

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Programação e Desenvolvimento de Software 2

Estratégias de depuração e ferramentas

Prof. Douglas G. Macharet
douglas.macharet@dcc.ufmg.br

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO



Introdução

- Boas práticas de programação
 - Reduzem chance de erros (eles vão existir!)
 - Medidas proativas
 - Programação defensiva
 - Testes de unidade
- Meu programa não funciona! E agora?!
~~Vou reescrever tudo do zero!~~
 - Medidas reativas
 - Depuração

DCC UFMG PDS 2 - Estratégias de depuração e ferramentas 2

Depuração

- Verificação e validação
 - Relacionados ao estabelecimento da existência de falhas (inconsistências) em um programa
- Depuração (debugging)
 - Relacionado à localização e reparação de falhas

Primeiro bug na computação (Harvard Mark II).

DCC UFMG PDS 2 - Estratégias de depuração e ferramentas 3

Depuração

Motivação

- Depurar grandes programas é difícil (arte?)
- Um bom programador deve conhecer uma ampla variedade de estratégias de depuração
- Um bom programador deve conhecer/usar ferramentas que facilitam a depuração
 - Debuggers
 - Sistemas de controle de versão
 - IDEs


A Arte de Depurar: <https://msdn.microsoft.com/pt-br/library/cc517980.aspx>

DCC UFMG PDS 2 - Estratégias de depuração e ferramentas 4

Depuração

Quando realizar

- Código dá *crash*, parando a execução
- Resultados diferentes dos esperados
- Necessidade de melhorar o desempenho
- Entender melhor como o código funciona



Six Stages of Debugging

1. That can't happen.
2. That doesn't happen on my machine.
3. That shouldn't happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?

DCC UFMG PDS 2 - Estratégias de depuração e ferramentas 5

Depuração vs. Testes

- Depurar (detetive)
 - O que fazer para tentar consertar o programa?
- Testar (vândalo)
 - O que fazer para tentar quebrar o programa?


When debugging, novices insert corrective code; experts remove defective code.

— R. Pattis

DCC UFMG PDS 2 - Estratégias de depuração e ferramentas 6

Depuração

Tipos de erros

- Sintáticos
 - Erros associados ao fato de que a sintaxe da linguagem não está sendo respeitada
- Semânticos
 - Erros associados a um uso indevido de algumas declarações do programa
- Lógica 
 - Erros associados ao fato de que a especificação (comportamento desejado) não é respeitada

Depuração

Tipos de erros – Sintáticos

```
int main() {
    int a = 5
    return 0;
}
```

- Falta adicionar ';' ao final.

```
int main() {
    int x = (3 + 5;
    return 0;
}
```

- Falta fechar o parênteses.

Depuração

Tipos de erros – Sintáticos

- Leia e entenda as mensagens de erro!

```
main.cpp: In function 'int main()':
main.cpp:15:3: error: expected ';' or ',' before 'return'
    return 0;
    ^
```

- **-Wall:** Exibe na tela todos os warnings que ele encontrar no código. Um warning não é um erro, mas sim uma advertência sobre o uso incorreto (não recomendado) de alguma função/instrução da linguagem.
- **-pedantic:** Esta flag faz com que o compilador seja mais "pedante", emitindo warnings para todas as partes do código que podem estar erradas.

Depuração

Tipos de erros – Semânticos

```
int main() {
    int i;
    i++;
    return 0;
}
```

- Variável não inicializada.
- Ocorre apenas um Warning.

```
int main() {
    int a = "hello";
    return 0;
}
```

- Atribuição incorreta de tipo.
- Erro de compilação.

Depuração

Tipos de erros – Momento da detecção

- Tempo de compilação
 - Erros de sintaxe e erros semânticos estáticos são indicados pelo próprio compilador
- Tempo de execução
 - Erros semânticos dinâmicos e erros lógicos não podem ser detectados pelo compilador (difícil)
 - É necessário depurar o código!

Depuração

Tipos de erros – Momento da detecção

```
int main() {
    int a, b, x;
    a = 10;
    b = 0;
    x = a / b;
    return 0;
}
```

O programa compila, e o problema só será detectado durante a execução do programa.

Depuração

Tipos de erros – Lógica

- Geralmente detectados durante execução
 - Retorno de um resultado incorreto
 - Loops infinitos
 - *Segmentation fault*
- Geralmente bastante imprevisíveis
 - De acordo com entradas (bem) específicas
 - Dependentes de plataforma ou hardware

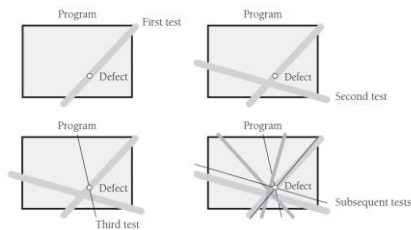
Depuração

Estratégias

- Reproduzir o problema
 - Determinar condições / Estabilizar
- Identificar o local e provável causa
 - Dados → Hipótese → Experimentos (repetir)
- Fazer a correção do erro
 - Considerar a real razão do problema
- Avaliar a solução e procurar erros similares
 - Cuidado com possíveis efeitos colaterais!

