ESTRUTURA DE DADOS

Prof. Nilton

Aula de hoje

- Alocação estática e dinâmica
- Introdução a ponteiros
- Exemplos
- Exercícios

Endereços de memória

A memória RAM de qualquer computador é uma sequência de bytes. Cada byte armazena um de 256 possíveis valores. Os bytes são numerados sequencialmente e o número de um byte é o seu endereço (= address).

Cada objeto na memória do computador ocupa um certo número de bytes consecutivos. Um char ocupa 1 byte. Um int ocupa 4 bytes e um double ocupa 8 bytes em muitos computadores. O número exato de bytes de um objeto é dado pelo operador sizeof: a expressão sizeof (int), por exemplo, dá o número de bytes de um int no seu computador.

Cada objeto na memória tem um endereço. Na maioria dos computadores, o endereço de um objeto é o endereço do seu primeiro byte. Por exemplo, depois das declarações

Endereços de memória

```
char c;
int i;
struct {
      int x, y;
} ponto;
 int v[3];
```

os endereços das variáveis poderiam ser os seguintes:

```
c       89421
i       89422
ponto   89426
v[0]    89434
v[1]    89438
v[2]    89442
v[3]    89446
```

Endereços de memória

O endereço de um objeto (como uma variável, por exemplo) é dado pelo operador &. Se i é uma variável então

&i

é o seu endereço. (Não confunda esse uso de & com o operador lógico and, que se escreve && em C.) No exemplo acima, &i vale 89422 e &v[3] vale 89446.

Um exemplo: O segundo argumento da função da biblioteca scanf é o endereço da variável onde deve ser depositado o objeto lido do dispositivo padrão de entrada:

```
int i;
scanf ("%d", &i);
```

Exercício 1

Tamanhos. Compile e execute o seguinte programa:

```
int main (void) {
   typedef struct {
      int dia, mes, ano;
   } data;
   printf ("sizeof (data) = %d\n", sizeof (data));
   cout << "sizeof(data) = \n" << sizeof(data);
}
```

Alocação de espaço

Alocação estática

Até agora, todas as estruturas que utilizamos tinham tamanho predefinido, ou seja, os valores sempre foram definidos na construção do programa ou em tempo de compilação. Esse tipo de alocação pode implicar desperdício de recursos, já que nem sempre é possível determinar qual o espaço necessário para armazenar todas as informações que o programa irá utilizar ou subestimar seu tamanho.

Chamamos esse tipo de alocação de estática.

Alocação de espaço

Alocação dinâmica

Na alocação dinâmica, os espaços são alocados durante a execução do programa. Isso significa que a memória será alocada conforme a necessidade do programa, não havendo reserva antecipada de espaço.

Com essa abordagem, resolvemos o principal problema da alocação estática: o dimensionamento de espaço, assim não subestimamos e não superestimamos o espaço necessário.

Introdução a Ponteiros

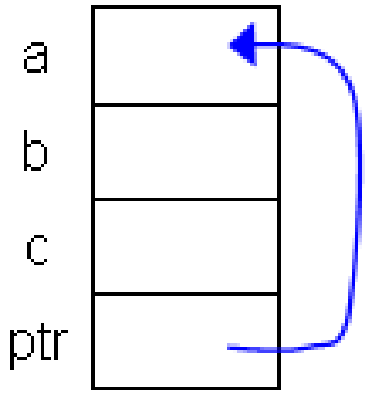
Ponteiros (ou apontadores) são variáveis que armazenam um **endereço de memória**. Portanto, podemos considerar endereços de memória como um tipo de dados.

Quando utilizamos outros tipos de variáveis, trabalhamos com o **conteúdo de um endereço de memória**.

Com os ponteiros, acessamos **o próprio endereço** da memória e não seu conteúdo.

Um ponteiro é uma variável capaz de armazenar um endereço de memória ou o endereço de outra variável.

