

Curso de C

Procedimentos e Funções

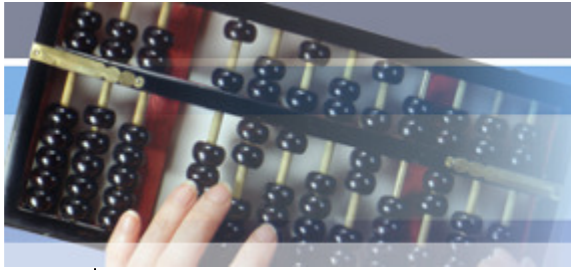
Funções

Roteiro:

- Funções
 - Declaração e chamada
- Funções importantes
- Exemplos de funções
- Variáveis
 - Globais, locais e parâmetros

Funções

O que são funções
O porquê de usá-las



Funções

Funções:

- São um trechos de código fora do programa principal
- Implementam um subalgoritmo do programa
- São utilizadas (chamadas) em diversos pontos do programa

Funções

Vantagens:

- Organiza o código:
 - Cada função é um algoritmo
 - Dividir um programa complicado em várias funções simples
- Evita repetição de código semelhante
 - Escrever o código apenas uma vez
 - Usar a mesma função várias vezes para variáveis diferentes
- Simplifica e agiliza a programação



Funções

Exemplo: comparar a média das duas melhores notas do aluno A e do aluno B

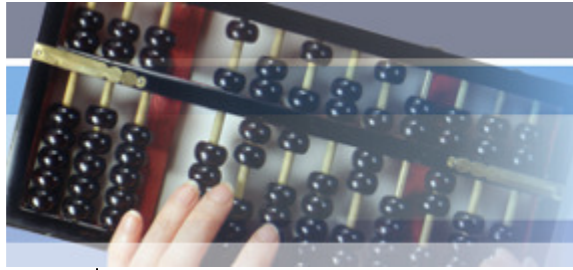
Algoritmo:

- Ler 3 notas dos alunos A e B
- Calcular a média de A
- Calcular a média de B
- Comparar as médias
- Imprimir resultados

Funções

Alguns problemas no exemplo ...

- Ler 3 notas dos alunos A e B
- Calcular a média de A
- Calcular a média de B
- Comparar as médias
- Imprimir resultados



Funções

Alguns problemas no exemplo ...

- Ler 3 notas dos alunos A e B
- Calcular a média de A
- Calcular a média de B
- Comparar as médias
- Imprimir resultados

Repetição do
mesmo algoritmo



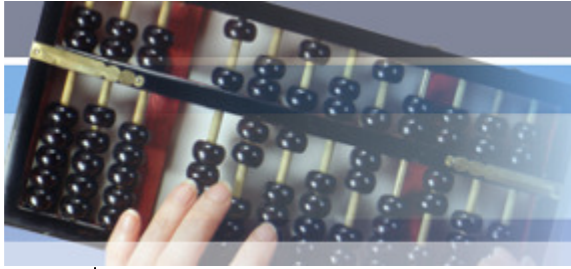
Funções

Alguns problemas no exemplo ...

- Ler 3 notas dos alunos A e B
- Calcular a média de A
- Calcular a média de B
- Comparar as médias
- Imprimir resultados

Repetição do
mesmo algoritmo

Solução: Uma função genérica para calcular a média



Funções

Alguns problemas no exemplo ...

- Ler 3 notas dos alunos A e B
- Calcular a média de A
- Calcular a média de B
- Comparar as médias
- Imprimir resultados

Funcoes\Nota01\Nota01.vcproj

Funções

Alguns problemas no exemplo ...

- Ler 3 notas dos alunos A e B
- Calcular a média de A
- Calcular a média de B
- Comparar as médias
- Imprimir resultados

Programa principal
mistura vários
algoritmos:

- Leitura
- Cálculo de média
- Comparação
- Impressão

Funcoes\Nota01\Nota01.vcproj

Funções

Alguns problemas no exemplo ...

- Ler 3 notas dos alunos A e B
- Calcular a média de A
- Calcular a média de B
- Comparar as médias
- Imprimir resultados

Programa principal
mistura vários
algoritmos:

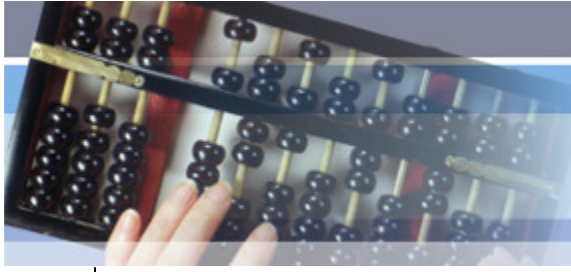
- Leitura
- Cálculo de média
- Comparação
- Impressão

Funcoes\Nota01\Nota01.vcproj

Solução: Uma função para cada algoritmo

Funções

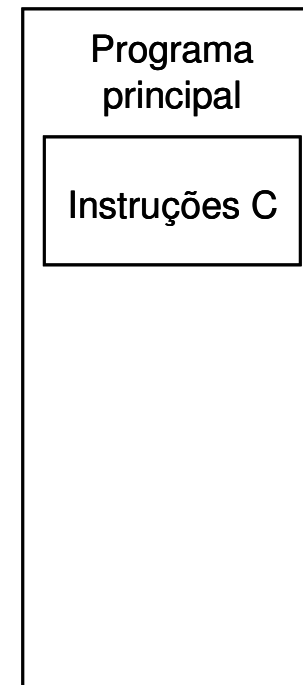
Chamada de Funções



Funções

Chamada de função:

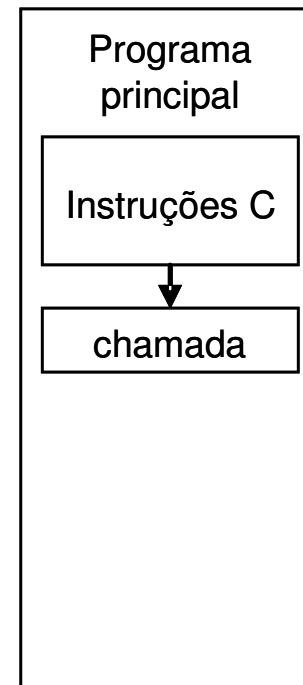
1. Interrupção do programa principal
2. Passagem de dados de entrada para a função
3. Execução do código da função
4. Retorno do resultado
5. Continuação do programa principal



Funções

Chamada de função:

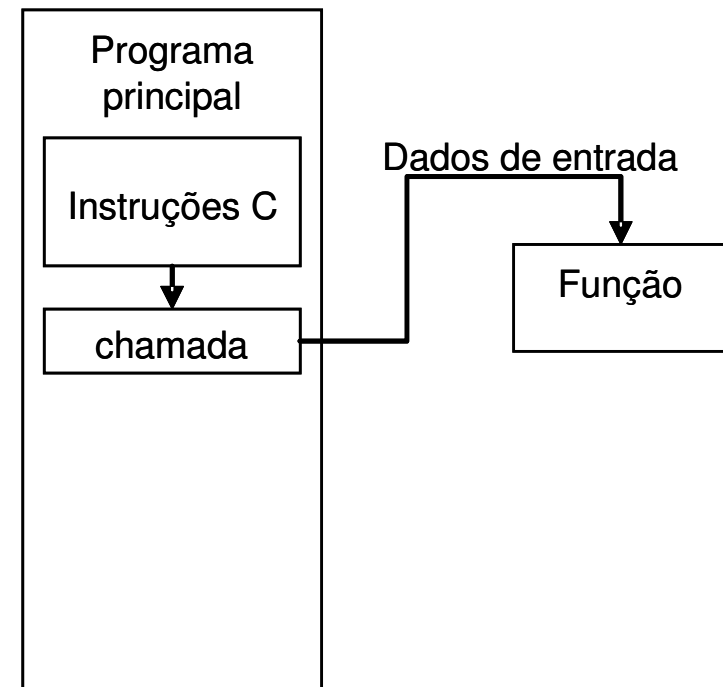
1. Interrupção do programa principal
2. Passagem de dados de entrada para a função
3. Execução do código da função
4. Retorno do resultado
5. Continuação do programa principal



Funções

Chamada de função:

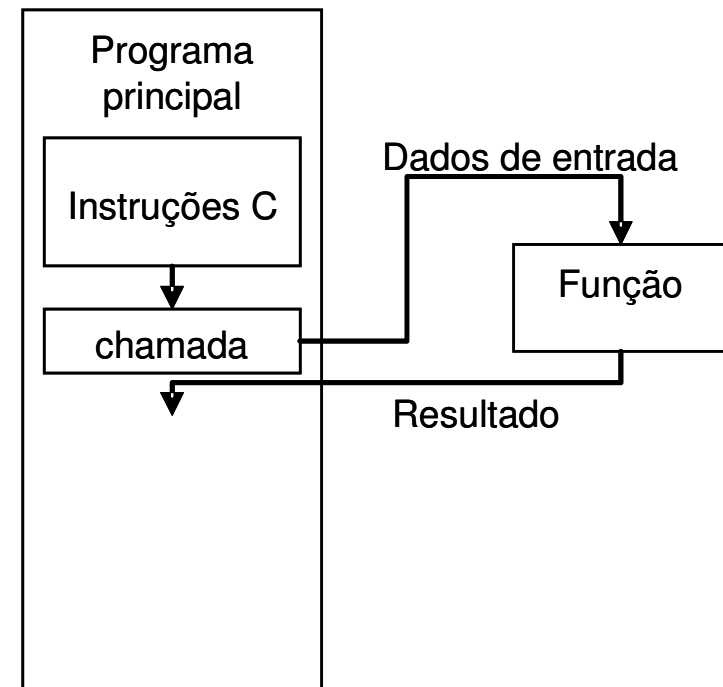
1. Interrupção do programa principal
2. Passagem de dados de entrada para a função
3. Execução do código da função
4. Retorno do resultado
5. Continuação do programa principal



Funções

Chamada de função:

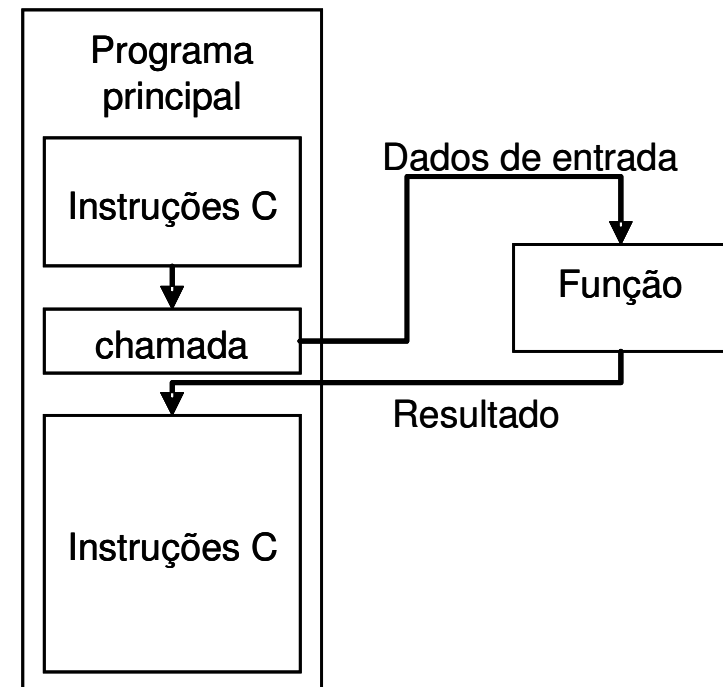
1. Interrupção do programa principal
2. Passagem de dados de entrada para a função
3. Execução do código da função
4. Retorno do resultado
5. Continuação do programa principal



Funções

Chamada de função:

1. Interrupção do programa principal
2. Passagem de dados de entrada para a função
3. Execução do código da função
4. Retorno do resultado
5. Continuação do programa principal





Funções

Chamada de função:

Para chamar uma função:

```
sentença (s) ;
```

```
...
```

```
resultado = nome(entr1, entr2, ...);
```

```
...
```

```
sentença (s) ;
```

Funções

Chamada de função:

Para chamar uma função:

```
sentença (s) ;
```

```
...
```

```
resultado = nome(entr1, entr2, ...);
```

```
...
```

```
sentença (s) ;
```

Nome da função

Funções

Chamada de função:

Para chamar uma função:

```
sentença (s) ;
```

```
...
```

```
resultado = nome (entr1, entr2, ...);
```

```
...
```

```
sentença (s) ;
```

Valores de entrada
(parâmetros)

Nome da função

Funções

Chamada de função:

Para chamar uma função:

```
sentença (s) ;
```

```
...
```

```
resultado = nome (entr1, entr2, ...);
```

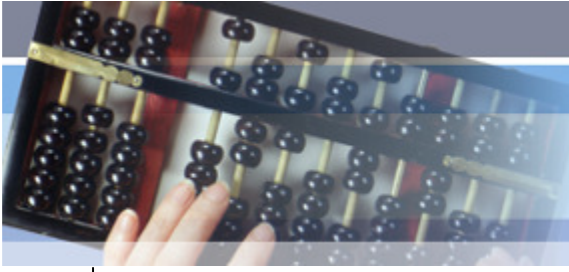
Resultado

Valores de entrada
(parâmetros)

```
...
```

```
sentença (s) ;
```

Nome da função

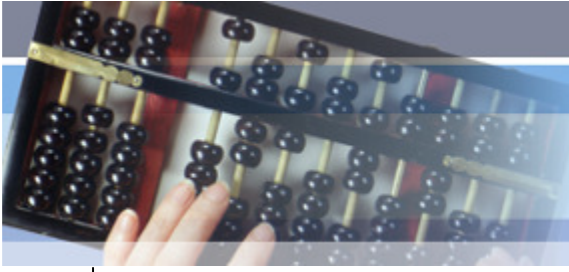


Exemplo: Calcular distância entre dois pontos

```
#include <stdio.h>
#include <math.h>
int main(int argc, char *argv[]) {
    double x1, x2, y1, y2, d;
    scanf("%lf, %lf, %lf, %lf", &x1, &y1, &x2, &y2);
    d = sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
    printf("Distância: %f", d);

    return 0
}
```

Funcoes\Chamada01\Chamada01.vcproj



Exemplo: Calcular distância entre dois pontos

```
#include <stdio.h>
#include <math.h>
int main(int argc, char *argv[]) {
    double x1, x2, y1, y2, d;
    scanf("%lf, %lf, %lf, %lf", &x1, &y1, &x2, &y2);
    d = sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
    printf("Distância: %f", d);

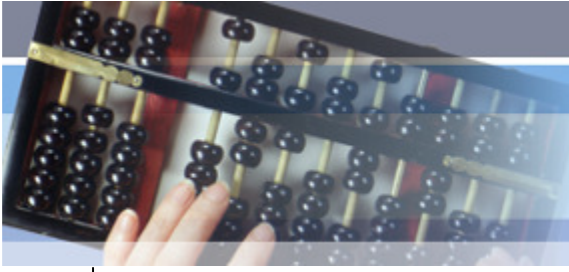
    return 0
}
```

Diagram illustrating the calculation of distance between two points using the Euclidean formula. The code uses `sqrt` to calculate the distance `d` based on input coordinates `x1, y1, x2, y2`. Annotations highlight the `sqrt` function and the input parameters.

Função

Valores de entrada
(parâmetros)

Funcoes\Chamada01\Chamada01.vcproj



Exemplo: Calcular distância entre dois pontos

```
#include <stdio.h>
#include <math.h>
int main(int argc, char *argv[]) {
    double x1, x2, y1, y2, d;
    scanf("%lf, %lf, %lf, %lf", &x1, &y1, &x2, &y2);
    d = sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
    printf("Distância: %f", d);

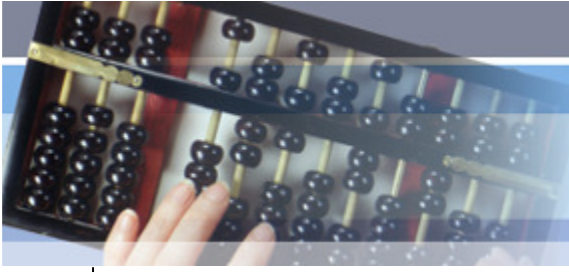
    return 0;
}
```

Interrompe o programa para executar **sqrt**

Função

Valores de entrada (parâmetros)

Funcoes\Chamada01\Chamada01.vcproj



Exemplo: Calcular distância entre dois pontos

```
#include <stdio.h>
#include <math.h>
int main(int argc, char *argv[]) {
    double x1, x2, y1, y2, d;
    scanf("%lf, %lf, %lf, %lf", &x1, &y1, &x2, &y2);
    d = sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
    printf("Distância: %f", d);
    return 0;
}
```

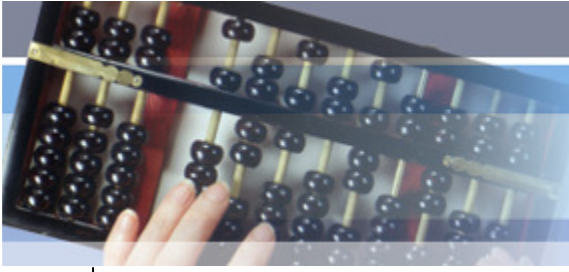
Interrompe o programa para executar **sqrt**

Função

Valores de entrada (parâmetros)

Resultado

Funcoes\Chamada01\Chamada01.vcproj



Exemplo: Calcular distância entre dois pontos

```
#include <stdio.h>
#include <math.h>
int main(int argc, char *argv[]) {
    double x1, x2, y1, y2, d;
    scanf("%lf, %lf, %lf, %lf", &x1, &y1, &x2, &y2);
    d = sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
    printf("Distância: %f", d);
    return 0;
}
```

Interrompe o programa para executar **sqrt**

Depois continua aqui

Valores de entrada (parâmetros)

Função

Resultado

Funcoes\Chamada01\Chamada01.vcproj

Funções

Definição de Funções



Funções

Definição de funções:

```
tipo nome (parâmetros) {  
    declarações de variáveis  
    ...  
    instruções;  
    ...  
    return valor;  
}
```

Funções

Nome:

- Identifica o trecho de código da função
- O nome é usado na chamada
- Formação igual a nome de variável

Nome da função

```
tipo nome (parâmetros) {  
    •declarações  
    •instruções;  
    return valor;  
}
```

Funções

Nome:

Nome da função: `media`

```
... media (...) {
```

- Declarações
- Instruções

```
...  
}
```

Corpo da função

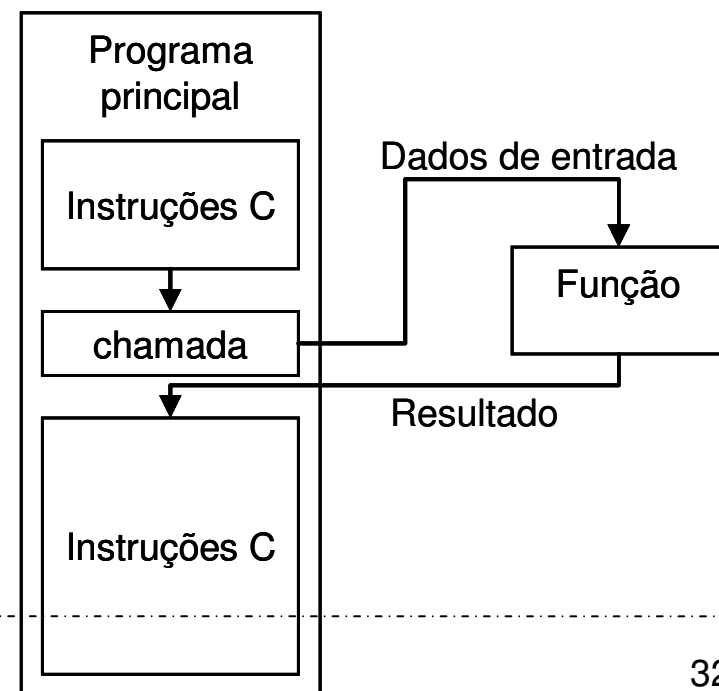
```
tipo nome (parâmetros) {  
    • declarações  
    • instruções;  
    return valor;  
}
```

Funções

Parâmetros:

- Definir dados de entrada.
- São variáveis locais, declaradas e atribuídas na chamada da função.
- Lista de tipos e nomes.

```
tipo nome(parâmetros) {  
    •declarações  
    •instruções;  
    return valor;  
}
```



Funções

Parâmetros:

Lista de 3 parâmetros:

Variáveis locais da função: `float nota1, nota2 e nota3`

```
... media(float nota1, float nota2,  
          float nota3) {
```

- Declarações

- Instruções

...

```
}
```

```
tipo nome(parâmetros) {  
    •declarações  
    •instruções;  
    return valor;  
}
```



Funções

Parâmetros na chamada de função:

- Ao chamar a função, o valor de cada parâmetro deve ser informado
- Regras:
 - Um valor para cada parâmetro
 - Valores compatíveis com o tipo do parâmetro
 - Valores na mesma ordem que na declaração

Chamada de função:

nome (*parametro1*, *parametro2*, ...)



Funções

Parâmetros na chamada de função:

```
float resultado;  
resultado = potencia(2.0f, 5.0f);
```

Quando a
função executa

Declaração da função **potencia**:

```
... potencia(float base, float expoente) {  
    ...  
    // Corpo da função  
    ...  
}
```

Funções

Parâmetros na chamada de função:

```
float resultado;  
resultado = potencia(2.0f, 5.0f);
```

Quando a
função executa

base = 2.0f

Declaração da função **potencia**:

```
... potencia(float base, float expoente) {  
    ...  
    // Corpo da função  
    ...  
}
```

Funções

Parâmetros na chamada de função:

```
float resultado;  
resultado = potencia(2.0f, 5.0f);
```

Quando a
função executa

base = 2.0f

expoente = 5.0f

Declaração da função **potencia**:

```
... potencia(float base, float expoente) {  
    ...  
    // Corpo da função  
    ...  
}
```

Funções

Parâmetros na chamada de função:

- Tipos: `int`, `long int`, `char`, `float`, etc
- São variáveis *locais* declaradas no corpo da função e que vão ser inicializadas com cópias dos valores correspondentes às expressões na chamada da função!

```
... maior_elemento(char c, int tamanho) {  
    // Corpo da função  
}
```

Funções

Função sem parâmetros:

Lista vazia de parâmetros

```
... funcaoSemParametros () {  
    •Declarações  
    •Instruções  
    ...  
}
```

Chamada de função:
`funcaoSemParametros ()`

```
tipo nome () {  
    •declarações  
    •instruções;  
    return valor;  
}
```

Funções

Corpo da função:

- Código para o algoritmo da função.
- Declara variáveis locais, além dos parâmetros.
- Parâmetros são variáveis locais, já pré-declaradas.

```
tipo nome(parâmetros) {  
    •declarações  
    •instruções;  
    return valor;  
}
```

Funções

Corpo da função:

```
... media(float nota1, float nota2, float nota3) {  
  
    float media;  
    if ((nota1 <= nota2) && (nota1 <= nota3)) {  
        media = (nota2 + nota3) / 2.0f;  
    } else if ((nota2 <= nota1) && (nota2 <=   
nota3)) {  
        media = (nota1 + nota3) / 2.0f;  
    } else {  
        media = (nota1 + nota2) / 2.0f;  
    }  
    ...  
}
```

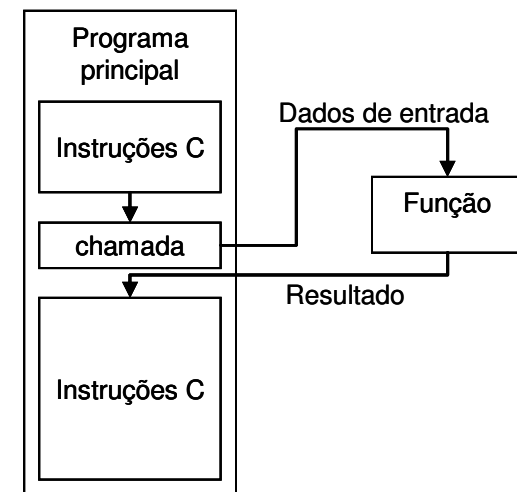
Funções

Valor de retorno:

- Tipo da função:
 - Indica o domínio do resultado da função
 - Qualquer tipo válido para variáveis

Tipo da
função

```
tipo nome (parâmetros) {  
    •declarações  
    •instruções;  
    return valor;  
}
```



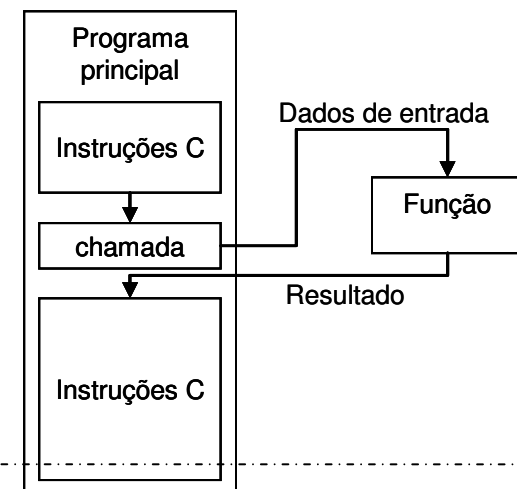
Funções

Valor de retorno:

- Comando `return`
- Termina a execução da função.
- Define o resultado (valor de retorno).
- Deve ser compatível com o tipo da função.

```
tipo nome(parâmetros) {  
    •declarações  
    •instruções;  
    return valor;  
}
```

Valor do resultado →





Funções

Valor de retorno:

```
float media(float nota1, float nota2, float nota3) {  
    float media;  
    if ((nota1 <= nota2) && (nota1 <= nota3)) {  
        media = (nota2 + nota3) / 2.0f;  
    } else if ((nota2 <= nota1) && (nota1 <= nota3)) {  
        media = (nota1 + nota3) / 2.0f;  
    } else {  
        media = (nota1 + nota2) / 2.0f;  
    }  
    return media;  
}
```

Funções

Função sem valor de retorno:

```
void comparaNotas(float mediaA, float mediaB) {  
    if (mediaA > mediaB) {  
        printf("Aluno A");  
    } else if (mediaA < mediaB) {  
        printf("Aluno B");  
    } else {  
        printf("Alunos A e B");  
    }  
    return;  
}
```

Tipo de retorno: **void**

Esta função não retorna resultado.

Funções

Exemplo completo: programa principal

```
int main(int argc, char *argv[]) {  
    float notaA1, notaA2, notaA3, mediaA;  
    float notaB1, notaB2, notaB3, mediaB;  
  
    scanf("%f %f %f", &notaA1, &notaA2, &notaA3);  
    scanf("%f %f %f", &notaB1, &notaB2, &notaB3);  
  
    mediaA = mediaMelhoresNotas(notaA1, notaA2,  
notaA3);  
    mediaB = mediaMelhoresNotas(notaB1, notaB2,  
notaB3);  
  
    comparaNotas(mediaA, mediaB);    Funcoes\Nota02\Nota02.vcproj  
    return 0;  
}
```

Funções

Funções importantes

Funções importantes

Função `main`:

- Contém o programa principal
- Várias declarações possíveis:
 - `void main(void) { ... }`
 - `int main() { ... }`
 - `int main(int argc, char *argv[])`



Funções importantes

Leitura e Escrita:

- Ler dados do usuário e escrever na tela
- **#include <stdlib.h>**
- Várias funções disponíveis:
scanf, printf, getchar, putchar, gets,
puts, ...
- Exemplo:
printf("Resultado: %d", valor);
scanf("%d %d", &linhas, &colunas);



Funções importantes

Manipulação de arquivos:

- Ler e escrever dados em arquivos ou fluxos de dados.
- **#include <stdio.h>**
- Várias funções disponíveis:
fopen, fclose, fscanf, fprintf, rewind,
fflush, fputc, fputs, fgetc, fgets,
ftell, fseek, etc...



Funções importantes

Matemáticas:

- Realizar operações matemáticas.
- **#include <math.h>**
- Várias funções disponíveis:
 `sqrt, sqr, pow, exp, log, log10`
 `sin, asen, sinh, cos, acos, cosh, tan,`
 `atan, atan2, tanh`
 `floor, ceil, abs, fabs`



Funções importantes

Texto e caracteres:

- Modificar texto (vetores de caracteres).
- **#include <string.h>**
- Várias funções disponíveis:
`strcat, strdup, strlen, strcmp, strcpy,`
`strstr`

Funções

Exemplos de Funções



Exemplos de Funções

Matemática:

```
float potencia(float base, int expoente) {  
    float resultado = 1.0;  
    int i;  
    for (i = 0; i < expoente; i++) {  
        resultado = resultado * base;  
    }  
    return resultado;  
}
```

Funcoes\Potencia01\Potencia01.vcproj

Funcoes\Potencia02\Potencia02.vcproj



Exemplos de Funções

Lógica de programa:

```
int le_numero(int min, int max) {  
    int leitura;  
  
    scanf("%d", &leitura);  
    while ((leitura < min) || (leitura > max)) {  
        printf("Digite entre %1d e %1d)\n", min, max);  
        printf("Digite novamente: ");  
        scanf("%d", &leitura);  
    }  
    return leitura;  
}
```

Funcoes\Entrada01\Entrada01.vcproj



Exemplos de Funções

Gerar/obter dados:

```
#include <time.h>
int main(int argc, char argv[]) {
    int fim, inicio = clock();
    int i;

    for (i = 0; i < 20000; i++) {
        printf("Um texto para imprimir... %d\n", i);
    }
    fim = clock();
    printf("Tempo necessário: %d\n", fim - inicio);
    return 0;
}
```

Funcoes\Time01\Time01.vcproj



Exemplos de Funções

Organização de código:

Ruim:

```
char opcao;  
scanf("%c", &opcao);  
if (opcao == 'P') {  
    // Código para calcular raizes de polinômios  
    // ...  
} else if (opcao == 'M') {  
    // Código para calcular determinantes de matrizes  
    // ...  
} else if (opcao == 'D') {  
    // Código para calcular derivadas de funções  
    // ...  
}
```

Exemplos de Funções

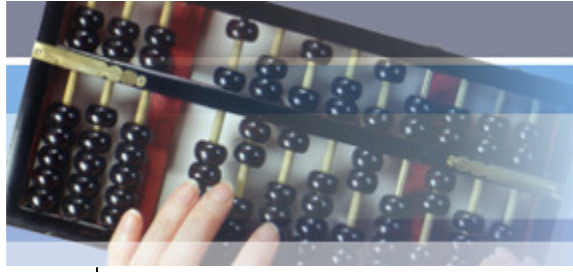
Organização de código:

```
void resolver_polinimio(void) {  
    // Código ...  
}  
void calc_determinante(void) {  
    // Código ...  
}  
void derivar_funcao(void) {  
    // Código ...  
}
```

```
scanf("%c", &opcao);  
if (opcao == 'P') {  
    resolver_polinomio();  
} else if (opcao == 'M') {  
    calc_determinante();  
} else if (opcao == 'D') {  
    derivar_funcao();  
}  
}
```

Funções

Tempo de vida de variáveis



Variáveis

Tempo de vida:

Tempo que o valor da variável permanece na memória:

- do início até final do **programa**, ou
- do início até final da **função**
- no início da função: **cria** espaço na memória
- no fim da função: **libera** espaço usado na memória

Variáveis

Tempo de vida:

- Variáveis declaradas localmente em funções:
 - Tempo de vida: durante a execução da função
 - Durante a execução a variável é visível, acessível
 - Durante a execução estão associadas à posições de memória
 - Em duas execuções, podem ocupar posições de memória diferentes

```
{  
    int v = 0;  
    ...  
}
```

} Vida da variável **v**

Variáveis

Tempo de vida:

- Declaração dentro do corpo da função
 - Visibilidade apenas na função

```
int funcao(void) {  
    int v = 0;  
    ...  
}  
  
int main(void) {  
    ...  
    a();  
}
```

Variáveis

Tempo de vida:

- Declaração dentro do corpo da função
 - Visibilidade apenas na função

```
int funcao(void) {  
    int v = 0;  
    ...  
}  
  
int main(void) {  
    ...  
    a();  
}
```

} Visibilidade da variável **v**

Variáveis

Tempo de vida:

- Declaração dentro do corpo da função
 - Visibilidade apenas na função

```
int funcao(void) {  
    int v = 0;  
    ...  
}  
  
int main(void) {  
    ...  
    a();  
}
```

} Visibilidade da variável **v**

} Aqui a variável **v** é invisível

Variáveis

Tempo de vida:

- Variáveis declaradas fora do corpo das funções
 - Sobrevivem por toda a execução do programa
 - Permanecem ocupando a mesma posição de memória

```
int v = 0;
```

```
int funcao(void) {  
    ...  
}
```

```
int main(void) {  
    ...  
    a();  
}
```

Vida de variável **v**

Variáveis

Tempo de vida:

```
int x = 0;
int a(void) {
    int v = 0;
    ...
}
int b(void) {
    int w = 0;
    ...
}
int main(void) {
    ...
}
```

Variáveis

Tempo de vida:

```
int x = 0;
int a(void) {
    int v = 0;
    ...
}
int b(void) {
    int w = 0;
    ...
}
int main(void) {
    ...
}
```

} Vida da
variável **v**

} Vida da
variável **w**

Variáveis

Tempo de vida:

```
int x = 0;
int a(void) {
    int v = 0;
    ...
}
int b(void) {
    int w = 0;
    ...
}
int main(void) {
    ...
}
```

Vida da
variável **v**

Vida da
variável **w**

Vida da
variável **x**

Variáveis

Tempo de vida:

```
int v = 0;

int funcaoA(void) {
    int w = 0;
    ...
}

int funcaoB(void) {
    ...
}

int main(void) {
    ...
}
```

Variáveis

Tempo de vida:

```
int v = 0;
int funcaoA(void) {
    int w = 0;
    ...
}

int funcaoB(void) {
    ...
}

int main(void) {
    ...
}
```

} **w** inacessível

Variáveis

Tempo de vida:

```
int v = 0;
int funcaoA(void) {
    int w = 0;
    ...
}
int funcaoB(void) {
    ...
}
int main(void) {
    ...
}
```

} w inacessível

} w acessível

Variáveis

Tempo de vida:

```
int v = 0;
int funcaoA(void) {
    int w = 0;
    ...
}
int funcaoB(void) {
    ...
}
int main(void) {
    ...
}
```

} w inacessível

} w acessível

} w inacessível

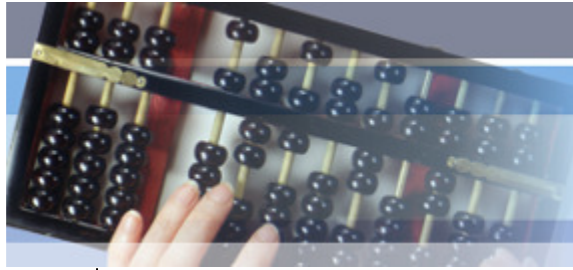
Variáveis

Tempo de vida:

```
int v = 0;
int funcaoA(void) {
    int w = 0;
    ...
}
int funcaoB(void) {
    ...
}
int main(void) {
    ...
}
```

Diagram illustrating the lifetime of variables in the provided C code:

- v** is **sempre acessível** (always accessible) throughout the entire program.
- w** is **inacessível** (inaccessible) outside of `funcaoA` and `funcaoB`.
- w** is **acessível** (accessible) inside `funcaoA`.
- w** is **inacessível** (inaccessible) inside `funcaoB`.



Variáveis

Sobreposição de nomes:

- Condições:
 - Variáveis com mesmo nome
 - Sobreposição de escopo
- Conseqüência:
 - Declaração mais recente torna inacessível outras declarações

Variáveis

Sobreposição de nomes:

```
int w = 0;

int funcao (void) {
    int w = 1;
    ...
}

int main(void) {
    ...
}
```

Variáveis

Sobreposição de nomes:

```
int w = 0;
```

} w acessível

```
int funcao (void) {
```

```
    int w = 1;
```

```
    ...
```

```
}
```

```
int main(void) {
```

```
    ...
```

```
}
```

Variáveis

Sobreposição de nomes:

```
int w = 0;
```

```
int funcao (void) {  
    int w = 1;  
    ...  
}
```

```
int main(void) {  
    ...  
}
```

} w acessível

} w acessível, mas
sobreposição w anterior

Variáveis

Sobreposição de nomes:

```
int w = 0;
```

```
int funcao (void) {  
    int w = 1;  
    ...  
}
```

```
int main(void) {  
    ...  
}
```

} **w** acessível

} **w** acessível, mas
sobre põe **w** anterior

} **w** anterior
novamente acessível

Variáveis

Variável global:

- Declarada fora das funções
- Tempo vida:
 - até fim do programa
 - acessível à todas as funções do programa, exceto quando há sobreposição
- Inicialização:
 - ao iniciar o programa

```
int v = 0;

void f1(void) {
    ...
}
void f2(void) {
    ...
}
```

Variáveis

Variável global:

```
int v = 0;
void f1(void) {
    v++;
    printf("f1: v = %d\n", v);
}
void f2(void) {
    v+=2;
    printf("f2: v = %d\n", v);
}
int main(int argc, char argv[]) {
    f1();
    f2();
    f1();
}
```

Funcoes\Visibilidade01\Visibilidade01.vcproj

Resultado:

f1: v =

f2: v =

f1: v =

Variáveis

Variável global:

```
int v = 0;
void f1(void) {
    v++;
    printf("f1: v = %d\n", v);
}
void f2(void) {
    v+=2;
    printf("f2: v = %d\n", v);
}
int main(int argc, char argv[]) {
    f1();
    f2();
    f1();
}
```

Funcoes\Visibilidade01\Visibilidade01.vcproj

Resultado:

f1: v = 1
f2: v = 3
f1: v = 4

Variáveis

Variável local:

- Declarada dentro das funções
- Tempo vida:
 - do início até o fim da função
 - nova execução não lembra valor anterior
- Inicialização:
 - no início da função

```
void f1(void) {  
    int v = 0;  
    ...  
}  
void f2(void) {  
    int w = 0;  
    ...  
}
```

Variáveis

Variável local:

```
void f1() {  
    int v = 0;  
    v++;  
    printf("f1: v = %d\n", v);  
}  
void f2() {  
    int v = 0;  
    v+=2;  
    printf("f2: v = %d\n", v);  
}  
int main(int argc, char argv[]) {  
    f1();  
    f2();  
    f1();  
}
```

