

Profa. Michele Fúlvia Angelo

**PGCA 028 – Tópicos Especiais em Tecnologia  
Computacional I – Introdução à Programação de  
Computadores  
Aula 9**

Universidade Estadual de Feira de Santana

# Sumário

- Funções e Procedimentos

# Novas necessidades...

- Programas cada vez maiores
- Aumento da complexidade
- Repetição de trechos de código
- Como atacar um problema de cada vez?
  - “dividir-para-conquistar”: dividir problemas em subproblemas menores e mais tratáveis, e se ainda for muito complexa, dividi-se em subproblemas ainda menores e assim sucessivamente.
- Como evitar repetições?
  - Modularização

# Modularização

- Simplificar a codificação
- Evitar a repetição do mesmo trecho de código
- Reaproveitar código existente
- Facilitar modificações
- Organizar o programas em partes que podem ser compreendidas isoladamente
- Melhorar a estruturação do programa
- Facilitar o entendimento do programa

# Exemplo

- Números primos entre 1 e N
  - Solicite o valor de N
  - Para cada número entre 1 e N
    - Teste se o número é primo
    - Se for primo, mostre o número

```
int main() {  
    int N, num;  
    scanf("%d", &N);  
    for(num=1; num<=N; num++){  
        if( ehprimo(num)==1 ){  
            printf("%d", num );  
        }  
    }  
}
```

Mas e este comando:  
**ehprimo(num)?**

Um sub-programa que  
precisa ser definido!

