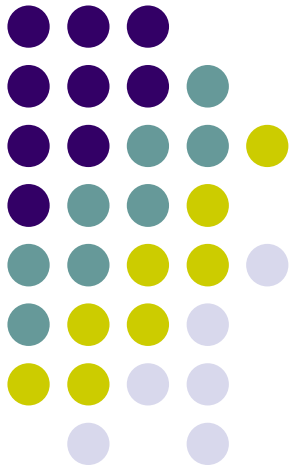
Paradigmas de Linguagens de Programação

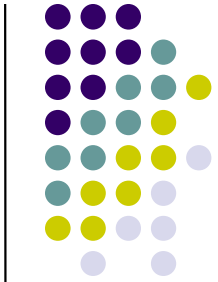
02: Estruturas de Controle



Prof. Márcio Puntel

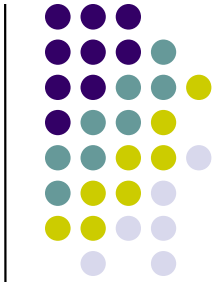
Marcio.puntel@ulbra.edu.br

Programação em Bloco Monolítico



- Inviabiliza grandes sistemas
- Um único programador, pois não há divisão do programa
- Indução a erros por causa da visibilidade de variáveis e fluxo de controle irrestrito
- Dificulta a reutilização do código
- Ineficiência de programação

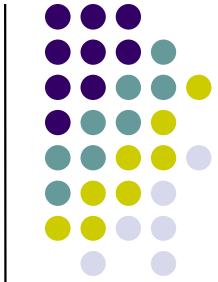
Processo de Resolução de Problemas Complexos



- Resolução de vários problemas menos complexos
- Aumento das possibilidades de reutilização
- Facilita o entendimento do programa
- Segmentação do programa
- Encapsulamento dos dados: agrupamento de dados e processos logicamente relacionados

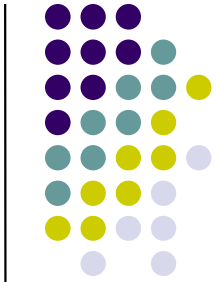
Dividir para conquistar...

Sistemas de Grande Porte



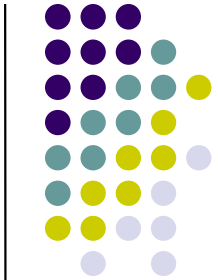
- Características
  - Grande número de entidades de computação e linhas de código
  - Equipe de programadores
  - Código distribuído em vários arquivos fonte
  - Conveniente não recompilar partes não alteradas do programa

Sistemas de Grande Porte



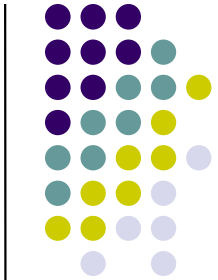
- Subdivisão em Módulos
  - Unidade que pode ser compilada separadamente
  - Propósito único
  - Interface apropriada com outros módulos
  - Reutilizáveis e Modificáveis
  - Pode conter um ou mais tipos, variáveis, constantes, funções, procedimentos
  - Deve identificar claramente seu objetivo e como o atinge

Abstração



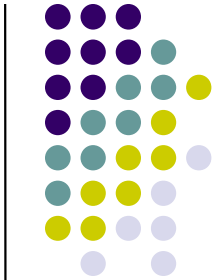
- Fundamental para a Modularização
- Seleção do que deve ser representado
- Possibilita a divisão do trabalho em níveis
- Exemplos de uso na computação
  - Comandos do SO
  - Linguagens de Programação
    - LP é abstração sobre o hardware
    - LP oferece mecanismos para criar abstrações

Abstração e Modularização



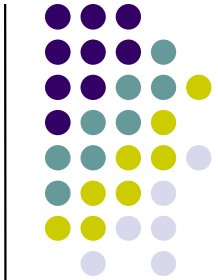
- Foco na distinção entre
  - O que uma parte do programa faz
    - foco do programador que usa a abstração
  - Como isso é implementado
    - Foco do programador que implementa a abstração

