

Cap. V – VARIÁVEIS ESTRUTURADAS

5.1 – Variáveis Indexadas Numéricas

5.1.1 – A necessidade de variáveis indexadas

- Seja um programa para contar os votos de uma eleição de 8 candidatos:

```
#include <stdio.h>
void main ( ) {
    int n, c0, c1, c2, c3, c4, c5, c6, c7, nulos, voto, i;

    printf ("Apuracao de eleicao:\n\n\tNumero de votos: ");
    scanf ("%d",&n); printf ("\n");
    c0 = c1 = c2 = c3 = c4 = c5 = c6 = c7 = nulos = 0;
    for (i = 1; i <= n; i++) {
        printf ("\t\tVoto: "); scanf ("%d", &voto);
        switch (voto) {
            case 0: c0++; break; case 1: c1++; break;
            case 2: c2++; break; case 3: c3++; break;
            case 4: c4++; break; case 5: c5++; break;
            case 6: c6++; break; case 7: c7++; break;
            default: nulos++;
        }
    }
    printf ("\n\nResultado: \n\n\tc0: %d; c1: %d", c0, c1);
    printf ("\n\tc2: %d; c3: %d", c2, c3);
    printf ("\n\tc4: %d; c5: %d", c4, c5);
    printf ("\n\tc6: %d; c7: %d", c6, c7);
    printf ("\n\t\nulos: %d", nulos);
}
```

E se o número de candidatos fosse maior que 100?
--

- Solução: um nome comum a todos os candidatos e um **índice** para identificar cada candidato:

C[0], C[1], C[2], C[3], C[4], C[5], C[6] e C[7]

- Programa para a mesma eleição usando uma variável indexada:

```
#include <stdio.h>
void main ()
{
    int n, nulos, voto, i; int c[8] = {0};
    printf ("Apuracao de eleicao:\n\n\tNumero de votos: ");
    scanf ("%d",&n); printf ("\n"); nulos = 0;
    for (i = 1; i <= n; i++) {
        printf ("\t\tVoto: "); scanf ("%d",&voto);
        if (voto >= 0 && voto <= 7) c[voto]++;
        else nulos++;
    }
    printf ("\n\nResultado:\n\n");
    for (i = 0; i <= 7; i++)
        printf ("\tc[%d]: %d\n", i, c[i]);
    printf ("\tnulos: %d", nulos);
}
```

<p>Programa muito mais simples e elegante</p>

- **int c[8]** : reserva espaço para c[0], c[1], . . . , c[7]
- **int c[8] = {2, 5, 7, 3, 1}**; faz as seguintes inicializações:
c[0] ← 2; c[1] ← 5; c[2] ← 7; c[3] ← 3; c[4] ← 1;
c[5] ← 0; c[6] ← 0; c[7] ← 0

- **Exemplo 5.1:** Ordenação de números pelo método *Bubble Sort*:

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
void main () {
    int n, i, p, aux, vetor[50]; char trocou;

    printf ("Ordenacao de numeros pelo Bubble-Sort\n\n");
    printf ("\tNumero de elementos: "); scanf ("%d",&n);
    printf ("\n\tElementos: ");
    for (i=0; i<n; i++) scanf ("%d", &vetor[i]);
    printf ("\n\nVetor desordenado:\n\n");
    for (i=0; i<n; i++) printf ("% -4d", vetor[i]);
    for (trocou = TRUE, p = n-2; p>=0 && trocou == TRUE; p--) {
        for (trocou = FALSE, i = 0; i<=p; i++)
            if (vetor[i] > vetor[i+1]) {
                aux = vetor[i]; vetor[i] = vetor[i+1];
                vetor[i+1] = aux; trocou = TRUE;
            }
    }
    printf ("\n\nVetor ordenado:\n\n");
    for (i=0; i<n; i++) printf ("% -4d", vetor[i]);
}
```

- Variável indexada com um índice é chamada de variável indexada unidimensional ou simplesmente de **vetor**.

- **Exemplo 5.2:** Procura binária em vetores

Esquema do programa:

- Ler uma relação de números supostamente ordenada;
- Se ela estiver desordenada, emitir mensagem de erro e encerrar a execução
- Senão verificar se alguns números pertencem a essa relação

```
/*          Declaracoes globais          */
```

```
#include <stdio.h>
#include <conio.h>
#define TRUE 1
#define FALSE 0
```

```
/*          Cabecalho e declaracoes locais          */
```

```
void main () {
    int n, i, inf, sup, med, posic, vetor[50], num;
    char c, achou, okrel;
```

```
/*          Leitura e escrita da relacao de numeros          */
```

```
printf ("P R O C U R A   B I N A R I A\n\n");
printf ("Numero de elementos: "); scanf ("%d",&n);
printf ("\nElementos: ");
for (i=0; i<n; i++) scanf ("%d", &vetor[i]);
printf ("\n\nRelacao dos numeros:\n\n");
for (i=0; i<n; i++) printf ("% -4d", vetor[i]);
```

```
/*          Verificacao da ordenacao da lista de numeros          */
```

```
for (i=0, okrel = TRUE; i < n-1 && okrel == TRUE; i++)
    if (vetor[i] > vetor[i+1]) okrel = FALSE;
if (okrel == FALSE) printf ("\n\n\tRelacao desordenada\n");
```

```
/* Procura de varios numeros na relacao */
```

```
else {  
    printf ("\n\n");  
    do {  
        printf ("Procura numero? (s/n): ");  
        do c = getche(); while (c != 's' && c != 'n');  
        if (c == 's') {  
            printf ("\n\tNumero: "); scanf ("%d", &num);  
            achou = FALSE; inf = 0; sup = n-1;  
            do {  
                med = (inf + sup) / 2;  
                if (num == vetor[med]) {  
                    achou = TRUE; posic = med;  
                }  
                else if (num < vetor[med]) sup = med-1;  
                else inf = med+1;  
            } while (!achou && sup >= inf);  
        }  
    }  
}
```

