Universidade Positivo

Algoritmos de Programação

Aula 22

Prof.ª Mariane Cassenote

mariane.cassenote@up.edu.br

Conteúdo

- Conceitos relacionados à modularização de código
- Procedimentos e Funções
- Continuidade da introdução à Linguagem C

Ao final do componente curricular será possível aplicar os conceitos de modularização, procedimentos e funções.

Modularidade

Modularidade

- O conceito de modularidade tem sido adotado desde meados da década de 1950
- Problemas complexos exigem algoritmos complexos para sua solução
- É possível dividir um grande problema em problemas menores (**dividir para conquistar**)
- Cada problema menor (ou módulo) tem um algoritmo mais simples
- Exemplo da empresa

PRESSMAN, R. S. Engenharia de Software. São Paulo: Makron Books, 1995.

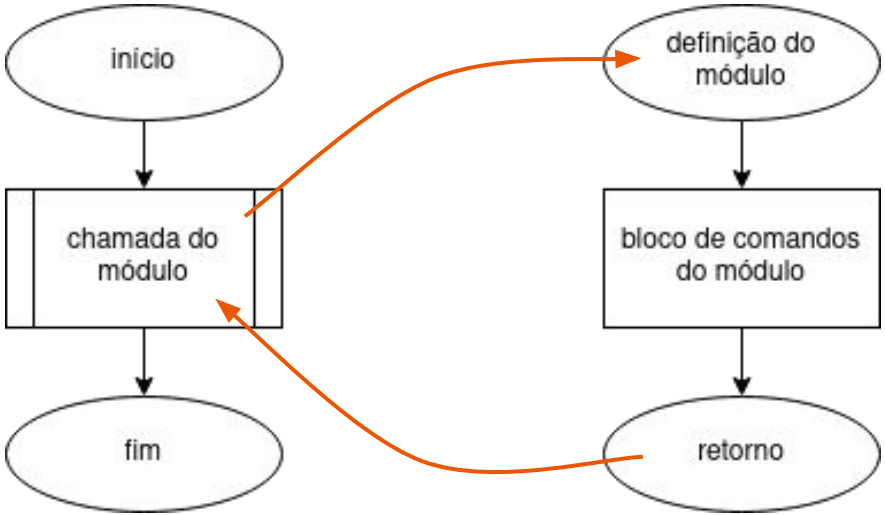
Modularidade

- Módulos também são utilizados quando um bloco de código se repete na resolução de um problema
- Um módulo é escrito somente uma vez e pode ser chamado sempre que necessário (reaproveitamento de código)
- Legibilidade do código
  - Se o programa principal é muito extenso, fica difícil ler e entender o que está sendo feito
  - Com módulos fica mais fácil visualizar as diferentes partes do programa

PRESSMAN, R. S. Engenharia de Software. São Paulo: Makron Books, 1995.

Modularidade

A execução de um programa sempre começa pela `main()` (função principal)



MANZANO, José Augusto Navarro Garcia; OLIVEIRA, Jayr Figueiredo de. Algoritmos: lógica para desenvolvimento de programação de computadores. 22. ed. São Paulo: Ed. Érica, 2009.

