Universidade Federal do Rio Grande do Sul
Departamento de Informática Aplicada
INF01121 – Modelos de Linguagem de Programação

Aula 4 – Abstrações de Controle

(subprogramas, funções, procedimentos, passagem de parâmetros, implementação)

Prof. Lucas Mello Schnorr

8 de Agosto de 2012

Plano da aula de hoje

Introdução

- Conceito de abstração
  - Nome $\rightarrow$ parte do programa potencialmente complexo
  - Grau de separação entre
    - Linguagem de programação
    - Arquitetura do computador
  - Reduzir os detalhes, diminuir a complexidade

Introdução

- Conceito de abstração
  - Nome $\rightarrow$ parte do programa potencialmente complexo
  - Grau de separação entre
    - Linguagem de programação
    - Arquitetura do computador
  - Reduzir os detalhes, diminuir a complexidade

Existem dois tipos de abstração

Abstração de Controle	Abstração de Dados
(operações e ações)	(representar informações)

Introdução

Abstração de Controle $\rightarrow$ Subprogramas

- Separa-se as operações repetitivas do programa
- Podem ser chamadas em diferentes momentos

Introdução

Abstração de Controle $\rightarrow$ Subprogramas

- Separa-se as operações repetitivas do programa
- Podem ser chamadas em diferentes momentos

Vantagens

- Permite economizar memória
- Diminui o tempo de codificação
- Aumenta a legibilidade do código

Subprogramas

Subprogramas

Características gerais

- Ponto único de entrada
- Unidade de programa chamadora é suspensa
  $\rightarrow$ fica aguardando o resultado do subprograma
- Controle retorna ao chamador
  $\rightarrow$ Ao final da execução do subprograma

Subprogramas

Características gerais

- Ponto único de entrada
- Unidade de programa chamadora é suspensa
  $\rightarrow$ fica aguardando o resultado do subprograma
- Controle retorna ao chamador
  $\rightarrow$ Ao final da execução do subprograma

Tipos de subprogramas

Função	Procedimento
(retorna um valor)	(não retorna um valor)

Subprogramas

Alguns conceitos

- **Definição de um subprograma**
  $\rightarrow$ interface de acesso e ações
- **Chamada de um subprograma**
  $\rightarrow$ solicitação explícita de sua execução
- **Ativo** ou não

Subprogramas

- **Cabeçalho** (primeira linha da definição)
  - Tipo (função ou procedimento)
  - Nome do subprograma
  - Parâmetros (que são opcionais)

Fortran	`Subroutine Soma (parametros)`
Ada	`procedure Soma (parametros)`
Python	`def soma (parametros):`
C	`void soma (int a, int b);`
Objective-C	`- (void) somaVariable: (int) a comVariavel: (int) b;`

Subprogramas

- **Cabeçalho** (primeira linha da definição)
  - Tipo (função ou procedimento)
  - Nome do subprograma
  - Parâmetros (que são opcionais)

Fortran	`Subroutine Soma (parametros)`
Ada	`procedure Soma (parametros)`
Python	`def soma (parametros):`
C	`void soma (int a, int b);`
Objective-C	`- (void) somaVariable: (int) a comVariavel: (int) b;`

- **Perfil dos parâmetros**
  - Quantidade, ordem e os tipos dos parâmetros
- **Protocolo** de um subprograma

  $\rightarrow$ Perfil dos parâmetros mais o tipo do retorno

  $\rightarrow$ C (conhecido como protótipo), Objective-C

Implementação de Subprogramas

- Forma mais comum: pilha
  - Cada subprograma tem um espaço reservado na pilha
  - $\rightarrow$ Frame

Implementação de Subprogramas

- Forma mais comum: pilha
  - Cada subprograma tem um espaço reservado na pilha
  - $\rightarrow$ Frame
- **Chamada**
  - Salvar o estado atual
  - Realizar o processo de passagem de parâmetros
  - Registrar o endereço de retorno para o chamado
  - Trasferir o controle
- **Retorno**
  - Salvar o valor de retorno (se função)
  - Transferir o controle de volta ao chamador