Tipos de Abstrações



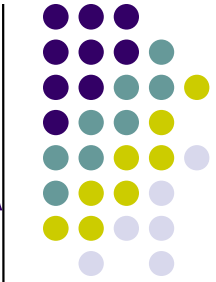
- Abstrações de Processos
  - Abstrações sobre o fluxo de controle do programa
  - Suprogramas – funções da biblioteca padrão do Pascal
- Abstrações de Dados
  - Abstrações sobre as estruturas de dados do programa
  - Tipos de Dados – tipos da biblioteca padrão do Pascal

Abstrações de Processos



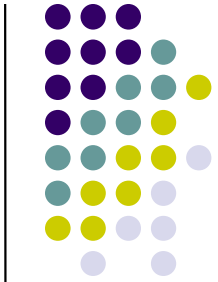
- Subprogramas
  - Permitem segmentar o programa em vários blocos logicamente relacionados
  - Servem para reusar trechos de código que operam sobre dados diferenciados
  - Modularizações efetuadas com base no tamanho do código possuem baixa qualidade
  - Propósito único e claro facilita legibilidade, depuração, manutenção e reutilização

Perspectivas do Usuário e do Implementador do Subprograma



- Usuário
  - Interessa o que o subprograma faz
  - Interessa como se usa o subprograma
  - Como faz é pouco importante ou não é importante
- Implementador
  - O que realmente importanta é como implementar a funcionalidade exigida pelo subprograma

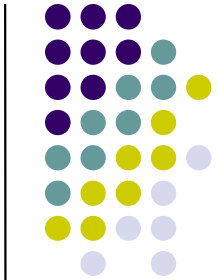
Perspectivas do Usuário e do Implementador Sobre Função



```
int fatorial(int n) {
   if (n<2) {
      return 1;
   } else {
      return n * fatorial (n – 1);
   }
}
```

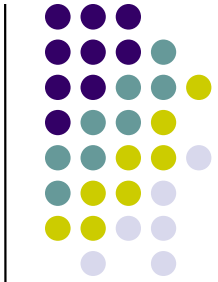
- Usuário
  - Função fatorial é mapeamento de n para n!
- Implementador
  - Uso de algoritmo recursivo

Parâmetros



- A ausência reduz:
  - A Legibilidada: pois os valores devem ser passados através de variáveis globais
  - A Redigibilidade: torna-se é necessário incluir operações para atribuir os valores desejados às variáveis globais
  - A Confiabilidade: pois é preciso atribuir valores as variáveis globais a cada chamada ao procedimento

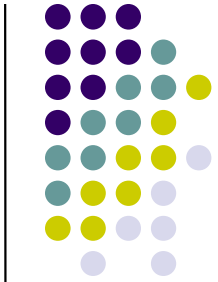
Parâmetros, exemplo de uso



```
int volume () {
  return a * l * c;
}
main() {
   int a = 1, l = 2, c = 3;
   int v1, v2;
   v1 = volume(a, l, c);
    a2 = 4, c2 = 5, l2 = 6
    v2 = volume(a, l, c);
}
```

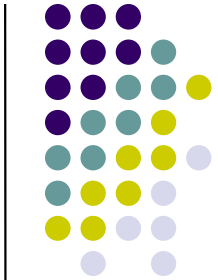
```
int volume (int a, int l, int c) {
   return a * l * c;
}
main() {
    int v1, v2;
    v1 = volume(1, 2, 3);
    v2 = volume(4, 5, 6);
}
```

Parâmetros Reais, Formais e Argumentos



- Parâmetro formal
  - Identificadores listados no cabeçalho do subprograma e usados no seu corpo
- Parâmetro real
  - Valores, identificadores ou expressões utilizados na chamada do subprograma
- Argumento
  - Valor passado do parâmetro real para o parâmetro formal

