



SSC 0721 – Teste e Inspeção de Software

Critérios de fluxo de dados

Prof. Marcio E. Delamaro

delamaro@icmc.usp.br

Critérios Baseados em Fluxo de Dados

- ✓ Critérios pertencentes à Técnica de Teste Caixa Branca.
- ✓ Complementares aos critérios baseados em fluxo de controle.
- ✓ Busca testar o uso das variáveis em um programa, ou seja, como os dados são usados nas computações.

Motivação

- ✓ Teste de fluxo de dados é uma ferramenta poderosa para o uso incorreto de valores resultante de erros de codificação.
- ✓ Tornou-se popular com a publicação do trabalho de Rapps e Weyuker
“It is our belief that, just as one would not feel confident about a program without executing every statement in it as part of some test, one should not feel confident about a program without having seen the effect of using the value produced by each and every computation.”

Critérios de fluxo de dados

- ✓ Usam informação sobre como as variáveis são utilizadas
- ✓ Cada vez que um valor é atribuído a uma variável, é necessário verificar se esse valor está correto
- ✓ Essa verificação é feita quando esse valor for acessado
- ✓ Tipos de utilização de variáveis

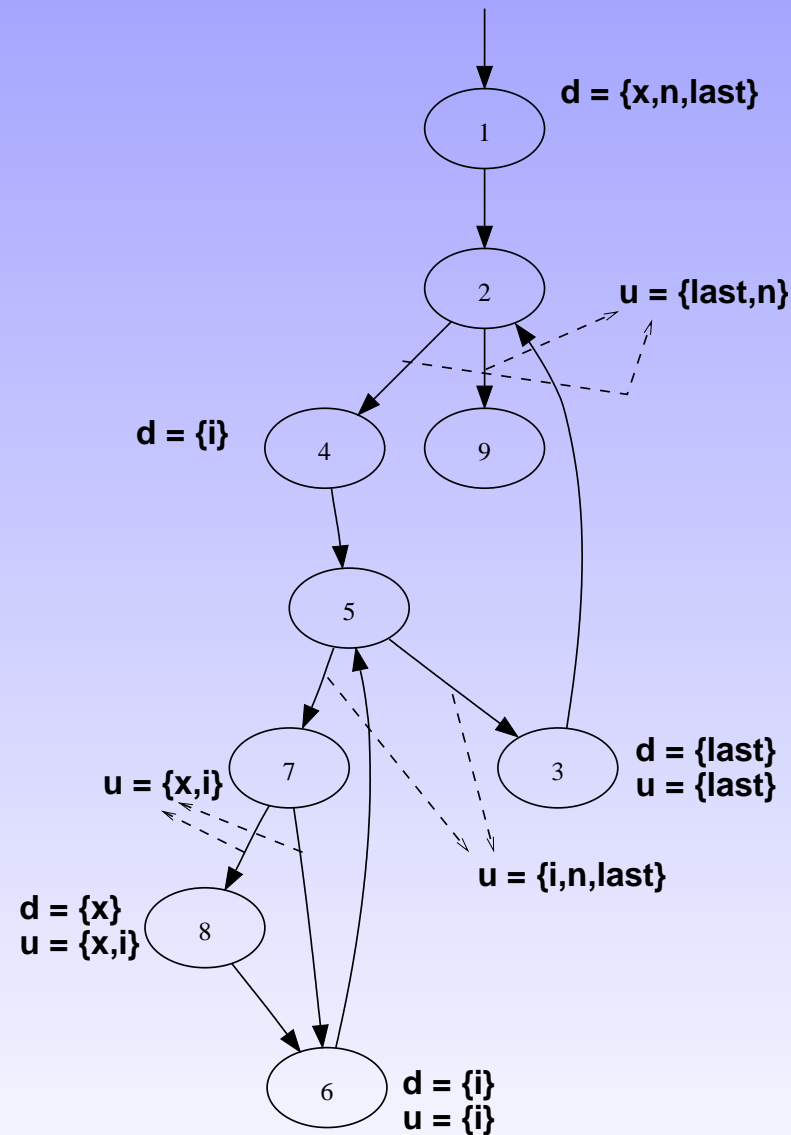
Critérios de fluxo de dados

- ✓ Tipos de utilização de variáveis
 - ★ **Definição:** é toda referência feita a uma variável que faz com que o valor dessa variável possa ser alterado
 - $a = 10;$
 - $\text{read}(a);$
 - ★ **Uso:** são todas as demais referências a variáveis, ou seja, quando o valor armazenado na variável é utilizado mas não modificado.
 - predicativo ou p-uso: $\text{if } (a > 10)$
 - computacional ou c-uso: $a = b * c$

