TA1 – Introduction

OPERATING SYSTEMS COURSE

THE HEBREW UNIVERSITY

SPRING 2020

Today's Plan

- Administrations
- Nand Review
- Virtualization
- Exercise 1
- Strace
- Extras - "Live" demo
  - SSH
  - Valgrind
  - GDB
  - Clion - only slides, we will not talk about it

Administrations

צוות הקורס

- דוד חי – מרצה
- נטע רוזן-שיף - מרצה
- איתן ליפשיץ – מתרגל
- עידן רפאלי – מתרגל
- איהאב זחאיקה – מתרגל
- יובל יעקבי - מתרגל
- אמיתי דבח – צאר
- שמעון בצלאל - צאר

מנהלות

- 4 תרגילי חובה – חובה לקבל לפחות 60 בממוצע התרגילים.
- ייתכן תרגיל רשות.
- התרגילים יכללו שאלות תאורטיות ומשימות תכנותיות.
- לוודא שהתרגילים מתקמפלים.
- ייתכן שיהיו ראיונות על תרגילים 2-4.
- כנסו לאתר הקורס ועיינו בנהלים בקפידה.
- אפס סובלנות להעתקות
- שימוש ב-moodle לצורך הודעות, שאלות, פורום תלמידים, הגשות וכו'.
- כתובת הדואר של הקורס os@cs.huji.ac.il

Course Motivation (1)

- How can you play music, scan for viruses and read a document with a single processor?
- What happens when you press a button?
- What happens when you use “write” or “read” of a file?
- Reading a file is extremely slow, how does the computer keep being responsive?
- What does “segmentation fault” mean?

Course Motivation (2)

- How can we run programs that use more memory than we actually have?
- How can you speed up programs by using multi-core system?
  - There are a few main problems that arise from this, we will study how to solve them.
- Sending messages between computers (communication)

Course Motivation (3)

THE MOST IMPORTANT COURSE FOR THE INDUSTRY

Computer Hardware

Live Coding

- 67100
- ssh to the “aquarium“
- Huji Git
  - This TA on git
  - git tutorial
  - git cheatsheet
- vim
  - cheatsheet
  - tutorial 1 , tutorial 2
- Address Sanitizer

CPU (Central Processing Unit)

- CPU contains registers.
- Important registers are:
  - IR – Instruction Register
  - PC – Program Counter
  - SP – Stack Pointer

- Executes a set of instructions:
  - **Data handling** – set a register, store , load
  - **Arithmetic operations** - Bitwise operations, compare, and basic mathematics operations.
  - **Control flow** – branch, conditional branch

- Each instruction is represented by an unique binary number (a 'word'). For example, the following 32-bit word:
  000000100001000101000000000100000

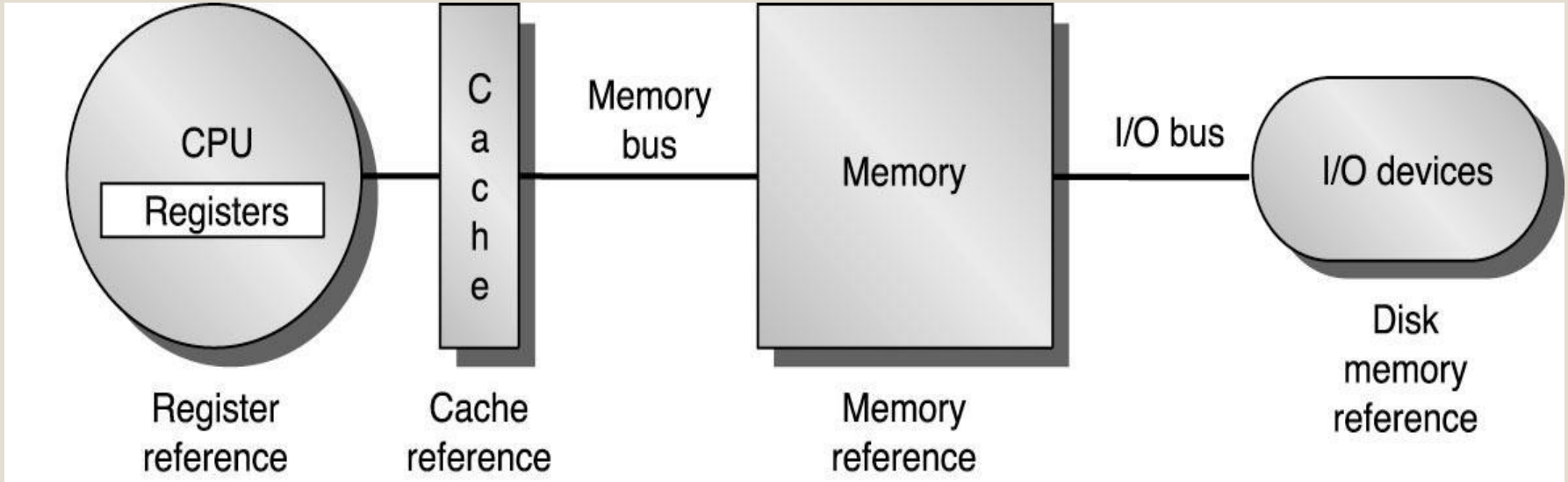
  means: add $8, $16, $17

CPU Instruction Life-Cycle

- Instruction fetch
- Decode instruction and registers fetch
- ALU operation
- Memory access if required (lw, sw)
- Register write-back (lw, add)

- For Example:
  - add $8, $16, $17 ☐ Reg[8]=Reg[16]+Reg[17]
  - lw $1, 32($2) ☐ Reg[1] = M[Reg[2]+32]
  - sw $3, 12($4) ☐ M[Reg[4]+12] = Reg[3]

Typical Memory Hierarchy



Size:	500 bytes	64 KB	512 MB	100 GB
Speed:	0.25 ns	1 ns	100 ns	5 ms
Current Size	~same	8MB	8GB	2TB

© 2003 Elsevier Science (USA). All rights reserved.

Memory Hierarchy

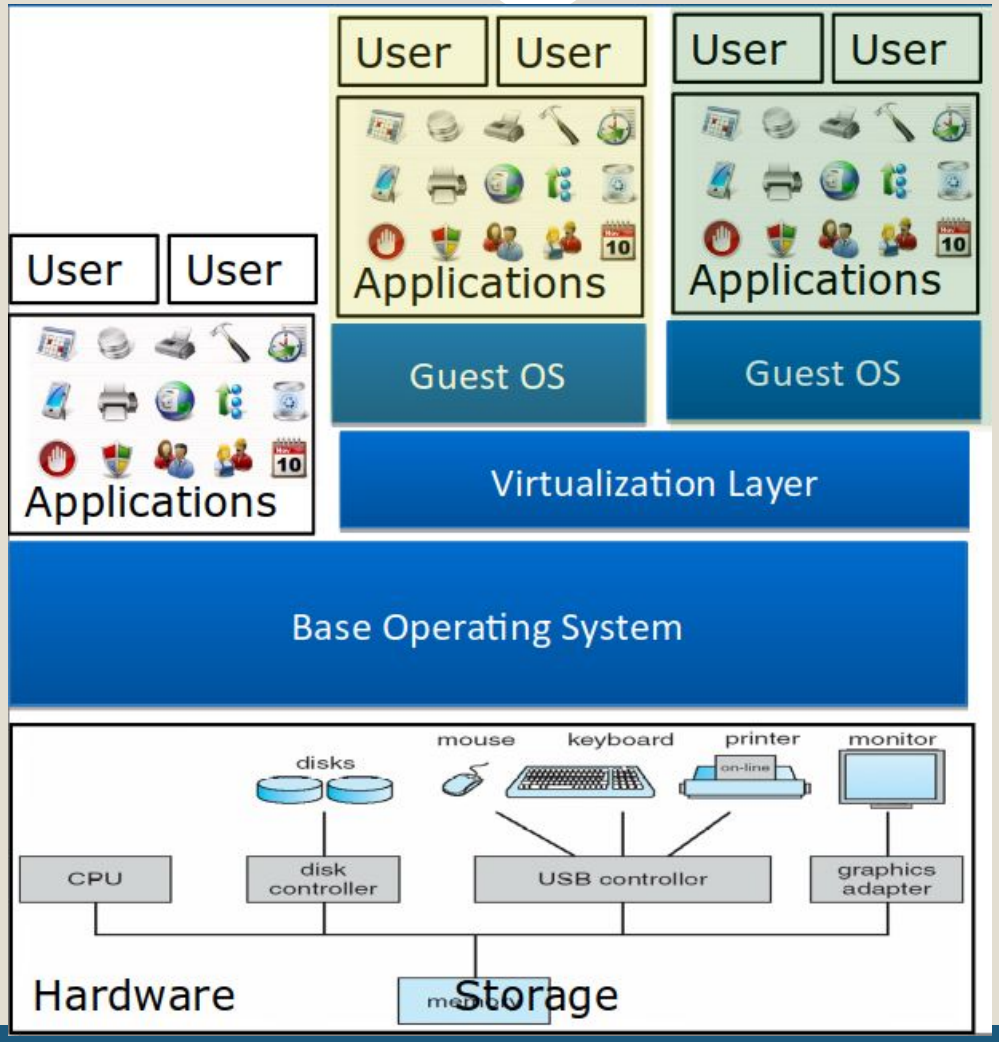
- Main Memory - located on chips inside the computer (outside CPU).
- The program instructions and the process data are kept in main memory.
- External Memory - disk. Information stored on a disk is not deleted when the computer turned off.
- The main memory has less storage capacity than the hard disk. The hard disk can write and read information to and from the main memory. The access speed of main memory is much faster than a hard disk.
- Programs are stored on the disk until they are loaded into memory, then they use the disk as both the source and destination of the information for their processing.

