INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação

ENSINO MÉDIO INTEGRADO - INFORMÁTICA
Disciplina de Linguagem de Programação
Aula 12: Função

Gil Eduardo de Andrade

Conceitos Preliminares

(https://docs.microsoft.com/pt-br/cpp/c-language/?view=msvc-170)

Definição

O conceito de função pode ser definido como um conjunto de comandos (linhas de código) agrupado num bloco devidamente nomeado, que ao ser invocado executa todas as instruções nele contidas. Ele permite o reaproveitamento do código construído, evitando que um mesmo trecho do programa seja repetido várias vezes (copiar + colar).

Principais Benefícios do Uso de Funções

A utilização do conceito de função possibilita que:

- a alteração do código ocorra de uma forma mais rápida, visto que a alteração é necessária apenas dentro da função.
- os blocos do programa sejam reduzidos, tornando mais fácil sua compreensão.
- a leitura do código-fonte seja simplificada.
- a lógica do programa seja separada em blocos, que podem ser compreendidos e mantidos de modo isolado;

Sintaxe

```
tipo nome_da_funcao(tipo parametro);
```

- **tipo:** indica o tipo de retorno da função, ou seja, o valor retornado como resultado para sua execução. Exemplo: ***int***, ***float***, ***double***;

- **nome_da_funcao:** indica o nome pelo qual a função deve ser chamada (invocada);
- **tipo parametro:** indica o tipo e nome do parâmetro (valor) que será passado a função;

Utilização

Divide-se em três etapas:

- **Declaração ou protótipo:** efetuada antes da função *main()*, indica que uma nova função está sendo criada, adicionalmente as já disponibilizadas pela linguagem.
- **Chamada ou invocação:** efetuada dentro do fluxo de código com intuito de executar a codificação definida para a função;
- **Implementação:** efetuada após a função *main()*, especifica o código que será executado quando a função for chamada / invocada;

Sintaxe

- Protótipo:

```
tipo nome_da_funcao(parametros);
```

- Chamada ou Invocação:

```
nome_da_funcao(parametros);
```

- Implementação:

```
tipo nome_da_funcao(parametros) {
    // Implementação
}
```

Passagem de Parâmetros por Valor

Quando utilizamos a passagem de parâmetro ***"por valor"***, a variável que é passada a função não tem seu valor modificado quando o código especificado para a mesma é executado.

Passagem de Parâmetros por Referência

A passagem de parâmetro ***"por referência"*** permite que, ao invés do valor, o endereço de memória (referência) de uma variável seja passado como parâmetro a uma função. Para que seja possível então receber esse endereço de memória (referência) o parâmetro da função deve ser um ponteiro. Sendo assim, quando efetuamos a passagem de parâmetro ***"por referência"***, o ponteiro que recebe o endereço de memória da variável passada pode alterar o seu conteúdo, alterando seu valor ao final da execução da função.

Vetores como Parâmetro

Considerando o uso de ponteiros como parâmetro, torna-se possível também passar um conjunto de valores as funções. Esse conjunto de valores podem ser, por exemplo, vetores (ou ponteiros alocados) com valores inteiros ou caracteres. Porém, ao invés de utilizar o operador ***"&"*** como fazemos na passagem ***"por referência"*** de uma variável, agora passamos um vetor (ou ponteiro já alocado) com um tamanho pré-definido;

Codificação – Linguagem de Programação C

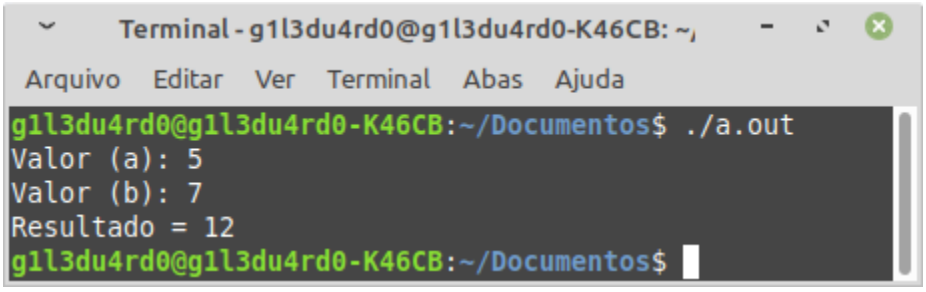
Função - Sem Retorno e Sem Parâmetro

```c
#include <stdio.h>

#include <stdio_ext.h>

#include <stdlib.h>

// variáveis globais

int resultado, val_a, val_b;
```

```c
// protótipo da função (declaração)
void somar();
int main() {
   printf("Valor (a): ");
   scanf("%i", &val_a);
   printf("Valor (b): ");
   scanf("%i", &val_b);
   // chamada da função (invocação)
   somar();
   printf("Resultado = %i", resultado);
   printf("\n");
   return 0;
}
// implementação da função
void somar() {
   resultado = val_a + val_b;
}
```

