Memcheck Tool

A Memory Error Detector for C and C++ programs

https://valgrind.org/docs/manual/mc-manual.html

Carmen Torres Lopez
Elisa Gonzalez Boix
Maarten Vandercammen

Could we check our programs for memory leaks?

http://valgrind.org

- Instrumentation framework for building dynamic analysis tools.
- Includes a **memory error detector (Memcheck)** to automatically detect memory management errors.
- Other tools:
  - a cache and branch-prediction profiler
  - a call-graph generating cache
  - branch-prediction profiler
  - two different heap profilers
  - two thread error detectors

What does Memcheck Detect for C programs?

https://valgrind.org/docs/manual/mc-manual.html

1. **Accessing memory you shouldn't**, e.g. accessing an address in the heap memory beyond the size that was requested, accessing memory after it has been freed, etc.
2. **Using undefined values**
3. **Incorrect freeing of heap memory**, such as double-freeing heap blocks.
4. **Overlapping `src` and `dst` pointers in `memcpy` and related functions**.
5. **Passing a fishy (presumably negative) value** to the `size` parameter of a memory allocation function.
6. **Memory leaks**.

Memcheck by Example 1

```c
#include <stdlib.h>

void f(void)
{
      int* x = malloc(10 * sizeof(int));
      x[10] = 0;        // problem 1: heap block overrun
}                       // problem 2: memory leak -- x not freed

int main(void)
{
  f();
  return 0;
}
```

Tip: Fix errors in the order they are reported, as later errors can be caused by earlier ones.

Error message for problem 1: Heap block overrun

```
==19182== Invalid write of size 4
==19182==    at 0x804838F: f (example.c:6)
==19182==    by 0x80483AB: main (example.c:11)
```

Kind of error

Stack trace

Process ID

https://valgrind.org/docs/manual/quick-start.html

Memcheck by Example 2

```
#include <stdlib.h>

void f(void)
{
     int* x = malloc(10 * sizeof(int));
     x[10] = 0;      // problem 1: heap block overrun
}                    // problem 2: memory leak -- x not freed

int main(void)
{
  f();
  return 0;
}
```

Error message for problem 2: Memory leaks

```
==19182== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==19182==    at 0x1B8FF5CD: malloc (vg_replace_malloc.c:130)
==19182==    by 0x8048385: f (a.c:5)
==19182==    by 0x80483AB: main (a.c:11)
```

Definitely lost: the program is leaking memory.

The stack trace tells you where the leaked memory was allocated.

Valgrind @ Eclipse in Linux

Linux Tools Project Valgrind plugin: http://www.eclipse.org/linuxtools/projectPages/valgrind/

Step 0. Install Valgrind in your computer. Write in the terminal:

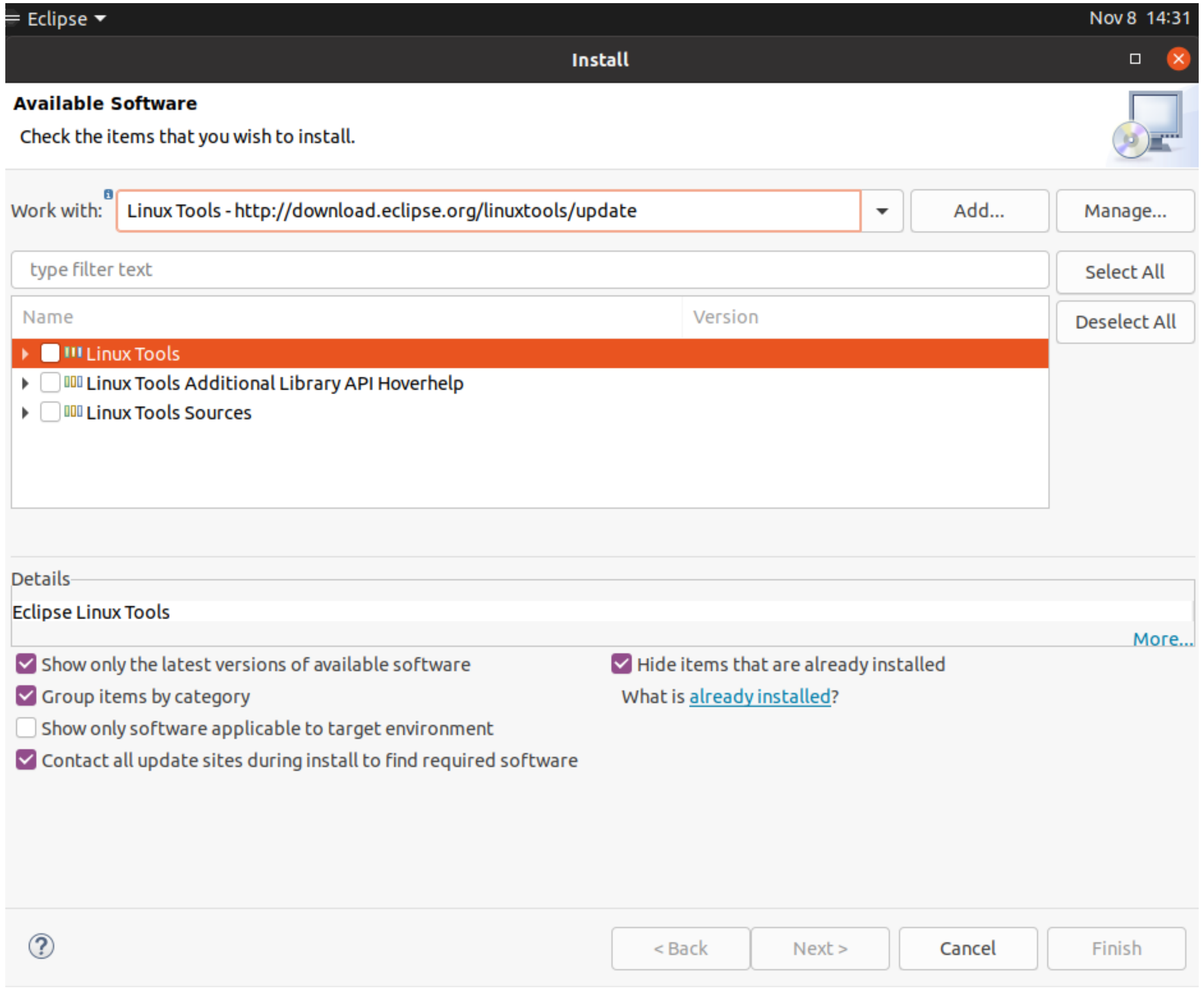
```
sudo apt-get install valgrind
```

Step 1. Install plugin Linux Tools Project Valgrind: http://wiki.eclipse.org/Linux_Tools_Project/PluginInstallHelp

Step 2. Create new project with your C program and run Valgrind (see next slides)

Step 1: Install Linux Tools plugin in Eclipse

Select the three Linux Tools options shown below



Available Software

Check the items that you wish to install.

Work with: Linux Tools - http://download.eclipse.org/linuxtools/update

Name	Version
☐ Linux Tools	
☐ Linux Tools Additional Library API Hoverhelp	
☐ Linux Tools Sources	