```
/* Resposta: o numero estah ou nao na relacao */
```

```
        if (achou) printf ("\n%d estah na posicao %d da  
            relacao\n\n", num, posic);  
        else printf ("\n%d nao estah na  
relacao\n\n", num);  
    }  
} while (c == 's');  
}  
}
```

5.1.2 – Variáveis indexadas multidimensionais

- Variáveis indexadas podem ter mais de um índice
- Variáveis indexadas com dois índices são chamadas variáveis indexadas bidimensionais ou simplesmente **matrizes**
- **Exemplo 5.3:** Contabilização dos acidentes nos cruzamentos de ruas ortogonais

```
#include <stdio.h>
void main () {
    int i, j, n, h, v, cruza[10][10] = {0};

    printf ("Mapa de acidentes:\n\n\tNumero de acidentes:
");
    scanf ("%d", &n); printf ("\n");
    for (i=1; i<=n; i++) {
        printf ("\tCruzamento: "); scanf ("%d%d", &h, &v);
        if (h>=0 && h<=9 && v>=0 && v<=9) cruza[h][v]++;
    }
    printf ("\n\n");
    for (i=0; i<=9; i++) {
        for (j=0; j<=9; j++) {
            printf ("%3d", cruza[i][j]);
        }
        printf ("\n");
    }
}
```

Pode ser mais sofisticado. (Ver página seguinte)

```
}
```

- O trecho marcado no programa anterior pode ser trocado pelo trecho a seguir, para um mapa um pouco mais elaborada:

```
for (j=0; j<=9; printf ("__%d", j), j++);  
printf ("__");  
for (i=0; i<=9; i++) {  
    printf ("\n %d|", i);  
    for (j=0; j<=9; j++) {  
        if (cruza[i][j]) printf ("%3d", cruza[i][j]);  
        else printf ("  ");  
    }  
    printf (" \n |%32c", '|');  
}  
printf ("\n -");  
for (j=0; j<=9; printf ("---"), j++);  
printf ("--");
```

- **Obs:** Pode-se trabalhar com variáveis de mais de duas dimensões.
- **Exemplo 5.4:** Achar a transposta de uma matriz quadrada:

$$\begin{bmatrix} 5 & -2 & 7 & 3 \\ 12 & 7 & -5 & -2 \\ 8 & 1 & 0 & 0 \\ 6 & -24 & 2 & 9 \end{bmatrix} \Rightarrow \begin{bmatrix} 5 & 12 & 8 & 6 \\ -2 & 7 & 1 & -24 \\ 7 & -5 & 0 & 2 \\ 3 & -2 & 0 & 9 \end{bmatrix}$$

Original	Transposta
#include <stdio.h>	
void main () {	
int mat[10][10], i, j, n, aux;	
printf ("Dimensao da matriz quadrada: ");	
scanf ("%d",&n);	
printf ("\nElementos da matriz: \n");	
for (i = 0; i < n; i++)	
for (j = 0; j < n; j++) scanf ("%d", &mat[i][j]);	
printf ("\nMatriz original:\n\n");	
for (i = 0; i < n; i++) {	
for (j = 0; j < n; j++) printf ("%4d", mat[i][j]);	
printf ("\n");	
}	
for (i = 1; i < n; i++)	
for (j = 0; j < i; j++) {	
aux = mat[i][j];	
mat[i][j] = mat[j][i];	
mat[j][i] = aux;	
}	
printf ("\nMatriz transposta:\n\n");	
for (i = 0; i < n; i++) {	
for (j = 0; j < n; j++) printf ("%4d", mat[i][j]);	
printf ("\n");	
}	
}	

5.1.3 – Definição de novos tipos de variáveis

- Até agora, os tipos usados nas declarações das variáveis são aqueles disponibilizados pela Linguagem C (**int**, **long**, **char**, **float**, **double**, etc.).
- A declaração **typedef** possibilita ao programador criar os seus próprios tipos.
- **Exemplo 5.5:** Declarações de novos tipos:

typedef int inteiro;

permite que variáveis sejam declaradas do tipo inteiro; a declaração `inteiro x;`

poderá ser feita no lugar de `int x;`

typedef int vetor[10]; **typedef** vetor matriz[10];

permitem que as declarações `vetor vet;` `matriz mat;`

sejam feitas no lugar de `int vet[10]`, `mat[10][10]`;

A declaração **typedef** `vetor matriz[10];` é equivalente à

typedef int `matriz[10][10];`

- **Exemplo 5.6:** Multiplicação de matrizes

Dadas 2 matrizes A (m por n) e B (n por p), achar outra matriz C (m por p) tal que $C = A * B$

Esquema:

$$\begin{bmatrix} A_{0,0} & A_{0,1} & \dots & A_{0,n-1} \\ A_{1,0} & A_{1,1} & \dots & A_{1,n-1} \\ \vdots & \vdots & & \vdots \\ A_{m-1,0} & A_{m-1,1} & \dots & A_{m-1,n-1} \end{bmatrix} \quad \begin{bmatrix} C_{0,0} & C_{0,1} & \dots & C_{0,p-1} \\ C_{1,0} & C_{1,1} & \dots & C_{1,p-1} \\ \vdots & \vdots & & \vdots \\ C_{m-1,0} & C_{m-1,1} & \dots & C_{m-1,p-1} \end{bmatrix}$$

$$C_{i,j} = \sum_{k=0}^{n-1} A_{i,k} * B_{k,j}$$

$$\begin{bmatrix} B_{0,0} & B_{0,1} & \dots & B_{0,p-1} \\ B_{1,0} & B_{1,1} & \dots & B_{1,p-1} \\ \vdots & \vdots & & \vdots \\ B_{n-1,0} & B_{n-1,1} & \dots & B_{n-1,p-1} \end{bmatrix}$$

O número de linhas da matriz B deve ser igual ao número de colunas da matriz A .

A multiplicação de matrizes não é comutativa, ou seja,

$$A * B \neq B * A.$$

Programa:

