Funções e Procedimentos - Parte 1

Prof. Tiago A. Almeida
talmeida@ufscar.br

Funções e Procedimentos

✓ Funções

- São procedimentos que retornam um único valor ao final de sua execução
  - ★ `x = sqrt(4);`

✓ Procedimentos

- São estruturas que agrupam um conjunto de comandos, que são executados quando o procedimento é chamado
  - ★ `scanf("%d", &x);`

Porque usar funções?

- ✓ Evitar que os blocos do programa fiquem grandes demais e, por consequência, mais difíceis de ler e entender.
- ✓ Separar o programa em partes que possam ser logicamente compreendidos de forma isolada.
- ✓ Permitir o reaproveitamento de código já construído (por você ou por outros programadores).
- ✓ Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa, minimizando erros e facilitando alterações.

Declarando uma função

- Uma função é declarada da seguinte forma:

```
tipo nome (tipo parâmetro1, tipo parâmetro2, ...,
tipo parâmetroN) {

    comandos;

    return valor de retorno;
}
```

- Toda função deve ter um **tipo**. Esse tipo determina qual será o tipo de seu **valor de retorno**.
- Os parâmetros de uma função determinam qual será o seu comportamento, se comportando como variáveis que são iniciadas quando a função é chamada.

Declarando uma função

- ✓ Uma função pode não ter parâmetros, basta não informá-los.
- ✓ A expressão contida dentro do comando `return` é chamada de valor de retorno, e corresponde a resposta de uma determinada função. Esse comando é sempre o último a ser executado por uma função, e nada após ele será executado.
- ✓ As funções só podem ser declaradas fora de outras funções. Lembre-se que o corpo do programa principal (`int main()`) é uma função.

Exemplo de função

- A função abaixo soma dois valores, passados como parâmetros:

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

Invocando uma função

- Uma forma clássica de realizarmos a invocação (ou chamada) de uma função é atribuindo o seu valor a uma variável:

```
x = soma(4, 2);
```

- Como o resultado da chamada de uma função é uma expressão, ela pode ser usada em qualquer lugar que aceite uma expressão:

```
printf("Soma de a e b: %d\n", soma(a, b));
```

- Veja um exemplo em `soma.c`.

Invocando uma função

- Para cada um dos parâmetros da função, devemos fornecer uma expressão de mesmo tipo, chamada de parâmetro real. O valor destas expressões são copiados para os parâmetros da função.
- Ao usar variáveis como parâmetros reais, estamos usando apenas os seus valores para avaliar a expressão.
- Se forem variáveis, os parâmetros reais passados pela função não necessariamente possuem os mesmos nomes que os parâmetros que a função espera.
- O valor das expressões que fornecem os parâmetros reais não é afetado por alterações nos parâmetros dentro da função.
- Veja um exemplo em `parametros.c`.

O tipo `void`

- O tipo `void` é um tipo especial, utilizado principalmente em funções.
- Ele é um tipo que representa o "**nada**", ou seja, uma variável desse tipo armazena conteúdo indeterminado, e uma função desse tipo retorna um conteúdo indeterminado.
- Este tipo é utilizado para indicar que uma função não retorna nenhum valor.

Procedimentos em C

- Procedimentos em linguagem C nada mais são que funções do tipo `void`. Por exemplo, o procedimento abaixo imprime o número que for passado para ele como parâmetro:

```
void imprime (int numero) {
    printf ("Número %d\n", numero);
}
```

- Podemos ignorar o valor de retorno de uma função e, para esta chamada, ela será equivalente a um procedimento.

Invocando um procedimento

- Para invocarmos um procedimento, devemos utilizá-lo como utilizaríamos qualquer outro comando, ou seja:

```
procedimento (parametros);
```

- Esta é a forma como chamamos usualmente as funções `printf` e `scanf`.

- Veja um exemplo em `imprime.c`.

A função `main`

- O programa principal é uma função especial, que possui um tipo fixo (`int`) e é invocada automaticamente pelo sistema operacional quando este inicia a execução do programa.
- Quando utilizado, o comando `return` informa ao sistema operacional se o programa funcionou corretamente ou não. O padrão é que um programa retorne zero caso tenha funcionado corretamente ou qualquer outro valor caso contrário.

```
int main() {
        printf("Hello, World!\n");
        return 0;
}
```

Declarando funções após o `main`

- Até o momento, aprendemos que devemos declarar as funções antes do programa principal, mas o que ocorreria se declarássemos depois?

- Veja os exemplos em `depois.c` e `depois2.c`. Para deixar mais aparente os problemas, compile com a opção `-Wall`.

Declarando uma função sem defini-la

- Para organizar melhor um programa ou para escrever um programa em vários arquivos podemos declarar uma função sem implementá-la ou defini-la.
- Para declarar uma função sem a sua implementação. Substituímos as chaves e seu conteúdo por ponto-e-virgula.

```
tipo nome (tipo parâmetro1, tipo parâmetro2, ...,
tipo parâmetroN);
```

- A declaração de uma função deve vir sempre antes do seu uso. A sua definição pode aparecer em qualquer lugar do programa.

