# Memory Layout  (Virtual address space of a C process)

**High memory**

Stack illustrated after the call
func(72,73) called from main(),
assuming func defined by:
 func(int x, int y) {
   int a;
   int b[3];
  /* no other auto variables */

Assumes int = long = char * of
size 4 and assumes stack at high
address and descending down.

## Expanded view of the stack

**STACK**

| System |
| --- |
| env<br>argv<br>argc |
| auto variables for<br> main() |
| auto variables for<br> func() |

← mfp – frame pointer (for main)

← stack pointer
(grows downward if func()
 calls another function)

*available for
stack growth*

**SHARED MEMORY**

| malloc.o (lib*.so) |
| --- |
| printf.o (lib*.so) |

↑↓ library functions if
dynamically linked
(usual case)

*available for
heap growth*

← brk point

| Heap<br>(malloc arena) |
| --- |

**Stack**

| Offset from current frame pointer (for func()) | main() auto variables | Contents |
| --- | --- | --- |
| +12 | 73 | y |
| +8 | 72 | x |
| +4 | ra | return address |
| frame pointer → 0 | mfp | caller's frame pointer |
| points here −4 | garbage | a |
| −8 | garbage | b[2] |
| −12 | garbage | b[1] |
| −16 | garbage | b[0] |

stack pointer →
(top of stack)
points here

All auto variables and parameters
are referenced via offsets from the
frame pointer.

The frame pointer and stack pointer
are in registers (for fast access).

When funct returns, the return value
is stored in a register.  The stack pointer
is move to  the y location, the code
is jumped to the return address (ra),
and the frame pointer is set to mfp
(the stored value of the caller's frame
pointer). The caller moves the return
value to the right place.

**DATA**

| global variables |
| --- |

uninitialized data (bss)

| "...%d..." |
| --- |

initialized data

**TEXT** — compiled code (a.out)

| malloc.o (lib*.a) |
| --- |
| printf.o (lib*.a) |

↑↓ library functions if
statically linked
(not usual case)

| file.o |
| --- |
| main.o        func(72,73) |
| crt0.o (startup routine) |

← ra (return address)

**Low memory**