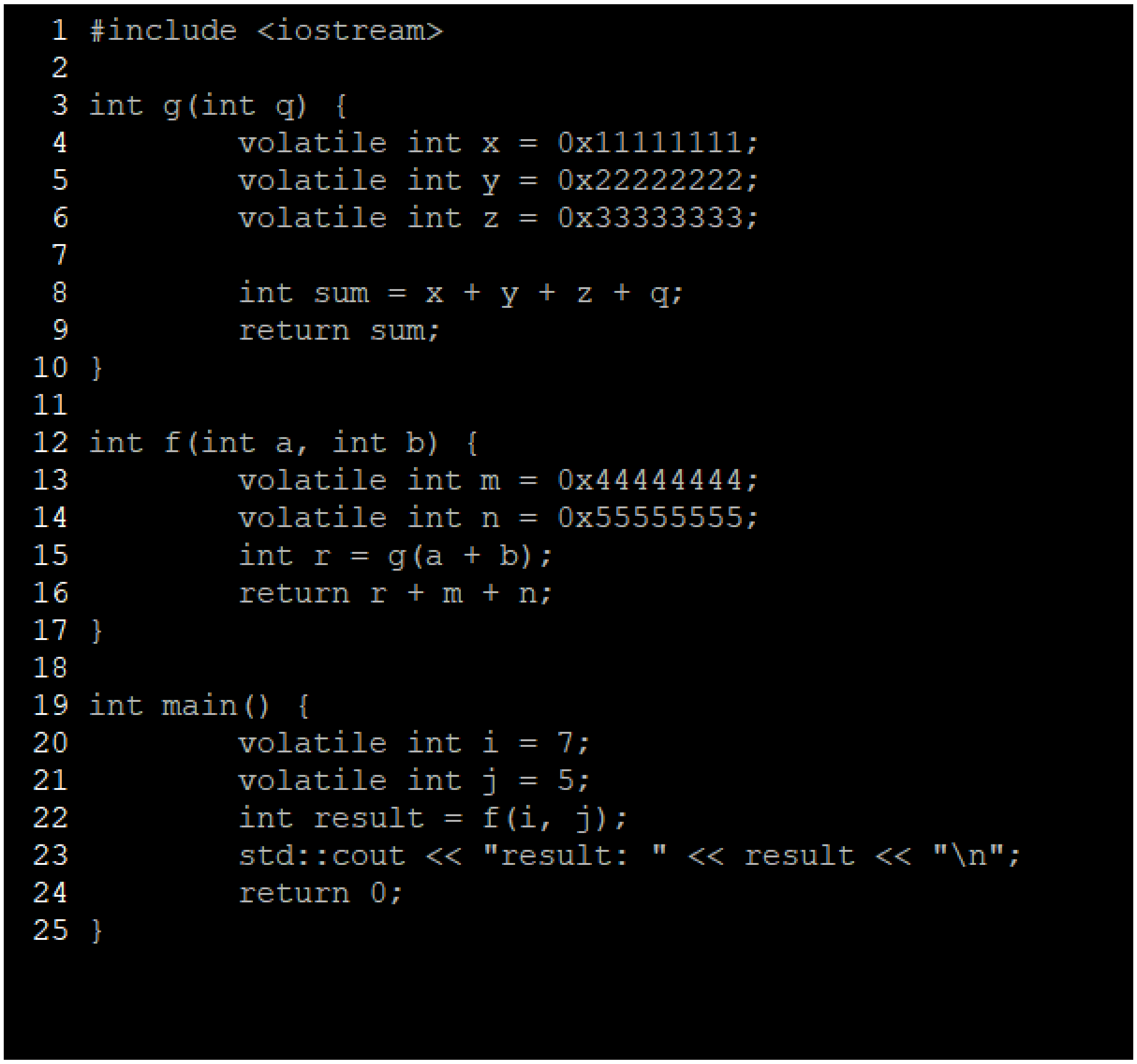
CMP SCI 3780 Project 2

Task 1:



```
#include <iostream>

int g(int q) {
        volatile int x = 0x11111111;
        volatile int y = 0x22222222;
        volatile int z = 0x33333333;

        int sum = x + y + z + q;
        return sum;
}

int f(int a, int b) {
        volatile int m = 0x44444444;
        volatile int n = 0x55555555;
        int r = g(a + b);
        return r + m + n;
}

int main() {
        volatile int i = 7;
        volatile int j = 5;
        int result = f(i, j);
        std::cout << "result: " << result << "\n";
        return 0;
}
```

