Discussion Section 6

- HW 5 tips and questions?
- Motif-finding algorithms
- If time: using valgrind to find memory leaks/out of bounds bugs

HW 5 output

- What you report:
    - Nucleotide histogram
    - Background frequency
    - Count matrix (-10 to 10 nucleotides)
    - Frequency matrix (-10 to 10 nucleotides)
    - Weight matrix (-10 to 10 nucleotides)
    - Maximum score
    - Score histogram for CDS
    - Score histogram for all positions
    - List of non-CDS positions with score >= 10

HW 5 tips

- Looking only for 'CDS' features

HW 5 tips

- Looking only for 'CDS' features
- 'complement' indicates the reverse complement

HW 5 tips

- Looking only for 'CDS' features
- 'complement' indicates the reverse complement
- 'ORIGIN' section contains the actual sequence

HW 5 tips

- Looking only for 'CDS' features
- 'complement' indicates the reverse complement
- 'ORIGIN' section contains the actual sequence
- Positions downstream of the translation start site could be noncontiguous

HW 5 tips

- Looking only for 'CDS' features
- 'complement' indicates the reverse complement
- 'ORIGIN' section contains the actual sequence
- Positions downstream of the translation start site could be noncontiguous
  - join(1000..1008,1200..1500)

HW 5 tips

- Looking only for 'CDS' features
- 'complement' indicates the reverse complement
- 'ORIGIN' section contains the actual sequence
- Positions downstream of the translation start site could be noncontiguous
  - join(1000..1008,1200..1500)
- Precision matters! (use doubles in C++)

Watch out for multi-line joins

```
CDS             join(10183..10943,11138..11246,11408..11525,11697..11815,
                 12006..12056,12284..12445,12661..12792,12989..13135,
                 13293..13400,13597..13661,13848..13957,14104..14208,
                 14364..14440,14606..14773,14909..15013)
                 /locus_tag="PTSG_00005"
                 /codon_start=1
                 /product="hypothetical protein"
                 /protein_id="EGD71989.1"
                 /db_xref="GI:326426419"
                 /translation="MMMMMMMMRPCCSLPSTWWLVVVVLAAACCAATPTAAAVPAAAP
                 AEAADPSVVNVGQFVVSLDEDGVLSAVRNPAQMPNPHLAWHSTGEILEVAASKMYLHG...“
```

Weight matrix definition

- $\log_2$(frequency of base in start site/background frequency of base)

- use -99 if frequency is zero (alternative to pseudocounts)

Score histogram for CDS and all sites

- Bins labeled with integer values
  - Round scores down to determine the bin
- Print all bins with at least one count
- Put all scores less than -50 into one bin

```
Score Histogram All:
-5 101880
-4 76413
-3 54704
-2 38081
-1 27202
0 21440
1 18671
2 18825
3 19072
4 18675
5 17308
6 14429
7 10595
8 6915
9 3886
10 1850
11 699
12 225
13 46
14 4
lt-50 6132782
```

HW 5 questions?

More general motif-finding problem

Sequence 1 G T A C T A T C C A G C T A T C G G T

Sequence 2 T A G G G C A A C T T T T C A G T C A

Sequence 3 A C G T C A T A T G G A T C T C G G A

Sequence 4 T T C A A A G C A A C C C A A A T A G

Sequence 5 C T T G G A A C T G G T T A T C A G T

Sequence 6 A C G A T G C C A T T A C C A T A A T

Sequence 7 A A A G A T C A G T A T G G C A C T A

More general motif-finding problem

- Basic idea:
  - Given a set of *t* sequences of length *n*
    - Find a set of *k*-mers with maximum consensus score
    - One *k*-mer from each sequence

More general motif-finding problem

Sequence 1 G T A C T A T C C A G C T A T C G G T

Sequence 2 T A G G G C A A C T T T T C A G T C A

Sequence 3 A C G T C A T A T G G A T C T C G G A

Sequence 4 T T C A A A G C A A C C C A A A T A G

Sequence 5 C T T G G A A C T G G T T A T C A G T

Sequence 6 A C G A T G C C A T T A C C A T A A T

Sequence 7 A A A G A T C A G T A T G G C A C T A

More general motif-finding problem

Sequence 1 A T C C A G C T

Sequence 2 G G G C A A C T

Sequence 3 A T G G A T C T

Sequence 4 A A G C A A C C

Sequence 5 T T G G A A C T

Sequence 6 A T G C C A T T

Sequence 7 A T G G C A C T

More general motif-finding problem

Sequence 1	A	T	C	C	A	G	C	T
Sequence 2	G	G	G	C	A	A	C	T
Sequence 3	A	T	G	G	A	T	C	T
Sequence 4	A	A	G	C	A	A	C	C
Sequence 5	T	T	G	G	A	A	C	T
Sequence 6	A	T	G	C	C	A	T	T
Sequence 7	A	T	G	G	C	A	C	T

A	5	1	0	0	5	5	0	0

More general motif-finding problem

```
Sequence 1 A T C C A G C T          A 5 1 0 0 5 5 0 0
                                    T 1 5 0 0 0 1 1 6
Sequence 2 G G G C A A C T

Sequence 3 A T G G A T C T

Sequence 4 A A G C A A C C

Sequence 5 T T G G A A C T

Sequence 6 A T G C C A T T

Sequence 7 A T G G C A C T
```

