que haja suficiente espaço livre em disco para operações normais e aumentar o espaço em disco conforme o necessário.
  - Monitorar tarefas a serem executadas na base de dados e assegurar que o desempenho não seja comprometido por tarefas muito onerosas submetidas por alguns usuários.

Usuários finais

A meta final de um sistema de base de dados é proporcionar um **ambiente para a recuperação de informações** e para o **armazenamento de novas informações** na base de dados. Há quatro tipos de usuários de sistemas de base de dados, diferenciados pelas suas expectativas de interacção com o sistema:

- **Usuários sofisticados**: são utentes que estão familiarizados com o SGBD e realizam consultas complexas e interagem com o sistema escrevendo programas.
- **Usuários especialistas**: são utentes sofisticados que escrevem aplicações especializadas de bases de dados que não podem ser classificadas como aplicações tradicionais em processamento de dados, por exemplo, sistemas especialistas, sistemas de base de conhecimento, etc.
- **Usuários navegantes**: são utilizadores que interagem com o sistema chamando um dos programas aplicativos já escritos. Acedem à base de dados casualmente.
- **Analistas de sistemas e programadores de aplicações**: os analistas determinam os requisitos dos utilizadores finais e desenvolvem especificações para transacções que atendam a estes requisitos, e os programadores implementam estas especificações como programas, testando, depurando, documentando e proporcionando a manutenção do mesmo. É importante que, tanto analistas quanto programadores, estejam a par dos recursos oferecidos pelo SGBD.

Linguagem de Base de Dados

Um sistema de base de dados proporciona dois tipos de linguagens: uma **específica** para os esquemas da base de dados e outra para **expressar** consultas e actualizações.

Linguagem de definição de dados

Para a definição dos esquemas lógico ou físico pode-se utilizar uma linguagem chamada **DDL** (*Data Definition Language* - Linguagem de Definição de Dados). O **SGBD** possui um compilador **DDL** que permite a execução das declarações para identificar as descrições dos esquemas e para armazená-las em tabelas que constituem um arquivo especial chamado **dicionário de dados** ou **directório de dados**.

Um dicionário de dados é um arquivo de **metadados**, isto é, dados a respeito de dados. Num sistema de base de dados, esse arquivo ou directório é consultado antes de os dados reais serem modificados.

Num **SGBD** em que a separação entre os níveis lógico e físico é bem clara, é utilizada uma outra linguagem, a **SDL** (*Storage Definition Language* – Linguagem de Definição de Armazenamento) para a especificação do nível físico. A especificação do esquema conceitual fica por conta da **DDL**.

Linguagem de manipulação de dados

Uma vez a base de dados organizada, usa-se uma linguagem para fazer a manipulação dos dados, a DML (*Data Manipulation Language* – Linguagem de Manipulação de Dados). Tal como foi referido antes, por manipulação de dados entende-se:

- A recuperação de informações armazenadas na base de dados
- A inserção de novas informações na base de dados
- A remoção de informações na base de dados
- A modificação de informações na base de dados

No nível físico, precisamos definir algoritmos que permitam o acesso eficiente aos dados. Nos níveis mais altos de abstracção, enfatizamos a facilidade de uso. O objectivo é proporcionar uma interacção eficiente entre homens e sistema.

Modelagem de Dados utilizando o Modelo Entidade Relacionamento (E/R)

O Modelo E/R propõe que a realidade seja visualizada sob três pontos de vista. Assim, há três conceitos fundamentais no Modelo E/R: **entidade, atributo** e **relacionamento** ou seja:

- Entidades – objectos que compõem a realidade.
- Atributos – tipos de informação ou características que se deseja conhecer sobre os objectos que compõem a realidade.
- Relacionamento – forma como estes objectos interagem entre si.

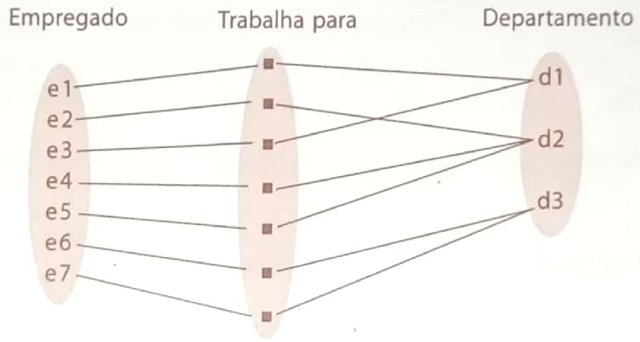


Fig. 1.10 Exemplo de um relacionamento

Entidades

Tal como foi referido antes, o objecto básico tratado pelo Modelo E/R é a «entidade», do latim, *entitas*, que significa ser, existência; é algo que possui existência distinta e separada, real ou imaginária.

Em inglês, o conceito de entidade recebe o nome que demonstra bem o seu significado, que é *entity type*, ou seja, um tipo de entidade.

Animal, pessoa, funcionário, residência, electrodoméstico, móvel, imóvel, material, aeronave e aluno são exemplos de substantivos concretos que representam objectos simples e do mesmo tipo, sendo, portanto, entidades.

Toda a **entidade** deve possuir um **identificador único** ou chave primária. Este identificador único é um dos critérios para identificar uma entidade. Sempre que não for possível achar este identificador ou chave primária, então não estará caracterizada uma entidade.

Atributos

Uma entidade, funcionário da Mozal, por exemplo, representa um tipo, no qual são classificados todos os funcionários da Mozal. No entanto, **cada indivíduo possui características próprias** que devem ser diferenciadas, como, por exemplo, o facto de que cada funcionário da Mozal possui um nome, um salário, uma categoria, uma data de nascimento, entre outras coisas. Essas características do mesmo tipo são utilizadas pela Mozal para contratar, administrar, pagar e despedir os funcionários.

Esses tipos de características (ou tipos de informação) são denominados **atributos** de uma entidade. Em inglês, o conceito de atributo recebe o nome de *attribute type*, ou seja, um tipo de atributo.

Comparando a Modelagem de Dados com a Análise Sentencial, pode-se dizer que, se cada entidade é uma palavra que representa um substantivo concreto ou abstrato, então o atributo é o seu «adjectivo», pois ele caracteriza a entidade.

Exemplo

Numa base de dados que se destina a servir os alunos da 12.ª classe na sua escola:

a) Os alunos, os professores e as disciplinas são entidades.
b) O aluno, o professor e a disciplina são tipos de entidade.

- identificados, respectivamente, por Aluno, Professor, Disciplina.

c) N.º de aluno (nAluno), nome de aluno (nomeA) e data de nascimento (datanasc) são atributos do tipo de entidade Aluno; código da disciplina (código), designação (designação) e notas (notas) são atributos do tipo de entidade Disciplina.
d) 45, Yuri Obri, 1994-03-01 são valores dos atributos.

Tipos de atributos

Os atributos de uma entidade podem desempenhar diversos papéis, de modo que eles podem ser classificados, como se segue:

Atributo simples ou atómico

Ocorre quando uma característica da entidade é representada por um **único atributo**. Por exemplo, na entidade Funcionário, temos os seguintes atributos simples: Nome, Sexo, Altura, etc.

Atributo composto

Ocorre quando uma característica da entidade é representada por um **conjunto de atributos** (dois ou mais atributos). Por exemplo, na entidade Funcionário, temos o seguinte atributo concatenado: Endereço (composto pelos atributos Rua, Número, Bairro, Cidade e Distrito).

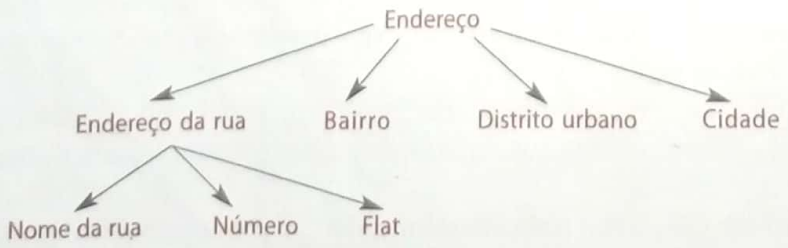


Fig. 1.11 Atributo composto

Chave primária (*primary key*)

Também conhecida como **Identificador único**. É o atributo de uma entidade cujo conteúdo individualiza uma única ocorrência desta entidade.

Regra: cada tipo de entidade deve possuir uma chave primária.

- No exemplo anterior, cada entidade do tipo Aluno pode ser identificada pelo nAluno e cada entidade do tipo Disciplina pode ser identificada pelo código.
- Alunos distintos terão números diferentes; duas disciplinas diferentes terão códigos diferentes.

Chave estrangeira (*foreign key*)

É um atributo pertencente a uma entidade, mas que é a chave primária de uma outra entidade. A chave estrangeira implementa o relacionamento entre as entidades.

Cardinalidade dos relacionamentos

O Modelo E/R, como toda a representação, não é a própria realidade, mas foi desenvolvido para estar o mais próximo possível dela. Por isso, além de representar as relações de posse, envolvimento, composição e geração (entre outras), incorporou, também, um outro conceito para melhorar o conhecimento sobre as políticas e regras. Este conceito é chamado **Cardinalidade do relacionamento**. A cardinalidade indica o número de relacionamentos dos quais uma entidade pode participar e pode ser: 1:1, 1:N, M:N.

Relacionamento um-para-um (1:1)

Indica que uma única ocorrência de uma entidade se pode relacionar com apenas uma única ocorrência de outra entidade.

Por exemplo: Funcionário (1) chefia (1) Departamento.

Lê-se: (um departamento possui um funcionário que exerce o papel de chefe; por sua vez, um funcionário-gerente pode gerir apenas um departamento de cada vez).

Relacionamento um-para-muitos (1:N ou 1:M)

Indica que uma ocorrência de uma entidade se pode relacionar com muitas ocorrências doutra entidade. No entanto, a recíproca não é verdadeira. Este tipo de relacionamento é muito comum (no mundo dos negócios).

Por exemplo: Funcionário (1) possui (N) Dependente

(Lê-se: um funcionário pode possuir vários dependentes; mas cada dependente pertence a apenas um funcionário).

Outro exemplo: Cliente (1) solicita (N) Cotação

(Lê-se: um cliente pode solicitar muitas cotações de vendas; no entanto, cada cotação somente pode ter sido solicitada por um cliente). Pode-se representar como 1:N ou 1:M.

Relacionamento muitos-para-muitos (N:M)

Indica que várias ocorrências de uma entidade se podem relacionar com muitas ocorrências doutra entidade. Pode-se representar como **N:M** ou como **M:N** ou, ainda, como **N:N** ou **M:M**.

Geralmente, um relacionamento desse tipo pode ser convertido e simplificado pela criação de uma entidade intermediária e de dois relacionamentos do tipo 1:N (um-para-muitos).

Por exemplo: Pedido (N) vende (M) Produto.

(Lê-se: em cada pedido podem ser vendidos muitos produtos diferentes – um para cada linha do pedido; por outro lado, um produto pode ser vendido por diversos pedidos).

Algumas dicas para a construção de diagramas E/R

- A presença de um **substantivo** usualmente indica uma **entidade.**
- A presença de um **verbo** é uma forte indicação de um **relacionamento.**
- Um **adjectivo**, que é uma qualidade, é uma forte indicação de um **atributo.**
- Um **advérbio** temporal, qualificando o verbo, é uma indicação de um atributo do **relacionamento.**

De recordar que segundo a Gramática:

- **Substantivo** é a palavra que nomeia os seres. O conceito de seres deve incluir os nomes de pessoas, de lugares, de instituições, de grupos, de indivíduos e de entes de natureza espiritual mitológica.
- **Verbo** é a palavra que se flexiona em número (singular/plural), pessoa (primeira, segunda, terceira), modo (indicativo, subjuntivo, imperativo), tempo (presente, pretérito, futuro) e voz (activa, passiva, reflexiva). Pode indicar acção (fazer, copiar), estado (ser, ficar), fenómeno natural (chover, anoitecer), ocorrência (acontecer, suceder), desejo (aspirar, almejar) e outros processos.
- **Adjectivo** é a palavra que caracteriza o substantivo, atribuindo-lhe qualidades (ou defeitos) e modos de ser, ou indicando-lhe o aspecto ou o estado.
- **Advérbio** é a palavra que caracteriza o processo verbal, exprimindo circunstâncias em que esse processo se desenvolve.

Entidade

Uma entidade corresponde à representação de todo e qualquer substantivo, concreto ou abstracto, sobre o qual se precisa de armazenar e/ou recuperar informações.

Documento

Breve historial dos Sistemas de Bases de Dados

O processamento de dados tem impulsionado o crescimento dos computadores desde os primeiros dias dos computadores comerciais. Na verdade, a automatização das tarefas de processamento de dados já existia antes mesmo dos computadores. Cartões perfurados, inventados por Herman Hollerith, foram usados no início do século XX para registar dados do censo dos Estados Unidos, e sistemas mecânicos foram usados para processar os cartões perfurados e tabelar os resultados. Mais tarde, os cartões perfurados passaram a ser amplamente usados como um meio de inserir dados em computadores.

As técnicas de armazenamento e processamento de dados evoluíram ao longo dos anos.

- **Década de 50 e início da década de 60:**
  Os computadores se tornam parte efectiva do custo das empresas juntamente com o crescimento da capacidade de armazenamento. Foram desenvolvidos dois modelos de dados principais: modelo em rede (CODASYL - *Comitee for Data Systems Language*) e o modelo hierárquico (IMS - *Information Management System*).

Modelo de dados em rede:
- Os primeiros trabalhos foram realizados em 1964 por Charles Bachman.
- Dados são representados por uma colecção de registos e os relacionamentos por meio de *links*.
- É representado por um diagrama constituído por caixas e linhas.
- São usados apenas relacionamentos muitos-para-muitos.

Modelo de dados hierárquico:
- Também se utilizava de registos para representar os dados e *links* para os relacionamentos.
- São organizados na forma de uma árvore com raiz.
- Como exemplo: *Clipper, Dbase 2, Fox Pro, COBOL.*

Portanto,
o Processamento de Dados usando fitas magnéticas para armazenamento.
- Fitas fornecem apenas acesso sequencial.
  - Cartões perfurados para entrada.
  - Os programas realizavam seu trabalho de forma específica.
  - Os utentes precisavam conhecer a estrutura física da Base de Dados para poderem realizar uma consulta.

- **Final das décadas de 60 e 70:**
  - Surgiram os discos rígidos e suas facilidades.
  - Os dados não necessitam mais de processamento sequencial.
  - Discos rígidos permitem acesso directo aos dados.
  - Modelos de dados de rede e hierárquico em largo uso.
  - Ted Codd define o modelo de dados relacional, que se tornou um marco em como pensar em Bases de dados.
  - Processamento de transação de alto desempenho (para a época).
  - Peter Chen propõe o modelo Entidade-Relacionamento (E/R) para projectos de bases de dados dando uma nova e importante percepção dos conceitos de modelos de dados. Assim como as linguagens de alto nível, a modelagem E/R possibilita ao projectista concentrar-se apenas na utilização dos dados, sem se preocupar com a estrutura lógica de tabelas.

O termo Sistema de Gerenciamento de Bases de Dados Relacional (SGBDR – RDBMS em inglês) foi definido durante este período.

- **Década de 80:**
  No início dos anos 80, a comercialização de sistemas relacionais começa a tornar-se uma febre entre as organizações.
  - Protótipos relacionais de pesquisa evoluem para sistemas comerciais
  - SQL se torna o padrão do sector
  - Sistemas de bases de dados paralelos e distribuídos
  - Sistemas de bases de dados orientados a objectos

- **Década de 1990:**
  No início dos anos 90, há indício de uma leve crise económica nas indústrias e algumas empresas sobrevivem oferecendo alguns produtos a custos muito elevados.
  Muito desenvolvimento acontece em ferramentas de desenvolvimento para o *desktop* para as aplicações (*client tolls*), tais como: *PowerBuilder (Sybase)*, *Oracle Developer, Visual Basic (Microsoft)*, entre outros. O modelo cliente-servidor (*client-server*) passa a ser uma regra para futuras decisões de mercado e assiste-se ao desenvolvimento de ferramentas de produtividade como *Excel/Access (Microsoft)* e ODBC, também é marcado como o início dos protótipos de *Object Database Management Systems* (ODBMS).
  - Grandes aplicações de suporte a decisão e exploração de dados
  - Grandes data *warehouses* de vários *terabytes*
  - Explosão da *Web*, maior utilização de sistemas de bases de dados
  - Base de dados voltados para consultas, utilização da Linguagem SQL

- **Década de 2000:**
  - Padrões *XML* e *Xquery*, um novo conceito em Bases de Dados
  - Administração de banco de dados
  - Computação autónoma, redução de esforços da administração de sistemas

- **Actualmente**
  Assim como a informática evolui na parte de *hardware*, com o *software* não é diferente isso é bem mais rápido. É perceptível como a necessidade das pessoas determina a operabilidade dos sistemas de bases de dados, pois cada um possui sua funcionalidade e peculiaridade de forma que torna cada vez mais amplo o campo de actuação do profissional.

Exercícios de consolidação

1. O que entende por base de dados?
2. Quais são os objectos principais de uma base de dados?
3. Quais são as funções de um SGBD?
4. O que significa manipular uma base de dados?
5. Caracterize um sistema de base de dados.
6. Em que áreas de actividade são usadas hoje intensamente as aplicações de bases de dados?
7. Dê exemplos de três SGBD que aprendeu.
8. Indique três vantagens de um SGBD.
9. Em que condições se torna dispensável o recurso a um SGBD?
10. Quais são os níveis de abstracção de um SGBD?
11. Caracterize o chamado nível conceptual.
12. Quais são os modelos de dados que conhece?
13. Caracterize o modelo relacional.
14. Classifique os SGBD.
15. Quando é que se diz que um SGBD é distribuído?
16. Quanto à arquitectura dos sistemas de base de dados, caracterize os chamados sistemas Cliente-Servidor.
17. Dentre os vários usuários duma base de dados, destaca-se o DBA. Quais são as principais tarefas deste?
18. Caracterize os usuários finais de uma base de dados.
19. Qual é a função dos analistas de sistemas e programadores de aplicações?
20. O que entende por um dicionário de dados num SBD?
21. O que é uma DML-*Data Manipulation Language*?
22. O Modelo E/R propõe que a realidade seja visualizada sob três pontos de vista: entidade, atributo e relacionamento. O que entende por entidade? Dê exemplos de três entidades à sua escolha.
23. Comparando a Modelagem de Dados com a Análise Sentencial, pode-se dizer que, se cada entidade é uma palavra que representa um substantivo concreto ou abstrato, então o atributo é _____, pois ele caracteriza a entidade.
24. Distinga chave primária (*primary key*) duma chave estrangeira (*foreign key*).
25. A cardinalidade indica o número de relacionamentos nos quais uma entidade pode participar. Caracterize, dando um exemplo, o relacionamento 1:N.
26. Assinale com um círculo as respostas correctas.

    a) O *software* encarregado da gestão e armazenamento da informação num sistema de bases de dados designa-se por:

    A. OLTP **B.** SGBD **C.** DSS

    b) A existência de Informação divergente sobre o mesmo facto, numa base de dados, designa-se por:

    A. Redundância **B.** Inconsistência **C.** Independência

    c) O processo automático de actualização de informação redundante, numa base de dados designa-se por:

    A. Propagação de actualizações
    B. Manutenção da integridade referencial
    C. *Cascade update related records*

    d) A chave escolhida como atributo identificador de uma tabela para o relacionamento com as restantes tabelas da base de dados, designa-se por:

    A. Chave candidata **B.** Chave externa **C.** Chave primária

Criar uma Base de Dados com o *MS Access*

Até agora aprendeu os conceitos gerais sobre bases de dados sem entrar em grandes pormenores sobre os aplicativos específicos (SGBD) que os desenham, apesar de já a isso termos feito referência. A título de exemplo, vamos aprender a criar bases de dados com o *MS Access*.

O ***Microsoft Access*** (nome completo ***Microsoft Office Access***), também conhecido por ***MSAccess***, é um sistema de gestão de banco de dados da *Microsoft*, incluído no pacote do *Microsoft Office Professional*, que combina o *Microsoft Jet Database Engine* com uma *interface* gráfica do utilizador (*graphical user interface*). Ele permite o desenvolvimento rápido de aplicações que envolvem tanto a modelagem e estrutura de dados como também a *interface* a ser utilizada pelos usuários. O *Access* cria bases de dados **relacionais**, o que significa que os dados são armazenados por assunto ou tarefa em várias tabelas separadas, mas estão relacionados e podem ser reunidos nas formas especificadas pelo utilizador.

Na terminologia do modelo relacional, cada tabela é chamada **relação**; uma linha de uma tabela é chamada **tupla**; o nome de cada coluna é chamado **atributo**; o tipo de dado que descreve cada coluna é chamado **domínio**.

As bases de dados do *Access* consistem em **objectos** dos quais os quatro mais importantes são:

- **Tabelas:** armazenam os dados em linhas e colunas. Todas as bases de dados contêm uma ou mais tabelas.
- **Consultas:** obtêm e processam os dados. Podem combinar os dados de diferentes tabelas, actualizar os dados e executar cálculos com base nos mesmos.
- **Formulários:** controlam a introdução de dados e as vistas de dados. Fornecem sugestões visuais que facilitam o trabalho com os dados.
- **Relatórios:** resumem e imprimem os dados. Convertem os dados nas tabelas e consultas em documentos para comunicar ideias.

Existem duas formas principais de se iniciar a criação de uma base de dados no *MS Access:* utilizando um modelo pré-pronto, ou partindo do zero.

Usar um modelo (*template*) pré-pronto

Recorrer a um modelo oferecido pelo próprio *Access* é a forma mais rápida de iniciar a criação de uma base de dados. Isso porque esses padrões (*template*) já contêm todas as tabelas, consultas, formulários e relatórios necessários para executar tarefas específicas.

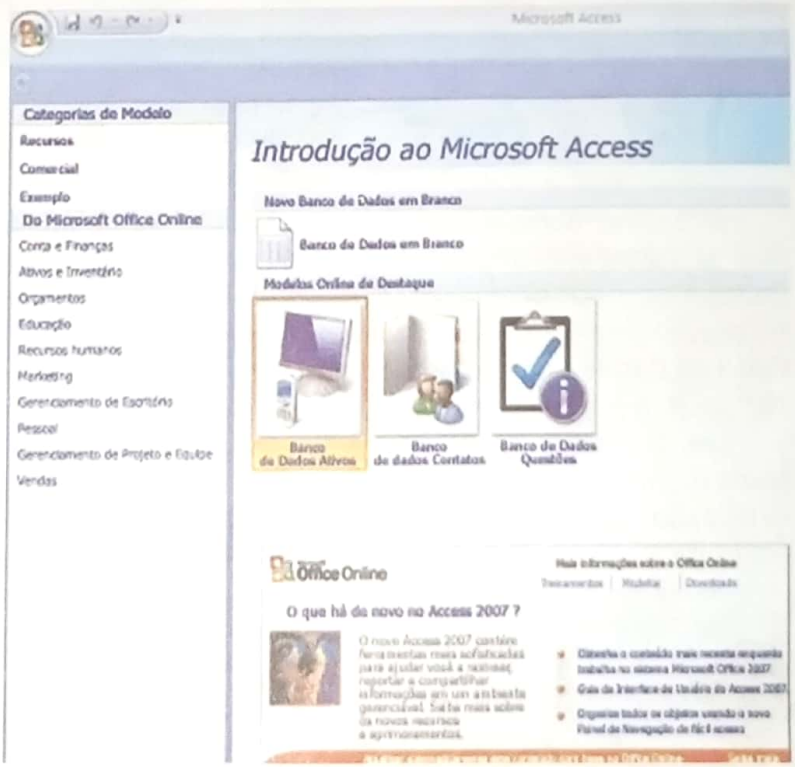


Fig. 1.12 Criação de uma base de dados

Quando se executa o *MS Access*, vários modelos são oferecidos na página Introdução ao *Microsoft Office Access*, e outros vão ficando disponíveis à medida que se clique nos *links* no painel Categorias de Modelo.

Podemos clicar num padrão à nossa escolha. O *Access* sugere um nome de ficheiro para a base de dados a criar, na caixa Nome do ficheiro. Entretanto, nós podemos mudar esse nome do ficheiro se preferirmos. Também é possível mudar a pasta sugerida pelo *Access* para guardar a base de dados.

Depois de termos escolhido o padrão da nossa base de dados, atribuído um nome e estarmos posicionados na pasta correcta, clicamos em **Criar**. O *Access* vai criar e abrir a base de dados. Será exibido um formulário no qual pode-se começar a **Inserir** dados. Se o modelo contém dados de amostra, é possível excluir cada registo clicando na barra sombreada à esquerda dele, indo para a guia **Início**, no grupo **Registos** e clicando em **Excluir**.

Daqui para frente é só trabalhar com a base criada. Para começar a inserir dados, clicamos na primeira célula vazia do formulário. Pode-se usar o **Painel de Navegação** para procurar outros formulários ou relatórios que se queira usar.

Pode acontecer, por exemplo, que não se queira usar nenhum dos modelos de base de dados oferecidos pelo Access. Nesse caso, há a opção de **baixar** (fazer o *download*) outros padrões no site da *Microsoft*, a partir do próprio *software*.

Criar uma Base de Dados sem usar um modelo pré-pronto

Alternativamente, pode-se preferir não adoptar nenhum dos modelos oferecidos pelo programa ou pelo site da *Microsoft*. A saída, então, é iniciar uma base de dados a partir do zero, isto é, usando as suas próprias tabelas, formulários e relatórios. Para tal, na tela inicial em **Novo Banco de Dados** em Branco do *MS Access* clique em **Banco de Dados em Branco**.

Fig. 1.13 Ícon inicial – base de dados vazio

Atribui um nome à Base de Dados na caixa **Nome do Ficheiro** e clica-se em **Criar.** O *Access* cria a Base de Dados com uma tabela vazia chamada **Tabela1** e, depois, abre essa tabela no modo **Folha de Dados**. Esse modo permite inserir dados imediatamente e também deixa que o *Access* crie a estrutura da tabela. Daí para a frente, é só começar a digitar e trabalhar com os dados. Inserir informações no modo **Folha de Dados** é muito parecido com o modo de trabalhar numa planilha do *Excel* que aprendemos na 10.ª classe. A estrutura da tabela é criada à medida que os dados são inseridos.

Um caso típico em que se pode criar a Base de Dados a partir do zero é quando se tem dados numa outra aplicação que se deseja importar para o *Access*. Isso porque o modelo possui uma estrutura de dados já definida, que pode exigir trabalho adicional para adaptar os dados pré-existentes à sua estrutura.

Importar dados

Na Base de Dados já criada, aceda ao grupo **Importar** em **Dados Externos**, e clique no comando para o tipo de ficheiro que se pretende importar.

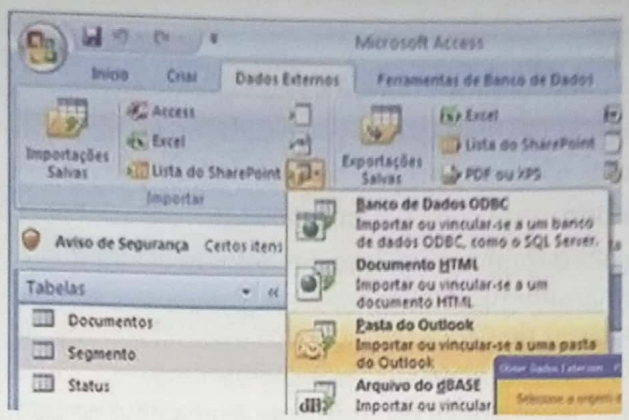


Fig. 1.14 Caixa de importação de dados

Por exemplo, se se estiver a importar dados do *Word*, clique no ícone do *Word*. Se não estiver visível o tipo de programa desejado, clique em **Mais**.

Fig. 1.15 Opções de importação

Na caixa de diálogo **Obter Dados Externos**, clique em **Procurar** para localizar o ficheiro de dados que será a fonte. Depois é preciso informar a maneira que essas informações importadas vão entrar na nova base de dados, clicando na opção desejada em **Especificar** como e onde se deseja armazenar os dados na base de dados actual. Agora, se se importar objectos ou vincular tabelas de uma base de dados do *Access*, a caixa de diálogo **Importar Objectos** ou **Vincular Tabelas** aparece. É só escolher os items desejados e clicar em **OK**.

Exercícios de consolidação

1. Crie uma Base de Dados no Access com as tabelas *alunos*, *disciplinas* e *notas*.
2. Defina os tipos e os tamanhos dos campos.
3. Formate o campo *nota* de forma a ter 2 casas decimais.
4. Insira dados para cada tabela.
5. Apague e modifique registos.
6. Defina a chave de cada tabela.
7. Impeça a inserção de notas maiores do que 20 e menores do que zero.
8. Impeça o lançamento de notas com mais de 120 dias de atraso.
9. Qual dos elementos seguintes não é um objecto de Base de Dados?
   A. Tabela **B.** Relatório **C.** Consulta **D.** Folha de cálculo
10. Qual das seguintes afirmações descreve uma Base de Dados relacional?
    A. Fornece uma relação entre números inteiros.
    B. Consiste em tabelas separadas de dados relacionados.
    C. Obtém dados relacionados com as respectivas consultas.
11. Qual das seguintes afirmações é verdadeira relativamente a uma Base de Dados relacional no *MS Access*?
    A. Os dados são armazenados em tabelas separadas e relacionadas.
    B. Os dados são duplicados, por isso é mais fácil encontrá-los.
    C. Os dados são armazenados em formulários.
12. Se uma tabela não estiver bem organizada, qual das seguintes situações poderá ocorrer quando actualizar a Base de Dados?
    A. Os dados poderão ser movidos para uma tabela separada.
    B. Os dados poderão ser duplicados ou eliminados acidentalmente.
    C. As pessoas poderão introduzir dados inválidos.
13. Qual das seguintes afirmações é uma forma de criar uma tabela?
    A. Criando um relatório. **B.** Criando uma consulta. **C.** Utilizando um assistente.
14. Quais são as vantagens de escolher o tipo de dados correcto para cada campo?
    A. Impedir introduções de dados inválidos e melhorar o desempenho da Base de Dados.
    B. Definir relações entre tabelas e impor estruturas de dados exclusivas.
    C. Definir um valor predefinido e aplicar formatações.
15. Em que local altera as propriedades de um campo?
    A. Janela Relações **B.** Vista Folha de dados **C.** Vista Estrutura
16. O que é uma chave externa?
    A. Um ou mais campos de tabela que estão relacionados com o campo ou campos de chave primária noutra tabela.
    B. Um campo é apresentado apenas numa relação um-para-muitos.
    C. Uma chave para uma casa de férias.
17. Numa relação muitos-para-muitos.
    A. Muitas tabelas estão associadas a muitas outras tabelas.
    B. Vários índices estão associados a várias tabelas.
    C. Uma terceira tabela contém as chaves primárias de duas tabelas relacionadas.

Objecto relatório

No *Microsoft Office Access* pode-se criar uma variedade de relatórios diferentes, do mais simples ao mais complexo. Primeiro, é preciso considerar a fonte de registo de relatório. Se o relatório for uma lista simples de registo ou um resumo agrupado sobre qualquer coisa, então aí deve-se, primeiro, determinar os campos que contêm os dados que se deseja ver no relatório e em que tabelas ou consultas estão localizados.

Depois de escolher a fonte de registo, poder-se-á ponderar a utilização ou não do Assistente de Relatório para criar o relatório desejado. O **Assistente de Relatório** é um recurso do *Access* que apresenta várias questões e gera um relatório baseado nas suas respostas.

Escolher uma fonte de registo

Um relatório consiste em informações enviadas de tabelas ou consultas, bem como em informações armazenadas com o *design* do relatório, como os rótulos, os títulos e as imagens. As tabelas ou as consultas que fornecem dados subjacentes também são conhecidas como a fonte de registos de relatório. Se os campos que se deseja incluir existirem numa única tabela, então usa-se essa tabela como a fonte do registo. Se os campos estiverem contidos em mais de uma tabela, aí será necessário usar-se uma ou mais consultas como a **fonte do registo**.

Criar um relatório usando a ferramenta Relatório

A ferramenta **Relatório** fornece o modo mais rápido de criar um relatório, porque ela gera um relatório imediatamente sem solicitar nenhuma outra informação. O **Relatório** exibe todos os campos da tabela ou da consulta subjacente. A ferramenta **Relatório** pode não criar o produto final que se deseja, mas é bastante útil como um meio de analisar rapidamente os dados subjacentes. Pode-se salvar o relatório e modificá-lo no modo de exibição *Layout* ou no modo *Design* para que ele atenda melhor às exigências.

Para isso, vá ao **Painel de navegação**, clique na **tabela** ou na consulta na qual se deseja basear o relatório.

Na guia **Criar**, no grupo Outros, clique em **Relatório**  . O *Access* cria e exibe o **Relatório** no modo de exibição *Layout*.

Criar um relatório utilizando o Assistente de Relatório

É possível usar o **Assistente de Relatório** para ser mais selectivo sobre os campos que aparecerão no relatório. Também se pode especificar como os dados são agrupados e classificados e usar-se os campos de mais de uma tabela ou consulta, desde que se tenha especificado anteriormente um relacionamento entre as tabelas e as consultas.

Na guia **Criar,** no grupo **Relatórios**, clique em **Assistente de Relatório**. Depois é só seguir-se as orientações e, por fim, clicar em **Concluir**.

Quando se visualizar o relatório, ele será exibido do mesmo modo em que será impresso.

Criar um relatório utilizando a ferramenta Relatório em branco

Se não se estiver interessado em utilizar a ferramenta **Relatório** ou o **Assistente de Relatório**, pode usar-se a ferramenta **Relatório** em branco para se criar um relatório a partir do nada. Esse pode ser um meio rápido de se criar um relatório, especialmente se se pretende colocar apenas alguns campos no relatório. O procedimento a seguir explica como usar a ferramenta **Relatório** em branco.

Na guia **Criar**, no grupo **Relatórios**, clique em **Relatório em Branco** . Um relatório em branco é exibido no modo de exibição *Layout*, e o painel **Lista** de campos é exibido do lado direito da janela do *Microsoft Access*.

No painel **Lista de Campos**, clique no sinal de adição próximo da tabela, ou tabelas, que contém os campos que se deseja ver no relatório.

De seguida é só arrastar-se cada campo para o relatório, um de cada vez, ou manter-se pressionada a tecla *CTRL* e selecionar-se vários campos e, em seguida, arrastá-los para o relatório ao mesmo tempo.

Pode-se usar as ferramentas no grupo **Controlos** na guia **Formatação** para adicionar um logótipo, título, número de página ou data e hora ao relatório.

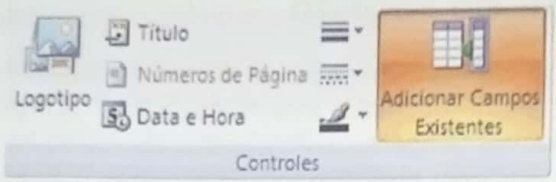


Fig. 1.16 Adicionar Controlos

Ajustar o relatório no modo *Layout*

Depois de criar um relatório, pode-se ajustar o seu *design*, trabalhando no modo *Layout*. Utilizando-se os dados reais do relatório como guia, pode-se ajustar a largura da coluna, reorganizar-se as colunas e adicionar-se níveis de agrupamento e totais. Pode-se inserir novos campos no *design* do relatório e definir-se as propriedades para os relatórios e seus controlos.

Para se abrir um relatório no modo de exibição *Layout*, clique com o botão direito do cursor no nome do relatório no **Painel de navegação** e, em seguida, em modo *Layout* .

O *Access* mostra o relatório no modo *Layout*.

Pode-se usar a folha de propriedades para modificar as propriedades do relatório e os seus controlos e secções. Para exibir a folha de propriedades, pressione *F4*.

Usa-se o painel **Lista de Campos** para adicionar campos da tabela ou consulta adjacente ao *design* do relatório. Para exibir o painel **Lista de Campos**, realize uma das acções a seguir:

- Na guia Formatar, no grupo **Controlos**, clique em **Adicionar Campos Existentes** .
- Pressione ALT+F8.

É possível adicionar-se campos arrastando-os do painel **Lista de Campos** para o relatório.

Ajustar o seu relatório no modo *Design*

Também se pode ajustar o *design* do relatório, trabalhando no modo *Design*. Pode-se adicionar novos controlos e campos ao relatório, adicionando-os à grade de *design*. A folha de propriedades fornece acesso a um grande número de propriedades que se pode definir para personalizar o relatório.

Para alternar para o modo de exibição *Design*, clique com o botão direito do cursor no nome do relatório no Painel de navegação e, em seguida, clique em Modo *Design* .

O *Access* mostra o relatório no modo *Design*.

Pode-se usar a folha de propriedades para se modificar as propriedades do próprio relatório e dos seus controlos e secções. Para se exibir a folha de propriedades pressiona-se *F4*.

Pode-se usar o painel **Lista de Campos** para adicionar campos da tabela ou consulta adjacente ao *design* do relatório. Para exibir o painel **Lista de Campos,** realize uma das acções a seguir:

Na guia Formatar, no grupo **Controlos**, clique em **Adicionar Campos Existentes** .

Pressione *ALT+F8*.

