Funções e Procedimentos

Programação e Desenvolvimento de Software - I
Heitor Ramos

- Separação lógica do programa
- Blocos com propósito claro
- Mapeiam valores de entrada para uma saída **parâmetros → retorno**

```c
// main não recebe nada a retorna int
int main(void);

// sqrt computa a raiz de double
double sqrt(double x);

// define se um número é primo
int is_prime(int x);

// soma dois números
int soma_dois(int x, int y);

// troca o valor do endereço de 2 ponteiros
void troca_valores(int *x, int *y);
```

Detalhes

- Toda função deve ter um tipo. Esse tipo determina qual será o tipo do seu valor de retorno

- Os parâmetros de uma função determinam qual será o seu comportamento. Parâmetros são variáveis que são iniciadas quando a função é chamada

- Procedimentos retornam void (não retornam nenhum valor)

- Funções retornam algo diferente de void
  - Mapeiam uma entrada para um retorno

- Para facilitar, vamos chamar tudo de funções (é como a linguagem trata)

Detalhes

- Existem funções sem parâmetros

```
double eulers_number();
```

- Funções só podem ser declaradas fora de outras funções

- A palavra chave `return` indica o resultado da função
  - Não é necessária em funções do tipo void
  - Encerra a execução da função

```
int soma_tres(int x) {
    return x + 3;
}
```

Problema 1

Vamos criar uma função que soma dois números

Soma dois números

```c
#include <stdio.h>

int soma_xy(int x, int y) {
  return x + y;
}

int main(void) {
  printf("1: - %d\n", soma_xy(10, 2));
  int x = 9;
  int y = 1;
  printf("2: - %d\n", soma_xy(x, y));
}
```

Olhando para a memória

```c
#include <stdio.h>
int soma_xy(int x, int y) {
  return x + y;
}
int main(void) {
  printf("1: - %d\n",
         soma_xy(10, 2));
  int x = 9;
  int y = 1;
  printf("2: - %d\n",
         soma_xy(x, y));
}
```

Função `int main(void)`	Nome	Endereço	Valor
	x	0x7fff7a481e80	9
	y	0x7fff7a481e84	1

Variáveis com nomes iguais mas end. diferentes

```c
#include <stdio.h>
int soma_xy(int x, int y) {
  return x + y;
}
int main(void) {
  printf("1: - %d\n",
         soma_xy(10, 2));
  int x = 9;
  int y = 1;
  printf("2: - %d\n",
         soma_xy(x, y));
}
```

Função `int main(void)`	Nome	Endereço	Valor
	x	0x7fff7a481e80	9
	y	0x7fff7a481e84	1

Função `int soma_xy (int, int)`	Nome	Endereço	Valor
	x	0x7fff6b481f90	9
	y	0x7fff6b481f94	1

Passagem por Valor

- Quando chamamos uma função, copiamos os valores passados para a mesma
- Por isso chamamos de passagem por valor
- Mesmo se 2 variáveis tiverem o mesmo nome, elas existem em locais diferentes da memória do computador
- Uma mudança em x dentro da função não muda x do main

Qual a saída?

```c
#include <stdio.h>

int soma_xy(int x, int y) {
  x += y;
  return x;
}

int main(void) {
  int x = 9;
  int y = 1;
  printf("1: - %d\n", soma_xy(x, y));
  printf("2: - %d\n", x);
}
```

Escopo Local

- Sabendo que o endereço é diferente para as variáveis dentro de cada função, dizemos que o escopo das mesmas é local

- Nos exemplos anteriores, x e y tem 2 versões:
  - Uma dentro do escopo da função main
  - Outra dentro do escopo da função soma_xy

Escopo Global

- De qualquer forma, nada impede um programador de declarar variáveis com um escopo global
- “Fora das funções”
- Embora ensinamos tal prática, não recomendamos a mesma! Organize seu código para usar apenas escopo local ou passagem por referência
- Use o escopo global apenas com constantes (**const**)

