INE5231 Computação Científica I

Prof. A. G. Silva

09 de maio de 2017

Conteúdo programático

- **O computador - [3 horas-aula]**
- **Representação de algoritmos - [3 horas-aula]:**
- **Linguagens de programação estruturadas [3 horas-aula]**
- **Introdução à programação em C [6 horas-aula]**
- **Programas envolvendo processos de repetição e seleção [6 horas-aula]**
- **Variáveis estruturadas unidimensionais homogêneas [9 horas-aula]**
- **Variáveis estruturadas multidimensionais homogêneas [6 horas-aula]**
- **Subdivisão de problemas e subprogramação [6 horas-aula]**
- **Variáveis estruturadas heterogêneas [6 horas-aula]**
- **Programação utilizando uma linguagem de computação técnica numérica [6 horas-aula]**

Curso de C

Procedimentos e Funções

Funções

Roteiro:

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

Funções

Funções:

- São 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

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

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

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

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

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:

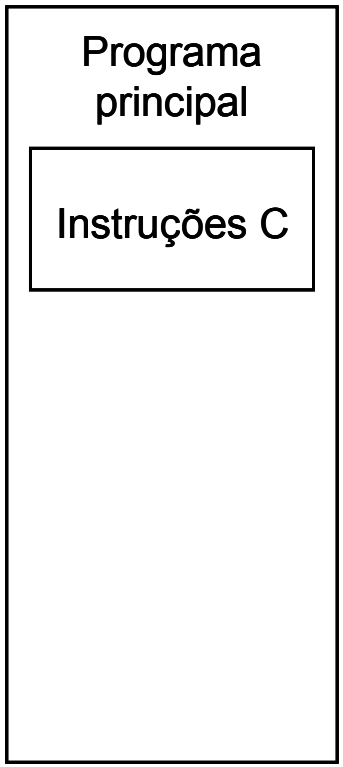
- Leitura
- Cálculo de média
- Comparação
- Impressão

Funcoes\Nota01\Nota01.vcproj

Solução: Uma função para cada algoritmo

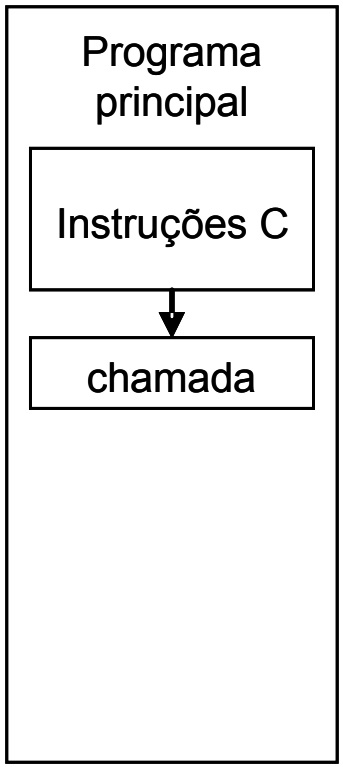
Chamada de função:

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



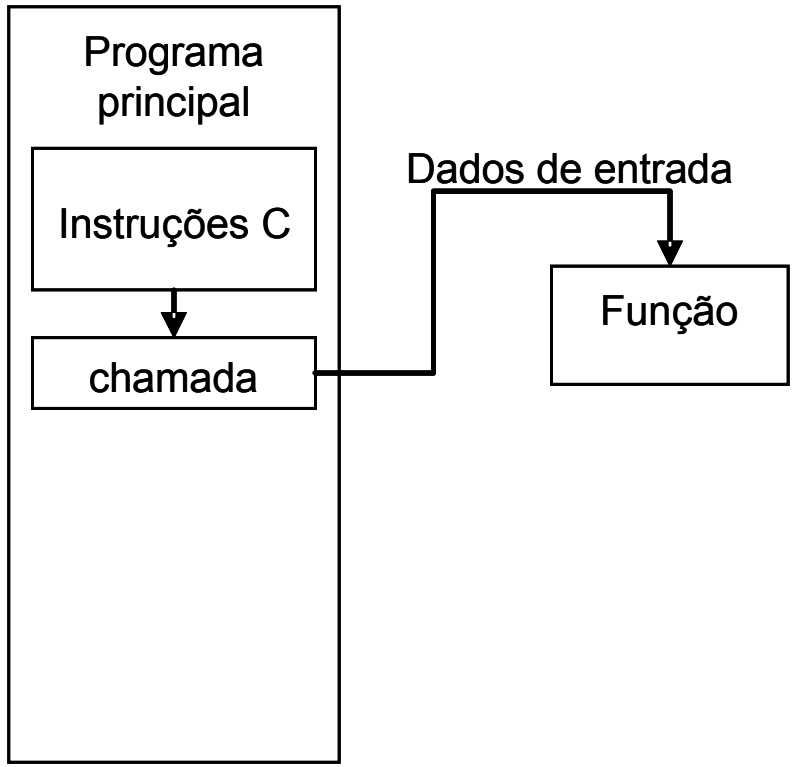
Chamada de função:

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



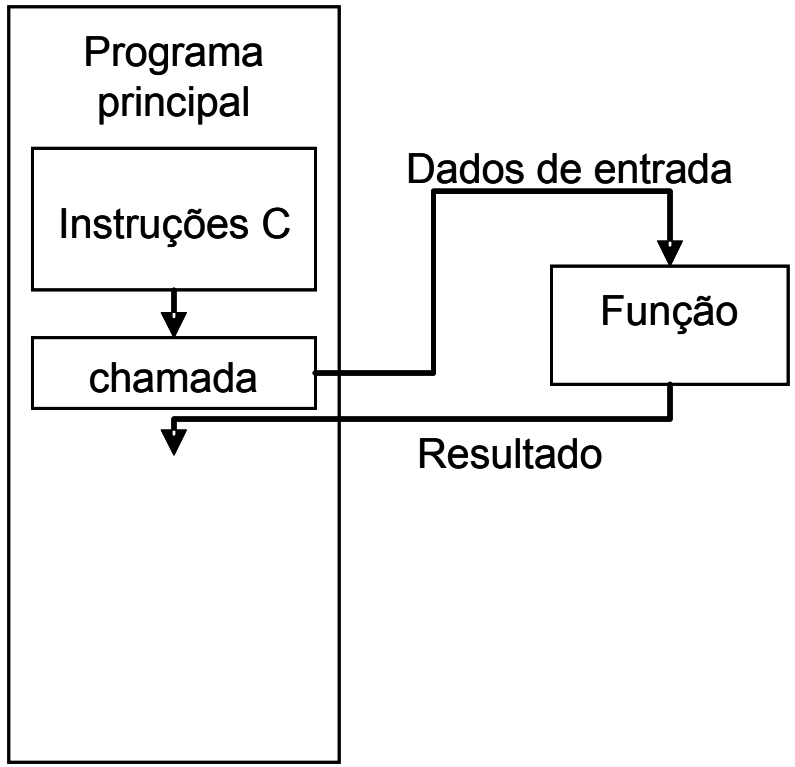
Chamada de função:

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



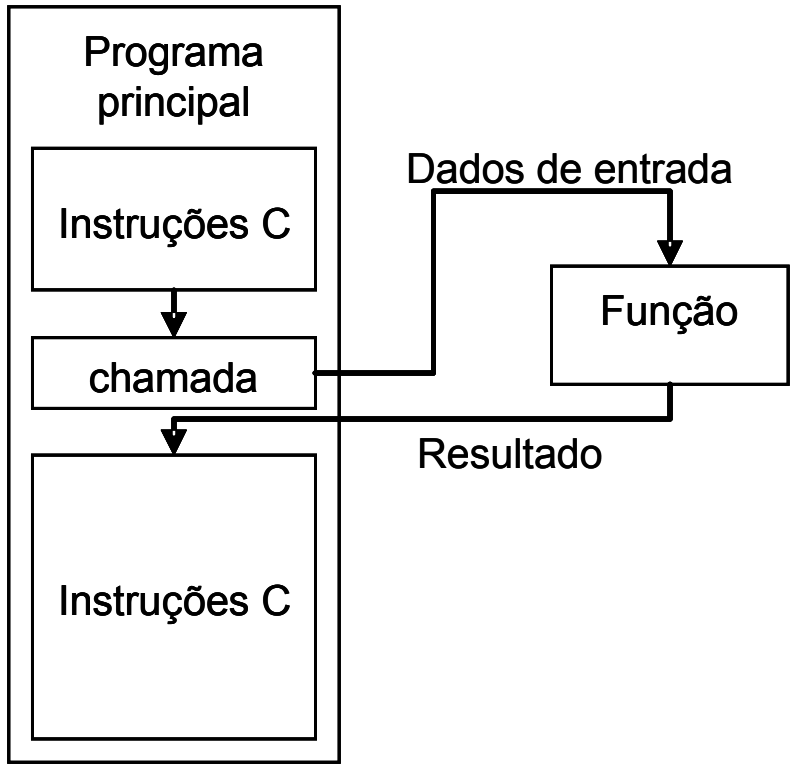
Chamada de função:

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



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



Chamada de função:

Para chamar uma função:

```
sentença(s);
...
resultado = nome(entr_1, entr_2, ...);



...
sentença(s);
```

Chamada de função:

Para chamar uma função:

```
sentença(s);
...
resultado = nome(entr_1, entr_2, ...);


...
sentença(s);
```

Nome da função

Funções

Chamada de função:

Para chamar uma função:

```
sentença(s);
...
resultado = nome(entr_1, entr_2, ...);



...
sentença(s);
```

Valores de entrada (parâmetros)

Nome da função

Chamada de função:

Para chamar uma função:

```
sentença(s);
...
resultado = nome(entr_1, entr_2, ...);
...
sentença(s);
```

Valores de entrada (parâmetros)

Resultado

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: %lf", 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: %lf", d);

   return 0;
}
```

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: %lf", 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: %lf", 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)*
   printf("Distância: %lf", d);

   return 0;
}
```

Interrompe o programa para executar **sqrt**

Depois continua aqui

Valores de entrada (parâmetros)

Função

Resultado

Funcoes\Chamada01\Chamada01.vcproj

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`

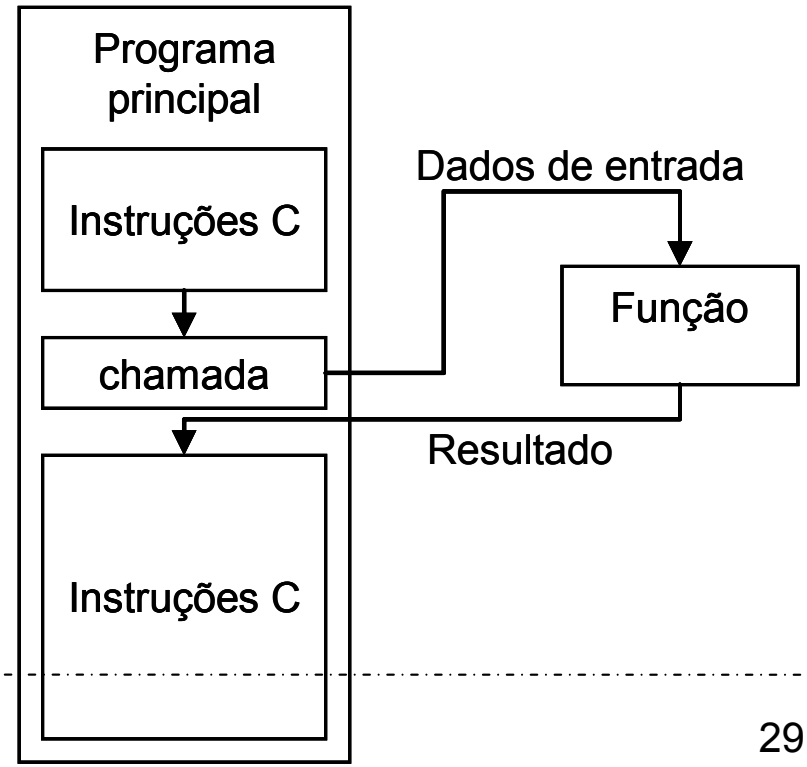
```
... media(...) {

   •Declarações
   •Instruções

   ...
}
```

Corpo da função

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

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;
}
```

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, ...*)

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
      ...
}
```

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
      ...
}
```

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
}
```

Função sem parâmetros:

Lista vazia de parâmetros

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

Chamada de função:
`funcaoSemParametros()`

```
tipo nome(void) {
      •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;
}
```

