GDB: GNU Debugger - Useful commands

- **Breakpoint**: `break/b` *`location`*
  also possible to watch only a specific thread:
  `break/b` *`location`* `thread` *`thread-id`*
- **Watchpoint**: Stops whenever the value of an expression changes
  `watch/w` *`expression`*; e.g.: `watch foo`
- **Catchpoint**: Stop for certain kinds of programm events
  `catch` *`event`* ,e.g. where event = exception, assert, syscall, signal, ...

- `continue/c/fg`: Resuming programm execution
- `step/s`: Executing just one more step
- `ptype` *`expression`*: Print a description of the type of the expression.
- `list` *`function`*: Print lines (default 10 lines) from a source file *centered around the beginning of the function*
- `backtrace/bt`: “A backtrace is a summary of how your program got where it is.”

Well structured documentation on the details of GDB:
`https://sourceware.org/gdb/current/onlinedocs/gdb/` [gdb16]
A helpful GDB cheat sheet: `http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf` [che16].

Memory management

A very well written and detailed introduction on memory management and OS internals can be found in “Understanding the Linux Kernel” by D. Bovet and M. Cesati. Memory addressing is explained in chapter 2. The dynamic memory allocation by the kernel is illustrated in chapter 8. [BC05] `http://www.johnchukwuma.com/training/UnderstandingTheLinuxKernel3rdEdition.pdf`
A brief summary of the most important aspects of the stack and the heap is given in the following based on [Gri12].

Stack

- limit on stack size (OS-dependent)
- may clutter up your memory -> programm killed by OS!
- very fast access
- don’t have to explicitly de-allocate variables
- space is managed efficiently by CPU, memory will not become fragmented
- local variables only
- variables cannot be resized

Heap

- variables can be accessed globally
- no limit on memory size
- (relatively) slower access
- no guaranteed efficient use of space, memory may become fragmented over time as blocks of memory are allocated, then freed
- you must manage memory (you’re in charge of allocating and freeing variables)
- variables can be resized using realloc()

Listing 1: Stack

```c
#include <stdio.h>

double multiplyByTwo (double input) {
  double twice = input * 2.0;
  return twice;
}

int main (int argc, char *argv[])
{
  int age = 30;
  double salary = 12345.67;
  double myList[3] = {1.2, 2.3, 3.4};

  printf("double your salary is %.3f\n",
     ↪ multiplyByTwo(salary));

  return 0;
}
```

Listing 2: Heap

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

double *multiplyByTwo (double *input) {
  double *twice = malloc(sizeof(double));
  *twice = *input * 2.0;
  return twice;
}

int main (int argc, char *argv[])
{
  int *age = malloc(sizeof(int));
  *age = 30;
  double *salary = malloc(sizeof(double));
  *salary = 12345.67;
  double *myList = malloc(3 * sizeof(double));
  myList[0] = 1.2;
  myList[1] = 2.3;
  myList[2] = 3.4;

  double *twiceSalary = multiplyByTwo(salary);

  printf("double your salary is %.3f\n", *twiceSalary);

  free(age);
  free(salary);
  free(myList);
  free(twiceSalary);

  return 0;
}
```

Bibliography

[BC05] Daniel P Bovet and Marco Cesati. *Understanding the Linux kernel.* " O'Reilly Media, Inc.", 2005.

[che16] Debugging with gdb: the gnu source-level debugger. `http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf`, November 2016. (last accessed: 2016-10-31).

[gdb16] Debugging with gdb: the gnu source-level debugger. `https://sourceware.org/gdb/current/onlinedocs/gdb/`, November 2016. (last accessed: 2016-10-31).

[Gri12] Paul Gribble. Memory: Stack vs heap. `http://gribblelab.org/CBootcamp/7_Memory_Stack_vs_Heap.html`, August 2012. (last accessed: 2016-10-31).