Técnicas avançadas em C

By Natal

Tópicos da aula

- Ponteiro de funções
- Variáveis genéricas
- Funções de parâmetros variados

Ponteiro de funções

Quando queremos utilizar dentro de nossa função outra que pode se comportar de diferentes maneiras, dependendo do que desejamos.

Para isso, podemos utilizar **funções como parâmetro de outras funções**.

Ponteiro de funções

Ex:

- Busca por qualquer critério de uma estrutura
- Ordenação por diferentes ordens ou métodos (quick sort, qsort)
- Encapsulamento de operações

Ponteiro de funções

```
int opera(int a, int b, int op) {
        if (op == 1) // soma
                return a + b;
        if (op == 2) // subtração
                return a - b;
        if (op == 3) // multiplicação
                return a * b;
}
```

Ponteiro de funções

```
int opera(int a, int b, int (*op)(int, int)) {
        return op(a, b);
}

int soma(int a, int b) {
        return a + b;
}

// na main
opera(x, y, soma);
```

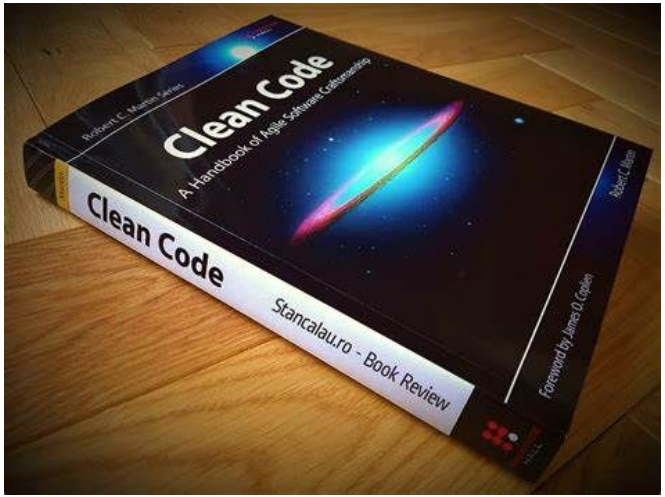
Ponteiro de funções

Neste caso, o código ficou maior, mas ficou mais claro!!

- Apesar de funções de 1 linha, cada uma possui apenas 1 tarefa bem definida

Em outros casos, essa técnica pode também evitar a repetição de código.

Tudo depende do bom senso do programador.



Ponteiro de funções

Outra notação:

```
typedef int (*func_op)(int, int);

int opera(int a, int b, func_op f) {
        return f(a, b);
}
```

Dúvidas e código

Variáveis genéricas

Variáveis do tipo **void*** (assim como **int***, **char***, etc) armazenam um endereço de memória.

Para essas variáveis, o endereço pode ser de qualquer tipo de dado. Portanto permite elaborar funções e métodos mais generalizados.

Variáveis genéricas

Ex:

- malloc, realloc, free
- fread, fwrite
- TAD

Variáveis genéricas

OBS:

sizeof(void *) = sizeof(int *) = sizeof(double *) = … = 4 bytes ou 8 bytes
(Depende se o computador é 32 ou 64 bits)

O que muda é a maneira como o endereço da memória é interpretado.

Variáveis genéricas

```
void *end = malloc(sizeof(int));
int *v = (int *)end;
```

```
double a = 2.3;
void *end_a = &a;
printf(“%lf”, *(double *)end_a);
```

Dúvidas e código

Funções de parâmetros variados

Quando não sabemos exatamente quantos parâmetros iremos utilizar em uma função.

Definiremos então um padrão que os **n** parâmetros passados irão seguir.

Funções de parâmetros variados

Ex:

- printf, fprintf
- scanf, fscanf

Funções de parâmetros variados

Para isso, utilizamos da biblioteca **stdarg.h** para termos as seguintes ferramentas:

- o tipo de dado (estrutura) **va_list**, que se refere à lista de parâmetros variáveis passados.
- a função **va_start()** (recebe o va_list e o último parâmetro que precede os parâmetros variáveis) para permitir o acesso aos parâmetros variáveis da função.

Funções de parâmetros variados

- a função **va_arg()** (recebe o va_list e o tipo da variável que se espera retornar) para retornar o próximo elemento da lista de parâmetros variáveis.
- a função **va_end()** (recebe o va_list) para finalizar o acesso à lista de parâmetros variáveis da função.

Funções de parâmetros variados

```
void printStrings(char *str1, …) {
        printf(“%s ”, str1);
        va_list args;
        va_start(args, str1);

        char *aux;
        while ((aux = va_arg(args, char*))) printf(“%s ”, aux);
        va_end(args);
}

// na main
printStrings(“ola”, “bixos”, “lindos”, “!!”, NULL);
```

Dúvidas e código