Adicionar controlos ao relatório

Alguns controlos são criados automaticamente, como o controlo caixa de texto acoplado que é criado quando se adiciona um campo do painel **Lista de Campos** no seu relatório. Muitos outros controlos podem ser criados no modo de exibição *Design* usando-se as ferramentas no grupo **Controlos** da guia *Design*.

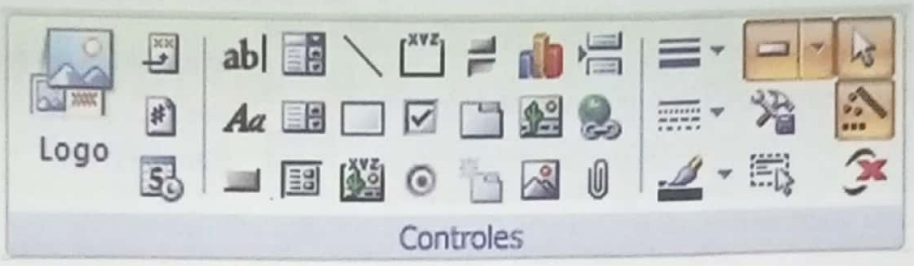


Fig. 1.17 caixa de controlo

Salvar o trabalho

Depois de salvar o *design* do relatório, pode-se executar o relatório quando for necessário. O *design* do relatório permanece o mesmo, mas obtêm-se dados actuais sempre que se imprime o relatório.

Salvar o *design* de relatório

1. Clique no botão *Microsoft Office* e, em seguida, clique em **Salvar** ou pressione *CTRL+S*.

   Alternativamente, clique em **Salvar** na Barra de Ferramentas de Acesso Rápido.

2. Se o relatório não tiver nenhum título, digite um nome na caixa Nome do Relatório e, em seguida, clique em **OK.**

Salvar o *design* de relatório com um novo nome

1. Clique no botão *Microsoft Office* e, em seguida, clique em **Salvar como**.
2. Na caixa de diálogo Salvar como, digite um nome da caixa **Salvar Relatório como**, seleccione **Relatório** na caixa **Como** e clique em **OK.**

Objecto formulário

Ao criarmos uma tabela, definimos a estrutura da Base de Dados. Posteriormente, podemos criar a *interface* que servirá para preenchermos e consultarmos os dados dessa tabela, o formulário.

Portanto, um formulário é utilizado, em primeiro lugar, para introduzir, eliminar ou apresentar dados numa Base de Dados. Pode também ser utilizado como uma caixa de diálogo personalizada que aceita dados inseridos pelo utilizador e executa uma acção com base nos mesmos, ou ainda como um painel de navegação que abre outros formulários e relatórios numa Base de Dados.

Para a criação do formulário acedemos em **Criar**. Nesse menu estão disponíveis vários botões numa secção dedicada a **Formulários**:

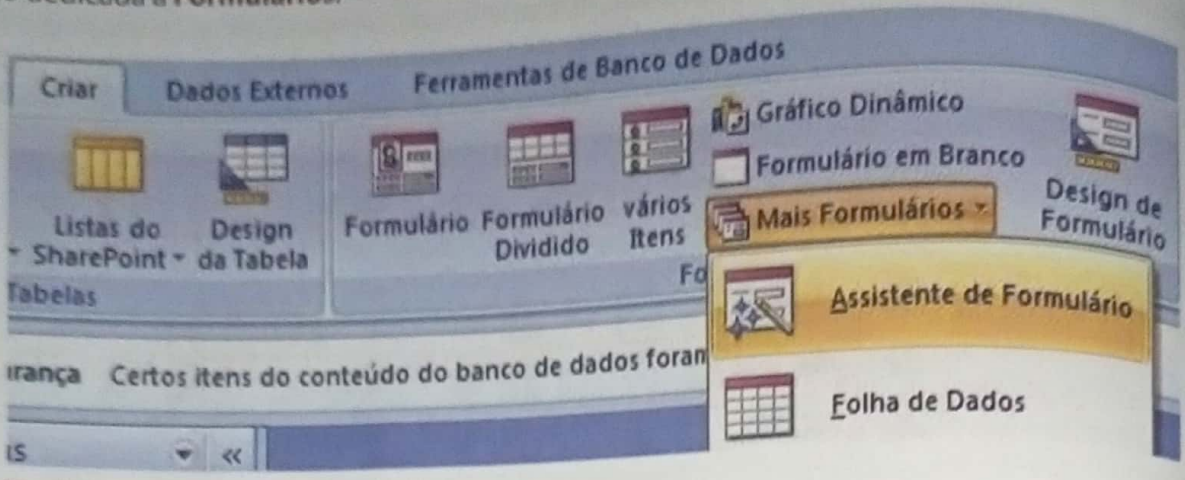


Fig. 1.18 Caixa de criação de formulários

Aí escolhemos a opção que melhor se ajusta ao nosso trabalho. À semelhança da criação do relatório, aqui também podemos optar por usar o **Assistente de Formulário**; para o efeito clique em **Mais Formulários** e de seguida escolha **Assistente de Formulário**.

A janela que surgir perguntará que campos da tabela se pretende incluir no formulário. Clica no botão com os caracteres >> para escolher e, em seguida, clique em **Avançar**. Na janela seguinte, escolhe o *layout* de preferência. Feita a escolha, clique em **Avançar**. O *Access* mostrará agora uma janela onde poderá escolher um estilo (visual) para o formulário. Escolha a sua opção preferida. Clique em **Avançar** e, depois, em **Concluir**.

Se quiser fazer alguma alteração clique no botão **Modo de Exibição** e escolha a opção Modo *Design*.

Fig. 1.19 Modo Design

Neste modo, poderá fazer uma série de alterações no formulário.

Consultas a uma Base de Dados – a linguagem SQL

As operações de consulta (e actualização) de uma Base de Dados denominam-se *queries*.

As regras do modelo relacional levaram ao desenvolvimento de uma linguagem, o SQL (*Structured Query Language* – Linguagem de Consulta Estruturada), que permite efectuar eficientemente *queries*.

A história do SQL começa em 1970 com a publicação, por E. F. Codd, no *ACM Journal*, de um artigo intitulado *A Relational Model of Data for Large Shared Data Banks*.

O modelo proposto por Codd é hoje considerado a base de trabalho para qualquer Sistema de Gestão de Base de Dados Relacional (SGBDR).

A linguagem SQL pertence à 4.ª Geração das Linguagens de Programação, da qual é talvez, a única sobrevivente. Não é, no entanto, uma evolução das linguagens de 3.ª Geração (Pascal, BASIC, C, COBOL, FORTRAN – sobre isso discutiremos no próximo capítulo), já que estas têm características bem diferentes.

A linguagem SQL destina-se, por isso e pela sua simplicidade, não só a informáticos, como também a gestores, utilizadores, administradores de bases de dados, etc.

No entanto, a sua principal diferença em relação às linguagens de 3.ª Geração é a ausência nestas de um objectivo pré-definido, coisa que no SQL está bem determinada: **proporcionar a interface entre o SGBDR e o utilizador, através da manipulação de dados**.

Com a linguagem SQL é possível:

- Criar, alterar e remover todas as componentes de uma base de dados, como tabelas, índices, *views*, etc.
- Inserir, alterar e apagar dados.
- Interrogar a base de dados.
- Controlar o acesso dos utilizadores à base de dados, e às operações a que cada um deles tem acesso.
- Obter a garantia da consistência e integridade dos dados.

A linguagem SQL é composta por vários conjuntos de comandos:

- **DDL** (*Data Definition Language*): comandos para definir ou modificar a composição das tabelas, apagar tabelas, criar índices, definir *views* e especificar direitos de acesso a tabelas e *views*.

Os principais comandos DDL são: Create (Criar), Drop (remover) e Alter (alterar).

- **DML**
  - (*Interactive Data Manipulation Language*): inclui uma linguagem de consulta baseada em álgebra relacional e em cálculo relacional sobre registos; inclui também comandos para inserir, apagar e modificar registos na base de dados.
  - (*Embedded Data Manipulation Language*): projectada para ser usada a partir de linguagens de programação de uso geral, da 3.ª geração.

Os principais comandos DML são: *Select* (Selecção de Dados), *Insert* (Inserção de Dados), *Update* (Actualização de Dados) e *Delete* (Exclusão de Dados).

Comandos SQL

SELECT

Significa **seleccionar**, ou seja escolher. O selecionador nacional de futebol escolhe os jogadores que hão-de fazer parte dos Mambas; e escolhe-os, claro, dentre as dezenas que jogam nas equipas do país. O comando *SELECT* do *SQL* significa exactamente o mesmo e tem o mesmo papel que o seleccionador dos Mambas.

Portanto, a interrogação de qualquer base de dados relacional faz-se utilizando o comando SELECT. A sintaxe do comado é a seguinte:

```
SELECT Campo1 , Campo2 , ... , CampoN , *
FROM  Tabela1 , ... , TabelaK
[WHERE condição]
[GROUP BY ... ]
[HAVING ... ]
[ORDER BY ... ]
```

As cláusulas entre [ ] são opcionais; no entanto, sempre que aparecerem terá de ser pela ordem indicada.

Se a cláusula *WHERE* for omitida, a **condição** será considerada **verdadeira**.

Se indicarmos * no *SELECT*, em vez de campos serão seleccionados todos os campos das tabelas envolvidas.

O resultado de uma consulta de *SQL* com o comando *SELECT* é **sempre** uma tabela.

Podemos repetir campos no *SELECT*; as repetições terão nomes atribuídos pelo SGBDR.

A ordem dos campos de saída é a que consta no *SELECT* e pode ser diferente daquela que é usada na criação da tabela.

A forma mais simples da instrução SELECT

```
SELECT { * | table.* | [table.]field1 [, [table.]field2 [, ...]]}
FROM table;
```

onde:

* especifica que todos os campos devem ser seleccionados.

table especifica o nome da tabela que contém os campos e os registos seleccionados.

field especifica os nomes dos campos que são seleccionados.

Cláusula WHERE

A cláusula *WHERE* admite os seguintes operadores:

Relacionais	
=	igual a
>	maior que
>=	maior ou igual a
<	menor que
<=	menor ou igual a
<> (mais vulgar) ou !=	diferente

Lógicos	
cond1 AND cond2	conjunção
cond1 OR cond2	disjunção
NOT condição	negação

Especiais	
campoN [NOT]BETWEEN valor1 AND valor2	Verifica intervalos de valores.
campoN [NOT]IN (valor1, ..., valorN)	Verifica conjuntos de valores.
campoN IS [NOT] NULL	Trata de valores NULL.
campoN LIKE '.....'	Compara *strings*.
tabela1 [NOT] CONTAINS tabela2	Compara tabelas.
[NOT] EXISTS tabela	Verifica tabelas vazias.

Nota: um campo tem valor *NULL* quando não está preenchido.

2.ª variante do *SELECT*

```
SELECT fieldlist
FROM table
WHERE condition;
```

onde:

fieldlist é a lista de comandos a serem seleccionados.

table é a tabela em referência dos campos a serem seleccionados.

condition uma condição que os registos seleccionados verificam; podem ser utilizados operadores relacionais, operadores lógicos e os operadores *In*, *Is between* e *like*.

Para o *Access*, * quer dizer 0 ou mais caracteres e ? quer dizer 1 caracter; no entanto, noutros SGBDRs o sinal % corresponde ao * e o sinal _ corresponde ao ?

Se quisermos procurar os próprios caracteres * ou ?, teremos de os preceder de um carácter que anule o contexto especial dos mesmos (carácter de «escape») e indicar o carácter de escape:

Exemplo

procura nomes de clientes cuja cidade tem o carácter ?

```
SELECT nome
FROM clientes
WHERE cidade LIKE '*=?*' ESCAPE '='
```

Existem outras variações aplicadas ao padrão de pesquisa, como, por exemplo, '[abc]*': qualquer *string* começada pelo carácter a, b ou c.

Precedência dos operadores:

Ordem decrescente ↓		
	( )	parêntesis
	*, /	multiplicação, divisão
	+, -	soma, subtracção
	NOT	negação lógica
	AND	conjunção lógica
	OR	disjunção lógica

Cláusula *FROM*

A cláusula *FROM* especifica o(s) nome(s) da(s) tabela(s) em que se encontram os registos a seleccionar. Quando é indicada mais do que uma tabela, a instrução *SELECT* produz um conjunto de registos cujos campos são os indicados na lista de campos.

FROM

```
SELECT fieldlist
FROM tableexpression
[ WHERE condition ];
```

onde:
tableexpression especifica o(s) nome(s) da(s) tabela(s) que contém(êm) os campos e os registos seleccionados.

Cláusula *ORDER BY*

A cláusula *ORDER BY* ordena apenas os registos seleccionados pelos valores de um conjunto de campos especificados.

ORDER BY

```
SELECT fieldlist
FROM tableexpression
WHERE condition
ORDER BY field1 [ASC | DESC ][, field2 [ASC | DESC ][, ...]];
```

onde:
ASC especifica a ordem ascendente e DESC especifica a ordem descendente.

Cláusula AS (*Aliases*)

É possível definir novos nomes (*aliases*) para o nome das tabelas e/ou dos campos utilizados numa instrução *SELECT*.

Exemplo

```
SELECT idade AS idade_agora, idade +5 AS idade_depois
FROM alunos;
```

Funções de agregação

Estas funções possibilitam o resumo dos resultados de uma selecção de registos, como alternativa à selecção de registos. Podem ser utilizadas as funções Sum, AVG, Max, Min e Count.

Cláusula *GROUP BY*

A utilização de funções de agregação é feita frequentemente em conjunto com a cláusula *GROUP BY*. Esta especifica os conjuntos de registos seleccionados que são objecto da(s) função(ões).

Quando é utilizada a cláusula *GROUP BY*, só podem ser indicados na cláusula *SELECT* os atributos incluídos na cláusula *GROUP BY* (para além daqueles que são objecto de uma função).

Cláusula *GROUP BY ... HAVING*

A cláusula *GROUP BY* pode ser utilizada em conjunto com *HAVING* para restringir os registos seleccionados daqueles que verificam a condição especificada em *HAVING*.

Outras instruções *SQL*

A linguagem *SQL* inclui outras instruções para a definição/alteração do esquema de uma base de dados:

CREATE TABLE

Este comando cria a tabela solicitada e obedece à seguinte forma:

CREAT TABELA <tabela>
(<descrição das colunas>).
(<descrição das chaves>).
PRIMARY KEY (colunas).
FOREIGN KEY (colunas).
REFERENCES <tabela>.

Onde:

<tabela> é o nome da nova tabela a ser criada.

<descrição das colunas> é uma lista de colunas (campos) e os seus respectivos tipos de dados (*smallint, char, maney, varchar, integer, decimal, float, real, date, time, timestamp, logical, number*).

<descrição das chaves> Ré a lista de colunas que são tratadas como chave estrangeira.

Alguns campos podem receber o valor *NULL* (nulo) e o campo definido como chave primária, além de não poder receber *NULL*, deve ser um campo *UNIQUE* (sem repetições – chave primária).

INSERT

É o comando de inserção de dados:

Comando *INSERT*

INSERT into nome_da_tabela
[(lista_de_campos)]
VALUES (lista de valores);

Para incluir um registo numa tabela chamada Clientes, por exemplo, com os campos Nome, Idade, Endereço e Telefone, utilizamos os seguintes comandos SQL:

INSERT INTO Clientes (NOME, IDADE, TELEFONE, ENDEREÇO) values ('Remka', '25', '82450045', 'Beco dos apaixonados, nº. 13')

DELETE

Comando DELETE

DELETE FROM nome da_tabela
[WHERE lista_de_condições*];*

Por exemplo, vamos excluir a Remka da lista de clientes:

DELETE FROM Clientes **WHERE** Nome = 'Remka'

O comando acima irá excluir todos os registos que tenham o valor «Remka» no campo «Nome».

UPDATE

É o comando de edição; é o valor ou expressão que vai passar para determinado campo.

Comando *UPDATE*

UPDATE nome_da_tabela
SET nome_do_campo_1=valor_1, ..., nome_do_campo_n=valor_n
[WHERE lista_de_condições]

Alterando a tabela do exemplo anterior, vamos alterar somente o telefone e o endereço da Remka:

UPDATE Clientes **SET** Telefone = '84450045', Endereço = 'Beco dos Ricaços'
WHERE Nome = 'Remka'

Considere a base de dados Moçambola.mdb
Equipas (e_número, e_nome, e_cidade, e_director)

e_número	e_nome	e_cidade	e_presidente
12	Matolinhas	Matola	Cossa
15	Chiveve	Beira	Simango
20	Wampula	Nampula	Nhapepe
24	Pembinhas	Pemba	Abú

Treinadores (t_número, t_nome, t_telefone, t_equipa)

e_número	e_nome	t_telefone	t_equipa
1	Tico Tico	823311440	12
2	Mexer	845500999	12
3	Dominguez	841122330	15
4	Kapango	820077441	20
5	Dário Monteiro	822200220	24
6	Paito	829911337	24

Experiência (ex_equipa, ex_treinador, ex_tipo, ex_anos)

ex_equipa	ex_treinador	ex_tipo	ex_anos
12	1	Juniores	10
12	1	Seniores	5
12	2	Iniciados	2
12	2	Juniores	3
12	2	Juvenis	4
15	3	Juniores	15
24	5	Juvenis	12

Bolas (b_equipa, b_ref, b_fabricante)

b_equipa	b_ref	b_fabricante
12	1	Adidas
12	9	Reebok
12	13	Jabulane
15	1	Adidas
20	3	Olimpic
20	4	Nike
24	18	Reebok
24	21	Jabulane

Jogadores (j_número, j_nome, j_idade)

j_número	j_nome	j_idade
853	Hugo	21
325	Yuri	32
1311	Sandro	28
7917	Alex	25
7002	Gui	27
8024	Samito	25
1943	Ivenildo	24
1054	Felix	27
6465	Deron	24
3980	Ternílio	21
6638	Saló	33

Filiação (f_jogador, f_equipa, f_anos, f_média)

f_jogador	f_equipa	f_anos	f_média
853	15	5	0,3
853	20	3	0,32
325	12	10	0,257
1311	20	1	0,283
7917	12	1	0,223
7917	15	7	0,246
7917	24	2	0,24
7002	24	3	0,29
8024	15	1	0,195
8024	20	3	0,227
1943	15	4	0,298
1054	12	6	0,307
6465	20	12	0,31
6465	24	3	0,28
3980	12	5	0,25
3980	20	2	0,24
3980	24	8	0,265
6638	20	7	0,283

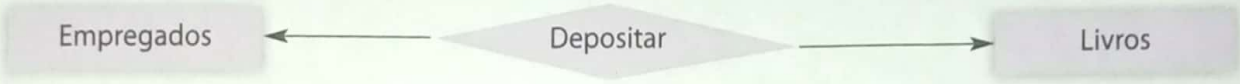
Base de Dados MOÇAMBOLA.mdb

I. Selecções Simples de Tabelas

1. Seleccione todos os dados da Tabela Equipas.
2. Seleccione todos os dados da Equipa número 12.
3. Seleccione o número e o nome de todas as equipas.
4. Seleccione o número e o nome dos jogadores com mais de anos de idade.
5. Seleccione os treinadores que treinaram juniores durante 5 ou mais anos.
6. Seleccione os registos de experiência dos treinadores que treinaram juniores ou que tenham mais do que 10 anos de experiência.
7. Seleccione os números das equipas que utilizam bolas da marca Jabulane.
8. Seleccione os números das equipas que utilizam bolas da marca Adidas, mas eliminando as linhas repetidas.
9. Seleccione os jogadores que têm idades compreendidas entre 25 e 29 anos, ordenados por idade.
10. Seleccione os registos das bolas fabricadas pelas marcas *Reebok* e *Olimpic*.
11. Seleccione os registos dos jogadores cujo nome começa pela letra S.

II. Funções agredadas ou compostas

1. Seleccione o número (quantidade) de equipas que disputam o Moçambola.
2. Seleccione o número (quantidade) de fabricantes diferentes que produzem bolas usadas no campeonato.
3. Seleccione o número (quantidade) de jogadores com idade superior a 30 anos.
4. Determine o número total de anos de experiência do treinador António do Académico (equipa número 12).
5. Os empregados de um gabinete de advogacia decidiram criar uma pequena biblioteca para partilharem os livros que cada um tem. Numa primeira fase pretendem uma base de dados que registe quem disponibiliza os livros (oferecidos ou somente emprestados) e a identificação de cada livro. Considere que todos os livros têm cotas diferentes.

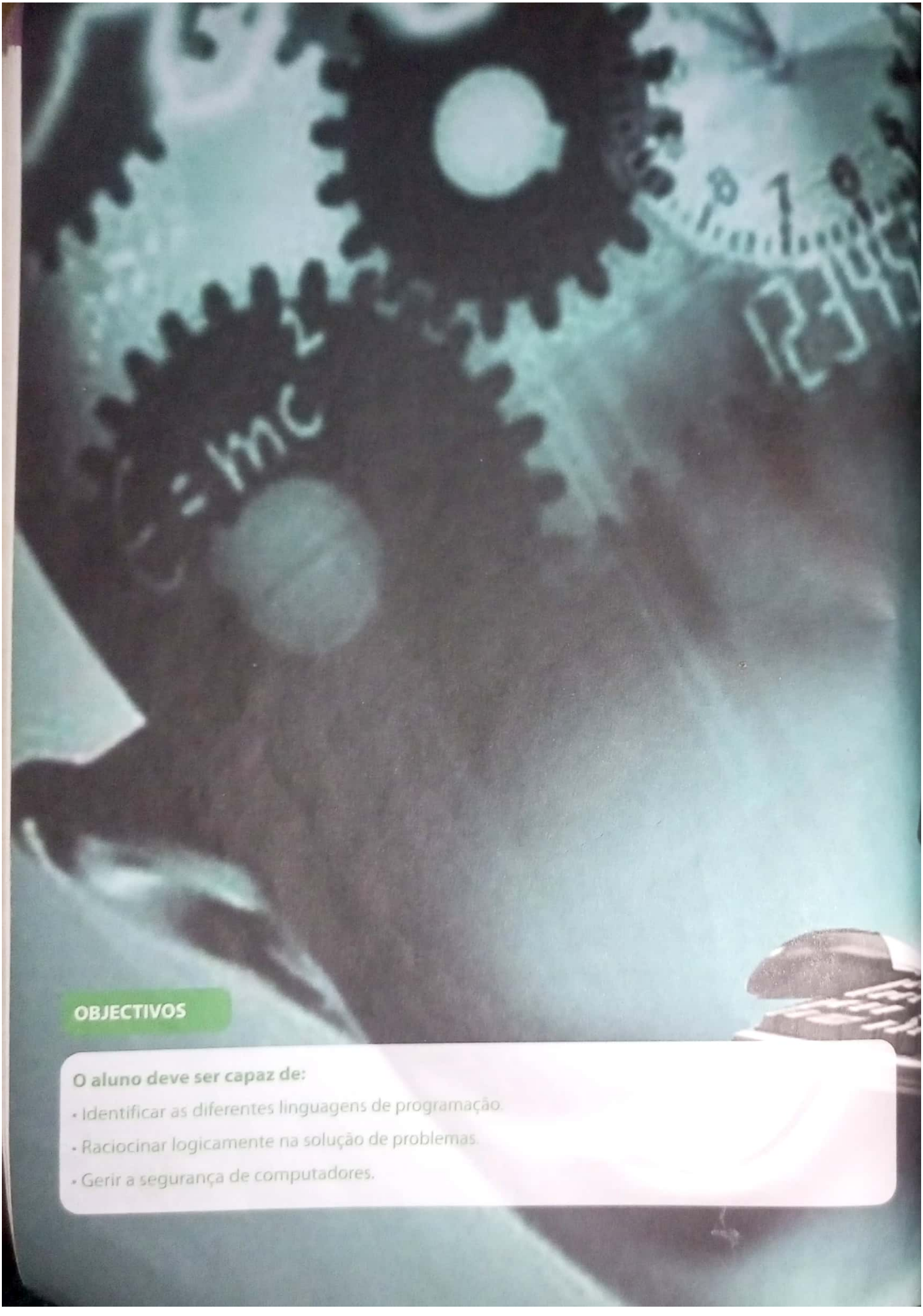


Empregados (*nome, bi, ncontribuinte, morada, telefone*)
Livros (*cota, titulo, autor, assunto, editora, ano, oferecido, biEmpregado*)

a) Crie uma base de dados vazia com o nome **BDLivros.mdb**
b) Crie as duas tabelas (T_Empregados, T_Livros) necessárias para esta base de dados tendo em conta as seguintes características:

T_empregados		
Nome do Campo	**Tipo de Dados**	**Características Gerais**
BI	Número Texto ou	Chave primária Nenhum valor pré-definido
N_Contribuinte	Número Texto Texto	Os dígitos do número de contribuinte deverão aparecer separados por um espaço entre cada 3 dígitos. Exemplo: 204 125 325 Nenhum valor pré-definido/sem valor
Nome	Texto ou Número	Tamanho 80 É um campo obrigatório.
Morada		Tamanho 255
Telefone		Exemplo de como deverá aparecer: (258) 214 978

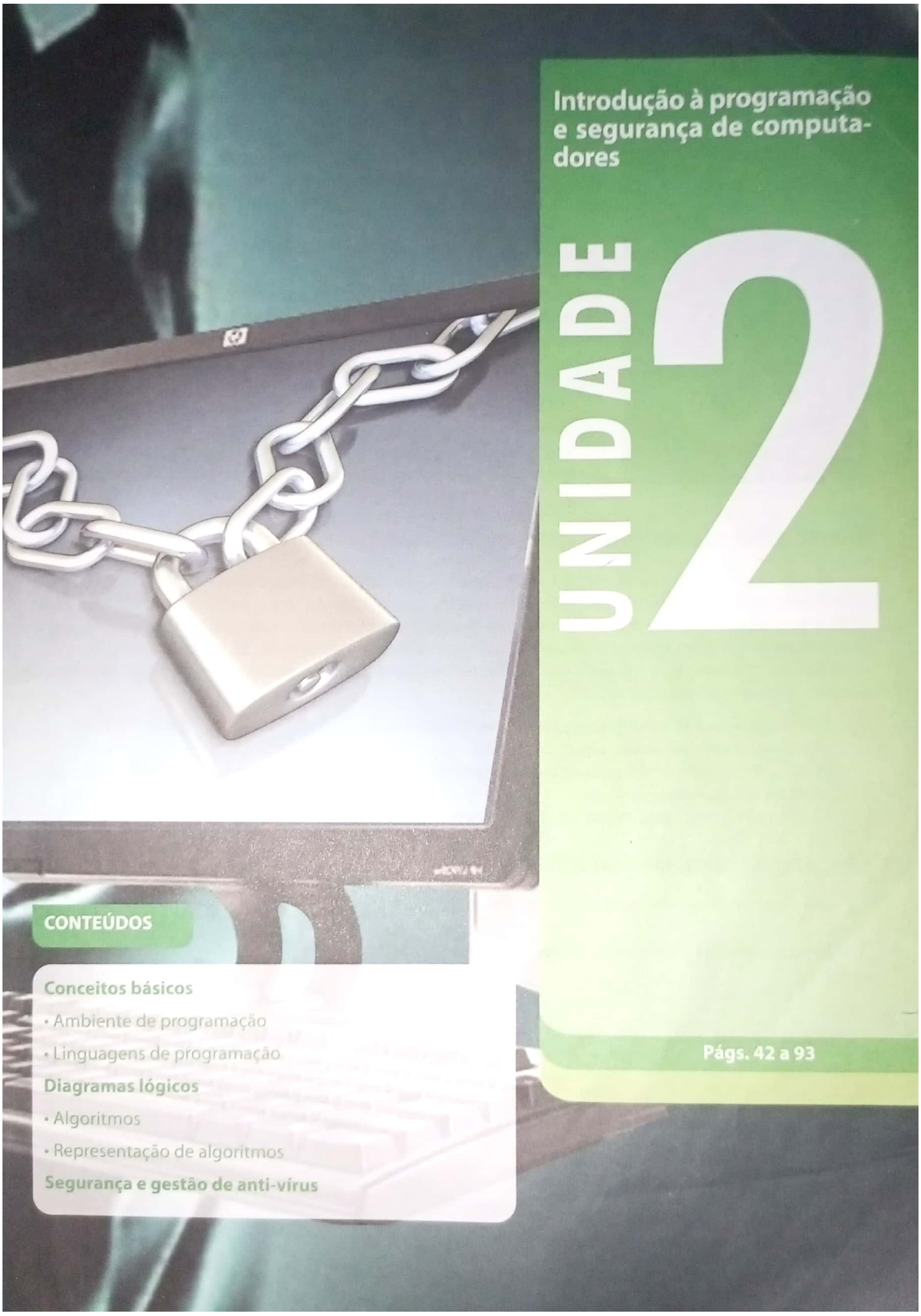
T_livros		
Nome do campo	**Tipo de Dados**	**Características Gerais**
Cota	Número	Chave primária
Título	Texto	Tamanho 50 Campo obrigatório
Autor	Texto	Tamanho 80
Assunto	Assistente de pesquisa	Deverá aparecer uma lista pré-definida com os seguintes valores: economia, política, poesia, informática, diversos.
Editora	Texto	Tamanho 80
Ano	Número	Tamanho: inteiro Valor pré-definido: ano actual (use, para tal, funções adequadas) Apenas deverá aceitar anos superiores a 1900. A mensagem, caso o ano esteja incorrecto, deverá ser «O ano de publicação tem de ser superior a 1900» Legenda: Ano de Publicação



OBJECTIVOS

O aluno deve ser capaz de:

- Identificar as diferentes linguagens de programação.
- Raciocinar logicamente na solução de problemas.
- Gerir a segurança de computadores.



Introdução à programação e segurança de computadores

UNIDADE 2

CONTEÚDOS

Conceitos básicos

- Ambiente de programação
- Linguagens de programação

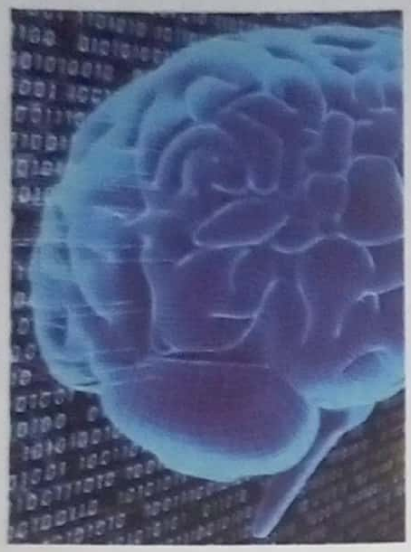
Diagramas lógicos

- Algoritmos
- Representação de algoritmos

Segurança e gestão de anti-vírus

Págs. 42 a 93

Introdução à programação



A automatização é um processo em que uma tarefa deixa de ser desempenhada pelo Homem e passa a ser realizada por máquinas, sejam dispositivos mecânicos (como as máquinas industriais), electrónicos (como os computadores), ou de natureza mista (como os robôs). Para que a automatização de uma tarefa seja bem-sucedida é necessário que a máquina que vai realizá-la seja capaz de desempenhar cada uma das etapas constituintes do processo a ser automatizado com eficiência, de modo a garantir a repetibilidade do mesmo. Assim, é necessário que seja especificado com clareza e exactidão o que deve ser realizado em cada uma das fases do processo a ser automatizado, bem como a sequência em que estas fases devem ser realizadas.

À especificação da sequência ordenada de passos que deve ser seguida para a realização de uma tarefa, garantindo a sua repetibilidade, dá-se o nome de **algoritmo** e à transformação deste para uma forma compreensível pelo computador chama-se **Programa de computador**.

Conceito de algoritmo

O uso de **algoritmos** é quase tão antigo quanto a matemática, embora com o passar do tempo a própria matemática se tenha esquecido dele. Porém, com o advento das máquinas de calcular e, mais tarde dos computadores, o uso de algoritmos ressurgiu com grande vigor, como uma forma de indicar o caminho para a solução dos mais variados problemas desde os mais simples do nosso dia a dia aos mais complexos. Por exemplo, a maneira como uma pessoa toma banho é um algoritmo. Outros algoritmos frequentemente encontrados são:

- Instruções para se utilizar um aparelho electrodoméstico.
- Uma receita para preparar algum prato.
- Guia de preenchimento para declaração de um imposto como IRPS (Imposto de Rendimento para Pessoas Singulares)
- A maneira como as contas de água, luz e telefone são calculadas mensalmente, etc.

O que é um algoritmo?

São vários os conceitos para algoritmo. Eis alguns exemplos:

Um conjunto finito de regras que provê uma sequência de operações para resolver um tipo de problema específico.

Knuth

Sequência ordenada, e não ambígua, de passos que levam à solução de um dado problema.

Tremblay

Processo de cálculo ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições, as regras formais para a obtenção do resultado ou da solução do problema.

Aurélio

E ainda, segundo Knuth, o termo algoritmo é derivado do nome de um matemático persa do século IX, chamado Abu Jafar Maomé Bin Musa al-Khowarizm (1825).

Portanto, um **algoritmo** pode ser definido como uma sequência finita de passos (instruções) para resolver um determinado problema. Sempre que desenvolvemos um algoritmo estamos a estabelecer um padrão de comportamento que deverá ser seguido (uma norma de execução de acções) para alcançar o resultado de um problema.

Mas atenção! algoritmo não é a solução do problema, pois, se assim fosse, cada problema teria um único algoritmo. O algoritmo é um caminho para a solução de um problema, e, em geral, os caminhos que levam à uma solução são muitos. A solução é obtida por meio da execução do algoritmo, seja mentalmente, ou manualmente, usando lápis e papel ou por meio do computador.

Algoritmos. Porquê ?

Vejamos o que algumas pessoas importantes para a Ciência da Computação disseram a respeito de algoritmo:

- *Knuth* – Professor da Universidade de Stanford, autor da colecção «*The art of computer programming*: *A noção de algoritmo é básica para toda a programação de computadores.*
- *Wirth* – Professor da Universidade de Zurique, autor de diversos livros da área e responsável pela criação de linguagens de programação como *Algol, Pascal* e *Modula -2*: *O conceito central da programação e da ciência da computação é o conceito de algoritmo.*

Portanto, a importância do algoritmo está no facto de termos de **especificar uma sequência de passos lógicos** para que o computador possa executar uma tarefa qualquer, pois o mesmo por si só não tem vontade própria, faz apenas o que mandamos. Com uma ferramenta algorítmica, podemos conceber uma solução para um dado problema, independente de uma linguagem específica e até mesmo do próprio computador.

Características

Todo o **algoritmo** deve apresentar algumas características básicas:

- Ter fim.
- Não dar margem à dupla interpretação (não ambíguo).
- Capacidade de receber dado(s) de entrada do mundo exterior.
- Poder gerar informações de saída para o mundo externo ao do ambiente do algoritmo.
- Ser efectivo (todas as etapas especificadas no algoritmo devem ser alcançáveis num tempo finito).

O **algoritmo 1** é um exemplo simples de algoritmo (sem condições ou repetições) para a troca de um pneu.

Algoritmo 1: Troca de pneu do carro

1. Desligar o carro.
2. Pegar as ferramentas (chave e macaco).
3. Pegar o sobressalente.
4. Afrouxar ligeiramente os 4 parafusos do pneu furado.
5. Suspender o carro com o macaco.
6. Desenroscar os 4 parafusos do pneu furado.
7. Colocar o sobressalente.
8. Enroscar os 4 parafusos.
9. Baixar o carro com o macaco.
10. Guardar as ferramentas.

Um algortimo, quando programado num computador, é constituído por pelo menos três partes, sendo elas:

1. Entrada de Dados.
2. Processamento de Dados
3. Saída de Dados

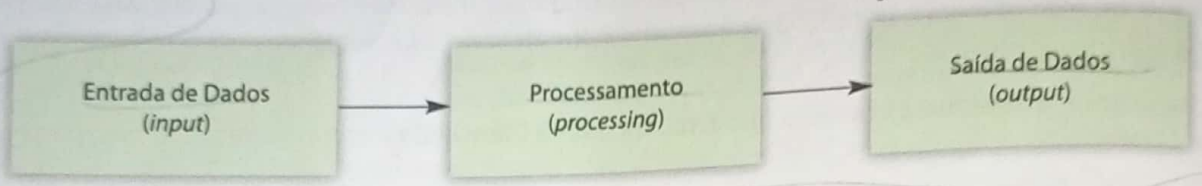
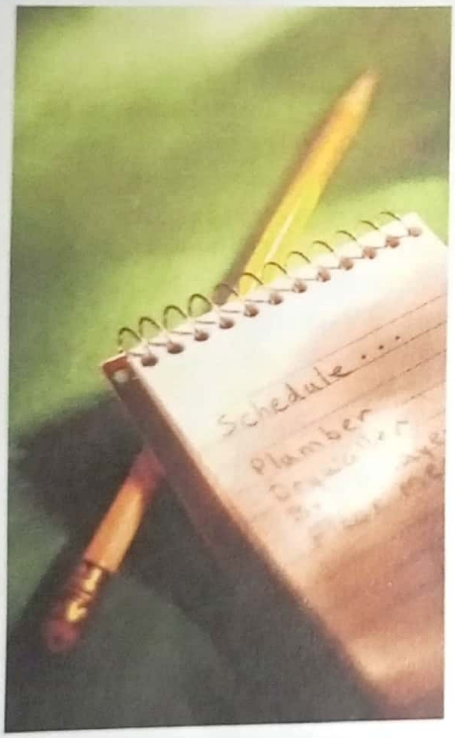


Fig. 2.1 Partes básicas de um algoritmo



Na **entrada**, são fornecidas as informações necessárias para que o algoritmo possa ser executado. Estas informações podem ser fornecidas na altura em que o programa está a ser executado ou podem estar embutidas no mesmo.

