

Ponteiros - Parte 2

Algoritmos e Programação 2

Prof. Dr. Anderson Bessa da Costa

Universidade Federal de Mato Grosso do Sul

Ponteiros: Tipos

Ponteiros: Tipos

- Qual o tamanho de um ponteiro?

```
#include <stdio.h>

int main () {
    int *p;
    printf("%lu\n", sizeof(p));
    return 0;
}
```

8

Ponteiros: Tipos (cont.)

- Qual o tamanho de um ponteiro?

```
#include <stdio.h>

int main () {
    int *p;
    char *test;

    printf("%lu\n", sizeof(p));
    printf("%lu\n", sizeof(test));

    return 0;
}
```

8
8

Tamanho do Ponteiro

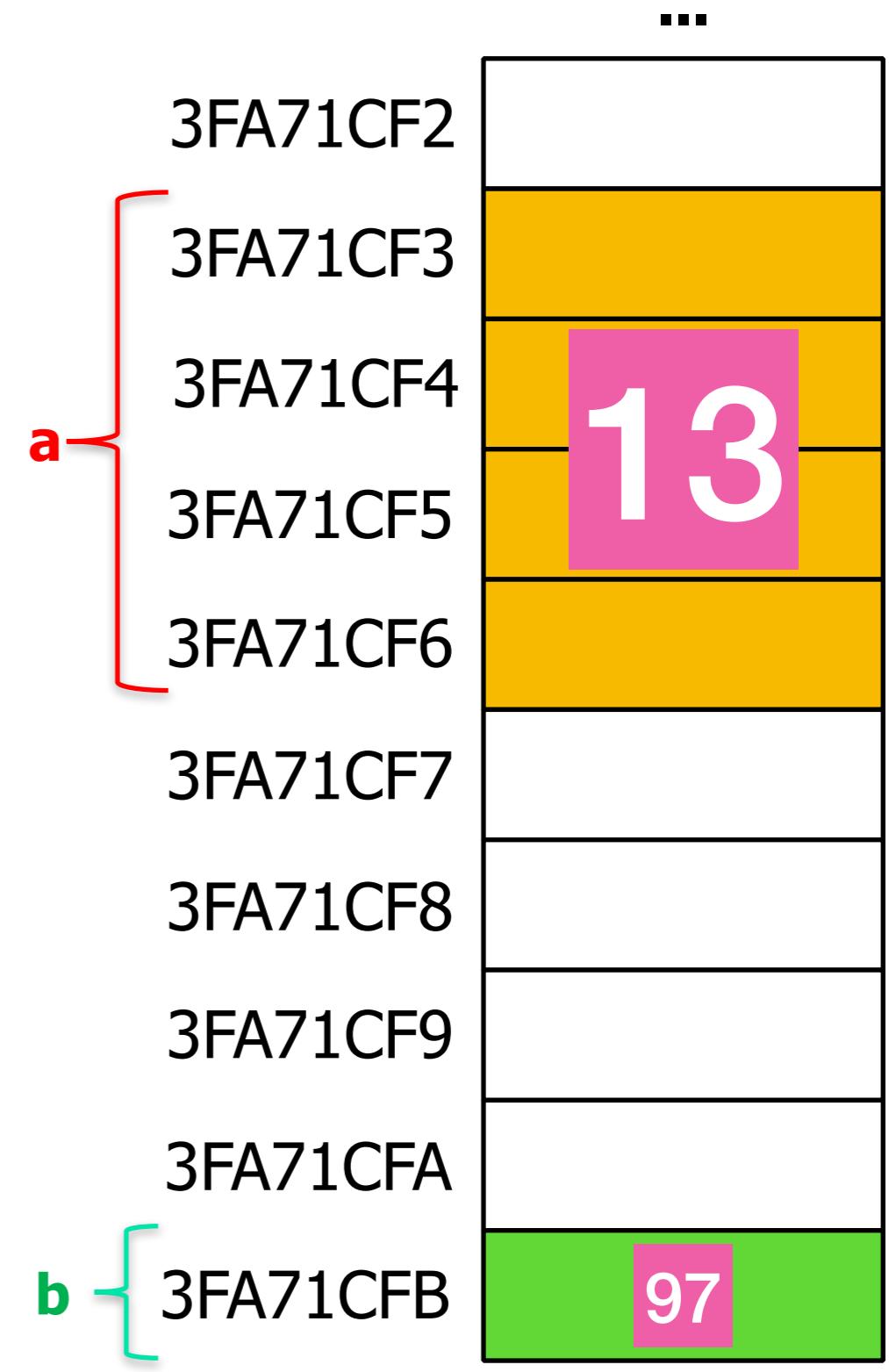
```
#include <stdio.h>

int main () {
    int a, *p;
    char b;

    a = 13;
    b = 'a';
    p = &a;
    printf("%d\n", a);
    printf("%p\n", &a);
    printf("%p\n", p);

    return 0;
}
```

```
13
3FA71CF3
3FA71CF3
```



Tamanho do Ponteiro (cont.)