Virtualization

Virtualization

- **Virtualization** refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources (Wikipedia).
- A **virtual machine** (VM) is an emulation of computer system. There can be many VMs on each physical computer system, and each VM may have its own “virtual” resources and operating system, so that it “feels” like working with physical machine.

Virtualization



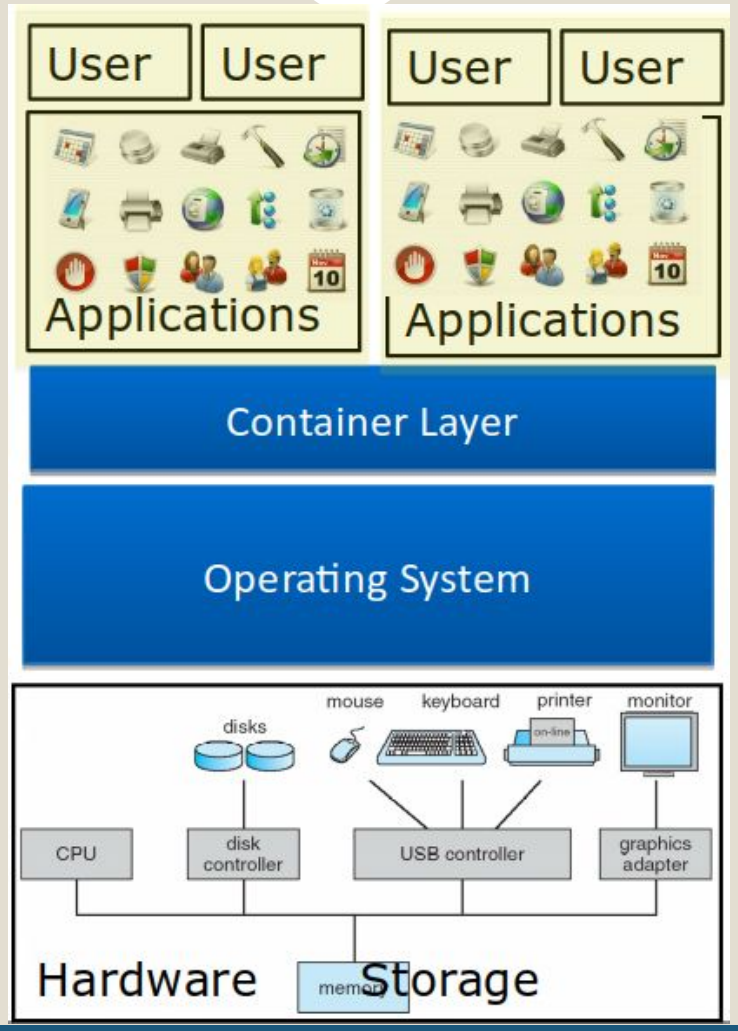
Virtualization

- Running several VMs, each with its own OS, consumes many system resources of the physical machine – disk space, RAM, CPU cycles…
- That's a lot of overhead
- Containers come to the rescue!

Virtualization

- A **container** is an OS-level virtualization, that is an OS which allows the existence of multiple isolated user space instances.
- A program running inside of a container can only see the container's contents and devices assigned to the container.
- Multiple containers can run on the same machine and share the OS kernel with other containers.

Virtualization



Ex1 Motivation

Ex1 Motivation

- Different instructions have different running times
- Function call - overhead of maintaining the working environment of each function
- System call – extra overhead of switching to kernel space

Ex1 Motivation

- In Ex1 you'll compare the running times of different kind of operations:
  - Directly on the computer
  - Inside the VM
  - Inside the container

Debugging System Calls

Debugging by Watching

- **strace** is a debugging utility to monitor the system calls
  - Easy to use
  - Fast debugginng
- **strace** command
  - Shows system calls, arguments, and return values
  - **-t** to display when each call is executed
  - **-T** to display the time spent in the call
  - **-e** to limit the types of calls
  - **-o** to redirect the output to a file
  - **-s** limit the length of print strings.

Background – basic commands

int open(const char *pathname, int flags);

- opens a file
- Returns file descriptor (fd), which identifies the file in future operations
- fd=0 -> standard input, fd=1 -> standard outpout, fd=2 -> standard error

ssize_t read(int fd, void *buf, size_t count);

- Reads from fd to buf between 1 to count bytes
- Returns the number of bytes that were read (zero for eof)

ssize_t write(int fd, const void *buf, size_t count);

- Writes from buf to fd between 1 to count bytes
- Returns the number of bytes that were written

Strace example: “strace ls /python/”

- open("/usr/share/locale/en_GB/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
- write(2, "ls: ", 4) = 4
- write(2, "cannot access /python/", 22) = 22
- open("/usr/share/locale/en_US/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
- open("/usr/share/locale/en/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
- open("/usr/share/locale/en_GB/LC_MESSAGES/libc.mo", O_RDONLY) = 3
- fstat(3, {st_mode=S_IFREG|0644, st_size=1474, ...}) = 0
- mmap(NULL, 1474, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f30b2df1000
- close(3) = 0
- write(2, ": No such file or directory", 27) = 27
- write(2, "\n", 1) = 1
- close(1) = 0

Strace example: “strace ls python/”

- stat("python/", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
- openat(AT_FDCWD, "python/", O_RDONLY|O_NONBLOCK|O_DIRECTORY|O_CLOEXEC) = 3
- fcntl(3, F_GETFD) = 0x1 (flags FD_CLOEXEC)
- getdents(3, /* 8 entries */, 32768) = 240
- getdents(3, /* 0 entries */, 32768) = 0
- close(3) = 0
- fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
- mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x79ac55f24000
- write(1, "proj1 proj2 proj3 proj4 proj"..., 41) = 41
- close(1) = 0

Strace example: "strace wc sample2.in"

- stat("sample2.in", {st_mode=S_IFREG|0777, st_size=490, ...}) = 0
- open("sample2.in", O_RDONLY) = 3
- read(3, "\" The path of the righteous man "..., 16384) = 490
- open("/usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache", O_RDONLY) = 4
- fstat(4, {st_mode=S_IFREG|0644, st_size=26066, ...}) = 0
- mmap(NULL, 26066, PROT_READ, MAP_SHARED, 4, 0) = 0x7f81a4c88000
- close(4) = 0
- read(3, "", 16384) = 0
- fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 4), ...}) = 0
- mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f81a4c87000
- write(1, " 1 96 490 sample2.in\n", 23) = 23
- close(3) = 0
- close(1) = 0

Valgrind (Debugging)

Valgrind

- Framework debugging and profiling code on Linux system.
- The most used tool is Memcheck, which can detect memory errors.

Ex1.c

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

typedef struct Foo {
        int arr[3];
        int bar;
} Foo;

int main(int argc, char *argv[]) {
        Foo t;
        int i;
        printf("start\n");
        t.bar = 6;
        printf("t.bar =  %d\n", t.bar);
        for (i=0; i<3; i++) {
                t.arr[i] = i+1;
        }
        t.arr[i] += t.arr[0] + t.arr[1];
        printf("t.arr[2] = %d\n", t.arr[2]);
        printf("t.bar = %d\n", t.bar);
        return 0;
}
```

Ex1.c


```
<70|0>orenstal@pond:~/os17/tirgutl1% gcc -g -Wall ex1.c -o example
<71|0>orenstal@pond:~/os17/tirgutl1% ./example
start
t.bar = 6
t.arr[2] = 3
t.bar = 9
<72|0>orenstal@pond:~/os17/tirgutl1%
<72|0>orenstal@pond:~/os17/tirgutl1% valgrind example
==11813== Memcheck, a memory error detector
==11813== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==11813== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==11813== Command: example
==11813==
start
t.bar = 6
t.arr[2] = 3
t.bar = 9
==11813==
==11813== HEAP SUMMARY:
==11813==     in use at exit: 0 bytes in 0 blocks
==11813==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==11813==
==11813== All heap blocks were freed -- no leaks are possible
==11813==
==11813== For counts of detected and suppressed errors, rerun with: -v
==11813== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
<73|0>orenstal@pond:~/os17/tirgutl1%
```

Ex2.c

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

void foo(int n) {
        int i;
        int *a = (int*) malloc(n*sizeof(int));

        a[0] = 1;
        printf("a[0] = 1\n");
        a[1] = 1;
        printf("a[1] = 1\n");

        for (i=2; i<=n; i++) {
                a[i] = a[i-1] + a[i-2];              printf("a[%d] =
    %d\n", i, a[i]);
        }

        free(a);
}

int main(int argc, char *argv[]) {
        foo(10);
        return 0;
}
```

