INF1007 - PROGRAMAÇÃO II

LISTA DE EXERCÍCIOS 4

1. Considere uma aplicação que utiliza pontos, retângulos e círculos para representações gráficas. Um ponto é composto pelas suas coordenadas x e y, um retângulo é composto por dois pontos e um círculo é definido pelo seu ponto central e raio:

Ponto
X
Y

Retângulo
Ponto1
Ponto2

Círculo
PontoCentro
Raio

Escreva um programa que defina os novos tipos estruturados `Ponto`, `Retangulo` e `Circulo`, como especificados acima. E em seguida, implemente as seguintes funções para a manipulação de objetos geométricos definidos por pontos, retângulos e círculos:

- `criaPonto` – a função recebe como parâmetro as coordenadas x e y e retorna a estrutura de um ponto criado com base nos parâmetros;

- `criaRetangulo` – a função recebe como parâmetro dois pontos e deve alocar dinamicamente a estrutura de um retângulo definido pelos pontos recebidos por parâmetro. A função deve retorna o endereço de memória para a estrutura alocada dinamicamente;

- `criaCirculo` – a função recebe como parâmetro o ponto central do círculo e o seu raio, e deve alocar dinamicamente a estrutura de um círculo definido pelos parâmetros. A função deve retorna o endereço de memória para a estrutura alocada dinamicamente;

- `areaRetangulo` – a função recebe como parâmetro o endereço de memória da estrutura de um retângulo e retorna a área deste retângulo;

- `areaCirculo` – a função recebe como parâmetro o endereço de memória da estrutura de um círculo e retorna a área deste círculo;

- `distancia` – a função recebe como parâmetro dois pontos e retorna distancia entre estes pontos;

- `pertence` – a função recebe como parâmetro um ponto e o endereço de memória da estrutura de um círculo e retorna 1 se o ponto estiver dentro do circulo ou 0 caso não estiver. Para isso você deve utilizar a função `distancia`;

- `criaCircunscrito` – a função recebe como parâmetro o endereço de memória da estrutura de um retângulo e retorna o maior circulo circunscrito neste retângulo. O círculo deve ser alocado dinamicamente.

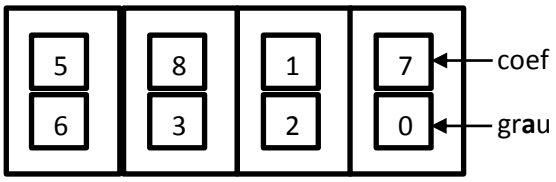
Utilize a main abaixo para testar o seu programa:

```
int main (void)
{
  Retangulo *rect = criaRetangulo(criaPonto(10,10), criaPonto(20,25));
  Circulo *circ = criaCirculo(criaPonto(30,15), 12);
  Circulo *circ_rect;
  float area_circ, area_rect, dist;
  Ponto pt;

  area_circ = areaCirculo(circ);
  printf("Area Circulo: %f\n", area_circ);
  area_rect = areaRetangulo(rect);
  printf("Area Retangulo: %f\n", area_rect);
  dist = distancia(circ->p, criaPonto(rect->p1.x +
                                ((rect->p2.x - rect->p1.x)/2), rect->p1.y +
                                (rect->p2.y - rect->p1.y)/2));
  printf("Distancia entre centro do circulo e centro do retangulo: %f\n",
                                                                    dist);

  pt = criaPonto(28, 19);

  if (pertence(pt, circ) == 1)
    printf("O ponto pertence ao circulo!\n");
  else
    printf("O ponto não pertence ao circulo!\n");

  circ_rect = criaCircunscrito(rect);
  printf("Posicao do Circulo Circunscrito: %f %f\n", circ_rect->p.x,
                                                     circ_rect->p.y);
  printf("Raio do Circulo Circunscrito: %f\n", circ_rect->r);
  return 0;
}
```

2. Implemente um programa para manipulação de polinômios, onde o tipo estruturado `Termo` representa um termo de um polinômio:

```
struct termo
{
  int coef;    /* coeficiente do termo*/
  int grau;    /* grau do termo       */
};
typedef struct termo Termo;
```

Considere que um vetor de termos é utilizado para armazenar um polinômio. Por exemplo, o polinômio $5x^6 + 8x^3 + x^2 + 7$ (ou seja, $5x^6 + 8x^3 + 1x^2 + 7x^0$) estaria armazenado em um vetor de Termos de tamanho 4, sempre ordenado decrescentemente por grau:

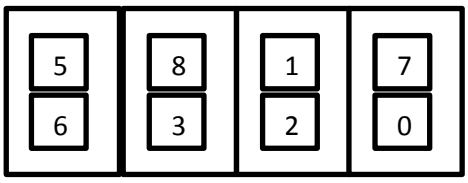


Escreva a função `criaVetorSoma`, que

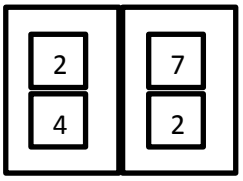
- Recebe um primeiro vetor de termos e seu tamanho (representando um polinômio) e um segundo vetor de termos e seu tamanho (representando um segundo polinômio). A função recebe também o endereço de uma variável onde deve ser devolvido o tamanho do novo vetor a ser criado pela função.
- A função deve criar um novo vetor de termos correspondente à soma dos 2 polinômios recebidos, do tamanho exato necessário. A função retorna o novo vetor, ou seja, o endereço do primeiro elemento desse novo vetor, devolvendo também o número de elementos desse novo vetor.

Por exemplo, tendo como entrada os seguintes polinômios:

$5x^6 + 8x^3 + x^2 + 7$

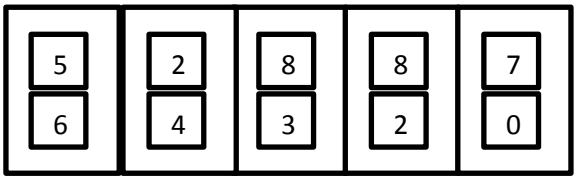


$2x^4 + 7x^2$



O vetor resposta, de tamanho 5, seria o correspondente ao polinômio:

$5x^6 + 2x^4 + 8x^3 + 8x^2 + 7$



Em seguida, implemente a função principal do programa para criar dois polinômios e utilizar a função `criaVetorSoma` para somar os dois polinômios e exibir o polinômio resultante na tela.

3. O departamento de transportes do Rio de Janeiro mantem um registro de todos os carros registrados. O registro de cada carro possui as seguintes informações:

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

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

Data
Mês
Ano

Carro
Placa
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 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 veiculo no seguinte formato:

      “MARIO, seu veiculo possui muitas multas!”

Onde MARIO deve ser substituído pelo nome do dono do veiculo. 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 veiculo no seguinte formato:

  "MARIO, seu veiculo é muito antigo!"

  Onde MARIO deve ser substituído pelo nome do dono do veiculo. 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 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.

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.

```
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;
}
```