Programação Estruturada

Funções

Professores Emílio Francesquini e Carla Negri Lintzmayer

2018.Q3

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

Funções

Funções

- Um ponto chave na resolução de um problema complexo é conseguir “quebrá-lo” em subproblemas menores.
- Ao criarmos um programa para resolver um problema, é crítico quebrar um código grande em partes menores, fáceis de serem entendidas e administradas.
- Isto é conhecido como modularizacão, e é empregado em qualquer projeto de engenharia envolvendo a construção de um sistema complexo.

Funções

Funções

São estruturas que agrupam um conjunto de comandos, que são executados quando a função é chamada/invocada.

- Vocês já usaram algumas funções como **scanf** e **printf**.
- Algumas funções podem devolver algum valor ao final de sua execução:

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

- Vamos aprender como criar e usar funções em C.

Por que utilizar 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 compreendidas 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.

Definindo uma função

Uma função é definida da seguinte forma:

```
1 tipo_retorno nome(tipo parâmetro1,..., tipo
  ↪  parâmetroN) {
2     comandos;
3     return valor_de_retorno;
4 }
```

- Toda função deve ter um tipo (**int, char, float, void, etc**). Esse tipo determina qual será o tipo de seu valor de retorno.
- Os **parâmetros** são variáveis, que são inicializadas com valores indicados durante a invocação da função.
- O comando **return** devolve para o invocador da função o resultado da execução desta.

Definindo uma função: exemplo

A função abaixo recebe como parâmetro dois valores inteiros. Ela faz a soma destes valores e devolve o resultado.

```
1  int soma(int a, int b) {
2      int c;
3      c = a + b;
4      return c;
5  }
```

- Note que o valor de retorno (variável **c**) é do mesmo tipo da função.
- Quando o comando **return** é executado, a função para de executar e retorna o valor indicado para quem fez a invocação (ou chamada) da função.

Definindo uma função: exemplo

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

Qualquer função pode invocar esta função **soma**, passando como parâmetro dois valores inteiros, que serão atribuídos para as variáveis **a** e **b**, respectivamente.

```c
int main() {
    int r;
    r = soma(12, 90);
    printf("r = %d\n", r);
    r = soma (-9, 45);
    printf("r = %d\n", r);
    return 0;
}
```

Definindo uma função: exemplo

```c
#include <stdio.h>

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

int main() {
    int res, x1 = 4, x2 = -10;
    res = soma(5, 6);
    printf("Primeira soma: %d\n", res);
    res = soma(x1, x2);
    printf("Segunda soma: %d\n",res);
    return 0;
}
```

Execução de um programa com funções

- Qualquer programa começa executando os comandos da função **main**.
- Quando se encontra a chamada para uma função, o fluxo de execução passa para ela e são executados os comandos até que um **return** seja encontrado ou o fim da função seja alcançado.
- Depois disso, o fluxo de execução volta para o ponto onde a chamada da função ocorreu.

Definindo uma função: exemplo

A lista de parâmetros de uma função pode ser vazia.

```c
int leInteiro() {
    int c;
    printf("Digite um número: ");
    scanf("%d", &c);
    return c;
}
```

O retorno será usado pelo invocador da função:

```c
int main() {
    int r;
    r = leInteiro();
    printf("Numero digitado: %d\n", r);
    return 0;
}
```

Definindo uma função: exemplo

```c
#include <stdio.h>

int leInteiro() {
    int c;
    printf("Digite um numero: ");
    scanf("%d", &c);
    return c;
}

int main() {
    int r;
    r = leInteiro();
    printf("Numero digitado: %d\n", r);
    return 0;
}
```

Exemplo de função

A expressão contida dentro do comando **return** é chamada de *valor de retorno* (é a resposta da função). Nada após esse comando será executado.

```c
#include <stdio.h>

int leInteiro() {
    int c;
    printf("Digite um numero: ");
    scanf("%d", &c);
    return c;
    printf("Bla bla bla!\n");
}

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

int main() {
    int x1, x2, res;
    x1 = leInteiro();
    x2 = leInteiro();
    res = soma(x1, x2);
    printf("Soma eh: %d\n", res);
    return 0;
}
```