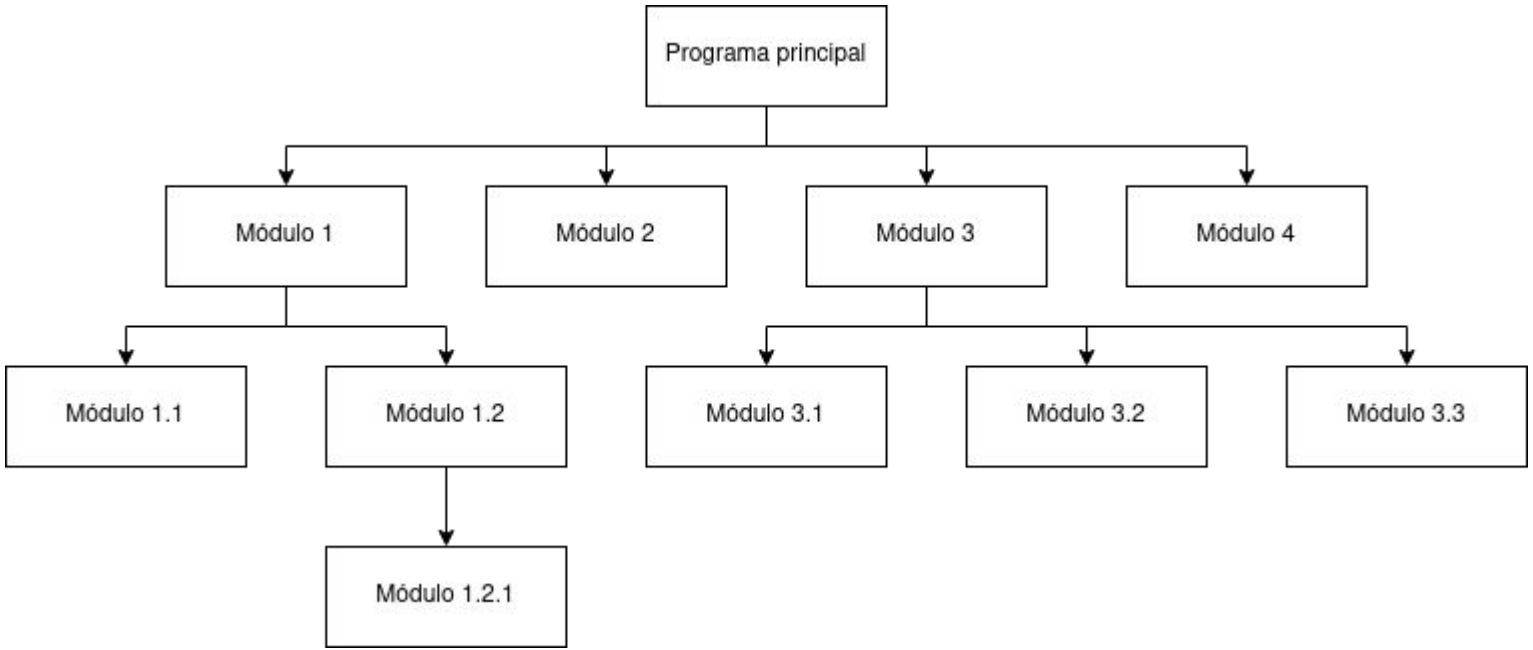
Modularidade

- A parte principal do programa é suspensa durante a execução de um módulo, o que implica na existência de somente um módulo em execução em um determinado momento (exceto em situações de paralelismo)
- Quando a execução do módulo é encerrada, o fluxo de execução do programa volta à linha de instrução em que o módulo foi chamado na parte principal do programa
- Cada módulo pode chamar outros módulos

MANZANO, José Augusto Navarro Garcia; OLIVEIRA, Jayr Figueiredo de. Algoritmos: lógica para desenvolvimento de programação de computadores. 22. ed. São Paulo: Ed. Érica, 2009.

Modularidade

A `main()` deve ser a última função do arquivo. Implementação de baixo para cima.



MANZANO, José Augusto Navarro Garcia; OLIVEIRA, Jayr Figueiredo de. Algoritmos: lógica para desenvolvimento de programação de computadores. 22. ed. São Paulo: Ed. Érica, 2009.

Modularidade

- Módulos podem ser:
    - **Procedimentos**: não retornam valor
    - **Funções**: retornam valor
- Módulos tem a mesma estrutura de um algoritmo: tem início, meio e fim e podem ter variáveis locais
- A execução dos módulos é temporária

Modularidade

- Parâmetros de entrada ou argumentos em módulos:
  - Fazem a troca de informações entre módulos
  - É opcional, depende do que o módulo deverá realizar
  - Se um módulo não recebe nenhum parâmetro, sua lista é vazia ou `void`
- O código do módulo deve ser escrito antes de sua chamada. Ou seja, se estiverem no mesmo arquivo, os módulos devem ser escritos **acima da `main()`**

Modularidade - em C

Variáveis criadas dentro de um módulo são conhecidas somente por ele (**locais**)

FUNÇÃO COM RETORNO VAZIO

```
void <nome> (<parâmetros – opcional>) {
    <declaração de variáveis – opcional>
    <bloco de comandos>
}
```

Se for um **procedimento**, o **retorno** não tem valor.
Se for **função**, tem.

