Programação II
Notas de Aula

Ricardo Dorneles

UCS

31 de julho de 2018

Índice

Matrizes

- Matrizes são conjuntos de variáveis organizados em uma estrutura de duas dimensões (i.e. linhas e colunas)
- Da mesma forma que vetores, para referenciar um elemento de uma matriz deve-se especificar o número da linha e da coluna que se quer referenciar
- Ao declarar uma matriz deve-se especificar o número de linhas e colunas da mesma, bem como o tipo de valores com que ela trabalha (float, int...):
  - Ex: int Mat[5][5];

	0	1	2	3	4
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	0	1

Matrizes

- Matrizes podem ter mais de duas dimensões, mas nesse semestre trabalharemos apenas com matrizes bidimensionais.
- Assim como nos vetores, o número da primeira linha é 0, e o número da primeira coluna de cada linha é 0.
- A forma de referenciar um elemento de uma matriz na linguagem C é nome_da_matriz[número da linha][número da coluna].
- Um elemento de uma matriz é uma variável, e pode ser utilizado em qualquer situação que uma variável simples, especificando o nome da matriz e a linha e coluna do elemento a ser referenciado.

Operações sobre matrizes

- Assim como vetores, operações sobre matrizes são feitas elemento a elemento
- Normalmente constituem de dois **for** encadeados, para gerar todas as combinações de linhas e colunas. Ex:

```
for (i=0;i<3;i++){
        for (j=0; j<3;j++){
                M[i][j]=0;
        }
}
```

i	j	
0	0	M[0][0]=0;
0	1	M[0][1]=0;
0	2	M[0][2]=0;
1	0	M[1][0]=0;
1	1	M[1][1]=0;
1	2	M[1][2]=0;
2	0	M[2][0]=0;
2	1	M[2][1]=0;
2	2	M[2][2]=0;

- A figura abaixo ilustra os índices (linha e coluna) dos elementos de uma matriz 5x5.
- M00000100 - Faça um programa leia uma matriz M[5][5]e calcule e escreva a soma dos elementos da diagonal principal (em negrito na figura abaixo).

	0	1	2	3	4
0	**0,0**	0,1	0,2	0,3	0,4
1	1,0	**1,1**	1,2	1,3	1,4
2	2,0	2,1	**2,2**	2,3	2,4
3	3,0	3,1	3,2	**3,3**	3,4
4	4,0	4,1	4,2	4,3	**4,4**

- A figura abaixo ilustra os índices (linha e coluna) dos elementos de uma matriz 5x5.
- M00000150 - Faça um programa que leia uma matriz M[5][5] e calcule e escreva a soma dos elementos da diagonal secundária (em negrito na figura abaixo).

	0	1	2	3	4
0	0,0	0,1	0,2	0,3	**0,4**
1	1,0	1,1	1,2	**1,3**	1,4
2	2,0	2,1	**2,2**	2,3	2,4
3	3,0	**3,1**	3,2	3,3	3,4
4	**4,0**	4,1	4,2	4,3	4,4

- Pode-se identificar a posição de um elemento em relação à diagonal principal (acima, abaixo ou exatamente na diagonal principal) pela relação entre seus índices (linha e coluna)
  - Elementos **exatamente** na diagonal principal: linha $=$ coluna
  - Elementos **acima** da diagonal principal: linha $<$ coluna
  - Elementos **abaixo** da diagonal principal: linha $>$ coluna

	0	1	2	3	4
0	**0,0**	0,1	0,2	0,3	0,4
1	1,0	**1,1**	1,2	1,3	1,4
2	2,0	2,1	**2,2**	2,3	2,4
3	3,0	3,1	3,2	**3,3**	3,4
4	4,0	4,1	4,2	4,3	**4,4**

- De forma semelhante pode-se identificar a posição de um elemento em relação à diagonal secundária (acima, abaixo ou exatamente na diagonal principal) pela soma de seus índices (linha e coluna). Em uma matriz NxN:
  - Elementos **exatamente** na D.S.: linha + coluna = N - 1
  - Elementos **acima** da D.S.: linha + coluna < N - 1
  - Elementos **abaixo** da D.S: linha + coluna > N - 1

	0	1	2	3	4
0	0,0	0,1	0,2	0,3	**0,4**
1	1,0	1,1	1,2	**1,3**	1,4
2	2,0	2,1	**2,2**	2,3	2,4
3	3,0	**3,1**	3,2	3,3	3,4
4	**4,0**	4,1	4,2	4,3	4,4

- Na declaração de uma matriz devem ser especificados o tipo de cada elemento, o número de linhas e o número de colunas.
- Ex:

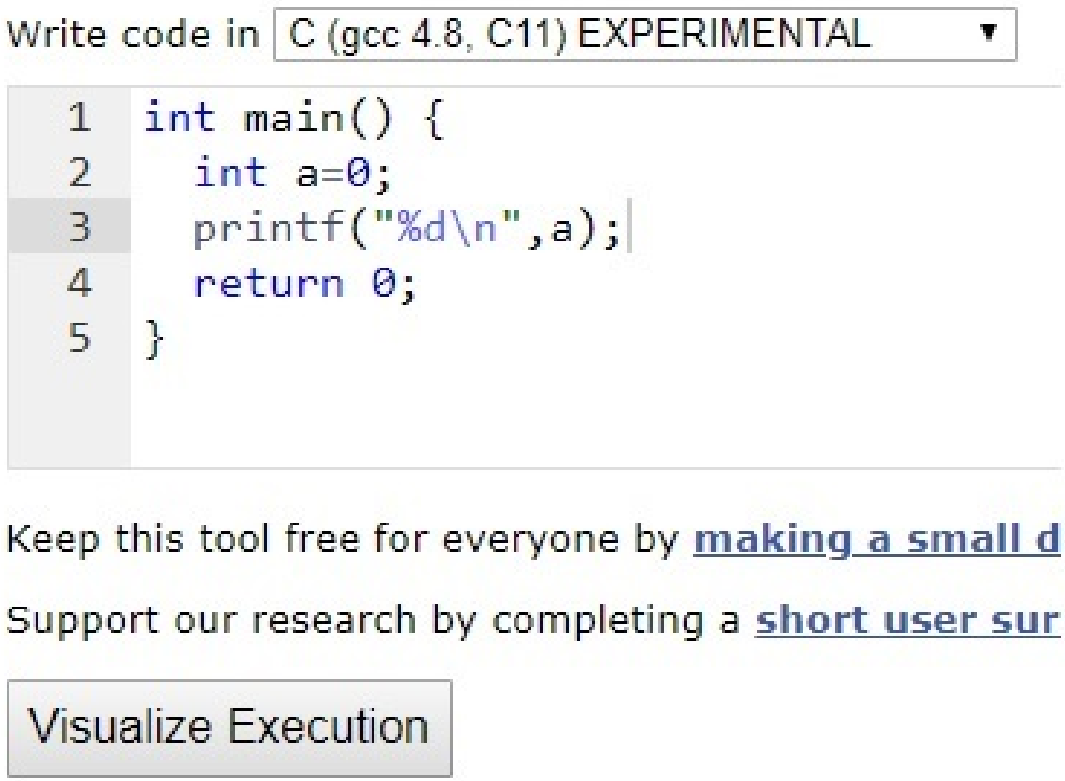
  int m[5][5]; /* matriz de 25 elementos do tipo int, de linhas 0 a 4, cada linha tendo as colunas 0 a 4 */
- Pode-se inicializar os elementos de uma matriz já na declaração:

  int m[3][3]={{1,2,3},{4,5,6},{7,8,9}};

  // declara uma matriz de 3 linhas (de 0 a 2) e 3 colunas (de 0 a 2) do tipo int, já inicializadas.

Ferramenta fortemente recomendada

- http://pythontutor.com/ → Start visualizing your code now
- Selecionar linguagem
- Editar código (p.ex. código abaixo)
- Visualize execution



1) Faça um programa que leia uma matriz m[3][3]. Calcule e escreva, após, a soma de todos os elementos da matriz.

- A figura abaixo ilustra os índices (linha e coluna) dos elementos de uma matriz 5x5.

	0	1	2	3	4
0	**0,0**	0,1	0,2	0,3	0,4
1	1,0	**1,1**	1,2	1,3	1,4
2	2,0	2,1	**2,2**	2,3	2,4
3	3,0	3,1	3,2	**3,3**	3,4
4	4,0	4,1	4,2	4,3	**4,4**

2) Faça um programa que leia uma matriz m[4][4] e calcule e escreva:

a) A soma dos elementos da diagonal principal.

b) A soma dos elementos da diagonal secundária.

c) A soma dos elementos ACIMA da diagonal principal.

d) A soma dos elementos ACIMA da diagonal secundária.

e) A soma dos elementos ACIMA da diagonal principal e ABAIXO da secundária

3) Faça um programa que leia uma matriz m[4][4] e:

a) Troque todos os elementos da linha 1 pelo elemento correspondente na linha 3.

b) Troque cada elemento da coluna 0 pelo elemento correspondente na coluna 2.

Escreva ao final a matriz alterada.

- 4) Faça um programa que leia uma matriz M[5][5], possivelmente com elementos repetidos. Leia, a seguir, 5 valores e, para cada um, verifique se o valor ocorre ou não na matriz, escrevendo a posição (linha e coluna) em que foi encontrada a primeira ocorrência do mesmo e, caso ele não exista na matriz, a mensagem "Não tem".
- 5) Faça um programa que leia uma matriz m[6][5]. Escreva, ao final, a soma dos elementos de cada linha e a soma dos elementos de cada coluna.
- 6) Faça um programa que leia uma matriz M[5][5] e escreva o maior valor existente na matriz, bem como a linha e coluna onde o valor ocorre.

Exercícios de matrizes em geral:

- 1) Escreva um programa que leia duas matrizes numéricas A[3][4] e B[3][4] e gere e escreva uma matriz inteira C[3][4], tal que um elemento C[i][j]= 1 se os elementos nas mesmas posições das matrizes A e B forem iguais, e 0 em caso contrário. O programa deve, ao final, escrever a matriz C.
- 2) Uma matriz identidade é uma matriz que possui 1 em todos os elementos da diagonal principal, e 0 em todas as outras posições. Faça um programa que leia uma matriz M[5][5] e verifique se é uma matriz identidade escrevendo uma mensagem apropriada.

- 3) Faça um programa que leia uma matriz M[5][5] e escreva o número da linha que contenha a maior soma de seus elementos. Considere que a matriz só contem valores positivos.
- 4) Faça um programa que leia uma matriz M[5][5] e escreva o número da linha que contenha a maior soma de seus elementos. Considere que a matriz pode conter valores positivos e negativos.

5) Faça um programa que leia uma matriz M[5][5] e gere dois vetores SomaLin[5] e SomaCol[5], com a soma dos elementos de cada linha e a soma dos elementos de cada coluna da matriz M. Escreva ao final os vetores Somalin e Somacol.

- 6) Uma matriz é dita Diagonalmente Dominante se atende às duas condições abaixo::

  a) em todas as linhas o elemento da diagonal principal é maior ou igual à soma dos outros elementos da linha e,
  b)há pelo menos uma linha em que o elemento da diagonal principal é MAIOR que a soma dos outros elementos da linha (não basta que seja igual).

  Faça um programa que leia uma matriz M[4][4] e verifique se é diagonalmente dominante escrevendo:

  1 - Se é diagonalmente dominante;
  0 - Se não é diagonalmente dominante

- 7) Faça um programa que leia uma matriz M[5][5], onde cada posição contem um número entre 0 e 9 e cada linha da matriz representa um número de 5 dígitos. O programa deve calcular a soma dos 5 números contidos na matriz colocando o resultado em um vetor Soma[6]. Escreva ao final o vetor Soma.
- 8) Faça um programa que leia uma matriz M[5][5], onde cada posição contem um número entre 0 e 9 e cada linha da matriz representa um número de 5 dígitos. O programa deve encontrar a linha que contem o maior dos 5 números e escrever o número.

- 9) Faça um programa que leia uma matriz M[5][5], onde cada posição contem um número entre 0 e 9 e cada linha da matriz representa um número de 5 dígitos. O programa deve ordenar os 5 números em ordem crescente, e escrever a matriz com os números ordenados.
- 10) Faça um programa que leia dois valores D e DS, correspondendo a um dia do mês e um dia da semana (1-domingo, 2-segunda,... 7-sábado) e preencha uma matriz Folhinha[6][7] com o calendário correspondente ao mês do dia digitado. A primeira coluna da matriz contem os domingos, a segunda coluna contem as segundas e assim por diante. O programa deve escrever a matriz gerada. As posições não utilizadas da matriz devem conter 0's.

- 11) Faça um programa que leia uma matriz M[5][5] e um valor N e multiplica cada valor da matriz M por N e coloca o resultado em um vetor V[25]. Escreva ao final o vetor V.
- 12) Faça um programa que leia uma matriz M[5][6] e divide todos os 6 elementos de cada linha pelo valor do menor elemento EM MÓDULO da linha. Escrever a matriz modificada.
- 13) Faça um programa que leia uma matriz M[5][5], possivelmente com elementos repetidos. Leia, a seguir, 5 valores e, para cada um, verifique se o valor ocorre ou não na matriz, escrevendo a posição (linha e coluna) em que foi encontrada a primeira ocorrência do mesmo e, caso ele não exista na matriz, a mensagem "Não tem".

- 14) Uma matriz é dita Matriz de Toeplitz se, em cada diagonal paralela à diagonal principal, todos os elementos são iguais. Assim, um exemplo de Matriz de Toeplitz é:

1	3	5	4	6
2	1	3	5	4
8	2	1	3	5
7	8	2	1	3
9	7	8	2	1

Faça um programa que leia uma matriz M[1..5,1..5] e verifique se é uma Matriz de Toeplitz, escrevendo:

1 - Se é uma matriz de Toeplitz;
0 - Se não é uma matriz de Toeplitz

- 15) - Uma matriz é dita Circulante cada elemento é igual ao elemento imediatamente acima à esquerda, e se o primeiro elemento de cada coluna é igual ao último elemento da coluna anterior. Assim, um exemplo de Matriz Circulante é:

1	9	7	8	2
2	1	9	7	8
8	2	1	9	7
7	8	2	1	9
9	7	8	2	1

Faça um programa que leia uma matriz M[1..5,1..5] e verifique se é uma Matriz Circulante, escrevendo:

1 - Se é uma matriz Circulante;

0 - Se não é uma matriz Circulante

- 16) Faça um programa que leia uma matriz M[1..3,1..3] e para cada linha divida toda a linha pelo seu elemento da primeira coluna. Após isso, a partir da segunda linha, de cada linha subtraia toda a a primeira linha elemento a elemento.
  ex. a matriz:

2	4	6
3	9	6
2	8	4

1	2	3
1	3	2
1	4	2

1	2	3
0	1	-1
0	2	-1

  Escreva a matriz alterada.

- 17) Um quadrado mágico de ordem N (sendo N um número ímpar) é um arranjo de número de 1 a NxN em uma matriz quadrada de tal modo que a soma de cada linha, coluna e diagonal é a mesma. A matriz abaixo representa um quadrado mágico de ordem 5.

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

- A regra para gerá-lo é relativamente fácil de observar:
  - Comece com 1 no meio da primeira linha.
  - Siga para cima e para a esquerda diagonalmente (quando sair do quadrado suponha que os lados superior e inferior estão unidos ou que os lados da direita e da esquerda estão unidos, conforme for o caso).
  - Em cada quadrado que passar coloque o valor do quadrado anterior mais 1 (um).
  - Quando a próxima casa estiver já preenchida, desça um quadrado e continue seguindo a diagonal até ter preenchido todos os quadrados.
- Faça um programa que lê um número N ímpar, menor ou igual a 99, e gere e escreva, para o número lido, o seu quadrado mágico.

18) Faça um programa que leia duas matrizes M[3][3] e N[3][3] (primeiro a matriz M, e após a matriz N) e calcule o produto matricial MxN colocando o resultado em uma matriz P[3][3]. Escreve ao final a matriz P.

