Funções

Linguagem de programação
Prof. Allan Rodrigo Leite

Funções

- Função é um trecho de código que realiza uma tarefa bem definida
  - Funções contribuem para:
    - Tornar o código mais compreensível, pois o comportamento do programa é quebrado em partes menores
    - Permitem o reaproveitamento de um trecho de código entre diferentes programas, similar às bibliotecas externas `stdlib.h`, `stdio.h`, etc.

- Exemplos
  - Calcular a área de um círculo com base em um dado raio
  - Converter a temperatura em graus Celsius para Fahrenheit
  - Calcular a média de um aluno dadas as notas do semestre
  - Calcular uma série de tamanho N de Fibonacci

Funções

- Uma função possui os seguintes componentes
  - Dados de entrada
    - São os argumentos da função
    - Podem ser valores primitivos ou referência (ponteiros)
  - Comportamento da função
    - Trecho de código a ser executado ao invocar a função
  - Retorno da função
    - Tipo de dados que será retornado ao final da execução da função
    - Podem ser valores primitivos ou referências (ponteiros)
    - Quando não há retorno, deve ser informado `void`

Funções

- Exemplos de funções que já conhecemos
  - `main`: bloco principal do programa
  - `printf`: imprime uma mensagem na console
  - `scanf`: leitura de valores digitados, armazenando-os em uma variável
  - `pow`: cálculo da potência de um dado número real e base
  - `sqrt`: cálculo da raiz quadrada de um dado número real

Funções

- Vantagens da abstração de código com funções
  - Precisamos conhecer a implementação da função `printf` para usá-la?

  - Para utilizar uma função é necessário apenas conhecer:
    - Objetivo da função, isto é, qual a tarefa que ela deve desempenhar
    - Como a função deve ser utilizada
    - Argumentos de entrada e o retorno esperado da função

  - A abstração ajuda a reduzir a complexidade de um programa

Definindo funções

- Há 3 situações distintas entre a criação e utilização de uma função
  - Declaração ou assinatura
  - Definição
  - Chamada

- Declaração
  - Consiste em especificar a assinatura da função
    - Nome da função
    - Quais são os parâmetros de entrada
    - O que será retornado
  - Também conhecida como protótipo (interface) da função

Definindo funções

- Declarando uma função
  - Normalmente as declarações são realizadas no início do programa
    - Devido ao compilador do C utilizar a estratégia *top-down*
  - Sintaxe

```
<tipo retorno> nomedafuncao(<tipo argumento 1> arg1,
                            <tipo argumento 2> arg2,
                            ...
                            <tipo argumento N> argN);
```

  - Exemplo

```
float converteCelsiusFahrenheit(float celsius);
float fatorial(int n);
int soma(int, int);
```

Definindo funções

- Definindo uma função
  - Consiste em especificar o comportamento da função
    - Trecho de código que será executado ao invocar a função
  - Sintaxe

```
<tipo retorno> nomedafuncao(<tipo argumento 1> arg1,
                            ...
                            <tipo argumento N> argN) {
     //comportamento da função
}
```

  - Exemplo

```
float converteCelsiusFahrenheit(float celsius) {
     float fahrenheit = (1.8 * celsius) + 32;
     return fahrenheit;
}
```

Definindo funções

- Definindo uma função
  - Consiste em especificar o comportamento da função
    - Trecho de código que será executado ao invocar a função
  - Sintaxe

```
<tipo retorno> nomedafuncao(<tipo argumento 1> arg1,
                            ...
                            <tipo argumento N> argN) {
    //comportamento da função
}
```

  - Exemplo

```
float converteCelsiusFahrenheit(float celsius) {
    float fahrenheit = (1.8 * celsius) + 32;
    return fahrenheit;
}
```

Quando a função não possui retorno, deve ser utilizado o tipo `void`

Definindo funções

- Definindo uma função
  - Consiste em invocar uma função declarada
  - Sintaxe

```
var = nomedafuncao(<tipo argumento 1> arg1,
                   <tipo argumento 2> arg2,
                   ...
                   <tipo argumento N> argN);

nomedafuncao(<tipo argumento 1> arg1,
             <tipo argumento 2> arg2,
             ...
             <tipo argumento N> argN);
```

  - Exemplo

```
float fahrenheit = converteCelsiusFahrenheit(35.3);
```

Passagem de parâmetros

- Existem dois métodos de passagem de parâmetros para funções
  - Passagem por valor
    - Permite usar dentro de uma função uma cópia do valor de uma variável
    - Porém, não permite alterar o valor da variável original (somente a cópia)
    - Passagem de parâmetro por valor ao usar variáveis primitivas ou estruturas

