

PROGRAMAÇÃO E ALGORITMOS (LEI)

Universidade da Beira Interior, Departamento de Informática
Hugo Pedro Proença, 2016/2017

Resumo



□ Alocação de Memória

- Estática

- Dinâmica

 - malloc

 - calloc

 - realloc

 - free

□ Exercícios

Gestão de Memória

- Ao contrário de outras linguagens de programação, o C disponibiliza um conjunto de funções de baixo nível para a gestão de recursos (memória).
 - ▣ Ausência de *Garbage Collector* (Java)
 - ▣ *Processo que administra os recursos utilizados por um programa. Caso eles não estejam a ser utilizados liberta-os automaticamente.*
- Esta característica aumenta a responsabilidade do programador.
 - ▣ Responsável pela eficiente gestão de recursos
- Optimização de recursos, evitando-se a necessidade de um processo (exigente em termos computacionais) em permanente execução.

Gestão de Memória



- Regra principal

- Todos os recursos explicitamente alocados durante a execução de um programa devem ser explicitamente libertados.

Alocação de Recursos



- Considere o seguinte exercício:
 - “Implemente um programa em linguagem C que receba números interior e mostre no écran a maior sequência de valores positivos recebidos. O programa deverá terminar ao ser introduzido o valor 0”.

- Problemas:
 - Quantos números vai ser necessário receber?
 - Qual a quantidade de memória necessária para os receber?

Alocação de Recursos

- Uma primeira abordagem será a alocação estática de recursos, por vezes referida como “em tempo de compilação”:
 - ▣ O programador (no código-fonte) estima um valor que julga suficiente e pede a respectiva alocação.
 - ▣ A alocação acontece no momento em que o programa começa a execução (no caso da função main), ou no início do processo de execução da função.

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

```
int main(){  
int v[10000];  
(...)  
}
```

Alocação de Recursos

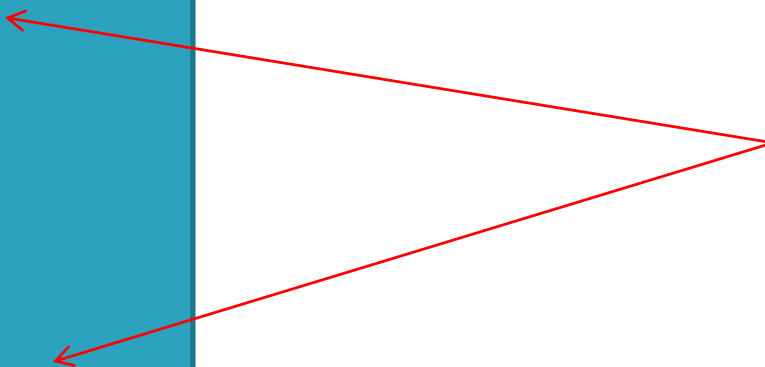
□ Exemplo:

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

```
void func(){  
int v[10000];  
(...)  
}
```

```
int main(){  
int v[10000];  
(...)  
}
```

Alcance Local
(dentro da função em
que estão definidas)



Recursos Dinâmicos

- ❑ No caso da alocação dinâmica, os recursos são pedidos e libertados durante a execução do programa, no momento da execução de uma dada linha.
- ❑ Compete ao programador definir o mecanismo de alocação / libertação, por forma a que a administração dos recursos seja correcta.
- ❑ Alocação:
 - ❑ malloc()
 - ❑ calloc()
 - ❑ realloc()
- ❑ Libertação
 - ❑ free()

Apontadores

- O que é um apontador?
 - Variável que contém um endereço de memória.
 - Exemplos: `int x=5; float z=3.1415; char c;`

5

x, 4 bytes

3.1415

z, 8 bytes

a

c, 1 byte

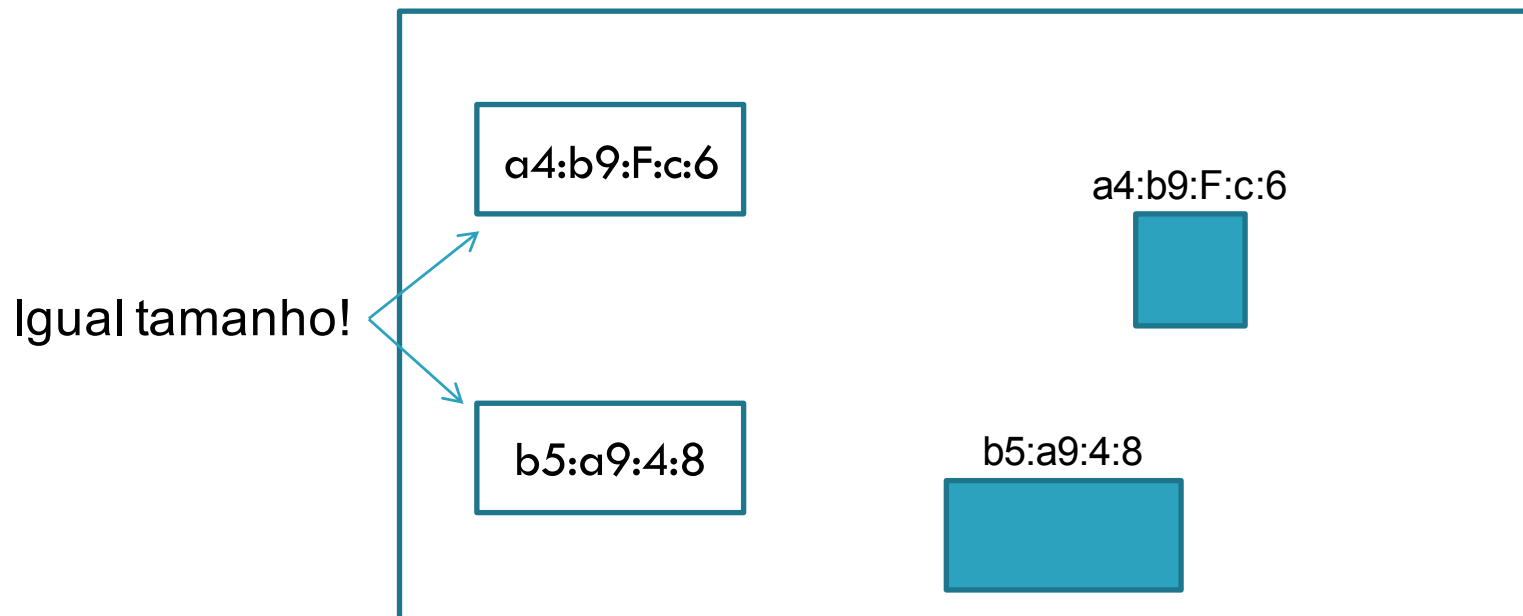
- Apontador:
 - Espaço ocupado depende do limite de endereçamento da máquina, mas tipicamente tem o mesmo tamanho de um inteiro. Exlº: **`int *p;`**