Details

Eclipse Linux Tools

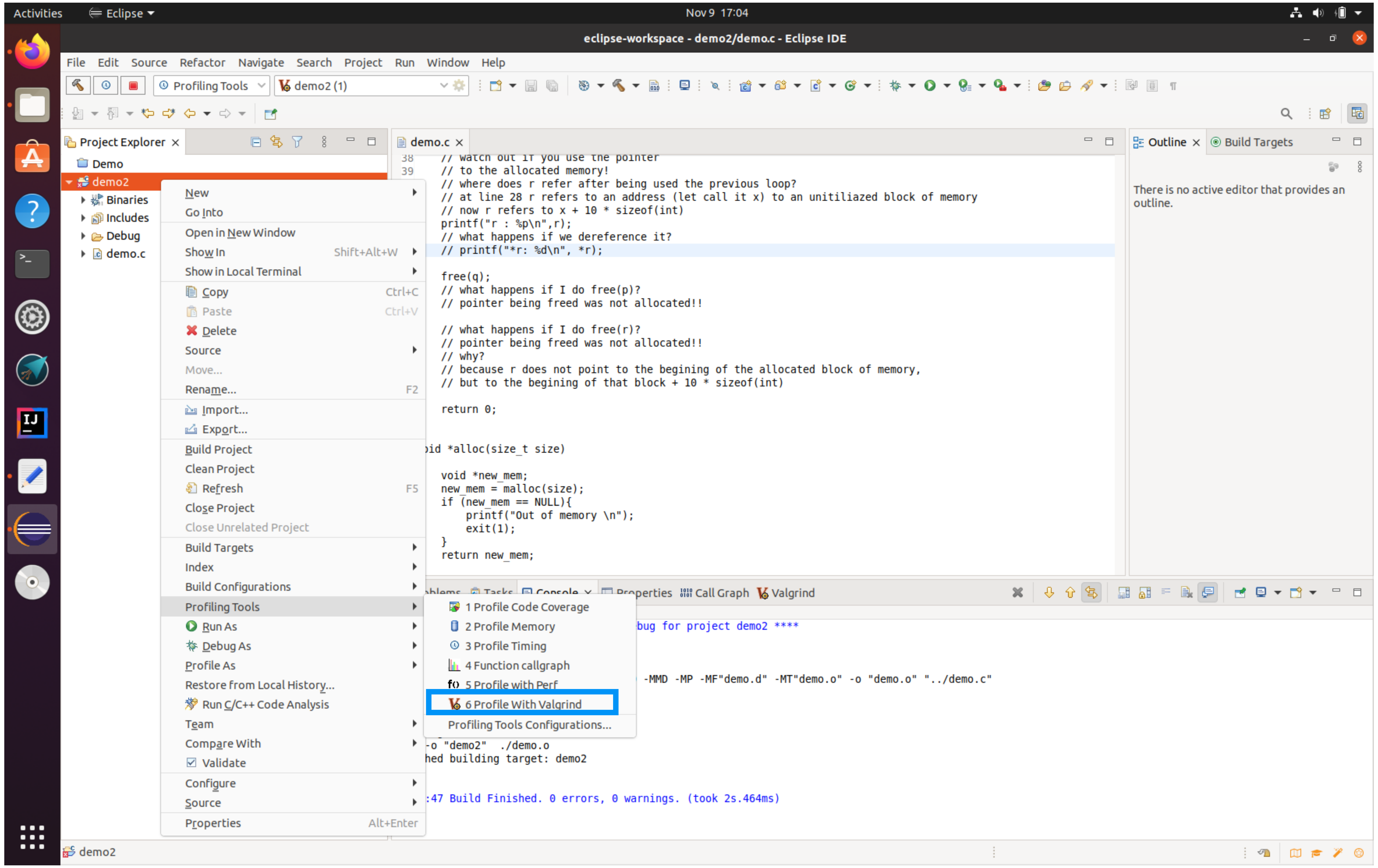
- ☑ Show only the latest versions of available software
- ☑ Group items by category
- ☐ Show only software applicable to target environment
- ☑ Contact all update sites during install to find required software
- ☑ Hide items that are already installed

What is already installed?

Step 2: Run Memcheck in Eclipse

Right click on the project and select **Profiling Tools** -> **Profile with Valgrind**

Demo program in Files/HOC/Code/Chapter3b/mallocEx2



```
    // watch out if you use the pointer
    // to the allocated memory!
    // where does r refer after being used the previous loop?
    // at line 28 r refers to an address (let call it x) to an unitiliazed block of memory
    // now r refers to x + 10 * sizeof(int)
    printf("r : %p\n",r);
    // what happens if we dereference it?
    // printf("*r: %d\n", *r);

    free(q);
    // what happens if I do free(p)?
    // pointer being freed was not allocated!!

    // what happens if I do free(r)?
    // pointer being freed was not allocated!!
    // why?
    // because r does not point to the begining of the allocated block of memory,
    // but to the begining of that block + 10 * sizeof(int)

    return 0;


oid *alloc(size_t size)

    void *new_mem;
    new_mem = malloc(size);
    if (new_mem == NULL){
        printf("Out of memory \n");
        exit(1);
    }
    return new_mem;
```

