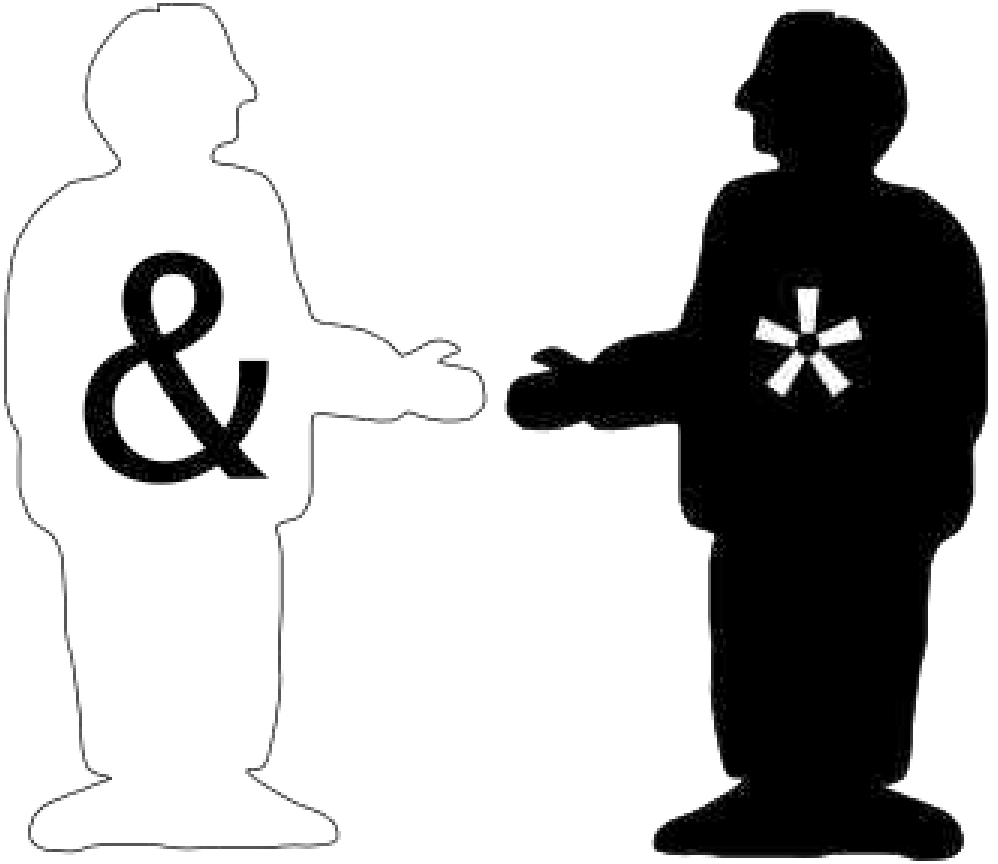
Ponteiros e Alocação Dinâmica

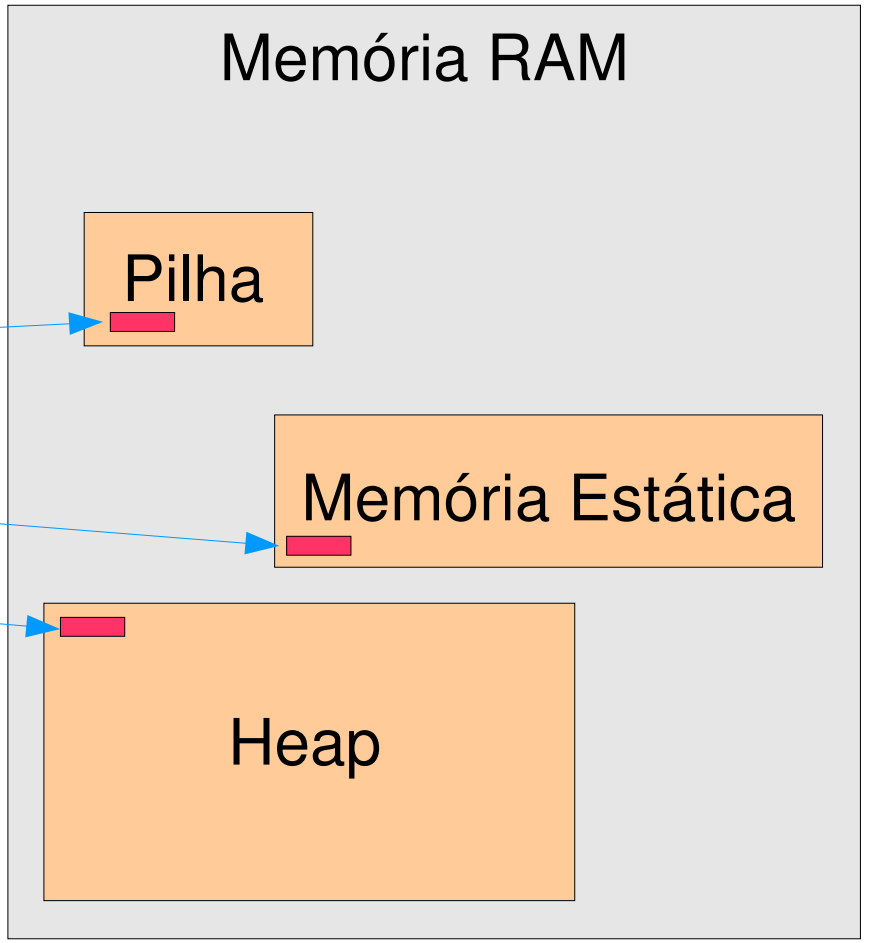


Ponteiros (ou Apontadores)

Ponteiros são variáveis que guardam o endereço de memória (localização) de alguma outra coisa.

Ponteiros são declarados com um * antes do nome da variável:

```
int *p;
float *q;
char *r;
```