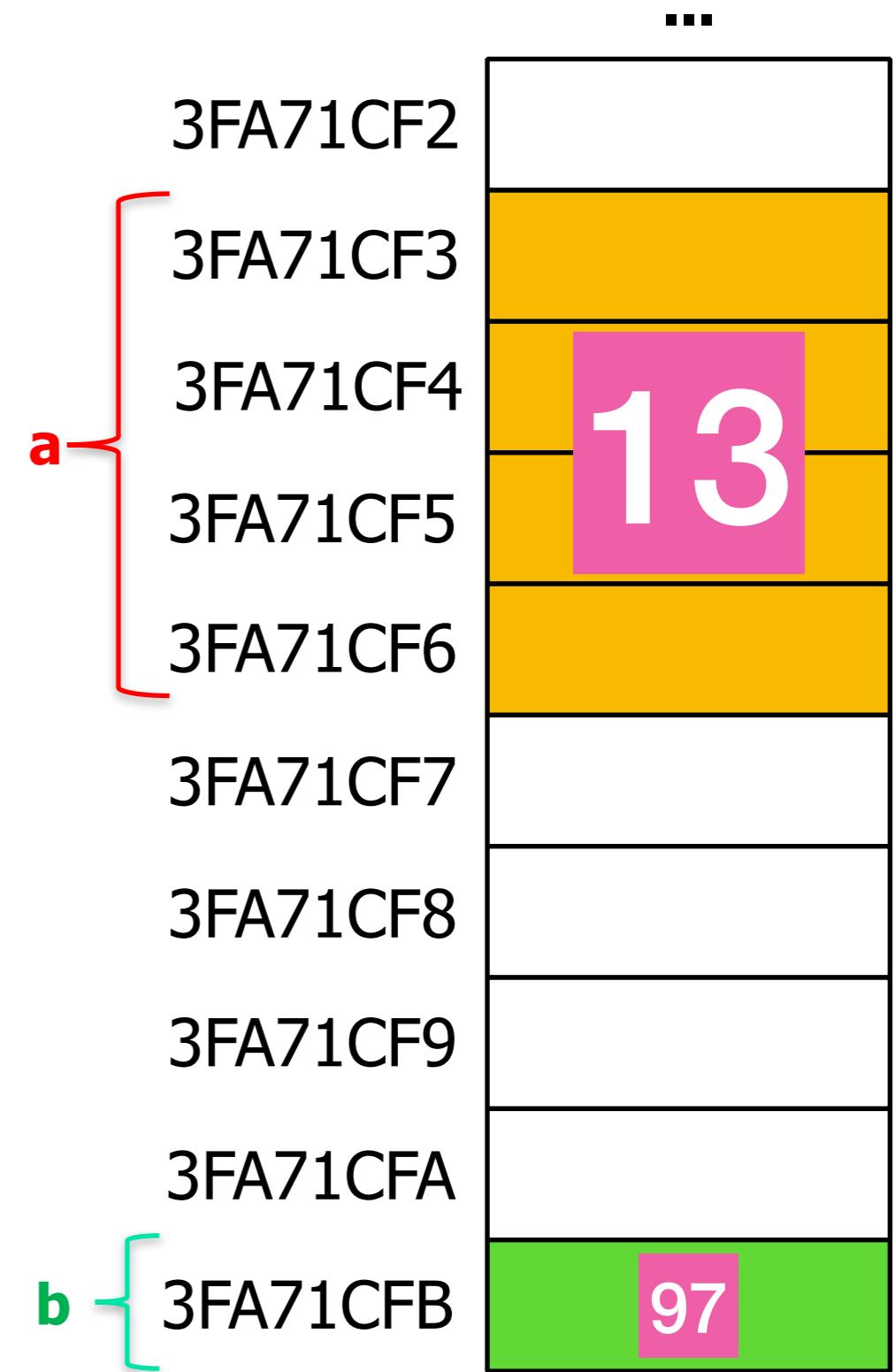
```
#include <stdio.h>

int main () {
    int a;
    char b, *p;

    a = 13;
    b = 'a';
    p = &b;
    printf("%c\n", b);
    printf("%p\n", &b);
    printf("%p\n", p);

    return 0;
}
```

```
'a'
3FA71CFB
3FA71CFB
```



Tipo do Ponteiro

- Por que os ponteiros do tipo **int** e **char** possuem o mesmo tamanho?
 - Oras .. ambos armazenam a mesma coisa; um endereço
- Então para que serve o tipo?
 - Serve para quando você referenciar *, o ponteiro saber **quantos bytes deve considerar** a partir do endereço inicial e como deve ser **interpretado** esses bytes;

Qualificador Const

Qualificador const

- Variáveis declaradas com qualificador **const** não podem ser alteradas
 - Constantes!!!
- Declare parâmetros **const** quando desejar que a função não altere o valor (por exemplo de um vetor)
- Tentar alterar uma variável **const** produz um erro

Qualificador const

- **const** também pode ser utilizado em conjunto com ponteiros

```
<const> int * <const> myPtr = &x;
```

- Observe que o **const** pode aparecer em dois locais com significados diferentes

Qualificador const (cont.)

