# Técnicas de Programação

## Tratamento de Erros e Depuração de Código II

Profa. Elaine Venson

elainevenson@unb.br

Semestre: 2012-1

# Conteúdo

- Definições
- Categorias de erros e tipos de erros
- Processo de depuração
- Ferramentas

# Definição

- Depuração é o processo de identificar a **causa raiz** de um erro e corrigi-lo
- Diferente do **teste**, que é o processo de **detectar** o erro inicial
- A depuração é considerada pelos desenvolvedores a atividades mais **difícil** da programação

# Definição

- Depuração é o processo de identificar a **causa raiz** de um erro e corrigi-lo

- Diferente do **teste**, que é o processo de **detectar** o erro inicial

- A depuração é considerada pelos desenvolvedores a atividades mais **difícil** da programação

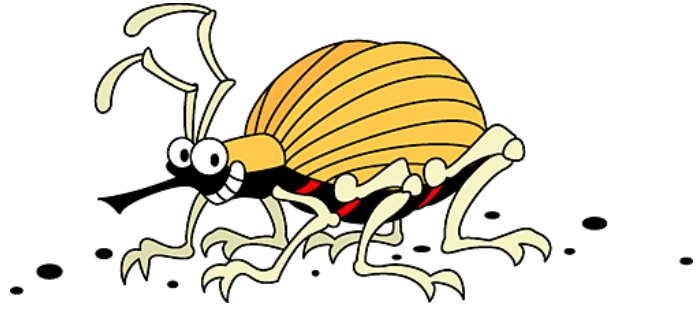
# Definições

- **Erro**: algo que foi feito de maneira errada. Uma ação do programador que resulta em um código contendo um defeito.

- **Defeito/Falta**: é a consequência de um erro. Um problema *latente*. Defeitos podem ser identificados em uma revisão de código.

- **Falha**: é a manifestação de um defeito. São identificados na execução do software.

- **Bug**: coloquialismo, geralmente usado como sinônimo de defeito.



# Motivação

| | 3 programadores mais rápidos | 3 programdores mais lentos |
|---|---|---|
| Tempo médio de depuração (minutos) | 5,0 | 14,1 |
| Quantidade média de defeitos não encontrados | 0,7 | 1,7 |
| Quantidade média de defeitos gerados na correção de defeitos | 3,0 | 7,7 |

Fonte: "Some Psychological Evidence on How People Debug Computer Programs" (Gould 1975)

# Defeitos como oportunidades

- Conhecer melhor o programa
- Aprender com os tipos de erros cometidos
- Analisar a qualidade do código do ponto de vista de outra pessoa
- Aprender sobre como resolver problemas
- Aprender sobre como corrigir defeitos

# Categorias de erros

① Erro de compilação

- É o mais fácil de detectar e corrigir
- Exemplos: erro de sintaxe, chamada a função com parâmetros errados, problema de “linkagem”, etc.

② Falha de execução

- O programa encerra de maneira repentina
- Mais difícil de investigar que o erro de compilação

# Categorias de erros

③ Comportamento inesperado

- O programa executa sem falhar, mas não apresenta o resultado esperado
- Geralmente relacionado a um problema de lógica

# Tipos de falha de execução

- Erros de sintaxe
  - A maioria é identificada pelo compilador
  - Mas alguns não são detectados
  - Exemplos:
    - Trocar == por = ou && por &
    - Esquecer um ponto e vírgula ou incluir em algum lugar errado
    - Não delimitar corretamente loops
    - Problema no uso dos parêntesis
  - A melhor forma de evitar esses erros é manter habilitados os avisos (warnings) do compilador

# Tipos de falha de execução

- Erros de geração do executável (*build*)
  - Um problema na geração do executável
  - O sistema de build não atualiza o executável por algum problema (dependência, timestamp do antigo executável, …)
  - O programador executa o programa sem perceber que é a versão antiga
  - Nesta situação é necessário excluir arquivos executáveis gerados e gerar o executável do zero (*rebuild*)

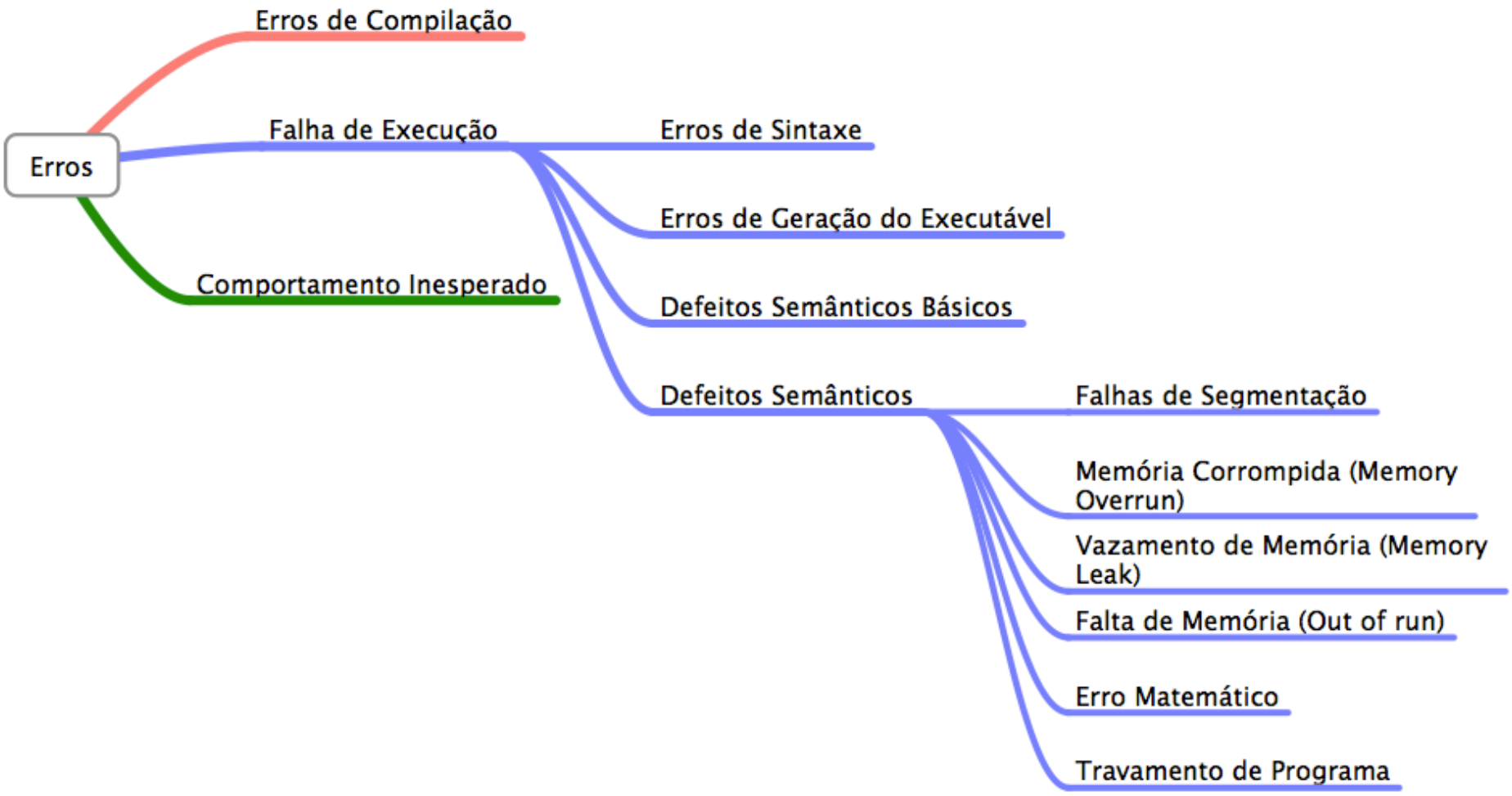
# Tipos de falha de execução

- Defeitos semânticos básicos
  - A maioria das falhas de execução devem-se a erros simples
  - Exemplos:
    - Usar variáveis não inicializadas
    - Realizar cálculos que não tratam overflow
    - Comparar igualdade de variáveis de ponto fluante
  - Esse tipo de erro pode ser detectado por ferramentas de análise sintática

# Tipos de falha de execução

- Defeitos semânticos
  - Erros não detectados por ferramentas de inspeção de código
  - Mais difíceis de serem encontrados
  - Exemplos:
    - Variável utilizada em local errado
    - Parâmetros de entrada de função não validados
    - Chamar uma API de forma errada

# Erros


Erros
Erros de Compilação
Falha de Execução
Erros de Sintaxe
Erros de Geração do Executável
Defeitos Semânticos Básicos
Defeitos Semânticos
Falhas de Segmentação
Memória Corrompida (Memory Overrun)
Vazamento de Memória (Memory Leak)
Falta de Memória (Out of run)
Erro Matemático
Travamento de Programa
Comportamento Inesperado

# Defeitos Semânticos

- Falhas de segmentação (*Segmentation Fault*)
  - Ou Falhas de Proteção (*Protecion Fault*)
  - Ocorre quando áreas de memória não alocadas são acessadas
  - O SO aborta a aplicação e apresenta uma mensagem de erro
  - Situações que causam este tipo de erro:
    - Erros envolvendo ponteiros e aritmética de ponteiros
  - Exemplo:
    - scanf("%d", number);

# Defeitos Semânticos

- Memória corrompida (*Memory Overruns*)
  - Ocorre quando uma área de memória alocada para uma estrutura de dados é sobrescrita indevidamente
  - Os sintomas podem ser erros aleatórios e podem ocorrer muito tempo depois da memória ter sido corrompida
  - É um problema comum e difícil de diagnosticar

# Defeitos Semânticos

- Vazamento de Memória (*Memory Leaks*)
  - É uma ameaça constante em linguagens que não tem *garbage collection*
  - Quando memória alocada (usando **malloc** em C ou **new** em C++) não é liberada (com **free** em C ou **delete** em C++)
  - O programa vai consumindo a memória e às vezes o problema só é percebido com a degradação do tempo de resposta do computador

# Defeitos Semânticos

- Falta de Memória (*Running out of memory*)
  - Erro menos comum, mas uma falha em potencial
  - Retorno de alocacão de memória ou chamadas ao sistema de arquivos deve sempre ser verificado

# Defeitos Semânticos

- Erros Matemáticos
  - Exceções de ponto flutuante, construções matemáticas incorretas, overflow/underflow, divisão por zero
- Travamento
  - Problemas na lógica do programa
  - Loops infinitos, problemas na cláusula de encerramento
  - Deadlock em códigos que usam threads
  - Código orientado a eventos aguardando eventos que nunca ocorrem

# Depuração

- Abordagens **ineficientes**:
  - Tentativa e erro
  - Não gastar tempo tentando entender o problema
  - Corrigir o defeito da forma mais óbvia

```
x = Compute( y )
if ( y = 17 )
   x = $25.15

      -- Compute() doesn't work for y = 17, so fix it
```

# Depuração

- Uma abordagem eficiente depuração é derivada do **método científico**:
  1. Coletar informações através de experimentos
  2. Formular uma hipótese a partir das informações levantadas
  3. Elaborar um experimento para confirmar ou invalidar a hipótese
  4. Confirmar ou invalidar a hipótese
  5. Repetir conforme necessário

# Depuração

- Abordagem de depuração efetiva:
  1. Estabilizar a falha
  2. Localizar a fonte da falha (defeito)
     a. Levantar as informações que produziram a falha
     b. Analisar as informações e formular uma hipótese sobre o defeito
     c. Determinar como confirmar ou invalidar a hipótese (através de teste ou inspeção do código)
     d. Confirmar ou invalidar a hipótese
  3. Corrigir o defeito
  4. Testar a correção
  5. Procurar por erros similares

