

# Linguagens de Programação

## Conceitos e Técnicas



Variáveis e Constantes

Prof.

# Variáveis

- *"Uma vez que o programador tenha entendido o uso de variáveis, ele entendeu a essência da programação". [Dijkstra]*
- Abstração para uma ou mais células de memória responsáveis por armazenar o estado de uma entidade de computação

```
unsigned x;
```

```
x = 7;
```

```
x = x + 3;
```

# Variáveis

Passo 1 – Espaço de memória é alocado e associado à variável x

x	FF00	00000000
	FF01	00000000
	FF02	00000000
	FF03	00000000

Passo 2 – Valor 7 atribuído a x é colocado no espaço de memória associado a x

x	FF00	00000000
	FF01	00000111
	FF02	00000000
	FF03	00000000

Passo 3 – Valor corrente de x é somado a 3 e resultado é colocado no espaço de memória associado a x

x	FF00	00000000
	FF01	00001010
	FF02	00000000
	FF03	00000000

# Propriedades de Variáveis

## ■ Nome

### ■ Nomeadas ou Anônimas

## ■ Endereço

```
char l;  
char *m;  
m = &l + 10;  
printf("&l = %p\nm = %p\n*m = %c\n", &l, m, *m);
```

## ■ Sinonímia

```
int r = 0;  
int &s = r;  
s = 10;
```

# Propriedades de Variáveis

## ■ Tipo

- Especificação explícita (C), sintática (FORTRAN) ou semântica (ML)

## ■ Valor

- Configuração de Bits Corrente e Tipo

## ■ Tempo de Vida

- Locais, Globais e Anônimas
- Transientes ou Persistentes

# Propriedades de Variáveis

## ■ Escopo de Visibilidade

```
int x = 15;
void main() {
    int x;
    x = 10;
    printf("x = %d\n", x);
}
```

# Atualização de Variáveis Compostas

## ■ Completa ou Seletiva

```
struct data { int d, m, a; };  
struct data f = {7, 9, 1965};  
struct data g;  
g = f;      // completa  
g.m = 17;   // seletiva
```

# Constantes

## ■ Predefinidas

```
char x = 'g';  
int y = 3;  
char* z = "bola";  
/* int *w = &3; */  
*z = 'c';
```

## ■ Declaradas em C

```
const float pi = 3.1416;  
float raio, area, perimetro;  
raio = 10.32;  
area = pi * raio * raio;  
perimetro = 2 * pi * raio;
```



# Constantes

## ■ Declaradas em C++

```
int* x;  
const int y = 3;  
const int* z;  
// y = 4;  
// y++;  
// x = &y;  
z = &y;
```

# Armazenamento de Variáveis e Constantes



- Transientes
  - Memória Principal
- Persistentes
  - Memória Principal
  - Memória Secundária

# Memória Principal

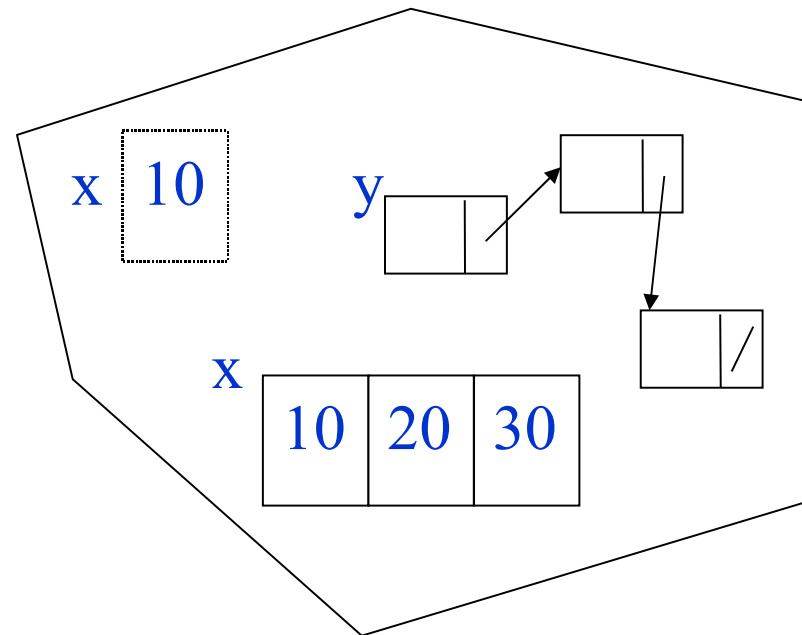
- Enorme sequência contígua e finita de bits - vetor de tamanho finito com elementos do tamanho da palavra do computador
- Alocação de Variáveis e Constantes
  - Tempo de Carga (FORTRAN)
    - | Super e subdimensionamento das variáveis
    - | Variáveis locais alocadas desnecessariamente
    - | Impedimento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

# Pilha e Monte

- LPs ALGOL-like
- Pilha
  - Variáveis Locais
  - Parâmetros
  - Regra de alocação bem definida
- Monte
  - Variáveis de Tamanho Dinâmico
  - Regra de alocação indefinida

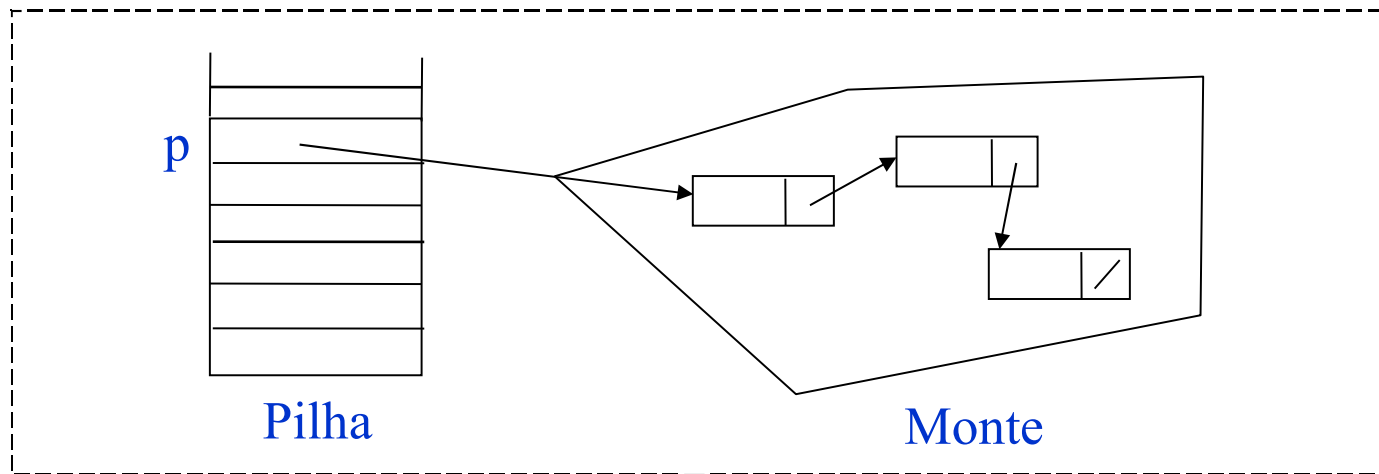
# Pilha e Monte

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



# Pilha e Monte

- Porque não usar só o Monte?
  - Menos Eficiente por causa de LED e LEO
- Ponteiro como Ligação entre Pilha e Monte



# Gerenciamento da Pilha

## ■ Uso de Registros de Ativação

constantes locais
variáveis locais
parâmetros
link dinâmico
link estático
endereço de retorno

# Gerenciamento da Pilha

```
procedure Principal is
```

```
  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;
```



# Gerenciamento da Pilha

```
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;
```

# Gerenciamento da Pilha

```
begin --Sub1
    a := 1; b:= 2; c:= 3;
    Sub3(19);                                -- 5
end Sub1;
begin --Principal
    x := 0;
    Sub1;                                    -- 6
end Principal;
```

# Gerenciamento do Monte

- LED e LEO
- Momento da Alocação sempre bem definido
- Momento da Desalocação definido por
  - Programador: mais eficiente, menos confiável e mais trabalhoso
  - LP: implementação mais complexa, falta de controle sobre a desalocação

# Coletor de Lixo

## ■ Java

- Dois Espaços: De e Para
  - | Movimenta de Um para o Outro
- Percorre todas as referências a partir da pilha movendo o que encontra
  - | Parada pode comprometer eficiência
- Coletor torna alocação no monte quase tão eficiente quanto na pilha

# Coletor de Lixo

## ■ Python

### ■ Contagem de Referências

- | Um contador por objeto
- | Toda atribuição ou passagem de parâmetro altera contadores dos objetos envolvidos
- | Objetos liberados assim que zeram seus contadores
- | Problema com referências cíclicas

```
l = []
```

```
l.append(l)
```

```
del l
```

# Memória Secundária

- Persistência de Dados
- Arquivos Seriais e Diretos
  - Operações de Abertura e Fechamento
  - Conversão de Dados para Formato Sequencial Binário

```
#include <stdio.h>
struct data {
    int d, m, a;
};
struct data d = {7, 9, 1999};
struct data e;
```

# Memória Secundária

```
void main() {  
    FILE *p;  
    char str[30];  
    printf("\n\n Entre com o nome do arquivo:\n");  
    gets(str);  
    if (!(p = fopen(str,"w"))) {  
        printf ("Erro! Impossivel abrir o arquivo!\n"); exit(1);  
    }  
    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);  
}
```

# Gerenciadores de Bancos de Dados

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



# Persistência Ortogonal

- Mesmos tipos para variáveis persistentes e transientes
- Nenhuma distinção entre o código que lida com variáveis persistentes e o que lida com variáveis transientes
- Identificação de persistência através da percepção da continuidade do uso
- Eliminação de Conversões de Entrada e Saída (30% do código)
- Não existem ainda na prática

# Serialização e Desserialização

- 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

# Serialização e Desserialização

- Ao restaurar uma variável da memória secundária, os ponteiros devem ser ajustados de modo a respeitar as relações existentes anteriormente entre as variáveis anônimas
- JAVA oferece, C++ não
- Mecanismo de JAVA compensa diferenças entre diferentes ambientes computacionais
- Ainda não é o ideal, mas facilita muito a vida do programador

# Serialização e Desserialização

```
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);
    }
    Impares() {
        j = j + 2;
        prox = new Impares(9);
    }
}
```

# Serialização e Desserialização

```
public String toString() {  
    String s = "" + i;  
    if(prox != null)  
        s += " : " + prox.toString();  
    return s;  
}  
  
public static void main(String[] args) {  
    Impares w = new Impares(6);  
    System.out.println("w = " + w);  
    try {  
        ObjectOutputStream out =  
            new ObjectOutputStream(  
                new FileOutputStream("impares.dat"));
```

# Serialização e Desserialização

```
out.writeObject("Armazena Impares");
out.writeObject(w);
out.close();
ObjectInputStream in =
    new ObjectInputStream(
        new FileInputStream("impares.dat"));
String s = (String)in.readObject();
Impares z = (Impares)in.readObject();
System.out.println(s + ", z = " + z);
in.close();
} catch(Exception e) {
    e.printStackTrace();
}
}
```

# Serialização em Python

```
from pickle import *
f = open("exemplopickle.txt", "w")
pickle.dump(54.3,f)
pickle.dump([11,22,33,55],f)
f.close()
f = open("exemplopickle.txt", "rw")
x = pickle.load(f)  # x = 54.299999999999997
type(x)             # <type 'float'>
y = pickle.load(f)  # y = [11, 22, 33, 55]
type(y)             # <type 'list'>
```