Funções e Procedimentos

Informática – IFSULDEMINAS

Primeiro Semestre de 2013

Roteiro

Introdução

Considere o problema a seguir:

Problema

Dado um número x, calcule o resultado da expressão $x^7 + x^4$.

Expressão $x^7 + x^4$

```
int i, pot1, pot2, soma, x;

printf("Qual o valor de x?\n"),
scanf("%d", &x);

pot1 = 1; // calcula a primeira potencia
for (i = 1; i <= 7; i++) {
    pot1 = pot1 * x;
}

pot2 = 1; // calcula a segunda potencia
for (i = 1; i <= 4; i++) {
    pot2 = pot2 * x;
}

soma = pot1 + pot2;

printf("Resultado: %d\n", soma);
```

Expressão $x^7 + x^4$ – Problemas

- Escrevemos dois blocos de código muito parecidos.
- O código se torna muito grande.
- É difícil ver o que cada bloco faz

E se pudéssemos perguntar o resultado de x^n?

Expressão $x^7 + x^4$ – Problemas

- Escrevemos dois blocos de código muito parecidos.
- O código se torna muito grande.
- É difícil ver o que cada bloco faz

E se pudéssemos perguntar o resultado de x^n?

Expressão $x^7 + x^4$ – Problemas

- Escrevemos dois blocos de código muito parecidos.
- O código se torna muito grande.
- É difícil ver o que cada bloco faz

E se pudéssemos perguntar o resultado de x^n?

Expressão $x^7 + x^4$ – Problemas

- Escrevemos dois blocos de código muito parecidos.
- O código se torna muito grande.
- É difícil ver o que cada bloco faz

E se pudéssemos perguntar o resultado de x^n?

Simplificando ...

```
int i, pot1, pot2, soma, x;

printf("Qual o valor de x?\n"),
scanf("%d", &x);

pot1 = potencia(x, 7); // calcula a primeira potencia
pot2 = potencia(x, 4); // calcula a segunda potencia
soma = pot1 + pot2;

printf("Resultado: %d\n", soma);
```

Simplificando ...

Mas...

- ...o compilador C não vai entender minha pergunta.

A não ser que...

- ...digamos a ele como calcular uma potencia!

Adicionando uma função ao vocabulário do compilador

```
int potencia(int base, int expoente) {
    int i, resultado;

    resultado = 1;
    for (i = 1; i <= expoente; i++) {
        resultado = resultado * base;
    }

    return resultado; // retorna base elevada ao expoente
}
```

Simplificando ...

Mas...

- ...o compilador C não vai entender minha pergunta.

A não ser que...

- ...digamos a ele como calcular uma potencia!

Adicionando uma função ao vocabulário do compilador

```
int potencia(int base, int expoente) {
    int i, resultado;

    resultado = 1;
    for (i = 1; i <= expoente; i++) {
        resultado = resultado * base;
    }

    return resultado; // retorna base elevada ao expoente
}
```

Simplificando ...

Mas...

- ...o compilador C não vai entender minha pergunta.

A não ser que...

- ...digamos a ele como calcular uma potencia!

Adicionando uma função ao vocabulário do compilador

```
int potencia(int base, int expoente) {
    int i, resultado;

    resultado = 1;
    for (i = 1; i <= expoente; i++) {
            resultado = resultado * base;
    }

    return resultado; // retorna base elevada ao expoente
}
```

Funções

Funções

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

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

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

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

Porque utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais e 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.

Porque utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais e 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.

Porque utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais e 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.

Porque utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais e 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.

Declarando uma função

Uma função é declarada da seguinte forma:

Declaração de função

```
tipo nome(tipo parâmetro1, ..., 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 são variáveis que serão utilizadas pela função. Tais variáveis são inicializadas com valores na chamada de execução da função.

Declarando uma função

Uma função é declarada da seguinte forma:

Declaração de função

```
tipo nome(tipo parâmetro1, ..., 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 são variáveis que serão utilizadas pela função. Tais variáveis são inicializadas com valores na chamada de execução da função.

Declarando uma função

Uma função é declarada da seguinte forma:

Declaração de função

```
tipo nome(tipo parâmetro1, ..., 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 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) {
    return a + b;
}
```

- 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.

Declarando uma função

- Se uma função não tiver parâmetros, basta não informá-los.
- A expressão após o comando `return` é chamada de valor de retorno e corresponde à resposta da função.
- O comando `return` é sempre o último a ser executado por uma função. Nada depois dele será executado!
- As funções só podem ser declaradas fora de outras funções.

Declarando uma função

- Se uma função não tiver parâmetros, basta não informá-los.
- A expressão após o comando `return` é chamada de valor de retorno e corresponde à resposta da função.
- O comando `return` é sempre o último a ser executado por uma função. Nada depois dele será executado!
- As funções só podem ser declaradas fora de outras funções.