20) As pirâmides têm a base quadrada, sendo que a única forma de se atingir o topo é seguir em espiral pela borda. Escreva um programa que leia um valor N e, a seguir, uma matriz quadrada A[n][n] de ordem no máximo igual a 10, e verifique se a matriz é equivalente a uma pirâmide inca, ou seja, se partindo-se do canto superior esquerdo da matriz, no sentido horário, em espiral, a posição seguinte na ordem é o inteiro consecutivo da posição anterior. O programa deve escrever:

1 - Se a matriz forma uma pirâmide inca;
0 - Em caso contrário.

21) Faça um programa que leia uma matriz M[3][20], onde cada posição contem um número entre 0 e 9 e cada linha da matriz representa um número de 20 dígitos. O programa deve calcular a soma dos 3 números contidos na matriz colocando o resultado em um vetor Soma[21]. Escreva ao final o vetor Soma. Utilize o programa para calcular a soma dos números: 12345678901234567890, 93579135799357913579 e 99999999999999999999

22) Faça um programa que leia uma matriz M[5][5], onde cada posição contem um número entre 0 e 9 e cada linha da matriz representa um número de 5 dígitos. O programa deve encontrar a linha que contem o maior dos 5 números e escrever o número.
23) Faça um programa que leia uma matriz M[5][5], onde cada posição contem um número entre 0 e 9 e cada linha da matriz representa um número de 5 dígitos. O programa deve ordenar os 5 números em ordem crescente, e escrever a matriz com os números ordenados.

24) Faça um programa que leia uma matriz M[5][5] e um valor N e multiplica cada valor da matriz M por N e coloca o resultado em um vetor V[25]. Escreva ao final o vetor V.
25) Faça um programa que leia uma matriz M[5][6] e divide todos os 6 elementos de cada linha pelo valor do menor elemento EM MÓDULO da linha. Escrever a matriz modificada.

- 26) Uma matriz é dita triangular superior se todos os elementos abaixo da diagonal principal são iguais a zero, e há pelo menos um elemento nulo acima da diagonal principal.

  Da mesma forma, uma matriz é dita triangular inferior se todos os elementos acima da diagonal principal são iguais a zero, e há pelo menos um elemento não nulo abaixo da diagonal principal.

  E uma matriz é dita diagonal se os elementos não nulos ocorrem somente na diagonal principal.

  Faça um programa que leia uma matriz M[1..5,1..5] e escreva:

  0 - Se a matriz é uma matriz diagonal;
  1 - Se é triangular superior;
  2 - Se é triangular inferior;
  3 - Se não é nenhuma das anteriores

- 27) Escrever um programa que lê uma matriz M[1..3,1..3], contendo uma posição de jogo da velha, com valores 0 (casa livre), 1 (marcada com cruzinha) ou 5 (marcada com bolinha) e escreva:

  1 - se o jogador 1 venceu o jogo (alguma linha, coluna ou diagonal com o mesmo valor);
  2 - se o jogador 2 venceu o jogo;
  3 - se o jogo terminou empatado (não há mais lances e ninguém ganhou);
  4 - se o jogo ainda não terminou (há lances por jogar e ninguém ainda venceu)

28) Na Teoria de Sistemas, define-se como elemento minimax de uma matriz o menor elemento da linha em que se encontra o maior elemento da matriz. Escreva um programa que leia uma matriz A[5][5] e determine o seu elemento minimax. O programa deve, ao final, escrever o valor do elemento minimax, bem como a linha e coluna onde ocorreu.

Variáveis do tipo caracter (char)

- São variáveis que podem conter um caracter, podendo ser um uma letra, dígito ou caracter especial. Os caracteres são armazenados utilizando uma representação chamada ASCII (American Standard Code for Information Interchange), que representa cada caracter por um valor numérico entre 0 e 255.
- Caracteres de tabulação
  9 - Horizontal Tab - HT
  10 - Line Feed - LF
  13 - Carriage Return (Retorno de Carro) - CR

Índice

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

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

- Caracteres especiais

32 - Space - SPC		
33 - !	34 - "	35 - #
36 - $	37 - %	38 - &
39 - '	40 - (	41 -)
42 - *	43 - +	44 - ,
45 - -	46 - .	47 - /

- 48 a 57 - Dígitos de 0 a 9
- 65 a 90 - Letras maiúsculas de A a Z
- 97 a 122 - Letras minúsculas de a a z

- Variáveis do tipo caracter são declaradas com o tipo char.
  - Ex: char a,b,c;
- Constantes do tipo char são representadas pelo caracter entre aspas simples. Ex: let='T'; (a variável let recebe o caracter 'T')
- Variáveis do tipo char podem ser usadas como inteiros de 8 bits ou como caracteres. As duas atribuições a seguir são totalmente equivalentes, uma vez que o caracter '0' na tabela ASCII está na posição 48: char c='0',b=48;
- O especificador de formato para variáveis char é %c.

```
for (c='0';c<='9';c++)
        printf("%c",c)
```

- Para converter um caracter de dígito em seu valor numérico correspondente basta subtrair do caracter o valor 48, que corresponde ao dígito '0'.

  char c='5'; int valor=c-'0'; (ou valor=c-48;)

- O programa a seguir escreve a posição na tabela ASCII e o caracter das posições 32 a 126.

```
#include ¡stdio.h¿
#include ¡stdlib.h¿
int main() {
    int i;
    for ( i = 32; i < 127; i++ )
        printf( "%c[%d]\n", i , i );
    return 0;
}
```

Função getch()

- Variáveis char podem ser lidas com a função getch( ), da biblioteca conio.h.
- Ex:

  c=getch( ); /* fica esperando o usuário digitar um caracter e quando for digitado o caracter é colocado na variável c. */
- Faça um programa que fique lendo teclas até que seja digitada a tecla 'f' e, quando forem tecladas as teclas das setas, escreva uma mensagem correspondente à tecla teclada. Ex: "Tecla para cima", "Tecla para a esquerda".
- Teclas diferentes das setas e de 'f' devem ser ignoradas.

```
#include <stdio.h>
#include <conio.h>
int main()
      {
      unsigned char c,d;
      c=getch();
      d=getch();
      printf("c=%d d=%d\n",c,d);
      }
```

- Também podem ser lidas com o scanf, especificador %c, mas ao digitar um caracter e o Enter, o Enter ficará pendente no buffer de teclado, e a próxima leitura de char lerá o enter, fazendo com que leituras sejam perdidas.

```
#include ¡stdio.h¿
int main()
{
char c,d;
scanf("%c",&c);
scanf("%c",&d);
printf("c=%c d=%c\n",c,d);
}
```

Strings

- São vetores de caracteres utilizado para manipulação de textos em C.
- Declaração: char nome[10]; // vetor de 10 posições do tipo caracter.
  - Podem ser inicializados na declaração: char nome[10]="Maria";
- Obs: Uma das posições do vetor é utilizada para armazenar o delimitador de fim de string (valor inteiro 0, representado por '\0').
  - O delimitador de fim de string é colocado automaticamente pelas funções de manipulação de string, ou ao inicializar um string na declaração.
- Assim, para armazenar N caracteres é necessário declarar um vetor com N+1 posições.

Índice

- Leitura de um string:

  scanf("%s",nome);
  - Obs: Como um string é um vetor, não é necessário o & para a leitura do mesmo.
  - Da mesma forma como ocorre ao ler inteiros, o scanf usa espaço em branco como separador de strings. Assim, ao tentar ler o nome "Maria da Silva", um scanf("%s",nome) lerá apenas "Maria".
- Uma alternativa para ler todos os caracteres de um string até o Enter (incluindo espaços em branco e quaisquer caracteres especiais) é o comando fgets.
  - fgets(variável string, número máximo de caracteres, stdin);
  - Ex: fgets(linha,200,stdin); // onde linha é um string de 200 caracteres e stdin é a especificação de entrada via teclado

```
#include <stdio.h>
#include <string.h>
int main()
    {
    char linha[200];
    fgets(linha,200,stdin);
    printf("linha=%s",linha);
    int i;
    for (i=0;i<strlen(linha);i++)
        printf("linha[%d]=%d %c\n",i,linha[i],linha[i]);
    }
```

- Escrita de um string:

  printf("%s",nome);

1) Faça um programa que leia um string e o escreva.

```
#include ¡stdio.h¿
int main()
{
char nome[10];
scanf("%s",nome);
printf("%s",nome);
}
```

Obs: O scanf utiliza o espaço em branco como delimitador de fim de string na leitura. Pode-se ler strings com o comando gets(string), que lê todos os caracteres até ser digitado ENTER. O gets consome o Enter, mas ele não fará parte do string lido.

Funçoes para manipulação de strings (colocar no início #include < *string.h* >)

- **strcmp**(str1,str2) - compara dois strings em ordem lexicográfica e retorna:
  - 0 - se os strings são iguais
  - <0 - se str1 é menor que str2 (considerando ordem lexicográfica)
  - >0 - se str1 é maior que str2 (considerando ordem lexicográfica)

ex:

```
if (strcmp(nome1,nome2)<0)
    printf("%s",nome1)
else printf("%s",nome2);
```

- A implementação de várias funções de manipulação de strings consiste de um único for. Uma implementação para a função strcmp seria:

```
int strcmp(char st1[], char st2[]){
        for (i=0;st1[i]==st2[i] && st1[i]!='\0';i++);
        return st1[i]-st2[i];}
```

- 2) Faça um programa que leia dois nomes e escreva-os em ordem alfabética.
- Pode-se declarar uma lista de strings como uma matriz de caracteres:
  - char nomes[20][15]; // declara uma lista de 20 strings, cada um com até 14 caracteres (e mais um espaço para o delimitador de fim de string)
- Nesse caso, se ao referenciar nomes forem usados dois índices (ex: nomes[i][j]) está-se se referindo ao char posição [i][j]. Se for usado apenas um índice (nomes[i]) é uma referência a todo o string da linha i.

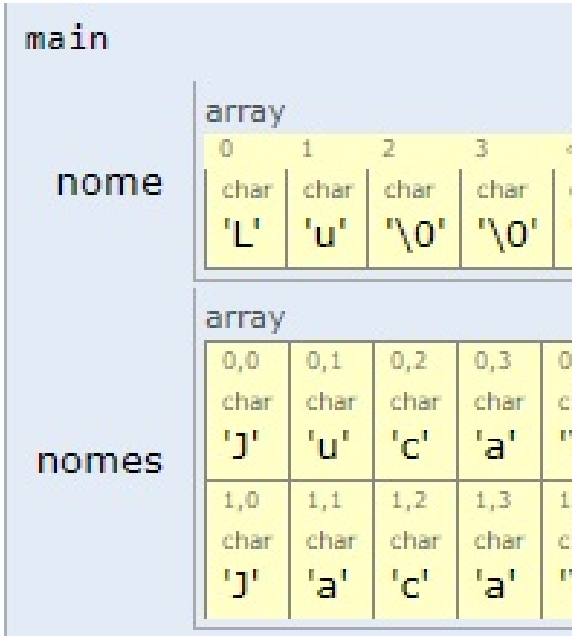
- Pode-se inicializar um string na declaração:

```
char nome[20]="João";
```

- também pode-se inicializar uma lista de strings (lâmina seguinte):

```
char nomes[5][15]={"João","Maria","Zefa","Juca","Jeca"};
char unidades[10][10]={"zero","um","dois","tres","quatro",
                       "cinco","seis","sete","oito","nove"};
```

```
1
2  int main() {
3    char nome[5]="Lu";
4    char nomes[2][5]={"Juca","Jaca"};
5    return 0;
6  }
```



- **strcpy**(destino, origem) - copia o string de origem para o string de destino.

  Ex:

  ```
  char nome[20];
  strcpy(nome,"João"); // copia "João" para o string nome
  ```

- Seguindo o modelo da função strcmp, faça uma função equivalente à função strcpy.

- **strlen**(str) - retorna o tamanho do string em str.
  Ex:

  a=strlen("João"); // a variável "a" recebe o valor 4
- Seguindo o modelo da função strcmp, faça uma função equivalente à função strlen.

- **strcat**(str1,str2) - concatena uma cópia de str2 no final de str1.

  Ex:

  ```
  strcpy(st1,"João");
  strcat(st1," da Silva");//st1 ficará com "João da Silva"
  ```

- Seguindo o modelo da função strcmp, faça uma função equivalente à função strcat.

- char * **itoa** (int value, char * str, int base); - Converte um inteiro para um string (stdlib.h, mas não é padrão de C)
- int **atoi**(const char *str) - converte um string para um inteiro

- **sscanf** - le um valor a partir de um string, no mesmo formato que o scanf. Ex:
  - char linha[ ]="123 23.2 Joao";
  - sscanf(linha,"%d%f%s",&x,&y,z);

- **sprintf** - um printf, só que a saída é colocada em um string
  - sprintf(linha,"x=%d y=%f z=%s\n",x,y,z);

- char * **strstr**(char *st1, char *st2) - verifica se o segundo string ocorre no primeiro. O código na lâmina seguinte exemplifica seu uso.

- long int **strtol**(const char *str, char **endptr, int base) - converte a parte inicial de um string em um valor long, e atualiza o pointer endptr para apontar para o próximo caracter após a parte convertida (está na stdlib).
  - char linha[]="12 15 18",*pl=linha;
  - printf("%d\n",strtol(pl,&pl,10));
  - printf("%d\n",strtol(pl,&pl,10));
  - printf("%d\n",strtol(pl,&pl,10));

```
#include <stdio.h>
#include <string.h>
int main() {
      char st1[ ]="Watson-Watson-Watson Watson";
      char *p=st1;
      int cont=0;
      while ((p=strstr(p,"Watson"))!=NULL)
            {
            cont++;
            p=p+1;
            }
      printf("%d",cont);
}
```

- char *strtok (char *st,char delimitadores) - retorna o substring que ocorre até o primeiro delimitador da lista de delimitadores. Ex: strtok("123+14","+-*") retorna o substring "123", que é o substring até o primeiro delimitador da lista "+-*". No caso, o substring até a primeira ocorrência de '+'.
- As chamadas subsequentes de strtok devem passar NULL como primeiro parâmetro. Quando não houver mais substrings que atendam à condição, a função retorna NULL;

- A cada chamada a função coloca '\0' no delimitador atualizando um pointer estático para a posição seguinte, preparando para a próxima chamada.
- Ex:

```
char linha[]="Joao da Silva,Maria da Silva,Jose da Silva";
char *p=strtok(linha,",");
while (p)
    {
    printf(" %s\n",p);
    p=strtok(NULL,",");
    }
```

- Se ocorrerem duas ou mais ocorrências contíguas de delimitadores, a função strtok as trata como um único delimitador. A função strtoke, abaixo, retorna uma string vazia quando houver dois delimitadores contíguos:

```
char* strtoke(char *str, const char *delim)
{
static char *start = NULL;
char *token = NULL;
if (str) start = str;
if (!start) return NULL;
token = start;
start = strpbrk(start, delim);
if (start) *start++ = '\0';
return token;
}
```

widthheight itemize

char *strpbrk (char *st,char delimitadores) - retorna um pointer para a posiçãoo da primeira ocorrência de um delimitador da lista.

As chamadas subsequentes de strpbrk devem passar NULL como

- 1.Fazer um programa que leia uma palavra e diga se ela é ou não um palíndromo. Ex: ARARA, ovo.
- 2.Idem ao anterior, mas sem distinção de maiúsculas e minúsculas. Ex: Arara, Ovo, OsSo.
  - Para converter um char de minúscula para maiúscula, utilize a função

    char toupper(char letra)
  - que recebe um caracter e retorna o caracter maiúsculo equivalente
- 3.Fazer um programa que leia uma letra (L) e um número (N), a seguir gere uma string contendo N letras L.
- 4.Fazer um programa que leia um número inteiro positivo, converta-o em uma string e escreva-a.
- 5.Fazer um programa que leia um número inteiro positivo, converta-o em hexadecimal, armazenando o resultado em uma string e escreva-a.

