PROGRAMAÇÃO DE COMPUTADORES V - TCC- 00.323

Modulo 6 : Funções

Escopo de Variáveis: Globais x Locais

Aura - Erick
aconci@ic.uff.br, erickr@id.uff.br

Roteiro

- Funções
- Escopo de Variáveis
- Variáveis Globais x Variáveis Locais
- **Recursividade**

Motivação

- Trechos de código podem ser repetir ao longo de um programa
- Reuso de código

Exemplo 1

```
main() {
  float A,B,C;
  scanf("%f %f",&A, &B);
  C = (A * B)/2;
  printf("%f \n", C);
  A = A/2;
  B = B/2;
  C = (A * B)/2;
  printf("%f \n", C);
  system("pause");
}
```

Trecho se repete 2 vezes

Exemplo 2

1. Leia vetor um vetor A de 10 posições de inteiros
2. **Ordene o vetor A**
3. Leia um vetor B de 10 posições de inteiros
4. **Ordene o vetor B**
5. Multiplique o vetor A pelo vetor B, e coloque o resultado num vetor C
6. **Ordene o vetor C**

Operação de ordenação do vetor é repetida 3 vezes

Problemas desta “repetição”

- Programa muito grande, porque tem várias “partes repetidas”
- Erros ficam difíceis de corrigir (e se eu esquecer de corrigir o erro em uma das N repetições daquele trecho de código?)
- Pequenas mudanças precisam ser refeitas em todos os trechos!

Solução: subprogramação

- Definir o trecho de código que se repete como uma "função" que é chamada no programa
- A função é definida uma única vez, e chamada várias vezes dentro do programa

Voltando ao Exemplo 1

```
#include <stdio.h>

float calculaC(float A,float B) {
  float C;
  C = (A * B)/2;
  printf("%f \n", C);
  return C;
}

main() {
  float A,B,C;
  scanf("%f %f", &A, &B);
  C = calculaC(A,B);
  A = A/2;
  B = B/2;
  C = calculaC(A,B);
  system("pause");
}
```

Definição da função

Chamada da função

Chamada da função

Fluxo de Execução

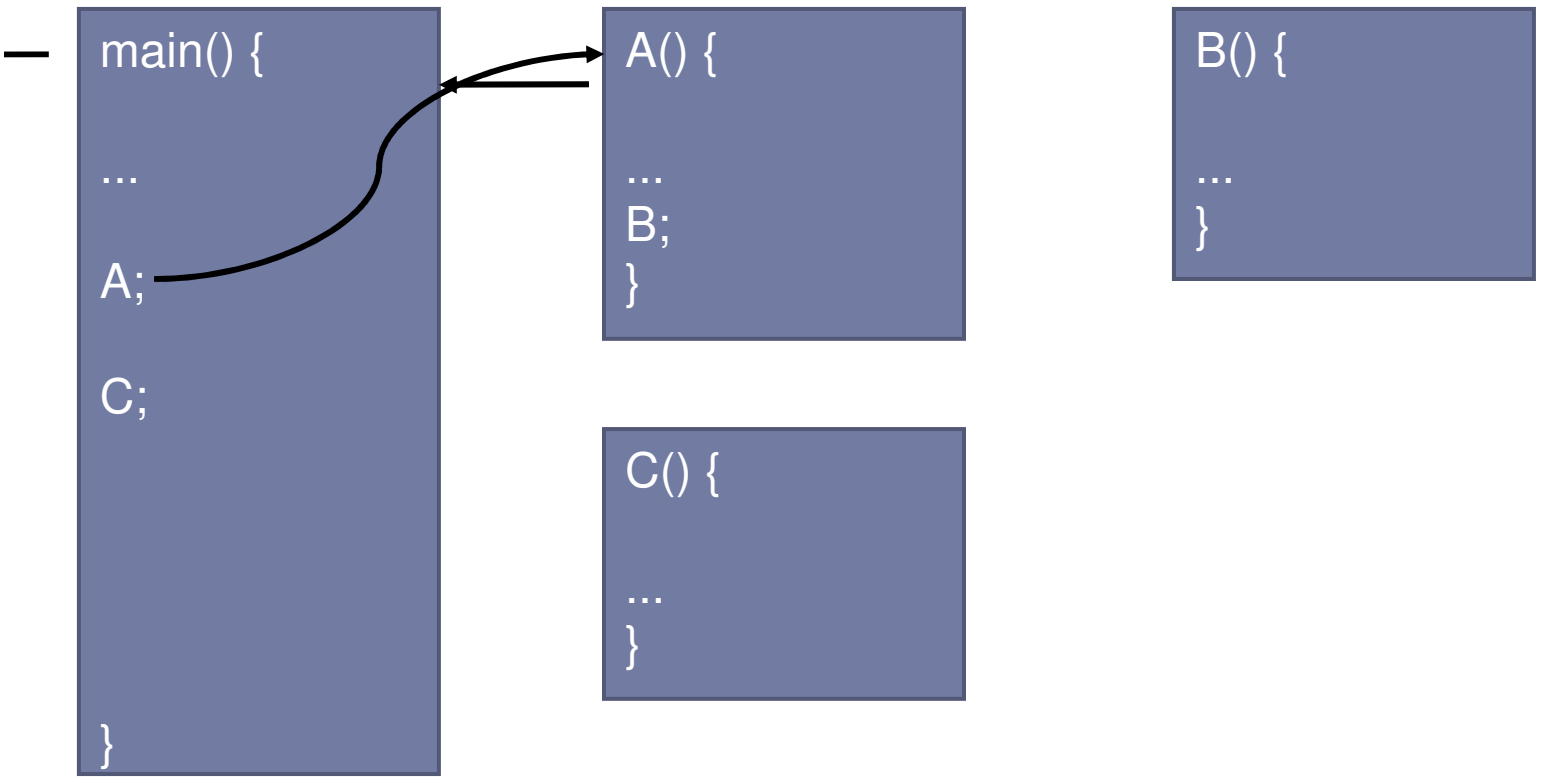
```
main() {

...

A;

C;



}
```

```
A() {

...
B;
}
```

