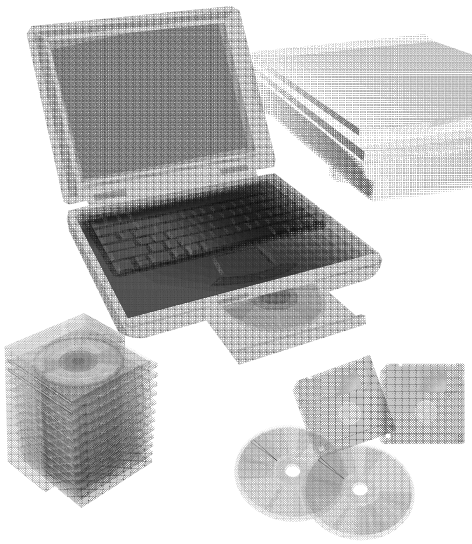
Introdução à Computação II



Enumerações, Registros, Uniões e Campos de Bit

Profa.: ***Adriane Beatriz de Souza Serapião***

adriane@rc.unesp.br

Tipos de Dados

- **Um *tipo de dados* define um conjunto de valores de mesma natureza, isto é, eles podem ser transformados uns nos outros por meio de operadores comuns.**
- **Em outras palavras ainda, um tipo de dados consiste em um conjunto de valores munido de operações básicas sobre esses valores.**
- **Os tipos podem ser *escalares* ou *estruturados.***

Tipos Escalares

- **Os tipos escalares são constituídos por valores indivisíveis ou atômicos.**
- **Os tipos escalares, por sua vez, podem ainda ser classificados em:**
  - **Tipos *primitivos* ou *básicos***
  - **Tipos definidos pelo programador**
- **Os tipos primitivos são geralmente implementados por hardware, mais especificamente, pela CPU.**

Tipos Escalares Primitivos

- **Os tipos escalares primitivos da linguagem C são:**
  - **Inteiro, denotado pelo identificador padrão `(unsigned)(short) (long) (long) int`**
  - **Real, denotado pelo identificador padrão `float, (long) double`**
  - **Caracter, denotado por `char`**
  - **Lógico, denotado por `bool`**

Tipos escalares definidos pelo usuário

- **Além dos quatro tipos primitivos básicos fornecidos pela linguagem, o programador pode definir algum tipo próprio adequado para seu programa em particular.**
- **Por exemplo, ele deseja construir um programa cujos itens de dados sejam os *naipes* de cartas de baralho: *ouros*, *copas*, *espadas* e *paus.***
- **Ou os *dias da semana: domingo*, *segunda*, *terça*, *quarta*, *quinta*, *sexta* e *sábado.***

Tipos Enumerados

- **A linguagem C não fornece esses tipos, mas permite que o programador os defina a partir da palavra-chave *enum*.**
- **Estes tipos são definidos pela *enumeração* de seus valores.**
- **Associa cada nome a um número. Eles são atômicos e ordinais.**
- **Sintaxe:**

```
enum nome_enumeração {opção1, opção2, opção3,
  ...} variável;
```

Tipos Enumerados

- **Enumeração**
  - ***Etiqueta*** ⇨ **Uso indireto.**
  - ***Lista_de_enumeradores*** ⇨ **Constantes simbólicas.**
    - **Valores inteiros de *0* a *n*.**
    - **Exemplos:**

```
enum naipe {ouros0, copas1, espadas2, paus3};

enum diasSemana {domingo0, segunda1, terca2,
   quarta3, quinta4, sexta5, sabado6};
```


Tipos Enumerados

- **Enumeração**
  - ***Lista_de_enumeradores*:**
    - **Primeiro enumerador ⇨ Atribuição do valor *0*.**
    - **Cada enumerador sucessivo ⇨ Atribuição de um valor *maior* do que o de seu predecessor.**
      - **Possibilidade de especificação explícita de um valor para um enumerador particular.**

Tipos Enumerados

- **Enumeração**
  - ***Lista_de_enumeradores*:**
    - **Enumeradores *não* precisam estar associados a valores *únicos*.**
    - **O nome de *cada* enumerador deve ser *único* no escopo de definição da *enum*.**
    - **A conversão de um inteiro para um enumerador requer um *cast* explícito.**
      - **Os resultados *não* são *definidos*.**