Grafo def-uso

- ✓ Estende o GFC
- ✓ Adiciona anotações de uso e definição de variáveis
- ✓ Cada c-uso é associado ao vértice do comando onde o uso ocorre
- ✓ Cada p-uso é associado às arestas que ele determina
- ✓ Uso e definição global

Grafo def-uso – exemplo



Modelo de fluxo de dados

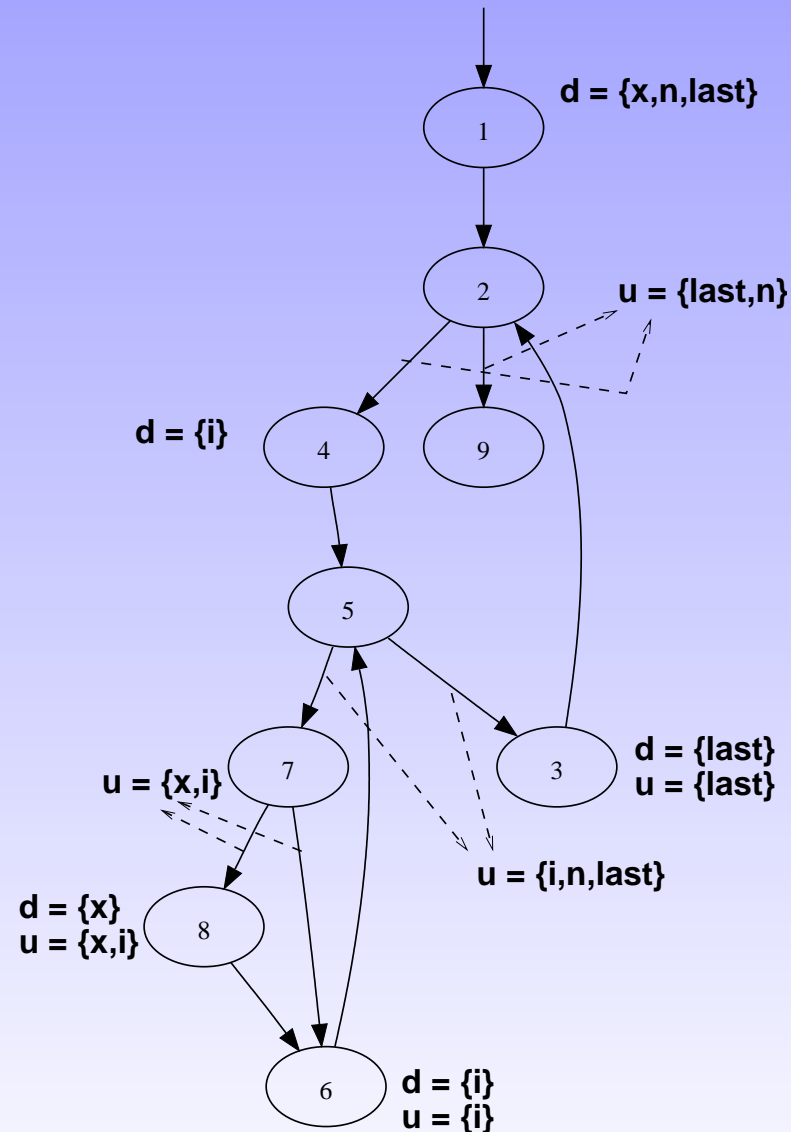
- ✓ Como se dão as definições e os usos
- ✓ Definição e uso de arrays. $v[i] = 10;$
- ✓ Utilização de ponteiros ou referências a objetos.
 $x \rightarrow idade = 10$
- ✓ Chamada de subrotinas. Passagem de parâmetros por referência
- ✓ Uso de variáveis globais. Onde se dá a definição?

