MC-102 — Aula 13
Funções II

Eduardo C. Xavier

Instituto de Computação – Unicamp

28 de Junho de 2017

Roteiro

Variáveis locais e variáveis globais

- Uma variável é chamada local se ela foi declarada dentro de uma função. Nesse caso ela existe somente dentro da função, e após o término da execução desta, a variável deixa de existir. **Variáveis parâmetros também são variáveis locais**
- Uma variável é chamada global se ela for declarada fora de qualquer função. Essa variável é visível em todas as funções. Qualquer função pode alterá-la e ela existe durante toda a execução do programa.

Organização de um Programa

- Em geral um programa é organizado da seguinte forma:

```
#include <stdio.h>
#include <outras bibliotecas>

Protótipos de funções

Declaração de Variáveis Globais

int main(){
  Declaração de variáveis locais
  Comandos;
}

int fun1(Parâmetros){ //Parâmetros também são variáveis locais
  Declaração de variáveis locais
  Comandos;
}

int fun2(Parâmetros){ //Parâmetros também são variáveis locais
  Declaração de variáveis locais
  Comandos;
}
...
...
```

Escopo de variáveis

- O escopo de uma variável determina de quais partes do código ela pode ser acessada, ou seja, de quais partes do código a variável é visível.
- A regra de escopo em C é bem simples:
  - As variáveis globais são visíveis por todas as funções.
  - As variáveis locais são visíveis apenas na função onde foram declaradas.

Escopo de variáveis

```c
#include<stdio.h>

void fun1();
int fun2(int local_b);

int global;

int main() {
  int local_main;
  /* Neste ponto são visíveis global e local_main */
}

void fun1() {
  int local_a;
  /* Neste ponto são visíveis global e local_a */
}

int fun2(int local_b){
  int local_c;
  /*Neste ponto são visíveis global, local_b e local_c*/
}
```

Escopo de variáveis

- É possível declarar variáveis locais com o mesmo nome de variáveis globais.
- Nesta situação, a variável local "esconde" a variável global.

```c
#include <stdio.h>

void fun();

int nota = 10;

int main(){
    nota = 20;
    fun();
}

void fun() {
  int nota;
  nota = 5;
  /* Neste ponto nota é a variável local de fun. */
}
```

Exemplo 1

```c
#include <stdio.h>

void fun1();
void fun2();

int x;
int main(){
  x = 1;
  fun1();
  fun2();
  printf("main: %d\n", x);
}

void fun1(){
  x = x + 1;
  printf("fun1: %d\n",x);
}

void fun2(){
  int x = 3;
  printf("fun2: %d\n",x);
}
```

O que será impresso ?

Exemplo 2

```c
#include <stdio.h>

void fun1();
void fun2();

int x = 1;
int main(){
  int x=1;
  fun1();
  fun2();
  printf("main: %d\n", x);
}

void fun1(){
  x = x + 1;
  printf("fun1: %d\n",x);
}

void fun2(){
  int x = 4;
  printf("fun2: %d\n",x);
}
```

O que será impresso ?

Exemplo 3

```c
#include <stdio.h>

void fun1();
void fun2(int x);

int x = 1;
int main(){
  x=2;
  fun1();
  fun2(x);
  printf("main: %d\n", x);
}

void fun1(){
  x = x + 1;
  printf("fun1: %d\n",x);
}

void fun2(int x){
  x = x + 1 ;
  printf("fun2: %d\n",x);
}
```

O que será impresso ?

Variáveis locais e variáveis globais

- O uso de variáveis globais deve ser evitado pois é uma causa comum de erros:
  - Partes distintas e funções distintas podem alterar a variável global, causando uma grande interdependência entre estas partes distintas de código.
- A legibilidade do seu código também piora com o uso de variáveis globais:
  - Ao ler uma função que usa uma variável global é difícil inferir seu valor inicial e portanto qual o resultado da função sobre a variável global.

