Agregados Heterogêneos

0	1	João	700
1	2	Maria	2000
2	3	Jesus	5000
3	4	Mateus	3000
4	5	Noé	500
5	6	Moises	700

São estruturas que podem agregar informações de diferentes tipos.

Agregados Heterogêneos

- Coleção de elementos onde é possível utilizar um índice de qualquer tipo **imutável**.

  OBS: Vetores, Matrizes e Strings, os índices são sempre inteiros.

- Principais operações: armazenar e recuperar dados heterogêneos.

Agregados Heterogêneos

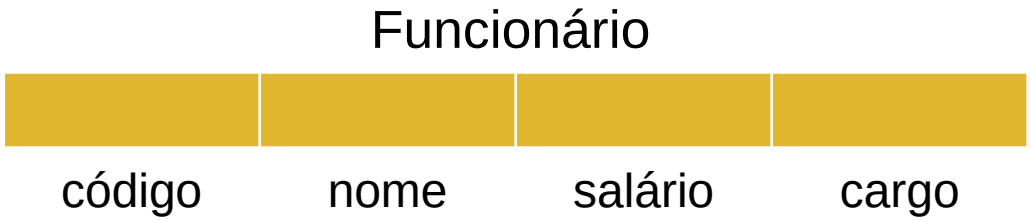
- Na linguagem C os agregados heterogêneos podem ser implementados através do uso de vetores combinados com **registros**.
- Registro é uma coleção de dados de diversos tipos diferentes;

Registro

- Um registro agrupa dados que possuem algum relacionamento lógico;
- Exemplo: dados de um funcionário.
- Os dados de um funcionário podem ser armazenados em campos de um registro.

Registro

- Deve ter um nome diferente para cada campo;
- Exemplo:
  - Registro → Funcionário
  - Campos → código, nome, salário, cargo.



Na Linguagem C, o tipo de dado registro é denominado ESTRUTURA.

Estrutura

Sintaxe da declaração de um tipo estrutura para funcionário:

```
struct funcionario {
        int codigo;
        char nome[30];
        float salario;
        char cargo;
}
```

Estrutura

Por exemplo, podemos criar uma **estrutura** para armazenar informações sobre um produto hipotético com código, descrição e preço:

```
struct produto {
        int codigo;
        char tamanho;
        float preco;
}
```

Estrutura

```
struct produto {
        int codigo;
        char tamanho;
        float preco;
}
```

Para criar um produto específico temos que definir uma variável do tipo definido (produto):

```
struct produto copo;
```

Estrutura

Para armazenar dados de um produto específico temos que atribuir valores para os campos:

```
struct produto copo;
copo.código = 771;
copo.tamanho = ‘P’;
copo.preco = 5.20;
```

Estrutura

```
struct produto {
        int codigo;
        char tamanho;
        float preco;
}
```

Podemos mostrar os campos:

```
...
printf("%i", copo.codigo);
printf("%s", copo.tamanho);
printf("%f", copo.preco);
```

Estrutura

```
struct produto {
        int codigo;
        char tamanho;
        float preco;
}
```

Podemos alterar valores dos campos:

```
copo.tamanho = 'G';
```

Estrutura

```
struct produto copo;
copo.codigo = 771;
copo.tamanho = ‘P’;
copo.preco = 5.20;
printf(“%i”, copo.codigo);
printf(“%c”, copo.tamanho);
printf(“%f”, copo.preco);
copo.tamanho = ‘G’;
//Podemos mostrar novamente
```

Estrutura

```
Void mostraCampos(struct produto o){
        printf(“\n%i”, o.codigo);
        printf(“\n%c”, o.tamanho);
        printf(“\n%f\n”, o.preco);
}
```

Estrutura

```
struct produto {
        int codigo;
        char tamanho;
        float preco;
}
```

```
Void mostraCampos(struct produto o){
        printf("\n%i", o.codigo);
        printf("\n%c", o.tamanho);
        printf("\n%f\n", o.preco);
}
```

Estrutura

```
struct produto {
        int codigo;
        char tamanho;
        float preco;
}
```

```
Void mostraCampos(struct produto o){
        printf("\n%i", o.codigo);
        printf("\n%c", o.tamanho);
        printf("\n%f\n", o.preco);
}
```

```
Int main() {
    struct produto copo;
    copo.codigo = 771;
    copo.tamanho = 'P';
    copo.preco = 5.20;
    mostraCampos(copo);
    copo.tamanho = 'G';
    mostraCampos(copo);
}
```

EXERCÍCIO

1) Foi realizada uma pesquisa de algumas características físicas de cinco habitantes de uma região. De cada habitante foram coletados os seguintes dados:
   - Sexo (M ou F),
   - cor dos olhos (A = Azuis ou C = Castanhos),
   - cor dos cabelos (L = Louros, P = Pretos ou C = Castanhos),
   - idade.