- `int *const myPtr = &x;`
 - Ponteiro constante para um **int** (uma vez atribuído um endereço para o ponteiro, não pode ser alterado)
- `const int *myPtr = &x;`
 - Ponteiro para um **const int** (pode-se alterar para onde o ponteiro está apontando, porém não pode-se alterar o conteúdo)
- `const int *const myPtr = &x;`
 - Ponteiro constante para um **const int**

```
1 /* Fig. 7.10: fig07_10.c
2  Converting lowercase letters to uppercase letters
3  using a non-constant pointer to non-constant data */
4
5 #include <stdio.h>
6 #include <ctype.h>
7
8 void convertToUppercase( char *sPtr ); /* prototype */
9
10 int main( void )
11 {
12     char string[] = "characters and $32.98"; /* initialize char array */
13
14     printf( "The string before conversion is: %s", string );
15     convertToUppercase( string );
16     printf( "\nThe string after conversion is: %s\n", string );
17
18     return 0; /* indicates successful termination */
19
20 } /* end main */
21
```

Ambos **sPtr** e ***sPtr** são modificáveis

fig07_10.c

(1 of 2)

```
22 /* convert string to uppercase letters */  
23 void convertToUppercase( char *sPtr )  
24 {  
25     while ( *sPtr != '\0' ) { /* current character is not '\0' */  
26  
27         if ( islower( *sPtr ) ) { /* if character is lowercase, */  
28             *sPtr = toupper( *sPtr ); /* convert to uppercase */  
29         } /* end if */  
30  
31         ++sPtr; /* move sPtr to the next character */  
32     } /* end while */  
33  
34 } /* end function convertToUppercase */
```

Ambos **sPtr** e ***sPtr** são modificados pela função **convertToUppercase**

The string before conversion is: characters and \$32.98
The string after conversion is: CHARACTERS AND \$32.98

```
1 /* Fig. 7.11: fig07_11.c
2  Printing a string one character at a time using
3  a non-constant pointer to co
4
5 #include <stdio.h>
6
7 void printCharacters( const char *sPtr );
8
9 int main( void )
10 {
11     /* initialize char array */
12     char string[] = "print characters of a string";
13
14     printf( "The string is:\n" );
15     printCharacters( string );
16     printf( "\n" );
17
18     return 0; /* indicates successful termination */
19
20 } /* end main */
21
```

Variável ponteiro **sPtr** é modificável, mas o dado que aponta, ***sPtr**, não é

fig07_11.c

(1 of 2)

```
22 /* sPtr cannot modify the character to which it points,  
23   i.e., sPtr is a "read-only" pointer */  
24 void printCharacters( const char *sPtr )  
25 {  
26     /* loop through entire string */  
27     for ( ; *sPtr != '\0'; sPtr++ ) { /* no initialization */  
28         printf( "%c", *sPtr );  
29     } /* end for */  
30 } /* end function printCharacters */
```

fig07_11.c

(2 of 2)

sPtr é modificado pela função **printCharacters**

The string is:

print characters of a string

```

1 /* Fig. 7.12: fig07_12.c
2  Attempting to modify data through a
3  non-constant pointer to constant data. */
4 #include <stdio.h>
5 void f( const int *xPtr ); /* prototype */
6
7 Variável ponteiro xPtr é modificável, mas o dado
8 int main( void )
9 {
10    int y;      /* define y */
11
12    f( &y );    /* f attempts illegal modification */
13
14    return 0;    /* indicates successful termination */
15
16 } /* end main */
17
18 /* xPtr cannot be used to modify the
19  value of the variable to which it points */
20 void f( const int *xPtr )
21 {
22    *xPtr = 100; /* error: cannot modify a const object */
23 } /* end function f */

```

fig07_12.c

***xPtr** tem o qualificador **const**, então tentar modificar seu valor causa um erro

Compiling...
FIG07_12.c
c:\books\2006\chtp5\examples\ch07\fig07_12.c(22) : error C2166: l-value
specifies const object
Error executing cl.exe.