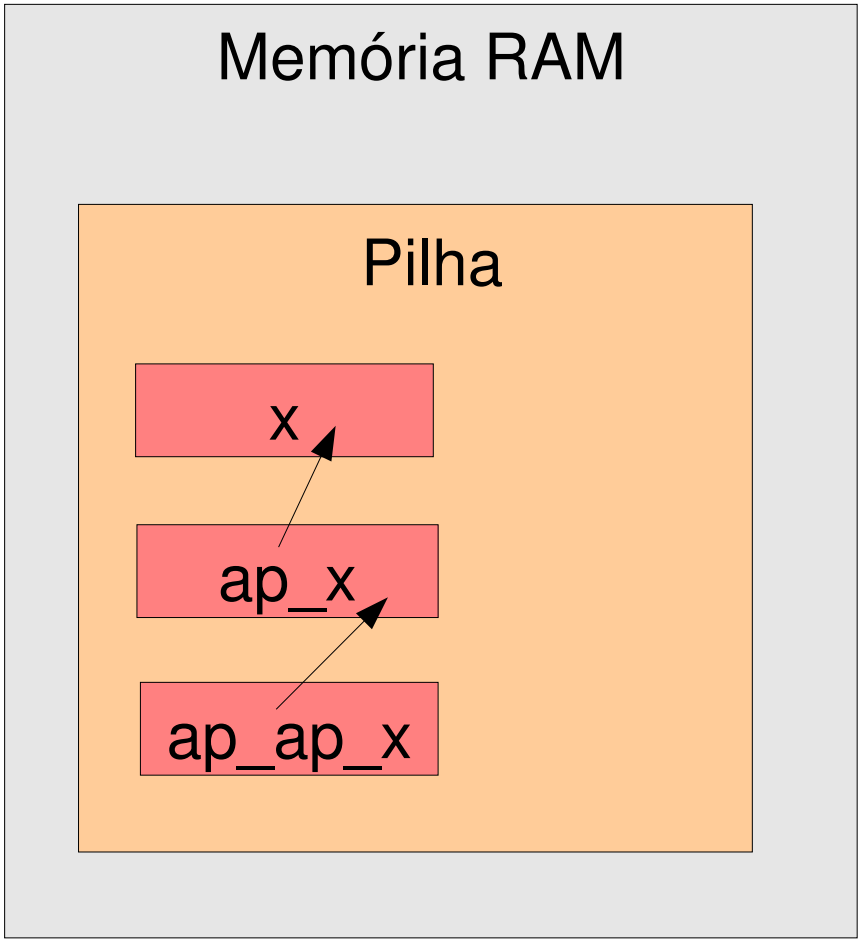


São apontadores para um inteiro, float e caractere, respectivamente.

Operador &

O operador & obtem o endereço de uma variável:

```
int x;
int *ap_x;
int **ap_ap_x;

ap_x = &x;
ap_ap_x = &ap_x;
```

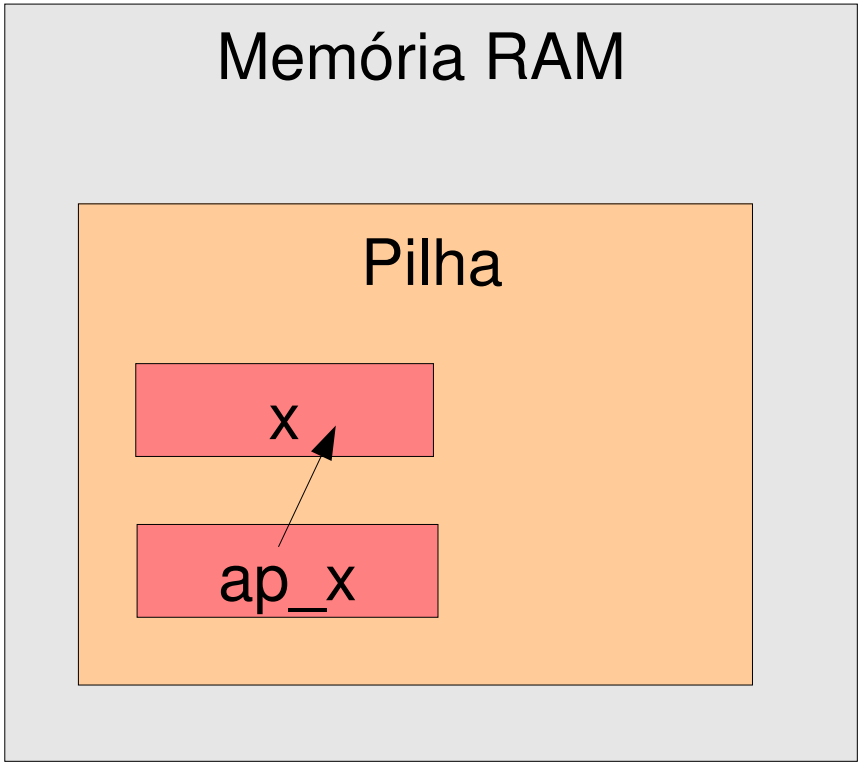


Como apontadores também são variáveis, eles também ocupam memória e podemos obter o endereço do apontador e ter apontadores para apontadores (múltiplos *).

