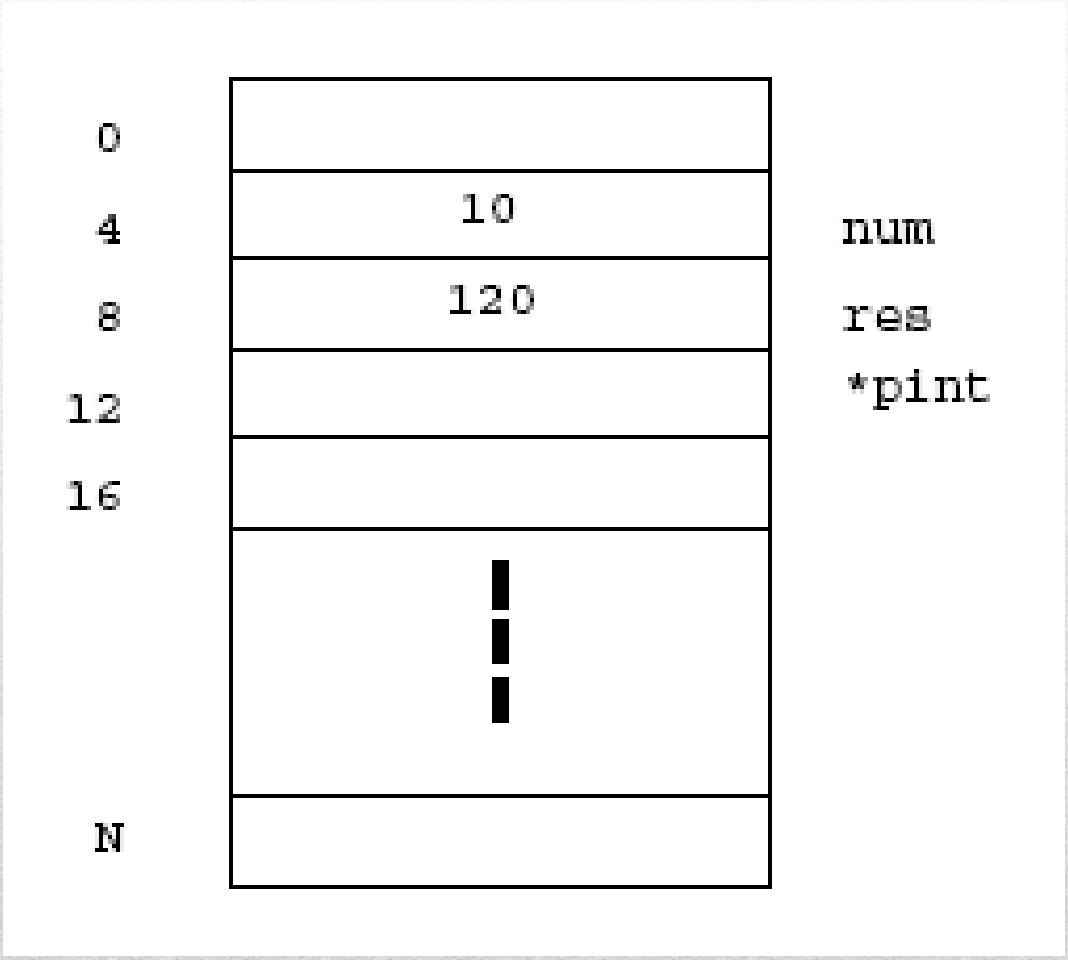
Ponteiros

Ponteiros

Ponteiros geralmente são usados em situações onde é necessário conhecer a localização da memória (o endereço) onde está armazenada a variável e não o seu conteúdo .

Um ponteiro é uma variável que contém um endereço de uma posição de memória e não o conteúdo da posição.