- 6.Fazer um programa que leia uma string até que seja um número hexadecimal válido, após converta esse número lido para decimal, armazenando-o em uma variável inteira e escreva-a.
- 7.Fazer um programa que leia uma string e a partir desta gere uma nova duplicando cada caracter da string original. Escreva a nova string. Ex: "OI" => "OOII"; "PROVA 1" => "PPRROOVVAA 11"
- 8.Fazer um programa que leia uma string e a partir desta gere uma nova contendo um espaço em branco entre cada caracter da string original. Escreva a nova string. Ex: "LIVRO DE C" => "L I V R O D E C"

- 9. Faça um programa em C que inverta os caracteres de uma string. Por exemplo, se a string for "UNIVASF", deve ser convertida a "FSAVINU".
- 10. Faça um programa em C que leia uma string s, um caracter chAtual, um caracter chNovo e substitua todo caracter chAtual de s pelo chNovo. O programa deve retornar também o número de substituições.
- 11. Faça um programa em C que leia uma string s, um caracter ch, um inteiro pos e insira o caracter ch na posição pos da string s.
- 12. Faça um programa em C que leia uma string s1, uma string s2, um inteiro pos e insira a string s2 em s1 na posição pos.

- 13. Faça um programa em C que leia uma string s1, uma string s2, um inteiro n e copie os n primeiros caracteres da string s1 na string s2.
- 14. Faça um programa em C que leia uma string s1, uma string s2, um inteiro n e copie os n últimos caracteres da string s1 na string s2.
- 15. Faça um programa em C que leia uma string s1, uma string s2, um inteiro n, um inteiro inicio e copie os n caracteres a partir da posição inicio da string s1 na string s2.

- 16.Fazer um algoritmo que leia o nome completo de uma pessoa, separando-o em partes. Considerar como delimitador(es) espaço(s) em branco. Escreva cada parte do nome em uma nova linha. Exemplo:

Lê	Escreve
Paulo Henrique Silveira	Paulo Henrique Silveira
Pedro Silva	Pedro Silva

- 17.Fazer um algoritmo que leia o nome de uma pessoa e após gere o seu email (utilizando letras minúsculas) considerando a primeira letra dos primeiros nomes completando com as letras do último nome até, no máximo, oito caracteres. Exemplos:

Nome da Pessoa	Email
Paulo Henrique Silveira	phsilvei
Pedro Silva	psilva

- 18. Idem ao exercício anterior, mas armazene os emails gerados em um vetor evitando geração de emails duplos, trocando a(s) última(s) letras do mesmo por números sequenciais a partir de um. Exemplos:

Nome da Pessoa	Email
Paula Silva	psilva
Paulo Renato Henrique Miguel Silveira	prhmsilv
Plínio Roberto Humberto Moraes Silva	prhmsil1
Pedro Silva	psilva1
Patricia Silva	psilva2

- 19. Fazer um algoritmo que leia nomes e salários de dez funcionários e após escreva somente os nomes em ordem decrescente de salário e finalmente escreva os nomes em ordem alfabética e os respectivos salários.
- 20. Fazer um algoritmo que leia uma string contendo uma data no formato DD/MM/AAAA. Separe o dia, o mês e o ano, armazenando-os em três variáveis inteiras. Caso o formato não seja esse, armazene -1 nessas variáveis e finalmente escreva essas três variáveis.

- 21. Faça um programa que leia um número inteiro entre 0 e 99 e gere um string com a representação do número por extenso, e escreva o string ao final.

Tabela de precedência de operadores em C

Prec.	Símbolo	Função	Associatividade
1	++ - -	Incremento e decremento pósfixo	esquerda para direita
1	()	Parênteses (chamada de função)	
1	[ ]	Elemento de array	
1	.	Seleção de campo de structure	
1	->	Seleção de campo de structure a partir de ponteiro	
2	++ - -	Incremento e decremento prefixo	direita para a esquerda
2	+ -	Adição e subtração unária	
2	! ~	Não lógico e complemento	
2	(tipo)	Conversão de tipo de dado	
2	*	Desreferência	
2	&	Referência (endereço de elemento)	
3	* / %	Multiplicação, divisão, e módulo (resto)	esquerda para a direita
4	+ -	Adição e subtração	
5	<< >>	Deslocamento de bits à esquerda e à direita	
6	< <=	"menor que" e "menor ou igual que"	
6	> >=	"maior que" e "maior ou igual que"	
7	== !=	"Igual a" e "diferente de"	
8	&	E bit a bit	
9	^	Ou exclusivo para bits	
10	|	Ou para bits	
11	&&	E lógico	
12	||	Ou lógico	
13	c ? t : f	Condição ternária	direita para esquerda
14	=	Atribuição	
14	+= -=	Atribuição por adição ou subtração	
14	*= /= %=	Atribuição por multiplicação, divisão ou módulo (resto)	
14	<<= >>=	Atribuição por deslocamento de bits	
14	&= ^= |=	Atribuição por operações lógicas	
15	,	vírgula	esquerda para direita

Funções

- Funções são trechos de código que podem ser chamados a partir de diversos pontos diferentes do programa.
- Ao invés de escrever um trecho de código diversas vezes, escreve-se o código apenas uma vez e ele é chamado diversas vezes.
- A linguagem C possui diversas funções prontas. Essas funções encontram-se em bibliotecas, declaradas nas cláusulas #include.
- Assim #include $<$ *math.h* $>$ informa ao compilador que o programa utilizará funções matemáticas da biblioteca math.h.

Índice

- Além das funções pré-definidas na linguagem, o programador pode especificar suas próprias funções. Algumas vantagens do uso de funções são:
  - Reduzem o tamanho do programa, eliminando a repetição de código;
  - Melhoram a legibilidade do programa, "quebrando" um programa grande em partes menores, individualmente mais compreensíveis.
- Ao especificar uma função, o programador deve especificar que valores que ela irá receber, o tipo de valores que irá receber, e o tipo de valor que será retornado por ela (se for o caso). Os valores passados para uma função, que ela utilizará para efetuar os cálculos necessários, se chamam "parâmetros" ou "argumentos" da função.
- Os parâmetros enviados à função devem ser em mesma quantidade, tipo e ordem dos parâmetros definidos no cabeçalho da função.

- O formato geral de especificação de uma função é assim:

```
<tipo de valor de retorno> <nome da função> (<lista de parâmetros e tipo de cada um>)
    {
    <comandos>
    }
```

- Por exemplo, uma função que receba um valor inteiro e retorne o fatorial desse número pode ser escrita assim:

```
int fat (int n)
    {
    int i,f=1;
    for (i=1;i<=n;i++)
        f=f*i; // cálculo do fatorial
    return f; // especifica o valor que será retornado
    }
```

- E na chamada da função deve ser passado o parâmetro esperado.
- No caso da função fat, deve ser passado o valor de quem será calculado o fatorial.
- Uma chamada de função que retorne um valor deve ser feita sempre dentro de um comando onde o valor de retorno será imediatamente utilizado.
- Exemplos da chamada da função dentro do programa seriam algo como:

```
a=fat(4);

a=fat(b);

printf("O fatorial de %d = %d\n",b,fat(b));
```

- Funções podem chamar outras funções. Nesse caso, a função chamada deve estar declarada antes do ponto onde ela é chamada.
- Caso isso não seja possível, pode-se usar um **protótipo** que é simplesmente a linha do cabeçalho, sem o corpo da função, que deve estar desenvolvido mais adiante.

- Exemplo:

```
int poi()
        {
        printf("Oi");
        ptchau();
        }
int ptchau()
        {
        printf("Tchau");
        poi();
        }
```

- Como a função poi chama a ptchau, e a ptchau chama a poi, não é possível ordená-las de modo que a chamada da função fique após a declaração da mesma. A solução para isso é o uso de um

- Exemplo:

```
int ptchau(); // protótipo da ptchau
int poi()
        {
        printf("Oi");
        ptchau();
        }
int ptchau()
        {
        printf("Tchau");
        poi();
        }
```

- Uma função pode utilizar variáveis declaradas dentro dela, chamadas **variáveis locais**, mas pode também utilizar variáveis declaradas fora de qualquer função. Essas variáveis são chamadas de **variáveis globais** e podem ser utilizadas para passar informações entre funções.
- Variáveis locais só podem ser referenciadas dentro da função onde são declaradas.
- Ex:

```
#include <stdio.h>

int i,b; // variáveis globais

int fat(int n)

{

int f=1; // f é local a fat

for (i=1;i<=n;i++) f=f*i;

// como i não foi declarado em fat,

// é utilizado o i global

return f;

}

int main()

{

int x; // x é local a main

scanf("%d",&a); // a é global

printf("O fatorial de %d = %d\n",a,fat(a));

}
```

- O comando **return** é utilizado para retornar a execução para o ponto onde a função foi chamada.
- Ele pode ser executado em qualquer ponto da função e causa o encerramento imediato da execução da função.
- Se a função deve retornar algum valor, o valor a ser retornado deve ser especificado no comando return.
- Em algumas situações uma função não deve retornar nenhum valor.
- Nesse caso ela deve ser especificada como função do tipo **void**.

- Por exemplo, a uma função que receba um valor N e escreva os números de 1 a N:

```
void esc1aN (int N)
{
    int i;
    for (i=1;i<=N;i++)
        printf("%d ",i);
    return;
}
```

- E a chamada de uma função sem valor de retorno é feita como se fosse um comando da linguagem. Por exemplo, a função anterior seria chamada

  esc1aN(5);

  for (j=1;j<5;j++) esc1aN(j);

Exercícios:

- 1) Faça uma função que receba o dia, mês e ano de nascimento de uma pessoa, e o dia, mês e ano atual, e retorne sua idade.
- 2) Faça uma função que receba um valor N e retorne o primeiro divisor maior que 1 do valor N.
- 3) Faça uma função que receba um valor N e retorne a quantidade de divisores do valor N.
- 4) Usando uma das duas funções anteriores, faça uma função que recebe um valor N e verifique se ele é primo, retornando 1 se ele é primo, e 0 se não for primo. Use essa função para escrever um programa que escreva os 100 primeiros números primos.

- 5) Usando a função anterior, faça uma função proxprimo(int N) que receba um valor N e retorne primeiro valor primo maior que N. Use essa função para escrever os primeiros 100 números primos.
- 6) Faça um programa que escreva os 50 primeiros números primos.
- 7) Faça um programa que leia 2 números N1 e N2 e escreva a soma dos números primos entre N1 e N2 (incluindo N1 e N2 se algum deles for primo)..
- 8) Faça um programa que leia 2 números N1 e N2 e escreva o produto dos números primos entre N1 e N2 (incluindo N1 e N2 se algum deles for primo).

- 9) Faça um programa que leia um número N e escreva os N primeiros números primos maiores que 100.
- 10) Faça um programa que leia um número inteiro N e escreva o maior número primo menor do que N.
- 11) Faça um programa que leia um número inteiro N e escreva o menor número primo maior do que N.
- 12) Um número primo é um número natural maior que 1, que é divisível somente por 1 e por ele mesmo. Faça um programa que leia um número inteiro N e escreva o número primo mais próximo a ele. Se N for primo, considere que o mais próximo é o próprio N. Se houver dois números à mesma distância, escreva os dois em ordem crescente.

- 13) A conjectura de Goldbach diz que todo número par maior que 2 pode ser representado como a soma de dois números primos. Assim, 4=2+2, 6=3+3, 8=3+5... Faça um programa que leia um número N, par, e escreva, em ordem crescente, os dois números primos que o compõem. No caso de haver mais de um par de números (p.ex:20=3+17 e 20=7+13) escreva o par que tiver o menor número primo.

- 14) Faça uma funçao que receba o valor de dois resistores e um terceiro parâmetro definindo o tipo de associação dos resistores (0 para em série, 1 para em paralelo) e retorne o valor da associação dos dois resistores.

- 15) Faça uma função que receba um vetor v[10] e retorne o valor do maior elemento do vetor.

- 16) Faça uma função que receba um vetor v[10] e escreva todos os elementos do vetor.

- 17) Faça uma função que receba um vetor v[10] e ordene-o em ordem crescente.
- 18) Faça uma função que receba dois vetores v[10] e w[10] e verifique se os vetores são iguais, retornando 1 se são iguais, 0 se tem pelo menos um elemento diferente
- 19) Faça uma função que receba uma matriz M[10][10] e retorne o valor do maior elemento da matriz.
- 20) Faça uma função que receba uma matriz M[10][10] e retorne a soma dos elementos da diagonal principal.
- 21) Faça uma função que receba uma matriz M[10][10] e retorne a soma dos elementos da diagonal secundária.

- 22) Faça um programa que escreva os primeiros 100 dígitos cuja soma dos dígitos formantes é 10.
- 23) O número de combinações de N diferentes objetos em grupos de P objetos é dado por $\frac{N!}{P!(N-P)!}$. Faça uma função que receba uma quantidade N de objetos e o tamanho P dos grupos a serem formados, e calcule e retorne a quantidade de grupos que podem ser formados.
- 24) O MDC (máximo divisor comum) entre dois números n1 e n2 é o maior número que é divisor de n1 e de n2. Faça uma função que receba dois números e escreva seu MDC.

Ponteiros

- Um ponteiro é uma variável que contém um endereço de memória. Esse endereço é normalmente uma posição de outra variável na memória ou o endereço de uma estrutura em memória alocada dinâmicamente (durante a execução).
- Usam-se ponteiros principalmente para:
  - Fazer passagem de parâmetros por referência em funções;
  - Acessar estruturas alocadas dinamicamente;

Índice

- Uma declaração de ponteiros consiste no formato:

  $< tipo >$ *<nome da variável>

- onde tipo é qualquer tipo válido em C.
- Exs:

  int *pta; // pointer para valor inteiro

  float *ptf; // pointer para float

- O tipo base do ponteiro define que tipo de variáveis o ponteiro pode apontar.
- Basicamente, qualquer tipo ponteiro pode apontar para qualquer lugar, na memória.
- Mas, para a aritmética de ponteiros é feita através do tipo base.
- Os operadores utilizados para trabalhar com ponteiros são * e &.
- O *, quando colocado antes do ponteiro em uma referência, referencia a posição de memória apontada por ele.
- Ex:
- int *a;
- *a=3; // a posição de memória apontada por a recebe 3

- E o operador &, aplicado a uma variável, retorna o endereço dela.
- Ex:

```
int a, *pta;

pta=&a; // pta recebe o endereço da variável a.

*pta=5; // a variável apontada por pta (ou seja, a variável a,
recebe 5)
```

- 1) Seja o seguinte trecho de programa:

```
int i=3,j=5;
int *p, *q;
p = &i;
q = &j;
```

- Qual é o valor das seguintes expressões ?
- a) p = = &i;
- b) *p - *q
- c) **&p
- d) 3* - *p/(*q)+7
- resposta: 1, -2, 3, 6

Tipos de passagem de parâmetros

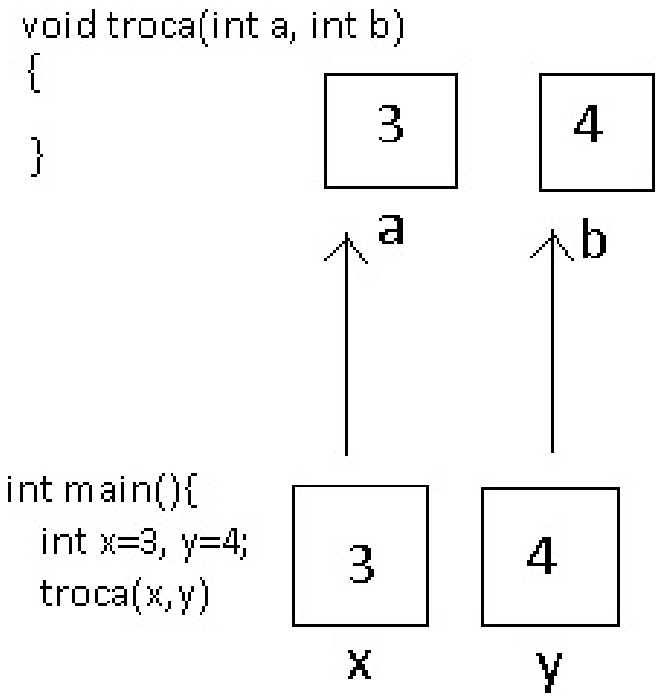
- Existem diversas formas de passagem de parâmetros em funções, e cada linguagem define que formas de passagem a linguagem oferece.
- O que será escrito (valores de **a** e **b**) no código a seguir?

```
void troca(int x, int y)
{
int aux;
aux=x;
x=y;
y=aux;
}
```

```
int main()
{
int a,b;
a=3;
b=5;
troca(a,b);
printf("%d %d\"n,a,b);
}
```

Passagem por valor

- No código anterior, apesar de **x** e **y** terem sido trocados entre si dentro da função, os valores de **a** e **b** ao retornar do procedimento permaneceram inalterados.
- Isso ocorre porque, na linguagem C, quando é feita uma chamada de função, os parâmetros formais (declarados no cabeçalho) não ocupam a mesma posição de memória dos parâmetros atuais (os parâmetros enviados) e na chamada é passada uma cópia do valor do parâmetro, então as operações feitas sobre o parâmetro dentro da função são feitas sobre os parâmetros formais, e não tem efeito nenhum na variável passada.
- Essa forma de passagem de parâmetro, que é a forma utilizada em C, se chama **passagem por cópia** ou **passagem por valor**, e é utilizada para a passagem de **parâmetros de entrada**, ou seja, parâmetros pelos quais não retornam resultados.