Introdução a Ponteiros




```
#include <stdio.h>
void main()
{
  int a;
  int b;
  int c;
  int *ptr;  // declara um ponteiro para um inteiro
             // um ponteiro para uma variável do tipo inteiro
  a = 90;
  b = 2;
  c = 3;
  ptr = &a;
  cout << "Valor de ptr: " << ptr << " Conteúdo de ptr: " << *ptr;
  cout << "B: " << b << " C: " << c;
}
```

Imprimindo um ponteiro

```
#include <iostream>
using namespace std;
void main()
{
  int x;
  int *ptr;
  ptr = &x;
  cout << "O endereço de X é: " << ptr << endl;
}
```

Acessando o Conteúdo de uma Posição de Memória através de um Ponteiro

Usa-se o operador de **referência** (*):

```
#include <iostream>
using namespace std;
void main()
{
  int x;
  int *ptr;
  x = 5;
  ptr = &x;
  cout << "O valor da variável X é: " << *ptr << endl;
  *ptr = 10;
  cout << "Agora, X vale: " << *ptr << endl;
}
```

Alocação Dinâmica de Memória

Durante a execução de um programa, pode-se alocar dinamicamente memória para usar como variáveis do programa. Em C, a alocação é feita com a função **malloc** (e, para liberar memória, usa-se a função **free**). Em C++, faz-se o mesmo tipo de alocação usando o operador **new** (para liberar memória, operador **delete**). No exemplo a seguir, pode-se observar o uso das funções **malloc** e **free**:

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

void main()
{
 int *ptr_a;
 ptr_a = malloc(sizeof(int));
 // cria a área necessária para 01 inteiro e
 // coloca em 'ptr_a' o endereço desta área.
```

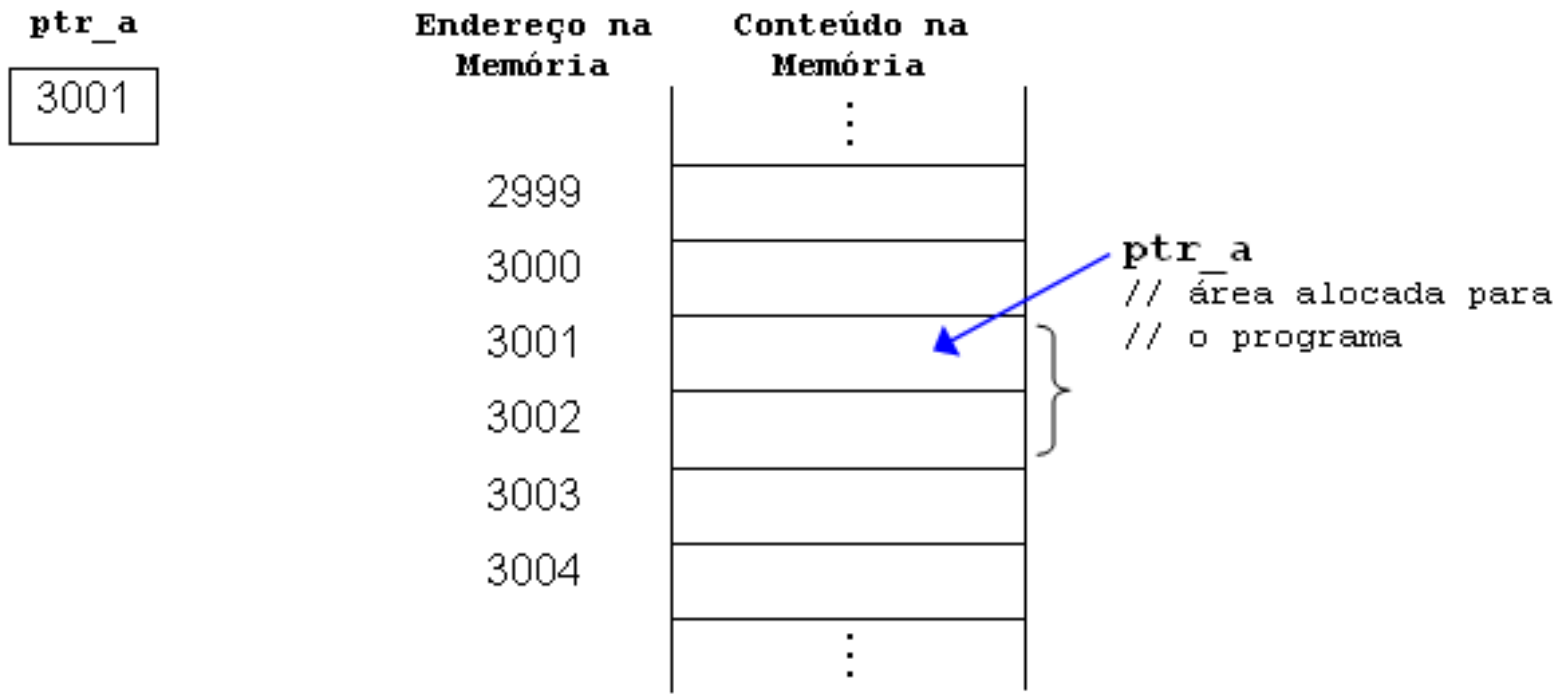
```c
 if (ptr_a == NULL)
 {
   printf("Memória insuficiente!\n");
   exit(1);
 }

  printf("Endereço de ptr_a: %p\n",  ptr_a);
  *ptr_a = 90;
  printf("Conteúdo de ptr_a: %d\n", *ptr_a);  //
imprime 90
  free(ptr_a);  // Libera a área alocada
}
```

Alocação dinâmica de memória