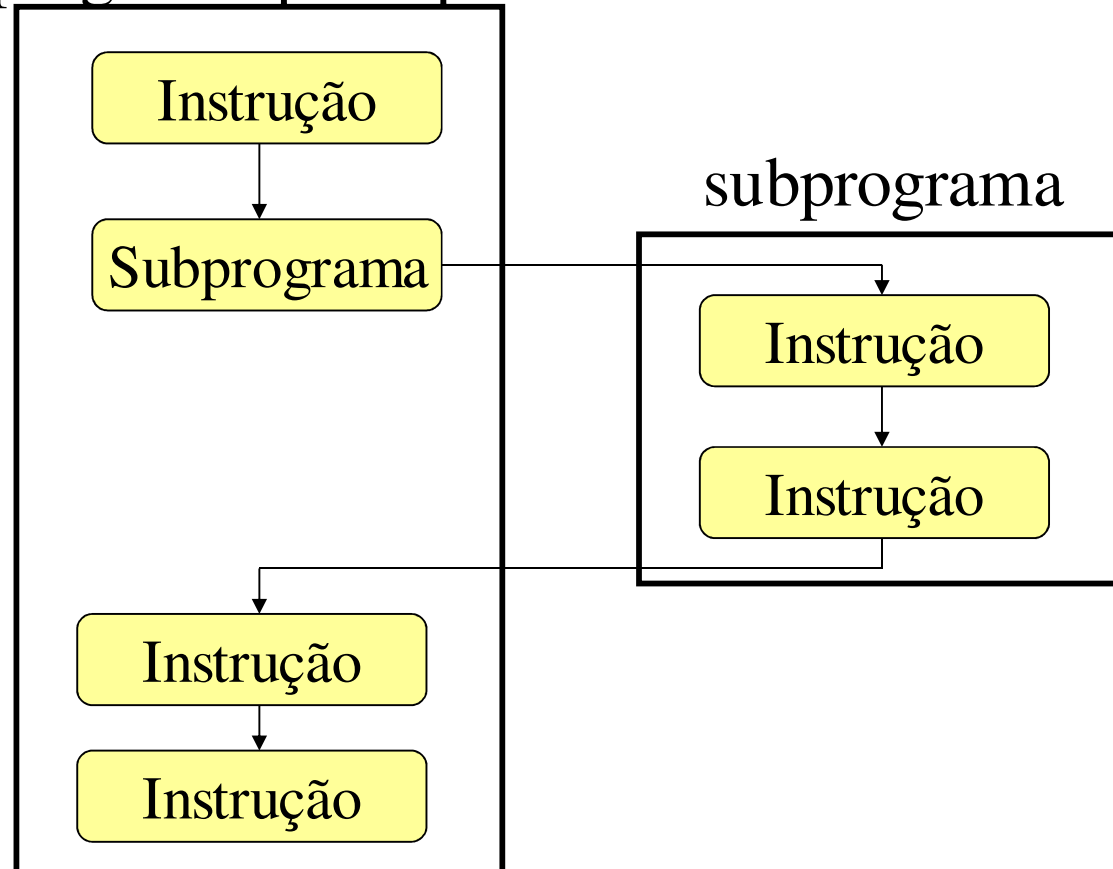
# Subprogramas

- Um subprograma não pode ser executado diretamente.
- Ele precisa ser "invocado" ou "chamado" pelo programa principal.
- Esta chamada desvia o fluxo de controle para o subprograma.
- Um subprograma pode invocar outro subprograma e assim sucessivamente.
- Em C todos os subprogramas estão no mesmo nível, por isso não é possível definir um subprograma dentro de outro subprograma .
- Podem ser parametrizados, auxiliando o reaproveitamento.



# Subprogramas

- Desvio do fluxo de controle

programa principal



# Função

- Função matemática:
  - `sqrt(x)` => raiz quadrada
  - Uso: `y=sqrt(x);`
- Já utilizamos funções no C:
  - `tam=strlen(str);`
- Valor(es) de entrada  Função  Valor Saída



# Função

- Declaração e definição:

```
tipo_saída nome_funcao (parametros)  
{  
  
    /*variáveis*/  
    comandos;  
  
    return resultado;  
}
```

- Chamada:

```
nome_funcao (parametros)
```

# Função

- Lista de parâmetros
  - Após o nome da função e deve estar entre parênteses
  - Cada parâmetro (ou argumento) precedido por seu tipo
  - Mais de um parâmetro: separar por vírgula
- Saída (Retorno)
  - Tipo do valor retornado especificado na definição
  - Não indicado, assume-se retorno do tipo inteiro
  - O comando *return* indica o valor retornado pela função
  - Este valor pode ser representado por uma constante, uma variável ou por expressões, respeitando o tipo
  - Uma função encerra assim que encontra o comando *return*
  - *return* sem valor faz a função finalizar

# Exemplo

- Se fosse um programa:

```
int main(){  
    int fat;  
    int x;  
  
    scanf("%d",&x);  
    fat=fatorial(x);  
    printf("%d",fat);  
}
```

# Exemplo

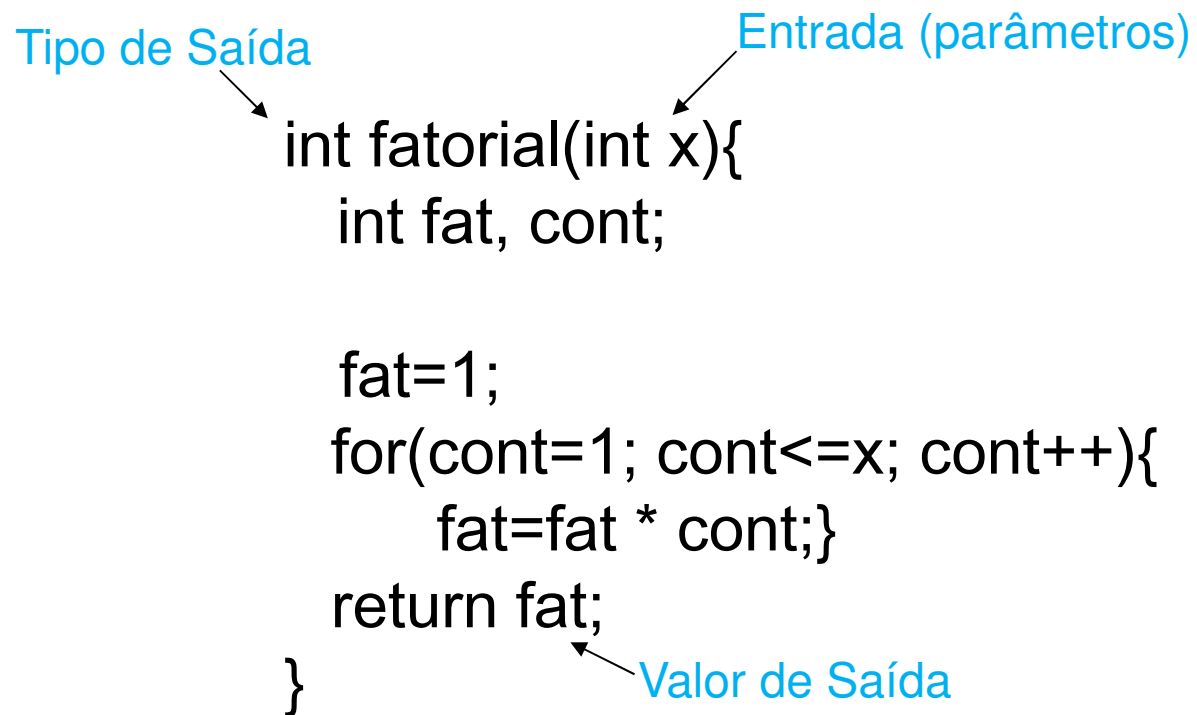
- Para uma função:

Tipo de Saída

Entrada (parâmetros)

```
int fatorial(int x){  
    int fat, cont;  
  
    fat=1;  
    for(cont=1; cont<=x; cont++){  
        fat=fat * cont;}  
    return fat;  
}
```

Valor de Saída



# Exemplo

- Verifica se um número é primo

```
int ehprimo(int numero){  
    int pr, dv;  
  
    pr = 1;  
    for(dv = 2; dv < numero; dv++){  
        if ((numero % dv) == 0){  
            pr = 0;  
        }  
    }  
    return pr;  
}
```

# Exemplo

```
int ehprimo(int numero){  
    int pr, dv;
```

declaração e definição

```
    pr = 1;  
    for(dv = 2; dv < numero; dv++){  
        if ((numero % dv) == 0){  
            pr = 0;  
        }  
    }  
    return pr;  
}
```

```
int main() {  
    int N, num;  
    scanf("%d", &N );  
    for(num=1; num<=N; num++){  
        if( ehprimo( num )==1 ){  
            printf("%d", num );  
        }  
    }  
}
```

chamada

# Função

- Protótipo de função

- Trecho de código que permite declarar uma função antes de defini-la, já que na linguagem C toda a função precisa ser declarada antes de ser chamada por outras funções
- Útil quando os programadores querem que a função `main()` esteja no início do arquivo fonte

```
tipo_saída nome_funcao(parametros);
```

# Exemplo

`int ehprimo(int numero);` ← declaração

```
int main() {  
    int N, num;  
    scanf("%d", &N );  
    for(num=1; num<=N; num++){  
        if( ehprimo( num )==1 ){ ← chamada  
            printf("%d", num );  
        }  
    }  
}
```

```
int ehprimo(int numero){  
    int pr, dv; ← definição  
  
    pr = 1;  
    for(dv = 2; dv < numero; dv++){  
        if ((numero % dv) == 0){  
            pr = 0;  
        }  
    }  
    return pr;  
}
```



# Função

- Função main ()
  - É a função principal de um programa em linguagem C
  - Um programa começa a ser executado do início da função main() e termina quando a última linha é executada ou quando o comando *return* é chamado
  - O tipo padrão da função main() é int

# Escopo de Variáveis

- As variáveis podem ser declaradas basicamente em três lugares:
  - Dentro de funções ou blocos de comando
  - Fora de todas as funções
  - Na lista de parâmetros das funções.
- As variáveis dentro das funções ou blocos de comandos são chamadas de variáveis locais.
- As que aparecem fora de todas as funções chamamos de variáveis globais.
- Aquelas que aparecem na lista de parâmetros são os parâmetros formais.

# Escopo de Variáveis

- Parâmetros são locais.
- Variáveis/parâmetros das funções não retêm seus valores de uma chamada para outra.
- Variáveis do programa principal podem ser acessadas dentro das funções se passadas como parâmetros.

# Variáveis Locais

- Elas passam a existir quando do início da execução do bloco de comandos ou função, onde foram definidas, e são destruídas ao final da execução do bloco (tempo de vida).
- Uma variável local só pode ser referenciada, ou seja usada, dentro das funções ou blocos onde foram declaradas.
- Como as variáveis locais deixam de existir ao final da execução da função ou bloco, elas são invisíveis para outras funções ou blocos do mesmo programa.

# Variáveis Locais

- Observe que um bloco de comandos se inicia em um "{" e termina em um "}".
- O bloco de comandos mais usado para definir uma variável é a função.
- Todas as variáveis que serão usadas dentro de um bloco de comandos precisam ser declaradas antes do primeiro comando do bloco.
- Declarações de variáveis, incluindo sua inicialização, podem vir logo após o abre chaves que inicia um bloco de comandos, não somente o que começa uma função.

```
#include<stdio.h>
void main(){
    int i;
    for (i=0; i<10; i++){
        int t;
        scanf("%d", &t);
        printf("%d\n", i*t);
    }
}
```

- Existem algumas vantagens em se declarar variáveis dentro de blocos.
  - Como as variáveis somente passam a existir quando o bloco passa a ser executado, o programa ocupa menos espaço de memória. Isto porque se a execução do bloco for condicional, a variável pode nem ser alocada.
  - Como a variável somente existe dentro do bloco pode-se controlar melhor o uso da variável, evitando erros de uso indevido da variável.