```
#include <stdio.h>
#define N 10
typedef int vetor[N]; typedef vetor matriz[N];

void main ()
{
    matriz A, B, C;
    int m, n, p, i, j, k, aux;

    /*
       Imprimir títulos; Ler e imprimir as matrizes A e B
    */

    for (i = 0; i < m; i++)
        for (j = 0; j < p; j++) {
            aux = 0;
            for (k = 0; k < n; k++)
                aux += A[i][k] * B[k][j];
            C[i][j] = aux;
        }

    /*
       Imprimir a matriz C
    */

}
```

- **Exercício das sub-matrizes:**

Uma **matriz B (20 por 20)** de números inteiros é subdividida em **25 sub-matrizes (4 por 4)**. Construir uma outra **matriz A (5 por 5)** na qual cada elemento contenha a **soma dos elementos** de cada uma das 25 sub-matrizes mencionadas. **Exemplo:**

Matriz B (20 por 20):

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	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220
221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260
261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280
281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320
321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340
341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360
361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380
381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400

Matriz A (5 por 5):

520	584	648	712	776
1800	1864	1928	1992	2056
3080	3144	3208	3272	3336
4360	4424	4488	4552	4616
5640	5704	5768	5832	5896

$A[i][j]$ corresponde à

sub-matriz

$B[4i : 4i+3][4j : 4j+3]$

- **Declarações:**

```
int A[5][5], B[20][20];
int i, j, x, y, aux; char c;
```

- **Formação da matriz B (20 por 20):**

```
for (i = 0; i < 20; i++)
    for (j = 0; j < 20; j++)
        B[i][j] = 20 * i + j + 1;
```

- **Escrita da matriz B (20 por 20):**

```
printf ("Matriz B (20 X 20)? (s/n): ");
c = getche ();
if (c == 's' || c == 'S') {
    printf ("\n\n");
    for (i = 0; i < 20; i++) {
        for (j = 0; j < 20; j++)
            printf ("%4d", B[i][j]);
        printf ("\n");
    }
}
```

- **Esboço do cálculo da matriz A (5 por 5):**

```
for (i = 0; i < 5; i++)
    for (j = 0; j < 5; j++)
        Cálculo do elemento A[i][j];
```

- **Sabendo que A[i][j] corresponde à sub-matriz:**

B[4i : 4i+3] [4j : 4j+3]

- **Cálculo do elemento A[i][j]:**

```
aux = 0;
for (x = 4*i; x <= 4*i + 3; x++)
    for (y = 4*j; y <= 4*j + 3; y++)
        aux += B[x][y];
A[i][j] = aux;
```

- **Cálculo completo da matriz A (5 por 5):**

```
for (i = 0; i < 5; i++)
    for (j = 0; j < 5; j++) {
        aux = 0;
        for (x = 4*i; x <= 4*i + 3; x++)
            for (y = 4*j; y <= 4*j + 3; y++)
                aux += B[x][y];
        A[i][j] = aux;
    }
```

- **Escrita da matriz A (5 por 5):**

```
printf ("\n\nMatriz A (5 por 5)? (s/n): ");
c = getche ();
if (c == 's' || c == 'S') {
    printf ("\n\n");
    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++)
            printf ("%10d", A[i][j]);
        printf ("\n");
    }
}
```

- Outro método: sejam as mesmas matrizes do exemplo anterior:

Matriz B (20 por 20):

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	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140
141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220
221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260
261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280
281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320
321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340
341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360
361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380
381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400

Matriz A (5 por 5):

520	584	648	712	776
1800	1864	1928	1992	2056
3080	3144	3208	3272	3336
4360	4424	4488	4552	4616
5640	5704	5768	5832	5896

B[i][j] será somado ao elemento

$A[\lfloor i/4 \rfloor][\lfloor j/4 \rfloor]$

ou simplesmente

$A[i/4][j/4]$

- ```
for (i = 0; i < 20; i++)
 for (j = 0; j < 20; j++)
 A[i/4][j/4] += B[i][j];
```

Os elementos de A devem ser zerados no início.

- **Exercício dos polinômios:**

Um polinômio  $P(x)$  de grau  $n$  é dado pela fórmula:

$$P(x) = A_0 + A_1x^1 + A_2x^2 + \dots + A_nx^n$$

Os coeficientes de polinômios podem ser armazenados em vetores mediante a declaração

```
typedef float polin [100];
```

Fazer um programa para ler dois polinômios e calcular:

- a) O polinômio soma desses polinômios lidos.
- b) O polinômio produto deles.

- Deve-se obter telas do tipo:

## **TRABALHO COM POLINOMIOS**

**Grau do Polinomio P1: 3**

**Coeficientes de P1 (0 a 3):**

**3 5 -2 4**

**Grau do Polinomio P2: 3**

**Coeficientes de P2 (0 a 3):**

**4 -3 0 -4**

**Polinomio P1: Grau 3**

$$+ (3) + (5)*X + (-2)*X**2 + (4)*X**3$$

**Polinomio P2: Grau 3**

$$+ (4) + (-3)*X + (-4)*X**3$$

**Somar os polinomios? (s/n): s**

**Polinomio Soma PS: Grau 2**

$$+ (7) + (2)*X + (-2)*X**2$$

**Multiplicar os polinomios? (s/n): s**

**Polinomio Produto PM: Grau 6**

$$+ (12) + (11)*X + (-23)*X**2 + (10)*X**3 + (-32)*X**4 + (8)*X**5 + (-16)*X**6$$

- O programa será apresentado por partes.

- Declarações globais, definindo o tipo *polin*:

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