Passagem de parâmetros por valor

- Quando passamos argumentos a uma função, os valores fornecidos são copiados para os parâmetros formais da função. Este processo é idêntico a uma atribuição.
- Desta forma, alterações nos parâmetros dentro da função não alteram os valores que foram passados:

```
void nao_troca(int x, int y) {
    int aux;
    aux = x;
    x = y;
    y = aux;
}
```

- Veja o exemplo em `nao_troca.c`.

Variáveis locais e variáveis globais

- Uma variável é chamada local se ela foi declarada dentro de uma função. Nesse caso, ela existe somente dentro daquela função e após o término da execução da mesma, a variável deixa de existir.
- Uma variável é chamada global se ela for declarada fora de qualquer função (ou seja, no mesmo lugar onde registros, tipos enumerados e funções são declarados). Essa variável é visível em todas as funções, qualquer função pode alterá-la e ele existe durante toda a execução do programa.

Variáveis globais

```
#include <stdio.h>

int variavel_global;

int main () {
  variavel_global = 0;
  printf ("%d", variavel_global);
}
```

- Veja outro exemplo em `global.c`

Escopo de variáveis

- O escopo de uma variável determina de que partes do código ela pode ser acessada.
- A regra de escopo em C é bem simples:
  - As variáveis globais são visíveis por todas as funções.
  - As variáveis locais são visíveis apenas na função onde foram declaradas.

Escopo de variáveis

```
int global;

void a() {
   int local_a;
   /* Neste ponto são visíveis global e local_a */
}

int main() {
   int local_main;
   a();
   /* Neste ponto são visíveis global e local_main */
}
```

Escopo de variáveis

- É possível declarar variáveis locais com o mesmo nome de variáveis globais.
- Nesta situação, a variável local “esconde” a variável global.

```
int nota;

void a() {
   int nota;
   /* Neste ponto nota é a variável local. */
}
```

- Veja mais detalhes em `escopo.c`

Exercício

1. Escreva um programa que receba dois naturais n_1 e n_2, calcule e imprima na tela (use funções):
   - $n_1 + n_2$
   - $n_1 - n_2$
   - $n_1 * n_2$
   - n_1 / n_2
   - $(n_1)^{n2}$
   - imprimir todos os pares de 1 a n_1
   - imprimir todos os ímpares de 1 a n_2

Funções e Procedimentos - Parte 2

Prof. Tiago A. Almeida
talmeida@ufscar.br

Registros em funções

- Registros podem ser passados como parâmetros de uma função, assim como qualquer outro tipo.
- O registro deve ser declarado antes da função.
- O parâmetro formal recebe uma cópia do registro, da mesma forma que em uma atribuição envolvendo registros.
- Uma função pode retornar um registro, que é novamente copiado como resultado da expressão

- Veja o exemplo em `registro.c`.

Vetores em funções

- Vetores também podem ser passados como parâmetros de uma função, **porém não podem ser usados diretamente como variáveis de retorno!**
  - Normalmente, o tamanho do vetor também é passado como parâmetro da função juntamente com o vetor

```
int maiorElemento(int vet[], int tamanho){
  int i, maior = vet[0];

  for (i = 0; i < tamanho; i++) {
    if (maior < vet[i]) {
      maior = vet[i];
    }
  }
  return(maior);
}
```

- Veja `vetor.c`

Vetores em funções

- Ao contrário dos outros tipos e registros, vetores têm um comportamento diferente quando usados como parâmetros ou valores de retorno de funções.
- Por padrão, ao se indicar o tipo de um vetor, este sempre é interpretado pelo compilador como o **endereço** do primeiro elemento do vetor.
- Desta forma, os vetores são sempre passados por referência, ou seja, não são criadas cópias do vetor dentro das funções e o próprio vetor é alterado!

- Veja os exemplos em `vetor_parametro.c`, `vetor_vs_variavel.c`, `vetor_erro.c` e `vetor_global.c`

Matrizes em funções

- Quando o arranjo é multi-dimensional a possibilidade de não informar o tamanho na declaração se restringe a primeira dimensão apenas

```
void imprimirMatriz(int mat[][10], int n_linhas)
{
   int i, j;

   for (i = 0; i < n_linhas; i++)
   {
      for (j = 0; j < 10; j++)
         printf("%2d ", mat[i][j]);
      printf("\n");
   }
}
```

- Veja `matriz.c`

Exercício

1. Escreva um programa que receba dois vetores v_1 e v_2 com t números naturais (máximo 100). Calcule e imprima na tela (use funções):
   - vetor resultante de $v_1 + v_2$
   - vetor resultante de $v_1 - v_2$
   - a soma dos elementos de v_1
   - a soma dos elementos de v_2
   - o maior elemento de v_1
   - o menor elemento de v_2
   - imprimir todos os pares de v_1
   - imprimir todos os ímpares de v_2

Exercícios

Prof. Tiago A. Almeida
talmeida@ufscar.br