# Variáveis Locais

```
#define N 5
```

```
void le_vetor(int v[],int n);  
void altera_vetor(int vh[], int n);  
void mostra_vetor(int v[],int n);
```

```
int main ()  
{  
    int lim=5;  
    int vet1[N];
```

```
    le_vetor(vet1,lim);  
    mostra_vetor(vet1,lim);
```

```
    getch();  
}
```

```
void le_vetor(int v[],int n){  
    int i;  
    printf("Digite os elementos do vetor\n");  
    for(i=0;i<n;i++){  
        scanf("%d",&v[i]);  
        int k;  
        k=v[i];  
    }  
    altera_vetor(v,n);
```

```
}
```

```
void altera_vetor(int vh[],int n){  
    int i;  
    for(i=0;i<n;i++){  
        vh[i]=vh[i]*2;  
    }  
}
```

```
void mostra_vetor(int v[],int n){  
    int i;  
    printf("Os elementos do vetor alterados sao\n");  
    for(i=0;i<n;i++){  
        printf("%d ",v[i]);  
    }  
}
```

# Variáveis Globais

- As variáveis globais são definidas fora de qualquer função e são portanto disponíveis para qualquer função.
- Este tipo de variável pode servir como uma canal de comunicação entre funções, uma maneira de transferir valores entre elas.
- Por exemplo, se duas funções tem de partilhar dados mais uma não chama a outra, uma variável global tem de ser usada.

# Parâmetros Formais

- As variáveis que aparecem na lista de parâmetros da função são chamadas de parâmetros formais da função.
- Eles são criados no início da execução da função e destruídos no final.
- Parâmetros são valores que as funções recebem da função que a chamou. Portanto, os parâmetros permitem que uma função passe valores para outra.



# Passagem de Parâmetros

```
float calcula_media(int a, int b, int c){  
    float media;  
    media = (a+b+c) / 3;  
    return media;  
}
```

...

```
int main(){  
    ...  
    m = calcula_media(10,20,30);  
}
```

- a,b,c assumem os valores 10, 20 e 30.

# Passagem de Parâmetros

```
float calcula_media(int a, int b, int c){  
    float media;  
    media = (a+b+c) / 3;  
    return media;  
}
```

```
...  
int main(){  
    ...  
    x=10; y=20; z=30;  
    m = calcula_media(x,y,z);  
}
```

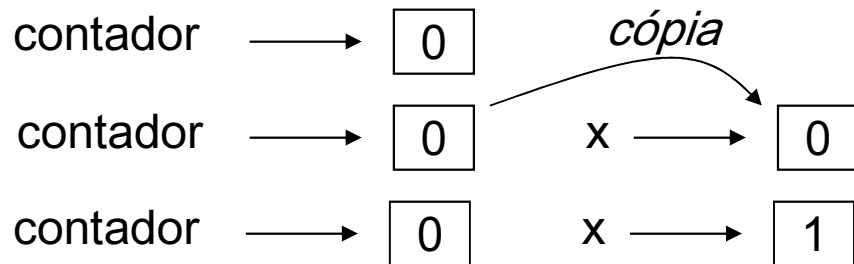
- a,b,c assumem cópia dos valores de x, y e z.
  - estão em lugares diferentes da memória
  - alterações em a,b e c não alteram x, y, z

# Passagem de Parâmetros

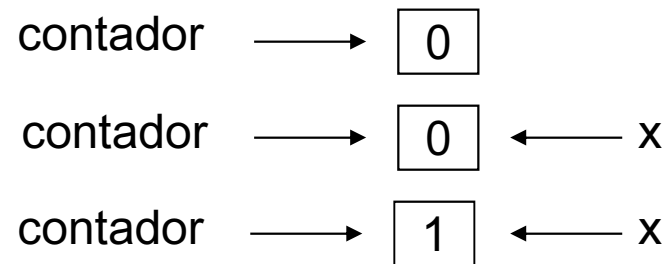
- Passagem por valor
  - Passagem do conteúdo da variável
  - Parâmetro do mesmo tipo da variável
  - Independência entre variáveis e parâmetros
  - Valores das variáveis são copiados para os parâmetros
  - Mudanças em parâmetros não afetam as variáveis
- Passagem por referência
  - Passagem do endereço da variável
  - Parâmetro do tipo apontador
  - Dependência entre variáveis e parâmetros
  - Parâmetros apontam para o conteúdo das variáveis
  - Mudanças em parâmetros refletem nas variáveis