FIG07_12.exe - 1 error(s), 0 warning(s)

```
1 /* Fig. 7.13: fig07_13.c
2  Attempting to modify a constant pointer to non-constant data */
3 #include <stdio.h>
4
5 int main( void )
6 {
7     int x; /* define x */
8     int y; /* define y */
9
10    /* ptr is a constant pointer to an integer that can be modified
11       through ptr, but ptr always points to the same */  
Ponteiro ptr não é modificável, mas o dado que  
aponta, *ptr, pode ser alterado
12    int * const ptr = &x;
13
14    *ptr = 7; /* allowed: *ptr is not const */
15    ptr = &y; /* error: ptr is const; cannot assign new address */
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

```
Compiling...
FIG07_13.c
c:\books\2006\chtp5\Examples\ch07\FIG07_13.c(15) : error C2166: l-value
    specifies const object
Error executing cl.exe.

FIG07_13.exe - 1 error(s), 0 warning(s)
```

fig07_14.c

```
1 /* Fig. 7.14: fig07_14.c
2  Attempting to modify a constant pointer to constant data. */
3 #include <stdio.h>
4
5 int main( void )
6 {
7     int x = 5; /* initialize x */
8     int y;      /* define y */
9
10    /* ptr is a constant pointer to a constant integer. ptr always
11       points to the same location; the integer at that location
12       cannot be modified */
13    const int *const ptr = &x;
14
15    printf( "%d\n", *ptr );
16
17    *ptr = 7; /* error: *ptr is const; cannot assign new value */
18    ptr = &y; /* error: ptr is const; cannot assign new address */
19
20    return 0; /* indicates successful termination */
21
22 } /* end main */
```

Nem o ponteiro **sPtr** nem o dado que aponta,
***sPtr**, é modificável

Compiling...

FIG07_14.c

```
c:\books\2006\chtp5\Examples\ch07\FIG07_14.c(17) : error C2166: l-value
    specifies const object
```

```
c:\books\2006\chtp5\Examples\ch07\FIG07_14.c(18) : error C2166: l-value
    specifies const object
```

```
Error executing cl.exe.
```

```
FIG07_12.exe - 2 error(s), 0 warning(s)
```

Aritmética em Ponteiros

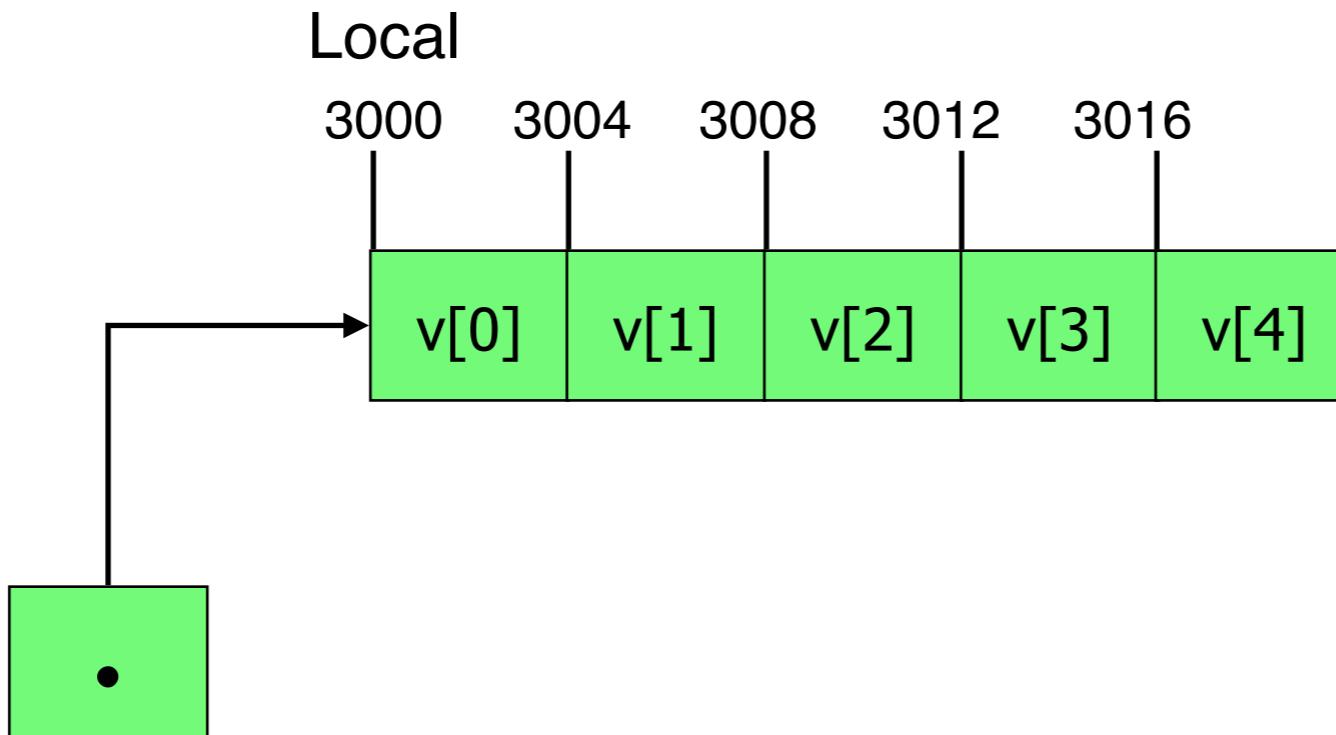
Aritmética de Ponteiros

- Considere v com 5 elementos em uma máquina onde um **int** possui 4 bytes
- Considere $vPtr$ um ponteiro do tipo **int**

