Processamento de Dados II
Prof. Max Davi

Ponteiros e vetores

Vetores como ponteiros

Quando você declara uma matriz da seguinte forma:

tipo_da_variável nome_da_variável [tam1][tam2] ... [tamN];

o compilador C calcula o tamanho, em bytes, necessário para armazenar esta matriz. Este tamanho é:

tam1 x tam2 x tam3 x ... x tamN x tamanho_do_tipo

O compilador então aloca este número de bytes em um espaço livre de memória. O nome da variável que você declarou é na verdade um ponteiro para o tipo da variável da matriz.

Vetores como ponteiros

Qual motivo de usar a seguinte notação?

nome_da_variável[índice]

Isto pode ser facilmente explicado desde que você entenda que a notação acima é *absolutamente equivalente* a se fazer:

***(nome_da_variável+índice)**

Vetores como ponteiros

Fica claro, por exemplo, porque é que, no C, a indexação começa com zero. É porque, ao pegarmos o valor do primeiro elemento de um vetor, queremos, de fato, *nome_da_variável e então devemos ter um índice igual a zero. Então sabemos que:

nome_da_variável* é equivalente a *nome_da_variável[0]

Vetores como ponteiros

Vamos ver agora um dos usos mais importantes dos ponteiros: a varredura sequencial de uma matriz. Quando temos que varrer todos os elementos de uma matriz de uma forma sequencial, podemos usar um ponteiro, o qual vamos incrementando. Qual a vantagem? Considere os programas a seguir para zerar uma matriz:

```
#include <stdlib.h>
#include <stdio.h>
float matrx [50][50];
float *p;
int count;
int main ()
{
p=matrx[0];
for (count=0;count<2500;count++)
     {
     *p=0.0;
     p++;
     }
return(0);
}
```

Vetores como ponteiros

No programa o único cálculo que deve ser feito é o de um incremento de ponteiro. Fazer 2500 incrementos em um ponteiro é muito mais rápido que calcular 2500 deslocamentos completos.

IMPORTANTE:

Há uma diferença entre o nome de um vetor e um ponteiro que deve ser frisada: um ponteiro é uma variável, mas o nome de um vetor não é uma variável. Isto significa, que não se consegue alterar o endereço que é apontado pelo "nome do vetor".

Vetores como ponteiros

Exemplo:

```
int vetor[10];
int *ponteiro, i;
ponteiro = &i;

/* as operações a seguir são invalidas */

vetor = vetor + 2; /* ERRADO: vetor não é variável */
vetor++; /* ERRADO: vetor não é variável */
vetor = ponteiro; /* ERRADO: vetor não é variável */
```

Vetores como ponteiros

```
/* as operações abaixo são validas */

ponteiro = vetor; /* CERTO: ponteiro é variável */
ponteiro = vetor+2; /* CERTO: ponteiro é variável */
```

Ponteiros como vetores

Sabemos agora que, na verdade, o nome de um vetor é um ponteiro constante. Sabemos também que podemos indexar o nome de um vetor. Como consequência podemos também indexar um ponteiro qualquer.

Ponteiros como vetores

Exemplo:

```
#include <stdio.h>
int matrx [10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int *p;
int main ()
{
    p=matrx;
    printf ("O terceiro elemento do vetor e:
%d",p[2]);
    return(0);
}
```

Obs: Podemos ver que p[2] equivale a *(p+2).

Endereços de elementos de vetores

Nesta seção vamos apenas ressaltar que a notação

&nome_da_variável[índice]

é válida e retorna o endereço do ponto do vetor indexado por índice. Isto seria equivalente a nome_da_variável + indice. É interessante notar que, como consequência, o ponteiro nome_da_variável tem o endereço &nome_da_variável[0], que indica onde na memória está guardado o valor do primeiro elemento do vetor.

Vetores de ponteiros

Podemos construir vetores de ponteiros como declaramos vetores de qualquer outro tipo. Uma declaração de um vetor de ponteiros inteiros poderia ser:

int *pmatrx [10];

No caso acima, pmatrx é um vetor que armazena 10 ponteiros para inteiros.

Ponteiros com Strings

Toda string que o programador insere no programa é colocada num banco de strings que o compilador cria. No local onde está uma string no programa, o compilador coloca o endereço do início daquela string (que está no banco de strings).

Para uma string que vamos usar várias vezes, podemos fazer:

```
char *str1="String constante.";
```

Aí poderíamos, em todo lugar que precisarmos da string, usar a variável str1.

