Valgrind

- Uninitialized variables
- Memory leaks
- Double frees
- Illegal memory access
- Stack overflow
- Thread errors
- Also profiling etc.

Argelander Institut für Astronomie

Building code for valgrind

- Use `-g` for debugging
- Turn off optimization with `-O0`

```
gcc -g -O0 example1.c -o example1
```

- Run code through *valgrind*'s default tool ***memcheck***

```
valgrind ./example1
```

```
==4409== Memcheck, a memory error detector
==4409== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==4409== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==4409== Command: ./example1
==4409==
hello world
==4409==
==4409== HEAP SUMMARY:
==4409==     in use at exit: 0 bytes in 0 blocks
==4409==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==4409==
==4409== All heap blocks were freed -- no leaks are possible
==4409==
==4409== For counts of detected and suppressed errors, rerun with: -v
==4409== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Uninitialized variables

```
#include <stdio.h>

int main(void)
{
  int i;
  printf("hello world i=%d\n",i);
  return 0;
}
```

```
> gcc -g -O0 example2.c -o example2
> ./example2
hello world i=0
```

Uninitialized variables

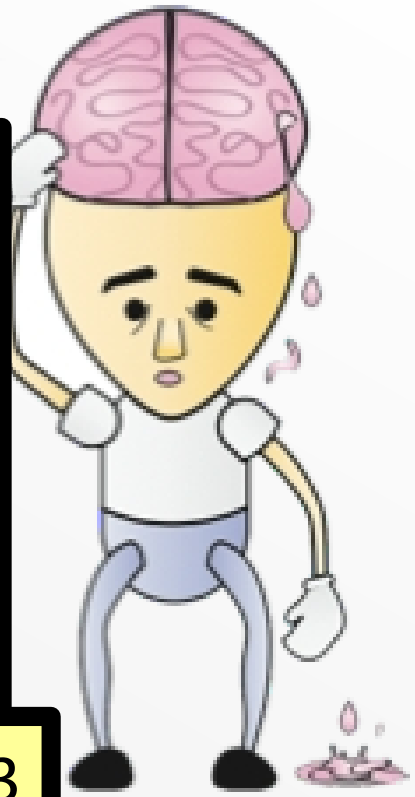
```
#include <stdio.h>

int r
{
   int
   pri
   ret
}
```

```
> valgrind ./example2
=4476== Memcheck, a memory error detector
....
==4476== Conditional jump or move depends on uninitialised value(s)
==4476==    at 0x4E8147E: vfprintf (vfprintf.c:1660)
==4476==    by 0x4E8B388: printf (printf.c:33)
==4476==    by 0x400548: main (example2.c:6)
==4476==
==4476== Use of uninitialised value of size 8
==4476==    at 0x4E8093B: _itoa_word (_itoa.c:179)
==4476==    by 0x4E845E6: vfprintf (vfprintf.c:1660)
==4476==    by 0x4E8B388: printf (printf.c:33)
==4476==    by 0x400548: main (example2.c:6)
....
==4476==
hello world i=0
==4476==
....
==4476== ERROR SUMMARY: 6 errors from 6 contexts (suppressed: 0 from 0)
```

Memory leaks

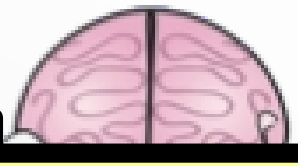
```
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
  double *x = malloc(sizeof(double)*512);
  printf("x[0]=%g at %p\n”);
  return(0);
}
```



```
> gcc -g -O0 example3.c -o example3
> ./example3
X[0]=0 at 0x1d4e010
```


Argelander Institut für Astronomie

Memory leaks



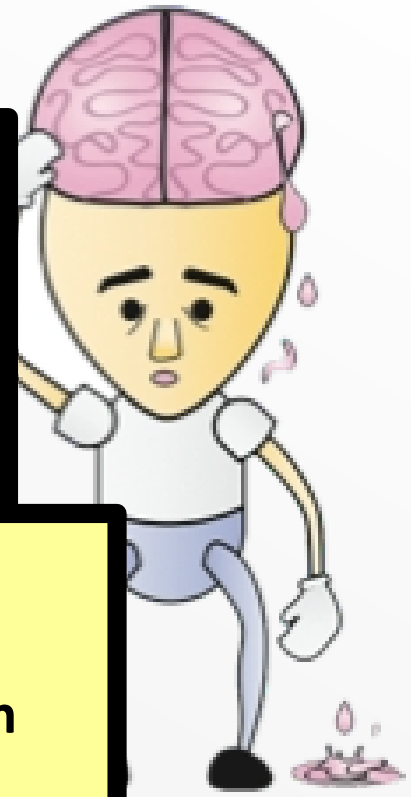
```
#include
#include
int main
{
  double
  printf
}
```