Ex2.c


```
<73|0>orenstal@pond:~/os17/tirgutl1% gcc -g -Wall ex2.c -o example
<74|0>orenstal@pond:~/os17/tirgutl1% ./example
a[0] = 1
a[1] = 1
a[2] = 2
a[3] = 3
a[4] = 5
a[5] = 8
a[6] = 13
a[7] = 21
a[8] = 34
a[9] = 55
a[10] = 89
<75|0>orenstal@pond:~/os17/tirgutl1%
```

```
<75|0>orenstal@pond:~/os17/tirgutl1% valgrind example
==11990== Memcheck, a memory error detector
==11990== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==11990== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==11990== Command: example
==11990==
a[0] = 1
a[1] = 1
a[2] = 2
a[3] = 3
a[4] = 5
a[5] = 8
a[6] = 13
a[7] = 21
a[8] = 34
a[9] = 55
==11990== Invalid write of size 4
==11990==    at 0x40065F: foo (ex2.c:14)
==11990==    by 0x4006BE: main (ex2.c:22)
==11990==  Address 0x51d9068 is 0 bytes after a block of size 40 alloc'd
==11990==    at 0x4C2BBCF: malloc (vg_replace_malloc.c:299)
==11990==    by 0x4005E1: foo (ex2.c:6)
==11990==    by 0x4006BE: main (ex2.c:22)
==11990==
==11990== Invalid read of size 4
==11990==    at 0x400675: foo (ex2.c:15)
==11990==    by 0x4006BE: main (ex2.c:22)
==11990==  Address 0x51d9068 is 0 bytes after a block of size 40 alloc'd
==11990==    at 0x4C2BBCF: malloc (vg_replace_malloc.c:299)
==11990==    by 0x4005E1: foo (ex2.c:6)
==11990==    by 0x4006BE: main (ex2.c:22)
==11990==
a[10] = 89
==11990==
==11990== HEAP SUMMARY:
==11990==     in use at exit: 0 bytes in 0 blocks
==11990==   total heap usage: 2 allocs, 2 frees, 1,064 bytes allocated
==11990==
==11990== All heap blocks were freed -- no leaks are possible
==11990==
==11990== For counts of detected and suppressed errors, rerun with: -v
==11990== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
<76|0>orenstal@pond:~/os17/tirgutl1%
```

Ex3.c

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

void foo(int n) {
	int *a = (int*) malloc(n*sizeof(int));
	int i;

	for (i=0; i<n; i++) {
		a[i] = i*i;
		printf("a[%d] = %d\n", i, a[i]);
	}
}

int main(int argc, char *argv[]) {
	foo(10);
	return 0;
}
```

Ex3.c


```
<76|0>orenstal@pond:~/os17/tirgutl1% gcc -g -Wall ex3.c -o example
<77|0>orenstal@pond:~/os17/tirgutl1% ./example
a[0] = 0
a[1] = 1
a[2] = 4
a[3] = 9
a[4] = 16
a[5] = 25
a[6] = 36
a[7] = 49
a[8] = 64
a[9] = 81
<78|0>orenstal@pond:~/os17/tirgutl1%
```

```
<78|0>orenstal@pond:~/os17/tirgutl1% valgrind example
==12168== Memcheck, a memory error detector
==12168== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==12168== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==12168== Command: example
==12168==
a[0] = 0
a[1] = 1
a[2] = 4
a[3] = 9
a[4] = 16
a[5] = 25
a[6] = 36
a[7] = 49
a[8] = 64
a[9] = 81
==12168==
==12168== HEAP SUMMARY:
==12168==     in use at exit: 40 bytes in 1 blocks
==12168==   total heap usage: 2 allocs, 1 frees, 1,064 bytes allocated
==12168==
==12168== LEAK SUMMARY:
==12168==    definitely lost: 40 bytes in 1 blocks
==12168==    indirectly lost: 0 bytes in 0 blocks
==12168==      possibly lost: 0 bytes in 0 blocks
==12168==    still reachable: 0 bytes in 0 blocks
==12168==         suppressed: 0 bytes in 0 blocks
==12168== Rerun with --leak-check=full to see details of leaked memory
==12168==
==12168== For counts of detected and suppressed errors, rerun with: -v
==12168== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
<79|0>orenstal@pond:~/os17/tirgutl1%
```

Ex4.c

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

#define DIM 1000000

int main(int argc, char *argv[]) {
        float *a;
        int i;

        a = (float*) malloc(DIM*sizeof(float));

        for (i=0; i<DIM; i++) {
                a[i] = i;
                if (i == DIM-1000) {                    printf("i
    == %d", (DIM-1000));        a[DIM+i] = a[i];
                        a[DIM-1] = a[i];
                }
        }
        printf("Done");
        free(a);
        return 0;
}
```

Ex4.c


```
<22|139>orenstal@river-01:~/os17/tirgutl1% gcc -g -Wall ex4.c -o example
<23|0>orenstal@river-01:~/os17/tirgutl1% ./example
Segmentation fault
<24|139>orenstal@river-01:~/os17/tirgutl1%
```

```
<24|139>orenstal@river-01:~/os17/tirgutl1% valgrind example
==56049== Memcheck, a memory error detector
==56049== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==56049== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==56049== Command: example
==56049==
==56049== Invalid write of size 4
==56049==    at 0x40060F: main (ex4.c:16)
==56049==  Address 0x59792a0 is not stack'd, malloc'd or (recently) free'd
==56049==
==56049==
==56049== Process terminating with default action of signal 11 (SIGSEGV)
==56049==  Access not within mapped region at address 0x59792A0
==56049==    at 0x40060F: main (ex4.c:16)
==56049==  If you believe this happened as a result of a stack
==56049==  overflow in your program's main thread (unlikely but
==56049==  possible), you can try to increase the size of the
==56049==  main thread stack using the --main-stacksize= flag.
==56049==  The main thread stack size used in this run was 8388608.
i == 999000==56049==
==56049== HEAP SUMMARY:
==56049==     in use at exit: 4,000,000 bytes in 1 blocks
==56049==   total heap usage: 2 allocs, 1 frees, 4,001,024 bytes allocated
==56049==
==56049== LEAK SUMMARY:
==56049==    definitely lost: 0 bytes in 0 blocks
==56049==    indirectly lost: 0 bytes in 0 blocks
==56049==      possibly lost: 0 bytes in 0 blocks
==56049==    still reachable: 4,000,000 bytes in 1 blocks
==56049==         suppressed: 0 bytes in 0 blocks
==56049== Rerun with --leak-check=full to see details of leaked memory
==56049==
==56049== For counts of detected and suppressed errors, rerun with: -v
==56049== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Segmentation fault
<25|139>orenstal@river-01:~/os17/tirgutl1%
```

Ex5.c

```
#include<stdlib.h>
#include<stdio.h>
#define DIM 1000000

int main(int argc, char *argv[]) {
        float *a;
        int i;
        a = (float*) malloc(DIM*sizeof(float));
        printf("~1 ");

        for (i=0; i<DIM; i++) {
                a[i] = i;
                if (i == DIM-1000) {                    printf("i
    == %d", (DIM-1000));            a[DIM+i] = a[i];
                        a[DIM-1] = a[i];                printf("
    ~2 ");
                }
        }

        printf("a[0] = %f\n", a[0]);
        printf("Done");
        free(a);
        return 0;
}
```

Ex5.c


```
<82|139>orenstal@pond:~/os17/tirgutl1% gcc -g -Wall ex5.c -o example
<83|0>orenstal@pond:~/os17/tirgutl1% ./example
Segmentation fault
<84|139>orenstal@pond:~/os17/tirgutl1%
```

```
<29|139>orenstal@river-01:~/os17/tirgutl1% valgrind example
==56315== Memcheck, a memory error detector
==56315== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==56315== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==56315== Command: example
==56315==
==56315== Invalid write of size 4
==56315==    at 0x400622: main (ex5.c:16)
==56315==  Address 0x59792a0 is not stack'd, malloc'd or (recently) free'd
==56315==
==56315==
==56315== Process terminating with default action of signal 11 (SIGSEGV)
==56315==  Access not within mapped region at address 0x59792A0
==56315==    at 0x400622: main (ex5.c:16)
==56315==  If you believe this happened as a result of a stack
==56315==  overflow in your program's main thread (unlikely but
==56315==  possible), you can try to increase the size of the
==56315==  main thread stack using the --main-stacksize= flag.
==56315==  The main thread stack size used in this run was 8388608.
~1 i == 999000==56315==
==56315== HEAP SUMMARY:
==56315==     in use at exit: 4,000,000 bytes in 1 blocks
==56315==   total heap usage: 2 allocs, 1 frees, 4,001,024 bytes allocated
==56315==
==56315== LEAK SUMMARY:
==56315==    definitely lost: 0 bytes in 0 blocks
==56315==    indirectly lost: 0 bytes in 0 blocks
==56315==      possibly lost: 0 bytes in 0 blocks
==56315==    still reachable: 4,000,000 bytes in 1 blocks
==56315==         suppressed: 0 bytes in 0 blocks
==56315== Rerun with --leak-check=full to see details of leaked memory
==56315==
==56315== For counts of detected and suppressed errors, rerun with: -v
==56315== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Segmentation fault
<30|139>orenstal@river-01:~/os17/tirgutl1%
```