```
vPtr = v // vPtr aponta para o primeiro elemento v[0]
vPtr += 2 // soma ao vPtr 8 bytes
```

Aritmética de Ponteiros (cont.)

`vPtr = v // vPtr aponta para o primeiro elemento v[0]`

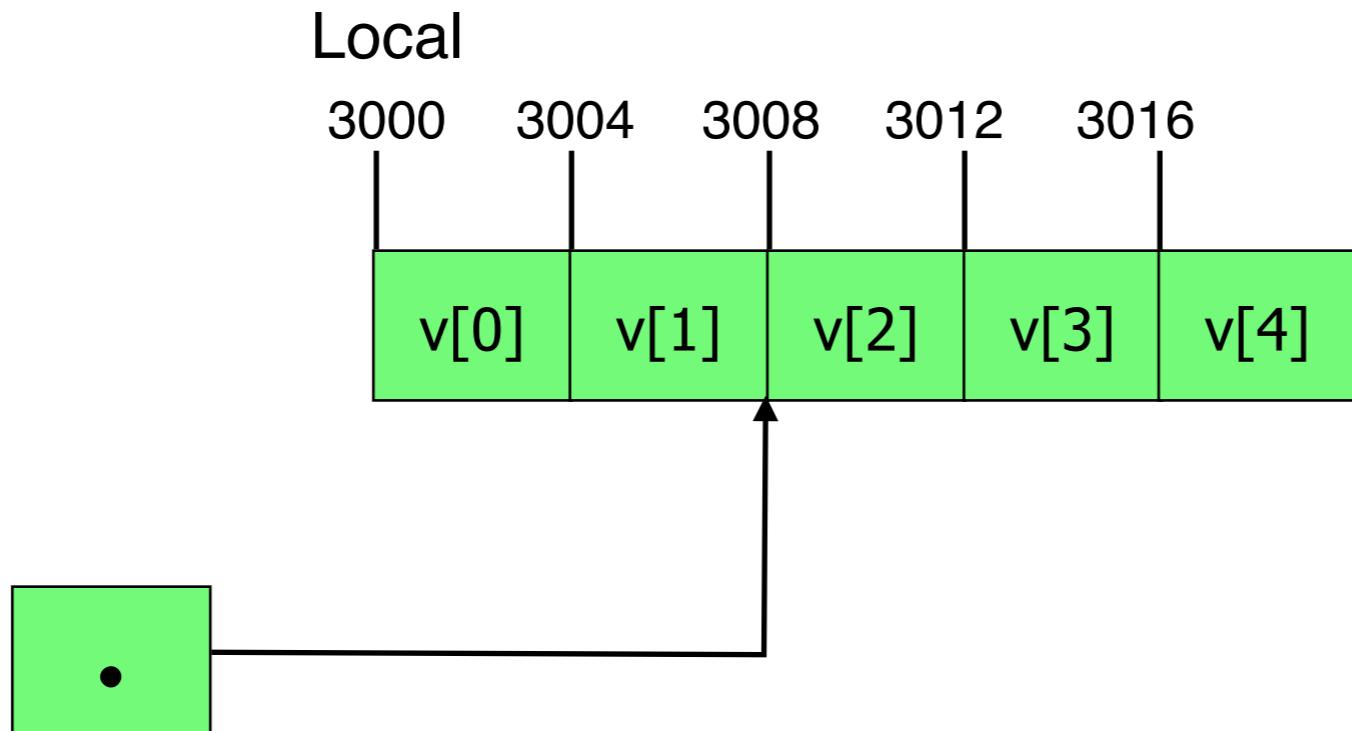


Variável ponteiro `vPtr`

Array `v` e uma variável ponteiro `vPtr` que aponta para `v`.

Aritmética de Ponteiros (cont.)

```
vPtr = v // vPtr aponta para o primeiro elemento v[0]
vPtr += 2 // soma ao vPtr 8 bytes
```



Variável ponteiro vPtr

O ponteiro vPtr depois da aritmética de ponteiros.

Ponteiros e Arrays

Relação entre Ponteiros e Arrays

- Ponteiros e arrays estão altamente relacionados
 - Arrays são como ponteiros constantes
 - Ponteiros podem utilizar a notação de acesso de arrays

Exemplo Ponteiro e Array

```
int main () {
    int v[5] = {123, 345, 678, 146, 121};
    int *p;

    // p aponta para v (alternativa 1)
    p = v;

