

# Ponteiros

## Funções e Arquivos

---

Prof. Edson Alves - UnB/FGA

2018

1. Ponteiros e funções
2. Ponteiros e arquivos

# Ponteiros e funções

---

# Passagem de parâmetros

- Os parâmetros das funções e métodos podem ser passados de duas maneiras: por valor ou por referência
- Na passagem por valor, os valores das variáveis passadas para a função ou método são copiados para os parâmetros formais
- Na passagem por referência, as funções recebem em seus parâmetros referências para as variáveis passadas como parâmetros
- A passagem por cópia tem maior custo de execução do que a passagem por referência, mas garante que as variáveis passadas como parâmetros não serão modificadas pela função

# Passagem de parâmetros em C/C++

- Em C, a única forma de passagem de parâmetros é por valor
- C++ suporta ambos métodos de passagem
- Para se utilizar a passagem por referência em C, deve-se declarar o parâmetro como um ponteiro para o tipo de dado desejado
- Efetivamente, a "passagem por referência" do C é uma passagem por cópia, onde o ponteiro é copiado e o acesso à variável é feita através do ponteiro
- Embora o mesmo seja verdadeiro para C++, a linguagem suporta passagem por referência nativamente
- O mecanismo de referência de C++ simplifica a notação, mas deixa apenas implícito que os valores podem ser modificados pela função

## Exemplo de passagem de parâmetros em C

```
1 #include <stdio.h>
2 #include <time.h>
3
4 typedef struct _Registro {
5     char nome[128];
6     char sexo;
7     float peso;
8     int idade;
9 } Registro;
10
11 void por_valor(Registro registro) {}
12 void por_referencia(Registro *registro) {}
13
14 static const unsigned long quantidade_chamadas = 1e8;
15
16 int main()
17 {
18     Registro registro;
19     time_t inicio, fim;
20     unsigned long i;
21
```

## Exemplo de passagem de parâmetros em C

```
22     inicio = clock();
23
24     for (i = 0; i < quantidade_chamadas; i++)
25         por_valor(registro);
26
27     fim = clock();
28
29     printf("Valor: %.8f s\n", 1.0*(fim - inicio)/CLOCKS_PER_SEC);
30
31     inicio = clock();
32
33     for (i = 0; i < quantidade_chamadas; i++)
34         por_referencia(&registro);
35
36     fim = clock();
37
38     printf("Referência: %.8f s\n", 1.0*(fim - inicio)/CLOCKS_PER_SEC);
39
40     return 0;
41 }
```

# Exemplo de passagem de parâmetros em C++

```
1 #include <iostream>
2
3 using namespace std;
4
5 void swap_por_valor(int x, int y) {
6     int temp = x;
7     x = y;
8     y = temp;
9 }
10
11 void swap_com_ponteiros(int *x, int *y) {
12     int temp = *x;
13     *x = *y;
14     *y = temp;
15 }
16
17 void swap_por_referencia(int& x, int& y) {
18     int temp = x;
19     x = y;
20     y = temp;
21 }
```



# Exemplo de passagem de parâmetros em C++

```
23 int main()
24 {
25     int x = 1, y = 2;
26
27     cout << "Valores iniciais: x = " << x << ", y = " << y << endl;
28
29     swap_por_valor(x, y);
30     cout << "swap_por_valor(): x = " << x << ", y = " << y << endl;
31
32     swap_por_referencia(x,y);
33     cout << "swap_por_referencia(): x = " << x << ", y = " << y << endl;
34
35     swap_com_ponteiros(&x, &y);
36     cout << "swap_com_ponteiros(): x = " << x << ", y = " << y << endl;
37
38     return 0;
39 }
```

# Ponteiros para funções

## Sintaxe para declaração de ponteiros para funções

```
tipo_retorno (*nome_ponteiro)([parametros]);
```

- O ponto de entrada de uma função ocupa uma posição na memória: logo podemos declarar ponteiros para funções
- O endereço de uma função pode ser obtido através do nome da função, como no caso de vetores
- O ponteiro para função pode ser usado para fazer uma chamada a função, com a mesma sintaxe de chamadas comuns. Exemplo:  

```
void * (*ptr)(size_t, size_t);  
ptr = calloc;  
...  
int *p = (int *) ptr(2, 8);
```
- Ponteiros para funções permitem a construção de programas dinâmicos e podem aumentar a organização do código

## Exemplo de uso de ponteiros de funções

```
1 #ifndef PLUGIN_H
2 #define PLUGIN_H
3
4 typedef struct _Plugin {
5     void *handle;
6     const char * (*symbol) ();
7     int (*operation) (int, int);
8 } Plugin;
9
10 extern Plugin * load(const char *path);
11 extern void release(Plugin *plugin);
12
13 #endif
```