EXERCÍCIO

1) Foi realizada uma pesquisa de algumas características físicas de cinco habitantes de uma região. De cada habitante foram coletados os seguintes dados: sexo, cor dos olhos (A = Azuis ou C = Castanhos), cor dos cabelos (L = Louros, P = Pretos ou C = Castanhos) e idade.

- **Crie uma estrutura *pessoa*;**
- **Crie campos para os dados;**

EXERCÍCIO

1) Foi realizada uma pesquisa ...

- **Crie uma estrutura *pessoa*;**
- **Crie campos para os dados;**

```
struct pessoa {
        char cabelo;
        char olhos;
        char sexo;
        int idade;
};
```

EXERCÍCIO

1) Foi realizada uma pesquisa de algumas características físicas de **cinco** habitantes de uma região. De cada habitante foram coletados os seguintes dados: sexo, cor dos olhos (A = Azuis ou C = Castanhos), cor dos cabelos (L = Louros, P = Pretos ou C = Castanhos) e idade.

Faça funções que:

a) leia os dados e armazene em um vetor de estruturas;

Exercício 1

```
#define quant 5
int main() {
    struct pessoa hab[quant];
    // ler os dados e armazenar em um vetor
    leitura(hab, quant);
...
```

Exercício 1

```
void leitura(struct pessoa v[], int n){
    // Le os dados e armazena em um vetor de estruturas.
    for (int k=0; k<n; ++k){
        printf("\nSEXO (M/F): ");   scanf(" %c", &v[k].sexo);
        printf("\nOLHOS (A/C): ");   scanf(" %c", &v[k].olhos);
        printf("\nCABELO (L/P/C): ");   scanf(" %c", &v[k].cabelo);
        printf("\nIDADE: ");   scanf("%i", &v[k].idade);
        printf("\n");
    }
}
```

EXERCÍCIO

1) Foi realizada uma pesquisa de algumas características físicas de cinco habitantes de uma região. De cada habitante foram coletados os seguintes dados: sexo, cor dos olhos (A = Azuis ou C = Castanhos), cor dos cabelos (L = Louros, P = Pretos ou C = Castanhos) e idade.

Faça funções que:

a) Leia os dados e armazene em um vetor de estruturas;

b) Determine a média de idade das pessoas com olhos castanhos e cabelos pretos. Mostre esse resultado no PP.

Exercício 1

```
int main() {
    struct pessoa hab[quant];
    // ler os dados e armazenar em um vetor
    leitura(hab, quant);
    // Determinar a média de idade das pessoas com
    // olhos castanhos e cabelos pretos.
    printf("\nIdade media = %5.1f\n", idadMedia(hab,quant));
...
```

Exercício 1

```
float idadMedia(struct pessoa hab[], int n){
    /* Determina a média de idade das pessoas com
    olhos castanhos e cabelos pretos. */
    int cont = 0, soma = 0;
    for (int k=0; k<n; ++k){
        if (hab[k].olhos == 'C' && hab[k].cabelo == 'P')
            cont += 1;
            soma = soma + hab[k].idade;
    }
    if (cont > 0)
        return (float)soma/cont;
    return 0;
}
```

EXERCÍCIO

1) Foi realizada uma pesquisa ...

Faça funções que:

a) Leia os dados e armazene em um agregado;

b) Determine a média de idade das pessoas com olhos castanhos e cabelos pretos. Mostre esse resultado no PP.

c) Determine e devolva a maior idade entre os habitantes.

Exercício 1

```
int main() {
    struct pessoa hab[quant];
    // ler os dados e armazenar em um vetor
    leitura(hab, quant);
    // Determinar a média de idade das pessoas com
    // olhos castanhos e cabelos pretos.
    printf("\nIdade media = %5.1f\n", idadMedia(hab,quant));
    printf("\nMaior idade = %i\n", maiorIdade(hab,quant));
...
```

Exercício 1

```
int maiorIdade(struct pessoa hab[], int n){
    // Determina a maior idade entre os habitantes.
    if (n <= 0)
        return 0;
    int maior = hab[0].idade;
    for (int k=0; k<n; ++k)
        if (hab[k].idade > maior)
            maior = hab[k].idade;
    return maior;
}
```

Exercício 1

1) Foi realizada uma pesquisa ...