Invocando uma função

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

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

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

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

Invocando uma função

Na chamada da função, para cada um dos parâmetros desta devemos fornecer um valor de mesmo tipo e na mesma ordem da declaração.

```c
#include <stdio.h>

int somaComMensagem(int a, int b, char texto[100])
 ↪  {
    int c = a + b;
    printf("%s %d\n", texto, c);
    return c;
}

int main() {
    somaComMensagem(4, 5, "Resultado da soma:");
    return 0;
}
```

Invocando uma função

A saída do programa anterior será:

```
Resultado da soma: 9
```

Já a chamada abaixo gerará um erro de compilação:

```
int main() {
    somaComMensagem(4, "Resultado da soma:",  5);
    return 0;
}
```

Invocando uma função

- Ao chamar uma função passando variáveis **simples** como parâmetros, estamos usando apenas os seus valores, que serão copiados para as variáveis parâmetros da função.
- Os valores das variáveis na chamada da função não são afetados por alterações dentro da função.

```c
#include <stdio.h>

int incr(int x) {
    x = x + 1;
    return x;
}

int main() {
    int a = 2, b;
    b = incr(a);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

- O que será impresso? O valor de **a** é alterado pela função **incr**?

O tipo void

O tipo void

- O “tipo” **void** é um tipo especial.
- Ele representa “nada”, ou seja, uma variável desse tipo armazena conteúdo indeterminado, e uma função desse tipo retorna um conteúdo indeterminado.
- Em geral, este tipo é utilizado para indicar que uma função não retorna nenhum valor.

O tipo void

- Por exemplo, a função abaixo imprime o número que for passado para ela como parâmetro e não devolve nada.
- Neste caso não utilizamos o comando **return**.

```
1  void imprimeInteiro(int numero) {
2      printf("Número %d\n", numero);
3  }
```

O tipo void

```c
#include <stdio.h>

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

int main() {
    imprimeInteiro(10);
    imprimeInteiro(20);
    return 0;
}
```

A função main

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.

```c
#include <stdio.h>

int main() {
    printf("Ola turma!\n");
    return 0;
}
```

Protótipo de funções

Protótipo de funções: definindo funções depois da main

Até o momento, aprendemos que devemos definir as funções antes do programa principal. O que ocorreria se declarássemos depois?

```c
#include <stdio.h>

int main() {
    float a = 0, b = 5;
    printf("%f\n", soma(a, b));
    return 0;
}

float soma(float op1, float op2) {
    return op1 + op2;
}
```

Protótipo de funções: declarando uma função sem defini-la

- Para organizar melhor um programa e podermos implementar funções em partes distintas do arquivo fonte, utilizamos **protótipos de funções**.
- Protótipos de funções correspondem a primeira linha da definição de uma função, contendo: tipo de retorno, nome da função, parâmetros e por fim **um ponto e vírgula**.

```
1  tipo_retorno nome(tipo parâmetro1, ..., tipo
   ↪  parâmetroN);
```

- O protótipo de uma função deve aparecer antes do seu uso.
- Em geral coloca-se os protótipos de funções no início do seu arquivo do programa.

Protótipo de funções: declarando uma função sem defini-la

Em geral o programa é organizado da seguinte forma:

```
1  #include <stdio.h>
2  #include <outras bibliotecas>
3
4  /* Protótipos de funções */
5  int fun1(lista de parâmetros);
6
7  int main() {
8      comandos;
9  }
10
11 /* Declarações de funções */
12 int fun1(lista de parâmetros) {
13     comandos;
14 }
```

Protótipo de Funções: Exemplo 1

```c
#include <stdio.h>

float soma(float op1, float op2);
float subtrai(float op1, float op2);

int main() {
    float a = 0, b = 5;
    printf("soma = %f\nsubtracao = %f\n", soma (a, b),
    ↪  subtrai(a, b));
    return 0;
}

float soma(float op1, float op2) {
    return op1 + op2;
}

float subtrai(float op1, float op2) {
    return op1 - op2;
}
```

Funções Podem Invocar Funções

Funções Podem Invocar Funções

- Nos exemplos anteriores apenas a função **main** invocava funções por nós definidas.
- Isto não é uma regra. Qualquer função pode invocar outra função (exceto a **main**, que é invocada apenas pelo sistema operacional).
- Veja o exemplo no próximo slide.

Funções Podem Invocar Funções

O que será impresso?

