UFRJ

INSTITUTO DE COMPUTAÇÃO UFRJ

Programação de Computadores II

Ponteiros

– Alocação dinâmica

Profa. Giseli Rabello Lopes

Sumário

- Relembrando...
- Ponteiros e funções (continuação)
- Alocação dinâmica de memória
- Exemplos
- Exercícios

Adaptado de material preparado por Profa. Silvana Rossetto e Prof. Adriano Joaquim de Oliveira Cruz

Relembrando...

- Argumentos na passagem de parâmetros por referência:
  - Se a variável for um vetor, seu nome corresponde ao endereço do seu primeiro elemento
  - Se a variável não for um vetor/matriz, então, caso se queira alterar a mesma, ela deve ser precedida de **&** ao se invocar a função

O operador seta (->)

- Outro operador associado a ponteiros é a seta (->)
  - Quando um ponteiro indica uma variável do tipo estrutura, para que se acesse os atributos desta variável, ao invés da combinação do operador '*' com o operador ponto '.' pode-se utilizar o operador ->

Exemplo de uso do operador seta (->)

```
#include <stdio.h>
typedef struct
{
    char nome [30];
    int matricula;
} tAluno;
int main ()
{
    tAluno *x;
    tAluno y;
    x = &y;
    (*x).matricula = 2031;
    // x->matricula = 2031; //forma alternativa
    printf ("%d\n", y.matricula);
    return 0;
}
```

Exercício

- Considere o tipo de dado **`tEmpregado`** definido ao lado para armazenar informações sobre um empregado. Implemente uma função para reajustar o salário de um dado empregado em 10%, caso ele tenha nascido antes de 1980, e de 7%, caso ele tenha nascido a partir de 1980.

```
typedef struct
{
    int dia, mes, ano;
} tData;

typedef struct
{
    char nome[40];
    tData dataNascimento,
dataAdmissao;
    char sexo;
    float salario;
} tEmpregado;
```

Exercício resolvido

```
#include <stdio.h>
//Definição das estruturas dadas na ordem do exercício

void reajuste_salarial (tEmpregado *emp)
{
        if (emp->dataNascimento.ano<1980)
                emp->salario=emp->salario*1.1;
        else
                emp->salario=emp->salario*1.07;
}
```

Outra alternativa para a função

```
void reajuste_salarial (tEmpregado *emp)
{
        if ((*emp).dataNascimento.ano<1980)
                (*emp).salario=(*emp).salario*1.1;
        else
                (*emp).salario=(*emp).salario*1.07;
}
```

```
int main()
{
	tEmpregado empregado;

	printf("Informe o nome do funcionario: ");
	gets(empregado.nome);

	printf("Informe a data de nascimento (dia/mes/ano): ");
	scanf("%d/%d/%d", &empregado.dataNascimento.dia,
&empregado.dataNascimento.mes, &empregado.dataNascimento.ano);

	printf("Informe a data de admissao (dia/mes/ano): ");
	scanf("%d/%d/%d", &empregado.dataAdmissao.dia,
&empregado.dataAdmissao.mes, &empregado.dataAdmissao.ano);

	printf("Informe o sexo: ");
	getchar();
	empregado.sexo=getchar();

	printf("Informe o salario: ");
	scanf("%f",&empregado.salario);

	reajuste_salarial(&empregado);
	printf("--> Salario reajustado:
%.2f\n",empregado.salario);
	return 0;
}
```

```
//Exemplo incluindo entrada de dados em função separada

void ler_dados(tEmpregado * saida)
{
	printf("Digite o nome do empregado: ");
	gets(saida->nome);
	printf("Digite a data de nascimento (D/M/A): ");
	scanf("%d/%d/%d", &saida->dataNascimento.dia, &saida-
>dataNascimento.mes, &saida->dataNascimento.ano);
	printf("Digite a data de admissao (D/M/A): ");
	scanf("%d/%d/%d", &saida->dataAdmissao.dia, &saida-
>dataAdmissao.mes, &saida->dataAdmissao.ano);
	getchar();
	printf("Digite o sexo: ");
	scanf("%c", &saida->sexo);
	printf("Digite o salario: ");
	scanf("%f", &saida->salario);
}

int main()
{
	tEmpregado empregado;
	ler_dados(&empregado);
	reajuste_salarial(&empregado);
	printf("Novo salario: %.2f\n", empregado.salario);
	return 0;
}
```

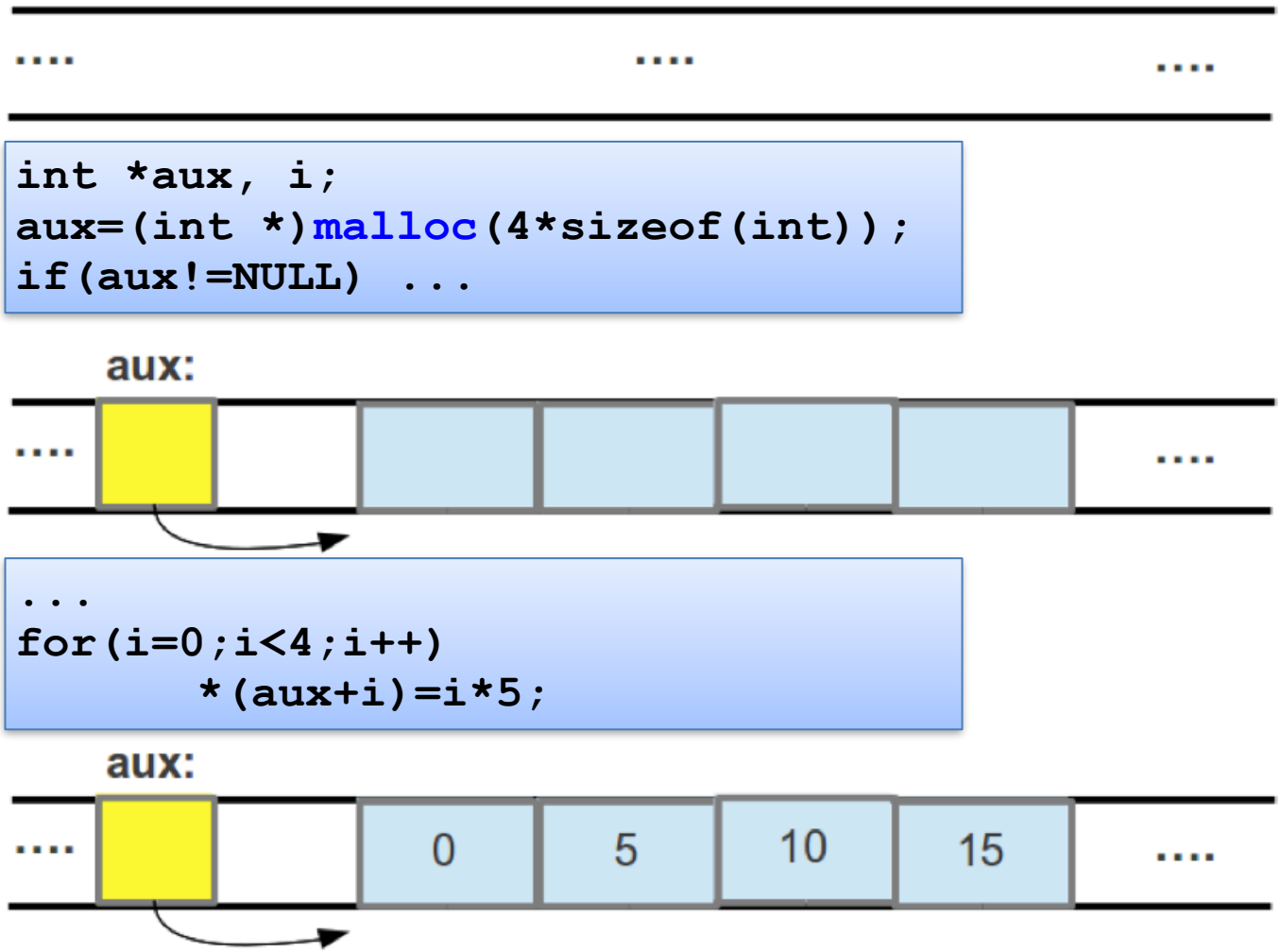
Alocação dinâmica de memória

- Ponteiros permitem **gerenciar a alocação de áreas de memória durante a execução do programa**
- Neste esquema, o programador pode reservar o número exato de posições que o programa requer
- Para isso é preciso usar funções existentes na biblioteca **`stdlib.h`**:
  - Pedem ao sistema operacional para separar pedaços da memória e devolvem ao programa que pediu o endereço inicial deste local

Funções da biblioteca **`stdlib.h`** para alocação de memória

- **`void *malloc(size_t size)`**: reserva **`size`** bytes de espaço na memória (não inicializa espaço alocado), retorna um ponteiro para o espaço reservado ou **`NULL`** no caso de algum erro ocorrer
- **`void *calloc(size_t num, size_t size)`**: reserva espaço na memória para um vetor de **`num`** itens, cada item tem tamanho **`size`** e todos os bits do espaço são inicializados com **`0`**, retorna um ponteiro para o espaço reservado ou **`NULL`** no caso de algum erro ocorrer
- **`void free(void *pont)`**: libera o espaço de memória apontado por **`pont`**

Exemplo de uso da função `malloc()`



Exemplo de uso de **calloc** e **free**

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
        float *v;
        int i, tam;
        printf ("Qual o tamanho do vetor? ");
        scanf ("%d" , &tam);
        v = (float *) calloc (tam, sizeof (float));
        if (!v)
        {
                printf ("Erro alocacao memoria");
                return -1;
        }
        for (i=0; i<tam; i++)
        {
                printf ("Elemento %d? " , i) ;
