Manipulação

de

Strings

Lendo e Escrevendo Strings

Uma string é um vetor de caracteres (texto):

char nome[tamanho];

O texto que vamos gravar em uma string não precisa ocupar todos os caracteres do vetor. Por isso, utiliza-se o '\0' para indicar onde o texto termina.

Exemplo:
```
char nome[10];
nome[0] = 'J';
nome[1] = 'o';
nome[2] = 's';
nome[3] = 'é';
nome[4] = '\0';
```

Lendo e Escrevendo Strings

Este tipo de inicialização somete pode ser realizado na declaração da string.

Exemplo:

```
char nome[10] = "José";
        ou
char nome[10] = {'J','o','s','é','\0'};
```

Lendo e Escrevendo Strings

Utilizando %s no scanf

Exemplo:

```
char nome[10];
scanf("%s", &nome[0]);
```

Podemos também utilizar apenas o nome do vetor, pois ele representa um ponteiro para o primeiro elemento.

```
scanf("%s", nome);
```

Atenção!!!! A função scanf realiza a leitura até encontrar um espaço, depois encerra a leitura e coloca o caracter terminador '\0'.

Lendo e Escrevendo Strings

Utilizando %s no printf

Imprime uma sequência de caracteres até encontrar o finalizador de string '\0'.

Exemplo:

```
char nome[10];
scanf("%s", nome);

printf("Seu nome é %s.", nome );
```

Lendo e Escrevendo Strings

Utilizando 'scanf' sem truncas palavras separadas por espaços.

Para ler uma string sem truncar palavras separadas por espaços usando scanf, é possível usar o especificador de formato %[^ \n].

Exemplo:

```
char str[100];
scanf(" %99[^\n]", str);
```

Lendo e Escrevendo Strings

Exemplo:

```
char str[100];
scanf(" %99[^\n]", str);
```

- '%[^\n]' diz para scanf ler todos os caracteres até encontrar uma nova linha (\n).
- O espaço antes do % serve para consumir qualquer espaço em branco (espaços, tabulações, quebras de linha) antes de ler a string.
- 99 limita o número máximo de caracteres lidos para 99 para evitar o estouro do buffer, deixando espaço para o caractere nulo final.
- str é o ponteiro para a string onde os caracteres lidos serão armazenados.

Esta instrução irá ler a string até encontrar uma quebra de linha ou até atingir o limite de 99 caracteres (o que ocorrer primeiro), permitindo que palavras separadas por espaços sejam lidas corretamente.

Lendo e Escrevendo Strings

Utilizando função gets

Armazena uma sequência de caracteres em uma string finalizando com '\0' ao teclar <enter>.

Atenção!!! Não é possível especificar o número máximo de caracteres a serem lidos, é possível ler caracteres além do tamanho da string passada como parâmetro, causando uma falha de segurança conhecida como *buffer overflow*.

Exemplo:

```
char nome[10];
gets(nome);

printf("Seu nome é %s.", nome);
```

Lendo e Escrevendo Strings

Utilizando função fgets()

Sintaxe: char * fgets (char * string, int tamanho, FILE * fluxo);

Armazena uma sequência de caracteres (indicando o limite) em uma string finalizando com '\0' ao teclar <enter>.

A função lê até o (*tamanho* - 1) pois devemos contar o espaço para o NULL ('\0') no final da *string*.

Exemplo:

```
char nome[10];
fgets(nome, 10, stdin); // stdin indica a entrada padrão é lida do teclado

printf("Seu nome é %s.", nome);
```

fgets() – Ajuste de entrada

A função fgets() captura o caracter salto de linha '\n', e muitas vezes este caracter não é desejado no resultado final dos programas. Sendo assim, segue codificação que elimina este caracter da string.

```
6   #include <stdio.h>
7   #include <locale.h>
8   #include <string.h>
9
10
11  int main()
12  {
13      char nome[60];
14
15
16      setlocale(LC_ALL, "Portuguese");
17
18      printf("Entre com seu nome: ");
19      fflush(stdin);              // limpa buffer de teclado
20      fgets(nome, sizeof(nome), stdin);    // captura os caracteres via teclado
21      nome[strlen(nome)-1] = '\0';         // elimina o salto de linha que é incluido pela função fgets;
22
23      printf("\nSeu nome é %s.", nome);
24  }
25
```

Manipulação de strings

Função strcpy()

Copia uma string origem para uma string destino

Sintaxe: strcpy (string_dest, string_orig);

Exemplo:

```
#include <stdio.h>
#include <string.h>
int main ( )
{
   char texto [200], novotexto[200], console[20];
   printf("Digite o texto: ");
   gets(texto);
   strcpy(novotexto, texto); //Copia texto para novotexto
   strcpy(console, "Adeus"); //Copia Adeus para console
   printf ("%s %s", console, novotexto);
}
```

Manipulação de strings

Função strcat()

Concatena uma string com outra.