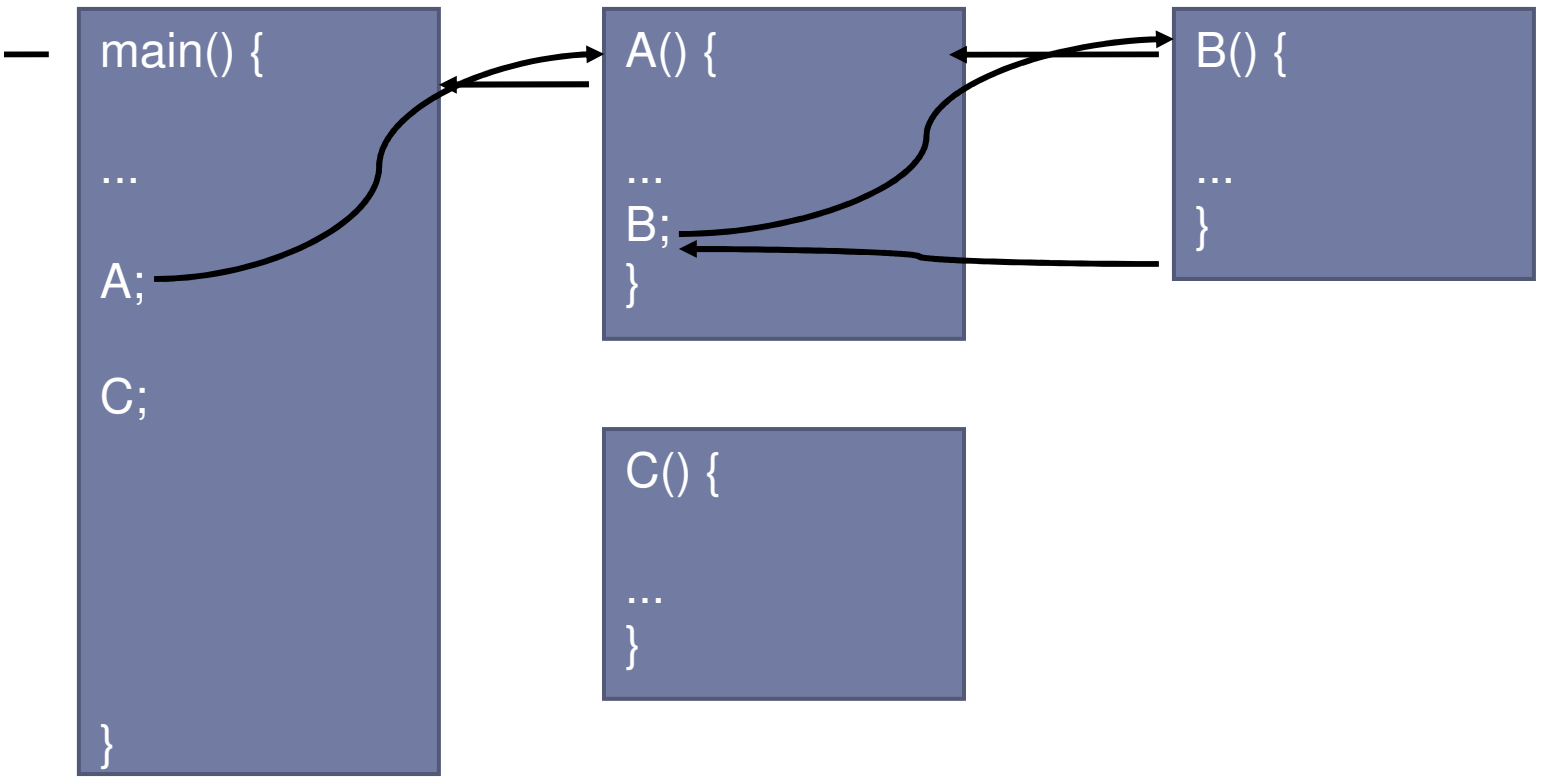
```
B() {

...
}
```

```
C() {

...
}
```