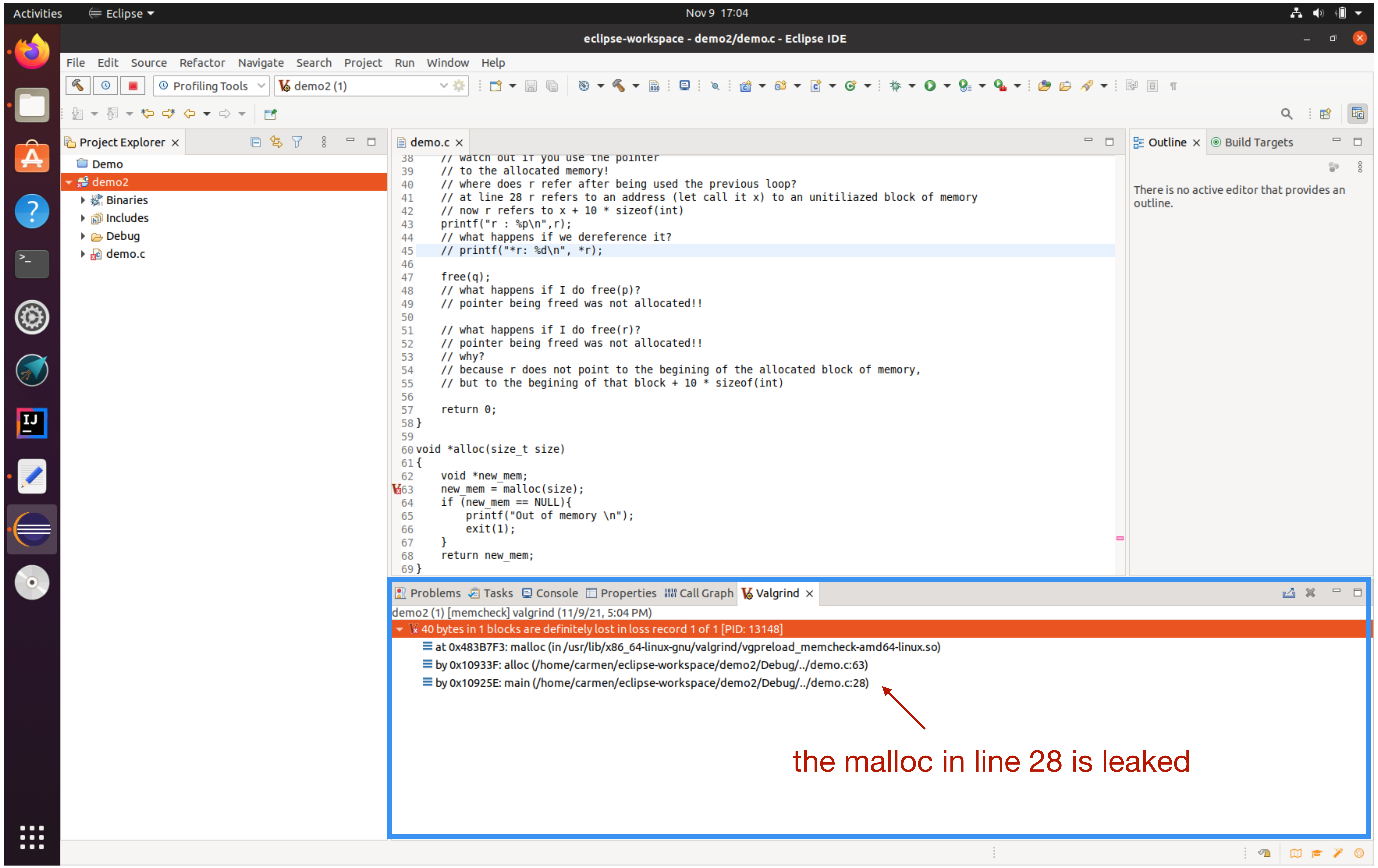
```
bug for project demo2 ****

 -MMD -MP -MF"demo.d" -MT"demo.o" -o "demo.o" "../demo.c"

-o "demo2"  ./demo.o
hed building target: demo2

:47 Build Finished. 0 errors, 0 warnings. (took 2s.464ms)
```

