C

Guilherme P. Telles

IC

30 de março de 2023

Arrays

Array

- Mecanismo para reunir um conjunto finito de elementos do mesmo tipo e dar ao conjunto um nome único.
- Cada elemento do array pode ter seu valor atribuído ou selecionado diretamente e em tempo constante, através de um índice inteiro que indica a posição do elemento no array.
- Em qualquer lugar onde uma expressão ou variável do tipo `T` deveria ser usada, pode-se usar um elemento de array do tipo `T`.
- O tamanho de um array é especificado na criação e não muda.
- Um array pode ter mais de uma dimensão. Com uma dimensão é chamado de vetor, com duas de matriz.

Vetor

- A definição de um vetor tem a forma

```
tipo nome[n]
```

- O tamanho `n` deve ser integral e positivo.
- Os índices dos elementos do vetor vão de 0 a n-1.
- Os elementos do vetor são armazenados em posições consecutivas na memória.

Acesso

- O operador `[]` é usado para acesso a elementos do vetor.
- Os limites do vetor não são verificados.
- Um acesso fora da faixa válida tem efeito difícil de prever e depende do ambiente.
  - Pode retornar ou modificar outra variável do programa.
  - Pode tentar retornar ou modificar uma região fora do programa.

exm-vetor-1.c

```c
#include <stdio.h>

int main() {

  int A[8], i;

  for (i=0; i<8; i++)
    A[i] = i;

  for (i=0; i<8; i++)
    printf("%d ",A[i]);
  printf("\n");
}
```

exm-vetor-2.c

```c
#include <stdio.h>

#define size 8

int main() {

  int A[size], i;

  for (i=0; i<size; i++)
    A[i] = i;

  for (i=0; i<size; i++)
    printf("%d ",A[i]);
  printf("\n");
}
```

exm-vetor-3.c

```c
#include <stdio.h>

int main() {

  int i, n = 8;

  int A[n];

  for (i=0; i<n; i++)
    A[i] = i;

  for (i=0; i<n; i++)
    printf("%d ",A[i]);
  printf("\n");
}
```

exm-vetor-4.c

```c
#include <stdio.h>

int main() {

  int n;
  scanf("%d",&n);

  int A[n], i;

  for (i=0; i<n; i++)
    A[i] = i;

  for (i=-10; i<2*n; i++)
    printf("%d ",A[i]);
  printf("\n");

  for (i=-10; i<2*n; i++)
    A[i] = i+1010;

  for (i=-10; i<2*n; i++)
    printf("%d ",A[i]);
  printf("\n");
}
```

Inicialização

- A inicialização de um vetor tem a forma

  `tipo nome[c] = {s1,...,sc}`
  `tipo nome[] = {s1,...,sc}`

- Se a lista tem tamanho menor que `c`, os últimos elementos são inicializados com zero.
- Se a lista é maior que `c`, os últimos elementos são ignorados.
- Vetores declarados com tamanho definido por variável não podem ser inicializados.

```c
int main() {

  int A[5] = { 10, 20, 30, 40, 50 };
  int B[5] = { 10, 20 };
  int C[5] = { 0 };
  int D[5] = { 10, 20, 30, 40, 50, 60 };

  printf("A: ");
  for (int i=0; i<5; i++)
    printf("%d ",A[i]);
  printf("\nB: ");

  for (int i=0; i<5; i++)
    printf("%d ",B[i]);
  printf("\nC: ");

  for (int i=0; i<5; i++)
    printf("%d ",C[i]);
  printf("\nD:");

  for (int i=0; i<5; i++)
    printf("%d ",D[i]);
  printf("\n");
}
```

sizeof

- `sizeof` retorna o número de bytes ocupados pelo array.

exm-sizeof-2.c

```c
#include <stdio.h>

int main() {

  int A[10];
  int B[10][20];
  int C[10][20][30];

  printf("A %ld\n",sizeof(A));
  printf("B %ld\n",sizeof(B));
  printf("C %ld\n",sizeof(C));

  return 0;
}
```

Vetores e apontadores

- O nome de um array unidimensional é um apontador constante para o endereço do elemento na posição 0.
- Vamos supor que

```
int V[] = {1,2,3,4,5};
```

então

`V` é equivalente a `&V[0]`.

exm-vetor-6.c

```c
#include <stdio.h>

int main() {

  int V[5] = {100,200,300,400,500};

  printf("V[0]: %d\n", V[0]);
  printf("*V: %d\n", *V);
}
```

Matrizes

- Um par de colchetes adicionais acrescenta uma dimensão a um array.

```
int M[5][3];
```