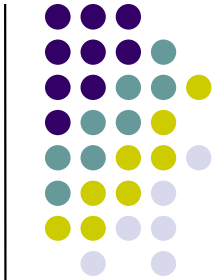
Parâmetros Reais, Formais e Argumentos



```
float area (float r) {
   return 3.1416 * r * r;
}
main() {
   float diametro, resultado;
   diametro = 2.8;
   resultado = area (diametro/2);
}
```

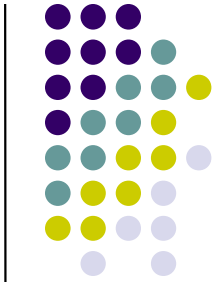
Parâmetro Formal: **r**
Parâmetro Real: **diametro/2**
Argumento: **1.4**

Valores Default de Parâmetros



```
int soma (int a[], int inicio = 0, int fim = 7, int incr = 1){
   int soma = 0;
   for (int i = inicio; i < fim; i+=incr) soma+=a[i];
   return soma;
}
main() {
   int [] pontuacao = { 9, 4, 8, 9, 5, 6, 2};
   int ptotal, pQuaSab, pTerQui, pSegQuaSex;
   ptotal = soma(pontuacao);
   pQuaSab = soma(pontuacao, 3);
   ...
}
```

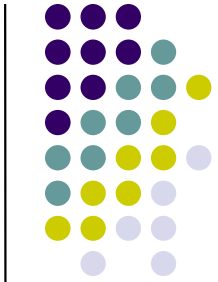
Número de Parâmetros Variável



```
main() {
   printf (“%d\n”, ou (1, 3 < 2));
   printf (“%d\n”, ou (2, 3 > 2, 7 > 5));
   printf (“%d\n”, ou (3, 1 != 1, 2 != 2, 3 != 3));
}
```

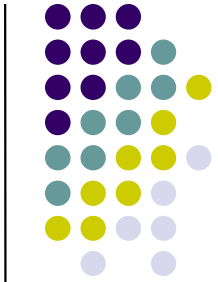
- Oferece maior flexibilidade à LP
- Reduz a confiabilidade pois não é possível verificar os tipos dos parâmetros em tempo de compilação

Tipos de Dados



- Forma de modularização usada para implementar abstrações de dados
- Agrupam dados correlacionados em uma entidade computacional
- Usuários enxergam o grupo de dados como um todo
  - Não se precisa saber como entidade é implementada ou armazenada

Tipos de Dados Simples

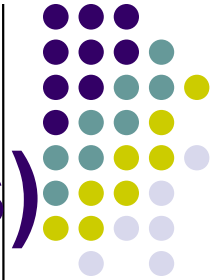


- Agrupam dados relacionados em uma única entidade nomeada
- Aumentam reusabilidade, redigibilidade, legibilidade e confiabilidade

```
typedef struct pilha {
   int elem[10];
   int topo;
} tPilha;
tPilha global;
```

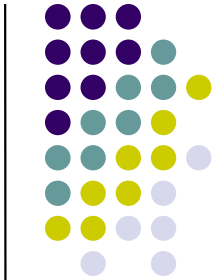
Não permite ocultar informações...

Tipos Abstratos de Dados (TADs)



