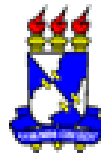


Persistência

Prof. Alberto Costa Neto
alberto@ufs.br

Linguagens de Programação



Departamento de Computação
Universidade Federal de Sergipe

Retrospectiva

- Quanto ao Tempo de Vida
 - Variáveis Estáticas
 - Variáveis *Stack*-Dinâmicas
 - Variáveis *Heap*-Dinâmicas
 - Variáveis Persistentes



Armazenamento de variáveis

- Transientes
 - Memória Principal
 - Pilha
 - Heap
- Persistentes
 - Memória Principal
 - Memória Secundária



Memória Principal

- Enorme sequência contígua e finita de bits
 - Vetor de tamanho finito com elementos do tamanho da palavra do computador
- Alocação de Variáveis e Constantes
 - Tempo de Carga (FORTRAN)
 - Super e subdimensionamento das variáveis
 - Variáveis locais alocadas desnecessariamente
 - Impedimento de uso de recursividade
 - Alocação Dinâmica Contígua no Vetor de Memória
 - Esgotamento rápido do vetor
 - Desalocação e realocação pouco eficientes



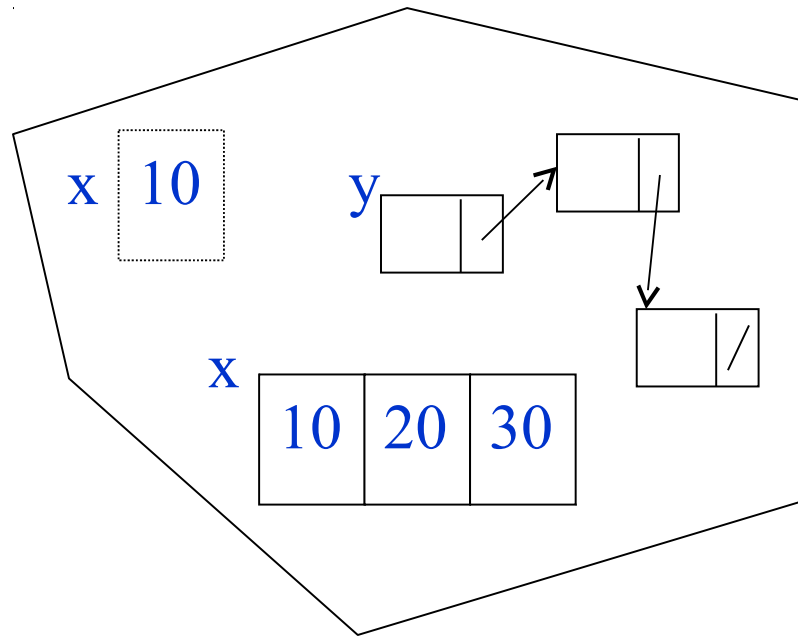
Pilha e Heap

- LPs ALGOL-like
- Pilha
 - Variáveis Locais
 - Parâmetros
 - Regra de alocação bem definida
- Heap
 - Variáveis de tamanho dinâmico
 - Regra de alocação indefinida