Tipos Enumerados

- **Declaração:**
  - `enum {dinheiro, cheque, vale_refeicao, cartao} forma_pagamento;`

- **Seleção:**

```
switch (forma_pagamento) {
   case dinheiro: ...
      break;
   case cheque: ...
      break;
   case vale_refeicao: ...
      break;
   case cartao: ...
      break; }
```

Tipos Enumerados

- **Uso de enumerações:**

```
enum {
        dinheiro = ‘d’,
        cheque   = ‘c’,
        vale_refeicao = ‘v’,
        cartao = ‘k’
} forma_pagamento;
```


Tipos Enumerados

- **É claro que os operadores relacionais podem ser aplicados aos valores de tipos enumerados.**

- **Por exemplo:**

```
domingo < segunda é true, porque (domingo = 0) <
  (segunda = 1)
ouros > espadas é false, porque (ouros = 0) <
  (espadas = 2)
paus = espadas é false, porque (paus = 3) e (espadas =
  2)
```

Tipos Enumerados

- **Supondo-se que se quisesse escrever o nome dos dias da semana, usando um vetor de strings:**

```
enum DiasDaSemana {Domingo, Segunda, Terca,
        Quarta, Quinta, Sexta, Sabado};

void main()
{
   enum DiasDaSemana j;
   char vetor[7][] = {“Domingo”, “Segunda”, “Terça”,
   “Sabado”}; {vetor é uma constante tipada}
                “Quarta”, “Quinta”, “Sexta”,

   for (j=domingo; j++; j<=sabado)
        printf(“%s\n”,vetor[j]);
}
```


Tipos Enumerados

- **Exemplo 1 - Uso de tipos de dados enumerados para acesso aos itens de um *array.***

```
#include <stdio.h>
int main( )
{
   int Novembro[5][7]={{0,0,1,2,3,4,5},{6,7,8,9,10,11,12},
   {13,14,15,16,17,18,19},{20,21,22,23,24,25,26},
   {27,28,29,30,0,0,0}};
   enum dias {Domingo, Segunda, Terca, Quarta, Quinta, Sexta,
   Sabado};
   enum semana {semana01, semana02, semana03, semana04,
   semana05};

   printf (“Quinta da primeira semana de Novembro eh dia
%d\n“, Novembro [semana01][Quinta]);
 }
```

Tipos Enumerados

Exemplo 2

```
enum Dias // Declaracao do tipo enum Dias
{
    Sabado, // Sabado = 0 (por definicao)
    Domingo = 0, // Domingo = 0 (por atribuicao do usuario)
    Segunda, // Segunda = 1
    Terca, // Terca = 2
    Quarta, // Quarta = 3
    Quinta, // Quinta = 4
    Sexta // Sexta = 5
} Hoje; // Criacao da variavel Hoje do tipo Dias

int  Sexta; // Definicao incorreta (redefinicao de Sexta)

enum Dias Ontem; // Correta em C e C++
Dias Amanha; // Correta apenas em C++

Ontem = Quinta;

int d = Quinta; // Correta (atribuicao de 2 a d)
Ontem = 0; // Incorreta (sem conversao de tipo)
Ontem = (Dias)0;      // Correta, mas com resultados indefinidos
```


Tipos Enumerados

Exemplo 3 – programa que codifica dias da semana como inteiros (sendo sabado = 5 e domingo = 6) para verificar se o dia do pagamento cai no final de semana e altera a dia para a segunda-feira seguinte.

```
#include <stdio.h>

enum TpSemana {SEG, TER, QUA, QUI, SEX, SAB, DOM};
/* prototipo da funcao que dada a data, retorna o dia da semana */
enum TpSemana diaDaSemana( int dia, int mes, int ano );

int main(){
  int diaPgto, mesPgto, anoPgto;
  int diaSem;
  printf("Entre com a data de pagamento (dd mm aa): ");
  scanf("%d %d %d", &diaPgto, &mesPgto, &anoPgto);
  diaSem = diaDaSemana( diaPgto, mesPgto, anoPgto );
  if( diaSem == SAB )
    diaPgto = diaPgto + 2;
  else if( diaSem == DOM )
    diaPgto++;
 printf("Data do pagamento: %d/%d/%d\n", diaPgto, mesPgto, anoPgto);
}
```

