Programação em computadores II

Ponteiros

Prof. Dr. Fábio Rodrigues de la Rocha

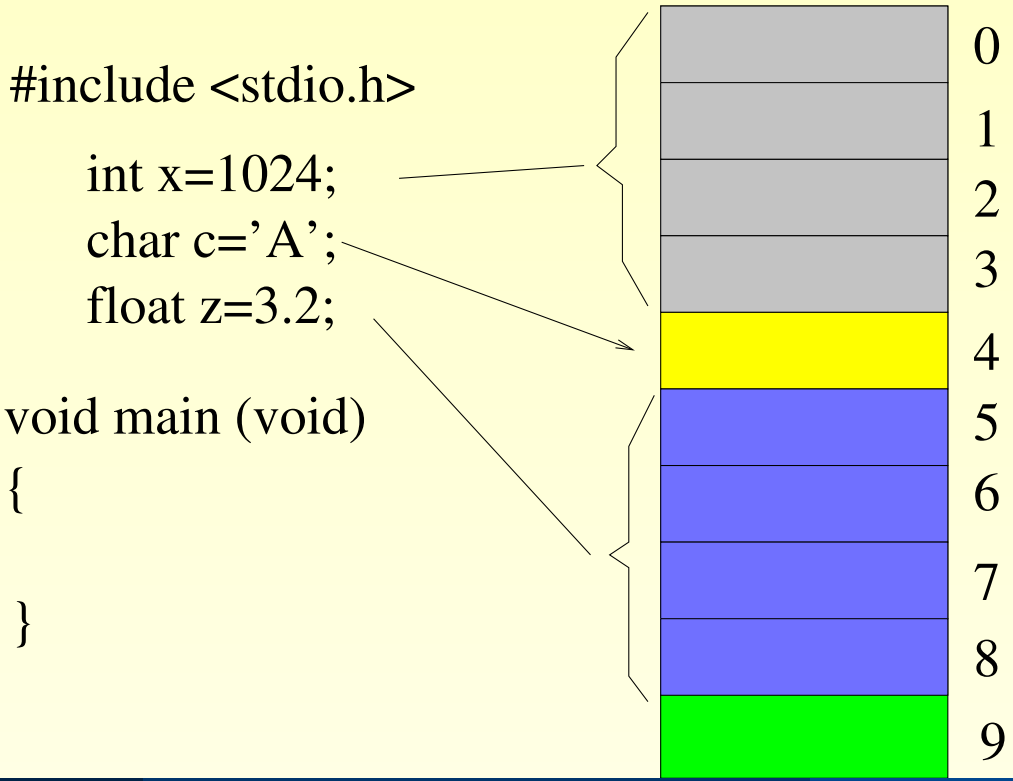
Introdução

- As variáveis vistas até agora são utilizadas para armazenar e manipular valores de determinados tipos;
- Variáveis do tipo ponteiro são utilizadas para guardar e manipular endereços de memória;
- As principais utilidades dos ponteiros são:
  - Passagem de parâmetros por referência, em subprogramação;
  - Alocação dinâmica de variáveis indexadas;
  - Encadeamento de estruturas;
  - Generalização de código;

