Objetivos	a. Demonstrar o funcionamento de diversas ED sob os aspectos: ocupação de memória, armazenamento e recuperação de conteúdo, operações principais b. Utilizar ponteiros e procedimentos da linguagem C para alocar dinamicamente porções de memória
Referências gerais	SCHILDT, H. *C Completo e Total*. São Paulo: Pearson, 1997. TENENBAUM, A.M.; LANGSAM, Y.; AUGENSTEIN, M.J. *Estruturas de Dados usando C*. São Paulo: Pearson Makron Books, 1995.

Objetivo específico desta aula prática: entender e utilizar a alocação dinâmica de memória (ling. C), comparando-a com a alocação estática ("fixa").

Suporte teórico 1. Funções e operações da linguagem C que permitem alocar memória dinamicamente (isto é, durante a execução do programa) e utilizar essa memória por meio de um ponteiro.

Função / operador	Descrição	Exemplo	Obs.

`malloc`	Solicita ao sistema operacional a alocação de bytes da memória "livre" (chamada de *heap*). Em caso de sucesso, o sistema operacional retorna o endereço inicial dessa área (o qual deve ser armazenado em um ponteiro).	`#include<stdlib.h>`
		`...`
		`int * p;`
		`...`
		`p = (int *) malloc ( 10 * sizeof(int) );`
		`if ( p != NULL ) {`
		`  ...`
		`  free ( p );`
		`}`
			No exemplo, é alocada memória suficiente para armazenar 10 números inteiros. A memória é adjacente; assim, só é preciso saber seu endereço inicial (armazenado em **p**).

`*`	Dá acesso ao conteúdo da área apontada pelo seu operando	`// notação ponteiro-deslocamento`
		`*(p+0) = 5;`
		`*(p+1) = 9;`
		
		`// comandos equivalem a...`
		`p[0] = 5;`
		`p[1] = 9;`
			Preenche, na memória *heap* alocada, as áreas correspondentes ao primeiro e segundo números inteiros.

`[ ]`	Dá acesso ao conteúdo da área apontada pelo seu operando	`// notação ponteiro-índice`
		`printf( "%d\n", p[0] );`
		`printf( "%d\n", p[1] );`
			Seguindo o exemplo, exibirá os valores **5** e **9**.

Atividade 1. Crie um programa em linguagem C no qual você:

a) declara dois ponteiros, **i_ptr** e **d_ptr** (um para "int" e outro para "double");
b) aloca dinamicamente memória para duas áreas (uma contendo elementos "int" e outra contendo elementos "double", apontadas respectivamente pelos ponteiros **i_ptr** e **d_ptr**), sendo os tamanhos dessas áreas (**i_qtd** e **d_qtd**) definidos pelo usuário durante a execução;
c) preenche, com valores aleatórios, todas as áreas alocadas;
d) exibe endereços e conteúdos de todas as variáveis que estão na memória "do programa" (memória *stack*);
e) exibe endereços e conteúdos dos elementos alocados dinamicamente (memória *heap*).

Atividade 2. Com base nos resultados impressos na Atividade 1, elabore um desenho que represente a ocupação de cada porção da memória (*stack* e *heap*) pelas variáveis do seu programa. Inclua os 'prints' de sua execução.

Atividade 3. Crie um programa em linguagem C no qual você declara um ponteiro para o tipo "struct" assim definido: `typedef struct { int codigo; char status; double ticket; float * ultimas_compras } CLIENTE;` . Em seguida, o programa aloca dinamicamente memória para uma área com dois `CLIENTEs` e, em cada uma delas, área para armazenar seis valores de `ultimas_compras`. Em seguida, o programa preenche os elementos com valores quaisquer. Por fim, exibe:

a) endereço e conteúdo do ponteiro que está na memória "do programa" (memória *stack*);
b) endereços e conteúdos de todos os elementos alocados dinamicamente (memória *heap*).

Atividade 4. Com base nos resultados impressos na Atividade 3, elabore um desenho que represente a ocupação de cada porção da memória (*stack* e *heap*) pelas variáveis do seu programa. Inclua os 'prints' de sua execução.