Conceitos

- ✓ Um **caminho livre de definição** em relação a uma variável x do vértice i ao vértice j ou à aresta (j, k) é um caminho que não possui nenhuma definição de x , exceto, possivelmente, no vértice inicial, i ou no vértice k
- ✓ Uma **associação definição/uso de uma variável x** é uma tripla $\langle x, i, j \rangle$ ou $\langle x, i, (j, k) \rangle$ tal que x é definida no nó i e possui um uso no nó j ou na aresta (j, k) . Além disso deve existir um caminho livre de definição c.r.a. x dessa definição até esse uso.

Associação – exemplo

- ✓ $\langle i, 4, 8 \rangle$
- ✓ $\langle i, 4, (7, 6) \rangle$
- ✓ $\langle i, 4, (7, 8) \rangle$
- ✓ $\langle x, 8, 8 \rangle$
- ✓ $\langle x, 8, (7, 6) \rangle$
- ✓ $\langle x, 8, (7, 8) \rangle$



Critérios

- ✓ Critérios são baseados na execução das associações def-uso
- ✓ Critério **todas-definições** requer que para cada definição de variável, um uso seja exercitado
- ✓ Critério **todos-usos** requer que para cada definição de variável, todos os usos existentes sejam exercitados
- ✓ Outros: todos-p-usos-algum-c-uso, todos-c-usos-algum-p-uso, ...

Critérios – exemplo

| Variável | Definição | Uso | Variável | Definição | Uso |
|----------|-----------|-------|----------|-----------|-------|
| i | 4 | 6 | last | 1 | 3 |
| i | 4 | 8 | last | 1 | (5,3) |
| i* | 4 | (5,3) | last | 1 | (5,7) |
| i | 4 | (5,7) | last | 1 | (2,4) |
| i | 4 | (7,6) | last* | 1 | (2,9) |
| i | 4 | (7,8) | n | 1 | (5,7) |
| i | 6 | 8 | n | 1 | (5,3) |
| i | 6 | (5,3) | n | 1 | (2,4) |
| i | 6 | (5,7) | n | 1 | (2,9) |
| i | 6 | (7,6) | x | 0 | 8 |
| i | 6 | (7,8) | x | 0 | (7,6) |
| last | 3 | (5,3) | x | 0 | (7,8) |
| last | 3 | (5,7) | x | 8 | 8 |
| last | 3 | (2,4) | x* | 8 | (7,6) |
| last | 3 | (2,9) | x | 8 | (7,8) |

Observações

- ✓ Complementa os critérios de fluxo de controle
- ✓ Eficaz em revelar alguns tipos de defeitos, principalmente relacionados ao dados
- ✓ Na prática são critérios que não fizeram muito sucesso
 - ★ Complexidade
 - ★ Requisitos não executáveis
 - ★ Ferramentas

Ferramenta – JaBUTi

- ✓ Java Bytecode Understanding and Testing
- ✓ Fluxo de controle e fluxo de dados
- ✓ Linguagem Java (bytecode da JVM)
- ✓ Critérios (in)dependente de exceções

JaBUTi

JaBUTi v. 1.0 -- C:\Users\delamaro\Documents\Palestras\JAI2010\submetido\progs\buble.jbt

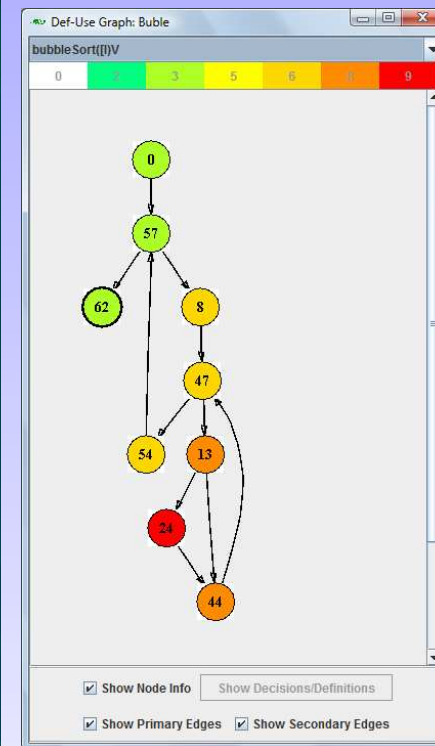
File Tools Visualization QualiPSo Summary Test Case Reports Properties Update Help