Ex6.c

```
#include<stdlib.h>
#include<stdio.h>
#define DIM 1000000

int main(int argc, char *argv[]) {
        float *a;
        int i;

        a = (float*) malloc(DIM*sizeof(float));
        printf("~1 ");      fflush(stdout);

        for (i=0; i<DIM; i++) {
                a[i] = i;
                if (i == DIM-1000) {                                    printf("i
    == %d", (DIM-1000));                fflush(stdout);
                a[DIM+i] = a[i];                        a[DIM-1] = a[i];
                        printf(" ~2 ");     fflush(stdout);
                }
        }

        printf("a[0] = %f\n", a[0]);  fflush(stdout);
        printf("Done");
        free(a);
        return 0;
}
```

Ex6.c


```
<32|139>orenstal@river-01:~/os17/tirgutl1% gcc -g -Wall ex6.c -o example
<33|0>orenstal@river-01:~/os17/tirgutl1% ./example
~1 i == 999000Segmentation fault
<34|139>orenstal@river-01:~/os17/tirgutl1%
```

```
<34|139>orenstal@river-01:~/os17/tirgutl1% valgrind example
==56517== Memcheck, a memory error detector
==56517== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==56517== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==56517== Command: example
==56517==
~1 i == 999000==56517== Invalid write of size 4
==56517==    at 0x4006D0: main (ex6.c:18)
==56517==  Address 0x59792a0 is not stack'd, malloc'd or (recently) free'd
==56517==
==56517==
==56517== Process terminating with default action of signal 11 (SIGSEGV)
==56517==  Access not within mapped region at address 0x59792A0
==56517==    at 0x4006D0: main (ex6.c:18)
==56517==  If you believe this happened as a result of a stack
==56517==  overflow in your program's main thread (unlikely but
==56517==  possible), you can try to increase the size of the
==56517==  main thread stack using the --main-stacksize= flag.
==56517==  The main thread stack size used in this run was 8388608.
==56517==
==56517== HEAP SUMMARY:
==56517==     in use at exit: 4,000,000 bytes in 1 blocks
==56517==   total heap usage: 2 allocs, 1 frees, 4,001,024 bytes allocated
==56517==
==56517== LEAK SUMMARY:
==56517==    definitely lost: 0 bytes in 0 blocks
==56517==    indirectly lost: 0 bytes in 0 blocks
==56517==      possibly lost: 0 bytes in 0 blocks
==56517==    still reachable: 4,000,000 bytes in 1 blocks
==56517==         suppressed: 0 bytes in 0 blocks
==56517== Rerun with --leak-check=full to see details of leaked memory
==56517==
==56517== For counts of detected and suppressed errors, rerun with: -v
==56517== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Segmentation fault
<35|139>orenstal@river-01:~/os17/tirgutl1%
```

Ex7.c

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

#define DIM 1000000

int main(int argc, char *argv[]) {
        float *a;
        int i;

        a = (float*) malloc(DIM*sizeof(float));

        for (i=0; i<DIM; i++) {
                a[i] = i;
                if (i == DIM-1000) {                    printf("i
    == %d\n", (DIM-1000));          a[DIM+i] = a[i];
                                a[DIM-1] = a[i];
                }
        }
        printf("Done\n");
        free(a);
        return 0;
}
```

Ex7.c


```
<36|0>orenstal@river-01:~/os17/tirgutl1% gcc -g -Wall ex7.c -o example
<37|0>orenstal@river-01:~/os17/tirgutl1% ./example
i == 999000
Segmentation fault
<38|139>orenstal@river-01:~/os17/tirgutl1%
```

```
<38|139>orenstal@river-01:~/os17/tirgutl1% valgrind example
==56759== Memcheck, a memory error detector
==56759== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==56759== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==56759== Command: example
==56759==
i == 999000
==56759== Invalid write of size 4
==56759==    at 0x40065F: main (ex7.c:16)
==56759==  Address 0x59792a0 is not stack'd, malloc'd or (recently) free'd
==56759==
==56759==
==56759== Process terminating with default action of signal 11 (SIGSEGV)
==56759==  Access not within mapped region at address 0x59792A0
==56759==    at 0x40065F: main (ex7.c:16)
==56759==  If you believe this happened as a result of a stack
==56759==  overflow in your program's main thread (unlikely but
==56759==  possible), you can try to increase the size of the
==56759==  main thread stack using the --main-stacksize= flag.
==56759==  The main thread stack size used in this run was 8388608.
==56759==
==56759== HEAP SUMMARY:
==56759==     in use at exit: 4,000,000 bytes in 1 blocks
==56759==   total heap usage: 2 allocs, 1 frees, 4,001,024 bytes allocated
==56759==
==56759== LEAK SUMMARY:
==56759==    definitely lost: 0 bytes in 0 blocks
==56759==    indirectly lost: 0 bytes in 0 blocks
==56759==      possibly lost: 0 bytes in 0 blocks
==56759==    still reachable: 4,000,000 bytes in 1 blocks
==56759==         suppressed: 0 bytes in 0 blocks
==56759== Rerun with --leak-check=full to see details of leaked memory
==56759==
==56759== For counts of detected and suppressed errors, rerun with: -v
==56759== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Segmentation fault
<39|139>orenstal@river-01:~/os17/tirgutl1%
```

Ex8.c

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

int main(int argc, char *argv[]) {
        int y;
        y += 1;

        printf("Done\n");
        return 0;
}
```

Ex8.c


```
<91|0>orenstal@pond:~/os17/tirgutl1% gcc -g -Wall ex8.c -o example
ex8.c: In function 'main':
ex8.c:5:4: warning: 'y' is used uninitialized in this function [-Wuninitialized]
  y += 1;
  ~~^~~~
<92|0>orenstal@pond:~/os17/tirgutl1%
<92|0>orenstal@pond:~/os17/tirgutl1% gcc -g ex8.c -o example
<93|0>orenstal@pond:~/os17/tirgutl1% ./example
Done
<94|0>orenstal@pond:~/os17/tirgutl1%
<94|0>orenstal@pond:~/os17/tirgutl1%
<94|0>orenstal@pond:~/os17/tirgutl1% valgrind example
==12788== Memcheck, a memory error detector
==12788== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==12788== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==12788== Command: example
==12788==
Done
==12788==
==12788== HEAP SUMMARY:
==12788==     in use at exit: 0 bytes in 0 blocks
==12788==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==12788==
==12788== All heap blocks were freed -- no leaks are possible
==12788==
==12788== For counts of detected and suppressed errors, rerun with: -v
==12788== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
<95|0>orenstal@pond:~/os17/tirgutl1%
```

Ex9.c

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

int main(int argc, char *argv[]) {
	int arrLen = 10;
	int* arr;
	int i;
	arr = (int*) malloc(arrLen*sizeof(int));

	for(i=0; i<arrLen; i++) {
		arr[i] = i;
	}
	free(arr);

	for(i=0; i<arrLen; i++) {
		arr[i] = i*2;
	}

	printf("Done\n");
	return 0;
}
```

```
<95|0>orenstal@pond:~/os17/tirgutl1% gcc -g -Wall ex9.c -o example
<96|0>orenstal@pond:~/os17/tirgutl1% ./example
Segmentation fault
<97|139>orenstal@pond:~/os17/tirgutl1%
<97|139>orenstal@pond:~/os17/tirgutl1%
<97|139>orenstal@pond:~/os17/tirgutl1%
<97|139>orenstal@pond:~/os17/tirgutl1% valgrind example
==12883== Memcheck, a memory error detector
==12883== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==12883== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==12883== Command: example
==12883==
==12883== Invalid write of size 4
==12883==    at 0x4005FD: main (ex9.c:18)
==12883==  Address 0x51d9040 is 0 bytes inside a block of size 40 free'd
==12883==    at 0x4C2CDFB: free (vg_replace_malloc.c:530)
==12883==    by 0x4005DA: main (ex9.c:15)
==12883==  Block was alloc'd at
==12883==    at 0x4C2BBCF: malloc (vg_replace_malloc.c:299)
==12883==    by 0x40059C: main (ex9.c:9)
==12883==
Done
==12883==
==12883== HEAP SUMMARY:
==12883==     in use at exit: 0 bytes in 0 blocks
==12883==   total heap usage: 2 allocs, 2 frees, 1,064 bytes allocated
==12883==
==12883== All heap blocks were freed -- no leaks are possible
==12883==
==12883== For counts of detected and suppressed errors, rerun with: -v
==12883== ERROR SUMMARY: 10 errors from 1 contexts (suppressed: 0 from 0)
<98|0>orenstal@pond:~/os17/tirgutl1%
```

Ex10.c

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

int compute(int len, int* arr) {
        int sum = 0, i;
        for(i=0; i<len; i++) {
                sum += arr[i];
        }
        free(arr);
        return sum;
}

int main(int argc, char *argv[]) {
        int arrLen = 10, sum, i;
        int* arr;
        arr = (int*) malloc(arrLen*sizeof(int));

        for(i=0; i<arrLen; i++) {
                arr[i] = i;
        }
        sum = compute(arrLen, arr);
        free(arr);
        printf("sum = %d", sum);
        return 0;
}
```

