9

Registros

" O mundo é um lugar perigoso para se viver, não por causa daqueles que fazem o mal, mas por causa daqueles que observam e deixam que o mal aconteça."

- Albert Einsten -

9.1- Motivação

Na vida real, existem inúmeros problemas que manipulam conjuntos cujos elementos são de tipo de dados diferentes. Por exemplo: Armazenar a avaliação final de uma turma com cinquenta alunos. Com os conhecimentos que possuímos, iremos utilizar três vectores: O primeiro para armazenar os números dos alunos, o segundo para armazenar os seus nomes e o terceiro para armazenar as suas notas. Mas, este tipo de solução, para além de ser pouco eficiente, torna os programas difíceis de ler e de compreender.

As linguagens de programação modernas, possuem um tipo de dados estruturado que permite agrupar numa única variável um conjunto de dados de tipos diferentes. Essa variável é chamada de registro e será objecto de estudo neste capítulo.

9.2- Definição de registros

Entende-se por **registro** um conjunto finito de campos de tipos diferentes agrupados num identificador. Sua sintaxe é descrita por:

```
struct  Tipo_do_registro
{
    tipo1      campo1;
    tipo2      campo2;
          .
          .
          .
    tipon      campon;
 };
```

onde:

Tipo_do_registro é um nome criado pelo programador
$tipo_i$ é um tipo elementar de dados ou tipo estruturado de dados
$campo_i$ é uma variável

Na Linguagem C os registros são chamados de ***estruturas***

Por exemplo, a informação para definir a data de aniversário de uma pessoa é descrita pelo seguinte comando:

```
struct tipdata
{
    int  dia;
    int  mes;
    int  ano;
} ;
```

e a informação para definir a matrícula de um aluno num curso universitário é

descrita pelo seguinte comando:

```
struct  tipdisciplina
{
    int     nmatricula;
    char   coddisciplina[7];
     float   nota[3];
    float   media;
};
```

A palavra reservada **struct** informa ao compilador que estamos em presença de um registro. O compilador calcula a quantidade de memória necessária para sua manipulação mas não reserva esse espaço de memória.

9.3- Declaração de registros

A declaração de um registro (estrutura em C) consiste em reservar um espaço em memória para manipular a sua informação. Este espaço é reservado com a associação do Tipo_do_registro à uma variável. Essa associação possui a seguinte sintaxe:

```
struct   Tipo_de_registro   variável;
```

Por exemplo, o comando:

```
struct  tipdata  aniversario;
```

faz referência à variável aniversario que é do tipo data, enquanto o comando

```
struct  tipdisciplina  matricula, cursos;
```

faz referência às variáveis matricula e cursos que são do tipo disciplina.

A linguagem C também permite que se faça a declaração de uma variável do tipo registro no instante de sua definição. Por exemplo:

```
struct tipdata
{
     int  dia;
     int  mes;
     int  ano;
} aniversario, natal;
```

declara que as variáveis aniversario e natal são do tipo data. Mas se necessitarmos de declarar apenas uma variável, o Tipo_de_registro deve ser omitido. Por exemplo:

```
struct
{
    int  dia;
    int  mes;
    int  ano;
} natal;
```

9.4- Visibilidade de variáveis

O conceito de visibilidade de variáveis do tipo elementar aplica-se às variáveis do tipo registro. Se definirmos um registro no interior de uma função ou de um procedimento este será apenas visível nessa função ou nesse procedimento.

Constitui uma boa práctica de programação, definir os registros como variáveis globais, dessa forma eles serão visíveis em todo o programa.

9.5- Acesso à um campo de um registro

Os campos de uma variável do tipo registro podem ser acedidos através do nome da variável seguido do operador ponto e do nome do campo. Por exemplo, o segmento de código:

```
natal.dia = 25;
natal.mes = 12;
```

armazena no campo dia da variável natal o valor 25 e no campo mes da mesma variável o valor 12, enquanto o comando:

```
printf(" %.2f ",cursos.nota[1]);
```

mostra no monitor de vídeo o conteúdo da segunda nota parcial da variável cursos.

9.6- Inicialização de registros

Vimos que inicializar uma variável consiste em prepará-la para receber dados. Se um registro possuir campos do tipo string (cadeia de caracteres), a sua inicialização consiste em armazenar no primeiro byte desses campos o caracter nulo. Por exemplo, o comando de atribuição de valor:

```
cursos.coddisciplina[0] = '\0';
```

inicializa o registro cursos. Mas se o registro possuir campos do tipo numérico, a sua inicialização consiste em armazenar zeros nesses campos. Por exemplo, o segmento de código:

```
natal.dia = 0;
natal.mes = 0;
natal.ano = 0;
```

inicializa a variável natal do tipdata.

9.7- Carregamento de dados

O carregamento de dados, consiste em preencher os campos de uma variável do tipo registro com alguma informação. Por exemplo:

```
cursos.nmatricula =  3209918;
strcpy(cursos.coddisciplina, "INF101");
cursos.nota[0] = 12.0;
cursos.nota[1] = 9.7;
cursos.nota[2] = 11.3;
```

A linguagem C também permite que uma variável seja carregada na sua declaração. Por exemplo, o comando:

```
struct tipdata data = {21,05,1959};
```

armazena na variável data do tipo data os seguintes valores:

```
data.dia = 21;
data.mes=05;
data.ano= 1959;
```

e o comando:

```
struct tipdisciplina   cursos = {0,'\0',0.0,0.0,0.0,0.0};
```

inicializa a variável cursos do tipo disciplina.

Para consolidar estes conceitos, vamos escrever um pequeno programa que carrega e imprime as notas de um determinado aluno.

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct                                  /* Definição e declaração do registro */
{
   int      nmatricula;
   char     coddisciplina[7];
   float    nota[3];
   float    media;
 } cursos;


int main()
{
  cursos.nmatricula =  3209918;                  /* Carregamento de dados   */
  strcpy(cursos.coddisciplina, "MAT105");
  cursos.nota[0] = 12.0;
  cursos.nota[1] = 9.7;
  cursos.nota[2] = 11.3;
  cursos.media  = (cursos.nota[0] + cursos,nota[1] + cursos.nota[2])/3.0;

  printf(" \n Matricula:  %d  ",cursos.nmatricula);    /* Impressão de resultados */
  printf(" \n Média     :  %.2f ",cursos.media);
  system("PAUSE");
  return 0;
}
```

9.8- Atribuição entre registros

A informação contida numa variável do tipo registro, pode ser atribuída a outra variável do mesmo tipo, se estas possuírem o mesmo número de campos e esses campos forem do mesmo tipo. Por exemplo, no segmento de código que descrevemos em seguida, carregamos na variável data um conjunto de valores que são copiados para a variável segdata do mesmo tipo.

```
#include <stdio.h>
#include <stdlib.h>

struct
{
     int  dia;
     int  mes;
     int  ano;
} data, segdata;

int main()
{

      data.dia = 12;
      data.mes= 07;
      data.ano = 1972;
```

```
        .
        .
      segdata = data;                /* atribui um registro a outro */
        .
        .
  }
```

Se depois do comando *segdata = data*; colocarmos uma instrução para imprimir o conteúdo dessa variável teremos a seguinte informação:

```
Dia : 12
Mês: 07
Ano : 1972
```

Isso quer dizer, que esse comando é equivalente aos comandos de atribuição de valor:

```
segdata.dia  = data.dia;
segdata.mes= data.mes;
segdata.ano = data.ano;
```

9.9- Comando typedef

Uma das principais desvantagens da utilização de registros, está na declaração das variáveis que têm de ser precedidas pela palavra reservada **struct**.

Para declararmos uma variável do tipo registro com a mesma sintaxe que declaramos as variáveis do tipo elementar necessitamos de utilizar a palavra reservada **typedef** que possui a seguinte sintaxe:

```
typedef  Tipo_de_variável   sinonimo;
```

Para criarmos um sinónimo de uma variável do tipo registro, utilizamos a seguinte sintaxe:

```
typedef struct
{
    int      nmatricula;
    char     coddisciplina[7];
    float    nota[3];
    float    media;
 } tipmatricula;
```

Agora podemos associar o tipo matricula às variáveis cursos e fundamentos com a mesma sintaxe que declaramos às variáveis do tipo elementar, ou seja:

```
tipmatricula  cursos, fundamentos;
```

9.10- Registro dentro de registros

Por definição, um registro pode conter variáveis do tipo elementar, vectores, ponteiros e outros registros.

Mas, para inserirmos uma variável do tipo registro numa outra variável do mesmo tipo, devemos respeitar o seguinte princípio: A variável inserida deverá ser declarada antes de ser usada. Por exemplo, dada a definição do registro:

```
typedef struct
{
    int   dia;
    int   mes;
    int   ano;
} tipdata;
```

podemos utilizar esta definição para inserir o campo data de matricula, na definição do registro do tipomatricula, ou seja:

```
typedef struct
{
   int        nmatricula;
   char       coddisciplina[7];
   tipodata   data;
   float      nota[3];
   float      media;
 } tipmatricula;
```

Para aceder a informação de um determinado mês da data de matrícula, basta utilizar a seguinte notação:

```
cursos.data.mes
```

onde cursos é uma variável do tipo tipo matricula e data uma variável do tipo data.

9.11- Vectores de registros

Suponhamos sem perda da generalidade, que pretendemos matricular cinquenta alunos na disciplina de Introdução à programação. A nossa intuição, leva-nos a pensar em definir um vector com cinquenta elementos, em que cada elemento é um registro do tipo matricula, ou seja, um **vector de registros.**

A declaração de um vector de registros é análoga a declaração de um vector cujos elementos são do tipo elementar, temos:

```
tipmatricula  fundamentos[MAX_ALUNOS];
```

onde

```
#define  MAX_ALUNOS  50
```

O acesso à um determinado elemento do vector é feito por uma variável indexada que indica a sua posição relativa, seguido do operador ponto e do nome do campo. Por exemplo, o acesso ao número de matrícula do quinto aluno é descrito pela notação:

```
fundamentos[4].nmatricula;
```

mas o acesso a segunda nota parcial do vigésimo aluno é descrito pela notação:

```
fundamentos[19].nota[1];
```

e a alteração do código da disciplina do décimo aluno para INF301 é feita pelo comando:

```
strcpy(fundamentos[9].coddisciplina,"INF301");
```

Para consolidar os conhecimentos, vamos descrever como declarar uma estrutura de dados para armazenar a informação relativa aos pilotos de uma companhia de avião. Suponhamos, sem perda da generalidade, que essa companhia possui no máximo 80 pilotos e pretendemos armazenar os seguintes dados: nome, data de nascimento, data de obtenção do brevet, categoria profissional e o nível de remuneração que inclui salários de base, subsídio de produtividade, subsídio horas extraordinárias e desconto para segurança social.

Data é uma entidade que possui os seguintes campos: dia, mês e ano. Essa entidade é definida pelo seguinte registro.