No **processamento** são avaliadas todas as expressões algébricas, relacionais e lógicas, assim como todas as estruturas de controlo existentes no algoritmo (condição e/ou repetição).

Na **saída**, todos os resultados do processamento (ou parte deles) são enviados para um ou mais dispositivos de saída, como: monitor, impressora, ou até mesmo a própria memória do computador.

Por exemplo, considere o **algoritmo 2** que tem como objectivo calcular a área de uma circunferência dada, como deve estar lembrado, por $A = \pi r^2$. Para calcular a área é necessário saber os valores do raio r e do π. Considerando que o valor de π é constante o mesmo poderá ser gravado (definido) dentro do próprio algoritmo, e a entrada para o processamento desse algoritmo consistirá nesse valor juntamente com o valor do raio R (que deve ser informado pelo usuário pelo teclado). O processamento do algoritmo será a realização do cálculo πr^2 e a atribuição do resultado dessa expressão para a variável A. A parte da saída consistirá na escrita do valor de A no monitor.

Algoritmo 2: Cálculo da área da circunferência

1. $\pi \leftarrow$ 3,14 {entrada para o processamento - *input*}
2. Leia R {entrada para o processamento - *input*}
3. A $\leftarrow \pi \cdot r^2$ {processamento}
4. Escreva A {saída - *output*}

Síntese

«Algoritmo é um conjunto finito de regras, bem definidas, para a solução de um problema num tempo finito e com um número finito de passos.»

Formas de representação

Ao longo dos anos surgiram muitas formas de representação dos algoritmos, algumas utilizando linguagens semelhantes às linguagens de programação, ou as próprias linguagens de programação e outras utilizando formas gráficas de representação dos algoritmos. Porém, não há um consenso em relação à melhor. Dentre as formas de representação, nos últimos anos, deu-se acentuada preferência por formas estruturadas, cuja principal vantagem é a de facilitar a legibilidade e a compreensão dos algoritmos.

Dentre as formas de representação de algoritmos mais conhecidas destacam-se:

- A descrição narrativa
- O fluxograma convencional
- O pseudocódigo, também conhecido como linguagem estruturada

Descrição narrativa

Forma pela qual os algoritmos são expressos directamente em linguagem natural, podendo-se fazer uso do português para descrevê-los.

Receita de bolo

1. Providencie manteiga, ovos, 2 kg de farinha, etc.
2. Misture os ingredientes.
3. Despeje a mistura na forma de bolo.
4. Leve a forma ao forno.
5. Espere 20 minutos.
6. Retire a forma do forno.
7. Deixe esfriar.
8. Prove.

Esta representação é pouco aplicada na prática porque o uso de linguagem natural muitas vezes dá lugar a más interpretações, ambiguidades e imprecisões. Por exemplo, logo na primeira linha do exemplo anterior, não se diz que quantidade de manteiga, nem de ovos!

Fluxograma

É uma representação gráfica de algoritmos em que formas geométricas diferentes implicam acções (instruções, comandos) distintas. Tal propriedade facilita o entendimento das ideias contidas nos algoritmos.

Por via de regra, o fluxograma resume-se a um **único símbolo inicial**, por onde a execução do algoritmo começa, e a **um** ou **mais símbolos finais**, que são pontos onde a execução do algoritmo se encerra. Partindo do símbolo inicial, há sempre **um único caminho** orientado a ser seguido, representando a existência de uma única sequência de execução das instruções. Isto pode ser melhor visualizado pelo facto de que, apesar de vários caminhos poderem convergir para uma mesma figura do diagrama, há sempre um **único saindo desta**. Excepções a esta regra são os símbolos finais, dos quais não há nenhum fluxo a sair, e os símbolos de decisão, de onde pode haver mais de um caminho de saída (normalmente dois), representando uma **bifurcação** no fluxo.

A figura na página seguinte mostra as principais formas geométricas usadas em fluxogramas.

Vantagens:

- É uma das ferramentas mais conhecidas.
- Figuras dizem muito mais que palavras.
- É um padrão universal.

Desvantagens:

- Dedica pouca atenção aos dados, não oferecendo recursos para descrevê-los ou representá-los.
- Complica-se à medida que o algoritmo cresce.

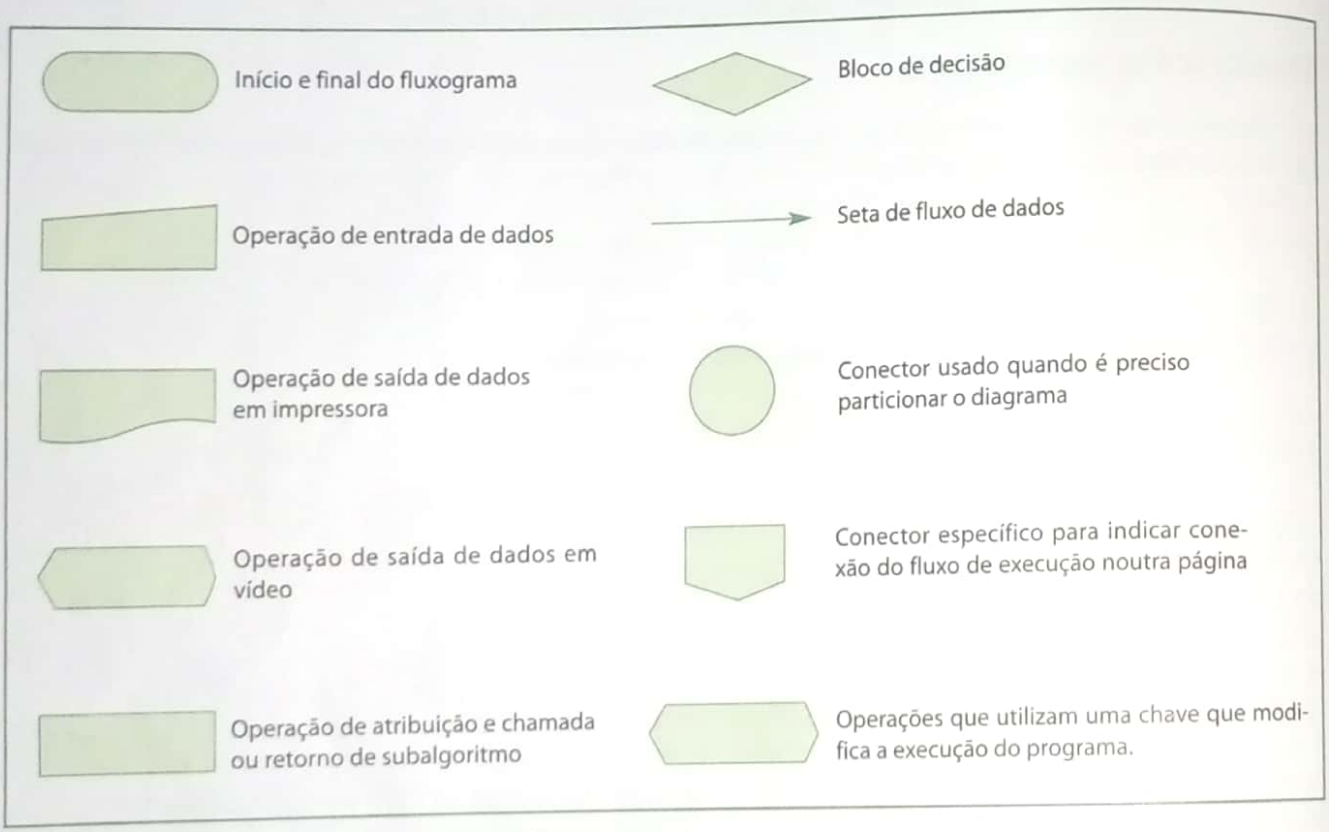


Fig. 2.2 Principais formas geométricas usadas em fluxogramas

Pseudocódigo

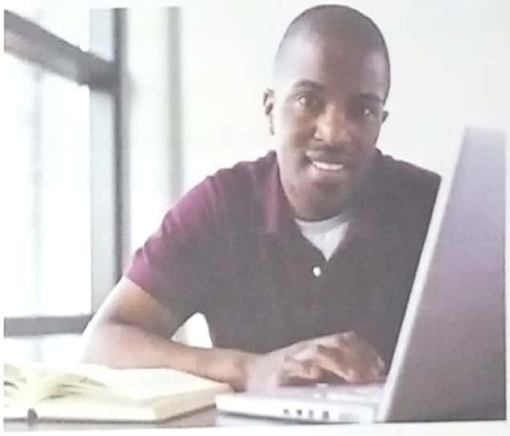
Forma de representação de algoritmos que se assemelha muito ao modo como os programas são escritos. Esta forma de representação permite que os algoritmos nela representados possam ser traduzidos, quase que directamente, para uma linguagem de programação.

Um **pseudocódigo** começa com a indicação **Início** e termina com o termo **Fim**. Entre ambos seguir-se-ão os passos necessários à resolução do problema. É boa prática em programação que o conjunto de declarações sejam estruturados (utilizando um pequeno avanço), por forma a que, com um olhar rápido, nos apercebamos da estrutura do programa, como no exemplo que se segue.

Exercícios resolvidos

Soma de dois números reais lidos a partir do teclado

```
Início
  reais: x, y, z
  Apresenta «Introduza o valor de x - »
  ler x
  Apresenta «Introduza o valor de y - »
  ler y
  z = x + y
  Apresentar «A soma de», x, «com» y « = » z.
Fim
```



Interpretemos este algoritmo:

- A primeira declaração **Início** especifica o início do programa.
- A segunda declaração especifica a criação das variáveis, **x**, **y** e **z** do tipo real, ou seja, a preparação de três locais em memória, a que corresponderão os nomes x, y e z, onde serão guardados valores reais.
- A terceira declaração **Apresenta «Introduza o valor de x -»** indica que será enviada para o ecrã a mensagem fixa que se encontra entre aspas.
- A quarta declaração **ler x**, especifica que será lido um valor a partir do teclado, que corresponderá ao valor a guardar na variável x.
- A quinta declaração **Apresenta «Introduza o valor de y -»** indica que será enviada nova mensagem para o ecrã.
- A sexta declaração **ler y**, especifica que será lido um valor a partir do teclado, que corresponderá ao valor a guardar na variável y.
- A sétima declaração **z = x + y** especifica que os valores guardados nas variáveis x e y serão somados e o seu valor colocado na posição de memória correspondente à variável z. A oitava declaração **Apresentar «A soma de», x, «com» y, « = », z** especifica a apresentação em ecrã de uma mensagem composta por uma sequência de mensagens fixas e valores das variáveis, respectivamente:

  A soma de (valor da variável x) **com** (valor da variável y) = (valor da variável z).

- A última mensagem **Fim** especifica o fim do programa.

Supondo que se executava este programa e se pretendiam somar os valores **1,75** e **3,5** então no ecrã teríamos a seguinte sequência:

Introduza o valor de x - 1,75	(1,75 é valor introduzido pelo operador)
Introduza o valor de y - 3,5	(3,5 é valor introduzido pelo operador)
A soma de 1,75 com 3,5 = 5,25	(mensagem de resposta)

O mesmo algoritmo poderia ser apresentado na forma de **fluxograma** utilizando uma simbologia própria que nos permite uma visão gráfica rápida do algoritmo, muito eficaz em variadíssimas situações.

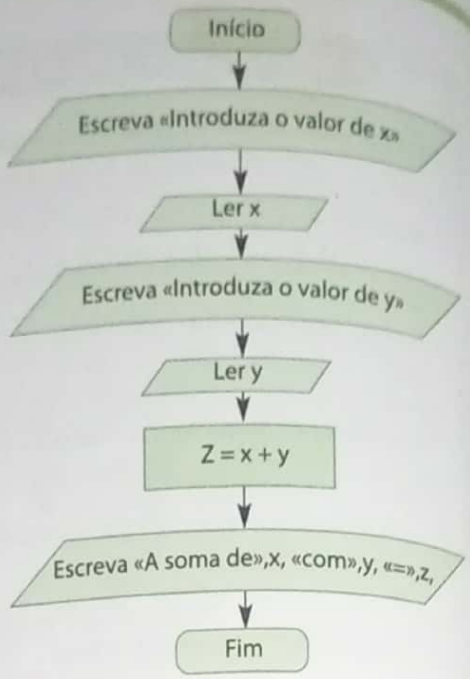


Fig. 2.3 Fluxograma soma de dois números

Como se pode verificar, existe uma correspondência directa entre cada linha do pseudocódigo e do fluxograma, pelo que podemos converter directamente um no outro.

Vantagens do Pseudocódigo:

- Pode-se usar o português como base.
- Pode-se definir quais e como os dados estarão estruturados.
- Passagem quase imediata do algoritmo para uma linguagem de programação qualquer.

Desvantagens:

- Exige a definição de uma linguagem não real para o trabalho.
- Não está padronizado.

Um ambiente para escrever algoritmos

A seguir vamos descrever uma máquina hipotética na qual escreveremos os nossos algoritmos. O nosso computador hipotético apresentará a seguinte organização:

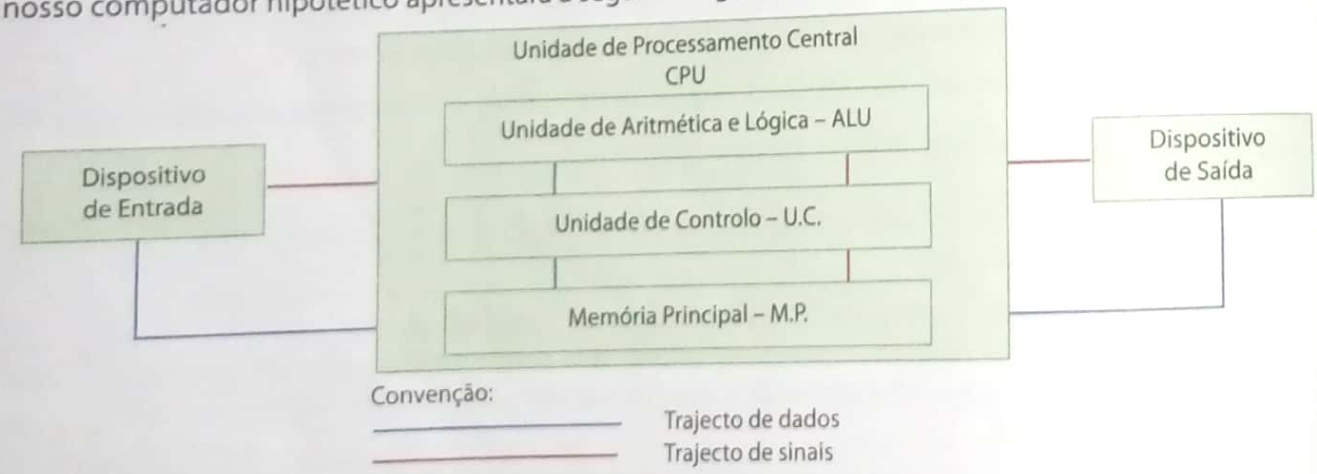


Fig. 2.4 Esquema de um computador hipotético

Actividades

Fazendo uma breve recapitulação da arquitectura do computador segundo Von Neumann que vimos na 10.ª classe, completa os espaços em branco relativos às partes básicas do nosso computador hipotético:

1. Guarda o algoritmo a ser executado e os dados a serem utilizados pelo mesmo: _____
_____.
2. É o meio pelo qual os dados que serão trabalhados pelo algoritmo vão ser introduzidos no nosso computador hipotético: _____.
3. Exerce controlo sobre as demais partes do nosso computador. É uma verdadeira gerente que distribui tarefas às outras unidades: _____.
4. Parte responsável pelas operações matemáticas e avaliações lógicas: _____
_____.
5. É o meio de que se dispõe para a apresentação dos resultados obtidos: _____
_____.

Funcionamento do computador

Todos os computadores, independentemente dos seus tamanhos, são conceitualmente semelhantes ao esquema da fig. 2.4 da página anterior (há naturalmente algumas pequenas diferenças). Pelo que, pode-se afirmar que existem 4 (quatro) operações básicas que qualquer computador pode executar:

- **Operações de entrada e saída de dados:** ler dados do teclado, escrever dados no ecrã e imprimir dados através da impressora são exemplos destas operações.
- **Operações aritméticas:** são utilizadas na realização de operações matemáticas (adição, subtracção, multiplicação e divisão).
- **Operações lógicas e relacionais:** têm aplicabilidade em comparações e testes de condições lógicas.
- **Movimentação de dados entre os vários componentes:** as operações aritméticas são executadas na Unidade Lógica e Aritmética, necessitando da transferência dos dados para essa unidade e da volta do resultado final para ser guardado na memória.

Agora, partindo destas quatro operações como é que seria resolvido o problema de calcular a soma de dois números A e B?

1.º passo: operação de entrada dos números A e B (*input*).

2.º passo: movimento do valor dos números A e B entre a memória e a UAL.

3.º passo: operação aritmética de somar os dois números A e B.

4.º passo: movimentação do resultado da UAL para guardar na memória.

5.º passo: operação de saída do resultado, que está guardado na memória, para o dispositivo de saída desejado (*output*).

Quer dizer, resumindo, pode-se afirmar que **escrever algoritmos** ou, se assim o quisermos, **programar** consiste em dividir qualquer problema em muitos pequenos passos, usando uma ou mais das quatro operações básicas citadas. Esses passos que compõem o algoritmo são denominados **comandos**.

Síntese

A aprendizagem de algoritmos não se consegue a não ser através de muitos exercícios!
É que os algoritmos não se aprendem:

- Copiando algoritmos.
- Estudando algoritmos.

Algoritmos só se aprendem:

- Construindo algoritmos.
- Testando algoritmos.

Estruturas básicas da construção de algoritmos

Basicamente existem três estruturas de controlo nas quais se baseiam os algoritmos: sequencial, decisão e repetição.

Estrutura sequencial

A **estrutura sequencial** é aplicada quando a solução do problema pode ser decomposta em passos individuais. Nesse caso, os comandos do algoritmo fazem parte de uma sequência, onde é relevante a ordem na qual os mesmos se encontram, pois serão executados um de cada vez, estritamente, de acordo com essa ordem.

Consideremos, por exemplo, o seguinte problema:

Exercícios resolvidos

Dados três valores positivos, *a*, *b*, *c*, determinar a sua média aritmética, harmónica, geométrica e ponderada com pesos respectivos de 1, 2 e 3. As tarefas a serem executadas para a solução deste problema podem ser descritas da seguinte forma:

Resolução

1. Obter os valores de *a*, *b*, *c*
2. Calcular a média aritmética pela fórmula $ma = \frac{a+b+c}{3}$
3. Calcular a média harmónica pela fórmula $mh = \frac{3}{\frac{1}{a}+\frac{1}{b}+\frac{1}{c}}$
4. Calcular a média geométrica pela fórmula $mg = \sqrt{a \times b \times c}$
5. Calcular a média ponderada pela fórmula $mp = \frac{1 \times a + 2 \times b + 3 \times c}{1+2+3}$
6. Apresentar os resultados obtidos: *ma; mh; mg; mp.*
7. Terminar.

A **execução desta tarefa** é correctamente cumprida se executarmos todos os passos (instruções) **na sequência em que elas aparecem**, da primeira até a última, sem omissões e repetições.

Um algoritmo com estas características é denominado **algoritmo de estrutura sequencial.**

Consideremos agora o problema de calcular o fluxo de corrente alternada de um circuito.

Exercícios resolvidos

Cálculo do fluxo de corrente alternativa dados, a voltagem, a resistência, a frequência, a indutância e a capacitância. As variáveis que representam estas grandezas são:

I = corrente (amperes)
R = resistência (ohms)
C = capacitância (farads)
E = voltagem (volts)
L = indutância (henrys)
F = frequência (ciclos/seg)

Resolução

Um algoritmo para resolver este problema testa o seguinte:

Algoritmo Cálculo Corrente
Início
reais: E, R, F, L, C
Apresenta «Introduza os valores de E, R, F, L, C - »
ler E, R, F, L, C

$$I = \frac{E}{\sqrt{R^2 + \left[2\pi FL - \frac{1}{2\pi FL}\right]^2}}$$

Apresentar «Fluxo de corrente alternada», I, « = », I
Fim

O passo referente ao cálculo de I pode ser decomposto em diversas instruções em vez de se usar uma só. Mas, independentemente da decomposição ou não, o algoritmo resume-se nas seguintes instruções básicas:

- Ler valores.
- Calcular valores.
- Escrever resultados.
- Parar.

Todo o algoritmo sequencial tem exactamente esta estrutura.

Estrutura de decisão

Também conhecida por estrutura condicional, nela há a subordinação da execução de um ou mais comandos dependente da veracidade de uma condição.

A estrutura de decisão pode ser de três tipos, nomeadamente:

- Simples
- Dupla
- Múltipla

Estrutura de decisão simples

Consideremos o seguinte algoritmo:

Algoritmo Médias

Início
inteiros: i, a, b, c, x
Apresenta «Introduza os valores de i, a, b, c - »
ler i, a, b, c

Se i = 1 então $\frac{a+b+c}{3}$

Se i = 2 então $\sqrt{a \times b \times c}$

Se i = 3 então $\frac{3}{\frac{1}{a}+\frac{1}{b}+\frac{1}{c}}$

Se i > 3 ou i < 1 então x = 0

Apresentar «Valor da média» , x, « = », x
Fim

O **tipo de decisão** apresentado neste algoritmo é denominado **simples**, pois decide entre a execução ou não de uma ou mais instruções conforme a condição for verdadeira ou falsa.

Estrutura de decisão dupla

Consideremos agora o algoritmo sobre o cálculo de raízes duma equação quadrática como aprendeu na 10.ª classe.

Algoritmo Raízes

Início
reais: a, b, c, x, x_1, x_2, d
Apresenta «Introduza os valores de a, b, c - »
ler a, b, c
Se a = 0 então Início

$$x = -\frac{c}{b}$$

Apresentar «valor de », x, « = », x
Fim
Se não Início
D = $b^2 - 4ac$
Se d < 0 então Apresentar («Equação sem raízes reais»)
Se não Início

$x_1 = \frac{-b+\sqrt{d}}{2a}$ e $x_2 = \frac{-b-\sqrt{d}}{2a}$

Apresentar «As raízes são», x_1 , x_2
Fim

Neste caso o algoritmo **decide** sempre **entre duas acções** ou **conjunto de acções**, executando uma delas e desprezando a outra, conforme a condição for **verdadeira** ou **falsa**.

Assim,

- Se $a = 0$ e $b \neq 0$ calcula e escreve a raiz da equação do primeiro grau.
- Se $a = 0$ e $b = 0$ escreve somente o valor de c.
- Se $a \neq 0$ e $b^2 - 4ac < 0$ escreve a mensagem «Equação sem raízes reais».
- Se $a \neq 0$ e $b^2 - 4ac > 0$ calcula e escreve as raízes da equação do segundo grau que tem a, b, c por coeficientes.

Este tipo de decisão é denominado decisão dupla.

Estrutura de decisão múltipla

Embora os algoritmos com decisão múltipla possam ser resolvidos com o uso de instruções de decisão simples, uma forma mais elegante e que tem correspondência nas linguagens de programação é o uso da instrução *Case*, cuja forma é a seguinte:

```
Case <expressão aritmética> com
        <valor 1>: instrução 1
        <valor 2>: instrução 2
        <valor 3>: instrução 3
        .....
        <valor n>: instrução n
        Senão instrução
Fim_Case
```

Onde:

<valor 1>, <valor 2>, ..., <valor n> são valores assumidos pela expressão aritmética. Para cada valor pode-se especificar uma acção ou um conjunto de acções a serem executadas.

```
Algoritmo Cálculo Médias

    Início
    inteiros: i
    reais: a, b, c, x
    Apresenta «Introduza os valores de i, a, b, c - »
    ler i, a, b, c
    Case i com
    1. x = (a + b + c) / 3
    2. x = √(a×b×c)
    3. x = 3 / (1/a + 1/b + 1/c)
    Senão x = 0
    Apresentar «Valor da média», x, « = », x
    Fim
```

Com a introdução das instruções de decisão, o número de algoritmos que podem ser construídos como solução para um dado problema aumenta consideravelmente.

Por exemplo, consideremos o problema:

Exercício resolvido

Encontrar o máximo divisor comum de dois números «m» e «n» inteiros e positivos.

Resolução 1

Seja «r» o resto da divisão de «m» por «n» com $0 \leq r < n$
Se r = 0 então escrever «n» como resposta e terminar.

Fazer «m» assumir o valor de «n»
«n» assumir o valor de «r» e
Voltar ao passo inicial.

Resolução 2

Se «m» ≥ «n» fazer «m» igual ao resto de «m» dividido por «n» e repetir este mesmo passo.
Trocar os valores de «m» e «n» entre si.
Se «n» = 0 então escrever «m» como resposta e terminar.
Voltar ao passo inicial.

Resolução 3

Dividir «m» por «n» e fazer «r» ser o resto dessa divisão.
Se r = 0 escrever «n» como resposta e terminar.
Dividir «n» por «r» e fazer «m» ser o resto.
Dividir «r» por «m» e fazer «n» ser o resto.
Se m = 0 escrever «r» como resposta e terminar.
Se n = 0 escrever «m» como resposta e terminar.
Voltar ao passo inicial.

E como Peter Druker alguma vez disse:
«Iterar é humano, a recursão é divina»

Então podemos ainda considerar a quarta solução.

Resolução 4

Se $n \neq 0$ então MDC = MDC (n, mod(m,n))
Senão MDC = m.

Onde: mod é a função módulo (resto inteiro da divisão de «m» por «n»).

E certamente estas soluções não são as únicas para a resolução deste problema.

Documento

Tendências actuais

A evolução das linguagens de programação continua, tanto na indústria quanto na pesquisa. Algumas das tendências actuais incluem:

- Mecanismos para a adição de segurança e verificação da confiabilidade para a linguagem: verificação estática prolongada, controlo de fluxo de informação, estático segurança em *threads*.
- Mecanismos alternativos de modularidade: mixins, delegação de programação, programação orientada a aspectos.
- Desenvolvimento de *software* orientado a componentes.
- Metaprogramação, reflexão ou acesso a árvores de sintaxe abstractas.
- Maior ênfase na distribuição e mobilidade.
- Integração com bases de dados, incluindo *XML* e bancos de dados relacionais.
- Suporte para *Unicode* de forma que o código-fonte não esteja restrito aos caracteres contidos no código *ASCII*; permitindo, por exemplo, o uso de *scripts* não-latinos ou pontuação estendida.
- *XML* para a *interfaces* gráficas (*XUL, XAML*).

Pessoas de destaque na história das linguagens de programação

- John Backus, inventor do *Fortran*.
- John McCarthy, inventor do *LISP*.
- Alan Cooper, desenvolveu o *Visual Basic*.
- Edsger W. Dijkstra, desenvolveu o *framework* de programação adequada.
- James Gosling, desenvolveu a linguagem *Oak*, precursora do *Java*.
- Anders Hejlsberg, desenvolveu o *Turbo Pascal* e o *C+*.
- Grace Hopper, desenvolveu o *Flow-Matic*, influenciando o COBOL.
- Kenneth E. Iverson, desenvolveu o *APL*.
- Bill Joy, inventor do vi, um dos primeiros autores do *BSD Unix*, e criador do *SunOS*, que se tornou no sistema operacional *Solaris*.
- Alan Kay, pioneiro da programação orientada ao objecto e autor do *Smalltalk*.
- Brian Kernighan, co-autor do primeiro livro sobre a linguagem *C* junto com Dennis Ritchie, co-autor das linguagens *AWK* e *AMPL*.
- John von Neumann, autor do conceito de Sistema Operacional.
- Dennis Ritchie, inventor do *C*.
- Bjarne Stroustrup, desenvolveu o *C++*.
- Ken Thompson, inventor do *Unix*.
- Niklaus Wirth inventor do *Pascal* e *Modula*.

Estrutura de repetição

Também conhecida por «*looping*» ou laço, esta estrutura permite que **tarefas individuais sejam repetidas** um número determinado de vezes ou tantas vezes quantas uma condição lógica permita.

Na prática, um algoritmo não é executado para um único conjunto de valores conforme visto nos casos anteriores, em algoritmos com repetição, aparecem, com frequência, conceitos como **variável contador** e **variável acumulador.**

Variável contador

É uma variável qualquer que recebe um valor inicial (geralmente 0) e é incrementado em algum outro ponto do algoritmo, a um valor constante (geralmente 1), por exemplo,

```
.
.
.
Cont = 0
.
.
.
Cont = cont +1
```

O significado de:

- **Cont = 0** é armazenar, na posição de memória de nome Cont, o valor zero.
- **Cont = cont + 1** é armazenar na posição de memória Cont o valor que já estava nesta posição acrescida de uma unidade.

Por exemplo, pretende-se calcular e escrever a média das três notas trimestrais T1, T2 e T3, obtidas por cada um dos 30 alunos de uma turma. Para cada aluno será lido o seu número e as suas três notas.

Para esta questão precisamos de uma nova instrução:

```
Enquanto <condição>
    Início
        <sequência de instruções>
Fim_Enquanto...
```

Assim, teríamos o seguinte algoritmo, por exemplo:

```
Algoritmo Média

  Num, Cont : Inteiros
  T1, T2, T3, Média: reais
  Início
     Cont = 0
     Enquanto Cont < 30
        Início
          Ler (num, t1, t2, t3)

        Média = (T1+T2+T3)/3
```

```
Escrever (num, Média)
Cont = Cont + 1
Fim_Enquanto
Fim
```

Suponha agora que se queira, no final, saber a média geral da turma. Aí necessitamos de outro conceito, que é o de **variável acumulador**.

Variável acumulador

É qualquer **variável que recebe um valor inicial constante** (geralmente 0) e é **incrementada** em algum outro ponto do algoritmo a um valor variável, por exemplo,

```
.
.
.
soma = 0
.
.
.
soma = soma + valor
```

O significado de:

- **Soma = 0** é armazenar, no endereço de memória de nome soma o valor zero.
- **Soma = soma + valor** é armazenar na posição de memória soma o valor que já estava nesta posição acrescido do valor da variável valor.

```
Algoritmo Média Geral
    Num, Cont : Inteiros
      Soma, T1, T2, T3, Média, Média Geral: reais
      Início
      Soma = Cont = 0
      Enquanto Cont < 30
      Início
      Ler (num, t1, t2, t3)
      Média = (T1 + T2 + T3) / 3

    Escrever (num, Média)
    soma = soma + Média
    Cont = cont + 1
    Fim_Enquanto
    Média Geral = soma/30
    Escrever (Média Geral)
    Fim
```

Em ambos os casos tem-se um conjunto de instruções que é repetido um certo número de vezes, isto é, o que caracteriza um algoritmo de repetição.

Exercício resolvido

Verificar se um dado número «m» lido é primo.

Resolução

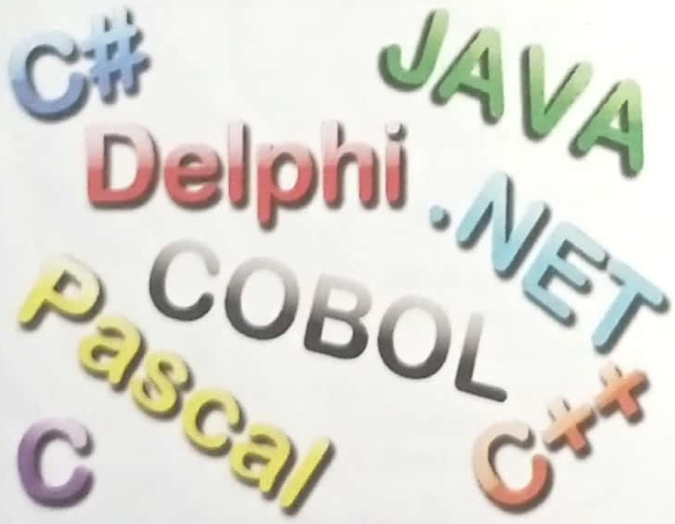
Algoritmo Divisores
c, m, i : Inteiros
Inicio
Ler (m)
Enquanto m > 0
Inicio
C = 2
Para i = 2 até $\frac{m}{2}$

Se $\frac{m}{i} = \text{int}\left(\frac{m}{i}\right)$ então c = c + 1

Fim_Para i
Se c > 2 então escrever (m, «possui» c, «divisores»)

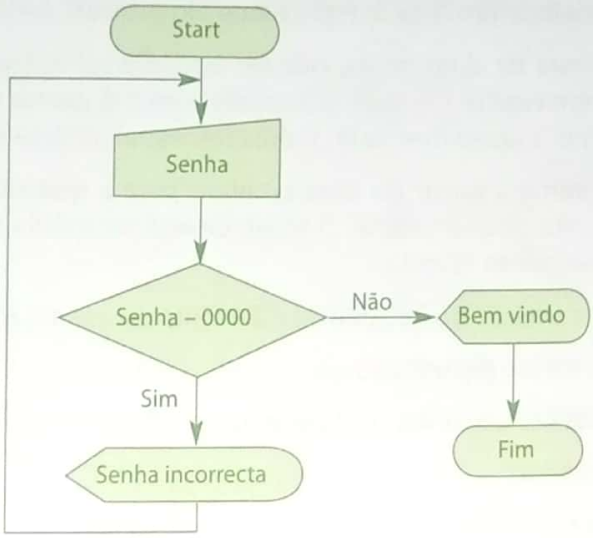
Senão escrever (m, «é número Primo»)
Fim_Enquanto...
Fim

A repetição do tipo «Para» só é utilizada quando se conhece *a priori* o número de valores que devem ser lidos; caso contrário, usa-se a instrução **enquanto**, pois esta é mais geral.



Actividades

1. Escreva um algoritmo que lê 10 valores para m, inteiros e positivos, e, para cada m lido, calcula o factorial de m, escrevendo-o juntamente com o valor lido.
2. Interprete o seguinte fluxograma:



Exercício resolvido

Desenvolva um algoritmo que calcula o valor de π a partir da seguinte expressão matemática:

$$\pi = 4\sum_{k=0}^{n}(-1)^{k}\left[\frac{1}{2k+1}\right]$$

onde n deve ser informado pelo usuário e corresponde ao grau de precisão no cálculo do valor de π.