Características dos tipos enumerados

- **Criam sequências ordenadas de identificadores (NÃO de strings) com valor de posição variando de 0 a *n*-1.**
- **Permitem que em vez de usar-se valores inteiros que sejam codificações, use-se nomes significativos referentes ao problema em questão, cujo valor é a posição que ocupam na lista de nomes que integram.**
- **Não são compostos por strings de texto.**
- **Não há qualquer palavra reservada associada a eles.**

Características dos tipos enumerados

- **Não há qualquer operação específica para eles.**
- **Identificadores de um tipo enumerado só representam a si próprios.**
- **Não podem ser usados diretamente em scanfs e printfs!**
- **Um valor não pode integrar mais de um tipo.**
- **Valores de tipos enumerados são constantes do tipo correspondente.**
- **Somente operadores relacionais podem ser usados com tipos enumerados.**

Tipos Estruturados

- **Os tipos estruturados permitem manipular vários valores ao mesmo tempo.**

- **Os tipos estruturados, por sua vez, podem ainda ser classificados em:**
  - **Tipos *primitivos* ou *básicos*:**
    - **Cadeias de caracteres (strings)**
  - **Tipos definidos pelo programador:**
    - ***Arranjos (arrays)***
    - ***Registros***
    - ***Conjuntos***
    - ***Arquivos***

Registros

- **“Variáveis compostas heterogêneas”.**

- **São conjuntos de dados logicamente relacionados, mas de tipos diferentes (inteiro, real, string, etc.).**

Registros

- **A estrutura de dados registro é extremamente versátil.**
- **Com ela podemos representar, sob uma mesma estrutura, vários itens de dados de tipos diferentes.**
- **Formalmente, o tipo registro pode ser visto como um produto cartesiano de tipos.**
- **Neste caso, um valor particular do tipo registro é uma tupla, com a vantagem adicional de podermos identificar cada componente da tupla com um nome.**

Registros

- **Sintaxe da estrutura**
  - **A linguagem C tem o seguinte construtor primitivo de tipos registro:**

```
struct identificador {
                    lista_de_campos
              } variáveis;
```

  - **A *lista de campos* pode ser definida como uma sequência de:**

tipo campo

Registros

- **Ou seja:**

```
struct <identificador_do_tipo> {
  <tipo_do_campo1>   <identificador_do_primeiro_campo>;
  <tipo_do_campo2>   <identificador_do_segundo_campo>;
  <tipo_do_campo3>   <identificador_do_último_campo> ;
} <variáveis>;
```

- **Onde:**

<identificador_do_tipo> é o nome que você deseja dar ao novo tipo de dados (registro).
<identificador_do_n-ésimo_campo> é o nome que você deseja dar ao n-ésimo atributo (campo).
<tipo_do_campon> é o tipo do atributo (tipo).

Registros

- **Sintaxe (cont.)**
  - **Um *campo* é denotado por um identificador e o *tipo* pode ser qualquer um.**
  - **A definição de um campo é separada de outra por ponto-e-vírgula.**

- **Exemplo**

```
struct Funcionario {
        char nome[30];
        int salario
    };
```

Registros

- Declaração de ***Structs***

Sem **Alocação de Espaço**	*Com* **Alocação de Espaço**

struct *Exemplo01*
{
  char *nome[40];*
  char *espec[80];*
  int *quant;*
} ;
	struct *Exemplo01*
	{
	  char *nome[40];*
	  char *espec[80];*
	  int *quant;*
	} Almoxarifado;

struct Exemplo01 Almoxarifado;

Declaração da variável Almoxarifado como do tipo estrutura Exemplo01 (alocação da memória neste ponto)

Alocação da memória na declaração da estrutura.
Definição de um tipo de estrutura Exemplo01 e declaração de variável daquele tipo.

