

Licenciatura em Engenharia Informática – 2023/24

Programação

3: Estruturas

Francisco Pereira (xico@isec.pt)

Problema: Gerir os clientes de um banco

- Informação sobre cada cliente:
 - Nome, número de conta e saldo
- Hipótese 1:
 - Usar 3 variáveis simples para cada cliente

```
char nome[100];  
char nconta[15];  
int montante;
```

- Solução pouco prática. Porquê?

- Os tópicos relativos a um cliente estão relacionados:
 - Como expressar isso no código?



Utilizando
estruturas

- Estrutura:
 - Objeto que agrupa variáveis relacionadas
 - As componentes podem ser de tipos diferentes

- Dois passos:
 1. Criar um novo tipo estruturado
 2. Declarar variáveis do novo tipo

1. Criar um tipo estruturado

- Criar um novo tipo chamado `struct dados`
- Definir
 - Nome do novo tipo de dados
 - Características (campos)

```
struct dados {  
    char nome[100];  
    char nconta[15];  
    int montante;  
};
```

Feito apenas
1 vez!

**Ainda não foram
declaradas variáveis!**

2. Declarar variáveis

Sempre que
necessário

m

```
struct dados m;
```

nome	
<input type="text"/>	
nconta	montante
<input type="text"/>	<input type="text"/>

n

```
struct dados n;
```

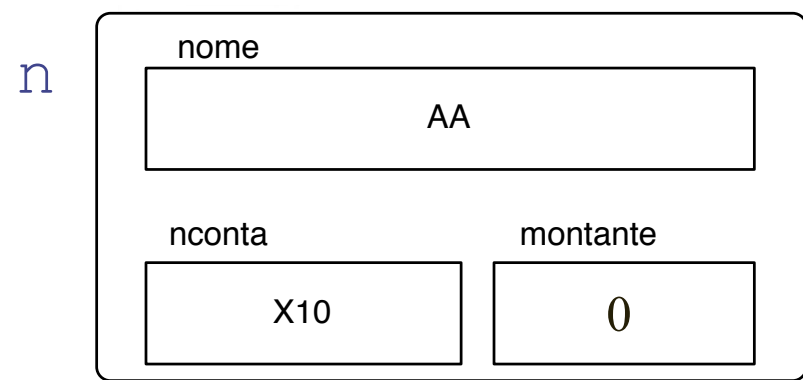
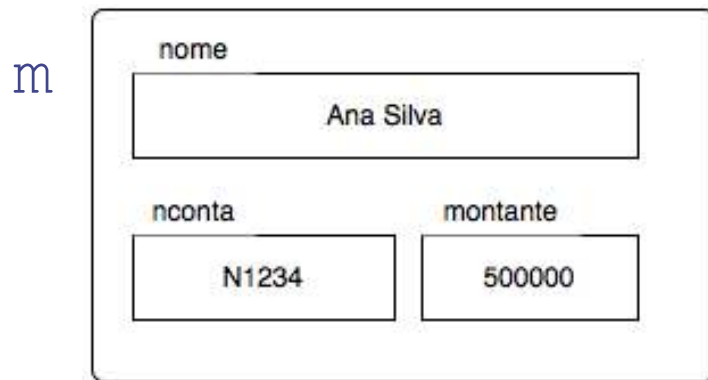
nome	
<input type="text"/>	
nconta	montante
<input type="text"/>	<input type="text"/>

- Acesso aos campos:
 - Operador •
 - Utilização: `nome_da_variável.nome_do_campo`
 - Exemplo:

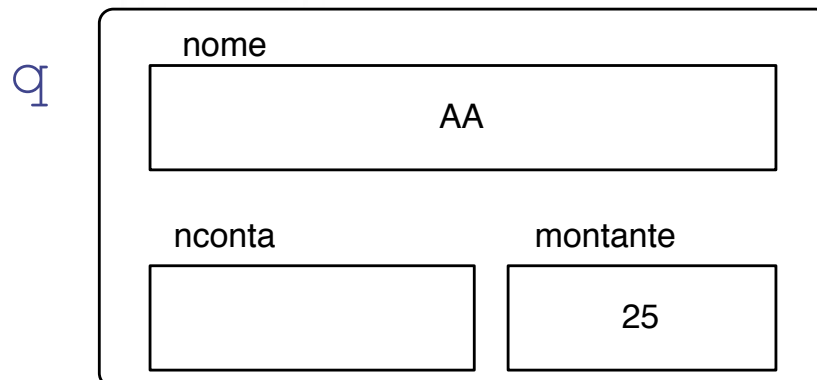
```
m.montante = 1200;  
  
scanf(" %99[^\n]", m.nome);  
  
if(strcmp(m.nconta, "X1234") == 0)  
    m.montante += 100;
```


