**Exercício.1. Programa dos alunos faltosos**

Escrever um programa que cadastre o nome, a matrícula, as faltas e as 4 notas dos alunos de uma turma, cuja quantidade de alunos é informada pelo usuário. O programa deve mostrar os dados dos alunos com número de faltas acima do número médio de faltas da turma.

--> Defina o tipo ALUNO:

--> Faça uma função que retorne um aluno preenchido com dados do teclado

--> Faça uma função que retorne um vetor de alunos, criado dinamicamente e preenchido com dados do teclado usando a função que preenche um aluno). O número de alunos é recebido pela função que também recebe o endereço de uma variável para preencher com o número médio de faltas da turma. A função retorna o endereço da 1ª pos do vetor criado para armazenar a turma

--> Faça uma função que receba o endereço que armazena um aluno exibindo seus dados na tela

--> Faça uma função que receba os dados da turma e a média de faltas, exibindo os dados dos alunos faltosos

}

--> Faça a main para atender o enunciado

**Exercício 2**. Uma instituição de pesquisa recolheu amostras de estados brasileiros a respeito do salário da população. Tendo o seguinte código:

struct assalariados {

char nome[81];

char sexo;

int idade;

float salario;

char estado[3];

};

typedef struct assalariados Assalariados;

Assalariados cadastra(char \*nome, char sexo, int idade, float salario, char \*estado);

void relatorio(Assalariados vAssalariados[], int numAssalariados);

void imprime(Assalariados vAssalariados[], int numAssalariados);

int main(void)

{

char \* estados[27] = {"AC","AL","AM","AP","BA","CE","DF","ES","GO","MA", "MG","MS","MT","PA","PB","PE","PI","PR","RJ","RN","RO","RR","RS","SC","SE", "SP","TO"};

Assalariados \*pessoas;

int numAssalariados=9;

pessoas = (Assalariados \*) malloc (numAssalariados \* sizeof(Assalariados));

pessoas[0] = cadastra("Fulano0 de tal", 'M',45,1500.00, "RJ");

pessoas[1] = cadastra("Ciclano0", 'M',50,2500.00, "RS");

pessoas[2] = cadastra("Beltrano0", 'M',42,500.00, "RS");

pessoas[3] = cadastra("Fulano1 de tal", 'F',45,1500.00, "RJ");

pessoas[4] = cadastra("Ciclano1", 'M',30,2500.00, "RS");

pessoas[5] = cadastra("Beltrano1", 'M',42,500.00, "RS");

pessoas[6] = cadastra("Fulano2 de tal", 'M',45,1500.00, "AC");

pessoas[7] = cadastra("Ciclano2", 'F',30,2500.00, "RS");

pessoas[8] = cadastra("Beltrano2", 'M',42,500.00, "RS");

imprime(pessoas, numAssalariados);

relatorio(pessoas,numAssalariados, estados, 27);

}

a. Implemente a função cadastra, que atribui os parâmetros recebidos aos campos da estrutura Assalariados.

b. Implemente a função imprime, que imprime todos os elementos do vetor de Assalariados.

c. Implemente a função contabilizaEstado, que recebe a sigla de um estado, o vetor de assalariados e o número de elementos deste vetor, contabilizando a quantidade de homens por estado com mais de 40 anos que recebem salários superiores a R$ 1000,00.

d. Implemente a função relatório, que imprime a quantidade de homens por estado com mais de 40 anos que recebem salários superiores a R$ 1000,00. Caso a quantidade no estado analisado seja zero, então não imprime nada. A mensagem de impressão deve ser:

Estado=%s com %d homens maiores de 40 anos e salário maior que R$ 1000,00

**Exercício 4.** Uma empresa registra as seguintes informações sobre os produtos que comercializa: código (int), descrição ( string 40) classificação (“alimento”, “têxtil”), tipo (“importado”, “nacional”), valor (em reais para as nacionais e em dólar para as importadas) e a nota (valor inteiro de 1 a 10). Para sua campanha de Natal, deseja separar os produtos mais desejados (os que tiveram a maior nota) nacionais dos importados em duas tabelas distintas, criadas dinamicamente, com tamanho exato necessário para armazenar os produtos. Faça um programa em C, modularizado, que atenda esta empresa sabendo que a empresa comercializa 10 produtos e que no final as tabelas geradas devem ser exibidas.