    // p aponta para v (alternativa 2)
    p = &v[0];

    return 0;
}
```

Exemplo Notação de Acesso

```
#include <stdio.h>

int main () {
    int v[5] = {123, 345, 678, 146, 121};
    int *p;

    // p aponta para v (alternativa 1)
    p = v;

    // modos de acessar o elemento v[3]

    // notacao array
    printf("%d %d\n", v[3], p[3]);

    // notacao ponteiro offset
    printf("%d %d\n", *(v+3), *(p+3));

    return 0;
}
```

```
% ./a.out
146 146
146 146
```

```

1  /* Fig. 7.20: fig07_20.cpp
2   Using subscripting and pointer notations with arrays */
3
4 #include <stdio.h>
5
6 int main( void )
7 {
8     int b[] = { 10, 20, 30, 40 }; /* initialize array b */
9     int *bPtr = b;                /* set bPtr to point to array b */
10    int i;                      /* counter */
11    int offset;                 /* counter */
12
13    /* output array b using array subscript notation */
14    printf( "Array b printed with:\nArray subscript notation\n" );
15
16    /* loop through array b */
17    for ( i = 0; i < 4; i++ ) {
18        printf( "b[ %d ] = %d\n", i, b[ i ] );
19    } /* end for */
20
21    /* output array b using array name and pointer/offset notation */
22    printf( "\nPointer/offset notation where\n"
23           "the pointer is the array name\n" );
24
25    /* loop through array b */
26    for ( offset = 0; offset < 4; offset++ ) {
27        printf( "* ( b + %d ) = %d\n", offset, *( b + offset ) );
28    } /* end for */
29

```

Outline

fig07_20.c

(1 of 3)

Notação array subscript

Notação Pointeiro/offset

```

30  /* output array b using bPtr and array subscript notation */
31  printf( "\nPointer subscript notation\n" );
32
33  /* loop through array b */
34  for ( i = 0; i < 4; i++ ) {
35      printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
36  } /* end for */
37
38  /* output array b using bPtr and pointer/offset notation */
39  printf( "\nPointer/offset notation\n" );
40
41  /* loop through array b */
42  for ( offset = 0; offset < 4; offset++ ) {
43      printf( "*(% bPtr + %d) = %d\n", offset, *( bPtr + offset ) );
44  } /* end for */
45
46  return 0; /* indicates successful termination */
47
48 } /* end main */

```

Array b printed with:

Array subscript notation

```

b[ 0 ] = 10
b[ 1 ] = 20
b[ 2 ] = 30
b[ 3 ] = 40

```

Notação ponteiro subscript

Notação ponteiro offset

Outline

fig07_20.c

(2 of 3)

(continued on next slide...)

Pointer/offset notation where
the pointer is the array name

```
* ( b + 0 ) = 10
* ( b + 1 ) = 20
* ( b + 2 ) = 30
* ( b + 3 ) = 40
```

fig07_20.c
(3 of 3)

Pointer subscript notation

```
bPtr[ 0 ] = 10
bPtr[ 1 ] = 20
bPtr[ 2 ] = 30
bPtr[ 3 ] = 40
```

Pointer/offset notation

```
* ( bPtr + 0 ) = 10
* ( bPtr + 1 ) = 20
* ( bPtr + 2 ) = 30
* ( bPtr + 3 ) = 40
```

```
1 /* Fig. 7.21: fig07_21.c
2   Copying a string using array notation and pointer notation. */
3 #include <stdio.h>
4
5 void copy1( char * const s1, const char * const s2 ); /* prototype */
6 void copy2( char *s1, const char *s2 ); /* prototype */
7
8 int main( void )
9 {
10    char string1[ 10 ];           /* create array string1 */
11    char *string2 = "Hello";     /* create a pointer to a string */
12    char string3[ 10 ];           /* create array string3 */
13    char string4[] = "Good Bye"; /* create a pointer to a string */
14
15    copy1( string1, string2 );
16    printf( "string1 = %s\n", string1 );
17
18    copy2( string3, string4 );
19    printf( "string3 = %s\n", string3 );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
24
```

Outline

fig07_21.c

(1 of 2)

Outline

fig07_21.c

(2 of 2)