- O acesso a membros é feito pela composição de operadores `[]`.

```
M[3][2] += M[1][1] + 1;
```

exm-vetor-10.c

```c
#include <stdio.h>

#define m 4
#define n 6

int main() {

  int A[m][n];
  int i,j;

  for (i=0; i<m; i++)
    for (j=0; j<n; j++)
      A[i][j] = i*n + j;

  for (i=0; i<m; i++) {
    for (j=0; j<n; j++) {
      printf("%2d ", A[i][j]);
    }
    printf("\n");
  }
}
```

Inicialização

- A inicialização de um array multidimensional é por linhas.
- Cada dimensão pode ser separada por `{ }`.
- Valores omitidos são inicializados como zero.
- O tamanho da primeira dimensão pode ser omitido e será igual ao número de pares de chaves correspondentes.

```c
int M[2][3] = {11,12,13,21,22,23};
```

```c
int M[2][3] = {{11,12,13},{21,22,23}};
```

```c
int M[][3] = {{11,12,13},{21,22,23}};
```

Matrizes e apontadores

- Supondo `int M[m][n]`,
  - O nome `M` é equivalente a `&M[0]`, um vetor com `m` ints.
  - `M[i]` pode ser pensado como a linha `i`.
  - A partir de `&M[0][0]` está o espaço consecutivo para `m*n` ints.
  - Se os elementos forem acessados na ordem em que estão alocados, índices mais a direita variam mais depressa.

Mais de duas dimensões

- Basicamente adicionamos mais pares de colchetes.
-
- Supondo `int A[`n_1`]...[`n_k`]`,
  - O nome de um array `A` de `k` dimensões é equivalente a `&A[`0_2`]...[`0_k`]`, um array `k`-1 dimensões.
  - A partir de `&A[`0_1`]...[`0_k`]`, está o espaço consecutivo para n_1`*...*`n_k ints.
- Se os elementos forem acessados na ordem em que estão alocados, índices mais a direita variam mais depressa.

exm-vetor-7.c

```c
#include <stdio.h>

int main() {

  int C[2][2][3] = {{{111,112,113},{121,122,123}},
                    {{211,212,213},{221,222,223}}};

  for (int f=0; f<2; f++) {
    for (int l=0; l<2; l++) {
      for (int c=0; c<3; c++) {
        printf("%d ",C[f][l][c]);
      }
      printf("\n");
    }
    printf("\n\n");
  }
}
```

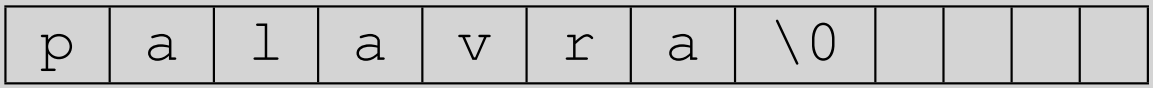
- Arrays multi-dimensionais definidos desta forma não são muito adequados em vários casos porque eles não podem ser passados para uma função sem fixar a maioria das dimensões na definição da função.
- Freqüentemente usamos matrizes alocadas dinamicamente, como veremos depois.

Cadeias de caracteres

Cadeias de caracteres (strings)

- Uma string em C é um vetor de `char` que tem o caractere com valor 0 (`'\0'`) marcando o fim da cadeia.

```
char s[12] = "palavra";
```



- Observe que o tamanho da string é menor que o tamanho do array.
  - Então temos duas idéias distintas: o tamanho do array e o tamanho da string dentro do array.

- A inicialização

```c
char s[] = "abc";
```

é equivalente a

```c
char s[] = {'a', 'b', 'c', '\0'};
```

- O `\0` é verificado por funções de saída, que imprimem até encontrá-lo, e por outras funções que processam strings.
- As funções de entrada normalmente acrescentam o `\0`.
- É responsabilidade do programador definir o vetor grande o bastante para armazenar o `\0`.

exm-strings-1.c

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

int main(void) {

  char s[20] = "ab   c  def";

  s[3] = 'z';

  for (int i=0; i<strlen(s); i++)
    printf(">%c<\n",s[i]);

  printf("\ntamanho do vetor: %ld\n",sizeof(s)/sizeof(char));
  printf("tamanho da string: %ld\n",strlen(s));
}
```

exm-strings-2.c

```c
#include <stdio.h>

int main(void) {

  char s[20] = "abcdef";

  int i = 0;
  while (s[i] != 0) {          // s[i] != '\0'
    printf(">%c<\n", s[i]);
    i++;
  }

  printf("\ntamanho do vetor: %d\n",sizeof(s)/sizeof(char));
  printf("tamanho da string: %d\n",strlen(s));
}
```

exm-strings-3.c

