Departamento
de Ciência da
COMPUTAÇÃO

UDESC
Joinville

Linguagem C

alocação dinâmica de memória

André Tavares da Silva

andre.silva@udesc.br

Alocação dinâmica de memória

- A alocação dinâmica é o processo para alocar (reservar) memória em tempo de execução.
- Ela é utilizada normalmente quando não se sabe ao certo quanto de memória será necessária para o armazenamento de informações, podendo ser determinadas em tempo de execução conforme a necessidade do programa. Dessa forma evita-se o desperdício de memória.
- A alocação dinâmica é muito utilizada em problemas de estrutura de dados, por exemplo, listas encadeadas, pilhas, filas, arvores binárias e grafos.

Alocação dinâmica de memória

- No padrão C ANSI existem 4 funções para alocações dinâmica pertencentes a biblioteca stdlib.h
  - malloc(), calloc(), realloc() e free()
- As mais utilizadas são as funções malloc() e free(). Existem outras, mas não são funções padrões.
- As funções malloc() e calloc() são responsáveis por alocar memória, a realloc() por realocar a memória e por ultimo a free() fica responsável por liberar a memória alocada.

malloc()

- Recebe como parâmetro o número de bytes de memória que se deseja alocar.
- Retorna um ponteiro do tipo *void*, permitindo ser atribuído a qualquer tipo de ponteiro. Se for retornado o valor zero (*NULL*), significa que não foi possível realizar a alocação de memória

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

```
char *nome = (char*) malloc(100);
int size, *ptr;
scanf("%d", &size);
ptr = (int*) malloc(size * sizeof(int));
```

free()

- Ao ser alocado um espaço de memória dinamicamente, é necessário liberá-lo quando este não for mais necessário.
- Para liberar a memória, deve-se passar o ponteiro que aponta para o início da memória alocada para free(). O sistema sabe quantos bytes deve liberar, já que este tamanho é armazenado numa "tabela de alocação" interna

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

```
int *ptr = (int*) malloc(100 * sizeof(int));

...

free(ptr);
```

calloc()

- Também serve para alocar memória.
- Recebe como parâmetro a quantidade de elementos e o tamanho do tipo de dados.
- Retorna um ponteiro do tipo void.

```
void *calloc (unsigned int num, unsigned int size);
```

```
int size, *ptr;
scanf("%d", &size);
ptr = (int*) calloc(size, sizeof(int));
```

realloc()

- Modifica o tamanho da memória previamente alocada apontada por um endereço para o tamanho especificado por um novo valor.
- O ponteiro para o bloco é retornado porque realloc() pode precisar mover o bloco para aumentar seu tamanho. Se isso ocorrer, o conteúdo do bloco antigo é copiado para o novo.

```
void *realloc (void *ptr, unsigned int size);
```

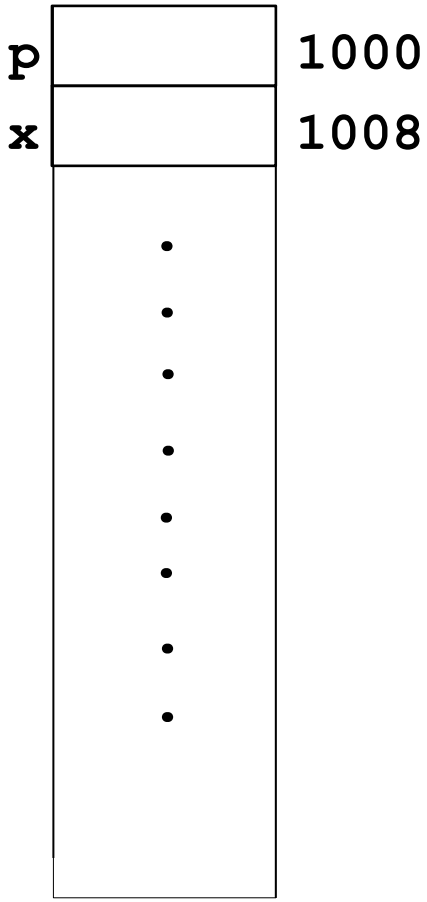
```
int *ptr = malloc(50*sizeof(int));
...
ptr = (int*) realloc(ptr, 100*sizeof(int));
```

EXEMPLO 1: ALOCANDO UM *FLOAT*

Exemplo 1: alocando um *float*

```
float *p, x;
```

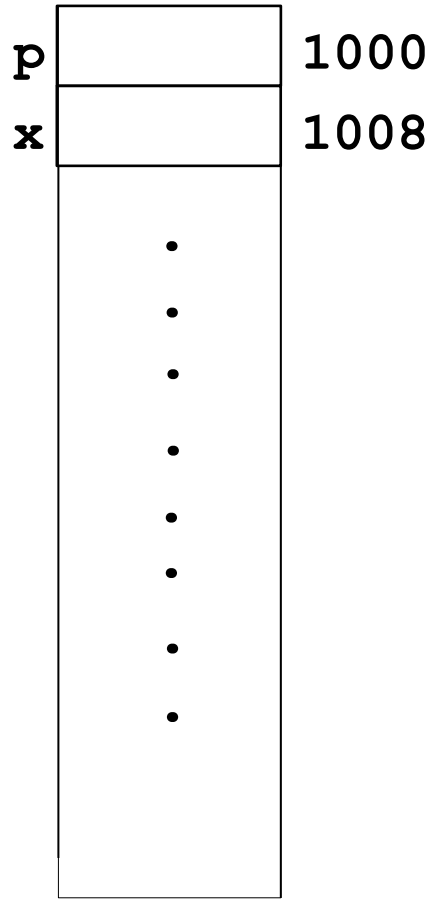
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
```

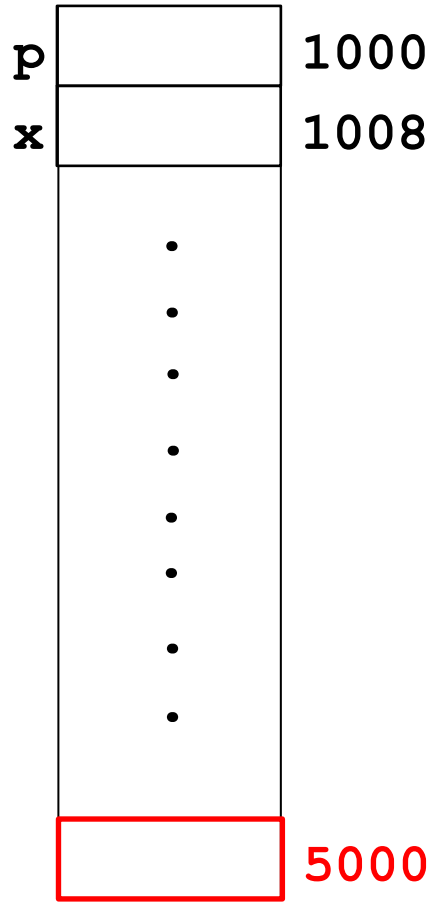
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
```

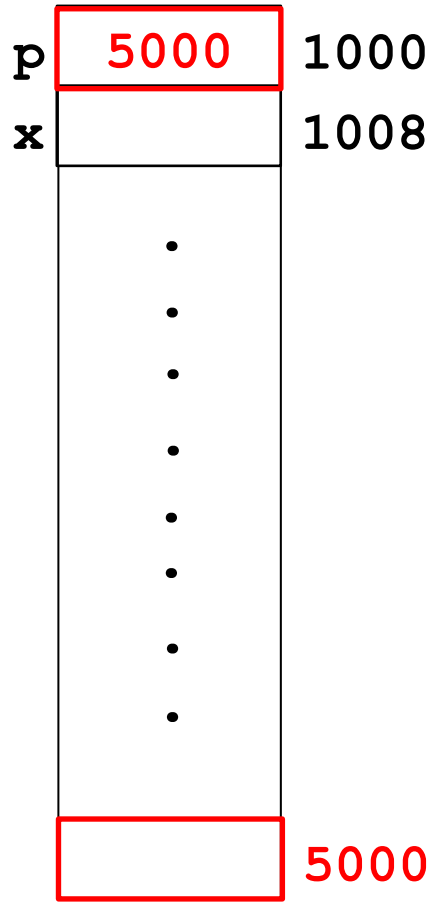
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
```

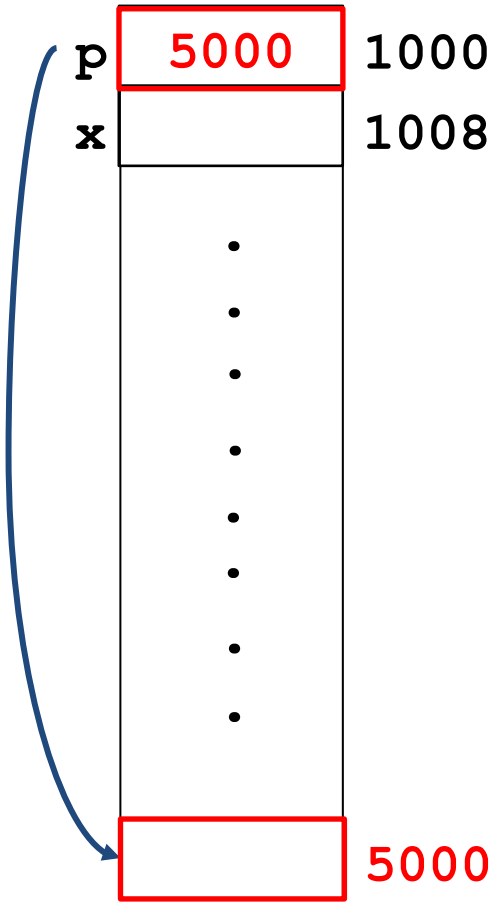
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
```

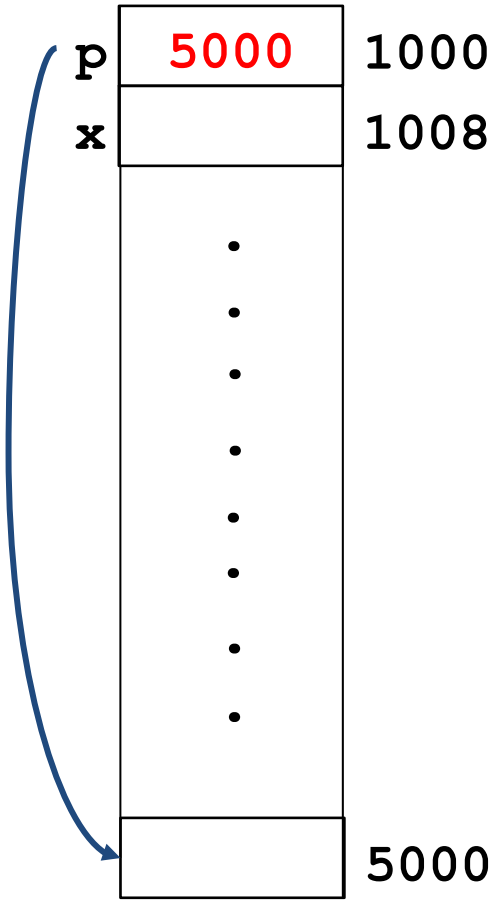
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
```

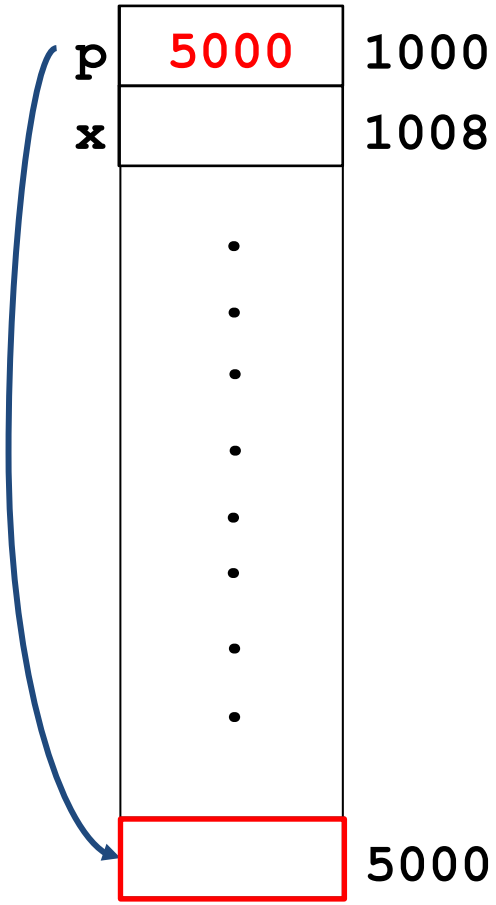
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
```

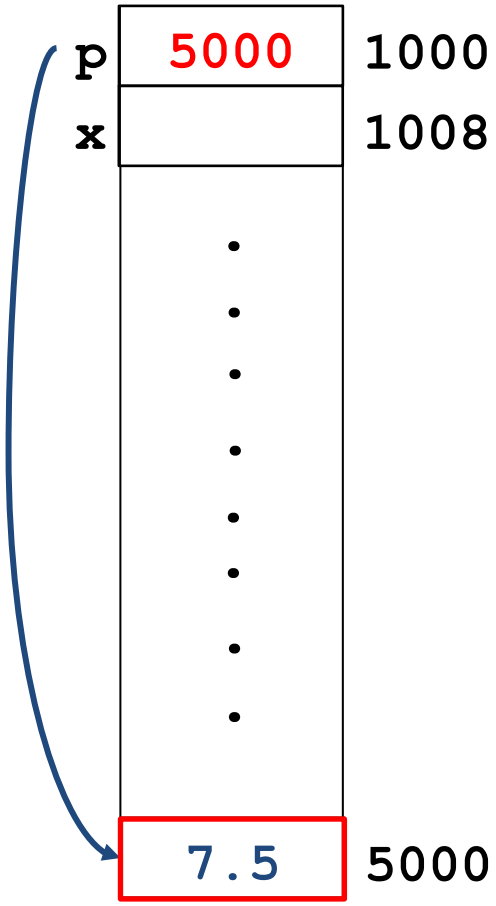
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p); // usuário digitou 7.5
```

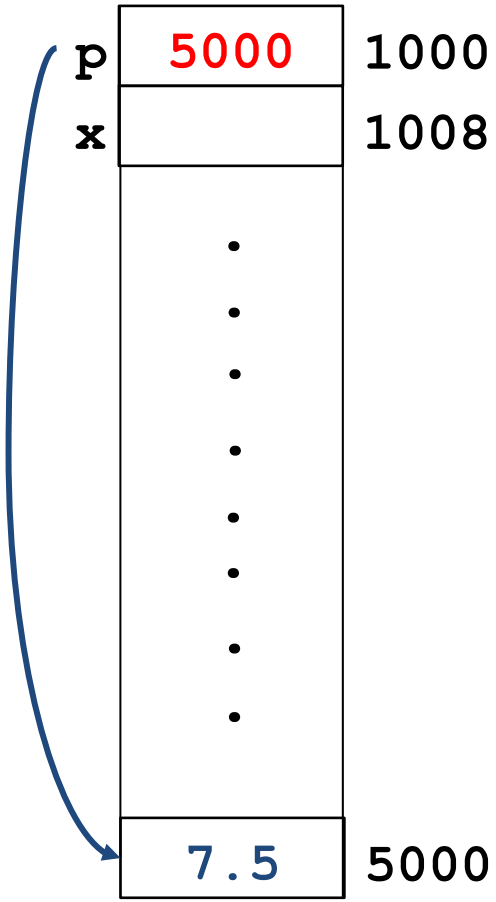
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p);
```

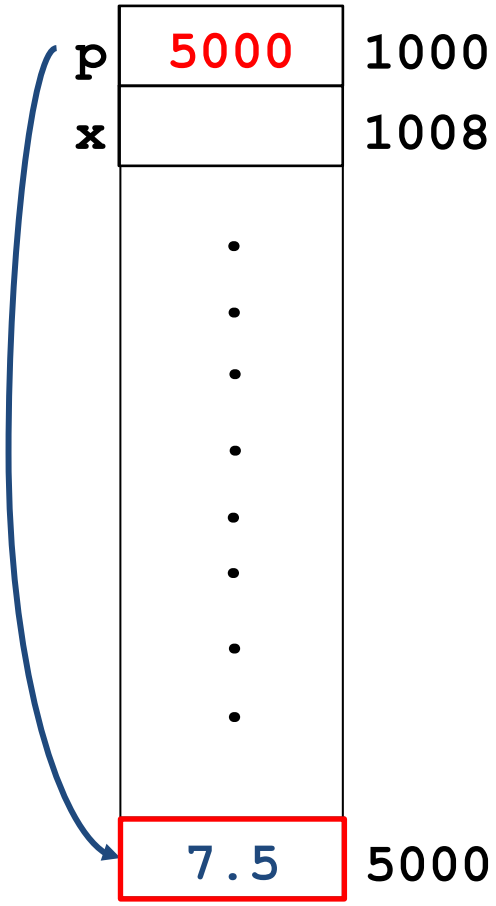
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p); // 7.5
```

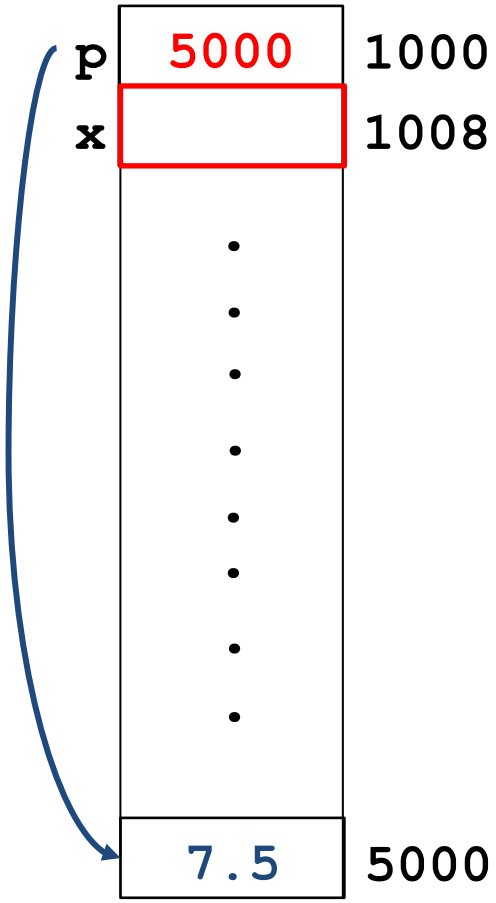
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p); // 7.5
x = *p;
```