1. Defina uma estrutura em C com o nome de *pdt* que tenha os campos apropriados para guardar todas as informações de um produto. Defina também um novo tipo de dados com o nome de *TPRODUTO*, correspondendo a essa estrutura.

struct pdt{

int cod;

char descrição[41], classificacao[9], tipo[10];

float valor;

int nota;

}**;**

typedef struct pdt TPRODUTO;

1. Utilizando a main abaixo, teste as funções marcadas como implementar

#define MAX 10

int main (void)

{

TPRODUTO vet\_Produtos[MAX] = { {1,"casaco", "textil", "importado",50, 9},

{2,"casaco", "textil", "nacional", 130,10},

{3,"damasco", "alimento", "importado",13, 10},

{4,"banana passa", "alimento", "nacional",13, 10},

{5,"sapato", "textil", "importado",180, 4},

{6,"sapato", "textil", "nacional",180, 8},

{7,"bolsa", "textil", "importado", 230,8},

{8,"bolsa", "textil", "nacional",330, 7},

{9,"vinho", "alimento", "importado",160, 10},

{10,"vinho", "alimento", "nacional",120, 9}};

//Continue a implementação do programa! return 0;

}

d) Faça a função maiores\_valores() que recebe a tabela de produtos e o endereço de 4 variáveis: maior\_nac, qtos\_nac, maior\_imp e quantos\_imp. Deve percorrer a tabela, encontrando a maior nota nacional, a maior nota importada, quantos produtos nacionais possuem a maior nota bem como quantos produtos importados possuem a maior nota.

void compara (int \* max, int val, int \* qt)

{

if ( \*max < val)

{

\*max = val;

\*qt = 1;

}

else

if ( \*max == val)

(\*qt)++;

}

void maiores\_valores(TPRODUTO tab[],int n, int\*maxN,int \* qtN, int \* maxI, int \* qtI)

{

int i;

(\*maxN)= (\*maxI)=-1;

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

{

if (strcmp(tab[i].tipo,"importado")==0)

compara(maxI,tab[i].nota,qtI);

else

compara(maxN,tab[i].nota,qtN);

}

return ;

}

**IMPLEMENTAR:**e)Escreva a função cria\_vetor()que recebe a quantidade de elementos de um vetor que será dinamicamente alocado por esta função. A função aloca um vetor com qtd elementos do tipo TPRODUTO e retorna o endereço do primeiro elemento deste vetor ou interrompe o programa se não puder criá-lo.

TPRODUTO\*cria\_vetor( int n)

{

TPRODUTO \* v=(TPRODUTO\*)malloc(n\*sizeof(TPRODUTO));

If ( v== NULL)

{

printf( “Erro”);

exit(1);

}

return v;

}

**IMPLEMENTAR:**f) Escreva a função separaBonsemTabelas(), que recebe a tabela de produtos preenchida e as tabelas de Nacionais e Importados vazias, a maior nota nacional e maior nota importada, incluindo, nas respectivas tabelas, os produtos com estas notas. Quando copiados para a nova tabela, os produtos importados, devem ter seu valor modificado pela função *modifica\_valor()* descrita abaixo

**IMPLEMENTAR:**g) Escreva a função modifica\_valor()que recebe o ENDEREÇO de um produto importado e modifica seu valor do seguinte modo:

--> para produtos texteis: acrescentar 10% do valor atual

--> para produtos alimentícios :diminuir 10% do valor atual

**IMPLEMENTAR:i**) Escreva a função exibeProdutosNaFaixadePreço que recebe um vetor de Produtos, limite inferior e limite superior de preço e exibe o código, descrição e tipo de todos os produtos que estão nesta faixa de preço

**IMPLEMENTAR:**j) Escreva a versão recursiva da função exibeProdutosNaFaixadePreço

**Exercício 5.** Considere o tipo Candidato abaixo:

struct medidas

{

float altura;

float peso;

};

typedef struct medidas Medidas;

struct candidato

{

int inscricao;

char nome[51];

int idade;

Medidas med;

};

typedef struct candidato Candidato;

Você irá escrever um programa que crie e manipule um vetor de candidatos.

Escreva as funções:

🡪 exibeCandidato: recebe o endereço de um candidato e exibe seus dados na tela

🡪 exibeTodosCandidatos: a função recebe um vetor de candidatos e o número de candidatos, exibindo os dados e o IMC ( peso/altura2) de todos os candidatos; Deve utilizar a função anterior

🡪 buscaInscricao: a função recebe um vetor de candidatos, o número de candidatos e uma inscrição. A função retorna o endereço do candidato, se ele for encontrado, ou NULL, caso contrário.