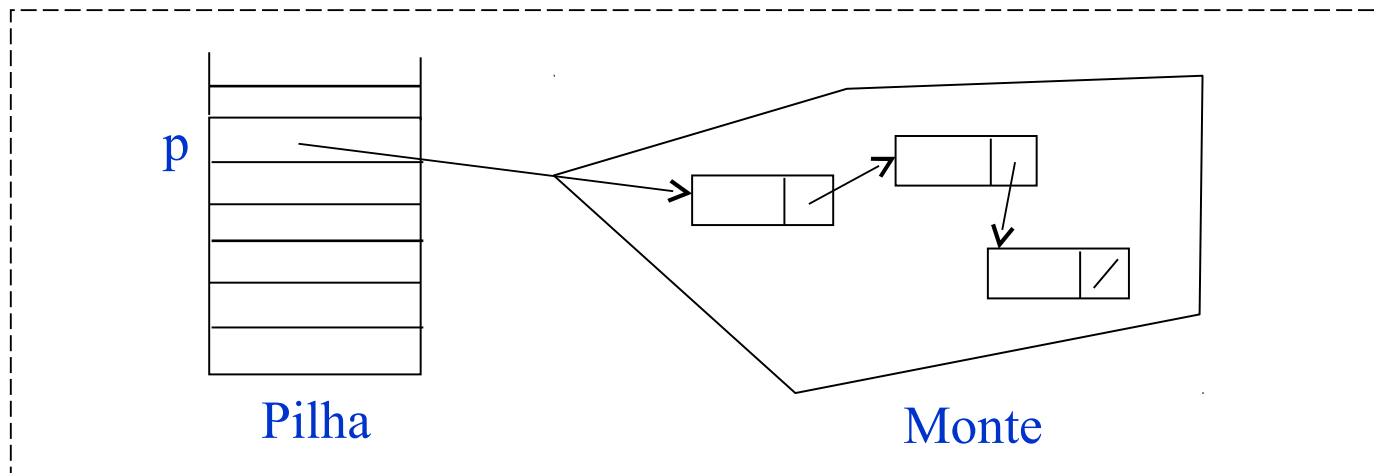
Pilha e Heap

f	z	10
	y	9
	x	10
p	b	9
	a	10



Pilha e Heap

- Porque não usar só o Monte (Heap)?
 - Menos Eficiente por causa da manutenção de:
 - Lista de Espaços Disponíveis (LED)
 - Lista de Espaços Ocupados (LEO)
- Ponteiro como Ligação entre Pilha e Heap



Gerenciamento da Pilha

- Uso de Registros de Ativação (RA)

constantes locais
variáveis locais
parâmetros
link dinâmico para quem o chamou
link estático para onde foi declarado
endereço de retorno