Registros

- Mais exemplos

```
struct complexo {
                    float re, im;
                };

struct Aluno {
                    char matr[7];
                    char nome[30];
                    enum curso = {Adm, Comp, Eng,
                                  Fis, Mat, Med};
                };
```

Registros

- **Mais exemplos**

CADASTRO PESSOAL		
Nome:		
Endereço:		
Bairro:	Cidade:	UF:
Data de nascimento: / /	Profissão:	

Registros

```
struct TipoCadastro {
   char nome[40];
   char endereco[100];
   char bairro[40];
   char cidade[40];
   char uf[3];
   int dia_nasc, mes_nasc, ano_nasc;
   char profissao[40];
};

struct TipoCadastro cadastro;
```

struct Identificador {

campos;

};

Ex: printf("Nome: %s\n", cadastro.nome);

Registros dentro de registros

```
struct TipoCadastro {
            char nome[40];
            struct endereco {
                        char rua[100];
                        char bairro[40];
                        char cidade[40];
                        char uf[3];
                      };
            int dia_nasc, mes_nasc, ano_nasc;
            char profissao[40];
          };
```

Ex: printf("Estado: %s\n", cadastro.endereco.uf);

Registros dentro de registros

```
struct endereco {
                    char rua[100];
                    char bairro[40];
                    char cidade[40];
                    char uf[3];
                  };

struct TipoCadastro {
            char nome[40];
            struct endereco adr;
            int dia_nasc, mes_nasc, ano_nasc;
            char profissao[40];
          } cadastro;
```

Ex: printf("Estado: %s\n", cadastro.adr.uf);

Registros

- **No exemplo anterior, em C, para atribuir um valor a um determinado campo do registro:**

```
cadastro.nome = "Jose da Silva"; ou
   cadastro.nome = puts(variável); ou
         scanf("%s\n", &cadastro.nome);


cadastro.endereco.rua = "Rua sem saida"; ou
   cadastro.endereco.rua = puts(variável); ou
        scanf("%s\n",&cadastro.endereco.rua);


cadastro.endereco.uf = "SC"; ou
   cadastro.endereco.uf = puts(variável);
         scanf("%s\n, &cadastro.endereco.uf);


cadastro.dia_nasc = 18; ou cadastro.dia_nasc=scanf("%d",
   &variável); ou scanf("%d\n, &cadastro.dia_nasc);
```


Array de registros

tipo identificador **array** $[f_1]$ ... $[f_n]$;

```
struct camisas {
  char descricao[100];
  float preco;
  int estoque;
};

struct camisas mercadorias[100];
```

printf("Estoque da camisa cod.55: %d", mercadorias[55].estoque);

Registros

- O tipo registro representa todos os valores compostos que tem aquela estrutura.
- Cada valor do tipo registro é uma tupla.
- Por exemplo, um valor do tipo `complexo` definido anteriormente pode ser o par (1.0, 2.0) que, neste caso, representa o número complexo $1 + 2i$.
- Um valor do tipo `Aluno` pode ser a tripla (32768, 'Antônio Lopes', Mat).
- Um valor do tipo `Funcionario` pode ser o par ('Gerusa Sampaio', 150000).

Registros

- Infelizmente, o C não fornece nenhum mecanismo para denotar uma constante do tipo registro.
- Se quisermos representar um valor específico de um tipo registro, temos que definir uma variável com esse tipo e depois armazenar os valores individuais de cada campo por meio de uma atribuição ou uma leitura.
- A criação de uma variável do tipo registro é feita usando a notação normal de declaração de variável.

Acesso e Seleção de Campos de Registros

- **Em C, para acessar ou selecionar um componente de uma variável registro, usamos o operador de seleção de campo.**
- **O *operador de seleção de campo* é denotado por . (um ponto) (para ponteiros é ->).**
- **Então, para referenciar um componente de um registro, o nome da variável registro é seguido por um ponto e o respectivo identificador do campo.**
- **O nome da variável sozinho referencia todo o registro.**

Registros - Exemplo em C

- **Sejam as declarações:**

```
struct Data {
      int dia;
      enum mes={jan, fev, mar, abr,
             mai, jun, jul, ago,
             sept, out, nov, dez};
      int ano;
   } nasc;
```