Resolução

Algoritmo

```
k, n: inteiros
pi: real
escreve (digita n)
   leia(n)
   para k de 0 até n passo 1 faça
   se k % 2 = 0 então
    pi  = pi + 1/(2k + 1)
   senão
   pi =  pi – 1/(2k + 1)
   Fim-se
Fim-para
```

Exercícios de consolidação

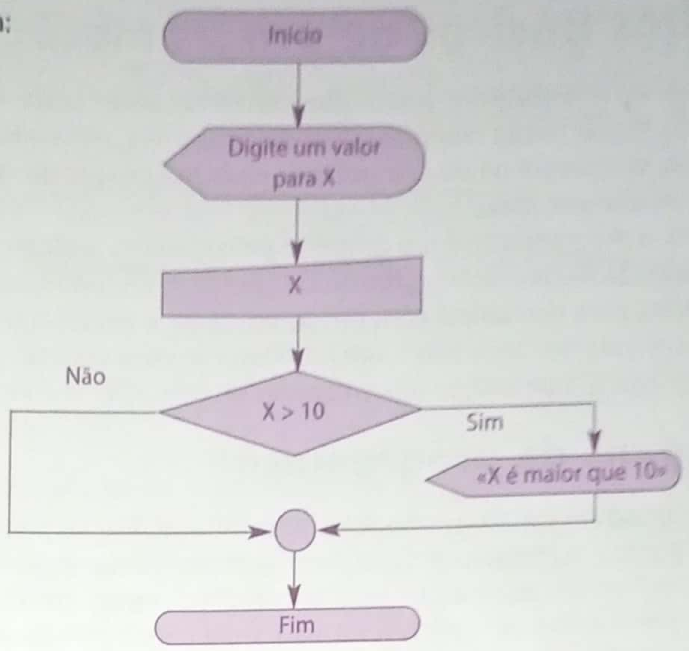
1. Defina, por palavras suas, o que é algoritmo.
2. Cite alguns algoritmos que podemos encontrar na vida quotidiana.
3. Para si qual é a característica mais importante num algoritmo? Justifique.
4. Um algoritmo não pode conter um comando como «Escreve todos os números inteiros positivos». Porquê?
5. Cite as formas básicas para se representar algoritmos, definindo-as.
6. Na sua opinião, qual é a melhor forma de se representar algoritmos? Justifique.
7. Elabore um fluxograma para ler duas notas, calcular a média semestral e informar o aluno se está reprovado ou aprovado, em exame. Considerar a média para a dispensa igual ou superior a 14,0 valores. Para ter direito a exame, o aluno deverá ter média semestral mínima não inferior a 10 valores.
8. Elabore um novo fluxograma a partir do desenvolvido para a questão anterior, informando se o aluno está aprovado ou não após o exame. O aluno deverá ter média semestral não inferior a 10,0 valores obedecendo ao seguinte cálculo:

   média final após exame = (média de frequência x 2 + nota do exame)/3.
9. Reescreva os exercícios 7 e 8 em pseudocódigo.
10. Quais são as estruturas básicas de controlo dos algoritmos? Explique cada uma delas.
11. O que é uma variável contador?
12. Qual é a diferença entre uma variável contador e uma acumulador?
13. Considere o algoritmo em «Moz» dado a seguir e responda (C1, C2, C3, C4 e C5 representam comandos quaisquer):

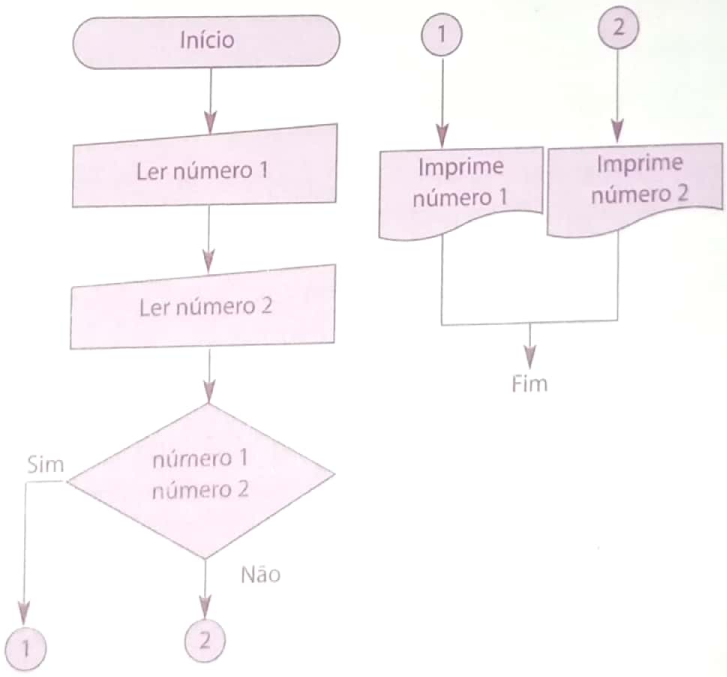
```
ALGORITMO
DECLARE
   b1, b2, b3 LÓGICO
SE b1
ENTÃO C1;
SENÃO SE b2
   ENTÃO INICIO
                SE b3
                ENTÃO C2;
                SENÃO C3;
                C4;
    FIM
C5;
FIM_ALGORITMO
```

a) Se b1 = verdadeiro, b2 = verdadeiro e b3 = falso, que comandos serão executados pelo algoritmo?
b) Se b1 = falso, b2 = verdadeiro e b3 = falso, que comandos serão executados?
c) Quais valores lógicos b1, b2 e b3 devem receber para que somente o comando C5 seja executado?
d) Se b1 = falso, b2 = verdadeiro e b3 = verdadeiro, que comandos serão executados?

14. Interprete o seguinte fluxograma:



15. O que o algoritmo abaixo irá imprimir como resultado caso sejam fornecidos os números 10 (como número 1) e 20 (como número 2)?



16. Escreva um algoritmo que, dado os 3 lados de um triângulo, determine se ele é equilátero, isósceles ou escaleno.

17. Escreva um algoritmo para entrar com três números e imprimir o maior entre eles.

18. Escreva um algoritmo para entrar com 8 números e imprimir a quantidade de números maiores que 4.

19. Construa um algoritmo capaz de representar uma estrutura do tipo Repete-Até.

20. Construa um algoritmo capaz de representar uma estrutura do tipo Faça-Enquanto.

21. Construa sobre um algoritmo que represente o critério de avaliação de um aluno da 12.ª classe. (Consulte o seu DT, como será calculada a sua nota final no final do ano lectivo).

Conceitos básicos de programação

Para que um **computador possa desempenhar uma tarefa** é necessário que esta seja detalhada passo a passo, numa forma compreensível pela máquina, utilizando aquilo a que se chama **programa**. Neste sentido, um programa de computador nada mais é que um **algoritmo escrito** numa forma compreensível pelo computador.

Portanto, a programação é um processo pelo qual um problema ou uma situação pode ser automatizada com a ajuda de um controlador ou computador, tornando, assim, todo o processo mais fácil.

Entretanto, para comunicar com um computador é necessário que ele entenda o utilizador e que o utilizador o entenda também. Isto é, um problema de comunicação que se resolve utilizando uma linguagem (ou linguagens) que ambos sejam capazes de descodificar com precisão.

Linguagens de programação

Uma **linguagem de programação** é uma notação formal para a descrição de algoritmos que serão executados por um computador. Quer dizer, o objectivo da programação é a codificação de algoritmos, listas de instruções que especificam uma sequência de operações que resolvam uma certa questão.

Agora, como todas as notações formais, uma linguagem de programação tem dois componentes: **sintaxe** e **semântica**. A sintaxe consiste num conjunto de regras formais, que especificam a composição de programas a partir de letras, dígitos e outros símbolos, por exemplo, **regras de sintaxe** podem especificar que cada parênteses aberto numa expressão aritmética deve corresponder a um parênteses fechado, e que dois comandos quaisquer devem ser separados por um ponto e vírgula.

As **regras de semântica** especificam o «significado» de qualquer programa, sintacticamente válido, escrito na linguagem.

Linguagem de programação = Símbolos + Regras de sintaxe

Um algoritmo serve, tal qual como já vimos, para resolver um determinado problema para qualquer conjunto de valores das suas variáveis: um programa trabalhará sempre igualmente sejam quais forem os seus dados de entrada.

Assim, um programa é um conjunto de instruções que forma a representação, inteligível pelo computador, de um algoritmo.

Uma **instrução** é uma **cadeia de símbolos de certo alfabeto**. Esta cadeia formar-se-á de acordo com determinadas regras sintácticas e construir-se-á de tal forma que terá um certo sentido (é a semântica da frase). Deste modo, uma linguagem de programação possui e fica definida por um alfabeto, certas regras de sintaxe e uma semântica.

Sucede exactamente como nas linguagens naturais, ainda que nestas últimas a sintaxe e a semântica sejam tão complexas que nenhuma pode ser ainda definida totalmente.

Nos **primeiros computadores**, para introduzir um programa na memória era necessário algum procedimento material para «gravar» um conjunto de uns e zeros em cada divisão, até completar o número total de instruções por que é constituído o programa.

Antes da introdução do programa havia que escrevê-lo numa linguagem que o computador entendesse (que futuramente se denominaria linguagem máquina), quer dizer, à base de zeros e uns sobre um documento ou papel. Depois cada instrução contida nesta informação escrita deve introduzir-se de alguma forma na zona de memória respectiva até completar a carga do programa.

Como se pode imaginar, este trabalho era fastidioso, sujeito a uma elevada percentagem de erros e requeria muito tempo. Por isso, foi necessário desenvolver **ferramentas** que **permitissem simplificar** estes procedimentos. Estas ferramentas são as chamadas linguagens de programação.

Um programa de computador é escrito numa determinada linguagem, a linguagem de programação. Uma **linguagem de programação** é um conjunto de ferramentas, regras de sintaxe e símbolos ou códigos que nos permitem escrever programas de computador destinados a instruir o computador para a realização das suas tarefas.

Classificação das linguagens de programação

Genericamente, as linguagens de programação dividem-se em dois grandes grupos:

- Linguagens de baixo nível
- Linguagens de alto nível

Linguagens de baixo nível

Já sabemos que tudo o que o computador faz é executado sob as ordens de um programa e que a única linguagem que ele entende é a dos *bits* (10.ª classe). Esta linguagem consiste na representação dos dados por sequências de zeros e uns, que desencadeiam **determinadas acções no processador**; por isso, esta linguagem é conhecida por **linguagem máquina**.

As linguagens baseadas em código máquina designam-se por **linguagens de baixo nível**, pois elas encontram-se ao nível do *hardware* e são executadas directamente pela CPU obedecendo a instruções que contêm um código de operação e um ou mais endereços de memória. Estas linguagens são pouco práticas.

Linguagem simbólica ou de montagem

No final dos anos 40, desenvolveu-se uma nova linguagem semelhante à linguagem máquina mas com os códigos de operação substituídos por mnemónicas e com os endereços de memória correspondentes aos dados, substituídos por nomes simbólicos, permitindo o controlo dos dispositivos internos do computador através de comandos e variáveis. A esta linguagem simbólica de montagem chamou-se **Linguagem** *Assembly* (tradução literal do inglês: «de montagem»).

Um programa em linguagem *Assembly* é convertido para linguagem máquina por um tradutor (ou programa de montagem) denominado *Assembler*.

Linguagens de alto nível

Foi já nos anos 50 que foram desenvolvidas as chamadas **linguagens de alto nível**, que permitem a programação através de comandos com palavras da língua inglesa.

Através desses comandos, possibilitou-se que fossem criadas estruturas de dados e descritos procedimentos lógicos para a solução de tarefas com mais facilidade. Dentro das linguagens de alto nível podemos encontrar diferentes paradigmas de programação.

Do acima exposto resulta que qualquer linguagem de programação deve estar situada entre dois extremos: o da linguagem natural do Homem (muito clara, porém lenta) e o da linguagem de máquina (muito rápida, porém complexa). Este é o conceito de nível de linguagem: alto nível para as linguagens mais próximas da linguagem humana; baixo nível para as linguagens mais semelhantes à linguagem de máquina.

Paradigmas de programação

O que é um paradigma de programação?

É o modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas características comuns. Por detrás de uma linguagem de programação está sempre um **paradigma** representado. Dentro da nossa abordagem consideraremos os paradigmas que se seguem.

Paradigma imperativo - características

- Programas centrados no conceito de um estado (modelado por variáveis) e acções (comandos) que manipulam o estado.
- Paradigma também denominado procedural, por incluir subrotinas ou procedimentos como mecanismo de estruturação.

Primeiro paradigma a surgir e ainda é o dominante.

Fazem parte deste paradigma, entre outras, as seguintes linguagens de programação:

- *Pascal*
- *Algol*
- *Fortran*
- *C*
- *Basic*
- *Modula*

Paradigma funcional - características

- Programas são funções que descrevem uma relação explícita e precisa.
- Estilo declarativo: não há o conceito de estado nem comandos como atribuição.
- Aplicação: prototipação em geral e inteligência artificial

Exemplo de linguagens deste paradigma:

- Lisp (*List Processing Language*) – criado nos finais dos anos 50 por J. McCarthy do MIT
- Logo – criado nos anos 60, inspirado no LISP e na teoria da construção da inteligência de Jean Piaget.
- Scheme – um dialecto do *LISP* criado em 1975.
- ML (Meta Language) – criado por R. Milner em 1978.
- Miranda, uma linguagem puramente funcional, desenvolvida em 1985 por D.A.Turner.

Paradigma *lógico*

Criado nos anos 70, em Marseille, por Alain Colmerauer, o PROLOG implementa a lógica de predicados numa linguagem computacional. A lógica de predicados permite a construção de uma representação do conhecimento baseada em regras, facilitando o desenvolvimento de sistemas de Inteligência Artificial, tal como os chamados Sistemas Especialistas.

Exemplo:

- *PROLOG (PROgramming in LOGic)*
- CLP

Paradigma orientado a objectos

O ancestral comum de praticamente todas as Linguagens Orientadas a Objectos (LOO) é *Simula*, desenvolvida nos anos 60 por Dahl, Myhraug e Nygard. Hoje, já são mais de 2000 as linguagens de programação de alto nível. Dessas, cerca de um quarto é **orientado a objectos** ou **baseado em objectos.**

Exemplos de Linguagens Orientadas para Objectos (LOO):

- *SmallTalk*
- *Eiffel*
- *C++*
- *Object Pascal*
- *Java*

A programação orientada para objectos assenta na implementação em computador de modelos de objectos do mundo real. Assim, um programa é uma agregação de objectos. A programação orientada para objectos é, talvez, o paradigma de maior expressão no mercado actualmente.

Síntese

Níveis de abstracção num computador

Na utilização do computador é possível identificar vários níveis de abstracção, sendo os três mais relevantes os seguintes:

- Nível da Linguagem máquina (em binário): instruções e variáveis totalmente codificadas em binário, sendo a codificação sempre associada a um dado processador.
- Nível da Linguagem *Assembly*: equivalente ao nível anterior, mas em vez da notação puramente binária, usa mnemónicas para especificar as operações pretendidas, bem como os valores ou localizações dos operandos.
- Nível das Linguagens HLL (*High Level Languages*), como o *Java, C++, Pascal, Fortran, Delphi,* etc...

Hoje, aprender a programar significa centrar-se mais em linguagens de alto nível, é aí onde se desenvolvem os aplicativos, cujo processo de programação compreende os seguintes passos:

1º **Definição e análise do problema** – um programa deve ser projectado de modo a permitir resolver os problemas a que se propõe, de um modo simples e eficaz. Por isso é que nesta fase é imperioso proceder-se à análise das necessidades, em termos de entrada (*Input*), a introduzir, de saídas (*Output*) de modo a obter, mapas, relatórios, para impressora, monitor ou ficheiros.

Desenho do algoritmo – definição dos procedimentos necessários à resolução do problema, desde a validação dos dados (de modo a corrigir possíveis erros de operação) até à definição dos procedimentos de leitura e escrita em periféricos.

Codificação e teste do programa – após a definição do problema e da escolha da linguagem de programação a utilizar, procede-se então à elaboração do respectivo código, e ao teste e correcção de erros.

Elaboração da documentação – esta fase encerra a elaboração do programa permitindo o conhecimento geral do programa, dos meios humanos e físicos (*hardware*) necessários, e ainda a descrição dos procedimentos de operação e manutenção do programa.

Tradutores

Para que um computador possa «entender» um programa escrito numa linguagem de alto nível, torna-se necessário um meio de tradução entre a linguagem utilizada no programa e a linguagem máquina. Este meio pode ser de dois tipos: *compilador* e *interpretador*.

Compilador

Os **compiladores** são programas de *software* especiais que traduzem e convertem os programas escritos em linguagem de alto nível noutro tipo de programas. Como resultado da tradução, é criado um novo programa correspondente ao inicial, mas escrito em linguagem máquina, portanto **executável**. Estes programas executáveis (*.exe) podem ser realizados fora dos ambientes de programação (C e *Pascal*, por exemplo).

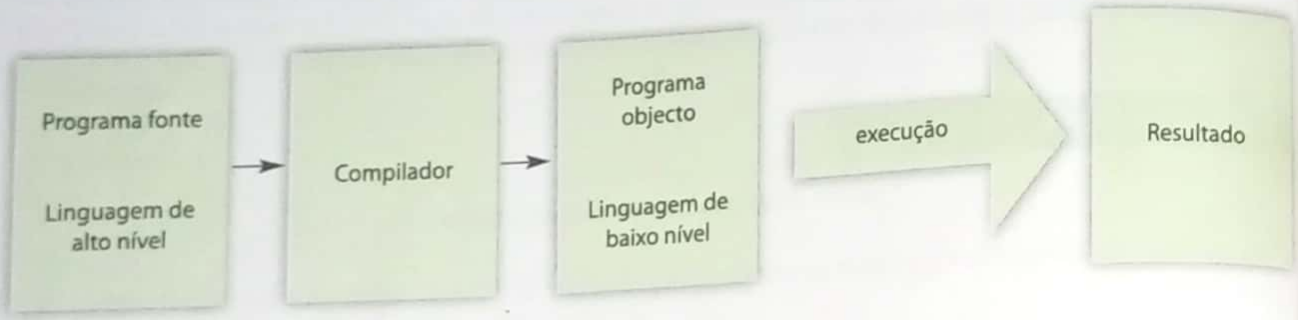


Fig. 2.5 Processo de compilação

Interpretador

Os **interpretadores** são programas de *software* especiais que traduzem e convertem os programas escritos em linguagens de alto nível em código capaz de ser interpretado e executado pelo processador. Os interpretadores fazem a interpretação de cada instrução do programa fonte executando-a dentro de um ambiente de programação. No entanto, a interpretação do código implica a presença do interpretador sempre que o programa é corrido (interpretado).

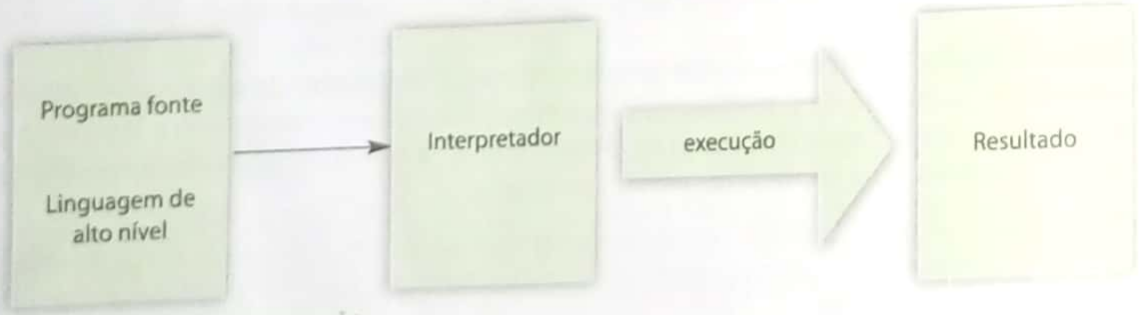


Fig. 2.6 Processo de interpretação

Linguagem *Pascal*

A **linguagem de programação** *Pascal* foi criada no início da década de 70 pelo Prof. Niklaus Wirth da Universidade de Zurique, na Suíça. O objectivo era desenvolver uma linguagem de programação disciplinada para ensinar a programação estruturada. Portanto, o *Pascal* foi desenvolvido para ser uma ferramenta educacional. Foi baptizada pelo seu idealizador em homenagem ao grande matemático Blaise Pascal, filósofo e matemático francês que viveu entre 1623 e 1662, e inventor de uma das primeiras máquinas lógicas conhecidas.

Apesar do seu propósito inicial, o *Pascal* começou a ser utilizado por programadores de outras linguagens, tornando-se, para surpresa do próprio Niklaus, um produto comercial. Contudo, somente no final de 1983 é que a empresa americana *Borland International* lançou o *Turbo Pascal*.

Turbo Pascal

É um Ambiente Integrado de Desenvolvimento (IDE - *Integrated Development Environment*), consistindo num conjunto de ferramentas de desenvolvimento integrado. Entre as ferramentas que compõem o *Turbo Pascal* temos:

- Editor de Código-Fonte
- Compilador
- Link-Editor
- Depurador
- Ajuda On-Line da Linguagem e do próprio IDE

Formato de um Programa Pascal

O *Pascal* é uma linguagem de programação altamente estruturada que possui uma rigidez definida, embora a sua estrutura de programa seja flexível. Cada secção ou parte de um programa em *Pascal* deve aparecer numa sequência apropriada e ser sistematicamente correcta, senão ocorrerá um erro.

Por outro lado, no *Pascal* não há regras específicas para o uso de espaço, linhas quebradas, requisições e assim os comandos podem ser escritos no formato livre em quase todos os estilos que o programador deseja utilizar.

Um programa escrito em Pascal tem o seguinte formato:

PROGRAM <identificador>;
<bloco>.

O <bloco>, por sua vez, está dividido em seis áreas, onde somente a última é obrigatória e devem obedecer à sequência abaixo:

- Área de declaração de uso de unidades.
- Área de declaração de constantes.
- Área de declaração de tipos.
- Área de declaração de variáveis.
- Área de declaração de procedimentos e funções.
- Área de comandos.

Declaração de uso de unidades

Um programa *Pascal* pode fazer uso de algumas unidades padrão que estão disponíveis no Sistema Turbo, tais como:

CRT, DOS, PRINTE, GRAPH, etc.

A área de declaração de uso de unidades possui o seguinte formato:

USES <unidade> , ... , <unidade> ;

Exemplo:

USES CRT, PRINTER;

Declaração de constantes

Serve para associarmos nomes às constantes utilizadas no programa. Possui o seguinte formato:

CONST <identificador>=<valor>;...;<identificador>=<valor>;

Exemplo:

```
CONST
VAZIO = ' ';
  PI = 3.1416;
 MAX = 10;
  OK = TRUE;
```

Declaração de tipos

Serve para definirmos novos tipos e estruturas de dados. A linguagem *Pascal* suporta os seguintes tipos de dados:

Simples

- **INTEGER** – envolve os números inteiros, que por sua vez também podem ser *SHORTINT, BYTE, WORD* e *LONGINT.*
- **REAL** – abrange os números reais, que podem ser *SINGLE, DOUBLE, EXTENDED* e *COMP.*
- **CHAR** – representa um único caracter, escrito entre apóstrofos (').
- **BOOLEAN** – representa um valor lógico. Utiliza apenas duas constantes lógicas: *TRUE* (verdadeiro) e *FALSE* (falso).
- **STRING** (dados estruturados) – formado por um conjunto de elementos do tipo *CHAR*. Fazem parte ainda *ARRAY, RECORD, FILE, SET* e *TEXT*.

Declaração de variáveis

Da Matemática sabemos que uma variável é a representação simbólica dos elementos de um certo conjunto. Nos programas destinados a resolver um problema no computador, a cada variável corresponde uma posição de memória, cujo conteúdo pode variar ao longo do tempo durante a execução de um programa. Embora a variável possa assumir diferentes valores, ela **só pode armazenar um valor a cada instante**.

Assim, todas as variáveis que serão utilizadas num programa devem ser declaradas dentro do mesmo. A declaração de uma variável tem como finalidade:

- Especificar o tipo de dado que poderá ser armazenado na variável.
- Alocar um espaço na memória onde possa ser armazenado o conteúdo da variável.
- Dar um nome (identificador) à variável.

Possui o seguinte formato:

```
VAR
<lista-de-identificadores> : <tipo>;
...
<lista-de-identificadores> : <tipo>;
```

Onde:
<lista-de-identificadores> é uma lista de variáveis de um mesmo tipo, separadas por vírgula.
<tipo> é o nome de um dos tipos predefinidos ou criados pelo programador.

Exemplo:

```
VAR
  X, Y, Z: REAL;
  I, J: INTEGER;
  ACHOU: BOOLEAN;
  LETRA: CHAR;
  PALAVRA, FRASE: STRING;
```

Declaração de procedimentos e funções

Nesta área são definidos os procedimentos e as funções utilizados pelo programa.

Área de comandos

É nesta área onde é inserido o algoritmo do programa. Os comandos são separados entre si pelo delimitador ponto e vírgula. A forma geral é:

```
BEGIN
<comando> ;
... ;
<comando>
END
```

Comentários

Um **comentário** na linguagem *Pascal* é idêntico à nossa linguagem algorítmica, pois utilizamos chavetas:

{ comentário }

Comandos básicos

Atribuição

A operação de **atribuição** permite que se forneça um valor a uma certa variável. Este comando tem a seguinte forma:

```
<identificador> := <expressão>
```

Exemplos:

A := 2 NOME := 'Yuri' A := B + C

Entrada

Um comando de **entrada** serve para que o programa solicite dados no momento em que o mesmo está a ser executado. Em geral a unidade de entrada é o teclado. Este comando tem a seguinte forma:

```
READ (<identificador-1>,...,<identificador-n>)
ou
READLN (<identificador-1>,...,<identificador-n>)
```

Com o *READ* o cursor **permanece** na mesma linha após a execução do comando; com o *READLN* o **cursor muda** para a próxima linha.

Saída

Um comando de **saída** serve para que o programa mostre ao usuário os resultados desejados. A unidade de saída padrão é o monitor de vídeo, podendo ser também a impressora.

Considerando a unidade de saída padrão o monitor de vídeo, o comando seria:

```
WRITE (<expressão-1>,...,<expressão-n>) ...
ou ...
WRITELN (<expressão-1>,...,<expressão-n>)
```

Exemplo:

```
A:=1;        B:=2;
writeln ('Soma de ',A,' e ',B,' = ',A+B)
```

Estruturas de decisão

Tal como vimos antes, as estruturas de decisão são utilizadas para **tomar** uma **decisão baseada** no **resultado da avaliação** de uma condição de controlo e selecciona uma ou mais acções possíveis (comandos) para serem executados pelo computador. No *Pascal*, existem três tipos de estrutura de decisão:

IF-THEN-ELSE

O comando *IF* é equivalente ao comando *SE* da linguagem algorítmica que já vimos, e deve ser utilizada da seguinte forma:

```
IF <condição> THEN
<comando1>
[ ELSE
<comando2> ]
```

Neste caso, se a <condição> resultar no valor *TRUE*, será executado o <comando1>; caso contrário, será executado o <comando2>.

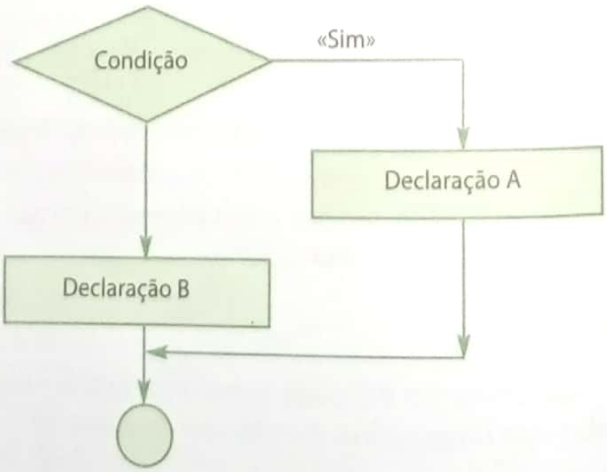
A <condição> deve ser uma expressão lógica. O <comando1> e <comando2> podem ser um comando simples ou um comando composto. Um comando composto é formado por dois ou mais comandos, separados por ponto e vírgula e delimitados por *BEGIN* e *END*.

Neste pequeno exemplo, o comando *WRITELN* só será executado se a condição N>0 for verdadeira, caso contrário não vai acontecer nada.

Exemplo

IF_THEN

```
Program IF_THEN;
    {Ler um número inteiro e exibi-lo se for positivo}
VAR
  N : integer;
BEGIN
    READLN(N);
   IF N > 0 THEN
     WRITELN(N)
END.
```



Agora, neste caso, a mensagem que será exibida dependerá do resultado da expressão lógica N>0. Tente interpretar o que vai acontecer na execução deste pequeno programa.

Exemplo

IF-THEN-ELSE

```
Program IF_THEN_ELSE;
      {Lê um número e determina se é maior que zero ou não}
VAR
  N : integer;
BEGIN
   READLN(N);
   IF N > 0 THEN
         WRITELN (N,' é maior que zero' )
   ELSE
        WRITELN (N,' não é maior que zero')
END.
```

CASE-OF

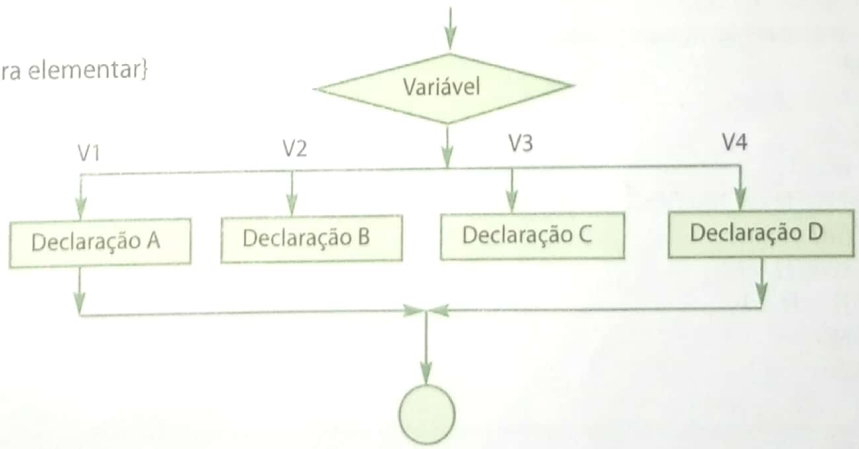
Utilizada quando se deseja executar um entre vários comandos dependendo do resultado de uma expressão. A estrutura de decisão múltipla do *Pascal* é o CASE, e obedece à seguinte sintaxe:

```
CASE <expressão> OF
<lista-de-constantes-1> : <comando-1>;
<lista-de-constantes-2> : <comando-2>;
...
[ELSE <comando-n>]
END
```

Exemplo

CASE-OF

```
Program CASE_OF;
     {Simulador de uma calculadora elementar}
USES
   CRT;
VAR
   X,Y : integer;
   OP : char;
BEGIN
  CLRSCR;
    WRITE('Digite os operandos: ');
    READLN(X,Y);
    WRITE('Digite o operador: ');
    READLN(OP);
 CASE OP OF
        '+' : WRITELN(X + Y);
        '-' : WRITELN(X - Y);
        '*','x','X' : WRITELN(X * Y);
```



```
    '/' : WRITELN(X div Y);
  ELSE WRITELN('operação inválida');
  END {CASE};
READKEY;
END.
```

Neste exemplo, a mensagem que será exibida dependerá do conteúdo da variável OP.

Estruturas de repetição

Tal como vimos antes, a estrutura de repetição permite que uma sequência de comandos seja executada repetidamente enquanto uma determinada condição seja satisfeita.

No *Pascal*, existem três tipos de estrutura de repetição: com teste no início (*WHILE*), com teste no final (*REPEAT*) e automático (*FOR*).

A estrutura de controlo *WHILE* permite que um comando simples ou composto seja executado repetidamente, enquanto uma condição de controlo seja Verdadeira. É equivalente ao comando Enquanto da linguagem algorítmica. A forma geral do *WHILE* é:

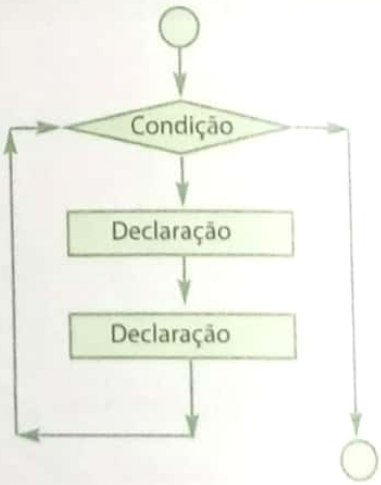
WHILE <condição> *DO* <comando>

Como o teste da <condição> é realizado no início do laço, o <comando> será executado zero ou mais vezes, dependendo da avaliação da <condição>.

Nesta estrutura, enquanto a condição permanecer verdadeira, são executadas as declarações A e B, sendo que quando não for satisfeita sai do ciclo e continua o programa. De notar que, se a condição nunca for satisfeita, nunca serão executadas as declarações do ciclo.

Exemplo

```
WHILE
 Program WHILE;
    {escrever os números inteiros de 1 a 100}
VAR
   N : integer;
BEGIN
   N := 1;
 WHILE N <= 100 DO
BEGIN
   WRITELN(N);
   N := N + 1
 END
END.
```



Neste exemplo, o comando *WRITE* será executado repetidas vezes enquanto a variável *N* possuir um valor igual ou inferior a 100. Toda a estrutura de repetição deve possuir uma condição de parada.

Existem basicamente dois tipos de controlo de estruturas de repetição: controlo por contador e controlo por entrada (*flag*).

Controlo por contador

Um **contador** é uma variável do **tipo inteiro** que deve receber inicialmente (antes do laço) o valor 1 e dentro do laço (no final) deve ser incrementada em 1, ou seja, adicionar-se o valor 1 ao conteúdo da própria variável. A condição para interromper o laço será: contador ≤ número de repetições.

```
Program LACO_CONTADO;
VAR
    NUM, SOMA, CONT : integer;
BEGIN
     SOMA := 0;
     CONT := 1;
 WHILE CONT <= 100 DO
BEGIN
     WRITE('Digite um número inteiro: ');
      READLN(NUM);
           SOMA := SOMA + NUM;
  END
   WRITELN('A soma é ',SOMA);
END.
```

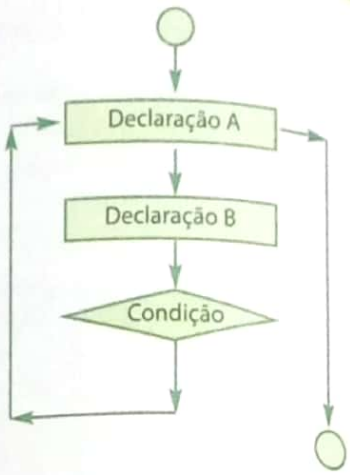
Controlo por entrada (*flag*)

Quando não conhecemos o número de repetições e este for determinado por um valor que será lido (a que chamamos *flag*), devemos utilizar um controlo de repetições por entrada, também conhecido como **controlo por sentinela**.

Por exemplo, consideremos o programa a seguir que lê um conjunto de números inteiros e exibe o valor médio dos mesmos, cuja condição de saída do laço será a leitura do valor 0 (*flag*).

```
Flag
Program LACO_COM_FLAG;
VAR
    NUM,CONT,SOMA,MÉDIA : integer;
BEGIN
     SOMA := 0;
     CONT := 0;
  WHILE TRUE DO {loop infinito}
BEGIN
   WRITE('Digite um número inteiro (0 para encerrar): ');
  READLN(NUM);
        IF NUM = 0 THEN
        break
   SOMA := SOMA + NUM;
   CONT := CONT + 1;
 END;
        MÉDIA := SOMA div CONT;
        WRITELN('Média = ',MÉDIA);
END.
```

Repetição com teste no final (*repeat-until*)



A estrutura de controlo REPEAT permite que um comando simples ou composto seja executado repetidamente até que uma condição de controlo seja FALSA. A forma geral do REPEAT é:

```
REPEAT <comando> UNTIL <condição>
```

A <condição> deve ser uma expressão lógica.

Nesta estrutura, as declarações A e B serão executadas até que a condição seja satisfeita. Neste caso, contrariamente ao anterior, as declarações serão executadas pelo menos uma vez.

```
Program REPEAT;
      {escrever os núm. inteiros de 1 a 100}
VAR
   N : Integer;
BEGIN
   N := 1;
 REPEAT
      WRITELN(N);
      N := N + 1
 UNTIL N > 100
END.
```

Este programa é equivalente ao do WHILE, onde o comando WRITELN é executado repetidas vezes até que a variável *N* possua um valor superior a 100.

A estrutura *REPEAT* também é bastante utilizada para se repetir um programa diversas vezes, até que o usuário deseje sair do mesmo. Observe o programa seguinte:

```
Repeat
Program REPEAT_2;
     {calcula a média de 2 números dados repetidas vezes}
 USES
     CRT;
VAR
     N1, N2, MÉDIA : real;
     RESP : char;
BEGIN
     CLRSCR;
 REPEAT
      WRITE('Digite os dois números: ');
      READLN (N1,N2);
      MÉDIA := (N1+N2)/2;
      WRITELN (MEDIA);
      WRITE ('Deseja repetir o programa (s/n)? ');
      RESP := READKEY;
 UNTIL (RESP='N') OR (RESP='n')
END.
```

Repetição automática (*FOR*)

Esta estrutura permite que um comando simples ou composto seja repetido um número específico de vezes. A sua forma geral é:

```
FOR <var> := <vi> TO <vf> DO <comando>
```

Onde:

<var> é uma variável de controlo, do tipo inteira, que assumirá o valor inicial <vi> e será incrementada do valor 1 após cada repetição do laço. A repetição será finalizada quando o conteúdo de <var> for superior ao valor final <vf>.

Também nesta estrutura, se tivermos de executar uma sequência de declarações, podemos utilizar os delimitadores, início e fim.

```
FOR
Program FOR;
    {escreve os números inteiros de 1 a 100}
VAR
   N : integer;
BEGIN
   For N := 1 TO 100 DO
    WRITELN(N)
END.
```

A estrutura de repetição *FOR* é especialmente indicada para quando o número de repetições é previamente conhecido. Caso contrário, devemos utilizar o *WHILE* ou o *REPEAT*, dependendo do caso.

```
FOR_DOWNTO
Program FOR_DOWNTO;
  {escreve os números inteiros de 100 a 1}
VAR
   N : integer;
BEGIN
   FOR N := 100 DOWNTO 1 DO
   WRITELN(N)
END.
```

A linguagem de programação *Pascal*, que acabámos de introduzir, é muito extensa e não cabe no âmbito deste livro escolar. Porém, julgamos que a introdução aqui feita poderá servir, por um lado, como uma motivação para se interessar mais pela programação que, na verdade, é actualmente uma área muito activa e apaixonante para os jovens, pois traz consigo a automatização e a robótica e, por outro lado, esperamos que tenha ajudado a entender um pouco mais a complexidade e ao mesmo tempo a beleza do mundo da informática, bem como da sua relação com outras ciências como a matemática, mas também as aplicadas.

De certeza que ao longo do seu percurso encontrará pessoas ligadas à informática e até programadores que vão dizer que programam há anos mas que nunca precisaram de nenhuma teoria de algoritmos! Não é verdade. Sempre que tiveram que desenvolver um programa ou uma aplicação, fizeram-no partindo, – consciente ou inconscientemente – de algum algoritmo.