Sintaxe: strcat (string_resultado, string_adicionada);

Exemplo:

```
#include <stdio.h>
#include <string.h>
int main ( )
{
   char texto [200], novotexto[200] = "Meu livro";
   printf("Digite o texto: ");
   gets(texto);
   strcat(novotexto, texto);
   printf ("%s ", novotexto);
}
```

Manipulação de strings

Função strcmp()

Compara uma string com outra.
Retorno da função: = 0, se s1 for igual s2
< 0 se s1 for menor que s2
> 0 se s1 for maior que s2

Sintaxe: strcmp (nome_str1, nome_str2);

Exemplo:

```
#include <stdio.h>
#include <string.h>
int main ( )
{
   char texto1 [100], texto2[200];
   printf("Digite o primeiro texto: ");
   gets(texto1);
   printf("Digite o segundo texto: ");
   gets(texto2);
   if (strcmp(texto1, texto2) == 0)
         printf ("As strings sao iguais");
   else
         printf ("As strings sao diferentes");
}
```

Manipulação de strings

Função strlwr()

Converte uma String em minúscula.

Sintaxe: strlwr (nome_string);

Função strupr()

Converte a String em maiúscula.

Sintaxe: strupr (nome_string);

OBS: Essas funções são da biblioteca ctype.h

Manipulação de strings

Strings que representam números

Uma cadeia de caracteres decimal é qualquer sequência de dígitos demais (caracteres 0(zero) a 9) possivelmente precedidos dos caracteres + ou -. Uma string decimal é qualquer string ASCII que representa uma cadeia de caracteres decimal.

Consulte a tabela ASCII:

http://www.asciitable.com/

A pergunta é:

Como converter um string decimal no correspondente número do tipo int?

Fatec Santana de Parnaíba — CPS Centro Paula Souza — São Paulo Governo do Estado

Tabela ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	[Sem título]		116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125			117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Tabela ASCII – Códigos Extendidos

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	n
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Manipulação de strings

Função atoi()

A função atoi() da biblioteca `stdlib` recebe uma string decimal e devolve o correspondente `int`.

```
int i;
char numero[7];
printf("Entre com uma string decimal: ");
fgets(numero, sizeof(numero), stdin);
numero[strlen(numero)-1] = '\0';

i = atoi(numero);

printf("%d", i);
```

Manipulação de strings

Função atol()

A função atol() da biblioteca `stdlib` recebe uma string decimal e devolve o correspondente long.

```
long i;
char numero[10];
printf("Entre com uma string decimal: ");
fgets(numero, sizeof(numero), stdin);
numero[strlen(numero)-1] = '\0';

i = atol(numero);

printf("%ld", i);
```

Manipulação de strings

Função atof()

A função atof() da biblioteca `stdlib` recebe uma string decimal e devolve o correspondente double (ponto flutuante).

```
double i;
char numero[10];
printf("Entre com uma string decimal: ");
fgets(numero, sizeof(numero), stdin);
numero[strlen(numero)-1] = '\0';

i = atof(numero);

printf("%f", i);
```

Desafio

Desenvolva um programa que capture uma string com o máximo 90 caracteres. Após a entrada da string, o programa deve totalizar o número de caracteres da string (quantidade de repetições) e apresente ao usuário o resumo da totalização.
O totalizador deve ser um vetor de inteiros que corresponda a todos os caracteres da tabela ASCII. Dica utilize se necessário as funções itoa() ou atoi() da biblioteca C.