```c
#include <stdio.h>

int fun1(int a);
int fun2(int b);

int main() {
    int c = 5;
    c = fun1(c);
    printf("c = %d\n", c);
    return 0;
}

int fun1(int a) {
    a = a + 1;
    a = fun2(a);
    return a;
}

int fun2(int b) {
    b = 2 * b;
    return b;
}
```

Escopo de Variáveis: variáveis locais e globais

Variáveis locais e variáveis globais

- Uma variável é chamada local se ela foi declarada dentro de uma função. Nesse caso ela existe somente dentro da função e, após o término da execução desta, a variável deixa de existir. **Variáveis parâmetros também são variáveis locais.**
- Uma variável é chamada global se ela for declarada fora de qualquer função. Essa variável é visível em todas as funções. Qualquer função pode alterá-la e ela existe durante toda a execução do programa.

Organização de um programa

Em geral um programa é organizado da seguinte forma:

```
1  #include <stdio.h>
2  #include <outras bibliotecas>
3
4  protótipos de funções
5
6  declaração de Variáveis Globais
7
8  int main() {
9      declaração de variáveis locais à main
10     comandos;
11 }
12
13 int fun1(parâmetros) { /* parâmetros também são variáveis locais
   ↪  */
14     declaração de variáveis locais à fun1
15     comandos;
16 }
17
18 ...
```

Escopo de variáveis

- O escopo de uma variável determina de quais partes do código ela pode ser acessada, ou seja, de quais partes do código a variável é visível.
- 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

```c
#include <stdio.h>

void fun1();
int fun2(int local_b);

int global;

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

void fun1() {
    int local_a;
    /* Neste ponto são visíveis global e local_a */
}

int fun2(int local_b) {
    int local_c;
    /* Neste ponto são visíveis global, local_b e local_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.

```c
#include <stdio.h>

void fun();

int nota = 10;

int main() {
    nota = 20;
    fun();
    printf("%d\n", nota);
    return 0;
}

void fun() {
    int nota;
    nota = 5;
    /* Neste ponto nota é a variável local de fun. */
}
```

Exemplo 1

```c
#include <stdio.h>

void fun1();
void fun2();

int x;

int main() {
    x = 1;
    fun1();
    fun2();
    printf("main: %d\n", x);
    return 0;
}

void fun1() {
    x = x + 1;
    printf("fun1: %d\n", x);
}

void fun2() {
    int x = 3;
    printf("fun2: %d\n", x);
}
```

O que será impresso?

Exemplo 2

```c
#include <stdio.h>

void fun1();
void fun2();

int x = 1;

int main() {
    int x = 1;
    fun1();
    fun2();
    printf("main: %d\n", x);
    return 0;
}

void fun1() {
    x = x + 1;
    printf("fun1: %d\n", x);
}

void fun2() {
    int x = 4;
    printf("fun2: %d\n", x);
}
```

O que será impresso?

Exemplo 3

```c
#include <stdio.h>

void fun1();
void fun2(int x);

int x = 1;

int main() {
    x = 2;
    fun1();
    fun2(x);
    printf("main: %d\n", x);
    return 0;
}

void fun1() {
    x = x + 1;
    printf("fun1: %d\n", x);
}

void fun2(int x) {
    x = x + 1;
    printf("fun2: %d\n", x);
}
```

O que será impresso?

Variáveis locais e variáveis globais

- O uso de variáveis globais deve ser evitado pois é uma causa comum de erros:
  - Partes distintas e funções distintas podem alterar a variável global, causando uma grande interdependência entre estas partes distintas de código.
- A legibilidade do seu código também pode piorar com o uso de variáveis globais:
  - Ao ler uma função que usa uma variável global é difícil inferir seu valor inicial e portanto qual o resultado da função sobre a variável global.