FUNÇÃO COM RETORNO

```
tipo de retorno <nome> (<parâmetros – opcional>) {
    <declaração de variáveis – opcional>
    <bloco de comandos>
    return <valor de retorno>;
}
```

Modularidade - em C

FUNÇÃO COM RETORNO VAZIO

```c
#include <stdio.h>

void mult(int a, int b) {
  printf("%i \n", a * b);
}

int main() {
  int a = 8, b = 3;
  mult(a, b);
  return 0;
}
```

FUNÇÃO COM RETORNO

```c
#include <stdio.h>

int mult(int a, int b) {
  return a * b;
}

int main() {
  int resultado, a = 8, b = 3;
  resultado = mult(a, b);
  printf("%i \n", resultado);
  return 0;
}
```

Modularidade em C

- Se uma função não retorna valor, deve ser declarada como **`void`**
- Se uma função não possui parâmetros, seus parênteses devem ficar vazios ou apresentar o termo **`void`**
- Cada **parâmetro** na definição da função deve ser precedido por seu **tipo**
- Ao chamar uma função, **não devemos passar os tipos dos parâmetros**, somente seus nomes

EDELWEISS, Nina. Algoritmos e programação com exemplos em Pascal e C. Porto Alegre Bookman 2014 1 recurso on-line (Livros didáticos UFRGS 23).

Modularidade em C

- Funções sem retorno de valores e sem parâmetros
  - `void nome_funcao() {...}`
  - `void nome_funcao(void) {...}`
- Função sem retorno de valores com dois parâmetros
  - `void nome_funcao(int param1, float param2) {...}`
- Função com retorno de valor inteiro e dois parâmetros
  - `int nome_funcao (float param1, int param2) {...}`

Na declaração da função, os parâmetros devem ser precedidos por seu **tipo** e ser separados por **vírgula**

EDELWEISS, Nina. Algoritmos e programação com exemplos em Pascal e C. Porto Alegre Bookman 2014 1 recurso on-line (Livros didáticos UFRGS 23).

Modularidade em C

- Para retorno de valores em uma função, utilizar **`return`**
  - `return <valor>;`
  - `return <variável>;`
  - `return <constante>;`
  - `return <expressão>;`
  - `return <função>;`

EDELWEISS, Nina. Algoritmos e programação com exemplos em Pascal e C. Porto Alegre Bookman 2014 1 recurso on-line (Livros didáticos UFRGS 23).

A função `main()`

- A função principal de um programa **`main()`** é uma função especial invocada automaticamente quando se inicia a execução do programa (executada antes de qualquer outra coisa)
- Seu tipo padrão é **int**
- O comando return que se coloca ao final do bloco da **main()** informa o Sistema Operacional (SO) se o programa terminou sua execução corretamente ou não
- Por convenção:
  - **return 0** indica ao SO que tudo correu bem
  - qualquer valor diferente de zero indica ao SO que houve erro no programa

Para main sem retorno, utilizar **`void main(){ ... }`**

Para praticar

Leia dois números inteiros *a* e *b* do teclado e imprima a sequência de valores entre esses números.

O programa deve ser executado até que o usuário decida parar.

Escreva uma função para leitura de *a* e *b* do teclado e um procedimento para exibir a sequência de números.

Para praticar - em C

A execução sempre inicia pela **main()**

```c
#include <stdio.h>

int lerPositivo() {
    int num;

    do {
        printf("Informe um numero positivo: ");
        scanf("%d", &num);
    } while (num < 0);

    return num;
}

void mostraNumeros(int nA, int nB) {
    int i;

    for (i = nA; i <= nB; i++) {
        printf("%d ", i);
    }
    printf("\n"); // somente quebra de linha
}
```

```c
int main() {
    char opcao;
    int nA, nB;

    do {
        nA = lerPositivo();
        nB = lerPositivo();

        mostraNumeros(nA, nB);

        printf("Deseja continuar? [S/N]: ");
        scanf(" %c", &opcao); // espaço no scanf de char
        printf("\n"); // somente para quebra de linha
    } while (opcao == 'S'); // aspas simples em char

    return 0;
}
```