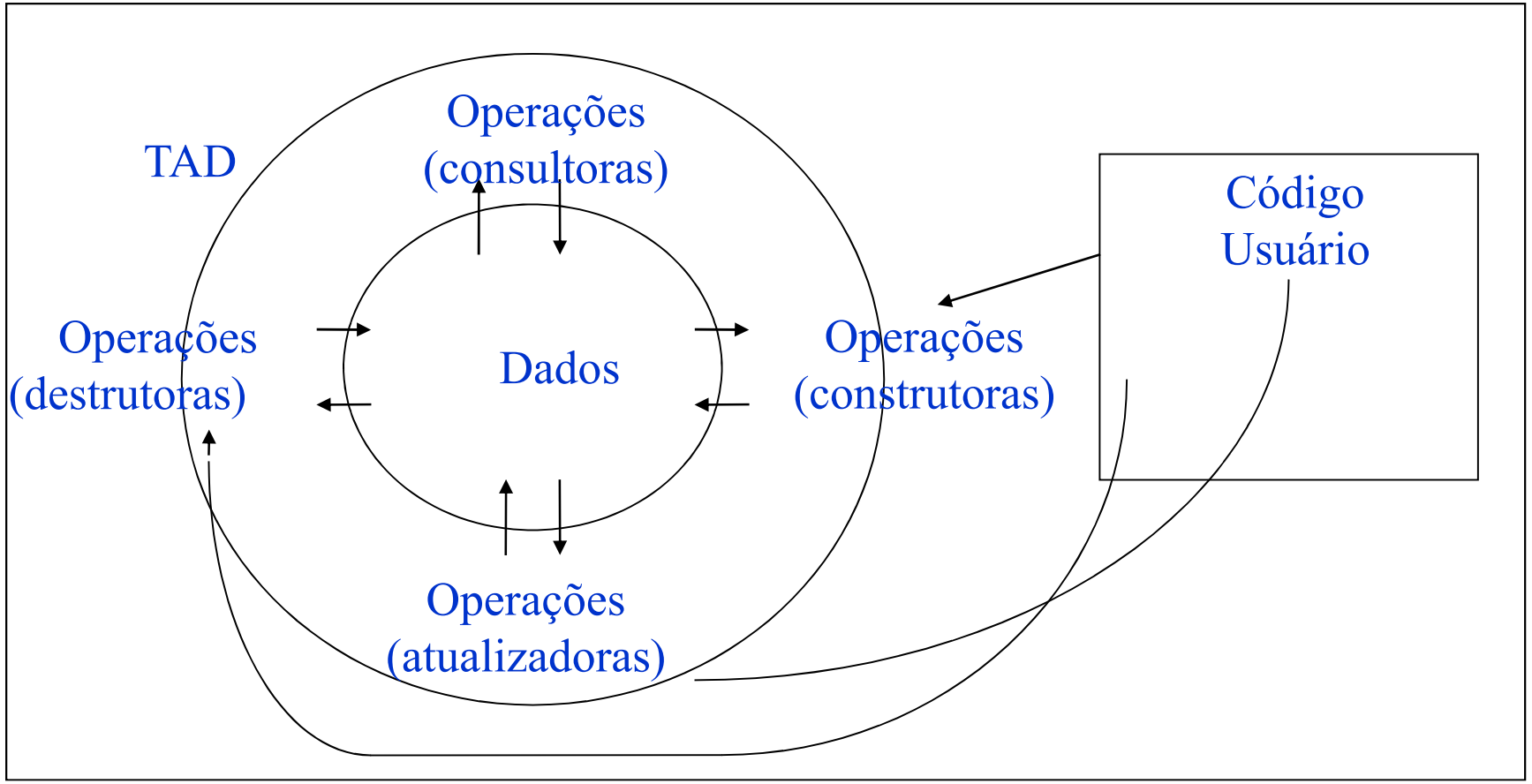
- Conjuntos de valores com comportamento uniforme definido por operações
- Em LPs, TADs possuem representação e operações especificadas pelo programador
- O usuário do TAD utiliza sua representação e operações como uma caixa preta
- Essencial haver ocultamento da informação para tornar invisível a implementação
- Interface são os componentes públicos do TAD (tipicamente, operações)

Tipos Abstratos de Dados

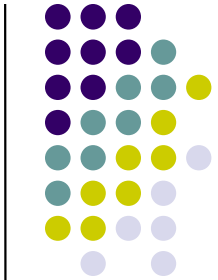


- Encapsulam e protegem os dados
- Resolvem os problemas existentes com tipos simples
- Quatro tipos diferentes de operações
  - Construtoras
  - Consultoras
  - Atualizadoras
  - Destrutoras