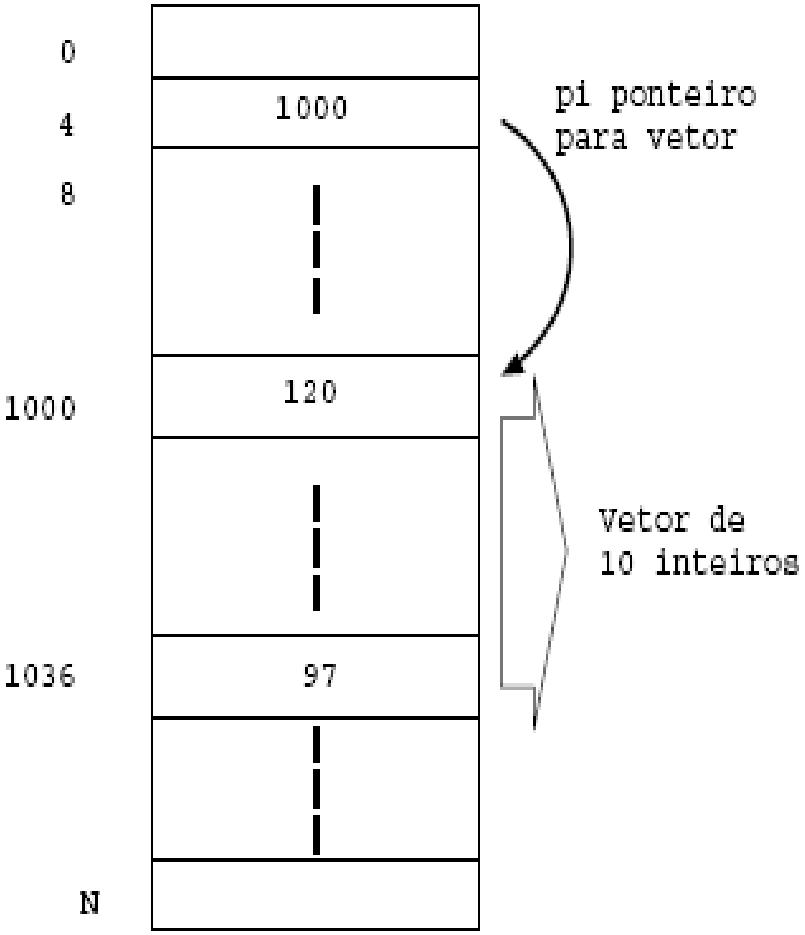
Exemplo: A Figura a seguir mostra o mapa de um trecho de memória que contém duas variáveis inteiras (num, res) ocupando 4 bytes cada uma e mais um ponteiro (pint), que também ocupa 4 bytes (endereços pulam de quatro em quatro bytes).


0
4
8
12
16
N
10
120
num
res
*pint

Aplicações:

Quando se deseja que uma função retorne mais de um valor. Neste caso a função recebe como argumentos não os valores dos parâmetros mas sim ponteiros que apontem para seus endereços. Assim esta função pode modificar diretamente os conteúdos destas variáveis, que após o fim da função estarão disponíveis para a função que chamou.

Com ponteiros, é possível reservar as posições de memória necessárias para armazenamento destas áreas somente quando for necessário e não quando as variáveis são declaradas.



Na figura, o ponteiro pi aponta para a área de memória que contém um vetor de 10 inteiros.

No início do programa é definida uma variável ponteiro e seu tipo.

Durante a execução do programa, reserva-se a área necessária para guardar os dados após descobrir o tamanho do vetor.

O tamanho do vetor é dado na sua declaração mantida até o final do programa.

Declaração de Ponteiros:

Os ponteiros, como as variáveis, precisam ser declarados.

Forma geral da declaração de um ponteiro é a seguinte:

```
tipo *nome;
```

Onde tipo é qualquer tipo válido em C e nome é o nome da variável ponteiro.

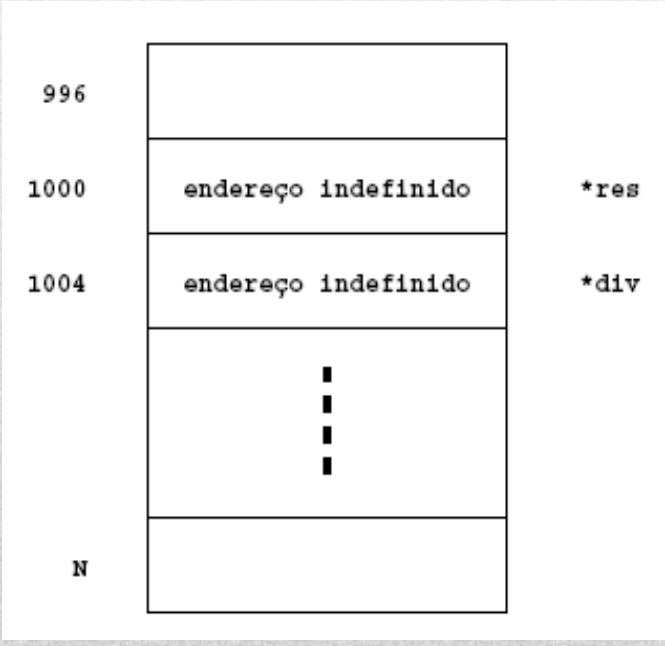
Por exemplo:

```
int *res; /* ponteiro para inteiro */
float *div; /* ponteiro para ponto flutuante */
```

Como as variáveis, os ponteiros também devem ser inicializados antes de serem usados.