Implementação de Subprogramas

- Forma mais comum: pilha
  - Cada subprograma tem um espaço reservado na pilha
  - $\rightarrow$ Frame
- **Chamada**
  - Salvar o estado atual
  - Realizar o processo de passagem de parâmetros
  - Registrar o endereço de retorno para o chamado
  - Trasferir o controle
- **Retorno**
  - Salvar o valor de retorno (se função)
  - Transferir o controle de volta ao chamador
- Recursividade: problema de "estouro" (Overflow) de pilha

Implementação de Subprogramas

- Forma mais comum: pilha
  - Cada subprograma tem um espaço reservado na pilha
  - $\rightarrow$ Frame
- **Chamada**
  - Salvar o estado atual
  - Realizar o processo de passagem de parâmetros
  - Registrar o endereço de retorno para o chamado
  - Trasferir o controle
- **Retorno**
  - Salvar o valor de retorno (se função)
  - Transferir o controle de volta ao chamador

- Recursividade: problema de "estouro" (Overflow) de pilha
  - GCC 1.4.6 $\rightarrow$ ***SplitStacks***

1. Pilha sempre tem espaço: uma Área de guarda
2. Função verifica se o *frame* cabe na área de guarda
3. Se maior $\rightarrow$ expansão do tamanho da pilha

Parâmetros

Parâmetros

Duas maneiras para acessar dados a serem processados

- Acesso direto a variáveis não-locais
  $\rightarrow$ em geral são variáveis de escopo global
- Passagem de parâmetros ao subprograma

Parâmetros

Duas maneiras para acessar dados a serem processados

- Acesso direto a variáveis não-locais
  $\rightarrow$ em geral são variáveis de escopo global
- Passagem de parâmetros ao subprograma

- Maior parte dos subprogramas tem parâmetros
  - Controlam parte do seu comportamento
  - Especificam dados sobre os quais operar

Parâmetros

Conceitos

- **Parâmetros formais**
  - Nomes dos parâmetros na declaração de um subprograma
- **Parâmetros reais**
  - Argumentos passados no momento da chamada

Parâmetros

Conceitos

- **Parâmetros formais**
  - Nomes dos parâmetros na declaração de um subprograma
- **Parâmetros reais**
  - Argumentos passados no momento da chamada

Correspondência entre parâmetros formais e reais

- Possibilidades
  - Pela posição (**posicionais**)
    $\rightarrow$ útil quando existem poucos parâmetros
    Ex.: C
  - Através de nomes (**nomeados**)
    $\rightarrow$ quando a lista de parâmetros é longa

Parâmetros

Conceitos

- **Parâmetros formais**
  - Nomes dos parâmetros na declaração de um subprograma
- **Parâmetros reais**
  - Argumentos passados no momento da chamada

Correspondência entre parâmetros formais e reais

- Possibilidades
  - Pela posição (**posicionais**)
    $\rightarrow$ útil quando existem poucos parâmetros
    Ex.: C
  - Através de nomes (**nomeados**)
    $\rightarrow$ quando a lista de parâmetros é longa
  - Correspondência **mista**
    Ex.: Ada e Fortran90

Parâmetros

Conceitos

- **Parâmetros formais**
  - Nomes dos parâmetros na declaração de um subprograma
- **Parâmetros reais**
  - Argumentos passados no momento da chamada

Correspondência entre parâmetros formais e reais

- Possibilidades
  - Pela posição (**posicionais**)
    $\rightarrow$ útil quando existem poucos parâmetros
    Ex.: C
  - Através de nomes (**nomeados**)
    $\rightarrow$ quando a lista de parâmetros é longa
  - Correspondência **mista**
    Ex.: Ada e Fortran90
- Valores-padrão para parâmetros formais
  Ex.: Ada, Fortran90 – C++ (neste devem aparecer por último)