Qual a saída?

```c
#include <stdio.h>
int x = 9;
int y = 1;
int soma_xy(int x, int y) {
  x += y;
  return x;
}
int soma_xy2(int y) {
  x += y;
  return x;
}
int soma_xy3() {
  return x+y;
}
int main(void) {
  printf("1: - %d\n", soma_xy(10, 2));
  printf("2: - %d\n", soma_xy(x, y));
  printf("3: - %d\n", soma_xy2(20));
  printf("4: - %d\n", soma_xy3());
}
```

Qual a saída?

```c
#include <stdio.h>
int x = 9;
int y = 1;
int soma_xy(int x, int y) {
  x += y;
  return x;
}
int soma_xy2(int y) {
  x += y;
  return x;
}
int soma_xy3() {
  return x+y;
}
int main(void) {
  printf("1: - %d\n", soma_xy(10, 2));
  printf("2: - %d\n", soma_xy(x, y));
  printf("3: - %d\n", soma_xy2(20));
  printf("4: - %d\n", soma_xy3());
}
```

Sempre usa x e y local. Não afeta o global

Muda o valor global de x

Apenas lê os globais

Chamando funções

- Funções podem chamar outras funções
- Afinal, o main faz isso com frequência
- Chamamos funções do
  stdio
  math
  stdlib
  ...

Qual a saída?

```c
#include <stdio.h>
int x = 9;
int y = 1;
int soma_xy(int x, int y) {
  return x+y;
}
int soma_xy2(int y) {
  x = soma_xy(10, 2);
  x += y;
  return x;
}
int soma_xy3() {
  y = soma_xy2(20);
  return x+y;
}
int main(void) {
  printf("A saida foi: - %d\n",
soma_xy3());
}
```

Passagem por Referência

- Como fazer com que 2 funções usem dados armazenados no mesmo espaço de memória?

- Problema: Escreva uma função que troca 2 números

Qual o problema?

```c
#include <stdio.h>

void troca_valores(int x, int y) {
  printf("endereço de x, y troca: %p %p\n",
         &x, &y);
  int aux = 0;
  aux = x;
  x = y;
  y = aux;
}

int main(void) {
  int x = 7;
  int y = 10;
  printf("endereço de x, y main: %p %p\n",
          &x, &y);
  printf("%d %d\n", x, y);
  troca_valores(x, y);
  printf("%d %d\n", x, y);
  return 0;
}
```

Cada função usa endereços distintos!

```c
#include <stdio.h>

void troca_valores(int x, int y) {
  printf("%p %p\n", &x, &y);
  int aux = 0;
  aux = x;
  x = y;
  y = aux;
}

int main(void) {
  int x = 7;
  int y = 10;
  printf("%p %p\n",&x, &y);
  printf("%d %d\n", x, y);
  troca_valores(x, y);
  printf("%d %d\n", x, y);
  return 0;
}
```

Função	Nome	Endereço	Valor
`int main(void)`	x	0x7fff7a481e80	7
	y	0x7fff7a481e84	10

Função	Nome	Endereço	Valor
`int troca_valores (int, int)`	x	0x7fff6b481f90	10
	y	0x7fff6b481f94	7

Operadores & e *

& ➜ retorna o endereço de memória de uma variável. Em outras palavras, um ponteiro

* ➜ utilizado para indicar ponteiros

```
// y é uma variável com valor 10
// y = 10; y++ = 11;
int  y = 10;

// y_ptr é uma variável com o valor do endereço de y
// y_ptr = 0x7fff6b481f90; y_ptr++ = 0x7fff6b481f94;
int *y_ptr = &y;
```

Operadores & e *

Se já tivermos um ponteiro, o operador * retorna o valor armazenado no endereço do mesmo

```
// y é uma variável com valor 10
// y = 10; y++ = 11;
int  y = 10;

// y_ptr uma variável com o valor do endereço de y
// y_ptr = 0x7fff6b481f90; y_ptr++ = 0x7fff6b481f94;
int *y_ptr = &y;
// muda o valor de y para 11
(*y_ptr)++;
```

Função Correta

```c
#include <stdio.h>

void troca_valores(int* x_ptr, int* y_ptr) {
  // note que não usa & no print.
  // já tenho 2 ponteiros
  printf("troca: %p %p\n",
         x_ptr, y_ptr);
  int aux = 0;
  aux = *x_ptr;
  *x_ptr = *y_ptr;
  *y_ptr = aux;
}

int main(void) {
  int x = 7;
  int y = 10;
  printf("main: %p %p\n",
         &x, &y);
  printf("%d %d\n", x, y);
  troca_valores(&x, &y);
  printf("%d %d\n", x, y);
  return 0;
}
```