Exemplo: "Marcio Silva"
**** Resumo ****
M = 1
a = 2
r = 1
c = 1
i = 2
o = 1
S = 1
l = 1
v = 1
branco = 1
Total de caracteres = 12

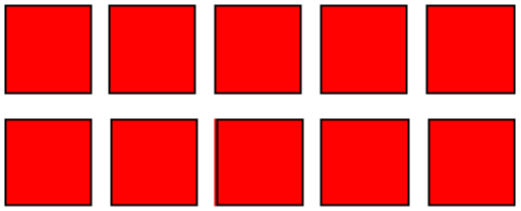
Matrizes

Batalha Naval com Desafio

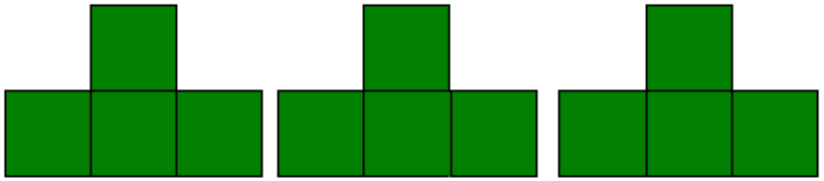
	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								

PEÇAS:

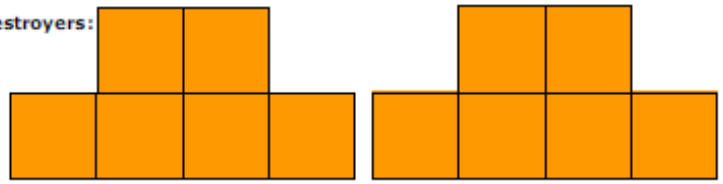
10 Submarinos



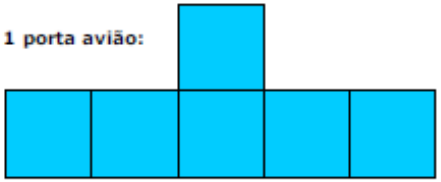
3 cruzadores:



2 destroyers:



1 porta avião:



Definições

Existem definições diferentes para variáveis compostas no mundo acadêmico.

Exemplo:

Para o autor Herbert Schidt (livro C Completo e Total), as variáveis compostas são representadas apenas por **matrizes** (unidimensionais, bidimensionais e multidimensionais). E uma string é uma matriz unidimensional.

Para o autor H. M. Deitel (livro C++ Como Programar), as variáveis compostas são representadas apenas por arrays (unidimensionais, bidimensionais e multidimensionais). E uma string é um array unidimensional.

Para outros autores vetores e arrays são a mesma coisa e sempre são unidimensionais. Já as matrizes podem ser bidimensionais e multidimensionais, ou seja, a matriz é um vetor de vetores.

Seguindo a Ementa

Conforme ementa do curso vamos seguir a seguinte definição:

Vetores e arrays são a mesma coisa e sempre são unidimensionais. Já as matrizes podem ser bidimensionais e multidimensionais, ou seja, a matriz é um vetor de vetores.

Matrizes bidimensionais

Matrizes bidimensionais são listas de informações do mesmo tipo, que são armazenadas em posições contíguas de memória em uma ordem que pode ser acessada por dois índices. A forma mais simples de matriz multidimensional é uma matriz bidimensional – um vetor de vetores unidimensionais. Para declararmos uma matriz bidimensional de valores inteiros de tamanho 10,20, escrevemos:

```
int valores[10][20];
```

Verifique que desta forma estamos declarando 200 variáveis do tipo inteiro, ou seja, 10 x 20 = 200.

Matriz bidimensional (representação)

Neste exemplo, num[0][0] tem o valor 1, num[0][1], o valor 2, num[0][2], o valor 3 e assim por diante.

num [t] [i]

	Coluna 0	Coluna 1	Coluna 2	Coluna 3
Linha 0	1	2	3	4
Linha 1	5	6	7	8
Linha 2	9	10	11	12

Matrizes bidimensionais (utilização de memória)

Matrizes bidimensionais são armazenas em uma matriz linha-coluna, onde o primeiro índice indica a linha e o segundo, a coluna. Isso significa que o índice mais à direita varia mais rapidamente do que o mais à esquerda quando acessamos os elementos da matriz na ordem em que eles estão realmente armazenados na memória.
Para matrizes bidimensionais podemos utilizar a seguinte fórmula para calcular o número de bytes de memória necessários para armazená-la:

quantidade de memória = tamanho do índice 1 * tamanho do índice 2 * tamanho do tipo

