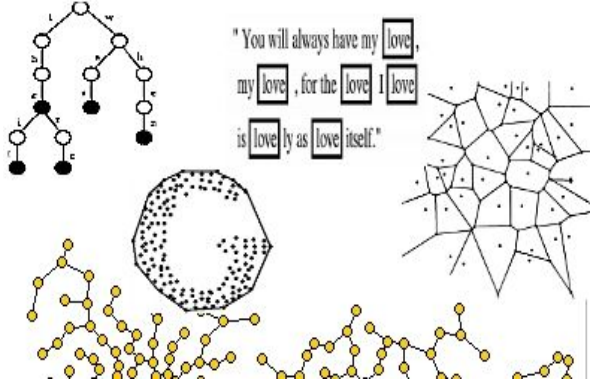


Programação Estruturada
TCC-00347

Prof. Aline Paes / alinepaes@ic.uff.br

Funções

Slides baseados nas aulas do prof. Casanova, PUC-Rio

Universidade Federal Fluminense

Instituto de Computação

Tópicos

- Definição de funções sem ser a *main*
- Pilha de execução
- Variáveis globais e estáticas

Funções

```
#include <stdio.h>

int larger(int a, int b)
{
  if (a > b)
    return a;
  return b;
}

int main()
{
  int greatest = larger(100, 1000);
  printf("%i is the greatest!\n", greatest);
  return 0;
}
```

retorna um valor inteiro

Dois argumentos inteiros

retorno da função

Função está sendo chamada aqui

variável greatest recebe o que a função retorna

Caso não queira retornar nada da função pode usar
void nome_fn () {
...
}
Não precisa ter declaração de return

Comunicação entre funções

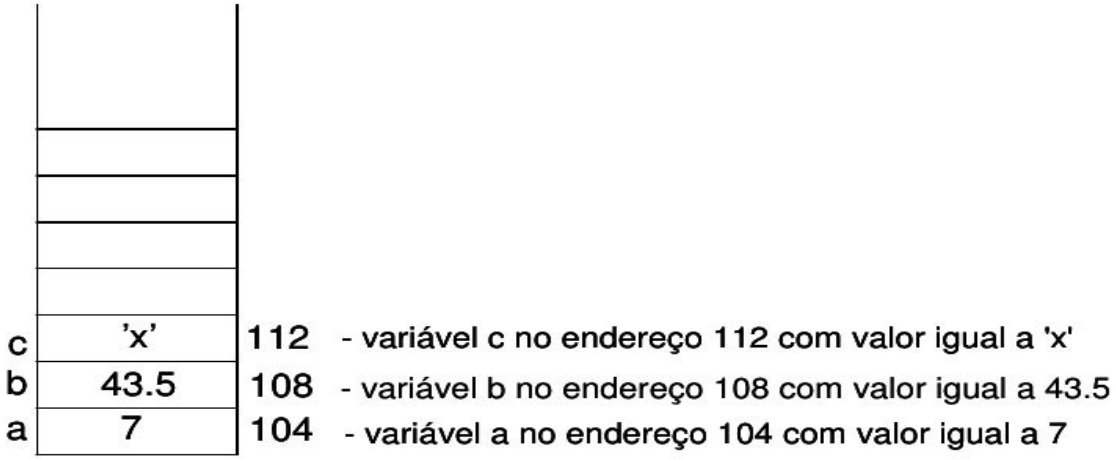
- Funções são independentes entre si
- Variáveis são vistas apenas dentro da função que a declarou
  - são criadas cada vez que a função é executada
  - deixam de ser existir quando a função termina
  - **são variáveis locais**

Comunicação entre funções

- Como transferir os dados entre as funções?
  - através dos parâmetros e do retorno da função chamada
- Passagem de parâmetros é feita por valor
  - são criadas novas variáveis para os parâmetros, inicializadas com o valor corrente na função chamadora

Pilha de execução

- Uma área na memória onde os valores de variáveis são mantidos
- Usadas para inserir o valor de parâmetros e retorno
  - permite a troca de informações entre as funções



