

Funções parte 1

Rafael Beserra Gomes

UFRN

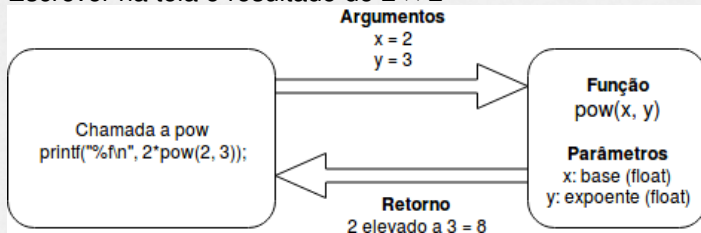
Material compilado em 6 de maio de 2018.

Licença desta apresentação:

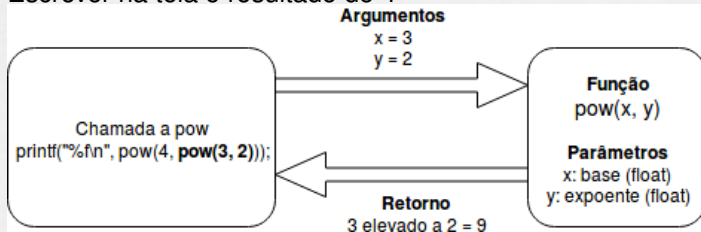


<http://creativecommons.org/licenses/>

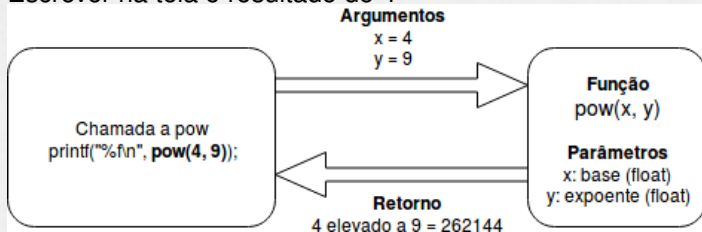
Escrever na tela o resultado de 2×2^3



Escrever na tela o resultado de 4^{3^2}



Escrever na tela o resultado de 4^3

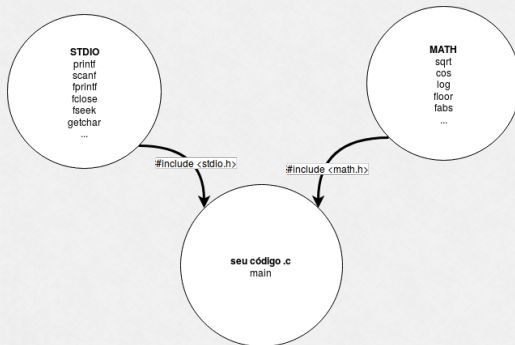


Já utilizamos (*chamamos*) várias funções:

- `pow(x, y)`: **retorna** x^y
- `sqrt(x)`: **retorna** a raiz quadrada de `x`
- `printf(...)`: escreve na saída padrão e **retorna** a quantidade de caracteres escritos
- `scanf(...)`: lê da entrada padrão e **retorna** a quantidade de elementos lidos com sucesso
- `malloc(x)`: reserva `x` bytes na memória e **retorna** o endereço base
- `srand(x)`: altera a semente para a geração de números aleatórios

Quais funções há em C e como utilizá-las?

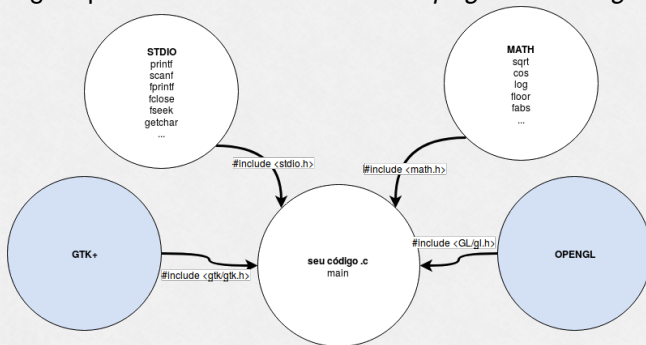
- as funções estão organizadas em **bibliotecas**
- verifique a **assinatura** da função
exemplo: **double sqrt(double x);**
- consulte **man 3 nomefuncao** ou
<http://www.cplusplus.com/reference/clibrary/>



Você pode incluir outras bibliotecas, exemplo:

- **gtk+**: para criar interfaces gráficas
- **opengl**: para gráficos 3D
- ...dentre tantas outras

Provavelmente exigirá a instalação da biblioteca (exemplo, para usar o gtk+ pode fazer no ubuntu: *sudo apt-get install libgtk-3-dev*)



Sintaxe para criar funções


```
<tipo de retorno>identificadorDaFuncao(<params separados por ,>) {  
    <instrução 1>  
    <instrução 2>  
    <...>  
    <instrução n>  
}
```

- **Assinatura da função:** o tipo de retorno + identificador da função + parâmetros
- A instrução **return** especifica o valor de retorno e encerra a função
- Escolha **void** caso nada queira retornar
- Funções dentro de funções são proibidas em ISO C!

Exemplos

Exemplo 1 (disponível no SIGAA):

```
1 #include <stdio.h>
2
3 int soma(int a, int b) {
4     return a + b;
5 }
6
7 int main() {
8
9     printf("3 + 5 = %d\n", soma(3, 5));
10    printf("4 + 7 = %d\n", soma(4, 7));
11
12    return 0;
13 }
```

- **Nome da função:** soma
- **Parâmetros:** a e b, ambos **int**
- **Retorno:** **int** o valor da soma a + b



Exercício em sala

Crie uma função chamada **maiorDeDois** que receba como **parâmetros** dois inteiros (**a** e **b**) e que **retorne** o maior deles

Por exemplo:

- **maiorDeDois(5, 2)** deve retornar 5
- **maiorDeDois(5, 7)** deve retornar 7

Exemplo 2 (disponível no SIGAA):

```
1 #include <stdio.h>
2
3 //retorna a + (a+1) + ... + (b-1) + b (sem usar formula fechada)
4 int somatorio(int a, int b) {
5     int soma = 0, i;
6     for(i = a; i <= b; i++) {
7         soma += i;
8     }
9     return soma;
10 }
11
12 int main() {
13
14     printf("Somatorio de 1 a 5 = %d\n", somatorio(1, 5));
15     printf("Somatorio de 10 a 40 = %d\n", somatorio(10, 40));
16
17     return 0;
18 }
```



Exercício em sala

Crie uma função chamada **escreverIntervalo** que receba como parâmetros dois inteiros (**a** e **b**) e que escreva na tela os números entre **a** e **b**. Não há valor de **retorno** (use **void**).

Por exemplo: **escreverIntervalo(2, 5)** deve escrever na tela 2 3 4 5.

Há boas e más escolhas na hora de implementar uma função
Em que aspectos as duas implementações a seguir não são boas?

```
1 int somaDoisNumeros() {  
2     scanf("%d", &x);  
3     scanf("%d", &y);  
4     return x + y;  
5 }
```

```
1 void somaDoisNumeros(int x, int y) {  
2     printf("%d\n", x + y);  
3 }
```

Por que usar funções?

- evita repetir código manualmente
- facilita alterações
- encapsulamento (não precisamos saber as instruções de uma função para usá-la!)
- melhor organização do código

Escopo

Escopo das variáveis:

- **variáveis locais**: parâmetros e variáveis declarados dentro de funções: podem ser usadas somente **dentro da função**¹.
- **variáveis globais**: variáveis declaradas fora das funções: podem ser usadas em qualquer parte após a declaração

```
int D; // Variável Global

int triplo(int C) {
    int B = 1; // Variável Local
    return B*C;
}

int * alocaVetorTamanho10() {
    int B = 10; // Variável Local
    int *v = malloc(sizeof(int)*B);
    return v;
}

int main() {
    int A = 4; // Variável Local
    int *u;

    printf("%d\n", triplo(A));
    u = alocaVetorTamanho10();

    return 0;
}
```