Para praticar - em C

A execução sempre inicia pela **main()**

```c
#include <stdio.h>

int lerPositivo() {
    int num;

    do {
        printf("Informe um numero pos
        scanf("%d", &num);
    } while (num < 0);

    return num;
}

void mostraNumeros(int nA, int nB) {
    int i;

    for (i = nA; i <= nB; i++) {
        printf("%d ", i);
    }
    printf("\n"); // somente quebra d
}
```

```
> ./main
Informe um numero positivo: 3
Informe um numero positivo: 7
3 4 5 6 7
Deseja continuar? [S/N]: S

Informe um numero positivo: 5
Informe um numero positivo: 9
5 6 7 8 9
Deseja continuar? [S/N]: S

Informe um numero positivo: 1
Informe um numero positivo: 13
1 2 3 4 5 6 7 8 9 10 11 12 13
Deseja continuar? [S/N]: N

>
```

```c
int main() {
    char opcao;

rPositivo();
rPositivo();

umeros(nA, nB);

'Deseja continuar? [S/N]: ");
%c", &opcao); // espaço no scanf de char
'\n"); // somente para quebra de linha
cao == 'S'); // aspas simples em char
}
```

Para praticar

Leia três valores do teclado e imprima o maior valor dentre os três.

Escreva uma função para leitura dos valores do teclado e uma função para decidir qual dos números é maior.

Para praticar - em C

A execução sempre inicia pela **main()**

```c
#include <stdio.h>

int maiorValor(int a, int b, int c) {
    int maior;

    maior = a;
    if (b > maior) {
        maior = b;
    }

    if (c > maior) {
        maior = c;
    }

    return maior;
}
```

A variável **maior** foi declarada nesta função e é diferente da variável da função principal

```c
int lerPositivo() {
    int num;

    printf("Informe um numero positivo: ");
    scanf("%d", &num);

    return num;
}

int main() {
    int a, b, c, maior;

    a = lerPositivo();
    b = lerPositivo();
    c = lerPositivo();

    maior = maiorValor(a, b, c);

    printf("O maior valor e: %d", maior);

    return 0;
}
```

Para praticar - em C

A execução sempre inicia pela **main()**

```
> ./main
Informe um numero positivo: 2
Informe um numero positivo: 5
Informe um numero positivo: 7
O maior valor e: 7>
```

```
int c) {

    maior = a;
    if (b > maior) {
        maior = b;
    }

    if (c > maior) {
        maior = c;
    }

    return maior;
}
```

```
> ./main
Informe um numero positivo: 2
Informe um numero positivo: 7
Informe um numero positivo: 3
O maior valor e: 7>
```

```
int lerPositivo() {
    int num;

    printf("Informe um numero positivo: ");
    scanf("%d", &num);

    return num;
}


c, maior;

itivo();
itivo();
c = lerPositivo();

maior = maiorValor(a, b, c);

printf("O maior va

return 0;
}
```

```
> ./main
Informe um numero positivo: 8
Informe um numero positivo: 6
Informe um numero positivo: 2
O maior valor e: 8>
```

Modularidade

Avaliar o problema de forma diferenciada:

- Antes de iniciar o algoritmo, identifique quais são as tarefas a serem executadas
- Em seguida, identifique qual será o bloco principal de código, que vai controlar a chamada de todas as tarefas distribuídas em módulos
- Depois de definido o bloco principal, deve-se definir a funcionalidade de cada módulo
- O bloco principal “terceiriza” tarefas para os módulos

Modularidade no Portugol WebStudio

FUNÇÃO COM RETORNO VAZIO

```
programa {
  funcao vazio mult(inteiro a, inteiro b) {
    escreva(a * b)
  }
  funcao inicio() {
    inteiro a = 8, b = 3
    mult(a, b)
  }
}
```