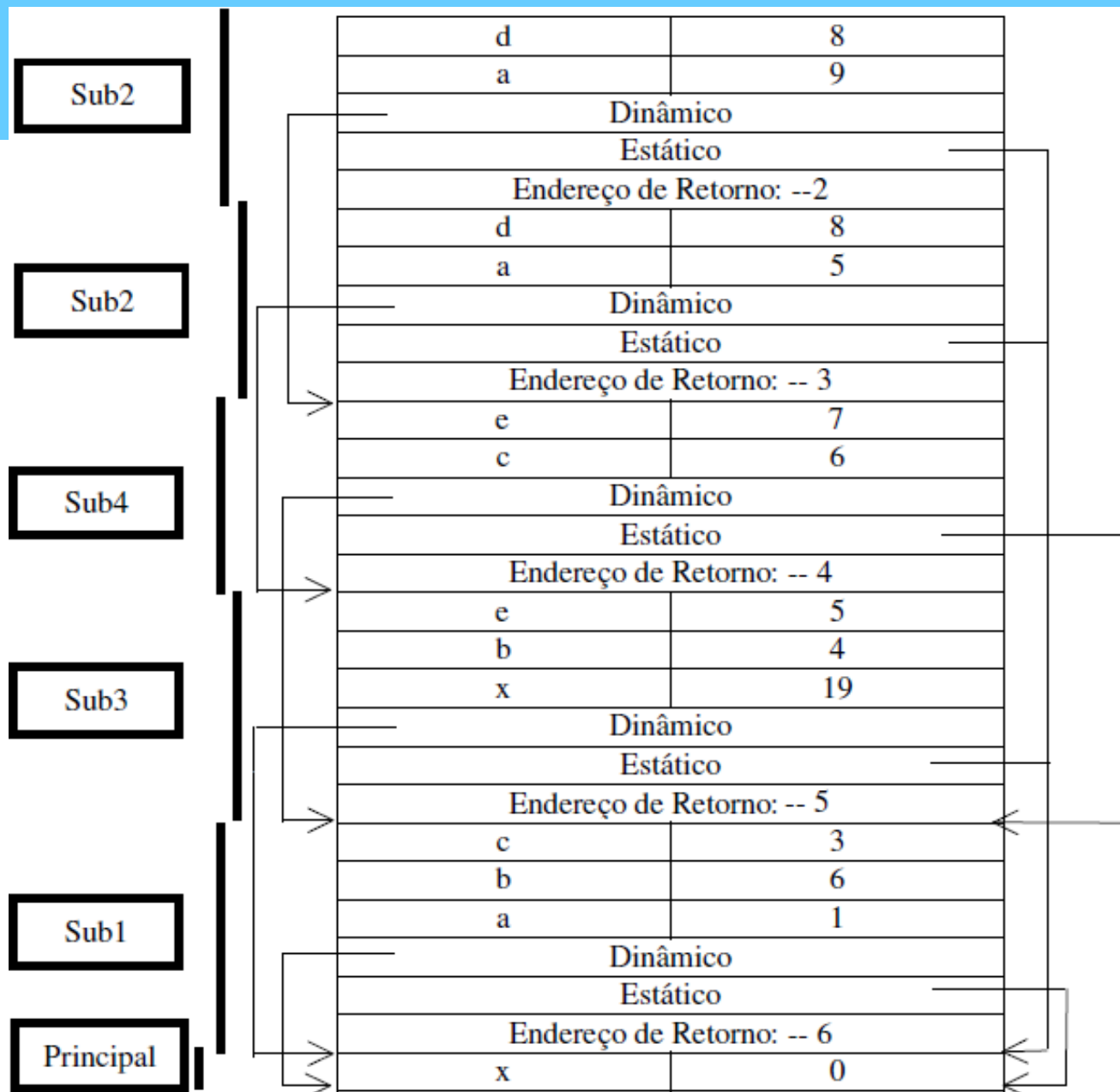


Gerenciamento da Pilha

```
procedure Principal is -- ADA
  x: integer;
  procedure Sub1 is
    a, b, c: integer;
    procedure Sub2 is
      a, d: integer;
    begin --Sub2
      a := b + c;
      d := 8; -- 1
      if a < d then
        b := 6;
        Sub2; -- 2
      end if;
    end Sub2;
  procedure Sub3 (x: integer) is
    b, e: integer;
```

```
    procedure Sub4 is
      c, e: integer;
    begin --Sub4
      c := 6; e:= 7; Sub2; -- 3
      e := b + a;
    end Sub4;
  begin --Sub3
    b := 4; e := 5; Sub4; -- 4
    b := a + b + e;
  end Sub3;
  begin --Sub1
    a := 1; b:= 2; c:= 3; Sub3(19); -- 5
  end Sub1;
begin --Principal
  x := 0; Sub1; -- 6
end Principal;
```





Gerenciamento do Heap

- LED e LEO
- Momento da Alocação sempre bem definido
- Momento da Desalocação definido por
 - Programador: mais flexível (podendo ser mais eficiente), menos confiável e mais trabalhoso
 - LP: implementação mais complexa, falta de controle sobre a desalocação
- Coletor de Lixo de JAVA torna alocação no heap quase tão eficiente quanto na pilha



Variáveis Persistentes

- Tempo de vida transcende a execução de um programa
- Oposto às variáveis transientes (estáticas, stack-dinâmicas, heap-dinâmicas)
- “Persistência de Dados”
 - Relacionado ao armazenamento e recuperação de dados em memória secundária
 - Exemplo: arquivos e banco de dados



Variáveis Persistentes

- Variáveis tipo Arquivo
 - Seriais ou Diretos: diferença na forma de acesso
 - Operações de Abertura e Fechamento
 - Conversão de dados em memória para formato binário
 - Exemplo em C:



Variáveis Persistentes

```
#include <stdio.h>
struct data { int d, m, a; };
struct data d = { 20, 04, 2011 }
struct data e;
void main() {
    FILE *p;
    char str[30];
    printf("\n\n Entre com o nome do arquivo:\n");
    gets(str);
    p = fopen(str, "w");
    fwrite (&d, sizeof(struct data), 1, p);
    fclose(p);
    p = fopen(str, "r");
    fread (&e, sizeof(struct data), 1, p);
    fclose (p);
    printf ("%d/%d/%d\n", e.a, e.m, e.d);
}
```