More general motif-finding problem

Sequence 1 A T C C A G C T

Sequence 2 G G G C A A C T

Sequence 3 A T G G A T C T

Sequence 4 A A G C A A C C

Sequence 5 T T G G A A C T

Sequence 6 A T G C C A T T

Sequence 7 A T G G C A C T

A	5	1	0	0	5	5	0	0
T	1	5	0	0	0	1	1	6
G	1	1	6	3	0	1	0	0

More general motif-finding problem

Sequence 1 A T C C A G C T

Sequence 2 G G G C A A C T

Sequence 3 A T G G A T C T

Sequence 4 A A G C A A C C

Sequence 5 T T G G A A C T

Sequence 6 A T G C C A T T

Sequence 7 A T G G C A C T

A	5	1	0	0	5	5	0	0
T	1	5	0	0	0	1	1	6
G	1	1	6	3	0	1	0	0
C	0	0	1	4	2	0	6	1

More general motif-finding problem

Sequence 1 A T C C A G C T

Sequence 2 G G G C A A C T

Sequence 3 A T G G A T C T

Sequence 4 A A G C A A C C

Sequence 5 T T G G A A C T

Sequence 6 A T G C C A T T

Sequence 7 A T G G C A C T

A	5	1	0	0	5	5	0	0
T	1	5	0	0	0	1	1	6
G	1	1	6	3	0	1	0	0
C	0	0	1	4	2	0	6	1

Consensus:
A T G C A A C T

More general motif-finding problem

Sequence 1 A T C C A G C T

Sequence 2 G G G C A A C T

Sequence 3 A T G G A T C T

Sequence 4 A A G C A A C C

Sequence 5 T T G G A A C T

Sequence 6 A T G C C A T T

Sequence 7 A T G G C A C T

A	5	1	0	0	5	5	0	0
T	1	5	0	0	0	1	1	6
G	1	1	6	3	0	1	0	0
C	0	0	1	4	2	0	6	1

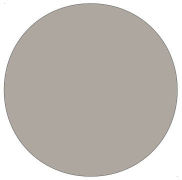
Consensus:
A T G C A A C T

Score:
5+5+6+4+5+5+6+6
=42

Motif search tree representation

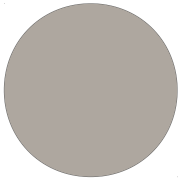
Motif search tree representation

Root



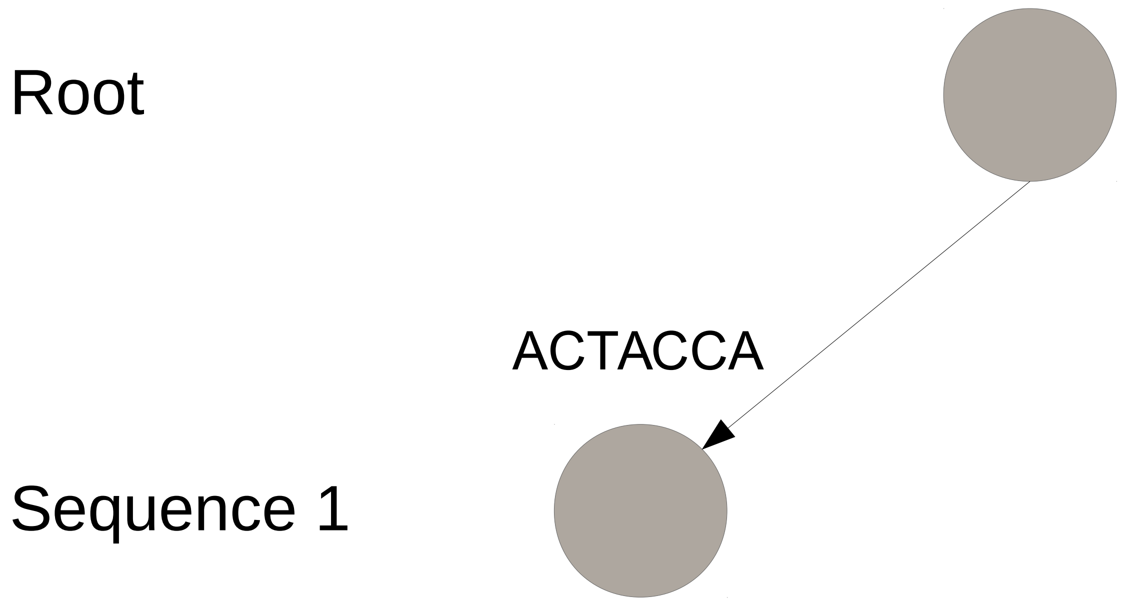
Motif search tree representation

Root



Sequence 1

Motif search tree representation

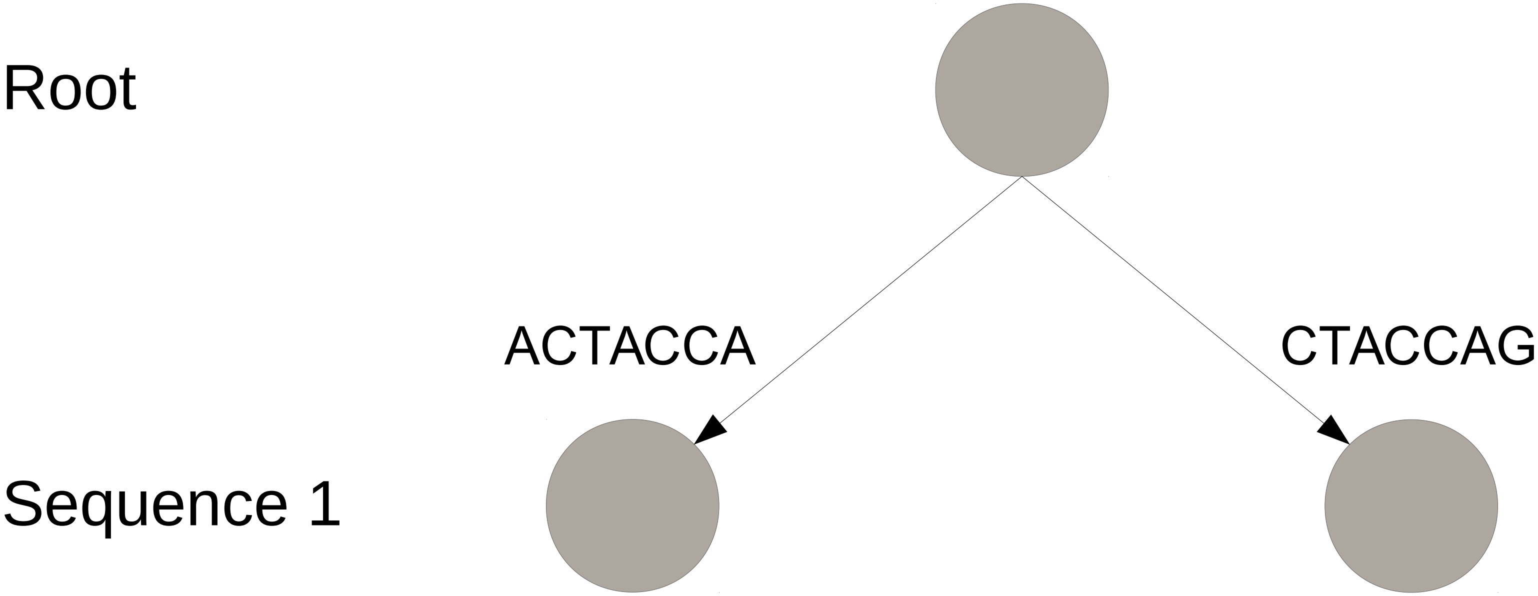


Root

ACTACCA

Sequence 1

Motif search tree representation



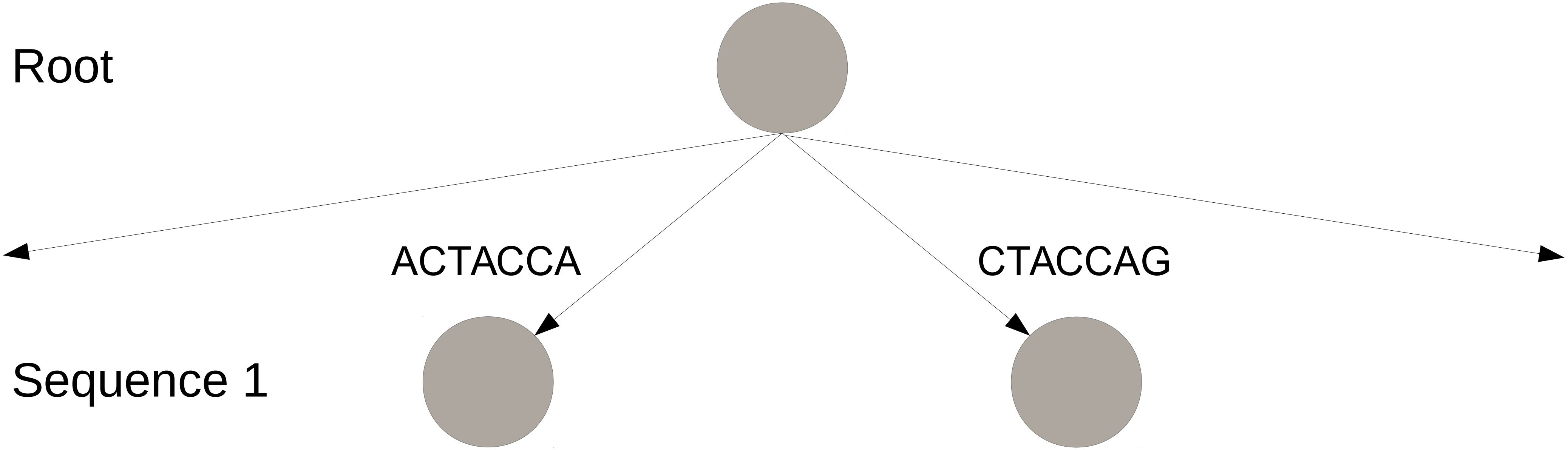
Root

ACTACCA

CTACCAG

Sequence 1

Motif search tree representation



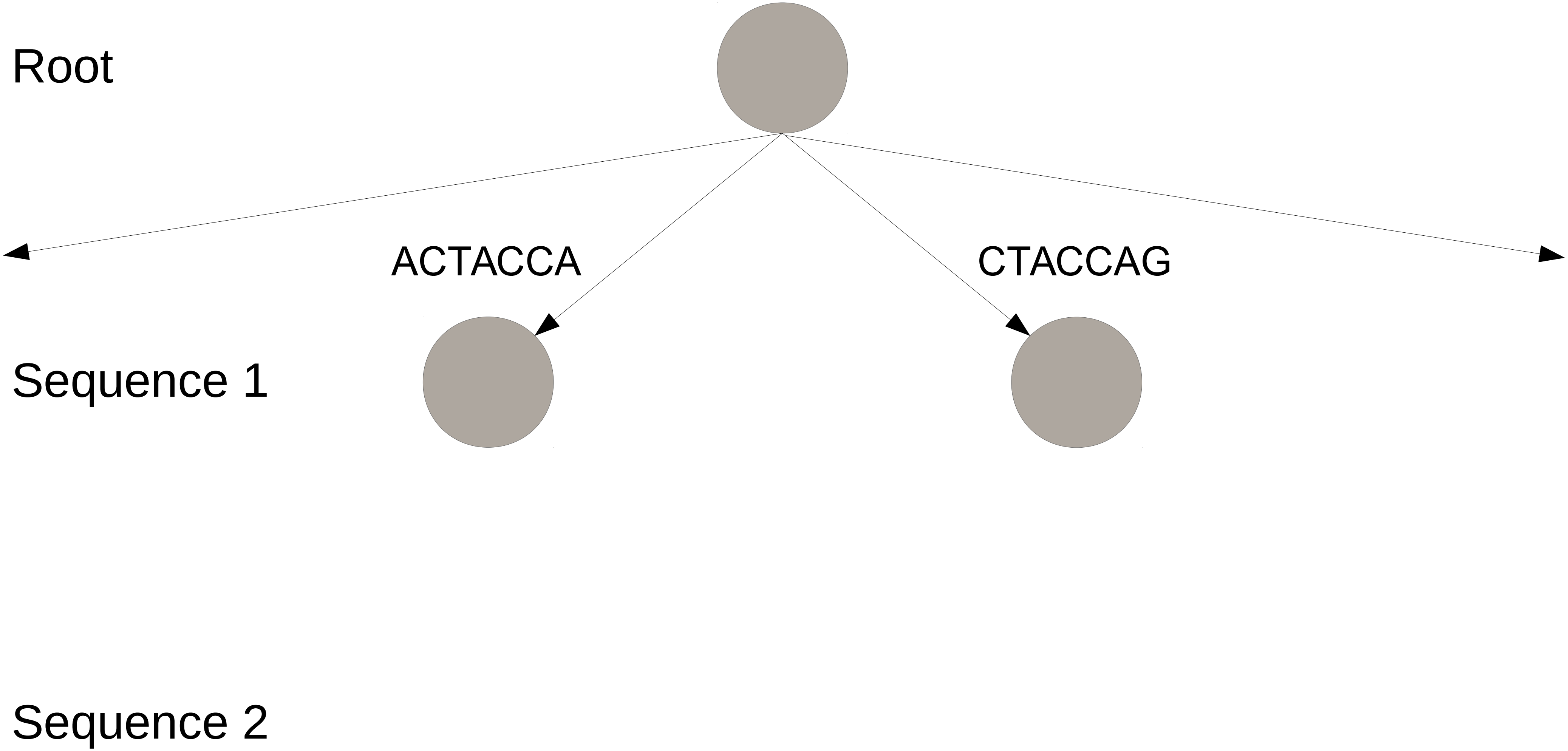
Root

ACTACCA

CTACCAG

Sequence 1

Motif search tree representation



Root

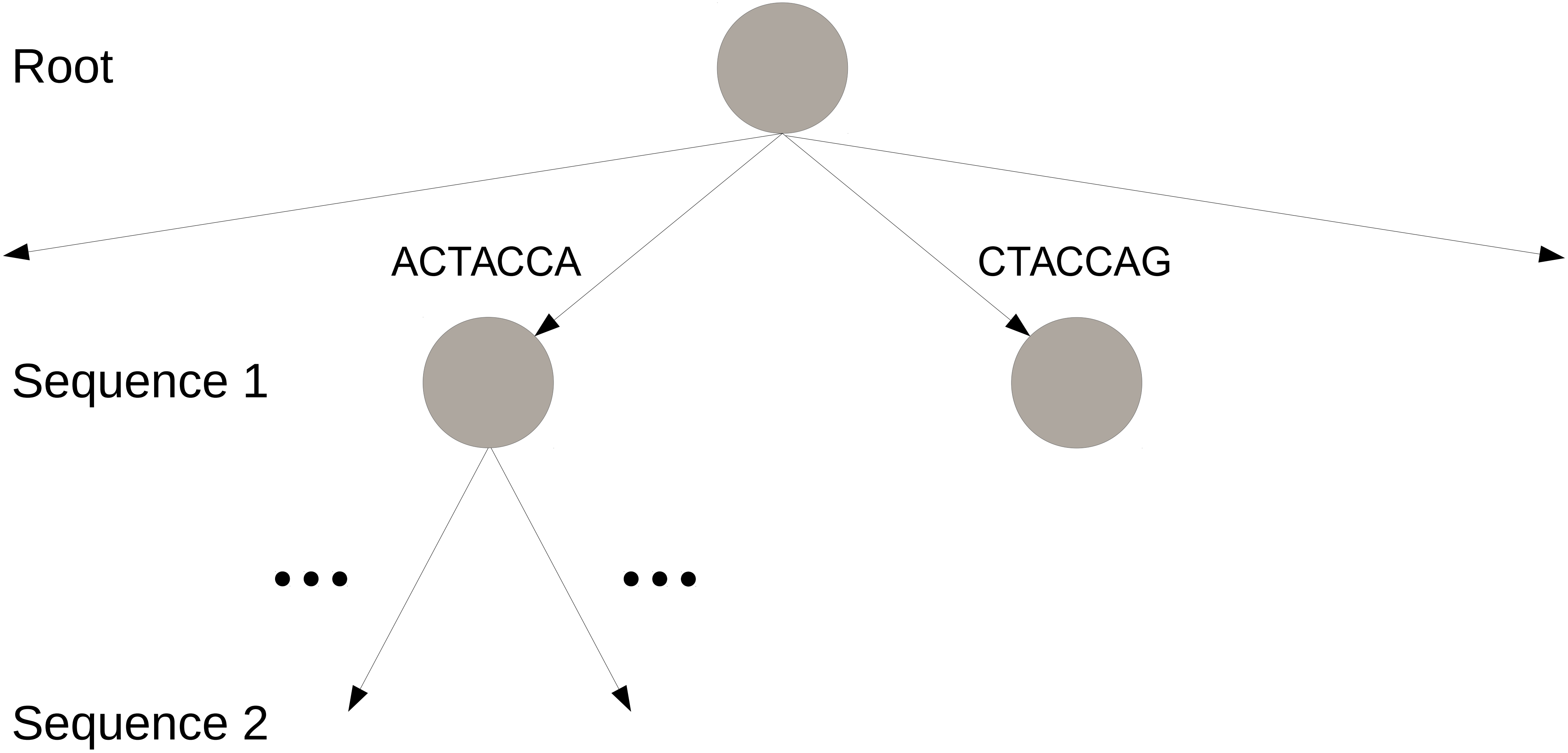
ACTACCA

CTACCAG

Sequence 1

Sequence 2

Motif search tree representation



Root

ACTACCA

CTACCAG

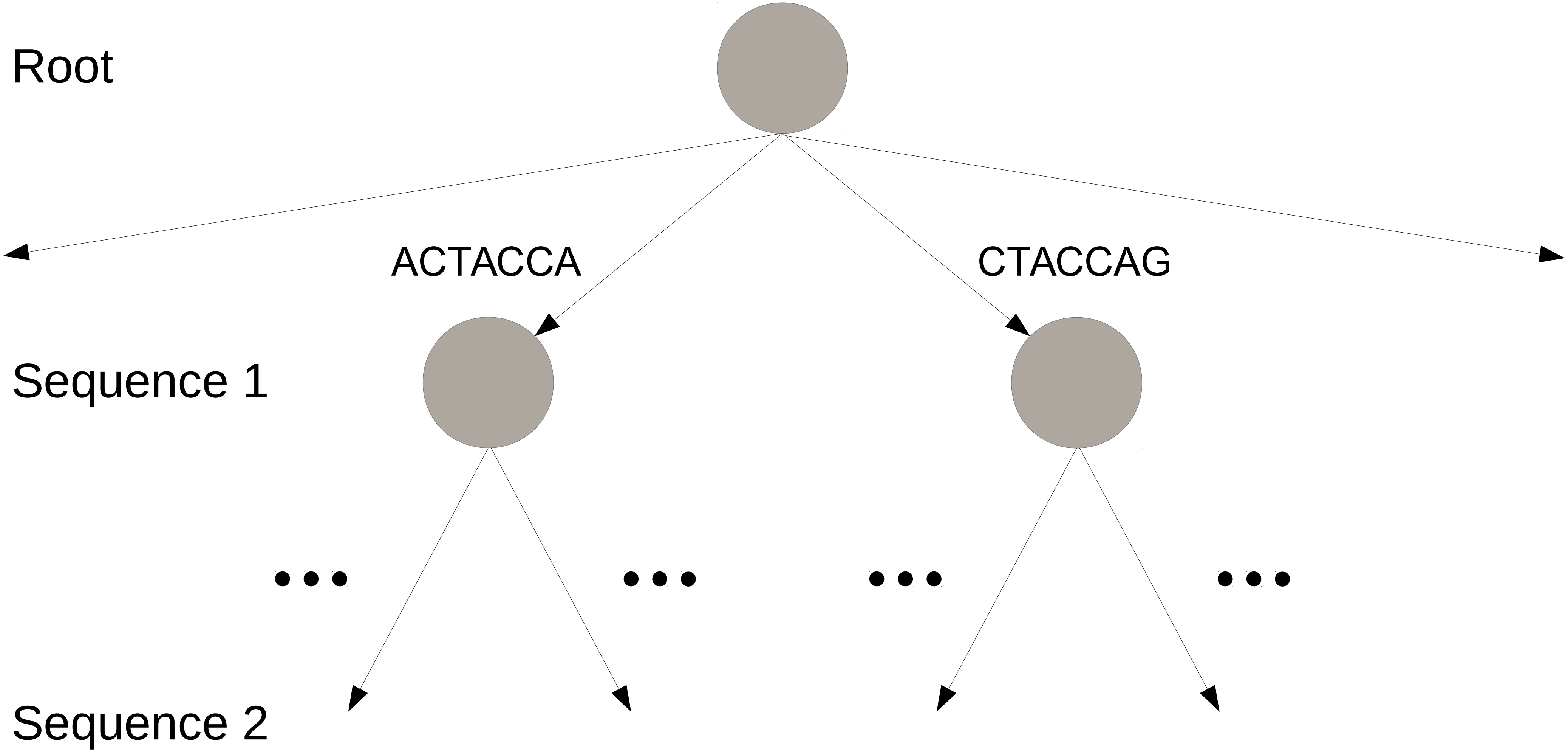
Sequence 1

• • •

• • •

Sequence 2

Motif search tree representation



Root

ACTACCA

CTACCAG

Sequence 1

• • •

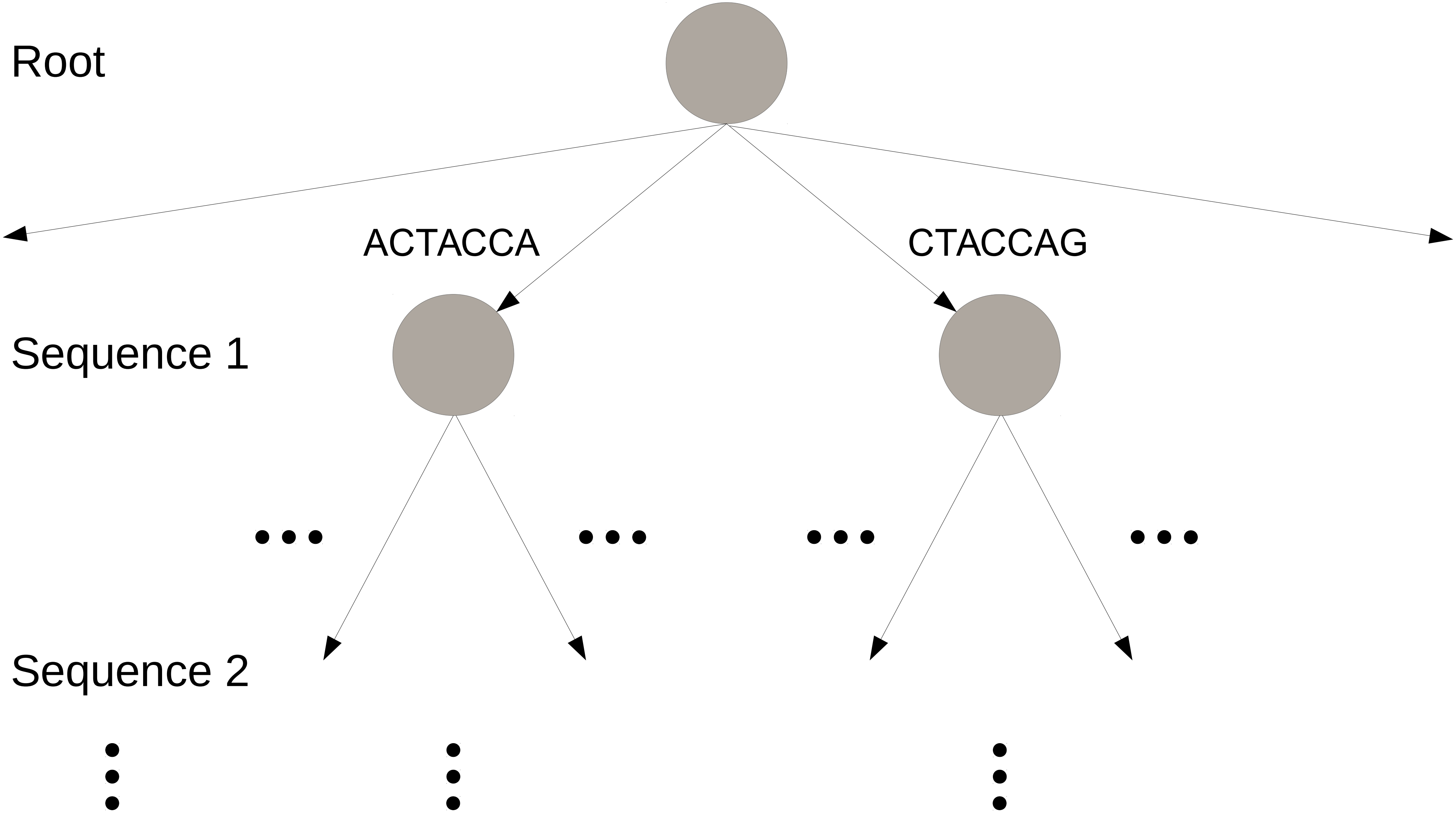
• • •

• • •

• • •

Sequence 2

Motif search tree representation



Motif search tree representation

- Each vertex represents a motif start location

Motif search tree representation

- Each vertex represents a motif start location

- Each level (except the root) corresponds to a sequence (*t* levels)

Motif search tree representation

- Each vertex represents a motif start location

- Each level (except the root) corresponds to a sequence (*t* levels)

- *n* – *k* + 1 children per vertex (possible motif start locations in next sequence)

Motif search tree representation

- Each vertex represents a motif start location
- Each level (except the root) corresponds to a sequence (*t* levels)
- *n* – *k* + 1 children per vertex (possible motif start locations in next sequence)
- Leaf vertices are complete motif sets

Motif search tree brute force solution

- Brute force algorithm:

Motif search tree brute force solution

- Brute force algorithm:
  - Traverse the tree in some order

Motif search tree brute force solution

- Brute force algorithm:
  - Traverse the tree in some order
  - At each leaf, calculate the score for the set of starting positions

Motif search tree brute force solution

- Brute force algorithm:
  - Traverse the tree in some order
  - At each leaf, calculate the score for the set of starting positions
  - Keep track of the best score seen

Motif search tree brute force solution

- Brute force algorithm:
  - Traverse the tree in some order
  - At each leaf, calculate the score for the set of starting positions
  - Keep track of the best score seen

- O(kn^t)

Motif search tree brute force solution

- Brute force algorithm:
  - Traverse the tree in some order
  - At each leaf, calculate the score for the set of starting positions
  - Keep track of the best score seen
- O(kn^t)
- What could we do better?

Motif search tree branch-and-bound improvement

- Basic idea

Motif search tree branch-and-bound improvement

- Basic idea
  - At each vertex, determine a bound on the score

Motif search tree branch-and-bound improvement

- Basic idea
  - At each vertex, determine a bound on the score
  - If the bound is too low, don't use this branch

Motif search tree branch-and-bound improvement

- Basic idea
  - At each vertex, determine a bound on the score
  - If the bound is too low, don't use this branch
- More specifically

Motif search tree branch-and-bound improvement

- Basic idea
  - At each vertex, determine a bound on the score
  - If the bound is too low, don't use this branch

- More specifically
  - Given the partial consensus for *i* sequences chosen

Motif search tree branch-and-bound improvement

- Basic idea
  - At each vertex, determine a bound on the score
  - If the bound is too low, don't use this branch

- More specifically
  - Given the partial consensus for *i* sequences chosen
    - The rest of the sequences can improve the score by at most (t - i) x k

Motif search tree branch-and-bound improvement

- Basic idea
  - At each vertex, determine a bound on the score
  - If the bound is too low, don't use this branch

- More specifically
  - Given the partial consensus for *i* sequences chosen
    - The rest of the sequences can improve the score by at most (*t* - *i*) x *k*
      - When does this happen?

Motif search tree branch-and-bound improvement

- Basic idea
  - At each vertex, determine a bound on the score
  - If the bound is too low, don't use this branch

- More specifically
  - Given the partial consensus for *i* sequences chosen
    - The rest of the sequences can improve the score by at most (*t* - *i*) x *k*
      - When does this happen? The rest match the partial consensus

Motif search tree branch-and-bound improvement

- Basic idea
  - At each vertex, determine a bound on the score
  - If the bound is too low, don't use this branch

- More specifically
  - Given the partial consensus for *i* sequences chosen
    - The rest of the sequences can improve the score by at most (*t* - *i*) x *k*
      - When does this happen? The rest match the partial consensus
  - So if current score + (*t* - *i*) x *k* is less than the best score so far, don't bother checking

Another way to consider the problem: median string

Another way to consider the problem: median string

- Just phrased a different way

Another way to consider the problem: median string

- Just phrased a different way
  - Given a set of *t* sequences, each of length *n*

Another way to consider the problem: median string

- Just phrased a different way
  - Given a set of *t* sequences, each of length *n*
    - Find the string V of length *k* that minimized the Hamming distance between V and one *k*-mer from each sequence

Another way to consider the problem: median string

- Just phrased a different way
    - Given a set of *t* sequences, each of length *n*
        - Find the string V of length *k* that minimized the Hamming distance between V and one *k*-mer from each sequence
        - Hamming distance is just the number of different positions

Another way to consider the problem: median string

- Just phrased a different way
  - Given a set of *t* sequences, each of length *n*
    - Find the string V of length *k* that minimized the Hamming distance between V and one *k*-mer from each sequence
    - Hamming distance is just the number of different positions

- How many possibilities for V?

Another way to consider the problem: median string

- Just phrased a different way
  - Given a set of *t* sequences, each of length *n*
    - Find the string V of length *k* that minimized the Hamming distance between V and one *k*-mer from each sequence
    - Hamming distance is just the number of different positions

- How many possibilities for V?
  - 4^k

Median string search tree representation

Median string search tree representation

- Each vertex represents a base at a certain position in the median string

Median string search tree representation

- Each vertex represents a base at a certain position in the median string

- Each level (except the root) corresponds to a median string position (1st level is position 1, etc.)

Median string search tree representation

- Each vertex represents a base at a certain position in the median string
- Each level (except the root) corresponds to a median string position (1st level is position 1, etc.)
- 4 children per vertex (one for each possible next base in the median string

Median string search tree representation

- Each vertex represents a base at a certain position in the median string

- Each level (except the root) corresponds to a median string position (1st level is position 1, etc.)

- 4 children per vertex (one for each possible next base in the median string

- Leaf vertices are complete median strings

Median string search tree brute force solution

- Brute force algorithm:

Median string search tree brute force solution

- Brute force algorithm:
    - For each leaf, check for the best-scoring match in each sequence individually

Median string search tree brute force solution

- Brute force algorithm:
  - For each leaf, check for the best-scoring match in each sequence individually

- $O\left(4^k nt\right)$

Median string search tree brute force solution

- Brute force algorithm:
  - For each leaf, check for the best-scoring match in each sequence individually

- $O\left(4^k nt\right)$

- Can we use branch-and-bound again?

Median string search tree branch-and-bound improvement

Median string search tree branch-and-bound improvement

- What can we do while checking scores for a candidate median string?

Median string search tree branch-and-bound improvement

- What can we do while checking scores for a candidate median string?
  - If we've found the smallest distance match for a sequence, what does that tell us about the best total score for the candidate?

Median string search tree branch-and-bound improvement

- What can we do while checking scores for a candidate median string?
  - If we've found the smallest distance match for a sequence, what does that tell us about the best total score for the candidate?
  - In general, how does the score change as we look at more sequences?

Median string search tree branch-and-bound improvement

- What can we do while checking scores for a candidate median string?
  - If we've found the smallest distance match for a sequence, what does that tell us about the best total score for the candidate?
  - In general, how does the score change as we look at more sequences?
  - As soon as the current score for the candidate is greater than the best (lowest) score seen, move on to the next candidate

Branch-and-bound methods can help in practice, but don't actually improve the worst-case time

Greedy motif search

Greedy motif search

- Scan each sequence only once

Greedy motif search

- Scan each sequence only once
  - Find the best *k*-mer pair match between two sequences

Greedy motif search

- Scan each sequence only once
  - Find the best *k*-mer pair match between two sequences
  - Add on the best-matching *k*-mer from each other sequence one at a time

Greedy motif search

- Scan each sequence only once
  - Find the best *k*-mer pair match between two sequences
  - Add on the best-matching *k*-mer from each other sequence one at a time

- CONSENSUS

Greedy motif search

- Scan each sequence only once
  - Find the best *k*-mer pair match between two sequences
  - Add on the best-matching *k*-mer from each other sequence one at a time

- CONSENSUS
  - Uses a greedy search as described except it stores *m* *k*-mers at each step

Greedy motif search

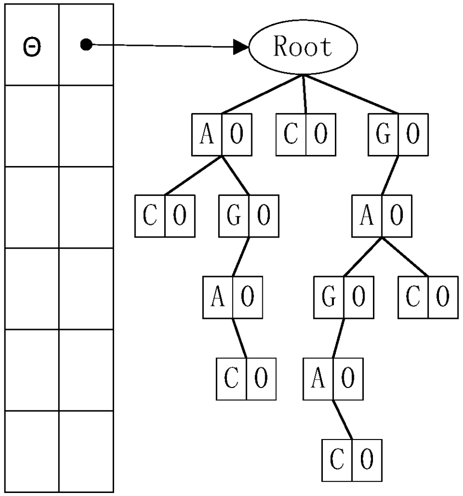
- Scan each sequence only once
  - Find the best *k*-mer pair match between two sequences
  - Add on the best-matching *k*-mer from each other sequence one at a time

- CONSENSUS
  - Uses a greedy search as described except it stores *m* *k*-mers at each step
    - Less likely to miss better ones

The WEEDER algorithm (2014)

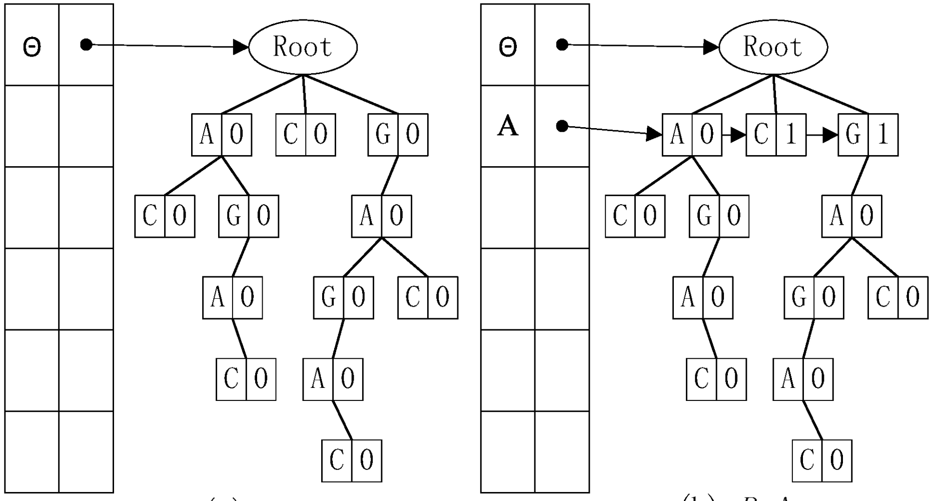
- Specifically looking for transcription factor (TF) binding sites
- Uses a range of motif sizes similar to observed TF binding sites
- Allows a specified number of differences (mutations) *d*
- Uses a 'mismatched' suffix tree to search sequences for candidate motif occurrences