Results of Memcheck in Eclipse


Activities
Eclipse
Nov 9 17:04
eclipse-workspace - demo2/demo.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Profiling Tools
demo2 (1)
Project Explorer
Demo
demo2
Binaries
Includes
Debug
demo.c
Outline
Build Targets
There is no active editor that provides an outline.
Problems Tasks Console Properties Call Graph Valgrind

```
38      // watch out if you use the pointer
39      // to the allocated memory!
40      // where does r refer after being used the previous loop?
41      // at line 28 r refers to an address (let call it x) to an unitiliazed block of memory
42      // now r refers to x + 10 * sizeof(int)
43      printf("r : %p\n",r);
44      // what happens if we dereference it?
45      // printf("*r: %d\n", *r);
46
47      free(q);
48      // what happens if I do free(p)?
49      // pointer being freed was not allocated!!
50
51      // what happens if I do free(r)?
52      // pointer being freed was not allocated!!
53      // why?
54      // because r does not point to the begining of the allocated block of memory,
55      // but to the begining of that block + 10 * sizeof(int)
56
57      return 0;
58  }
59
60  void *alloc(size_t size)
61  {
62      void *new_mem;
63      new_mem = malloc(size);
64      if (new_mem == NULL){
65          printf("Out of memory \n");
66          exit(1);
67      }
68      return new_mem;
69  }
```

```
demo2 (1) [memcheck] valgrind (11/9/21, 5:04 PM)
  40 bytes in 1 blocks are definitely lost in loss record 1 of 1 [PID: 13148]
    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
    by 0x10933F: alloc (/home/carmen/eclipse-workspace/demo2/Debug/../demo.c:63)
    by 0x10925E: main (/home/carmen/eclipse-workspace/demo2/Debug/../demo.c:28)
```

the malloc in line 28 is leaked

Valgrind @ CLion in Linux

Valgrind Memcheck in CLion: https://www.jetbrains.com/help/clion/memory-profiling-with-valgrind.html#start

Note: This option only works for Linux, Windows and older (pre Sierra) versions of MacOS

Alternative for Valgrind in MacOS

Run Address Sanitizer

1.1 For detecting dangling pointer dereferences:

```
$ clang -fsanitize=address -g program.c
$ ./a.out
```

Step 1: you need to compile and link your program using clang with `-fsanitize`

Step 2: run the executable

If you get the following error

```
clang: error: unsupported option '-fsanitize=leak' for target 'x86_64-apple-darwin20.6.0
```

- Install LLVM on Homebrew (https://stackoverflow.com/questions/53456304/mac-os-leaks-sanitizer)

```
brew install llvm@8
```

- Overwrite default Clang

```
echo 'export PATH="/usr/local/opt/llvm@8/bin:$PATH"' >> ~/.bash_profile
```

