

Licenciatura em Engenharia Informática – 19/20

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?

Estruturas em C

- Os tópicos relativos a um cliente estão relacionados:
 - Como expressar isso no código?
- Estrutura:
 - Objeto que agrupa variáveis relacionadas
 - As componentes podem ser de tipos diferentes

Utilizando
estruturas

Utilizar Estruturas em C

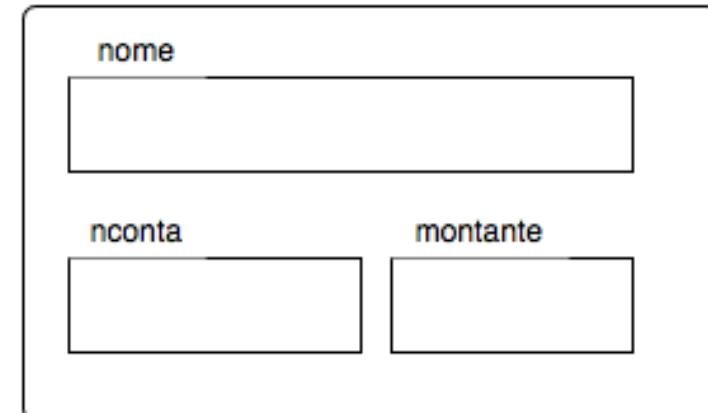
- Dois passos:
 1. Criar um novo tipo estruturado
 2. Declarar variáveis do novo tipo

1. Criar um tipo estruturado

- Criar um novo tipo chamado struct dados

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

Feito apenas
1 vez!



Atenção

**Ainda não foram
declaradas variáveis!**

2. Declarar variáveis

Sempre que
necessário

```
struct dados m;
```

nome

nconta

montante

m

```
struct dados n;
```

nome

nconta

montante

n

Operações com estruturas

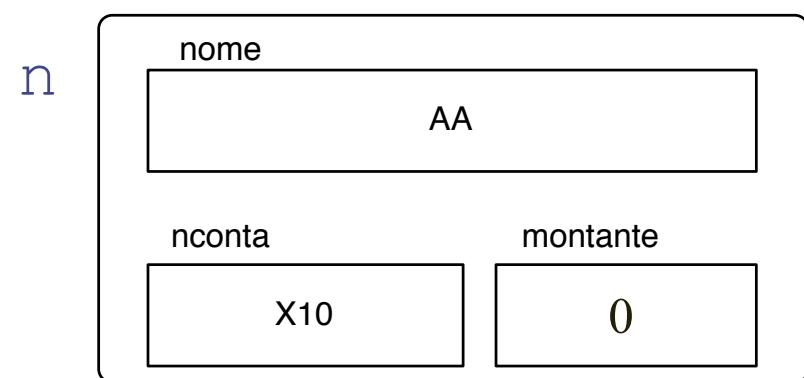
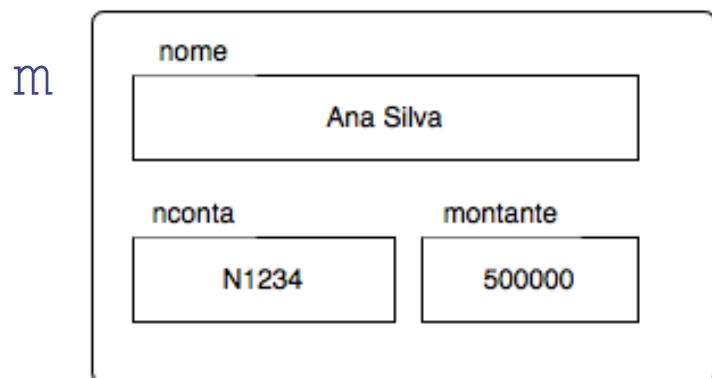
- Acesso aos campos:
 - Operador •
 - Utilização: nome_da_variável.nome_do_campo
 - Exemplo:

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

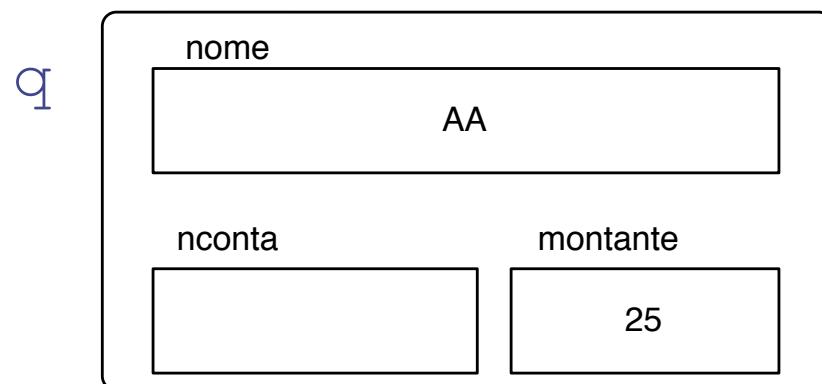
Inicialização na Declaração

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

struct dados n = {"AA", "X10"};
```



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



C99

Operações com estruturas

- Operador de atribuição (=)
 - Pode ser utilizado entre estruturas do mesmo tipo
- Não é permitido utilizar operadores de comparação (==, !=) entre duas estruturas
 - Comparação tem que ser feita campo a campo

Diretiva `typedef`

- 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

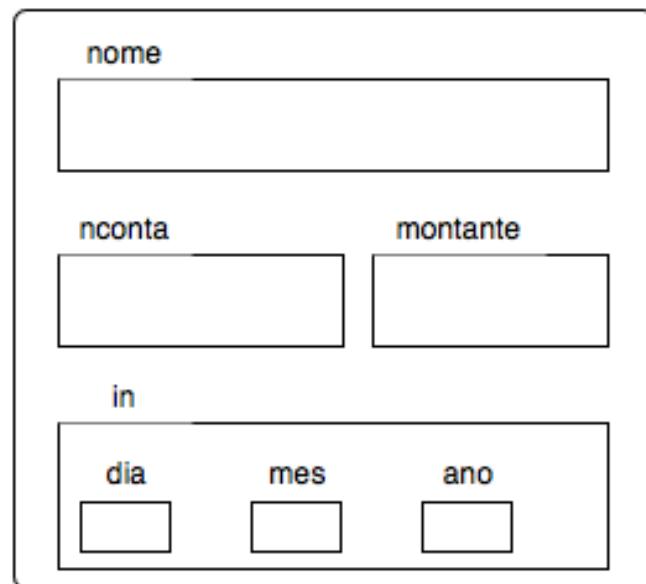
- Estruturas encadeadas:
 - Os campos de uma estrutura podem ser estruturas
- Exemplo:
 - Adicionar data em que se tornou cliente

Estruturas encadeadas

- 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



As regras de acesso
mantêm-se

Estruturas e Funções: Argumentos

- 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);
}
```

Estruturas e Funções: Valor devolvido

- 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("Nº conta: ");
    scanf(" %14s", t.nconta);
    printf("Saldo: ");
    scanf("%d", &(t.montante));
    return t;
}
```

Variável
temporária

Estruturas e Ponteiros

```
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);
```

Estruturas e Ponteiros

- 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

Estruturas e Ponteiros

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;
    }
}
```

Arrays de estruturas

- Problema
 - Armazenar informação relativa a diversos clientes
- Solução
 - Utilizar um array de estruturas

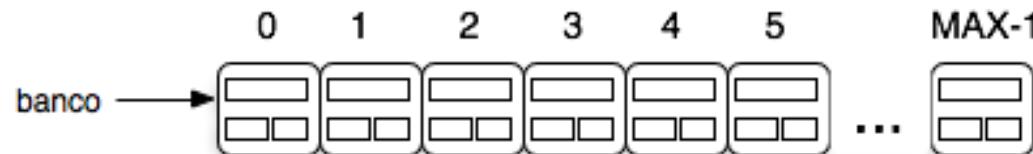
Abordagem 1

- Utilizar um array de estruturas com tamanho fixo

Nº de clientes
armazenados

```
#define MAX 100

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



Arrays de estruturas: Operações básicas

- 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 ()

Arrays de estruturas: Função 1

- 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)

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

Ponteiro para o
número
de clientes

Arrays de estruturas: Função 4

- 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

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

Duas alternativas

Arrays de estruturas: Função 4

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

    printf("Nº de conta do cliente a eliminar: ");
    scanf(" %14s" st);

    for(i=0; i<*n && strcmp(st, tab[i].nconta)!=0; i++)
        ;

    if(i==*n)
        printf("Cliente não existe\n");
    else
    {
        tab[i] = tab[*n-1];
        (*n)--;
    }
}
```

