Programação Estruturada

Aula 11 - Funções

Yuri Malheiros (yuri@ci.ufpb.br)

Introdução

- Uma função é uma série de comandos que foram agrupados e que possui um nome
- O nome função vem da matemática, mas nem sempre uma função em C equivale a uma função matemática
    - Uma função pode não ter argumentos, nem retorno
- Até agora nossos programas tinham apenas uma função: `main`

Introdução

- Cada função é essencialmente um mini programa
- Ela pode ter declarações de variáveis e comandos
- Usando funções, nós dividimos um programa em partes menores
    - Isto organiza e facilita a compreensão e modificação do programa

Introdução

- Funções são reusáveis
- Podemos usá-las em diversas partes dos nossos programas
    - Evitamos código repetido com elas

Definindo e chamando uma função

- A função a seguir calcula a média de dois números do tipo double:

```
double media(double a, double b) {
  return (a+b)/2;
}
```

- A palavra `double` no início é o tipo de retorno da função
- `a` e `b` são os parâmetros da função
  - Esses valores devem ser passados na chamada da função
- Cada parâmetro deve ter um tipo
  - Nesse caso, `a` e `b` são do tipo `double`

Definindo e chamando uma função

```
double media(double a, double b) {
  return (a+b)/2;
}
```

- Entre chaves temos o corpo da função
- Neste caso temos apenas `return (a+b)/2;`
- Ao executar o `return` a função encerra a sua execução e o valor é retornado para a parte do código que a função foi chamada

Definindo e chamando uma função

- Para chamar uma função, escrevemos o seu nome seguido da lista de argumentos entre parênteses
  - `media(10.0, 8.0)` - `10.0` é copiado para o parâmetro `a` e `8.0` para o `b`
  - `media(x, y)` - o valor de `x` é copiado para o parâmetro `a` e o valor de `y` para o `b`
- Qualquer expressão pode ser usada como argumento
- O compilador vai usar os argumentos passados e executar o corpo da função

Definindo e chamando uma função

```c
#include <stdio.h>

double media(double a, double b) {
    return (a+b)/2;
}

int main(void) {
    printf("Média: %lf\n", media(10.0, 8.0));
    return 0;
}
```

Definindo e chamando uma função

- Vamos modificar o programa anterior para receber os valores do usuário

Definindo e chamando uma função

- Nem toda função retorna um valor
- Por exemplo, uma função que tem como objetivo imprimir uma mensagem
- Para indicar que uma função não tem retorno, usamos o tipo `void`

Definindo e chamando uma função

```c
void print_progresso(double atual, double total) {
    double porcentagem = atual/total*100;

    printf("Progresso: %.2lf/%.2lf (%.2lf%%)\n", atual, total, porcentagem);
}

int main(void) {
    print_progresso(25,70);
    return 0;
}
```

Definindo e chamando uma função

- Algumas funções não tem parâmetros
- Para isso, colocamos a palavra `void` nos parâmetros da função

```
void print_c_ou_nao_c(void) {
  printf("C ou não C, eis a questão\n");
}
```

- Para chamá-la: `print_c_ou_nao_c();`

Definindo e chamando uma função

- Forma geral de uma definição de função:

```
tipo_do_retorno nome_da_funcao(parâmetros) {
  declarações
  comandos
}
```

- Variáveis declaradas dentro da função só existem dentro da função

Definindo e chamando uma função

- Para uma função que retorna valores, podemos usá-las dentro de expressões
- `media(10.0, 8.0) + 1.0`
- `resultado = media(10.0, 8.0)`

Exercício

- Escreva um programa que recebe um número e exibe se ele é primo ou não
- Faça uma função separada para checar se o número é primo

Declaração de função

- As definições das funções estão sempre acima do `main`
- Se invertermos a ordem, o compilador mostrará mensagens de erro
- Toda função deve ter sido definida antes de ser chamada

Declaração de função

- Podemos usar uma declaração de função para poder ordenar as funções como desejarmos
    - `tipo_do_retorno nome_da_funcao(parâmetros);`

Declaração de função

```c
#include <stdio.h>

void print_progresso(double atual, double total);

int main(void) {
    print_progresso(25,70);
    return 0;
}

void print_progresso(double atual, double total) {
    double porcentagem = atual/total*100;

    printf("Progresso: %.2lf/%.2lf (%.2lf%%)\n", atual, total, porcentagem);
}
```