Exemplo: int valores[10][20]; << se um inteiro ocupa 4 bytes na memória >>>>

qtde_memoria = 10 * 20 * 4 << total de memória ocupada é 800 bytes >>

Matrizes multidimensionais

A linguagem C permite matrizes com mais de duas dimensões. O limite exato, se existe, é determinado por seu compilador. A sintaxe geral da declaração de uma matriz multidimensional é:

tipo nome_variavel [tamanho1] [tamanho2] [tamanho3]... [tamanhoN];

Exemplo: int valores[10][20][2]; << se um inteiro ocupa 4 bytes na memória >>>>

qtde_memoria = 10 * 20 * 2 * 4 << total de memória ocupada é 1.600 bytes >>

Matrizes multidimensionais

Matrizes de três ou mais dimensões não são frequentemente usadas devido à quantidade de memória de que eles necessitam. Por exemplo, uma matriz de quatro dimensões do tipo caractere e com o tamanhos 10,6,9,4 requer: 10 * 6 * 9 * 4, ou seja, 2.160 bytes de memória.

O armazenamento necessário cresce exponencialmente com o número de dimensões.

Em matrizes multidimensionais, toma-se tempo do computador para calcular cada índice. Isso significa que acessar um elemento em uma matriz multidimensional é mais lento do que acessar um elemento de um vetor unidimensional.

Matrizes multidimensionais (utilização de memória)

Para matrizes multidimensionais podemos utilizar a seguinte fórmula para calcular o número de bytes de memória necessários para armazená-la:

quantidade de memória = tam. índice 1 * tam. índice 2 * .... * tam. índice N * tamanho do tipo

Exemplo: int valores[10][20][2]; << se um inteiro ocupa 4 bytes na memória >>>>

qtde_memoria = 10 * 20 * 2 * 4 << total de memória ocupada é 1.600 bytes >>

Matrizes multidimensionais – declaração

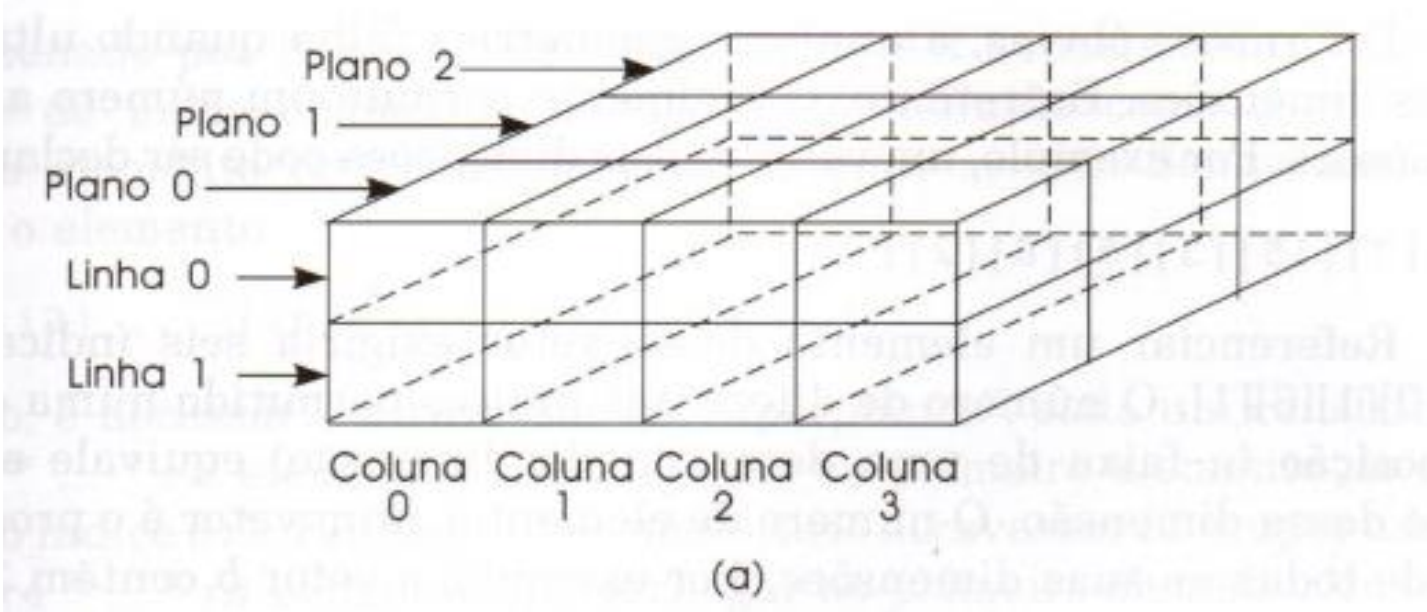
A linguagem C permite a declaração de matrizes com mais de duas dimensões. Por exemplo, uma matriz tridimensional pode ser declarada por:

```
int b[3][2][4];
```

