

Funções I

Instituto de Computação – Unicamp

12 de Abril de 2012

- printf
- scanf
- sqrt
- pow
- ...

Roteiro

- 1 Funções
- 2 O tipo void
- 3 A função `main`
- 4 Protótipo de funções
- 5 Exemplo Maior

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.

Funções

Funções

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

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

Funções

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

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

Definindo uma função

Uma função é definida da seguinte forma:

```
tipo nome(tipo parâmetro1, ..., tipo parâmetroN) {  
    comandos;  
    return valor de retorno;  
}
```

- **nome**: identifica a função; é usado na chamada/invocação da função; definido da mesma forma que definimos nomes de variáveis.
- Toda função deve ter um **tipo**. Esse tipo determina qual será o tipo de seu valor de **retorno**.
- Os **parâmetros** são variáveis de entrada de uma função. Tais variáveis são inicializadas com valores na **chamada** da função.
- No corpo da função podemos escrever **qualquer código que escreveríamos no "main", inclusive declarar variáveis**

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 compreendidos de forma isolada**.
- Permitir o **reaproveitamento de código** já construído (por você ou por outros programadores). **Ex: sqrt**
- **Evitar** que um **trecho de código** seja **repetido** várias vezes dentro de um mesmo programa, minimizando erros e facilitando alterações.

Exemplo de função

A função abaixo soma dois valores, passados como parâmetro:

```
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 para de executar 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);
}
```

Em que ponto deste código o programa começa a execução?

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Definindo uma função

Uma função pode não ter parâmetros, 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("Soma e: %d\n",soma(x1,x2));
}
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

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);
}
```

Obs: Seu programa sempre começa executando os comandos da função **main**.

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

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.**

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

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

int main(){
    int x1, x2;
    x1 = leNumero();
    x2 = leNumero();
    printf("Soma e: %d\n",soma(x1,x2));
}
```

Não imprime *bla bla bla bla*!

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

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.

Chamando/Invocando uma função

Parâmetros na chamada da função:

- Ao chamar a função, o valor de cada parâmetro deve ser informado
- Um valor para cada parâmetro
- Valores do mesmo tipo do parâmetro
- Valores na mesma ordem que na declaração
- 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.

Chamando/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));
```

Chamando/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, b;  
  
    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 que acontece quando chamamos uma função

- 1 Interrompe o programa principal (aquele que está no “main”)
- 2 Passa dados para a função
- 3 Desvia a execução para a função
- 4 Executa o código da função
- 5 Retorna o resultado
- 6 Continua a execução do programa principal

O tipo void

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

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

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.
- Este tipo é utilizado para indicar que uma função não retorna nenhum valor.

O tipo void

```
#include <stdio.h>  
  
void imprime (int numero) {  
    printf ("Número %d\n", numero);  
}  
  
int main () {  
    imprime (10);  
    imprime (20);  
  
    return 0;  
}
```

Precisamos atribuir para alguma variável a chamada de uma função void?

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

```
#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, ocorre um erro de compilação!

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 uma função sem defini-la

- Para organizar melhor um programa e podermos implementar funções em partes distintas do arquivo é utilizado **protótipo 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**.

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

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

Protótipo 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);
}
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Exemplo Maior

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

```
divisor = 2;
eprimo=1;
while( (divisor<=candidato/2) && (eprimo) ){
    if(candidato % divisor == 0)
        eprimo=0;
    divisor++;
}
if(eprimo)
    printf(" %d, ", candidato);
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Protótipo 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);
}
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Exemplo Maior

- Agora usamos este código para imprimir os n primeiros números primos:
- Veja no próximo slide.

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺

Exemplo Maior

```
int main(){
    int divisor=0, n=0, eprimo=0, candidato=0, primosImpr=0;
    printf("\n Digite numero de primos a imprimir:");
    scanf("%d",&n);
    if(n>0){
        printf("2, ");
        primosImpr=1;
        candidato=3;
        while(primosImpr < n){
            divisor = 2;
            eprimo=1;
            while( (divisor <= candidato/2) && (eprimo) ){
                if(candidato % divisor == 0)
                    eprimo=0;
                divisor++;
            }
            if(eprimo){
                printf("%d, ",candidato);
                primosImpr++;
            }
            candidato=candidato+2;//Testa proximo numero
        }
    }
}
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍

Exemplo Maior

```
#include <stdio.h>

int ePrimo(int candidato); //retorna 1 se candidato e primo; e 0 caso contrário
int valorAbs(int x); //retorna valor absoluto de x

int main(){
    int divisor=0, n=0, eprimo=0, candidato=0, primosImpr=0;

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

    printf("2, ");
    candidato = 3;
    primosImpr = 1;
    while(primosImpr < n){
        if(ePrimo(candidato)){
            printf("%d, ",candidato);
            primosImpr++;
        }
        candidato=candidato+2;
    }
}
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍

Exemplo Maior

- Se o número de primos a ser impresso é negativo usaremos o valor absoluto deste.
- Como refazer este código de forma elegante e clara 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.

Exemplo Maior

```
int valorAbs(int x){
    if(x < 0)
        return -1*x;
    else
        return x;
}

int ePrimo(int candidato){
    int divisor, eprimo;
    divisor = 2;
    eprimo=1;
    while( (divisor <= candidato/2) && (eprimo) ){
        if(candidato % divisor == 0)
            eprimo=0;
        divisor++;
    }
    if(eprimo)
        return 1;
    else
        return 0;
}
```

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍

- O código é mais claro quando utilizamos funções.
- Também é mais fácil fazer alterações.
- Exemplo: queremos otimizar o teste de primalidade, e para tanto não vamos testar todos os divisores $2, \dots, (\text{candidato} - 1)$.
 - ▶ Testar se número é par maior que 2 (não é primo).
 - ▶ Se for ímpar, testar divisores ímpares $3, 5, 7, \dots$
- O uso de funções facilita modificações no código. Neste caso altera-se apenas a função **ePrimo**.

- Escreva uma função para calcular a soma de três números reais.
- Escreva uma função para calcular a subtração de dois inteiros.
- Escreva um programa que pega n números inteiros e que determina quais destes números são pares. Escreva uma função que determina se o número é par.

Exemplo Maior

Função **ePrimo** é alterada para:

```
int ePrimo(int candidato){
    int divisor, eprimo;

    if( (candidato>2) && (candidato % 2 == 0) )//se for par > 2
        return 0;

    divisor = 3;
    eprimo=1;
    while( (divisor <= candidato/2) && (eprimo) ){
        if(candidato % divisor == 0)
            eprimo=0;
        divisor = divisor + 2;
    }
    if(eprimo)
        return 1;
    else
        return 0;
}
```