Lista de exercícios 7 – Registros

1. Considere o registro visto em aula para representar retângulos alinhados aos eixos cartesianos:

```
struct Retangulo {
    int pse_x;   // Ponto superior esquerdo, coordenada x
    int pse_y;   // Ponto superior esquerdo, coordenada y
    int pid_x;   // Ponto inferior direito, coordenada x
    int pid_y;   // Ponto inferior direito, coordenada y
};
```

   Crie as seguintes funções para manipular retângulos:

```
/* Área do retângulo */
int area(struct Retangulo ret)
/* Perímetro do retângulo */
int perimetro(struct Retangulo ret)
/* Translada o retângulo em x e y, isto é, faz a operação
   pse_x += x, pse_y += y, pid_x += x, pid_y += y */
struct Retangulo translada(struct Retangulo ret, int x, int y)
```

2. Crie um registro chamado Complexo para representar números complexos do tipo $a + bi$. Basicamente, o registro armazena dois valores de ponto flutuante (a parte real e a parte imaginária). Feito isso, crie as seguintes funções:

```
/* Retorna a parte real do número */
float real(struct Complexo num)
/* Retorna a parte imaginária do número */
float imag(struct Complexo num)
/* Retorna a soma de dois números complexos */
struct Complexo soma(struct Complexo num1, struct Complexo num2)
/* Imprime o número no formato a + bi */
void imprime(struct Complexo num)
```

3. Crie o seguinte registro para representar um conjunto de **valores únicos**, isto é, sem repetição, possuindo no máximo 100 elementos:

```
struct Conjunto {
    int elementos[100]; // Elementos do conjunto
    int n;              // Número de elementos no conjunto
};
```

   Crie as seguintes funções para manipular conjuntos:

```
/* Cria um novo conjunto vazio */
struct Conjunto novo_conjunto(void);
/* Retorna 1 se o valor pertencer ao conjunto, 0 caso contrário */
int pertence(struct Conjunto conjunto, int valor);
/* Adiciona um valor ao conjunto. Se o valor já pertencer ao conjunto,
   ele não é adicionado novamente */
```

```c
struct Conjunto adiciona(struct Conjunto conjunto, int valor);
/* Retorna um novo conjunto contendo a união entre dois conjuntos */
struct Conjunto uniao(struct Conjunto conjunto1, struct Conjunto conjunto2);
/* Retorna um novo conjunto contendo a intersecção entre dois conjuntos */
struct Conjunto intersecao(struct Conjunto conjunto1, struct Conjunto conjunto2);
```

Você não precisa fazer checagens para verificar se está sendo ultrapassado o limite de 100 elementos. Dicas: Primeiramente, veja o exercício de lista feito em aula. Para a função `adiciona`, chame a função `pertence` antes de adicionar para verificar se o item já existe no conjunto. Para a função `uniao`, basta criar um novo conjunto e chamar a função `adiciona` para cada elemento de conjunto1 e conjunto2. Para a função `intersecao`, itere sobre todos os elementos de conjunto1 e adicione cada elemento ao conjunto de interseção se o elemento também pertencer ao conjunto 2.

4. Considere os seguintes registros para representar pessoas e funcionários de uma empresa:

```c
#define MAX_TAMANHO 50   // Tamanho máximo do nome e cargo de uma pessoa
#define MAX_FUNC 1000    // Número máximo de funcionários

/* Armazena informações sobre uma pessoa*/
struct Pessoa {
    char nome[MAX_TAMANHO];
    int idade;
    char endereco[MAX_TAMANHO];
};

/* Armazena informações sobre um funcionário */
struct Funcionario {
    struct Pessoa pessoa;
    char cargo[MAX_TAMANHO]; // Cargo do funcionário
    float salario;
};

/* Armazena informações sobre uma empresa */
struct Empresa {
    struct Funcionario funcionarios[MAX_FUNC];
    int n_funcionarios;     // Número de funcionários
};
```

Crie as seguintes funções para manipular esses registros:

```c
/* Recebe as informações de uma pessoa */
struct Pessoa recebe_pessoa(void);
/* Recebe as informações de um funcionário */
struct Funcionario recebe_funcionario(void);
/* Recebe as informações de uma empresa */
struct Empresa recebe_varios_funcionarios(void);
/* Imprime os dados de uma empresa */
void imprime(struct Empresa empresa);
```