Operador *

O operador * unário faz o contrário do &: dado um apontador, acessa o conteúdo apontado por ele:

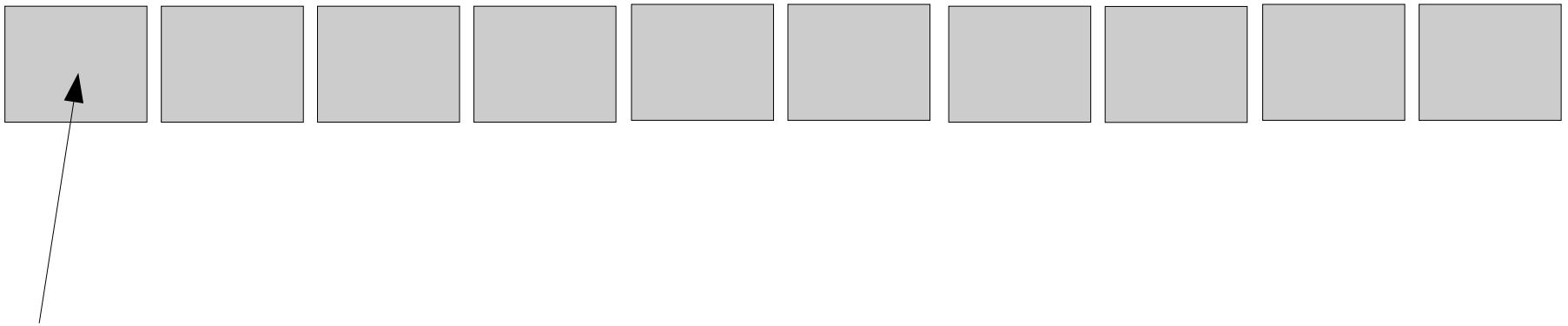
```
int x, y,*ap_x;

ap_x = &x;
*ap_x = 5; // atribui 5 na
      // posição apontada por ap_x

y = *ap_x; // lê o conteúdo da memória
  // apontada por ap_x e atribui a y
```



Vetores: a verdade

Para o C, um vetor é um apontador para a sua primeira posição (índice 0):

```
int v[10];
```



v ou &v[0] *(para o C, int v[ ] e int *v são sinônimos)*

Vetores: a verdade

Para passar vetores como parâmetros de funções, sempre declaramos o tipo do parâmetro como um apontador para o tipo do vetor. O C não tem como saber o tamanho do vetor.