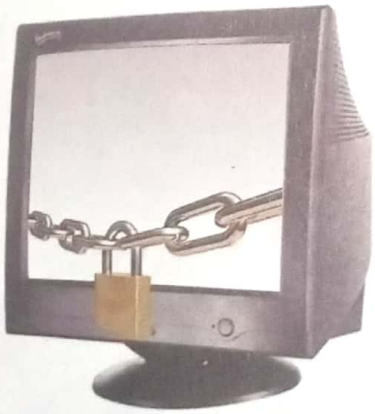
1. Defina, por palavras suas, os seguintes termos:
   a) Programa
   b) Linguagem de programação
   c) Tradutor
2. Qual é a diferença entre linguagem de baixo nível e linguagem de alto nível?
3. Diferencie um algoritmo de um programa.
4. Um mês antes das eleições municipais, um determinado partido político encomendou uma pesquisa de opinião sobre as intenções de voto dos eleitores. Foram entrevistadas 50 pessoas que indicaram as suas intenções de acordo com as seguintes opções: A candidato A, B candidato B, C indecisos. Desenvolva um algoritmo que faça a leitura das intenções de voto dessas 50 pessoas e que informe no fim a percentagem de intenções para cada uma das opções existentes (candidatos A e B, e indecisos).
5. Escreva o tipo de dado ideal para se representar as seguintes informações:
   a) O número da conta bancária
   b) A altura de uma pessoa em metros
   c) A placa de matrícula de um veículo
   d) O número de filhos de uma pessoa
   e) A população de um país
   f) A cor de um objecto
6. Qual é o formato básico de um programa *Pascal*?
7. Quais são as áreas que podem existir num bloco do *Pascal*? Qual delas é obrigatória?
8. Qual é a finalidade de declararmos uma variável?
9. Qual é a finalidade de um comentário dentro de um programa? Como deve ser escrito em *Pascal*?
10. Escreva um programa para calcular a área de um triângulo, sendo dados a sua base e a sua altura.
11. Escreva um programa para calcular e exibir o perímetro de uma circunferência, sendo dado o valor do seu raio.
12. Escreva um programa para calcular e exibir o valor de xy, sendo dados a base (x) e o expoente (y).
13. Escreva os comandos necessários para ler:
    a) As 3 notas de um aluno.
    b) O nome, o peso e a altura de uma pessoa.
14. Qual é a diferença entre os comandos *WRITE* e *WRITELN*?
15. Como podemos direccionar a saída de um programa para a impressora? Dê exemplos.
16. Qual é a utilidade da estrutura de decisão?
17. Quais são os comandos de decisão existentes no *Pascal*?
18. Quais são as estruturas de repetição existentes no *Pascal*?
19. Qual é a principal diferença entre o *WHILE-DO* e o *REPEAT-UNTIL*?
20. Em que situações é mais indicado o uso da estrutura *FOR*?

Introdução à segurança informática

Com o advento dos sistemas de informação e da caracterização da sociedade actual como uma aldeia global, isto é, em rede, a generalidade das organizações tornou-se fortemente dependente dos seus sistemas informáticos para gerir as suas actividades e suportar a tomada de decisões. Quer dizer, a segurança de sistemas de informação na vida das organizações hoje reveste-se de uma importância igual à concepção e desenvolvimento dos próprios sistemas de informação. O princípio reside no facto de que pior que não ter um bom sistema de informação é tê-lo e perdê-lo no momento em que os utilizadores, os procedimentos e os compromissos estão dependentes do sistema informático.

Por isso, não é de admirar que os responsáveis pelos sistemas informáticos das organizações se preocupem cada vez mais com os efeitos desastrosos que teria uma ameaça ou um ataque que comprometesse o funcionamento desses sistemas e a informação que possuem.

O que é segurança informática?



É a **segurança de sistemas** onde computadores e redes informáticas são elementos centrais, componentes importantes ou estão envolvidos de qualquer modo.

Objectivos da segurança informática

O **sistema de informação** define-se geralmente como o conjunto dos dados e dos recursos materiais e de *software* da organização que permite armazená-los ou fazê-los circular. O sistema de informação representa um património essencial da organização, que convém proteger.

A segurança informática, geralmente, consiste em garantir que os recursos materiais ou *software* de uma organização sejam utilizados unicamente no âmbito previsto.

A segurança informática visa geralmente seis objectivos principais:

1. A **integridade**, ou seja, garantir que os dados são efectivamente os que se crê serem. Verificar a integridade dos dados consiste em determinar se os dados não foram alterados durante a comunicação (de maneira fortuita ou intencional).

2. A **confidencialidade**, consiste em assegurar que só as pessoas autorizadas têm acesso aos recursos partilhados, isto é, tornar a informação ininteligível para terceiros além dos actores da transação.
3. A **disponibilidade**, permite manter o bom funcionamento do sistema de informação, ou seja, garantir que um determinado recurso ou serviço esteja sempre acessível.
4. O **registo**, que permite o arquivamento de determinadas operações, para análise *a posteriori*, de modo a saber quem fez o quê e quando, especialmente quando se detecta alguma anomalia no funcionamento de um certo serviço ou sistema. Por exemplo, numa época em que o cartão de crédito é cada vez mais utilizado em negócios electrónicos, é importante haver um registo de todas as transacções de forma que se alguém não autorizado fizer uso indevido de um cartão, a acção seja facilmente detectada.
5. A **não-repudiação**, permite garantir que uma transacção não pode ser negada. O acto de não-repúdio pode ser realizado em três fases distintas, nomeadamente:
   - Na criação, quando o autor de uma mensagem ou de um documento não pode negar a sua autoria e o seu envio, por exemplo, se o documento estiver assinado.
   - Na submissão, quando o autor de uma mensagem ou de um documento obtém uma prova do seu envio como, por exemplo, no correio registado.
   - Na recepção, quando o destinatário de uma mensagem não pode negar que a recebeu como, por exemplo, no correio registado com aviso de recepção.
6. A **autenticação**, consiste em assegurar que só as pessoas autorizadas têm acesso aos recursos, isto é, a autenticação consiste em garantir a identidade de um utilizador, ou seja, garantir a cada um dos correspondentes que o seu parceiro é efectivamente aquele que crê ser. Isto pode ser conseguido através da utilização de:
   - Segredos entre os participantes, como senhas ou combinações de *username/password*.
   - Dispositivos únicos como *tokens* de segurança, *smartcards* e cartões de «batalha naval».

Um *token* de segurança, também por vezes denominado *token de hardware*, *token* de autenticação ou *token* criptográfico, é um pequeno dispositivo físico que um utilizador transporta de modo a ter autorização de acesso a um determinado serviço como, por exemplo, uma rede informática. Um *token* pode armazenar uma assinatura digital, dados biométricos como uma impressão digital ou até incorporar um pequeno teclado para introdução do número de identificação pessoal, mais conhecido por *PIN* (*personal identification number*).

Fig. 2.7 *Token*

Um *smartcard* é um pequeno cartão de plástico com um microprocessador (*chip*) incorporado de modo a ter capacidade de armazenamento e memória. É cada vez maior o número de cartões de débito e crédito com *smartcards* incorporados.

Fig. 2.8 *Smartcard*

Um cartão de «batalha naval» é um pequeno cartão que contém uma matriz de elementos que permitem a um utilizador a realização de determinadas operações, como, por exemplo, transacções electrónicas bancárias.

Fig. 2.9 Um cartão «batalha naval»

- Métodos biométricos como impressões digitais, *scan* da íris ou retina e análise da voz.

Síntese

Garantias de Segurança

As características que a informação deve possuir para garantir a sua segurança podem ser classificadas em confidencialidade, integridade, autenticação, autorização, registo e não-repúdio.

Ameaças à segurança

Uma ameaça (em inglês *threat*), no contexto informático, é qualquer acção efectuada com o intuito de comprometer a segurança do fluxo de informação entre duas entidades.

A **ameaça** representa o tipo de acção susceptível de prejudicar em absoluto, enquanto a **vulnerabilidade** (em inglês *vulnerability*, chamada às vezes falha ou brecha) representa o nível de exposição à ameaça num contexto específico. Por último, a **medida defensiva** é o conjunto das acções implementadas para a prevenção da ameaça.

Considere a situação mais simples, em que um emissor envia uma mensagem a um receptor com informação confidencial. Se um terceiro interveniente (atacante) pretender realizar um ataque à comunicação, a acção pode ser levada a cabo sobre:

- A mensagem
- O canal de comunicação
- A infra-estrutura do emissor ou do receptor

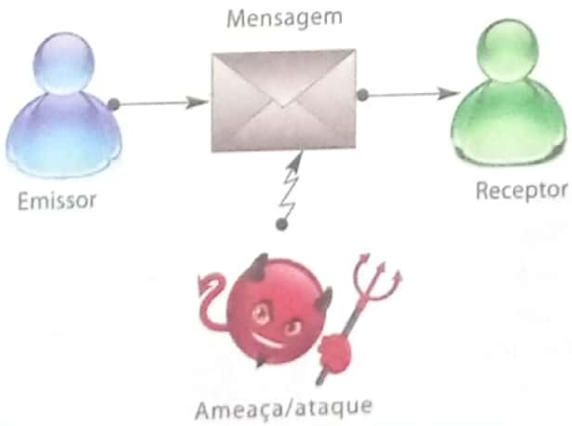


Fig. 2.10 Esquema de um ataque a uma mensagem

As medidas defensivas a aplicar não são unicamente soluções técnicas, mas igualmente medidas de formação e sensibilização para os utilizadores, bem como um conjunto de regras claramente definidas. A fim de poder proteger um sistema, é necessário identificar as potenciais ameaças, e por conseguinte

conhecer e prever a maneira de proceder do atacante. Em termos gerais, os ataques a que os fluxos de informação estão sujeitos podem ser classificados em seis categorias:

- Modificação
- Repetição
- Intercepção
- Disfarce
- Repúdio
- Negação de serviço (*denial of service*)

Modificação

Consiste na **alteração dos dados** de uma mensagem em trânsito. A alteração pode ocorrer de forma acidental ou maliciosa, quando, por exemplo, num negócio, um agente não autorizado altera uma encomenda de 10 unidades por parte de uma entidade para 100 unidades.

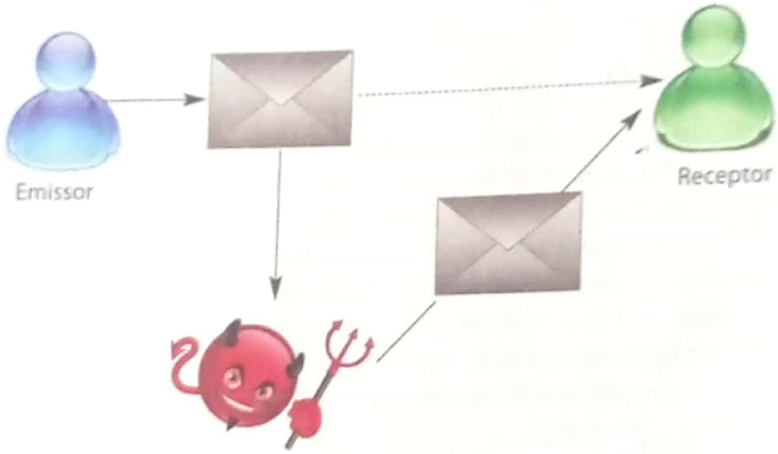


Fig. 2.11 Esquema de um ataque de modificação

Repetição

Acontece quando uma operação já realizada **é repetida**, sem autorização, de modo a obter o mesmo resultado. Considere, por exemplo, o caso em que um fornecedor utiliza sucessivamente os dados enviados por um comprador para efectuar o pagamento, obtendo de forma ilícita vários pagamentos adicionais.

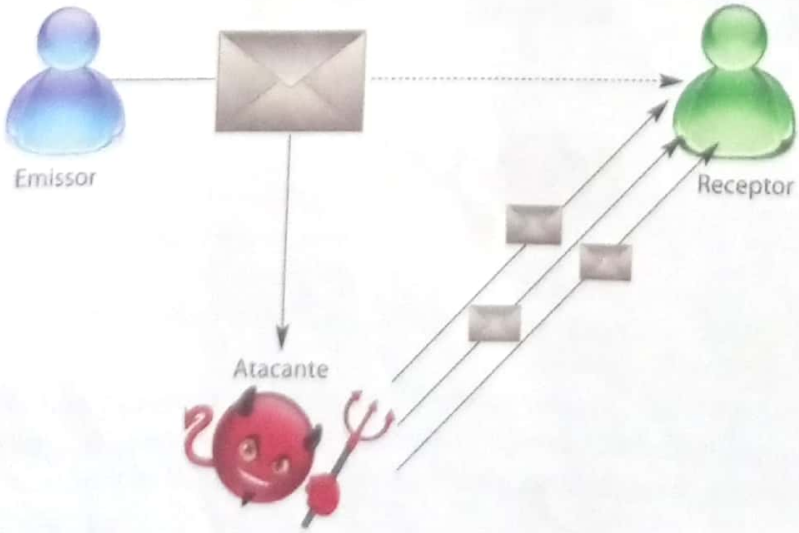


Fig. 2.12 Esquema de um ataque de repetição

Intercepção

Ocorre quando se verifica o **acesso não autorizado a uma mensagem**, que, contudo, não tem a possibilidade de alterar. Um exemplo desse ataque é a «escuta» da informação trocada entre duas sucursais de uma empresa por uma empresa concorrente.

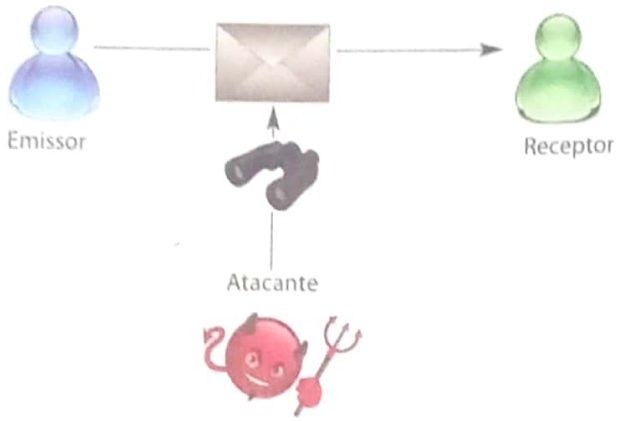


Fig. 2.13 Esquema de um ataque de intercepção

Disfarce

Consiste em apresentar uma **identidade falsa** perante um determinado interlocutor. Isto pode acontecer, por exemplo, quando um agente não autorizado pretende ocultar a sua própria identidade ou quando assume a identidade de outrem com o intuito de prejudicar o detentor daquela identidade.

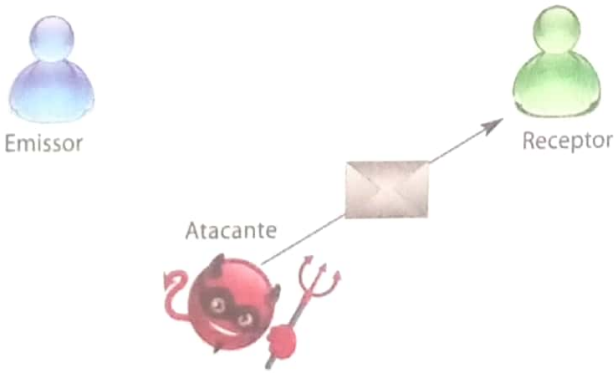


Fig. 2.14 Esquema de um ataque de disfarce

Repúdio

Consiste na **negação de participação** numa determinada comunicação ou operação quando de facto se fez parte dela. Acontece por exemplo quando um comprador nega a autoria e/ou o envio de uma mensagem com uma ordem de pagamento, ou quando um vendedor nega ter recebido o cancelamento de uma encomenda.

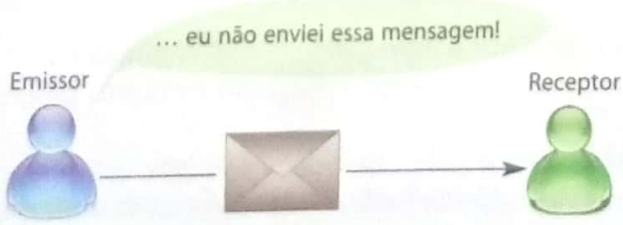


Fig. 2.15 Esquema de um ataque de repúdio

Negação de serviço (*denial of service*)

Consiste na **realização de um conjunto de acções** com o objectivo de dificultar o bom funcionamento de um sistema, por exemplo, saturando uma infra-estrutura de comunicação ou restringindo todas as mensagens para um destino específico.

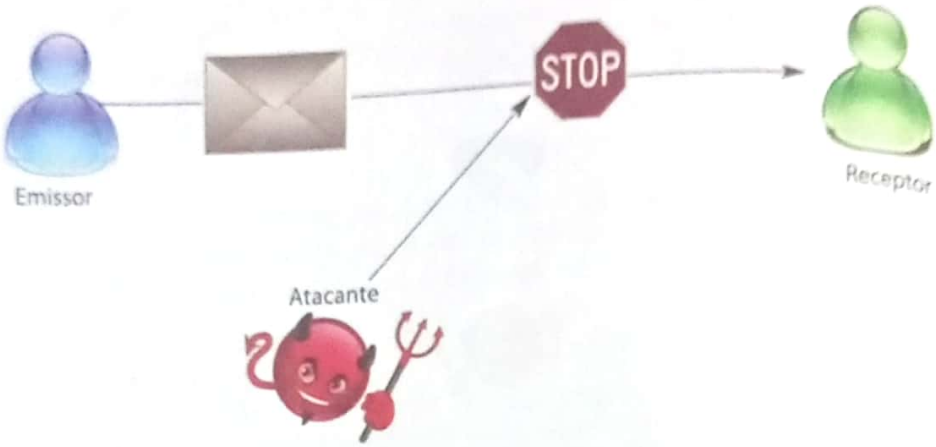


Fig. 2.16 Esquema de um ataque de negação de serviços (DoS)

Estas seis categorias podem ser agrupadas em duas classes de acordo com a metodologia utilizada no ataque: os **ataques activos** e os **ataques passivos**.

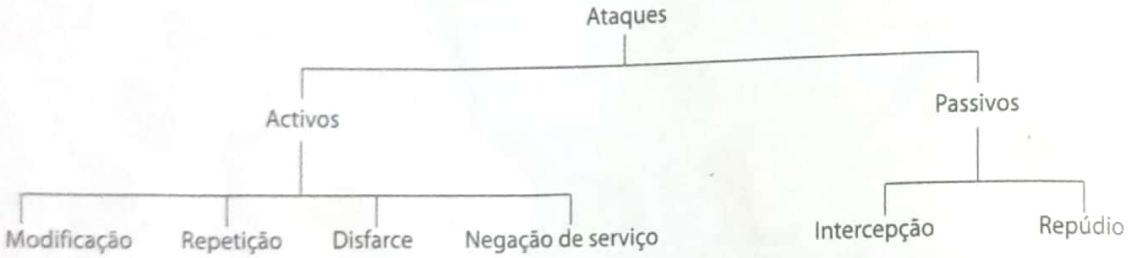


Fig. 2.17 Classificação dos ataques de acordo com a metodologia

Vírus e antivírus

Os **vírus informáticos** são uma das maiores condicionantes actuais para uma utilização fiável de computadores e sistemas de informação.

O que é um vírus informático?

Um **vírus informático** é um programa que é introduzido num computador sem o conhecimento do utilizador, com a intenção de ser multiplicado e afectar a operação de outros programas ou do próprio computador.

Os **vírus** são **programas caracterizados** pela capacidade que possuem de se copiarem a si próprios, «infectando» ficheiros e a memória do computador, realizando acções destrutivas sobre a informação e seus suportes, ao mesmo tempo que procuram esconder a sua existência e acções do utilizador.

O comportamento de um vírus de computador é semelhante ao de um vírus biológico, nomeadamente durante o processo de contaminação. Enquanto um vírus biológico utiliza as células vivas para se reproduzir, o vírus de computador serve-se de programas executáveis para se multiplicar por outras aplicações.

Que tipo de vírus existem?

Os primeiros vírus de computador surgiram na década de 80 e, desde então, esse número tem aumentado exponencialmente. O gráfico seguinte ilustra o número de vírus conhecidos no período entre 1990 e 2005.

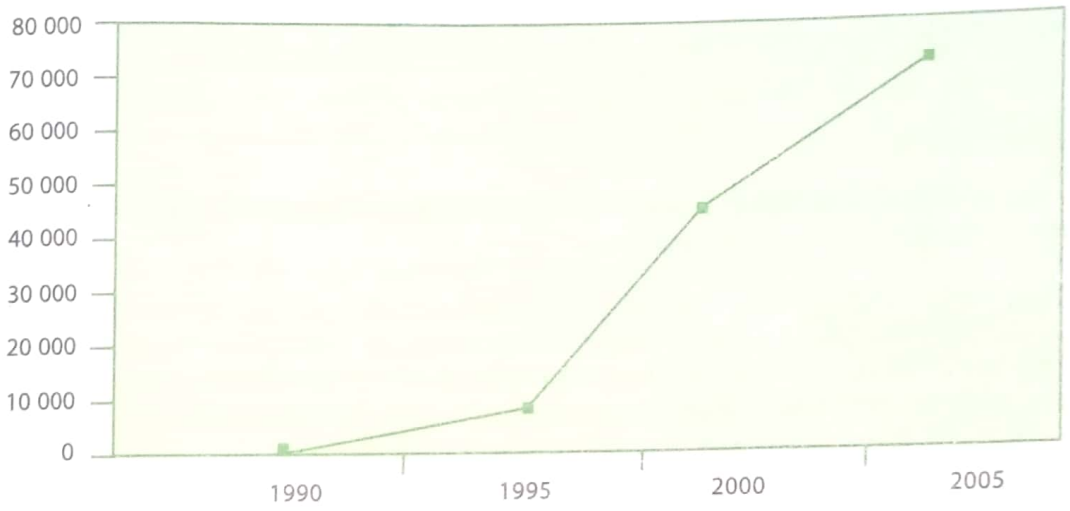


Fig. 2.18 N.º de vírus conhecidos no período entre 1990 – 2005

Hoje em dia, no vocabulário comum, o termo vírus de computador é muitas vezes estendido para referir também *worms* (verme), *trojan horses* (cavalos de Tróia) e outros tipos de programas maliciosos. Não vamos aqui fazer uma listagem exaustiva dos vírus existentes, mas na medida do possível vamos apresentar os principais grupos de vírus:

Boot sector infector vírus

São os vírus que se copiam a si próprios para o *boot* do disco, ou seja, para a zona do suporte de informação que é sempre acedida em primeiro lugar. Desta forma, o vírus é «carregado» para a memória antes do utilizador realizar qualquer outra operação. Uma vez em memória, **infecta todos os outros suportes de informação**.

Companion vírus

Não infectam ficheiros, mas **criam uma cópia de um ficheiro executável** (*.exe*), atribuíndo-lhe um nome idêntico embora com uma extensão *.com*. Como para o sistema operativo a prioridade de um ficheiro .com é superior, isto torna o programa original muito difícil de encontrar pelo computador.

Cluster vírus

Replicam-se automaticamente no sistema de ficheiros, «forçando» o computador a carregá-los em memória primeiro que o programa que se pretende activar, por forma a infectá-lo.

File infector vírus

São os vírus que **adicionam o seu próprio código de funcionamento aos programas** (ficheiros executáveis: *.exe, .com*). A execução do programa infectado espalha o vírus para outros ficheiros.

Macro vírus

Associam-se aos Macros da aplicação em uso, replicando-se através dos vários documentos, à medida que estes são abertos e\ou o *macro* que contém o vírus é executado.

Overwriting vírus

Destrói o programa original, copia o seu código para o interior deste. Estas acções tornam impossivel a recuperação do ficheiro original, embora se mantenha a capacidade de eliminação do vírus do sistema.

Polymorphic vírus

Vírus com **capacidade de mutação do seu código**, replicando-se pelo sistema com diferentes formas de encritação, dificultando acções de detecção.

Stealth vírus

Armazenam uma imagem do sistema antes da infecção deste, por forma a que quando os programas antivírus realizam uma verificação ao sistema, o próprio vírus apresenta uma imagem desta «pré-infecção», tentando dar a ideia que o sistema se encontra limpo.

Outras ameaças

Existem ainda outros programas maliciosos e ameaças que, apesar de não serem vírus, representam também perigo para a segurança do computador:

- ***Worm*** – programa que se propaga de computador para computador geralmente muito rapidamente, mas que não afecta outros programas. O principal prejuízo causado por um *worm* é a perda de capacidade de processamento do computador.
- ***Trojan horse*** – programa malicioso que tenta geralmente passar despercebido. O principal objectivo de um *trojan horse* é retirar informação privada de um computador, como, por exemplo, *passwords* de acesso a contas bancárias. Contudo, ao contrário de um vírus, um *trojan horse* não se multiplica.
- ***Logic bomb*** – programa que é inserido no sistema operativo ou em certas aplicações de modo a «explodir» quando um determinado evento ocorre, por exemplo, um dia da semana ou uma data em particular. Uma vez detonada, a bomba pode alterar ou apagar ficheiros ou até bloquear o computador.
- ***Spyware*** – programa que permanece escondido no sistema para monitorizar as acções do utilizador e recolher informação confidencial, como *passwords*, registos dos sistemas e outros ficheiros, enviando estes dados para uma entidade externa na *Internet*.
- ***Keylogger*** – pequena aplicação que tem como objectivo capturar tudo o que é digitado através do teclado, especialmente números de cartões de crédito e *passwords*.
- ***Hijacker*** – programa que altera a página inicial do *browser*, impedindo o utilizador de a modificar e de aceder a outras páginas, enquanto é apresentada publicidade.
- ***Adware*** – programa que exibe publicidade não desejada, frequentemente em janelas novas (*pop-ups*), e que podem redireccionar a página de pesquisa apenas para endereços comerciais.
- ***Phishing*** – tipo de fraude electrónica, através de uma mensagem de *e-mail*, que consiste em enganar uma pessoa induzindo-a a revelar informação sensível ou confidencial como *passwords* e números de cartões de crédito.

- **Spam** – são mensagens de *e-mail* anónimas, não solicitadas e enviadas maciçamente. Normalmente o *spam* tem fins publicitários, no entanto, o termo também engloba certas mensagens políticas, apelos à caridade, fraudes financeiras (*phishing*) e mensagens que servem para propagar vírus e outros programas maliciosos.
- **Hoax** – mensagem de *e-mail* que avisa as pessoas da falsa existência de perigosos vírus, sugerindo que a mensagem seja distribuída a todos os contactos.

Quais os efeitos de um vírus

Os principais motivos para a criação de um vírus por parte de um *hacker* (pirata informático) são **vandalismo, distribuição de mensagens políticas, ataque a produtos** de empresas específicas e **roubo** de *passwords* para proveito financeiro próprio. Quer dizer, como qualquer programa, o vírus apenas faz aquilo para que foi criado.

Os **efeitos negativos** de um vírus de computador podem variar desde a realização de operações mais ou menos inofensivas, como uma simples exibição no monitor de uma mensagem irritante, até à destruição de ficheiros do sistema ou formatação do disco rígido.

Como se propagam os vírus entre computadores

Ao infectarem ficheiros e ao serem carregados para a memória, o meio de transmissão actualmente mais relevante é a **comunicação de informação à distância**, nomeadamente nos computadores ligados em rede, seja a nível de uma rede corporativa ou de acesso à *internet*. Isto significa que, perante o grande desenvolvimento da *Internet*, esta tornou-se no principal veículo de transmissão de vírus, principalmente pela partilha de ficheiros e envio-recepção de correio electrónico. Embora actualmente com menor relevância, não é ainda de menosprezar a importância da transmissão dos vírus através de suportes de informação partilhados entre diferentes computadores, tais como *CDs* e *DVDs*.

Como proteger o computador/a informação

Existem algumas regras fundamentais no que diz respeito à protecção da informação quanto a vírus:

- Adquirir um programa antivírus que possibilite detecção, eliminação e protecção em relação a todos os tipos de vírus, mantendo-se residente em memória durante o funcionamento do computador.

Antivírus

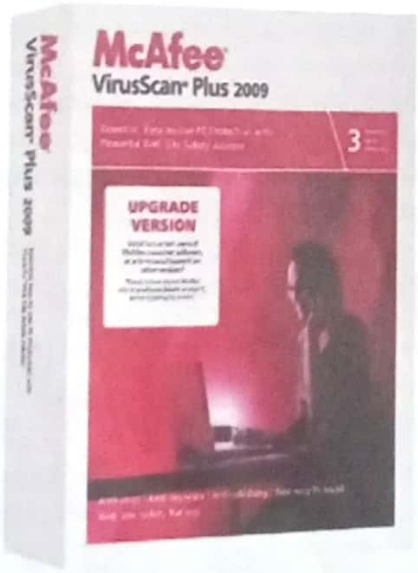
Um antivírus é um programa destinado a bloquear a acção dos vírus, analisando os dados que passam pela memória do computador ou o conteúdo dos ficheiros.

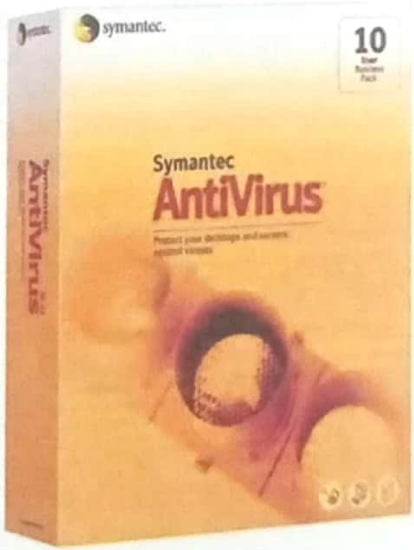
- Seleccionar um antivírus que lhe dê a possibilidade de assistência *on-line* ou até presencial, assegurando.
  - Rápida actualização dos ficheiros de dados sobre novos vírus.
  - Acesso regular a *upgrades*.
  - Assistência técnica para os casos difícieis.
- Instalar todas as actualizações (*updates*) disponibilizadas pelo fabricante do sistema operativo que utiliza diariamente. O *Windows*, por exemplo, fornece a possibilidade de transferir automaticamente as últimas actualizações de segurança da *Microsoft* pela aplicação *Automatic Updates*.
- Instalar um antivírus e manter a base de dados de vírus sempre actualizada, uma vez que todos os meses são descobertos cerca de 500 vírus novos.

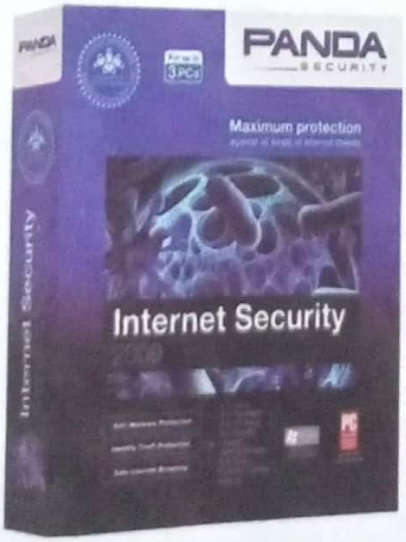
- Instalar (se possível) um sistema operativo alternativo como o *Linux*, uma vez que este sistema operativo possui características de segurança adicionais que o tornam praticamente inviolável.
- Utilizar um *browser* e um programa de *e-mail* alternativo, uma vez que os mais populares *browsers* e programas de *e-mail* da *Microsoft* – *Internet Explorer* e *Outlook*, respectivamente – são frequentemente atacados por *hackers*.
- Evitar descarregar programas ou ficheiros da *Internet* de fontes não fiáveis.
- Evitar abrir uma mensagem de *e-mail* quando o emissor é desconhecido.
- Evitar (se possível) enviar documentos por *e-mail* em formato *Word*, uma vez que estes documentos contêm *macros* e, portanto, são mais sujeitos a vírus de *macro*. Uma solução alternativa é enviar no formato RTF (*rich text format*).
- Instalar uma *firewall* e um sistema de detecção de intrusão.

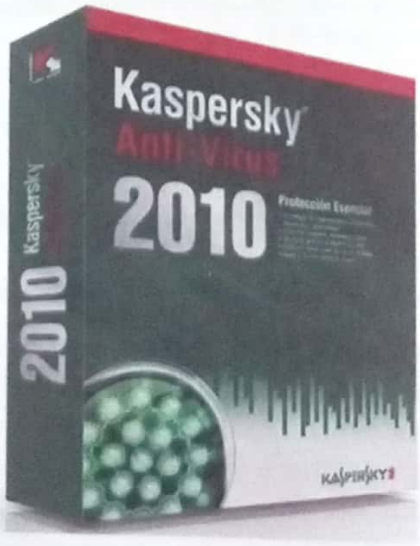
Alguns dos programas anti-vírus mais conceituados no mercado são:

- McAfee Vírus Scan
- Symantec Anti-Vírus
- Panda Anti-Vírus
- Kaspersky Anti-Vírus









Uma outra medida para protecção de computadores e informação é a **manutenção preventiva**, que consiste fundamentalmente em verificar, periodicamente, o adequado funcionamento dos vários tipos de *hardware* e *software*, bem como a própria organização da informação. O sistema operativo *windows* vem munido de algumas ferramentas de manutenção preventiva nos seus acessórios. Veja a figura ao lado:

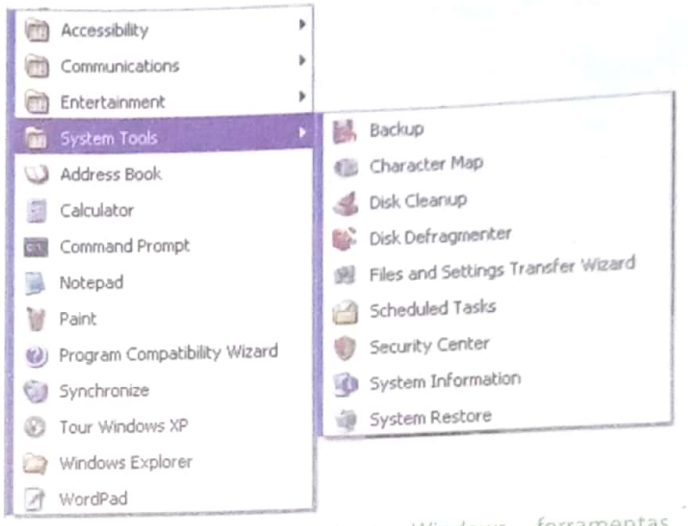


Fig. 2.19 Manutenção preventiva no *Windows* – ferramentas disponíveis

A **gravação periódica** da informação à medida que a vamos desenvolvendo é também uma das medidas aconselhadas para a protecção de dados. É verdade que existem alguns programas que já fazem isto automaticamente, em intervalos de 1 a 60 minutos, mas é bom não andarmos distraídos, pois a perda de informação pode-se transformar num pesadelo nos nossos projectos.

A utilização de *software* genuíno (original) nos nossos sistemas não só é uma forma de respeitar os direitos de autor do mesmo, como também permite a obtenção de suporte técnico e actualizações por parte do utilizador caso seja necessário, constituíndo-se assim numa outra forma de protecção.

As **medidas de segurança** a aplicar não devem ser unicamente soluções técnicas, mas igualmente medidas de formação e sensibilização para os utilizadores. Muitos dos danos sobre a informação ocorrem devido à falta de conhecimentos sobre o *software* utilizado e sobre o que fazer como medidas preventivas de segurança da informação.

Não menos importante para a protecção de equipamentos informáticos ou electrónicos é a utilização das chamadas UPS (*uninterruptable power supply*), que têm como função manter a corrente eléctrica durante uma falha de energia, protegendo desse modo a informação que está a ser processada. Associado a esta medida está a utilização de uma instalação eléctrica adequada, com as chamadas «tomadas amarelas» ou ainda a utilização de estabilizadores de tensão que minimizam picos de voltagem e descargas eléctricas.

O ambiente de trabalho, isto é, o sítio físico onde os computadores estão instalados contribui também para a segurança dos mesmos. Temperaturas extremas, fortes campos magnéticos, poeiras, infiltrações de água são alguns dos principais perigos para os equipamentos informáticos.

Protecção contra acesso por terceiros

Um dos elementos de segurança da informação mais utilizados pelas organizações até hoje e até por pessoas singulares é o uso de *passwords* (palavra-passe).

O que é uma password?

Uma *password* é um conjunto de caracteres, normalmente alfanuméricos (letras e números), usado para controlar o acesso de utilizadores não autorizados aos sistemas informáticos.

As *passwords* podem ser agrupadas em quatro categorias principais:

Password individual de sistema

Trata-se de uma *password* que normalmente **restringe** o acesso a um único computador. Estas passwords impossibilitam o funcionamento do próprio computador e ficam armazenadas no seu programa de configuração, o *Setup*.

Password de rede

Esta possibilita a um utilizador obter **acesso** a determinadas áreas de um computador central, o servidor, e à informação aí armazenada.

Password de programa

Este tipo de *password* condiciona o **acesso a um programa**, normalmente em relação ao software de aplicação.

Password de ficheiro

Protegem documentos, impedindo a sua abertura, visualização ou impressão. As mais recentes aplicações de *office automation*, por exemplo, *MS Word, Excel*, etc. Têm já opções de gravação de documentos com *password* individual.