```
#include <iostream>

using namespace std;

void main()

{

 int *ptr_a;

 ptr_a = new int;

 // cria a área necessária para 01 inteiro e

 // coloca em 'ptr_a' o endereço desta área.
```

```
 if (ptr_a == NULL)
  {
    cout << "Memória insuficiente!" << endl;
    exit(1);
  }
  cout << "Endereço de ptr_a: " <<  ptr_a << endl;
  *ptr_a = 90;
  cout << "Conteúdo de ptr_a: " << *ptr_a << endl;
  delete ptr_a;
}
```

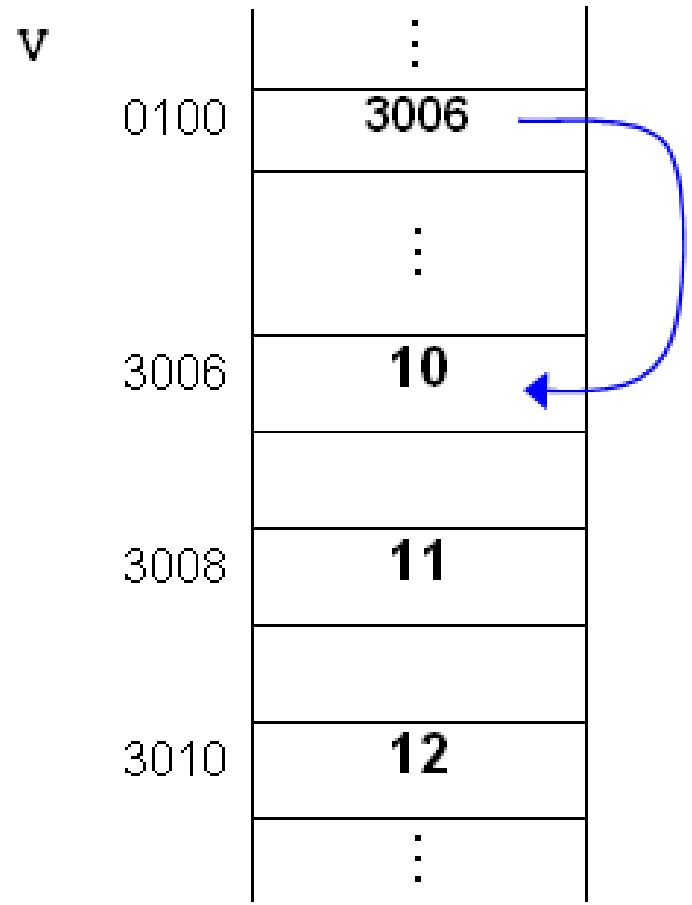
Alocação dinâmica de memória



Alocação de vetores

Nos exemplos anteriores, foi alocado um espaço para armazenar apenas um dado do tipo int. Entretanto, é mais comum utilizar ponteiros para alocação de vetores. Para tanto, basta especificar o tamanho desse vetor no momento da alocação. Nos exemplos abaixo, apresenta-se a alocação de vetores com malloc e new. Após a alocação de uma área com vários elementos, ela pode ser acessada exatamente como se fosse um vetor.

Alocação de vetores



Exemplo em C