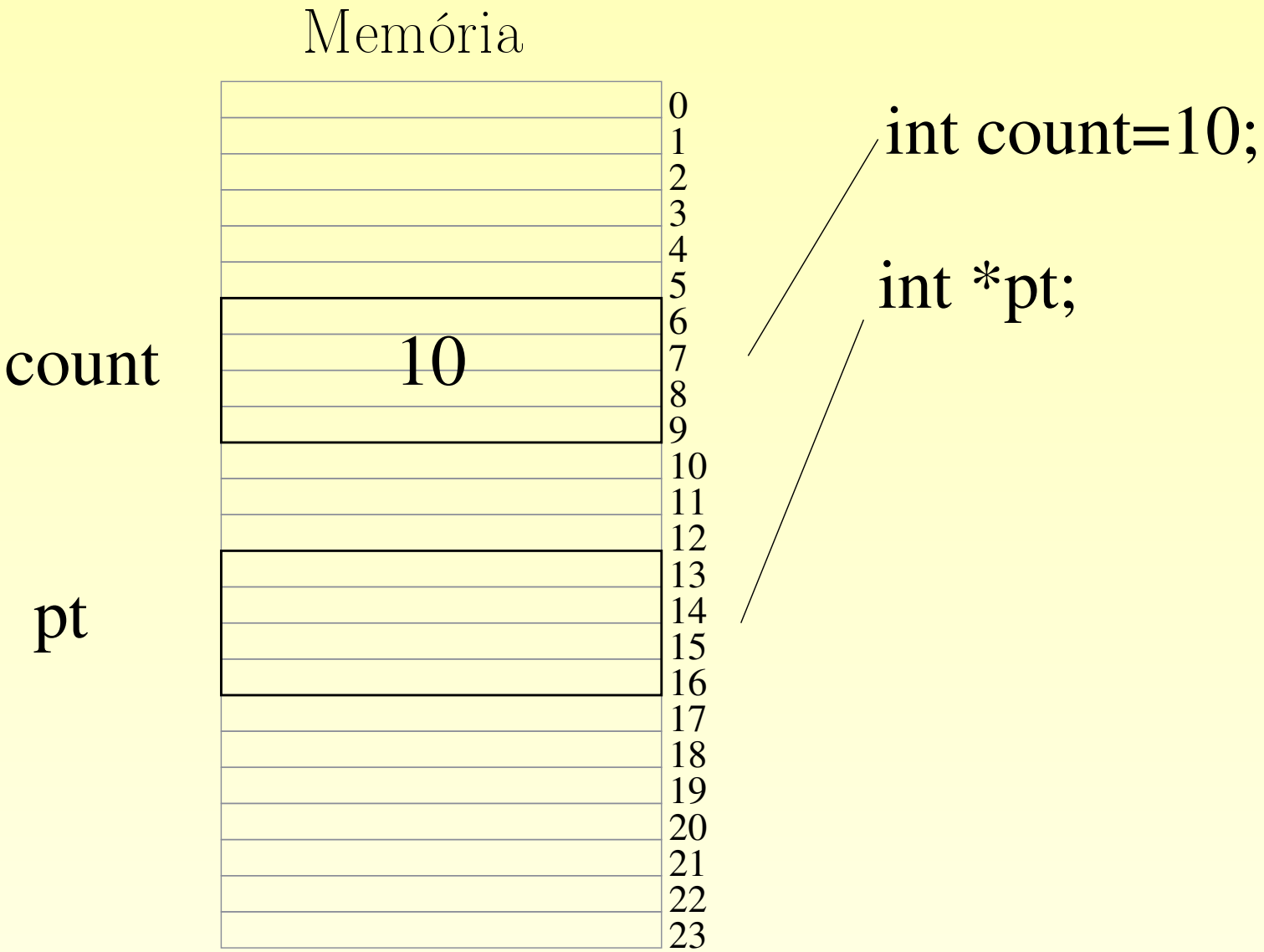
Como as variáveis ficam na memória ?

int 32 bits = 4 bytes
char 8 bits = 1 byte
float 32 bits=4bytes
double 64bits=8bytes