All-Nodes-ei All-Edges-ei All-Uses-ei All-Pot-Uses-ei All-Nodes-ed All-Edges-ed All-Uses-ed All-Pot-Uses-ed

0 1 2 3 4 5 6 7 8 9

```
/* 0001 */ public class Buble {
/* 0002 */
/* 0003 */ public static void bubbleSort(int[] x) {
/* 0004 */ int n = x.length; /* 1 */
/* 0005 */ for (int last = 1; /* 1 */; last < n /* 2 */; last++ /* 3 */)
/* 0006 */ {
/* 0007 */ for (int i = 0 /* 4 */; i < n - last /* 5 */; i++ /* 6 */)
/* 0008 */ {
/* 0009 */ if (x[i] > x[i+1]) /* 7 */ {
/* 0010 */ int aux = x[i]; /* 8 */
/* 0011 */ x[i] = x[i+1]; /* 8 */
/* 0012 */ x[i+1] = aux; /* 8 */
/* 0013 */ }
/* 0014 */ }
/* 0015 */ }
/* 0016 */ } /* 9 */
/* 0017 */
/* 0018 */
/* 0019 */ static public void main(String args[])
/* 0020 */ {
/* 0021 */ int[] v = new int[args.length];
/* 0022 */ for (int i = 0; i < v.length; i++)
/* 0023 */ v[i] = Integer.parseInt(args[i]);
/* 0024 */ bubbleSort(v);
/* 0025 */ for (int i = 0; i < v.length; i++)
/* 0026 */ System.out.print(v[i] + " ");
/* 0027 */ System.out.println();
/* 0028 */ }
/* 0029 */ }
```

JaBUTi: Coverage Tool File: Buble Line: 1 of 34 Coverage: All-Nodes-ei Highlighting: All Priorized



JaBUTi

- ✓ Execução manual ou integrada ao JUnit
- ✓ Priorização dos requisitos de teste
- ✓ Identificação de requisitos não executáveis
- ✓ Análise de cobertura por critério/classe/método
- ✓ Ferramenta de slicing
- ✓ Métricas estáticas OO
- ✓ <http://ccsl.icmc.usp.br>

JaBUTi – análise de cobertura

The screenshot displays the JaBUTi v. 1.0 application window. The title bar indicates the file path: C:\Users\delamaro\Documents\Palestras\JAI2010\submetido\progs\buble2.jbt. The menu bar includes File, Tools, Visualization, QualiPSO, Summary, Test Case, Reports, Properties, Update, and Help. Below the menu bar, there are radio buttons for selecting analysis options: All-Nodes-ei, All-Edges-ei, All-Uses-ei (selected), All-Pot-Uses-ei, All-Nodes-ed, All-Edges-ed, All-Uses-ed, and All-Pot-Uses-ed. The main area is titled "All-Uses-ei Testing Requirements". Below this, there are dropdown menus for "Buble" and "bubbleSort(I)V". A set of buttons includes "Activate All", "Deactivate All", "Feasible All", and "Infeasible All". A table lists testing requirements with columns for Covered, Active, Infeasible, and the requirement text. At the bottom, a summary table shows Method Coverage (27 of 28, 96%) and Total Coverage (47 of 50, 94%).

| Covered | Active | Infeasible | Testing Requirement |
|-------------------------------------|-------------------------------------|-------------------------------------|---------------------|
| <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <L@3, 8,(47, 54)> |
| <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <L@0, 24,(13, 44)> |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <L@3, 8,(47, 13)> |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <L@3, 8,(13, 44)> |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <L@3, 8,(13, 24)> |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <L@3, 8,44> |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <L@2, 54,(47, 54)> |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <L@2, 54,(47, 13)> |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <L@3, 44,24> |
| <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <L@2, 0,(47, 54)> |