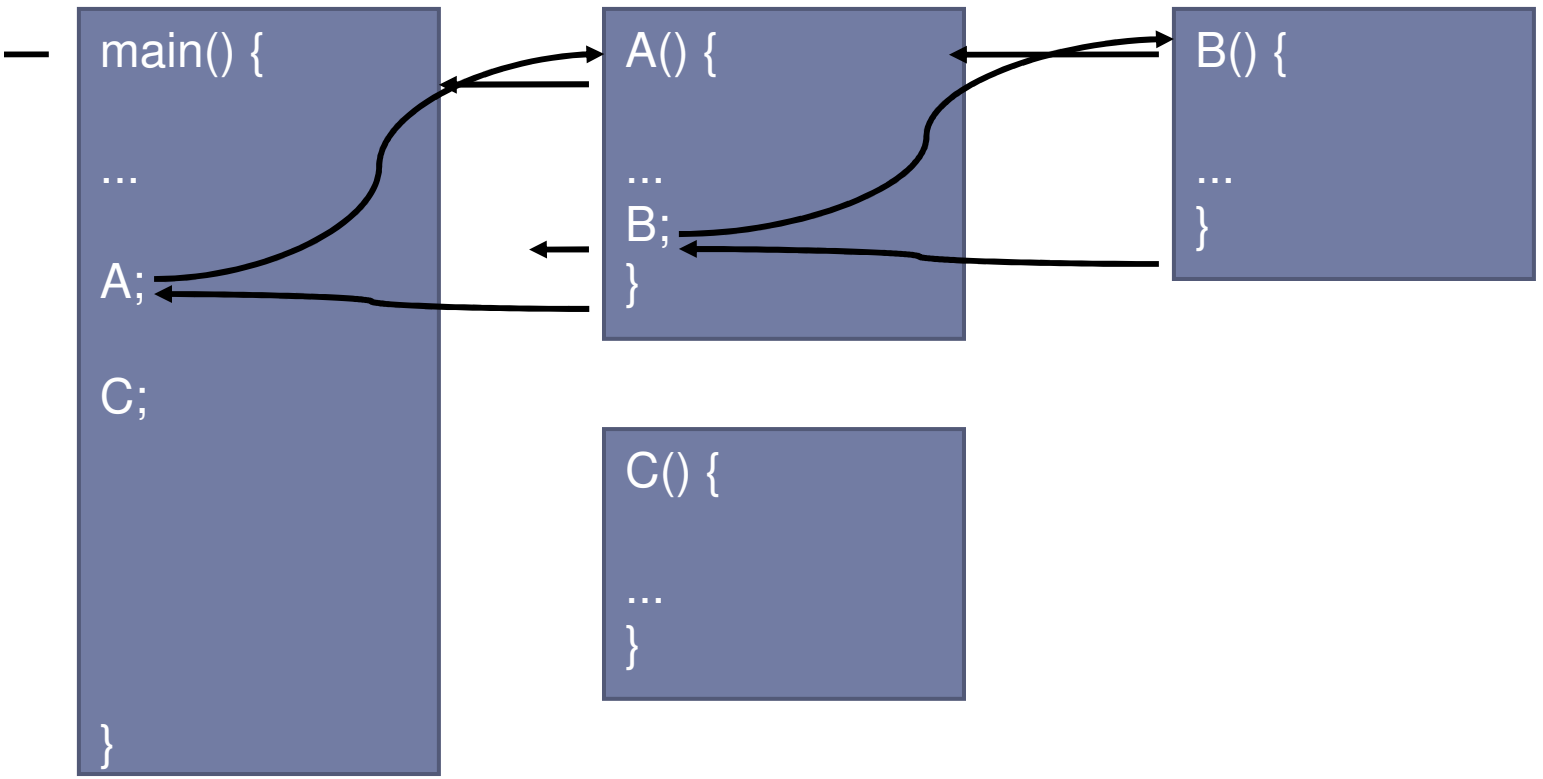
Fluxo de Execução