```c
#include <stdio.h>

int main(void) {

  char chr;
  int i = 0;
  char s[10];
  int t = 0;

  // Leitura potencialmente problemática:
  while (scanf("%c",&chr) != EOF) {
    s[i++] = chr;
  }
  s[i] = '\0';

  printf(">%s<\nt=%d\n",s,t);
}
```

- Relembrado: o nome de um vetor é um apontador para o prmeiro elemento dele.

exm-strings-4.c

```
int main() {

  char s1[] = "abcdef";
  char* s2 =  "ghijkl";
  char* pc;

  for (pc = s1; *pc != '\0'; pc++)
    putchar(*pc);

  for (pc = s2; *pc; pc++)
    putchar(*pc);

  printf("\n");
  return 0;
}
```

Entrada

- `scanf` possui alguns especificadores de conversão para strings:
  - `s`: lê caracteres até encontrar um branco.
  - `nc`: lê n caracteres, não acrescenta `\0`.
  - `[ ]`: lê caracteres enquanto fizerem parte do conjunto especificado.
  - `[^ ]`: lê caracteres enquanto não fizerem parte do conjunto especificado.
  - `n` e `[` não descartam brancos à esquerda.

- Entrada: `abacate abacaxi\n`

```
char str[30];
```

`scanf("%s",str)`	`abacate\0`
`scanf("%10c",str)`	`abacate ab`
`scanf("%[abc]",str)`	`abaca\0`
`scanf("%[^e]",str)`	`abacat\0`
`scanf("%[^\n]",str)`	`abacate abacaxi\0`

Entrada

- `[]` e `[^]` permitem especificar faixas de carateres com hífen, p.ex. 32-46, h-j.
- Para incluir `]` ele deve ser o primeiro caractere, o hífen deve ser o último.

Múltiplas strings

- Múltiplas strings podem ser manipuladas como matrizes de caracteres ou como vetores de apontadores para caractere.

```
char nomes[5][10] = {"Carlos","Dora","Lia","Lea",""};
```

```
char* nomes[] = {"Carlos","Dora","Lia","Lea",""};
```

```
int k = 10;
char S[k][51];

for (i=0; i<k; i++)
  scanf("%s ", S[i]);
```

string.h

- Declara várias funções para manipular strings.
- Declara o tipo `size_t`, que é grande o bastante para armazenar o tamanho do maior vetor que pode ser construído na linguagem.

- `size_t strlen(const char *s);`

  Retorna o número de caracteres em `s`, sem contar o `\0`.

- `char* strcat(char* dest, const char* src);`

  Copia `src` no fim de dest `dest`, sobrescrevendo o `\0` em `dest` e incluindo o `\0` de `src`. Retorna um apontador para `dest`.
  O vetor `dest` deve ser grande o suficiente.

- `char *strncat(char *dest, const char *src, size_t n);`

  Similar a `strcat`, porém apenas os `n` primeiros caracteres de `src` são copiados.

- `int strcmp(const char *s1, const char *s2);`

  Compara `s1` e `s2` e retorna um inteiro menor, igual ou maior que zero se o `s1` for respectivamente, lexicograficamente menor, igual ou maior que `s2`.

- `int strncmp(const char *s1, const char *s2, size_t n);`

  Similar a `strcmp`, porém no máximo os `n` primeiros caracteres de `s1` e `s2` são comparados.

- `char *strcpy(char *dest, const char *src);`

  Copia `src` em `dest`, incluindo o `\0`. Retorna um apontador para `dest`. O vetor `dest` deve ser grande o suficiente.

- `char *strncpy(char *dest, const char *src, size_t n);`

  Similar a `strcpy`, no máximo os primeiros `n` caracteres de `src` serão copiados. Se `\0` não estiver entre os n caracteres, `dest` não será terminada por `\0`.

Funções de busca

`strchr`	retorna um apontador para a primeira ocorrência de um caractere
`strrchr`	retorna um apontador para a última ocorrência de um caractere
`strpbrk`	retorna um apontador para a primeira ocorrência de um caractere de um conjunto
`strspn`	retorna o maior prefixo de uma cadeia que tenha apenas caracteres em um conjunto
`strstr`	retorna um apontador para o início da primeira ocorrência de uma subcadeia

Funções de conversão em `stdlib` e `stdio`

`atoi`	string para int
`atol/strtol`	string para long
`atoll/strtoll`	string para long long
`strtof`	string para float
`strtod`	string para double
`strtoul`	string para unsigned long
`strtoull`	string para unsigned long long
`sprintf`	conversões para string

Aritmética de apontadores

Aritmética de apontadores

- Relembrando: o nome de um vetor é um apontador para o primeiro elemento dele.
- Se `p` é um apontador para algum elemento de um vetor então a sentença `p+1` resulta no endereço para recuperar ou armazenar o elemento seguinte.
- Se `p` e `q` são apontadores para elementos de um vetor, `p-q` resulta no número de elementos do vetor entre `p` e `q`.

- Supondo