Alocação dinâmica de memória

Em muitas aplicações, a quantidade de memória a alocar só se torna conhecida durante a execução do programa. Para lidar com essa situação é preciso recorrer à alocação dinâmica de memória. A alocação dinâmica é administrada pelas funções malloc, realloc e free, que estão na biblioteca stdlib. Para usar essa biblioteca, inclua a correspondente interface no seu programa:

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

Função Malloc

A função malloc (o nome é uma abreviatura de memory allocation) aloca espaço para um bloco de bytes consecutivos na memória RAM (random access memory) do computador e devolve o endereço desse bloco. O número de bytes é especificado no argumento da função. No seguinte fragmento de código, malloc aloca 1 byte:

```
char *ptr;
ptr = malloc (1);
scanf ("%c", ptr);
```

Função Malloc

O endereço devolvido por malloc é do tipo genérico void *. O programador armazena esse endereço num ponteiro de tipo apropriado. No exemplo anterior, o endereço é armazenado no ponteiro ptr, que é do tipo ponteiro-para-char. A transformação do ponteiro genérico em ponteiro-para-char é automática; não é necessário escrever ptr = (char *) malloc (1);.

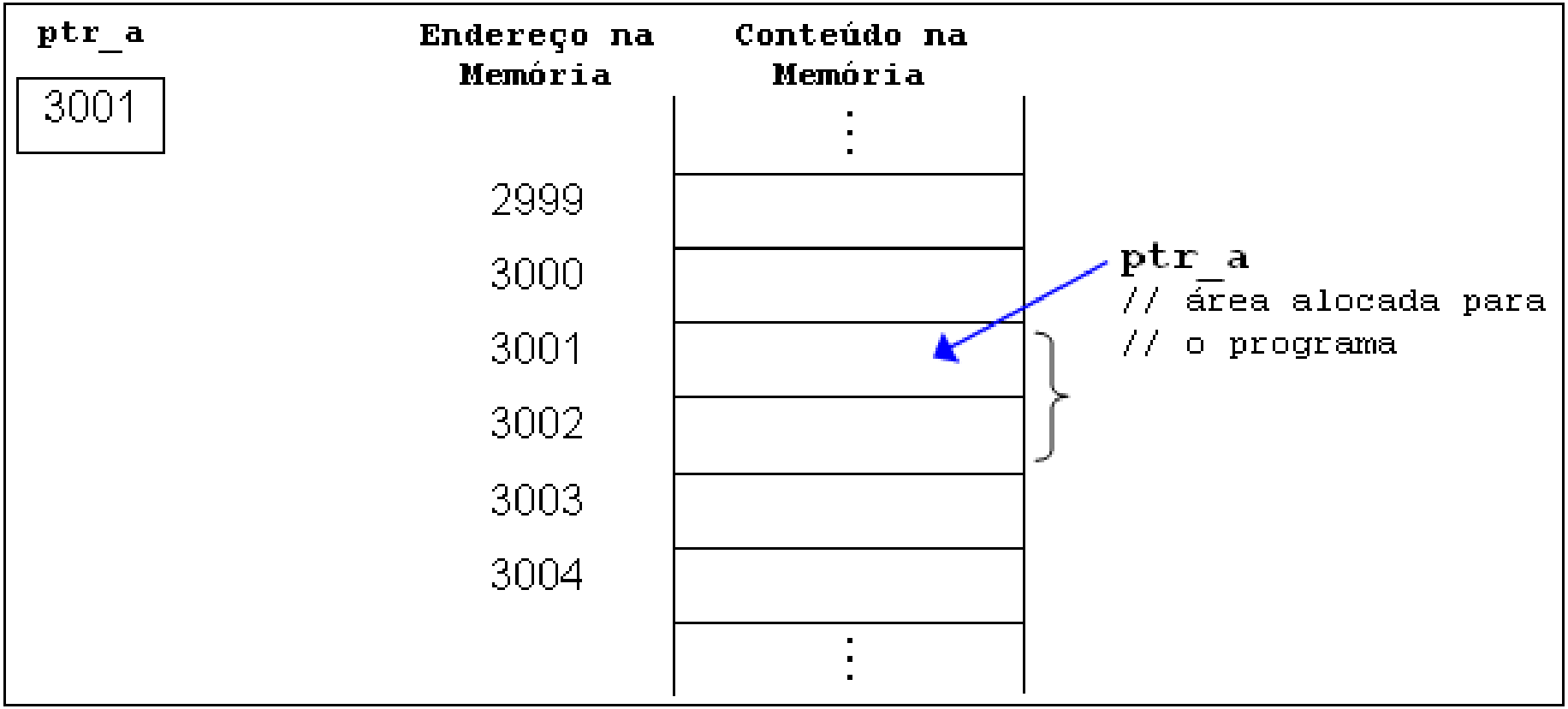
```
#include <stdio.h>
#include <stdlib.h> //necessário para usar as funções malloc() e free()
#include <conio.h>
float *v; //definindo o ponteiro v
int i, num_componentes;
int main(void) {
  printf("Informe o numero de componentes do vetor\n");
  scanf("%d", &num_componentes);
  v = (float *) malloc(num_componentes * sizeof(float));
  //Armazenando os dados em um vetor
  for (i = 0; i < num_componentes; i++) {
      printf("\nDigite o valor para a posicao %d do vetor: ", i+1);
      scanf("%f",&v[i]);
  }
  // ------ Percorrendo o vetor e imprimindo os valores ----------
  printf("\n********** Valores do vetor dinamico **********\n\n");
  for (i = 0;i < num_componentes; i++){
      printf("%.2f\n",v[i]);
  }
  //liberando o espaço de memória alocado
  free(v);
  return 0;
}
```