Program + method. I wrote a simple C++ program with three functions that call each other: main → f → g. Each function takes at least one parameter by value and creates local variables with recognizable hex values. I compiled with debug info and no optimization (-g -O0) on delmar.umsl.edu and used gdb to pause in each function, examine the call chain, and inspect stack memory.

Evidence (screenshots I included):

- Breakpoint hit in g, bt, and info frame (shows call order and saved regs).
- g locals/addresses and memory dump near $rbp (x/2gx $rbp, x/24wx $rbp-0x30).
- Returned to f with finish; bt + info frame.
- f locals/addresses and memory dump near $rbp.
- Returned to main with finish; bt + info frame.
- main locals/addresses and memory dump near $rbp.

Key observations: stack grows downward. On x86-64 Linux, stack addresses decrease as new frames are pushed. Each activation record contains:

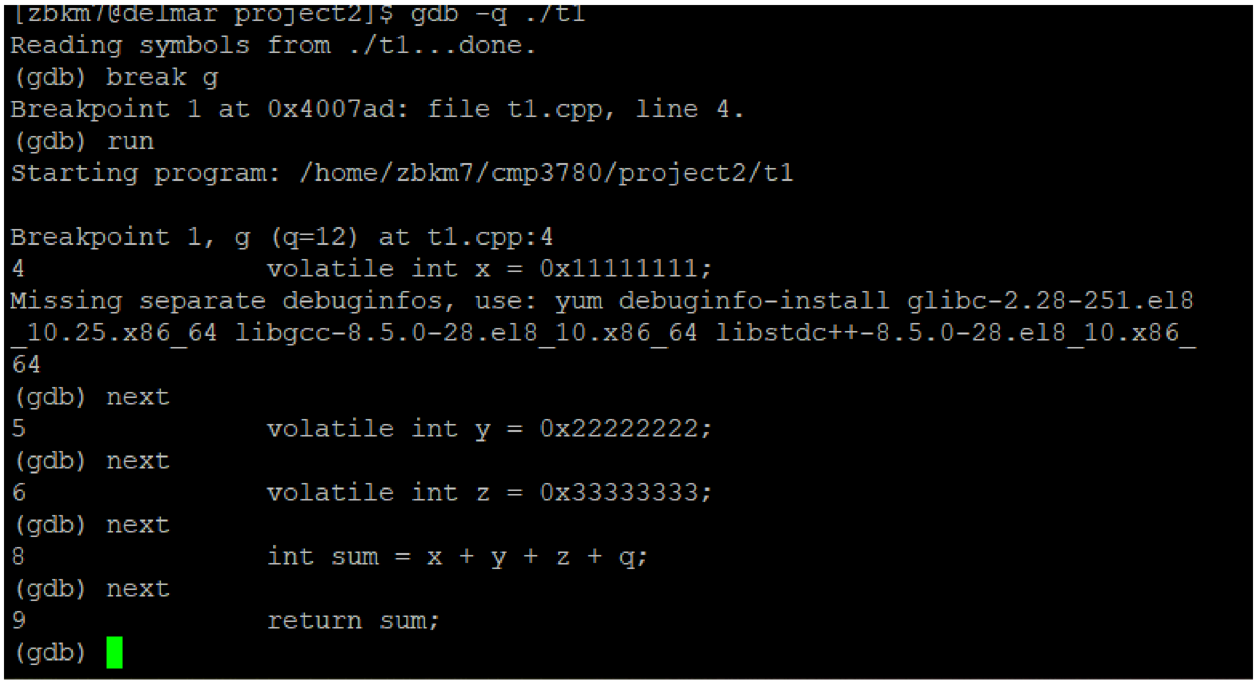
- **Non-variable data:** saved RBP (previous frame pointer) and saved RIP (return address) → 8 B + 8 B = 16 B.
- **Variable area:** the function's locals and any on-stack copies of parameters, usually placed at negative offsets from RBP (e.g., rbp-0x04, rbp-0x08, …).
- Possible padding/alignment so the stack obeys ABI rules (often 16-byte alignment).
-