🡪 mostraIMCBons: a função que recebe um vetor de candidatos, o número de candidatos e exibe todos os candidatos com IMC normal ( entre 18.5 e 24.9)

Para testar as funções utilize o vcand abaixo:

Candidato vcand[8] = {

{111,"bia",15,{1.60,60.0}},

{222,"edu",17,{ 1.80,86.0 }},

{333,"leo",14,{ 1.75,91.0 }},

{666,"rui",13,{ 1.90,70.0 }},

{555,"vik",18,{ 1.50,50.0 }},

{888,"ana",16,{ 1.60,55.6 }},

{444,"teo",16,{ 1.70,70.0 }},

{777,"lia",15',{ 1.65,55.4 }}} ;

**Exercício 6.** O departamento de transportes do Rio de Janeiro mantém um registro de todos os carros registrados. O registro de cada carro possui as seguintes informações:

* + Placa – cadeia de caracteres identificando a placa do veículo (máximo 8 caracteres);
  + Dono – cadeia de caracteres identificando o nome do proprietário do veículo (máximo 40 caracteres);
  + Data – Data de fabricação do veículo (contendo mês e ano);
  + Multas – Número de multas associadas ao veículo;

Para criar um programa para acessar os dados do sistema do departamento de transportes você precisa implementar dois tipos estruturados: Carro e Data.

|  |  |  |
| --- | --- | --- |
| **Data** |  | **Carro** |
| Mês | Placa |
| Ano | Dono |
|  | | DataFab |
| Multas |

Escreva um programa que defina os novos tipos estruturados Carro e Data, como especificados acima. E em seguida, implemente as seguintes funções para a manipulação dos cadastros de veículos:

* + exibeCadastro – a função recebe como parâmetro uma estrutura do tipo Carro e imprime na tela todas as informações sobre o carro no formato exemplificado abaixo:

ABC1234 (25/03) - MARIO – 5 multas

* + exibeMaisMultados – a função que recebe um vetor de Carros e exibe na tela somente os carros que tiverem mais de 5 multas. A função deve utilizar a função exibeCadastro criada no item anterior.
  + exibeMaisAntigo – a função recebe um vetor de Carros e exibe na tela somente o registro do carro mais antigo existente. A função deve utilizar a função exibeCadastro criada no item anterior.
  + geraAdvertenciaMultas – a função recebe como parâmetro uma estrutura do tipo Carro e retorna uma cadeia de caracteres alocada dinamicamente com o tamanho exato necessário para armazenar uma mensagem de advertência para o dono do veículo no seguinte formato:

“MARIO, seu veículo possui muitas multas!”

onde MARIO deve ser substituído pelo nome do dono do veículo. Você deve utilizar a função strcat da biblioteca string.h para concatenar as cadeias de caracteres.

* geraAdvertenciaData – a função recebe como parâmetro uma estrutura do tipo Carro e retorna uma cadeia de caracteres alocada dinamicamente com o tamanho exato necessário para armazenar uma mensagem de advertência para o dono do veículo no seguinte formato:

“MARIO, seu veículo é muito antigo!”

Onde MARIO deve ser substituído pelo nome do dono do veículo. Você deve utilizar a função strcat da biblioteca string.h para concatenar as cadeias de caracteres.

* exibeAdvertencias – a função recebe um vetor de Carros e a quantidade de elementos. Exibe na tela mensagens de advertência para os carros que possuírem mais de 5 multas e também para os carros que tiverem mais de 20 anos e meio. A função deve utilizar as funções geraAdvertenciaMultas e geraAdvertenciaData criadas nos itens anteriores para gerar as mensagens de advertência.
* separaZerados – a função recebe um vetor de Carros e o endereço de uma variável. Conta quantos motoristas não possuem multas, armazenando na variável cujo endereço foi recebido, cria um vetor dinamicamente para armazená-los, copia os motoristas sem multas para o vetor criado e retorna o vetor criado.
* exibeZerados – a função que recebe um vetor de Carros e a quantidade de elementos, e exibe-os na tela. A função deve utilizar a função exibeCadastro .