Depuração

Estratégias



Depuração

Estratégias (raiz)

- printf / cout
 - “Acompanhamento” da execução do programa
 - Selecionar pontos chave alcançados ou não, exibir os valores de variáveis importantes
- Prós
 - Simples, fácil e rápido
- Cons
 - Pode prejudicar a legibilidade do código
 - Várias compilações para diferentes testes
 - Não é possível pausar ou controlar a execução
 - Não é possível modificar valores de variáveis

Depuração

Exemplo 1

- Calcular o valor da seguinte série
 - Entradas: x e n

$$\frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!}$$

- Por onde começar?
 - Modularizar em 2 funções

Depuração

Exemplo 1

```
#include <iostream>
#include <cmath>

using namespace std;

int fatorial(int n) {
    int fat = 1;
    for (int i = 1; i <= n; i++)
        fat = fat * i;
    return fat;
}

double series(double x, int n) {
    double xpow, seriesValue;
    for (int k = 1; k <= n; k++) {
        xpow = pow(x, k);
        seriesValue += xpow / fatorial(k);
    }
    return seriesValue;
}
```

- Esse código funciona?

```
int main() {
    cout << series(2, 3) << endl;
    return 0;
}
```

- Como descobrir o erro?

- fatorial(int n)
 - int fat = 1;
 - for (int i = 1; i <= num; i++)
- series(double x, int n)
 - for (int k = 0; k <= n; k++)
 - double seriesValue = 0.0;

Depuração

Dicas gerais

- Pense antes de escrever
- Procure por problemas comuns
- Dividir para conquistar
- Adicione mais testes internos
- Mostre o valor de variáveis importantes
- Concentre-se em mudanças recentes
- Utilize ferramentas auxiliares

DCC

PDS 2 - Estratégias de depuração e ferramentas

19

GDB

▪ GNU Debugger

- Depurador “padrão” de C/C++
- Permite acompanhar o que está acontecendo dentro do programa enquanto é executado
 - Adicionar breakpoints
 - Analisar o código uma linha de cada vez
 - Verificar valores de variáveis durante a execução

<https://www.gnu.org/software/gdb/>
<https://betterexplained.com/articles/debugging-with-gdb/>

DCC

PDS 2 - Estratégias de depuração e ferramentas

20

GDB

- Pode ser utilizado pela linha de comando
 - Utilizar a flag de compilação “-g” (debug)



▪ Ferramentas auxiliares

- <https://gdbgui.com>
- <https://www.onlinegdb.com/>

DCC

PDS 2 - Estratégias de depuração e ferramentas

21

GDB

Exemplo – Identifique e corrija os erros!

```
#include<iostream>

using namespace std;

int findMax(int *array, int len, int max) {
    max = array[0];
    for(int i=1; i <= len; i++) {
        if(max < array[i]) {
            max = array[i];
        }
    }
    return 0;
}
```

```
int main() {
    int arr[5] = {17, 21, 44, 2, 60};
    int max;
    if (findMax(arr, 5, max) != 0) {
        cout << "Ocorreu erro!" << endl;
        exit(1);
    }
    cout << "Valor maximo e: " << max << endl;
    return 0;
}
```

```
$ g++ -std=c++11 -g gdbExample.cpp -o gdbExample
$ gdb gdbExample
```

DCC

PDS 2 - Estratégias de depuração e ferramentas

22

Valgrind

- Ferramentas que ajudam a detectar erros associados ao gerenciamento de memória
 - Memory leak, erros alocação ou desalocação, ...
- Ferramentas auxiliares
 - Valkyrie: <http://valgrind.org/downloads/guis.html>
 - Dr. Memory: <http://drmemory.org/>

<http://valgrind.org/docs/manual/quick-start.html>

DCC

PDS 2 - Estratégias de depuração e ferramentas

23

Valgrind

Exemplo 1 – Identifique e corrija os erros!

```
#include<iostream>

using namespace std;

int main() {
    int a[10];
    for (int i = 0; i < 9; i++)
        a[i] = i;

    for (int i = 0; i < 10; i++){
        cout << a[i] << endl;
    }

    return 0;
}
```

```
$ g++ -std=c++11 -g valgrind_ex01.cpp -o valgrind_ex01
$ valgrind --leak-check=full ./valgrind_ex01
```

<http://valgrind.org/docs/manual/mmc-manual.html#mc-manual.uninitvals>

DCC

PDS 2 - Estratégias de depuração e ferramentas

24

Valgrind

Exemplo 2 – Identifique e corrija os erros!

```
#include<iostream>

using namespace std;

struct TADExemplo {
    int atributo;
};

int main() {
    TADExemplo *c = new TADExemplo();
    c->atributo = 10;

    delete c;

    cout << c->atributo << endl;

    c->atributo = 99;
    cout << c->atributo << endl;

    delete c;

    return 0;
}
```

https://en.wikipedia.org/wiki/Undefined_behavior
<http://valgrind.org/docs/manual/mc-manual.html#mc-manual.leaks>

DCC 

PDS 2 - Estratégias de depuração e ferramentas

25

Valgrind

Exemplo 3 – Identifique e corrija os erros!

```
#include<list>

using namespace std;

struct TADExemplo {
    int atributo;
};

int main() {
    list<TADExemplo> itens;

    itens.push_back(new TADExemplo());
    itens.push_back(new TADExemplo());

    return 0;
}
```

<http://valgrind.org/docs/manual/mc-manual.html#mc-manual.leaks>

DCC 

PDS 2 - Estratégias de depuração e ferramentas

26

Considerações finais

Como NÃO fazer depuração

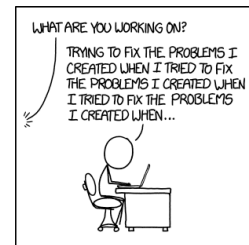
- Encontrar defeitos adivinhando (na sorte)
- Fazer alterações aleatórias até funcionar
- Não fazer um backup do original e não manter um histórico das alterações feitas
- Corrigir o erro com a solução mais óbvia e simples, sem entender a razão do problema
 - O sintoma é removido, mas não a causa
 - O erro “sumiu”, o problema está resolvido! 😞

DCC 

PDS 2 - Estratégias de depuração e ferramentas

27

Considerações finais



<https://xkcd.com/1739/>

DCC 

PDS 2 - Estratégias de depuração e ferramentas

28