Interpretation: how the memory layout reflects execution and stack organization

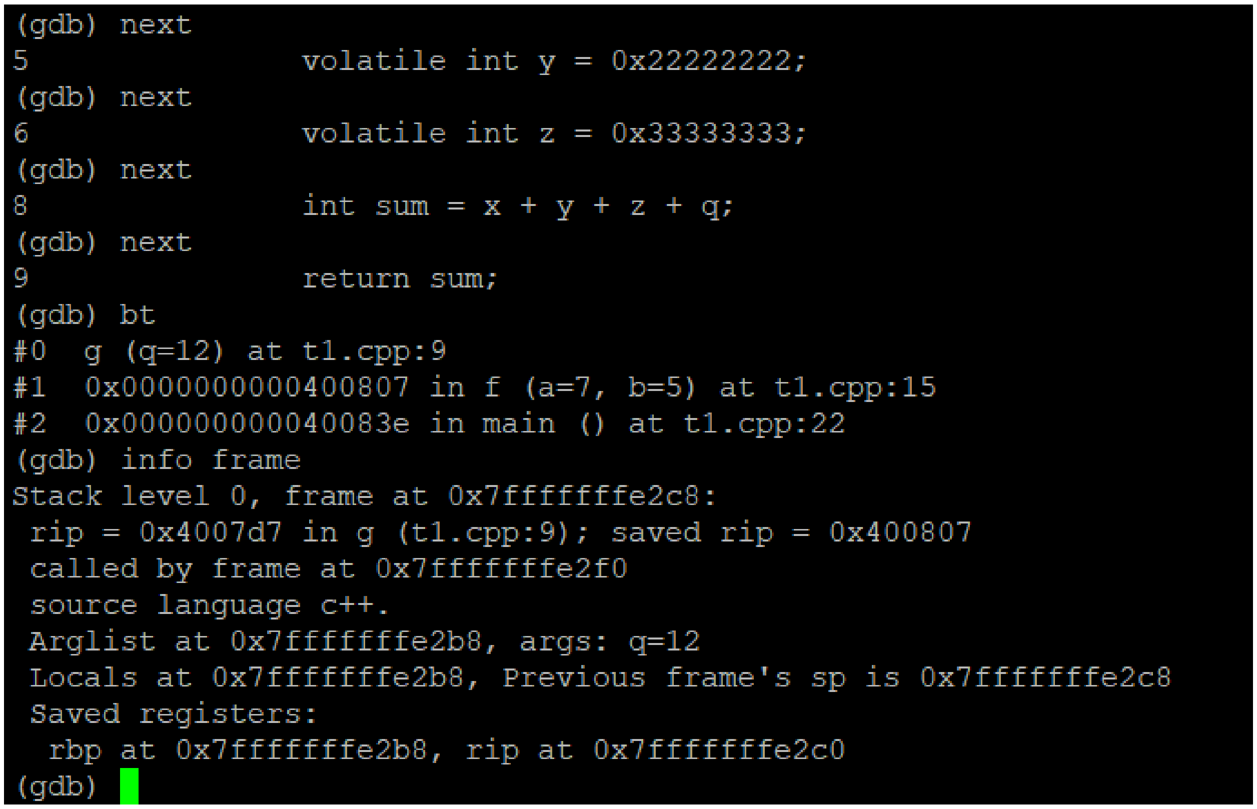
- Entering each function pushes a new activation record: saved RBP, saved RIP, then space for locals/param copies at negative offsets from RBP.
- **The stack grows downward:** addresses decrease as we move from caller to callee.
- The order of calls is visible in the backtrace and in the saved return addresses: main saved RIP points to the runtime startup; f saved RIP points back to main; g saved RIP points back to f.
- The space between frames directly shows how much additional stack was used by each call on this run: 40 B when f called g and 32 B when main called f.
- The locals appear at predictable offsets: -0x04, -0x08, -0x0c, … (4-byte ints), and our memory dumps show their recognizable hex patterns

(0x11111111, 0x22222222, 0x33333333, and the computed sum = 0x66666672).