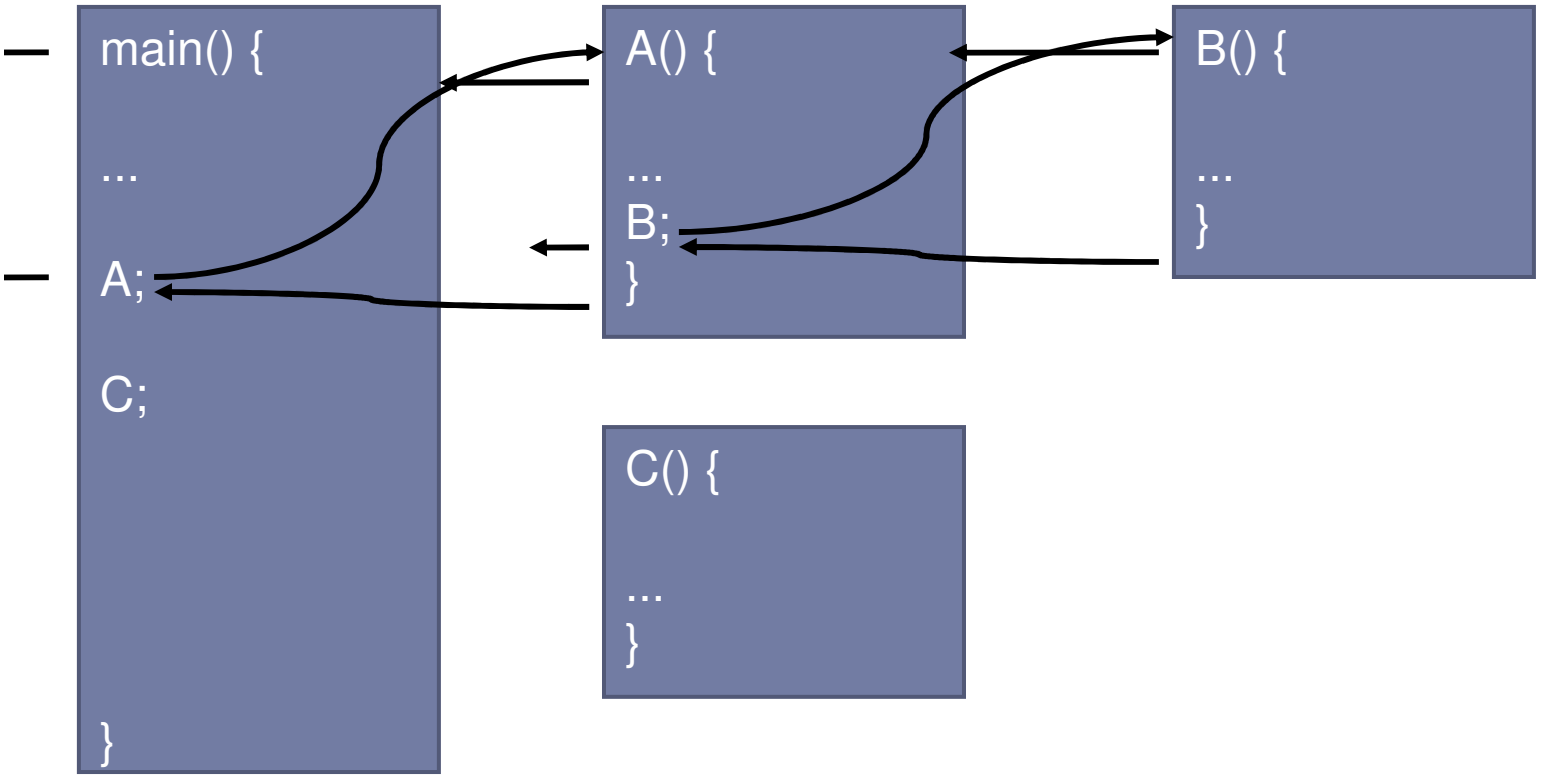
Fluxo de Execução



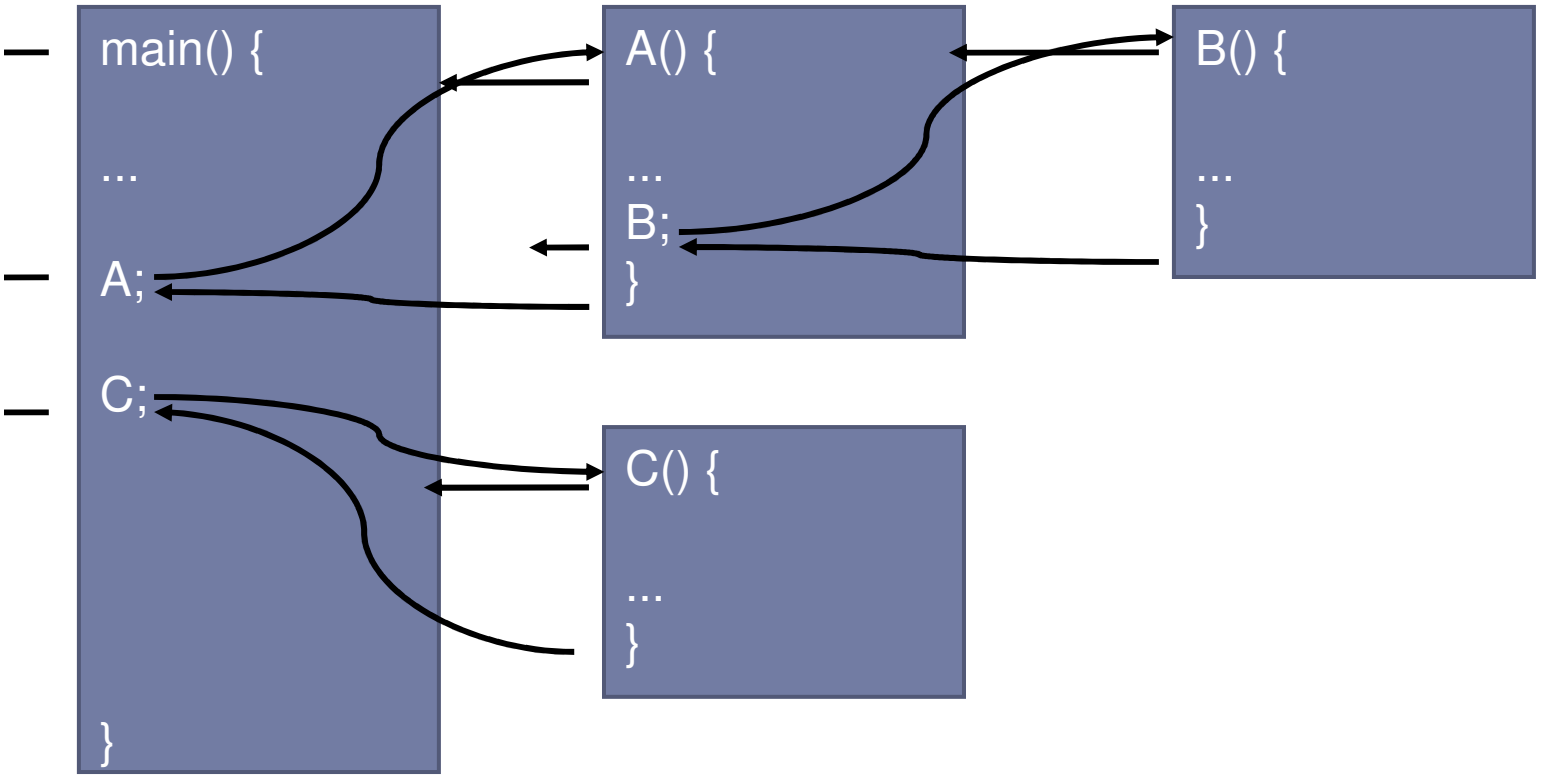
Fluxo de Execução



Fluxo de Execução



Fluxo de Execução



main()