## Exemplo de uso de ponteiros de funções

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <dlfcn.h>
4 #include "plugin.h"
5
6 Plugin * load(const char *path)
7 {
8     void *handle = NULL;
9     Plugin *plugin = NULL;
10    const char * (*symbol)() = NULL;
11    int (*operation) (int, int) = NULL;
12
13    if (!path)
14        return NULL;
15
16    handle = dlopen(path, RTLD_LAZY | RTLD_GLOBAL);
17
18    if (!handle) {
19        fprintf(stderr, "Load plugin error: %s\n", dlerror());
20        return NULL;
21    }
```

## Exemplo de uso de ponteiros de funções

```
23     operation = dlsym(handle, "operation");
24     symbol = dlsym(handle, "symbol");
25
26     if (!operation || !symbol) {
27         fprintf(stderr, "Load plugin error: %s\n", dlerror());
28         dlclose(handle);
29         return NULL;
30     }
31
32     plugin = (Plugin *) malloc(sizeof(Plugin));
33
34     if (!plugin) {
35         dlclose(handle);
36         return NULL;
37     }
38
39     plugin->handle = handle;
40     plugin->symbol = symbol;
41     plugin->operation = operation;
42     return plugin;
43 }
```

## Exemplo de uso de ponteiros de funções

```
45 void release(Plugin *plugin)
46 {
47     if (!plugin)
48         return;
49
50     if (plugin->handle)
51         dlclose(plugin->handle);
52
53     free(plugin);
54 }
```

# Exemplo de uso de ponteiros de funções

```
1 /* soma.c
2  *
3  * Compile com a linha abaixo:
4  *
5  * $ gcc -o soma.so -shared soma.c -fPIC
6  */
7 #include <stdlib.h>
8 #include "plugin.h"
9
10 const char * symbol()
11 {
12     return "+";
13 }
14
15 int operation(int a, int b)
16 {
17     return a + b;
18 }
```

# Exemplo de uso de ponteiros de funções

```
1 /* subtracao.c
2  *
3  * Compile com a linha abaixo:
4  *
5  * $ gcc -o subtracao.so -shared subtracao.c -fPIC
6  */
7 #include <stdlib.h>
8 #include "plugin.h"
9
10 const char * symbol()
11 {
12     return "-";
13 }
14
15 int operation(int a, int b)
16 {
17     return a - b;
18 }
```



## Exemplo de uso de ponteiros de funções

```
1 /* calculadora.c
2 *
3 * Compile com a linha abaixo:
4 *
5 * $ gcc -o calculadora calculadora.c plugin.c -I. -ldl
6 *
7 * Os plugins devem estar no diretório 'plugins'.
8 */
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12 #include <sys/types.h>
13 #include <dirent.h>
14 #include "plugin.h"
15
16 Plugin ** load_plugins(const char *directory, int *qtd) {
17     DIR *dir = NULL;
18     Plugin **plugins;
19     struct dirent *entry;
20     int size = 0;
21     char path[4096];
```

## Exemplo de uso de ponteiros de funções

```
23     dir = opendir(directory);
24
25     if (!dir)
26         return NULL;
27
28     while (entry = readdir(dir))
29         size++;
30
31     closedir(dir);
32
33     plugins = (Plugin **) calloc(size, sizeof(Plugin *));
34
35     if (!plugins) {
36         fprintf(stderr, "Sem memoria\n");
37         return NULL;
38     }
39
40     *qtd = 0;
41     dir = opendir(directory);
42
43     if (!dir) return plugins;
```

## Exemplo de uso de ponteiros de funções

```
45     while (entry = readdir(dir)) {
46         if (!strcmp(entry->d_name, ".") || !strcmp(entry->d_name, ".."))
47             continue;
48
49         sprintf(path, "%s/%s", directory, entry->d_name);
50         plugins[*qtd] = load(path);
51
52         if (plugins[*qtd]) {
53             (*qtd)++;
54             printf("Plugin %s loaded\n", path);
55         }
56     }
57
58     closedir(dir);
59
60     return plugins;
61 }
62
```

## Exemplo de uso de ponteiros de funções

```
63 int main() {
64     Plugin **plugins = NULL;
65     char symbol[128], line[4096];
66     int x, y, qtd = 0, i;
67
68     plugins = load_plugins("plugins", &qtd);
69
70     if (!plugins) {
71         fprintf(stderr, "Can't load plugins\n");
72         return -1;
73     }
74
75     while (1) {
76         printf("Insira a expressao: ");
77         fgets(line, 4096, stdin);
78
79         if (!strcmp(line, "quit\n"))
80             break;
81
82         sscanf(line, "%d %s %d", &x, symbol, &y);
83     }
```

## Exemplo de uso de ponteiros de funções