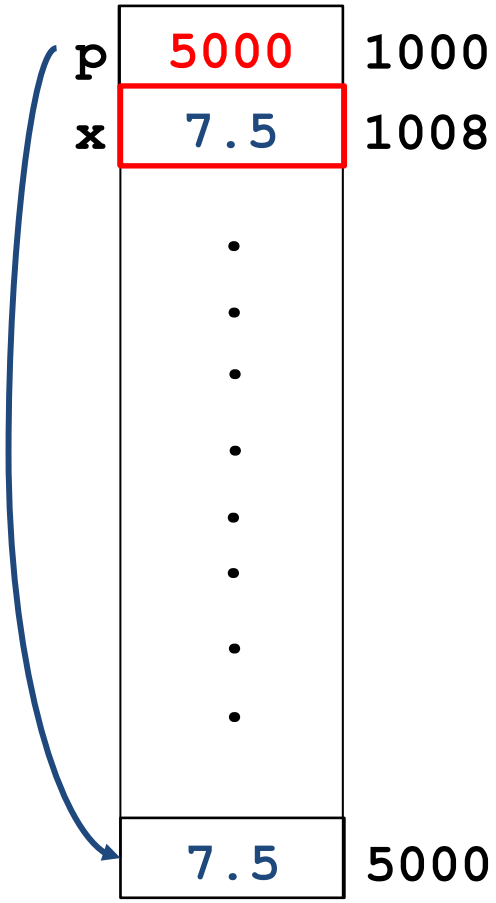
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p); // 7.5
x = *p;
```

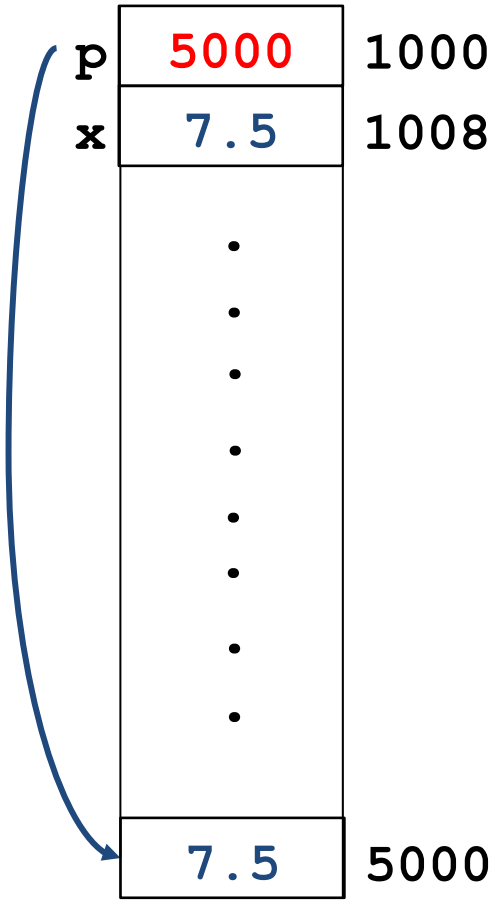
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p); // 7.5
x = *p;
printf("end. de x:%d\n", &x);
```

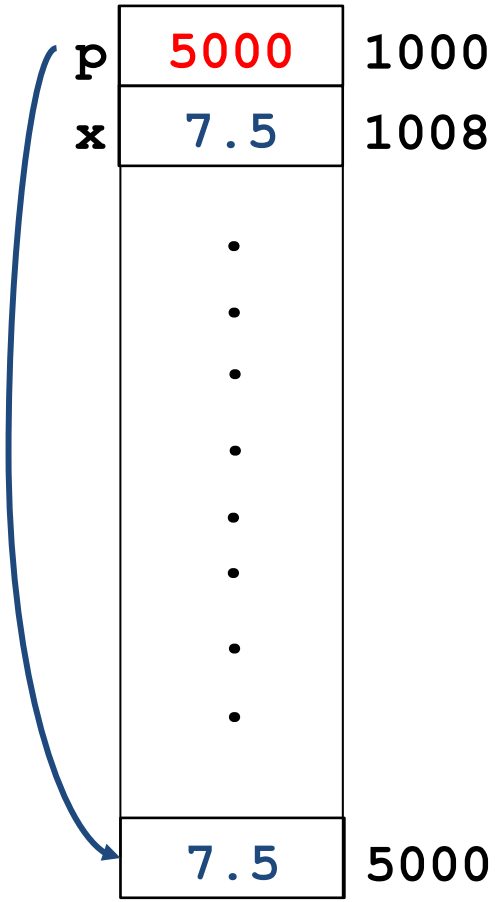
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p); // 7.5
x = *p;
printf("end. de x:%d\n", &x); //1008
```

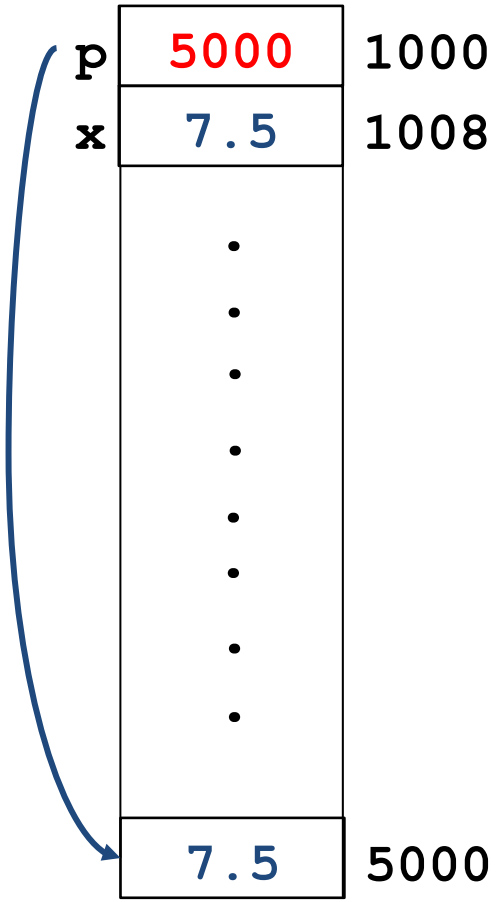
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p); // 7.5
x = *p;
printf("end. de x:%d\n", &x); //1008
printf("end. alocado:%d\n", p);
```

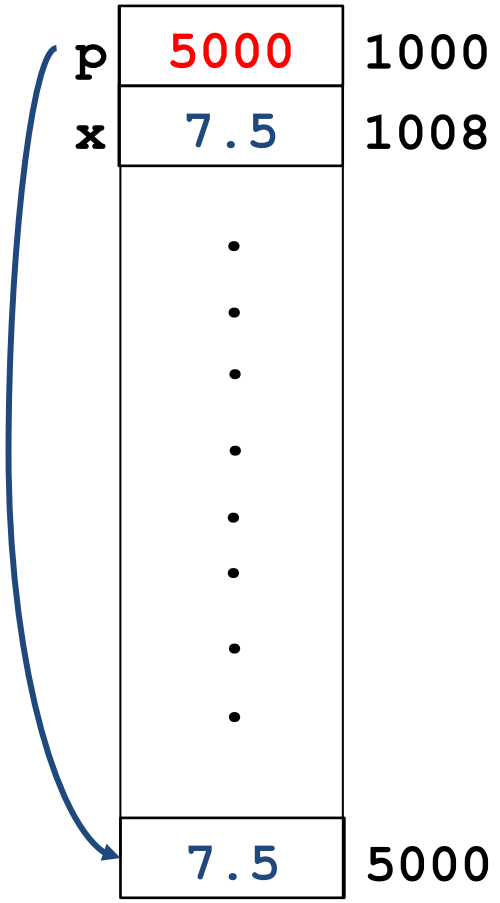
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p); // 7.5
x = *p;
printf("end. de x:%d\n", &x); //1008
printf("end. alocado:%d\n", p); //5000
```

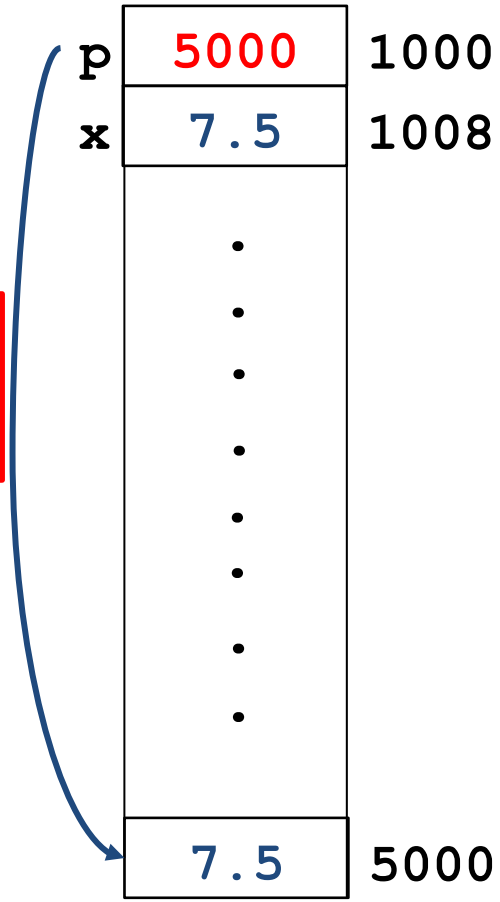
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p); // 7.5
x = *p;
printf("end. de x:%d\n", &x); //1008
printf("end. alocado:%d\n", p); //5000
```