```
typedef
{
   int dia, mes, ano;
} tipodata;
```

Remuneração é uma entidade que possui os seguintes atributos: salário de base, salário de produtividade e desconto para INSS. Essa entidade é definida pelo seguinte registro:

```
typedef
{
  float salbase, produtividade, inss;
} tiposalario;
```

Piloto é uma entidade que possui os seguintes campos: nome e sobrenome. Essa entidade é definida pelo seguinte registro.

```
typedef
{
  char nome[MAX_NOME],sobrenome[MAX_NOME];
} tipopiloto;
```

Para terminar, o registro da companhia é constituído pelas seguintes entidades:

```
typedef
{
  tipopiloto  funcionario;
  tipodata datanasc, databrevet;
  tiposalario  remuneracao
  char  categoria[MAX_CAT];
 } tipoempresa;
```

Todos os pilotos da companhia são definidos por um vector do tipoempresa que está associado a um índice que indica o número de pilotos inseridos na estrutura. Seja ultpiloto esse índice.

```
tipoagenda  companhia[MAX_PILOTO];
int ultpiloto;
```

onde:

```
#define MAX_PILOTOS 80
#define MAX_NOME   40
#define MAX_NOME   10
```

9.12- Passagem de parâmetros

A passagem de parâmetros de variáveis do tipo registro são feitas de forma análoga à passagem de parâmetros de variáveis do tipo simples. Por exemplo, a função:

```
float calc_media(tipmatricula curso)
{
   return (curso.nota[0] + curso.nota[1] + curso.nota[2]) / 3.0;
}
```

recebe como argumento uma variável do tipomatricula passado por valor e devolve via comando return a média das provas parcelares. O procedimento

```
void   troca (tipomatricula *a, tipomatricula *b)
{
   tipomatricula  temp;
   temp = *a;
   *a = b*;
   *b = temp;
}
```

recebe como argumento duas variáveis do tipomatricula passados por referência e troca o seu conteúdo. O procedimento:

```
void carregar_dados(tipmatricula *mat)
{
    printf(" \n Entre com o número do aluno: ");
    scanf("%5d", &(*mat).nmatricula);
    printf("\n Entre com o código da disciplina: ");
    gets((*mat).coddisciplina);
    printf(" \n Entre com a 1ª nota parcial : ");
    scanf(" %4.2f ",&(*mat).nota[0]);
    printf("\n Entre com a 2ª nota parcial : ");
    scanf(" %4.2f ",&(*mat).nota[1]);
    printf("\n Entre com a 3ª nota parcial : ");
    scanf(" %4.2f ",&(*mat).nota[2]);
    printf("\ n Entre com o dia: ");
    scanf(" %2d ",&(*mat).data.dia);
    printf("\ n Entre com o mês: ");
    scanf(" %2d ",&(*mat).data.mes);
    printf("\ n Entre com o ano: ");
    scanf(" %2d ",&(*mat).data.ano);
}
```

recebe como argumento uma variável do tipomatricula passada por referência. Vamos estuda-lo com mais atenção. Podemos observar que as instruções de leitura, apresentam uma notação um pouco pesada. Por exemplo, no comando

```
gets((*mat).coddisciplina);
```

temos o operador indirecto e o operador ponto. Como o operador ponto tem maior precedência do que o operador indirecto essa notação pode provocar erros de compilação, se o programador esquecer-se de colocar a variável mat entre parêntesis.

Para tornar a leitura do programa mais simples e evitar esses erros, a linguagem C possui o operador seta (->). Com essa notação, a expressão anterior possui a seguinte sintaxe.

```
gets(mat->coddisciplina);
```

Vamos reescrever este procedimento com essa notação.

```
void carregar_dados(tipmatricula *mat)
{
    printf(" \n Entre com o número do aluno : ");
    scanf("%5d", &mat->nmatricula);
    printf(" \n Entre com o código da disciplina: ");
    gets(mat->coddisciplina);
    printf(" \n Entre com a 1ª nota parcial: ");
    scanf(" %4.2f ",&mat->nota[0]);
    printf(" \n Entre com a 2ª nota parcial: ");
    scanf(" %4.2f ",&mat->nota[1]);
    printf("\n Entre com a 3ª nota parcial: ");
    scanf(" %4.2f ",&(*mat).nota[2]);
    printf(" \ n Entre com o dia: ");
    scanf(" %2d ",&mat->data.dia);
    printf(" \ n Entre com o mês: ");
    scanf(" %2d ",&mat->data.mes);
    printf(" \ n Entre com o ano: ");
    scanf(" %2d ",&mat->data.ano);
}
```

9.14- Retorno de Registros

Para devolver via comando **return** um registro completo, basta declarar o tipo da função como registro. Por exemplo, desenvolver uma função para copiar os dados de um aluno, consiste em:

```
tipomatricula  copiar_dados(tipmatricula curso)
{
   tipmatricula back;
   back = curso;
   return back;
}
```

cujo protótipo é descrito por:

```
tipmatricula copiar_dados(tipmatricula curso);
```

Para consolidar os conhecimentos, vamos desenvolver uma pequena aplicação, para gerir uma agenda de telefones. Suponhamos que cada linha dessa agenda possui a seguinte informação: Número do telefone, nome e sobrenome.

A nossa primeira preocupação, consiste em definir a estrutura de dados do problema. Por recomendação, a estrutura de dados deve ser declarada como variável global. Desse modo temos:

```
typedef struct
{
  int      ntelefone;
  char   nome[10];
  char   sobrenome[10];
} tipotefefone;

tipotelefone agenda[MAX];
int numtelefones;
```

onde

```
#define  MAX    20
```

Para além dessa constante necessitaremos de definir as seguintes constantes simbólicas:

```
#define    TRUE    1
#define   FALSE  0
```

Vamos em seguida definir a função principal que fará a gestão dessa agenda. Suponhamos que essa aplicação possui um menu com as seguintes opções:

1- Inserir dados
2- Excluir dados
3- Mostrar dados
4- Sair do programa

Sua implementação é descrita pelo seguinte código:

```
int main()
{
   int processar = TRUE, opcao;
   inicializar_agenda();
   while (processar)
    {
        opcao = selecionar_menu();
        if (opcao == 1)
             carregar_dados();
        else if (opcao == 2)
                  apagar_dados();
             else if (opcao == 3)
                       imprimir_dados();
                  else if (opcao == 4)
                            processar = FALSE;
                       else
                            printf(" \n Erro: opção inválida");
    }
    system("PAUSE");
    return 0;
```

```
}
```

O procedimento inicializar_agenda() consiste em preparar o vector de registros para receber os dados. Para isso basta efectuar a seguinte operação:

```
void inicializar_agenda()
{
   numtelefones = -1;
}
```

A função selecionar_menu(), consiste em limpar a tela, mostrar o menu de opções e solicitar ao utilizador uma opção.

```
int selecionar_menu()
{
  int opcao;
  system("CLS");
  printf("   MENU DE OPÇOES ");
   printf(" \n 1-  Inserir dados");
   printf(" \n 2-   Remover dados ");
   printf(" \n 3-  Mostrar dados ");
   printf(" \n 4- Sair do programa");
  printf(" \n Entre com uma opção : ");
  scanf( " %d ", opcao);
  return opcao;
}
```

O procedimento carregar_dados() consiste em armazenar no vector os dados digitados pelo utilizador. Mas antes de efectuar essa operação, necessitamos de verificar se o vector não está cheio e se os dados satisfazem os requisitos do sistema.

```
void carregar_dados()
{
    tipotelefone aux;
    int valido;
    if (cheio())
       printf(" \n Erro: Vector está cheio ");
    else {
            aux = ler_dados_agenda();
            valido = validar_dados_agenda(aux);
            if (valido)
               armazenar_dados_agenda(aux);
            else
                printf("Erro : Dados do telefone inconsistentes );
          }
}
```

O função ler_dados_agenda() consiste em ler os dados de um telefone e devolver essa informação via comando return. Faça-o como exercício.

A função validar_dados_agenda() recebe como argumento um registro do tipotelefone e devolve via comando return o valor TRUE se os dados satisfazem os requisitos do problema e FALSE no caso contrário. Faça-o como exercício.

O procedimento armazenar_dados_agenda() recebe como argumento um registro do tipotelefone com os dados já validados. Sua finalidade consiste em movimentar o índice que faz referência ao último elemento inserido para a próxima posição e armazenar esses dados nessa posição. Faça-o como exercício.

A função cheio() devolve TRUE se todos os elementos do vector estiverem ocupados e FALSE no caso contrário. Faça-o como exercício.

O procedimento apagar_dados(), consiste em remover um registro do vector cujo número de telefone é digitado pelo utilizador. Mas, antes de efectuar essa operação, necessitamos de verificar se o vector está vazio e se o número de telefone é valido. Se essas condições forem verdadeiras, faremos uma busca até encontrar um elemento cujo número de telefone é igual ao valor digitado. Se o elemento for encontrado, procedemos a sua remoção, no caso contrário emitimos uma mensagem de erro. Esta descrição é implementada pelo seguinte procedimento.

```
void apagar_dados();
{
  int telefone, pos;
  if (vazio())
     printf(" \n Erro: Vector está vazio");
  else {
          telefone = ler_numero_telefone();
          if (valida_numero_telefone(telefone))
            {
               pos = busca(telefone);
               if (pos == -1)
                  printf(" \n Erro: Elemento não cadastrado");
               else
                   remover(pos);
            }
          else
              printf("\ n Erro: Número telefone inválidio ");
      }
 }
```

A função vazio() devolve TRUE se o vector não possui elementos e FALSE no caso contrário. Com base na função inicializar_agenda() facilmente fará a sua implementação.

A função busca(telefone) recebe como argumento o número do telefone e devolve via comando return, o valor -1 se esse número não for encontrado. No caso contrário, devolve a posição relativa do elemento. Faça-o como exercício.

O procedimento remove(pos) recebe como argumento a posição relativa do elemento a remover e procede a sua eliminação. Faça-o como exercício.

O procedimento imprimir_dados() consiste em percorrer o vector até ao último elemento inserido e mostrar na tela o conteúdo dos seus elementos.

