MC102 – Algoritmos e Programação de Computadores

Instituto de Computação

UNICAMP

Segundo Semestre de 2013

Roteiro

Funções

- Um aspecto importante na resolução de um problema complexo é conseguir dividi-lo em subproblemas menores.
- Ao criarmos um programa para resolver um determinado problema, uma tarefa crítica é dividir o código grande em partes menores, fáceis de serem compreendidas e mantidas.

Funções

Funções

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

Exemplo:

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

Funções

As funções podem retornar um valor ao final de sua execução.

Exemplo:

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

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, evitando inconsistências e facilitando alterações.

Definindo uma função

Uma função é definida da seguinte forma:

```
tipo nome(tipo parâmetro1, ..., tipo parâmetroN) {
  comandos;
  return valor_de_retorno;
}
```

- Toda função deve ter um tipo que determina seu valor de retorno.
- Os parâmetros são variáveis que serão utilizadas pela função. Tais variáveis são inicializadas com valores na chamada de execução da função.

Exemplo de função

A função abaixo soma dois valores, passados como parâmetros:

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

- Notem que o valor de retorno é do mesmo tipo definido no retorno da função.
- Quando o comando `return` é executado, a função termina sua execução e retorna o valor indicado para quem fez a chamada da função.

Exemplo de função

Agora podemos usar esta função:

```
#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;
}
```

Importante: seu programa sempre começa executando os comandos da função `main`.

Definindo uma função

Uma função pode não ter parâmetros, neste caso, basta não informá-los:

```
#include <stdio.h>

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

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

int main() {
  int x1, x2;
  x1 = leNumero();
  x2 = leNumero();
  printf("Valor da soma: %d\n", soma(x1,x2));

  return 0;
}
```

Definindo uma função

A expressão contida dentro do comando `return` é chamado de valor de retorno (é a resposta da função). Nada após ele será executado.

```
#include <stdio.h>

int leNumero() {
 int n;
 printf("Digite um numero:");
 scanf("%d", &n);

 return n;
 printf("bla bla bla bla!"); /* nao imprime esta mensagem */
}

int main() {
  int x1, x2;
  x1 = leNumero();
  x2 = leNumero();
  printf("Soma: %d\n", x1 + x2);

  return 0;
}
```

Definindo uma função

- As funções só podem ser definidas fora de outras funções.
- Lembre-se que o corpo do programa principal (`main()`) é uma função.

Invocando uma função

Uma forma comum 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));
```

Invocando uma função

- Para cada um dos parâmetros da função, devemos fornecer um valor, de mesmo tipo, na chamada da função.
- Ao chamar uma função passando variáveis 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.

Invocando uma função

```
#include <stdio.h>

int somaEsquisita(int x, int y) {
  x = x + 1;
  y = y + 1;
  return (x + y);
}

int main() {
  int a = 10, b = 5;

  printf("Soma de a e b: %d\n", a + b);
  printf("Soma de x e y: %d\n", somaEsquisita(a, b));
  printf("a: %d\n", a);
  printf("b: %d\n", b);

  return 0;
}
```

Os valores de a e b não são alterados por operações feitas em x e y!

O tipo `void`

- O tipo `void` é um tipo especial.
- Este tipo é utilizado para indicar que uma função não retorna nenhum valor.

O tipo void

- Por exemplo, a função abaixo imprime o número inteiro que for passado para ela como parâmetro:

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

O tipo void

```
#include <stdio.h>

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

int main() {
  imprime(10);
  imprime(20);

  return 0;
}
```

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;
}
```

Definindo funções depois do `main`

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

Declarando funções depois do main

```
#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);
}
```

Dependendo do compilador, ocorrerá um erro de compilação!

Declarando uma função sem defini-la

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

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

- O protótipo de uma função deve vir sempre antes do seu uso.
- É comum colocar os protótipos de funções no início do arquivo do programa.

Protótipos de funções

```
#include <stdio.h>

float soma(float op1, float op2);

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

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

Protótipos de funções

```
#include <stdio.h>

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

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

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

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

Números primos

- Em aulas anteriores, vimos como testar se um número é primo:

```
divisor = 2;
primo = 1;

while (primo && (divisor <= candidato / 2)) {
  if (candidato % divisor == 0)
    primo = 0;
  divisor++;
}

if (primo)
  printf(%d", candidato);
```

Números primos

- É fácil escrever um programa para imprimir os n primeiros números primos (veja no próximo slide).

Números primos

```
#include <stdio.h>
int main() {
  int divisor = 0, n = 0, primo = 0, candidato = 2, impressos = 0;

  printf("Numero de primos a imprimir: ");
  scanf("%d", &n);

  while (impressos < n) {
    divisor = 2;
    primo = 1;
    while (primo && (divisor <= candidato / 2)) {
      if (candidato % divisor == 0) primo = 0;
      divisor++;
    }
    if (primo) {
      printf("%d\n", candidato);
      impressos++;
    }
    candidato++;
  }

  return 0;
}
```

Números primos

- Se o número de primos a ser impresso é negativo, usaremos o valor absoluto deste.
- Como refazer este código 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).
- Vamos criar também uma função que retorna o valor absoluto de um número.
- Depois fazemos chamadas para estas funções.

Números primos

```
#include <stdio.h>

int testaPrimo(int candidato); /* verifica se um candidato eh primo */
int valorAbs(int x); /* calcula o valor absoluto de x */

int main() {
  int divisor = 0, n = 0, primo = 0, candidato = 2, impressos = 0;

  printf("Numero de primos a imprimir: ");
  scanf("%d", &n);
  n = valorAbs(n);

  while (impressos < n) {
    if (testaPrimo(candidato)) {
      printf("%d\n", candidato);
      impressos++;
    }
    candidato++;
  }

  return 0;
}
```

Números primos

```
int valorAbs(int x) {
  if (x < 0)
    return -x;
  else
    return x;
}

int testaPrimo(int candidato) {
  int divisor, primo;
  divisor = 2;
  primo = 1;

  while (primo && (divisor <= candidato / 2)) {
    if (candidato % divisor == 0)
      primo = 0;
    divisor++;
  }

  if (primo)
    return 1; /* se for primo, retorna 1 (verdadeiro) */
  else
    return 0; /* se nao for, retorna 0 (falso) */
}
```

Números primos

- O código é mais claro quando utilizamos funções.
- Também é mais fácil de fazer alterações.
- Exemplo: melhorar o teste de primalidade.
  - Testar se o candidato é um número par maior que 2 (não é primo).
  - Se for ímpar, testar apenas divisores ímpares (3, 5, 7, etc).
- O uso de funções facilita modificações no código. Neste caso, altera-se apenas a função `testaPrimo`.

Números primos

```
int testaPrimo(int candidato) {
  int divisor, primo;

  if (candidato % 2 == 0)
    return (candidato == 2);

  divisor = 3;
  primo = 1;
  while (primo && (divisor <= candidato / 2)) {
    if (candidato % divisor == 0)
      primo = 0;
    divisor = divisor + 2;
  }

  return primo;
}
```