Mismatched suffix tree



(a) P_0=Θ

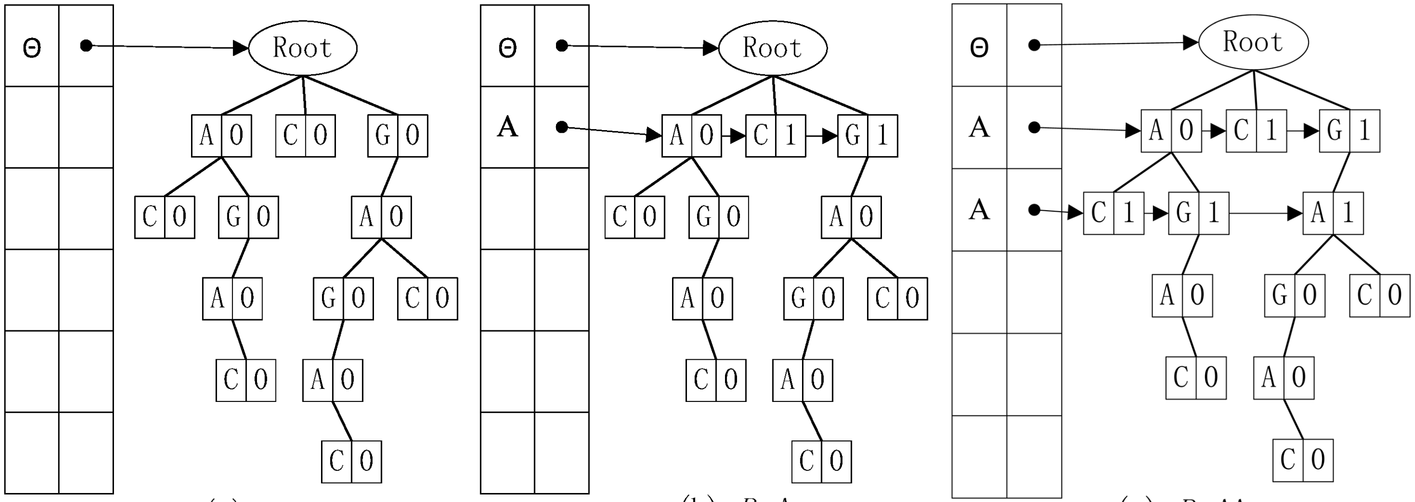
Mismatched suffix tree



(a) P_0=Θ

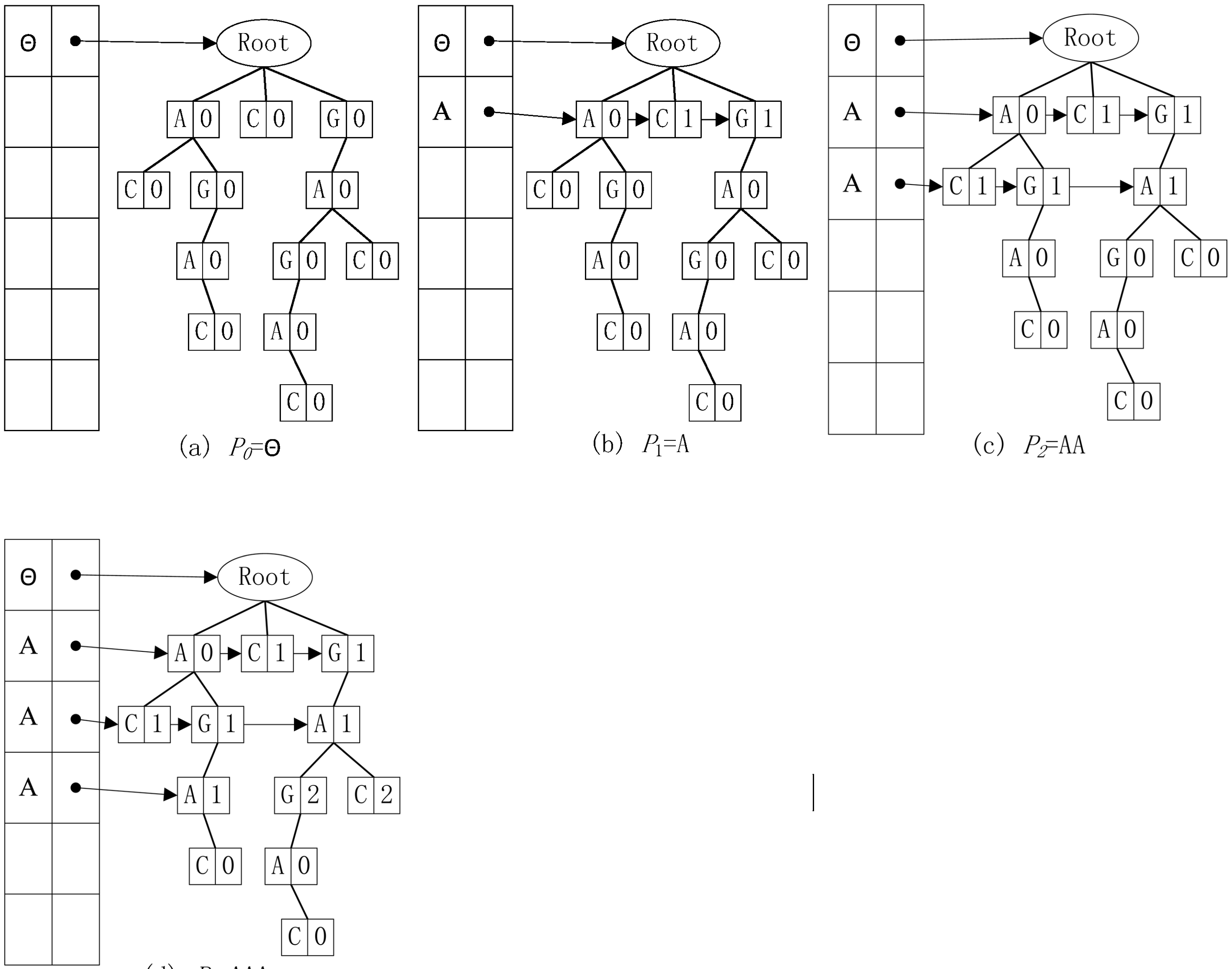
(b) P_1=A

Mismatched suffix tree



(a) P_0=Θ (b) P_1=A (c) P_2=AA

Mismatched suffix tree



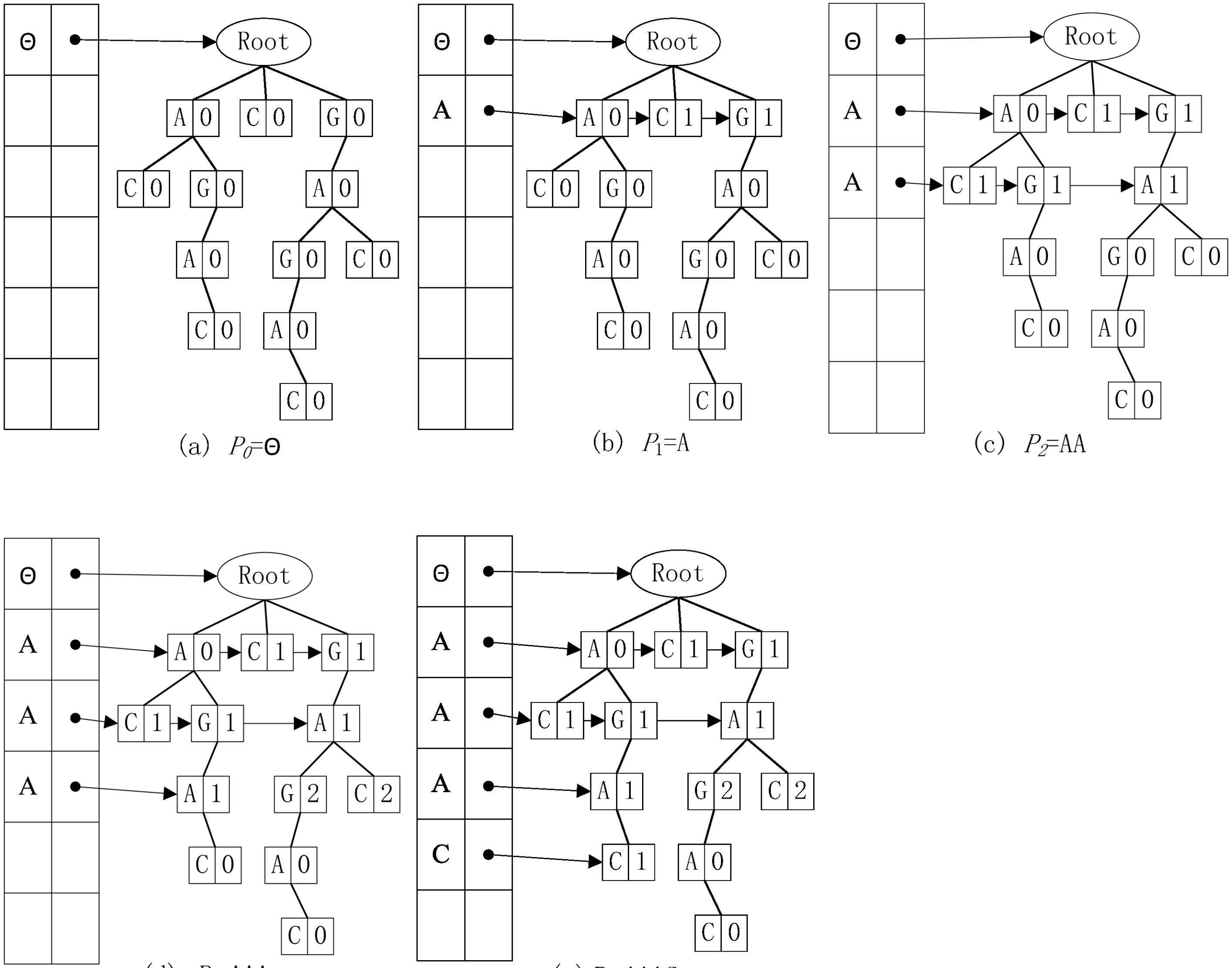
(a) P_0=Θ

(b) P_1=A

(c) P_2=AA

(d) P_3=AAA

Mismatched suffix tree



(a) P_0=Θ

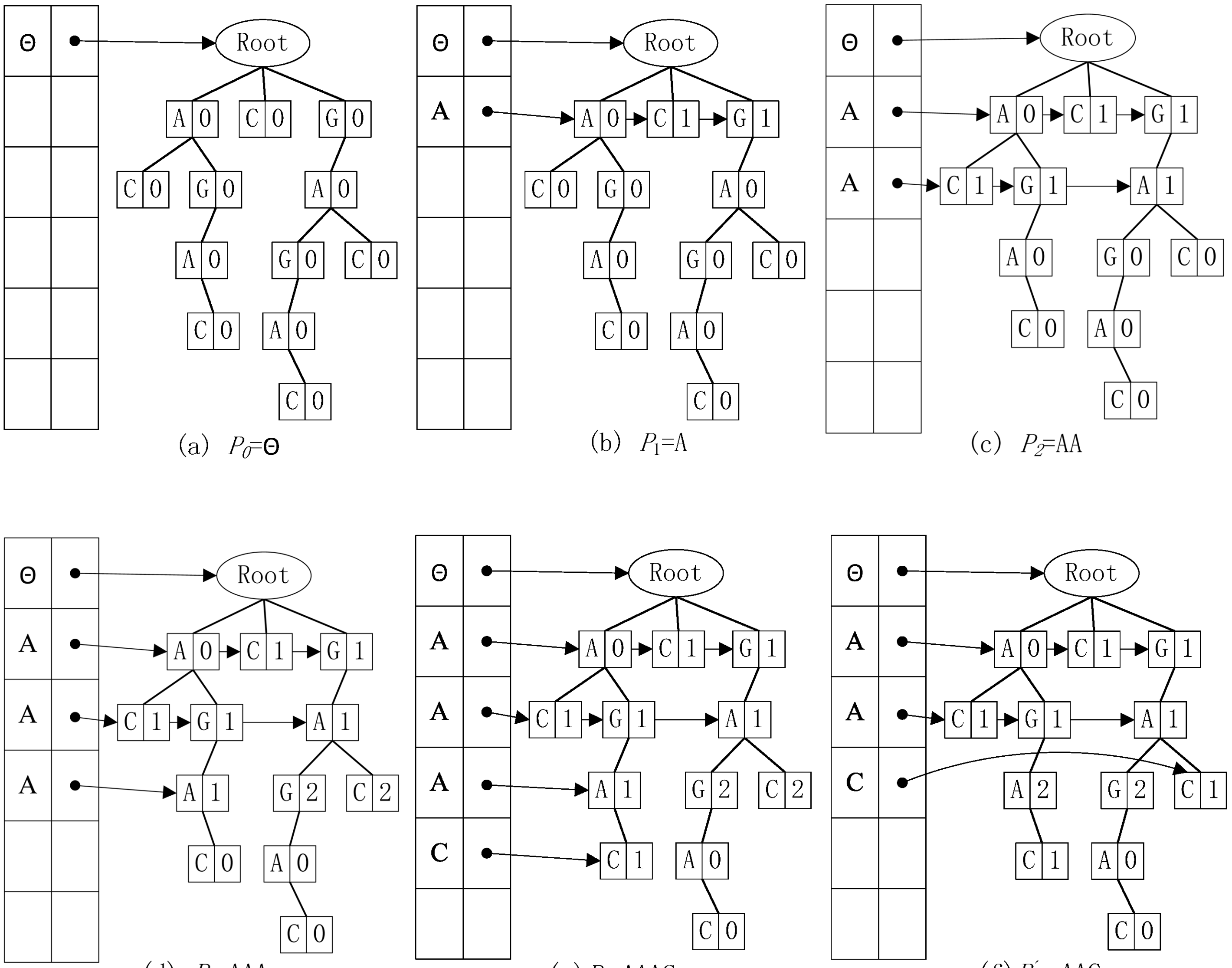
(b) P_1=A

(c) P_2=AA

(d) P_3=AAA

(e) P_4=AAAC

Mismatched suffix tree



(a) P_0=Θ

(b) P_1=A

(c) P_2=AA

(d) P_3=AAA

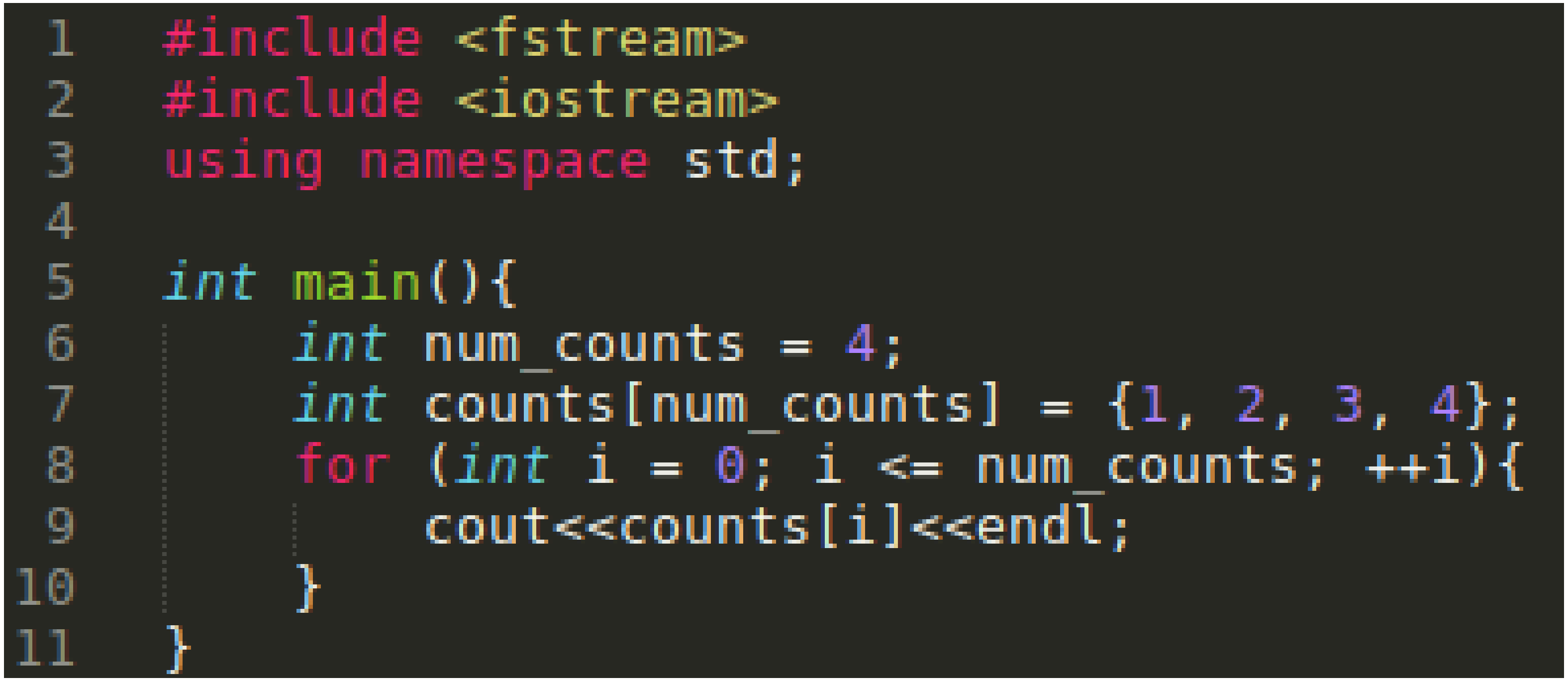
(e) P_4=AAAC

(f) P'_3=AAC

Using valgrind to check for memory bugs

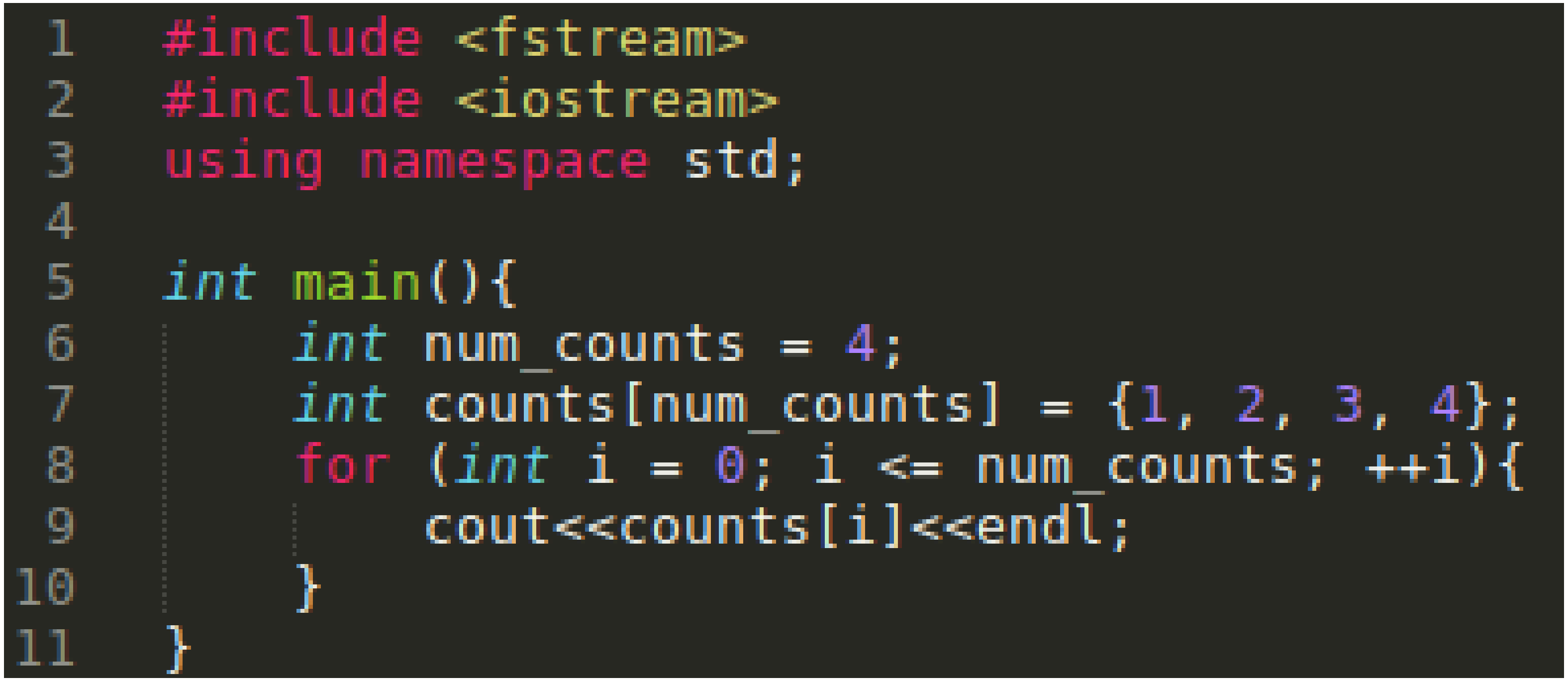
- Valgrind is a command line tool for profiling and checking program memory use
- If you compile with g++, then you just add the '-g' flag when compiling
- You can then run your program with valgrind and it gives detailed memory usage info
  - Sometimes a bit too detailed

Valgrind example #1:

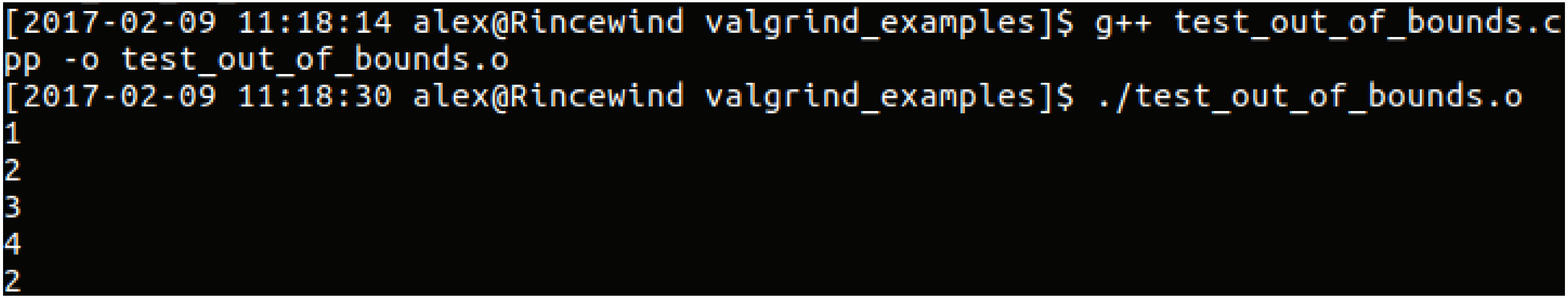


```
#include <fstream>
#include <iostream>
using namespace std;

int main(){
    int num_counts = 4;
    int counts[num_counts] = {1, 2, 3, 4};
    for (int i = 0; i <= num_counts; ++i){
        cout<<counts[i]<<endl;
    }
}
```

Valgrind example #1:

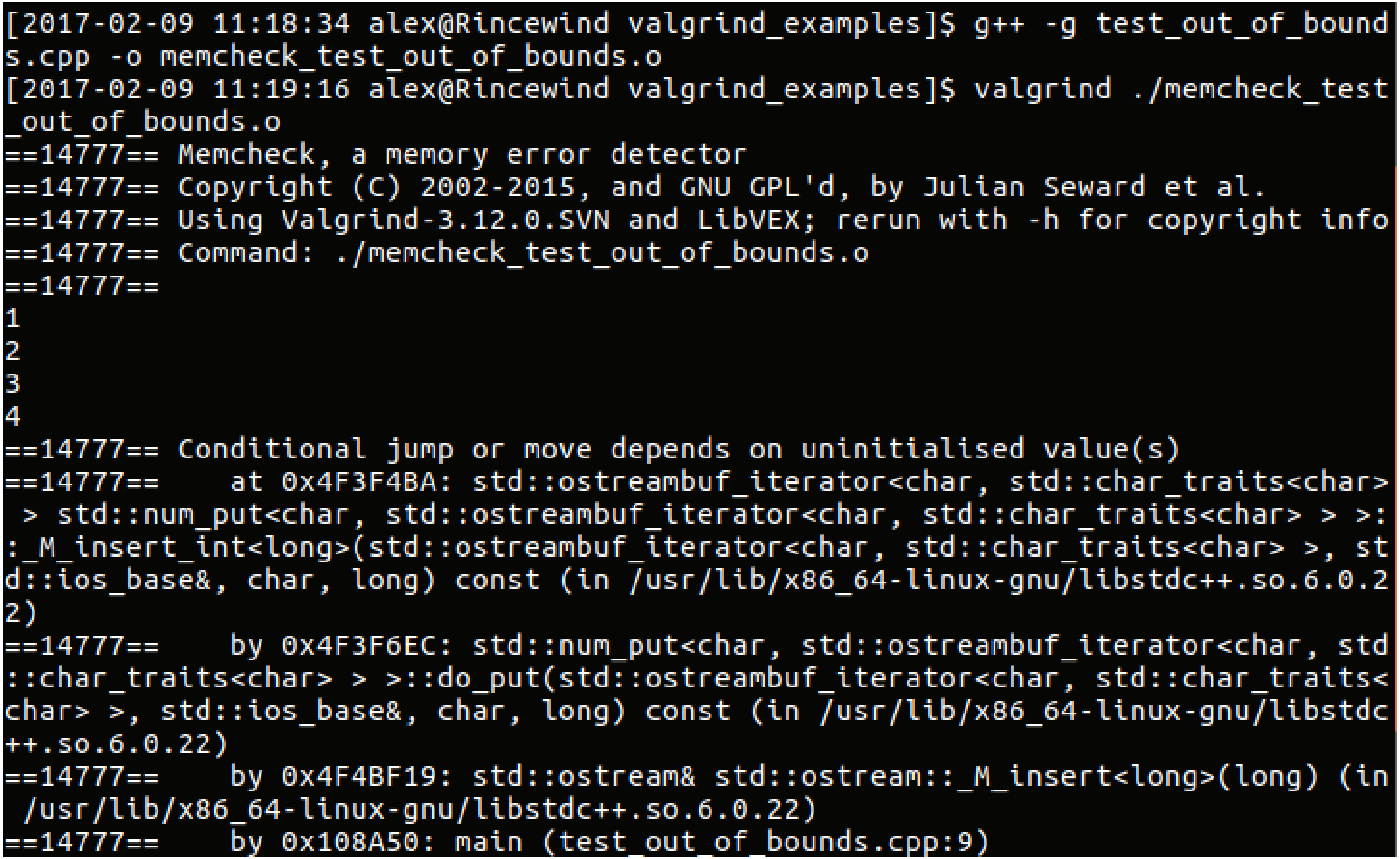


```cpp
#include <fstream>
#include <iostream>
using namespace std;

int main(){
    int num_counts = 4;
    int counts[num_counts] = {1, 2, 3, 4};
    for (int i = 0; i <= num_counts; ++i){
        cout<<counts[i]<<endl;
    }
}
```



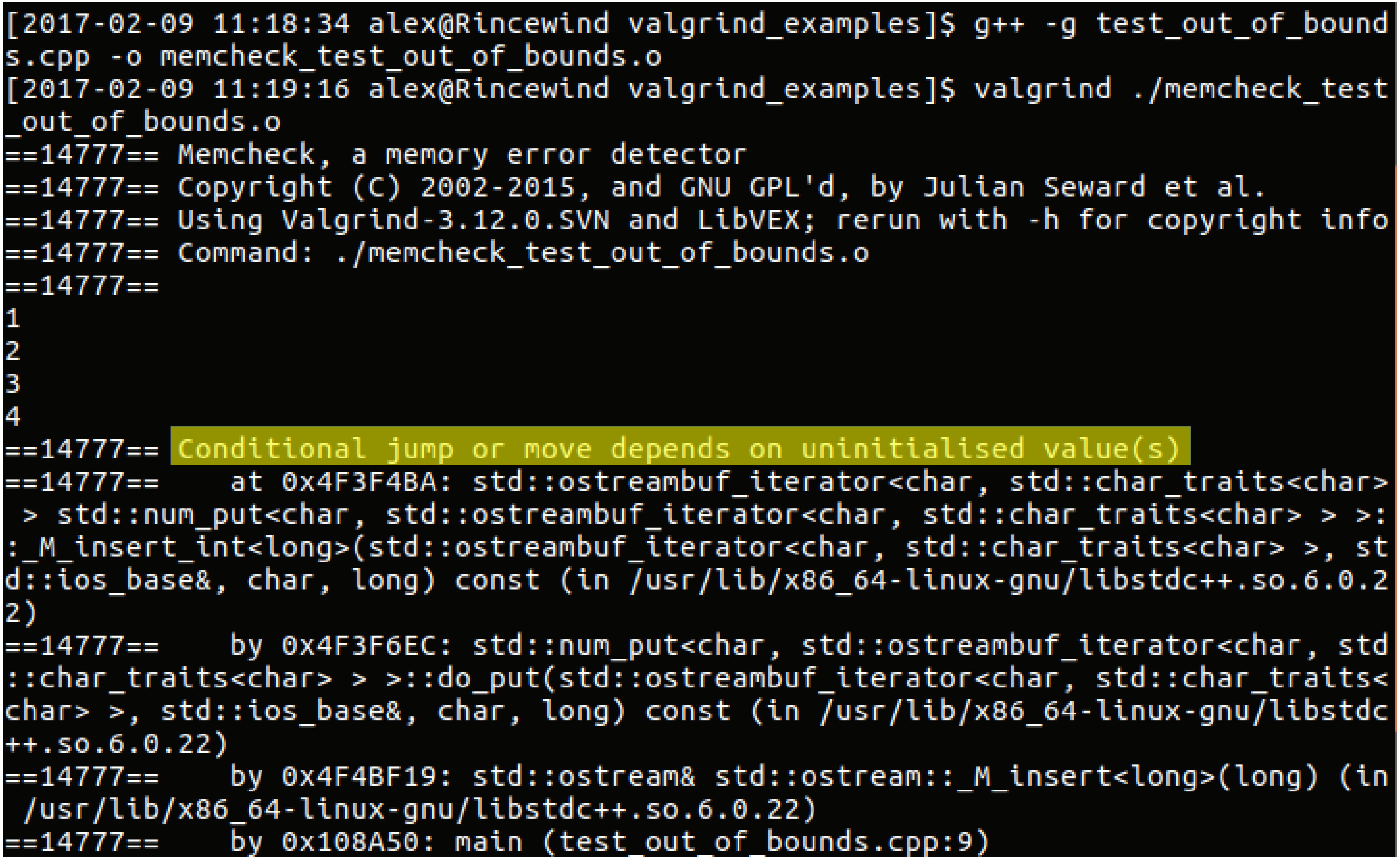
```
[2017-02-09 11:18:14 alex@Rincewind valgrind_examples]$ g++ test_out_of_bounds.c
pp -o test_out_of_bounds.o
[2017-02-09 11:18:30 alex@Rincewind valgrind_examples]$ ./test_out_of_bounds.o
1
2
3
4
2
```

Valgrind example #1:



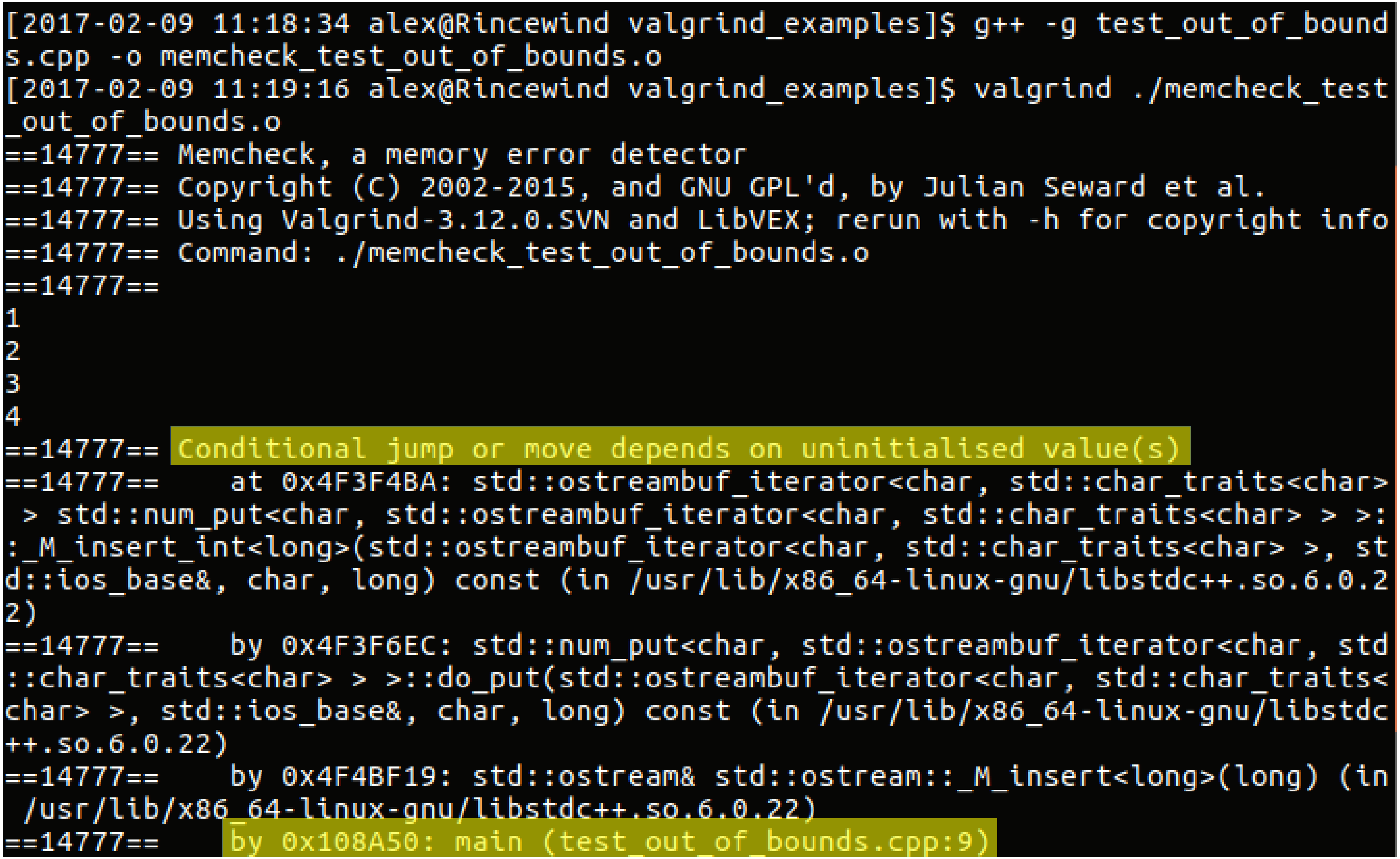
```
[2017-02-09 11:18:34 alex@Rincewind valgrind_examples]$ g++ -g test_out_of_bound
s.cpp -o memcheck_test_out_of_bounds.o
[2017-02-09 11:19:16 alex@Rincewind valgrind_examples]$ valgrind ./memcheck_test
_out_of_bounds.o
==14777== Memcheck, a memory error detector
==14777== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==14777== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==14777== Command: ./memcheck_test_out_of_bounds.o
==14777==
1
2
3
4
==14777== Conditional jump or move depends on uninitialised value(s)
==14777==    at 0x4F3F4BA: std::ostreambuf_iterator<char, std::char_traits<char>
 > std::num_put<char, std::ostreambuf_iterator<char, std::char_traits<char> > >:
:_M_insert_int<long>(std::ostreambuf_iterator<char, std::char_traits<char> >, st
d::ios_base&, char, long) const (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.2
2)
==14777==    by 0x4F3F6EC: std::num_put<char, std::ostreambuf_iterator<char, std
::char_traits<char> > >::do_put(std::ostreambuf_iterator<char, std::char_traits<
char> >, std::ios_base&, char, long) const (in /usr/lib/x86_64-linux-gnu/libstdc
++.so.6.0.22)
==14777==    by 0x4F4BF19: std::ostream& std::ostream::_M_insert<long>(long) (in
 /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.22)
==14777==    by 0x108A50: main (test_out_of_bounds.cpp:9)
```

Valgrind example #1:



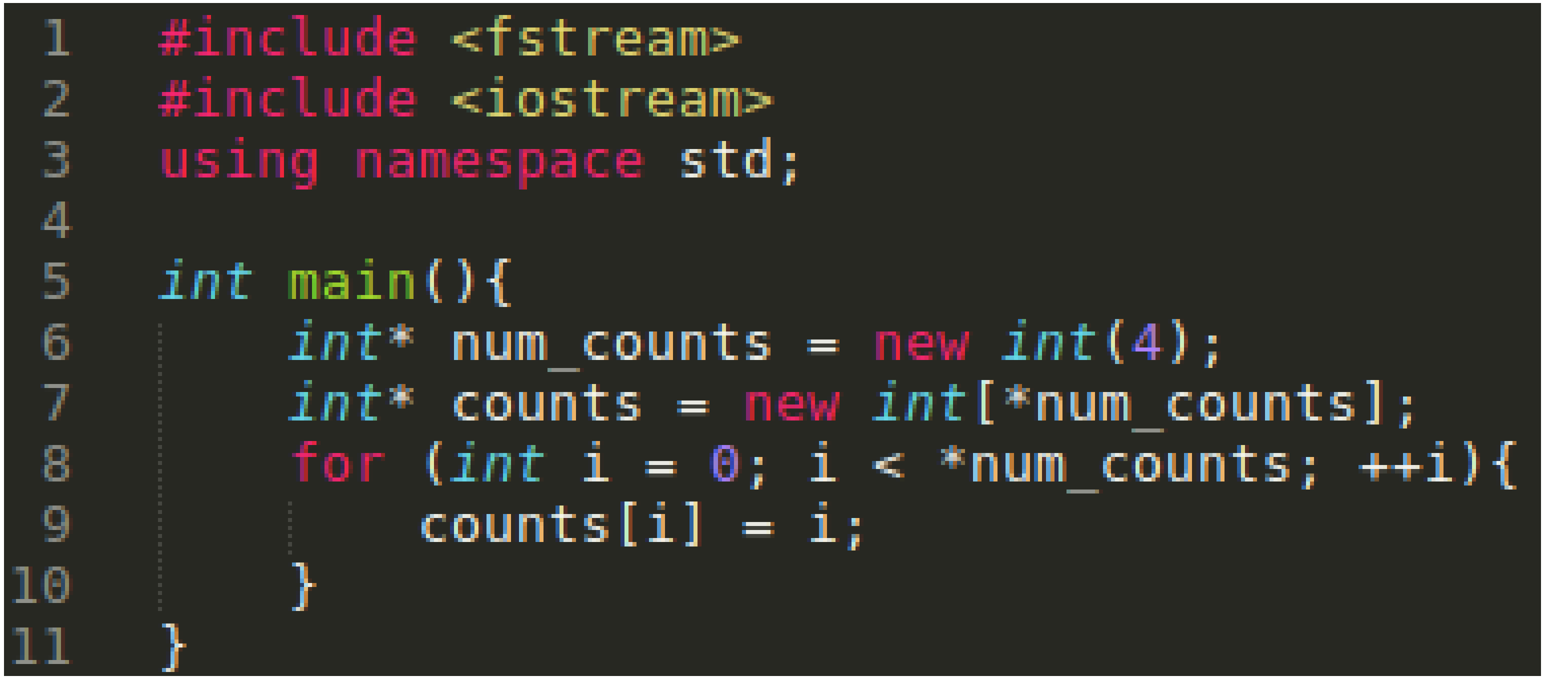
```
[2017-02-09 11:18:34 alex@Rincewind valgrind_examples]$ g++ -g test_out_of_bound
s.cpp -o memcheck_test_out_of_bounds.o
[2017-02-09 11:19:16 alex@Rincewind valgrind_examples]$ valgrind ./memcheck_test
_out_of_bounds.o
==14777== Memcheck, a memory error detector
==14777== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==14777== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==14777== Command: ./memcheck_test_out_of_bounds.o
==14777==
1
2
3
4
==14777== Conditional jump or move depends on uninitialised value(s)
==14777==    at 0x4F3F4BA: std::ostreambuf_iterator<char, std::char_traits<char>
 > std::num_put<char, std::ostreambuf_iterator<char, std::char_traits<char> > >:
:_M_insert_int<long>(std::ostreambuf_iterator<char, std::char_traits<char> >, st
d::ios_base&, char, long) const (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.2
2)
==14777==    by 0x4F3F6EC: std::num_put<char, std::ostreambuf_iterator<char, std
::char_traits<char> > >::do_put(std::ostreambuf_iterator<char, std::char_traits<
char> >, std::ios_base&, char, long) const (in /usr/lib/x86_64-linux-gnu/libstdc
++.so.6.0.22)
==14777==    by 0x4F4BF19: std::ostream& std::ostream::_M_insert<long>(long) (in
 /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.22)
==14777==    by 0x108A50: main (test_out_of_bounds.cpp:9)
```

Valgrind example #1:



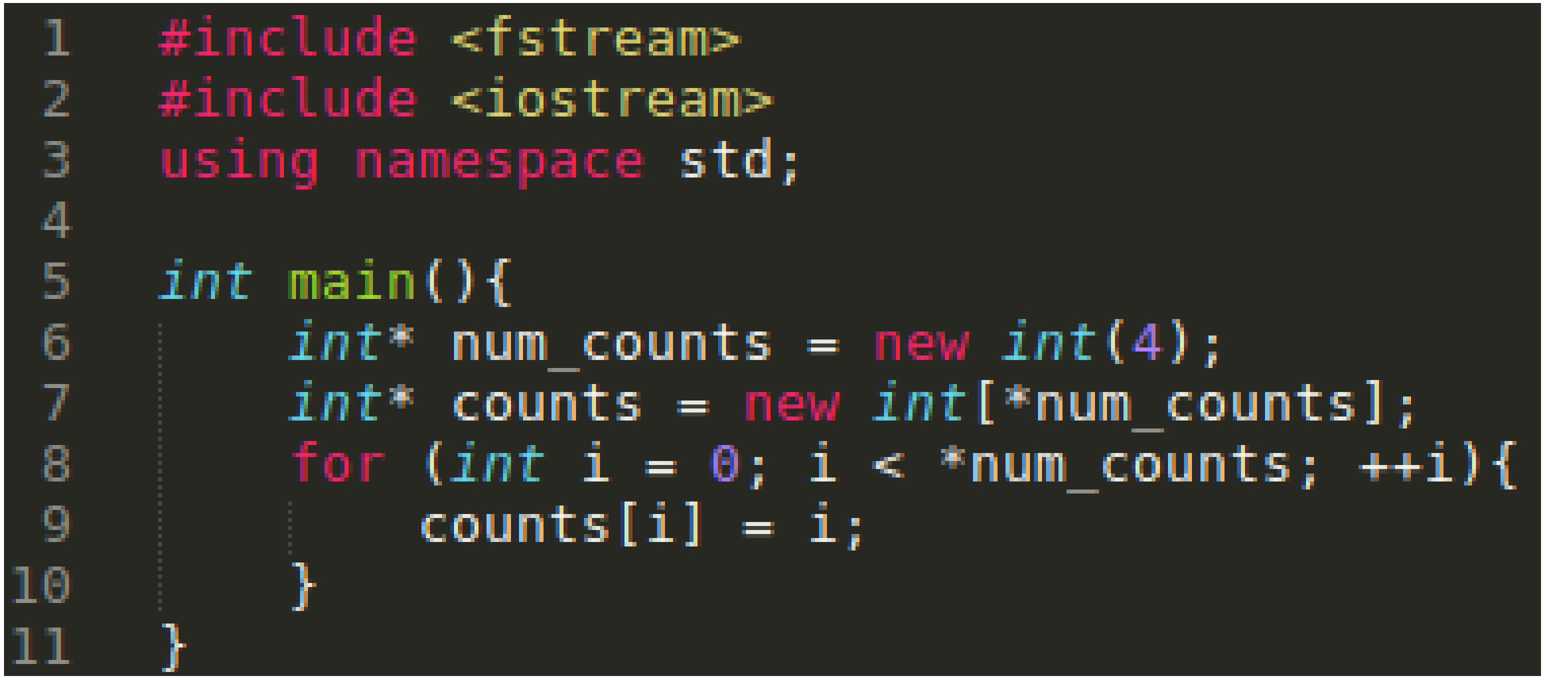
```
[2017-02-09 11:18:34 alex@Rincewind valgrind_examples]$ g++ -g test_out_of_bound
s.cpp -o memcheck_test_out_of_bounds.o
[2017-02-09 11:19:16 alex@Rincewind valgrind_examples]$ valgrind ./memcheck_test
_out_of_bounds.o
==14777== Memcheck, a memory error detector
==14777== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==14777== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==14777== Command: ./memcheck_test_out_of_bounds.o
==14777==
1
2
3
4
==14777== Conditional jump or move depends on uninitialised value(s)
==14777==    at 0x4F3F4BA: std::ostreambuf_iterator<char, std::char_traits<char>
 > std::num_put<char, std::ostreambuf_iterator<char, std::char_traits<char> > >:
:_M_insert_int<long>(std::ostreambuf_iterator<char, std::char_traits<char> >, st
d::ios_base&, char, long) const (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.2
2)
==14777==    by 0x4F3F6EC: std::num_put<char, std::ostreambuf_iterator<char, std
::char_traits<char> > >::do_put(std::ostreambuf_iterator<char, std::char_traits<
char> >, std::ios_base&, char, long) const (in /usr/lib/x86_64-linux-gnu/libstdc
++.so.6.0.22)
==14777==    by 0x4F4BF19: std::ostream& std::ostream::_M_insert<long>(long) (in
 /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.22)
==14777==    by 0x108A50: main (test_out_of_bounds.cpp:9)
```