```
void imprimir_tela()
{
   int i;
   for (i= 0; i <= ultimo; i++)
      {
         printf("\n  %s ",agenda[i].nome);
         printf( \n  %s ",agenda[i].sobrenome);
         printf(" \n %d ",agenda[i].ntelefone);
      }
}
```

Para terminar, implemente os subprogramas, monte o programa, compile e execute. Verá que a aplicação possui um erro, porque permite que se carregue o mesmo número do telefone para várias pessoas. Como exercício, proceda a sua correcção e volte a correr o programa.

9.15 - Comando Selectivo

As linguagens de programação de alto nível, possuem um comando especial que é uma generalização do comando condicional e aplica-se quando pretendemos tomar uma decisão entre múltiplas alternativas possíveis. Sua sintaxe é descrita por:

```
switch (expressão)
{
   case l1 : comandos1;
   case l2 : comandos2;
         .
         .
         .
   case ln : comandosn;
   default      : comandos;
}
```

e caracteriza-se por:

1- Calcula o valor da expressão
2- Procura o rótulo l_i ($1 \leq i \leq n$) tais que l_i = valor da expressão .

3- Se encontrou esse rótulo executa a sequencia de comandos$_i$.
4- Se não encontrou executa a sequencia de comandos associado a clausula **default** (opcional).

Em termos gerais, esse comando é equivalente ao comando condicional encadeado.

```
if (expressão == l1)
   comandos1;
else if (expressão == l2)
        comandos2;
              .
              .                    .
        else if (expressão == ln)
              comandosn;
           else
              comandos;
```

Vamos refazer a função principal do exemplo anterior. Observe como o código fica mais elegante e legível.

```
int main()
{
   int processar = TRUE, opcao;
   inicializar_agenda();
   while (processar)
    {
        opcao = selecionar_menu();
        switch (opcao)
        {
          case 1:
                  carregar_dados();
                  break;
          case 2:
                   apagar_dados();
                   break;
          case 3:
                  imprimir_dados();
                  break;
          case 4:
                  processar = FALSE;
                  break;
            default:
                     printf(" \n Erro: opção inválida");
        }

    }
    system("PAUSE");
    return 0;
}
```

Os rótulos do comando selectivo devem ser únicos, ou seja, não se pode ter o mesmo rótulo para dois casos distintos.

Como rótulos indicam apenas o início da execução de uma acção, temos de colocar no seu término o comando **break** ou o comando **return**. A inexistência de um desses comandos fará com que a execução de uma acção prossiga para o próximo caso.

Existem alguns problemas computacionais que executam a mesma acção entre várias alternativas possíveis, um exemplo trivial é devolver o número de dias de um mês. Pelo calendário, os meses têm em geral 31 ou 30 dias. Mas o mês de fevereiro tem 29 dias se o ano é bissexto e 28 no caso contrário. Essa descrição é implementada pela seguinte função:

```
int Diasdomes(int ano, int mes)
{
  switch (mes)
  {
    case 1: case 3: case 5: case 7: case 8: case 10: case 12:
            return 31;
    case 4: case 6: case 9: case 11:
            return 30;
    case 2:
            if (bissexto(ano))
               return 29;
            else
               return 28;
  }
}
```

Apesar de tornar o programa mais legível, o comando selectivo, apresenta duas limitações: A primeira reside no facto de só permitir a utilização de expressões enumeradas. A segunda, reside no facto de termos de pesquisar a lista de rótulos do inicio ao fim para determinar se existe algum rótulo cujo valor é igual ao valor da expressão.

9.15 - Exercícios Resolvidos

Problema 9.15.1: A Direcção Nacional de Estatística efectuou um estudo sobre o nível de reprovações à matemática, nas escolas secundárias da cidade de Luanda. Para realizar este estudo, foram preenchidas fichas com os seguintes dados: código da escola, nome da escola, código da localidade, nome da localidade, número de alunos matriculados, número de alunos reprovados por faltas, número de alunos reprovados por nota e alunos aprovados. Desenvolver um programa para carregar esses dados.

Resolução: Numa primeira abordagem, a solução deste problema é descrita pelos seguintes passos:

Versão 1

1- Declarar as variáveis e as constantes
2- Enquanto há dados
3- Armazenar dados

Vamos analisar numa primeira fase a estrutura de dados envolvida no problema. Como a escola é uma entidade devemos definir um registro com a sua informação.

```
typedef struct
{
  int cod_escola;
  char nome[20];
  int cod_local;
  char local[20];
} tipoescola;
```

A informação relacionada às estatística de aproveitamento são independentes da entidade escola. Elas enquadram-se numa entidade estatísticas de avaliação escolar que pode ser definida pelo seguinte registro:

```
typedef struct
{
   int  total_alunos,  total_r_faltas, total_r_notas, total_aprovad;
} tipoestat;
```

A informação de um inquérito é uma sequencia de registros do tipo escola e do tipo estatística de aproveitamento escolar, ou seja:

```
typedef struct
{
  tipescola escola;
  tipestat  avaliacao;
} tipoinquerito;
```

Para terminar a estrutura de dados, o inquérito as escolas pode ser definido como sendo um vector do tipo inquérito. Associado a esse vector temos um contador que dá-nos o número de inqueridos armazenados.

```
tipoinquerito   inquerito[MAX_INQUERITO];
int  ultimo;
```

onde

```
# define  MAX_INQUERITO;
```

Como não conhecemos o número de inquéritos a processar, necessitamos de declarar um sentinela que está associado ao número da escola.

```
#define sentinela -1
```

O próximo passo, consiste em descrever com mais detalhe as acções para armazenar os dados.

Armazenar os dados, consiste em primeiro lugar, em inicializar o vector inquérito. Em seguida, ler um inquérito e verificar a consistência de dados. Se estes forem consistentes armazena-los no vector, no caso contrário, rejeitá-los.

Em função desses pressupostos, estamos em condições de refinar a versão anterior.

Versão 2

```
1-  Declarar as constantes
2-  Declarar as variáveis
3-  Inicializar o vector
4-  Ler os dados
5-  Enquanto há dados
6-      Validar os dados
7-      Se dados são consistentes
8-          Armazenar dados
9-          Ler próximos dados
10-     No caso contrário
11-         Imprimir mensagem de erro
12-         Ler os mesmos dados
```

Pelo enunciado, não conhecemos a priori, o número de escolas inquiridas. Isso quer dizer, que esse número pode ser maior do que a quantidade de elementos do vector. Então, antes de validar e armazenar os dados, necessitamos de verificar se o vector está cheio.

Por outro lado os utilizadores cometem erros de lançamento. Um erro muito frequente é a duplicação de dados. Isso quer dizer que antes de validar o inquerido necessitamos de verificar se ele já foi carregado.

Em função desses pressupostos, estamos em condições de refinar a versão anterior.

Versão 3

```
1-  Declarar as constantes
2-  Declarar as variáveis
3-  Inicializar o índice para último elemento inserido
4-  Ler os dados do inquerido
5-  Enquanto vector não cheio e há dados
6-      Verificar a consistência dos dados
7-      Se os dados forem consistentes
```

```
8-          Procurar o inquérito
9-          Se inquérito já carregado
10-               Imprimir mensagem de erro
11-               Ler próximos dados
12-         No caso contrário
13-               Armazenar dados
14-               Ler próximos dados
15-     No caso contrário
16-         Imprimir mensagem de erro
17-         Ler o mesmo dado
```

O próximo passo consiste em formalizar as seguintes acções: Inicializar o vector, vector cheio, ler os dados, procurar o inquérito, verificar a consistência e armazenar os dados. Essas acções serão descritas como subprogramas que ficarão como exercício.

Estamos em condições de apresentar um programa que é a solução do nosso problema:

```
/*-----------------------------------------------------------------------------------------------
Objectivo: Carregar os dados de um inquérito sobre a reprovação á matemática nas escolas publicas
-----------------------------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include  <string.h>
#include <locale.h>
#define  MAX 100
#define  SENTINELA -1
#define  TRUE   1
#define  FALSE  0
#define  NULO  '\0'

/*-----------------------------------------------------------------------------------------------
Declaração da estrutura de dados
 -----------------------------------------------------------------------------------------------*/
typedef struct
{
   int cod_escola;
   char nome[20];
   int cod_local;
   char local[20];
} tipoescola;

typedef struct
{
    int  tot_alunos,  tot_rep_faltas, tot_rep_notas, tot_aprovad;
} tipoestat;
```

```
typedef struct
 {
    tipoescola escola;
    tipoestat  avaliacao;
 } tipoinquerito;

tipoinquerito   inquerito[MAX_INQUERITO];
int  ultimo;


/*---------------------------------------------------------------------------------------------------
Prototipo de funções
 ---------------------------------------------------------------------------------------------------*/
void inicializar_vector();
int cheio();
int busca(int codigo);
void gravar_dados();
void carregar_dados(tipoinquerito *buffer);
int ver_consistencia(tipoinquerito buffer);

/*---------------------------------------------------------------------------------------------------
Objectivo : Inicializar o vector de estatísticas de inquérito
Valor de retorno : Zeros
 ---------------------------------------------------------------------------------------------------*/
void inicializar_vector()
{
   ultimo = -1;
}

/*---------------------------------------------------------------------------------------------------
Objectivo : Verificar se o vector está cheio
Valor de retorno: Verdadeiro ou Falso
---------------------------------------------------------------------------------------------------*/
int cheio();
{
   return  ultimo == MAX_INQUERITO-1;
}

/*---------------------------------------------------------------------------------------------------
Objectivo : Procurar uma escola no vector de inquérito
Parâmetros por valor: código da escola
Valor de retorno: -1 se não encontrou ou índice do inquérito no vector
---------------------------------------------------------------------------------------------------*/
int busca(int codigo)
{
  int ind;
  for (ind = 0; ind <= ultimo; ind ++)
       if (codigo == inquerito[ind].escola.cod_escola)  return ind;
  return -1;
}
```

```
/*-------------------------------------------------------------------------------------------------
Objectivo : Armazenar um inquérito no vector de estatísticas
Parâmetros por Valor : Buffer temporário
-------------------------------------------------------------------------------------------------*/
void gravar_dados(tipoinquerito buffer)
{
    ultimo++;
    inquerito[ultimo] = buffer;
}

/*-------------------------------------------------------------------------------------------------
Objectivo : Ler os dados de um inquérito
-------------------------------------------------------------------------------------------------*/
void  ler_dados(tipoinquerito *buffer)
{
   printf("\n Código da escola : ");
   scanf(" %d ", &buffer->escola.cod_escola);
   printf("\n Nome da escola :");
   gets(buffer->escola.nome);
   printf(" Código da localização :");
   scanf(" %d ",&buffer->escola.cod_local);
   printf("\n Nome da localização :");
   gets(buffer->escola.local);
   printf(" Total de alunos :");
   scanf(" %d ", &buffer->avaliacao.tot_alunos);
   printf(" \n  Total de reprovados por faltas: ");
   scanf(" %d ",&buffer->avaliacao.tot_rep_faltas);
   printf(" \n Total reprovados por notas : ");
   scanf(" %d ",&buffer->avaliacao.tot_rep_notas);
   printf("\n Total de alunos aprovados ");
   scanf("%d ",&buffer->avaliacao.tot_aprovad);
}