* material PUC-Rio

Pilha de execução: exemplo para função fatorial

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
int fat (int n);
int main (void)
{  int n = 5;
   int r;
   r = fat ( n );
   printf("Fatorial de %d = %d \n", n, r);
   return 0;
}
int fat (int n)
{  int f = 1;
   while (n != 0) {
      f *= n;
      n--;
   }
   return f;
}
```

declaração das variáveis *n* e *r*, locais à função *main*

declaração das variáveis *n* e *f*, locais à função *fat*

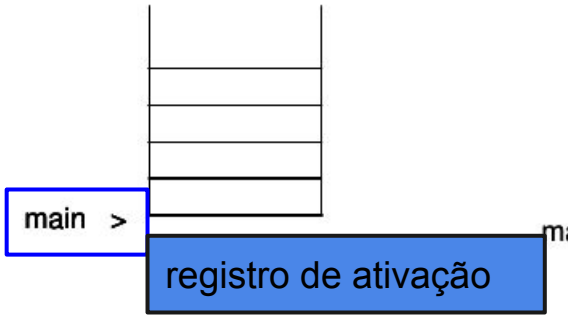
alteração no valor de *n* em *fat* não altera o valor de *n* em *main*

- Variável *n* será 0 ao final da função, mas no programa principal ainda será 5

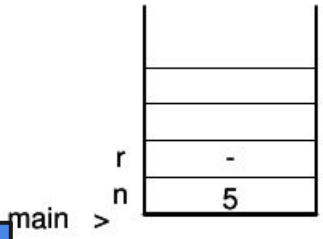
* material PUC-Rio

Comportamento da pilha de execução

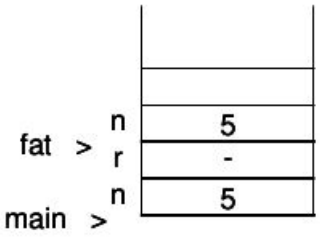
1 - Início do programa: pilha vazia



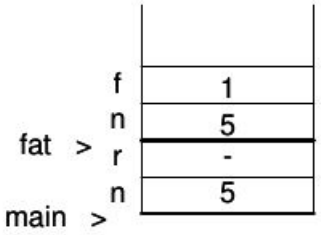
2 - Declaração das variáveis: n, r



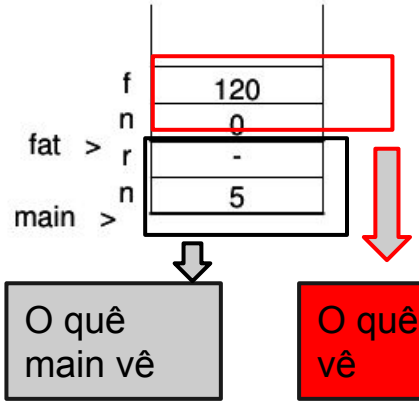
3 – Chamada da função : cópia do parâmetro



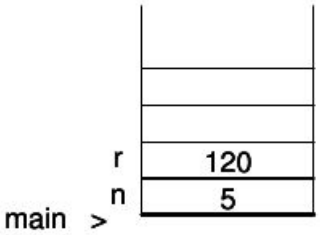
4 - Declaração da variável local: f



5 - Final do laço



6 - Retorno da função: desempilha



Pilha de execução

- Internamente, quando qualquer chamada de função é feita dentro de um programa, é criado um Registro de Ativação na Pilha de Execução do programa

- O registro de ativação armazena os parâmetros e variáveis locais da função bem como o “ponto de retorno” no programa ou função que chamou essa função.

- Ao final da execução dessa função, o registro é desempilhado e a execução volta ao ponto onde a função foi chamada

Variável global

- Declarada fora do corpo das funções
- Visível por todas as funções subseqüentes
  - se uma função modificar o seu valor, a modificação será vista nas demais funções
- não é armazenada na pilha de execução:
  - não deixa de existir quando a execução de uma função termina
  - existe enquanto o programa estiver sendo executado

Variável global

- Utilização de variáveis globais:
  - deve ser feito com critério
  - pode-se criar um alto grau de interdependência entre as funções
  - dificulta o entendimento e o reuso do código

Exemplo - variável global

```
/* Exemplo de variável global */

#include <stdio.h>

int s, p; /* variáveis globais */

void somaprod (int a, int b){
        s = a + b;
        p = a * b;
}
int main (void) {
        int x, y;
        scanf("%d %d", &x, &y);
        somaprod(x,y);
        printf("Soma = %d produto = %d\n", s, p);
        return 0;
}
```

Variável estática

- Declarada no corpo de uma função
  - visível apenas dentro da função em que foi declarada
- não é armazenada na pilha de execução
  - armazenada em uma área de memória estática
  - continua existindo antes ou depois de a função ser executada
- utilização de variáveis estáticas
  - quando for necessário recuperar o valor de uma variável atribuída na última vez que a função foi executada

Variável estática

- Declarada no corpo de uma função
  - visível apenas dentro da função em que foi declarada
- não é armazenada na pilha de execução
  - armazenada em u memória estática
  - continua existin is de a função ser executada
- utilização de v s
  - quando for ne valor de uma variável atribu e a função foi executada

Exemplo: variável estática