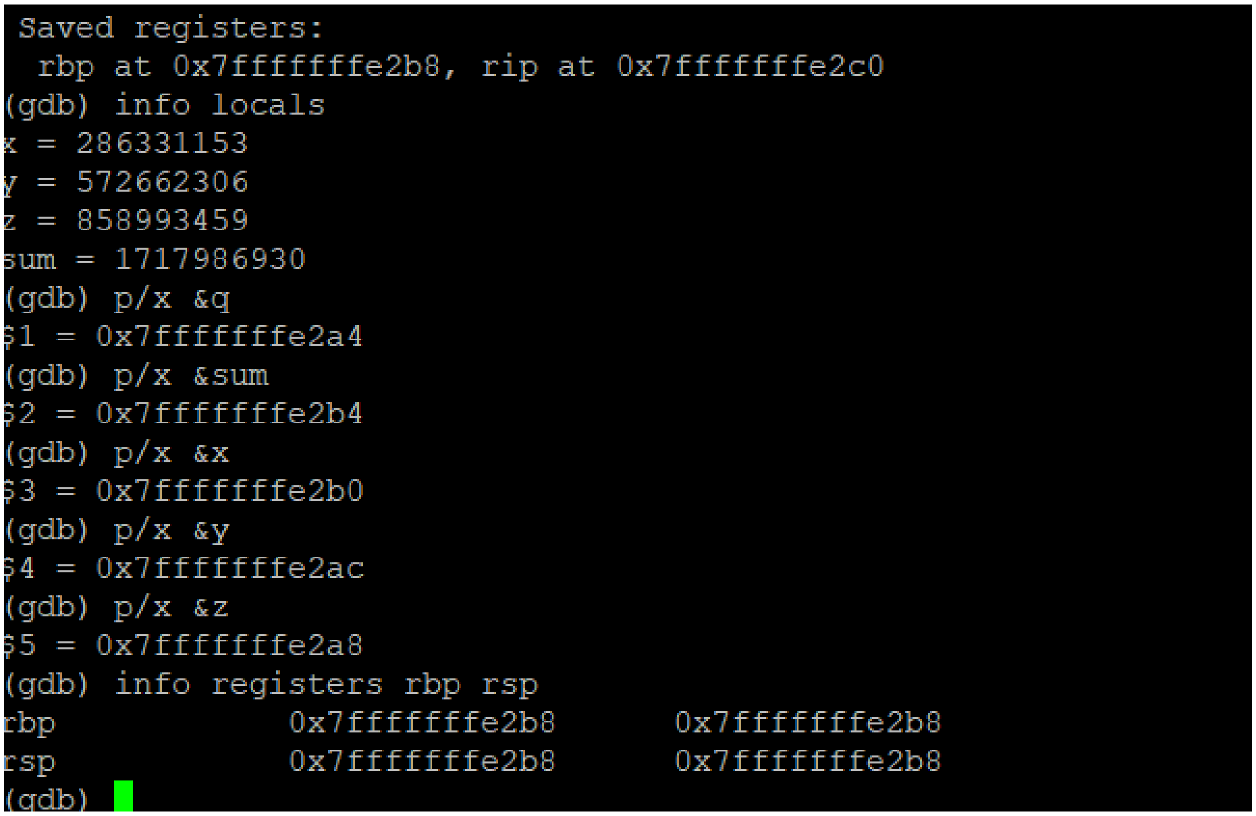
- Small differences between “locals size + 16 B” and the measured frame deltas are due to ABI alignment and compiler choices (e.g., register passing vs. creating debug stack slots). This is normal and worth noting in the write-up.