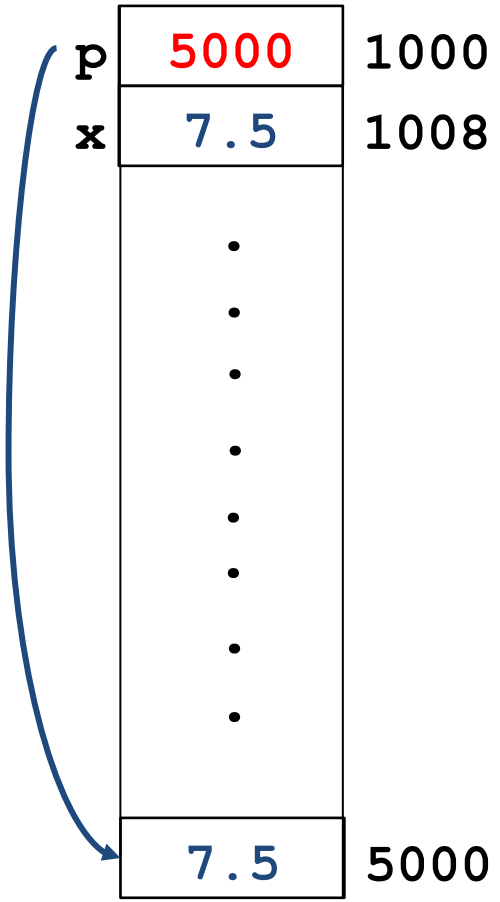
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p); // 7.5
x = *p;
printf("end. de x:%d\n", &x); //1008
printf("end. alocado:%d\n", p); //5000
free(p); // libera a memória
```

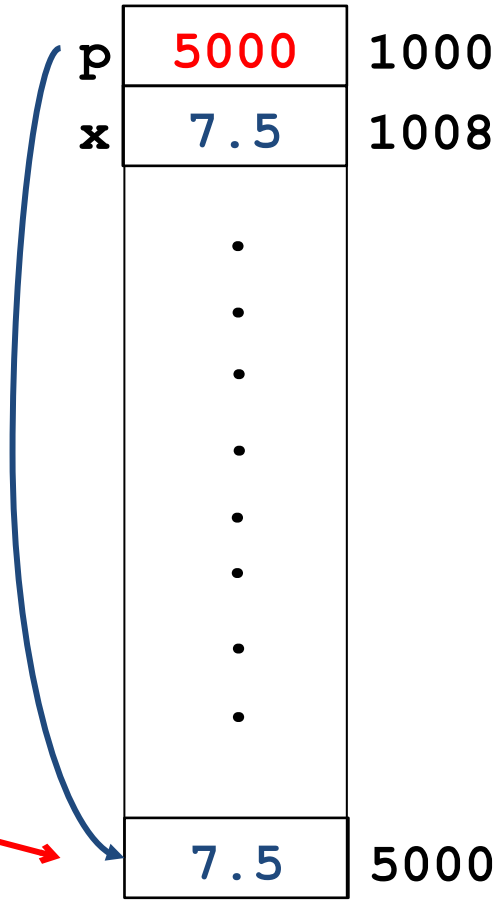
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p); // 7.5
x = *p;
printf("end. de x:%d\n", &x); //1008
printf("end. alocado:%d\n", p); //5000
free(p); // libera a memória
```

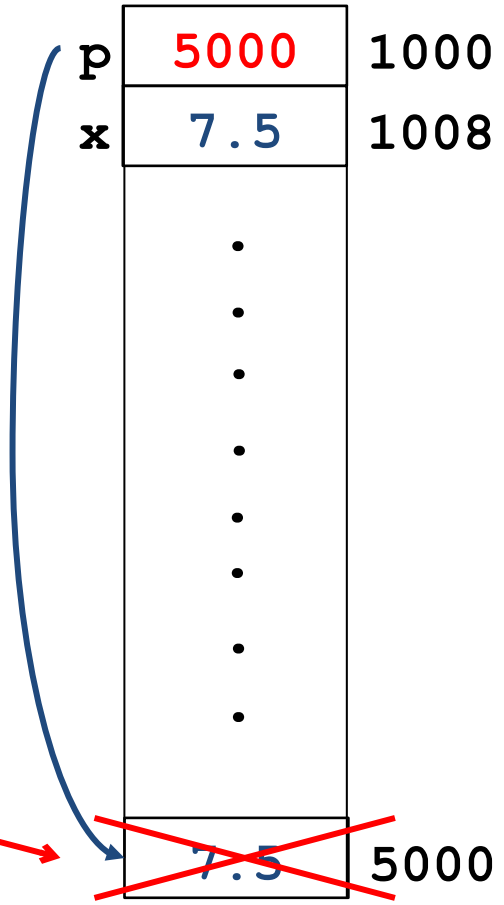
Modelo da Memória



Exemplo 1: alocando um *float*

```
float *p, x;
p = malloc( sizeof(float) );
scanf("%f", p);
printf("%f", *p); // 7.5
x = *p;
printf("end. de x:%d\n", &x); //1008
printf("end. alocado:%d\n", p); //5000
free(p); // libera a memória
```

Modelo da Memória



Liberação da memória

- Toda e memória alocada dinamicamente deve ser liberada quando não é mais necessária;

Liberação da memória

- Toda e memória alocada dinamicamente deve ser liberada quando não é mais necessária;
  - É responsabilidade do programa que alocou a memória;

Liberação da memória

- Toda e memória alocada dinamicamente deve ser liberada quando não é mais necessária;
  - É responsabilidade do programa que alocou a memória;

- Utiliza-se a função `free()`, passando o ponteiro como parâmetro;

Liberação da memória

- Toda e memória alocada dinamicamente deve ser liberada quando não é mais necessária;
  - É responsabilidade do programa que alocou a memória;

- Utiliza-se a função `free()`, passando o ponteiro como parâmetro;

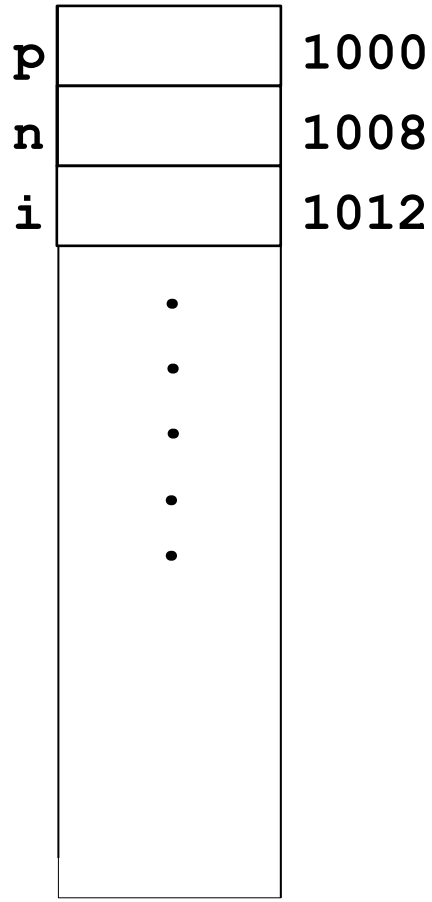
- A liberação não implica na remoção dos dados, mas informa ao S.O. que a área está disponível para novas alocações.

EXEMPLO 2: ALOCANDO UM VETOR

Exemplo 2: alocando um vetor

```
int *p, n, i;
```

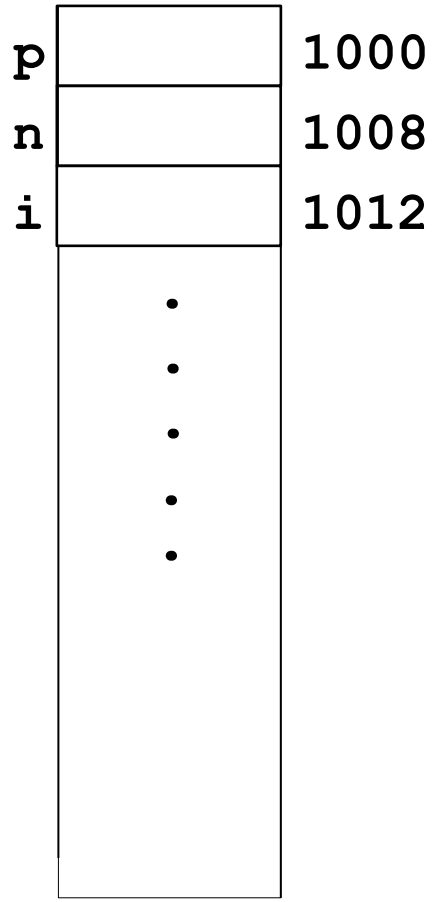
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
```

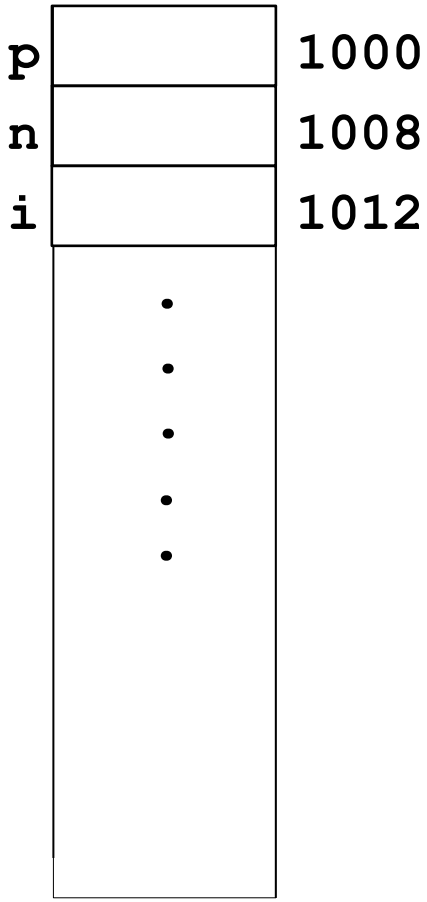
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
```