- **Para armazenar a data 02/12/2004 na variável nasc, podemos fazer:**

```
nasc.dia = 2; nasc.mes = dez; nasc.ano = 2004
```

Registros dentro de registros

```
struct Data {
                int dia;
                enum mes={jan, fev, mar, abr,
                          mai, jun, jul, ago,
                          sept, out, nov, dez};
                int ano;
      };

struct endereco {
                          char rua[100];
                          char bairro[40];
                          char cidade[40];
                          char uf[3];
                };

struct TipoCadastro {
                char nome[40];
                struct endereco adr;
                struct Data nasc;
                char profissao[40];
            } cadastro;
```


Registros - Exemplo em C

- **Programa para jogar cartas:**

```
1  /* JogoDeCartas.c
2     Programa para embaralhar e distribuir cartas usando estruturas */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  struct carta {
8     const char *face;
9     const char *naipe;
10 };
11
12 typedef struct carta Carta;
13
14 void preencherBaralho( Carta * const, const char *[],
15                        const char *[] );
16 void embaralhar( Carta * const );
17 void distribuir( const Carta * const );
18
```

Registros - Exemplo em C

```
19 int main()
20 {
21    Carta baralho[ 52 ];
22    const char *face[] = { "As", "Dois", "Tres",
23                           "Quatro", "Cinco",
24                           "Seis", "Sete", "Oito",
25                           "Nove", "Dez",
26                           "Valete", "Rainha", "Rei"};
27    const char *naipe[] = { "Copas", "Ouros",
28                           "Paus", "Espadas"};
29
30    srand( time( NULL ) );
31
32    preencherBaralho( baralho, face, naipe );
33    embaralhar( baralho );
34    distribuir( baralho );
35    return 0;
36 }
37
```


Registros - Exemplo em C

```
38 void preencherBaralho( Carta * const wBaralho, const char * wFace[],
39                 const char * wNaipe[] )
40 {
41    int i;
42
43    for ( i = 0; i <= 51; i++ ) {
44       wBaralho[ i ].face = wFace[ i % 13 ];
45       wBaralho[ i ].naipe = wNaipe[ i / 13 ];
46    }
47 }
48
49 void embaralhar( Carta * const wBaralho )
50 {
51    int i, j;
52    Carta temp;
53
54    for ( i = 0; i <= 51; i++ ) {
55       j = rand() % 52;
56       temp = wBaralho[ i ];
57       wBaralho[ i ] = wBaralho[ j ];
58       wBaralho[ j ] = temp;
59    }
60 }
61
```

Registros - Exemplo em C

```
62 void distribuir( const Carta * const wBaralho )
63 {
64    int i;
65
66    for ( i = 0; i <= 51; i++ )
67       printf( "%5s de %-8s%c", wBaralho[ i ].face,
68               wBaralho[ i ].naipe,
69               ( i + 1 ) % 2 ? '\t' : '\n' );
70 }
```


Registros - Exemplo em C

```
Quatro de Paus          Tres de Copas
Tres de Ouros        Tres de Espadas
 Quatro de Ouros         As de Ouros
 Nove de Copas           Dez de Paus
Tres de Paus           Quatro de Copas
Oito de Paus           Nove de Ouros
Dois de Paus          Rainha de Paus
Sete de Paus           Valete de Espadas
  As de Paus           Cinco de Ouros
  As de Espadas           Cinco de Paus
Sete de Ouros         Seis de Espadas
Oito de Espadas          Rainha de Copas
 Cinco de Espadas           Dois de Ouros
Rainha de Espadas             Seis de Copas
Rainha de Ouros       Sete de Copas
 Valete de Ouros         Nove de Espadas
Oito de Copas          Cinco de Copas
 Rei de Espadas            Seis de Paus
Oito de Ouros         Dez de Espadas
  As de Copas          Rei de Copas
 Quatro de Espadas           Valete de Copas
Dois de Copas          Valete de Paus
Dois de Espadas           Dez de Ouros
Sete de Espadas          Nove de Paus
 Rei de Paus            Seis de Ouros
  Dez de Copas          Rei de Ouros
```