```
typedef float polin[100];
```

- Declaração dos polinômios e de seu grau, além das variáveis auxiliares:

```
void main () {
 polin P1 = {0}, P2 = {0}, PS = {0}, PM = {0}, Paux;
 int n1, n2, ns, nm, naux, i, j;
 char c;
```

- Leitura do 1º. polinômio:

```
printf ("TRABALHO COM POLINOMIOS\n\n");
printf ("Grau do Polinomio P1: ");
scanf ("%d", &n1);
printf ("Coeficientes de P1 (0 a %d):\n\t", n1);
for (i = 0; i <= n1; i++)
 scanf ("%f", &P1[i]);
```

- Correção do grau do 1º. polinômio:

```
i = n1;
while (P1[i] == 0.0 && i != 0) i--;
n1 = i;
```

- Leitura do 2º. polinômio e correção de seu grau:

```
printf ("\nGrau do Polinomio P2: ");
scanf ("%d", &n2);
printf ("Coeficientes de P2 (0 a %d):\n\t", n2);
for (i = 0; i <= n2; i++)
 scanf ("%f", &P2[i]);
i = n2;
while (P2[i] == 0.0 && i != 0) i--;
n2 = i;
```

- Escrita do 1º. polinômio:

```
printf ("\n\nPolinomio P1: Grau %d\n\n\t", n1);
for (i = 0; i <= n1; i++)
 if (P1[i] != 0.0 || n1 == 0) {
 printf (" + (%g)", P1[i]);
 if (i != 0) {
 printf ("*X");
 if (i > 1)
 printf ("**%d", i);
 }
 }
}
```

- Escrita do 2º. polinômio:

```
printf ("\n\nPolinomio P2: Grau %d\n\n\t", n2);
for (i = 0; i <= n2; i++)
 if (P2[i] != 0.0 || n2 == 0) {
 printf (" + (%g)", P2[i]);
 if (i != 0) {
 printf ("*X");
 if (i > 1)
 printf ("**%d", i);
 }
 }
}
```

- Soma dos polinômios, correção de seu grau e escrita:

```

printf ("\n\nSomar os polinomios? (s/n): ");
c = getche ();
if (c == 's' || c == 'S') {
 ns = (n1 > n2) ? n1 : n2;
 for (i = 0; i <= ns; i++)
 PS[i] = P1[i] + P2[i];
 i = ns;
 while (PS[i] == 0.0 && i != 0) i--;
 ns = i;
 printf ("\n\nPolinomio Soma PS: Grau %d\n\n", ns);
 for (i = 0; i <= ns; i++)
 if (PS[i] != 0.0 || ns == 0) {
 printf (" + (%g)", PS[i]);
 if (i != 0) {
 printf ("*X");
 if (i > 1)
 printf ("**%d", i);
 }
 }
 }
}

```

- O produto dos polinômios segue o esquema:

```

PM = 0;
for (j = 0; j <= n2; j++)
 if (P2[j] != 0) {
 Paux = P1 * P2[j]; PM += Paux;
 }

```

- Código para o produto dos polinômios, correção de seu grau e escrita:

```

printf ("\n\nMultiplicar os polinomios? (s/n): ");
c = getche ();
if (c == 's' || c == 'S') {
 nm = n1 + n2;
 for (j = 0; j <= n2; j++)
 if (P2[j] != 0.0) {
 for (i = 0; i < 100; i++) Paux[i] = 0;
 for (i = 0; i <= n1; i++)
 Paux[i+j] = P1[i] * P2[j];
 naux = n1 + j;
 for (i = 0; i <= naux; i++)
 PM[i] = Paux[i] + PM[i];
 }
 printf ("\n\nPolinomio Produto PM: Grau %d\n\n", nm);
 for (i = 0; i <= nm; i++)
 if (PM[i] != 0.0 || nm == 0) {
 printf (" + (%g)", PM[i]);
 if (i != 0) {
 printf ("*X");
 if (i > 1)
 printf ("**%d", i);
 }
 }
 }
}

```

## 5.2 – Cadeia de Caracteres

- Constantes cadeias de caracteres: seqüência de caracteres entre aspas “ ”:

“abcDEF;”

- Caracteres componentes de uma cadeia:
  - Os caracteres entre aspas.
  - O caracter ‘\0’ (caractere nulo).

- A cadeia acima tem os seguintes caracteres:

‘a’ ‘b’ ‘c’ ‘D’ ‘E’ ‘F’ ‘;’ ‘\0’

- Variáveis do tipo cadeia de caracteres são variáveis indexadas do tipo **char**.

- Inicialização: a declaração

`char s[ ] = “abcde”;` equivale a

`char s[ ] = { ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘\0’ };`

reservando espaço para `s[0]`, `s[1]`, ..., `s[5]`

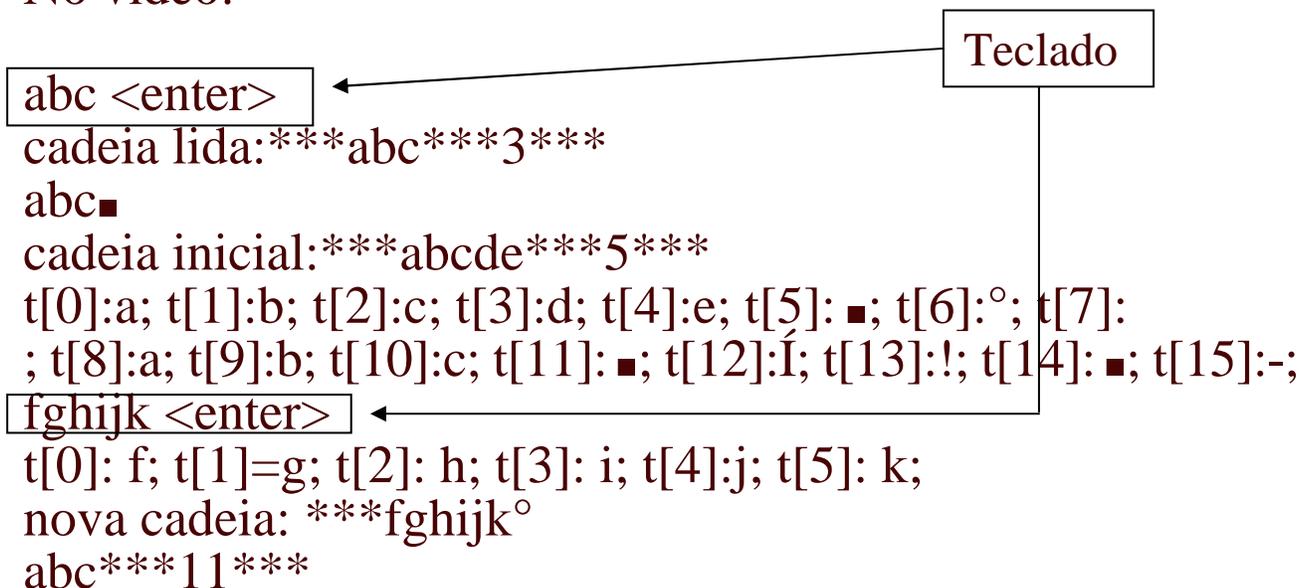
- Leitura de cadeias de caracteres: na função *scanf*, o formato **%s** lê uma seqüência de caracteres não brancos e diferentes de <enter>.
  - Cuidado para não ultrapassar a área reservada.
- Exibição de cadeias de caracteres: na função *printf*, o formato **%s** vai exibindo todos os caracteres, até encontrar um caractere nulo ‘\0’.
- **Função strlen(s)**: fornece o número de caracteres da cadeia armazenada na variável s, antes do caracter nulo ‘\0’.

strlen( ) está no arquivo <string.h>

- **Exemplo 5.7:** seja o programa.