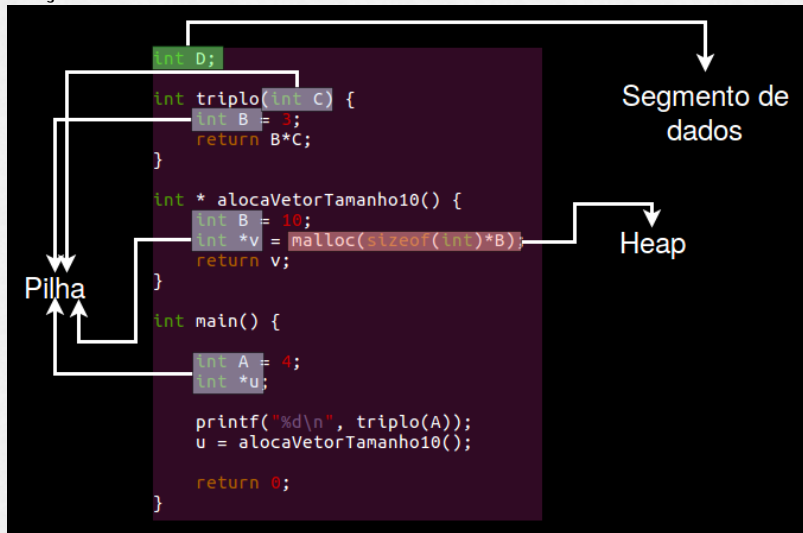
¹é possível acessar fora da função, mas não pelo identificador declarado

Pensando no ciclo de vida dos dados na memória:

- os dados podem estar na região da **pilha**, do **segmento de dados**, da **heap**

- as variáveis globais (declaradas fora das funções) são armazenadas no **segmento de dados** e valem por todo o programa
- as variáveis declaradas dentro das funções e parâmetros são armazenados na **pilha** e liberados ao encerrar a função
- o espaço de memória alocado na **heap** (usando malloc por exemplo) não é liberado ao encerrar a função

Exemplo de escopo de uma variável declarada dentro de uma função:



Passagem de parâmetros

Quando uma função é chamada, os argumentos são **copiados** para os parâmetros (não se referem ao mesmo dado na memória!)

```
1 #include <stdio.h>
2
3 void f(int x) {
4     x--;
5 }
6
7 int main() {
8
9     int k;
10    k = 3;
11
12    printf("k = %d\n", k);
13    f(k);
14    printf("k = %d\n", k);
15
16    return 0;
17 }
```

Disponível no SIGAA

Passando vetor como parâmetro (opção 1)

```
1 #include <stdio.h>
2
3 void funcao(int v[], int n) {
4     int i;
5     for(i = 0; i < n; i++)
6         printf("%d ", v[i]);
7     printf("\n");
8 }
9
10 int main() {
11
12     int vetor[] = {3, 4, 2, 1, 7};
13     funcao(vetor, 5);
14
15     return 0;
16 }
```

Disponível no SIGAA

Passando vetor como parâmetro (opção 2)

```
1 #include <stdio.h>
2
3 void funcao(int *v, int n) {
4     int i;
5     for(i = 0; i < n; i++)
6         printf("%d ", v[i]);
7     printf("\n");
8 }
9
10 int main() {
11
12     int vetor[] = {3, 4, 2, 1, 7};
13     funcao(vetor, 5);
14
15     return 0;
16 }
```

Disponível no SIGAA

A função main

- A função main é por padrão a executada quando o programa é carregado
- Retorna se houve falha no programa (útil para scripts). Então se o programa terminou com sucesso, retorne 0 (falso).
- Pode receber **argumentos de linha de comando**

Exemplo:

```
./a.out 3 teste 194
```

- útil para obter dados do usuário já na chamada do programa
- alguns comandos do bash utilizam argumentos de linha de comando

Teste o seguinte código (disponível no SIGAA):

```
1 #include <stdio.h>
2
3 int main(int argc, char **argv) {
4
5     int i;
6
7     printf("Quantidade de parametros: %d\n", argc);
8     for(i = 0; i < argc; i++) {
9         printf("Parametro %d: %s\n", i, argv[i]);
10    }
11
12    return 0;
13 }
```

argc é a quantidade de parâmetros de linha de comando e argv um vetor de strings com tais parâmetros

Teste o seguinte código (disponível no SIGAA):

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 int main(int argc, char **argv) {
6
7     if(strcmp(argv[1], "somar") == 0) {
8         printf("Soma: %d\n", atoi(argv[2]) + atoi(argv[3]));
9     } else {
10         if(strcmp(argv[1], "multiplicar") == 0) {
11             printf("Produto: %d\n", atoi(argv[2]) * atoi(argv[3]));
12         } else {
13             if(strcmp(argv[1], "subtrair") == 0) {
14                 printf("Subtracao: %d\n", atoi(argv[2]) - atoi(argv[3]));
15             } else {
16                 printf("Divisao: %d\n", atoi(argv[2])/atoi(argv[3]));
17             }
18         }
19     }
20
21     return 0;
22 }
```