/*-------------------------------------------------------------------------------------------------
Objectivo : Limpar o buffer temporário
Parâmetros por referência: buffer temporário
-------------------------------------------------------------------------------------------------*/
void  limpa_buffer(tipoinquerito *buffer)
{
   buffer->escola.cod_escola = ZERO;
   buffer->escola.local = ZEROS;
   buffer->avaliacao.tot_alunos = ZERO;
   buffer->avaliacao.tot_rep_faltas = ZERO;
   buffer->avaliacao.tot_rep_notas = ZERO;
   buffer->avaliacao.tot_aprovad = ZERO;
   strcpy(buffer->escola.nome,NULO);
   buffer->escola.local = NULO;
}
```

```
/*-------------------------------------------------------------------------------------------------
Objectivo : Verificar se os dados do inquérito são consistentes
Parâmetros por valor: Buffer temporário
Valor de retorno: Verdadeiro ou Falso
-------------------------------------------------------------------------------------------------*/
int ver_consistencia(tipoinquerito buffer)
{
   int consistente= TRUE;  float total;
   if (buffer.escola.cod_escola < ZERO)
       {
         printf("Erro: Código da escola inválido");
         consistente = FALSE;
       }
    if (buffer.escola.cod_local < ZEROS)
       {
         printf("Erro: Código da localização da escola inválida");
         consistente = FALSE;
       }
   if (buffer.avaliacao.tot_alunos < ZERO)
       {
          printf("Erro: total de alunos invalido");
         consistente = FALSE;
       }
   if  (buffer.avaliacao.tot_rep_faltas < ZERO)
       {
          printf("Erro:Total alunos reprovados por faltas inválido");
          consistente =  FALSE;
       }
   if (buffer.avaliacao.tot_rep_notas < ZERO)
       {
         printf("Erro:Total alunos reprovados por notas inválido");
         consistente = FALSE;
       }
   if (buffer.avaliacao.tot_aprovad < ZERO)
     {
       printf("Erro:Total alunos aprovados inválido");
       consistente = FALSE;
     }
   if (strlen(buffer.escola.nome) == ZEROS)
     {
       printf("Erro: Nome da escola inválido");
       consistente = FALSE;
     }
   if (strlen(buffer.escola.local) == ZEROS)
     {
       printf("Erro: Localização da escola inválida");
       consistente = FALSE;
     }
   return consistente;
}
```

```
/*-------------------------------------------------------------------------------------------------
Função principal
-------------------------------------------------------------------------------------------------*/
int main()
{
  tipoinquerito bufescola;
  system("CLS");
  setlocale(LC_ALL, "Portuguese");
  inicializar_vector();
  limpa_buffer(&bufescola);
  printf("\n Entre com os dados do inquérito ');
  ler_dados(&bufescola);
  while ((!cheio())  && (bufescola.escola.cod_escola != SENTINELA))
    {
       if (ver_consistencia(bufescola))
        {
            if (busca(bufescola.escola.cod_escola) != -1)
               {
                  printf("Erro: Ficha já foi lançada");
                  printf(" Entre com os dados da mesma escola ");
               }
            else
                {
                  gravar_dados(bufescola);
                  printf(" Entre com os dados da próxima escola ");
                }
            limpa_buffer(bufescola);
        }
       else
          {
             limpa_buffer(bufescola);
             printf("Entre com os dados da mesma escola");
          }
       system("CLS");
       carregar_dados(&bufescola);
    }
  if (bufescola.escola.cod_escola <> SENTINELA)
       printf(" Carregamento terminou com sucesso");
  else
       printf("Erro: vector cheio");
  system("PAUSE");
  return 0;
}
```

Quanto mais exaustivos forem os testes de validação maior será a consistência de dados armazenada no vector. É da responsabilidade do programador desenvolver esses testes.

Muitos autores aconselham que se faça uma limpeza ao buffer antes de qualquer processo de leitura. Esta recomendação não constitui uma má prática de programação. É um excesso de zelo.

Problema 9.15.2: Um Banco comercial, possui um cadastro de clientes com os seguintes campos: número da conta, nome do cliente e saldo. Desenvolva um programa para imprimir esse cadastro em ordem crescente de saldos.

Resolução: Vamos estudar um método muito simples de ordenação, denominado por **ordenação por selecção.**

Iniciamos o método com uma busca para encontrar o menor elemento do vector. Ao terminar essa busca, copiamos o menor elemento para à primeira posição de um vector auxiliar. Alteramos o conteúdo do vector original, para um valor especial. Suponhamos que esse vector seja do tipo inteiro e consideramos como valor especial o número 9999. Executamos em seguida, uma busca no vector original para encontrar o segundo menor elemento. Mas, nesse processo de busca, ignoramos o elemento cujo valor é igual à 9999. Ao terminar a busca, copiamos o segundo menor elemento para à segunda posição do vector auxiliar e alteramos o conteúdo no vector original para 9999. Realizamos este processo até que todos os elementos do vector original tenha sido colocados no vector auxiliar na ordem crescente.

Mas, esta abordagem apresenta as seguintes insuficiências: A primeira deve-se ao facto de necessitarmos de um vector adicional. A segunda deve-se ao facto de efectuarmos a busca a todo vector para procurarmos o menor elemento. A terceira e última deve-se ao facto, de necessitarmos de um valor especial. Se o vector possuir elementos cujo valor é igual ao valor especial, este método não funciona.

Veremos em seguida, uma versão melhorada deste método que elimina essas deficiências.

Selecionamos, numa primeira fase, o primeiro elemento do vector e consideramos que esse elemento é o menor. Percorremos em seguida, os restantes elementos e comparamos o seus conteúdos ao elemento que assumimos como menor. Se encontrarmos algum elemento menor do que o elemento que assumimos como menor, efectuamos a sua troca. Com essa operação, temos todas às garantias, que o elemento que está na primeira posição é o menor. Selecionamos, numa segunda fase, o segundo elemento e consideramos que esse elemento é o menor. Percorremos os restantes elementos do vector e comparamos os seus conteúdos ao elemento que assumimos como menor. Se encontrarmos algum elemento menor do que o elemento que assumimos como menor, efectuamos a sua troca. Com essa operação os dois primeiros elementos do vector estão ordenados. Repete-se o processo, até que todos os elementos tenham sido ordenados.

Em termos formais, a ordenação dos elementos de um vector com esse método, pode ser descrita pelos seguintes passos:

Versão 1

1- Para i que varia de 1 até ao ultimo elemento inserido
2- Armazenar na variável min o valor de i.
3- Para j que varia de i + 1 até ao último elemento inserido
4- Se elemento indexado por j > elemento indexado por min
5- Armazenar na variável mim o valor j.
6- Trocar elemento indexado por i pelo elemento indexado por min

Agora, vamos analisar a estrutura de dados envolvida no problema. Qualquer cliente de um banco possui os seguintes atributos: número, nome e saldo. Esses atributos fazem parte da entidade cliente. Logo, podemos declarar o seguinte registro:

```
typedef struct
{
   int    numero;
   char nome[45];
   float saldo;
} tipocli;
```

Os clientes do banco, podem ser definidos como um vector do tipo cliente com MAX_CLI elementos. Associado a esse vector, temos um contador dá-nos o número de clientes inseridos nessa estrutura.

```
tipocli   banco[MAX_CLI];
int ultimo;
```

onde:

```
#define MAX_CLI 50
```

Como entidade de saída, teremos esse vector ordenado por ordem crescente de saldos.

Como já formalizamos um método de ordenação e como as restantes acções são fáceis de implementar, apresentamos em seguida, um programa que é a solução do nosso problema:

```
/*-----------------------------------------------------------------------------------------------
Objectivo: Imprimir o cadastro de clientes de um banco, ordenados por ordem
           crescente de saldos.
-----------------------------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include  <string.h>
#include <locale.h>
#define   MAX_CLI 50
```

```
/*-------------------------------------------------------------------------------------------------
Definição de variáveis globais
-------------------------------------------------------------------------------------------------*/

 typedef struct
  {
      int    numero;
      char nome[45];
      float saldo;
   } tipocli;

tipocli   banco[MAX_CLI];
int       ultimo;

/*-------------------------------------------------------------------------------------------------
Protótipo de funções
-------------------------------------------------------------------------------------------------*/
int   carregar_dados();
int  cheio_vector();
int  validar_dados(tipocli buffer);
void  inicializar_vector();
void  ordenar();
void  imprimir();

/*-------------------------------------------------------------------------------------------------
Objectivo : Inicialização do vector
-------------------------------------------------------------------------------------------------*/
void  inicializar_vector()
{
     ultimo = -1;
}

/*-------------------------------------------------------------------------------------------------
Objectivo : Validar dados
-------------------------------------------------------------------------------------------------*/
int  validar_dados(tipocli buffer)

{
     /* Faça como exercício  /*

}

/*-------------------------------------------------------------------------------------------------
Objectivo : Limpar buffer de leitura e gravação
-------------------------------------------------------------------------------------------------*/
void  limpar_buffer(tipocli *buffer)

{
     /* Faça como exercício  /*
}
```

```
/*-------------------------------------------------------------------------------------------------
Objectivo : carregar dados no vector
-------------------------------------------------------------------------------------------------*/
int carregar_dados()
{
     /* Faça como exercício  /*
}

/*-------------------------------------------------------------------------------------------------
Objectivo : Verificar se o vector está cheio
-------------------------------------------------------------------------------------------------*/
int cheio_vector()
{
     /* Faça como exercício  /*
}

/*-------------------------------------------------------------------------------------------------
Objectivo : Ordenar um vector pelo método de selecção
-------------------------------------------------------------------------------------------------*/
void ordenar()
{
   int i, j;
   float min;
   tipocli  aux;
   for (i=0; i <= ultimo-1; i++)
     {
        min = i;
        for (j = i+1; j <= ultimo; j++)
          if ( banco[j].saldo >  banco[min].saldo )
                min = j;
        if (min != i)
          {
             aux = banco[i];
             banco[i] = banco[min];
             banco[min] = aux;
          }
     }
}

/*-------------------------------------------------------------------------------------------------
Objectivo : Imprimir resultados
-------------------------------------------------------------------------------------------------*/
void imprimir()
{
    int i;
    printf(" \ n Listagem de saldos de clientes  \n");
    for (i=0; i <= ultimo; i++)
      printf(" %d   %s   %f   \n ",banco[i].numero,  banco[i].nome,  banco[i].saldo);
}
```