Se for preciso, temos que passar o tamanho em um parâmetro separado.

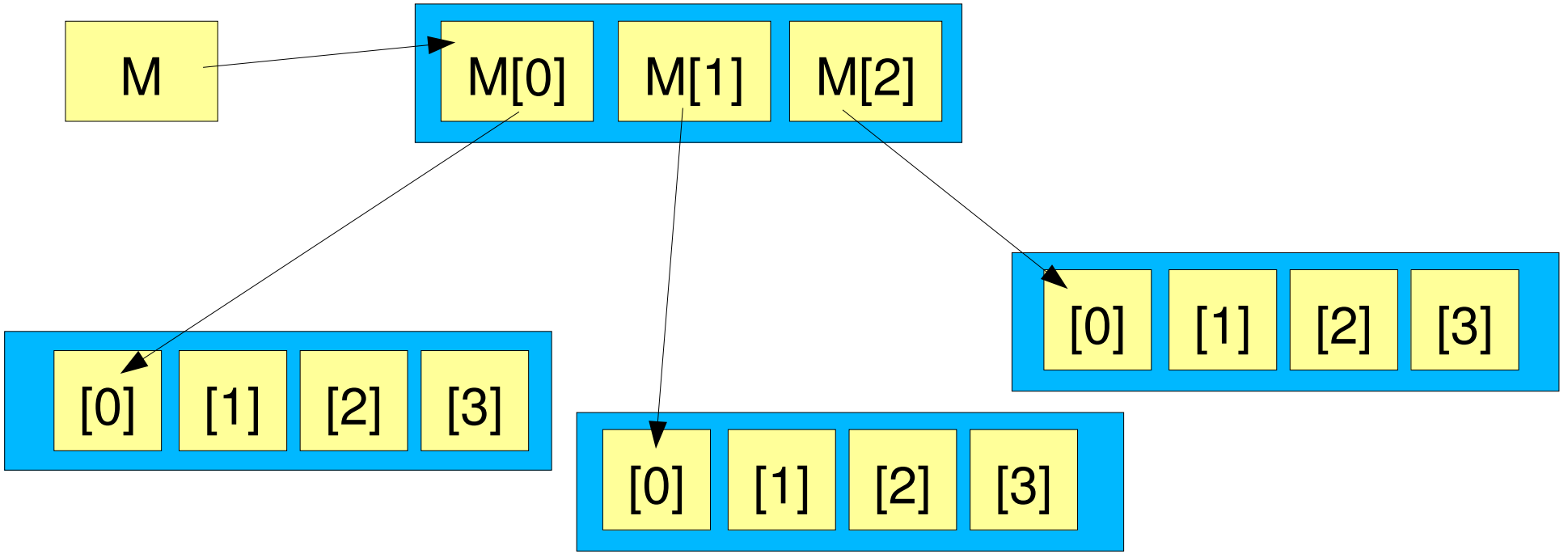
Vetores: a verdade

```
void selection_sort(int *v, int n) {
        int i,j,min,minpos;
        for(i=0;i<n-1;i++) {
                min = v[i];
                minpos = i;
                for(j=i+1;j<n;j++)
                        if (v[j] < min) {
                                min = v[j];
                                minpos = j;
                        }
                v[minpos] = v[i];
                v[i] = min;
        }
}
```

```
void main() {
        int x[200], i;

        for(i=0;i<200;i++)
                x[i] = rand()%500;

        selection_sort(x,200);
}
```