```
25 /* copy s2 to s1 using array notation */
26 void copy1( char * const s1, const char * const s2 )
27 {
28     int i; /* counter */
29
30     /* loop through strings */
31     for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ ) {
32         ; /* do nothing in body */
33     } /* end for */
34
35 } /* end function copy1 */
36
37 /* copy s2 to s1 using pointer notation */
38 void copy2( char *s1, const char *s2 )
39 {
40     /* loop through strings */
41     for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ ) {
42         ; /* do nothing in body */
43     } /* end for */
44
45 } /* end function copy2 */
```

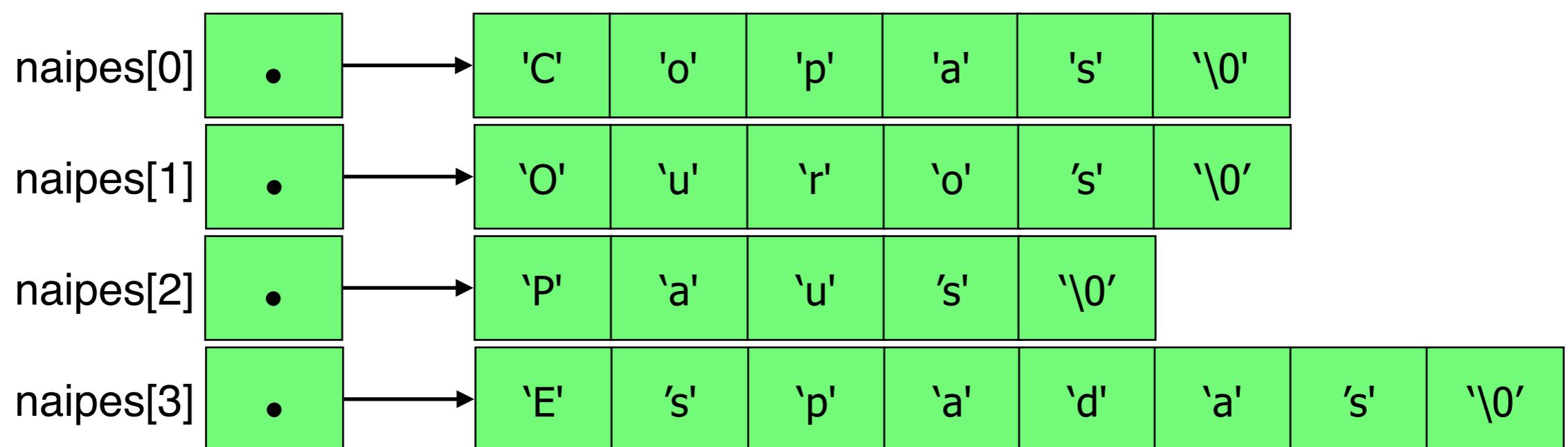
Condição do laço **for**

```
string1 = Hello
string3 = Good Bye
```

Vetor de Ponteiros

- `char *naipes[4] = {"Copas", "Ouros", "Paus", "Espadas"};`
- **Strings** são ponteiros para o primeiro caractere
- Cada elemento de **suit** é um ponteiro para um char
- As strings não são realmente armazenadas no array **suit**, apenas ponteiros para as strings são armazenadas

Figura: Vetor de ponteiros



Representação gráfica do vetor `naipes`.

Ponteiro para Função

Ponteiro para Função

- **Ponteiro** armazena endereço de uma função
- Análogo a nome de array endereçar o primeiro elemento
- Nome da função é endereço inicial do código que a define
- Ponteiro de funções podem ser
 - Passado para funções
 - Armazenado em arrays
 - Atribuído para outros ponteiros de função

Exemplo: Bubble sort

- Função **bubble sort** recebe um ponteiro para função
 - **Bubble** chama uma função auxiliar que determina se a ordenação é ascendente ou descendente
 - Um dos argumentos em **bubble sort** é um ponteiro para função:
`int (*compare) (int a, int b)`

```
1 /* Fig. 7.26: fig07_26.c
2   Multipurpose sorting program using function pointers */
3 #include <stdio.h>
4 #define SIZE 10
5
6 /* prototypes */
7 void bubble( int work[], const int size, int (*compare)( int a, int b ) );
8 int ascending( int a, int b );
9 int descending( int a, int b );
10
11 int main( void )
12 {
13     int order; /* 1 for ascending order or 2 for descending order */
14     int counter; /* counter */
15
16     /* initialize array a */
17     int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
18
19     printf( "Enter 1 to sort in ascending order,\n"
20             "Enter 2 to sort in descending order: " );
21     scanf( "%d", &order );
22
23     printf( "\nData items in original order\n" );
24
25     /* output original array */
26     for ( counter = 0; counter < SIZE; counter++ ) {
27         printf( "%5d", a[ counter ] );
28     } /* end for */
29
```

função **bubble** recebe um ponteiro
para função como argumento

```
30  /* sort array in ascending order; pass function ascending as an
31   argument to specify ascending sorting order */
32  if ( order == 1 ) {
33      bubble( a, SIZE, ascending );
34      printf( "\nData items in ascending order\n" );
35  } /* end if */
36  else { /* pass function descending */
37      bubble( a, SIZE, descending );
38      printf( "\nData items in descending order\n" );
39  } /* end else */
40
41 /* output sorted array */
42 for ( counter = 0; counter < SIZE; counter++ ) {
43     printf( "%5d", a[ counter ] );
44 } /* end for */
45
46 printf( "\n" );
47
48 return 0; /* indicates successful termination */
49
50 } /* end main */
```

dependendo da escolha do usuário, a função **bubble** uses either a função **ascending** ou **descending** para ordenar o array

Outline

fig07_26.c

(3 of 4)