Variáveis transientes

Variável persistente

Copia cadeia de bytes da variável transiente d num arquivo binário

Faz o inverso



Variáveis Persistentes

- Através de Gerenciadores de BD
 - Valores das variáveis transientes precisam ser transformados em registros de tabelas em um BD utilizando comandos SQL
 - Nesse caso, a LP não oferece um tipo de dados persistente
 - O programador escreve sentenças compostas por termos SQL e por valores das variáveis transientes
 - O gerenciador executa as sentenças para armazenar e recuperar os valores das variáveis persistentes
 - Exemplo em Java/SQL:



Variáveis Persistentes

```
int idadeMaxima = 50; Variável transiente
Statement comando = conexao.createStatement();
ResultSet resultados = Variável persistente
    comando.executeQuery(
        "SELECT nome, idade, nascimento FROM pessoa " +
        "WHERE idade < " + idadeMaxima); Comando SQL
while (resultados.next()) {
    String nome = resultados.getString("nome");
    int idade = resultados.getInt("idade");
    Date nascimento = resultados.getDate("nascimento");
} Variáveis transientes
```



Persistência Ortogonal

- LPs normalmente fornecem diferentes tipos para variáveis persistentes e transientes
 - Viola o princípio da completude de tipo!
- Idealmente, todos os tipos deveriam ser permitidos tanto para variáveis transientes quanto para persistentes
- “Persistência Ortogonal”
 - Nenhuma distinção no código que lida com variáveis persistentes e o que lida com variáveis transientes
 - Eliminação de conversões de E/S (30% do código)
- Grande Desafio!
 - Integração de S.O.s, BDs, LPs



(Des)Serialização

- Aspectos ao se implementar persistência
 - Variável transiente deve ser convertida de sua representação na memória primária para uma sequência de bytes na memória secundária
 - Ponteiros devem ser relativizados quando armazenados
 - Variáveis anônimas apontadas também devem ser armazenadas e recuperadas
 - Na restauração de uma variável, os ponteiros devem ser ajustados de modo a respeitar as relações existentes anteriormente entre as variáveis anônimas
- Layout da variável na memória primária \leftrightarrow
Representação serial na memória secundária



(Des)Serialização

- Exemplos em LPs:
- C++ não oferece serialização: Programador tem que programar na “mão” processos de serialização e desserialização
- Java provê interface Serializable
 - Objetos que implementam Serializable podem ser transformados numa sequência de bytes e vice-versa
 - Objetos referenciados também são serializados
 - Mecanismo compensa automaticamente diferenças entre ambientes
 - Exemplo: programação em Rede (Windows $\xrightarrow{\text{Obj serializado}}$ Unix)



(Des)Serialização

```
import java.io.*;
public class Impares implements Serializable {
    private static int j = 1;
    private int i = j;
    private Impares prox;
    Impares(int n) {
        j = j + 2;
        if (--n > 0)
            prox = new Impares(n);
    }
    public String toString() {
        String s = "" + i;
        if (prox != null)
            s += " : " + prox.toString();
        return s;
    }
}
```

Representa uma lista com os
n primeiros números ímpares



(Des)Serialização

```
public static void main(String[] args) {  
    Impares w = new Impares(6);  
    System.out.println("w = " + w);  
    try {  
        ObjectOutputStream out =  
            new ObjectOutputStream(  
                new FileOutputStream("impares.txt"));  
        out.writeObject(w);  
        out.close();  
        ObjectInputStream in =  
            new ObjectInputStream(  
                new FileInputStream("impares.txt"));  
        Impares z = (Impares)in.readObject();  
        System.out.println("z = " + z);  
        in.close();  
    } catch (Exception e) { e.printStackTrace(); }  
}
```

lista com os 6 primeiros
números ímpares



Sugestões de Leitura

- Concepts of Programming Languages (Sebesta)
 - Seção 5.4.3 e, Capítulo 10
- Programming Language Concepts and Paradigms (David A. Watt)
 - Seção 3.4.3
- Linguagens de Programação (Flávio Varejão)
 - Seção 4.3