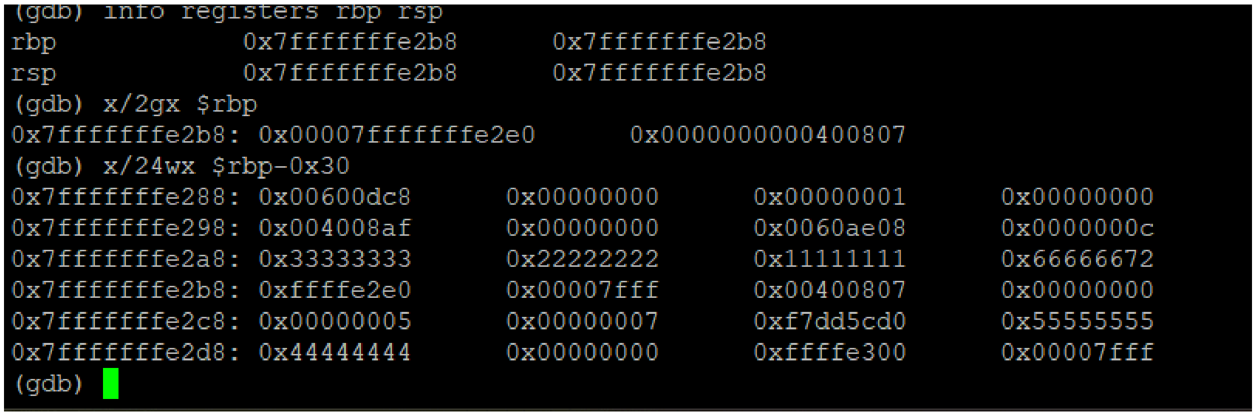
```
[zbkm7@delmar project2]$ gdb -q ./t1
Reading symbols from ./t1...done.
(gdb) break g
Breakpoint 1 at 0x4007ad: file t1.cpp, line 4.
(gdb) run
Starting program: /home/zbkm7/cmp3780/project2/t1

Breakpoint 1, g (q=12) at t1.cpp:4
4               volatile int x = 0x11111111;
Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-251.el8
_10.25.x86_64 libgcc-8.5.0-28.el8_10.x86_64 libstdc++-8.5.0-28.el8_10.x86_
64
(gdb) next
5               volatile int y = 0x22222222;
(gdb) next
6               volatile int z = 0x33333333;
(gdb) next
8               int sum = x + y + z + q;
(gdb) next
9               return sum;
(gdb)
```



```
(gdb) next
5               volatile int y = 0x22222222;
(gdb) next
6               volatile int z = 0x33333333;
(gdb) next
8               int sum = x + y + z + q;
(gdb) next
9               return sum;
(gdb) bt
#0  g (q=12) at t1.cpp:9
#1  0x0000000000400807 in f (a=7, b=5) at t1.cpp:15
#2  0x000000000040083e in main () at t1.cpp:22
(gdb) info frame
Stack level 0, frame at 0x7fffffffe2c8:
 rip = 0x4007d7 in g (t1.cpp:9); saved rip = 0x400807
 called by frame at 0x7fffffffe2f0
 source language c++.
 Arglist at 0x7fffffffe2b8, args: q=12
 Locals at 0x7fffffffe2b8, Previous frame's sp is 0x7fffffffe2c8
 Saved registers:
  rbp at 0x7fffffffe2b8, rip at 0x7fffffffe2c0
(gdb)
```