Esta inicialização pode ser feita na declaração ou através de uma atribuição.

Após a declaração o que temos é um espaço na memória reservado para armazenamento de endereços. O valor inicial da memória é indefinido como acontece com variáveis.



Um ponteiro pode ser inicializado com um endereço ou com o valor NULL. O valor NULL, que é equivalente a 0, é uma constante definida no arquivo <stdio.h> e significa que o ponteiro não aponta para lugar nenhum.

Operadores Especiais para Ponteiros

Existem dois operadores especiais para ponteiros: * e &.

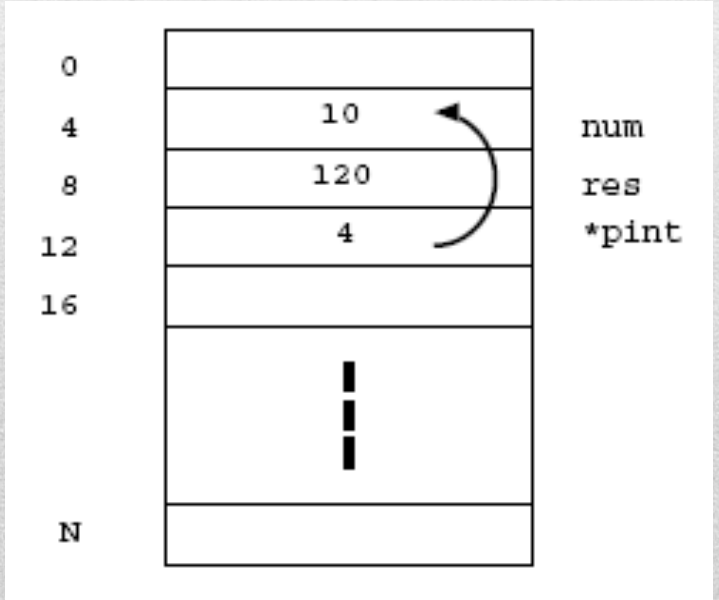
Os dois operadores são unários, isto é requerem somente um operando.

O operador & devolve o endereço de memória do seu operando.

Após a execução da instrução:

pint = # /*o endereco de num e carregado em pint */

a variável ponteiro pint termina com o valor 4.
O fato importante é que o ponteiro pint passou a apontar para a variável num.



O operador * é o complemento de &.

O operador * devolve o valor da variável localizada no endereço apontado pelo ponteiro.

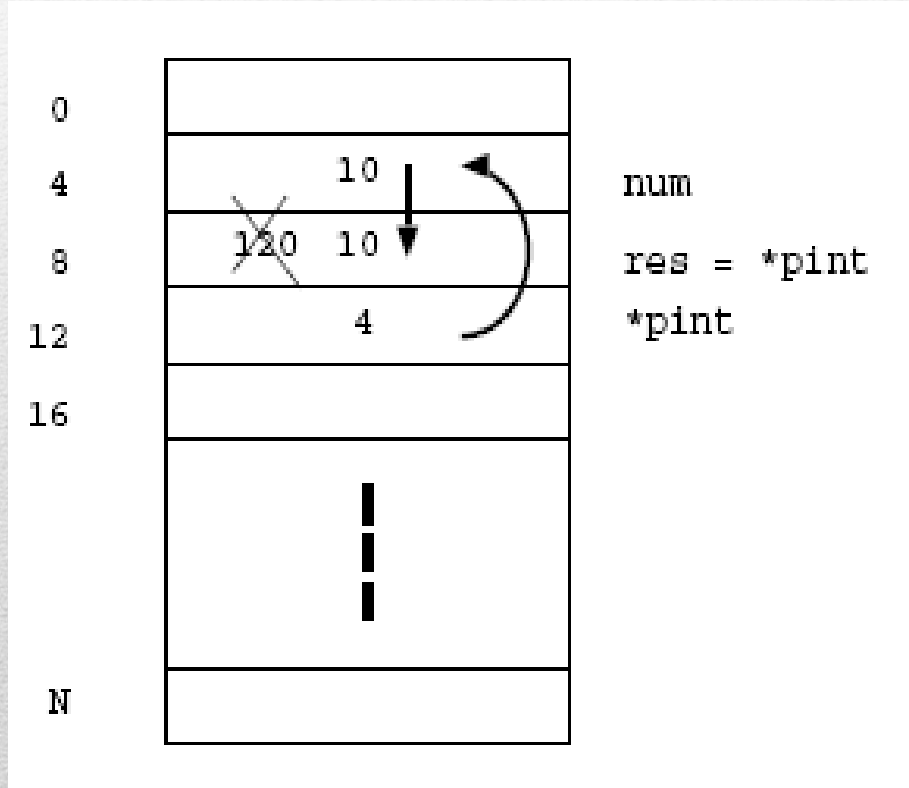
Por exemplo, considere que o comando

```
res = *pint;
```

foi executado logo após

```
pint = &num;
```

Isto significa que a variável res recebe o valor apontado por pint, ou seja a variável res recebe o valor 10.