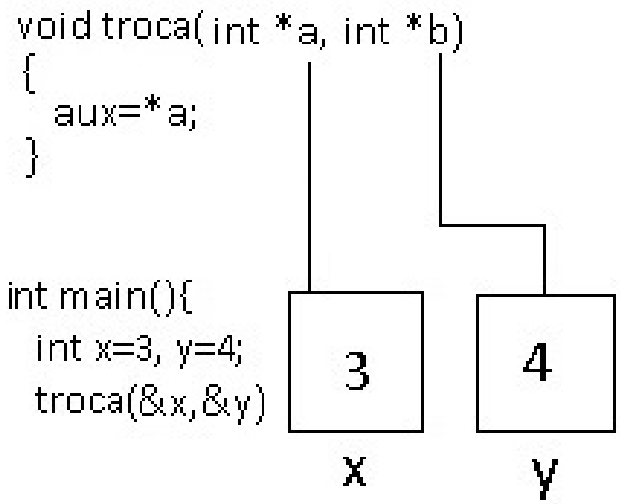


- Ao fazer a chamada são criadas a e b e após a cópia dos valores dos parâmetros atuais para os formais, não há mais ligação entre eles.
- Alterações nos parâmetros formais não tem efeito nenhum nos parâmetros atuais.

Passagem por referência (ou por endereço)

- Quando se deseja que as variáveis passadas como parâmetros tenham seus valores alterados dentro da função ou procedimento, utiliza-se a **passagem por referência** (ou **passagem por endereço**).
- Nesse tipo de passagem, ao invés de ser passada uma cópia do valor do parâmetro, é passado o endereço de memória da variável, de modo que alterações no parâmetro dentro da função ocorrem, realmente, na variável enviada como parâmetro.

- Para efetuar uma passagem por referência, o parâmetro formal deve ser declarado como um pointer para o tipo de valor a ser recebido (utilizando o operador *), e ao chamar a função envia-se o endereço da variável a ser enviada (utilizando-se o operador &).
- Por isso que no scanf as variáveis devem ser passadas com & e no printf não se usa o &.
  - A passagem de parâmetros do scanf é por referência e a passagem dos parâmetros do printf é por valor.
- Ex: void troca(int *x, int *y)



- Ao fazer a chamada **a** e **b** recebem os endereços dos parâmetros enviados, e através desses endereços acessam diretamente as variáveis enviadas.
- Alterações nas variáveis apontadas pelos parâmetros formais são efetuadas diretamente sobre os parâmetros atuais.

- Dentro da função, toda a referência à variável apontada pelo parâmetro formal deve ser feita através do operador *.

```
void troca(int *x, int *y)

{

int aux;

aux=*x; // leia-se "aux recebe o valor apontado por x"

*x=*y; // a posição apontada por x recebe o valor apontado por y

*y=aux; // a posição apontada por y recebe o valor em aux

}

int main()

{

int a,b;

troca(&a,&b); // envia os endereços de a e b

}
```

- A passagem por referência é utilizada para a passagem de parâmetros de saída, ou seja, parâmetros enviados para receber resultados de processamento.
- Normalmente são utilizados quando se precisa retornar mais de um valor, o que impede o uso de função, que só pode retornar um valor.
- A passagem por referência obriga que os parâmetros passados sejam variáveis, ao contrário da passagem por valor, em que os valores passados na chamada podem ser constantes ou expressões aritméticas

- Passagem de vetores em matrizes é sempre por referência, ou seja, sempre é passado o endereço inicial do vetor ou matriz.
- Para a passagem de vetores e matrizes, não deve ser usado o &.

- 1) Faça uma função que receba uma hora, minuto e segundo e retorne, nos mesmos parâmetros, a hora, minuto e segundo correspondentes ao segundo seguinte.
- 2) Faça uma função que receba uma data, formada por dia, mês e ano, e retorne, nos mesmos parâmetros, a data do dia seguinte. Considere que ano bisexto é aquele divisível por 4. Considere também que abril, junho, setembro e novembro tem 30 dias, fevereiro tem 28 (29 se o ano for bisexto) e todos os outros meses tem 31 dias.
- 3) Faça uma função que receba um vetor M[10] e retorne, no mesmo parâmetro, o vetor ordenado em ordem crescente.

- 4) Faça uma função que receba dois vetores V[10] e W[10] e retorne um terceiro vetor Z com os elementos que estão em V e W (interseção). Retorne também, em um quarto parâmetro o número de elementos que foram colocados em Z.
- 5) Faça uma função que receba dois vetores V[10] e W[10] e ordene os dois vetores, gere a seguir um terceiro vetor Z[20] com todos os elementos de V e W, também ordenado, retornando o vetor Z.

Atividades no site hackerrank.com

- Acesse o site: https://www.hackerrank.com/
- Efetue um cadastro no site ("sign up", canto superior direito da tela)
- Resolva os primeiros exercícios de "warm up" (aquecimento) submetendo-os até obter "accepted" em cada um, categoria "easy".
- Resolva o exercício Extra Long Factorials, primeiro exercício da categoria "medium" (para isso precisaremos de algumas funções extras...).

- Ao clicar em "visualize execution" é feita uma simulação da execução do código (aparece uma mensagem pedindo para esperar 10 segundos) e é gerada uma sequência de frames sobre a qual é exibida a simulação.
    - Eventualmente o servidor pode estar sobrecarregado, ou a duração da simulação excede o limite de frames permitido, e não será possível fazer a simulação.
- Como a exibição é feita após a simulação, não é possível interação com o usuário (ou seja, não há comandos de entrada (scanf))
- Dados de teste podem ser inseridos na declaração de variáveis. Ex na lâmina seguinte:

.

C (gcc 4.8, C11) EXPERIMENTAL!
see known bugs and report to philip@pgbovine.net

```
1  #include <stdio.h>
2  typedef struct nodo{int val;struct nodo *prox;} Tnodo;
3  int main() {
4  int m[10]={9,8,7,6,5,4,3,2,1,0},i,j,aux;
5  for (j=0;j<9;j++)
6  for (i=0;i<9;i++)
7     if (m[i]>m[i+1])
8     {
9       aux=m[i];
10     m[i]=m[i+1];
11     m[i+1]=aux;
12    }
13   return 0;
14 }
```

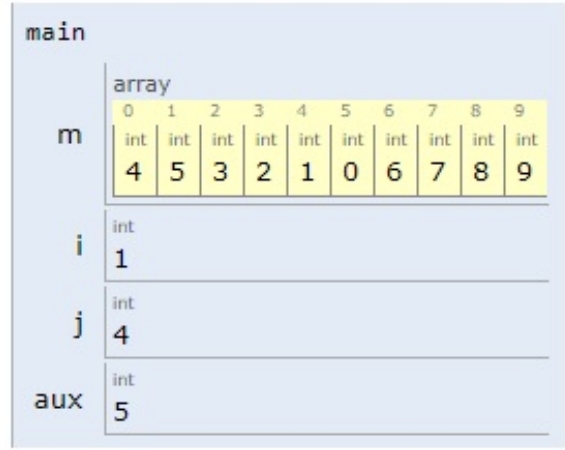
Edit code

line that has just executed
next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

<< First | < Back | Step 179 of 318 | Forward > | Last >>

Stack



- Os botões First e Last permitem ir até o primeiro e o último frame da simulação.
- Os botões Back e Forward permitem avançar ou voltar um frame.
- É possível setar pontos de parada no código para que, ao chegar no ponto selecionado, a execução seja pausada.
  - Isso é feito clicando uma vez sobre a linha a ser inserido o break point. A linha ficará em vermelho.
  - Ao clicar em Forward, ao passar sobre o ponto de parada a simulação será interrompida.
  - Ao clicar em Last os breakpoints serão ignorados
  - Para remover o breakpoint, basta clicar novamente sobre a linha.

Variáveis compostas heterogêneas (Structures)

- Uma *structure* é uma coleção de várias variáveis, possivelmente de tipos diferentes, agrupadas em uma única estrutura.
- São utilizadas para poder manipular conjuntos de informações relacionadas entre si, como por exemplo, os dados que compõem uma data, ou os dados de um cliente de banco.
- Cada uma das variáveis que compõem uma *structure* é chamada de **campo** da structure.

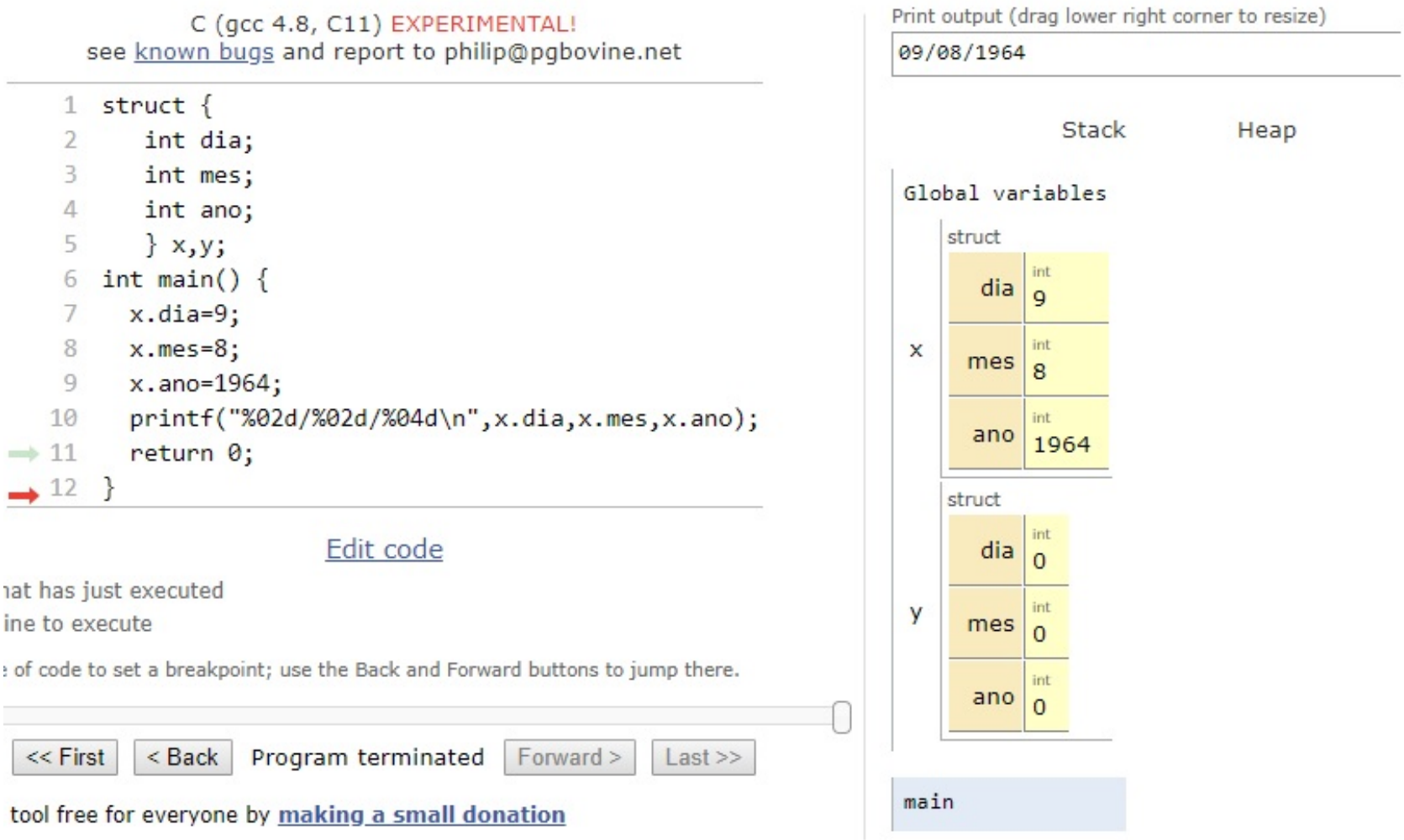
Índice

- Uma variável do tipo structure é declarada da forma a seguir:

  struct { lista de campos } nome;

- O exemplo abaixo declara uma variável chamada x do tipo structure com três campos inteiros:

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


C (gcc 4.8, C11) EXPERIMENTAL!
see known bugs and report to philip@pgbovine.net
1 struct {
2 int dia;
3 int mes;
4 int ano;
5 } x,y;
6 int main() {
7 x.dia=9;
8 x.mes=8;
9 x.ano=1964;
10 printf("%02d/%02d/%04d\n",x.dia,x.mes,x.ano);
11 return 0;
12 }
Edit code
hat has just executed
ine to execute
of code to set a breakpoint; use the Back and Forward buttons to jump there.
<< First
< Back
Program terminated
Forward >
Last >>
tool free for everyone by making a small donation
Print output (drag lower right corner to resize)
09/08/1964
Stack
Heap
Global variables
x
struct
dia
int
9
mes
int
8
ano
int
1964
y
struct
dia
int
0
mes
int
0
ano
int
0
main

- Para trabalhar com *structures*, pode-se associar um nome a um tipo de *structure*, que pode ser posteriormente utilizado em diversas declarações.
- Por exemplo, para a *structure* anterior pode-se usar o nome dma:

```
struct dma {

int dia;

int mes;

int ano;

};

struct dma x; /* um registro x do tipo dma */

struct dma y; /* um registro y do tipo dma */
```

- Para se referenciar um campo de uma structure, utiliza-se a notação estrutura.campo.

  x.dia = 31 /* o campo dia da estrutura x recebe 31 */

  x.mes = 8;

  x.ano = 1998;

C (gcc 4.8, C11) EXPERIMENTAL!
see known bugs and report to philip@pgbovine.net

```
 1  #include <stdio.h>
 2  // define tipo
 3  struct dma{int dia;int mes;int ano;};
 4  // define variável data global
 5  struct dma dmaglob={9,8,1964};
 6  int main() {
 7     // define variável data local
 8     struct dma dmalocal={1,4,1500};
 9     printf("Data=%02d/%02d/%04d\n",
10        dmalocal.dia,
11        dmalocal.mes,
12        dmalocal.ano);
13     return 0;
14  }
```

Edit code

s just executed
execute

ode to set a breakpoint; use the Back and Forward buttons to jump there.