Ex10.c


```
<98|0>orenstal@pond:~/os17/tirgutl1% gcc -g -Wall ex10.c -o example
<99|0>orenstal@pond:~/os17/tirgutl1% ./example
*** Error in `./example': double free or corruption (fasttop): 0x0000000001092010 ***
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x6ef45)[0x7fa1991f0f45]
/lib/x86_64-linux-gnu/libc.so.6(+0x746b6)[0x7fa1991f66b6]
/lib/x86_64-linux-gnu/libc.so.6(+0x74e9e)[0x7fa1991f6e9e]
./example[0x400644]
/lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xf0)[0x7fa1991a2730]
./example[0x4004a9]
======= Memory map: ========
00400000-00401000 r-xp 00000000 00:33 1887333283                         /cs/grad/orenstal/os17/tirgutl1/example
00600000-00601000 rw-p 00000000 00:33 1887333283                         /cs/grad/orenstal/os17/tirgutl1/example
01092000-010b3000 rw-p 00000000 00:00 0                                  [heap]
7fa194000000-7fa194021000 rw-p 00000000 00:00 0
7fa194021000-7fa198000000 ---p 00000000 00:00 0
7fa198f6c000-7fa198f82000 r-xp 00000000 00:0e 1945945728                 /lib/x86_64-linux-gnu/libgcc_s.so.1
7fa198f82000-7fa199181000 ---p 00016000 00:0e 1945945728                 /lib/x86_64-linux-gnu/libgcc_s.so.1
7fa199181000-7fa199182000 rw-p 00015000 00:0e 1945945728                 /lib/x86_64-linux-gnu/libgcc_s.so.1
7fa199182000-7fa199319000 r-xp 00000000 00:0e 1958619660                 /lib/x86_64-linux-gnu/libc-2.23.so
7fa199319000-7fa199519000 ---p 00197000 00:0e 1958619660                 /lib/x86_64-linux-gnu/libc-2.23.so
7fa199519000-7fa19951d000 r--p 00197000 00:0e 1958619660                 /lib/x86_64-linux-gnu/libc-2.23.so
7fa19951d000-7fa19951f000 rw-p 0019b000 00:0e 1958619660                 /lib/x86_64-linux-gnu/libc-2.23.so
7fa19951f000-7fa199523000 rw-p 00000000 00:00 0
7fa199523000-7fa199547000 r-xp 00000000 00:0e 1907667676                 /lib/x86_64-linux-gnu/ld-2.23.so
7fa1996e8000-7fa1996eb000 rw-p 00000000 00:00 0
7fa199743000-7fa199746000 rw-p 00000000 00:00 0
7fa199746000-7fa199747000 r--p 00023000 00:0e 1907667676                 /lib/x86_64-linux-gnu/ld-2.23.so
7fa199747000-7fa199748000 rw-p 00024000 00:0e 1907667676                 /lib/x86_64-linux-gnu/ld-2.23.so
7fa199748000-7fa199749000 rw-p 00000000 00:00 0
7ffe1e759000-7ffe1e77a000 rw-p 00000000 00:00 0                          [stack]
7ffe1e7ba000-7ffe1e7bc000 r--p 00000000 00:00 0                          [vvar]
7ffe1e7bc000-7ffe1e7be000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
Abort
<100|134>orenstal@pond:~/os17/tirgutl1%
```

Ex10.c


```
<100|134>orenstal@pond:~/os17/tirgutl1% valgrind example
==13036== Memcheck, a memory error detector
==13036== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==13036== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==13036== Command: example
==13036==
==13036== Invalid free() / delete / delete[] / realloc()
==13036==    at 0x4C2CDFB: free (vg_replace_malloc.c:530)
==13036==    by 0x400643: main (ex10.c:28)
==13036==  Address 0x51d9040 is 0 bytes inside a block of size 40 free'd
==13036==    at 0x4C2CDFB: free (vg_replace_malloc.c:530)
==13036==    by 0x4005C5: compute (ex10.c:11)
==13036==    by 0x400634: main (ex10.c:27)
==13036==  Block was alloc'd at
==13036==    at 0x4C2BBCF: malloc (vg_replace_malloc.c:299)
==13036==    by 0x4005F1: main (ex10.c:21)
==13036==
sum = 45==13036==
==13036== HEAP SUMMARY:
==13036==     in use at exit: 0 bytes in 0 blocks
==13036==   total heap usage: 2 allocs, 3 frees, 1,064 bytes allocated
==13036==
==13036== All heap blocks were freed -- no leaks are possible
==13036==
==13036== For counts of detected and suppressed errors, rerun with: -v
==13036== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
<101|0>orenstal@pond:~/os17/tirgutl1%
```

Valgrind Summary

- Valgrind can detect errors such as:
  - Memory leaks.
  - Wrong usage of memory-related functions (malloc, free, etc.).
  - Out of bounds error.
  - It may supply segmentation fault hint.
- Use it!

GDB (GNU Debugger)

GDB (GNU Debugger)

- Debugger that runs on many Unix-like systems and for many programming languages.
- Compile c / c++ programs with '-g' flag (enables debug symbols)
  - gcc –o myProg –g test.c
  - g++ -o myProg –g test.cpp
- Then, load the program into gdb
  - gdb myProg

GDB – Basic Commands

- run: starts executing the program.
- break: insert breakpoint in which the execution will suspend.
- next: executes the next line (even if it's function call), unless there is a suspending execution event (later today)
- step: steps into the next line.
- cont : continues the execution till the next suspending execution event or end of execution.
- print *var*: prints the value of *var*.

Ex1.c Reminder

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

typedef struct Foo {
        int arr[3];
        int bar;
} Foo;

int main(int argc, char *argv[]) {
        Foo t;
        int i;
        printf("start\n");
        t.bar = 6;
        printf("t.bar =  %d\n", t.bar);
        for (i=0; i<3; i++) {
                t.arr[i] = i+1;
        }
        t.arr[i] += t.arr[0] + t.arr[1];
        printf("t.arr[2] = %d\n", t.arr[2]);
        printf("t.bar = %d\n", t.bar);
        return 0;
}
```

Ex1.c


```
<70|0>orenstal@pond:~/os17/tirgutl1% gcc -g -Wall ex1.c -o example
<71|0>orenstal@pond:~/os17/tirgutl1% ./example
start
t.bar = 6
t.arr[2] = 3
t.bar = 9
<72|0>orenstal@pond:~/os17/tirgutl1%
<72|0>orenstal@pond:~/os17/tirgutl1% valgrind example
==11813== Memcheck, a memory error detector
==11813== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==11813== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==11813== Command: example
==11813==
start
t.bar = 6
t.arr[2] = 3
t.bar = 9
==11813==
==11813== HEAP SUMMARY:
==11813==     in use at exit: 0 bytes in 0 blocks
==11813==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==11813==
==11813== All heap blocks were freed -- no leaks are possible
==11813==
==11813== For counts of detected and suppressed errors, rerun with: -v
==11813== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
<73|0>orenstal@pond:~/os17/tirgutl1%
```

GDB (GNU Debugger)

- Why does t.bar print wrong value?
  - Let's debug it…

```
<104|0>orenstal@pond:~/os17/tirgutl1% gdb example
GNU gdb (Debian 7.11.1-2) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from example...done.
(gdb) break main
Breakpoint 1 at 0x400545: file ex1.c, line 13.
(gdb) run
Starting program: /cs/grad/orenstal/os17/tirgutl1/example

Breakpoint 1, main (argc=1, argv=0x7fffffffe4d8) at ex1.c:13
13              printf("start\n");
(gdb) next
start
14              t.bar = 6;
(gdb) print t.bar
$1 = 0
(gdb) next
15              printf("t.bar = %d\n", t.bar);
(gdb) print t.bar
$2 = 6
(gdb) cont
Continuing.
t.bar = 6
t.arr[2] = 3
t.bar = 9
[Inferior 1 (process 13267) exited normally]
(gdb)
```

GDB (GNU Debugger)

- We still didn't find the root cause..

- Break points:
  - break *function* (break *main*)
  - break <*line_num*> (break *4*)
  - break *filename:function* (break *temp.c:main*)
  - break *filename: <line_num>* (break *temp.c:2*)
  - break **address* (break **0x12345*)
  - Etc.

GDB (GNU Debugger)

- watch command:
  insert watch point that suspend the program execution upon watched point has changed.

- Watch points:
  - watch *var* (watch *arrSize*)
  - watch *condition* (watch *i==4*)
  - watch **address* (watch **0x12345*)

- But how watch can help us debugging our problem?