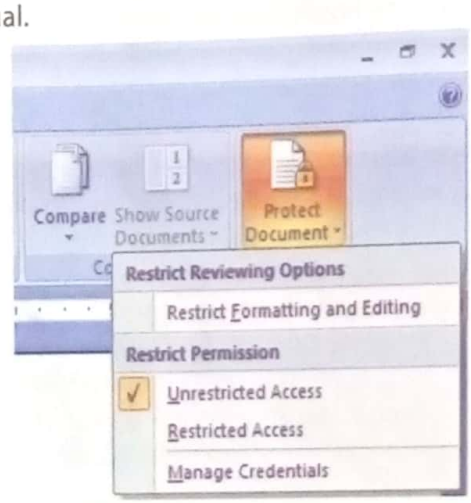


Fig. 2.20 Protecção de documentos no MS Word

O que não deve fazer de modo a garantir a segurança dos equipamentos informáticos

1. Não desligar o computador enquanto estiver a rodar um programa.
2. Não abrir a caixa da unidade do sistema com o cabo de alimentação de energia conectado.
3. Evitar conectar ou desconectar periféricos com o computador ligado (excepto dispositivos USB).
4. Não instalar o computador e periféricos externos sob a luz solar directa.
5. Não remover *CD's, DVD's* ou outros suportes de informação enquanto as luzes indicadoras de acesso às unidades estiverem acesas.
6. Não ligar e desligar continuamente os seus equipamentos. Aguarde pelo menos 30 segundos se desligou o computador e pretende ligá-lo de novo.
7. Não alterar a configuração do seu computador ou instalar periféricos se não estiver seguro de como fazê-lo.

Verifique o estado de segurança com o Centro de Segurança do Windows

A Central de Segurança do Windows é o ponto central da segurança do computador na plataforma *Windows*. Esta funcionalidade mostra o estado de segurança actual do computador e apresenta recomendações sobre as acções que se deve efectuar para tornar o computador mais seguro.

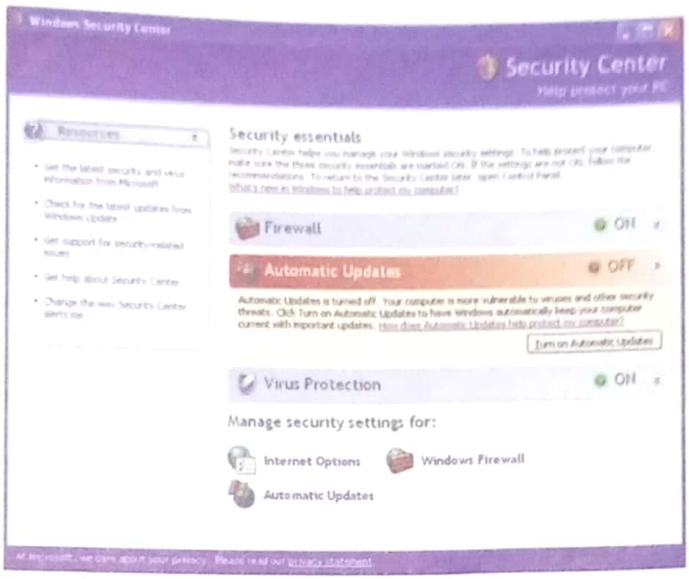


Fig. 2.21 Central de Segurança do Windows

A Central de Segurança do Windows **inspeciona** o *status* de vários componentes fundamentais da segurança do computador, inclusive configurações de **firewall**, actualizações automáticas do *Windows* e configurações de *software antimalware*, de segurança da *Internet* e do Controle de Conta de Usuário. Se o *Windows* detectar um problema em um destes componentes fundamentais da segurança (por exemplo, se o programa antivírus estiver obsoleto), a Central de Segurança exibe uma notificação e coloca um ícone da Central de Segurança na área de notificação.

Firewall

Uma *firewall* é um sistema ou uma combinação de sistemas (*hardware ou software*) que assegura o cumprimento de políticas de controlo de acesso (segurança) entre duas ou mais redes. Portanto, uma *firewall* pode ajudar a proteger o computador, impedindo que os *hackers* ou *software* malicioso consigam aceder-lhe.

Activar o firewall do Windows

1. Para abrir a Central de Segurança, clique no botão **Iniciar** , em **Painel de Controle**, em **Segurança** e, depois, em **Central de Segurança**.
2. Clique em *Firewall* e então clique em **Activar agora**. Se for solicitado a informar uma senha de administrador ou sua confirmação, digite a senha ou forneça a confirmação.

Actualizações automáticas (*automatic updates*)

Actualizações (*updates*) são adições ao *software* capazes de evitar ou corrigir problemas, aumentar a segurança do computador ou melhorar seu desempenho.

O *Windows* pode seguir uma **rotina de verificação das actualizações** para o seu computador e instalá-las automaticamente. Para o efeito ele recorre a Central de Segurança para verificar se a Actualização automática está activada. Se a actualização estiver desactivada, a Central de Segurança exibe uma notificação e coloca um ícone da Central de Segurança na área de notificação.

Activar a actualização automática

1. Para abrir a Central de Segurança, clique no botão **Iniciar**, em **Painel de Controle**, em **Segurança** e, depois, em **Central de Segurança**.
2. Clique em **Actualização automática** e então clique em **Activar agora**. Se for solicitado a informar uma senha de administrador ou sua confirmação, digite a **senha** ou forneça a **confirmação**.

Protecção contra malware

O *software* antivírus pode ajudar a proteger o computador contra vírus, *worms* e outras ameaças de segurança. O *software anti-spyware* pode ajudar a proteger o computador contra *spyware* e outro *software* potencialmente indesejado.

Instalar ou activar software anti-malware

1. Para abrir a Central de Segurança, clique no botão **Iniciar**, em **Painel de Controle**, em **Segurança** e, depois, em **Central de Segurança**.
2. Clique em **Protecção** contra *malware*, clique no botão em **Protecção contra vírus** ou **Protecção contra *spyware*** e outro *malware*, e então **escolha** a opção desejada.

Outras configurações de segurança

O *Windows* verifica também as configurações de segurança da *Internet* e as de Controle da Conta de Usuário para certificar se foram definidas nos níveis recomendados. Se as configurações de segurança da *Internet* e as de Controle da Conta de Usuário não foram definidas nos níveis de segurança recomendados, a Central de Segurança exibe uma **notificação e coloca um ícone da Central de Segurança** na área de notificação.

Restaurar as configurações da *Internet* para níveis recomendados

1. Clique no botão **Iniciar** → **Painel de Controle**, → **Segurança** → **Central de Segurança**.
2. Clique em **Outras configurações de segurança**.
3. Em Configurações de segurança da *Internet*, clique em **Restaurar configurações**.
   Siga um destes procedimentos:
   - Para redefinir automaticamente as configurações de segurança da Internet que devem estar no nível padrão, clique em **Restaurar minhas configurações de segurança da Internet** agora.
   - Para redefinir manualmente as configurações de segurança da *Internet*, clique em **Eu mesmo desejo restaurar minhas configurações de segurança da Internet**. Clique na **zona de segurança** cujas configurações deseja alterar e clique em **Nível personalizado**.

Configurações da Conta do Usuário

1. Clique no botão **Iniciar** → **Painel de Controle**, → **Segurança** → **Central de Segurança**.
2. Clique em **Outras configurações de segurança**.
3. Em Controle da Conta de Usuário, clique em **Activar** agora. Se for solicitado a informar uma senha de administrador ou sua confirmação, digite a senha ou forneça a confirmação.

Exercícios de consolidação

1. Em que situações não podemos utilizar a estrutura *FOR*?

2. Dado o trecho de programa abaixo:

```
...
READLn(N)
R := 1;
I := 2;
WHILE I <= N-1 DO
BEGIN
 R := R * 2;
 I := I + 1;
 END;
WRITE(R);
...
```

Reescreva-o utilizando:

a) O comando *FOR*.

b) O comando *REPEAT*.

3. Faça um programa que mostre todos os números inteiros pares de 2 a 100.

4. Faça um programa para gerar e exibir os números inteiros de 20 até 10, decrescendo de 1 em 1.

5. Qual das alternativas melhor define a segurança da informação?

A. Conjunto de medidas visando a protecção da informação contra qualquer acesso.

B. Conjunto de medidas visando a protecção da informação contra acesso e modificação autorizadas.

C. Conjunto de medidas visando a protecção da informação contra o acesso e modificação não autorizadas.

D. Conjunto de medidas com vista a evitar que a informação seja acedida, modificada ou eliminada de forma não autorizada.

6. O cumprimento de normas de segurança informática pelos usuários de uma organização, deve ser garantido pela aplicação formal de:

A. Conscientização e treinamento. **B.** Sanções e penalidade.

C. Termo de compromisso. **D.** Mecanismo de controle de acesso.

7. O programa malicioso que, uma vez instalado em um computador, permite a abertura de portas, possibilitando a obtenção de informações não autorizadas, é o:

A. *Firewall* **B.** *Trojan horse* **C.** *Spam killer* **D.** *Vírus de macro*

8. Tradicionalmente realiza a protecção de computadores de uma rede contra os ataques (tentativas de invasão) provindos de um ambiente externo. Trata-se de:

A. Roteador **B.** Antivírus **C.** *Password* **D.** *Firewall*

9. No capítulo segurança informática, a propriedade que traduz a confiança em que a mensagem não tenha sido alterada desde o momento da sua criação é:

A. Autencidade **B.** Confidencialidade **C.** Não-repúdio **D.** Integridade

10. Seleccione a melhor forma de privacidade para dados que estejam a trafegar numa rede:

A. Desactivação da rede e utilização dos dados apenas em «papel impresso».

B. Métodos de *Backup* e recuperação eficientes.

C. Criptografia.

D. Emprego de sistema de senhas e autenticação de acesso.

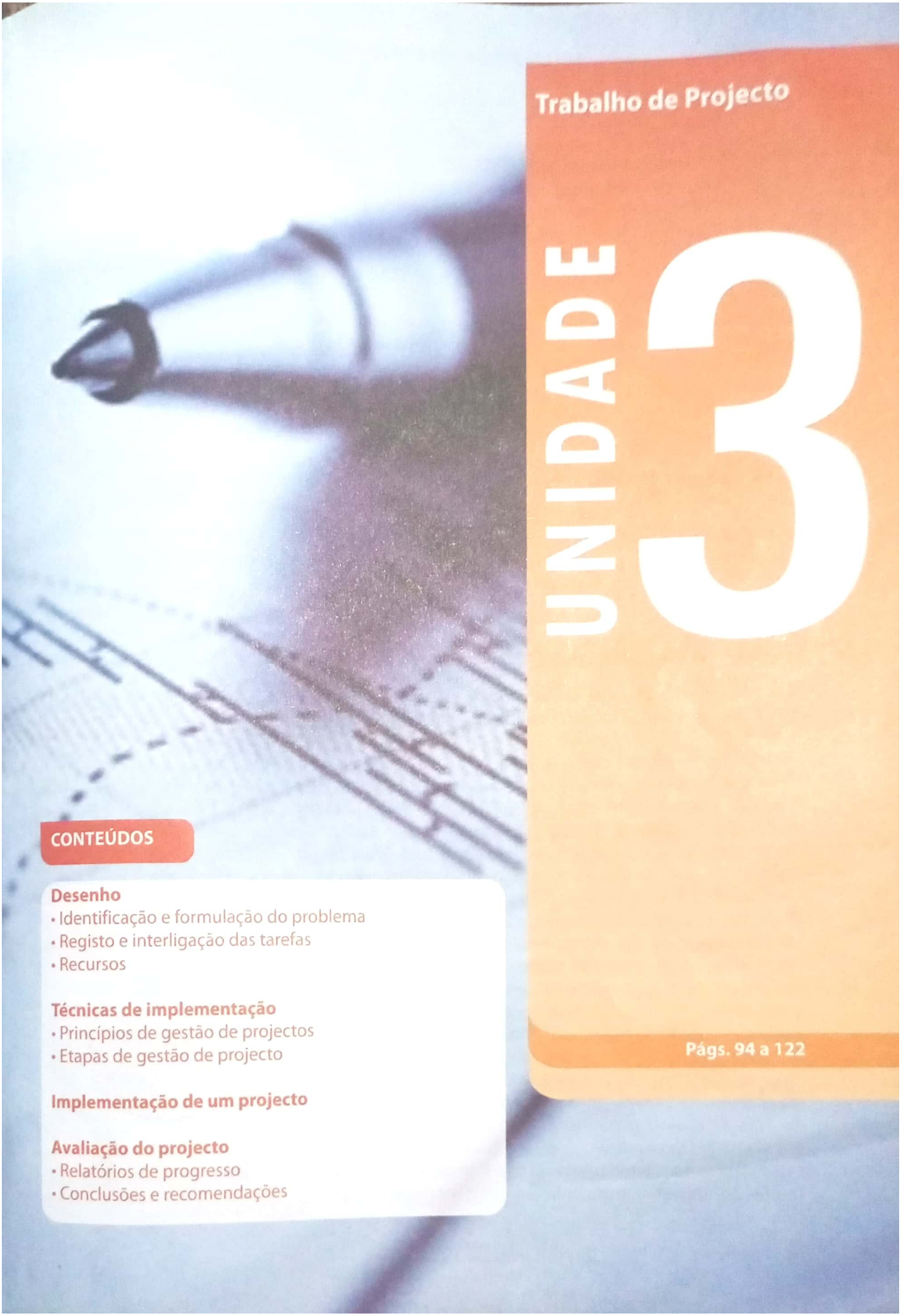
11. Um código malicioso que se altera em tamanho e aparência cada vez que infecta um novo programa é um vírus do tipo:

A. De Macro **B.** Parasita **C.** Camuflado **D.** Polimórfico

OBJECTIVOS

O aluno deve ser capaz de:

- Elaborar projectos simples
- Dominar as técnicas de implementação
- Implementar um projecto.
- Avaliar um projecto.



Trabalho de Projecto

UNIDADE 3

CONTEÚDOS

Desenho
- Identificação e formulação do problema
- Registo e interligação das tarefas
- Recursos

Técnicas de implementação
- Princípios de gestão de projectos
- Etapas de gestão de projecto

Implementação de um projecto

Avaliação do projecto
- Relatórios de progresso
- Conclusões e recomendações

Págs. 94 a 122

Introdução

Todos os dias ouvimos falar de projectos. Basta estarmos atentos aos meios de comunicação de massas (jornal, rádio, televisão, etc.) para ouvirmos diversas referências aos mais variados tipos de «projectos» em curso, qualquer que seja o domínio de actividade, por exemplo, um grande projecto de obras públicas, como o da construção da vila olímpica no Zimpeto para os jogos africanos em 2011, ou o da construção de um aeroporto internacional em Nacala, ou um projecto de intervenção com vista a evitar a extinção de uma espécie animal num determinado habitat, ou ainda um projecto com vista à melhoria da qualidade de vida dos jovens.

Mas também falamos de projectos na escola, considerando trabalhos em grupo, ou de equipa, com vista a estudar e documentar um certo assunto ou ainda o projecto de plantio de árvores de fruta e sombra no recinto da nossa escola. De facto, há algo em comum entre os projectos em geral e os projectos que efectuamos na escola. Tanto uns como outros recorrem a uma metodologia própria, embora, obviamente, a escalas diferentes.

O que é um projecto?

O termo «projecto» vem do latim *pro* + *jectare* e significa «lançar para a frente, atirar». Projectar significa investigar um tema, um problema, uma situação com o objectivo de a conhecer e, se possível, apresentar interpretações e/ou soluções novas.

Características de um projecto

O conceito de projecto pode ser definido de várias maneiras. Porém, há um conjunto de características fundamentais que lhe estão quase sempre associadas:

- Primeiro é que um projecto é sempre uma **actividade intencional**. A sua realização pressupõe um **objectivo** formulado pelos autores e executores do projecto ou apropriado por eles, que dá unidade e sentido às várias actividades, e está associada a um **produto final** que pode assumir formas muito variadas mas que procura responder ao objectivo inicial e reflecte o trabalho realizado.
- Um projecto pressupõe uma margem considerável de **iniciativa** e de **autonomia** daqueles que o realizam, os quais se tornam co-responsáveis pelo trabalho e pelas escolhas ao longo das sucessivas fases do seu desenvolvimento; é por isso que na escola ele representa uma rica forma de aprendizagem onde a ideia de cada aluno é muito importante.
- Geralmente, há um **grupo de pessoas** envolvidas na realização do projecto, pelo que a cooperação e o trabalho em conjunto assumem igualmente uma grande importância, ainda que haja também projectos individuais.
- A **autenticidade** é outra característica fundamental de um projecto. Aquilo que se pretende fazer constitui um problema genuíno para quem o faz e envolve alguma originalidade. Não se chama projecto à mera reprodução de um trabalho já feito por outros.
- Um projecto envolve **complexidade** e **incerteza**. São as tarefas complexas e problemáticas que precisam de ser

Fig. 3.1 Espírito de equipa

«projectadas». O objectivo central do projecto constitui um problema ou torna-se uma fonte geradora de problemas.

- Um projecto tem um **carácter prolongado e faseado**. Pela sua própria natureza, um projecto corresponde a um trabalho que se estende ao longo de um período de tempo mais ou menos longo e percorre várias fases desde a formulação do objectivo central até à apresentação dos resultados passando pelo planeamento e execução.

Assim, o **Trabalho de Projecto** é um método de trabalho que se centra na investigação, análise e resolução de problemas em grupo, possibilitando, desse modo, o desenvolvimento do sentido da responsabilidade, da solidariedade e do espírito de equipa.

Síntese

O Trabalho de Projecto permitir-te-á:

- Desenvolver uma cultura de autonomia, pesquisa e reflexão.
- Descobrir as tuas inclinações vocacionais ou caminhos profissionais.
- Aprofundar a tua capacidade de relacionar conhecimentos diversos.
- Desenvolver a capacidade de comunicar e exprimir opiniões publicamente.
- Desenvolver a capacidade de questionar e imaginar.

É que, na verdade, o Trabalho de Projecto é uma estratégia que implica um método de acção participativo, solidário, tendo em vista objectivos realizáveis e estabelecidos de comum acordo. A realização do trabalho em moldes de projecto é hoje indispensável em diversas esferas da actividade escolar e profissional. Esta metodologia permite a gestão integrada dos empreendimentos, incluindo a discussão das melhores estratégias, a realização de estudos iniciais, a coordenação dos esforços dos diversos intervenientes, o emprego racional dos recursos, a avaliação e a tomada de acções correctivas sempre que haja desvios relativamente aos planos inicialmente traçados.

Importante!

Um projecto é um empreendimento com determinados objectivos, levado a cabo adoptando estratégias adequadas, executando um conjunto de actividades coordenadas, realizadas por uma equipa de participantes ao longo de um tempo determinado, e empregando diversos recursos.

Elaboração do Projecto

Um projecto surge em resposta a um problema concreto, entendendo-se por «problema» a «diferença entre uma situação existente e uma outra que é desejada». **Elaborar um projecto** é, antes de mais nada, contribuir para a solução desse problema, transformando ideias em acções.

O documento chamado projecto é o resultado obtido ao «projectar-se» no papel tudo o que é necessário para o desenvolvimento de um conjunto de actividades a serem executadas: quais são os objectivos, que meios serão utilizados para atingi-los, que recursos serão necessários, onde serão obtidos e como serão avaliados os resultados.

A organização do projecto num documento auxilia-nos a sistematizar o trabalho em etapas a serem cumpridas, a compartilhar a imagem do que se quer alcançar, a identificar as principais deficiências a superar e a apontar possíveis falhas durante a execução das actividades previstas.

Importante!

A Metodologia do Trabalho de Projecto consiste na adopção de um conjunto de procedimentos, de técnicas e na escolha de instrumentos com vista a atingir os objectivos do projecto.

A **Metodologia do Trabalho de Projecto** assenta numa ordem lógica de procedimentos e operações que se interligam; por isso, há determinadas etapas que devem ser observadas na formulação de um projecto. São etapas que se sucedem no tempo, na maioria das vezes, com alguma sobreposição. Todas elas são essenciais à consecução dos objectivos e, por consequência, todas devem ser tratadas com igual atenção:

- Concepção
- Definição dos objectivos
- Selecção de estratégias
- Planeamento das actividades
- Produção documental
- Realização
- Avaliação

Fig. 3.2 Equiparação das Etapas na formulação do Projecto

Obviamente que estas etapas têm por sua vez diferentes subetapas, consoante o projecto em questão.

Concepção

O aluno ou grupo de alunos **formula uma ideia**, original ou não, com vista à resolução de um problema, quer dizer, o projecto será desenvolvido em torno de uma situação definida e assumida pelo grupo. Nesta etapa define-se o problema, quer **quantitativa**, quer **qualitativamente**, estabelecendo-se prioridades, indicando-se as causas prováveis e seleccionando-se os recursos e os grupos intervenientes. É uma etapa que se traduz naquilo que normalmente se chama **pesquisa – acção**.

O problema deverá ser relevante e significativo para cada um dos participantes, e deve ser tratado em ligação com o contexto (realidade) em que se insere e com as experiências dos alunos. Só assim haverá garantias para um real envolvimento de todos os alunos.

De modo a garantir-se este envolvimento de todos, o problema a solucionar deve ter as seguintes características:

- Ser autêntico, real.
- Ser relevante e significativo para cada um dos participantes.
- Ter uma ligação com o meio social dos participantes, i.e., ser possível resolvê-lo tendo em conta as condições do meio envolvente, partindo das experiências de cada aluno.
- Ser passível de ser investigado (exequível).
- Admitir vários caminhos de resolução.
- Reflectir vários ramos do saber.
- Ser susceptível de ser formulado através de um conjunto de questões.

Definição de objectivos

A formulação de objectivos deverá ser precisa, sem ambiguidades e deverá partir do conhecimento da realidade.

Algumas considerações sobre formulação e selecção de objectivos

Para uma correcta formulação e selecção de objectivos, é preciso distinguir entre: **finalidades** ou metas e **objectivos gerais** e **específicos**.

É de todo aconselhável que o projecto não se espartilhe em demasiadas finalidades (pois os seus promotores correm o risco de se perderem), escolhendo, por isso, uma só finalidade.

É necessário, então, escolher pontos intermédios de chegada: falamos da definição de objectivos gerais e específicos.

Os objectivos gerais indicam as grandes intenções de um projecto. Em regra, como também não são formuláveis em termos operacionais, carecem de datação e de localização precisas.

Os objectivos específicos devem permitir desmontar os objectivos gerais, pelo que terão que ser formulados em termos operativos, o que deixará avaliar sobre a sua concretização. Por outro lado, serão susceptíveis de ser atingidos a curto prazo e o seu enunciado não dará lugar a ambiguidades de interpretação sendo, sempre que possível, quantificados. Os objectivos específicos têm como alcance o sector de actividade em relação aos quais são definidos.

Selecção de estratégias

A selecção de estratégias deverá partir da articulação entre os objectivos e os recursos disponíveis. *Caminhante não há caminho, o caminho faz-se andando.*

Planeamento das actividades

Nesta fase são elaborados os documentos que servem de **suporte** à **implementação** e conclusão do projecto. Nestes documentos são definidas as datas de início / fim do projecto, as tarefas do projecto, os recursos (humanos ou materiais) essenciais para a execução das tarefas, os custos do projecto (em termos de salários e custos de materiais), e as dependências entre tarefas.

Ao iniciar a planificação de um projecto devemos ter em mente que vamos ter de responder, pelo menos, às seguintes perguntas:

- Qual é a data de início do projecto?
- Quais são as tarefas a executar?
- Quais são as datas de início e término das tarefas?
- Quem vai executar o trabalho?
- Como gerir os custos?

Documentação

Do estudo feito, elabora-se um dossier do projecto com a descrição das soluções encontradas, escolhas tecnológicas feitas, cálculos e dimensionamentos efectuados, estudos gráficos, desenhos efectuados, orçamentação efectuada, etc.

Nesta etapa as **TIC** constituem um veículo organizador da informação e ao mesmo tempo um facilitador do processo de sistematização da preparação do trabalho. Construir um cronograma, um organigrama e elaborar instrumentos de pesquisa com o processador de texto e identificar recursos na *Internet*, são exemplos da sua utilização. Por outro lado, discutir e confrontar ideias com os outros parceiros pode também ser feito através da comunicação **síncrona**, através de um *IRC/Chat* ou **assíncrona**, quer se utilize o correio electrónico, quer um fórum de discussão, permitindo aos alunos interagir numa comunidade mais alargada para partilhar informação, dados, recursos e ideias.

Realização

São executadas as acções definidas no plano de projecto. Isto implica a mobilização atempada dos recursos humanos e materiais e o acompanhamento das acções por parte da gestão do projecto.

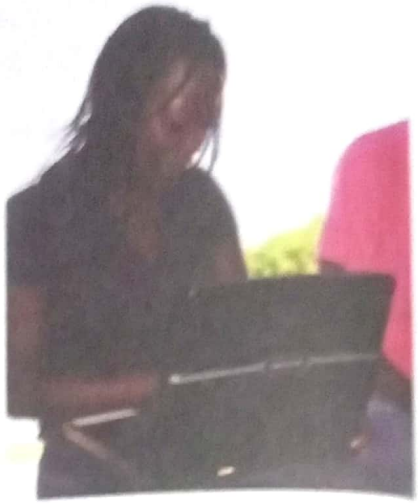


Fig. 3.3 As TIC podem dar o seu contributo

Avaliação

Na fase final do projecto é importante que o grupo faça uma reflexão sobre todo o processo, sobre as dificuldades que sentiu, sobre as estratégias que utilizou e sobre a forma como encontrou as soluções.

Divulgação dos resultados

Deverá haver um **relatório final** de divulgação dos resultados. Na divulgação dos resultados os alunos darão a conhecer aos seus pares, à comunidade educativa e/ou à comunidade em geral, o resultado do seu trabalho, dando significado à produção realizada.

Aqui também as TIC apresentam grandes potencialidades tanto na apresentação (que pode ser feita recorrendo a uma apresentação em *PowerPoint*) como na divulgação, que pode ser feita através do suporte de papel – brochuras, cartazes, relatórios, etc.

Ao longo destas fases, deve sempre estar presente um forte espírito de equipa e uma liderança firme rumo aos objectivos.

Fig. 3.4 Espírito de equipa e liderança firme

Quer dizer

O Trabalho de Projecto é uma metodologia que deve ser assumida em grupo e pressupõe um grande envolvimento de todos os participantes.

Resumo

Vivemos numa época em que se fala constantemente de projectos. Hoje em dia, nos mais diversos domínios de actividade, a concepção e o desenvolvimento de projectos surge frequentemente como uma tentativa de dar resposta a problemas complexos com que nos defrontamos. Por isso, achámos que era muito importante terminar este nível de ensino com algumas noções sobre a Metodologia de Trabalho de Projecto, pois assim sai munido de mais uma ferramenta. Tal como diz Jean-Pierre Boutinet, o projecto é uma figura emblemática da nossa modernidade. Ao contrário das sociedades tradicionais, o nosso tempo é caracterizado pelas «culturas de antecipação» que, sob o impulso da evolução científica e tecnológica, recorrem a figuras diversas (projectos, previsões, planificações) com o propósito de «explorar o futuro para domesticá-lo».

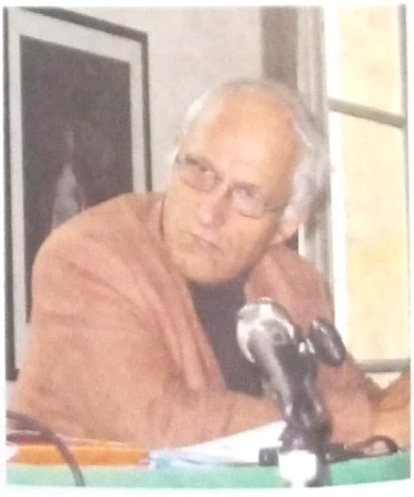


Fig. 3.5 Jean-Pierre Boutinet

Documento

Uma referência da nossa modernidade

Em oposição às sociedades tradicionais, a nossa cultura tecnológica fala cada vez mais de projecto: para se convencer de tal não é preciso mais do que prestar atenção ao vocabulário utilizado. Podemos certamente questionar-nos se isso proporciona uma ajuda aos indivíduos na determinação das suas intenções. E, logo que passa da fase de concepção à de realização, constituirá o projecto um guia eficaz para a acção, sobretudo quando se medem as distâncias (mesmo as falhas) que separam aquilo que foi projectado daquilo que será, na sua sequência, concretizado? O que se passa com o projecto de inserção dos jovens, com o projecto de planeamento de uma região consignada num esquema director, com o projecto de desenvolvimento de uma nação concretizado num plano?

Poderíamos multiplicar as referências para as situações concretas que recorrem ao projecto; na sua grande variedade, apresentam, pelo menos, uma constante: de forma bastante frequente, o projecto possui uma conotação positiva, aparece como naturalmente bom, daí esta valorização sistemática.

Jean-Pierre Boutinet, 1996
Antropologia do Projecto

Ferramentas de gestão de projectos

Introdução

Para o trabalho de projecto existe hoje uma gama disponível de ferramentas padrão de escritório e de recursos de consulta, pesquisa e comunicação através da *Internet*. Um exemplo dessas ferramentas específicas de grande interesse é uma aplicação de **Gestão de Projectos** como, por exemplo, o *Planner*, o *OpenProject* ou o *Microsoft Project*.

Estes programas foram especialmente concebidos para responder às necessidades de gestão dos projectos e permitem a representação dos projectos nas suas várias dimensões, ligadas aos recursos, custos, tarefas e a sua disposição no tempo.

Gráfico de *Gantt*

O **gráfico de Gantt** foi inicialmente divulgado pelo engenheiro americano Henry Gantt em 1910 e consiste numa representação gráfica do posicionamento óptimo das diferentes actividades de um projecto tendo em conta as durações e relações de precedência, bem como prazos de entrega e capacidades disponíveis.

Fig. 3.6 Henry Gantt

O diagrama de *Gantt* corresponde a um quadro onde:

- Colunas correspondem às unidades de tempo.
- Linhas correspondem às actividades a realizar.

Finalidades:

- Definir antecipadamente as fases de trabalho, de modo a evitarem-se operações repetidas.
- Acompanhar o plano, isto é, permitir a quem planifica acompanhar a forma como as etapas estão a ser executadas.
- Estabelecer uma sequência lógica, ou seja, a existência de uma sucessão óptima de etapas, eliminando à partida etapas desnecessárias.

Mapa de *Gantt*

Para se construir um Mapa de *Gantt* é necessário fazer-se uma listagem prévia das tarefas que devem ser realizadas e, posteriormente, construir-se o gráfico, em que a duração da actividade é representada por uma barra horizontal, cuja localização no diagrama (indicando o início e o fim da actividade) é o resultado das relações de precedência das diferentes actividades.

Exemplo

Consideremos o exemplo em que a nossa escola recebeu do MINED um fundo adicional e com ele pretende instalar uma sala de Informática.
Portanto, o título é: «*Projecto de instalação de uma sala de Informática*».
Sendo necessário efectuar um estudo das necessidades e efectuar a encomenda de computadores, pensa-se que três dias serão suficientes para esta tarefa. Posteriormente, é necessário preparar a sala em termos de mobiliário e disposição logística, actividade com uma duração de 6 dias. A recepção dos equipamentos é efectuada gradualmente e durante 9 dias. Por outro lado, à medida que os equipamentos vão chegando, é instalado o *software* necessário, durante 5 dias. Após a instalação do *software*, são efectuados testes e a configuração das aplicações, durante 4 dias, sendo dado por terminado o projecto.

Listagem das tarefas que devem ser realizadas:

Tarefas	Nome	Descrição	Duração	Precedências
1	A	Pré-estudo e encomenda	3 dias	------
2	B	Preparação do espaço	6 dias	1
3	C	Recepção dos equipamentos	9 dias	1
4	D	Instalação de software	5 dias	1
5	E	Configuração e testes	4 dias	4

Construção do diagrama de GANTT

DiasActividade	1	2	3	4	5	6	7	8	9	10	11	12
A	3d											
B				6d								
C				9d								
D				5d								
E									4d			

Para a sua realização, o critério consiste em traçar primeiro as actividades sem precedentes, seguidas das actividades que têm como precedentes as actividades já representadas, e assim sucessivamente.

Duma forma geral, são as seguintes as fases de construção do gráfico de Gantt:

- Definir a data de início do projecto.
- Listar todas as tarefas do projecto.
- Em projectos de certa complexidade, estabelecer uma hierarquia entre as tarefas.
- Inserir as tarefas por ordem de sequencialidade (naturalmente, é possível alterar a ordem das tarefas).

- Inserir a lista dos recursos do projecto; devemos indicar se cada recurso é material ou humano, dados adicionais do recurso e respectivo custo horário.
- Atribuir os recursos às actividades.
- Definir a duração de cada tarefa.
- Definir as dependências das tarefas.
- Rever todos os pormenores do gráfico e introduzir correcções ou optimizações, se necessário.

Planner

O **Planner** é uma ferramenta de gestão de projectos de uso geral, pertencente à categoria de *software* livre, como tal é gratuito, e também é multiplataforma, ou seja, existem versões para instalar em diferentes sistemas operativos, como o *Linux* e o *Windows*. No sistema operativo *Linux*, o programa vem com o ambiente **Gnome**, bastando assiná-lo durante o processo de instalação no passo apropriado ou instalá-lo a *posteriori*.

O programa oferece um conjunto de funções que estão disponíveis mediante quatro apresentações separadas e distintas, a que chamamos **vistas**. Como veremos, pode-mos chamar cada uma dessas vistas clicando sobre os ícones da barra de ferramentas à esquerda, na janela do *Planner*.

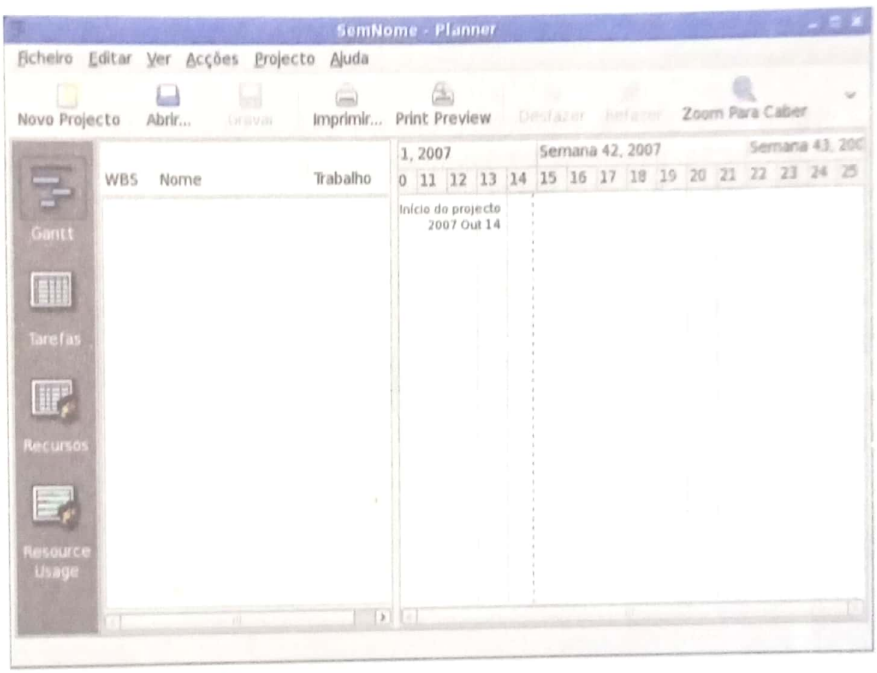


Fig. 3.7 Tela inicial do Planner

Vistas do projecto

São quatro as vistas em que o *Planner* nos mostra as informações do projecto segundo diferentes perspectivas:

Vista de *Gantt* é um gráfico de *Gantt*.

A vista de Gantt combina o gráfico de Gantt com uma versão abreviada da vista de tarefas, permitindo-nos:

- Visualizar uma representação gráfica da calendarização das actividades do projecto.
- Gerir relações entre tarefas, utilizando **clicar** e **arrastar**.

- Utilizar ferramentas de ampliação e redução para mostrar diversos níveis de pormenor.
- Visualizar os recursos atribuídos a cada tarefa.

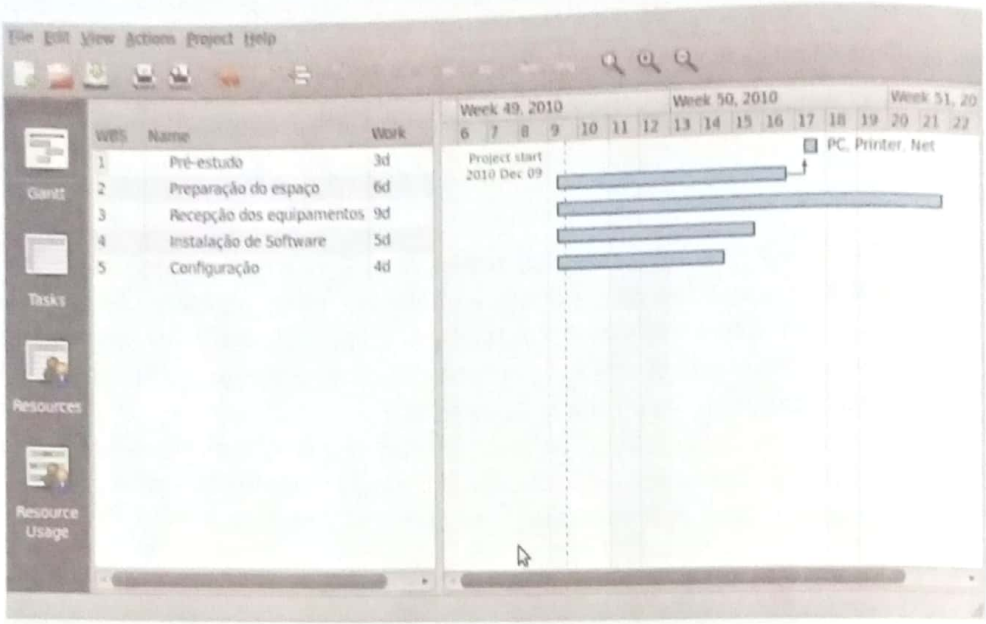


Fig. 3.8 Vista de Gantt

Vista de tarefas é uma lista das tarefas do projecto e as suas características.