Faça funções que:

a) Leia os dados e armazene em um agregado;

b) Determine a média de idade das pessoas com olhos castanhos e cabelos pretos. Mostre esse resultado no PP.

c) Determine e devolva a maior idade entre os habitantes.

d) Determine e devolva a quantidade de indivíduos do sexo feminino cuja idade está entre 18 e 35 (inclusive) e que tenham olhos azuis e cabelos louros.

Exercício 1

```
int main() {
    struct pessoa hab[quant];
    // ler os dados e armazenar em um vetor
    leitura(hab, quant);
    // Determinar a média de idade das pessoas com
    // olhos castanhos e cabelos pretos.
    printf("\nIdade media = %5.1f\n", idadMedia(hab,quant));
    printf("\nMaior idade = %i\n", maiorIdade(hab,quant));
    printf("\nQuantidade de loiras bonitas = %i\n", elas(hab,quant));
}
```

Exercício 1

```
int elas(struct pessoa hab[], int n){
    /* Calcula a quantidade de indivíduos do sexo feminino cuja
idade está
       entre 18 e 35 (inclusive) e que tem olhos azuis e cabelos
louros.*/
    int cont = 0;
    for (int k=0; k<n; ++k)
        if (hab[k].sexo =='F' && 18 <= hab[k].idade &&
hab[k].idade <= 35)
            if (hab[k].olhos == 'A' && hab[k].cabelo == 'L')
                cont += 1;
    return cont;
}
```

EXERCÍCIOS

2) Faça um programa que utilize os registros:

Clientes
cod_cli
nome
fone
endereco

Documentos
num_doc
cod_cli
data_venc
valor

EXERCÍCIOS

2) Faça um programa que utilize os registros:

Clientes
cod_cli
nome
fone
endereco

Documentos
num_doc
cod_cli
data_venc
valor

- Considere o máximo de 30 clientes e 100 documentos.
- Crie uma estrutura para clientes e outra para documentos.

Exercício 2

```
struct clientes {
    int cod_cli;
    char nome[30];
    char fone[20];
    char endereco[40];
    int ativo;
};
```

Exercício 2

```
struct clientes {
    int cod_cli;
    char nome[30];
    char fone[20];
    char endereco[40];
    int ativo;
};
```

```
struct documentos {
    int num_doc;
    int cod_cli;
    struct data data_venc;
    float valor;
    int ativo;
};
```

Exercício 2

```
struct clientes {
    int cod_cli;
    char nome[30];
    char fone[20];
    char endereco[40];
    int ativo;
};
```

```
struct data {
    int dia;
    int mes;
    int ano;
};
```

```
struct documentos {
    int num_doc;
    int cod_cli;
    struct data data_venc;
    float valor;
    int ativo;
};
```

Exercício 2

Crie um ***menu*** para a realização de cada uma das operações especificadas a seguir:

- **Cadastrar clientes** – não pode existir mais de um cliente com o mesmo código;
- **Cadastrar documentos** – um documento só pode ser cadastrado para um cliente que já exista;

Exercício 2

```
int main(void){
    struct clientes cli[nCli];
    int i;
    for(i=0;i<nCli;++i) cli[i].ativo = 0;
    struct documentos docs[nDocs];
    for(i=0;i<nDocs;++i) docs[i].ativo = 0;
. . .
```

Exercício 2

```
int main(void){
    struct clientes cli[nCli];
    int i;
    for(i=0;i<nCli;++i) cli[i].ativo = 0;
    struct documentos docs[nDocs];
    for(i=0;i<nDocs;++i) docs[i].ativo = 0;
    char resp;
    while (1) {
        printf ("\n\nMENU DE OPCOES:");
        printf ("\nA) Cadastrar clientes.");
        printf ("\nB) Cadastrar documentos.");
        printf ("\nS) Sair.");
        printf("\n\tQual a sua opcao?: ");    scanf(" %c", &resp);
        if(resp=='S') break;
        if (resp == 'A')
            cadastre_cli(cli);
        else if (resp == 'B')
            cadastre_doc(cli, docs);
    }
}
```

Exercício 2

Crie uma ***função*** para a realização de cada uma das operações especificadas a seguir:

- **Cadastrar clientes** – não pode existir mais de um cliente com o mesmo código (utilize o índice do vetor);

- Cadastrar documentos – um documento só pode ser cadastrado para um cliente que já exista;