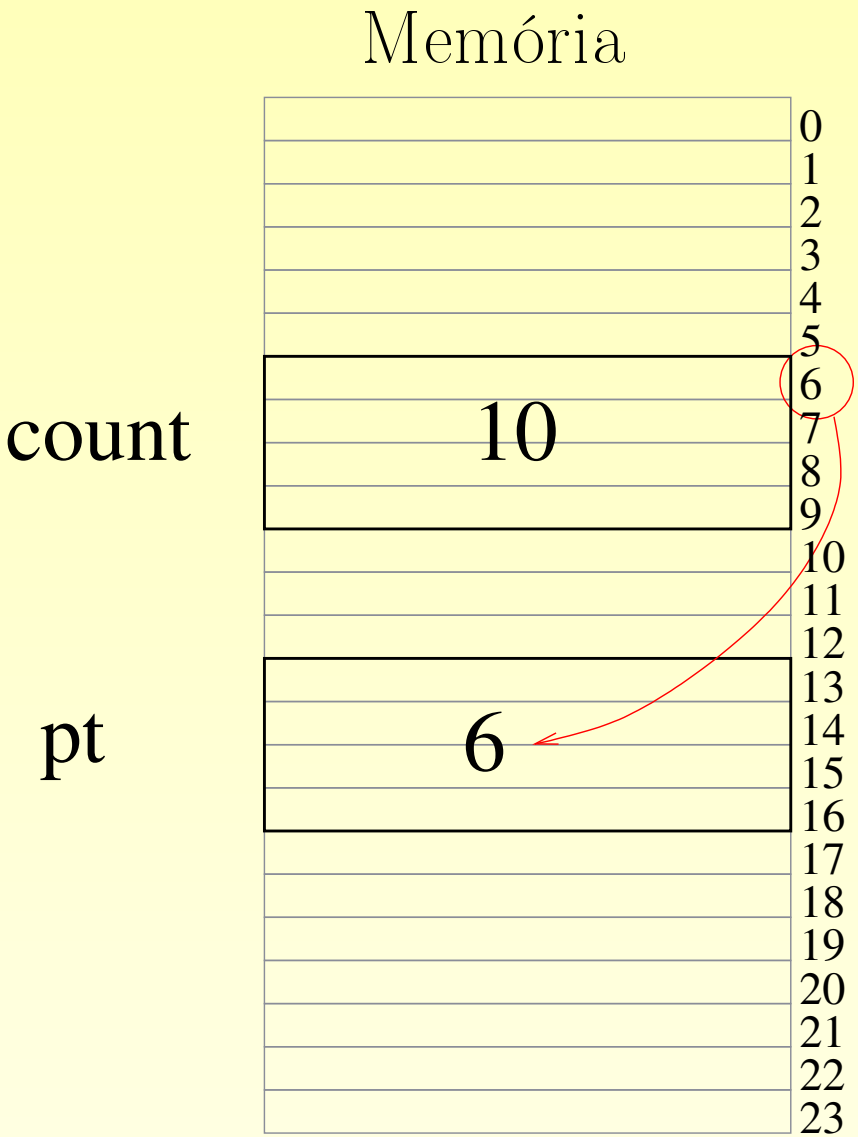
```
#include <stdio.h>

    int x=1024;
    char c='A';
    float z=3.2;

void main (void)
{

}
```



Ponteiros com variáveis simples:

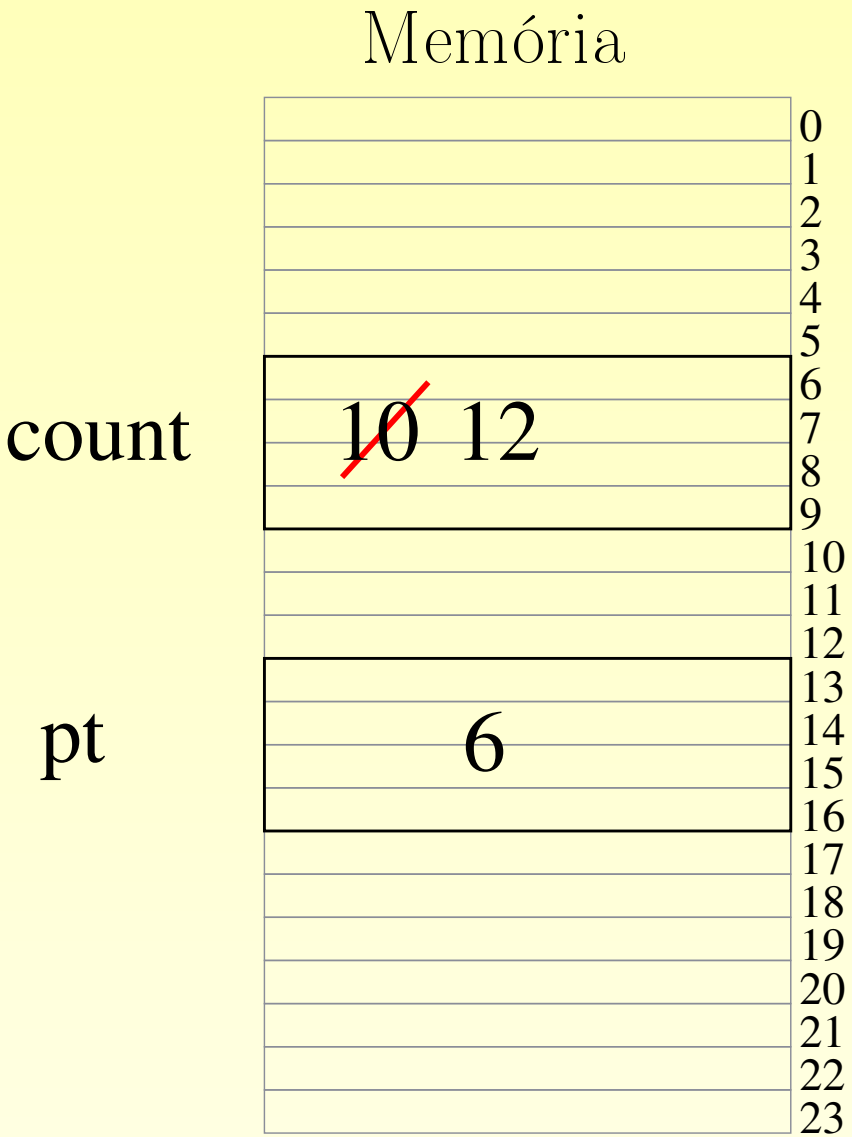


Ponteiros com variáveis simples:




```
int count=10;

int *pt;

pt=&count;

//pt=6 pois o
//endereco inicial
//de count vale 6
```

Ponteiros com variáveis simples:




```
int count=10;

int *pt;

pt=&count;

*pt=12;
//o conteudo da
//posicao armazenada
//em pt recebe 12
// ou seja
// a posicao 6
```

```c
#include <stdio.h>
void main (void)
{
    int num,valor;
    int *p;
    num=55;
    p=&num;          /* Pega o endereco de num */
    valor=*p;  /* Valor e igualado a num de uma maneira
        indireta */
    printf ("\n\n%d\n",valor);
    printf ("Endereco para onde o ponteiro aponta: %p\
       n",p);
    printf ("Valor da variavel apontada: %d\n",*p);
}
```

Importante!

- O caractere "*" antes do nome da variável a classifica como um ponteiro para o tipo utilizado na declaração da mesma. *Ex: int *a; declara a como sendo um ponteiro, ou sejam uma variável cujo conteúdo é um endereço (valor numérico)*;

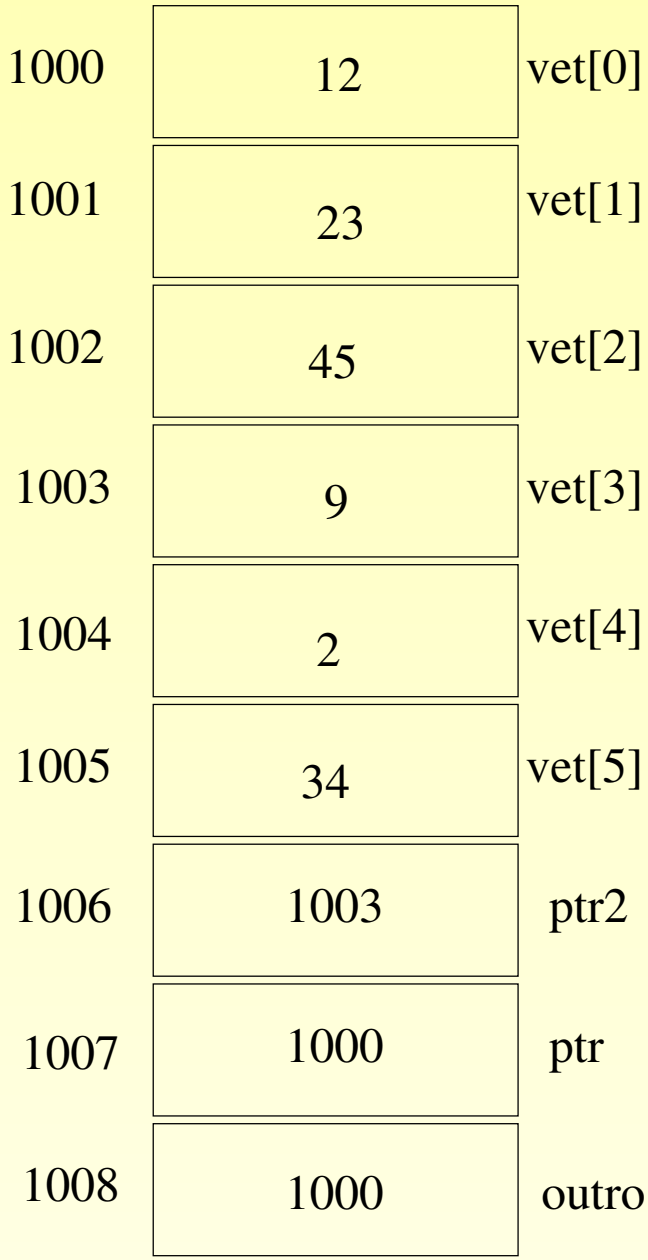
Importante!