```
52 /* multipurpose bubble sort; parameter compare is a pointer to
53   the comparison function that determines sorting order */
54 void bubble( int work[], const int size, int (*compare)( int a, int b ) )
55 {
56   int pass; /* pass counter */
57   int count; /* comparison counter */
58
59   void swap( int *element1Ptr, int *element2ptr ); /* prototype */
60
61   /* loop to control passes */
62   for ( pass = 1; pass < size; pass++ ) {
63
64     /* loop to control number of comparisons per pass */
65     for ( count = 0; count < size - 1; count++ ) {
66
67       /* if adjacent elements are out of order, swap them */
68       if ( (*compare)( work[ count ], work[ count + 1 ] ) ) {
69         swap( &work[ count ], &work[ count + 1 ] );
70       } /* end if */
71
72     } /* end for */
73
74   } /* end for */
75
76 } /* end function bubble */
77
```

Note que o que o programa considera "fora de ordem" é dependente do ponteiro da função que foi passado para a função **bubble**

```
78 /* swap values at memory locations to which element1Ptr and
79   element2Ptr point */
80 void swap( int *element1Ptr, int *element2Ptr )
81 {
82     int hold; /* temporary holding variable */
83
84     hold = *element1Ptr;
85     *element1Ptr = *element2Ptr;
86     *element2Ptr = hold;
87 } /* end function swap */
88
89 /* determine whether elements are out of order for an ascending
90   order sort */
91 int ascending( int a, int b ) ←
92 {
93     return b < a; /* swap if b is less than a */
94
95 } /* end function ascending */
96
97 /* determine whether elements are out of order for a descending
98   order sort */
99 int descending( int a, int b ) ←
100 {
101     return b > a; /* swap if b is greater than a */
102
103 } /* end function descending */
```

Passar para **bubble** a função **ascending**
irá apontar o programa aqui

Passar para **bubble** a função **descending**
irá apontar o programa aqui

Enter 1 to sort in ascending order,

Enter 2 to sort in descending order: 1

Data items in original order

2 6 4 8 10 12 89 68 45 37

Data items in ascending order

2 4 6 8 10 12 37 45 68 89

Enter 1 to sort in ascending order,

Enter 2 to sort in descending order: 2

Data items in original order

2 6 4 8 10 12 89 68 45 37

Data items in descending order

89 68 45 37 12 10 8 6 4 2

```
1 /* Fig. 7.28: fig07_28.c
2   Demonstrating an array of pointers to functions */
3 #include <stdio.h>
4
5 /* prototypes */
6 void function1( int a );
7 void function2( int b );
8 void function3( int c );
9
10 int main( void )
11 {
12     /* initialize array of 3 pointers to functions that each take an
13        int argument and return void */
14     void (*f[ 3 ])( int ) = { function1, function2, function3 };
15
16     int choice; /* variable to hold user's choice */
17
18     printf( "Enter a number between 0 and 2, 3 to end: " );
19     scanf( "%d", &choice );
20 }
```

Array de ponteiros para funções

Outline

```
21  /* process user's choice */
22  while ( choice >= 0 && choice < 3 ) {
23
24      /* invoke function at location choice in array f and pass
25          choice as an argument */
26      (*f[ choice ])( choice );
27
28      printf( "Enter a number between 0 and 2, 3 to end: " );
29      scanf( "%d", &choice );
30  } /* end while */
31
32  printf( "Program execution completed.\n" );
33
34  return 0; /* indicates successful termination */
35
36 } /* end main */
37
38 void function1( int a )
39 {
40     printf( "You entered %d so function1 was called\n\n", a );
41 } /* end function1 */
42
43 void function2( int b )
44 {
45     printf( "You entered %d so function2 was called\n\n", b );
46 } /* end function2 */
```

fig07_28.c

(2 of 3)

Função chamada é dependente da escolha do usuário

47

```
48 void function3( int c )
49 {
50     printf( "You entered %d so function3 was called\n\n", c );
51 } /* end function3 */
```

Enter a number between 0 and 2, 3 to end: 0

You entered 0 so function1 was called

Enter a number between 0 and 2, 3 to end: 1

You entered 1 so function2 was called

Enter a number between 0 and 2, 3 to end: 2

You entered 2 so function3 was called

Enter a number between 0 and 2, 3 to end: 3

Program execution completed.

Referências

- DEITEL, C Como Programar 6^a Edição. Pearson;
- FEOFILOFF, P. Algoritmos em Linguagem C, 1. ed. Rio de Janeiro: Elsevier, 2008;
- PIVA, D.J. et al. Algoritmos e programação de computadores. Rio de Janeiro: Elsevier, 2012;
- WIRTH, Niklaus. Algoritmos e estrutura de dados. Rio de Janeiro, RJ: LTC, 2009. 255p;