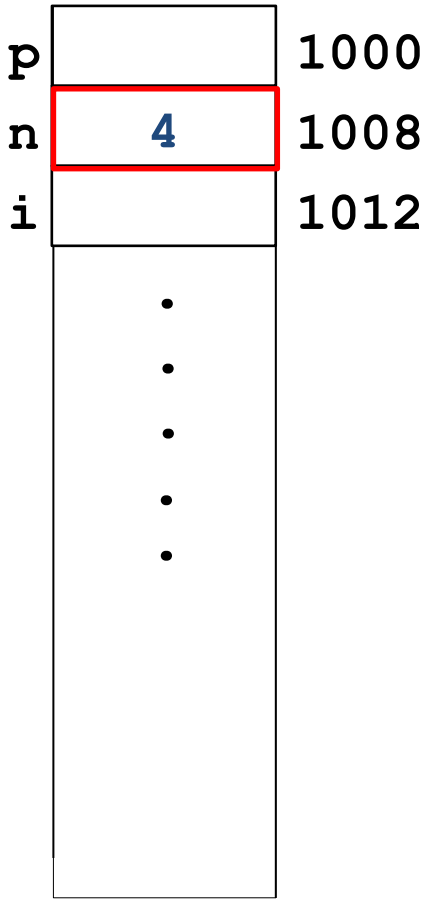
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);// usuário digitou 4
```

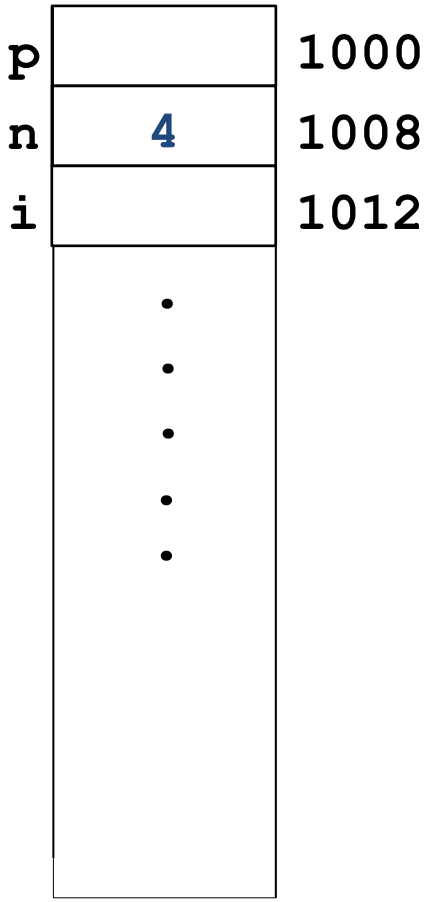
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
```

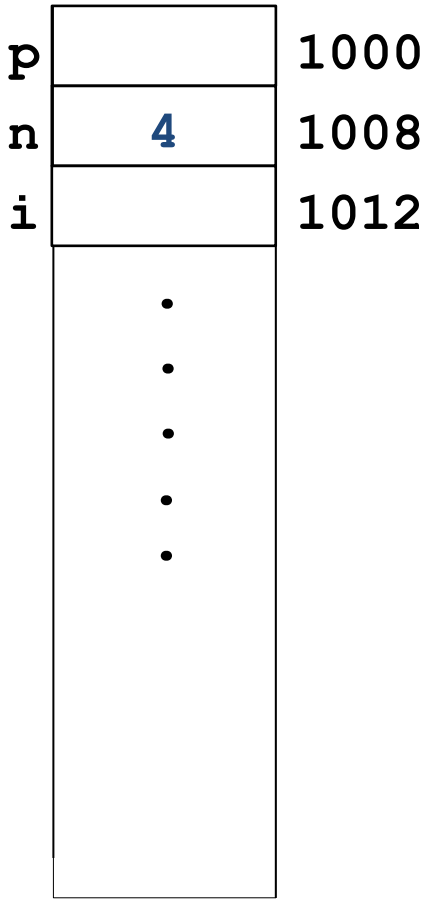
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
```

Modelo da Memória

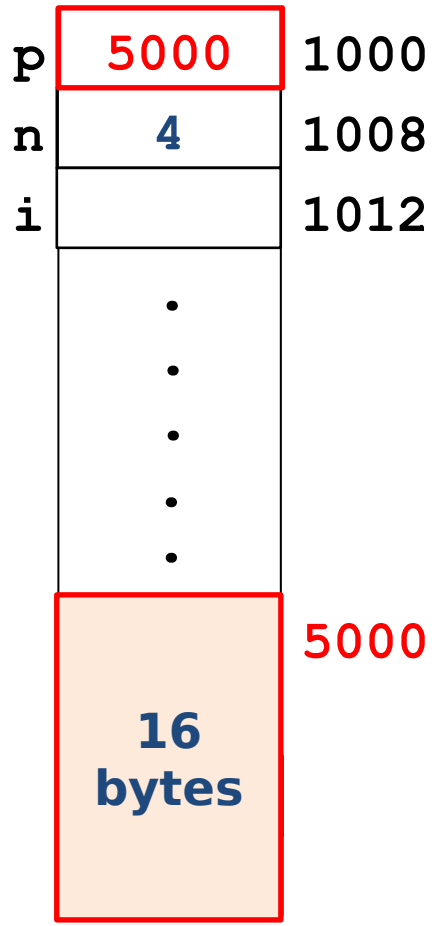


Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
```

```
// Aloca bloco de 16 bytes
// p aponta para o 1 byte do bloco
```

Modelo da Memória

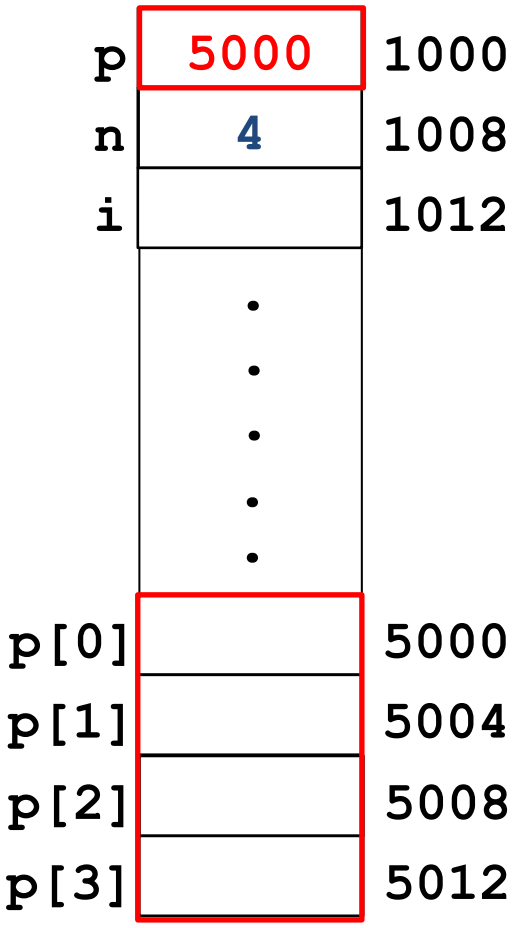


Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
```

```
// Aloca bloco de 16 bytes
// Como o ponteiro é int,
// o bloco é visto como um vetor
```

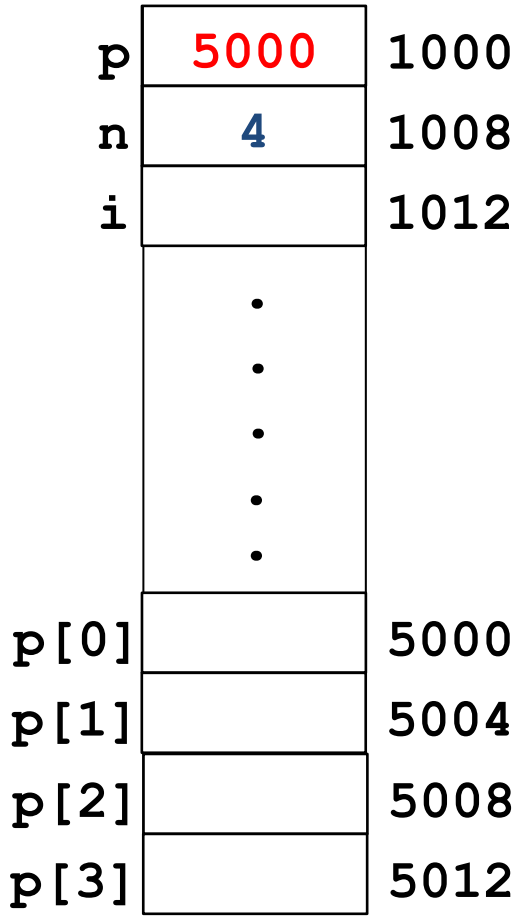
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
for( i = 0 ; i < n ; i++ ){



}
```