```
/*-------------------------------------------------------------------------------------------------
Função Principal
-------------------------------------------------------------------------------------------------*/
int main()
{
  setlocale(LC_ALL, "Portuguese");
  system("CLS");
  inicializar_vector();
  if (carregar_dados());
    {
      ordenar();
      imprimir();
    }
  else
       printf("Erro: Carregamento de dados inválido");
  system("PAUSE");
  return 0;
}
```

Vamos analisar o funcionamento deste programa com uma simulação. Suponhamos, sem perda da generalidade, que foram carregados os seguintes saldos no vector.

{73, 65,52,24,83,17,35,96,41,9}

Não ordenado		Processo de ordenação							Ordenado	
Indice	**Saldos**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	***73***	9	9	9	9	9	9	9	9	*9*
2	*65*	**65**	17	17	17	17	17	17	17	*17*
3	*52*	52	**52**	24	24	24	24	24	24	*24*
4	*24*	24	**24**	**52**	35	35	35	35	35	*35*
5	*83*	83	83	83	**83**	41	41	41	41	*41*
6	*17*	**17**	65	65	65	**65**	52	52	52	*52*
7	*35*	35	35	**35**	52	**52**	65	65	65	*65*
8	*96*	96	96	96	96	96	96	**96**	73	*73*
9	*41*	41	41	41	**41**	83	83	83	**83**	*83*
10	***9***	73	73	73	73	73	73	**73**	96	*96*

É importante salientar, que este é o metodo de ordenação mais simples e mais comum.

Nota: *Qualquer campo de um registo sobre o qual existe uma regra de formação bem definida pode ser utilizado como chave. Normalmente, as chaves são campos numéricos ou alfanuméricos que têm uma identificação única. Por exemplo, a identificação de um aluno de uma determinada universidade, é feita pelo código do aluno, a identificação de um cidadão é feita pelo seu bilhete de identidade. Esses campos são chaves para esses conjuntos de dados.*

9.16 - Exercícios Propostos

9.16.1- Defina uma estrutura de dados para armazenar a informação de um cadastro de empregados de uma empresa. Cada empregado é identificado pelos seguintes dados: número do empregado, nome do empregado, endereço que é constituído por: (rua, número, apartamento, telefone fixo e telefone móvel), situação militar, documentos que é constituído por (número da carta de condução, número do registro militar, número de contribuinte, número da conta bancária e tipo de grupo sanguíneo), estado civil e salário. Suponha que essa empresa possui no máximo 120 empregados.

9.16.2- Defina uma estrutura de dados para armazenar a informação sobre um catálogo de CDs de uma loja. Pretende-se organizar esse catálogo por género musical. Cada género contém a seguinte informação: código e nome do género. Entendemos por género musical: Gazz, Pop, Semba, Samba, Rock, etc. A loja, possui no máximo 4000 CDs. Cada CD possui a seguinte informação: código do CD, nome do autor, titulo do CD, código da editora, data de lançamento, código do género musical, data da última aquisição, data da última venda e número de unidades disponíveis.

9.16.3- Defina uma estrutura de dados para armazenar a informação de um concurso de ginástica desportiva masculina. O concurso possui no máximo 150 concorrentes. Para cada concorrente deve-se registar a seguinte informação: número de inscrição, nome, código da nacionalidade, classificação final e uma descrição das seis provas do concurso. A descrição de cada prova possui a seguinte informação: código da prova, nome da prova e pontuação obtida.

9.16.4- Defina uma estrutura de dados para armazenar a informação de uma empresa do sector mobiliário. Esta empresa possui no máximo 40 estabelecimentos comerciais. Para cada estabelecimento pretende-se registar a seguinte informação: código do estabelecimento, localização, data de abertura, número de empregados, número de moveis de tipos diferentes e descrição dos mesmos. Na descrição de cada móvel deve-se incluir a designação do móvel, o tipo de madeira utilizado na sua construção, a quantidade existente, a data da última compra e o preço de venda em Kwanzas. Considere que existem quatro tipos de madeira: Pinho. Mogno, Carvalho e a Nogueira. Admita que em cada estabelecimento há no máximo 50 tipos diferentes de móveis.

9.16.5- Defina uma estrutura de dados para armazenar a informação de um empresa ligada ao sector de entrega de encomendas. Suponhamos que a empresa possui no máximo 50 viaturas. Pretende-se registar para cada viatura a sua marca, data de compra, carga máxima em quilos, número de serviços efectuados e a descrição dos últimos 30 serviços. A descrição de cada serviço deve incluir a data em que foi efectuado, o preço cobrado, a descrição da mercadoria transportada e o nome do cliente. Vamos considerar que temos três tipos de encomendas: Rápidas, Normal e a Frágil.

9.16.6- Um cadenciario é uma ficha utilizada em armazéns para controlar o stock de mercadorias num determinado período, normalmente mensal. Essa

ficha é constituída pelos seguintes campos: Código do artigo, nome, classificação da mercadoria, stock inicial e movimentos mensais. A classificação da mercadoria é definida pela secção e família que são cadeias de caracteres, enquanto os movimentos mensais são constituídos por: data, quantidade entrada e quantidade saída. O campo data é constituído por dia, mês e ano. Suponha que o armazém possui cinco mil artigos.

9.16.7- Suponhamos que o leitor tenha conseguido um estágio na Secretaria acadêmica da sua Universidade. Como primeira tarefa, a dona Carla, solicitou um programa que tem como entrada uma lista de alunos com a seguinte informação: número do estudante, nome, curso, turma, notas das provas de avaliação, nota média dos projectos. Este programa deve funcionar com o seguinte menu:

1- Incluir alunos
2- Alterar a informação do aluno
3- Calcular a nota final
4- Imprimir os alunos por ordem decrescente de numérico
5- Imprimir os alunos por ordem crescente de nome
6- Sair do programa

9.16.8- O Departamento de recursos humanos de uma empresa necessita de um programa para colocar no jornal mural, os funcionários cuja data de aniversário coincide com esse dia. Para realizar essa tarefa, existe um ficheiro manual que tem de ser carregado com a seguinte informação: código do trabalhador, nome do trabalhador, área de trabalho, cargo e data de nascimento (ddmmaaaa). Com base na data de nascimento e na data do dia, imprimir a relação de trabalhadores que satisfaz esse requisito.

9.16.9- Numa determinada cidade vários proprietários de imóveis estão em atrasos com o pagamento do imposto de condomínio. Desenvolva um programa com subprogramas que possui o seguinte menu:

1- Cadastrar de imóveis
2- Calcular o imposto
3- Imprimir resultados
4- Sair do programa

O cadastro de imóveis consiste em incluir, alterar ou remover a informação relativa a cada imóvel. Para efeitos de consistência de informação, não podemos ter dois imóveis com o mesmo código. Cada imóvel possui seguinte informação:

- Código de identificação
- Endereço do imóvel
- Valor do imposto
- Número de meses em atraso

O valor das multas deve ser calculado com base na seguinte tabela:

Valor do imposto	% por mês de atraso
até 1.500,00 Kz	2
de 1.501,00 Kz até 5.000,00 Kz	4.5
de 5.001,00 Kz até 10.500,00 Kz	5.2
de 10.501,00 Kz até 17.000,00 Kz.	7.5
acima de 17.001,00 Kz	10.4

A impressão de resultados consiste numa listagem ordenado por código de imóvel com os seguintes campos: código do imóvel, valor do imposto, meses em atraso e multa a pagar.

9.16.10- Uma firma de exportação possui um arquivo manual com os dados do stock de mercadorias. Cada registro contém a seguinte informação: código do artigo, nome do artigo, nome do cliente, preço de venda por unidade. Estes registros estão organizados por ordem crescente de código de artigo. Desenvolva um programa com subprogramas para encontrar um determinado artigo dado o seu código.

9.16.11- Um provedor de acesso a Internet necessita de uma aplicação para proceder a cobrança dos seus serviços aos seus clientes. Este aplicação deve funcionar com o seguinte menu:

1- Cadastro de Clientes
2- Lançamento das horas de utilização
3- Cálculo da subscrição mensal
4- Impressão de resultados
5- Sair do programa

O cadastro dos clientes consiste na inclusão, alteração e remoção de um cliente. Não se pode alterar nem remover clientes que não tenham sido inseridos. Também se pode inserir mais do que um clientes com o mesmo código. Para cada cliente devemos armazenar a seguinte informação:

e-mail
Nome do cliente
Número de horas de acesso
Página (S = sim, N = não)
Data do contracto
Ultimo mês pago

O lançamento das horas de utilização consiste em armazenar a seguinte informação:

e-mail
Número de horas utilizadas
Mês da pagar

Durante o lançamento das horas a pagar, não se pode cobrar horas que já foram pagas pelo cliente.

O cálculo do valor da subscrição obedece as seguintes condições: Até 35 horas os clientes devem pagar uma tarifa de 4.500,00 Kz. As horas excedentes devem ser taxadas à 350,00 Kz por hora. Os clientes com páginas devem pagar uma taxa adicional de 2.500,00 Kz.

A impressão de resultados consiste em imprimir uma factura, ordenada por cliente com os seguintes campos: nome do cliente, e-mail, valor da tarifa, valor das horas excedentes, valor página e por fim o valor a pagar.

9.16.12- Para participar numa jornada de computação os alunos têm de pagar uma taxa para assistir cada palestra. No acto de inscrição os alunos preenchem o seguinte formulário: número do aluno, número de palestras, Sócio (0 = não 1 = sim). Existem alunos que são sócios da Sociedade Angolana de Computação (SAC). O valor a pagar no acto de inscrição obedece ao seguinte critério:

Número de palestras	Valor a pagar
1	250.00
2	500.00
3	750.00
Outros	100.00

Para os sócios da SAC há um desconto de 35%. Desenvolva um programa com subprogramas que calcule o valor arrecadado com o evento e quantos alunos matricularam-se em cada palestra.

9.16.13- O Ministério da Justiça possui um sistema com o cadastro dos cidadãos. Cada cidadão é identificado pelos seguintes dados: Número do Bilhete de Identificação, nome completo, data nascimento e sexo. Esses dados estão armazenados num vector. Desenvolva um procedimento para separar esse vector em dois: Um com os cidadãos de sexo masculino e outro com os cidadãos de sexo feminino.

9.16.14- O Departamento de trânsito da cidade de Luanda compilou informações sobre o excesso de velocidade num determinado intervalo de tempo. Para melhorar o controlo a cidade foi organizada 16 municípios. Deseja-se as estatísticas de violação por município. Para cada violação é preparado um lançamento com a seguinte informação: matrícula do carro, código do município, velocidade limite do município e velocidade do carro. Preparar um programa com subprogramas para produzir três relatórios: O primeiro, é uma lista das multas aplicadas, em que cada multa é calculada como a soma dos custos do tribunal Kz 25.000,00 mais uma percentagem por quilómetro em excesso que é determinada pela seguinte tabela:

Quilómetros em excesso	Percentagem por velocidade excessiva
1 a 10	10 %
11 á 20	20 %
21 à 30	30 %

31 à 40	40 %
41 à 50	50 %
51 à 60	60 %
61 à 70	70 %
71 à 80	80 %
81 à 90	90 %
91 à 100	100%

Vamos supor que não existem carros cuja velocidade em excesso seja superior a 100 quilómetros por hora.

O segundo é uma análise de violações por município. Para cada município imprimir o número de violações, a multa média o valor total de multas acumuladas, a multa máxima e a multa mínima.

O terceiro é uma análise de violações na cidade. Imprimir o total de violações, a multa média, o valor acumulado de multas, a multa máxima e a multa mínima.

9.16.15- Pretende-se lançar um novo perfume no mercado. Para analisar o grau de recepsão desse produto, efectuou-se um inquérito há um conjunto indeterminado de pessoas de ambos os sexos. Nesse inquérito essas pessoas assinalavam com um x a percepção que tiveram sobre o aroma do produto. Essa percepção possui a seguinte escala:

1- Desagrado intenso
2- Desagrado moderado
3- Desagrado fraco
4- Neutro
5- Agrado fraco
6- Agrado moderado
7- Agrado intenso

Desenvolva um programa com subprogramas que possui o seguinte menu:

1- Inserção do inquérito
2- Impressão de resultados
3- Sair do programa.

A inserção do inquérito consiste em armazenar a seguinte informação: Idade, sexo, percepção do aroma e nível de escolaridade do inquirido. O nível de escolaridade pode ser: 1- Superior, 2- Médio e 3- básico.

A impressão dos resultados consiste em imprimir as listagens: percepção do artigo por nível de escolaridade; percepção do artigo por sexo; percepção do artigo por nível de escolaridade e sexo. Os resultados devem estar ordenados por ordem decrescente.

9.16.16- Uma universidade deseja fazer um levantamento sobre o seu concurso de admissão. Para cada matrícula é fornecida a seguinte informação: Número do candidato, nome do candidato, sexo e código do curso. Desenvolva um programa com menus que permita: inserir as matrículas; inserir as

informações sobre os cursos; Imprimir por curso o número de candidatos com percentagem por sexos; Imprimir o curso com maior e o menor número de alunos por vagas.

9.16.17- A Direcção de vendas de uma empresa de matérias de construção, necessita de controlar as vendas mensais e anuais dos seus vendedores. O gerente dessa área, necessita de uma aplicação com menus que permita inserir os vendedores, inserir as suas vendas e analisar as seguintes informações: vendas por vendedor num determinado período; total de vendas por vendedor durante um período e vendedores ordenados por maior nível de vendas no período.

9.16.18- O Instituto Nacional de Estatística fez um inquérito há alguns habitantes do município da Samba. Para efectuar esse inquérito os moradores desse bairro preencheram uma ficha com a seguinte informação: idade, sexo, número de filhos e salário. Desenvolva um programa com subprogramas para imprimir a seguinte informação: Média de salário da população inquirida; Média do número de filhos; Percentagem de mulheres com salário superior à 45.000,00 Kz.

9.16.19- Uma papelaria possui um conjuntos de fichas com a informação de seus produtos. Cada ficha possui os seguintes campos: código do artigo, descrição, data da ultima aquisição, preço unitário e quantidade em stock. Suponha que essa papelaria possui 100 artigos diferentes. Desenvolva um programa para carregar esses produtos, alterar a quantidade de stock de um determinado produto e remover um produto desse cadastro.

9.16.20- A contabilidade de uma empresa necessita de uma aplicação para controlar as contas a receber de seus clientes. Cada conta possui a seguinte informação: número do documento, código do cliente, data de vencimento, data de pagamento, valor da conta e juros. Desenvolva um programa com subprogramas para carregar os clientes e seus pagamentos. Se a data do pagamento for superior a data de vencimento o programa deve calcular uma taxa de juros de 0.45 % por dia. Imprimir o diário de recebimentos que mostra os pagamentos e seus juros. No fim desse diário temos uma linha com totais.

9.16.21- O Hospital Municipal de Luanda, necessita de uma agenda para controlar a marcação de consultas dos seus paciêntes. Esse hospital atende as seguintes especialidades: Clinica Geral, Genecologia, Urologia e Cardiologia.

O seu software deve possuir o seguinte menu:

1- Cadastro das especialidades
2- Cadastro de médicos
3- Cadastro de pacientes
4- Marcação de consultas
5- Impressão de resultados
6- Sair da aplicação

O cadastro das especialidades consiste na inclusão, alteração, remoção e visualização dos dados das especialidades que o hospital atende. O registro de qualquer especialidade é constituído pelos seguintes campos:

Código da especialidade
Nome da especialidade
Sexo pacientes da especialidade (1=Ambos, 2=Homens, 3= mulheres)

Para efeitos de consistência as especialidades devem ter os seguintes códigos: 1= Clinica geral, 2= Genecologia, 3= Urologia 4= Cardiologia.

É importante observar que a especialidade de genecologia só se atende mulheres enquanto a especialidade de urologia só se atende homens.

O cadastro de médicos consiste na inclusão, alteração, remoção e visualização dos dados dos médicos. O registro de qualquer médico é constituído pelos seguintes campos:

Código do médico
Nome do médico
Sobrenome do medico
Número de telefone
Endereço
Código da especialidade
Número de pacientes agendados

Para efeitos de consistência não se pode armazenar mais do que um médico com o mesmo código.

Não se pode eliminar um médico com pacientes agendados, nem uma especialidade com um médico cadastrado.

Para marcar qualquer consulta o pacientes devem estar cadastrados. O cadastro de pacientes consiste na inclusão, alteração, remoção e visualização dos dados de um paciente. O registro de qualquer paciente é constituído pelos seguintes campos:

Código do paciente
Nome do paciente
Sobrenome do paciente
Sexo (2= Homem 3= Mulher)
Idade
Número de telefone
Endereço

Não se pode eliminar um paciente com uma consulta marcada, nem armazenar mais do que um paciente com o mesmo código.

A marcação de consultas consiste em agendar uma consulta para uma determinada data. Para realizar esse processo é necessário inserir os seguintes campos:

Código do paciente
Tipo de consulta
Código do médico
Data da consulta

Para garantir a qualidade da instituição a administração fixou que um médico não pode ter mais do que 10 pacientes agendados. É evidente que um paciente não pode marcar um consulta para mais de um médico, ou para um médico que não cadastrado.

A impressão de resultados consiste em: Mostrar todas as consultas realizadas por um determinado médico ordenado por forma ascendente de dada e a relação de médicos ordenada por número de pacientes.

9.16.22- Um determinado colégio necessita de uma aplicação para gerir os seus estudantes. Essa aplicação deverá possuir o seguinte menu:

1- Cadastro dos cursos
2- Cadastro das turmas
3- Cadastro de alunos
4- Cadastro de disciplinas
5- Lançamento das notas
6- Impressão de resultados
7- Sair do programa

O cadastro de cursos consiste na inclusão, alteração, remoção e visualização dos dados de um curso. O registro de qualquer curso é constituído pelos campos:

Código do curso
Nome do curso
Numero de semestres

O cadastro das turmas consiste na inclusão, alteração, remoção e visualização dos dados de uma turna. O registro de qualquer turma é constituído pelos campos:
Código do curso
Código da turma
Nome da turma

O cadastro de estudantes consiste na inclusão, alteração, remoção e visualização dos dados de um estudante. O registro de qualquer estudante é constituído pelos campos:

Código do estudante
Nome do estudante

Endereço do estudante
Número do telefone
Código do curso
Data de matricula

O cadastro das disciplinas consiste na inclusão, alteração, remoção e visualização das disciplinas de um curso. O registro de qualquer disciplina é constituído pelos campos:

Código da disciplinas
Nome da disciplina
Código do curso
Numero do semestre

O lançamento de notas consiste na inclusão, alteração, remoção e visualização das notas de um aluno. O registro de qualquer nota é constituído pelos campos:

Código da disciplina
Tipo de prova
Valor da nota
Data de lançamento

O tipo de prova está classificado por:

1- Normal
2- Substitutiva
3- Exame normal
4- Exame recurso

O método de avaliação possui o seguinte critério: Os alunos são submetidos a duas provas (P1 e P2) iniciais. Com base nessas provas, calcula-se a média M=(P1 + P2)/2. Se a média for maior ou igual a 10 e se nenhuma das notas for inferior à 7 o aluno passa. No caso contrário, o aluno é submetido à uma terceira prova P3, e a média final é calculada com base nessa prova.

Se P3 > 7 ; MF= (P3 + máximo(P1,P2))/2
Se P3 < 7; MF = (M + P3)/2

O aluno estará reprovado se a média for inferior a 10.
A impressão de resultados consiste em mostrar as seguintes informações: Uma pauta com os resultados finais; Uma pauta com as estatísticas de estudantes matriculados em cada curso dessa instituição de ensino que passaram e reprovaram.

Para garantir a consistência da informação, não se pode ter dois cursos, dois estudantes e duas disciplinas com o mesmo código.

9.16.23- A biblioteca central de uma universidade necessita de ser informatizada. Para proceder a essa Informatização, desenvolva uma aplicação que possuir o seguinte menu:

1- Cadastro de áreas
2- Cadastro de editoras
3- Cadastro de obras
4- Impressão de resultados
5- Sair da aplicação

O cadastro de área consiste na inclusão, alteração remoção e visualização de qualquer área. O registro de qualquer área é constituído pelos campos:

Código da área
Nome da área

Para efeitos de consistência, não se pode armazenar mais do que uma área com o mesmo código.

O cadastro de editoras consiste na inclusão, alteração, remoção e visualização de um determinada editora. O registro de qualquer editora é constituído pelos campos:

Código da editora
Nome da editora
Endereço da editora

De forma análogo as áreas, não se pode armazenar mais do que uma editora com o mesmo código.

O cadastro de obras consiste na inclusão, alteração, remoção e visualização de uma determinada obra. O registro de qualquer obra é constituído pelos campos:

Número da obra
Nome da obra
Nome do autor
Código da editora
Código a área
Número de exemplares
Data de aquisição

Apôs a inclusão de uma obra, não se pode remover a área nem a editora desse obra. Para além disso, não se pode armazenar mais do que uma obra com o mesmo número.

A impressão de resultados consiste em mostrar as seguintes informações: Todas as obras por editora e todas as obras por área.

9.16.25- Desenvolva uma aplicação para gerir uma carteira de clientes de um banco comercial. O programa deve apresentar o seguinte menu:

1- Cadastro de clientes
2- Cadastro de Contas