Tipos Abstratos de Dados



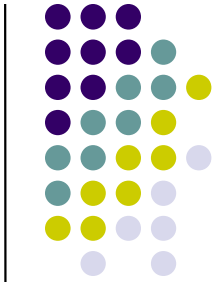
TADs Como Classes



- Estrutura de dados privada e operações da interface públicas
- Uso do operador de resolução de escopo
- Sintaxe diferenciada com parâmetro implícito na definição e chamada
- Podem haver vários construtores sempre chamados antes de qualquer outra operação

```
pilha = new tPilha;
```

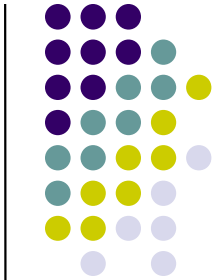
TADs Como Classes



- Atributo de classe
  - Compartilhado por todos objetos da classe
- Método de classe
  - Pode ser chamado independentemente da existência de um objeto da classe
  - Pode ser chamado pela própria classe ou pelos objetos da classe

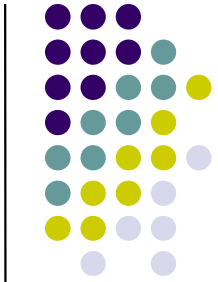
Permite ocultar informações...

Pacotes



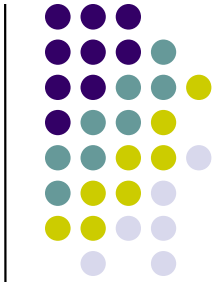
- Só subprogramas e tipos não são suficientes para sistemas de grande porte
  - Código com baixa granularidade
  - Possibilidade de conflito entre fontes de código
- Fontes de código são coleções de entidades reutilizáveis de computação
  - Bibliotecas ou Frameworks
  - Aplicações utilitárias
  - Aplicações completas

Pacotes



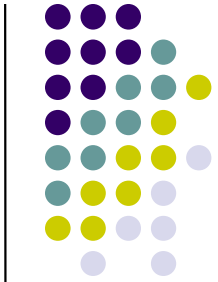
- Pacotes agrupam entidades de computação (exportáveis ou não)
- Usados para organizar as fontes de informação e para resolver os conflitos de nomes
- Por exemplo, o Delphi possui **packages** (geralmente contém um conjunto de componentes)

Modularização, Arquivos e Compilação Separada



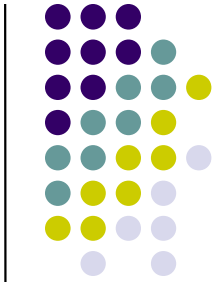
- Uso de arquivo único causa problemas
  - Redação e modificação se tornam mais difíceis
  - Reutilização apenas com processo de copiar e colar
- Divisão do código em arquivos separados com entidades relacionadas
  - Biblioteca de arquivos .c
  - Arquivos funcionam como índices
  - Reutilização através de inclusão
  - Necessidade de recompilação de todo o código

Vantagens da Modularização



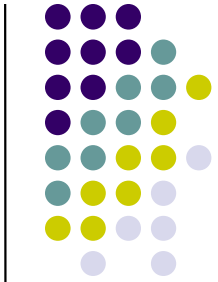
- Melhoria da legibilidade
  - Divisão lógica do programa em unidades funcionais
  - Separação do código de implementação do código de uso da abstração
- Aprimoramento da redigibilidade
  - Mais fácil escrever código em vários módulos do que em um módulo único
- Aumento da modificabilidade
  - Alteração no módulo é localizada e não impacta código usuário

Vantagens da Modularização



- Incremento da reusabilidade
  - Módulo pode ser usado sempre que sua funcionalidade é requerida
- Aumento da produtividade de programação
  - Compilação separada
  - Divisão em equipes
- Maior confiabilidade
  - Verificação independente e extensiva dos módulos antes do uso

Vantagens da Modularização



- Suporte a técnicas de desenvolvimento de software
  - Orientadas a funcionalidades (top-down)
    - Uso de subprogramas
  - Orientadas a dados (bottom-up)
    - Uso de tipos
  - Complementaridade dessas técnicas