# 1 – Estabilizar a falha

- Encontrar uma forma de **predizer** uma falha **intermitente** é uma das atividades mais desafiadoras da depuração

- **Simplificar** ao máximo o caso de teste, eliminando fatores que não influencia na geração da falha

# 2 – Localizar a fonte da falha

- Formular hipóteses de acordo com as informações do teste
- Criar novos casos de teste para as hipóteses formuladas

# Depuração

- A correção de falhas envolve duas atividades:
  1. Identificar o defeito que causou a falha
  2. Corrigir o defeito

- Fatores que influenciam a depuração:
  - Conhecimento a respeito do código
  - Controle sobre o ambiente de execução
  - Atitude de desconfiança em relação a tudo

# Depuração – erros de compilação

- Erros de compilação
  - Os compiladores geralmente **não interrompem** o processo de compilação ao encontrar o primeiro erro, continuam analisando o código até o final
  - Com isso é gerada uma lista de mensagens onde as últimas tendem a ser aleatórias e irrelevantes
  - As **primeiras mensagens** são mais significativas para identificar o defeitos do que as subsequentes
  - As mensagens apresentadas também dependem do compilador utilizado

# Depuração – identificar o defeito

- Erros de execução
  - Exigem um método mais elaborado:
    1. Identificar a falha: registrar a ocorrência e analisar o contexto
    2. Reproduzir o comportamento
    3. Localizar o defeito:
       - Dividir e conquistar: imprimir o resultado ou incluir break-point no ponto médio da execução
       - Dry run: rastrear a execução passo-a-passo
    4. Entender o problema: causa real
    5. Criar um teste para a situação
    6. Corrigir o defeito
    7. Testar

# Depuração – corrigir o defeito

- É uma atividade crítica!
- Há sempre o risco de causar **novos defeitos**
- Ao modificar o código, sempre devem ser analisadas as **consequências**
- Garantir que a **causa raiz** da falha foi identificada e não apenas um sintoma
- Verificar se **erros similares** podem estar presentes em módulos relacionados

# Considerações psicológicas

# Considerações psicológicas

```
if ( x < y )
    swap = x;
    x = y;
    y = swap;
```

```
if ( x < y ) {
    swap = x;
    x = y;
    y = swap;
}
```

# Ferramentas de depuração

- Depurador (debugger)
  - Ferramenta interativa que permite analisar a execução do programa internamente
  - Deve ser usado com moderação
  - Não deve ser usado como uma alternativa para entender o código
- Validador de acesso à memória
  - Inspeciona a execução de programa buscando encontrar vazamentos de memória e memória corrompida

# Ferramentas de depuração

- Rastreamento de chamadas de sistema
  - “Trace do Linux”
  - Apresenta todas as chamadas de sistemas originadas pela aplicação
  - Útil para verificar como o programa está interagindo com o ambiente
- Core dump
  - Termo do Unix, gerar uma fotografia do momento em que ocorreu a falha (memória, registradores, pilha de execução)

# Ferramentas de depuração

- Logging
  - Gerar informações sobre a execução da aplicação
  - Ferramentas mais sofisticadas apresentam níveis de log que podem ser filtrados
  - Ajuda a analisar as circunstâncias em que determinada falha ocorreu
  - Resultado semelhante pode ser alcançado incluindo código para imprimir mensagens e conteúdo de variáveis em pontos específicos do programa
  - Pode deixar o programa mais lento

# Ferramentas de depuração

- Analisador Estático
  - Ferramenta não interativa que inspeciona o código para identificar problemas pontenciais
  - Muitos compiladores realizam análise estática básica
  - Um boa recomendação é utilizar uma ferramenta de análise estática de uma empresa diferente do fabricante do compilador

# Ferramentas de depuração

- Profilers
- Comparadores de código fonte
- Test frameworks