                scanf ("%f", v+i);
        }
        for (i=0; i<tam; i++)
                printf ("v[%d]=%.2f\n",i,*(v+i));
        free (v);
        return 0;
}
```

Vetores de ponteiros

- Uma vez que ponteiros são variáveis, podemos criar **vetores de ponteiros**!
  (permite representar estruturas de dados bidimensionais)

- Exemplos:

```
char *pLinha[N]; //vetor de N ponteiros de caracteres
int *notas[4]; //vetor de 4 ponteiros para inteiros
```

Exemplo de uso de **malloc** e **free**

– Vetor de ponteiros

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main ()
{
        int tam,i;
        char *g[3];
        tam = sizeof(char);
        g[0] = (char *)malloc(13*tam);
        strcpy(g[0], "Computacao I");
        g[1] = (char *)malloc(23*tam);
        strcpy(g[1], "Sistemas de Informacao");
        g[2] = (char *)malloc(14*tam);
        strcpy(g[2], "Computacao II");
        for (i=0;i<3;i++)
                free(g[i]);
        return 0;
}
```

Funções da biblioteca **`stdlib.h`** para alocação de memória (cont.)

- **`void realloc(void *pont, size_t size)`**: altera o tamanho do objeto na memória apontado por **`pont`** para o tamanho especificado por **`size`**; o conteúdo do objeto será mantido até um tamanho igual ao menor dos dois tamanhos, novo e antigo; se o novo tamanho requerer movimento, o espaço reservado anteriormente é liberado; caso o novo tamanho for maior, o conteúdo da porção de memória reservada a mais ficará com um valor sem especificação; se o tamanho **`size`** for igual a 0 e **`pont`** não é um ponteiro nulo o objeto previamente reservado é liberado