Função Correta

```c
#include <stdio.h>

void troca_valores(int* x_ptr, int* y_ptr) {
  // note que não usa & no print.
  // já tenho 2 ponteiros
  printf("troca: %p %p\n",
         x_ptr, y_ptr);
  int aux = 0;
  aux = *x_ptr;
  *x_ptr = *y_ptr;
  *y_ptr = aux;
}

int main(void) {
  int x = 7;
  int y = 10;
  printf("main: %p %p\n",
         &x, &y);
  printf("%d %d\n", x, y);
  troca_valores(&x, &y);
  printf("%d %d\n", x, y);
  return 0;
}
```

Função `int main(void)`

Nome	Endereço	Valor
x	0x7f...80	7
y	0x7f...84	10

Função `int troca_valores (int*, int*)`

Nome	Endereço	Valor
x_ptr	0x6d...70	0x7f...80
y_ptr	0x6d...74	0x7f...84
*x_ptr	0x7f...80	7
*y_ptr	0x7f...84	10

Vetores e Matrizes em Funções

```c
int soma_todos_vetor(int n, int vetor[n]) {
  int soma = 0;
  for (int i = 0; i < n; i++)
    soma += vetor[i];
  return soma;
}

int soma_todos_vetor_ptr(int n, int* vetor) {
  int soma = 0;
  for (int i = 0; i < n; i++)
    soma += vetor[i];
  return soma;
}
```

Vetores e Matrizes em Funções

```c
// C99 e C11. Em C89 não funciona
int soma_todos_matriz(int n, int m, int mat[n][m]) {
  int soma = 0;
  for (int i = 0; i < n; i++) {
   for (int j = 0; j < m; j++) {
       soma += mat[i][j];
    }
  }
  return soma;
}
```

Vetores e Matrizes em Funções

```
int soma_todos_matriz_ptr(int n, int m, int** mat) {
  int soma = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        soma += mat[i][j];
    }
  }
  return soma;
}
```

Struct como parâmetro

- Podemos passar uma struct por valor ou por referência
- Temos duas possibilidades
  - Passar por parâmetro toda a struct
  - Passar por parâmetro apenas um campo específico da struct

Struct como parâmetro

- Passar por parâmetro apenas um campo específico da struct
  - Valem as mesmas regras vistas até o momento
  - Cada campo da struct é como uma variável independente. Ela pode, portanto, ser passada individualmente por ***valor*** ou por ***referência***

Struct como parâmetro

- Passar por parâmetro toda a struct
- Passagem por valor
  - Valem as mesmas regras vistas até o momento
  - A struct é tratada com uma variável qualquer e seu valor é copiado para dentro da função
- Passagem por referência
  - Valem as regras de uso do asterisco “*” e operador de endereço “&”
  - Devemos acessar o conteúdo da struct para somente depois acessar os seus campos e modificá-los.
  - Uma alternativa é usar o ***operador seta “->”***

Usando “*”

```c
struct ponto {
    int x, y;
};

void atribui(struct ponto *p){
    (*p).x = 10;
    (*p).y = 20;
}

struct ponto p1;

atribui(&p1);
```

```c
struct ponto {
    int x, y;
};

void atribui(struct ponto *p){
    p->x = 10;
    p->y = 20;
}

struct ponto p1;

atribui(&p1);
```

Exercícios

Escreva uma função chamada my_pow, que receba dois inteiros (x e y) como parâmetros e retorna o valor de x^y.

Exercícios

Utilizando a função criada no exercício anterior, crie uma função chamada soma_quadrados que retorna a soma dos quadrados dos *n* primeiros números naturais, onde *n* é um parâmetro fornecido para a função. Exemplo para *n* = 4: $1^2 + 2^2 + 3^2 + 4^2$

Exercícios

Escreva uma função chamada palíndromo, que receba uma string e retorna 1 se a string for um palíndromo, ou 0 caso contrário.