Valgrind example #2:

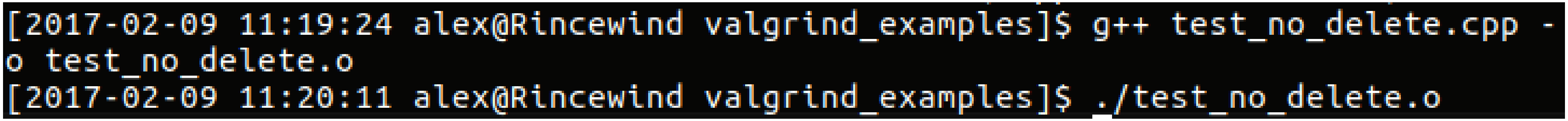


```cpp
#include <fstream>
#include <iostream>
using namespace std;

int main(){
    int* num_counts = new int(4);
    int* counts = new int[*num_counts];
    for (int i = 0; i < *num_counts; ++i){
        counts[i] = i;
    }
}
```

Valgrind example #2:

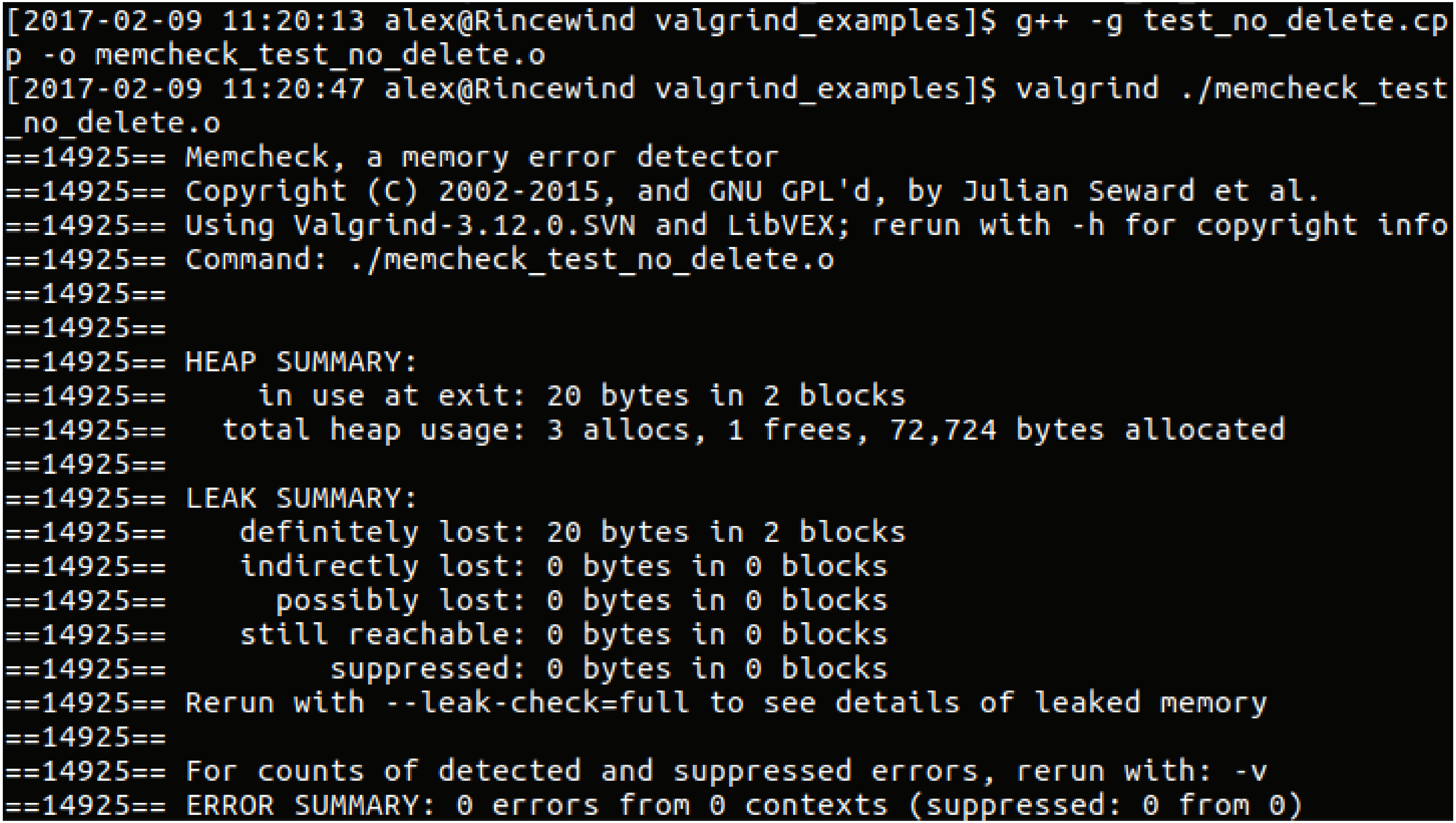


```cpp
#include <fstream>
#include <iostream>
using namespace std;

int main(){
    int* num_counts = new int(4);
    int* counts = new int[*num_counts];
    for (int i = 0; i < *num_counts; ++i){
        counts[i] = i;
    }
}
```



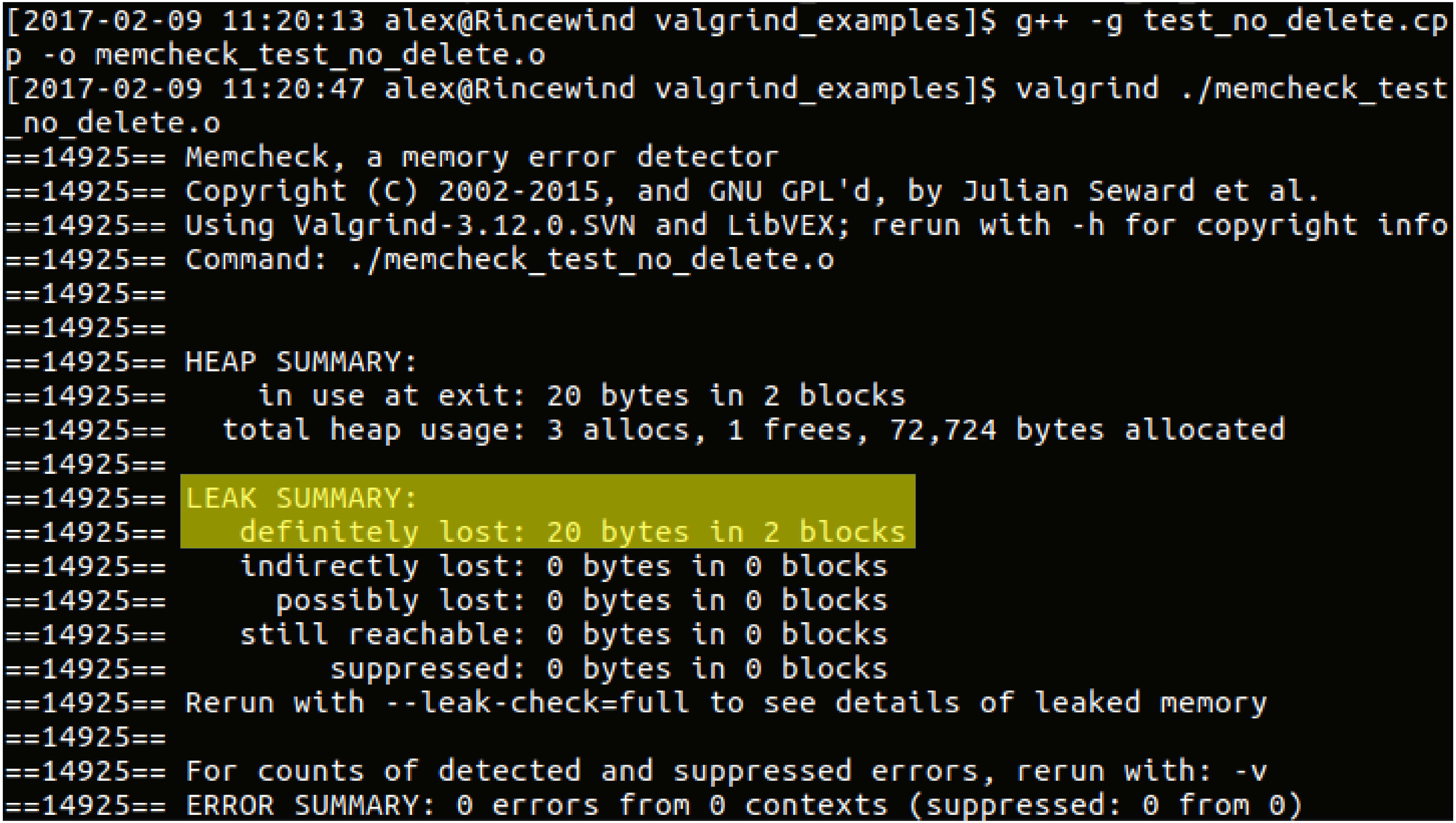
```
[2017-02-09 11:19:24 alex@Rincewind valgrind_examples]$ g++ test_no_delete.cpp -
o test_no_delete.o
[2017-02-09 11:20:11 alex@Rincewind valgrind_examples]$ ./test_no_delete.o
```

Valgrind example #2:



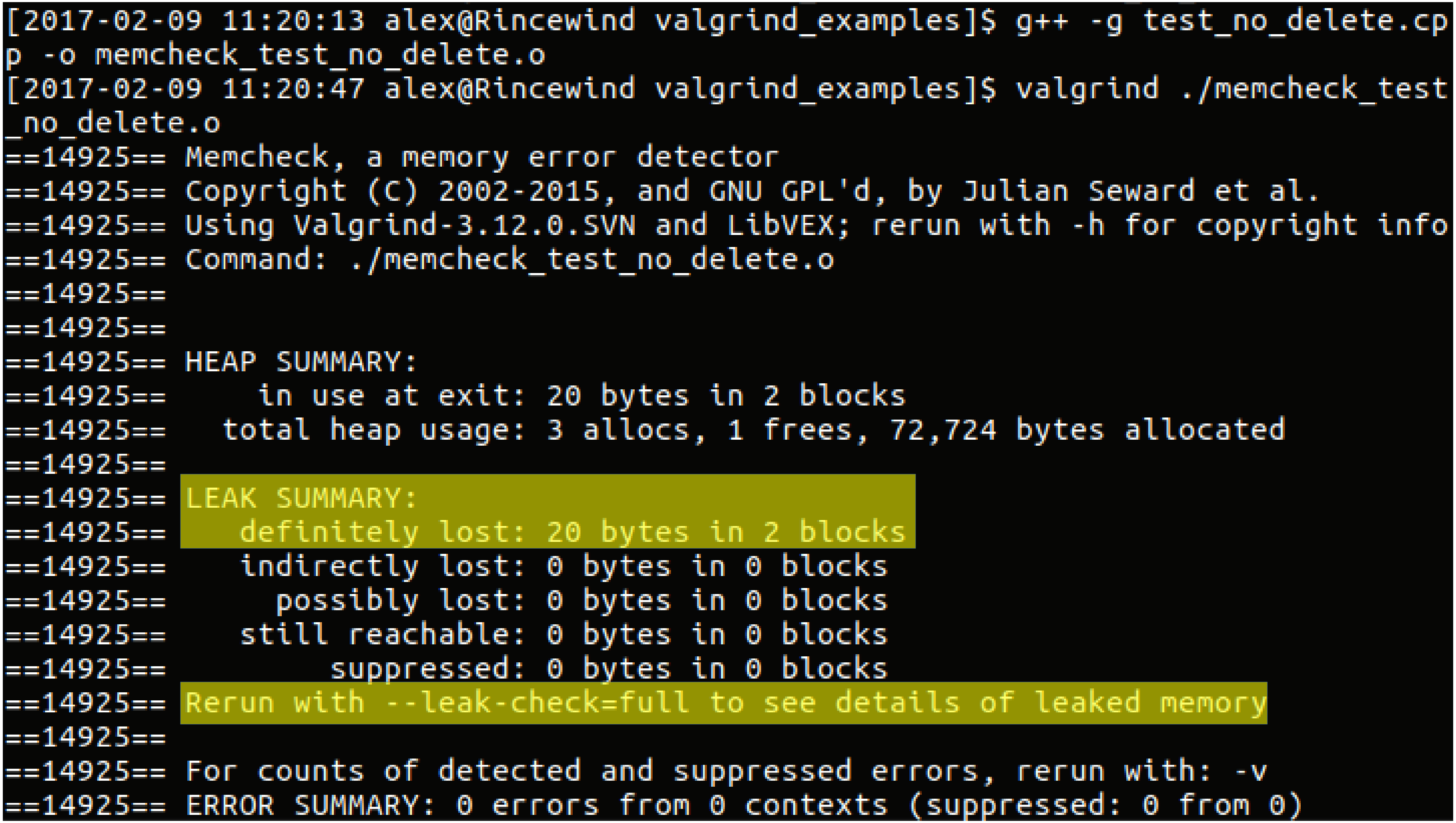
```
[2017-02-09 11:20:13 alex@Rincewind valgrind_examples]$ g++ -g test_no_delete.cp
p -o memcheck_test_no_delete.o
[2017-02-09 11:20:47 alex@Rincewind valgrind_examples]$ valgrind ./memcheck_test
_no_delete.o
==14925== Memcheck, a memory error detector
==14925== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==14925== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==14925== Command: ./memcheck_test_no_delete.o
==14925==
==14925==
==14925== HEAP SUMMARY:
==14925==     in use at exit: 20 bytes in 2 blocks
==14925==   total heap usage: 3 allocs, 1 frees, 72,724 bytes allocated
==14925==
==14925== LEAK SUMMARY:
==14925==    definitely lost: 20 bytes in 2 blocks
==14925==    indirectly lost: 0 bytes in 0 blocks
==14925==      possibly lost: 0 bytes in 0 blocks
==14925==    still reachable: 0 bytes in 0 blocks
==14925==         suppressed: 0 bytes in 0 blocks
==14925== Rerun with --leak-check=full to see details of leaked memory
==14925==
==14925== For counts of detected and suppressed errors, rerun with: -v
==14925== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Valgrind example #2:



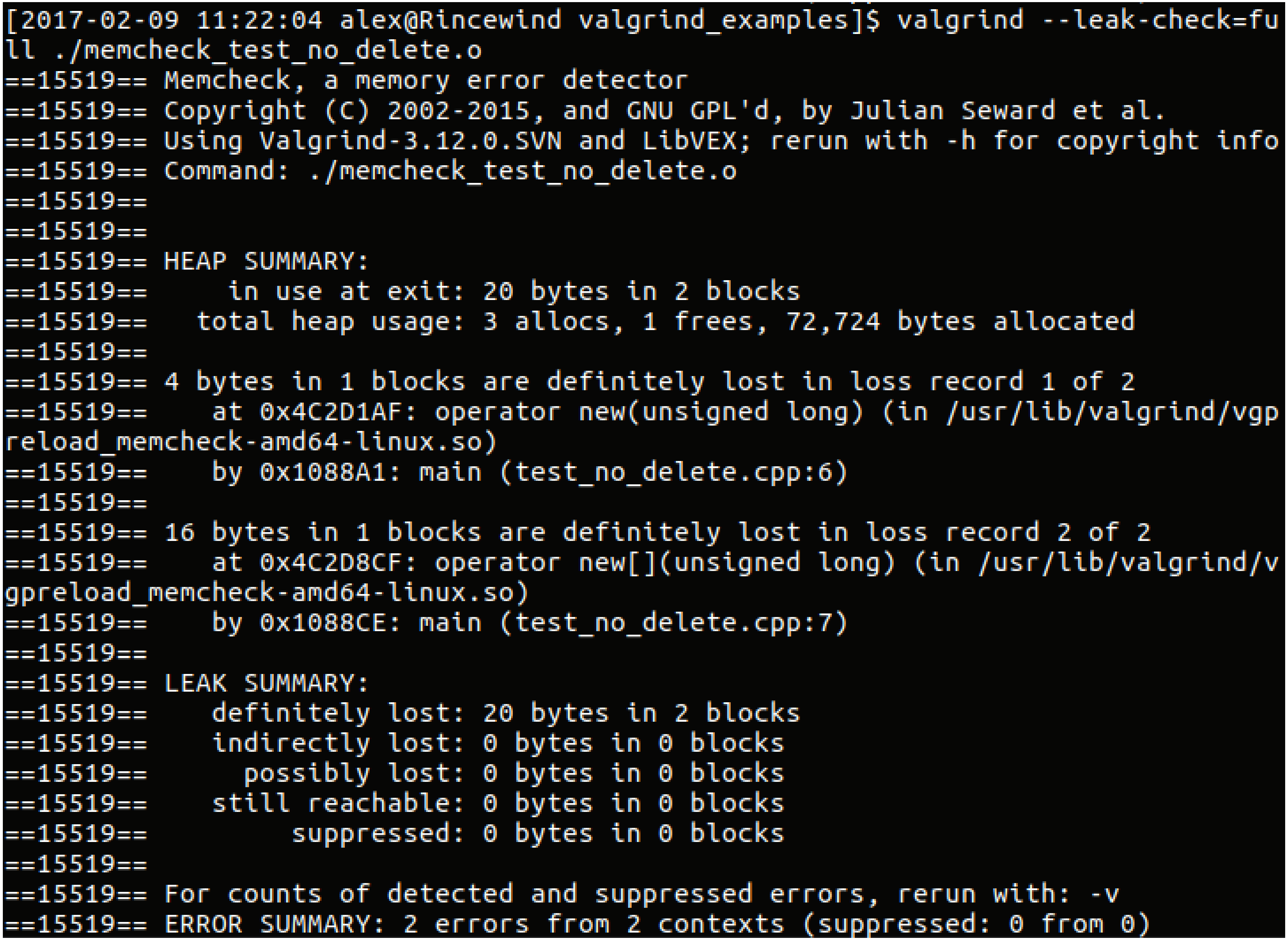
```
[2017-02-09 11:20:13 alex@Rincewind valgrind_examples]$ g++ -g test_no_delete.cp
p -o memcheck_test_no_delete.o
[2017-02-09 11:20:47 alex@Rincewind valgrind_examples]$ valgrind ./memcheck_test
_no_delete.o
==14925== Memcheck, a memory error detector
==14925== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==14925== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==14925== Command: ./memcheck_test_no_delete.o
==14925==
==14925==
==14925== HEAP SUMMARY:
==14925==     in use at exit: 20 bytes in 2 blocks
==14925==   total heap usage: 3 allocs, 1 frees, 72,724 bytes allocated
==14925==
==14925== LEAK SUMMARY:
==14925==    definitely lost: 20 bytes in 2 blocks
==14925==    indirectly lost: 0 bytes in 0 blocks
==14925==      possibly lost: 0 bytes in 0 blocks
==14925==    still reachable: 0 bytes in 0 blocks
==14925==         suppressed: 0 bytes in 0 blocks
==14925== Rerun with --leak-check=full to see details of leaked memory
==14925==
==14925== For counts of detected and suppressed errors, rerun with: -v
==14925== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Valgrind example #2:



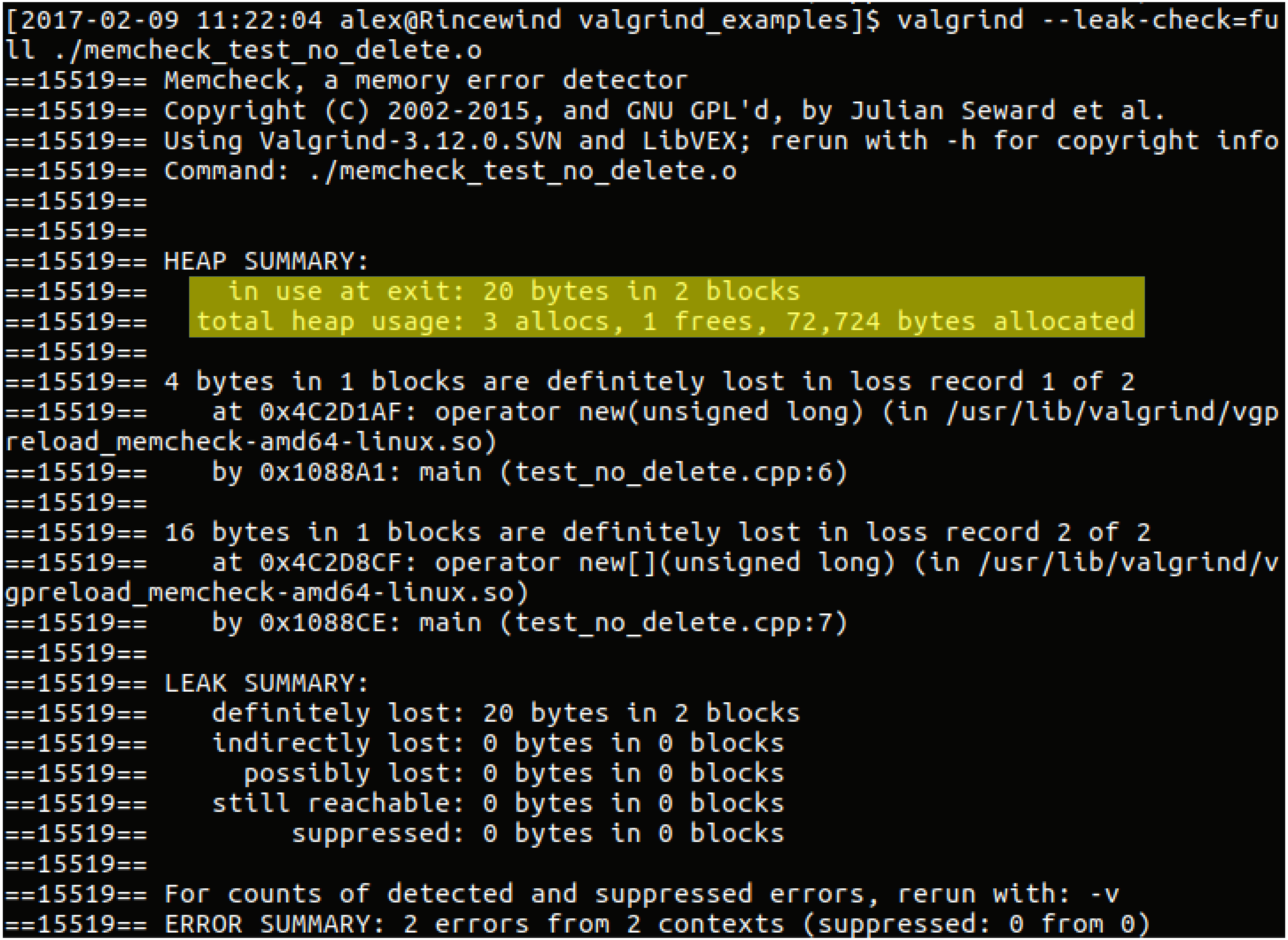
```
[2017-02-09 11:20:13 alex@Rincewind valgrind_examples]$ g++ -g test_no_delete.cp
p -o memcheck_test_no_delete.o
[2017-02-09 11:20:47 alex@Rincewind valgrind_examples]$ valgrind ./memcheck_test
_no_delete.o
==14925== Memcheck, a memory error detector
==14925== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==14925== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==14925== Command: ./memcheck_test_no_delete.o
==14925==
==14925==
==14925== HEAP SUMMARY:
==14925==     in use at exit: 20 bytes in 2 blocks
==14925==   total heap usage: 3 allocs, 1 frees, 72,724 bytes allocated
==14925==
==14925== LEAK SUMMARY:
==14925==    definitely lost: 20 bytes in 2 blocks
==14925==    indirectly lost: 0 bytes in 0 blocks
==14925==      possibly lost: 0 bytes in 0 blocks
==14925==    still reachable: 0 bytes in 0 blocks
==14925==         suppressed: 0 bytes in 0 blocks
==14925== Rerun with --leak-check=full to see details of leaked memory
==14925==
==14925== For counts of detected and suppressed errors, rerun with: -v
==14925== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Valgrind example #2:



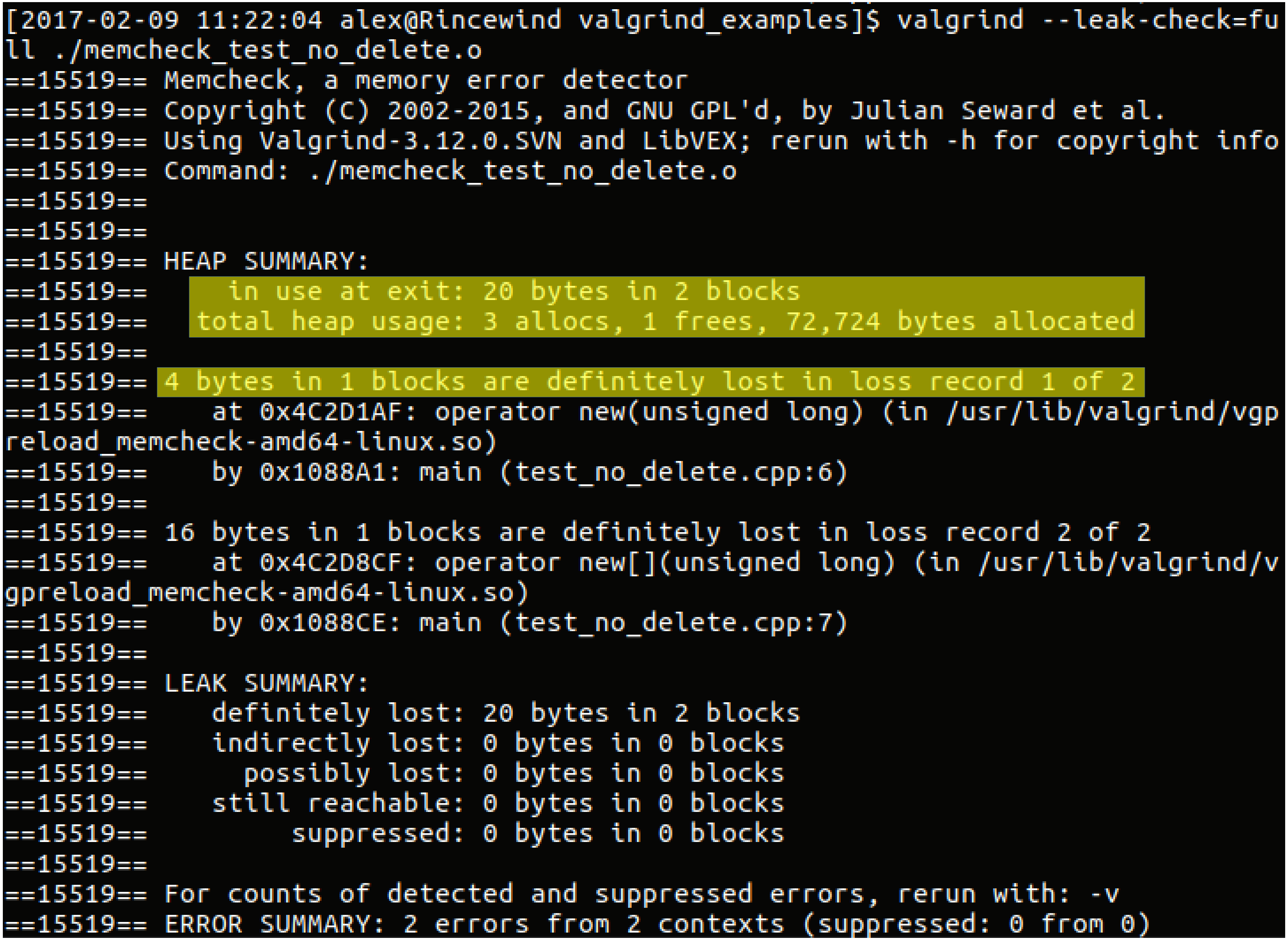
```
[2017-02-09 11:22:04 alex@Rincewind valgrind_examples]$ valgrind --leak-check=fu
ll ./memcheck_test_no_delete.o
==15519== Memcheck, a memory error detector
==15519== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15519== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==15519== Command: ./memcheck_test_no_delete.o
==15519==
==15519==
==15519== HEAP SUMMARY:
==15519==     in use at exit: 20 bytes in 2 blocks
==15519==   total heap usage: 3 allocs, 1 frees, 72,724 bytes allocated
==15519==
==15519== 4 bytes in 1 blocks are definitely lost in loss record 1 of 2
==15519==    at 0x4C2D1AF: operator new(unsigned long) (in /usr/lib/valgrind/vgp
reload_memcheck-amd64-linux.so)
==15519==    by 0x1088A1: main (test_no_delete.cpp:6)
==15519==
==15519== 16 bytes in 1 blocks are definitely lost in loss record 2 of 2
==15519==    at 0x4C2D8CF: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==15519==    by 0x1088CE: main (test_no_delete.cpp:7)
==15519==
==15519== LEAK SUMMARY:
==15519==    definitely lost: 20 bytes in 2 blocks
==15519==    indirectly lost: 0 bytes in 0 blocks
==15519==      possibly lost: 0 bytes in 0 blocks
==15519==    still reachable: 0 bytes in 0 blocks
==15519==         suppressed: 0 bytes in 0 blocks
==15519==
==15519== For counts of detected and suppressed errors, rerun with: -v
==15519== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Valgrind example #2:



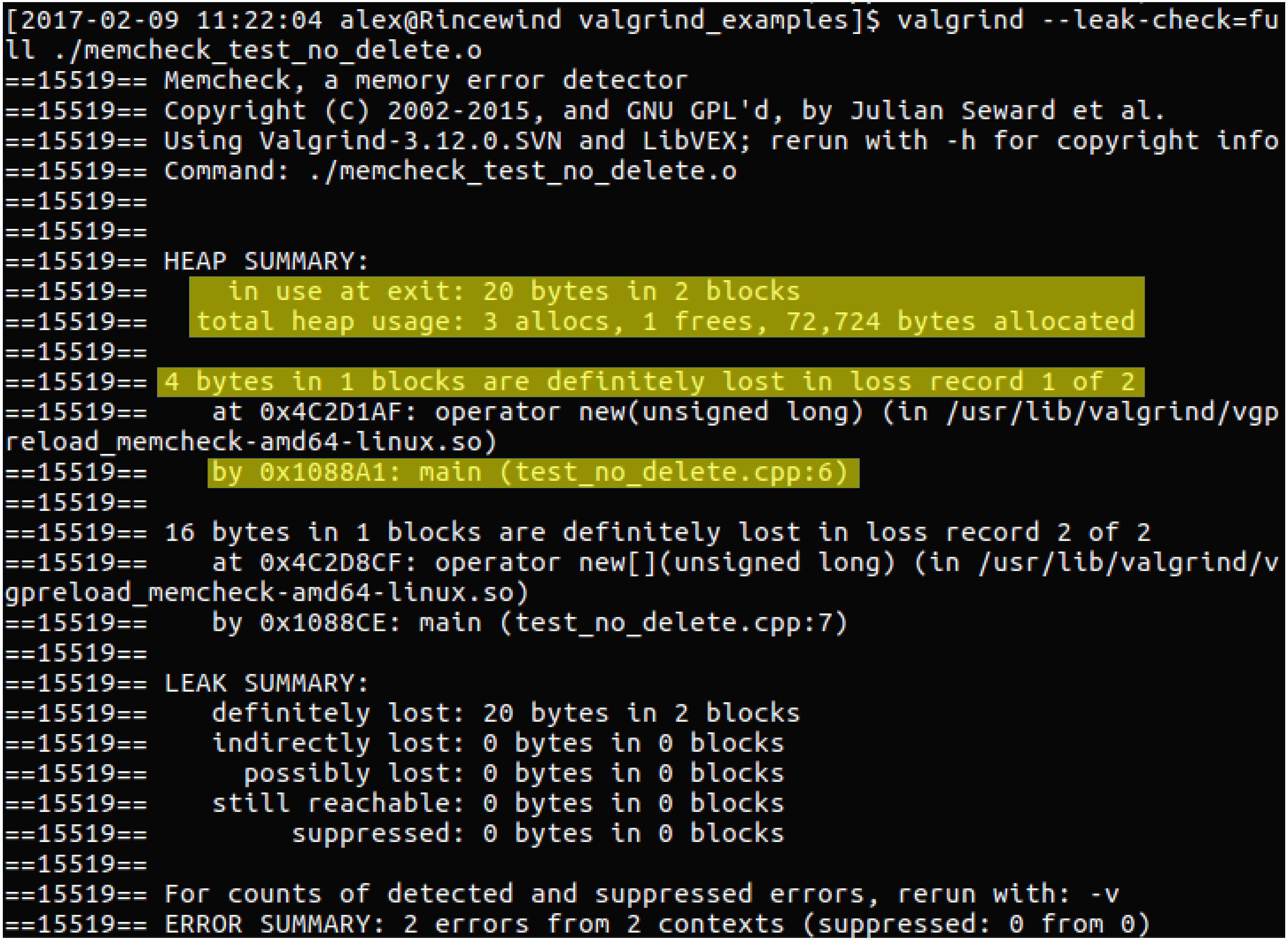
```
[2017-02-09 11:22:04 alex@Rincewind valgrind_examples]$ valgrind --leak-check=fu
ll ./memcheck_test_no_delete.o
==15519== Memcheck, a memory error detector
==15519== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15519== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==15519== Command: ./memcheck_test_no_delete.o
==15519==
==15519==
==15519== HEAP SUMMARY:
==15519==     in use at exit: 20 bytes in 2 blocks
==15519==   total heap usage: 3 allocs, 1 frees, 72,724 bytes allocated
==15519==
==15519== 4 bytes in 1 blocks are definitely lost in loss record 1 of 2
==15519==    at 0x4C2D1AF: operator new(unsigned long) (in /usr/lib/valgrind/vgp
reload_memcheck-amd64-linux.so)
==15519==    by 0x1088A1: main (test_no_delete.cpp:6)
==15519==
==15519== 16 bytes in 1 blocks are definitely lost in loss record 2 of 2
==15519==    at 0x4C2D8CF: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==15519==    by 0x1088CE: main (test_no_delete.cpp:7)
==15519==
==15519== LEAK SUMMARY:
==15519==    definitely lost: 20 bytes in 2 blocks
==15519==    indirectly lost: 0 bytes in 0 blocks
==15519==      possibly lost: 0 bytes in 0 blocks
==15519==    still reachable: 0 bytes in 0 blocks
==15519==         suppressed: 0 bytes in 0 blocks
==15519==
==15519== For counts of detected and suppressed errors, rerun with: -v
==15519== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Valgrind example #2:



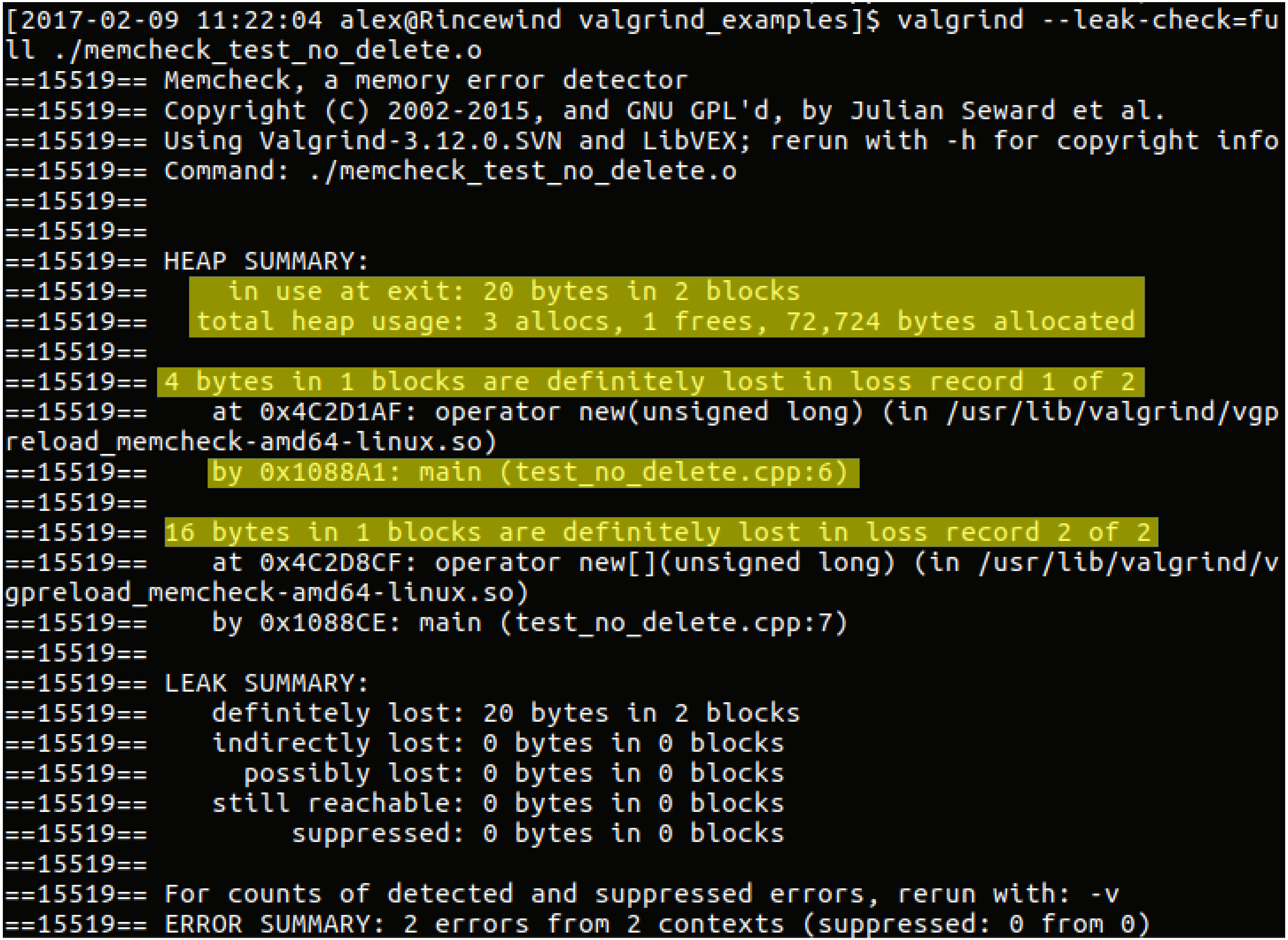
```
[2017-02-09 11:22:04 alex@Rincewind valgrind_examples]$ valgrind --leak-check=fu
ll ./memcheck_test_no_delete.o
==15519== Memcheck, a memory error detector
==15519== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15519== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==15519== Command: ./memcheck_test_no_delete.o
==15519==
==15519==
==15519== HEAP SUMMARY:
==15519==     in use at exit: 20 bytes in 2 blocks
==15519==   total heap usage: 3 allocs, 1 frees, 72,724 bytes allocated
==15519==
==15519== 4 bytes in 1 blocks are definitely lost in loss record 1 of 2
==15519==    at 0x4C2D1AF: operator new(unsigned long) (in /usr/lib/valgrind/vgp
reload_memcheck-amd64-linux.so)
==15519==    by 0x1088A1: main (test_no_delete.cpp:6)
==15519==
==15519== 16 bytes in 1 blocks are definitely lost in loss record 2 of 2
==15519==    at 0x4C2D8CF: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==15519==    by 0x1088CE: main (test_no_delete.cpp:7)
==15519==
==15519== LEAK SUMMARY:
==15519==    definitely lost: 20 bytes in 2 blocks
==15519==    indirectly lost: 0 bytes in 0 blocks
==15519==      possibly lost: 0 bytes in 0 blocks
==15519==    still reachable: 0 bytes in 0 blocks
==15519==         suppressed: 0 bytes in 0 blocks
==15519==
==15519== For counts of detected and suppressed errors, rerun with: -v
==15519== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Valgrind example #2:



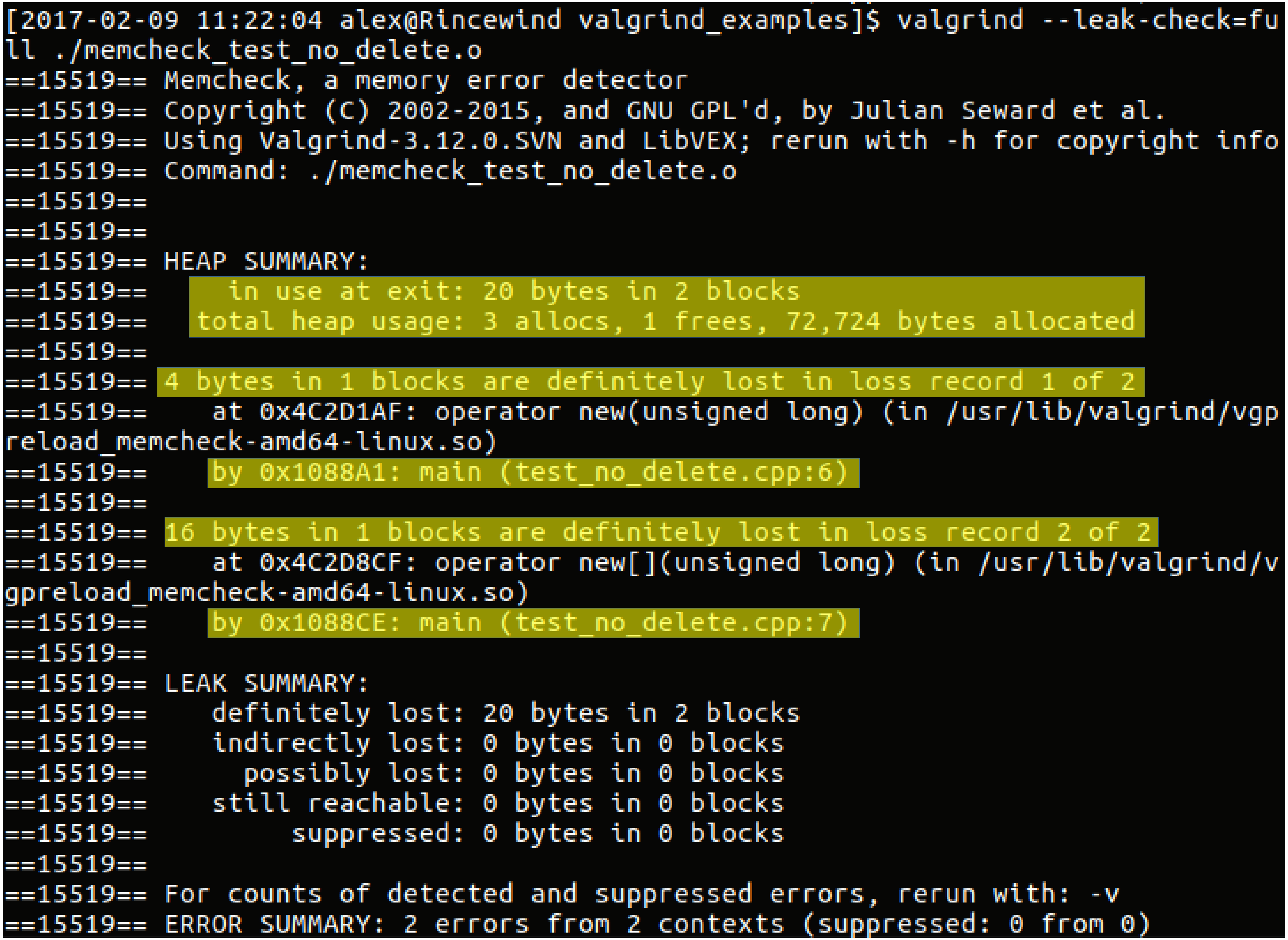
```
[2017-02-09 11:22:04 alex@Rincewind valgrind_examples]$ valgrind --leak-check=fu
ll ./memcheck_test_no_delete.o
==15519== Memcheck, a memory error detector
==15519== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15519== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==15519== Command: ./memcheck_test_no_delete.o
==15519==
==15519==
==15519== HEAP SUMMARY:
==15519==     in use at exit: 20 bytes in 2 blocks
==15519==   total heap usage: 3 allocs, 1 frees, 72,724 bytes allocated
==15519==
==15519== 4 bytes in 1 blocks are definitely lost in loss record 1 of 2
==15519==    at 0x4C2D1AF: operator new(unsigned long) (in /usr/lib/valgrind/vgp
reload_memcheck-amd64-linux.so)
==15519==    by 0x1088A1: main (test_no_delete.cpp:6)
==15519==
==15519== 16 bytes in 1 blocks are definitely lost in loss record 2 of 2
==15519==    at 0x4C2D8CF: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==15519==    by 0x1088CE: main (test_no_delete.cpp:7)
==15519==
==15519== LEAK SUMMARY:
==15519==    definitely lost: 20 bytes in 2 blocks
==15519==    indirectly lost: 0 bytes in 0 blocks
==15519==      possibly lost: 0 bytes in 0 blocks
==15519==    still reachable: 0 bytes in 0 blocks
==15519==         suppressed: 0 bytes in 0 blocks
==15519==
==15519== For counts of detected and suppressed errors, rerun with: -v
==15519== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Valgrind example #2:



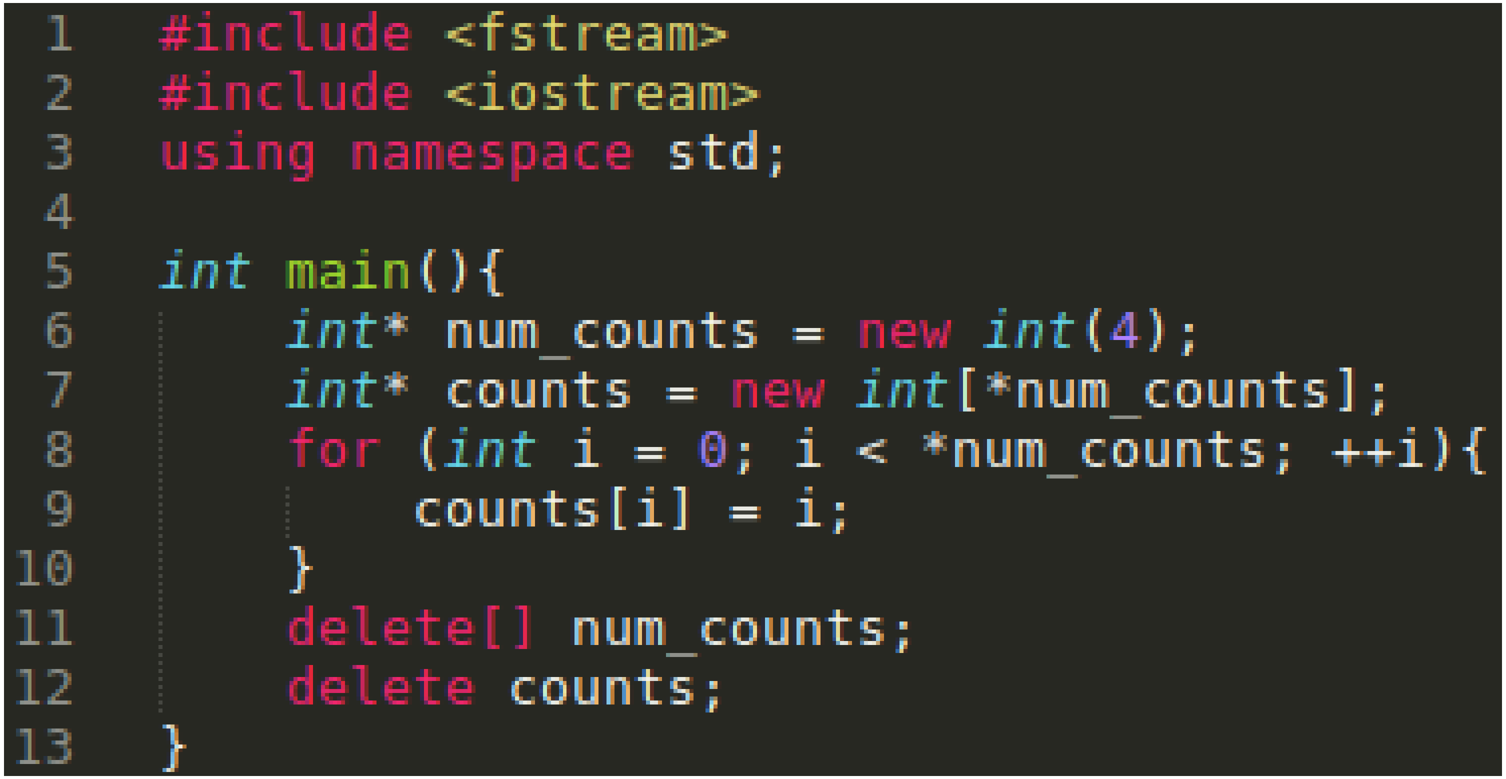
```
[2017-02-09 11:22:04 alex@Rincewind valgrind_examples]$ valgrind --leak-check=fu
ll ./memcheck_test_no_delete.o
==15519== Memcheck, a memory error detector
==15519== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15519== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==15519== Command: ./memcheck_test_no_delete.o
==15519==
==15519==
==15519== HEAP SUMMARY:
==15519==     in use at exit: 20 bytes in 2 blocks
==15519==   total heap usage: 3 allocs, 1 frees, 72,724 bytes allocated
==15519==
==15519== 4 bytes in 1 blocks are definitely lost in loss record 1 of 2
==15519==    at 0x4C2D1AF: operator new(unsigned long) (in /usr/lib/valgrind/vgp
reload_memcheck-amd64-linux.so)
==15519==    by 0x1088A1: main (test_no_delete.cpp:6)
==15519==
==15519== 16 bytes in 1 blocks are definitely lost in loss record 2 of 2
==15519==    at 0x4C2D8CF: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==15519==    by 0x1088CE: main (test_no_delete.cpp:7)
==15519==
==15519== LEAK SUMMARY:
==15519==    definitely lost: 20 bytes in 2 blocks
==15519==    indirectly lost: 0 bytes in 0 blocks
==15519==      possibly lost: 0 bytes in 0 blocks
==15519==    still reachable: 0 bytes in 0 blocks
==15519==         suppressed: 0 bytes in 0 blocks
==15519==
==15519== For counts of detected and suppressed errors, rerun with: -v
==15519== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Valgrind example #2:



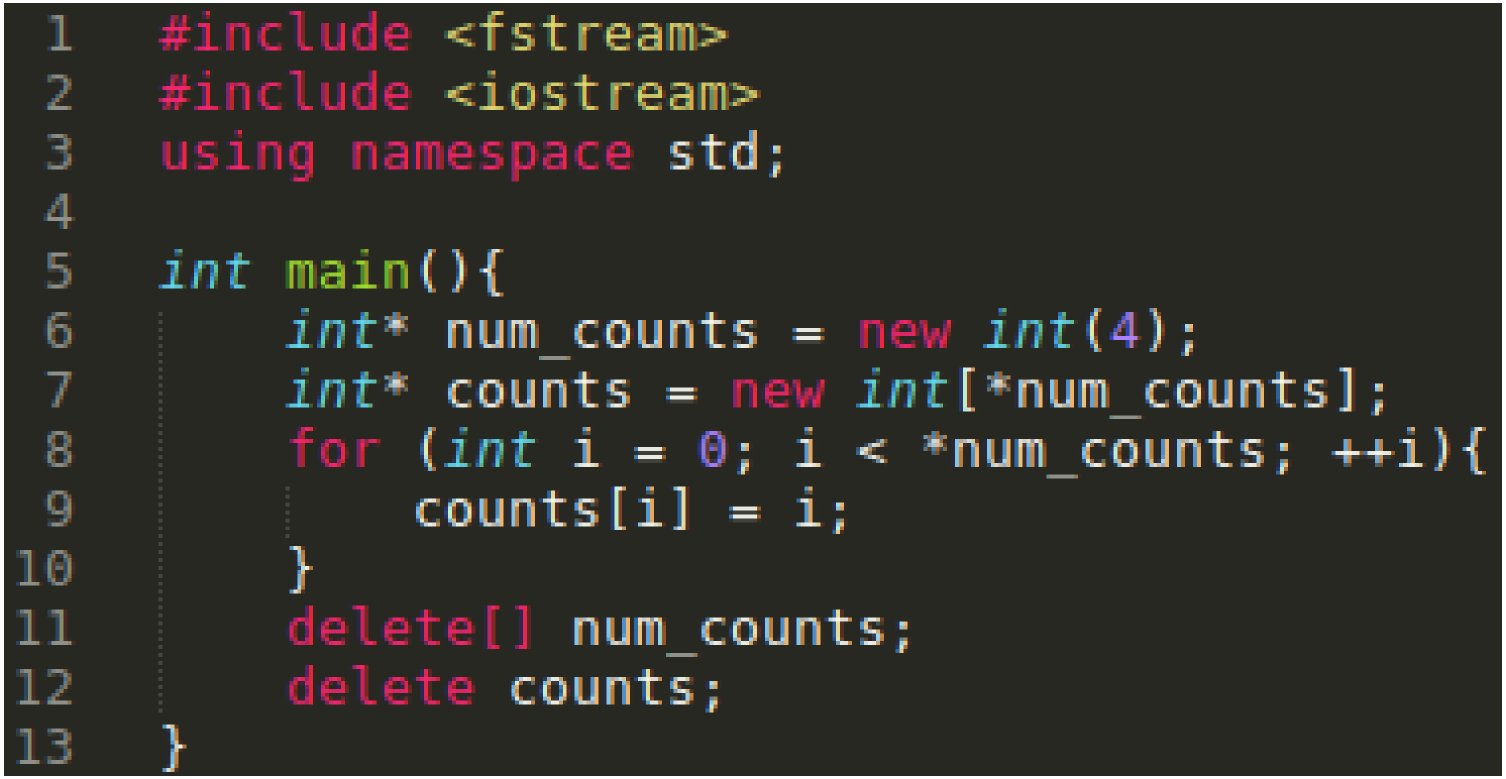
```
[2017-02-09 11:22:04 alex@Rincewind valgrind_examples]$ valgrind --leak-check=fu
ll ./memcheck_test_no_delete.o
==15519== Memcheck, a memory error detector
==15519== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15519== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==15519== Command: ./memcheck_test_no_delete.o
==15519==
==15519==
==15519== HEAP SUMMARY:
==15519==     in use at exit: 20 bytes in 2 blocks
==15519==   total heap usage: 3 allocs, 1 frees, 72,724 bytes allocated
==15519==
==15519== 4 bytes in 1 blocks are definitely lost in loss record 1 of 2
==15519==    at 0x4C2D1AF: operator new(unsigned long) (in /usr/lib/valgrind/vgp
reload_memcheck-amd64-linux.so)
==15519==    by 0x1088A1: main (test_no_delete.cpp:6)
==15519==
==15519== 16 bytes in 1 blocks are definitely lost in loss record 2 of 2
==15519==    at 0x4C2D8CF: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==15519==    by 0x1088CE: main (test_no_delete.cpp:7)
==15519==
==15519== LEAK SUMMARY:
==15519==    definitely lost: 20 bytes in 2 blocks
==15519==    indirectly lost: 0 bytes in 0 blocks
==15519==      possibly lost: 0 bytes in 0 blocks
==15519==    still reachable: 0 bytes in 0 blocks
==15519==         suppressed: 0 bytes in 0 blocks
==15519==
==15519== For counts of detected and suppressed errors, rerun with: -v
==15519== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Valgrind example #3:

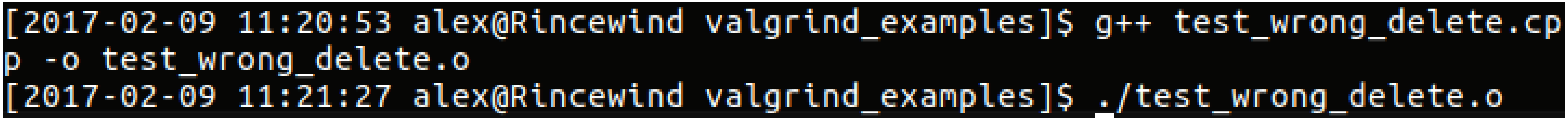


```
#include <fstream>
#include <iostream>
using namespace std;

int main(){
    int* num_counts = new int(4);
    int* counts = new int[*num_counts];
    for (int i = 0; i < *num_counts; ++i){
        counts[i] = i;
    }
    delete[] num_counts;
    delete counts;
}
```

Valgrind example #3:

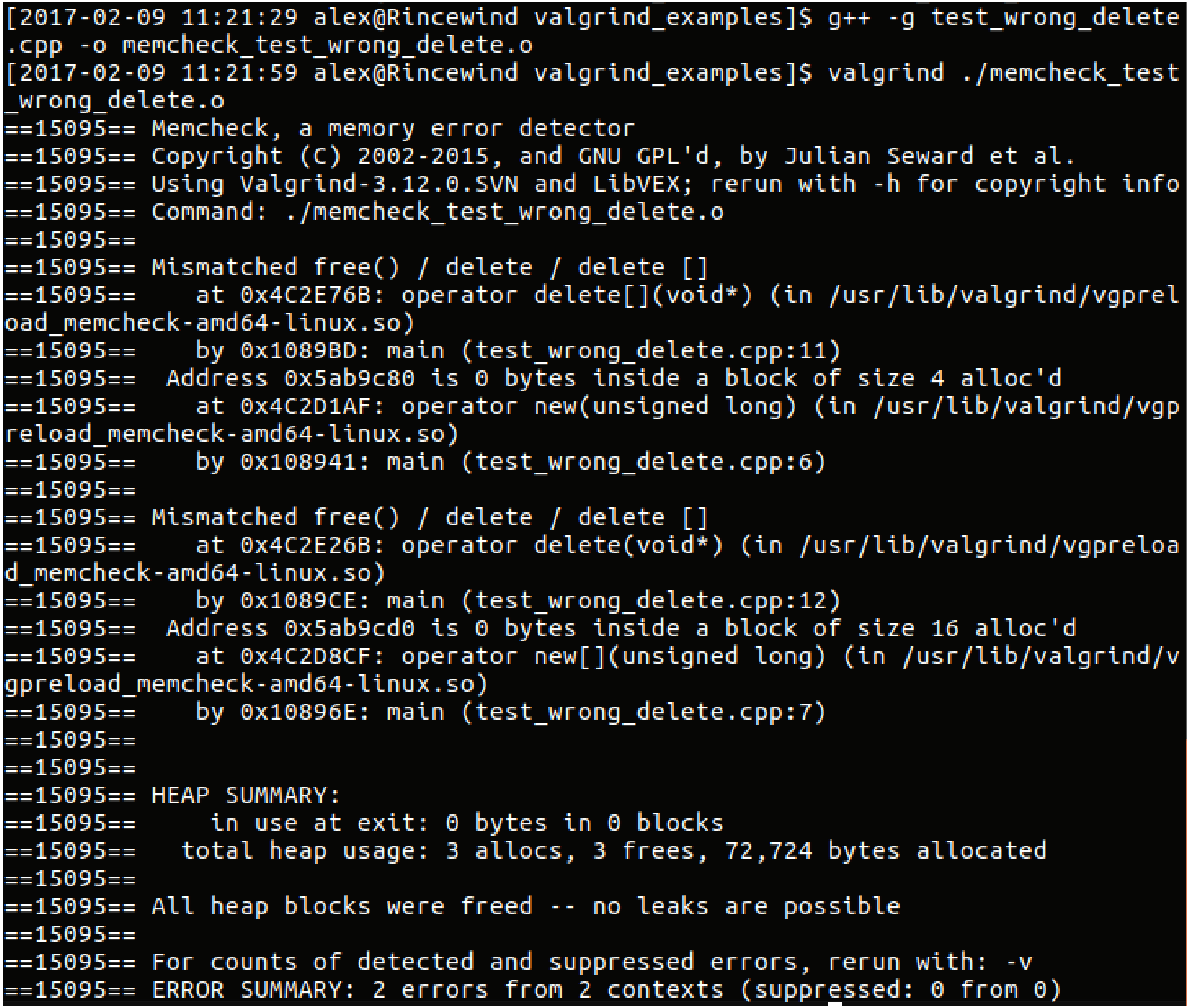


```
#include <fstream>
#include <iostream>
using namespace std;