Vetores: a verdade

int M[3][4];



Vetores multidimensionais são vetores de vetores. Passar vetores multidimensionais como parâmetro de funções pode ser complicado.

Alocação Dinâmica

O C possui 3 "regiões" de memória: a pilha, a memória estática, e o heap.

Na pilha são criadas as variáveis locais.

Na memória estática são criadas as variáveis globais e locais estáticas.

As variáveis da pilha e da memória estática precisam ter tamanho conhecido antes do programa ser compilado.

Como obter memória de forma dinâmica ?

Alocação Dinâmica

Com ponteiros e alguma ajuda do sistema operacional, podemos obter memória à medida que vamos precisando.

Funções em <stdlib.h> para alocação dinâmica de memória:

```
void *malloc(int tamanho);
```

Aloca um bloco de memória no heap com tamanho bytes e retorna um ponteiro. O ponteiro deve ser "casted" para o tipo de ponteiro a ser usado:

```
int *x;
x = (int *) malloc(sizeof(int) * 10000); // aloca um vetor de 10000 inteiros
x[9000] = 10;
```

Alocação Dinâmica

Funções em <stdlib.h> para alocação dinâmica de memória:

void free(void *apontador);

Desaloca um bloco de memória alocado com malloc.

Se você não desalocar a memória alocada com malloc, a memória não pode ser utilizada por outros programas do sistema, e eventualmente o computador pode ficar sem memória. Toda memória alocada é liberada quando o programa termina. Dizemos que programas que não desalocam a memória que consomem "vazam memória".

Você não pode desalocar memória que não foi alocada. Você não pode desalocar o mesmo bloco de memória duas vezes.

Alocação Dinâmica

```
void main() {
	int *x,i;

	x = (int *) malloc(sizeof(int) * 10000); // aloca um vetor de 10000 inteiros
	for(i=0;i<10000;i++)
		x[i] = rand() % 100;

	selection_sort(x,10000);
	free(x); // desaloca x
}
```

Alocação Dinâmica: realloc

A função realloc faz um bloco já alocado crescer ou diminuir, preservando o conteúdo já existente:

```
void * realloc(void *apontador, int novo_tamanho);
```

Exemplo:

```
int *x,i;
x = (int *) malloc(4000*sizeof(int));
for(i=0;i<4000;i++) x[i] = rand()%100;

x = (int *) realloc(x, 8000*sizeof(int));

x = (int *) realloc(x,2000*sizeof(int));
free(x);
```