<< First | < Back | Step 4 of 4 | Forward > | Last >>

Print output (drag lower right corner to resize)

```
Data=01/04/1500
```

Stack Heap

Global variables

	struct dma	
dmaglob	dia	int 9
	mes	int 8
	ano	int 1964

main

	struct dma	
dmalocal	dia	int 1
	mes	int 4
	ano	int 1500

- Diferentemente de vetores e matrizes, variáveis do tipo estrutura podem receber o valor de outra do mesmo tipo em uma única atribuição.
- Ex:

```
struct dma {

int dia;

int mes;

int ano;

} d1,d2;

d1=d2; /* atribui todos os campos ao mesmo tempo */
```

- Da mesma forma, structures podem ser passadas para funções e retornadas por funções.
- Exemplo: A função abaixo recebe a data de início de um evento e a duração do evento em dias.
- Ela devolve a data de fim do evento.

```
struct dma fim_evento (struct dma datainicio,
int duracao){
struct dma datafim;
. . .
. . .
datafim.dia = ...
datafim.mes = ...
datafim.ano = ...
return datafim;
```

- O código deve levar em conta a existência de meses com 31 dias, de meses com 30 dias, com 29 dias etc. Eis como essa função poderia ser usada:

```
int main( ) {

    struct dma a, b;

    int d;

    scanf( "%d %d %d", &a.dia, &a.mes, &a.ano);

    scanf( "%d", &d);

    b = fim_evento( a, d);

    printf( "%d %d %d\n", b.dia, b.mes, b.ano);

    }
```

Exercícios

1. Refaça os exercícios 2, 3 e 4 utilizando structures;
2. Complete o código da função fim_evento acima.
3. Escreva uma função que receba dois structs do tipo dma, cada um representando uma data válida, e devolva o número de dias que decorreram entre as duas datas.
4. Escreva uma função que receba um número inteiro que representa um intervalo de tempo medido em minutos e devolva o correspondente número de horas e minutos (por exemplo, converte 131 minutos em 2 horas e 11 minutos). Use uma struct como a seguinte:

```
struct hm {
    int horas;
    int minutos;
};
```

- Em C pode-se redefinir um tipo de dado dando-lhe um novo nome.
- Essa forma de programação simplifica a referência ao tipo.
- Para redefir-se o nome de um tipo de dado usa-se o comando **typedef**, que provém de *type definition*.
- **typedef** faz o compilador assumir que o novo nome é um certo tipo de dado, e então, passamos a usar o novo nome da mesma forma que usaríamos o antigo.

- A sintaxe do **typedef** é:

  typedef especificação_do_tipo_original nome_novo;

- Ex:

  typedef unsigned int **uint**;

  typedef int **boolean**;

  typedef struct dma{int d;int m;int a;} **tdata**;

- Pode-se definir o nome de uma estrutura de dados (struct) de duas maneiras:
  - a) Definindo o nome da estrutura e só depois definindo a estrutura; p.ex:

    typedef struct estrutura1 MinhaEstrutura;

    struct estrutura1 {

        int var1;
        float var2;

    };
  - b) Definindo a estrutura ao mesmo tempo que define o nome.

    typedef struct estrutura1 {

        int var1;
        float var2;

    } MinhaEstrutura;

- Tendo uma estrutura de dados definida, pode-se utilizá-la dentro de outra estrutura de dados.
- Por exemplo:

```
typedef struct estrutura1 MinhaEstrutura;

struct estrutura1 {
        int var1;
        float var2;
        };
struct estrutura2 {
        int var3;
        MinhaEstrutura var4;
        struct estrutura1 campo5;
};
```

- Como mostra o exemplo anterior, pode-se usar tanto o novo nome (MinhaEstrutura) quanto o tipo definido (struct estrutura1)
- Ex:

```
typedef struct data {
unsigned short dia;
unsigned short mes;
unsigned int ano;
} Data;
typedef struct aniversario {
char nome[50];
Data nascimento;
} Aniversario;
```

- Uma estrutura pode conter como campo uma outra estrutura.
- Para isso é conveniente que a estrutura interna tenha sido anteriormente definida (com typedef ou que tenha sido definido um nome para ela).
- Ex:

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

typedef struct {

char nome[30];

date nascimento;

} reg;

struct teste2 {int a,b;};

struct teste3 {

struct teste2 x;

int y;
```

- Estruturas podem ser inicializadas na declaração, da mesma forma como é feito com variáveis escalares e arrays.
- Para inicializar, a lista de valores iniciais dos campos é delimitada por .
- Esse tipo de atribuição só pode ser feito na declaração.

```
date v2={9,8,1964};

reg lista[2]={{"José",{9,8,1963}},

{"João",{9,8,1964}}};

reg x={"João",{9,8,1964}};
```

C (gcc 4.8, C11) EXPERIMENTAL!
see known bugs and report to philip@pgbovine.net

```
1  #include <stdio.h>
2  typedef struct {int dia,mes,ano;} date;
3
4  typedef struct {
5     char nome[10];
6     date nascimento;
7     } reg;
8  reg reg1={"Joao",{1,4,1500}};
9
10 struct teste2 {int a,b;};
11 struct teste3 {
12    struct teste2 x;
13    int y;
14    };
15 struct teste3 z;
16 int main() {
17    return 0;
18 }
```

Edit code

as just executed
) execute

ode to set a breakpoint; use the Back and Forward buttons to jump there.

<< First | < Back | Step 1 of 1 | Forward > | Last >>

Global variables

reg1
- struct reg
  - nome: array [0..9] = 'J' 'o' 'a' 'o' '\0' '\0' '\0' '\0' '\0' '\0'
  - nascimento: struct date
    - dia: int 1
    - mes: int 4
    - ano: int 1500

z
- struct teste3
  - x: struct teste2
    - a: int 0
    - b: int 0
  - y: int 0

main

1. Defina uma structure chamada cliente que contenha as seguintes informações:
   - Nome (char [30])
   - Data de nascimento (structure com dia, mês e ano)
   - Saldo: float
2. Crie um array de 10 elementos inicializados na declaração.
3. Liste em ordem alfabética do nome do cliente o nome e saldo de todos os clientes com saldo maior que R$10.000,00
4. Idem, em ordem de data de nascimento

Exercícios

1. Crie uma estrutura aluno contendo o seu nome (char 30), matrícula (char) e uma lista de até 5 disciplinas (cada disciplina é char 7) e turmas (cada turma é char 3) em que está matriculado. Crie uma lista com 10 alunos inicializando-a na declaração.

2. Faça um programa que leia o código de uma disciplina e o código da turma e liste os alunos da tuma.

- structures podem ser passadas como argumentos para funções, e podem ser retornados por funções. Para isso é necessário que o tipo de structure tenha sido definido antes.
- 1) Faça uma função MaisRico ( ) que receba um array de clientes e retorne o cliente com o maior saldo. Faça duas versões, uma que o retorne como valor de retorno da função, e outra que o retorne como parâmetro de retorno.
- 2) Faça uma função ListaClientes ( ) que recebe um array de clientes e os escreve (nome, data de nascimento e saldo) na ordem em que estão na lista
- 3) Faça uma função OrdenaPorSaldo ( ) que recebe um array de clientes e ordena a lista de clientes por ordem decrescente de saldo. Chame-a e utilize a função ListaClientes para verificar se a ordenação.
- 4) Idem para OrdenaPorNome ( ) e OrdenaPorDataNascimento ( ).

- A estrutura struct tm é uma *structure* definida em time.h e contem uma data e hora com os seguintes campos:

Nome do Campo	Tipo	Significado	Intervalo
tm_sec	int	seconds after the minute	0-61*
tm_min	int	minutes after the hour	0-59
tm_hour	int	hours since midnight	0-23
tm_mday	int	day of the month	1-31
tm_mon	int	months since January	0-11
tm_year	int	years since 1900	
tm_wday	int	days since Sunday	0-6
m_yday	int	days since January 1	0-365
tm_isdst	int	Daylight Saving Time flag	

- The Daylight Saving Time flag (tm_isdst) is greater than zero if Daylight Saving Time is in effect, zero if Daylight Saving Time is not in effect, and less than zero if the information is not available.
- * tm_sec é geralmente 0-59. O intervalo extra é para acomodar segundos "bisextos" em alguns sistemas
- time_t é um alias para um unsigned type, definido em time.h.

O código abaixo busca a hora e data do sistema e escreve valores numéricos correspondentes a cada elemento que a compõe (hora, minuto, segundo, dia, mes, ano e dia da semana). Altere-o para que escreva na tela a data e hora no formato:
Segunda-feira, 23 de abril de 1500, 16:23:47.

```
/* exemplo da função localtime */

#include < stdio.h >

#include < time.h >

#include < stdlib.h >

int main ()

{

time_t rawtime;

struct tm * timeinfo;

time ( &rawtime ); /* busca o número de segundos desde 1/1/1970 */

timeinfo = localtime ( &rawtime ); /* converte para uma estrutura de data-hora */

printf("hora=%d\n",timeinfo->tm_hour);

printf("minuto=%d\n",timeinfo->tm_min);

printf("segundo=%d\n",timeinfo->tm_sec);

printf("dia=%d\n",timeinfo->tm_mday);

printf("mes=%d\n",timeinfo->tm_mon+1);
```

E o operador &, aplicado a uma variável, retorna o endereço dela. Ex:

int a, *pta;

pta=&a; // pta recebe o endereço da variável a.

*pta=5; // a variável apontada por pta (ou seja, a variável a, recebe 5)

- 1) Seja o seguinte trecho de programa:

int i=3,j=5;

int *p, *q;

p = &i;

q = &j;

Qual é o valor das seguintes expressões ?

a) p = = &i;

b) *p - *q

Precedência de Operadores

- Qual a função de cada operador na expressão a seguir?

  3* - *p/(*q)+7

  - * = multiplicação
  - - = subtração unária (troca de sinal)
  - * = indireção (referência a posição apontada)
  - / = divisão
  - + = soma
- Qual a ordem em que os operadores serão avaliados?
  - Tabela de Precedência de Operadores

Tabela de precedência de operadores em C

Prec.	Símbolo	Função	Associatividade
1	++ - -	Incremento e decremento pósfixo	→
1	()	Parênteses (chamada de função)	
1	[ ]	Elemento de array	
1	.	Seleção de campo de structure	
1	->	Seleção de campo de structure a partir de ponteiro	
2	++ - -	Incremento e decremento prefixo	←
2	+ -	Adição e subtração unária	
2	! ~	Não lógico e complemento	
2	(tipo)	Conversão de tipo de dado	
2	*	Desreferência	
2	&	Referência (endereço de elemento)	
3	* / %	Multiplicação, divisão, e módulo (resto)	→
4	+ -	Adição e subtração	
5	<< >>	Deslocamento de bits à esquerda e à direita	
6	< <=	"menor que" e "menor ou igual que"	
6	> >=	"maior que" e "maior ou igual que"	
7	== !=	"Igual a" e "diferente de"	
8	&	E bit a bit	
9	^	Ou exclusivo para bits	
10	|	Ou para bits	
11	&&	E lógico	
12	||	Ou lógico	
13	c ? t : f	Condição ternária	←
14	=	Atribuição	
14	+= -=	Atribuição por adição ou subtração	
14	*= /= %=	Atribuição por multiplicação, divisão ou módulo (resto)	
14	<<= >>=	Atribuição por deslocamento de bits	
14	&= ^= |=	Atribuição por operações lógicas	
15	,	vírgula	→

Precedência de Operadores

Qual a função de cada operador na expressão a seguir?

3* - *p/(*q)+7

- * = multiplicação (precedência 3)
- - = subtração unária (precedência 2)
- * = indireção (precedência 2)
- / = divisão (precedência 3)
- + = soma (precedência 4)
- Em primeiro lugar será avaliado o operadores de indireção mais à direita, resultando a expressão:

  3* - *p/5+7
- Em seguida será avaliado o outro operador de indireção, já que tem mesma precedência da subtração unária e é associativo à direita, resultando

  3* - 3/5+7
- Em seguida será aplicada a subtração unária e em seguida será efetuada a multiplicação, resultando

  -9/5+7
- Agora é efetuada a divisão, resultando

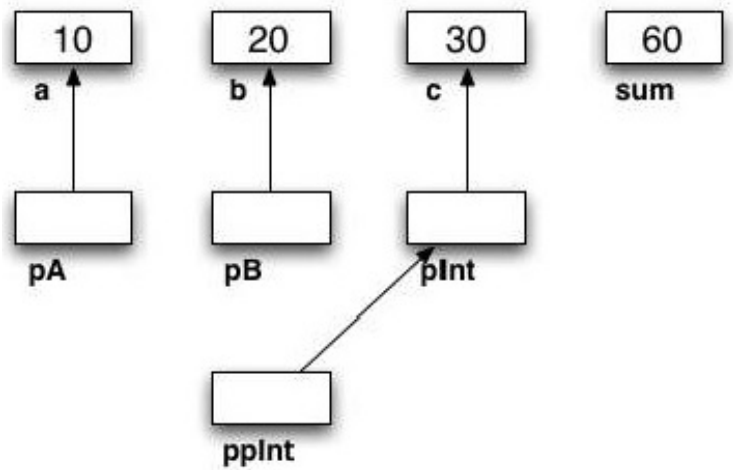
  -1+7
- Que resulta:

  6

- Pode-se escrever o valor de um ponteiro (endereço para o qual ele aponta) utilizando-se o especificador %p (mostra em hexa, 32 bits) ou %x (mostra em hexa, sem mostrar os 0's não significativos).
- Qual será a saída deste programa supondo que i ocupa o endereço 4094 na memória?

```
main() {
    int i=5, *p;
    p = &i;
    printf("%x %d %d %d %d \n", p,*p+2,**&p,3**p,**&p+4);
    printf("%d\n",(*p)++ + *q);
    printf("%d\n",i);
    printf("%d\n",++(*p) + *q);
}
```

```
void pointers2pointers()
   {
   int a=10, b=20, c=30, sum=0;
   int *pA=&a,*pB=&b,*pInt;
   int **ppInt = &pInt;
   (*ppInt)=pA;
   sum += (**ppInt);
   (*ppInt)=pB;
   sum+=(*pInt);
   *ppInt=&c;
   sum+=(**ppInt);
   printf("Soma é %d\n",sum);
   }
```



Um ponteiro pode ser usado no lado direito de um comando de atribuição para passar seu valor para outro ponteiro.