Função - Sem Parâmetro e Sem Retorno - Com Variáveis Globais

(Arquivo-fonte: 13 - Função/funcao_sem_parametro_sem_retorno.c)



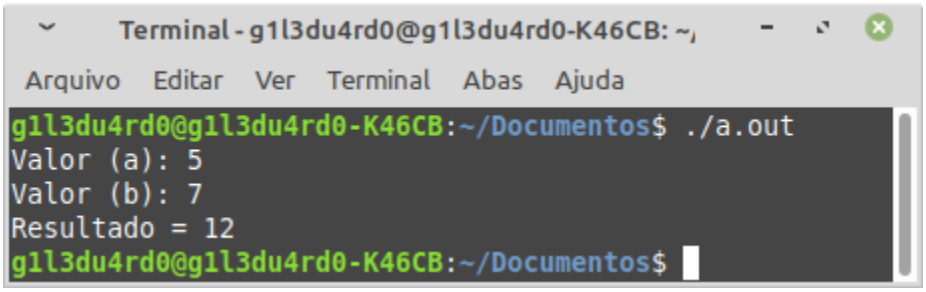
Resultado da execução do código anterior

Função - Sem Retorno e Com Parâmetro

```c
#include <stdio.h>

#include <stdio_ext.h>

#include <stdlib.h>

// variáveis globais

int resultado;

// protótipo da função (declaração)

void somar(int, int);

int main() {

   int val_a, val_b;

   printf("Valor (a): ");

   scanf("%i", &val_a);

   printf("Valor (b): ");

   scanf("%i", &val_b);

   // chamada da função (invocação)

   somar(val_a, val_b);

   printf("Resultado = %i", resultado);

   printf("\n");

   return 0;

}

// implementação da função

void somar(int a, int b) {

   resultado = a + b;

}
```

Função - Com Parâmetro e Sem Retorno - Com Variáveis Globais

(Arquivo-fonte: 13 - Função/funcao_sem_parametro_com_retorno.c)

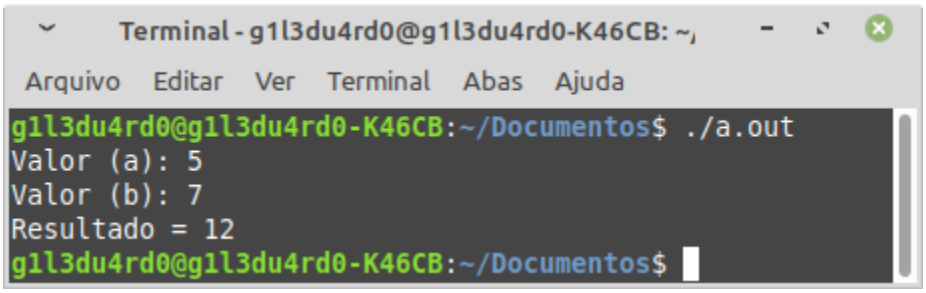


Resultado da execução do código anterior

Função - Com Retorno e Com Parâmetro

```c
#include <stdio.h>

#include <stdio_ext.h>

#include <stdlib.h>

// protótipo da função (declaração)

int somar(int, int);

int main() {

   int resultado, val_a, val_b;

   printf("Valor (a): ");

   scanf("%i", &val_a);

   printf("Valor (b): ");

   scanf("%i", &val_b);

   // chamada da função (invocação)

   resultado = somar(val_a, val_b);

   printf("Resultado = %i", resultado);

   printf("\n");

   return 0;

}

// implementação da função

int somar(int a, int b) {

   return a + b;
```

```
}

            Função - Com Parâmetro e Com Retorno - Sem Variáveis Globais

_____________________________________________________________________

(Arquivo-fonte: 13 - Função/funcao_com_parametro_com_retorno.c)
```