| | | |
|-----------------|----------|-----|
| METHOD COVERAGE | 27 of 28 | 96% |
| TOTAL COVERAGE | 47 of 50 | 94% |

JaBUTi: Coverage Coverage: All-Uses-ei Active Test Cases: 3 of 3

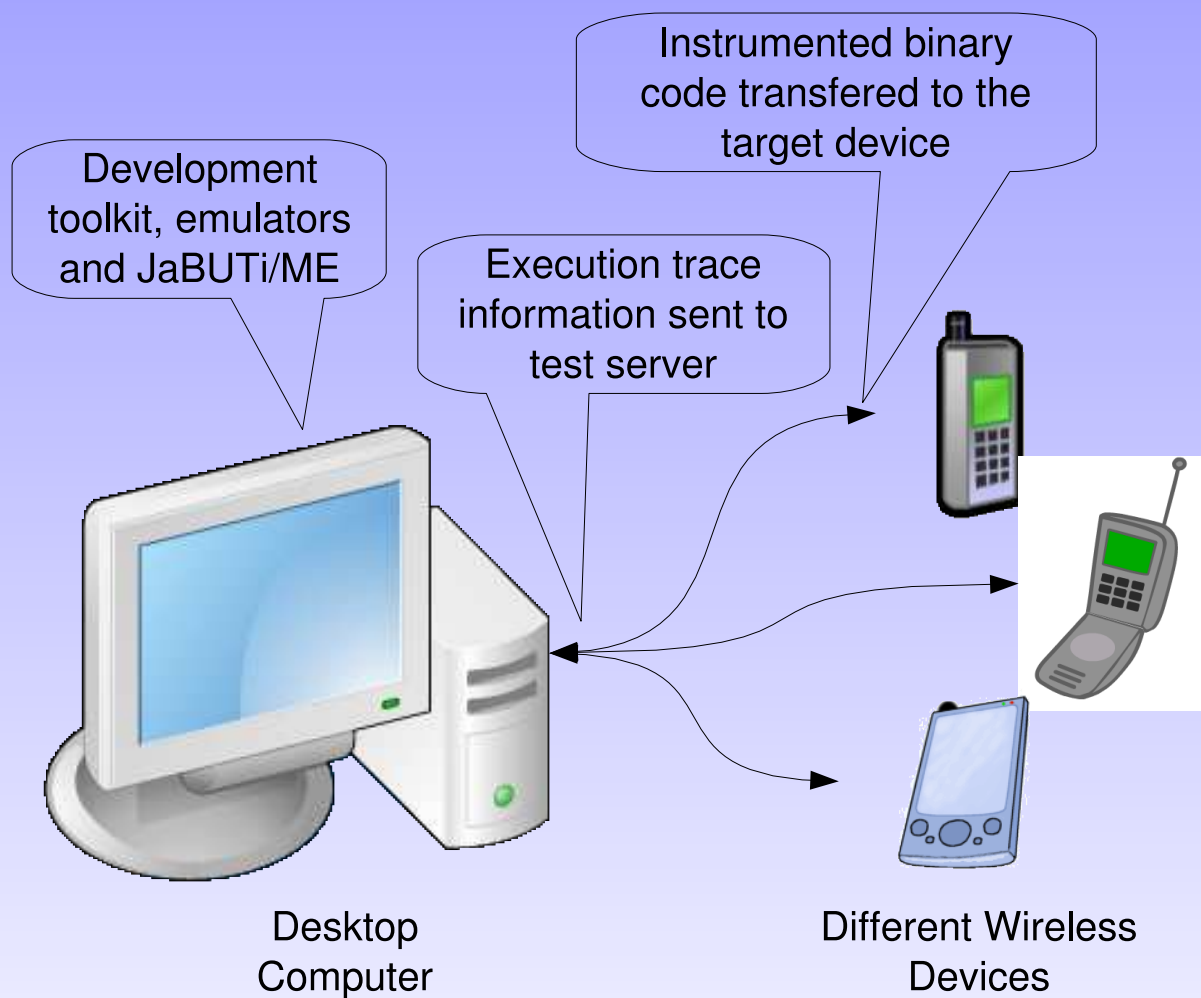
JaBUTi – slicing

The screenshot displays the JaBUTi v. 1.0 application window. The title bar shows the file path: C:\Users\delamaro\Documents\Palestras\JAI2010\submetido\progs\buble2.jbt. The menu bar includes File, Tools, Visualization, QualiPSo, Summary, Test Case, Reports, Properties, Update, and Help. Below the menu bar, there are radio buttons for selecting different coverage views: All-Nodes-ei (selected), All-Edges-ei, All-Uses-ei, All-Pot-Uses-ei, All-Nodes-ed, All-Edges-ed, All-Uses-ed, and All-Pot-Uses-ed. The main area is titled "All-Nodes-ei Coverage per Test Case" and contains a table with the following data:

| Success | Fail | Test Case | JUnit Name | Host | Total Coverage | Percentage |
|-------------------------------------|-------------------------------------|-----------|-------------------|-----------|----------------|------------|
| <input checked="" type="checkbox"/> | <input type="checkbox"/> | 0001 | ProberLoader 0001 | localhost | 16 of 18 | 88% |
| <input checked="" type="checkbox"/> | <input type="checkbox"/> | 0002 | ProberLoader 0002 | localhost | 15 of 18 | 83% |
| <input type="checkbox"/> | <input checked="" type="checkbox"/> | 0003 | ProberLoader 0003 | localhost | 10 of 18 | 55% |
| <input type="checkbox"/> | <input type="checkbox"/> | 0004 | ProberLoader 0004 | localhost | 15 of 18 | 83% |
| <input type="checkbox"/> | <input type="checkbox"/> | 0005 | ProberLoader 0005 | localhost | 16 of 18 | 88% |
| <input type="checkbox"/> | <input type="checkbox"/> | 0006 | ProberLoader 0006 | localhost | 8 of 18 | 44% |
| <input type="checkbox"/> | <input type="checkbox"/> | 0007 | ProberLoader 0007 | localhost | 16 of 18 | 88% |

Below the table, a summary bar shows "TOTAL COVERAGE" as "16 of 18" with a progress bar indicating 88%. At the bottom, there are three status boxes: "JaBUTi: Slice", "Coverage: All-Nodes-ei", and "Active Test Cases: 3 of 3".

JaBUTi/ME



Ferramenta – GCC

- ✓ Famoso compilador C
- ✓ Inclui opções que permitem gerar código instrumentado
- ✓ Execução gera arquivos com informação sobre cobertura de código (nós e desvios)

Ferramenta – gcov

```
-:      0:Source:buble.c
-:      0:Graph:buble.gcno
-:      0:Data:buble.gcda
-:      0:Runs:1
-:      0:Programs:1
-:      1:#include <stdio.h>
-:      2:
1:      3:void bubbleSort(int x[], int n) {
3:      4:     for (int last = 1; last < n; last++) {
5:      5:         for (int i = 0; i < n - last; i++)
3:      6:             if (x[i] > x[i+1]) {
#####:      7:                 int aux = x[i];
#####:      8:                 x[i] = x[i + 1];
#####:      9:                 x[i+1] = aux;
-:     10:             }
-:     11:         }
-:     12:     }
1:     13: }
```

Ferramenta – lcov

LCOV - code coverage report

Current view: [top level](#) - [submetido/progs](#) - [buble.c](#) (source / functions)

Test: **buble.info**
Date: 2010-02-19

| | Hit | Total | Coverage |
|-------------------|-----|-------|----------|
| Lines: | 13 | 16 | 81.2 % |
| Functions: | 2 | 2 | 100.0 % |
| Branches: | 9 | 10 | 90.0 % |

| Branch data | Line data | Source code |
|-------------|-----------|--|
| | 1 | : #include <stdio.h> |
| | 2 | |
| | 3 | 1 : void bubbleSort(int x[], int n) { |
| [+ +] | 3 | 3 : for (int last = 1; last < n; last++) { |
| [+ +] | 5 | 5 : for (int i = 0; i < n - last; i++) { |
| [- +] | 3 | 3 : if (x[i] > x[i+1]) { |
| | 0 | 0 : int aux = x[i]; |
| | 0 | 0 : x[i] = x[i + 1]; |
| | 0 | 0 : x[i+1] = aux; |
| | | } |
| | | } |
| | 1 | 1 : } |
| | | |
| | | void main(int argc, char* argv[]) |
| | 1 | 1 : { |
| | | int v[100]; |
| | | |
| [+ +] | 4 | 4 : for (int i = 0; i < argc-1; i++) |
| | 3 | 3 : v[i] = atoi(argv[i+1]); |
| | 1 | 1 : bubbleSort(v, argc-1); |
| [+ +] | 4 | 4 : for (int i = 0; i < argc-1; i++) |
| | 3 | 3 : printf("%d ", v[i]); |
| | 1 | 1 : printf("\n"); |
| | 1 | 1 : } |
| | | |
| | | • |

Generated by: [LCOV version 1.8](#)

Concluído