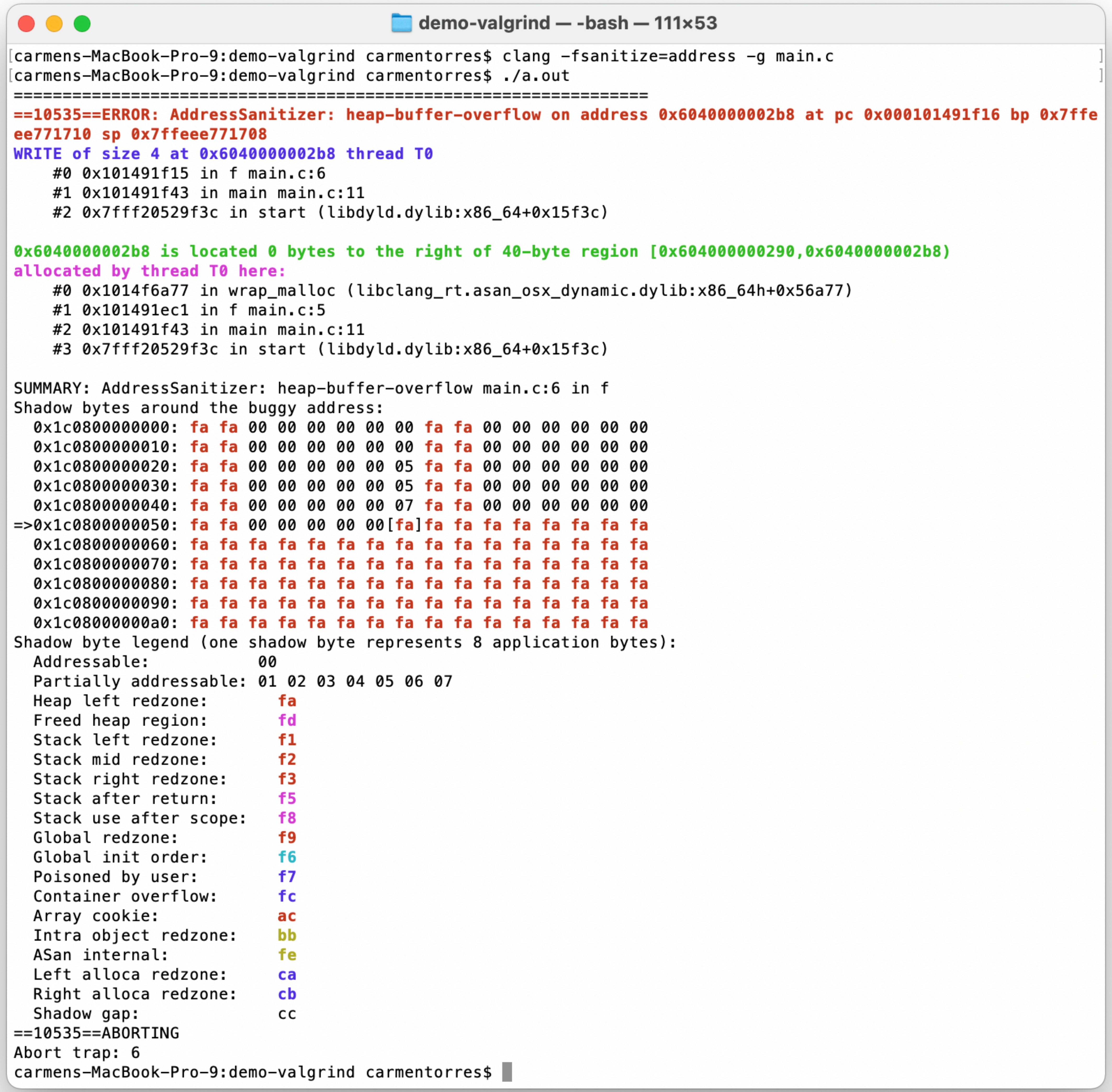
Address Sanitizer by Example 1

```c
#include <stdlib.h>

void f(void)
{
     int* x = malloc(10 * sizeof(int));
     x[10] = 0;       // problem 1: heap block overrun
}                     // problem 2: memory leak -- x not freed

int main(void)
{
  f();
  return 0;
}
```

Error message for problem 1: Heap block overflow



demo-valgrind — -bash — 111×53

```
carmens-MacBook-Pro-9:demo-valgrind carmentorres$ clang -fsanitize=address -g main.c
carmens-MacBook-Pro-9:demo-valgrind carmentorres$ ./a.out
=================================================================
==10535==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x6040000002b8 at pc 0x000101491f16 bp 0x7ffe
ee771710 sp 0x7ffeee771708
WRITE of size 4 at 0x6040000002b8 thread T0
    #0 0x101491f15 in f main.c:6
    #1 0x101491f43 in main main.c:11
    #2 0x7fff20529f3c in start (libdyld.dylib:x86_64+0x15f3c)

0x6040000002b8 is located 0 bytes to the right of 40-byte region [0x604000000290,0x6040000002b8)
allocated by thread T0 here:
    #0 0x1014f6a77 in wrap_malloc (libclang_rt.asan_osx_dynamic.dylib:x86_64h+0x56a77)
    #1 0x101491ec1 in f main.c:5
    #2 0x101491f43 in main main.c:11
    #3 0x7fff20529f3c in start (libdyld.dylib:x86_64+0x15f3c)

SUMMARY: AddressSanitizer: heap-buffer-overflow main.c:6 in f
Shadow bytes around the buggy address:
  0x1c0800000000: fa fa 00 00 00 00 00 00 fa fa 00 00 00 00 00 00
  0x1c0800000010: fa fa 00 00 00 00 00 00 fa fa 00 00 00 00 00 00
  0x1c0800000020: fa fa 00 00 00 00 00 05 fa fa 00 00 00 00 00 00
  0x1c0800000030: fa fa 00 00 00 00 00 05 fa fa 00 00 00 00 00 00
  0x1c0800000040: fa fa 00 00 00 00 00 07 fa fa 00 00 00 00 00 00
=>0x1c0800000050: fa fa 00 00 00 00 00[fa]fa fa fa fa fa fa fa fa
  0x1c0800000060: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0800000070: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0800000080: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c0800000090: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c08000000a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==10535==ABORTING
Abort trap: 6
carmens-MacBook-Pro-9:demo-valgrind carmentorres$
```