  - Passagem por referência
    - É passada para a função uma referência da variável
    - Assim, é possível alterar o conteúdo da variável original usando a referência
    - Neste caso, a referência se refere à um ponteiro
    - Passagem de parâmetro por valor ao usar vetores ou ponteiros

Passagem de parâmetros

- Exemplo de passagem por valor

```c
void teste(int a) {
  a = 10;
}

int main() {
   int a = 5;

   teste(a);
   printf("a = /%d\n", a); //Imprime 5 ou 10?

  return 0;
}
```

Passagem de parâmetros

- Exemplo de passagem por referência

```c
void teste(int a[]) {
  a[0] = 10;
}

int main() {
   int a[] = {5,6,7};

   teste(a);
   printf("a = %d\n", a[0]); //Imprime 5 ou 10?

  return 0;
}
```

Passagem de parâmetros

- Exemplo de passagem por referência

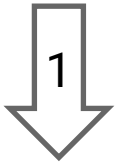
```c
void teste(int *a) {
  *a = 10;
}

int main() {
   int *a = malloc(sizeof(int));
   *a = 5;

   teste(a);
   printf("a = %d\n", *a); //Imprime 5 ou 10?

  return 0;
}
```

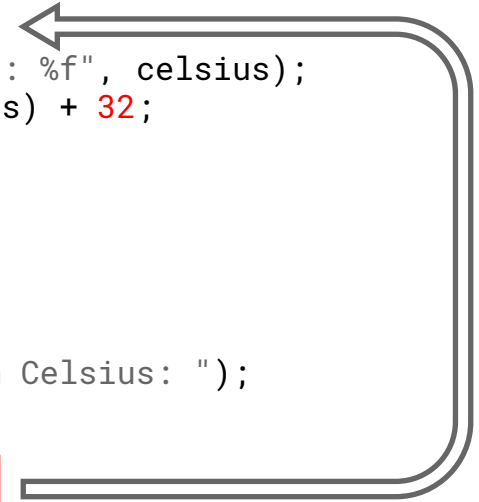
Pilha de execução

```c
#include <stdio.h>

float converte(float celsius) {
    printf("\nTemperatura em Celsius: %f", celsius);
    float fahrenheit = (1.8 * celsius) + 32;
    return fahrenheit;
}

int main() {
    float celsius, fahrenheit;

    printf("Entre com temperatura em Celsius: ");
    scanf("%f", &celsius);

    fahrenheit = converte(celsius);

    printf("\nTemperatura em Fahrenheit: %f\n", fahrenheit);
    return 0;
}
```

Pilha de execução

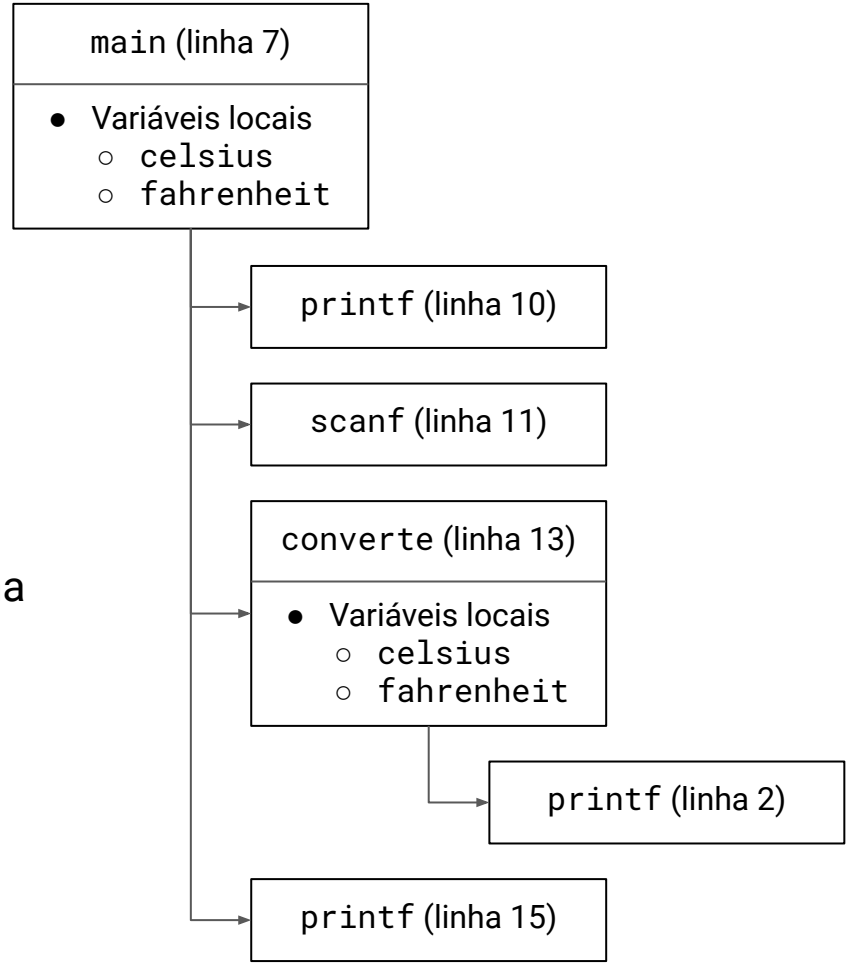
```c
#include <stdio.h>

01      float converte(float celsius) {
02          printf("\nTemperatura em Celsius: %f", celsius);
03          float fahrenheit = (1.8 * celsius) + 32;
04          return fahrenheit;
05      }
06
07      int main() {
08          float celsius, fahrenheit;
09
10          printf("Entre com temperatura em Celsius: ");
11          scanf("%f", &celsius);
12
13          fahrenheit = converte(celsius);
14
15          printf("\nTemperatura em Fahrenheit: %f\n", fahrenheit);
16          return 0;
17      }
```