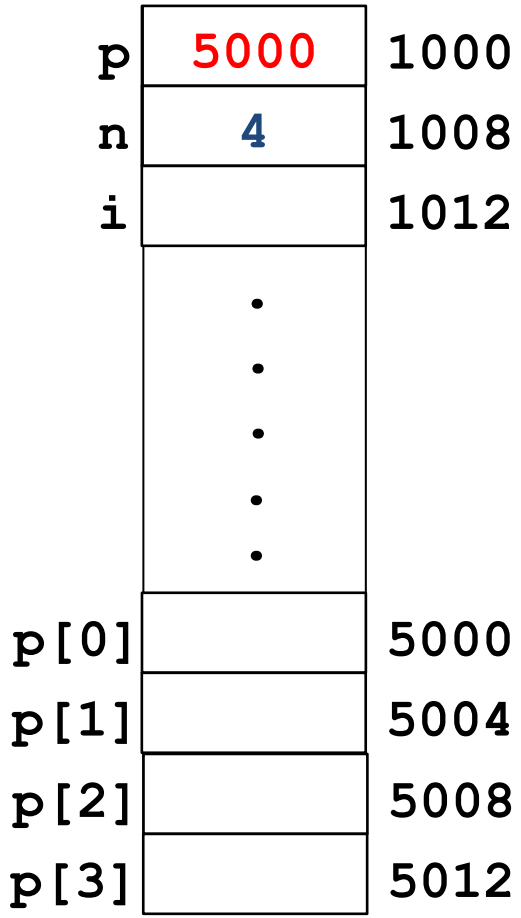
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
for( i = 0 ; i < n ; i++ ){
  printf("P[%d] = ", i);

}
```

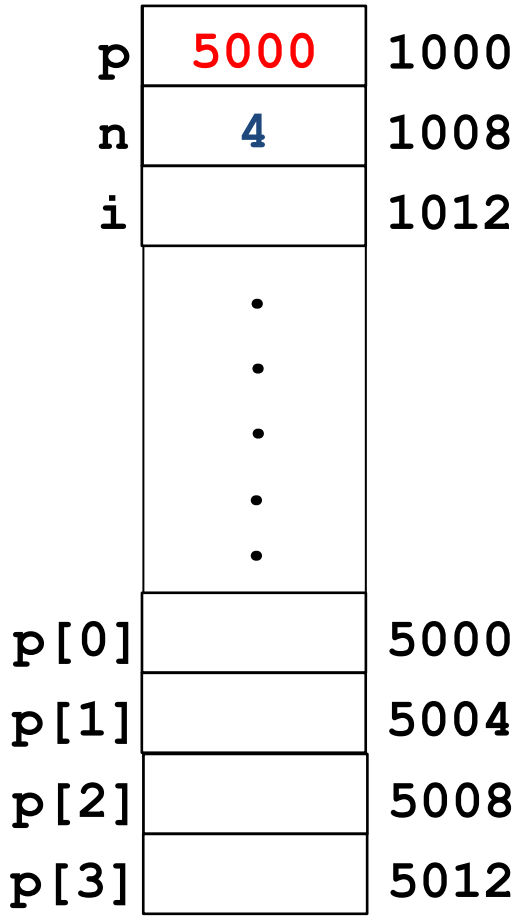
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
for( i = 0 ; i < n ; i++ ){
  printf("P[%d] = ", i);
  scanf("%d", p + i);
}
```

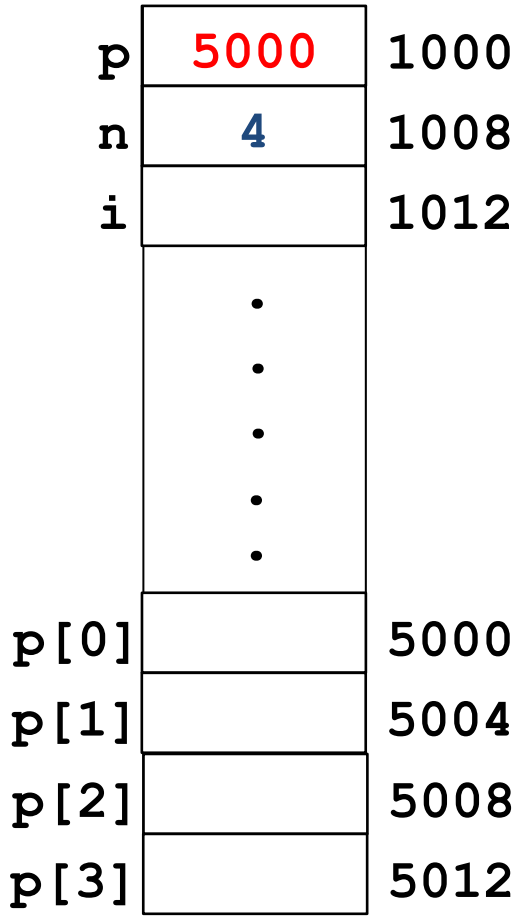
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
for( i = 0 ; i < n ; i++ ){
  printf("P[%d] = ", i);
  scanf("%d", p + i); // &p[i]
}
```

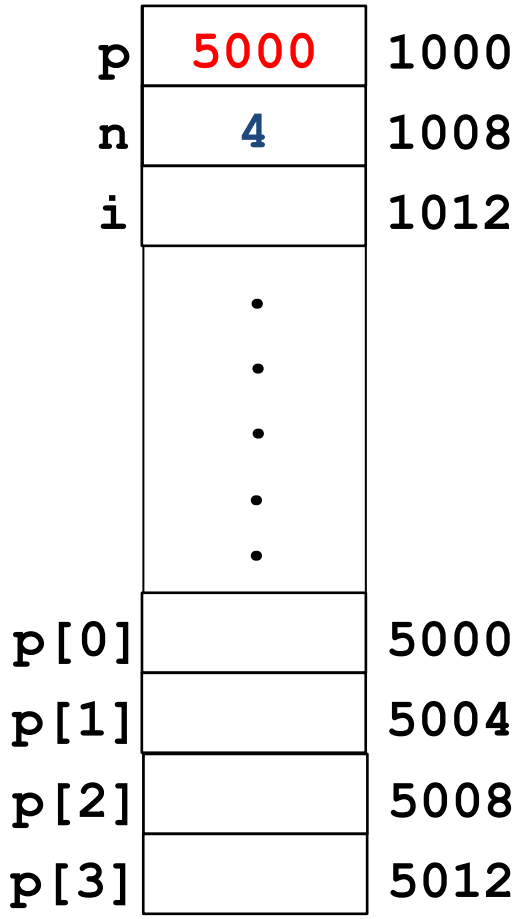
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
for( i = 0 ; i < n ; i++ ){
  printf("P[%d] = ", i);
  scanf("%d", p + i); // &p[i]
}
```

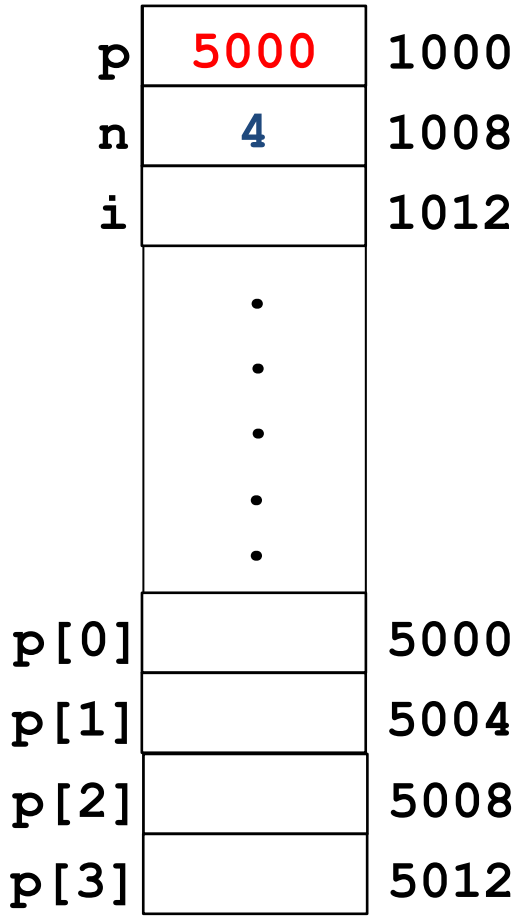
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
for( i = 0 ; i < n ; i++ ){
  printf("P[%d] = ", i);
  scanf("%d", p + i); // &p[i]
}