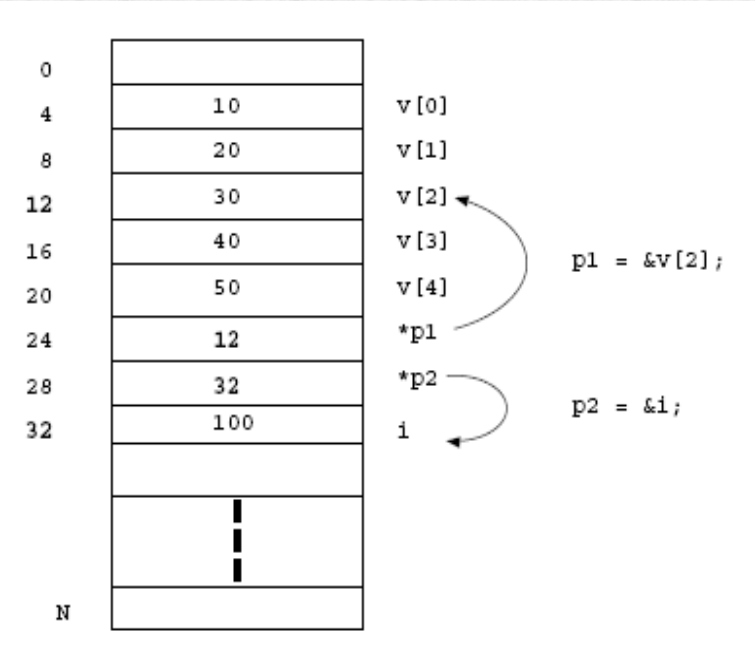
Atribuição de Ponteiros

Da mesma maneira que ocorre com uma variável comum, o conteúdo de um ponteiro pode ser passado para outro ponteiro do mesmo tipo.

Por exemplo, uma variável ponteiro declarada como apontador de dados inteiros deve sempre apontar para dados deste tipo.

Observar que em C é possível atribuir qualquer endereço a uma variável ponteiro. Deste modo é possível atribuir o endereço de uma variável do tipo float a um ponteiro do tipo int. No entanto, o programa não irá funcionar da maneira correta.

No exemplo a seguir, o endereço do terceiro elemento do vetor v é carregado em p1 e o endereço da variável i é carregado em p2. No final o endereço apontado por p1 é carregado em p2.



Exemplo de atribuição de ponteiros.

```
#include <stdio.h>
int main (void)
{
    int vetor [] = { 10, 20, 30, 40, 50 };
    int *p1 , *p2,  i = 100;

    p1 = & vetor [2];
    printf("%d\n", *p1 );
    p2 = &i;
    printf("%d\n", *p2 );
    p2 = p1;
    printf("%d\n", *p2 );
    return 0;
}
```

Os comandos printf imprimem os valo-res apontados pelos ponteiros respecti-vos, mostrando os seguintes valores:

30
100
30

Incrementando e Decrementando Ponteiros

Exemplos de operações com ponteiros.

```
int main (void)
{
    int vetor [] = { 10, 20, 30, 40, 50 };
    int *p1;
    p1 = &vetor [2];
    printf ("%d\n", *p1 );
    p1 ++;
    printf ("%d\n", *p1 );
    p1 = p1 + 1;
    printf ("%d\n", *p1 );
    return 0;
}
```

O primeiro printf imprime 30, que é o elemento de índice igual a 2 no vetor vetor. Após o incremento do ponteiro o segundo printf imprime 40 e o mesmo acontece com o terceiro printf que imprime 50.

Sempre que um ponteiro é incrementado (decrementado) ele passa a apontar para a posição do elemento seguinte (anterior).

O compilador interpreta o comando p1++ como: passe a apontar para o próximo número inteiro e, portanto, aumenta o endereço do número de bytes correto.

A operação abaixo faz com que o ponteiro p passe a apontar para o terceiro elemento após o atual.

```
p = p + 3;
```

Também é possível usar-se o seguinte comando *(p+1)=10;
Este comando armazena o valor 10 na posição seguinte àquela apontada por p.