Pilha de execução

- Definição
  - Pilha que armazena informações sobre as funções em execução
  - Mantém um registro do ponto em que cada função deve retornar
    - Quando a função invocada termina de executar
  - Variáveis declaradas dentro da função possuem escopo local
    - serão eliminadas da memória ao final da execução da função



Recursividade

- Recursão significa partir de uma função, chamar a mesma função
  - Produz uma execução iterativa
  - Normalmente substitui o uso de uma estrutura de repetição
  - A cada execução é gerada uma sequência na pilha de execução
- Exemplo do cálculo da potência

```
float pot(float base,int exp) {
    if (exp == 0) {
        return 1;
    } else {
        return base * pot(base,exp - 1);
    }
}
```

Implementação recursiva

```
float pot(float base, int exp) {
  float pot = 1;
  for (int i = 0; i < exp; i++) {
    pot *= base;
  }
  return pot;
}
```

Implementação iterativa

Recursividade

- Vantagens
  - Torna o código mais legível, simples e elegante
    - Sobretudo quando é necessário manter o estado das iterações do algoritmo

- Desvantagens
  - Quando a recursividade gera uma pilha de execução muito grande
    - O algoritmo demanda muita memória para as chamadas recursivas
    - Neste caso, a solução iterativa é mais eficiente pois requer menos memória

Exercícios

1. Faça um programa que solicite dois números positivos e, em seguida, exiba o produto destes números. Não utilize o operador aritmético `*`. Ao invés disso, utilize uma função recursiva para realizar este cálculo.
    - Exemplo, dado os números aleatório 7 e 3, a saída é 21.

2. Faça um programa que solicite dois números positivos e, em seguida, exiba o quociente inteiro da divisão desses números. Não utilize o operador aritmético `/`. Ao invés disso, utilize uma função recursiva para realizar este cálculo.
    - Exemplo, dado os números aleatório 19 e 3, a saída é 6.

3. A série de Fibonacci é uma sequência de números inteiros, começando por 1 e 1, onde cada termo subsequente corresponde à soma dos dois anteriores. Faça um programa que solicite o *n*-ésimo termo da série e retorne o número correspondente ao termo por meio de uma função recursiva.
    - Exemplo 1, dado o termo 7, a saída esperada é 13.
    - Exemplo 2, dado o termo 9, a saída esperada é 34.

Exercícios

4. Faça um programa que leia um número inteiro e retorne o fatorial deste número. Utilize uma função recursiva para realizar o cálculo do fatorial.

5. Faça um programa que leia uma cadeia de caracteres e retorne uma nova cadeia de caracteres invertida. Utilize uma função recursiva para realizar a inversão da sequência de caracteres.
   - Exemplo, dado o termo `"abc"`, a saída esperada é `"cba"`.

6. Faça um programa que, dado dois números inteiros `x` e `y`, retorne o máximo divisor comum entre os números. Utilize uma função recursiva para realizar o cálculo do máximo divisor comum.
   a. `mdc(x,y) = y`, se `x ≥ y` e `x mod y = 0`
   b. `mdc(x,y) = mdc(y, x)`, se `x < y`
   c. `mdc(x,y) = mdc(y,x mod y)`, caso contrário.

7. Considere uma escadaria com n degraus e você pode subir 1 ou 2 degraus por vez. Faça um programa que, dado n, retorne o número de maneiras únicas para subir a escada.
   - Exemplo, dado `n = 4`, existem 5 maneiras exclusivas
     - `[1,1,1,1]`, `[2,1,1]`, `[1,2,1]`, `[1,1,2]`, `[2, 2]`

Funções

Linguagem de programação
Prof. Allan Rodrigo Leite