Registros

- ***Typedef* e *Struct***
  - **Uso combinado ⇨ Declaração de um sinônimo (ou codinome) para uma *struct***

```
typedef struct /* Definicao de uma estrutura */
        {
          int num_item;
          char nome_item[30];
          char espec_item[60] ;
        } Estrutura;/* Codinome da struct */

Estrutura est1; /* Criação de uma variavel do
tipo Estrutura */
```


Estruturas definidas recursivamente

- **Quando duas estruturas referem-se uma à outra, uma delas deve ser declarada de modo incompleto (prototype).**

```
struct HUMANO;
struct PET {
  char nome[NOME_LIMITE];
  char species[NOME_LIMITE];
  struct HUMAN *dono;
} fido = {"Fido", "Canis lupus familiaris"};
struct HUMANO {
  char nome[NOME_LIMITE];
  struct PET pets[PET_LIMITE];
} sam = {"Sam", {fido}};
```

Não se pode declarar o membro dono neste ponto, pois ele não foi declarado ainda.

Exercício proposto 1

Exercício:

Implementar um programa em C que leia uma tabela de profissões composta de código, nome e descrição da profissão. A tabela comporta no máximo 100 profissões. Para terminar o cadastro digite -1 no campo código.

Então, você informa um código e o programa informa o nome e a descrição da profissão.

Exercício proposto 2

- **Faça um programa que realize o cadastro de contas bancárias com as seguintes informações: número da conta, nome do cliente e saldo. O banco permitirá o cadastramento de apenas 15 contas e não pode haver mais de uma conta com o mesmo número. Crie o menu de opções a seguir.**

  Menu de Opções:

  1. Cadastrar contas

  2. Visualizar todas as contas de um determinado cliente

  3. Excluir a conta com menor saldo (supondo não existir saldo iguais)

  4. Sair

```
Algoritmo
 Declare conta[15] Registro (num, saldo
    NUMÉRICO, nome LITERAL)
 Declare i , op, posi, achou, noconta,
    sldcliente, sldmenor NUMÉRICO
 Declare nomcliente LITERAL
 Para i ← 0 até 14 Faça
 Início
   conta[i].num ← 0
   conta[i].saldo ← 0
   conta[i].nome ← ‘’
 Fim
 Repita
   Escreva “Menu de Opções”
   Escreva “1 – Cadastrar contas”
   Escreva “2 – Visualizar contas cliente”
   Escreva “3 – Excluir conta menor sldo
   Escreva “4 – Sair”
   Escreva “Digite Opção: “
   Leia op
```

Exercício proposto 2 (cont.)

```
Caso (op)
 1: Inicio
      achou ← 0
      Escreva "Digite no. Conta: "
      Leia noconta
      Para i ← 0 até 15 Faça
      Inicio
       Se (noconta = conta[i].num)
       Então achou ← 1
      Fim
      Se achou = 1 Então
       Escreva "Conta já cadastrada"
      Senao Inicio
       posi ← 0
       i ← 0
       Enquanto (i < 15) Faça
       Inicio
        Se conta[i].num = 0 Então
        Inicio
         posi ← i
         i ← 15
        Fim
        i ← i + 1
       Fim
       Se posi = 0 Então
          Escreva " Impossivel cadastrar
novas contas"
          Senão Inicio
            Escreva "Digite Nome: "
            Leia nomcliente
            Escreva "Digite Saldo: "
            Leia sldcliente
            conta[posi].num ← noconta
            conta[posi].saldo ← sldcliente
            conta[posi].nome ← nomcliente
            Escreva "Conta cadastrada c/ sucesso."
          Fim
        Fim
      Fim
 2: Inicio
      Escreva "Digite nome: "
      Leia nomcliente
      achou ← 0
      Para i ← 0 até 15 Faça
      Inicio
       Se conta[i].nome = nomcliente Então
       Inicio
          Escreva conta[i].num, conta[i].saldo
          achou ← 1
       Fim
      Fim
      Se achou = 0 Então
          Escreva "Não existe conta p/ esse cliente"
Fim
```