As tarefas definidas para o projecto também são mostradas na vista de *Gantt*, mas a vista de tarefas apresenta mais pormenores sobre cada tarefa.

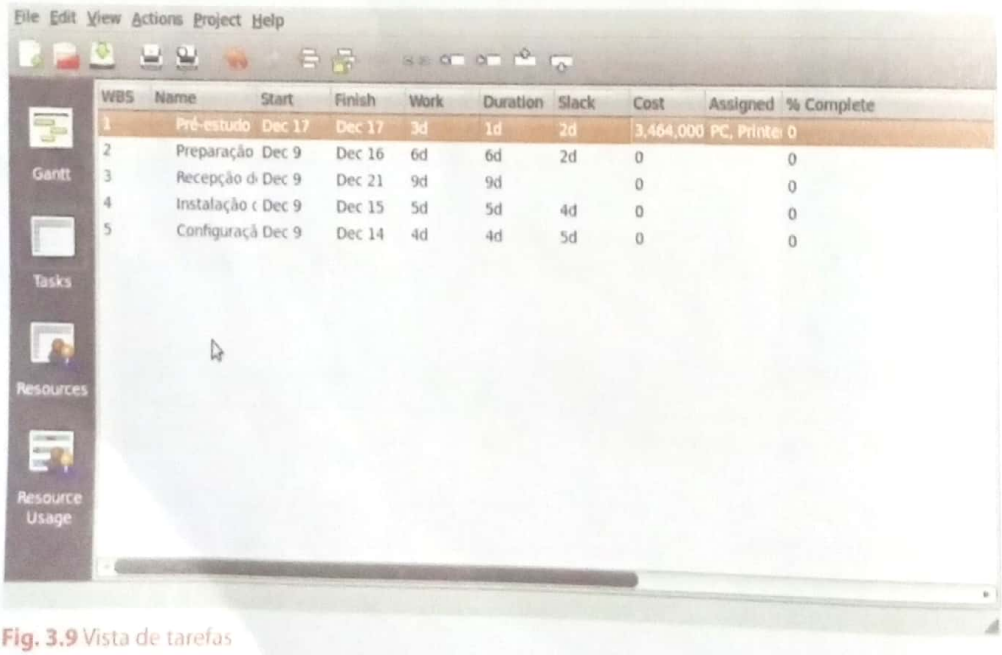


Fig. 3.9 Vista de tarefas

A **vista de tarefas** é usada para as seguintes funções:

- **Definição de tarefas** – decompor fases abrangentes do projecto em tarefas de menor âmbito, mais facilmente aplicáveis.
- **Sequenciamento de tarefas** – identificar dependências entre tarefas e outros constrangimentos, através do diálogo de propriedades da tarefa.

- **Estimativa da duração das tarefas** – estimar o tempo necessário à execução das tarefas.
- **Cálculo de custos das tarefas** – estimar os custos associados à execução das tarefas.

Vista de recursos é uma lista dos recursos que temos à disposição para o projecto, incluindo os respectivos dados básicos.

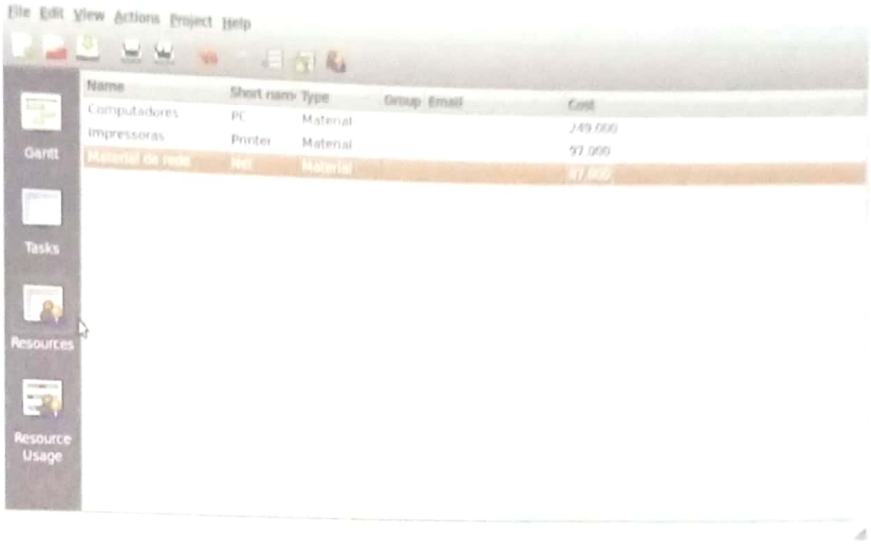


Fig. 3.10 Vista de recursos

A vista de recursos, também referenciada como *Resource Usage*, dá-nos acesso às seguintes funções:

- **Gestão da lista de recursos** do projecto, incluindo os recursos humanos, assinalados com o tipo **Trabalho**, e os recursos materiais, assinalados com o tipo **materiais**.
- **Gestão de grupos de recursos**; os recursos podem ser agrupados, lidando-se depois com as designações dos grupos de recursos, como pode acontecer, por exemplo, ao envolver-se numa obra um ou mais empreiteiros, cada um com os seus próprios recursos.
- **Gestão dos custos associados aos recursos**; a cada recurso pode ser atribuído um valor de custo horário.

Vista de utilização dos recursos é uma visualização do grau de utilização dos vários recursos atribuídos às tarefas.

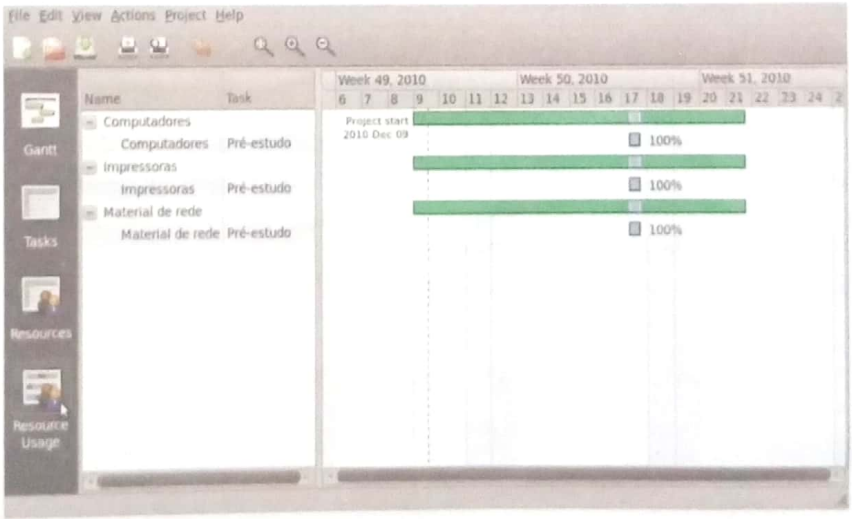


Fig. 3.11 Vista de utilização de recursos

A **vista de utilização dos recursos** apresenta a disponibilidade dos recursos baseada nas tarefas que lhes foram atribuídas. A forma gráfica é semelhante ao gráfico de *Gantt*, no entanto, neste caso a vista é organizada por recursos.

A **barra de resumo** mostra a disponibilidade geral do recurso. **Clicando** no pequeno triângulo do lado esquerdo do nome do recurso, a disponibilidade pormenorizada de cada recurso pode ser recolhida para cima, de modo que apenas seja visível a **barra de resumo**.

A codificação de cores significa o seguinte:

- **Verde** – que o recurso não está atribuído a qualquer tarefa nesse intervalo de tempo.
- **Azul** – que algo ligeiramente diferente, dependente do contexto. Na barra da tarefa, mostra que o recurso está parcial ou completamente atribuído à tarefa, com a percentagem de ocupação visível à frente; na barra de resumo, mostra que o recurso está completamente atribuído durante esse tempo.
- **Cinzento** – que o recurso está parcialmente atribuído nesse tempo.
- **Vermelho** – que o recurso está sobreutilizado nesse tempo.

Iniciar o *Planner* no ambiente *Gnome* (*Linux*)

Podemos iniciar o *Planner* do seguinte modo:

- Menu Aplicações → Seleccionar Produtividade → Gestão de Projectos

Iniciar o Planner no ambiente *Windows*

- Start → Programs → *Planner*

Quando o *Planner* arranca, aparece a janela representada na figura da página 104. Como podemos notar, a vista de *Gantt* é a pré-definida.

A janela do *Planner* contém os seguintes elementos:

- **Barra de menus** – os menus presentes nesta barra contêm todos os comandos necessários para trabalhar com ficheiros no *Planner*.
- **Barra de ferramentas** – contém um subconjunto dos comandos acessíveis na barra de menus.
- **Barra de estado** – apresenta informação sobre a actividade corrente do *Planner*, bem como a informação de contexto sobre os elementos do menu.

Abrir e guardar projectos

Para abrir um projecto, clicamos em **Ficheiro** → **Abrir**, ou no botão **Abrir** da barra de ferramentas. Aparece o diálogo **Abrir Ficheiro**. Seleccionamos o ficheiro do *Planner* que pretendemos abrir e seguidamente clicamos no botão **Abrir**. Os ficheiros do *Planner* são armazenados com a extensão *.planner*.

Para guardar um projecto, seleccionamos Ficheiro → **Gravar Como**... guardando como novo projecto, ou clicamos em Ficheiro → **Gravar** para guardar um projecto existente, mantendo-lhe o nome.

Editar as propriedades do projecto

Para **editar** as **propriedades do projecto**, seleccionamos **Projecto** → **Editar** as Propriedades do Projecto. Nesta caixa de diálogo podemos editar o nome do projecto, a sua data inicial, o nome do gestor do projecto, a organização em que se desenrola o projecto, a fase do projecto e o calendário pré-definido.

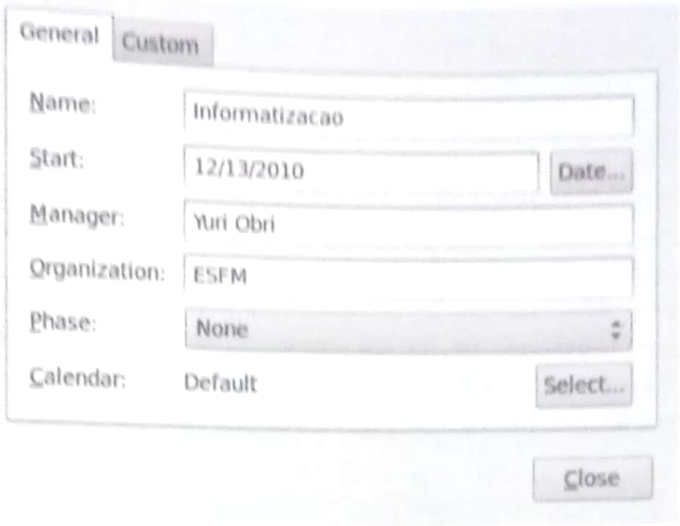


Fig. 3.12 Caixa de diálogo «Propriedades do projecto»

Atribuição de fases no projecto

As fases podem ser adicionadas seleccionando **Projecto** → **Editar** às Fases do Projecto, o que provoca a aparição de um diálogo com uma lista simples de fases de projecto e botões para as adicionar e remover.

Calendário

Cada projecto possui um **calendário pré-definido**. Este aplica-se a todos os recursos do projecto, a não ser que se especifique de outra forma no editor de recursos. Utilizamos o botão de selecção neste diálogo para alterar o calendário pré-definido.

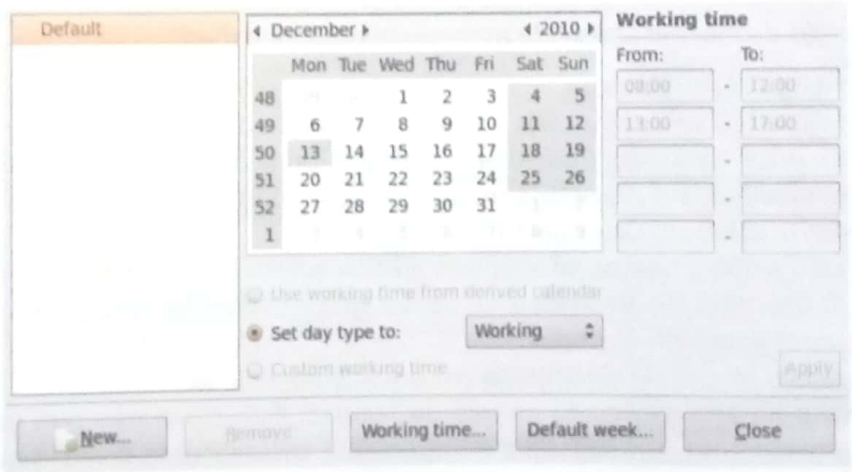


Fig. 3.13 O calendário

Editar os calendários do projecto

Para editar os calendários do projecto, seleccionamos **Projecto** → **Gerir os Calendários.** Os calendários do projecto ajudam-nos a planear, definindo, em termos de dias de trabalho e dias de descanso, quando é que os recursos podem ser usados e também quais as horas de trabalho que estão disponíveis nos dias de trabalho.

Trabalhar com as tarefas

Inserir tarefas

Para adicionar uma tarefa, podemos clicar no botão **Inserir Tarefa** da barra de tarefas, ou clicar na área de tarefas e seleccionar **Inserir Tarefa** no menu rápido que surge. Estas opções funcionam do mesmo modo na vista de *Gantt* e na vista de tarefas.

Para adicionar rapidamente várias tarefas, seleccionamos **Acções** → **Inserir Tarefas**, o que provoca a abertura do diálogo **Inserir Tarefa**, que permite a entrada rápida de várias tarefas. Basta inserir o nome da tarefa e a quantidade de esforço de trabalho e depois clicar em **Inserir**. A nova tarefa é adicionada e o diálogo permanece aberto e pronto para a inserção de outra tarefa.

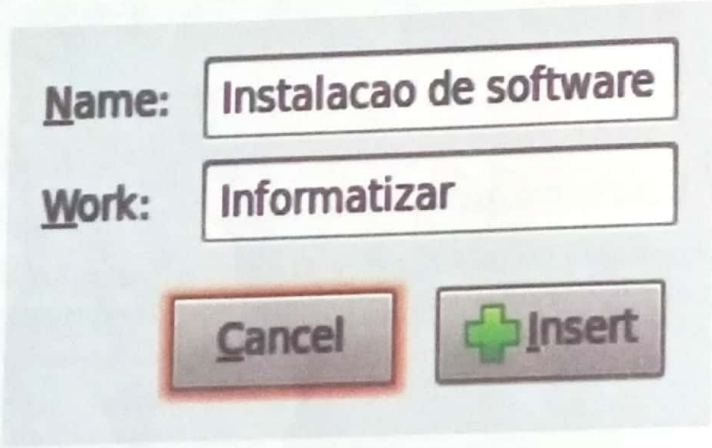


Fig. 3.14 Caixa de diálogo «Inserir tarefa»

Trabalhar com os recursos

A vista de recursos do *Planner* apresenta as seguintes propriedades:

- Nome – a designação do recurso.
- Abreviatura – nome curto ou as iniciais a serem mostradas no gráfico de *Gantt*.
- Tipo – os tipos disponíveis são Trabalho e Material; o primeiro desses tipos refere-se aos recursos humanos que trabalham no projecto, enquanto o segundo se aplica para o caso dos recursos materiais necessários à realização do projecto.
- Grupo – o grupo a que o recurso se encontra atribuído. Esta coluna oferece uma caixa de listagem com os grupos definidos. Se pretendermos utilizar grupos, antes de atribuir recursos temos de usar o editor de grupos para definir grupos.
- Email – o endereço de correio electrónico de contacto do recurso.
- Custo – o custo horário associado ao uso do recurso. Este custo não tem unidade monetária, pelo que em projectos internacionais é importante assumir uma única moeda a utilizar para gerir os custos.

Adicionar recursos

Podemos adicionar recursos por meio do botão **Inserir Recursos**, da barra de ferramentas, ou seleccionando **Acções** → **Inserir Recurso**. Para adicionar vários recursos de forma rápida, escolhemos **Acções** → **Inserir Recursos**, o que provoca a abertura do diálogo **Inserir um Recurso**, como se mostra na figura abaixo. Tal como no caso de **Adicionar Tarefas**, podemos rapidamente inserir diversos recursos e adicionar pormenores mais tarde.

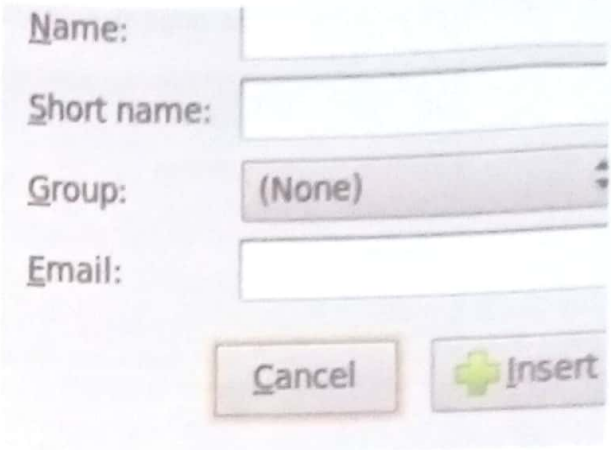


Fig. 3.15 Adicionar recursos

Imprimir

Podemos **imprimir** os dados do nosso projecto seleccionando **Ficheiro** → **Imprimir**, ou clicando no botão **Imprimir** da barra de tarefas.

O diálogo **Imprimir Projecto**, permite-nos escolher entre imprimir em impressoras disponíveis ou então gerar um ficheiro em formato **pdf**. Se escolhermos esta última possibilidade, as nossas opções de localização mudarão, para que indiquemos o caminho de armazenamento e o nome do ficheiro a gerar.

Open Project

OpenProj é uma alternativa *desktop* nova e robusta, desenvolvida sob licença de código aberto. É um *software* livre (*freeware*), e constitui uma forte alternativa contra a *Microsoft Project*. O *OpenProj* controla todos os aspectos de gestão de projectos, tais como planeamento e programação, gestão e alocação de recursos, simulação de processos alternativos, etc. Ele também fornece as funcionalidades necessárias para trabalhar com ambientes **multiprojectos**.

As suas principais funções estão concentradas no projecto de gráficos de **Gantt**, diagramas de rede *PERT*, e muitos outros gráficos.

Apesar de não ser um produto comercial, o *OpenProj* oferece todos os recursos necessários e suficientes para trabalhar em Projectos escolares, mas não só, e está disponível para as plataformas *Linux, Unix, Mac* e *Windows*.

OpenProj é uma aplicação concebida inteiramente em *Java*. Por conseguinte, é necessário ter o *plugin* instalado no computador para rodá-lo. Além disso, a aplicação só está disponível em inglês actualmente.

O ambiente do *OpenProj*

Para se instalar o programa basta baixá-lo (*download*) a partir do site oficial da *Projity* (**http://openproj.org**) que é a empresa responsável pelo desenvolvimento e distribuição do programa.

Após instalar o *software*, podemos começar a trabalhar num novo projecto. Quando executarmos o programa, vai aparecer uma janela de boas-vindas e, a partir daí, podemos seleccionar o botão **Novo Projecto**.

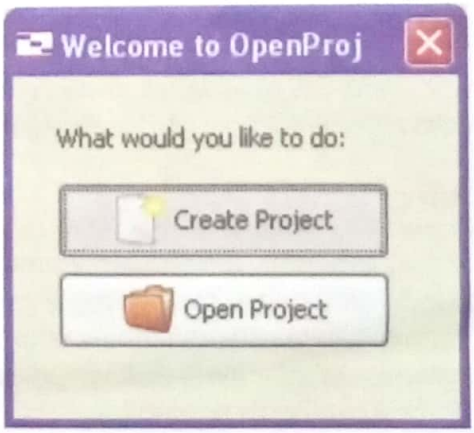


Fig. 3.16 Caixa de selecção

Abrir-se-á uma nova caixa, onde preenchemos os dados do projecto.

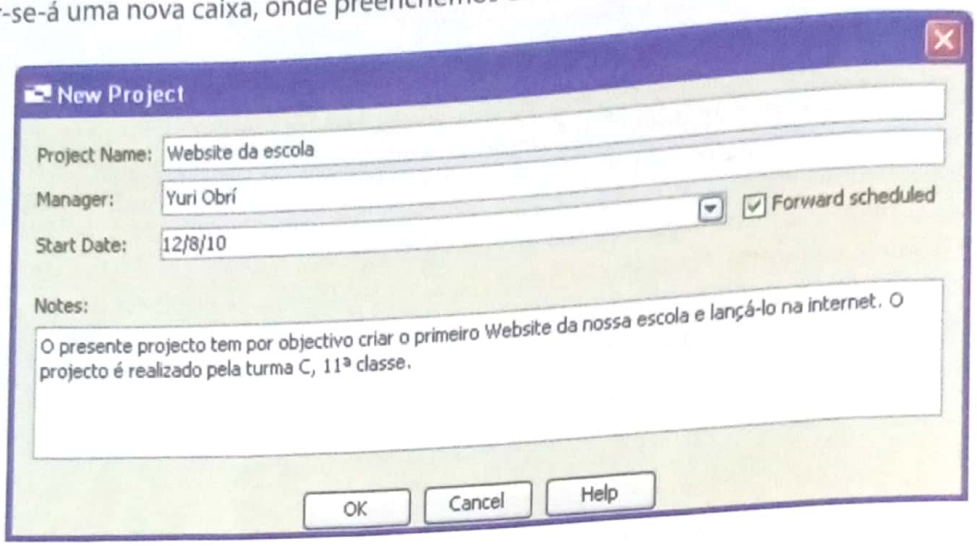


Fig. 3.17 Janela New Project – OpenProj

Automaticamente, vamos ver um gráfico de *Gantt* em branco onde podemos começar a trabalhar.

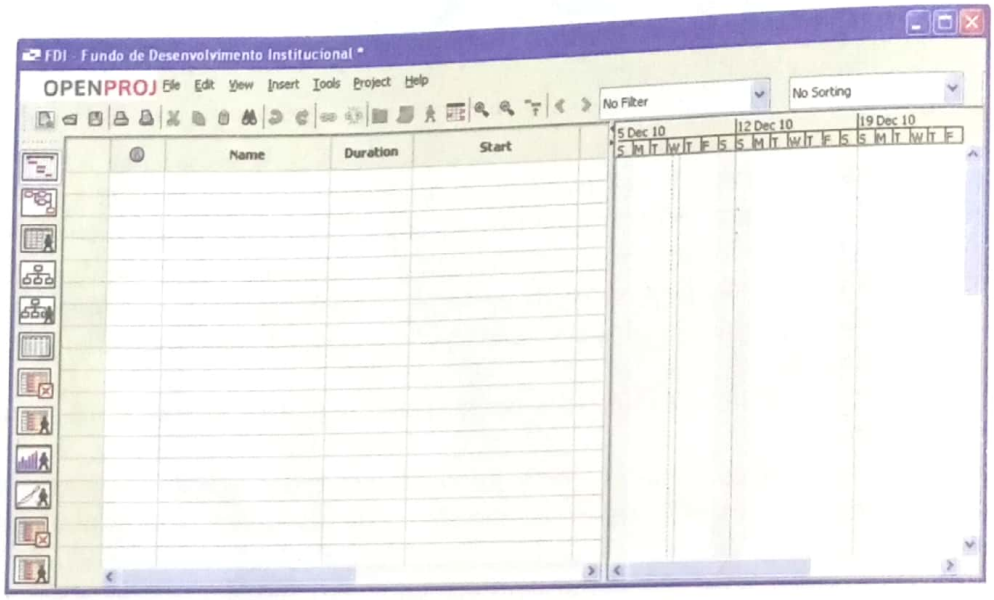


Fig. 3.18 Tela inicial do *OpenProj*

Uma das qualidades desta aplicação é a sua compatibilidade com o *MS Project da Microsoft*, isto é, com o *OpenProj* podemos abrir arquivos do *Microsoft Project*.

Microsoft Project

O *Microsoft Office Project* é uma aplicação concebida especialmente para a **gestão de projectos**. Ela dispõe de ferramentas flexíveis e combinadas que permitem gerir projectos com mais eficiência e eficácia, mantendo os seus autores informados sobre todos os detalhes do mesmo, bem como o controlo do trabalho, as agendas e as finanças do projecto incluindo o alinhamento das equipas de projecto tornando-os, assim, mais produtivos.

Características básicas do *Microsoft Project*

Dentre os diversos recursos disponíveis, destacam-se os seguintes:

- Utiliza tabelas no processo de entrada de dados.
- Existe um conjunto padrão de tabelas e o utilizador pode criar as suas próprias tabelas.
- É gerado automaticamente um Mapa de Gantt, auxiliando o processo de entrada de dados.
- Aceita relações de precedências entre tarefas (*Finish-to-Start, Start-to-Start, Finish-toFinish, e Start-to-Finish*).
- Permite tarefas recorrentes (ocorrem de forma repetitiva). Por exemplo, num projecto pode-se planear a realização de reuniões todas as segundas-feiras.
- Permite estabelecer níveis hierárquicos através de tarefas de resumo. Este aspecto é muito útil na criação da Estrutura de Decomposição do Trabalho.
- Permite a utilização de subprojectos.
- Possui recursos para agrupar, filtrar e classificar tarefas.
- Possui um conjunto padrão de relatórios e o utilizador pode criar os seus próprios relatórios.
- Permite a inclusão de campos do utilizador, que aceitam diversos tipos de operação.
- Permite a definição de semana de trabalho, expediente e feriados.
- O cálculo da rede pode ser feito do início para o fim ou do fim para o início.
- Permite o uso de datas programadas para as tarefas.
- Os recursos são ligados directamente às tarefas.
- Permite a redistribuição de recursos (ou nivelamento de recursos) manual ou automaticamente. Os custos são ligados directamente às tarefas na forma de custos fixos ou de custos em termos de valor/hora.

Criação do projecto

Para **planear** as **actividades de um projecto**, começamos por definir o nome do projecto. Para tal, recorremos ao menu *File - Save as*. Desta forma definimos o nome do nosso documento do *Project* que servirá de planeamento das nossas actividades. Antes de iniciarmos qualquer planeamento vamos ver as várias formas de visualização do *Microsoft Project*.

Formas de visualização

O *Microsoft Project* disponibiliza diversas **formas de inserção e visualização dos dados do projecto**. Esta diversidade de formatos permite, conforme o tipo de informação, ter uma visão mais abrangente do estado do projecto. Por defeito, a informação é visualizada graficamente através de um Mapa de *Gantt* (*Gantt Chart*). Mas podemos ver toda a informação de outras formas.

Para alterar a forma de visualização basta seleccionar no menu *View* a opção pretendida:

- ***Calendar***: é visualizado um calendário mensal com as tarefas previstas e respectivas durações.
- ***Gantt Chart***: as tarefas e respectivas durações são visualizadas graficamente com o decorrer do tempo.
- ***Network Diagram***: é visualizado um diagrama de rede com todas as tarefas e eventuais dependências.
- ***Task Usage:*** visualização das tarefas e todos os recursos afectos.
- ***Tracking Gantt***: forma de visualização que permite comparar o agendado inicialmente com a situação real.
- ***Resource Graph***: permite visualizar toda a informação de um ou mais recursos com o decorrer do tempo.
- ***Resource Sheet***: visualização dos recursos e informações relacionados. Esta forma permite também introduzir informações em forma de tabela.

- *Resource Usage*: é visualizada a lista de recursos com todas as tarefas agrupadas. Esta forma de visualização é adequada quando pretendemos saber, por exemplo, os custos associados a cada recurso num determinado período. Existem mais algumas vistas que não aparecem no menu *View*, mas se acedermos a ele e escolhermos a opção *More Views* temos acesso a uma caixa com todas as opções.

Iniciando o *MS Project*

À semelhança de todos os outros programas do pacote *MS Office da Microsoft*, para iniciar o *MS Project* clicamos no *Start – All Programs – Microsoft Office – MS Project*.

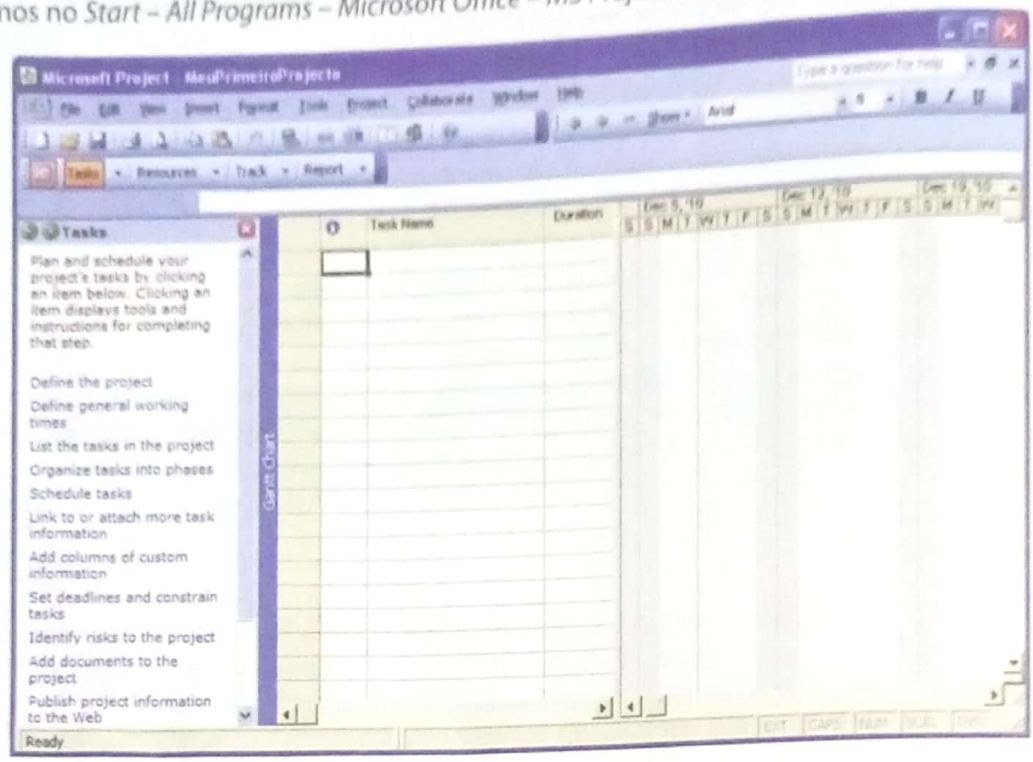


Fig. 3.19 Tela inicial do *MS Project*

1.ª etapa: criar um novo projecto

1. Clique em **Arquivo** e, em seguida, clique em **Novo**.
2. Verifique se a opção **Projecto Vazio** está seleccionada e clique em **Criar** no painel direito.

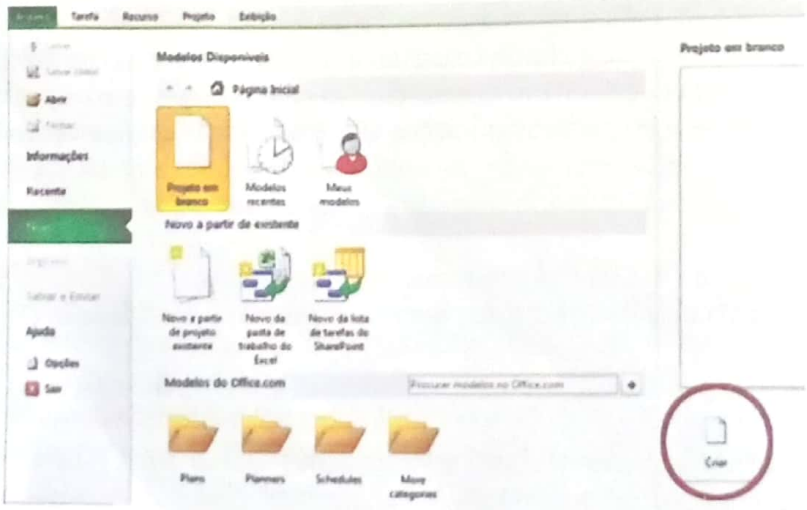


Fig. 3.20 Caixa de selecção

3. Na guia **Projecto**, no grupo **Propriedades**, clique em **Informações do Projecto**.

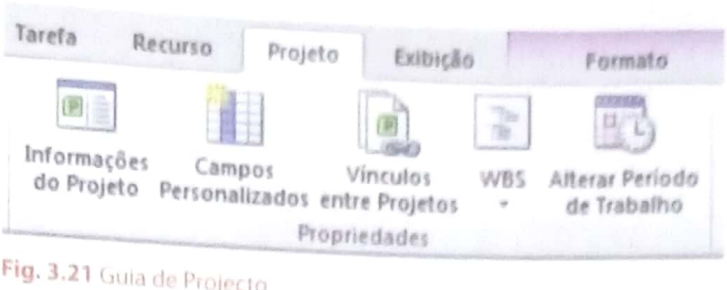


Fig. 3.21 Guia de Projecto

4. Agende o projecto na caixa de diálogo **Informações do Projecto**:

- Para indicar a data de início, clique em **Data de Início do Projecto** na caixa **Cronograma a partir de** e seleccione a data de início na caixa **Data de início**.
- Para indicar a data de término, clique em **Data de Término do Projecto** na **caixa Cronograma a partir de** e seleccione a data do término na caixa **Data do término**.

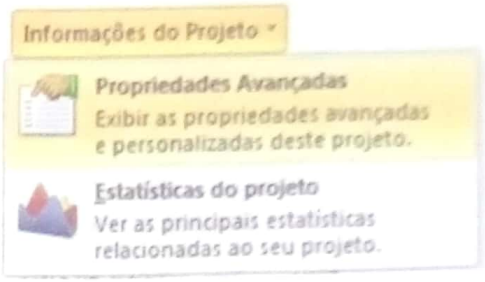


Fig. 3.22 Informações do Projecto

Definir propriedades de arquivo para o projecto

1. Abra o projecto.
2. Clique na guia **Arquivo** e, em seguida, clique em **Informações**.
3. No painel direito, clique em **Informações do Projecto** e, em seguida, clique em **Propriedades Avançadas**.
4. Na guia **Resumo**, digite informações importantes do projecto nas respectivas caixas.
5. Na guia **Personalizado**, digite as informações personalizadas nas caixas Nome, Tipo e Valor e clique em **Adicionar**.
6. Clique em **OK**.

2.ª etapa: adicionar tarefas ao projecto

Duma forma geral, os projectos começam com uma lista simples de tarefas que vai aumentando em complexidade à medida que este se vai concretizando. Depois de se criar ou importar-se a lista de tarefas, pode-se definir os relacionamentos entre elas.

3.ª etapa: estruturar o projecto

Para se tornar a lista de tarefas mais organizada e legível basta deslocar-se para a esquerda as tarefas do projecto de modo a criar-se uma estrutura resumida das tarefas e das subtarefas.

Escolher um método para organizar suas tarefas

Ao organizarmos as tarefas para um projecto, podemos agrupar aquelas que compartilham características semelhantes ou que serão concluídas no mesmo intervalo de tempo num resumo de tarefas, também conhecido como «**rede de tarefas**».

É possível usar-se as tarefas de rede para se mostrar as fases e subfases importantes do projecto. As tarefas de rede resumem os dados das suas subtarefas, que são as tarefas agrupadas abaixo delas. Recuando as tarefas na quantidade de níveis possíveis, podemos visualizar a organização do nosso projecto.

Existem dois métodos para organizar uma lista de tarefas:

- Com o **método de cima para baixo**, identificamos as fases importantes do projecto e dividimo-los em tarefas individuais. Este método fornece uma versão do plano que nos permite decidir sobre as fases mais importantes do projecto.
- Com o **método de baixo para cima**, listamos todas as tarefas possíveis primeiro e depois agrupamo-las em fases.

Criar resumo de tarefas e subtarefas

Como vimos antes, recuando as tarefas podemos criar uma estrutura de tópicos, originando a rede de tarefas e de subtarefas. Por padrão, as tarefas de resumo estão em negrito e recuadas para a esquerda, e as subtarefas são recuadas abaixo delas.

1. No modo de exibição Gráfico de *Gantt*, clique na **linha da tarefa** que deseja recuar como uma **subtarefa** ou recuar para a esquerda como uma **tarefa de resumo**.
2. Na faixa de opções, no grupo de tarefas, clique em **Recuo** para recuar a **tarefa** e transformá-la numa **subtarefa**.