Vetores em funções

- Vetores também podem ser passados como parâmetros em funções.
- Ao contrário dos tipos simples, vetores têm um comportamento diferente quando usados como parâmetros de funções.
- Quando uma variável simples é passada como parâmetro, seu valor é atribuído para uma nova variável local da função.
- No caso de vetores, **não é criado** um novo vetor!
- Isto significa que os valores de um vetor **são alterados** dentro de uma função!

Vetores em funções

```c
#include <stdio.h>

void fun1(int vet[], int tam){
  int i;
  for(i=0;i<tam;i++)
     vet[i]=5;
}

int main(){
  int x[10];
  int i;

  for(i=0;i<10;i++)
    x[i]=8;

  fun1(x,10);
  for(i=0;i<10;i++)
    printf("%d\n",x[i]);
}
```

O que será impresso?

Vetores em funções

- No exemplo anterior note que a função **fun1** recebe o vetor como parâmetro e um inteiro que especifica o seu tamanho.

```
void fun1(int vet[], int tam){
  int i;
  for(i=0;i<tam;i++)
     vet[i]=5;
}
```

- Esta é a forma padrão para se receber um vetor como parâmetro.
- Um vetor possui um tamanho definido, mas em geral usa-se menos posições do que o seu tamanho. Além disso a função pode operar sobre vetores de diferentes tamanhos, bastando informar o tamanho específico de cada vetor na variável **tam**.

Vetores em funções

- Vetores não podem ser devolvidos por funções.

```
#include <stdio.h>

int[] leVet() {
  int i, vet[100];
  for (i = 0; i < 100; i++) {
    printf("Digite um numero:");
    scanf("%d", &vet[i]);
  }
  return vet;
}
```

- O código acima não compila, pois não podemos retornar um **int[]** .

Vetores em funções

- Mas como um vetor é alterado dentro de uma função, podemos criar a seguinte função para leitura de vetores.

```
#include <stdio.h>

void leVet(int vet[], int tam){
  int i;
  for(i = 0; i < tam; i++){
    printf("Digite numero:");
    scanf("%d",&vet[i]);
  }
}
```

- A função abaixo faz a impressão de um vetor.

```
void escreveVet(int vet[], int tam){
  int i;
  for(i=0; i< tam; i++)
    printf("vet[%d] = %d\n",i,vet[i]);
}
```

Vetores em funções

- Podemos usar as funções anteriores no programa abaixo.

```
int main(){
  int vet1[10], vet2[20];

  printf(" ------- Lendo Vetor 1 ---------\n");
  leVet(vet1,10);
  printf(" ------- Lendo Vetor 2 ---------\n");
  leVet(vet2,20);

  printf(" ------- Imprimindo Vetor 1 ---------\n");
  escreveVet(vet1,10);
  printf(" ------- Imprimindo Vetor 2 ---------\n");
  escreveVet(vet2,20);

}
```

Exemplo 2 - Aposentadoria

- Vamos fazer uma aplicação que determina o montante final que teríamos caso tivéssemos a liberdade de aplicar os descontos previdenciários de nosso salário.
- A aplicação também deverá calcular o tempo de aposentadoria que teríamos considerando resgates mensais de valor igual ao salário de contribuição.

Exemplo 2 - Aposentadoria

- O programa terá como dados de entrada:
  - O salário mensal.
  - A taxa de juros real onde serão aplicados os recursos.
  - O tempo de contribuição em anos.

Exemplo 2 - Aposentadoria

- Primeiramente precisamos determinar qual é o valor de contribuição mensal.
- No site da previdência (para 2017) temos

Tabela: Contribuição Previdênciaria

Salário	**Alíquota**
Até R$1.659,38	8%
De R$ 1.659,39 a R$ 2.765,66	9%
De R$ 2.765,67 até R$ 5.531,31	11%

- No regime do INSS a aposentadoria é limitada à R$ 5.531,31, mas por simplificação, assumimos uma taxa de 11% sobre o salário total para salários maiores que R$ 2.765,67 (o valor da aposentadoria será igual ao salário).
- A contribuição patronal é sempre 20% do salário.