```
int V[n];
```

- `V` é equivalente a `&V[0]`
  `V+i` é equivalente a `&V[i]`

- `*V` é equivalente a `V[0]`
  `*(V+i)` é equivalente a `V[i]`

exm-aritmap-1a.c

```c
#include <stdio.h>

int main() {

  int V[10] = {0,1,2,3,4,5,6,7,8,9};

  printf("*V: %d\n", *V);
  printf("*(V+1): %d\n", *(V+1));
  printf("*(V+5): %i\n", *(V+5));
}
```

exm-aritmap-1b.c

```c
#include <stdio.h>

int main() {

  int V[] = {11,22,33,44,55,66,77,88,99};

  int* p = V+4;

  printf("*p: %i\n",*p);
  printf("*(p+1): %i\n", *(p+1));
  printf("*(p+2): %i\n", *(p+2));
}
```

exm-aritmap-1c.c

```c
#include <stdio.h>

int main() {

  int V[10] = {0,1,2,3,4,5,6,7,8,9};

  int* q = V+9;
  int* r = V+4;

  printf("r-q: %li\n", r-q);
  printf("q-r: %li\n", q-r);
}
```

exm-aritmap-1d.c

```c
#include <stdio.h>

int main() {

  int V[10] = {0,1,2,3,4,5,6,7,8,9};

  for (int i=2; i<7; i++) {
    printf("%d %d\n", V[i], *(V+i));
  }
  printf("\n");
}
```

exm-aritmap-1e.c

```c
#include <stdio.h>

int main() {

  int V[10] = {0,1,2,3,4,5,6,7,8,9};

  int* p = V+2;
  int* q = V+7;

  while (p != q) {
    printf("%d ",*p);
    p++;
  }
  printf("\n");
}
```

exm-aritmap-2.c

```c
#include <stdio.h>

int main(void) {

  char s[] = "abcdefghij";
  char *pc;

  for (pc = s; *pc != '\0'; pc++)
    printf("%s\n", pc);
}
```

exm-aritmap-3a.c

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

int main(void) {

  char* phrase = "it doesnt matter the way that you make it";
  char* pc;

  pc = strstr(phrase, "th");

  printf("frase: %s\n", phrase);
  printf("ocorrencia: %s\n", pc);
  printf("onde ocorreu: %ld\n", pc-phrase);
}
```

exm-aritmap-3b.c

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

int main(void) {

  char* phrase = "it doesnt matter the way that you make it";
  char* pc;

  pc = strstr(phrase, "th");

  printf("frase: %s\n", phrase);
  printf("ocorrencia: %s\n", pc);
  printf("onde ocorreu: %d\n", pc-phrase);

  pc = strstr(pc+1, "th");
  printf("ocorrencia: %s\n",pc);
  printf("onde ocorreu: %d\n",pc-phrase);
}
```

Arrays bidimensionais e apontadores

- Relembrando, se temos a matriz `int M[n][m]`,
  - O nome `M` é equivalente a `&M[0]`, um vetor com `m` ints.
  - A partir de `&M[0][0]` está o espaço consecutivo para `n*m` ints.
  - Se os elementos forem acessados na ordem em que estão alocados, índices mais a direita variam mais depressa. Ou seja, os elementos estão armazenados linha-a-linha.

exm-aritmap-5.c

```c
#include <stdio.h>

int main() {

  int M[3][5] = { {1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15} };

  for (i=0; i<3; i++) {
    for (j=0; j<5; j++)
      printf("%2d ", M[i][j]);
    printf("\n");
  }

  int i = 1;
  int j = 3;

  printf("%d\n", M[i][j] );
  printf("%d\n", *(M[i] + j) );
  printf("%d\n", (*(M + i))[j] );
  printf("%d\n", *((*(M + i)) + j) );
  printf("%d\n", *(&M[0][0] + 5*i + j) );
}
```

- Para qualquer array, o mapeamento entre valores de apontadores e índices do vetor é dado por uma função de mapeamento de armazenamento.
- P.ex., para matrizes e cubos:

```
int M[r][c];
M[i][j] = *(&M[0][0] + i*c + j)
```

```
int C[s][r][c];
C[i][j][k] = *(&C[0][0][0] + i*r*c + j*c + k)
```