```
#include <stdio.h>

void main() {

int x;

int *p1,*p2; /* declaração do ptr p1 e p2 com o tipo base int. */

p1 = &x;

p2 = p1;

printf("%p",p2); /* escreve o endereço de x em 32 bits, não seu valor */

return 0;

}
```

- Agora, tanto p1 quanto p2 apontam para x.
- O endereço de x é mostrado usando o modificador de formato de printf() %p, que faz com que printf() apresente um endereço no formato usado pelo computador hospedeiro.

Aritmética de Ponteiros

- Existem apenas duas operações aritméticas que podem ser usadas com ponteiros: adição e subtração.
- Os operadores permitidos no caso seriam: +, -, ++, - -.
- O incremento é sempre realizado do tamanho básico de armazenamento do tipo base.
- Isto é, se o tipo base for um inteiro e incrementarmos em uma unidade, o apontador apontará para o próximo inteiro (no caso do inteiro ocupar 4 bytes o incremento será de 4 bytes), no caso de um caractere (char) será de um byte.

Índice

Número de bytes dos tipos em C

tipo	Tamanho
short int	2
int	4
long int	4
long long int	8
double	8
float	4

```
int *ptri=3000;

char *ptrc=4000;

float *ptrf=4000;

ptri++; /* ptri apontará para o endereço 3004 */

ptrc++; /* ptrc apontará para o endereço 4001 */

ptrf++; /* ptrf apontará para o endereço 5004 */

ptri = ptri - 10; /* ptri apontará para o endereço 2964 */

ptrc = ptrc - 10; /* ptrc apontará para o endereço 3991 */

ptrf = ptrf - 10; /* ptrf apontará para o endereço 4964 */
```

- Além de adição e subtração entre um ponteiro e um inteiro, nenhuma outra operação aritmética pode ser efetuada com ponteiros.
- Isto é, não se pode multiplicar ou dividir ponteiros; não se pode aplicar os operadores de deslocamento e de mascaramento bit a bit com ponteiros e não se pode adicionar ou subtrair o tipo float ou o tipo double a ponteiros.

- Se a e b são variáveis inteiras e pA e pB ponteiros para int, quais das seguintes expressões de atribuição são ilegais?

a) pA = &a;

b) *pB = &b;

c) pA = &*&a;

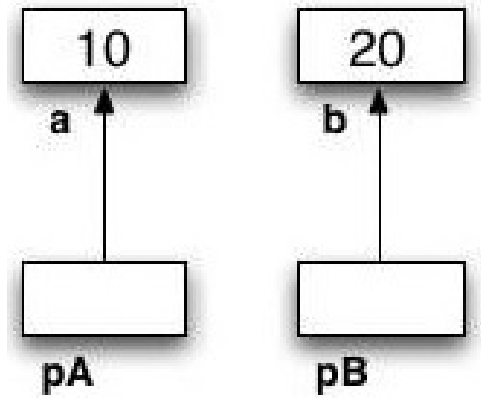
d) a = (*&)b;

e) a = *&b;

f) a = *&*&b;

g) pB = *pA;

h) a = (*pA)++ + *pB



- Resposta: as ilegais são B, D, G

- 6. Assumindo que pulo[] é um vetor do tipo int, quais das seguintes expressões referenciam o valor do terceiro elemento da matriz?
  - a) *(pulo + 2)
  - b) *(pulo + 4)
  - c) pulo + 4
  - d) pulo + 2
- 7. Supor a declaração: int mat[4], *p, x; Quais expressões são válidas? Justifique:
  - a) p = mat + 1;
  - b) p = mat++;
  - c) p = ++mat;
  - d) x = (*mat)++;

Alocação dinâmica de memória

- **Pointers** são utilizados para referenciar variáveis (normalmente structures, vetores e matrizes) alocadas dinamicamente, ou seja durante a execução do programa.
- As funções utilizadas para isso são malloc(), calloc() e free().
- Protótipos

#include < stdlib.h >

void * calloc (size_t nmemb , size_t size);

void * malloc (size_t size);

void free (void * ptr);

Pode-se considerar que size_t é equivalente a int.

Descrição

- **calloc** reserva memória para um vetor de nmemb elementos de tamanho size bytes cada e retorna um ponteiro para a memória reservada. A memória é inicializada com zero.
- **malloc** reserva size bytes e retorna um ponteiro para a memória reservada. A memória não é inicializada com zero.
- **free** libera a área de memória apontada por ptr, que foi previamente reservada por uma chamada a uma das funções malloc() ou calloc(). Comportamento indefinido ocorre se a área já foi liberada antes ou as funções não foram chamadas antes. Se o valor de ptr é NULL nada é executado.

- Retorno
  - Em **calloc()** e **malloc()**, o valor retornado é um ponteiro para a memória alocada, que é alinhada corretamente para qualquer tipo de variável ou NULL se o pedido não pode ser atendido.
  - O ponteiro retornado é do tipo void *. Para que ele possa ser atribuído a ponteiros de outro tipo é necessário fazer a conversão de tipo.
  - Para saber o tamanho da área necessária para uma variável, pode-se utilizar a função sizeof( ).
  - Ex:

```
struct cliente *pcli=(struct cliente *)malloc(sizeof(struct cliente));
```

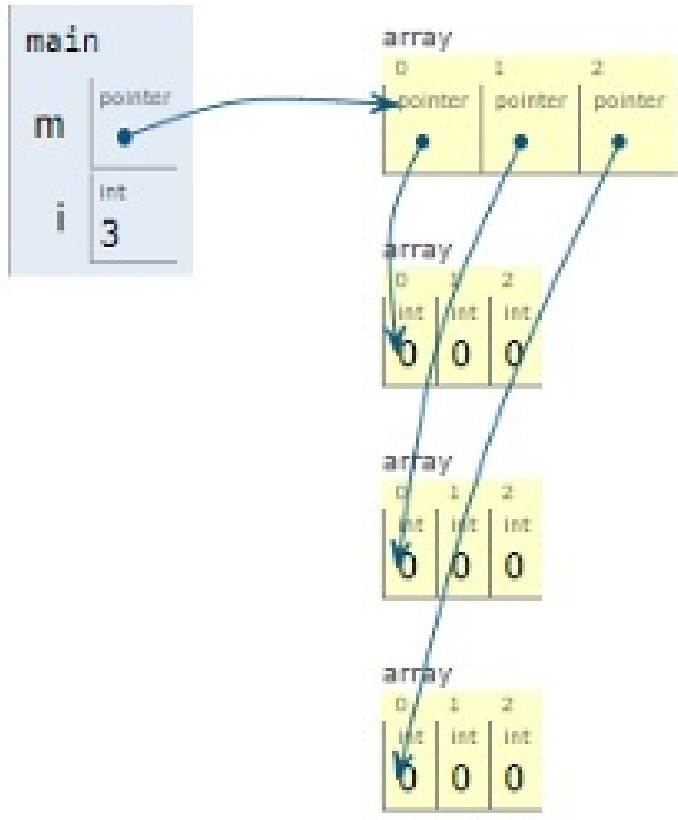
- **free()** não retorna nenhum valor.

Alocação dinâmica de matriz

- Uma forma de alocar uma matriz dinamicamente é criando um vetor de pointers, onde cada posição é um pointer para uma linha da matriz.
- O código abaixo exemplifica a criação de uma matriz de inteiros de 3 linhas e 3 colunas.

```
int **m=(int**)malloc(3*sizeof(int *));

int i;

for (i=0;i<3;i++)

    m[i]=(int*)malloc(3*sizeof(int));
```

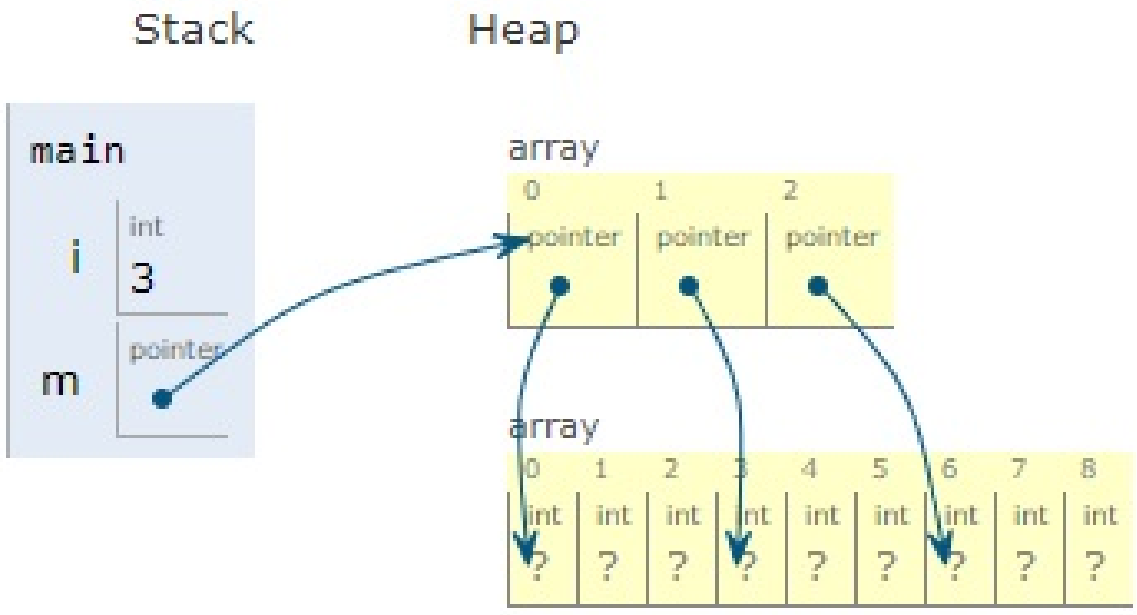
- E a estrutura alocada é mostrada na lâmina a seguir


main
m
pointer
i
int
3
array
0
1
2
pointer
pointer
pointer
array
0
1
2
int
int
int
0
0
0
array
0
1
2
int
int
int
0
0
0
array
0
1
2
int
int
int
0
0
0

- Uma alternativa é alocar a matriz toda como um único vetor e alocar um vetor de pointers que apontarão para o início de cada linha.

```
int **m=(int**)malloc(3*sizeof(int *));

m[0]=(int*)malloc(3*3*sizeof(int));

int i;

for (i=1;i<3;i++)

    m[i]=m[i-1]+3;
```

- A estrutura resultante é mostrada na lâmina a seguir


Stack
Heap
main
i
int
3
m
pointer
array
0
1
2
pointer
pointer
pointer
array
0
1
2
3
4
5
6
7
8
int
int
int
int
int
int
int
int
int
?
?
?
?
?
?
?
?
?

Comparação de Ponteiros

- É possível comparar dois ponteiros em uma expressão relacional.

```
if (p<q)
    printf("p aponta para uma memória mais baixa que q \n!");
```

- Geralmente, a utilização de comparação entre ponteiros é quando os mesmos apontam para um objeto comum.
- Exemplo disto é a pilha, onde é verificado se os ponteiros de início e fim da pilha estão apontando para a mesma posição de memória, significando pilha vazia.

Ponteiros e Matrizes

- Existe uma estreita relação entre matrizes e ponteiros. C fornece dois métodos para acessar elementos de matrizes: aritmética de ponteiros e indexação de matrizes.
- Aritmética de ponteiros pode ser mais rápida que indexação de matrizes.
- Normalmente utilizam-se ponteiros para acessar elementos de matrizes devido à velocidade de acesso.
- Exemplo
  - char str[80], *p1;
  - p1 = str;
  - Para acessar a string str pode-se utilizar estes dois mecanismos:
    - str[4] /* indexação de matrizes */
  - ou
    - *(p1 + 4) /* aritmética de ponteiros */
  - Os dois comandos devolvem o quinto elemento.

- Para uma melhor compreensão ou facilidade de programação as funções de indexação trabalham com ponteiros (como mostra a implementação abaixo, da função puts()).

```
/* Indexa s como uma matriz */
void put(char *s){
    register int t;
    for (t=0;s[t]; ++t) putchar(s[t]);
}
/* Acessa s como um ponteiro */
void put(char *s) {
    while (*s) putchar(*s++);
}
```

No caso da passagem de parâmetros é possível tratar uma matriz como se fosse um ponteiro.

```
#include <stdlib.h>

#include <stdio.h>

#include <string.h>

void le_tab(int *p) {

register int i;

for(i=0; i<20; i++)

      scanf("%d",(p+i));

}

void mostra_tab(int *p) {

register int i;

for(i=0; i<20; i++)

printf("%d",*(p+i));

}

void main(void) {

int mat[20];

le_tab(mat);

mostra_tab(mat);

}
```

Operadores bit a bit

- Os operadores bit a bit são comumente utilizados para trabalhar com dispositivos (pois os mesmos utilizam bytes ou palavras codificadas para comunicação), modo de armazenamento (um byte pode representar 8 informações binárias), e até compactação de dados (utilização de bits ociosos). A tabela a seguir mostra os operadores bit a bit suportados pela linguagem.

Índice

Operadores bit-a-bit

Operador	Ação
&	E (AND)
—	OU (OR)
^	OU exclusivo (XOR)
~	Complemento de um
<<	Deslocamento à esquerda
>>	Deslocamento à direita

- Os operadores bit a bit só podem ser utilizados sobre um byte ou uma palavra, isto é, aos tipos de dados char e int e variantes do padrão C. Operações bit não podem ser usadas em float, double, long double, void ou outros tipos mais complexos.

- E a tabela verdade de cada operador é:

A	B	A & B	A—B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

A	~ A
0	1
1	0

O Operador &

- O operador bit a bit & executa um e bit-a-bit para cada par de bits, produzindo um novo byte ou palavra.
- Para cada bit dos operandos, o operador & retorna o bit em 1 se ambos os bits dos operandos é 1.
- Caso algum bit dos operandos for 0, o operador retorna o bit 0.
- Este operador é mais utilizado para desligar bits (realizando a operação com um operando também chamado de máscara cujos bits que devam ser desligados estejam com valor 0, enquanto que os outros estão em 1) ou verificar se um determinado bit está ligado ou desligado (realizando a operação com uma máscara cujo bit que deva ser checado esteja com valor 1, enquanto que os outros estão em 0).

- Exemplo

```
unsigned char x = 7; /* 0000 0111 */

unsigned char y = 4; /* 0000 1010 */

unsigned char mascara = 252; /* 1111 1100 */

unsigned char res;

res = x & y; /* res = 0000 0010 */

res = y & mascara; /* res = 0000 1000 ? desligar os bits 0 e 1 */

res = y & 2 /* res = 0000 0010 ? bit ligado qdo res ¿ 0 */

res = y & 4 /* res = 0000 0000 ? bit desligado qdo res == 0 */
```

- Para gerar uma máscara com 0 na posição p (considerando as posições como numeradas a partir de 0 da direita para a esquerda) e 1 nas outras posições pode-se fazer:
  - Mascara =~(1 << p)

Operador "ou" bit-a-bit

- O operador bit a bit | executa um **ou lógico** para cada par de bits, produzindo um novo byte ou palavra.
- Para cada bit dos operandos, o operador | retorna o bit em 1 se algum dos bits dos operandos é 1.
- Caso ambos os bits dos operandos for 0, o operador retorna o bit 0.
- Este operador é mais utilizado para ligar (realizando a operação com um operando cujos bits que devam ser ligados estejam com valor 1, enquanto que os outros que não devem ser alterados estão em 0).

Exemplo

```
unsigned char x = 7; /* 0000 0111 */

unsigned char y = 4; /* 0000 1010 */

unsigned char mascara = 1; /* 0000 0001 */

unsigned char res;

res = x — y; /* res = 0000 1111 */

res = y | mascara; /* res = 0000 1011 ? ligar o bit 0 */

res = x | 8; /* res = 0000 1111 ? ligar o bit 3 */
```

- Para gerar uma máscara com 1 na posição p (considerando as posições como numeradas a partir de 0 da direita para a esquerda) e 0 nas outras posições pode-se fazer:
  - Mascara = 1 << p

- Não confunda os operadores relacionais && e || com & e |, respectivamente. Os operadores relacionais trabalham com os operandos como um único valor lógico (verdadeiro ou falso), e eles produzem somente dois valores 0 ou 1. Os operadores bit a bit podem produzir valores arbitrários pois a operação é realizada em nível de bit.

Operador de ou-exclusivo

- O operador bit a bit ^ executa um ou-exclusivo (XOR) lógico para cada par de bits, produzindo um novo byte ou palavra.
- Para cada bit dos operandos, o operador ^ retorna o bit em 1 se somente um dos bits dos operandos é 1.
- Caso os bits dos operandos forem iguais, o operador retorna o bit 0.
- Este operador é mais utilizado para inverter bits (realizando a operação com um operando cujos bits que devam ser invertidos estejam com valor 1, enquanto que os outros estão em 0).
- Trivia: o que resulta do comando abaixo?
  - a^=b^=a^=b;

Exemplo

unsigned char x = 7; /* 0000 0111 */

unsigned char y = 4; /* 0000 1010 */

unsigned char mascara = 3; /* 0000 0011 */

unsigned char res;

res = x y; / $*$ res = 00001101 $*$ /res = $\hat{y}$ mascara; /* res = 0000 1001 ? inverter os bits 0 e 1 */

res = y 8; / $*$ res = 00000010?$inverterobit3$ $*$ /

Operador de complemento ~

- O operador bit a bit ~ executa um não lógico (complemento de 1) no valor a sua direita (operador unário), produzindo um novo byte ou palavra com os bits invertidos.
- Exemplo

```
unsigned char x = 7; /* 0000 0111 */

unsigned char res;

res = ~x; /* res = 1111 1000 */

res = ~127; /* res = 1000 0000 */
```

Operadores de deslocamento

- Os operadores de deslocamento, ¡¡ e ¿¿, movem todos os bits de um operando para a esquerda ou direita, respectivamente. A forma geral do comando de deslocamento é:
- Sintaxe:
  - operando $<<$ número de bits
  - operando $>>$ número de bits
- Conforme os bits são deslocados para um direção, zeros ou uns são utilizados para preencher a extremidade contrária da direção (isto é, deslocamento para a direita coloca zeros ou uns nos bits mais significativos).

- No deslocamento à direita de valores com sinal (sem o modificador unsigned), o valor introduzido pela esquerda é um valor igual ao bit de sinal. Se o valor é negativo, com bit de sinal igual à 1, 1's são inseridos à esquerda.
- Valores inteiros são representados em **complemento de 2**.
  - Se o bit mais significativo é 1, o valor é negativo.
  - Pode-se obter o valor positivo correspondente invertendo todos os bits e somando 1 ao resultado.

- Estes operadores são utilizados para recebimento e envio de dados bit a bit (conversores analógico/digitais, portas seriais), e multiplicação (deslocamento para a esquerda) e divisão inteira (deslocamento para a direita) por 2.
- Os operadores de deslocamento podem ser utilizados com variáveis, constantes e até mesmo expressões. Entretanto, deve-se verificar a precedência de operadores quando trabalhando com expressões.
- Exemplo