```
84     for (i = 0; i < qtd; i++) {
85         if (!strcmp(symbol, plugins[i]->symbol())) {
86             printf("%d %s %d = %d\n", x, symbol,
87                 y, plugins[i]->operation(x, y));
88             break;
89         }
90     }
91
92     if (i == qtd)
93         printf("Operacao nao suportada\n");
94 }
95
96 for (i = 0; i < qtd; i++)
97     release(plugins[i]);
98
99 free(plugins);
100
101 return 0;
102 }
```

# Ponteiros e arquivos

---

# Leitura e escrita em arquivos em C

- Em C, a manipulação de arquivos é feita através da estrutura **FILE** e as funções associadas a ela, que fazem parte da biblioteca `stdio.h`
- A abertura de arquivos em C é feita através da função `fopen()`, cuja assinatura é

```
FILE *fopen(const char *path, const char *mode);
```

- Os parâmetros da função `fopen()` são o caminho para o arquivo (`path`) e o modo de abertura: “r” para leitura, “w” para escrita e “a” para anexar
- O retorno é um ponteiro para a estrutura **FILE**, que deve ser passada para as diversas funções que realizam a leitura e a escrita de dados em arquivo, além das funções de posicionamento do cursor

# Leitura e escrita em arquivos em C

- Entre as funções de escrita, as mais comuns são:  
`fprintf()`, `fputc()`, `fwrite()`
- Entre as funções de leitura, as mais comuns são:  
`fscanf()`, `fgetc()`, `fread()`
- As funções para manipulação do cursor são:  
`fseek()`, `ftell()`, `rewind()`
- Ao finalizar o uso de um arquivo, a estrutura **FILE** deve ser desalocada através da chamada da função `fclose()`
- Em C++, a manipulação de arquivos é feita pelas classes da biblioteca `fstream`
- Uma vez inicializados os fluxos de entrada e saída da `fstream`, a leitura e a escrita são feitas da mesma forma que são feitas com as classes `cin` e `cout`



# Exemplo de manipulação de arquivos em C

```
1 #ifndef CRIPTO_H
2 #define CRIPTO_H
3
4 typedef char byte;
5
6 extern byte cipher(byte message, byte key);
7 extern byte decipher(byte cipher, byte key);
8
9 #endif
```

# Exemplo de manipulação de arquivos em C

```
1 #include "cripto.h"
2
3 byte
4 cipher(byte message, byte key)
5 {
6     return message + key;
7 }
8
9 byte
10 decipher(byte cipher, byte key)
11 {
12     return cipher - key;
13 }
```

# Exemplo de manipulação de arquivos em C

```
1 /*
2  * Compile com a linha abaixo:
3  *
4  * $ gcc cripto.c main.c -o cipherfile
5  * $ gcc cripto.c main.c -o decipherfile
6  *
7  */
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11 #include "cripto.h"
12
13 int main(int argc, char *argv[]) {
14     FILE *in, *out;
15     byte (*op)(byte, byte) = NULL, key;
16     int c;
17
18     if (argc < 4) {
19         fprintf(stderr, "Uso: %s <entrada> <saida> <chave>\n", argv[0]);
20         return -1;
21     }
```

## Exemplo de manipulação de arquivos em C

```
23     in = fopen(argv[1], "r");
24
25     if (!in) {
26         fprintf(stderr, "Erro ao abrir %s para leitura\n", argv[1]);
27         return -2;
28     }
29
30     out = fopen(argv[2], "w");
31
32     if (!out)
33     {
34         fprintf(stderr, "Erro ao abrir %s para escrita\n", argv[2]);
35         fclose(in);
36         return -3;
37     }
38
39     if (!strcmp(argv[0], "./cipherfile"))
40         op = cipher;
41     else if (!strcmp(argv[0], "./decipherfile"))
42         op = decipher;
```

## Exemplo de manipulação de arquivos em C

```
43     else {
44         fprintf(stderr, "Operacao desconhecida\n");
45         fclose(in);
46         fclose(out);
47
48         return -4;
49     }
50
51     key = (byte) atoi(argv[3]);
52
53     while ((c = fgetc(in)) != EOF)
54         fputc(op((byte) c, key), out);
55
56     fclose(in);
57     fclose(out);
58
59     return 0;
60 }
```

1. **KERNIGHAN**, Bryan; **RITCHIE**, Dennis. *The C Programming Language*, 1978.
2. **SCHILDT**, Herbert. *C Completo e Total*, 1997.
3. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.
4. C Man Pages<sup>1</sup>.
5. C++ Reference<sup>2</sup>.

---

<sup>1</sup>Comando man no Linux.

<sup>2</sup><https://en.cppreference.com/w/>