Resultado da execução do código anterior

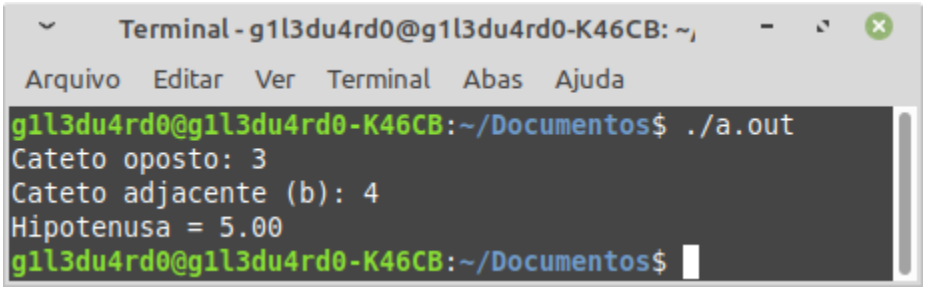
Função - Passagem de Parâmetro por Valor

```c
#include <stdio.h>

#include <stdio_ext.h>

#include <stdlib.h>

#include <math.h>

// protótipo da função (declaração)

double pitagoras(int, int);

int main() {

   int cateto_op, cateto_ad;

   double hipotenusa;

   printf("Cateto oposto: ");

   scanf("%i", &cateto_op);

   printf("Cateto adjacente (b): ");

   scanf("%i", &cateto_ad);

   // chamada da função (invocação)

   hipotenusa = pitagoras(cateto_op, cateto_ad);
```

```c
   printf("Hipotenusa = %.2lf", hipotenusa);
   printf("\n");
   return 0;
}
// implementação da função
double pitagoras(int co, int ca) {
   return sqrt(pow(co, 2) + pow(ca, 2));
}
```

Função - Passagem de Parâmetro por Valor

(Arquivo-fonte: 13 - Função/funcao_passagem_parametro_valor.c)



Resultado da execução do código anterior

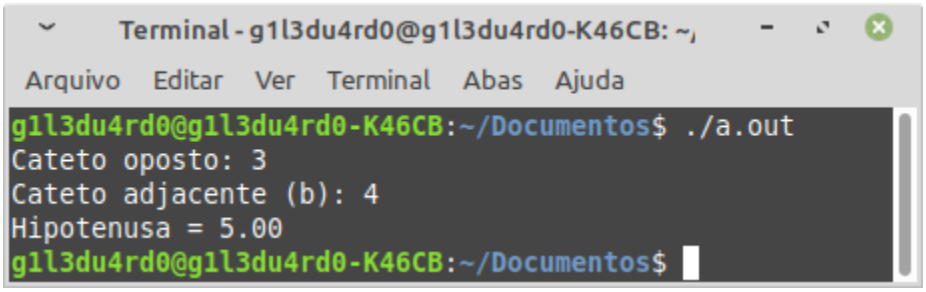
Função - Passagem de Parâmetro por Referência

```c
#include <stdio.h>
#include <stdio_ext.h>
#include <stdlib.h>
#include <math.h>
// protótipo da função (declaração)
void pitagoras(int, int, double*);
int main() {
   int cateto_op, cateto_ad;
```

```c
    double hipotenusa;

    printf("Cateto oposto: ");

    scanf("%i", &cateto_op);

    printf("Cateto adjacente (b): ");

    scanf("%i", &cateto_ad);

    // chamada da função (invocação)

    pitagoras(cateto_op, cateto_ad, &hipotenusa);

    printf("Hipotenusa = %.2lf", hipotenusa);

    printf("\n");

    return 0;

}

// implementação da função

void pitagoras(int co, int ca, double *hipo) {

    *hipo = sqrt(pow(co, 2) + pow(ca, 2));

}
```

Função - Passagem de Parâmetro por Referência

(Arquivo-fonte: 13 - Função/funcao_passagem_parametro_referencia.c)