```
#include <stdlib.h>
void main()
{
  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;
  // continua...
  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);
}
```

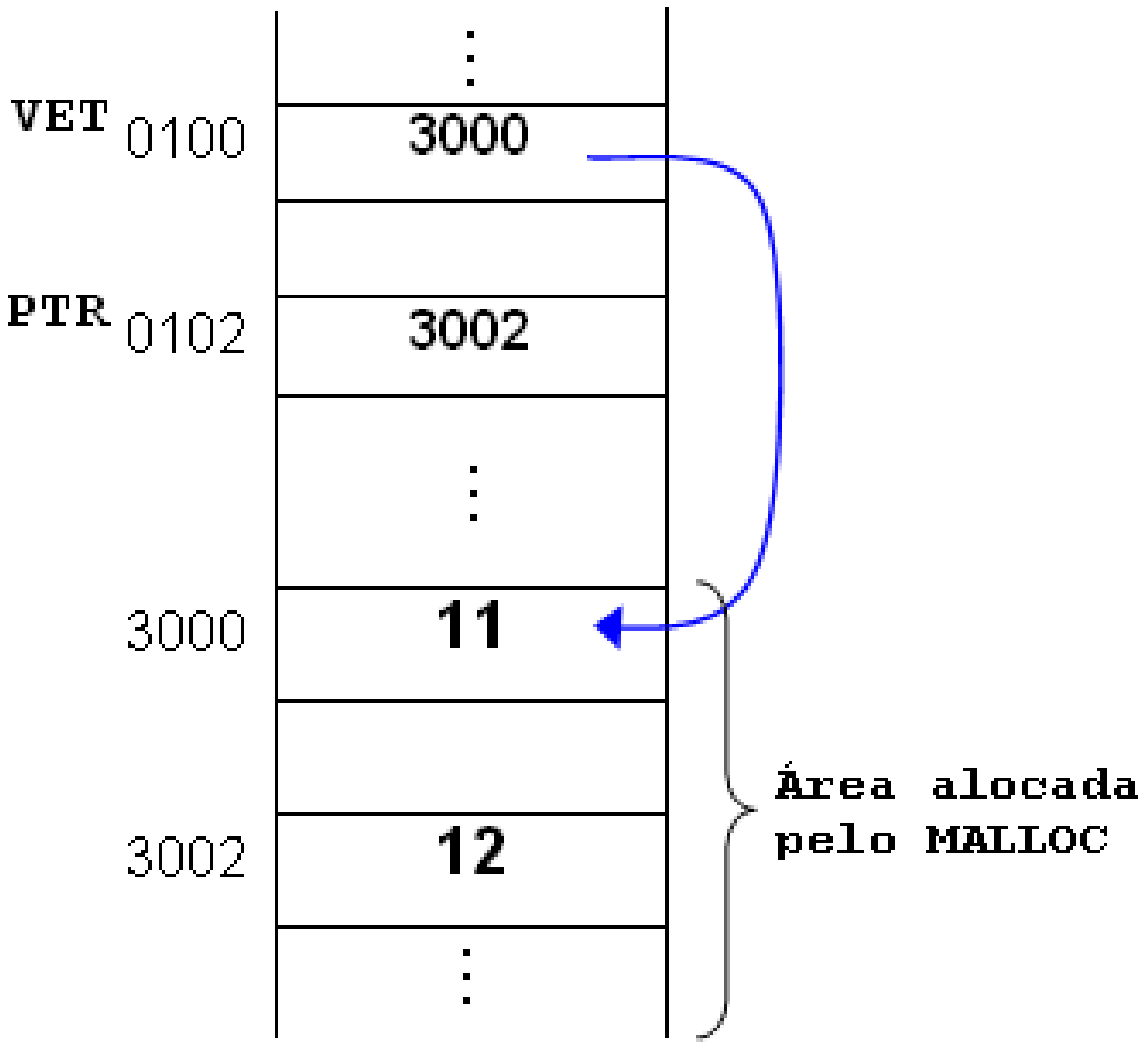
Exemplo em C++

```
#include <iostream>
using namespace std;
void main()
{
 int i;
 int *v;
 v = new 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;
 // continua...
 v[9] = 19;
  for(i = 0; i < 10; i++)
    cout << "v[" << i << "]: " << v[i] << endl;

  cout << "Endereço de 'v': " << v << endl; // imprime o endereço da área alocada para 'v'
  delete[] v;
}
```

Além de acessar os dados de uma área alocada dinâmicamente como se fosse um vetor, é possível acessá-los através do próprio ponteiro, utilizando a técnica chamada aritmética de ponteiros.

```
#include <stdlib.h>
void main()
{
  int *vet;
  int *ptr;

  vet = (int*)malloc(sizeof(int)*10);
  vet[4] = 44;  // 'vet' funciona como um vetor, depois de malloc

  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);

  // free(ptr);  // Liberar 'ptr' direto causa NULL POINTER ASSIGNMENT
  // Corrigindo, a forma correta é:
  ptr--;
  free(ptr);
}
```


VET 0100
3000
PTR 0102
3002
3000
11
3002
12
Área alocada
pelo MALLOC

Passando ponteiro por referencia

```
int main(){
  int a, *p;
  a = 100;
  p = &a;

  imprimir(p);
}
```

```
void imprimir(int *p){
  printf("Valor de p %d", *p);
}
```

Obrigado!!!