Exercício proposto 2 (cont.)

```
 3: Inicio
      i ← 0
      achou ← 0
      sldmenor ← 999999.9
      Enquanto (i < 15) Faça
      Inicio
         Se (conta[i].num = 0) Então
          i ← 15
         Senao Inicio
          achou ← 1;
          Se conta[i].saldo < sldmenor
Entao
          Inicio
            sldmenor ← conta[i].saldo
            posi ← i
          Fim
          i ← i + 1
         Fim
      Fim
      Se achou = 0 Então
          Escreva "Sem conta"
      Senão Inicio
          i ← posi
          Enquanto i < 14 Faça
          Inicio
           Se conta[i+1].num != 0
           Inicio
            conta[i].num ← conta[i+1].num
            conta[i].nome ← conta[i+1].nome
            conta[i].saldo ← conta[i+1].saldo
            i ← i + 1
           Fim
           Senão Inicio
            conta[i] ← 0
            conta[i].nome ← ""
            conta[i].saldo ← 0;   i ← 14
           Fim
          Fim
          Escreva "Conta excluida com sucesso"
        Fim
      Fim
   4: Inicio
        Escreva "Sair do Programa"
      Fim
   Default: Inicio
        Escreva "Opcao Invalida"
      Fim
  até op = 4
Fim_algoritmo
```

Exercício proposto 3

- Uma empresa prestadora de serviços armazena informações sobre os serviços prestados. Sabe-se que essa empresa pode atender a apenas três clientes em cada dia, ou seja, realiza, no máximo, três serviços diariamente.

- É de interesse da direção dessa empresa manter um histórico mensal (30 dias) sobre os serviços prestados. A empresa realiza 4 tipos de serviços: 1 – Pintura; 2 – Jardinagem; 3 – Faxina; 4 – Reforma Geral.

- Cada Serviço desenvolvido deve ser cadastrado com as seguintes informações: número do serviço, valor do serviço, código do serviço e código do cliente.

- Cadastre todos os tipos de serviços (codigo e descrição) que a empresa poderá realizar. Para isso, utilize um vetor de quatro posições.

Exercício proposto 3

- Mostre o seguinte menu de opções:
  1. Cadastrar os tipos de serviço
  2. Cadastrar os serviços prestados
  3. Mostrar os serviços prestados em um determinado dia
  4. Mostrar os serviços prestados dentro de um intervalo
  5. Mostrar um relatório geral (separado por dia), que exibe, inclusive, a descrição do tipo do serviço
  6. Finalizar

- Para opcão 1:
  - deve-se cadastrar os tipos de serviços oferecidos pela empresa, com código e descrição.

Exercício proposto 3 (cont.)

- **Para opção 2:**
  - **deve-se considerar que deverão ser cadastrados os serviços prestados ao longo do mês. Em cada dia podem ser cadastrados, no máximo 3 serviços**
  - **Utilize uma matriz capaz de armazenar em cada posição todas as informações referentes a um serviço prestado. Cada linha representa um dia do mês. Dessa maneira, considere a matriz com dimensão 30x3.**
  - **Solicite o dia que o serviço foi prestado e as demais informações**
  - **Lembre-se de que a emporesa só pode prestar os serviços que já tenham sido cadastrados no vetor de tipo de serviços.**
  - **Caso o usuário digite um código de tipo de serviço inválido, mostre uma mensagem de erro.**
  - **Quando o usuário tentar cadastrar mais de três serviços prestados em um mesmo dia, mostre uma mensagem de erro.**

Exercício proposto 3 (cont.)

- **Para opção 3:**
  - **Receba o dia que deseja consultar e mostre os respectivos serviços prestados.**
- **Para opção 4:**
  - **Receba o valor mínimo e o valor máximo e mostre os serviços prestados que estiverem neste intervalo.**
- **Para opção 5:**
  - **Mostre todos os serviços prestados, conforme o exemplo a seguir:**

Exercício proposto 3 (cont.)

Dia 01				
No. Serviço	Valor do Serviço	Código do Serviço	Descrição	Código do Cliente
100	R$ 200.00	1	Pintura	1
150	R$ 100.00	3	Faxina	5
Dia 02				
No. Serviço	Valor do Serviço	Código do Serviço	Descrição	Código do Cliente
301	R$ 600.00	4	Reforma	3
280	R$ 352.00	1	Pintura	2