- O caractere "&" antes do nome da variável faz com que o endereço da tal variável seja capturado. *Ex: int x; int *a; a = &x;*

Mas e ponteiros para variáveis do tipo vetor ?

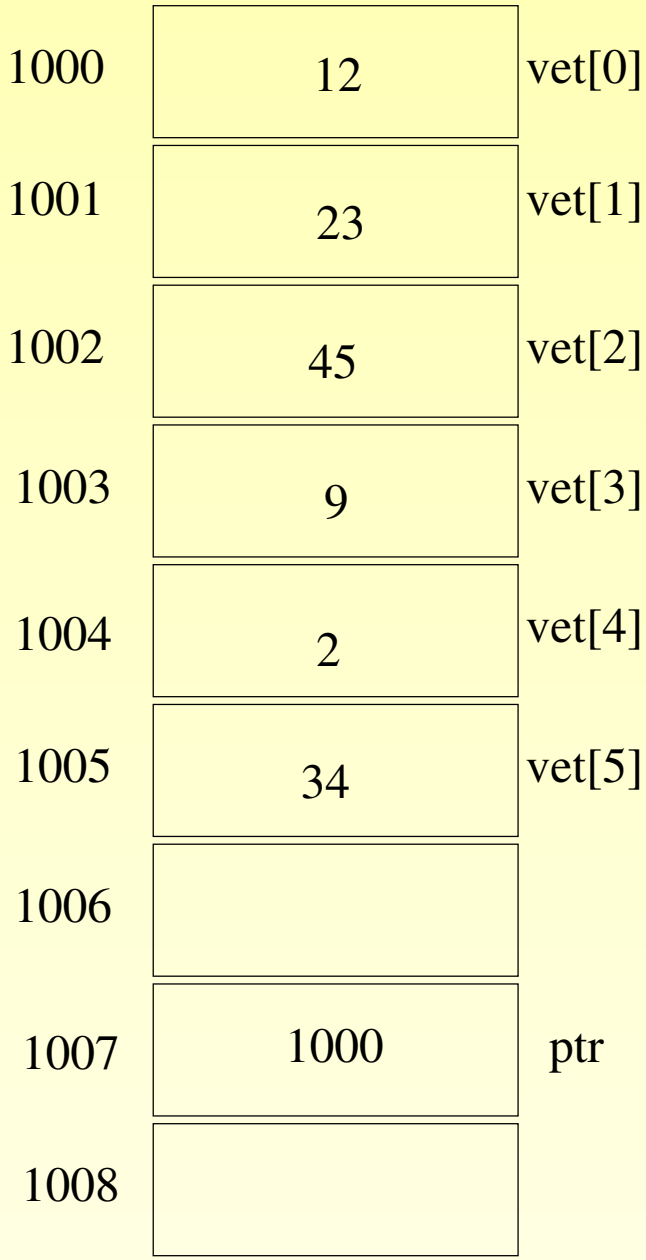
- O nome de um vetor é igual ao endereço de início dele na memória;
- char vet[]={12,23,45,9,2,34}; Caso o vetor vet inicie no endereço 1000, o nome vet=1000;
- além disso, vet = &vet[0], ou seja, o nome vet vale a mesma coisa que o endereço da sua posição 0;