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;
      }
      ...
}
```

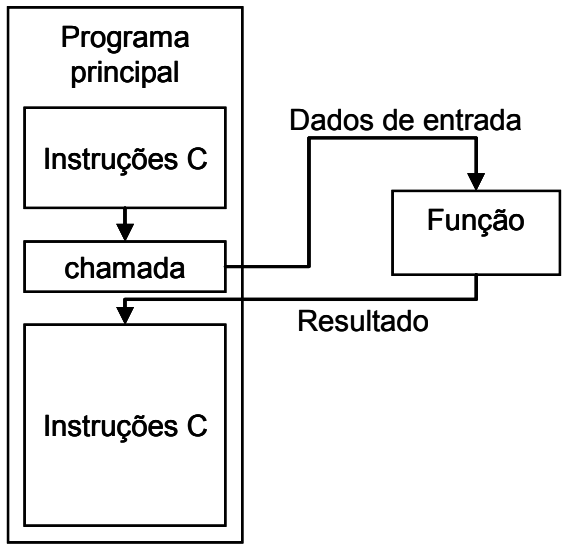
Funções

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;
}
```

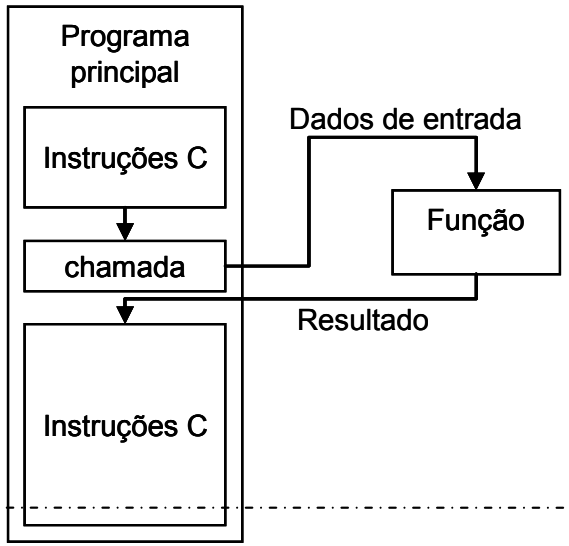


Valor de retorno:

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

Valor do resultado

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



Valor de retorno:

```
float mediaMelhoresNotas(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;
}
```

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.

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);
      return 0;
}
```

Funcoes\Nota02\Nota02.vcproj

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:

```
prinft("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, senh, cos, acos, cosh, tan,
  atan, atan2, tanh
floor, ceil, abs
```

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

Exemplo: Leitura de dados

```
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

Função para “medir tempo”:

```
#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 calcular_determinante(void) {
        // Código ...
}
void derivar_funcao(void) {
        // Código ...
}
```

```
scanf("%c", &opcao);
if (opcao == 'P') {
        resolver_polinomio();
        } else if (opcao == 'M') {
        calcular_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
- da chamada até final da função

Variáveis

Tempo de vida:

- Variáveis declaradas localmente em uma função:
  - Tempo de vida: durante a execução da função
  - Durante a execução a variável é visível, acessível
  - Durante a execução está associada à posições de memória
  - Em duas execuções, ocupam 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 da variável **v**

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**

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;                    }
                              }  w inacessível
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

w acessível

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) {
      ...
}
```

w inacessível

w acessível

w inacessível

v sempre acessível

Sobreposição de nomes:

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

Sobreposição de nomes:

```
int w = 0;

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

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

Sobreposição de nomes:

```
int w = 0;

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

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

`w` acessível

Sobreposição de nomes:

```
int w = 0;

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

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

`w` acessível

`w` acessível, mas sobrepõe `w` anterior

Sobreposição de nomes:

```
int w = 0;

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

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

`w` acessível

`w` acessível, mas sobrepõe `w` anterior

`w` anterior novamente acessível

Variáveis

Variável global:

- Declarada fora das funções
- Tempo vida:
  - até fim do programa
  - acessível a todas 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
. . . .
```

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( ... ) {
  . . .
}
```

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( ... ) {
  . . .
}
```

funcaoA conhecida aqui

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( ... ) {
  . . .
}
```

funcaoA conhecida aqui

funcaoA, funcaoB conhecidas aqui

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( ... ) {
  . . .
}
```

funcaoA conhecida aqui

funcaoA, funcaoB conhecidas aqui

funcaoA, funcaoB, funcaoC conhecidas aqui

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) {
   . . . }
```

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

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

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

Referências

- Notas do Prof. Arnaldo V. Moura e Daniel F. Ferber – Curso C – IC/Unicamp