- O main() também é uma função.
  - É a primeira função a ser executada num programa em C.
  - A partir dela outras funções podem ser chamadas

Elementos de uma função

- Declaração
- Variáveis
- Código
  - Retorno

Declaração de Funções

```
tipoRetorno nomeFuncao (tipo parametro; tipo
   parametro, ..., tipo parametro) {
variáveis locais
código
[retorno]
}
```

```
float calculaC (float A, float B) {
  float C;
  C = (A * B)/2;
  printf("%f \n", C);
  return C;
}
```

Variáveis

- Podem ser locais ou globais

- Variáveis locais
  - Declaradas dentro de uma função
  - São visíveis somente dentro da função onde foram declaradas
  - Passam a existir no início da execução da função
  - São destruídas ao término da execução da função

- Variáveis globais
  - Declaradas fora de todas as funções (inclusive do main()!)
  - São visíveis por TODAS as funções do programa

Uso de Variáveis Globais x Variáveis Locais

- Cuidado com variáveis globais
  - Dificultam o entendimento do programa
  - Dificultam a correção de erros no programa
    - Se a variável pode ser modificada por qualquer função do programa, encontrar um erro envolvendo o valor desta variável pode ser muito complexo
- Recomendação
  - Sempre que possível, usar variáveis LOCAIS

Variáveis

```
#include <stdio.h>
float X;

float calculaC(float D,float E){
  float F;
  F = (D * E)/2;
  printf("%f \n", F);
  return F;
}

main() {
  float A,B,C;
  scanf("%f %f",&A, &B);
  C = calculaC(A,B);
  A = A/2;
  B = B/2;
  C = calculaC(A,B);
  X = A;
  system("pause");
}
```

Variável Global

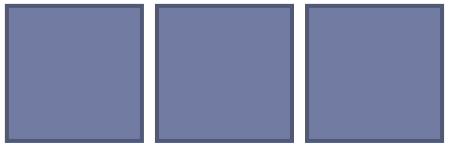
Parâmetros da Função

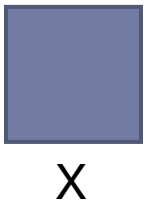
Variável Local

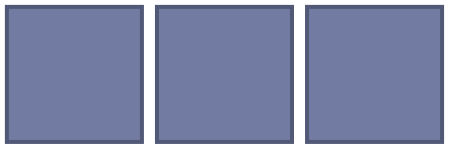
Variáveis Locais

Escopo de Variáveis

```
#include <stdio.h>
float X;

float calculaC(float D, float E) {
  float F;
  F = (D * E)/2;
  printf("%f \n", F);
  return F;
}

main() {
  float A,B,C;
  scanf("%f %f",&A, &B);
  C = calculaC(A,B);
  A = A/2;
  B = B/2;
  C = calculaC(A,B);
  X = A;
  system("pause");
}
```





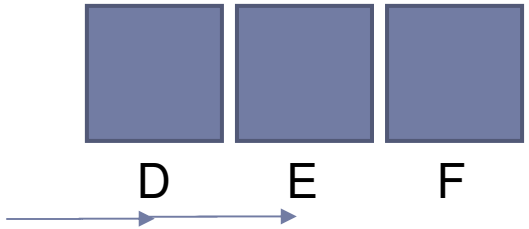


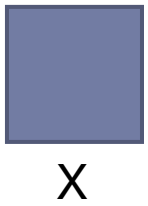
Parâmetros

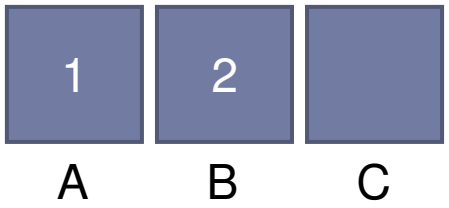
- Quando uma função é chamada, é necessário fornecer um valor para cada um de seus parâmetros
- Isso por ser feito informando o valor diretamente
  - C = calculaC(1,2);
- ou; Usando o valor de uma variável
  - C = calculaC(A,B);