// Entrada de dados
```

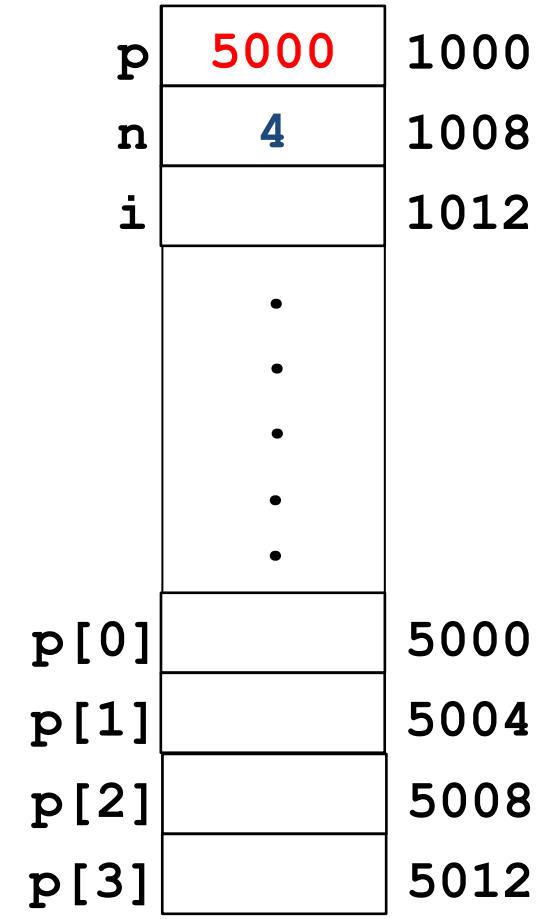
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
for( i = 0 ; i < n ; i++ ){
  printf("P[%d] = ", i);
  scanf("%d", p + i); // &p[i]
}
for( i = 0 ; i < n ; i++ )
  printf("P[%d] : %d\n", i, p[i]);
```

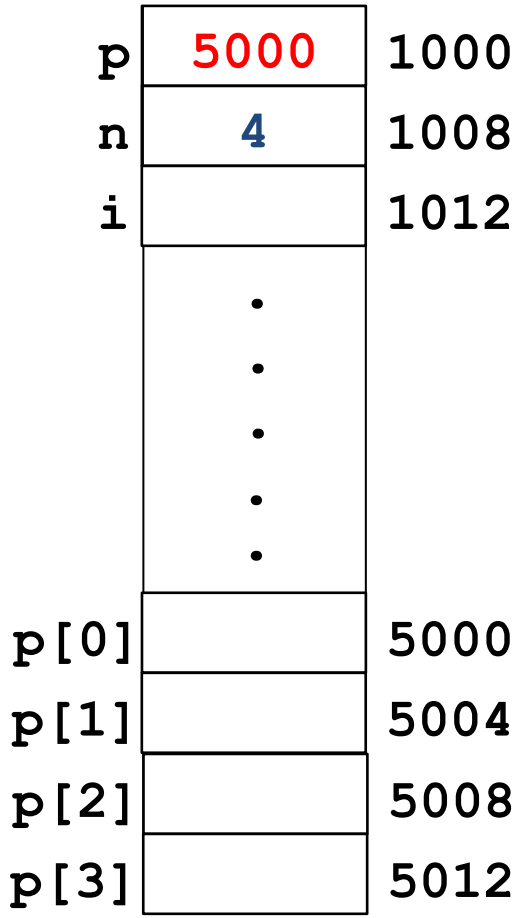
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
for( i = 0 ; i < n ; i++ ){
  printf("P[%d] = ", i);
  scanf("%d", p + i); // &p[i]
}
for( i = 0 ; i < n ; i++ )
  printf("P[%d] : %d\n", i, p[i]);

// Imprime os dados
```

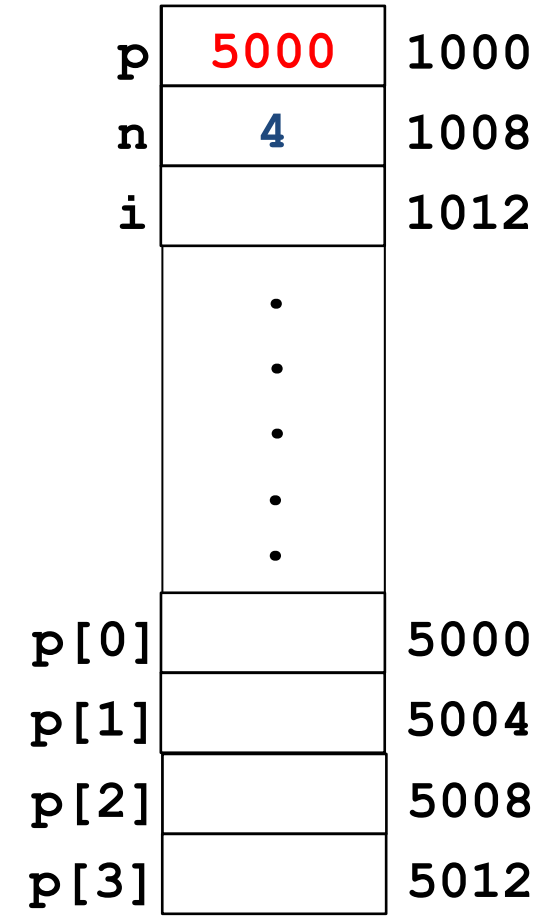
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
for( i = 0 ; i < n ; i++ ){
  printf("P[%d] = ", i);
  scanf("%d", p + i); // &p[i]
}
for( i = 0 ; i < n ; i++ )
  printf("P[%d] : %d\n", i, p[i]);

free(p); // libera a memória
```

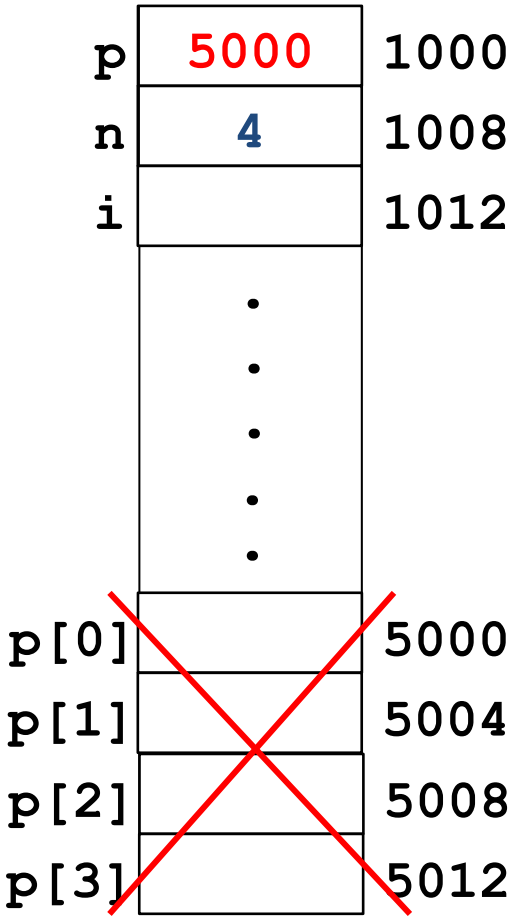
Modelo da Memória



Exemplo 2: alocando um vetor

```
int *p, n, i;
printf("Quantos valores? ");
scanf("%d", &n);
p = malloc( sizeof(int) * n );
for( i = 0 ; i < n ; i++ ){
  printf("P[%d] = ", i);
  scanf("%d", p + i); // &p[i]
}
for( i = 0 ; i < n ; i++ )
  printf("P[%d] : %d\n", i, p[i]);

free(p); // libera a memória
```

Modelo da Memória



EXEMPLO PRÁTICO: CONCATENAÇÃO DE STRINGS

Exemplos Práticos

- Concatenação de *strings*, gerando uma nova *string*;

- Função recebe duas *strings* e retorna uma *string* alocada dinamicamente:

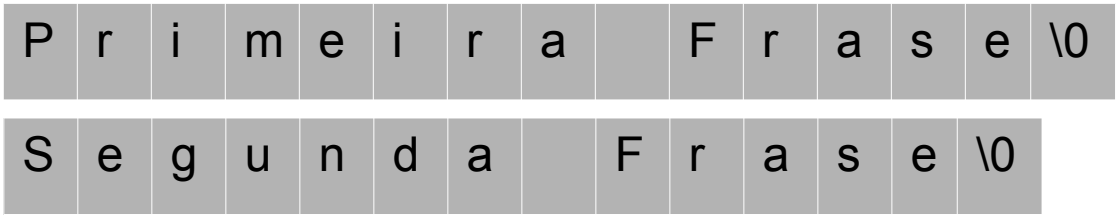
```
char * concatena( char *str1, char *str2 );
```