Funcoes\Visibilidade02\Visibilidade02.vcproj

Resultado:

f1: v =
f2: v =
f1: v =

Variáveis

Variável local:

```
void f1() {  
    int v = 0;  
    v++;  
    printf("f1: v = %d\n", v);  
}  
void f2() {  
    int v = 0;  
    v+=2;  
    printf("f2: v = %d\n", v);  
}  
int main(int argc, char argv[]) {  
    f1();  
    f2();  
    f1();  
}
```

Funcoes\Visibilidade02\Visibilidade02.vcproj

Resultado:

f1: v = 1
f2: v = 2
f1: v = 1

Funções

*Arquivos de cabeçalhos
Compilação em separado*



Arquivos Cabeçalho

Arquivos cabeçalho (“**headers**” ou **.h**):

A diretiva **#include**

- Permite incluir arquivos no ponto de chamada
- Texto do arquivo substitui a diretiva, no ponto de chamada
- Nome do arquivo entre **< >**
- Sistema operacional sabe onde encontrar o arquivo

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



Arquivos Cabeçalho

Seus próprios arquivos cabeçalho:

- Nomes com extensão .h
- Nomes devem aparecer entre aspas duplas (")
- Indicar diretório onde estão os arquivos
 - Default depende do sistema operacional
- Contém: definições de constantes comuns, etc. . .

```
...  
#include "consts.h"  
...
```

Arquivos Cabeçalho

Seus próprios arquivos cabeçalho:

- Nomes com extensão .h
- Nomes devem aparecer entre aspas duplas (")
- Indicar diretório onde estão os arquivos
 - Default depende do sistema operacional
- Contém: definições de constantes comuns, etc. . .

```
.....  
#include "consts.h"  
.....
```

```
/* arquivo .h com definição  
   de constantes */  
#define pi 3.141592  
#define true 1  
. . . .
```

Arquivos Cabeçalho

Seus próprios arquivos cabeçalho:

- Nomes com extensão .h
- Nomes devem aparecer entre aspas duplas (")
- Indicar diretório onde estão os arquivos
 - Default depende do sistema operacional
- Contém: definições de constantes comuns, etc. . .

Arquivo const.h

```
....  
#include "consts.h"  
....
```

```
/* arquivo .h com definição  
   de constantes */  
#define pi 3.141592  
#define true 1  
. . . .
```

Pragmas

Onde escrever o código de funções:

Até agora:

```
int funcaoA(int a) {  
    ...  
}
```

```
float funcaoB(char c, double r) {  
    ...  
}
```

```
char funcaoC(char c, int x) {  
    ...  
}
```

```
//
```

```
Int main( ... ) {  
    ...  
}
```

Pragmas

Onde escrever o código de funções:

Até agora:

```
int funcaoA(int a) {  
    ...  
}
```

```
float funcaoB(char c, double r) {  
    ...  
}
```

funcaoA conhecida aqui

```
char funcaoC(char c, int x) {  
    ...  
}
```

```
//
```

```
Int main( ... ) {  
    ...  
}
```

Pragmas

Onde escrever o código de funções:

Até agora:

```
int funcaoA(int a) {  
    ...  
}
```

```
float funcaoB(char c, double r) {  
    ...  
}
```

funcaoA conhecida aqui

```
char funcaoC(char c, int x) {  
    ...  
}
```

funcaoA, funcaoB conhecidas aqui

```
//  
Int main( ... ) {  
    ...  
}
```

Pragmas

Onde escrever o código de funções:

Até agora:

```
int funcaoA(int a) {  
    ...  
}
```

```
float funcaoB(char c, double r) {  
    ...  
}
```

funcaoA conhecida aqui

```
char funcaoC(char c, int x) {  
    ...  
}
```

funcaoA, funcaoB conhecidas aqui

```
//  
int main( ... ) {  
    ...  
}
```

funcaoA, funcaoB, funcaoC conhecidas aqui

Pragmas

Código das funções no final: pragmas

```
....  
int funcaoA(int a);  
float funcaoB(char c, double r);  
char funcaoC(char c, int x);  
//  
Int main( ... ) {  
    ... }  
//  
int funcaoA(int a) {  
    ... }  
float funcaoB(char c, double r) {  
    ... }  
char funcaoC(char c, int x) {  
    ... }
```

Pragmas

Código das funções no final: pragmas

```
....  
int funcaoA(int a);  
float funcaoB(char c, double r);  
char funcaoC(char c, int x);  
//  
Int main( ... ) {  
    ....}  
//  
int funcaoA(int a) {  
    .... }  
float funcaoB(char c, double r) {  
    .... }  
char funcaoC(char c, int x) {  
    .... }
```

} pragmas de funções

Pragmas

Código das funções no final: pragmas

```
....  
int funcaoA(int a);  
float funcaoB(char c, double r);  
char funcaoC(char c, int x);  
//  
Int main( ... ) {  
    ....}  
//  
int funcaoA(int a) {  
    ....}  
float funcaoB(char c, double r) {  
    ....}  
char funcaoC(char c, int x) {  
    ....}
```

} pragmas de funções

funcaoA, funcaoB, funcaoC conhecidas aqui

Pragmas

Código das funções no final: pragmas

```
....  
int funcaoA(int a);  
float funcaoB(char c, double r);  
char funcaoC(char c, int x);  
//  
Int main( ... ) {  
    ....  
//  
int funcaoA(int a) {  
    ....  
float funcaoB(char c, double r) {  
    ....  
char funcaoC(char c, int x) {  
    ....  
}
```

pragmas de funções

funcaoA, funcaoB, funcaoC conhecidas aqui

Corpo das funções

Funções

Fim do capítulo