To get nicer stack traces in error messages add
`-fno-omit-frame-pointer`

Address Sanitizer by Example 2

```c
#include <stdlib.h>

void f(void)
{
     int* x = malloc(10 * sizeof(int));
     x[10] = 0;       // problem 1: heap block overrun
}                     // problem 2: memory leak -- x not freed

int main(void)
{
  f();
  return 0;
}
```

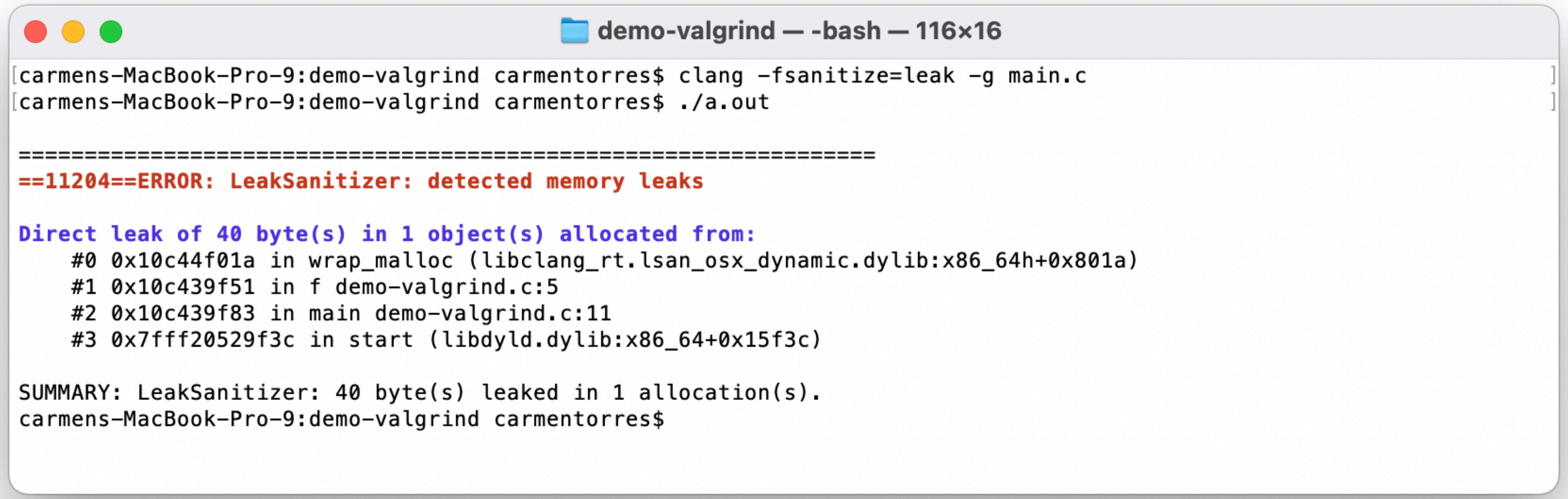
Run Address Sanitizer

For detecting memory leaks (Leak Sanitizer):

```
$ clang -fsanitize=leak -g program.c
$ ./a.out
```

https://clang.llvm.org/docs/LeakSanitizer.html

Error message for problem 2: Memory leaks



```
carmens-MacBook-Pro-9:demo-valgrind carmentorres$ clang -fsanitize=leak -g main.c
carmens-MacBook-Pro-9:demo-valgrind carmentorres$ ./a.out

=================================================================
==11204==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 40 byte(s) in 1 object(s) allocated from:
    #0 0x10c44f01a in wrap_malloc (libclang_rt.lsan_osx_dynamic.dylib:x86_64h+0x801a)
    #1 0x10c439f51 in f demo-valgrind.c:5
    #2 0x10c439f83 in main demo-valgrind.c:11
    #3 0x7fff20529f3c in start (libdyld.dylib:x86_64+0x15f3c)

SUMMARY: LeakSanitizer: 40 byte(s) leaked in 1 allocation(s).
carmens-MacBook-Pro-9:demo-valgrind carmentorres$
```

Leak Sanitizer shows memory leaks under the label “Direct leak”