Exemplo utilizando funções

Exemplo utilizando funções

Em aulas anteriores vimos como testar se um número em **candidato** é primo:

```
1  divisor = 2;
2  eh_primo = 1;
3  while (divisor <= candidato/2) {
4      if (candidato % divisor == 0) {
5          eh_primo = 0;
6          break;
7      }
8      divisor++;
9  }
10 if (eh_primo)
11     printf("%d, ", candidato);
```

E usamos isso para imprimir os *n* primeiros números primos:

Exemplo utilizando funções

```c
int main() {
    int divisor, n, eh_primo, candidato, primosImpressos;
    scanf("%d", &n);
    if (n >= 1) {
        printf("2, ");
        primosImpressos = 1;
        candidato = 3;
        while (primosImpressos < n) {
            divisor = 2;
            eh_primo = 1;
            while (divisor <= candidato/2) {
                if (candidato % divisor == 0) {
                    eh_primo = 0;
                    break;
                }
                divisor++;
            }
            if (eh_primo) {
                printf("%d, ", candidato);
                primosImpressos++;
            }
            candidato = candidato+2;
        }
    }
    return 0;
}
```

Exemplo utilizando funções

- Podemos criar uma função que testa se um número é primo ou não (note que isto é exatamente um bloco logicamente bem definido).
- Depois fazemos chamadas para esta função.

```
1  int eh_primo(int candidato) {
2      int divisor = 2;
3
4      while (divisor <= candidato/2) {
5          if (candidato % divisor == 0) {
6              return 0;
7          }
8          divisor++;
9      }
10
11     return 1;
12 }
```

Exemplo utilizando funções

```c
#include <stdio.h>

int eh_primo(int candidato); /* retorna 1 se candidato é primo, e 0 caso contrário */

int main() {
    int n, candidato, primosImpressos;

    scanf("%d", &n);

    if (n >= 1) {
        printf("2, ");
        primosImpressos = 1;
        candidato = 3;
        while (primosImpressos < n) {
            if (eh_primo(candidato)) {
                printf("%d, ", candidato);
                primosImpressos++;
            }
            candidato = candidato + 2;
        }
    }

    return 0;
}
```

Exercícios

Exercício

Escreva uma função que computa a potência a^b para valores a (double) e b (int) passados por parâmetro (não use bibliotecas como math.h). Sua função deve ter o seguinte protótipo:

```
1 double pot(double a, int b);
```

Use a função anterior e crie um programa que imprima todas as potências:

$$2^0, 2^1, \ldots, 2^{10}, 3^0, \ldots, 3^{10}, \ldots, 10^{10}.$$

Exercício

Escreva uma função que computa o fatorial de um número n passado por parâmetro. Sua função deve ter o seguinte protótipo:

```
1 long int fat(int n);
```

OBS: Caso $n \leq 0$, seu programa deve retornar 1.

Use a função anterior e crie um programa calcula o coeficiente binomial de dois números n e k lidos:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Pergunta: o que acontece se você define a função como sendo do tipo **int**?

Exercício

Escreva uma função em C para computar a raiz quadrada de um número positivo. Use a idéia abaixo, baseada no método de aproximações sucessivas de Newton descrito abaixo.

Seja Y um número. Sua raiz quadrada é raiz da equação

$$f(x) = x^2 - Y \ .$$

A primeira aproximação é $x_1 = Y/2$. A $(n+1)$-ésima aproximação é

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

A função deverá retornar o valor da vigésima aproximação.

Exercício

Faça um programa que teste quais números de uma sequência fazem parte da sequência de Fibonacci.

O programa deve receber inicialmente um inteiro n. Em seguida, deve receber n números inteiros positivos.

A resposta consiste de uma única linha, com os números da sequência da entrada que fazem parte da sequência de Fibonacci separados por um espaço.

Por exemplo, se a entrada é "`7 3 9 -7 4 1 0 3`", então a saída é "`3 1 0 3`".

Exercício

Escreva um programa que mostre os dois números primos mais próximos de um dado número n.

Por exemplo, se $n = 24$, os dois números primos mais próximos dele são 23 e 29. Se $n = 47$, então os dois números primos mais próximos dele são 47 e 53.