```
#include <stdio.h>
#include <string.h>
void main () {
 char s[10], c, t[] = "abcde"; int i;
 scanf ("%s", s);
 printf ("cadeia lida:***%s***%d***\n", s, strlen(s));
 for (i = 0; i <= strlen(s); i++) printf ("%c", s[i]);
 scanf ("%c", &c);
 printf ("\ncadeia inicial:***%s***%d***\n", t, strlen(t));
 for (i = 0; i <= 15; i++) printf (" t[%d]: %c;", i, t[i]);
 printf ("\n");
 for (i = 0; i <= 5; i++) {
 scanf ("%c", &t[i]); printf (" t[%d]: %c;", i, t[i]);
 }
 printf ("\nnova cadeia:***%s***%d***", t, strlen(t));
}
```

No vídeo:



- **Função `strcat` (*cadeia1*, *cadeia2*):** concatena as duas cadeias e coloca o resultado em *cadeia1*.
- **Função `strcmp` (*cadeia1*, *cadeia2*):** o resultado é  $< 0$ ,  $= 0$ , ou  $> 0$ , se *cadeia1* é lexicograficamente menor, igual ou maior que *cadeia2*.
- **Função `strcpy` (*cadeia1*, *cadeia2*):** a *cadeia2* é copiada em *cadeia1*, apagando o valor anterior de *cadeia1*.
- **Exemplo 5.8:** Procura binária de nomes.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
#define N 10
typedef char nome[N];
void main () {
 int n, i, inf, sup, med, posic; char c, achou, okrel;
 nome vetor[50], name;

 printf ("Numero de elementos: "); scanf ("%d",&n);
 printf ("\nEntre com os nomes\n");
 for (i=0; i<n; scanf("%s", vetor[i]), i++);
 printf ("\n\nRelacao de nomes:\n\n");
 for (i=0; i<n; printf("%s\n", vetor[i]), i++);
 for (i=0, okrel=TRUE; i<n-1 && okrel; i++)
 if (strcmp(vetor[i],vetor[i+1]) > 0) okrel = FALSE;
```

```

if (!okrel) printf ("\n\n\tRelacao desordenada\n");
else {
 printf ("\n\n");
 do {
 printf ("Procura? (s/n) ");
 do c = getche(); while (c != 's' && c != 'n');
 if (c == 's') {
 printf (" Nome: "); scanf ("%s", name);
 achou = FALSE; inf = 0; sup = n-1;
 do {
 med = (inf + sup) / 2;
 if (strcmp(name, vetor[med]) == 0) {
 achou = TRUE; posic = med;
 }
 else if (strcmp(name, vetor[med]) < 0)
 sup = med-1;
 else inf = med+1;
 } while (!achou && sup>=inf);
 if (achou) printf ("\n%s estah na posicao
 %d da relacao\n\n", name, posic);
 else printf ("\n%s nao estah na relacao
 \n\n",name);
 }
 } while (c == 's');
}
}

```

- A função *scanf* encerra a leitura de uma cadeia quando encontra um caractere *espaço\_em\_branco*.
- Cadeias contendo espaços em branco **não podem ser lidas** por essa função.
- Função *gets ( cad )*: lê uma cadeia de caracteres encerrada por *enter* e guarda na variável **cad** (do tipo cadeia de caracteres);

A cadeia lida pode ter espaços em branco.