# Passagem de Parâmetros

```
void incVal (int x){  
    x=x+1;  
}  
int main(){  
    int contador=0;  
    incVal(contador);  
}
```



```
void incRef (int *x){  
    *x=*x+1;  
}  
int main(){  
    int contador=0;  
    incRef(&contador);  
}
```



# Passagem de Parâmetros

```
int quadrado(int x){  
    return (x*x);  
}  
int main(){  
    int num=10;  
    num=quadrado(num);  
}
```

```
void quadrado(int *x){  
    *x=(*x)*(*x);  
}  
int main(){  
    int num=10;  
    quadrado(&num);  
}
```

# Passagem de Parâmetros

- Vetores como Parâmetros
  - Vetores também podem ser passados como parâmetros para funções, mas são sempre passados por referência
  - Implicação: ao alterar valores dos elementos de um vetor passado como parâmetro, as alterações são “percebidas” no escopo da função chamadora

```
void mostra_vetor (int v[LIM], int n);
```

equivale a

```
void mostra_vetor (int v[], int n);
```

equivale a

```
void mostra_vetor (int *v, int n);
```

# Passagem de Parâmetros

```
#define N 5
```

```
void le_vetor(int v[N],int n);  
void altera_vetor(int v[], int n);  
void mostra_vetor(int *v,int n);
```

```
int main ()  
{  
    int lim=5;  
    int vet1[N];  
  
    le_vetor(vet1,lim);  
    altera_vetor(vet1,lim);  
    mostra_vetor(vet1,lim);  
}
```

```
void le_vetor(int v[N],int n){  
    int i;  
    printf("Digite os elementos do vetor\n");  
    for(i=0;i<n;i++)  
        scanf("%d",&v[i]);  
}
```

```
void altera_vetor(int v[],int n){  
    int i;  
    for(i=0;i<n;i++)  
        v[i]=v[i]*2;  
}
```

```
void mostra_vetor(int *v,int n){  
    int i;  
    printf("Os elementos do vetor alterados  
sao\n");  
    for(i=0;i<n;i++)  
        printf("%d",v[i]);  
}
```

# Passagem de Parâmetros

- Matrizes como Parâmetros
  - Matrizes também podem ser passadas como parâmetros para funções, mas como os vetores, são sempre passados por referência

```
void mostra_matriz (int m[LIM1][LIM2], int linhas);
```

equivale a

```
void mostra_matriz (int m[][LIM2], int linhas);
```



# Passagem de Parâmetros

```
#define M 2
#define N 3

void le_matriz(int matriz[M][N], int linhas){
    int i, j;
    for (i = 0; i < linhas; i++)
        for (j = 0; j < N; j++)
            scanf("%d",&matriz[i][j]);
}

void altera_matriz(int matriz[][N], int linhas){
    int i, j;
    for (i = 0; i < linhas; i++)
        for (j = 0; j < N; j++)
            matriz[i][j]=matriz[i][j]*2;
}
```

```
void exibe_matriz(int matriz[][N], int linhas){
    int i, j;
    for (i = 0; i < linhas; i++)
        for (j = 0; j < N; j++)
            printf("matriz[%d][%d] = %d\n", i, j,
                matriz[i][j]);
}

int main(){

    int mat1[M][N];

    le_matriz(mat1, M);
    altera_matriz(mat1, M);
    exibe_matriz(mat1, M);

}
```

# Passagem de Parâmetros

```
void exhibe_vetor(int valores[], int num_de_elementos){  
    int i;  
    printf("Prestes a exhibir %d valores\n", num_de_elementos);  
    for (i = 0; i < num_de_elementos; i++)  
        printf("%d\n", valores[i]);  
}
```

```
int main () {  
  
    int notas[5] = {70, 80, 90, 100, 90};  
    int conta[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
    int pequeno[2] = {-33, -44};  
  
    exhibe_vetor(notas, 5);  
    exhibe_vetor(conta, 10);  
    exhibe_vetor(pequeno, 2);  
  
}
```