Exemplo 2 - Aposentadoria

- Vamos criar então duas funções que dado um salário, devolvem o valor de contribuição do empregado e empregador.

```
double contrib_empregado(double salario){
  if(salario <= 1659.38)
    return 0.08*salario;
  else if(salario <= 2765.66)
    return 0.09*salario;
  else
    return 0.11*salario;
}

double contrib_patronal(double salario){
  return 0.2 * salario;
}
```

Exemplo 2 - Aposentadoria

- Faremos aplicações mensais do valor de contribuição do salário.
- Dada a taxa de juros anual da aplicação precisamos da taxa mensal. Note que um montante M aplicado em 12 meses rende

$$M(1 + \text{juros\_mensal})^{12}$$

- Este valor deve ser igual ao valor M aplicado 1 ano com taxa juros_anual.

$$M(1 + \text{juros\_mensal})^{12} = M(1 + \text{juros\_anual})$$

$$\text{juros\_mensal} = \sqrt[12]{(1 + \text{juros\_anual})} - 1$$

Exemplo 2 - Aposentadoria

- Baseado na fórmula abaixo criamos a função que devolve a taxa de juros mensal, dada a taxa anual.

$$\text{juros\_mensal} = \sqrt[12]{(1 + \text{juros\_anual})} - 1$$

```
double taxa_anual_mensal(double juros_a){
  double juros_m;

  juros_m = pow(1+juros_a, 1.0/12.0) - 1;
  return juros_m;;
}
```

- A função **taxa_anual_mensal** usa a função **pow** da biblioteca **math.h**. A função **pow** devolve o primeiro parâmetro elevado ao segundo parâmetro.

Exemplo 2 - Aposentadoria

- Com as funções anteriores definidas, podemos agora criar a função que devolve o montante final, ao se fazer aplicações mensais durante um certo período:

```
double valor_final(double salario, double juros_a, int anos_contr);
```

- A função recebe como parâmetros o **salário**, a taxa de juros anual da aplicação **juros_a**, e o número de anos contribuídos **anos_contri**.

Exemplo 2 - Aposentadoria

- Suponha aplicações mensais de um valor C com uma taxa de juros mensal j.
- Após 1 período teremos o total de $C(1+j)$.
- Após 2 períodos teremos o total de $C(1+j)^2 + C(1+j)$.
- Após 3 períodos teremos o total de $C(1+j)^3 + C(1+j)^2 + C(1+j)$.
- Após n períodos teremos o total de

$$C \sum_{i=1}^{n} (1+j)^i = C \left[\frac{(1+j)[(1+j)^n - 1]}{j} \right]$$

Exemplo 2 - Aposentadoria

- Com a fórmula abaixo podemos criar a função **valor_final**.

$$C \sum_{i=1}^{n} (1+j)^i = C \left[\frac{(1+j)[(1+j)^n - 1]}{j} \right]$$

```
double valor_final(double salario, double juros_a, int anos_contr){
  double meses_contr, valor_contr, juros_m, final;

  meses_contr = 12*anos_contr;
  valor_contr = contrib_empregado(salario) + contrib_patronal(salario);
  juros_m = taxa_anual_mensal(juros_a);

  final = valor_contr*( (1+juros_m)*(pow(1+juros_m, meses_contr) -1)/juros_m);
  return final;
}
```

- Por simplificação estamos usando 12 salários anuais (não contamos o décimo terceiro).

Exemplo 2 - Aposentadoria

- Dado um montante final M, vamos agora computar por quanto tempo a aposentadoria está garantida considerando-se resgates mensais com valor igual ao do salário de contribuição.
- Vamos considerar que M sofre a perda de um resgate por mês, mas é corrigido pela mesma taxa da aplicação.
- Supondo M o montante final, S os saques mensais e j a taxa de juros da aplicação, quantos resgates podem ser realizados?