Passagem de Parâmetro

```
#include <stdio.h>
float X;

float calculaC(float D,float E) {
  float F;
  F = (D * E)/2;
  printf("%f \n", F);
  return F;
}

main() {
  float A,B,C;
  scanf("%f %f",&A, &B);
  C = calculaC(A,B);
  A = A/2;
  B = B/2;
  C = calculaC(A,B);
  X = A;
  system("pause");
}
```

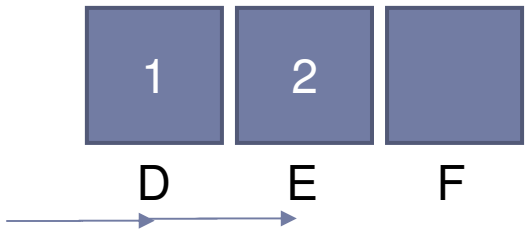


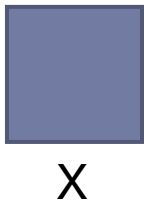


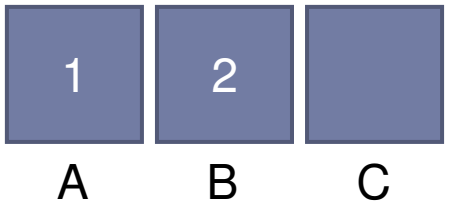


Passagem de Parâmetro

```
#include <stdio.h>
float X;

float calculaC(float D, float E) {
  float F;
  F = (D * E)/2;
  printf("%f \n", F);
  return F;
}

main() {
  float A,B,C;
  scanf("%f %f",&A, &B);
  C = calculaC(A, B);
  A = A/2;
  B = B/2;
  C = calculaC(A, B);
  X = A;
  system("pause");
}
```







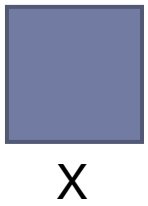
Tipos de passagem de Parâmetro

- Por valor: o valor da variável na chamada é copiado para a variável da função. Alterações não são refletidas na variável original

- Por referência: é como se o mesmo “escaninho” fosse usado. Alterações são refletidas na variável original

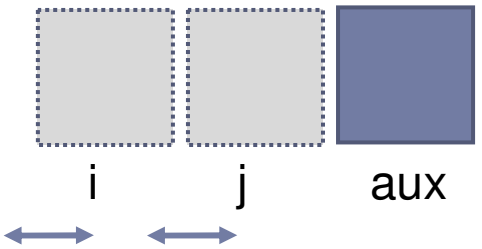
Passagem de Parâmetro por Valor

```
#include <stdio.h>
float X;

float calculaC(float D, float E) {
  float F;
  F = (D * E)/2;
  printf("%f\n", F);
  return F;
}

main() {
  float A,B,C;
  scanf("%f %f",&A, &B);
  C = calculaC(A,B);
  A = A/2;
  B = B/2;
  C = calculaC(A,B);
  X = A;
  system("pause");
}
```

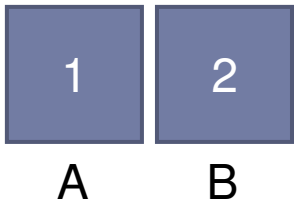


Passagem de Parâmetro por Referência

```
#include <stdio.h>

/* troca o conteúdo de 2 variáveis inteiras
   recebe i, j; devolve i, j com valores trocados */
void troca(int *i, int *j) {
      int aux;
      aux = *i;
      *i = *j;
      *j = aux;
}
```




```
void main () {
    int A,B;
     scanf("%d %d", &A, &B);
     printf("%d %d \n", A, B);
      troca(&A, &B);
     printf("%d %d \n", A ,B);
     A = A + 2;
     troca(&A, &B);
     printf("%d %d\n", A, B);
     system("pause");
}
```



Retorno das funções

- Função que retorna um valor deve usar **return**
  - Assim que o comando return é executado, a função termina
  - O valor retornado deve ser do mesmo tipo declarado para a função
- Função que não retorna valor deve ser declarada como tipo **void**
- Quando não se especifica o tipo de retorno de uma função, o compilador assume que o tipo de retorno é **int**