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;
}
```

Abordagem 1: Proposta de trabalho

- 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

Abordagem 2

- Utilizar um array de estruturas dinâmico
 - Dinamicamente o programa ajusta o tamanho do array ao número de clientes existentes

Gestão Dinâmica
de Memória

Exemplo Simples

Gerir dinamicamente
um array de inteiros

Requisição dinâmica de memória

- Durante a execução, um programa pode reservar espaço em memória para guardar informação

<stdlib.h>

```
void* malloc(size_t tam);
```

Ponteiro para o espaço reservado
(devolve NULL em caso de erro)

Número de
bytes a reservar

Exemplo Simples

- Reservar espaço para armazenar `tam` inteiros
 - O valor `tam` é indicado pelo utilizador

```
int main() {  
    int tam, *tab = NULL;  
  
    printf("Número de elementos: ");  
    scanf("%d", &tam);  
    tab = malloc(sizeof(int) * tam);  
    if(tab == NULL){  
        printf("Erro na alocação de memória");  
        return 0;  
    }  
    ...
```

Exemplo Simples

- Utilizar o array

```
...
// Inicializar
 inicializa(tab, tam);

// Mostrar
 mostra(tab, tam);
```

Exemplo Simples

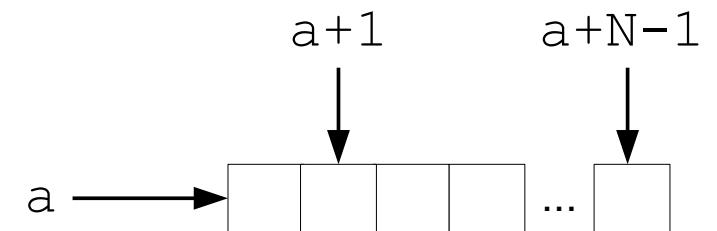
```
void inicializa(int a[], int n) {
    int i;

    for(i=0; i<n; i++)
        a[i] = 2*i;
}
```

Processamento habitual
de um array

```
void mostra(int a[], int n) {
    int i;

    printf("A tabela tem %d elementos:\n", n);
    for(i=0; i<n; i++)
        printf("%d\t", a[i]);
    putchar('\n');
}
```



Libertar memória

- É necessário libertar explicitamente os espaços de memória previamente reservados

```
void free(void *p);
```

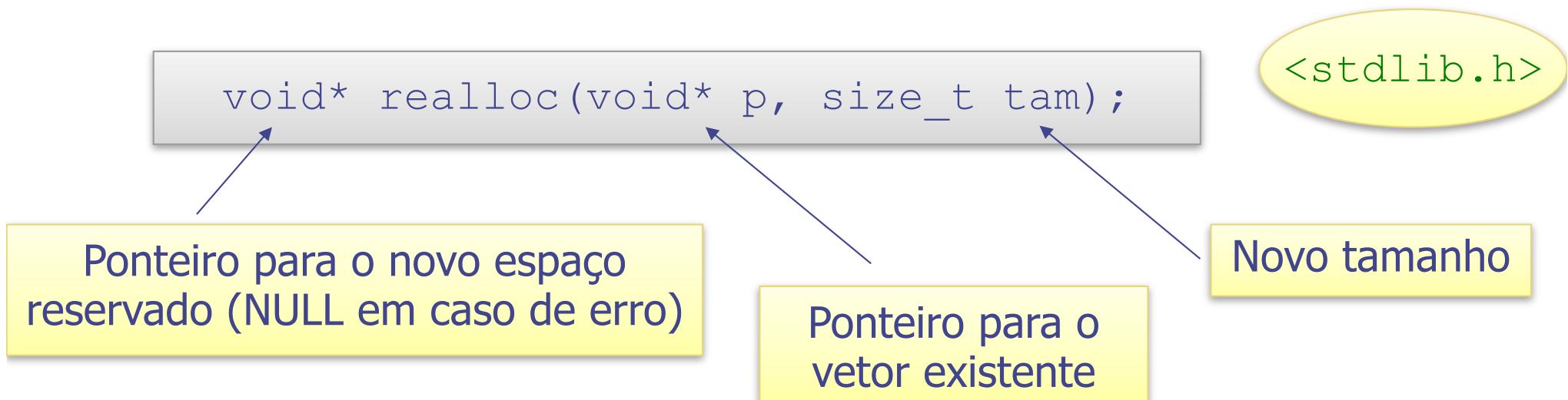
<stdlib.h>

Liberta a memória
referenciada por p