```
void imprime ( float a ){

        static int n = 1;
        printf(" %f ", a);
        if ((n % 5) == 0)
                printf(" \n ");
        n++;
}
```

Exercícios

1 - Um número *a* é dito *permutação* de um número *b* se os dígitos de *a* formam uma permutação dos dígitos de *b*. Exemplo: 5412434 é uma permutação de 4321445, mas não é uma permutação de 4312455.
Obs.: Considere que o dígito 0 (zero) não aparece nos números.
(a) Faça uma função *contadígitos* que dados um inteiro *n* e um inteiro *d*, $0 < d \leq 9$, devolve quantas vezes o dígito *d* aparece em *n*.
(b) Usando a função do item anterior, faça um programa que lê dois inteiros positivos *a* e *b* e responde se *a* é permutação de *b*.
(c) Desenhe a pilha de execução para o programa completo

2 - Escreva um programa que informa a soma de uma progressão aritmética, dados os valores inicial, final e o número de termos.

Considere a seguir que não temos o último termo, mas no lugar dele temos a razão.

Exercícios

3 - (a) Construa uma função *sufixo* que, dados dois inteiros positivos *a* e *b,* verifica se *b* corresponde aos últimos dígitos de *a*.

a	b	
567890	890	sufixo
1243	1243	sufixo
2457	245	não é sufixo
457	245	não é sufixo

(b) Usando a função do item anterior, faça um programa que lê dois inteiros positivos *a* e *b* e verifica se o menor deles é segmento do outro.

a	b	
567890	678	b é segmento de a
1243	2212435	a é segmento de b
235	236	não são segmentos

Exercícios

4 - Simule a execução do programa a seguir e desenhe sua pilha de execução, considerando a entrada como 2, 5, 3.0, 2.0.

```
#include <stdio.h>

float f1 (int x, int y) {
    float res;
    if (y != 0)
        res = (float) x / y;
    else
        res = (float) 1 / x;
    while (x > y) {
        res = res + (float) y / x;
        x = x - 1;
    }
    return(res);
}
```

```
int main() {
    int a, b;
    float c, d;

    puts("Digite quatro numeros.\n");
    scanf("%d %d %f %f", &a, &b, &c, &d);
    printf("a = %d  b = %d  c = %f  d = %f\n", a, b, c, d);
    while (a < b) {
        if (c > d) {
            d = f1(b,a);
            b = b - 1;
        }
        else{
            c = 1 / f1(a,b);
            a = a + 1;
        }
        printf("a = %d  b = %d  c = %f  d = %f\n", a, b, c, d);
    }
    return 0;
}
```

Exercícios

5 - Escreva uma função que recebe por parâmetro um valor inteiro e positivo *n* e retorna o valor de S, calculado pela fórmula

S = 1 + 1/1! + 1/2! + 1/3! + 1/N!

6 - Escreva uma função que recebe dois valores *x* e z e calcula e retorna x^z (sem utilizar funções ou operadores de potência prontos).

7 - Implemente uma função que retorne uma aproximação do valor de PI, de acordo com a fórmula de Leibniz abaixo. A função deve receber como entrada o número que indica a quantidade de termos na série. Após, escreva um programa completo que leia o número de termos via teclado e chame a função implementada

$$\pi \approx 4 * \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ldots\right)$$

$$\pi \approx 4 * \sum_{i=0}^{n-1} \frac{(-1)^i}{2 * i + 1}$$

Exercícios

8 - Implemente um programa que leia um inteiro *n* e n inteiros positivos, determinando o maior desses *n* inteiros.

9 - Escreva um programa que leia caracteres da entrada padrão e escreva para a saída padrão os caracteres lidos de código compreendido entre 32 e 127 (extremos incluídos), passando para maiúsculas as letras minúsculas. Por exemplo, a entrada As armas e os 7 barões, deve virar AS ARMAS E OS 7 BARES. Use apenas a função getchar().