Exemplo de subtração de ponteiros.

```
#include <stdio.h>
int main (void)
{
    float vetor [] = { 1.0, 2.0, 3.0, 4.0, 5.0 };
    float *p1 , *p2;
    p1 = & vetor [2]; /* endereco do terceiro elemento */
    p2 = vetor; /* endereco do primeiro elemento */
    printf(" Diferenca entre ponteiros %d\n", p1 -p2 );
    return 0;
} // imprime valor o 2
```

Não é possível multiplicar ou dividir ponteiros, e não se pode adicionar ou subtrair o tipo float ou o tipo double a ponteiros.

Comparação de Ponteiros

É possível comparar ponteiros em uma expressão relacional. No entanto, só é possível comparar ponteiros de mesmo tipo.

Exemplo de comparação de ponteiros.

```
#include < stdio.h>
int main (void)
{
    char *c, *v, a, b;
    scanf("%c %c", &a, &b);
    c = &a;
    v = &b;
    if (c == v)
        printf ("As variáveis estao na mesma posicao .");
    else
        printf("As variaveis nao estao na mesma posicao .");
    return 0;
}
```

Ponteiros e Vetores

Ponteiros e Vetores estão fortemente relacionados na linguagem C.

O nome de um vetor é um ponteiro que aponta para a primeira posição do vetor.

Na declaração de vetor, o compilador automaticamente reserva um bloco de memória para que o vetor seja armazenado.

Quando apenas um ponteiro é declarado a única coisa que o compilador faz é alocar um ponteiro para apontar para a memória, sem que espaço seja reservado.

Exemplo de alterações inválidas sobre ponteiros.

```
int list [5], i;
// Ponteiro list nao pode ser modificado pelo endereco de i
list = &i
// O ponteiro list nao pode ser incrementado
list ++;
```

Exemplo de notações de vetores.

```
#include < stdio.h>
int main (void)
{
    float v[] = {1.0 , 2.0, 3.0, 4.0, 5.0, 6.0, 7.0};
    int i;
    for (i = 0; i < 7; i++)
        printf ("%.1f ", v[i]);
    printf("\n");
    for (i = 0; i < 7; i++)
        printf ("%.1f ", *(v+i));
    return 0;
}
```

Exemplo de ponteiro variável.

```
#include <stdio.h>
int main (void)
{
    float v[] = {1.0 , 2.0, 3.0, 4.0, 5.0, 6.0, 7.0};
    int i;
    float *p;
    for (i = 0; i < 7; i++) printf("%.1 f ", v[i]);
    printf ("\n");
    for (i = 0; i < 7; i++) printf("%.1 f ", *(v+i));
    printf ("\n");
    for (i = 0, p = v; i < 7; i++, p++) printf("%.1 f ", *p);
    return 0;
}
```

Qual o valor de y no final do programa?
Tente primeiro descobrir e depois verifique no computador o resultado.

```
int main()
{
    int y, *p, x;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    x--;
    (*p) += x;
    printf ("y = %d\n", y);
    return(0);
}
```

```
void main(void)
{
        float v[] = {1.0, 2.0, 3.0, 4.0,
5.0, 6.0, 7.0, 8.0, 9.0};
        int i;
        for (i=0; i<9; i++) printf("%.1f
", v[i]);
        printf("\n");
        for (i=0; i<9; i++) printf("%.1f
", *(v+i)); }
```

O nome de um vetor é chamado de ponteiro constante e, portanto, não pode ter o seu valor alterado. Assim, os comandos abaixo não são válidos:

```
void main()
{
        int list[5], i;

        /* O ponteiro list nao pode ser modificado recebendo o
endereco de i */
       list = &i
       /* O ponteiro list nao pode ser incrementado */
       list++;
}
```

```
void main(void) {  float v[] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};  int i;  float *p;  for (i=0; i<9; i++) printf("%.1f ", v[i]);  printf("\n");  for
```

Para percorrer um vetor além da maneira mostrada é possível usar um ponteiro variável como ilustrado abaixo.

```
void main(void)
{
        float v[] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};
        int i;
        float *p;
        for (i=0; i<9; i++) printf("%.1f ", v[i]);
        printf("\n");
        for (i=0; i<9; i++) printf("%.1f ", *(v+i));
        printf("\n");
        for (i=0, p=v; i<9; i++, p++) printf("%.1f ", *p); }
```

Observe como o ponteiro p recebe seu valor inicial e a maneira que ele é incrementado.