2- Movimentos de contas
3- Imprimir resultados
4- Sair do programa

O cadastro de clientes consiste na inclusão, alteração e remoção de um cliente. Qualquer cliente possui os seguintes dados:

Código do cliente
Nome próprio
Sobrenome
Sexo
Data de nascimento
Número de contas abertas (no máximo 6)
Data de cadastro

Por consistência de informação não se pode armazenar mais do que cliente com o mesmo código.

O cadastro de contas consiste em relacionar um cliente a uma ou várias contas. Qualquer conta possui os seguintes campos:

Código do cliente
Código da conta
Tipo de conta
Saldo
Data de abertura

Por consistência de informação, não se pode armazenar dois clientes com o mesmo número de conta, nem o mesmo número de conta para duas contas diferentes do mesmo cliente. Queremos com isso dizer, que o código da conta não se repete.

O banco possui os seguintes tipos de contas:

1- Conta à ordem em Kwanzas
2- Conta à ordem em dólares
3- Conta à prazo em Kwanzas
4- Conta à prazo em dólares
5- Conta poupança em Kwanzas
6- Conta poupança em dólares

Os movimentos de conta são descritos pelas seguintes operações (tipos):

1- Abertura de conta
2- Depósito em Kwanzas
3- Depósito em dólares
4- Retirada em Kwanzas
5- Retirada em dólares
6- Depósito por transferência
7- Retirada por transferência

Para realizar movimentos de depósitos e retiradas nas contas do mesmo cliente ou entre contas de clientes diferentes, o utilizador deve digitar os seguintes dados:

Código do cliente
Tipo de conta
Tipo de movimento
Valor do movimento
Data de movimento

Para realizar movimentos de transferência nas contas do mesmo cliente ou entre contas de clientes diferentes o utilizador deve digitar os seguintes dados:

Código do cliente ordenador
Tipo de conta
Tipo de movimento
Valor do movimento

Código do cliente destinatário
Tipo de conta
Tipo de movimento
Valor do movimento
Data de movimento

É evidente que o banco não permite operações de retirada normal ou retirada por transferência de valores superiores ao saldo da conta. Para além disso estas operações só podem ser feitas em contas com a mesma moeda.

A impressão de resultados consiste em mostrar: A situação do cliente perante o banco, onde se apresenta o saldo que este possui em todas as contas e o extrato de qualquer uma das suas contas ordenado em ordem crescente de data. Esse extrato deve possuir os seguintes campos: data do movimento, descrição do movimento, valor movimentado e saldo em conta.

9.16.26- As grandes lojas de conveniência e supermercados possuem sistemas de cartões pre-pagos. Os clientes que utilizam esses cartões o fazem de forma individual ao através de uma empresa. Desenvolva uma aplicação que trate apenas dos clientes que estão associados as empresas. Essa aplicação deverá possuir o seguinte menu:

1- Cadastrar as empresas

2- Cadastrar os clientes
3- Cadastar os plafond
4- Movimentos de conta
5- Distribuição de plafond
6- Imprimir resultados
7- Sair do programa

O cadastro de empresas consiste na inclusão, alteração, remoção e visualização da ficha de uma empresa. A ficha de qualquer empresa é descrita pelos campos:

Código da empresa
Nome da empresa
Saldo em conta
Percentegem de bónus
Data de cadastro

Para garantir a consistência da informação, não se pode armazenar mais do que uma empresa com o mesmo código.

O cadastro de clientes consiste na inclusão, alteração, remoção e visualização de um cliente. Qualquer cliente é descrito pelos campos:

Código da empresa
Código do cliente
Nome do cliente
grupo de plafond
Saldo em conta
Data de cadastro

Para garantir a consistência da informação, não se pode armazenar mais do que um cliente com o mesmo código.

O cadastro de plafond consiste na inclusão, alteração, remoção e visualização de um grupo de plafonds. Os grupos de plafond servem para associar os trabalhadores de uma determinada empresa á um determinada valor de depósito. Qualquer plafond é descrito pelos campos:

Código da empresa
Código do grupro
Descrição do grupo
Valor a depositar

Os movimentos na conta das empresas são descritos pelas seguintes operações (tipos):

1- Abertura de conta
2- Deposito de valor
2- Retirada de valor

Para efectuar qualquer movimento o utilizador deve digitar os seguintes dados:

Código da empresa
Tipo de movimento
Valor do movimento
Data do movimento

Ao efectuar um deposito o sistema calcula o valor do bonus e armazena como saldo o valor do deposito + bonus. Ao efectuar uma retirada o sistema deve calcular o valor de bonus e devolver ao cliente o valor que ele realmente depositou.

A distribuição de plafond consiste em atribuir aos vários trabalhadores dessa empresa o valor que cada um tem direito em função do seu grupo e da percentagem do bonus que a empresa teve. Essa distribuição só pode ser realizada, se a empresa possui um saldo que permita cobrir essa operação. Ao realizar a operação o saldo da empresa deve ser actualizado, ou seja deve-se subtrair ao saldo da empresa o total distribuído.

A impressão de resultados consiste numa listagem com os saldos dos trabalhadores de uma determinada empresa ordenada por número do trabalhador. Essa listagem deve possuir os seguintes campos: número do trabalhador, nome do trabalhador, valor do deposito. No fim deverá existir uma linha de totais.

9.16.27- No exame de admissão à universidade, cada candidato tem direito à optar por 3 cursos nos 7 cursos oferecidos pela universidade. Cada candidato preenche uma ficha com os seguintes dados: Identificação do candidato, Nota final, 1ª opção, 2 opção e 3ª opção. A identificação ou chave, contêm o número de inscrição do candidato, O campo nota final contém a média das notas do candidato.

Desenvolva um programa para distribuir os candidatos entre os cursos, de acordo com a nota final e as opções apresentadas. No caso de empate serão atendidos em primeiro lugar, os candidatos que se inscreveram mais cedo, isto é, os candidatos que têm o número de inscrição para o exame menor.

9.16.28- Uma pizzaria pretende informatizar os serviços de entrega ao domicílio. Esta aplicação deverá possuir o seguinte menu:

1- Cadastro de pizzas
2- Cadastro de moto boy's
3- Cadastro de clientes
4- Cadastro de pedidos
5- Cadastro movimentos da pizza
6- Sair do programa

Cadastrar as pizzas consiste na inserção, alteração, remoção e visualização de dados de um registro. Cada registro contêm os seguintes campos:

Código da pizza
Nome da pizza
Preço de venda

Para garantir a consistência da informação, não se pode armazenar mais do que uma pizza com o mesmo código.

Cadastrar os moto boy's consiste na inserção, alteração, remoção e visualização de um registro. Cada registro contêm os seguintes campos:

Código do moto-boy
Nome do nome boy

Como restrição, não se pode armazenar mais do que um moto boy com o mesmo código.

Cadastrar os clientes consiste na inserção, alteração, remoção e visualização de dados de um cliente. Cada registro contêm os seguintes campos:

Número do telefone
Nome do cliente
Endereço do cliente.

Como restrição, não se pode armazenar mais do que um cliente com o mesmo número de telefone

Cadastrar os pedidos consiste na inserção, alteração, remoção e visualização de dados de um pedido. Qualquer pedido, possui os seguintes campos:

Número do pedido
Número do telefone do cliente
Código da pizza
Quantidade de pizzas

Todo o pedido só pode ser cadastrado se o cliente e a pizza estiverem cadastrados. No acto do cadastro a situação do pedido deve ser igual a um e o código do moto boy igual a zero. Toda a pizza que tiver sido pedida, não pode ser alterada ou removida do cadastro de pizzas. O mesmo acontece com todo o cliente.

Cadastrar o movimento da pizza consiste na preparação, entrega ao moto boy e entrega ao cliente.

Preparar um pizza consiste em entrar com o número do pedido e alterar o estado do pedido para número um (pizza em preparação)

Entregar a pizza ao moto boy consiste em entrar com o número do pedido e o código do moto boy e alterar o estado do pedido para o número dois. Se a pizza relacionada com esse pedido não foi para a preparação emitir uma mensagem de erro. Para garantir a consistência da informação, todo o moto boy que tiver uma pizza para entrega não pode ser alterado nem removido do cadastro de moto boy's.

Entregar a pizza ao cliente consiste em entrar com o código do pedido e verificar se o moto boy recebeu a pizza. Se essa acção já foi realizada alterar o estado do pedido para o número três e colocar zeros no código do moto boy. Se o pedido não foi entregue ao moto boy deve-se emitir uma mensagem de erro.

Imprimir os seguinte relatórios: Total de gastos com pizzas por cliente; Total de pizzas por situação; Código e nome do moto boy que fez mais entregas; Pizzas ordenadas em ordem decrescente de pedidos com os respetivos totais.

9.16.29- Desenvolva uma pequena aplicação para automatizar o controlo de stock's de uma empresa. A aplicação deve possui o seguinte menu:

1- Cadastro
2- Movimentações
3- Consultas
4- Termino

O cadastro consiste na manutenção das informações sobre os clientes e os produtos. Essa manutenção baseia-se nas seguintes funcionalidades

A manutenção de clientes consiste nas seguintes operações: inclusão, alteração, exclusão e visualização do registro do cliente. Deve-se tomar o cuidado em não inserir clientes com o mesmo código. A alteração consiste em corrigir qualquer campo do registro com excepção do seu código. A exclusão consiste em remover um registro. Deve-se tomar o cuidado em não excluir um cliente com talões. Cada registro contém os seguintes campos:

Código do cliente
Endereço
Telefone

A manutenção de produtos consiste nas seguintes operações: inclusão, alteração, exclusão e visualização do registro do produto. Deve-se tomar o cuidado em não inserir dois produtos com o mesmo código. A alteração consiste em corrigir de qualquer campo do registro, com exceção do seu código. A exclusão consiste em remover um produto. Deve-se tomar o cuidado em não excluir produtos com notas fiscais (talões). Cada registro contém os seguintes campos:

Código do produto
Descrição
Unidade de venda
Preço unitário
Quantidade em Stock
Tipo de produto

O módulo de movimentações, consiste na manutenção das informações sobre as entradas e compras de produtos, que estão sujeitas as seguintes regras:

Qualquer entrada ou compra deve possuir um único talão. Queremos com isso dizer que não podemos ter dois movimentos com o mesmo talão. Para qualquer nova entrada ou compra é armazenada a seguinte informação:

Código do talão
Código do cliente
Total geral do talão, que deve ser igual a zeros

Os produtos só podem inseridos num talão se estiverem devidamente carregados, se isso não acontecer, deve-se emitir uma mensagem de erro. A inserção de qualquer produto é feita com base na seguinte informação:

Código do talão
Código do produto
Quantidade movimentada
Preço de venda