Um elemento desta matriz é especificado por 3 (três) índices, tais como b[2][0][3]. O primeiro índice especifica um número do plano, o segundo índice, um número da linha e o terceiro, um número de coluna. Tais valores são úteis quando um valor é determinado por três entradas. Por exemplo, uma matriz de temperaturas poderia ser indexado por latitude, longitude e altitude.

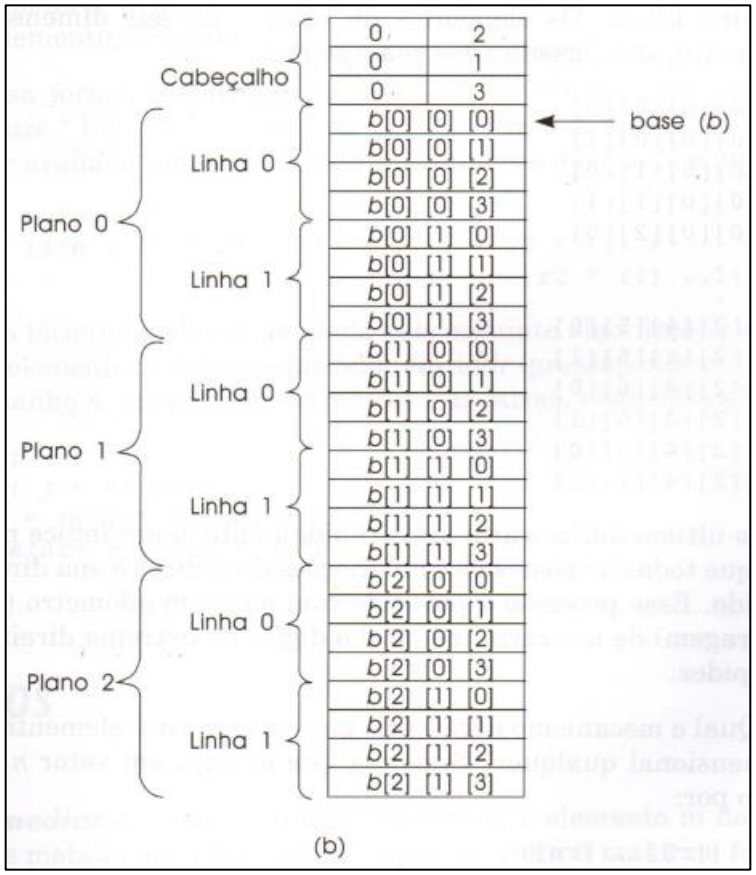
Matrizes multidimensionais
representação da matriz int b[3][2][4];

O primeiro índice especifica um número do plano, o segundo índice, um número da linha e o terceiro, um número de coluna. Tais valores são úteis quando um valor é determinado por três entradas. Por exemplo, uma matriz de temperaturas poderia ser indexado por latitude, longitude e altitude.



Matrizes multidimensionais
representação da matriz int b[3][2][4];

Representação de acesso aos elementos da matriz através de seus índices.



Conversões - Casts

Uma conversão permite que uma expressão seja tratada como sendo de um tipo específico. Uma conversão pode fazer com que uma expressão de um determinado tipo seja tratada como tendo um tipo diferente, ou pode causar uma expressão sem um tipo para obter um tipo. As conversões podem ser implícitas ou explícitas, e isso determina se uma conversão explícita é necessária. Por exemplo, a conversão de tipo **int** para tipo **long** é implícita, portanto, as expressões do tipo int podem ser tratadas implicitamente como tipo long . A conversão oposta, de tipo long para tipo int , é explícita e, portanto, uma conversão explícita é necessária.

```
int a = 123;
long b = a;          // conversão implicita de int para long
int c = (int) b;     // conversão explicita de long para int
```