```
> valgrind ./example3
==4611== Memcheck, a memory error detector
....
==4611== Conditional jump or move depends on uninitialised value(s)
==4611==    by 0x4005B8: main (example3.c:6)
....
x[0]=0 at 0x51fd040
==4611==
==4611== HEAP SUMMARY:
==4611==     in use at exit: 4,096 bytes in 1 blocks
==4611==   total heap usage: 1 allocs, 0 frees, 4,096 bytes allocated
==4611==
==4611== LEAK SUMMARY:
==4611==    definitely lost: 4,096 bytes in 1 blocks
==4611== Rerun with --leak-check=full to see details of leaked memory
==4611==
==4611== For counts of detected and suppressed errors, rerun with: -v
==4611== Use --track-origins=yes to see where uninitialised values come from
==4611== ERROR SUMMARY: 22 errors from 14 contexts (suppressed: 0 from 0)
```

Memory leaks

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
  double *x = malloc(sizeof(double)*512);

}
```



```
> valgrind --track-origins=yes --leak-check=full ./example3
....
==4687== Uninitialised value was created by a heap allocation
==4687==    at 0x4C2AB80: malloc (in
/usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4687==    by 0x40058E: main (example3.c:5)
....
```

Double frees

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
  double *x = malloc(sizeof(double)*512);
  free(x);
  free(x);
  return 0;
}
```

```
> gcc -g -O0 example4.c -o example4
> ./example4
./example4
*** Error in `./example4': double free or corruption (top): 0x0000000001479010 ***
Aborted (core dumped)
```

Double frees

```
> valgrind ./example4
==4757== Memcheck, a memory error detector
.....
==4757== Invalid free() / delete / delete[] / realloc()
==4757==    at 0x4C2BDEC: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4757==    by 0x4005AA: main (example4.c:7)
==4757==  Address 0x51fd040 is 0 bytes inside a block of size 4,096 free'd
==4757==    at 0x4C2BDEC: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4757==    by 0x40059E: main (example4.c:6)
==4757==
==4757==
==4757== HEAP SUMMARY:
==4757==     in use at exit: 0 bytes in 0 blocks
==4757==   total heap usage: 1 allocs, 2 frees, 4,096 bytes allocated
==4757==
==4757== All heap blocks were freed -- no leaks are possible
==4757==
==4757== For counts of detected and suppressed errors, rerun with: -v
==4757== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

```c
#include <stdlib.h>
#include <stdio.h>
#define SAFE_FREE(A) if((A)!=NULL){free(A);(A)=NULL;};

int main(void)
{
  double *x = malloc(sizeof(double)*512);
  SAFE_FREE(x);
  SAFE_FREE(x);
  return 0;
}
```

Illegal memory access

```
int main(void)
{
   double x[3];
   x[0]=1.0;
   x[1]=2.0;
   x[2]=3.0;
   x[3]=4.0;

   return 0;
}
```

```
> gcc -g -O0 example5.c -o example5
> ./example5
>
```

Illegal memory access

```
int main(void)
```

```
> gcc -g -O0 example5.c -o example5
>valgrind ./example5
==4848== Memcheck, a memory error detector
==4848== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==4848== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==4848== Command: ./example5
==4848==
==4848==
==4848== HEAP SUMMARY:
==4848==     in use at exit: 0 bytes in 0 blocks
==4848==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==4848==
==4848== All heap blocks were freed -- no leaks are possible
==4848==
==4848== For counts of detected and suppressed errors, rerun with: -v
==4848== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Illegal memory access

```
#include <stdlib.h>
int main(void)
{
  double *x=malloc(sizeof(double)*3);
  x[0]=1.0;
  x[1]=2.0;
  x[2]=3.0;
  x[3]=4.0;
  free(x);
  return 0;
}
```

Illegal memory access