Inicialização na Declaração

```
struct dados m = {"Ana Silva", "N1234", 500000};  
struct dados n = {"AA", "X10"};
```



```
struct dados q = {.montante=25, .nome="AA"};
```



C99

- Operador de atribuição (=)
 - Pode ser utilizado entre estruturas do mesmo tipo

```
m = n;
```

Válido se forem
estruturas do mesmo tipo

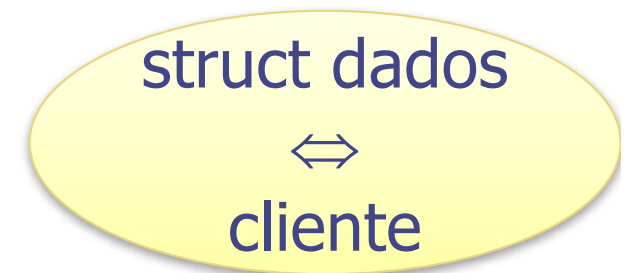
- Não é permitido utilizar operadores de comparação (==, !=) entre duas estruturas
 - Comparação tem que ser feita campo a campo

```
if (m == n)  
    printf("Sao Iguais\n");
```

Sempre
inválido

- Associa um novo nome a um tipo de dados

```
typedef struct dados cliente;  
  
struct dados {  
    char nome[100];  
    char nconta[15];  
    int montante;  
};
```



- Declarações válidas

```
struct dados a;
```



```
cliente b;
```



- Combinar criação do tipo com diretiva typedef

```
typedef struct dados {  
    char nome[100];  
    char nconta[15];  
    int montante;  
} cliente;
```

Não está a ser declarada
nenhuma variável

- Estruturas encadeadas:
 - Os campos de uma estrutura podem ser estruturas
- Exemplo:
 - Adicionar data em que se tornou cliente

- Solução

```
struct data {  
    int dia, mes, ano;  
};  
  
struct dados_d {  
    char nome[100];  
    char nconta[15];  
    int montante;  
    struct data in;  
};  
  
struct dados_d m;
```

m

nome

nconta

montante

in

dia

mes

ano

As regras de acesso
mantêm-se

- Exemplo:
 - Função que escreve a informação de um cliente passado como argumento
 - **A estrutura é passada por valor**

```
void escreve_info(cliente a)
{
    printf("Nome: %s\nNº conta: %s\tSaldo: %d\n",
          a.nome, a.nconta, a.montante);
}
```

- Exemplo
 - Função que inicializa uma estrutura do tipo cliente e devolve-a já preenchida

```
cliente obtem_info()  
{  
    cliente t;  
  
    printf("Nome: ");  
    scanf(" %99[^\n]", t.nome);  
    printf("N° conta: ");  
    scanf(" %14s", t.nconta);  
    printf("Saldo: ");  
    scanf("%d", &(t.montante));  
    return t;  
}
```

Variável
temporária


```
cliente a = {"Ana", "X100", 1000};
```

- Declarar:

```
cliente *p;
```

- Associar:

```
p = &a;
```

- Utilizar:


```
(*p).montante = 500;  
printf("%s\n", (*p).nome);
```

ou

$(*p).c \Leftrightarrow p->c$

```
p->montante = 500;  
printf("%s\n", p->nome);
```


- Desenvolver uma função que transfira um determinado montante entre 2 clientes
- Argumentos
 - **Endereços das estruturas** onde se encontra a informação dos 2 clientes
 - Valor a transferir
- Valor devolvido
 - 1: Ok
 - 0: Transferência cancelada por falta de saldo



Vão ser alteradas

Cliente
Origem

Cliente
Destino



```
int transfere(cliente *or, cliente *dest, int valor)
{
    if(or->montante < valor)
        return 0;
    else
    {
        or->montante -= valor;
        dest->montante += valor;
        return 1;
    }
}
```