Parâmetros – Alguns exemplos

- Parâmetros nomeados (em Python)

```
def soma(lista, inicio, fim):
...
soma(inicio = 1, fim = 2, lista = [4, 5, 6])
soma([4, 5, 6], fim = 1, inicio = 2)
```

Parâmetros – Alguns exemplos

- Parâmetros nomeados (em Python)

```
def soma(lista, inicio, fim):
...
soma(inicio = 1, fim = 2, lista = [4, 5, 6])
soma([4, 5, 6], fim = 1, inicio = 2)
```

- Parâmetros com valores-padrão (em Python)

```
def compute_pay(income, exemptions = 1, tax_rate):
...
pay = compute_pay(20000.0, tax_rate = 0.15)
```

Parâmetros – Alguns exemplos

- Parâmetros nomeados (em Python)

```
def soma(lista, inicio, fim):
...
soma(inicio = 1, fim = 2, lista = [4, 5, 6])
soma([4, 5, 6], fim = 1, inicio = 2)
```

- Parâmetros com valores-padrão (em Python)

```
def compute_pay(income, exemptions = 1, tax_rate):
...
pay = compute_pay(20000.0, tax_rate = 0.15)
```

- Parâmetros com valores-padrão (em C++)

```
float compute_pay(float income,
                  float tax_rate,
                  int exemptions = 1) { ... }
...
pay = compute_pay(20000.0, 0.15);
```

Procedimentos e Funções

Procedimentos e Funções

Conceitos dos dois tipos de subprogramas

- Procedimentos
  $\rightarrow$ coleções de instruções para computação parametrizada
  - Atuar sobre variáveis globais
  - E também sobre parâmetros formais
    $\rightarrow$ se permitirem a transferência dos dados
  - Não retornam um resultado

Procedimentos e Funções

Conceitos dos dois tipos de subprogramas

- Procedimentos
  → coleções de instruções para computação parametrizada
  - Atuar sobre variáveis globais
  - E também sobre parâmetros formais
    → se permitirem a transferência dos dados
  - Não retornam um resultado
- Funções
  → modeladas como funções matemáticas
  - Retorna um único valor como efeito desejado
  - Se não modifica nem parâmetros nem globais
    → funções confiáveis, puras – sem **efeito colateral**

Métodos de Passagem de Parâmetros

Métodos de Passagem de Parâmetros

- Maneira pelas quais se transmitem parâmetros
  - Deve ser escolhida no projeto da linguagem
  - Métodos semânticos e de implementação

Métodos de Passagem de Parâmetros

- Maneira pelas quais se transmitem parâmetros
  - Deve ser escolhida no projeto da linguagem
  - Métodos semânticos e de implementação

Três modelos semânticos

- Entrada
- Saída
- Entrada/Saída

Dois modelos conceituais

- Valor real é copiado
- Um caminho de acesso é transmitido

Métodos de Passagem de Parâmetros

- Maneira pelas quais se transmitem parâmetros
  - Deve ser escolhida no projeto da linguagem
  - Métodos semânticos e de implementação

Três modelos semânticos

- Entrada
- Saída
- Entrada/Saída

Dois modelos conceituais

- Valor real é copiado
- Um caminho de acesso é transmitido

Várias implementações $\rightarrow$ veremos as mais importantes

Passagem por valor

- Valor do parâmetro real $\rightarrow$ Inicializa o parâmetro formal
  - Se comporta como variável de escopo local
  - Semântica de modo entrada
  - Transferência de dados real
- Exemplo em linguagem C

Passagem por valor

- Valor do parâmetro real $\rightarrow$ Inicializa o parâmetro formal
  - Se comporta como variável de escopo local
  - Semântica de modo entrada
  - Transferência de dados real
- Exemplo em linguagem C

- Vantagem – acessos mais eficientes
- Desvantagem – memória extra e tempo de cópia
  $\rightarrow$ se os dados foram volumosos