```
Type "apropos word" to search for commands related to "word"...
Reading symbols from example...done.
(gdb) break main
Breakpoint 1 at 0x400545: file ex1.c, line 13.
(gdb) run
Starting program: /cs/grad/orenstal/os17/tirgutl1/example

Breakpoint 1, main (argc=1, argv=0x7fffffffe4d8) at ex1.c:13
13              printf("start\n");
(gdb) next
start
14              t.bar = 6;
(gdb) next
15              printf("t.bar = %d\n", t.bar);
(gdb) watch t.bar
Hardware watchpoint 2: t.bar
(gdb) cont
Continuing.
t.bar = 6

Hardware watchpoint 2: t.bar

Old value = 6
New value = 9
main (argc=1, argv=0x7fffffffe4d8) at ex1.c:22
22              printf("t.arr[2] = %d\n", t.arr[2]);
(gdb) cont
Continuing.
t.arr[2] = 3
t.bar = 9

Watchpoint 2 deleted because the program has left the block in
which its expression is valid.
__libc_start_main (main=0x400536 <main>, argc=1, argv=0x7fffffffe4d8, init=<optimized out>, f
    at ../csu/libc-start.c:325
325     ../csu/libc-start.c: No such file or directory.
(gdb) cont
Continuing.
[Inferior 1 (process 13605) exited normally]
(gdb)
```

GDB (GNU Debugger)

- Alternative way to catch it:
  - Print &t.bar (to get the address)
  - Watch *add_of_t.bar

```
(gdb) break main
Breakpoint 1 at 0x400545: file ex1.c, line 13.
(gdb) run
Starting program: /cs/grad/orenstal/os17/tirgutl1/example

Breakpoint 1, main (argc=1, argv=0x7fffffffe4d8) at ex1.c:13
13              printf("start\n");
(gdb) next
start
14              t.bar = 6;
(gdb) next
15              printf("t.bar = %d\n", t.bar);
(gdb) print &t.bar
$1 = (int *) 0x7fffffffe3dc
(gdb) watch *0x7fffffffe3dc
Hardware watchpoint 2: *0x7fffffffe3dc
(gdb) cont
Continuing.
t.bar = 6

Hardware watchpoint 2: *0x7fffffffe3dc

Old value = 6
New value = 9
main (argc=1, argv=0x7fffffffe4d8) at ex1.c:22
22              printf("t.arr[2] = %d\n", t.arr[2]);
(gdb) cont
Continuing.
t.arr[2] = 3
t.bar = 9

Hardware watchpoint 2: *0x7fffffffe3dc

Old value = 9
New value = 0
0x00007ffff7a6da94 in __run_exit_handlers (status=0, listp=0x7ffff7dd35f8 <__exit_funcs>, run_list_atexit=run_list_
35      exit.c: No such file or directory.
(gdb) cont
Continuing.
[Inferior 1 (process 14151) exited normally]
(gdb) q
<108|0>orenstal@pond:~/os17/tirgutl1%
```

GDB (GNU Debugger)

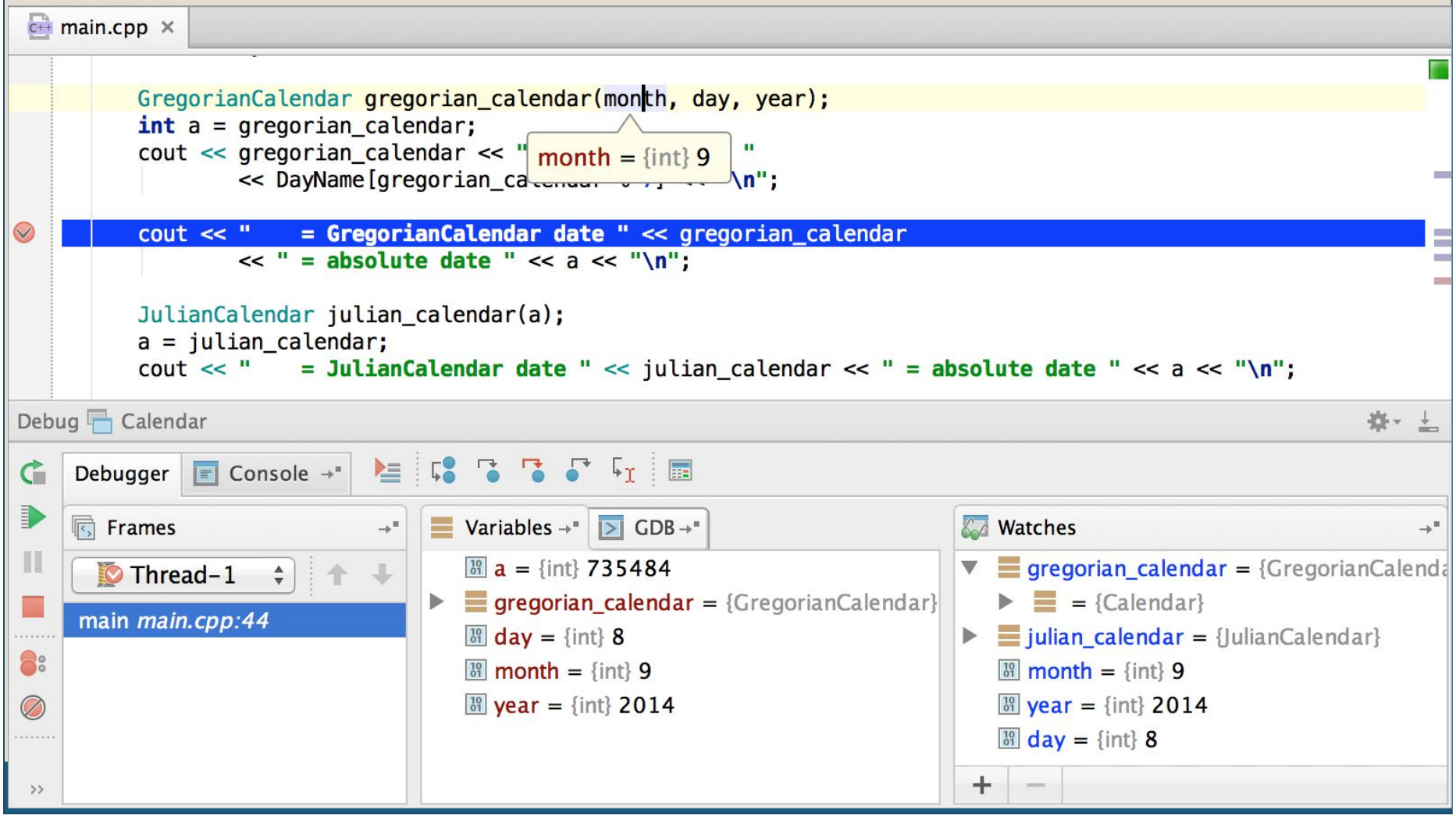
- More useful commands:
  - delete <break_num>: remove breakpoint
  - disable <watch_num>: remove watch point
  - help: help function
  - bt: prints the backtrace (in case the program crashed)
  - up: going to the calling function context (for example: to print variables values)
  - down: going back down the function stack, one function at a time.
  - info breakpoints prints a list of all the defined breakpoints and watches.

CLion

CLion

- Based on GDB.
- Has a lot of capabilities.
- Easy to use (no command line), very intuitive.

CLion

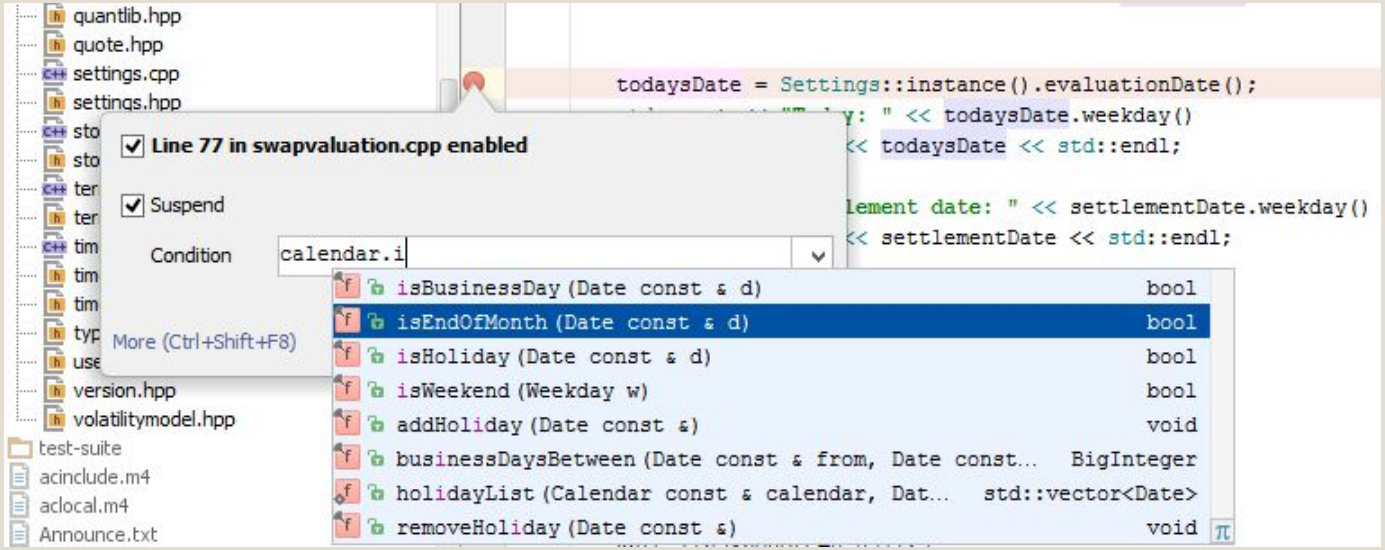

main.cpp
GregorianCalendar gregorian_calendar(month, day, year);
int a = gregorian_calendar;
cout << gregorian_calendar << "
month = {int} 9
<< DayName[gregorian_
\n";
cout << " = GregorianCalendar date " << gregorian_calendar
<< " = absolute date " << a << "\n";
JulianCalendar julian_calendar(a);
a = julian_calendar;
cout << " = JulianCalendar date " << julian_calendar << " = absolute date " << a << "\n";
Debug
Calendar
Debugger
Console
Frames
Thread-1
main main.cpp:44
Variables
GDB
a = {int} 735484
gregorian_calendar = {GregorianCalendar}
day = {int} 8
month = {int} 9
year = {int} 2014
Watches
gregorian_calendar = {GregorianCalenda
= {Calendar}
julian_calendar = {JulianCalendar}
month = {int} 9
year = {int} 2014
day = {int} 8

CLion

- Breakpoints:
  - Traditional breakpoint:

```
        if (engine_)
            registerWith(engine_);
        // trigger (lazy) recalculation
        update();
    }
```

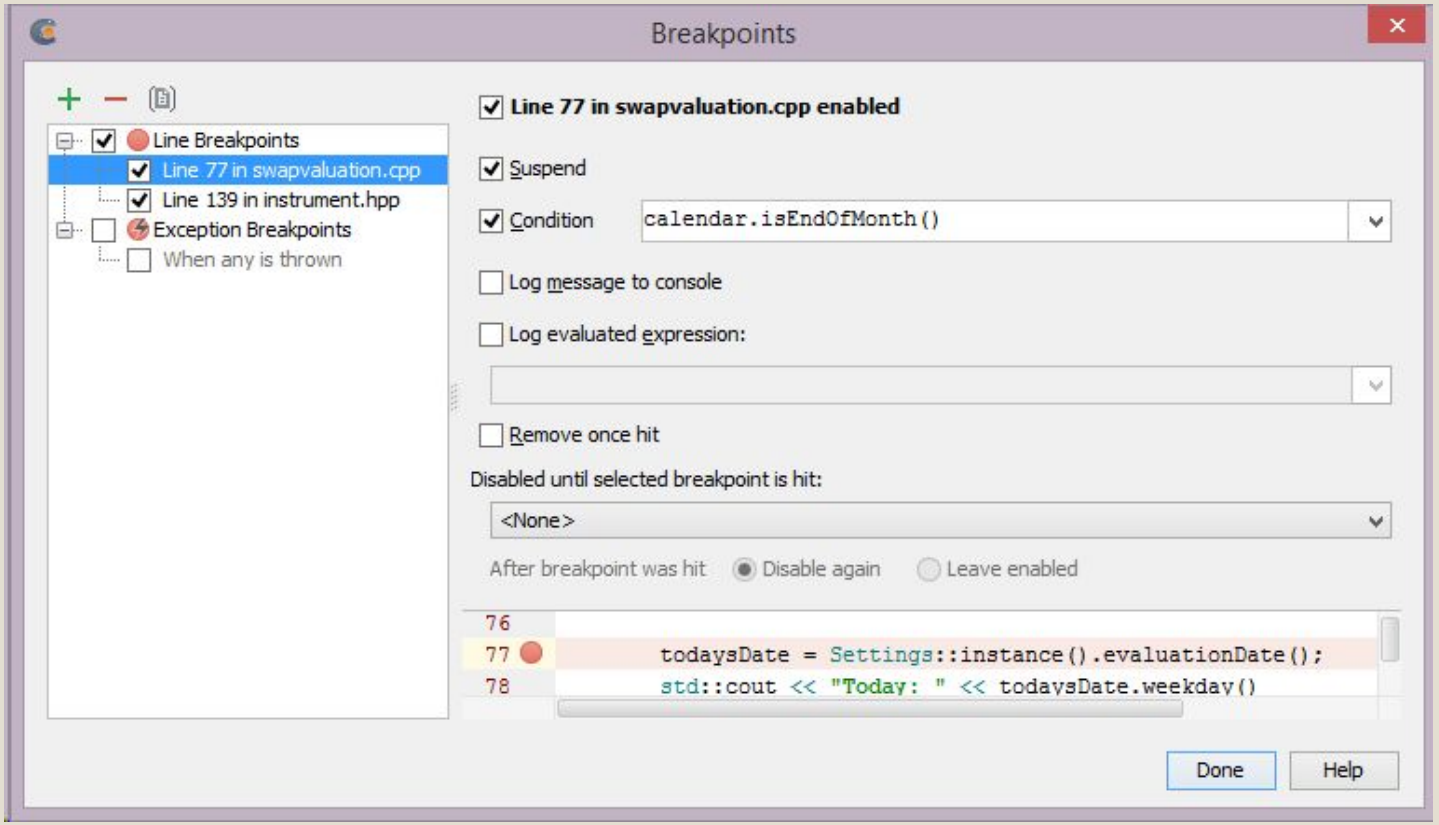
  - Conditional breakpoint:



The figures are taken from: https://blog.jetbrains.com/clion/2015/05/debug-clion/

CLion

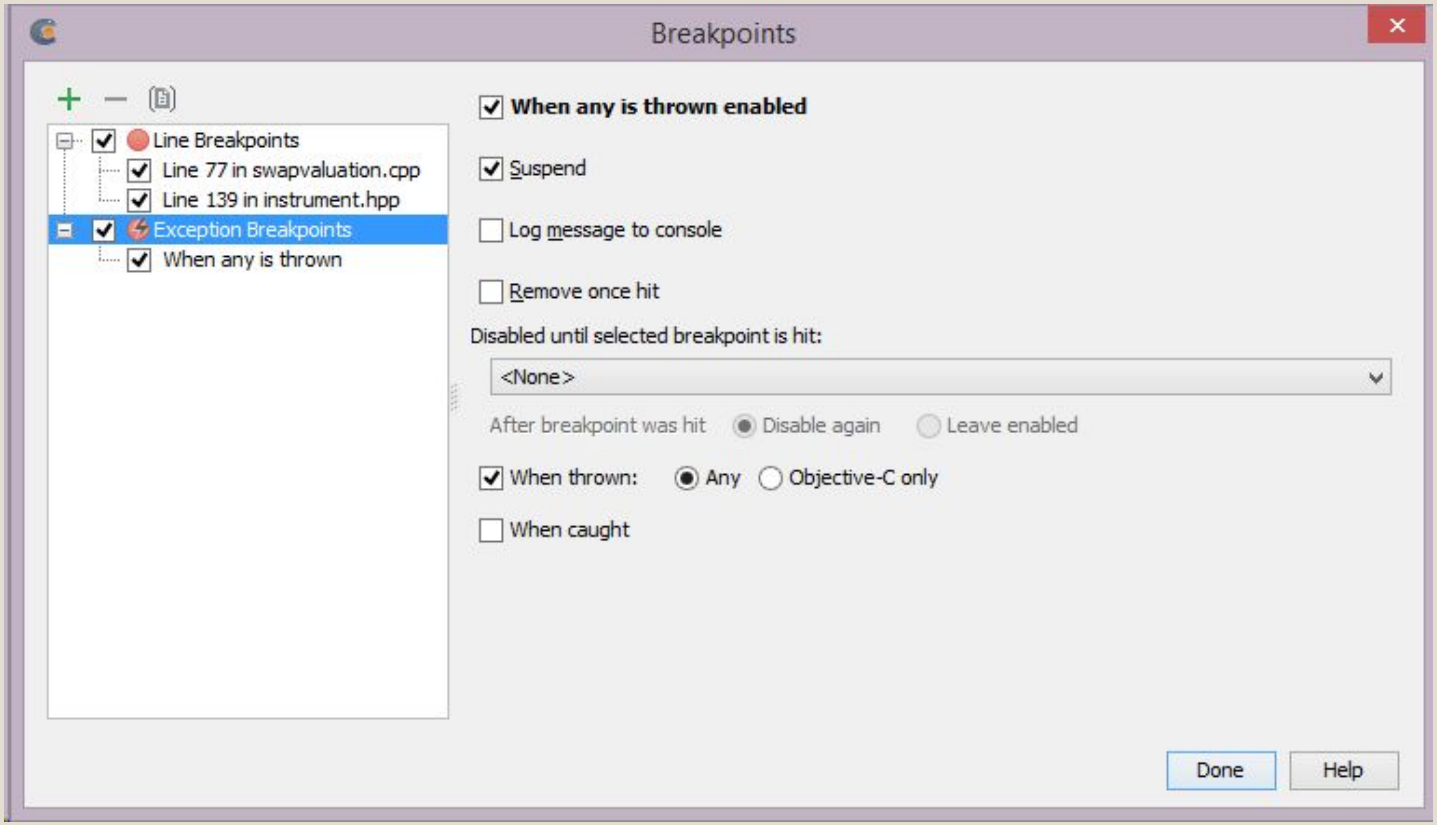
- Breakpoints management:
  - Traditional breakpoint:



The figures are taken from: https://blog.jetbrains.com/clion/2015/05/debug-clion/

CLion

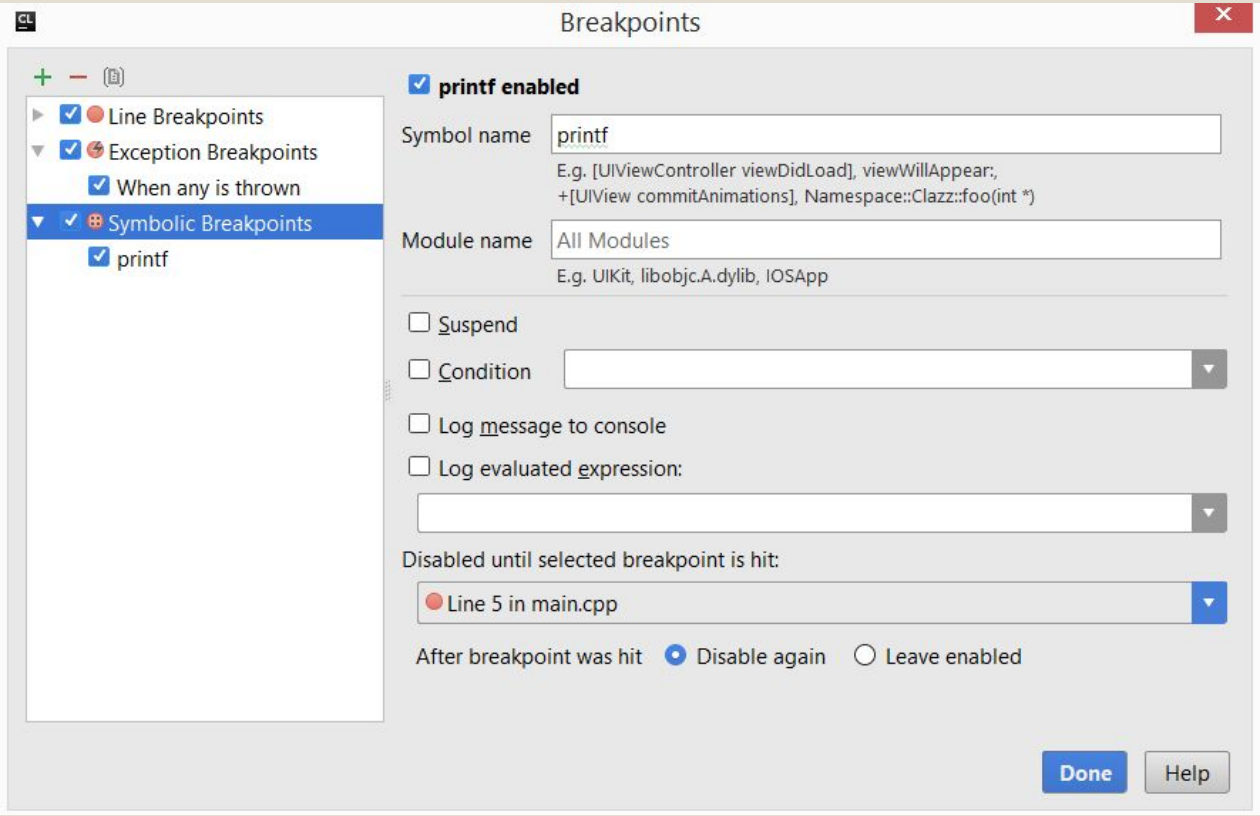
- Breakpoints:
  - Exception breakpoint:



The figures are taken from: https://blog.jetbrains.com/clion/2015/05/debug-clion/