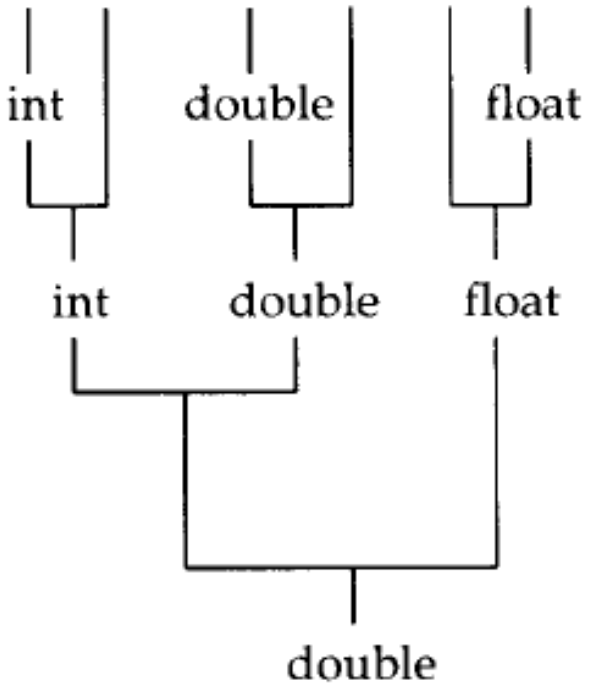
Conversões – Casts (Exemplo)

Para garantir que a expressão x / 2 (x dividido por 2) resulte em um valor do tipo **float**, escreva

```
(float) x / 2;
```

Conversões – Casts (Exemplos de conversões implícitas)

```
char ch;
int i;
float f;
double d;
result = (ch/i)  +  (f*d)  -  (f+i);
```



Exercício

Desenvolva um programa que armazene o resultado da divisão dos números entre 1 e 10 (inteiro), cada valor deverá ser dividido por 3, 5 e 7. O armazenamento das divisão deve considerar o ponto flutuante.

No final o programa deve imprimir os valores inteiros e seus respectivos resultados de divisão.
Utilize uma matriz bidimensional, e a conversão cast na solução deste algoritmo.

Exercício – Matriz bidimensional

Desenvolva um programa que armazene a informação de 5 funcionários com os seguintes dados: código do funcionário, quantidade de dias da jornada de trabalho no mês e quantidade de faltas não justificadas.
A saída do programa deve apresentar o seguinte relatório:

" Funcionário XXXX dos YY dias trabalhou ZZ dias.". Sendo que, XXX corresponde ao código do funcionário, YY corresponde a quantidade de dias da jornada de trabalho no mês e ZZ a quantidade de dias que o funcionário efetivamente trabalho.

Exercício – Matriz bidimensional (1)

Desenvolva um programa que preencha automaticamente uma matriz bidimensional 4 X 4 com o dobro da soma de seus índices. Após preenchimento execute a soma de todos os elementos das linhas 1 e 4 da matriz, e apresente o resultado da soma por linha (1 e 4) para o usuário. Utilize laços de repetição no desenvolvimento deste algoritmo.

Exercício – Matriz bidimensional (2)

Desenvolva um programa que preencha automaticamente uma matriz bidimensional 4 X 4 com o triplo da multiplicação de seus índices. Após preenchimento execute a soma de todos os elementos das colunas 2 e 3 da Matriz, e apresente o resultado da soma (por coluna 2 e 3) para o usuário. Utilize laços de repetição no desenvolvimento deste algoritmo.

Exercício – Matriz bidimensional (3)

Desenvolva um programa que preencha automaticamente uma matriz bidimensional 4 X 4 com o (dobro da soma de seus índices) + (triplo da multiplicação de seus índices). Após preenchimento execute a soma de todos os elementos diagonal principal da matriz, e apresente o resultado da soma para o usuário. Utilize laços de repetição no desenvolvimento deste algoritmo.

Questionário de apoio aos estudos

1. Qual a quantidade total de memória utilizada por uma matriz multidimensional de inteiros valor[10][5][6][4]?
2. O que é uma matriz bidimensional?
3. O que é uma matriz multidimensional?
4. Descreva a formula para calcular o total de bytes a ser utilizado em um vetor bidimensional.
5. Descreva a formula para calcular o total de bytes a ser utilizado em um vetor multidimensional.