- Problema
 - Armazenar e gerir informação relativa a diversos clientes do banco
- Solução
 - Utilizar um array de estruturas

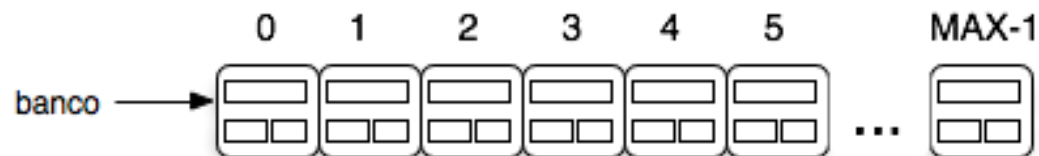
Abordagem 1

- Utilizar um array de estruturas com tamanho fixo

```
#define MAX 100

int main{
    cliente banco[MAX];
    int total=0;
    ...
}
```

Nº de clientes
armazenados



- Operações
 - Adicionar informação:
 - Adicionar um novo cliente
 - Eliminar informação:
 - Eliminar um cliente
 - Listar informação armazenada
 - Completa: Mostrar todos os clientes
 - Parcial: Procurar um subconjunto de clientes

`banco` e `total` são variáveis
locais da função `main()`

- Mostrar informação completa sobre todos os clientes

```
void escreve_todos(cliente tab[], int n){  
    int i;  
  
    for(i=0; i<n; i++)  
        escreve_info(tab[i]);  
}
```

Arrays de estruturas: Função 2

- Procurar e mostrar informação do cliente com o saldo mais elevado

```
void procura_mais_rico(cliente tab[], int n){
    int i, index=0;


    if(n==0)
    {
        printf("Sem Clientes\n");
        return;
    }

    for(i=1; i<n; i++)
        if(tab[i].montante > tab[index].montante)
            index = i;
    printf("Cliente com saldo mais elevado:\n");
    escreve_info(tab[index]);
}
```


Arrays de estruturas: Função 3

- Adicionar um cliente (inserção efetuada no final)

```
int adiciona_cliente(cliente tab[], int* n){  
  
    if(*n >= MAX){  
        printf("Tabela completa\n");  
        return 0;  
    }  
    else  
    {  
        tab[*n] = obtem_info();  
        (*n)++;  
        return 1;  
    }  
}
```



Ponteiro para o
número
de clientes

- Eliminar um cliente (identificação feita pelo nº de conta)
 - Estratégia:
 - Obter nº de conta
 - Procurar cliente no array
 - Se cliente existir então
 - Retirá-lo da posição em que se encontra
 - Reorganizar a informação armazenada
 - Atualizar nº de clientes

Duas alternativas

- Transferir o último cliente para esta posição.
- Mover uma posição para a esquerda todos os elementos a seguir a esta posição.

Arrays de estruturas: Função 4

```
int elimina_cliente(cliente tab[], int *n){
    char st[15];
    int i;

    printf("N° de conta do cliente a eliminar: ");
    scanf("%s" st);

    for(i=0; i<*n && strcmp(st, tab[i].nconta)!=0; i++)
        ;
    if(i==*n){
        printf("Cliente não existe\n"); return 0;
    }
    else
    {
        tab[i] = tab[*n-1];
        (*n)--;
        return 1;
    }
}
```

Arrays de estruturas: Função `main()`

```
#define MAX 100

int main() {
    cliente banco[MAX];
    int i, total=0;

    do {
        i = menu();
        switch(i) {
            case 1: adiciona_cliente(banco, &total); break;
            case 2: elimina_cliente(banco, &total); break;
            case 3: escreve_todos(banco, total); break;
            case 4: procura_mais_rico(banco, total); break;
        }
    } while(i != 5);
    return 0;
}
```

Arrays de estruturas: Função menu ()

```
int menu() {
    int i;

    puts("1 - Adiciona Cliente");
    puts("2 - Elimina Cliente");
    puts("3 - Lista Clientes");
    puts("4 - Mostra Mais Rico");
    puts("5 - Terminar");
    puts("Escolha uma opção: ");

    do{
        scanf(" %d", &i);
    }while(i<1 || i>5);

    return i;
}
```

- Alterar as funções desenvolvidas de forma a que os clientes no array estejam sempre ordenados pelo número de conta
 - Os números de conta são únicos