Exemplo – retorno float

```
float calculaC(float D, float E) {
  float F;
  F = (D * E) /2;
  printf("%f \n", F);
  return F;
}
```

Exemplo – retorno void

```
void imprimeMenu() {
  printf("Escolha uma opcao \n");
  printf("1 - Somar“);
  printf(“2 - Subtrair“);
  printf(“3 - Sair do Programa“);
}
```

Neste caso, o **return** não é usado

Chamada de função

- Se a função retorna um valor, pode-se atribuir seu resultado a uma variável

```
C = calculaC(A,B);
```

- Se a função não retorna um valor (é **void**), não se pode atribuir seu resultado a uma variável

```
imprimeMenu();
```

Dica para onde se usar textos na tela

- Como limpar a tela?

- A tela de comando tem 25 linhas por 80 colunas
- Cada compilador C tem uma função específica para limpar a tela (não é padrão). Para evitar problemas de compatibilidade entre compiladores, o modo mais seguro é usar um **for** dentro de outro:

```
for ( i = 1; i < 26 ; i++ ) {
     for ( j = 1 ; j < 81 ; j++ )
        printf(" ");
    printf(“ \n ");
    }
```

Exercícios

- Fazer uma nova solução sua para cada um dos exercícios da aula, pois existem diversas formas de se chegar ao mesmo resultado em programação!

Recursividade

Uma função pode chamar a si própria.

Uma função que faz assim é chamada **função recursiva.**

Há várias operações matemáticas recursivas, um exemplo bem conhecido é o **fatorial.**

Cálculo do fatorial de um número,

Fatorial de um número *n* é definido como o produto de todos os números naturais (não nulos) menores ou iguais a ***n***

Por exemplo, **5!** (lê-se "cinco fatorial") é igual a 5 x 4 x 3 x 2 x 1.

Por convenção **0!** = 1.

Uma maneira de definir o algoritmo de fatorial é:

$$n! = \begin{cases} 1, \text{ se } n = 0 \text{ ou } n = 1 \\ n\,(n-1)! \text{ se } n \geq 2 \end{cases}$$

E a implementação correspondente seria:

```
#include <stdio.h>
#include <stdlib.h>

int fat(int n)
{
   if (n)
       return n * fat(n-1);
   else return 1;
}

int main()
{
    int n;
    printf("\n \n Digite um valor para n: ");
    scanf("%d", &n);
    printf("\n  O fatorial de %d e' %d", n, fat(n) );
    return 0;
}
```

n ser verdade é não ser Zero!!

Exercícios

Usando recursividade de funções calcule as seqüências de Fibonacci , F(n), e de Lucas , U(n) .

O matemático Leonardo Pisa, conhecido como *Fibonacci*, propôs no século XIII, a seqüência numérica abaixo:
(1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...)

Essa seqüência tem uma **lei de formação** simples: **cada elemento**, a partir do terceiro, **é obtido somando-se os dois anteriores**.

Veja: 1+1=2, 2+1=3, 3+2=5 e assim por diante.

A seqüência de Fibonacci tem aplicações na análise de mercados financeiros, na ciência da computação e na teoria dos jogos. Também aparece em **configurações biológicas**, como, por exemplo, na disposição dos galhos das árvores ou das folhas em uma haste, no arranjo do cone da alcachofra, do abacaxi, ou no crescimento da samambaia.

Uma **generalização** da seqüência de Fibonacci são as **seqüências de Lucas, U(n)** .

$$U(0) = 0$$
$$U(1) = 1$$
$$U(n+2) = \boldsymbol{P}\, U(n+1) - \boldsymbol{Q}\, U(n)$$

onde ***P*** e ***Q*** são constantes.

Por exemplo: a seqüência normal de *Fibonacci* é o caso especial de ***P*** = 1 e ***Q*** = -1.

Outra ***seqüência de Lucas*** pode começa com V(0) = 2, V(1) = P.

Essas sequências têm aplicações na Teoria de Números e na prova que um dado número é primo.

Referências

- https://pt.wikibooks.org/wiki/Programar_em_C
- https://pt.wikibooks.org/wiki/Programar_em_C/Fun%C3%A7%C3%B5es
- https://pt.wikipedia.org/wiki/Sistema_octal

https://www.youtube.com/watch?v=Y19q6rgM9eo

http://www.inf.pucrs.br/~pinho/LaproI/Funcoes/AulaDeFuncoes.htm