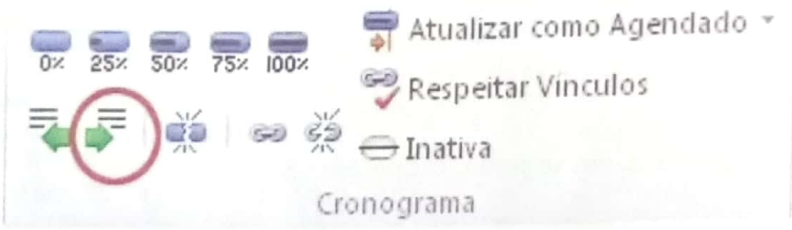


Fig. 3.23 Resumo de tarefas

Criar uma nova tarefa

Adicionar uma tarefa a uma lista de tarefas

1. Na guia **Exibir**, no grupo **Modos de Exibição de Tarefa**, clique em *Gráfico de Gantt*.
2. No campo **Nome da Tarefa** vazio, digite o nome da tarefa e pressione *Enter*.

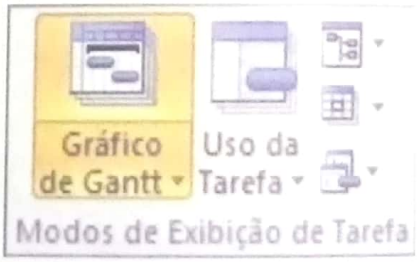


Fig. 3.24 Adicionar uma tarefa a lista de tarefas

Inserir uma tarefa entre tarefas existentes

1. Seleccione a linha **abaixo** donde se deseja que a nova tarefa apareça.
2. Na guia **Tarefa**, no grupo **Inserir**, clique na parte superior do botão **Tarefa**.
3. Digite o nome da **tarefa** na linha inserida. As identificações da tarefa são renumeradas automaticamente depois de se inserir uma tarefa.

Fig. 3.25 Grupo inserir

Adicionar uma tarefa ao modo de exibição diagrama de rede

1. Para alternar para o modo de exibição **Diagrama de Rede**, na guia **Exibir**, no grupo **Modos de Exibição de Tarefa**, clique em **Diagrama de Rede**.

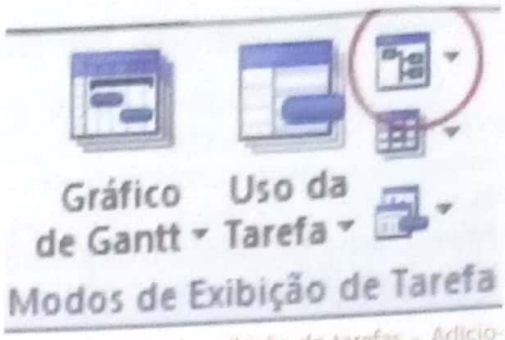


Fig. 3.26 Modo de exibição de tarefas – Adicionar tarefas.

2. Na guia **Tarefa**, no grupo **Inserir**, clique na parte superior do botão **Tarefa**.
3. Digite o nome da tarefa na caixa da nova tarefa.

Fig. 3.27 Inserir tarefas.

Relacionar as tarefas de um projecto

Em praticamente todos os projectos, as **tarefas** não são iniciadas todas ao mesmo tempo. Salvo raras excepções, as tarefas que compõem um projecto são executadas numa determinada ordem, ou relacionadas com outras tarefas. No *Microsoft Project* podemos definir esse relacionamento entre tarefas. Essa definição pode ser feita directamente no *Mapa de Gantt*.

Existem 4 tipos de relações entre tarefas:

- ***Finish-to-Start*** – uma determinada tarefa não se pode iniciar sem que outra esteja concluída.
- ***Start-to-Start*** – uma tarefa não se pode iniciar sem que outra o tenha sido. Esta situação ocorre quando se pretende que duas tarefas ocorram em simultâneo (pelo menos durante algum tempo).
- ***Finish-to-Finish*** – uma tarefa não pode ser concluída sem que outra o tenha sido também.
- ***Start-to-Finish*** – uma tarefa só pode ser concluída depois de outra ter sido iniciada.

Para definirmos qualquer um dos quatro tipos de relações, basta clicarmos com o botão direito do cursor sobre a tarefa que pretendemos atribuir a um dos outros tipos de relações e escolher a opção *Task Information*.

Planeamento dos recursos do projecto

Qualquer projecto **necessita de recursos**, que podem ser recursos humanos, recursos materiais, ou até mesmo recursos monetários. No *Microsoft Project* temos a possibilidade de definir recursos e afectá-los a tarefas. Vamos então ver, passo a passo, como se definem os recursos e como se afectam às tarefas que temos definidas.

Definir recursos

Para **definirmos** os Recursos que teremos disponíveis, apontamos o cursor para o menu *View* e escolhemos *Resource Sheet*. Aparece-nos uma folha onde podemos definir o nome do recurso, o tipo, as unidades, os custos e o calendário atribuído a esse recurso (se for recurso humano).

Janela de edição de dados dos recursos do projecto

Existem alguns aspectos a ter em conta quando estamos a definir os recursos. Se estivermos a definir os recursos humanos e os definirmos pela sua categoria (por exemplo, programador) então o valor total do número de programadores é definido no campo *Max Units*. Por exemplo: se tivermos quatro programadores, não vamos colocar quatro entradas para programador, mas no campo *Max Units* colocamos 400%. Se os recursos humanos forem definidos não pela categoria mas pelo seu nome, então criaremos quatro entradas para os quatro programadores, mas em que definimos os seus nomes no campo *Resource Name*. No campo *Type*, escolhemos o tipo de recurso: **Work** Recurso humano; **Material** Recurso Material; **Cost** Recurso Monetário.

Associar recursos a tarefas

Para **associarmos os recursos** às tarefas temos de voltar à vista de *Gantt (View - Gantt Chart)*. Os recursos são introduzidos na coluna *Resource Names* e são separados por «;» no caso de haver mais de um recurso associado a determinada tarefa.

4.ª etapa: calendarizar

Depois de termos uma ideia melhor do que é preciso ser feito e de como todas as partes se devem relacionar dentro do projecto, podemos começar a ajustar a agenda. Podemos **definir calendários** para o projecto inteiro, tarefas específicas e alocar recursos.

Podemos usar o calendário do **Projecto** (calendário base usado por um projecto) para visualizar os dias e as horas úteis gerais do projecto, além de períodos de folga (horas ou dias designados num calendário de recurso ou de projecto nos quais o **Projecto** não deve agendar tarefas porque não são períodos de trabalho). O período de folga pode incluir intervalos de almoço, fins-de-semana e feriados, por exemplo, regulares (como fins-de-semana e noites) e folgas especiais (como feriados).

Adicionar um dia de férias a um recurso

1. Clique na guia **Projecto** e, no grupo **Propriedades**, clique em **Alterar Período de Trabalho**.

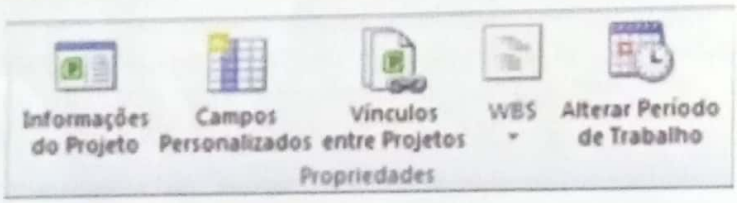


Fig. 3.28 Alterar período de trabalho.

2. Na caixa de diálogo **Alterar Período de Trabalho**, clique no recurso cujo calendário deseja alterar na lista **Para o calendário**.
3. Clique na guia **Excepções**.
4. Digite um **nome descritivo** para a excepção, como Dia de férias e as horas de início e de término para o horário em que a excepção ocorrerá.
5. Se a excepção tiver de ser repetida durante um período da agenda, clique em **Detalhes**.
6. Em **Padrão de recorrência**, seleccione a frequência entre **Diariamente e Anualmente** e seleccione **detalhes adicionais** sobre o **padrão de recorrência**.

Observação: Os detalhes do padrão de recorrência são alterados, dependendo se se quer criar um padrão diário, semanal, mensal ou anual.

7. Em **Intervalo de recorrência**, escolha a **hora de início da excepção** usando a caixa **Início** e seleccione **Termina após** ou **Termina em**.

8. Digite ou seleccione as **informações apropriadas**, com base na sua selecção de hora de término.
   - Se tiver seleccionado **Terminar após**, digite ou seleccione o **número de ocorrências da tarefa.**
   - Se tiver seleccionado **Terminar em**, digite ou seleccione **a data em que desejas que a tarefa recorrente termine**.

Adicionar um feriado ao calendário de um projecto

O *MS Project* não inclui um calendário de feriados pré-definido. Para adicionar os feriados a um projecto, especificamos um de cada vez no calendário do projecto. Se pretendes usar essa agenda de feriados para vários projectos, considera a hipótese de transformar o projecto num modelo ou de adicionar o calendário ao arquivo global. Se estiveres a usar o *Project Professional*, solicita ao teu administrador do servidor a adição do calendário ao modelo global da empresa.

1. Clique na guia **Projecto** e, no **grupo Propriedades**, clique em **Alterar Período de Trabalho**.

Fig. 3.29 Alterar período do projecto.

2. Na lista **Para o calendário**, clique no **calendário** que deseja alterar. O calendário do projecto actual é seguido por (**Calendário do Projecto**). O padrão é **Padrão** (**Calendário do Projecto**). Também é possível escolher **24 horas** ou **turno da noite**.
3. Na caixa de diálogo **Alterar Período de Trabalho**, clique na guia **Excepções**.
4. Digite um nome descritivo para a excepção, como **Feriado da Empresa** e as horas de início e de término para o horário em que a excepção ocorrerá.
5. Se a excepção tiver de ser repetida durante um período da agenda, clique em **Detalhes**.
6. Em **Padrão de recorrência**, seleccione a frequência entre **Diariamente** e **Anualmente** e seleccione detalhes adicionais sobre o padrão de recorrência. Os detalhes do padrão de recorrência serão alterados, caso queira criar um padrão diário, semanal, mensal ou anual.
7. Em **Intervalo de recorrência**, escolha a hora de **início** da excepção usando a caixa **Início** e seleccione **Termina** após ou **Termina em**.
8. Digite ou seleccione as informações apropriadas, com base na sua selecção de hora de término.
   - Se tiver seleccionado **Terminar após**, digite ou seleccione o número de ocorrências da tarefa.
   - Se tiver seleccionado **Terminar em**, digite ou seleccione a data em que deseja que a tarefa recorrente termine.

Qualquer tarefa agendada em torno do feriado será automaticamente reagendada para tomar em consideração o período de folga do feriado.

Visão geral dos calendários do *MS Project*

Calendário é o mecanismo de agendamento que determina o período de trabalho de recursos e tarefas. O *MS Project* usa quatro tipos de calendários: o **calendário base**, o **calendário do projecto**, o **calendário do recurso** e o **calendário da tarefa.**

O *MS Project* usa vários calendários para determinar a disponibilidade de um recurso e a maneira como as tarefas serão agendadas.

A disponibilidade diz respeito à questão: quando e por quanto tempo um recurso pode ser agendado para um trabalho atribuído. A disponibilidade é determinada pelos calendários do projecto e do recurso, pelas datas de início e término do recurso ou pelo nível em que o recurso estará disponível para o trabalho.

Assim, temos:

- **Calendário do projecto**: este é o calendário usado para designar a agenda de trabalho padrão para todas as tarefas de um projecto.
- **Calendário de recursos**: para cada recurso inserido, o *MS Project* criará calendários de recursos individuais baseados nas configurações do calendário Padrão. Naturalmente que podemos modificar esses calendários clicando em **Alterar Período de Trabalho** na guia **Geral** da caixa de diálogo **Informações sobre o Recurso**. Também podemos criar e atribuir calendários de recursos para recursos individuais ou grupos de recursos para indicar horas de trabalho específicas.
- **Calendário de tarefas**: as tarefas são agendadas com base nos períodos de trabalho (horas designadas num calendário de recurso ou de projecto durante as quais o trabalho pode ser realizado) no calendário do projecto. No entanto, podemos personalizar os períodos de trabalho a partir do calendário do projecto num calendário de tarefas se houver tarefas que tenham de ser concluídas em períodos diferentes, especialmente aquelas que são independentes de recursos. Um calendário de tarefas é especialmente útil para o equipamento que executa e conclui tarefas à noite e nos fins-de-semana, designados como **período de folga** no calendário do projecto.

Calendário base

É um calendário que pode ser usado como calendário de projecto e de tarefa que especifica os **períodos padrão** de trabalho e de **folga** para um conjunto de recursos. É diferente de um calendário de recurso, o qual especifica o **período de trabalho** e de **folga** de um recurso individual.

Um calendário base é usado como modelo em que o calendário do projecto, o calendário de recursos e o calendário de tarefas se baseiam. O **Project** oferece três calendários base:

- Padrão (dias da semana, das 8h às 17h, com uma hora de almoço)
- 24 horas
- Turno da Noite

Mas atenção, os calendários de período de trabalho e da disponibilidade dos recursos não se aplicam a recursos materiais, isto é, a suprimentos ou outros itens de consumo usados para executar as tarefas de um projecto.

5.ª Etapa: salvar e publicar

Periodicamente, precisamos de **salvar** o projecto para preservar as alterações feitas, fazer uma cópia de *backup* ou criar um modelo que possamos usar noutro **projecto** ou até publicar o projecto no *Project Server*.

Salvar um projecto: processo para **salvar** um projecto para usuários do *Project Standard* é diferente do que se aplica no caso dos usuários do *Project Professional* que estão conectados ao *Project Server*. Documentamos os dois processos nesta secção.

Salvar um projecto com o *Project Standard*

1. Clique em **Arquivo** e, em seguida, clique em **Salvar**.
2. Se esta for a primeira vez que está a salvar o projecto, atribui-lhe um nome na **caixa Nome do arquivo** e clique em **Salvar**.

Salvar um projecto como um modelo

Para reutilizar um projecto existente como base de um novo projecto, é possível salvá-lo como um modelo.

1. Clique em guia **Arquivo** e, em seguida, clique em **Salvar como**.
2. Usuários do *Project Professional*: se o projecto que desejas salvar como um modelo estiver salvo no *Project Server*, faça o seguinte:
   a) Na caixa de diálogo **Salvar** no *Project Server*, clique em **Salvar como Arquivo**.
   b) Na caixa de diálogo **Salvar como Arquivo**, clique em **Apenas campos** personalizados da empresa e itens do modelo global da empresa carregados na altura para salvar apenas os itens do modelo global da empresa usados nesse projecto.
3. Seleccione a unidade e a pasta em que deseje salvar o modelo.
4. Na caixa **Nome do arquivo**, digite um **nome para o modelo**.
5. Na caixa **Salvar como tipo**, clique em **Modelo** (**.mpt*) ou **Modelos do Microsoft Project 2007** (**.mpt*).
6. Clique em **Salvar**.
7. Marque as caixas de selecção dos dados que deseja remover do seu arquivo de projecto e, em seguida, clique em **Salvar**.

Trabalho de projecto - síntese

O **trabalho de projecto** é «um método de trabalho que requer a participação de cada membro de um grupo, segundo as suas capacidades, com o objectivo de realizar um trabalho conjunto, decidido, planificado e organizado de comum acordo».

O trabalho de projecto **deve ser orientado** para a **resolução de um problema**, considerado importante e real. Permite aprendizagens novas e ser estudado ou resolvido tendo em conta as condições da sociedade em que vivemos.

Duma forma geral, o trabalho de projecto desenvolve-se nas seguintes fases:

1. Escolha do tema.
2. Definição pormenorizada dos aspectos do tema que se quer trabalhar.
3. Planeamento do trabalho
4. Recolha de documentos, investigação, tomada de notas.
5. Elaboração do ponto da situação
6. Preparação da apresentação do trabalho aos pares
7. Apresentação do trabalho e participação na apresentação dos trabalhos dos outros grupos
8. Realização do balanço

E não esqueça!

A aprendizagem é um processo de descoberta em que cada um de nós deverá ser o seu próprio descobridor, coisa que os outros não poderão fazer por nós

John Dewey, 1996

Exercícios propostos

1. Quais das acções seguintes se referem a factos que se prestam a ser abordados adoptando-se uma Metodologia de Projecto? Justifique.

   a) Construir o estádio nacional de futebol.
   b) Realizar obras de reabilitação da nossa escola.
   c) Ir com os meus irmãos passar três semanas em casa da avó.
   d) Modernizar o aeroporto de Nacala.
   e) Faltar um dia à escola.
   f) Escrever um roteiro turístico da minha cidade.

2. Apresente mais três exemplos que lhe ocorram, de situações que se adequem à Metodologia do Trabalho de Projecto.
3. Apresente, de forma clara, três vantagens de trabalhar segundo a Metodologia de Projecto.
4. Apresente um exemplo de um tema que poderá tratar nas aulas e que apresente as características desejadas. Sugestão: pensa em temas que estudará no programa de outras disciplinas.
5. É importante planear e planificar o trabalho, concorda? Justifique a sua resposta, apresentando três vantagens de o fazer.
6. Na prática, que «ferramentas» se podem usar para planear um projecto?
7. Das várias possibilidades de apresentação dos resultados, quais são as que acha que lhe dariam mais prazer realizar?
8. Qual é o interesse de se efectuar o relatório do projecto?
9. Tendo em atenção o quadro seguinte, constrói o mapa de *GANTT*.

Nome-Tarefa	Duração	Precedência
A	3	-------
B	8	A
C	5	A
D	1	B
E	10	B
F	4	C
G	5	E
H	9	D
I	2	G

10. Que entende por Projecto?
11. Que diferenças significativas existem entre os *Software Planner* e *MS Project*?
12. O que significa Metodologia do Trabalho de Projecto e quais são os seus objectivos?
13. Organize um guião para usar a Metodologia do Trabalho de Projecto.
14. Seleccione um dos temas abaixo e proponha aos seus colegas trabalharem segundo a Metodologia do Trabalho de Projecto.
    Tema 1 _ Irradicação da toxicodependência na escola.
    Tema 2 _ Da escola que temos à escola que queremos - Desafios.
    Tema 3 _ Como atenuar o desemprego dos jovens no nosso país?
    Tema 4 _ Como acabar com a violência doméstica?
    Tema 5 _ A solidão no planeta global.
    Tema 6 _ A presença da *Internet* no quotidiano dos alunos.
    Tema 7 _ A poluição: uma fatalidade do mundo actual?
    Tema 8 _ Reciclar para salvar o planeta: campanha de reciclagem.

Teste I

Leia cuidadosamente o teste

Justifique convenientemente todas as respostas.

1. Uma base de dados tem alguma fonte de dados, algum grau de interacção com eventos do mundo real e uma audiência que está activamente interessada no seu conteúdo. Isso tudo é facilitado pelo SGBD.
   a) Que entende por um Sistema Gerenciador de Base de Dados?
   b) Dê exemplo de três SGBD que conhece.
   c) Indique os modelos de dados que conhece.
2. Preencha a tabela abaixo, segundo características da Abordagem de Base de Dados *versus* Processamento Tradicional de Arquivos:

Processamento Tradicional de Arquivos	Base de Dados	Vantagens da Base de Dados
Definição dos dados é parte do código de programas de aplicação.		Eliminação de redundâncias
	Capaz de permitir diversas aplicações.	Eliminação de redundâncias
		Facilidade de manutenção
Representação de dados ao nível físico.		Facilidade de manutenção
	Permite múltiplas visões.	Facilidade de consultas

3. Descreva as diferenças entre entidade e tipo de entidade.
4. Caracterize o modelo de dados relacional.
5. Qual é a estrutura fundamental numa base de dados relacionais?
6. A linguagem SQL é um padrão de linguagem de consulta comercial que usa uma combinação de construtores em Álgebra e Cálculo Relacional.
   a) Como é que se designam as operações de consulta (e actualização) de uma base de dados.
   b) Interprete o resultado da seguinte consulta em SQL:

Consulta SQL

```
SELECT DATANASC, ENDEREÇO
FROM EMPREGADO
WHERE PNOME = 'Yuri' AND MNOME = 'F' AND SNOME = 'Obri'
```

7. Suponha que a Videoteca do seu bairro lhe solicita para lhes criar uma base de dados para a gestão do aluguer dos diversos filmes que eles têm.
   a) Que informações relevantes recolheria primeiro?
   b) Que ferramenta informática escolheria para a criação dessa base de dados?

Bom trabalho

Teste II

Leia cuidadosamente o teste

Justifique convenientemente todas as respostas.

1. Diferencie um algoritmo de um programa.
2. Crie algortimos simplificados para executar cada uma das tarefas a seguir:
   a) Tirar uma fotografia.
   b) Cadastrar um cliente numa videoteca.
   c) Ler uma revista.
3. Represente um dos algoritmos acima na forma de um fluxograma.
4. Considere um algoritmo que calcule a área de um triângulo A = (b∗h)/2.
   a) Quais são os valores de entrada?
   b) Qual será o processamento do algoritmo?
   c) Qual o valor de saída?
   d) Quais são os valores que variam e quais permanecem constantes ou fixos durante a execução do algoritmo?
5. O que será impresso para cada um das instruções abaixo?
   a) n1 →4
      n2 →10
      Escreva(«n1+n2 =», n1 + n2)
   b) n1 →4
      n2 →10
      Escreva(n1,«+»,n2,«=», n1 + n2)
6. Elabore um algoritmo que leia um número e imprima uma das mensagens: é múltiplo de 3, ou, não é múltiplo de 3.
7. Construa a tabela de decisão para o algoritmo abaixo, conforme os possíveis valores de q1, q2, q3, especificando o que será impresso em cada caso no final da execução do algoritmo.

Algoritmo 1:

```
Var q1,q2,q3: logico
    Escreva «A»
     Se (q1) então
       Escreva «B»
       Escreva «C»
    Fim_se
   Se (q2) então
       Escreva «D»
    Fim_se
   Escreva «E»
       Se (q3) então
          Escreva «F»
       Senão
          Escreva «G»
      Fim_se
    Escreva «H»
```

Bom trabalho

Teste III

Leia cuidadosamente o teste

Justifique convenientemente todas as respostas.

I

1. Apresente, de forma clara, três vantagens de trabalhar segundo a Metodologia de Projecto.
2. Liste as etapas da Metodologia de Trabalho de Projecto e explique uma delas.
3. É importante planear e planificar o trabalho, não concorda? Justifique a sua resposta, apresentando três vantagens para o fazer.

II

1. Obra de reabilitação de uma sala para Laboratório de Informática.
   - Elabore o gráfico de *Gantt* do projecto definido pela tabela seguinte, utilizando a aplicação *Planner*.

Actividades	Duração	Recursos
Retirar todas as mobílias e restante recheio.	1 dia	1 carpinteiro, um ajudante, ferramentas de carpinteiro
Recolocar todas as mobílias e restante recheio.	1 dia	1 carpinteiro, um ajudante, ferramentas de carpinteiro
Substituir as molduras das janelas e portas das varandas.	2 dias	1 electricista, ferramentas de electricista
Retirar o material eléctrico.	6 horas	1 electricista, material eléctrico, ferramentas de electricista
Colocar o material eléctrico.	1 dia	1 pedreiro, ferramentas de pedreiro
Rasgar as paredes para as alterações eléctricas.	6 horas	1 electricista, material eléctrico, ferramentas de electricista
Inserir canalizações eléctricas.	6 horas	1 pedreiro, cimento, ferramentas de pedreiro
Rectificar as paredes com argamassa.	3 horas	–
Secagem da argamassa	2 dias	1 pedreiro, gesso, ferramentas de pedreiro
Acabamento da superfície das paredes	6 horas	1 carpinteiro, ferramentas de carpinteiro
Retirar rodapés e outras madeiras.	6 horas	1 carpinteiro, ferramentas de carpinteiro
Recolocar rodapés e outras madeiras.	1 dia	1 pintor, massa de reparação de paredes, ferramentas de pintura
Reparar pequenos defeitos nas paredes e tectos.	6 horas	1 pintor, tintas, ferramentas de pintura
Aplicar primeira de mão de pintura.	1 dia	1 pintor, tintas, ferramentas de pintura
Aplicar segunda de mão de pintura.	1 dia	

- Tenha em atenção que a ordem das actividades não está optimizada.
- Jogue com a possível simultaneidade de actividades, para tentar que a obra dure o mínimo de tempo possível.
- Ao definir os recursos, não se esqueça de definir cada recurso elementar de forma separada, e não agrupada, por exemplo, um carpinteiro, um ajudante de carpinteiro e ferramentas de carpinteiro são três recursos distintos.
- Define abreviaturas para os recursos, de modo a não sobrecarregar o gráfico de *Gantt*.
- Define também uma data futura de início de projecto, pois é de esperar que o projecto não se inicie logo na data em que é planeado.

Bom trabalho

Teste final

Leia cuidadosamente o teste

Justifique convenientemente todas as respostas.

I

1. A tecnologia aplicada aos métodos de armazenamento de informações vem crescendo e gerando um impacto cada vez maior no uso de computadores, em qualquer área em que os mesmos podem ser aplicados.
   a) Que entende por base de dados?
   b) Diferencie Base de Dados e Sistema de Base de Dados.
   c) Indique três áreas onde as Bases de Dados são usadas intensamente.
2. Qual é a diferença principal entre o processamento tradicional de arquivos e o conceito moderno de base de dados?
3. Indique, comentadas, três vantagens do uso de um SGBD.

II

1. Considere que se deseja desenvolver um algoritmo para calcular a quantidade de azulejos que são necessários para cobrir uma determinada parede.
   a) Quais são os valores de entrada?
   b) Qual será o processamento do algoritmo?
   c) Qual o valor de saída?
   d) Quais são os valores que variam e quais permanecem constantes ou fixos?
2. O que será impresso para cada um das instruções abaixo?
   a) n1 → 4
   n2 → 10
   Escreva (n1 + n2)
   b) n1 → 4
   n2 → 10
   Escreva b(«n1 + n2»)
3. Desenvolva um algoritmo que classifique um número de entrada fornecido pelo usuário como par ou ímpar.

III

2. É importante planear e planificar o Trabalho de Projecto, não concorda? Justifique a sua resposta, apresentando três vantagens para o fazer.
2. Qual o interesse de se efectuar o relatório do projecto?
3. Tendo em atenção o quadro seguinte, construa o mapa de *GANTT*.

Nome	Descrição	Duração	Precedências
A	Pré-estudo e encomenda	3 dias	-----
B	Preparação do espaço	6 dias	1
C	Recepção dos equipamentos	9 dias	1
D	Instalação de *software*	5 dias	1
E	Configuração e Testes	4 dias	4

Bom trabalho

Termo em Português	Em Inglês	Definição
Algoritmo	*Algorithm*	Conjunto finito de regras, bem determinadas, para a resolução de um problema, através de um número finito de operações.
Aplicações	*Application*	Programa que executa tarefas específicas que não são próprias do sistema operativo.
Analógico	*Analog*	Qualificativo de dados representados por meio de grandezas físicas que variam de modo contínuo.
Armazenamento	*Storage*	Conservação de dados num dispositivo de memória.
Autenticação		Processo que busca verificar a identidade digital do usuário de um sistema.
Backup (salvaguarda)	*Backup*	Cópia de dados de um dispositivo de armazenamento para o fim de outro a serem restaurados em caso da perda dos dados originais. No contexto de *software* usado para realizar a salvaguarda de ficheiros (*software* de salvaguarda), obtemos as chamadas «cópias de segurança» (*backup copies*). No contexto de equipamento que permita redaundância, temos por exemplo «fontes de alimentação de reserva» (*backup power supplies*) ou mesmo discos de reserva (*backup disks*).
Base de dados	*Data base*	Conjunto de dados inter-relacionados e armazenados segundo uma determinada estrutura.
Bifurcação		Ponto em que alguma coisa se divide em dois
Cláusula	*Clause*	Em certas linguagens de programação (por exemplo, PROLOG), é um dos elementos da definição de um predicado.
Compilador	*Compiler*	Programas que traduzem o código fonte de uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível.
Domínio	*Domain*	Parte de uma rede de computadores em que os recursos do processamento de dados estão sob o mesmo controlo.
Entidade	*Entity*	Do latim, *entitas*, que significa ser, existência. É algo que possui existência distinta e separada, real ou imaginária.
Ficheiro	*File*	Conjunto de *bytes* a que está associado um identificador e que é tratado como um todo.
Fluxograma	*Flowchart*	Representação gráfica de um problema, sua definição, análise e resolução, através de símbolos, que representam operações, dados, fluxos e equipamento.
Gateway	*Gateway*	Dispositivo para interligar redes diferentes.

Grafo	*Graph*	Conjunto de pontos chamados nós e de arcos unindo os nós. Os grafos dão lugar a representações desenhadas sob a forma de diagramas.
Intruso	*Hacker*	Indivíduos com conhecimento profundo de programação que elaboram e modificam *software* e *hardware* de computadores;
Informação	*Information*	Significado atribuído correntemente aos dados através das convenções a eles aplicadas.
Instância	*Instance*	Objecto que se adequa a uma descrição normalizada num certo nível.
Interface	*Interface*	Fronteira partilhada entre duas unidades funcionais, definida pelas suas características físicas comuns de interligação, características dos sinais e outras características apropriadas.
Internet	*Internet*	Rede alargada que é uma confederação de redes de computadores das instituições educacionais, governamentais e até individuais com base no protocolo TCP/IP.
Layout	*Layout*	É um esboço mostrando a distribuição física, tamanhos e pesos de elementos como texto, gráficos ou figuras num determinado espaço.
Linguagem de alto nível	*High-level language*	Linguagem de programação cujos conceitos e estruturas estão mais afastados das linguagens de assemblador e mais próximos da linguagem natural.
Linguagem de baixo nível	*Autocode*	Linguagem de programação muito próxima da linguagem-máquina na qual os códigos de operação são representados por mnemónicas e onde os endereços podem ser substituídos por etiquetas.
Linguagem máquina	*Machine language*	Linguagem artificial composta por instruções-máquina.
Modem	*Modem*	Unidade funcional que modula e desmodula sinais. É sobretudo utilizado na conversão, na porta serial de um computador, de sinais digitais em sinais analógicos modulados para envio através de uma linha telefónica analógica e vice-versa.
Palavra-senha	*Password*	Cadeia de caracteres que permite ao utilizador ter acesso parcial ou total a um sistema ou a um conjunto de dados.
Paradigma	*Paradigm*	Formato de alto nível para usar conhecimento com vista a resolver uma certa classe de problemas.
Projecto	*Project*	Empreendimento com objectivos, amplitude e duração pré-especificados.
Pseudocódigo	*Pseudocode*	Código no qual são escritas as instruções de programa, que utiliza representações simbólicas de códigos de operações e endereços e que requer a tradução para código-máquina, por meio de um compilador, antes da execução do programa.

Registo	*Record*	Conjunto de dados, ou de palavras relacionadas, tratados como um todo.
Segurança informática	*Computer system security*	Conjunto de medidas técnicas e administrativas aplicadas a um sistema de processamento de dados para proteger o equipamento, o suporte lógico e os dados contra modificações, destruição ou divulgação, quer acidental quer intencional.
Semântica	*Semantics*	Disciplina que se ocupa da significação das palavras e da evolução do seu sentido.
Sentinela	*Flag*	
Servidor	*Server*	Estação de dados que, numa rede local, proporciona serviços a outras estações de dados.
SGBD	*DBMS*	*Data Base Management System* – sistema de *software* que tem como função assegurar a gestão automatizada de uma base de dados.
Síncrono	*Synchronous*	Refere-se a dois ou mais processos cuja realização simultânea depende da ocorrência de um acontecimento específico, tal como um sinal de sincronização comum.
Sintaxe	*Syntax*	Conjunto de regras que regem o agrupamento de símbolos de uma dada linguagem.
Sistema informático	*Computer system*	Sistema composto por equipamento informático e pessoal associado, que executa funções de entrada, processamento, memorização, saída e controlo, destinadas a realizar uma sequência de operações sobre dados.
Token	*Token*	Dispositivos físicos que auxiliam o usuário quanto à segurança pessoal ao gerar uma senha temporária de protecção para as contas que ele utiliza.
Actualizar	*Upgrade*	Troca de um *hardware*, *software* ou *firmware* por uma versão melhor ou mais recente.
Utilizador final	*End user*	Pessoa, dispositivo, programa ou sistema informático que utiliza uma rede de computadores a fim de permitir o processamento de dados e a troca de informações.

Bibliografia

Campos, Luis de; Vilar, Sandro; Lucio, Levi. *Programação em VB6*, 6.ª edição, FCA – Editora Informática, 1999.

Chen, Peter. *Modelagem de dados: a abordagem entidade-relacionamento para projeto lógico*, Trad. Cecília Camargo Bartalotti, São Paulo, Makron Books, 1990.

Egypto, Cândido. *Introdução à Programação*, Apostila da ASPER, 2004.

Ferrari, F.; Cechinel, C. *Introdução a Algoritmos e Programação*, Apostila Versão 2.0, 2008.

Forja Castro, L. Ricardo. M. *Trabalho de Projecto Educativo* – Manual de formação no âmbito do projecto «Gerir o Trabalho de Projecto», Lisboa, Texto Editora, 1994.

Goodman, Erik. *Introduction to Project Management*, ECE 480.

Guerra, Isabel. *Introdução à Metodologia de Projecto* (doc. Int.), CET, Lisboa, 1994.

Guias Práticos Informática Microsoft Project, Porto Editora, 2007.

Guimarães, Ângelo de Moura. *Algoritmos e estruturas de dados*. LTC, 1985.

Heuser, Carlos Alberto. *Projeto de Banco de Dados*, 4.ª edição, Ed. Sagra, 2001.

Heuser, Carlos Alberto. *Projeto de banco de dados*. 3.ª edição, Porto Alegre: Sagra-Luzzatto, 2000 (Série Livros Didáticos; Número 4).

Kroenke, D.M. *Banco de Dado: Fundamentos, Projeto e Implementações*, 6.ª edição. Rio de Janeiro: Livros Técnicos e Científicos, 1998. 382 p.

Medina, M., Fertig, C. *Algoritmos e Programação: Teoria e Prática*, São Paulo, Novatec Editora, 2005.

Microsoft Project Made Simple, Mike Bailey, Oregon State University.

Neto, P. C., Infante, U. *Gramática da língua portuguesa*, São Paulo: Scipione, 1988.

Oliveira, A. B., Boratti, I. C. *Introdução à Programação de Algoritmos*, Florianópolis, Bookstore, 1999.

Orth, A. I. *Algoritmos e Programação com resumos das Linguagens Pascal e C*. Editora AIO, 2001

Paiva, S. R. *Algoritmos, Técnicas de Programação e Estruturas de Dados*. Apostila da ASPER, 1995.

Ponte, João Pedro e outros. *in* «projectos educativos Matemática Ensino Secundário», DES, Lisboa, 1998, pp.9-22.

Pressman, Roger S. *Software Engineering: A Practitioner's Approach*, McGraw-Hill, capítulo 7, 5.ª edição, 2001.

Ramos, Sérgio. *Introdução à Metodologia do Trabalho de Projecto*. ESMS, Aveiro – Portugal, 2007.

Rinaldi, Roberto. *Turbo Pascal 7.0: comandos e funções*. Érica, 1993.

Schimtz, Eber A.; Teles, Antonio A. S. *Pascal e técnicas de programação*. LTC, 1985.

Silberschatz, A.; Korth, H. F. ; Sudarshan. S. *Sistema de Banco de Dados*, 3.a edição, São Paulo: Makron Books, 1999. 778 p.

Tremblay, Jean-Paul; Bunt, Richard B. *Ciência dos computadores: uma abordagem algorítmica*. Mcgraw-Hill, 1983.

WebSite da cadeira de Projecto de Sistemas de Informação, do Departamento de Informática, da Faculdade de Ciências da Universidade de Lisboa (http://si.di.fc.ul.pt/psi)

Wirth, Niklaus. *Programação Sistemática em Pascal*. Campus, 1989.

Félix Singo

Pós – Doutorado em Informática pela Universidade Federal do Rio Grande do Sul – Brasil em 2012
Doutorado em Didáctica de Informática pela Universidade Técnica de Dresden – Alemanha, em 2002. Licenciado em Informática pela mesma Universidade. É, também, licenciado em Matemática pela Escola Superior de Gustrow – Alemanha.
É docente na Universidade Pedagógica, Departamento de Informática, desde 1999.
É director do Centro de Informática da UP – CIUP. É igualmente membro do Conselho Académico e Coordenador do curso de mestrado em Informática Educacional na UP.

HINO NACIONAL

Pátria Amada

Na memória de África e do Mundo
Pátria bela dos que ousaram lutar
Moçambique o teu nome é liberdade
O sol de Junho para sempre brilhará.

Coro

Moçambique nossa terra gloriosa
Pedra a pedra construindo o novo dia
Milhões de braços, uma só força
Ó pátria amada vamos vencer.

Povo unido do Rovuma ao Maputo
Colhe os frutos do combate pela Paz
Cresce o sonho ondulado na Bandeira
E vai lavrando na certeza do amanhã.

Flores brotando do chão do teu suor
Pelos montes, pelos rios, pelo mar
Nós juramos por ti, ó Moçambique
Nenhum tirano nos irá escravizar.

978-902-47-5494-6

LeYa
www.leya.co.mz
www.leyaonline.com

Texto Editores
E-mail: info@me.co.mz