Declaração de função

- Este tipo de declaração também é chamada de protótipo de função
- Apesar de não recomendável, podemos omitir o nome dos parâmetros no protótipo:
  - `void print_progresso(double, double);`

Argumentos

- Parâmetros aparecem na definição da função
- Argumentos são expressões que aparecem na chamada da função

Argumentos

- Em C, argumentos são passados por valor
- Na chamada da função, cada argumento é avaliado e o seu valor é copiado para o parâmetro correspondente
- Uma mudança num parâmetro durante a execução da função não afeta o argumento
- Cada parâmetro funciona como uma variável que foi inicializada com o valor do argumento correspondente

Argumentos

- A passagem por valor tem suas vantagens e desvantagens
- Como os valores são copiados, podemos usar os parâmetros como variáveis sem nos preocupar com as variáveis que podem ter sido usadas como argumento

```
int potencia(int x, int n) {
  int result = 1;

  while (n-- > 0) {
    result *= x;
  }

  return result;
}
```

Argumentos

- Por outro lado, a passagem por valor dificulta a escrita de algumas funções
- Suponha uma função que recebe um double e decompõe ele em dois valores: parte inteira e parte fracionária
- Uma função não pode retornar dois valores
    - Então poderíamos passar duas variáveis para função que receberiam esse valor

Argumentos

```
void decompor(double x, int parte_inteira, double parte_fracionaria) {
  parte_inteira = (int) x;
  parte_fracionaria = x - parte_inteira;
}
```

- Ao chamar `decompor(3.14159, i, d);` `i` e `d` não são afetadas, pois seus valores foram copiados para `parte_inteira` e `parte_fracionaria`

Argumentos

- Ao passar um valor para função na sua chamada, o compilador pode convertê-lo implicitamente se os tipos não forem iguais
- Por exemplo, se um `int` for passado para uma função esperando um `double`, o argumento é convertido para `double`

Argumentos - arrays

- Podemos usar arrays como argumentos

```
int func(int a[]) {
  ...
}
```

- Note que o tamanho do array não precisa ser especificado

Argumentos - arrays

- Como precisamos saber o tamanho do array e ele pode variar, então passamos um segundo argumento com esse valor

```
int soma_array(int a[], int n) {
  int i, soma=0;

  for (i=0; i<n; i++)
    soma += a[i];

  return soma;
}
```

Argumentos - arrays

- Para chamar a função, no argumento do array, usamos apenas o seu nome

```
int arr[6] = {1,2,3,4,5,6};

soma_array(arr, 6);
```

Argumentos - arrays

- Uma função pode modificar os elementos de um array

```c
void store_zeros(int a[], int n) {
    for (int i=0; i<n; i++)
        a[i] = 0;
}

int main(void) {
    int a[6] = {1,2,3,4,5,6};

    store_zeros(a, 6);

    for (int i=0; i<6; i++)
        printf("%d", a[i]);

    return 0;
}
```

Argumentos - arrays

- Se um parâmetro for um array multidimensional, apenas o tamanho da primeira dimensão pode ser omitida
    - `int func(int a[][10])`

return

- Uma função que não tenha o tipo de retorno `void` deve usar o comando `return` para retornar um valor
  - `return expressão;`
- Usualmente, a expressão de retorno é uma variável
  - `return resultado;`
- Mas podemos usar expressões mais complexas:
  - `return n >= 0 ? n : 0;`

return

- Se o tipo do valor retornado for diferente do tipo especificado como retorno da função, então o compilador converte implicitamente o valor
- Por exemplo, se uma função for declarada com o tipo de retorno `int` e o valor retornado for `double`, então o compilador converte o valor para `int`

return

- Ao executar o comando `return`, a função termina imediatamente

```
int func(int i) {
    if (i > 0) {
        return i;
    }

    return 0;
}
```

return

- Como fazer esse comportamento para uma função com retorno `void`?

```
void func(int i) {
    if (i > 0) {
        printf("%d", i);
        return;
    }

    printf("0");
}
```

- O `return` pode ser usado sem ser seguido por uma expressão em funções `void`

Terminando um programa

- Como `main` é uma função, ela precisa de um tipo de retorno
- Normalmente o tipo de retorno do `main` é `int`
- O valor retornado pelo `main` é um `status code`
- Isto pode ser usado pelo sistema operacional para testar se um programa terminou normalmente
    - 0 significa terminação normal
- Podemos checar o retorno no terminal através do comando:
    - `echo $?`