## Ponteiros e alocação dinâmica de memória

1) Faça um programa que leia um valor n e crie dinamicamente um vetor de n elementos e passe esse vetor para uma função que vai ler os elementos desse vetor. Depois, no programa principal, o vetor preenchido deve ser impresso. Além disso, antes de finalizar o programa, deve-se liberar a área de memória alocada.
2) Faça uma função que receba um valor n e crie dinamicamente um vetor de n elementos e retorne um ponteiro. Crie uma função que receba um ponteiro para um vetor e um valor n e imprima os n elementos desse vetor. Construa também uma função que receba um ponteiro para um vetor e libere esta área de memória. Ao final, crie uma função principal que leia um valor n e chame a função criada acima. Depois, a função principal deve ler os n elementos desse vetor. Então, a função principal deve chamar a função de impressão dos n elementos do vetor criado e, finalmente, liberar a memória alocada através da função criada para liberação.
3) Construa um programa (main) que aloque em tempo de execução (dinamicamente) uma matriz de ordem m x n (linha por coluna), usando 1+m chamadas a função malloc. Agora, aproveite este programa para construir uma função que recebendo os parametros m e n aloque uma matriz de ordem m x n e retorne um ponteiro para esta matriz alocada. Crie ainda uma função para liberar a área de memória alocada pela matriz. Finalmente, crie um novo programa (main) que teste/use as duas funções criadas acima.
4) Construa um programa (main) que aloque em tempo de execução (dinamicamente) uma matriz de ordem m x n (linha por coluna), usando 2 chamadas a função malloc. Agora, aproveite este programa para construir uma função que recebendo os parametros m e n aloque uma matriz de ordem m x n e retorne um ponteiro para esta matriz alocada. Crie ainda uma função para liberar a área de memória alocada pela matriz. Finalmente, crie um novo programa (main) que teste/use as duas funções criadas acima.
5) Criar um tipo abstrato de dados que represente uma pessoa, contendo nome, data de nascimento e CPF. Crie uma variável que é um ponteiro para este TAD (no programa principal). Depois crie uma função que receba este ponteiro e preencha os dados da estrutura e também uma uma função que receba este ponteiro e imprima os dados da estrutura. Finalmente, faça a chamada a esta função na função principal.
6) Idem a questão acima, mas construa as funções usando referências (&).
7) Crie um programa para manipular vetores. O seu programa deve implementar uma função que receba um vetor de inteiros V e retorne um outro vetor de inteiros alocado dinamicamente com todos os valores de V que estejam entre o valor mínimo e máximo (que também são passados como parâmetro para a função). A função deve obedecer ao seguinte protótipo: int* valores_entre(int *v, int n, int min, int max, int *qtd); A função recebe: • v: vetor de números inteiros; • n: a quantidade de elementos do vetor v; • min: valor mínimo a ser buscado; • max: valor máximo a ser buscado; A função deve: • Verificar a quantidade de elementos do vetor que sejam maiores do que min e menores que max; • Caso a quantidade seja maior do que 0 (zero), alocar dinamicamente uma área do exato tamanho necessário para armazenar os valores; • Copia os elementos do vetor que sejam maiores do que min e menores que max para a área alocada dinamicamente. A função retorna: • O endereço da área alocada dinamicamente, preenchida com os números maiores do que min e menores que max, ou NULL, caso essa relação de números não tenha sido criada; • A quantidade de

números carregados na área alocada dinamicamente, através do parâmetro qtd. Em seguida, crie a função principal do programa para inicializar um vetor de inteiros, exibir esses valores na tela e pedir para o usuário digitar o valor mínimo e máximo a ser buscado. Em seguida o programa deverá chamar a função valores_entre e exibir na tela os valores resultantes. Lembre-se de exibir uma mensagem de erro caso nenhum valor seja encontrado. Não se esqueça de liberar a memória alocada dinamicamente.

8) Crie uma função que receba como parâmetros dois vetores de inteiros, v1 e v2, e as suas respectivas quantidades de elementos, n1 e n2. A função deverá retornar um ponteiro para um terceiro vetor, v3, com capacidade para (n1 + n2) elementos, alocado dinamicamente, contendo a união de v1 e v2. Por exemplo, se v1 = {11, 13, 45, 7} e v2 = {24, 4, 16, 81, 10, 12}, v3 irá conter {11, 13, 45, 7, 24, 4, 16, 81, 10, 12}. O cabeçalho dessa função deverá ser o seguinte: int* uniao(int *v1, int n1, int *v2, int n2); Em seguida, crie a função principal do programa para chamar a função uniao passando dois vetores informados pelo usuário (ou declarados estaticamente). Em seguida, o programa deve exibir na tela os elementos do vetor resultante. Não esqueça de liberar a memória alocada dinamicamente.
9) Crie um programa que implemente o jogo “Bingo de Prog II”. Nesse jogo, o jogador deve selecionar a quantidade de números que ele gostaria de apostar (entre 1 e 20), e em seguida, informar os números escolhidos (valores entre 0 e 100). Após receber a aposta, o computador sorteia 20 números (entre 0 e 100) e compara os números sorteados com os números apostados, informando ao apostador a quantidade de acertos e os números que ele acertou. O seu programa deverá implementar as funções ler_aposta, sorteia_valores e compara_aposta. A função ler_aposta deve receber como parâmetro a quantidade de números que serão apostados e um vetor previamente alocado dinamicamente para armazenar a quantidade exata de números apostados. A função deve pedir para o usuário digitar os números apostados e armazena-los no vetor, garantindo que somente números dentro do intervalo de 0 a 100 sejam digitados. A função deve seguir o seguinte protótipo: void ler_aposta(int *aposta, int n); A função sorteia_valores deve receber como parâmetro a quantidade de números que serão sorteados e um vetor previamente alocado dinamicamente para armazenar a quantidade exata de números sorteados. A função deve sortear aleatoriamente os números (entre 0 e 100) e armazena-los no vetor. A função deve seguir o seguinte protótipo: void sorteia_valores(int *sorteio, int n); A função compara_aposta deve receber como parâmetro o vetor com os números apostados (aposta), o vetor com os números sorteados (sorteio), juntamente com os seus respectivos tamanhos (na e ns) e um ponteiro para uma variável inteira (qtdacertos), onde deve ser armazenada a quantidade de acertos. A função deve retornar o ponteiro para um vetor alocado dinamicamente contendo os números que o apostador acertou. A função deve seguir o seguinte protótipo: int* compara_aposta(int *aposta, int *sorteio, int *qtdacertos, int na, int ns); Em seguida, crie a função principal do programa utilizado as funções criadas anteriormente para implementar o jogo “Bingo de Prog II”. Lembre-se que os vetores aposta, sorteio e acertos devem ser alocados dinamicamente e a memória alocada deve liberada quando ela não for mais ser utilizada. Para sortear números aleatórios utilize a função rand da biblioteca stdlib.h. A função rand retorna um número aleatório em um determinado intervalo. Exemplo: x = rand() % 10; /* x vai receber um valor entre 0 e 10 */ Para garantir que novos números aleatórios sejam sorteados em cada execução do programa é necessário executar a função srand antes de sortear os números.

Exemplo: srand(time(NULL)); Para poder utilizar essas funções é necessário incluir no programa as bibliotecas stdlib.h e time.h. Exemplo de programa para sortear um número aleatório: #include #include #include int main(void) { int x; srand(time(NULL)); x = rand() % 10; /* x vai receber um valor entre 0 e 10 */ printf("%d", x); return 0; }

Fontes:

http://www.decom.ufop.br

http://edirlei.3dgb.com.br