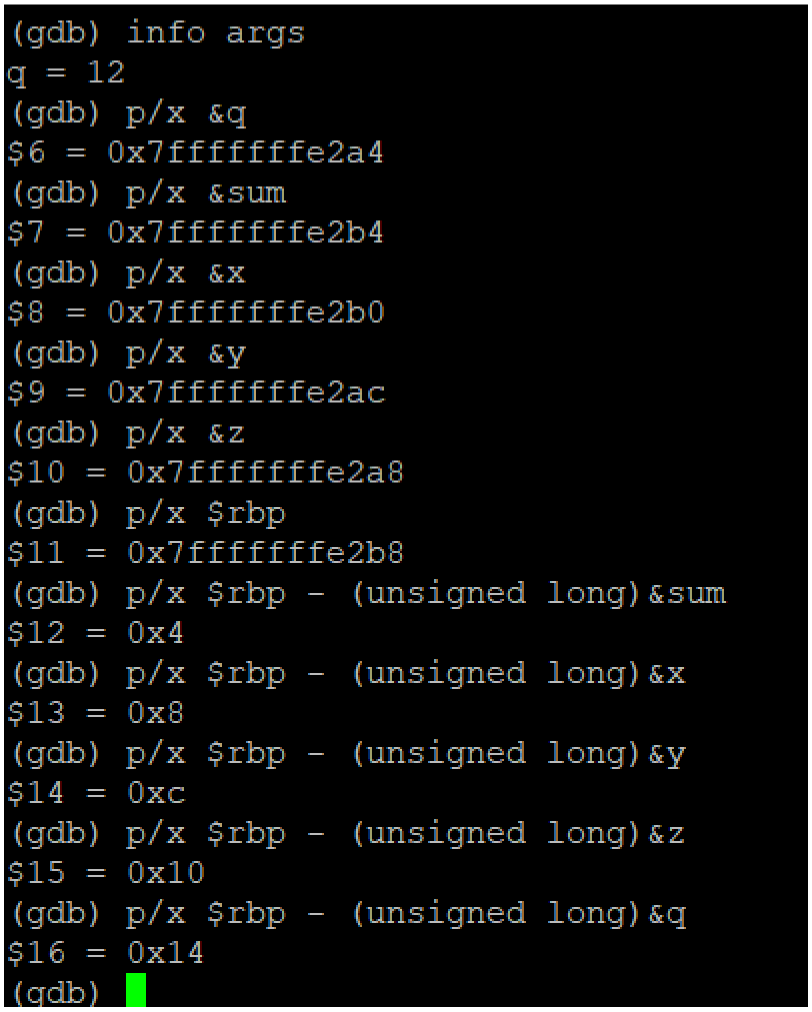


```
 Saved registers:
  rbp at 0x7fffffffe2b8, rip at 0x7fffffffe2c0
(gdb) info locals
x = 286331153
y = 572662306
z = 858993459
sum = 1717986930
(gdb) p/x &q
$1 = 0x7fffffffe2a4
(gdb) p/x &sum
$2 = 0x7fffffffe2b4
(gdb) p/x &x
$3 = 0x7fffffffe2b0
(gdb) p/x &y
$4 = 0x7fffffffe2ac
(gdb) p/x &z
$5 = 0x7fffffffe2a8
(gdb) info registers rbp rsp
rbp            0x7fffffffe2b8      0x7fffffffe2b8
rsp            0x7fffffffe2b8      0x7fffffffe2b8
(gdb)
```