```
int main() {  
  
    int *tab = NULL, *aux;  
  
    ...  
  
    free(tab);  
    return 0;  
}
```

Reallocação Dinâmica de Memória

- E se o número de valores a guardar se alterar durante a execução?
 - Alterar dinamicamente o tamanho do vetor



- Novo tamanho pode ser superior ou inferior
- Alterações feitas no final do array existente

Exemplo Simples

- Adicionar v novas posições ao array existente

```
int *aux, v;

printf("Variacao: "); scanf("%d", &v);

aux = realloc(tab, sizeof(int)*(tam+v));
if(aux != NULL) {
    tab = aux;
    inicializa(tab+tam, v);
    tam += v;
}
mostra(tab, tam);
...
```

Gestão dinâmica: Alguns erros comuns

```
int *p;  
  
p = malloc(sizeof(int)*4);  
  
p = malloc(sizeof(int)*10);
```

```
int *p, *q;  
  
p = malloc(sizeof(int)*4);  
q = p;  
free(q);  
*(p+2) = 12;
```

Exemplo

- Função que crie e devolva uma string, a partir de duas strings passadas como argumentos:

```
char* cria_str(char* st1, char* st2);
```

- As strings `st1` e `st2` não devem ser modificadas.
- O espaço deve ser requisitado dinamicamente.
- Devolve um ponteiro para o início da nova string (NULL em caso de erro).

Função `cria_str()`

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char* cria_str(char* st1, char* st2) {
    char*st= malloc(sizeof(char)*
                    (strlen(st1) + strlen(st2) + 1));
    if(st != NULL) {
        strcpy(st, st1);
        strcat(st, st2);
    }
    return(st);
}

int main() {
    char *p = cria_str("Ola" , " Mundo!");
    puts(p);
    free(p);
    return 0;
}
```

Abordagem 2

- Utilizar um array de estruturas dinâmico
 - Dinamicamente o programa ajusta o tamanho do array ao número de clientes existentes

Ponteiro para o array dinâmico

Nº de clientes armazenados

```
int main () {  
    cliente *banco = NULL;  
    int total=0;  
    ...  
    if (banco != NULL)  
        free(banco);  
    return 0;  
}
```

Libertar memória no final

Abordagem 2: Operações básicas

- Listar informação armazenada
 - Completa: Mostrar todos os clientes
 - Parcial: Procurar um subconjunto de clientes

- Alterar  Gestão Dinâmica do array
 - Adicionar um novo cliente
 - Eliminar um cliente

Sem alterações

Abordagem 2: Adicionar um cliente

- Adicionar um novo cliente
 - Cliente é adicionado no final do array
 - Informação obtida do utilizador

```
cliente* adiciona_cliente(cliente *tab, int* n);
```

Ponteiro para o
início do array
depois da adição

Ponteiro para o
início do array

Tamanho do array

Abordagem 2: Adicionar um cliente

```
cliente* adiciona_cliente(cliente *tab, int* n) {
    cliente *aux;

    aux = realloc(tab, sizeof(cliente) * (*n+1));
    if(aux != NULL) {
        tab = aux;
        tab[*n] = obtem_info();
        (*n)++;
    }
    return tab;
}
```

Abordagem 2: Eliminar um cliente

- Eliminar um cliente
 - Através do número de conta
 - Transfere o elemento da última posição para o local onde está o cliente a eliminar

```
cliente* elimina_cliente(cliente *tab, int *n);
```

Ponteiro para o
início do array
depois da eliminação

Ponteiro para o
início do array

Tamanho do array

Abordagem 2: Eliminar um cliente

```
cliente* elimina_cliente(cliente *tab, int *n){  
    char st[100];  
    int i;  
    cliente *aux, t;  
  
    printf("Nº de conta do cliente a eliminar: ");  
    scanf(" %s", st);  
  
    for(i=0; i<*n && strcmp(st, tab[i].nconta)!=0; i++)  
    ;  
    ...
```

Abordagem 2: Eliminar um cliente

```
if(i==*n){  
    printf("Cliente não existe\n"); return tab;  
}  
else if(*n==1){  
    free(tab); *n=0; return NULL;  
}  
else{  
    t = tab[i];  
    tab[i] = tab[*n-1];  
    aux = realloc(tab, sizeof(cliente) * (*n-1));  
    if(aux!=NULL){  
        tab = aux;  
        (*n)--;  
    }  
    else  
        tab[i] = t;  
    return tab;  
}  
}
```