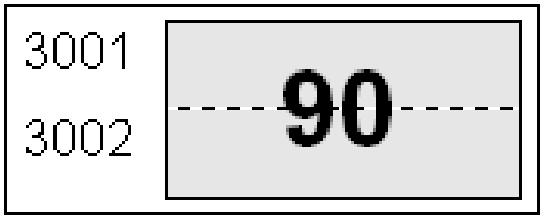
Função Malloc

- **Calcula o número de bytes necessários primeiramente, multiplicando o número de componentes do vetor pela quantidade de bytes que é dada pelo comando sizeof, portanto temos: num_componentes * sizeof(float)**
- **Reservada a quantidade de memória, para isso usamos malloc para reservar essa quantidade de memória, então temos: malloc(num_componentes * sizeof(float))**
- **Convertemos o ponteiro para o tipo de dados desejado. Como a função malloc retorna um ponteiro do tipo void, precisamos converter esse ponteiro para o tipo da nossa variável, no caso float, por isso usamos o comando de conversão explicita: (float *)**
- **juntando tudo e atribuindo em v temos o comando: v = (float *) malloc(num_componentes * sizeof(float));**

Função Free

- As variáveis alocadas estaticamente dentro de uma função, também conhecidas como variáveis automáticas ou locais, desaparecem assim que a execução da função termina. Já as variáveis alocadas dinamicamente continuam a existir mesmo depois que a execução da função termina. Se for necessário liberar a memória ocupada por essas variáveis, é preciso recorrer à função free.
- A função free desaloca a porção de memória alocada por malloc. A instrução free (ptr) avisa ao sistema que o bloco de bytes apontado por ptr está disponível para reciclagem. A próxima invocação de malloc poderá tomar posse desses bytes.


ptr_a
3001
Endereço na Memória
Conteúdo na Memória
2999
3000
3001
3002
3003
3004
ptr_a
// área alocada para
// o programa


3001
3002
90

Alocação de vetores

No exemplos anterior, foi alocado um espaço para armazenar apenas um dado. Entretanto, é mais comum utilizar ponteiros para alocação de vetores. Para tanto, basta especificar o tamanho desse vetor no momento da alocação.

No exemplo seguinte, apresenta-se a alocação de vetores com malloc. Após a alocação de uma área com vários elementos, ela pode ser acessada exatamente como se fosse um vetor.

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
    int i;
    int *v;
    v = (int*)malloc(sizeof(int)*10);  /* 'v' é um ponteiro para uma área que tem 10 inteiros. 'v' funciona exatamente como um vetor*/
    v[0] = 10;
    v[1] = 11;
    v[2] = 12;
    v[3] = 13;
    v[4] = 14;
    v[5] = 15;
    v[6] = 16;
    v[7] = 17;
    v[8] = 18;
    v[9] = 19;
```

```
    for(i = 0; i < 10; i++)
        printf("v[%d]: %d\n", i, v[i]);
    printf("Endereço de 'v': %p", v);  // imprime o endereço da área alocada para ‘v’
    free(v);
    return 0;
}
```

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
	int *vet;
	int *ptr;
	vet = (int*)malloc(sizeof(int)*10);
	ptr = vet;  /* 'ptr'  aponta para o início da área alocada por
‘vet */
	*ptr = 11;  // vet[0] = 11 /* coloca 11 na primeira posição da
área alocada */
	ptr++;	// avança o apontador
	*ptr = 12;  // vet[1] = 12
	printf("%p\n",  ptr);
	printf("%d\n", *ptr);
	ptr--;
	printf("%p\n",  ptr);
	printf("%d\n", *ptr);
	free(ptr);
	return 0;
}
```

Atividade Final

a) Faça um programa que leia o tamanho de um vetor de inteiros e reserve dinamicamente memória para esse vetor. Em seguida, leia os elementos desse vetor, imprima o vetor lido e mostre o resultado da soma dos números ímpares presentes no vetor.

Referências Bibliográficas

- Deitel H and Deitel P. - C: Como Programar, 6 edição, Pearson;

- Schildt H. - C Completo e Total – Makron Books;

- Ana Fernanda Gomes Ascencio e Edilene Aparecida Veneruchi de Campos - Fundamentos da Programação de Computadores: Algoritmos, Pascal, C, C++ e Java.