```
#include <stdlib.h>
```

```
> gcc -g -O0 example5a.c -o example5a
>valgrind ./example5a
  ==4905== Memcheck, a memory error detector
==4905== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==4905== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright info
==4905== Command: ./example5a
==4905==
==4905== Invalid write of size 8
==4905==    at 0x4005E0: main (example5a.c:8)
==4905==  Address 0x51fd058 is 0 bytes after a block of size 24 alloc'd
==4905==    at 0x4C2AB80: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4905==    by 0x40058E: main (example5a.c:4)
==4905==
....
==4905== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Stack overflow: recursion

```
void main(void)
{
   foo();
}
int foo(void)
{
   return foo();
}
```

```
> gcc -g -O0 example6.c -o example6
> ./example6
Segmentation fault (core dumped)
>
```

Stack overflow: recursion

```
void main(void)
{
```

```
> gcc -g -O0 example6.c -o example6
> valgrind ./example6
==4980== Memcheck, a memory error detector
....
==4980== Stack overflow in thread 1: can't grow stack to 0xffe801ff8
==4980==
==4980== Process terminating with default action of signal 11 (SIGSEGV)
==4980==  Access not within mapped region at address 0xFFE801FF8
==4980==    at 0x400501: foo (example6.c:7)
 .....
==4980== For counts of detected and suppressed errors, rerun with: -v
==4980== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Segmentation fault (core dumped)
```

Stack overflow: big arrays

```
void main(void)
{
   int x[10000000];
}
```

```
> gcc -g -O0 example7.c -o example7
> ./example7
Segmentation fault (core dumped)
>
```

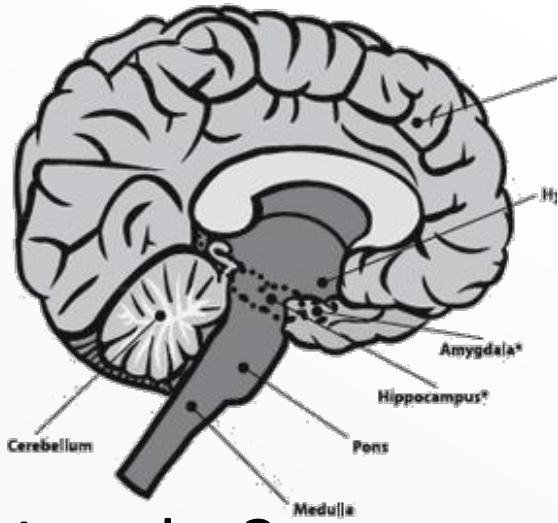
Stack overflow: big arrays

```
void main(void)
{
```

```
> gcc -g -O0 example7.c -o example7
> valgrind ./example7
....
==5110== Process terminating with default action of signal 11 (SIGSEGV)
==5110==  Access not within mapped region at address 0xFFC9DA1A8
==5110==    at 0x4004F8: main (example7.c:4)
==5110==  If you believe this happened as a result of a stack
==5110==  overflow in your program's main thread (unlikely but
==5110==  possible), you can try to increase the size of the
==5110==  main thread stack using the --main-stacksize= flag.
==5110==  The main thread stack size used in this run was 8388608
```

--help-debug	Print Valgrind help command plus debugging option.
--q	Show only the error message and ignore the others (--quiet)
--version	Print Valgrind software version.
- -leak-check = <no|summary|full>	exec memory-leak analysis. A detected memory-leak means a block of allocated memory has not been freed and to which all references have been lost. So the block can now not be deallocated. This flags shows how many memory leak have been matched. The option full shows a lot of detail. When doing leak detection Valgrind tracks all memory block allocations. When the program finishes it prints which blocks have not been freed.

Valgrind



- Framework for other tools, e.g.
- **Cachegrind** : how much do you hit the CPU cache?
  - `--tool=cachegrind`
  - View output with e.g. **kcachegrind**
- **Callgrind** : cache/branch prediction profiler
- **Helgrind** and **DRD** : thread error detectors
- **Massif/DHAT** : heap analysis
- **SGCheck** : array overrun detector (experimental!)

+ others

Argelander Institut für Astronomie