- **Exemplo 5.9:** utilização da função *gets ( )*

O programa a seguir pode ler, armazenar e imprimir várias frases encerradas por um *enter*

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

typedef char cadeia[30];

void main () {
 int i, n;
 cadeia frases [25];
```

```

printf ("Numero de frases (ate 25): ");
scanf ("%d", &n);
printf ("\n");
for (i = 0; i <= n-1; i++) {
 printf ("Digite a %da frase: ", i+1);
 fflush (stdin);
 gets (frases[i]);
}
printf ("\nListagem das frases:\n\n");
for (i = 0; i <= n-1; i++)
 printf ("%d) %s\n", i+1, frases[i]);
}

```

Resultados de uma execução:

Numero de frases (ate 25):

Digite a 1a frase:   
 Digite a 2a frase:   
 Digite a 3a frase:   
 Digite a 4a frase:

Digitadas

Listagem das frases:

- 1) Estou gostando de CES-10
- 2) A Linguagem C eh fantastica
- 3) The show must go on
- 4) Pink Floyd eh a melhor banda

## 5.3 – Tipos Enumerativos

- Tipos vistos até agora: **int**, **float**, **char**, e assemelhados (**long**, **double**, **unsigned**, etc.)
- Pode-se criar, em C, tipos para representar
  - dias da semana,
  - meses do ano,
  - cores,
  - estado civil de pessoas,
  - naipes de baralho, etc
- As declarações

**enum** diasemana {dom, seg, ter, qua, qui, sex, sab};

**enum** mês {jan, fev, mar, abr, mai, jun, jul, ago, set, out, nov, dez};

**enum** cor {branco, amarelo, verde, azul, marrom, preto};

criam os tipos **enum** diasemana, **enum** mês, **enum** cor.

- Variáveis podem ser declaradas usando esses tipos:

**enum** diasemana hoje, ontem, amanhã;

**enum** mês mescorrente, mesnascimento;

**enum** cor cor1, cor2, cor3;

- Atribuições e comparações podem ser feitas:

```
cor1 = azul;
if(ontem == seg) hoje = ter;
```

- Por “default”, internamente:

```
dom = 0, jan = 0, branco = 0, então
seg = 1, ter = 2, ... , sab = 6, etc...
```

- Pode-se declarar, alternativamente:

```
enum legume { vagem = 2, beterraba, cenoura, jiló = 8,
mandioca, chuchu = 9, abobora = 2, batata, cebola,
quiabo } veg1, veg2, veg3;
```

- Neste caso:
  - veg1, veg2, veg3 são variáveis do tipo enum legume;
  - internamente

```
vagem = abobora = 2
beterraba = batata = 3
cenoura = cebola = 4
quiabo = 5
jiló = 8
chuchu = mandioca = 9
```

- Pode-se também usar **typedef**:

```
enum diasemana { dom, seg, ter, qua, qui, sex, sab };
typedef enum diasemana diasemana;
diasemana hoje, ontem, amanha;
```

- **Exemplo 5.9:** Programa para ler o dia de hoje e calcular o dia de ontem e o de amanhã.

```
#include <stdio.h>
#define N 10
enum diasemana { dom, seg, ter, qua, qui, sex, sab };
typedef enum diasemana diasem;
typedef char nome[N];

void main(){
 diasem hoje, ontem, amanha;
 nome nomedia[7] = {"domingo", "segunda", "terca",
"quarta",
 "quinta", "sexta", "sabado"};
 do{
 printf ("Entre com o dia de hoje (0,1,2,3,4,5,6): ");
 scanf("%d", &hoje);
 if (hoje >= dom && hoje <= sab){
 ontem = (hoje + 6) % 7;
 amanha = (hoje + 1) % 7;
 printf("\n\thoje = %8s; ontem = %8s; amanha =
%8s;\n\n",
 nomedia[hoje], nomedia[ontem], nomedia[amanha]);
```

```
 }
 } while (hoje >= dom && hoje <= sab);
}
```

## 5.4 – Estruturas Não Homogêneas

- Variáveis indexadas e cadeias de caracteres são estruturas homogêneas:
  - conjuntos de elementos do mesmo tipo.
- Estruturas não homogêneas são muito úteis.
- **Exemplo 5.10:** cadastro das informações dos empregados de uma empresa.

Informações: nome, rua, número, sexo, estado civil

```
#include <stdio.h>
#define N 15
typedef char cadeia [N];
enum sexo {masc, fem};
enum estcivil {solt,cas,viuvo,desq,div};
typedef enum sexo sexo;
typedef enum estcivil civil;
struct endereco{cadeia rua; int numero;};
typedef struct endereco endereco;
struct empregado{
 cadeia nome; endereco ender;
 sexo sex; civil estcivil;
};
typedef struct empregado empregado;
```

```

void main(){
 int i, n = 3;
 empregado rel[10];

 printf ("Informacoes sobre Empregados da Empresa \n");
 for(i=0;i<n;i++) {
 printf ("\nEmpregado n.o %d:\n", i);
 printf ("\tNome : "); scanf("%s", rel[i].nome);
 printf ("\tRua : "); scanf("%s", rel[i].ender.rua);
 printf ("\tNumero: ");
 scanf("%d", &rel[i].ender.numero);
 printf ("\tSexo (0-Masc, 1-Fem) : ");
 scanf("%d", &rel[i].sex);
 printf ("\tEstado civil (0-solt,1-cas,2-viuv,3-desq,4-div) : ");
 scanf("%d", &rel[i].estcivil);
 }
 printf ("\nListagem dos Empregados:\n");
 for(i=0;i<n;i++)
 printf("% 17s% 17s% 5d% 3d% 3d\n",rel[i].nome,
 rel[i].ender.rua, rel[i].ender.numero,rel[i].sex,
 rel[i].estcivil);
}

```

- O operador ‘.’ especifica um campo da estrutura. Ex.:

```
rel[i].ender.rua; rel[i].sex;
```

- Inicialização de estruturas: similar à inicialização de variáveis indexadas.
- **Exemplo 5.11:** seja o programa.