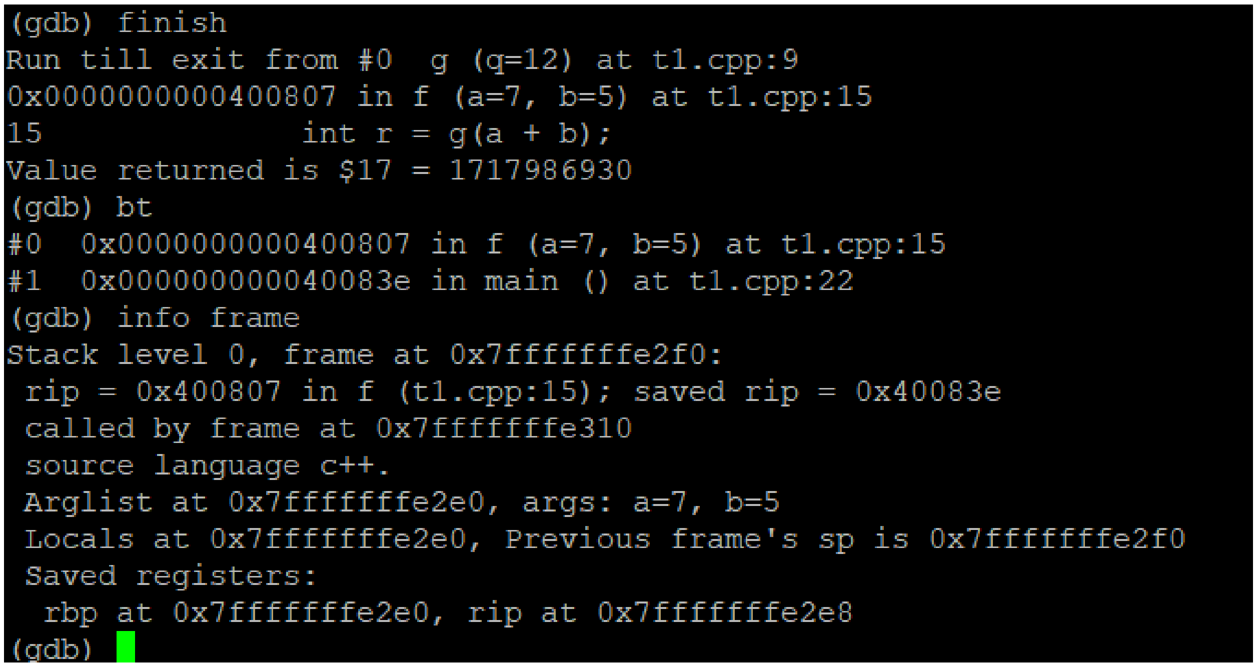
```
(gdb) info registers rbp rsp
rbp            0x7fffffffe2b8      0x7fffffffe2b8
rsp            0x7fffffffe2b8      0x7fffffffe2b8
(gdb) x/2gx $rbp
0x7fffffffe2b8: 0x00007fffffffe2e0      0x0000000000400807
(gdb) x/24wx $rbp-0x30
0x7fffffffe288: 0x00600dc8      0x00000000      0x00000001      0x00000000
0x7fffffffe298: 0x004008af      0x00000000      0x0060ae08      0x0000000c
0x7fffffffe2a8: 0x33333333      0x22222222      0x11111111      0x66666672
0x7fffffffe2b8: 0xffffe2e0      0x00007fff      0x00400807      0x00000000
0x7fffffffe2c8: 0x00000005      0x00000007      0xf7dd5cd0      0x55555555
0x7fffffffe2d8: 0x44444444      0x00000000      0xffffe300      0x00007fff
(gdb)
```

GDB memory view for g: saved RBP, saved return address, and locals (sum, x, y, z) plus parameter copy q in the activation record.