Exemplo 1



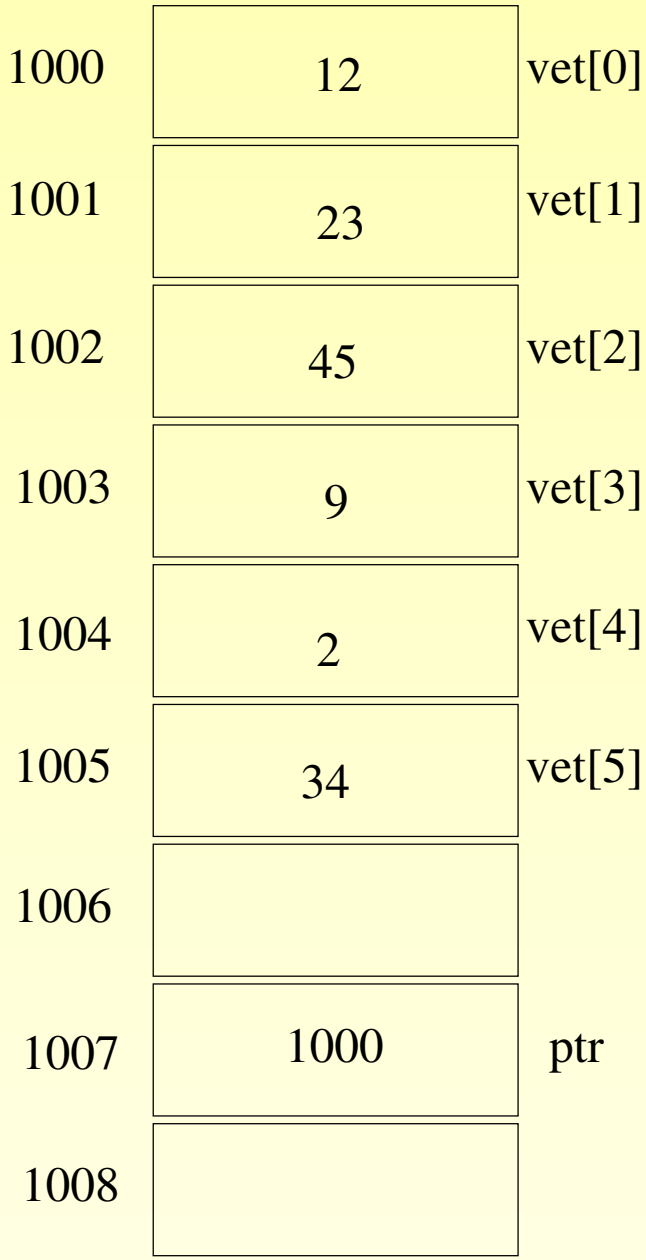

```
char vet[]={12,23,45,9,2,34};
char *ptr;  // ponteiro para char
char *outro; // outro ponteiro para char
char *ptr2; // mais um ponteiro
ptr = vet;
outro = &vet[0];
// tanto ptr quanto outro estao apontando
// para o mesmo endereco
ptr2 = &vet[3]; // ptr2  vale 1003
```

Exemplo 2 – parte 1




```
char vet[]={12,23,45,9,2,34};
char *ptr;  // ponteiro para char

int x;
ptr = vet; // ou ptr=&vet[0];
```

Exemplo 2 – parte 2




```
char vet[]={12,23,45,9,2,34};
char *ptr;  // ponteiro para char

int x;
ptr = vet; // ou ptr=&vet[0];

for (x=0;x<6;x++)
{
      *ptr=0;
       ptr++;
}
```

// O codigo acima faz ptr valer o endereco
// inicial do vetor (1000) no exemplo
// a linha *ptr=0; faz com que o conteudo
//da posicao 1000 receba 0
// depois o ponteiro e incrementado e passa a
// valer 1001, ou seja, na proxima vez
// o conteudo da posicao 1001 sera 0
// depois o conteudo da posicao 1002 ....

Mas e ponteiros para variáveis do tipo matriz ?