Declarando uma função

- Se uma função não tiver parâmetros, basta não informá-los.
- A expressão após o comando `return` é chamada de valor de retorno e corresponde à resposta da função.
- O comando `return` é sempre o último a ser executado por uma função. Nada depois dele será executado!
- As funções só podem ser declaradas fora de outras funções.

Declarando uma função

- Se uma função não tiver parâmetros, basta não informá-los.
- A expressão após o comando `return` é chamada de valor de retorno e corresponde à resposta da função.
- O comando `return` é sempre o último a ser executado por uma função. Nada depois dele será executado!
- As funções só podem ser declaradas fora de outras funções.

Declarando uma função

- Se uma função não tiver parâmetros, basta não informá-los.
- A expressão após o comando `return` é chamada de valor de retorno e corresponde à resposta da função.
- O comando `return` é sempre o último a ser executado por uma função. Nada depois dele será executado!
- As funções só podem ser declaradas fora de outras funções.

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("Média de a e b: %d\n", soma(a, b) / 2);
```

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("Média de a e b: %d\n", soma(a, b) / 2);
```

Invocando uma função

```
#include <stdio.h>

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

int main () {
  int n1, n2;

  printf ("Digite o valor de n1: ");
  scanf ("%d", &n1);
  printf ("Digite o valor de n2: ");
  scanf ("%d", &n2);
  printf ("Soma de n1 e n2: %d\n", soma(n1, n2));
  printf ("Soma de 5 e 10: %d\n", soma(5, 10));

  return 0;
}
```

Procedimentos

Procedimentos

São funções que não retornam valores.

- Toda função possui um tipo que determina qual será o tipo de seu valor de retorno.
- No caso de procedimentos o tipo é o vazio, ou seja, **void**.
- Como o tipo de retorno é vazio, procedimentos não devem ter o comando **return**.

Procedimentos

Procedimentos

São funções que não retornam valores.

- Toda função possui um tipo que determina qual será o tipo de seu valor de retorno.
- No caso de procedimentos o tipo é o vazio, ou seja, **void**.
- Como o tipo de retorno é vazio, procedimentos não devem ter o comando **return**.

Procedimentos

Procedimentos

São funções que não retornam valores.

- Toda função possui um tipo que determina qual será o tipo de seu valor de retorno.
- No caso de procedimentos o tipo é o vazio, ou seja, **void**.
- Como o tipo de retorno é vazio, procedimentos não devem ter o comando **return**.

Procedimentos

Procedimentos

São funções que não retornam valores.

- Toda função possui um tipo que determina qual será o tipo de seu valor de retorno.
- No caso de procedimentos o tipo é o vazio, ou seja, **void**.
- Como o tipo de retorno é vazio, procedimentos não devem ter o comando **return**.

Exemplo de procedimento

O procedimento abaixo soma dois valores passados como parâmetros e apresenta a soma ao usuário:

```
void soma(int a, int b) {
    int soma = a + b;
    printf("A soma de %d e %d é %d.", a, b, soma);
}
```

- Notem que o tipo de retorno é **void**.
- Não há comando **return**.

Exemplo de procedimento

O procedimento abaixo soma dois valores passados como parâmetros e apresenta a soma ao usuário:

```
void soma(int a, int b) {
    int soma = a + b;
    printf("A soma de %d e %d é %d.", a, b, soma);
}
```

- Notem que o tipo de retorno é **void**.
- Não há comando **return**.

Exemplo de procedimento

O procedimento abaixo soma dois valores passados como parâmetros e apresenta a soma ao usuário:

```
void soma(int a, int b) {
    int soma = a + b;
    printf("A soma de %d e %d é %d.", a, b, soma);
}
```

- Notem que o tipo de retorno é **void**.
- Não há comando **return**.

Invocando um procedimento

Procedimentos são simplesmente invocados.

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

- Saída: **A soma de 4 e 2 é 6.**
- Procedimentos não podem ser utilizados em expressões.

Invocando um procedimento

Procedimentos são simplesmente invocados.

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

- Saída: **A soma de 4 e 2 é 6.**
- Procedimentos não podem ser utilizados em expressões.

Invocando um procedimento

Procedimentos são simplesmente invocados.

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

- Saída: **A soma de 4 e 2 é 6.**
- Procedimentos não podem ser utilizados em expressões.

Invocando uma função

```
#include <stdio.h>

void soma (int a, int b) {
  int soma = a + b;
  printf("A soma de %d e %d é %d.", a, b, soma);
}

int main () {
  int n1, n2;
  printf ("Digite o valor de n1: ");
  scanf ("%d", &n1);
  printf ("Digite o valor de n2: ");
  scanf ("%d", &n2);
  soma(n1, n2);
  soma(5, 10);
  return 0;
}
```