CLion

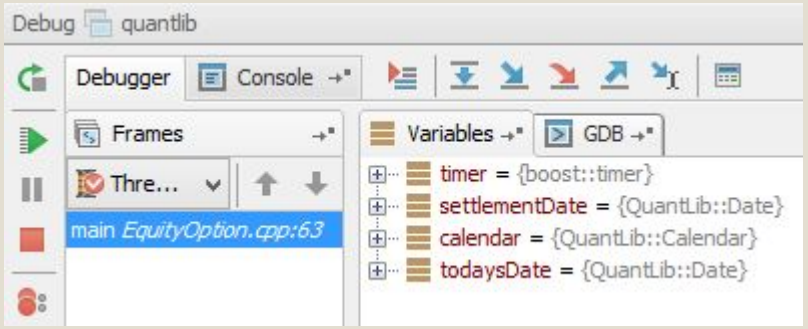
- Breakpoints:
  - Symbolic breakpoint:

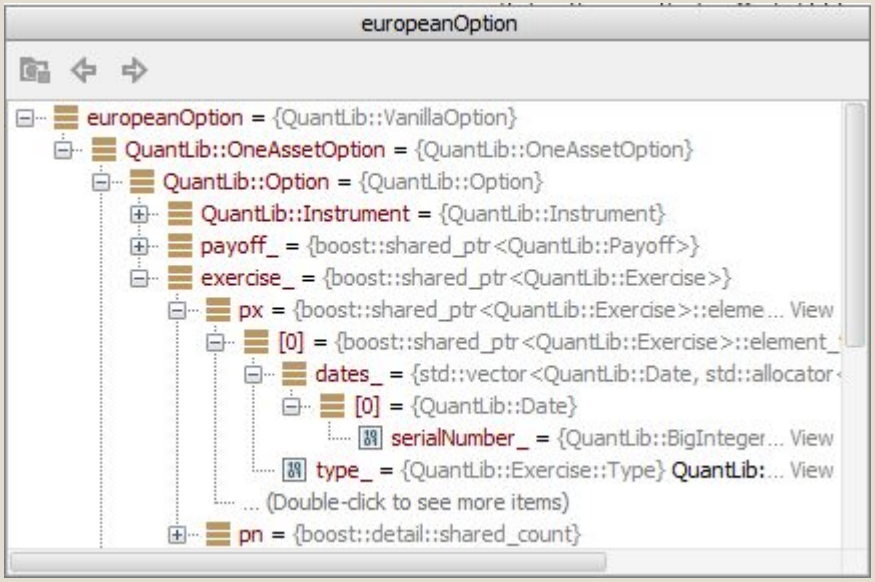


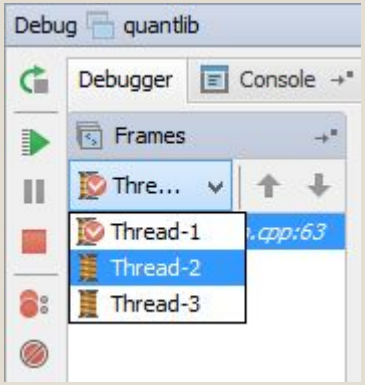
The figures are taken from: https://blog.jetbrains.com/clion/2015/05/debug-clion/

CLion

- What happens when a debugger hits a breakpoint?



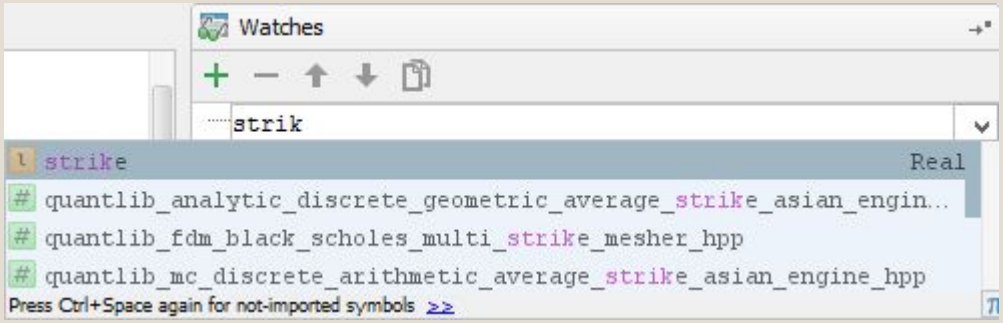


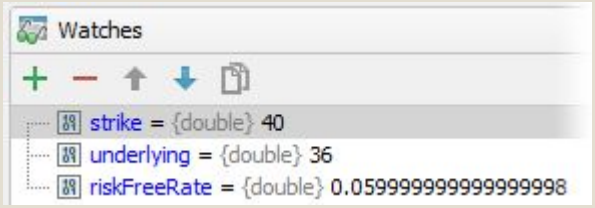


The figures are taken from: https://blog.jetbrains.com/clion/2015/05/debug-clion/

CLion

- Watches:





The figures are taken from: https://blog.jetbrains.com/clion/2015/05/debug-clion/