Exemplo 2 - Aposentadoria

- Supondo M o montante final, S os saques mensais e j a taxa de juros da aplicação, quantos resgates podem ser realizados?
- Vamos criar um algoritmo para determinar isto:

```
periodos = 0
Enquanto M - S > 0 faça:
  M = (M - S)
  M = M*(1+j) //O que sobrar é corrigido pala taxa j
   meses = meses + 1

meses = meses + M/S
```

Exemplo 2 - Aposentadoria

- Podemos implementar diretamente o algoritmo anterior em C, em um função:

```
double resgates_possiveis(double mont_final, double salario, double juros_a){
  double meses=0, juros_m;

  juros_m = taxa_anual_mensal(juros_a);
  while(mont_final-salario > 0){
    mont_final = mont_final - salario;
    mont_final = mont_final*(1+juros_m);
    meses++;
  }
  meses = meses + mont_final/salario;
  return meses;
}
```

Exemplo 2 - Aposentadoria

- Na função implementada, se M for grande e j for alta, a chamada da função entra em **loop infinito**.
- Por que??

```
double resgates_possiveis(double mont_final, double salario, double juros_a){
  double meses=0, juros_m;

  juros_m = taxa_anual_mensal(juros_a);
  while(mont_final-salario > 0){
    mont_final = mont_final - salario;
    mont_final = mont_final*(1+juros_m);
    meses++;
  }
  meses = meses + mont_final/salario;
  return meses;
}
```

Exemplo 2 - Aposentadoria

- Na função implementada, se M for grande e j for alta, a chamada da função entra em **loop infinito**. Por que??
- Nesta situação o rendimento de M é maior do que o valor resgatado por mês, o que garante uma aposentadoria vitalícia.
- Devemos corrigir a função para tratar este caso, devolvendo a constante **INT_MAX** definida em **limits.h** para avisar a função invocadora que estamos no caso de renda vitalícia.

```
double resgates_possiveis(double mont_final, double salario, double juros_a){
  double meses=0, juros_m;

  juros_m = taxa_anual_mensal(juros_a);

  if( (1+juros_m)*(mont_final-salario) > mont_final)
  //correcao do montante final deduzido salario ainda cresce
  //garantindo renda por tempo indeterminado
    return INT_MAX;

  while(mont_final-salario > 0){
    mont_final = mont_final - salario;
    mont_final = mont_final*(1+juros_m);
    meses++;
  }
  meses = meses + mont_final/salario;
  return meses;
}
```

Exemplo 2 - Aposentadoria

- Com todas as funções anteriores podemos criar a **main**:

```
int main(){
  double salario, juros_anual, final,  meses_resgate;
  int anos;

  printf("Salario: ");
  scanf("%lf", &salario);

  printf("Taxa Juros Real Anual Aplicacao em %% :");
  scanf("%lf", &juros_anual);
  juros_anual = juros_anual/100.0;

  printf("Anos Contribuicao: ");
  scanf("%d", &anos);

  final = valor_final(salario, juros_anual, anos);
  printf("Valor Final: %.2lf\n", final);

  meses_resgate = resgates_possiveis(final, salario, juros_anual);
  if(meses_resgate == INT_MAX)
    printf("Renda Vitalicia\n");
  else
    printf("Anos de aposentadoria: %.2lf\n", ((double)meses_resgate)/12.0);
}
```

Exemplo 2 - Aposentadoria

- Se considerarmos um salário de R$10.000,00, uma taxa **real** de 3% a.a, e 30 anos de contribuição teremos o montante final de R$1.798.434,93 garantindo 19.76 anos de aposentadoria.
- Neste exemplo considerei aplicações no tesouro direto IPCA+5%. A taxa de juros real de 3% foi obtida descontando-se o IR de 15% sobre os rendimentos totais (incluindo correção da inflação) e descontando também a inflação.

Exercício

- Escreva uma função em C para computar a raiz quadrada de um número positivo. Use a idéia abaixo, baseada no método de aproximações sucessivas de Newton. A função deverá retornar o valor da vigésima aproximação.

Seja Y um número, sua raiz quadrada é raiz da equação

$$f(x) = x^2 - Y.$$

A primeira aproximação é $x_1 = Y/2$. A $(n+1)$-ésima aproximação é

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$