- O nome de uma matriz é igual ao endereço de início dela na memória;
- float matrx [50][50]; Caso a matriz inicie no endereço 1000, o nome matrx=1000;
- além disso, matrx = &matrx[0][0], ou seja, o nome matrx vale a mesma coisa que o endereço da sua posição [0][0];

```
1  main ()
2  {
3      float matrx [50][50];
4      float *p;
5      int count;
6      p=&matrx[0][0];
7      for (count=0;count<2500;count++)
8          {
9                  *p=0.0;
10                 p++;
11         }
12 }
```

Exercícios:

Exercício 1

Faça um programa que preencha dois vetores de quatro elementos numéricos cada um e mostre um terceiro vetor que é resultante da intercalação deles conforme exemplo abaixo utilizando o conceito de ponteiros.
o vetor vet_1 possui os elementos $4, 5, 3, 1$
já o vetor vet_2 possui os elementos $3, 11, 7, 2$.
Assim, o vetor resultante será $vet_3 = 4, 3, 5, 11, 3, 7, 1, 2$

Código:ex1_slides_ponteiros.c

Exercícios:

Exercício 2

Faça um programa que preencha um vetor com oito números inteiros e mostre dois vetores resultantes. O primeiro vetor resultante deve conter os números positivos; o segundo deve conter os números negativos. Cada vetor resultante vai ter no máximo oito posições, que poderão ser completamente utilizadas. Apresente somente as posições utilizadas dos vetores (positivo e negativo).

Código:ex2_slides_ponteiros.c

Exercícios:

Exercício 3

Crie uma função semelhante a strlen que retorna a quantidade de elementos de uma string. Crie a função de forma a utilizar ponteiros;

Código:ex3_slides_ponteiros.c

Ponteiros genéricos

- Você reparou que as variáveis **ponteiro** sempre tem um tipo ? Assim, se preciso trabalhar com um número `float` e utilizarei um ponteiro, o ponteiro precisa ser do tipo `float`. O mesmo vale para `int`, `long`, etc. Chamados isso de ponteiros "tipados"

```
1 int   *x; // ponteiro para inteiro
2 float *y; // ponteiro para real
3 long  *z; // ponteiro para inteiro longo
4 char  *k; // ponteiro para caractere
5 printf("inteiro=%d\n",sizeof(x));
6 printf("float =%d\n",sizeof(y));
7 printf("long  =%d\n",sizeof(z));
8 printf("char  =%d\n",sizeof(k));
```

Ponteiros genéricos - parte 2

Conclusão:

Ponteiros possuem sempre o mesmo tamanho. Esse tamanho está ligado ao sistema onde o código foi compilado. Num compilador de 16 bits, um ponteiro terá 16 bits. Num compilador de 32 bits os ponteiros terão 32 bits.

Ponteiros genéricos - parte 3

Ponteiros genéricos:

Chama-se **ponteiro genérico** um ponteiro que pode ser utilizado para "apontar" para variáveis de qualquer tipo. Representa-se assim:

```
1 #include <stdio.h>
2 #include <string.h>
3 void *ponteiro; //ponteiro generico
4 float z=3.24;
5
6 void main (void) {
7  ponteiro=(void *)&z;
8  printf("Valor = %f",*(float *)ponteiro);
9 }
```

Ponteiros genéricos - parte 4

Ponteiros genéricos:

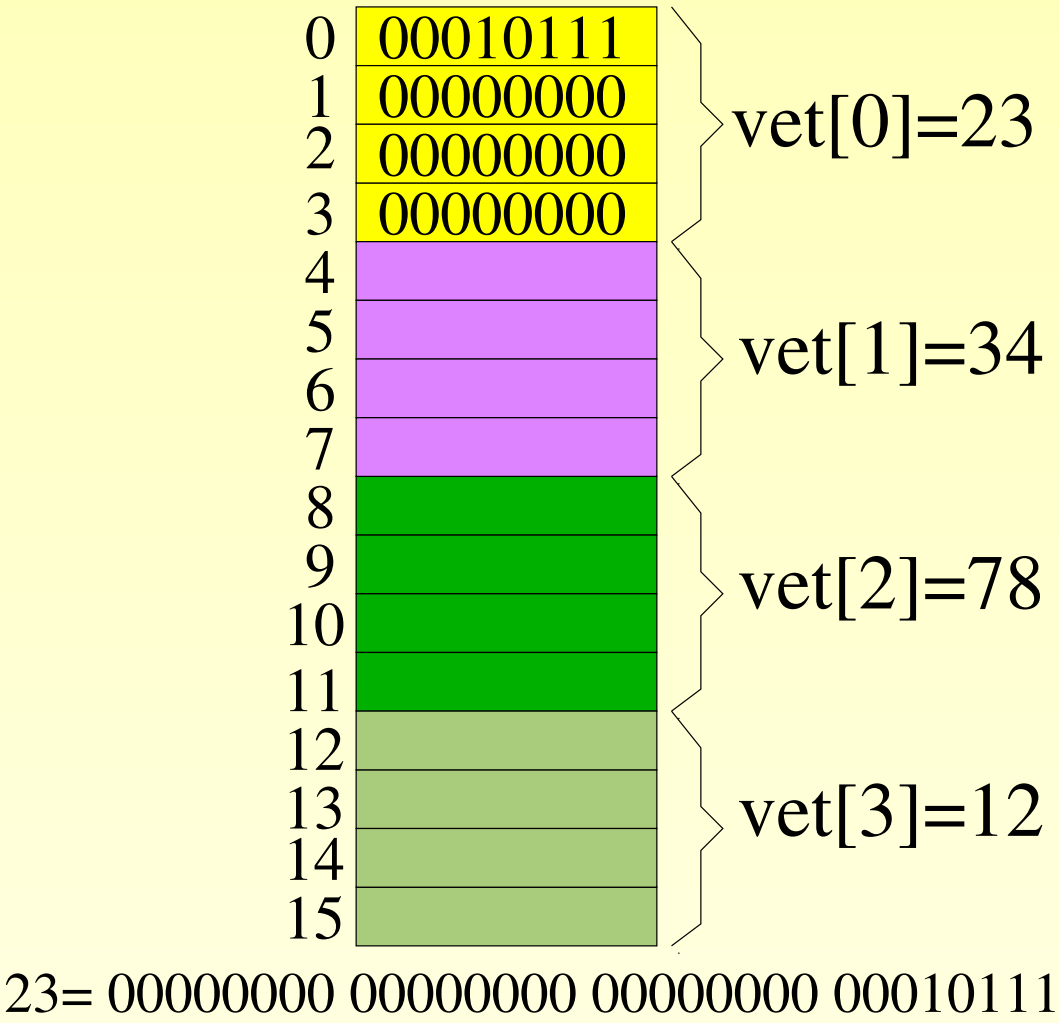
A expressão `ponteiro=(void *)&z` faz uma mudança forçada de tipo, ou seja, ela converte um endereço de um float para um endereço genérico.
Se ponteiros genéricos permitem trabalhar com variáveis de quaisquer tipos, por que preciso dos ponteiros "tipados"

Ponteiros genéricos - parte 4

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int vet[4]={23,34,78,12};
5
6  void main (void) {
7    int *ptr;
8    void * generico;
9    ptr=vet;
10   printf("terceiro elemento=%d\n",*(ptr+2));
11   generico=(void *) vet;
12   printf("terceiro elemento=%d\n",*((int *)(generico+8)
         ));
13 }
```

Ponteiros genéricos - parte 5

O código do slide anterior somente funciona como previsto pois como o ponteiro é tipado, ao somarmos o valor 2 ele pula para o terceiro inteiro na sequencia. Mesmo que um inteiro ocupe mais de 1 byte.



Uso Avançado - Ponteiros para funções

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

void mostra_msg ( void) {
  printf("Oi mundo\n");
}
void main (void)
{
     void (*p)(void); // Declara um ponteiro
    // para uma funcao que retorna void
    // e que tem um parametro void
     p=mostra_msg;// pega o endereco
     // da funcao
     p(); //chama indiretamente a funcao
}
```

Uso Avançado - Ponteiros para funções 2

```c
#include <stdio.h>
void mostra_msg ( void) {
    printf("Oi mundo\n");
}
void mostra_2 ( void) {
    printf("Tchau mundo\n");
}
void main (void) {
    void (*p[2])(void);

    p[0]=mostra_msg;
    p[1]=mostra_2;
    (*p[0])(); //chama mostra_msg
    (*p[1])(); //chama mostra_2
}
```