int main(){
    int* num_counts = new int(4);
    int* counts = new int[*num_counts];
    for (int i = 0; i < *num_counts; ++i){
        counts[i] = i;
    }
    delete[] num_counts;
    delete counts;
}
```



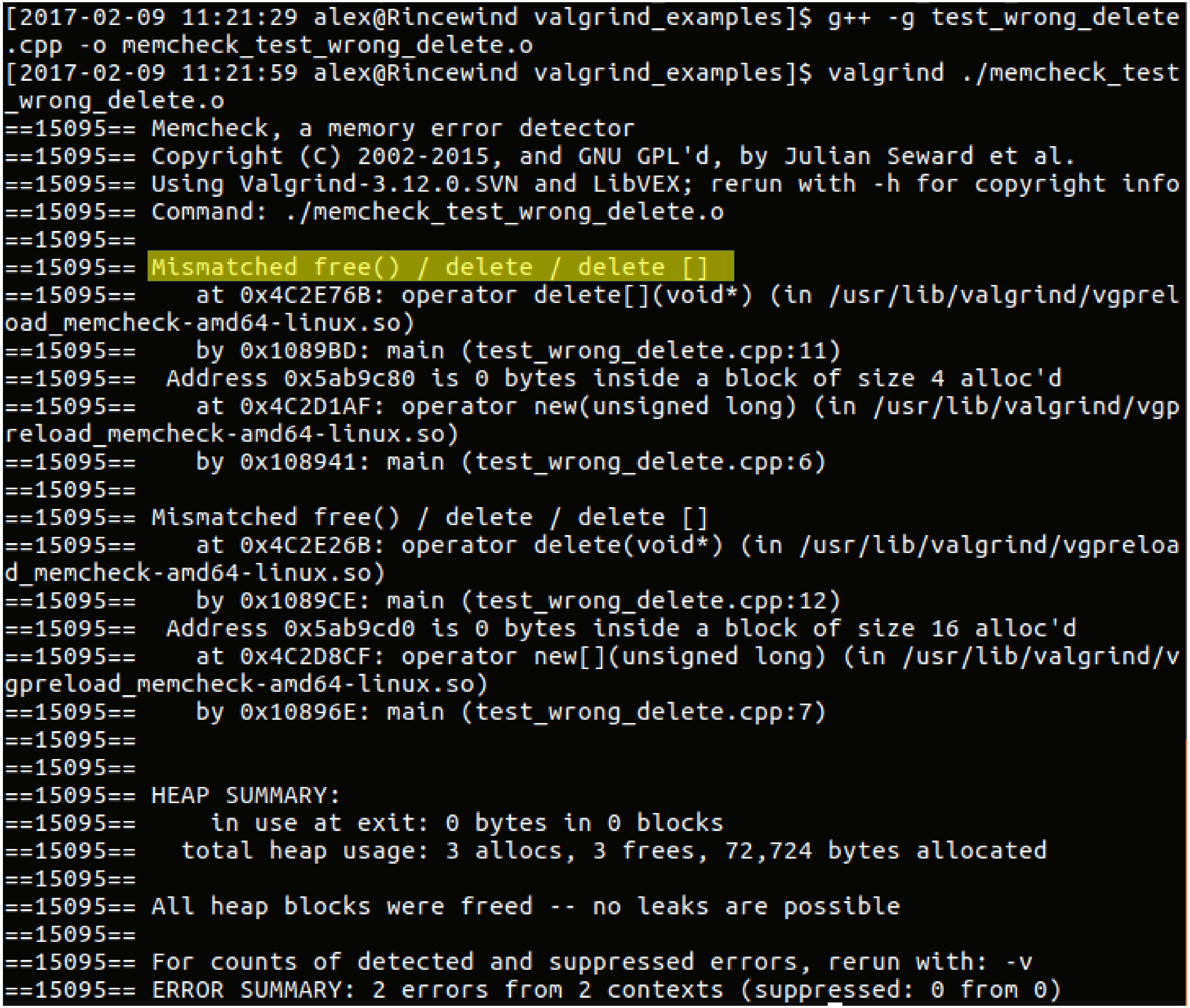
```
[2017-02-09 11:20:53 alex@Rincewind valgrind_examples]$ g++ test_wrong_delete.cp
p -o test_wrong_delete.o
[2017-02-09 11:21:27 alex@Rincewind valgrind_examples]$ ./test_wrong_delete.o
```

Valgrind example #3:



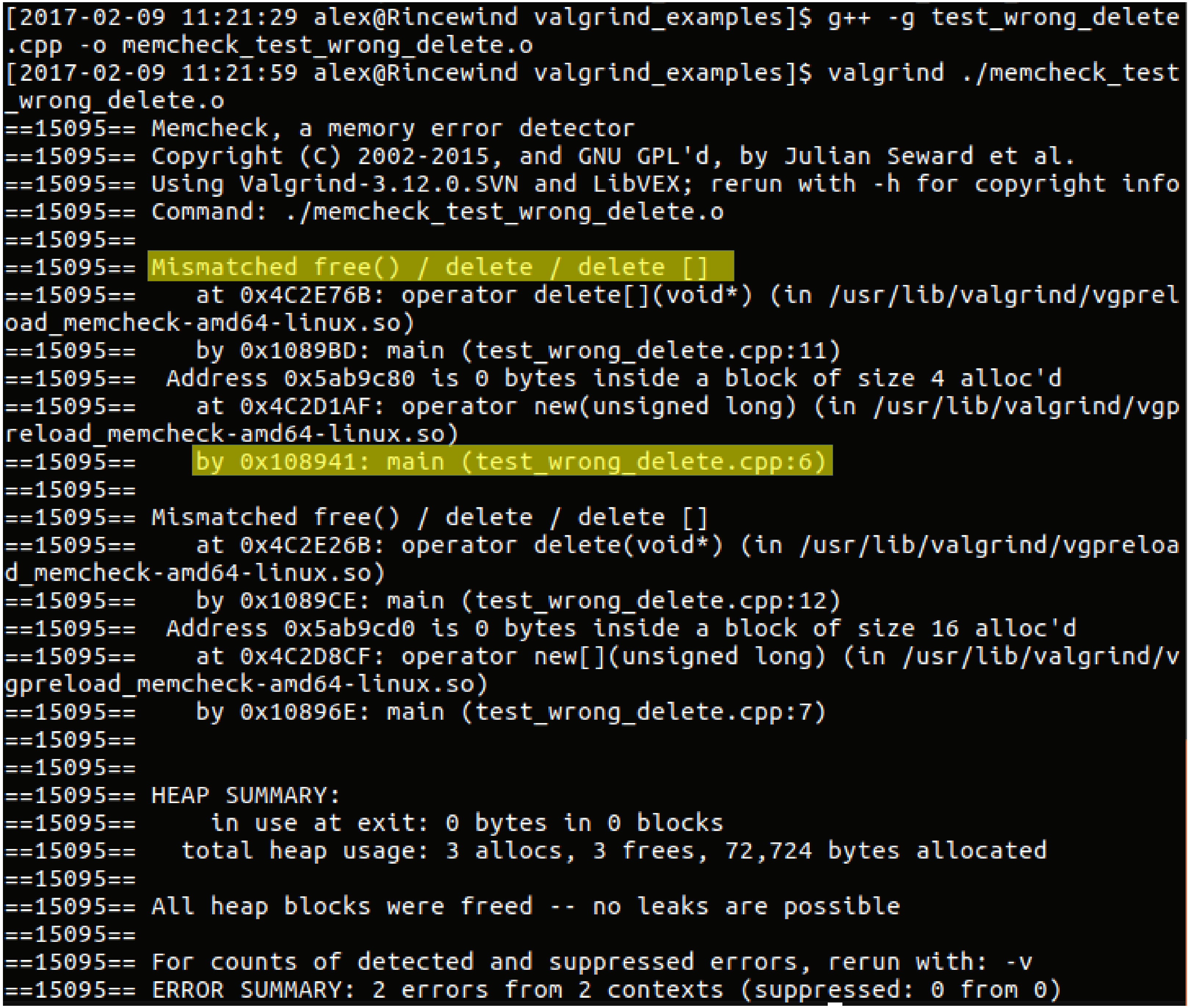
```
[2017-02-09 11:21:29 alex@Rincewind valgrind_examples]$ g++ -g test_wrong_delete
.cpp -o memcheck_test_wrong_delete.o
[2017-02-09 11:21:59 alex@Rincewind valgrind_examples]$ valgrind ./memcheck_test
_wrong_delete.o
==15095== Memcheck, a memory error detector
==15095== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15095== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==15095== Command: ./memcheck_test_wrong_delete.o
==15095==
==15095== Mismatched free() / delete / delete []
==15095==    at 0x4C2E76B: operator delete[](void*) (in /usr/lib/valgrind/vgprel
oad_memcheck-amd64-linux.so)
==15095==    by 0x1089BD: main (test_wrong_delete.cpp:11)
==15095==  Address 0x5ab9c80 is 0 bytes inside a block of size 4 alloc'd
==15095==    at 0x4C2D1AF: operator new(unsigned long) (in /usr/lib/valgrind/vgp
reload_memcheck-amd64-linux.so)
==15095==    by 0x108941: main (test_wrong_delete.cpp:6)
==15095==
==15095== Mismatched free() / delete / delete []
==15095==    at 0x4C2E26B: operator delete(void*) (in /usr/lib/valgrind/vgpreloa
d_memcheck-amd64-linux.so)
==15095==    by 0x1089CE: main (test_wrong_delete.cpp:12)
==15095==  Address 0x5ab9cd0 is 0 bytes inside a block of size 16 alloc'd
==15095==    at 0x4C2D8CF: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==15095==    by 0x10896E: main (test_wrong_delete.cpp:7)
==15095==
==15095==
==15095== HEAP SUMMARY:
==15095==     in use at exit: 0 bytes in 0 blocks
==15095==   total heap usage: 3 allocs, 3 frees, 72,724 bytes allocated
==15095==
==15095== All heap blocks were freed -- no leaks are possible
==15095==
==15095== For counts of detected and suppressed errors, rerun with: -v
==15095== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Valgrind example #3:



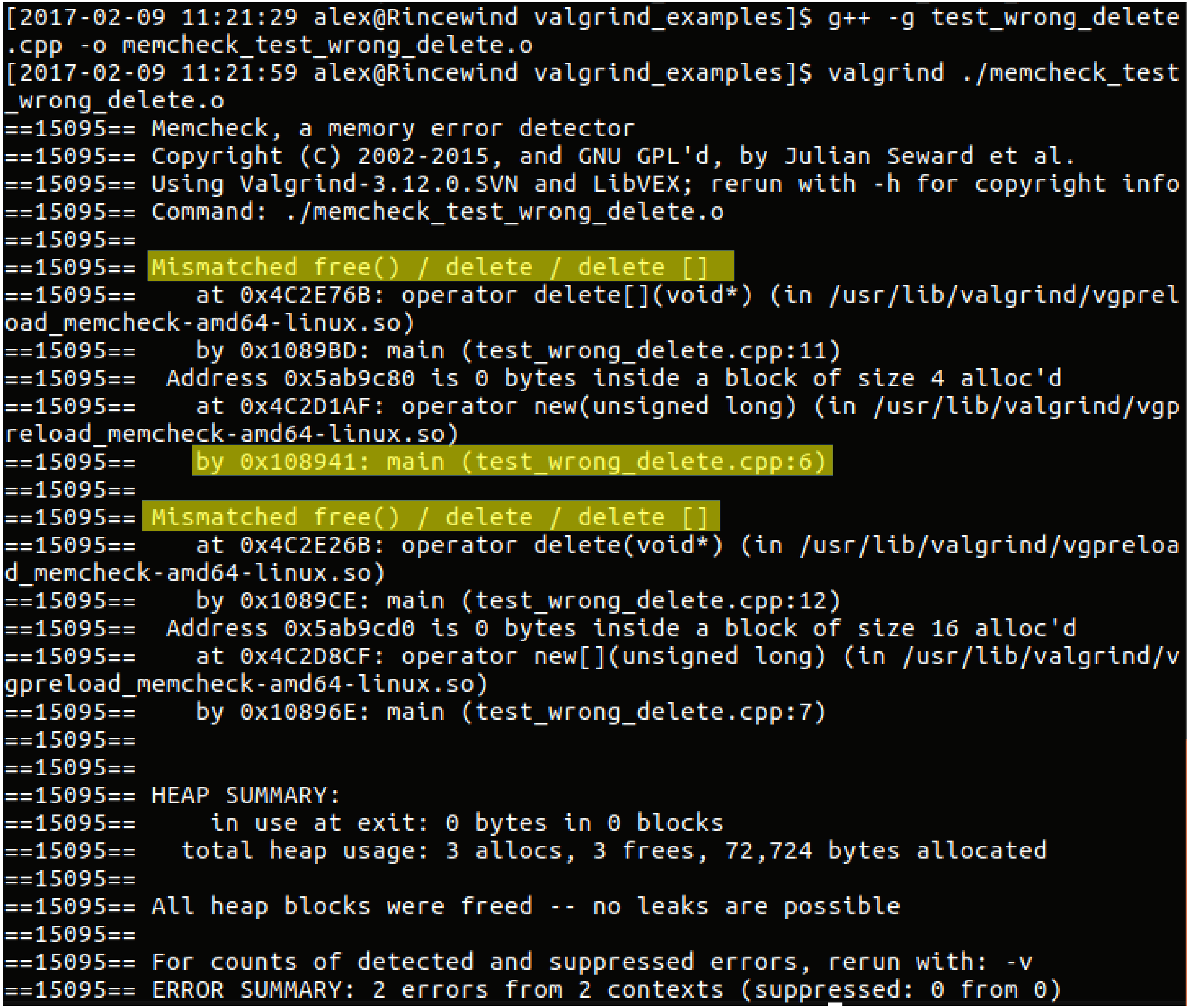
```
[2017-02-09 11:21:29 alex@Rincewind valgrind_examples]$ g++ -g test_wrong_delete
.cpp -o memcheck_test_wrong_delete.o
[2017-02-09 11:21:59 alex@Rincewind valgrind_examples]$ valgrind ./memcheck_test
_wrong_delete.o
==15095== Memcheck, a memory error detector
==15095== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15095== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==15095== Command: ./memcheck_test_wrong_delete.o
==15095==
==15095== Mismatched free() / delete / delete []
==15095==    at 0x4C2E76B: operator delete[](void*) (in /usr/lib/valgrind/vgprel
oad_memcheck-amd64-linux.so)
==15095==    by 0x1089BD: main (test_wrong_delete.cpp:11)
==15095==  Address 0x5ab9c80 is 0 bytes inside a block of size 4 alloc'd
==15095==    at 0x4C2D1AF: operator new(unsigned long) (in /usr/lib/valgrind/vgp
reload_memcheck-amd64-linux.so)
==15095==    by 0x108941: main (test_wrong_delete.cpp:6)
==15095==
==15095== Mismatched free() / delete / delete []
==15095==    at 0x4C2E26B: operator delete(void*) (in /usr/lib/valgrind/vgpreloa
d_memcheck-amd64-linux.so)
==15095==    by 0x1089CE: main (test_wrong_delete.cpp:12)
==15095==  Address 0x5ab9cd0 is 0 bytes inside a block of size 16 alloc'd
==15095==    at 0x4C2D8CF: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==15095==    by 0x10896E: main (test_wrong_delete.cpp:7)
==15095==
==15095==
==15095== HEAP SUMMARY:
==15095==     in use at exit: 0 bytes in 0 blocks
==15095==   total heap usage: 3 allocs, 3 frees, 72,724 bytes allocated
==15095==
==15095== All heap blocks were freed -- no leaks are possible
==15095==
==15095== For counts of detected and suppressed errors, rerun with: -v
==15095== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Valgrind example #3:



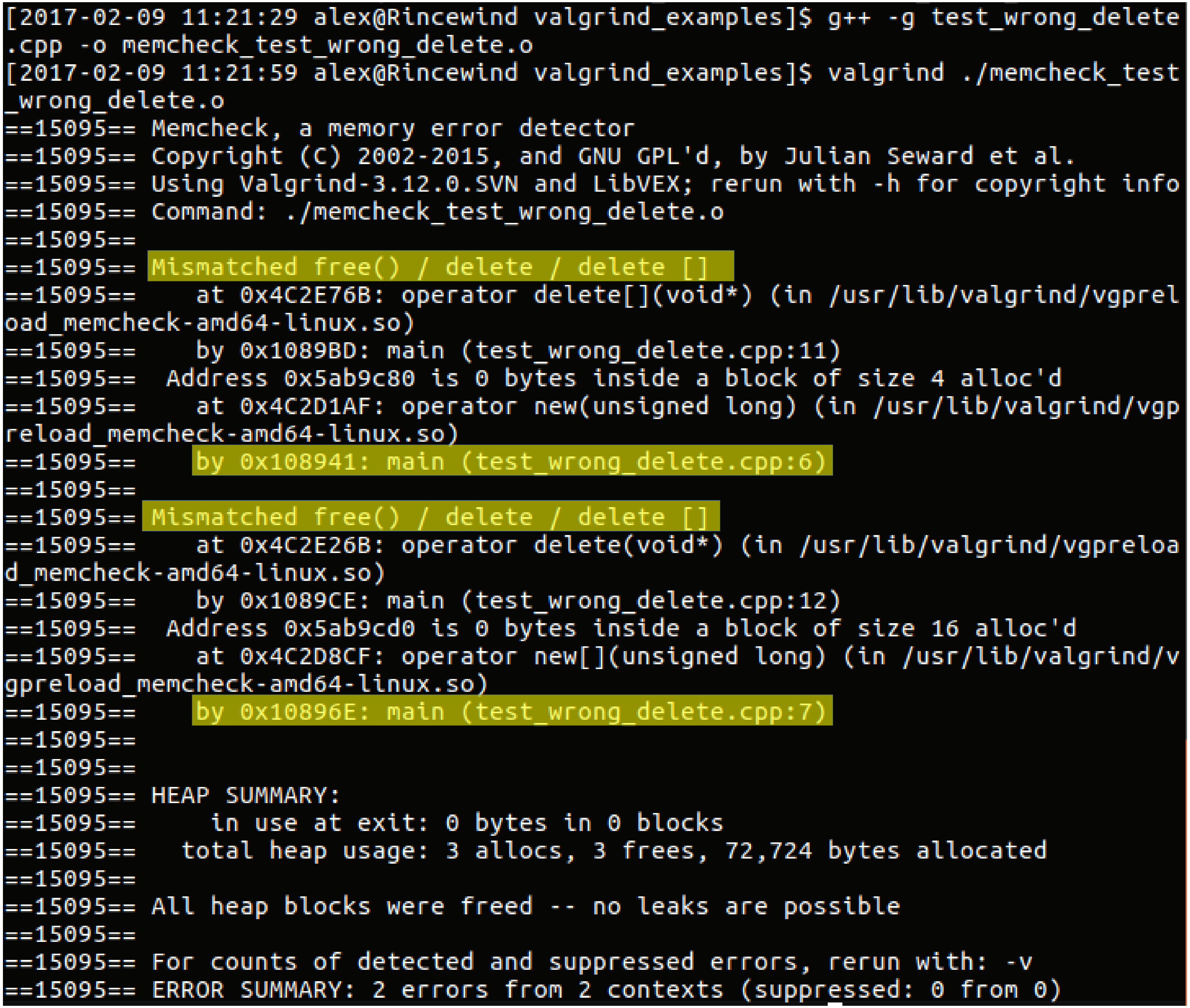
```
[2017-02-09 11:21:29 alex@Rincewind valgrind_examples]$ g++ -g test_wrong_delete
.cpp -o memcheck_test_wrong_delete.o
[2017-02-09 11:21:59 alex@Rincewind valgrind_examples]$ valgrind ./memcheck_test
_wrong_delete.o
==15095== Memcheck, a memory error detector
==15095== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15095== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==15095== Command: ./memcheck_test_wrong_delete.o
==15095==
==15095== Mismatched free() / delete / delete []
==15095==    at 0x4C2E76B: operator delete[](void*) (in /usr/lib/valgrind/vgprel
oad_memcheck-amd64-linux.so)
==15095==    by 0x1089BD: main (test_wrong_delete.cpp:11)
==15095==  Address 0x5ab9c80 is 0 bytes inside a block of size 4 alloc'd
==15095==    at 0x4C2D1AF: operator new(unsigned long) (in /usr/lib/valgrind/vgp
reload_memcheck-amd64-linux.so)
==15095==    by 0x108941: main (test_wrong_delete.cpp:6)
==15095==
==15095== Mismatched free() / delete / delete []
==15095==    at 0x4C2E26B: operator delete(void*) (in /usr/lib/valgrind/vgpreloa
d_memcheck-amd64-linux.so)
==15095==    by 0x1089CE: main (test_wrong_delete.cpp:12)
==15095==  Address 0x5ab9cd0 is 0 bytes inside a block of size 16 alloc'd
==15095==    at 0x4C2D8CF: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==15095==    by 0x10896E: main (test_wrong_delete.cpp:7)
==15095==
==15095==
==15095== HEAP SUMMARY:
==15095==     in use at exit: 0 bytes in 0 blocks
==15095==   total heap usage: 3 allocs, 3 frees, 72,724 bytes allocated
==15095==
==15095== All heap blocks were freed -- no leaks are possible
==15095==
==15095== For counts of detected and suppressed errors, rerun with: -v
==15095== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Valgrind example #3:



```
[2017-02-09 11:21:29 alex@Rincewind valgrind_examples]$ g++ -g test_wrong_delete
.cpp -o memcheck_test_wrong_delete.o
[2017-02-09 11:21:59 alex@Rincewind valgrind_examples]$ valgrind ./memcheck_test
_wrong_delete.o
==15095== Memcheck, a memory error detector
==15095== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15095== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==15095== Command: ./memcheck_test_wrong_delete.o
==15095==
==15095== Mismatched free() / delete / delete []
==15095==    at 0x4C2E76B: operator delete[](void*) (in /usr/lib/valgrind/vgprel
oad_memcheck-amd64-linux.so)
==15095==    by 0x1089BD: main (test_wrong_delete.cpp:11)
==15095==  Address 0x5ab9c80 is 0 bytes inside a block of size 4 alloc'd
==15095==    at 0x4C2D1AF: operator new(unsigned long) (in /usr/lib/valgrind/vgp
reload_memcheck-amd64-linux.so)
==15095==    by 0x108941: main (test_wrong_delete.cpp:6)
==15095==
==15095== Mismatched free() / delete / delete []
==15095==    at 0x4C2E26B: operator delete(void*) (in /usr/lib/valgrind/vgpreloa
d_memcheck-amd64-linux.so)
==15095==    by 0x1089CE: main (test_wrong_delete.cpp:12)
==15095==  Address 0x5ab9cd0 is 0 bytes inside a block of size 16 alloc'd
==15095==    at 0x4C2D8CF: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==15095==    by 0x10896E: main (test_wrong_delete.cpp:7)
==15095==
==15095==
==15095== HEAP SUMMARY:
==15095==     in use at exit: 0 bytes in 0 blocks
==15095==   total heap usage: 3 allocs, 3 frees, 72,724 bytes allocated
==15095==
==15095== All heap blocks were freed -- no leaks are possible
==15095==
==15095== For counts of detected and suppressed errors, rerun with: -v
==15095== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Valgrind example #3:



```
[2017-02-09 11:21:29 alex@Rincewind valgrind_examples]$ g++ -g test_wrong_delete
.cpp -o memcheck_test_wrong_delete.o
[2017-02-09 11:21:59 alex@Rincewind valgrind_examples]$ valgrind ./memcheck_test
_wrong_delete.o
==15095== Memcheck, a memory error detector
==15095== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15095== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==15095== Command: ./memcheck_test_wrong_delete.o
==15095==
==15095== Mismatched free() / delete / delete []
==15095==    at 0x4C2E76B: operator delete[](void*) (in /usr/lib/valgrind/vgprel
oad_memcheck-amd64-linux.so)
==15095==    by 0x1089BD: main (test_wrong_delete.cpp:11)
==15095==  Address 0x5ab9c80 is 0 bytes inside a block of size 4 alloc'd
==15095==    at 0x4C2D1AF: operator new(unsigned long) (in /usr/lib/valgrind/vgp
reload_memcheck-amd64-linux.so)
==15095==    by 0x108941: main (test_wrong_delete.cpp:6)
==15095==
==15095== Mismatched free() / delete / delete []
==15095==    at 0x4C2E26B: operator delete(void*) (in /usr/lib/valgrind/vgpreloa
d_memcheck-amd64-linux.so)
==15095==    by 0x1089CE: main (test_wrong_delete.cpp:12)
==15095==  Address 0x5ab9cd0 is 0 bytes inside a block of size 16 alloc'd
==15095==    at 0x4C2D8CF: operator new[](unsigned long) (in /usr/lib/valgrind/v
gpreload_memcheck-amd64-linux.so)
==15095==    by 0x10896E: main (test_wrong_delete.cpp:7)
==15095==
==15095==
==15095== HEAP SUMMARY:
==15095==     in use at exit: 0 bytes in 0 blocks
==15095==   total heap usage: 3 allocs, 3 frees, 72,724 bytes allocated
==15095==
==15095== All heap blocks were freed -- no leaks are possible
==15095==
==15095== For counts of detected and suppressed errors, rerun with: -v
==15095== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```