Em seguida, implemente a função principal do programa utilizando como base a main definida abaixo. O seu programa deverá primeiramente exibir os dados dos carros mais multados e também os dados do carro antigo utilizando as funções exibeMaisMultados e exibeMaisAntigo. Em seguida, utilizando a função exibeAdvertencias, o programa deverá exibir as advertências para os motoristas com mais multas e que possuem carros muito antigos. Finalmente, utilizando a função separa Zerados, copia os motoristas zerados e exibe-os utilizando a função exibe\_Zerados.

int main (void)

{

Carro vet\_carros[10] = {{"NVY4822", "Ana Silva", {5, 2010}, 2},

{"LAB5849", "Pedro Duarte", {9, 2004}, 8},

{"HMF4821", "Joao Filho", {8, 2000}, 3},

{"POD1842", "Maria Gomes", {2, 1994}, 5},

{"VFJ3284", "Silvio Lins", {10, 1998}, 6},

{"WED7452", "Marcia Morais", {1, 1991}, 2},

{"HEL1475", "Bruno Rodrigues", {12, 2005}, 8},

{"IKF3685", "Thais Silva", {3, 1990}, 4},

{"WPF4528", "Ricardo Costa", {7, 2009}, 10},

{"FNY1579", "Julia Neves", {4, 1997}, 7}};

//Continue a implementação do programa! return 0;

}

**Exercício 7.** A avaliação de um vendedor de uma loja é mapeada no tipo estruturado Avaliacao abaixo:

struct avaliacao

{ float valorVendidoNoMes;

int numeroDeFaltas;

};

typedef struct avaliacao Avaliacao;

Os dados de um vendedor são representados pelo tipo estruturado Vendedor abaixo:

struct vendedor

{ int inscricao;

char nome[51];

float salário-base;

Avaliacao aval;

};

typedef struct vendedor Vendedor;

1. Escreva em C a função **calculaExtra** que recebe uma avaliação e retorna um valor (float) correspondente a um valor extra a ser acrescido ao salário do vendedor no pagamento do mês. Esse valor extra deve ser computado da seguinte forma:

* Se o número de faltas é menor do que 2 e o valor vendido no mês é maior do que 2000, a função retorna 180.
* Se o número de faltas é maior ou igual a do 2 e menor do que 4 e o valor vendido é maior do que 2000, a função retorna 150.
* Se o número de faltas é maior ou igual a 4 e menor do que 6, mas o valor vendido é maior do que 5000, a função retorna 120.
* Do contrário a função retorna 0.

b) Para a folha de pagamento da loja é utilizado um vetor de vendedores. Escreva em C a função **calculaValorTotalDaFolha** que recebe o número de vendedores e o vetor dos vendedores da loja e calcula qual o valor total que a loja deverá disponibilizar para pagamentos naquele mês, que será o resultado da soma dos valores a serem pagos a todos os vendedores (valor para cada vendedor : salario + extra). A função deve utilizar a função calculaExtra do item anterior.

d) Escreva em C a função **obtemRelacaoDemitidos**, que recebe o número de vendedores e o vetor dos vendedores da loja e retorna um novo vetor de inteiros, alocado com o tamanho exato necessário, com as inscrições dos vendedores com número de faltas >= 10, que serão demitidos. A função deve também devolver em uma variável inteira, cujo endereço foi fornecido com argumento na chamada da função, o número de vendedores que serão demitidos. Caso não seja possível criar o novo vetor ou caso não existam vendedores com mais de 10 faltas, a função deve retornar NULL.

**MODIFICAÇÃO: Criar o tipo VENDEDORDEMITIDO, com o nº de inscrição e a Avaliação do vendedor. Esta função deve retornar um vetor de VENDEDORDEMITIDO**

e) Assumindo, para facilitar, que os valores vendidos no mês são todos diferentes, escreva a função **obtemMelhorVendedor**, que recebe o número de vendedores e o vetor dos vendedores da loja e retorna, sem alocar dinamicamente, nem usar qualquer outro vetor de char, um ponteiro para a cadeia contendo o nome do melhor vendedor da loja (o que tem o maior valor vendido no mês).

Em seguida, implemente a função principal do programa utilizando como base a main definida abaixo.

int main (void)

{

Vendedor vVendedor[10]= {{1, "Ana Silva", 1000,{3000,1}},

{2, "Pedro Duarte",2000,{3010,3}},

{3, "Joao Filho", 1000,{3020,4}},

{4, "Maria Gomes", 3000,{3030,15}},

{5, "Silvio Lins", 1000,{6000,15}},

{6, "Marcia Morais",2000,{6010,3}},

{7, "Bruno Rodrigues",3000,{6020,4}},

{8, "Thais Silva", 4000,{6030,7}},

{9, "Ricardo Costa", 5000,{3040,7}},

{10, "Julia Neves", 10000,{1000,0}}};

//Continue a implementação do programa! return 0;

}