Variáveis, endereços, ponteiros e arrays

- Variáveis são entidades "virtuais": só existem no âmbito do compilador. Na realidade, toda variável corresponde a um bloco de bytes na memória e o "nome" da variável é um "apelido" para se referir a essas posições na memória de uma forma mais confortável. Portanto, toda variável possui (corresponde a) um endereço na memória.
- De acordo com o tipo de informação que a variável representa, o tamanho e as operações com essa área de memória variam.
- Cada tipo de variável corresponde a um tipo de informação:
  - `int` → números inteiros
  - `char` → caracteres alfanuméricos
- Sintaxe para se referir a informações das variáveis em geral:
  - `nome_var`: refere-se ao conteúdo da variável (ver exceção abaixo – arrays)
  - `&nome_var`: refere-se ao endereço da variável
  - `tipo_var nome_var`: define (cria) uma variável do tipo `tipo_var`.

Ponteiros

- Variáveis do tipo ponteiro correspondem a informações do tipo endereço:
  - Ponteiros → endereços de memória
- Com ponteiros, existem dois endereços envolvidos:
  - O endereço do ponteiro (`&nome_var`). As variáveis do tipo ponteiro, como toda variável, também residem em um bloco de bytes da memória e têm endereço.
  - O conteúdo do ponteiro (`nome_var`), que é o endereço armazenado na variável.
- Sintaxe específica para variáveis do tipo ponteiro:
  - `*nome_var`: operação de desreferenciação (exclusiva de ponteiros). Refere-se ao conteúdo da área de memória para onde o conteúdo da variável aponta.
  - `tipo_var *nome_var`: define (cria) uma variável ponteiro denominada `nome_var` que aponta para uma variável (área de memória) do tipo `tipo_var`
  - `void *nome_var`: define (cria) um ponteiro genérico (não pode ser desreferenciado, ou seja, não se pode usar `*nome_var`)
- Na aritmética de ponteiros, ao se somar 1 ao conteúdo (valor) da variável, o ponteiro aumenta tantos bytes quanto o tamanho do dado para o qual ele aponta:
  - ```
    int *pi;
    ...
    pi += 1;  // pi passa a apontar 4 bytes a frente, se o tipo int ocupar 4 bytes.
    ```
  - ```
    char *pc;
    ...
    pc += 1;  // pc passa a apontar 1 byte a frente, pois tipo char ocupa 1 byte.
    ```
- Ponteiros, ao serem criados, não são alocados pelo compilador e, como todas as variáveis C, inicialmente contêm lixo (podem apontar para qualquer lugar). Portanto, até que sejam inicializados, não podem ser desreferenciados (usar `*nome_var`).

Arrays

- Arrays são conjuntos de variáveis, armazenadas consecutivamente na memória. Cada elemento (componente) do array se comporta de maneira equivalente a uma variável do mesmo tipo do array.
    - Arrays → n x variáveis
- Sintaxe específica para variáveis do tipo array:
    - `tipo_var nome_var[N]`: define (cria) N variáveis do tipo `tipo_var`
    - `nome_var`: refere-se ao endereço do primeiro elemento do array. Portanto, `nome_var` é do tipo ponteiro para `tipo_var` e equivalente a `&(nome_var[0])`.
        - ATENÇÃO: O uso do nome dos arrays é uma exceção à regra geral de que, para se referir ao endereço de uma variável, é preciso acrescentar um &.
        - ATENÇÃO: operações com arrays NUNCA realizam nenhuma operação em todos os elementos do array. Por exemplo, a expressão A = B, onde A e B são arrays, não tornará cada elemento de A igual ao elemento correspondente em B, mas seria uma tentativa de fazer com que o endereço do primeiro elemento de A seja igual ao de B, o que não é permitido.
    - `&nome_var`: raramente usada, também se refere ao endereço do primeiro elemento do array, porém é do tipo ponteiro para array de `tipo_var`. A diferença com relação ao anterior é que, na aritmética de ponteiros, ao se somar com 1 avança tantos bytes quanto o tamanho do array inteiro. Além disso, não é desreferenciado: `*(&A)` também é um endereço, igual a `&A`.
- Arrays tem comportamento similar aos ponteiros, comportando-se aproximadamente como ponteiros pré-alocados pelo compilador.
    - `nome_var[i]` equivale a `*(nome_var+i)`
    - Arrays não podem ter seu conteúdo (a área de memória para onde apontam) alterado: sempre apontam para a área de memória que foi alocada pelo compilador.
    - Ponteiros podem ser alterados (podem apontar para onde o usuário desejar).
- Arrays podem ser initializados, com os valores iniciais entre `{}`, no momento da criação:

```
int arr[5] = {3,8,1,-1,2};    // arr = [ 3 8 1 -1 2 ]
int arr[] = {3,8,1,-1,2};     // arr = [ 3 8 1 -1 2 ]
int arr[5] = {3,8,1};         // arr = [ 3 8 1 0 0 ]
int arr[5] = {};              // arr = [ 0 0 0 0 0 ]
int arr[5] = {3,8,1,-1,2,7};     ERRO DE COMPILAÇÂO
int arr[5]{3,8,1,-1,2};       // arr = [ 3 8 1 -1 2 ] C++ 2011
```

- Arrays multidimensionais podem ser implementados de forma equivalente com um array unidimensional, multiplicando-se os índices:

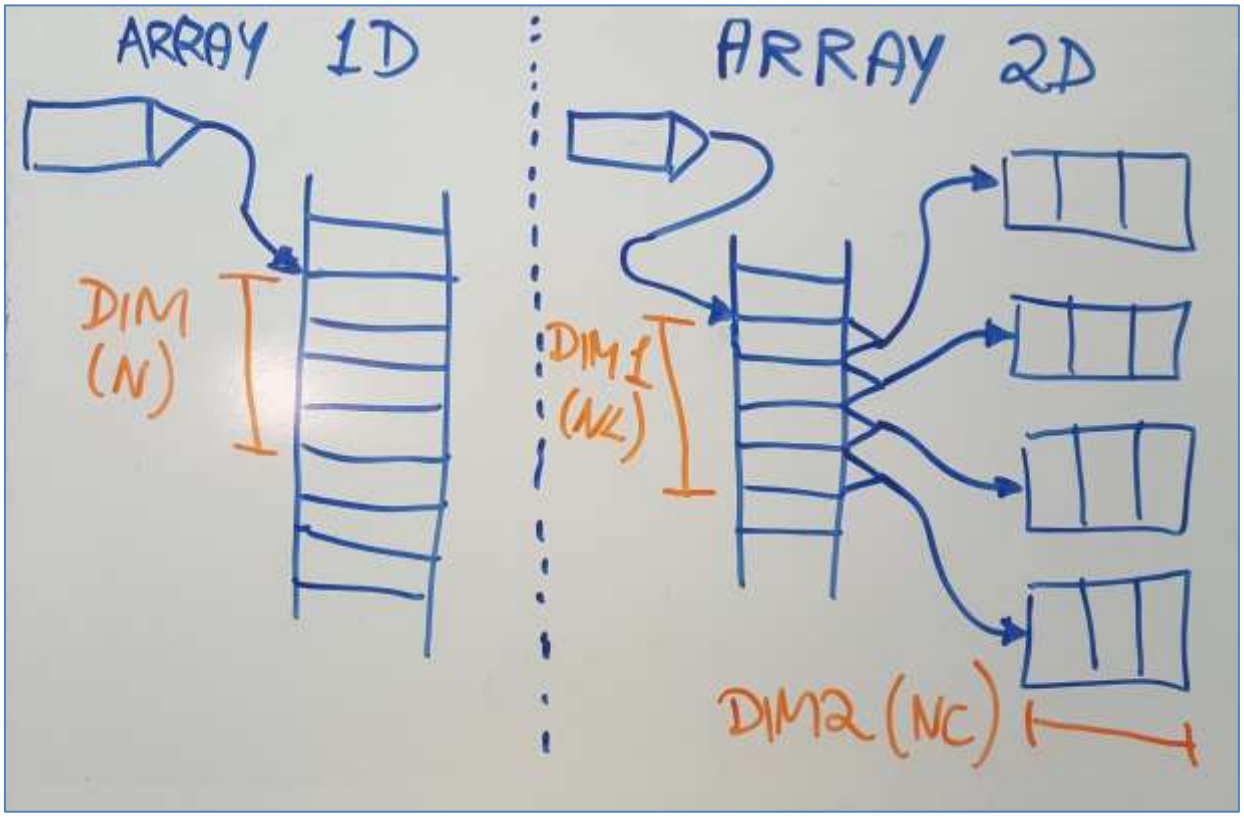
Array multidimensional	Array pseudo-multidimensional

`#define N_COL 5`
`#define N_LIN 3`
`int mat[N_LIN][N_COL];`
`int i,j;`
 	
`int main()`
`{`
`  for (i=0; i<N_LIN; i++)`
`    for (j=0; j<N_COL; j++)`
`      mat[i][j] = i+j;`
`}`
	`#define N_COL 5`
	`#define N_LIN 3`
	`int mat[N_LIN * N_COL];`
	`int i,j;`
	
	`int main()`
	`{`
	`  for (i=0; i<N_LIN; i++)`
	`    for (j=0; j<N_COL; j++)`
	`      mat[i*N_COL + j] = i+j;`
	`}`

Alocação de memória

- Alocar memória é solicitar ao sistema operacional que uma área de memória de tamanho especificado fique destinada exclusivamente para esse programa. É usada para dar um valor inicial válido a ponteiros (que contêm lixo ao serem criados).
- Sintaxe específica para alocação de memória:
  - `new tipo_var`: retorna o endereço para uma área de memória que pode conter uma variável do tipo `tipo_var`
  - `new tipo_var[N]`: retorna o endereço para uma área de memória que pode conter N variáveis do tipo `tipo_var`
  - `delete` libera uma área criada com `new`
  - `delete[]` libera uma área criada com `new[]`
- Semelhança entre arrays estáticos e arrays dinâmicos criados com ponteiros:
  - `tipo_var nome_var[N]` se comporta de forma similar a `const tipo_var *nome_var = new tipo_var[N]`

Arrays bidimensionais



Array de dimensão variável

Aumentando um array

- Ponteiro `arr`, dimensão `N`:
  1. Crie um ponteiro provisório, `prov`
  2. Aloque para prov uma área com `N+1` elementos
  3. Copie todos os elementos de `arr` para `prov`
  4. Copie o novo elemento na última posição de `prov`
  5. Libere a área para onde `arr` aponta
  6. Faça `arr` apontar para a nova área de memória
  7. Incremente a dimensão `N`

Passo a passo: