Paradigma Imperativo

Subprogramas

ERICK GALANI MAZIERO
erick.maziero@ufla.br

Departamento de Ciências da Computação
Universidade Federal da Lavras

UNIVERSIDADE FEDERAL DE LAVRAS

5 Subprogramas

Modularização

- Dividir um problema em problemas menores, que interagem entre si.
- Facilitar a solução de problemas complexos.
- Promover o reuso de código.
- Facilitar a manutenção e aumentar a legibilidade.
- Facilitar a realização de testes e detecção de erros.

Subprogramas

- Ponto único de entrada.
- Podem apenas desviar o fluxo de execução (procedimentos) ou devolver um valor (funções).
- Comunicam-se com o código chamador através de parâmetros ou argumentos.
- O chamador tem sua execução suspensa durante a execução do subprograma.

Subprogramas

- O controle retorna ao chamador quando a execução do subprograma se encerra.

Alternativas

- Corotinas
- unidades concorrentes

Subprogramas - Python

- Usa-se a palavra reservada `def`

```
def subprograma(a, b, c):
    print(a + b + c)
```

Parâmetros

Argumento: valor passado ao subprograma.

Parâmetro de chamada (reais): expressão que produz o argumento.

Parâmetro formal: identificador do argumento no interior do subprograma.

Vinculação entre parâmetros reais e formais: ordem dos parâmetros formais.

```
int Foo (int x, int y)
Foo(4,9)
```

Parâmetros

Tipo de passagem

Por valor

- Apenas o valor é passado para a subrotina;
- Após a execução da subrotina, a variável externa a subrotina utilizada para passar como parâmetro não possui seu valor alterado.

Parâmetros

Tipo de passagem

Por referência

- Neste caso o endereço de memória da variável é passada ao subprograma, e não apenas seu valor;
- Ao final da execução da subrotina o valor do parâmetro é alterado, uma vez que foi atribuído ao endereço que referencia a variável

Sobrecarga de subprogramas

Algumas LPs, como C++, C#, Java e Ada, permitem que o desenvolvedor utilize vários subprogramas diferentes com o mesmo nome, desde que os parâmetros utilizados sejam diferentes:

```
int foo(int x);
float foo(float f);
int foo(char c);
float foo(float f, char c, int b);
```

Sobrecarga de subprogramas

O polimorfismo introduzido pela sobrecarga pode ser ampliado com o uso de tipos genéricos. Em C++, por exemplo, pode-se usar para qualquer tipo que suporte comparação com >:

```
template <class Type> Type max(Type first, Type
second) {
      if (first > second)
            return first;
      else
            return second;
}
```

O *template* ao lado em C++ permite: o uso da função max para tipos como *int e float sem a necessidade de definir duas funções diferentes*

Subprogramas genéricos

- Modo de aumentar a reusabilidade de software

- *Polimorfismo ad hoc:* subprogramas sobrecarregados não precisam ter o mesmo código
- *Polimorfismo paramétrico:* subprograma que recebe parâmetros genéricos
  - **Subprogramas genéricos**
  - Funcionará desde que os operadores utilizados no subprograma estejam definidos para os parâmetros recebidos

Recursão

- Caracterizada por funções que se auto referenciam.
- Implementações recursivas tendem a ser mais “caras”.
- Na maioria das linguagens atuais cada nova chamada é registrada em uma pilha de execuções.
- Recursão de cauda tenta minimizar os custos calculando os valores ao longo das chamadas e não no retorno da função.
  - Dessa forma, ao final das chamadas, o valor a ser produzido já estará calculado.

Recursão

Estratégias:

- Dividir o problema em problemas menores
- Resolver os problemas menores
  - um ou mais casos base, em que nenhuma recursão é necessária.
- Combinar as soluções para chegar à solução final

Recursão mútua

Recursividade mútua:

- quando duas ou mais funções são definidas em termos uma da outra.

Ex:

Um número é ímpar se seu antecessor é par

Um número negativo é par se seu oposto for par

Zero deve ser considerado par

Recursão x Iteração

- Recursão tende a ser um pouco mais lenta por ser necessário registrar o estado atual do sistema para que a execução possa continuar de onde parou após a conclusão de cada nova chamada.
- Outro fator que beneficia implementações iterativas é o espaço destinado ao heap, que na maioria dos casos é maior que o espaço destinado ao fluxo de controle.

Exemplo: Recursão

```
int fatorial(int n) {
 if (n <= 1)
  return 1;
 else
  return (n * fatorial(n - 1))
}
```

chamada recursiva

Exemplo: Recursão (Python)

```
def fatorial(n):
 if (n <= 1):
    return 1
 else:
    return (n * fatorial(n - 1))
```

chamada recursiva

Blocos

Linguagens, como algumas baseadas em C, fornecem escopo locais, especificado pelos usuários:

```
{
 int temp;
 temp = list[upper];
 list[upper] = list[lower];
 list[lower] = temp;
}
```

a variável inteira *temp* estará disponível apenas nesse bloco, delimitado por *{ }*

Referência Bibliográfica

Sebesta, R. W. (2011). *Conceitos de Linguagens de Programação*. 9 ed. Bookman.

Capítulos 9 e 10