A00BF:FA00C

p, 4 bytes

Apontadores

- Um apontador contém informação sobre uma determinada localização e quantidade de memória:
 - Exlos: `int *a; double *b;`



Alocação Dinâmica

□ Função malloc()

- Está definida na biblioteca “stdlib.h”
- Inclusão de nova biblioteca “#include <stdlib.h>”

□ Protótipo:

- void *malloc (unsigned int size)
 - Recebe como parâmetro de entrada o total de bytes a alocar.
 - Retorna um apontador genérico para o bloco de memória alocado.

□ Exemplos:

□ `int *v=(int*)malloc(10 * sizeof(int));`

↑
cast

↑
Total de elementos

→
Espaço de 1 elemento

Alocação Dinâmica

- Após a alocação dinâmica, os recursos são utilizados exactamente da mesma forma como os recursos estáticos.

- Suporta tipos compostos:

```
typedef struct{
    int bi;
    char nome[80];
}Pessoa;
```

- Alocação de um conjunto de 100 pessoas:

- ▣ `Pessoa *v=(Pessoa *) malloc(100*sizeof(Pessoa))`

- Acesso aos recursos

- ▣ `v[0].nome //Nome da pessoa da primeira posição`

Alocação Dinâmica

□ Função calloc()

- Está definida na biblioteca “stdlib.h”
- Semelhante à função-base (malloc)
 - Aloca “x” elementos de um determinado tipo
 - Inicializa os recursos alocados, com o valor “0”.

□ Protótipo:

- `void *calloc (unsigned int number, unsigned int size)`
 - Recebe como parâmetros de entrada o número de elementos e o espaço ocupado por 1 unidade.
 - Retorna um apontador genérico para o bloco de memória alocado.

□ Exemplos:

- `int *v=(int*)calloc(10,sizeof(int));`

Alocação Dinâmica

□ Função realloc()

- Está definida na biblioteca “stdlib.h”
- Serve para realocar (aumentar ou diminuir de tamanho), um bloco previamente alocado.

□ Protótipo:

- `void *realloc (void *_old_pnt, unsigned int new_size)`
 - Recebe como parâmetros de entrada o apontador para o bloco previamente alocado e o novo tamanho do bloco.
 - Retorna um apontador genérico para o bloco de memória alocado.
- Tenha-se em atenção que o bloco (1º parâmetro) teve que ser previamente alocado de forma dinâmica.

Alocação Dinâmica

□ Exemplos:

```
int *v=(int*)malloc(10*sizeof(int)),i; //Aloca 10 inteiros;  
for (i=1;i<10;i++)  
    v=(int*)realloc(v,(10+i)*sizeof(int));
```

- Caso o bloco previamente alocado não tenha elementos, terá que estar a apontar para NULL.

```
int *v=NULL, i=0;  
v=(int*)realloc(v,(i+1)*sizeof(int));
```

Alocação Dinâmica

□ Função free()

- Serve para libertar todos os recursos dinamicamente alocados, qualquer que tenha sido a função utilizada no alocamento.
 - malloc(), calloc(), realloc()

□ Protótipo:

□ void free (void *pnt)

- Recebe um apontador para o bloco de memória.
- Tenha-se em atenção que o valor da variável não é alterado, apenas os recursos para onde aponta são libertados.

□ Exemplo:

```
free(v);
```

```
v=NULL;
```


Exercício

```
#include <stdio.h>
int main(){
    int *v=NULL, tot=0, x;
    do{
        printf("valor?\n");
        scanf("%d",&x);
        if (x!=0){
            tot++;
            v=(int*)realloc(v,tot*sizeof(int));
            v[tot-1]=x;
        }
    }while(x!=0);
    mostraMaiorSequenciaPositivos(v,tot);
}
```

Exercício

```
void mostraMaiorSequenciaPositivos(int *v, int tot){
    int pos=0,i, M=0,c=0;
    for (i=0;i<tot;i++){
        if (v[i]>0){
            c++;
            if (c>M){
                pos=i-M;
                M=c;
            }
        }
        else{
            c=0;
        }
    }
    for (i=pos;i<pos+M;i++)
        printf("%d",v[i]);
}
```