Passagem por resultado

- Nada é transmitido (zero entrada)
- No final da execução do subprograma
  - Valor do parâmetro formal $\rightarrow$ atribuído ao parâmetro real
- Semântica de modo saída

Passagem por resultado

- Nada é transmitido (zero entrada)
- No final da execução do subprograma
  - Valor do parâmetro formal $\rightarrow$ atribuído ao parâmetro real
- Semântica de modo saída

- Desvantagem $\rightarrow$ colisão de parâmetros reais
  $sub(p1, p1)$
- Nomes dos parâmetros formais foram diferentes
  - O resultado depende da última atribuíção
  - Problemas de portabilidade difícies de diagnosticar

Passagem por valor/resultado

- Também conhecido como passagem por cópia
- Modelo semântico de Entrada/Saída
- Na Entrada
  - Valor do parâmetro real $\rightarrow$ Inicializa o parâmetro formal
- Na Saída
  - Valor do parâmetro formal $\rightarrow$ atribuído ao parâmetro real

Passagem por valor/resultado

- Também conhecido como passagem por cópia
- Modelo semântico de Entrada/Saída
- Na Entrada
  - Valor do parâmetro real $\rightarrow$ Inicializa o parâmetro formal
- Na Saída
  - Valor do parâmetro formal $\rightarrow$ atribuído ao parâmetro real

- Combina passagem por valor e por resultado
  - As desvantagens da passagem por valor $\rightarrow$ quando os dados são grandes
  - Os problemas da passagem por resultado $\rightarrow$ Colisão de valores

Passagem por referência

- Modelo semântico de Entrada/Saída
- Um caminho de acesso é transmitido (endereço)

- Vantagens – eficiente em tempo e espaço
- Desvantanges – indireção

Passagem por referência

- Subprogramas como referência
  - Endereço de função em C, C++
  - `@selector` em Objective-C

Passagem por referência

- Subprogramas como referência
  - Endereço de função em C, C++
  - `@selector` em Objective-C
- Linguagens funcionais
  - Funções de ordem superior – (Higher-Order Functions)
    - Recebem um ou mais funções como parâmetro
    - Retornam uma função

Conclusão

- Abstrações de controle
- Principal mecanismo: subprogramas
- Métodos de passagem de parâmetros

Conclusão

- Abstrações de controle
- Principal mecanismo: subprogramas
- Métodos de passagem de parâmetros

- Lista de exercícios teóricos
- Moodle institucional
  - Slides da aula de hoje e das aulas anteriores
  - $\rightarrow$ Exercícios práticos para a próxima aula

Conclusão

- Abstrações de controle
- Principal mecanismo: subprogramas
- Métodos de passagem de parâmetros

- Lista de exercícios teóricos
- Moodle institucional
  - Slides da aula de hoje e das aulas anteriores
  - $\rightarrow$ Exercícios práticos para a próxima aula
- Não esqueçam do trabalho $\rightarrow$ data limite é 15/Agosto
- Segundo trabalho será especificado na semana que vem

Conclusão

- Abstrações de controle
- Principal mecanismo: subprogramas
- Métodos de passagem de parâmetros

- Lista de exercícios teóricos
- Moodle institucional
  - Slides da aula de hoje e das aulas anteriores
  - $\rightarrow$ Exercícios práticos para a próxima aula
- Não esqueçam do trabalho $\rightarrow$ data limite é 15/Agosto
- Segundo trabalho será especificado na semana que vem
- Primeira semana de setembro

Bibliografia e Referências

Principal

- Conceitos de Linguagens de Programação. Robert W. Sebesta. Quinta edição. Bookman.
- Programming Language Pragmatics. Michael L. Scott. Morgan Kaufmann.

Adicionais

- Programming Language Concepts. Carlo Ghezzi, Mehdi Jazayeri. John Wiley.
- Programming Languages: Design and Implementation. Terrence W. Pratt. Terceira edição. Prentice-Hall.
- Linguagens de Programação: Conceitos e Técnicas - Flavio Miguel Varejao. Editora Campus.