Não se pode inserir um produto no mesmo talão mais do que uma vez. Queremos com isso dizer que não podemos ter produtos repetidos no mesmo talão.

Para proceder a compra de qualquer produto, deve-se verificar se há mercadoria em stock para atender o pedido. Se não houver, emitir uma mensagem de erro.

Ao confirmar uma compra ou uma entrada de um produto, o sistema atualiza o stock, o total geral do talão e insere um registro no vector Itens_por_talão, onde tipo = 1 se for compra e 2 se for entrada.

Para remover qualquer entrada ou compra de um produto, deve-se solicitar ao utilizador para entrar com o número do talão e o código do produto. O sistema atualizará o stock, o total geral do talão e remove o correspondente registro no vector Itens_por_talão.

O módulo de consultas: deverá permitir executar as seguintes listagens:

Dado dois valores digitados pelo utilizador, mostrar todos os artigos cujo preço está entre esses valores.

Dado um cliente, mostrar todos os seus talões

Dado um talão, mostrar os artigos desse talão

O módulo de término, consiste no encerramento do programa

9.16.30- Para gerir o campeonato universitário de fusal "futebol salão", a associação de estudantes da Universidade solicitou uma aplicação que deve possui as seguintes funcionalidades:

1- Cadastrar Universidade
2- Cadastrar jogadores por Universidade
3- Cadastrar calendário da competição
4- Lançar os resultados dos Jogos
5- Imprimir os resultados
6- Sair do programa

O cadastro da Universidade consiste na inserção, alteração, remoção e visualização dos dados de uma universidade. Cada universidade possui os seguintes campos:

Codigo da Universidade
Nome da Universidade
Total de jogos
Total de victorias
Total de empates
Total de derrotas
Total de jogos marcados
Total de golos sofridos

Não se pode ter duas universidades com o mesmo código.

O cadastro de jogadores da Universidade, consiste na inserção, alteração, remoção e visualização dos dados da equipe de cada universidade. Cada jogador possui seguintes campos:

Codigo da Universidade
Numero do jogador
Nome
Sobrenome
Quantidade de golos marcados

Não se pode ter dois jogadores da mesma equipe com o mesmo código e não se pode cadastrar os jogadores de uma universidade se essa universidade não estiver sido previamente carregada.

O cadastro do calendário da competição, consiste em inserir por jornada o emparelhamento entre as equipes. Este cadastro possui os seguintes campos:

Número da jornada
Número de jogos lançados
Código da primeira equipe
Numero de golos
Código da segunda equipe
Numero de golos

Antes de efectuar qualquer lançamente é necessário lançar todo o calêndário da competição. Ao lançar o calêndário da competição devemos preencher com zeros os campos: número de jogos da competição, o numero de golos marcados pela equipe e o total de jogos marcados pelos jogadores da equipe.

O lançamento dos resultados de qualquer jogo, consiste na inserção, alteração e visualização dos seguintes campos:

Número da jornada
Código da primeira Universidade
Total de golos marcados pela Universidade
Código da segunda Universidade
Total de golos marcados pela Universidade

Antes de armazenar essa informação, o sistema deve verificar se os dados digitados pelo utilizador são consistentes. Só depois dessa verificação é que o sistema actualiza o cadastro do calêndário e o cadastro de universidades. Também devemos salientar que não se pode lançar os resultados de uma jornada sem antes ter terminado de lançar todos os resultados da jornada anterior e as estatística desses jogos. Para além disso, não se pode lançar os resultados de uma jornada que não estejam no calêndário.

O lançamento das estatísticas por jogo, consiste na inserção, alteração, e visualização dos seguintes campos:

Numero da jornada
Código da universidade
Código do jogador
Número de golos

Essas estatísticas só podem ser lançadas se já estiverem sido carregadas as informações sobre os resultados desse jogo. Deve-se ter o cuidado de analisar a consistencia dessas estatísticas. O total de golos marcados pelos jogadores não pode ser superior ao número de golos marcados pela equipe. Se a informação for coerente, o sistema deve actualizar o cadastro de jogadores por universidade e o cadastro de calêndário.

A impressão de resultados consiste nas seguintes listagens:

- Classificação do campeonato por equipes;

- Resultados das jornadas;
- Jogador com mais golos marcados
- A equipe com a melhor ataque
- A equipe com a melhor defesa

Para efeito de classificação, a vitoria soma 3 pontos o empate 1 e a derrota zero pontos.

Para efeito de desempate na classificação, está em primeiro lugar a equipe que tiver o maior número de vitórias. Se o empate persistir, está em primeiro lugar a equipe que tiver o maior número de empates. Se o empate persistir está em primeiro lugar a equipe que tiver o menor número de derrotas.

9.16.31- A Gestcarro é uma empresa de aluguer de viaturas que pretende uma aplicação para gerir este negócio. Essa aplicação deve possuir o seguinte menu:

1- Cadastro de viaturas
2- Cadastro de clientes
3- Cadastro de alugueres
4- Impressão de resultados
5- Pesquisa de informação
6- Sair da aplicação

O cadastro de viaturas consiste na inserção, alteração, remoção e visualização os dados de uma viatura. Cada viatura é descrita pelos seguintes campos:

Marca
modelo
cor
cilindrada
Ano de aquisição
nº do chassi
matricula
valor por dia

O cadastro de clientes consiste na inserção, alteração, remoção e visualização dos dados de um cliente. Cada cliente é descrito pelos seguintes campos:

Nome
Morada
Internet
Número do BI
Número da carta de condução

O cadastro de alugueres de viaturas consiste na inserção, alteração, remoção e visualização dos dados de uma operação de aluguer. O registro de qualquer operação é constituído pelos seguintes campos:

Identificação da viatura
Identificação do cliente
Data de inicio
Data de fim

Uma viatura só pode ser alugada se estiver livre e carregada no sistema. Ao confirmar o aluguer de uma viatura o sistema deve calcular o valor a pagar e verificar se a data de fim é superior ou igual a data de início.

A impressão de resultados consiste nas seguintes listagens: Viaturas alugadas e não alugadas; Viaturas alugadas ordenadas por data de entrega; Clientes com os respectivos valores a pagar ordenados por valor.

As pesquisas de informação consistem em: Dada uma determinada matricula de um carro mostrar todos os clientes que alugaram esse carro as suas dadas; Dada o número do bilhete de identidade de um cliente, mostrar todos os carros que ele alugou e suas dadas; Dado um determinado intervalo de tempo, mostrar todos as viaturas alugadas e seus respectivos clientes, ordenados por data.

9.16.32- O centro de informática da Universidade de Luanda, pretende manter um registro sobre o parque informático dessa instituição. Essa aplicação deve possuir o seguinte menu:

1- Cadastro de tabelas primárias
2- Cadastro de equipamentos
3- Cadastro de aplicações instaladas
4- Impressão de resultados
5- Pesquisa de informação
6- Sair da aplicação

O cadastro de tabelas primárias consiste em:

O cadastro de tipos de sistema operativo consiste na inserção, alteração, remoção e visualização da ficha de um sistema operativo. O armazenamento de cada ficha é feito pelos seguintes campos:

Tipo do sistema operativo
Nome do sistema operativo

O cadastro de tipos de aplicação consiste na inserção, alteração, remoção e visualização da ficha de uma aplicação. O armazenamento de cada ficha é feito pelos seguintes campos:

Tipo de aplicação
Nome da aplicação

O cadastro de equipamentos consiste na inserção, alteração, remoção e visualização da ficha de um equipamento. O armazenamento de cada ficha é feito pelos seguintes campos:

Código de equipamento
Data de aquisição
Garantia
CPU
Disco duro
Memória RAM
Tipo de sistema operativo
Endereço IP

Observe que não se pode incluir um equipamento se o tipo do sistema operativo que esse equipamento possui, não estiver cadastrado no sistema. Não se pode armazenar mais do que um equipamento com o mesmo código, nem com o mesmo endereço IP.

O cadastro de aplicações instaladas consiste na inserção, alteração, remoção e visualização da ficha de uma aplicação. O armazenamento de cada aplicação é feito pelos seguintes campos:

Código do equipamento
Tipo de aplicação
Versão
Data da Licença
Data de expiração

Observe que não se pode inserir uma aplicação num equipamento se ela não tiver sido inserido. De forma análoga, não se pode armazenar mais do que um código de equipamento com o mesmo número.

A impressão de resultados consiste nas seguintes listagens: Equipamentos cuja garantia expirou; Equipamentos com um determinado sistema operativo; Equipamentos com uma determinada aplicação; Equipamentos com o mesmo IP;

As pesquisas de informação consistem em: Todos os equipamentos; Equipamentos cuja garantia expirou; Equipamentos com um determinado tipo de sistema operativo; Equipamentos com uma determinada aplicação e equipamentos com um sistema operativo e com uma determinada aplicação.

Referencias Bibliográficas

[Asc 07] Ascencio G.F.A, Campos V.A.E. ;- *Fundamentos da programação de computadores*, 2ª Edição, Longman, 2007, Brasil

[Dam 99] Damas L.;- *Linguagem C*, FCA,1999, Portugal

[Far 99] Farrer H., Beker C.G, Faria E.F, Santos M.A.- *Algoritmos estruturados*, 3ª edição,LTC,1999, Brasil.

[Has 07] Hashimoto F.R, Morimoto H.C.;- *Notas de aulas de introdução à ciência de computação I,* Instituto de Matemática e Estatística da Universidade de São Paulo, 2007, Brasil

[Ker 89] Kernighan B W., Richie D. M.;- *C a linguagem de programação padrão Ansi*, 23ª Edição, Campus,1989, Brasil

[Lee 12] Lee O.,- *Notas de aulas de algoritmos e programação de computadores*, Instituto de Computação da Universidade Estadual de Campinas, 2012, Brasil.

[Miz 08] Mizrahi V. V.,- *Treinamento em Linguagem* C, 2ª Edição, Pearson Education, 2008, Brasil

[Oso 10] Osório F. S. ;- *Notas de aulas de Introdução à Ciência da Computação* I, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2010, Brasil

[Ran 04] Rangel J.L., Cerqueira R., Celes W.;- *Notas de aula de Introdução a estrutura de dados em C*, Departamento de Informática da Pontífice Universidade Católica do Rio de Janeiro, 2004, Brasil.

[Rob 95] Roberts E. S. ;- *The Art and Science of C*, Addison-Wesley, 1995

[Sal 98] Salvetti D.D, Barbosa M. M.; - *Algoritmos*, Makron books,1998, Brasil

[Sch 97] Schildt H.;- *C completo e total*, 3ª edição, Makron books, 1997, Brasil

[Tre 83] Trembley J. P, Brunt R. B. ; - *Ciências dos Computadores uma abordagem algorítmica*, Mcgraw-Hill,1983, Brasil

[Wir 87] Wirth N.;- *Programação Sistemática em Pascal, 6ª Edição,* Campus, 1987,Brasil.

[Wir 89] Wirth N.;- *Programação em Modula2, Série Ciência de Computação*, Livro técnico e científico, 1889, Brasil.