

## Licenciatura em Engenharia Informática – 2023/24

# Programação

## 4: Estruturas Dinâmicas

### 4.1: Arrays Dinâmicos

Francisco Pereira (xico@isec.pt)

---

- Armazenar informação de um cliente

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

- Operações:
  - Listar / Consultar clientes
  - Adicionar cliente
  - Eliminar cliente

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

Gestão Dinâmica  
de Memória

# Por Agora: Exemplo Simples

---

Gerir dinamicamente  
um array de inteiros

- 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

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

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

- Utilizar o array

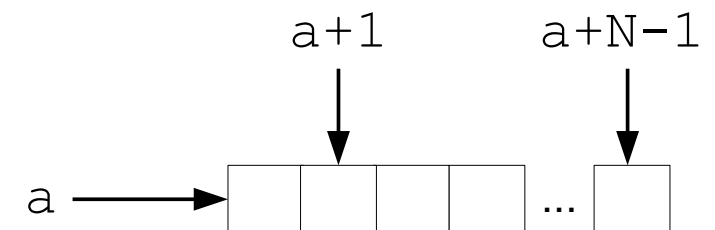
```
...  
// Inicializar  
inicializa(tab, tam);  
  
// Mostrar  
mostra(tab, tam);
```

# 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');  
}
```





- 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;  
}
```

# Gestão de memória dinâmica

```
int* f1() {  
    int *p = malloc(sizeof(int)*5);  
    if(p == NULL)  
        return NULL;  
    return p;  
}
```

1. Obter memória

```
void f2(int *a) {  
    free(a);  
}
```

3. Libertar memória

2. Passa pela  
função *main()*

```
int main() {  
    int *p=NULL;  
  
    p = f1();  
    f2(p);  
    return 0;  
}
```

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

```
void* realloc(void* p, size_t tam);
```

<stdlib.h>

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

Ponteiro para o vetor existente

Novo tamanho

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

- 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;
```

- 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;
}
```

- 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;  
  
    ...  
  
    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("N° 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;
}
}
```