```
unsigned char res, x = 7; /* 0000 0111 */

res = x << 1; /* res = 00001110 res = 14 */

res = x << 3; /* res = 01110000 res = 112 */

res = x << 2; /* res = 11000000 res = 192 */

res = x >> 1; /* res = 01100000 res = 96 */

res = x >> 2; /* res = 00011000 res = 24 */
```

Operadores bitwise de atribuição

- A Tabela a seguir mostra os operadores bit a bit de atribuição suportados pela linguagem.

Operador	Ação
x &= y	x = x & y
x |= y	x = x | y
x^= y	x = x^y
x >>= y	x = x >> y
x <<= y	x = x << y

Figure: Operadores bitwise reduzidos

- As expressões com este operadores são mais compactas e normalmente produzem um código de máquina mais eficiente.
- A execução da operação bit a bit ocorre por último após a avaliação da expressão à direita do sinal de igual.

Exercícios

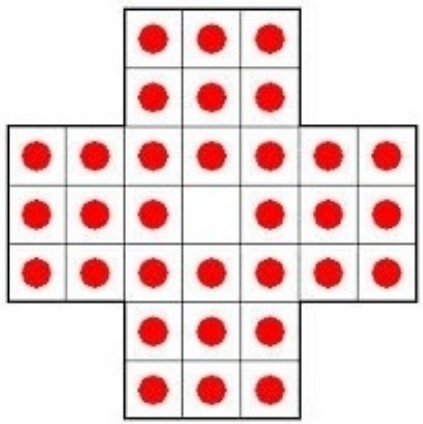
- 1) Uma forma de representar um conjunto é através de um vetor de bits. Por exemplo, um conjunto de valores entre 0 e 7 poderia ser representado como um unsigned char, onde cada bit i de 0 a 7 representaria a presença ou não do elemento i no conjunto. Assim, o conjunto 1,4,5 seria representado pelo char 00110010. Da mesma forma, um conjunto de valores entre 0 e 31 poderia ser representado por um unsigned int. Utilizando operadores binários, faça as funções a seguir:

1. void insere (unsigned int *conjunto, int valor) que insere um valor entre 0 e 31 em um conjunto.
2. int testa (unsigned int conjunto, int valor) que verifica se um dado valor está em um conjunto, retornando 0 ou 1.
3. void remove (unsigned int *conjunto, int valor) que remove um valor entre 0 e 31 de um conjunto.
4. void mostra (int conjunto) que mostra os valores que fazem parte do conjunto.
5. void mostrabin (int conjunto) mostra o valor do conjunto em binário.

- Idem, para as funções a seguir:

1. unsigned int intersecao (unsigned int c1, unsigned int c2) : retorna um conjunto com os elementos comuns (intersecção) dos conjuntos c1 e c2;
2. unsigned int uniao(unsigned int c1, unsigned int c2) : retorna um conjunto com os elementos que aparecem em pelo menos um dos conjuntos c1 e c2 (união);
3. unsigned int diferença(unsigned int c1, unsigned int c2) : retorna um conjunto com os elementos que estão em c1 mas não estão em c2;
4. void rota_esq (unsigned int *valor) : rotaciona os bits uma posição à esquerda, o bit mais significativo deve ir para a posição 0;
5. void rota_dir (unsigned int *valor) : rotaciona os bits uma posição à direita, o bit 0 deve ir para o bit mais significativo;

- 2) O desafio "resta-um" é jogado em um tabuleiro no formato abaixo, que pode ser representado por uma matriz 7 x 7 na qual são usadas somente as posições representadas na figura. Cada posição da matriz tem 1 para as casas ocupadas e 0 para as casas não ocupadas. Faça uma função que receba uma matriz M[7][7] de inteiros iguais a 0 ou 1, e retorne um inteiro de 32 bits sem sinal em que cada bit contem o valor correspondente a uma posição da matriz. Represente apenas os primeiros 32 bits dos 33 que são usados para representar o tabuleiro.



3)(2.5 pontos) Faça uma função que receba uma matriz M[4][4] de inteiros iguais a 0 ou 1, e retorne um inteiro de 16 bits sem sinal em que cada bit contem o valor correspondente a uma posição da matriz. O bit 15 deve conter o valor da posição m[0][0] e o bit 0 deve conter o valor da posição [3][3]. Ex: Se a matriz M contem:

0	1	0	1
1	1	1	1
0	1	0	1
0	0	0	0

A função deve retornar o valor inteiro correspondente ao binário 0101 1111 0101 0000.

4) Diz-se que um valor está no formato *big endian*, quando as partes mais significativas do valor aparecem nas primeiras posições. Assim, números decimais estão no formato *big endian* porque os dígitos de maior peso aparecem antes dos dígitos de menor peso. Da mesma forma, em valores representados em little endian, a parte de menor peso aparece no início. Ex: datas em formato americano (AAAA/MM/DD).

- Processadores da linha Intel x86 armazenam valores inteiros no formato little endian, ou seja, os bytes menos significativos estão nas posições inferiores de memória.
- O que é escrito pelo programa a seguir?

```
#include <stdio.h>

#include <stdlib.h>

int main()

{

unsigned int a=0xABCDEF12;

unsigned char *p=(unsigned char *)&a;

for (int i=0;i<4;i++)

      printf("p[%d]%x\n",i,p[i]);

system("pause");

}
```

- Faça uma função que receba um inteiro de 4 bytes em formato *little endian* e retorne, como um segundo parâmetro, o valor convertido para o formato *big endian*.

Campos de Bits

- C tem um método intrínseco para acessar um único bit dentro de um byte. Isso pode ser útil por diversas razões:
  - Se o armazenamento é limitado, você pode armazenar diversas variáveis Booleanas (verdadeiro/falso) em um byte.
  - Certos dispositivos transmitem informações codificadas nos bits dentro de um byte.
  - Certas rotinas de criptografia precisam acessar os bits dentro de um byte. Para acessar os bits, C usa um método baseado na estrutura. Um campo de bits é, na verdade, apenas um tipo de elemento de estrutura que define o comprimento, em bits, do campo. A forma geral de uma definição de campo de bit é:

- Para declarar um campo de bits, define-se o tipo do campo e a lista de sub-campos, com o número de bits em cada um:

  struct nome {

  tipo <lista de campo:comprimento>;

  } lista de variaveis;

  Ex:

  struct pixel {

  unsigned short int r : 5, g : 6, b : 5;

  } p1,p2,p3;

- Um campo de bit deve ser declarado como int, unsigned ou signed. Campos de bit de comprimento 1 devem ser declarados como unsigned, porque um único bit não pode ter sinal. (Alguns compiladores só permitem campos do tipo unsigned).
- Um exemplo de campos de bits é a comunicação via serial que devolve um byte de estado organizado como mostra a tabela a seguir:
- Estado da comunicação serial.

Bit	Significado quando ligado
0	alteração na linha clear-to-send
1	alteração em data-set-ready
2	borda de subida da portadora detectada
3	alteração na linha de recepção
4	clear-to-send
5	data-set-ready
6	chamada do telefone
7	sinal recebido

- Pode-se representar a informação em um byte de estado utilizando o seguinte campo de bits:
- Exemplo

struct status$_t ype\{unsigneddelta_cts$: 1;

unsigned delta_dsr : 1;

unsigned tr_edge : 1;

unsigned delta_rec : 1;

unsigned cts : 1;

unsigned dsr : 1;

unsigned ring : 1;

unsigned rec_line : 1;

} status;

- Para atribuir um valor a um campo de bit, simplesmente utiliza-se a forma para atribuição de outro tipo de elemento de estrutura.
- status.ring = 0;
- Não é necessário dar um nome a todo campo de bit. Isto torna fácil alcançar o bit que se deseja acessar, contornando os não usados. Por exemplo, se apenas cts e dtr importam, pode-se declarar a estrutura status_type desta forma:

```
struct status_type {
unsigned : 4;
unsigned cts : 1;
unsigned dsr : 1;
} status;
```

- Além disso, nota-se que os bits após dsr não precisam ser especificados se não são usados.
- Variáveis de campo de bit têm certas restrições:
  - Não pode obter o endereço de uma variável de campo de bit.
  - Variáveis de campo de bit não podem ser organizadas em matrizes.
  - Não pode ultrapassar os limites de um inteiro.
  - Não pode saber, de máquina para máquina, se os campos estarão dispostos da esquerda para a direita ou da direita para a esquerda.
  - Em outras palavras, qualquer código que use campos de bits pode ter algumas dependências da máquina.

- Finalmente, é válido misturar elementos normais de estrutura com elementos de campos de bit. O Exemplo abaixo define um registro de um empregado que usa apenas um byte para conter três informações: o estado do empregado, se o empregado é assalariado e o número de deduções. Sem o campo de bits, essa variável ocuparia três bytes.
- Exemplo

```
struct emp {
struct addr endereco;
float salario;
unsigned ativo : 1; /* ocioso ou ativo */
unsigned horas : 1; /* pagamento por horas */
unsigned deducao : 3; /* deduções de imposto */
};
```

Arquivos

- São conjuntos de dados armazenados em meio não volátil (normalmente disco). O acesso a um arquivo é sempre feito através de uma variável do tipo arquivo (FILE *).
- Ex: FILE *arqent, *arqvotos;
- Leitura e escrita em arquivos texto.
- O acesso a arquivos em C é feito através de variáveis do tipo "descritores de arquivo", que são structures que mantém a informação sobre arquivos em utilização. O acesso a um descritor de arquivo é feito através de um apontador para ele, declarado no programa através do tipo FILE * (stdlib.h)
- Ex: FILE *arqin,*arqout;

Índice

Tratamento de Arquivos

- Para utilizar um arquivo, inicialmente deve-se associar a variável arquivo ao arquivo em disco. Isso é feito com o comando FOPEN.

  arquivo=fopen(Nome do arquivo, modo de abertura)

- E os principais modos de abertura são:
  - "wt" - escrita texto
  - "rt" - leitura texto
- Ex.

arqent=fopen("c:\\notas","wt");

- Para gravar um dado em um arquivo texto, usa-se o comando fprintf, que funciona como um printf, apenas especificando como 1º parâmetro o arquivo onde será escrito o dado.
- Ex: fprintf(arquivo,"%d %d\n",a,b);
- Antes de terminar o programa, todos os arquivos devem ser fechados (fclose(arqout))

Exercícios:

1) Faça um programa que crie um arquivo numeros.dat na raiz do drive C ("C:\\"), com os números de 1 a 1000.

2) Faça um programa que leia 20 números do teclado e escreva os números pares no arquivo pares.dat e os ímpares no arquivo impares.dat, ambos na raiz do drive C.

3) Faça um programa que escreva os 100 primeiros números primos em um arquivo primos.dat.

Leitura de arquivos texto

- A leitura de um arquivo texto é feita atraves do comando
- fscanf(arquivo,especificador_de_tipo,lista_de_variaveis).
- Ex:

  fscanf(arqin,"%d",voto)

  A função fscanf retorna o número de elementos lidos. Em caso de fim de arquivo, ela retorna EOF. Assim, para ler um arquivo até o fim pode-se fazer:

  while(fscanf(arqin,"%d",&num)!=EOF)

  printf("%d\n",num);

- Outra função útil para a leitura de arquivos é a função feof(arquivo), que retorna verdadeiro se já foi encontrado o fim do arquivo.
- Uma forma de utilizá-lo é:

```
fscanf(arq,"%d",&num)

while (feof(arq)==FALSE)

{

fscanf(arq,"%d",num);

printf("%d\n",num);

}
```

- A leitura de um arquivo texto pode ser feita linha a linha, com o comando fgets (string, tamanho, arquivo). A função fgets retorna o apontador para o string lido (mesmo string que foi enviado como primeiro parâmetro), ou NULL se o arquivo chegou ao fim. Para ler um arquivo inteiro a função pode ser usada assim:

```
while (fgets(linha,200,arqin))
{
// processa a linha lida
}
```

1) Faça um programa que leia o arquivo resumo.txt contendo 1000 filmes no formato descrito na especificação do trabalho, e insira cada filme em uma lista ordenada de filmes, e cada ator em uma lista ordenada de atores. Liste ao final as duas listas.

Manipulação de Arquivos Binários

- Estrutura FILE: Para a manipulação de arquivos é utilizado a declaração de ponteiro (ponteiro de arquivo). Isto é, um ponteiro para informações que definem vários dados sobre o arquivo, como o seu nome, status, e a posição atual do arquivo. Um ponteiro de arquivo é uma variável ponteiro do tipo FILE .
- Todas as funções são realizadas utilizando o ponteiro. Para a declaração de um ponteiro de arquivo utiliza-se a seguinte sintaxe:

  FILE *<var>

- Exemplo

  FILE *fp;

Índice

ABERTURA DE ARQUIVOS

- A função fopen() abre arquivo e o associa ao descritor do arquivo. Ela retorna o ponteiro de arquivo associado a esse arquivo. Sua sintaxe é:

FILE *fopen(const char * <nome_arquivo>, const char * <modo_abertura>);

- O modo de abertura define a forma como é feito o acesso aos dados (somente leitura, leitura e escrita, etc). As formas principais são apresentadas na tabela a seguir.

Modo	Significado
rb	Abre um arquivo binário para leitura
wb	Cria um arquivo binário para escrita
ab	Anexa a um arquivo binário
r+b ou rb+	Abre um arquivo binário para leitura/escrita
w+b ou wb+	Cria um arquivo binário para leitura/escrita
a+b ou ab+	Anexa a um arquivo binário para leitura/escrita

- Exemplo

  FILE *arq; /* ponteiro de arquivo */

  arq = fopen("dados.dat","wb");

- Se ao abrir um arquivo para leitura o mesmo não existir a função fopen retorna um ponteiro nulo (NULL).

  arq = fopen("dados.dat","rb");

  if (arq == NULL)

      printf("Erro na abertura do arquivo");

Escrita de um Bloco de Dados

- Para se gravar um bloco de dados é utilizada a função fwrite(). Seu protótipo está definido a seguir:
- Sintaxe:

  size_t fwrite(void *buffer, size_t num_bytes, size_t count, FILE *fp);
- onde buffer é um ponteiro para uma região de memória que contém as informações que serão escritas no arquivo. O número de bytes de cada item a gravar é especificado por num_bytes. O argumento count determina quantos itens serão gravados. E, finalmente, fp é um ponteiro de arquivo para um arqyuvi aberto anteriormente.
- Esta função devolve o número de itens escritos. O número retornado pode ser menor que count quando o final de arquivo for atingido ou ocorrer um erro de gravação.

- Exemplo

```
int var_int;
FILE *arq;
arq = fopen("dados.dat","wb");
var_int = 5;
fwrite(&var_int,sizeof(var_int),1,arq);
```

- Faça uma função que gere um número randômico long long int, utilizando valores gerados pelo rand().
- use a função anterior em um programa que crie um arquivo "longos.dat" com 10.000 unsigned long long ints.
- A função rand() gera um int até 32767 (16 bits). Para gerar um unsigned long long int pode-se juntar vários inteiros de 15 bits.

Funções rand( ) e srand ( )

- Para gerar valores randômicos utiliza-se a função rand( ), da biblioteca stdlib.h.
- Cada chamada da função rand( ) retorna um valor inteiro entre 0 e RAND_MAX (constante definida em stdlib, normalmente igual a 32767, mas pode variar dependendo do sistema).
- Para inicializar o gerador de números randômicos no início do programa chama-se a função srand (time(NULL));

  Ex:

```
#include < stdio.h >

#include < stdlib.h >

#include < time.h >

int main ()

{

srand ( time(NULL) );

printf ("Um numero entre 0 e RAND_MAX (%d): %d\n", RAND_MAX, rand());

printf ("Um numero entre 0 e 99: %d\n", rand()%100);
```

```
unsigned long long int randao()
{
unsigned long long int valor=rand();
valor=valor*65536LL+rand();
valor=valor*65536LL+rand();
valor=valor*65536LL+rand();
return valor;
}
```

- O "LL" ao final do "65535" informa ao compilador que a expressão é long long int, evitando que ele a trunque para 32 bits (o fato de a variável valor ser long long int já deveria ser suficiente, mas...)
- Para não ficar escrevendo "unsigned long long int" uma dúzia de vezes, pode-se usar um typedef, como em:
  - typedef unsigned long long int uint64;

Leitura de um bloco de dados

- Para se ler um bloco de dados é utilizada a função fread(). Seu protótipo está definido a seguir:
- Sintaxe:

  size_t fread(void *buffer, size_t num_bytes, size_t count, FILE *fp);
- onde buffer é um ponteiro para uma região de memória que receberá os dados do arquivo. O número de bytes de cada item a ler é especificado por num_bytes. O argumento count determina quantos itens serão lidos. E, finalmente, fp é um ponteiro de arquivo para um arquivo aberto anteriormente.
- Esta função devolve o número de itens lidos. O número retornado pode ser menor que count quando o final de arquivo for atingido ou ocorrer um erro de leitura.

Fechamento de Arquivo

- A função fclose() fecha um arquivo que foi aberto através de uma chamada à fopen(). Esta função tem a seguinte sintaxe:
- Sintaxe:

  int fclose(FILE *fp);

- onde fp é o ponteiro de arquivo devolvido pela chamada à fopen(). O valor de retorno 0 significa uma operação de fechamento bem?sucedida.

- Exemplo

```
int var_int;
FILE *arq;
arq = fopen("dados.dat","rb");
fread(&var_int,sizeof(var_int),1,arq);
```

Acesso Aleatório : FSEEK()

- Além do acesso sequencial nas operações de arquivos, pode-se acessar diretamente uma posição no arquivo utilizando a função fseek(), que modifica o indicador de posição de arquivo. Seu protótipo é:

  int fseek (FILE *fp, long numbytes, int origem);

- Onde, numbytes, um inteiro longo, é o número de bytes a partir de origem, que se tornará a nova posição corrente, e origem é uma das seguintes macros definidas em STDIO.H.

Origem	Nome da Macro
Início do arquivo	SEEK_SET
Posição atual	SEEK_CUR
Final do arquivo	SEEK_END

- A função fseek() devolve 0 se a operação for bem-sucedida e um valor diferente de zero se ocorre um erro.

```
if (fseek(fp, atol(argv[2]), SEEK_SET)!=0)

    printf("Erro!!!");
```

- 1) Faça um programa que leia 10 valores inteiros do teclado e grave-os em um arquivo BINÁRIO dados.bin.
- 2) Faça um programa que leia o arquivo dados.bin, contendo valores inteiros, e escreva na tela os valores lidos.
- 3) Faça um programa que leia o arquivo dados.bin e troque, no mesmo arquivo, os valores pares por zero.
- 4) Utilize o programa do exercício 2 para verificar se a alteração no arquivo pedida no exercício 3 foi feita corretamente.

Função FTELL

- Esta função retorna a posição do ponteiro de um arquivo binário em relação ao seu começo.
- Esta função aceita um único argumento, que é o ponteiro para a estrutura FILE do arquivo.
- Seu protótipo é mostrado aqui:
- Sintaxe:

  long ftell (FILE *fp);

- Retorna um valor do tipo long, que representa o número de bytes do começo do arquivo até a posição atual.
- A função ftell() pode não retornar o número exato de bytes se for usada com arquivos em modo texto, devido à combinação CR/LF que é representada por um único caractere em C.

- Com o uso combinado das funções fseek e ftell pode-se descobrir o tamanho (em bytes) de um arquivo. Basta posicionar o indicador de posição do arquivo no último byte e chamar ftell para descobrir qual seu deslocamento em relação à posição zero:

```
fseek(arq,0L,SEEK_END);
long tamanho=ftell(arq);
```

- Para descobrir o número de itens (por exemplo, o número de inteiros se for um arquivo com inteiros) divide-se o tamanho do arquivo pelo tamanho de um item.
  - int numregs=tamanho/sizeof(int);

Rewind( )

- Esta função reposiciona o indicador de posição de arquivo no início do arquivo especificado como seu argumento. Seu protótipo é:
- void rewind(FILE *fp);

Condições de Erro

- Para determinar se um erro ocorreu utiliza-se a função ferror(), mas esta informação não basta para a solução por parte do usuário.
- Para isso utiliza-se a função perror() em conjunto com a função ferror(). O argumento de perror() é uma string fornecida pelo programa que normalmente é uma mensagem de erro que indica em que parte do programa ocorreu erro.
- Sintaxe:
- void perror (const char *str);
- Se for detectado um erro de disco, ferror() retornará um valor verdadeiro (não zero) e perror() imprimirá a seguinte mensagem
- Erro de Busca: Bad data
- A primeira parte da mensagem é fornecida pelo programa, e a segunda parte, pelo sistemaoperacional

- Medição de tempo em trechos de código pode ser feita utilizando a função time( ) da biblioteca time.h, que retorna o número de segundos transcorridos desde 1/1/1970.
- Pode ser utilizada da seguinte forma:

```
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
time_t t0,t1;
t0=time(NULL);
// trecho de código a ser medido
t1 = time(NULL);
printf("tempo:%d\n",t1-t0);
}
```

- 1) Faça um programa que ordene em ordem crescente os 10000 long long ints do arquivo "longos.dat". Considere que não há memória suficiente para manter mais do que 2000 registros em memória a cada vez.
- Alternativas:
  - a) Bubblesort (direto no disco)
  - b) Quicksort
  - c) Mergesort

Quicksort

Código do particionamento do quicksort

```
int separa( int v[], int p, int r)
    {
    int c = v[p], i = p+1, j = r, t;
    while (1) {
        while (i <= r && v[i] <= c) ++i;
        while (c < v[j]) - -j;
        if (i >= j) break;
        t = v[i], v[i] = v[j], v[j] = t;
        ++i; - -j;
        }
    v[p] = v[j], v[j] = c;
    return j;
```

Mergesort

- Faça um programa que leia dois vetores v[20] e w[20], ordene cada um em ordem crescente (coloque a ordenação como uma função) e gere um terceiro vetor z[40] com todos os valores de v e de w, também ordenado.
- Faça uma função que receba um vetor v[100] e 3 valores i1, i2 e f2, e efetue a intercalação ordenada do intervalo de i1 a i2-1, com o intervalo i2 a f2.

- Exercícios:

1. 1) Faça um programa que leia o arquivo arvore_binaria_16 (disponível no ucs virtual -> acervo da turma), com uma imagem BMP de 16 bits e a converta para tons de cinza. Use o mesmo cabeçalho do arquivo de origem, em que cada uma das 3 componentes de cada pixel é substituída pela média das cores. Desenvolva o código a partir do código gray_scale.cpp, disponível no acervo da turma.
2. 2) Faça um programa que leia o arquivo anterior gerando uma imagem com apenas a componente vermelha (zerando as componentes verde e azul).
3. 3) Idem, trocando a componente verde com a componente vermelha.
4. 4) Repita os exercícios anteriores com o arquivo enterprise.bmp (BMP 24)
5. 5) Faça um programa que leia o enterprise.bmp (BMP24 bits) e converta-o para um BMP16 bits.

- A imagem arvore_binaria_16 está em um padrão BMP565, em que as componentes R e B tem 5 bits e a componente G tem 6 bits. Para calcular a média deve-se considerar somente os 5 bits mais significativos da componente G.
- Supondo que se queira a média das componentes de um pixel px.

```
media=(px.r+px.b+px.g/2)/3;

px.r=media;

px.g=media*2; // volta a 6 bits

px.b=media;
```

- Utilizando o conjunto de caracteres ascii disponível em http://paulbourke.net/dataformats/asciiart/, altere o programa anterior para gerar um arquivo texto com a imagem do arquivo bmp lido.

Prova 2 - 2018/4

1) (2.5 pontos) Considere a declaração de struct a seguir, que contem as coordenadas X,Y de um ponto no plano cartesiano:

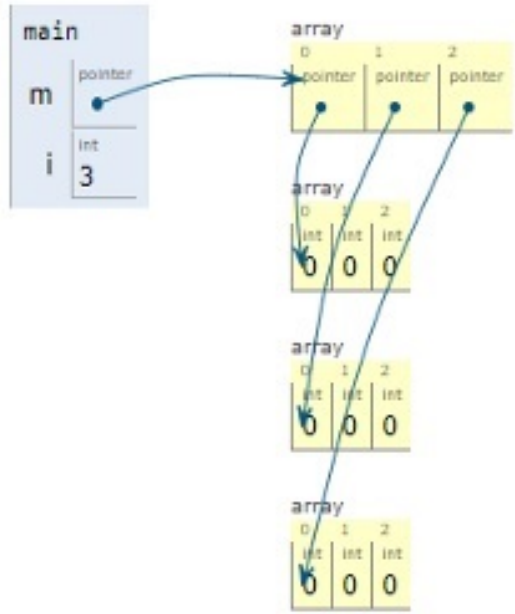
- typedef struct ponto {float X; float Y;} tponto;

A distância de um ponto à origem é dada por $\sqrt{X^2 + Y^2}$.
Faça uma função que receba um vetor de 100 pontos e retorne nos parâmetros MaisPróximo e MaisLonge as coordenadas dos pontos mais próximo e mais afastado da origem.Use o protótipo a seguir:

void maisproximo(tponto listap[100], tponto *MaisProximo, tponto *MaisLonge);

```
void maisproximo(tponto listap[100], tponto *MaisProximo, tponto *MaisLonge )

{

int i,posProx=0,posLonge=0;

float DistLonge,DistProx;

DistLonge=DistProx=listap[0].X*listap[0].X+listap[0].Y*listap[0].Y;

for (i=1;i<100;i++)

{

float dist=listap[i].X*listap[i].X+listap[i].Y*listap[i].Y;

if (dist>DistLonge)DistLonge=dist;posLonge=i;

if (dist<DistProx)DistProx=dist;posProx=i;

}

*MaisProximo=listap[posProx];

*MaisLonge=listap[posLonge];

}
```

2) (2.5 pontos) Faça uma função AlocaMat que receba dois valores N e M, representando o número de linhas e colunas de uma matriz de inteiros, aloque a matriz dinamicamente e zere todas as posições, retornando, como valor de retorno, o pointer para a matriz alocada.



```
int **alocaMat(int N, int M)
{
int i,j;
int **Mat=(int **)malloc(N*sizeof(int *));
for (i=0;i<N;i++)
Mat[i]=(int *)malloc(M*sizeof(int));
for (i=0;i<N;i++)
for (j=0;j<M;j++)
Mat[i][j]=0;
return Mat;
}
```

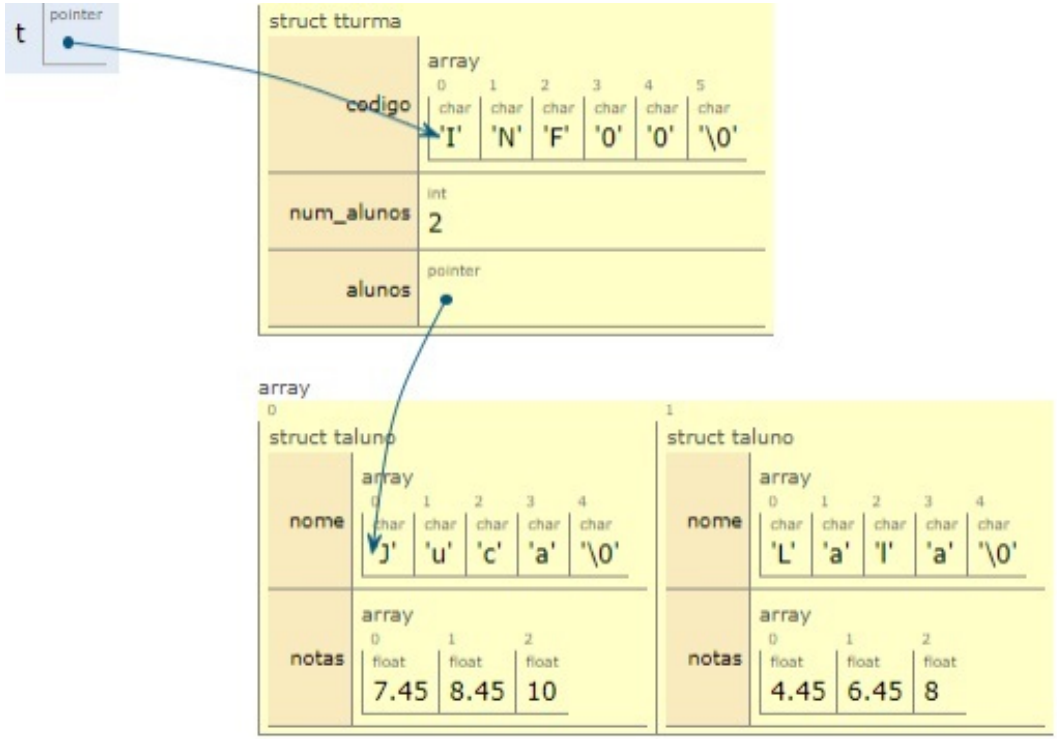
3) (2.5 pontos) Considere as declarações de struct a seguir:

typedef struct aluno {char nome[30]; float notas[3];} taluno;

typedef struct turma {char codigo[6]; int num_alunos; taluno alunos[];} tturma;

Faça uma função le_turma que recebe como parâmetro a quantidade de alunos de uma turma, aloca a struct da turma e do vetor de alunos, e lê os dados para a turma, retornando o pointer para a struct criada. Use o protótipo a seguir:

tturma * le_turma (int Quant);


t
pointer
struct tturma
codigo
array
0 1 2 3 4 5
char char char char char char
'I' 'N' 'F' '0' '0' '\0'
num_alunos
int
2
alunos
pointer
array
0
struct taluno
nome
array
0 1 2 3 4
char char char char char
'J' 'u' 'c' 'a' '\0'
notas
array
0 1 2
float float float
7.45 8.45 10
1
struct taluno
nome
array
0 1 2 3 4
char char char char char
'L' 'a' 'l' 'a' '\0'
notas
array
0 1 2
float float float
4.45 6.45 8

```
typedef struct aluno {char nome[30]; float notas[3];} taluno;

typedef struct turma {char codigo[6]; int num_alunos; taluno *alunos;} tturma;

tturma * le_turma (int Quant)

    {

    tturma *taux=(tturma*)malloc(sizeof(tturma));

    scanf("%s",taux->codigo);

    taux->num_alunos=Quant;

    taux->alunos=(taluno*)malloc(Quant*sizeof(taluno));

    int i;

    for (i=0;i<Quant;i++)

        scanf("%s%f%f%f",taux->alunos[i].nome,

    taux->alunos[i].notas[0],

    taux->alunos[i].notas[1],

    taux->alunos[i].notas[2]);

    return taux;

    }
```

4) (2.5 pontos) Faça uma função que receba um vetor de 10 strings de até 30 caracteres e retorna a lista de strings ordenada em ordem alfabética. Use o protótipo a seguir:
void ordena(char nomes[10][31])

```
void ordena(char nomes[5][31])
{
int i,j;
for (i=0;i<4;i++)
      for (j=0;j<4;j++)
            if (strcmp(nomes[j],nomes[j+1])>0)
                  {
                  char staux[31];
                  strcpy(staux,nomes[j]);
                  strcpy(nomes[j],nomes[j+1]);
                  strcpy(nomes[j+1],staux);
                  }
}
```