MC-102 — Aula 12
Funções e Procedimentos I

Instituto de Computação – Unicamp

Segundo Semestre de 2007

Roteiro

Procedimentos e funções

Procedimentos

São estruturas que agrupam um conjunto de comandos, que são executados quando o procedimento é chamado.

```
scanf("%d", &x);
```

Funções

São procedimentos que retornam um único valor ao final de sua execução.

```
x = sqrt(4);
```


Porque utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais e, por conseqüência, 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 chamado 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 (`main()`) é uma função.

Exemplo de função

A função abaixo soma dois valores, passados como parâmetro:

```
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);
```

Na verdade, o resultado da chamada de uma função é uma expressão e pode ser usada em qualquer lugar que aceite uma expressão:

Exemplo

```
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.

Exemplo

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


Declarando funções depois do main

Até o momento, aprendemos que devemos declarar as funções antes do programa principal, mas o que ocorreria se declarassemos 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.

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 */
}
```

Veja outro exemplo em `parametros.c`.

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

Reestruture o programa `verificar.c` em termos de funções e inclua a opção "todas as verificações". Este programa testa um número de acordo com as opções abaixo:

```
1 - Número primo
2 - Número par
3 - Quadrado perfeito
4 - Cubo perfeito
5 - Todas as verificações
6 - Sair
```