```
#include <stdio.h>
struct endereco{
 char rua[30];
 int numero;
 char bairro[15];
};
typedef struct endereco endereco;

void main(){
 endereco e1 = {"Av. Nove de Julho",1300,"Centro"}, e2;
 e2 = e1;
 e1.rua[14] = 'n';
 printf("%31s%10d%17s\n", e1.rua, e1.numero, e1.bairro);
 printf("%31s%10d%17s\n", e2.rua, e2.numero, e2.bairro);
}
```

Resultado:

```
Av. Nove de Junho 1300 Centro
Av. Nove de Julho 1300 Centro
```

e2=e1; equivale a: →

```
strcpy (e2.rua ,e1.rua);
e2.numero = e1.numero;
strcpy (e2.bairro ,e1.bairro);
```

- **Exemplo 5.12:** inicialização de vetor de estruturas.

```
#include <stdio.h>
struct complexo { float real, imag; };
typedef struct complexo complexo;

void main(){
 int i,j;
 complexo a[3][3]={
 {{1.0, -0.1},{2.0, -0.2},{3.0, 4.3}},
 {{4.0, -3.4},{5.0, 4.1},{6.0, -2.6}}
 };
 for (i = 0; i < 3; i++) {
 for (j = 0; j < 3; j++)
 printf ("%5.1f) + i(%5.1f) ", a[i][j].real, a[i][j].imag);
 printf ("\n");
 }
}
```

Resultado:

```
(1.0)+i(-0.1) (2.0)+i(-0.2) (3.0)+i(4.3)
(4.0)+i(-3.4) (5.0)+i(4.1) (6.0)+i(-2.6)
(0.0)+i(0.0) (0.0)+i(0.0) (0.0)+i(0.0)
```

Os elementos `a[2][0]`, `a[2][1]` e `a[2][2]` foram inicializados com 0.

## 5.5 – Alternação de Tipos

- Tipo *union*: define um conjunto alternativo de tipos que podem ser armazenados numa mesma posição de memória.
- **Exemplo 5.13:**

```
#include <stdio.h>
union int_or_float{ int i; float f; };
typedef union int_or_float numero;

void main(){
 numero n;
 n.i = 4444;
 printf ("i:%10d f:%16.10e\n", n.i, n.f);
 n.f = 4444;
 printf ("i:%10d f:%16.10e\n", n.i, n.f);
}
```

### Resultado:

```
i: 4444 f: 1.2992640975e+12
i: -8192 f: 4.4440000000e+03
```

- O mesmo local da memória é visto como inteiro e como real.
- Os membros de um tipo ‘union’ podem ser estruturas ou

outros tipos ‘union’.

- **Exemplo 5.14:** observar as declarações:

```
struct flor{
 char nome[10];
 enum {vermelho, branco, azul} cor;
};
struct fruta{
 char nome[10]; int calorias;
};
struct legume {
 char nome[10]; int calorias, t_cozimento;
};
typedef struct flor flor;
typedef struct fruta fruta;
typedef struct legume legume;

union flor_fruta_legume { flor fl; fruta ft; legume lg; };
typedef union flor_fruta_legume vegetal;
```

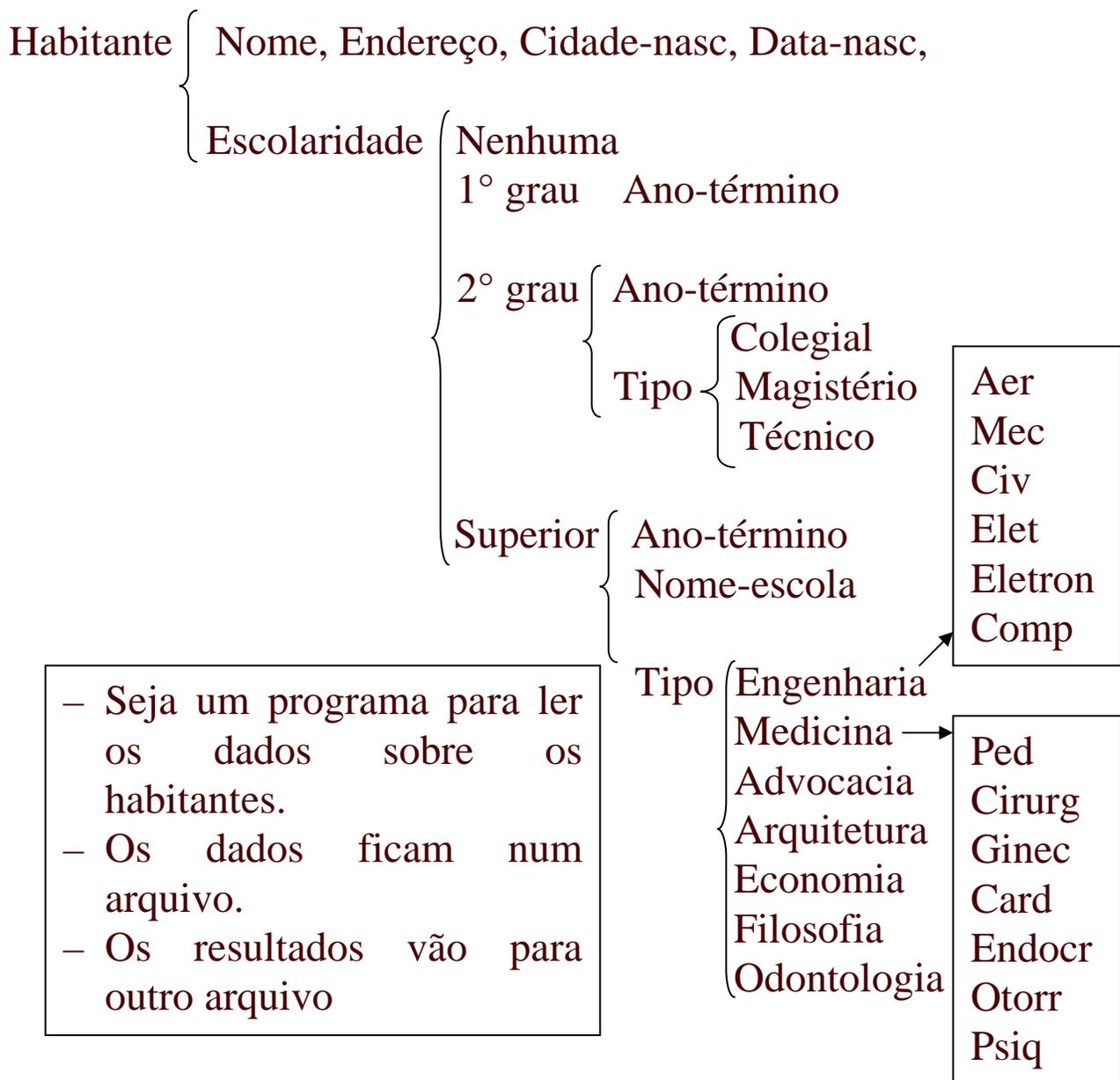
Pode-se usar comandos como:

```
vegetal veg;
veg.lg.t_cozimento = 7;
```

## 5.6 – Estruturas de Campos Alternativos

- São estruturas cujos campos podem ser do tipo *union*.
- **Exemplo 5.15:** aplicação de *union*'s dentro de *struct*'s: Seja um cadastro dos habitantes de uma cidade.

Informações sobre um habitante:



```
#include <stdio.h>
#include <conio.h>
#define N1 15
#define N2 10
```

```
/* Declaracoes dos tipos enumerativos */
```

```
enum escolaridade {nehn, pgrau, sgrau, sup};
typedef enum escolaridade escolaridade;
```

```
enum tseggrau {col, mag, tec};
typedef enum tseggrau tseggrau;
```

```
enum tsuperior {eng, med, adv, arq, econ, fil, odont};
typedef enum tsuperior tsuperior;
```

```
enum tengenharia {aer, mec, civ, elet, eletron, comp};
typedef enum tengenharia tengenharia;
```

```
enum tmedicina {ped, cir, gin, card, endocr, otorr, psiq};
typedef enum tmedicina tmedicina;
```

```
/* Declaracoes das estruturas */
```

```
struct primgrau { int ano_term; };
typedef struct primgrau primgrau;
```

```
struct seggrau { int ano_term; tseggrau tipo; };
typedef struct seggrau seggrau;
```

```
union fsuperior { tengenharia teng; tmedicina tmed; };
typedef union fsuperior fsuperior;
```

```
struct superior{
 int ano_term; char nome_escola[N1];
 tsuperior tipo; fsuperior form;
};
typedef struct superior superior;
```

```
struct data { int dia, mes, ano; };
typedef struct data data;
```

```
union formacao { primgrau pg; seggrau sg; superior sp; };
typedef union formacao formacao;
```

```
struct habitante {
 char nome[N1]; char ender[N1];
 char cid_natal[N2]; data data_nasc;
 escolaridade esc; formacao form;
};
```

```
typedef struct habitante habitante;
```

```

/*
Inicio dos comandos; leitura do numero de habitantes
*/

void main () {
 FILE *fl, *f2; habitante hab[10]; int i, n;
 fl = fopen("infile2.cpp", "r");
 f2 = fopen("outfile2", "w");
 fscanf (fl, "%d",&n);

/* Leitura dos dados sobre os habitantes */

 for (i = 0; i < n; i++) {
 fscanf (fl, "%s %s %s %d %d %d %d", hab[i].nome,hab[i].ender,
 hab[i].cid_natal,&hab[i].data_nasc.dia, &hab[i].data_nasc.mes,
 &hab[i].data_nasc.ano, &hab[i].esc);
 switch (hab[i].esc){
 case pgrau:
 fscanf(fl,"%d",&hab[i].form.pg.ano_term);
 break;
 case sgrau:
 fscanf(fl,"%d%d",&hab[i].form.sg.ano_term,
 &hab[i].form.sg.tipo);
 break;
 case sup:
 fscanf(fl,"%d%s%d",&hab[i].form.sp.ano_term,
 hab[i].form.sp.nome_escola, &hab[i].form.sp.tipo);
 switch (hab[i].form.sp.tipo){
 case eng:
 fscanf(fl,"%d",&hab[i].form.sp.form.teng);
 break;
 case med:
 fscanf(fl,"%d",&hab[i].form.sp.form.tmed);
 }
 }
 }
 }
}

```

```

/* Saida dos resultados */

for (i = 0; i < n; i++) {
 fprintf (f2, "\n%-15s %-15s %-10s%3d/%2d/%4d%3d",
 hab[i].nome,hab[i].ender, hab[i].cid_natal,
 hab[i].data_nasc.dia, hab[i].data_nasc.mes,
 hab[i].data_nasc.ano, hab[i].esc);
 switch (hab[i].esc){
 case pgrau:
 fprintf (f2, "%5d",hab[i].form.pg.ano_term);
 break;
 case sgrau:
 fprintf (f2, "%5d%3d", hab[i].form.sg.ano_term,
 hab[i].form.sg.tipo);
 break;
 case sup:
 fprintf (f2, "%5d%3d %-10s",
 hab[i].form.sp.ano_term, hab[i].form.sp.tipo,
 hab[i].form.sp.nome_escola);
 switch (hab[i].form.sp.tipo) {
 case eng:
 fprintf (f2, "%3d",hab[i].form.sp.form.teng);
 break;
 case med:
 fprintf (f2,
 "%3d",hab[i].form.sp.form.tmed);
 break;
 }
 }
 }
}

```