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

char * concatena( char *str1, char *str2 ) {
    int tamanho = strlen(str1)+strlen(str2)+1;
    char *str3 = (char*) malloc(tamanho);
    char *temp = str3+strlen(str1);
    strcpy(str3,str1);
    strcpy(temp,str2);
    return str3;
}

int main() {
    char frase1[] = "Primeira Frase";
    char frase2[] = "Segunda Frase";
    char *juntas = concatena(frase1, frase2);
    printf("%s\n", juntas);
    free(juntas);
    return 0;
}
```

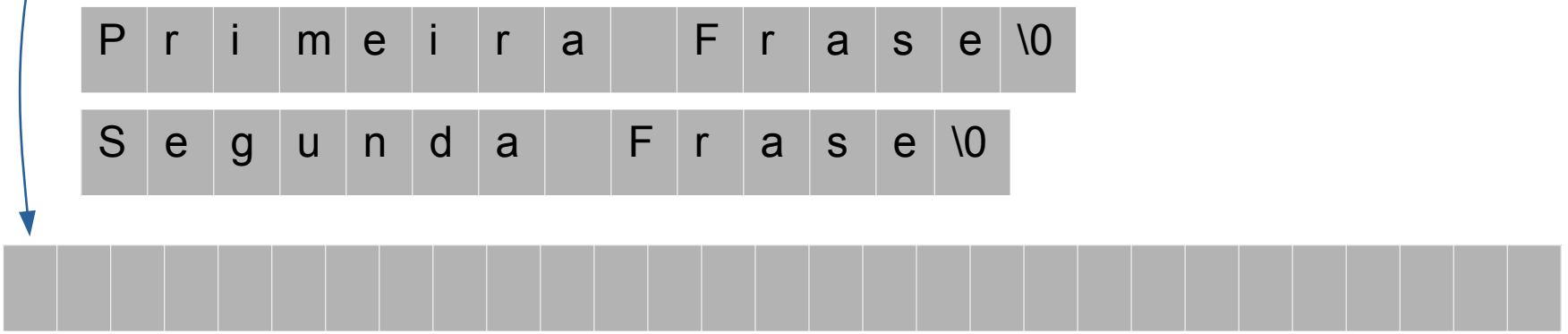
```
int main() {
    char frase1[] = "Primeira Frase";
    char frase2[] = "Segunda Frase";
    char *juntas = concatena(frase1, frase2);
    printf("%s\n", juntas);
    free(juntas);
    return 0;
}
```

```
char * concatena( char *str1, char *str2 ) {
    int tamanho = strlen(str1)+strlen(str2)+1;
    char *str3 = (char*) malloc(tamanho);
    char *temp = str3+strlen(str1);
    strcpy(str3,str1);
    strcpy(temp,str2);
    return str3;
}
```

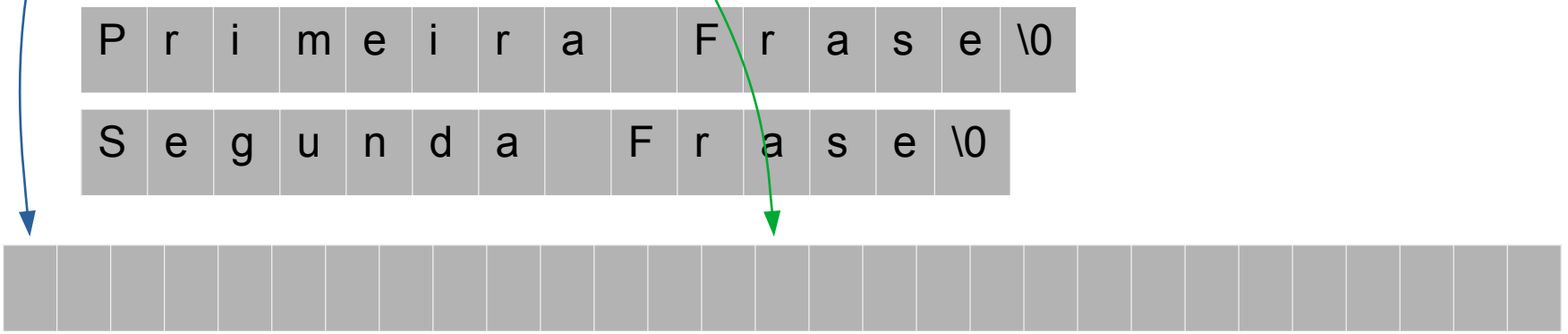
```
char * concatena( char *str1, char *str2 ) {
    int tamanho = strlen(str1)+strlen(str2)+1;
    char *str3 = (char*) malloc(tamanho);
    char *temp = str3+strlen(str1);
    strcpy(str3,str1);
    strcpy(temp,str2);
    return str3;
}
```



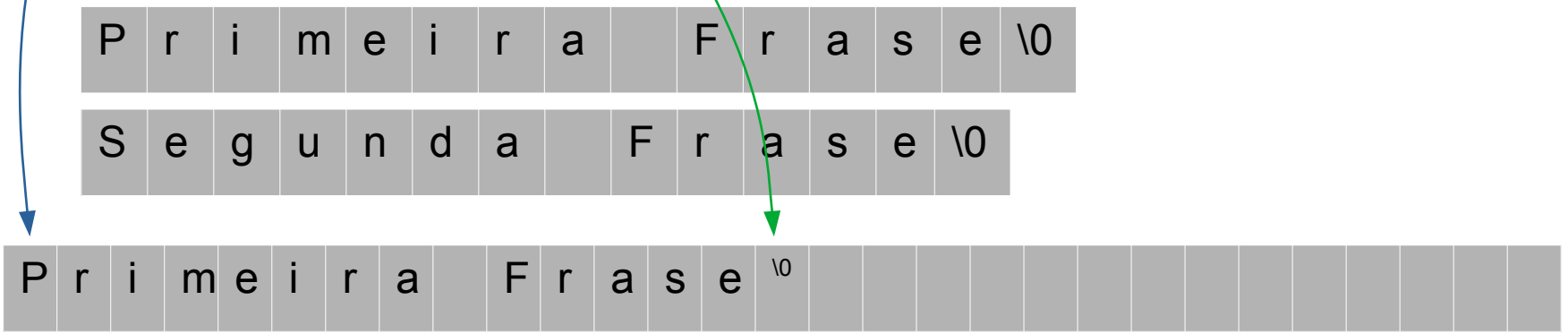
```
char * concatena( char *str1, char *str2 ) {
    int tamanho = strlen(str1)+strlen(str2)+1;
    char *str3 = (char*) malloc(tamanho);
    char *temp = str3+strlen(str1);
    strcpy(str3,str1);
    strcpy(temp,str2);
    return str3;
}
```



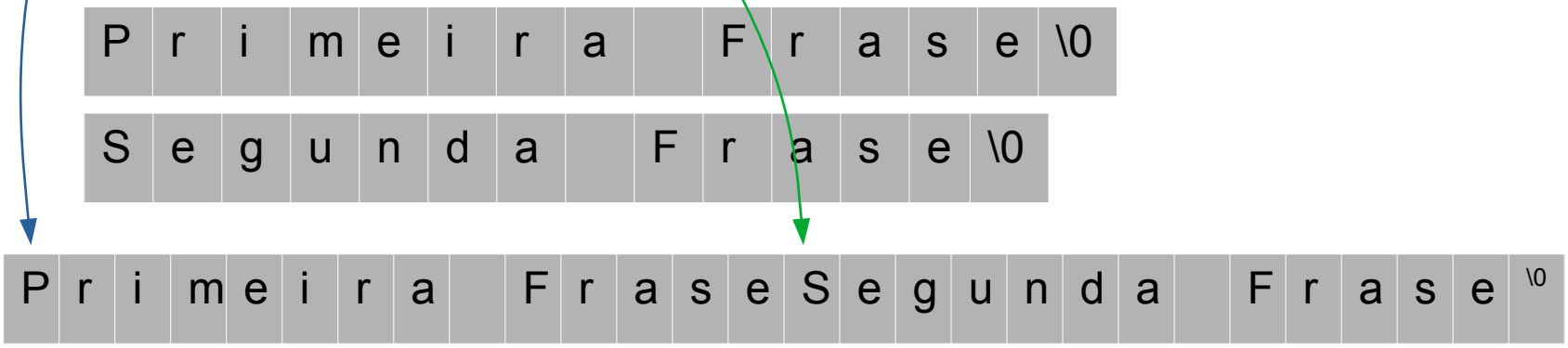
```
char * concatena( char *str1, char *str2 ) {
    int tamanho = strlen(str1)+strlen(str2)+1;
    char *str3 = (char*) malloc(tamanho);
    char *temp = str3+strlen(str1);
    strcpy(str3,str1);
    strcpy(temp,str2);
    return str3;
}
```



```
char * concatena( char *str1, char *str2 ) {
    int tamanho = strlen(str1)+strlen(str2)+1;
    char *str3 = (char*) malloc(tamanho);
    char *temp = str3+strlen(str1);
    strcpy(str3,str1);
    strcpy(temp,str2);
    return str3;
}
```



```
char * concatena( char *str1, char *str2 ) {
    int tamanho = strlen(str1)+strlen(str2)+1;
    char *str3 = (char*) malloc(tamanho);
    char *temp = str3+strlen(str1);
    strcpy(str3,str1);
    strcpy(temp,str2);
    return str3;
}
```



Exercícios

- Usando alocação dinâmica, faça um programa para ler as notas de uma turma e calcular a média e desvio padrão desta turma. A quantidade de alunos da turma é informado pelo usuário.
- Faça uma função que ordene um vetor inteiro de qualquer tamanho. Para testar a função, crie um programa que pergunte ao usuário o tamanho do vetor e gere o mesmo com valores aleatórios, mostre os valores gerados e depois o vetor ordenado.

Exercícios

- Construa um programa que aloque em tempo de execução (dinamicamente) uma matriz de ordem *m*x*n* (linha por coluna), usando 1+m chamadas a função malloc. Construa uma função que recebe os parametros m e n aloque uma matriz de ordem *m*x*n*, retornando um ponteiro para esta matriz alocada. Crie ainda uma função para liberar a área de memória alocada pela matriz. Crie então um novo programa que use as funções criadas.