FUNÇÃO COM RETORNO

```
programa {
  funcao inteiro mult(inteiro a, inteiro b) {
    retorne a * b
  }
  funcao inicio() {
    inteiro resultado, a = 8, b = 3
    resultado = mult(a, b)
    escreva(resultado)
  }
}
```

```
programa {
  funcao inteiro lerPositivo() {
    inteiro num
    faca {
      escreva("Informe um numero positivo: ")
      leia(num)
    } enquanto (num < 0)
    retorne num
  }

  funcao vazio mostraNumeros(inteiro nA, inteiro
  nB) {
    inteiro i
    para (i = nA; i <= nB; i++) {
      escreva(i, " ")
    }
    escreva("\n") // somente quebra de linha
  }
```

```
  funcao inicio() {
    caracter opcao
    inteiro nA, nB
    faca {
      nA = lerPositivo()
      nB = lerPositivo()
      mostraNumeros(nA, nB)
      escreva("Deseja continuar? [S/N]: ")
      leia(opcao) // espaço no scanf de char
      escreva("\n") // somente para quebra de linha
    } enquanto (opcao == 'S') // aspas simples em char
  }
}
```

```
programa {
  funcao inteiro lerPositivo() {
    inteiro num
    faca {
      escreva("Informe um nu
      leia(num)
    } enquanto (num < 0)
    retorne num
  }

  funcao vazio mostraNumeros
  nB) {
    inteiro i
    para (i = nA; i <= nB; i
      escreva(i, " ")
    }
    escreva("\n") // somente quebra de linha
  }
```

```
Informe um numero positivo: 1
Informe um numero positivo: 6
1 2 3 4 5 6
Deseja continuar? [S/N]: S

Informe um numero positivo: 2
Informe um numero positivo: 13
2 3 4 5 6 7 8 9 10 11 12 13
Deseja continuar? [S/N]: S

Informe um numero positivo: 6
Informe um numero positivo: 18
6 7 8 9 10 11 12 13 14 15 16 17 18
Deseja continuar? [S/N]: N


Programa finalizado. Tempo de execução: 21284 ms
```

```
  funcao inicio() {

    caracter opcao

                                    o()

                                    o()

                                    A, nB)

                                     continuar? [S/N]: ")

                                    paço no scanf de char

                                   ' somente para quebra de linha

                                     == 'S') // aspas simples em char
    }

}
```

A execução sempre inicia pela função principal

```
programa {
  funcao inteiro maiorValor(inteiro a, inteiro b,
  inteiro c) {
    inteiro maior
    maior = a
    se (b > maior) {
      maior = b
    }
    se (c > maior) {
      maior = c
    }
    retorne maior
  }
```

A variável **maior** foi declarada nesta função e é diferente da variável da função principal

```
  funcao inteiro lerPositivo() {
    inteiro num
    escreva("Informe um numero positivo: ")
    leia(num)
    retorne num
  }

  funcao inicio() {
    inteiro a, b, c, maior
    a = lerPositivo()
    b = lerPositivo()
    c = lerPositivo()
    maior = maiorValor(a, b, c)
    escreva("O maior valor e: ", maior)
  }
}
```

A variável **maior** foi declarada aqui e é diferente da variável da função `maiorValor()`

A execução sempre inicia pela função principal

```
Informe um numero positivo: 4
Informe um numero positivo: 5
Informe um numero positivo: 8
O maior valor e: 8
Programa finalizado. Tempo de execução: 5651 ms
```

```
    maior = a
    se (b > maior) {
      maior = b
    }
    se (c > maior) {
      maior = c
    }
    retorne maior
  }
```

A variável **maior** foi declarada nesta função e é diferente da variável da função principal

```
Informe um numero positivo: 2
Informe um numero positivo: 9
Informe um numero positivo: 1
O maior valor e: 9
Programa finalizado. Tempo de execução: 2590 ms
```

```
funcao inteiro lerPositivo() {
  inteiro num
  escreva("Informe um numero positivo: ")
  leia(num)
  retorne num
}
```

```
    c = lerPositivo()
    maior = maiorValor(a, b, c)
    escreva("O maior valor e: ", maior)
```

```
Informe um numero positivo: 6
Informe um numero positivo: 3
Informe um numero positivo: 1
O maior valor e: 6
Programa finalizado. Tempo de execução: 2447 ms
```

Para praticar

Escreva uma função chamada pesoIdeal() que recebe como parâmetros a altura e o sexo de uma pessoa (M masculino e F feminino) e retorna o seu peso ideal, utilizando as seguintes fórmulas:

- para sexo masculino: (72.7 * altura) – 58
- para sexo feminino: (62.1 * altura) – 44.7

Faça também o programa principal que recebe do usuário a sua altura, peso atual e sexo, e imprima na tela mensagem dizendo apenas se o usuário está acima ou abaixo de seu peso ideal, este calculado pela função pesoIdeal().

Próxima aula

- Retomada dos principais pontos desta aula
- Escopo de variáveis
- Passagem de parâmetros por valor e por referência
- Continuidade da introdução à Linguagem C

Universidade Positivo

Algoritmos de Programação

Aula 21

Prof.ª Mariane Cassenote

mariane.cassenote@up.edu.br