Exercício 2

```
void cadastre_cli(struct clientes cli[]){
    printf ("\n\nCADASTRO DE CLIENTES:");
    char loop = 'S';
    int k = 0;
    while (loop == 'S'){
        // Gerar Indice
        while (k < (nCli - 1) && cli[k].ativo)
            k++;
        if (!cli[k].ativo)
            cli[k].ativo = 1;
        else {
            printf("\n*** Memoria cheia ***");
            printf("\nDigite Enter p/ sair:");  scanf(" %c", loop);
            return;        }
        cli[k].cod_cli = k;
        printf("\nNome do cliente: ");  scanf("%s",cli[k].nome);
        printf("\nTelefone: ");     scanf("%s",cli[k].fone);
        printf("\nEndereco: ");     scanf("%s",cli[k].endereco);
        printf("\n\tDeseja cadastrar outro cliente (S/N)?: ");
        scanf(" %c", &loop);
    }
}
```

Exercício 2

Crie uma *função* para a realização de cada uma das operações especificadas a seguir:

- Cadastrar clientes – não pode existir mais de um cliente com o mesmo código;
- **Cadastrar documentos** – um documento só pode ser cadastrado para um cliente que já exista;

Exercício 2

```
void cadastre_doc(struct clientes cli[], struct
    documentos docs[]){
    printf ("\n\nCADASTRO DE DOCUMENTOS:");
    char loop, nome[30];
    printf("\nNome do cliente: ");  scanf("%s", nome);
    int num, d, codCli;
    codCli = codigo_cli(cli, nome);
    if (codCli == nCli)
        printf("Cliente inexistente.");
    else
            ...
```

Exercício 2

```
int codigo_cli(struct clientes cli[], char name[]){
    //Encontra o codigo de um cliente.
    int cod=nCli,    k = 0;

    while (!cli[k].ativo || strcmp(name, cli[k].nome) && k < (nCli - 1))
        k++;
    if (!strcmp(name, cli[k].nome) && cli[k].ativo)
        cod = cli[k].cod_cli;
    return cod;
}
```

Exercício 2

```
void cadastre_doc(struct clientes cli[], struct documentos docs[]){
    printf ("\n\nCADASTRO DE DOCUMENTOS:");
    char loop, nome[30];
    printf("\nNome do cliente: ");  scanf("%s", nome);
    int num, d, codCli;
    codCli = codigo_cli(cli, nome);
    if (codCli == nCli)
        printf("Cliente inexistente.");
    else
        loop = 'S';
        while (loop == 'S'){
            printf("\nNumero do documento: ");  scanf("%i", &num);
            while (!valida(docs, num)){
                printf("\nNumero do documento: ");
                scanf("%i", &num);
            }
              ...
```

Exercício 2

```
int valida(struct documentos docs[], int num){
    for (int d=0; d<nDocs; d++)
        if (num == docs[d].num_doc && docs[d].ativo){
            printf("*** Numero invalido ***");
            return 0;
        }
    return 1;
}
```

Exercício 2

```
void cadastre_doc(struct clientes cli[], struct documentos docs[]){
    printf ("\n\nCADASTRO DE DOCUMENTOS:");
    char loop, nome[30];
    printf("\nNome do cliente: ");  scanf("%s", nome);
    int num, d, codCli;
    codCli = codigo_cli(cli, nome);
    if (codCli == nCli)
        printf("Cliente inexistente.");
    else
        loop = 'S';
        while (loop == 'S'){
            printf("\nNumero do documento: ");  scanf("%i", &num);
            while (!valida(docs, num)){
                printf("\nNumero do documento: ");
                scanf("%i", &num);
            }
            d = geraIndice(docs);
            if (d == nDocs){
                printf("\n*** Memoria cheia ***");
                printf("Digite Enter p/ sair:");    scanf(" %c",&loop);
                return;
            }
              ...
```

Exercício 2

```
int geraIndice(struct documentos v[]){
    int i = 0;
    while (i < (nDocs - 1) && v[i].ativo)
        i++;
    if (!v[i].ativo){
        v[i].ativo = 1;
        return i;    }
    else
        return nDocs;
}
```

Exercício 2

```
void cadastre_doc(struct clientes cli[], struct documentos docs[]){
            ...

          d = geraIndice(docs);
          if (d == nDocs){
              printf("\n*** Memoria cheia ***");
              printf("Digite Enter p/ sair:");    scanf(" %c",&loop);
              return;
          }
          docs[d].num_doc = num;
          docs[d].cod_cli = codCli;
          printf("\nDia do vencimento: ");
          scanf("%i", &docs[d].data_venc.dia);
          printf("\nMes do vencimento: ");
          scanf("%i", &docs[d].data_venc.mes);
          printf("\nAno do vencimento: ");
          scanf("%i", &docs[d].data_venc.ano);
          printf("\nValor: ");
          scanf("%f", &docs[d].valor);
          printf("\nCadastrar outro doc? (S/N)?: ");
          scanf(" %c", &loop);
       }
}
```