Resultado da execução do código anterior

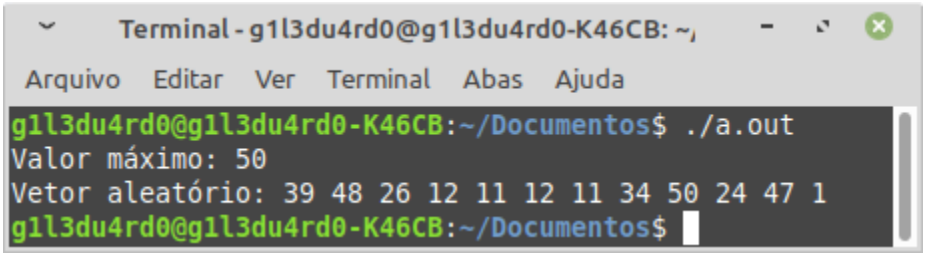
Função - Vetor por Parâmetro (int)

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

```c
#include <stdio_ext.h>
#include <stdlib.h>
#include <time.h>
#define TAM 12
// protótipo da função (declaração)
void aleatorio(int, int*);
int main() {
   int a, maximo, numeros[TAM];
   srand(time(NULL));
   printf("Valor máximo: ");
   scanf("%i", &maximo);
   aleatorio(maximo, numeros);
   printf("Vetor aleatório: ");
   for(a=0; a<TAM; a++) {
       printf("%i ", numeros[a]);
   }
   printf("\n");
   return 0;
}
// implementação da função
void aleatorio(int max, int *vet) {
   int a;
   for(a=0; a<TAM; a++) {
       vet[a] = rand()%(max+1);
   }
}
```

Função - Vetor como Parâmetro - Inteiro

(Arquivo-fonte: 13 - Função/funcao_passagem_vetor_inteiro.c)



Resultado da execução do código anterior

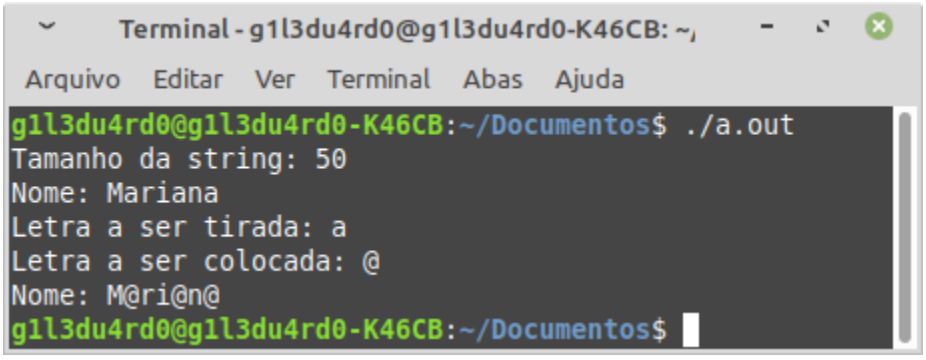
Função - Vetor por Parâmetro (char)

```c
#include <stdio.h>

#include <stdio_ext.h>

#include <stdlib.h>

#include <time.h>

// protótipo da função (declaração)

void substituir(char, char, char*);

int main() {

   int a, tam;

   char tirar, colocar, *nome;

   printf("Tamanho da string: ");

   scanf("%i", &tam);

   nome = malloc(tam * sizeof(char));

   printf("Nome: ");

   __fpurge(stdin);

   gets(nome);

   printf("Letra a ser tirada: ");

   __fpurge(stdin);

   scanf("%c", &tirar);
```

```c
    printf("Letra a ser colocada: ");
    __fpurge(stdin);
    scanf("%c", &colocar);
    substituir(tirar, colocar, nome);
    printf("Nome: %s", nome);
    printf("\n");
    return 0;
}
// implementação da função
void substituir(char b, char c, char *vet) {
    int a;
    for(a=0; vet[a] != '\0'; a++) {
        if(vet[a] == b) {
            vet[a] = c;
        }
    }
}
```

Função - Vetor como Parâmetro - String

(Arquivo-fonte: 13 - Função/funcao_passagem_vetor_string.c)



Resultado da execução do código anterior