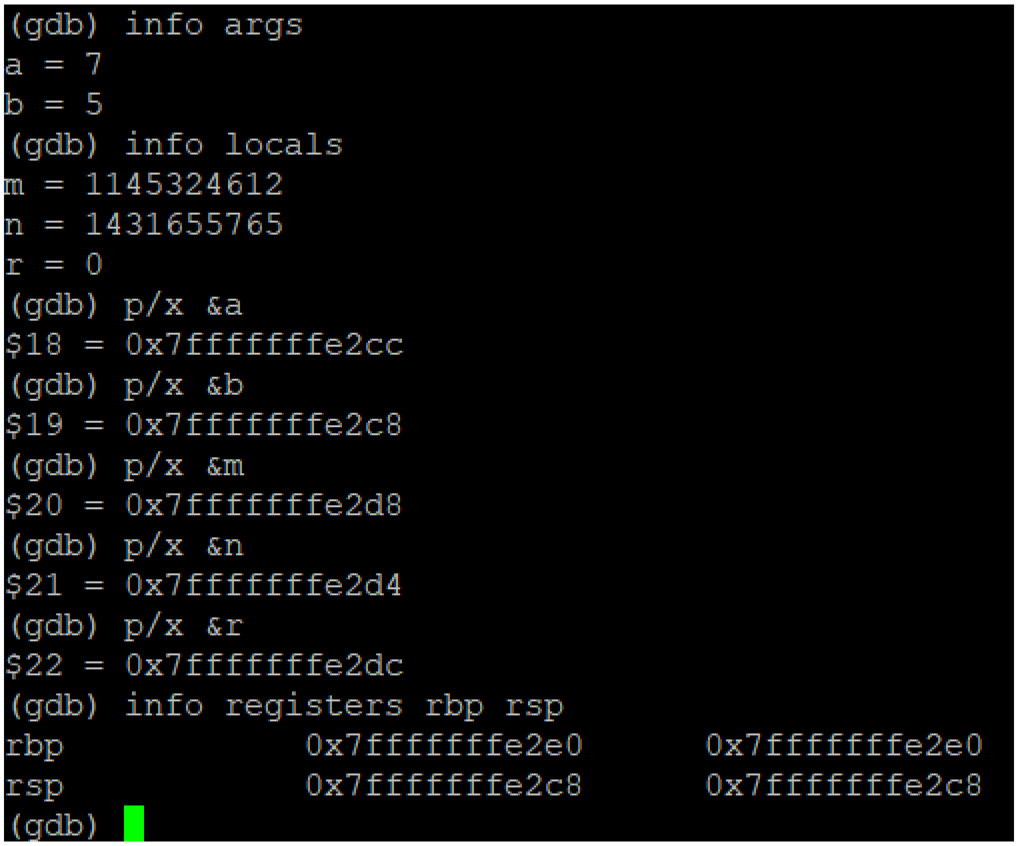


```
(gdb) info args
q = 12
(gdb) p/x &q
$6 = 0x7fffffffe2a4
(gdb) p/x &sum
$7 = 0x7fffffffe2b4
(gdb) p/x &x
$8 = 0x7fffffffe2b0
(gdb) p/x &y
$9 = 0x7fffffffe2ac
(gdb) p/x &z
$10 = 0x7fffffffe2a8
(gdb) p/x $rbp
$11 = 0x7fffffffe2b8
(gdb) p/x $rbp - (unsigned long)&sum
$12 = 0x4
(gdb) p/x $rbp - (unsigned long)&x
$13 = 0x8
(gdb) p/x $rbp - (unsigned long)&y
$14 = 0xc
(gdb) p/x $rbp - (unsigned long)&z
$15 = 0x10
(gdb) p/x $rbp - (unsigned long)&q
$16 = 0x14
(gdb)
```

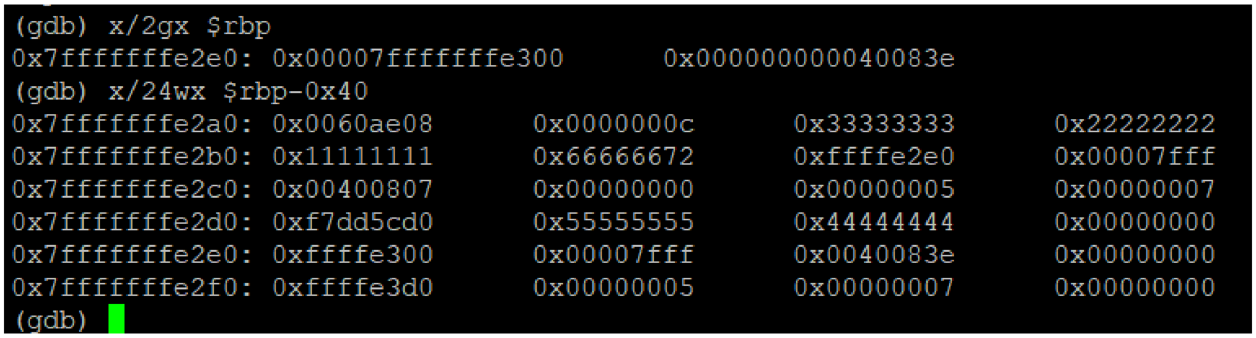
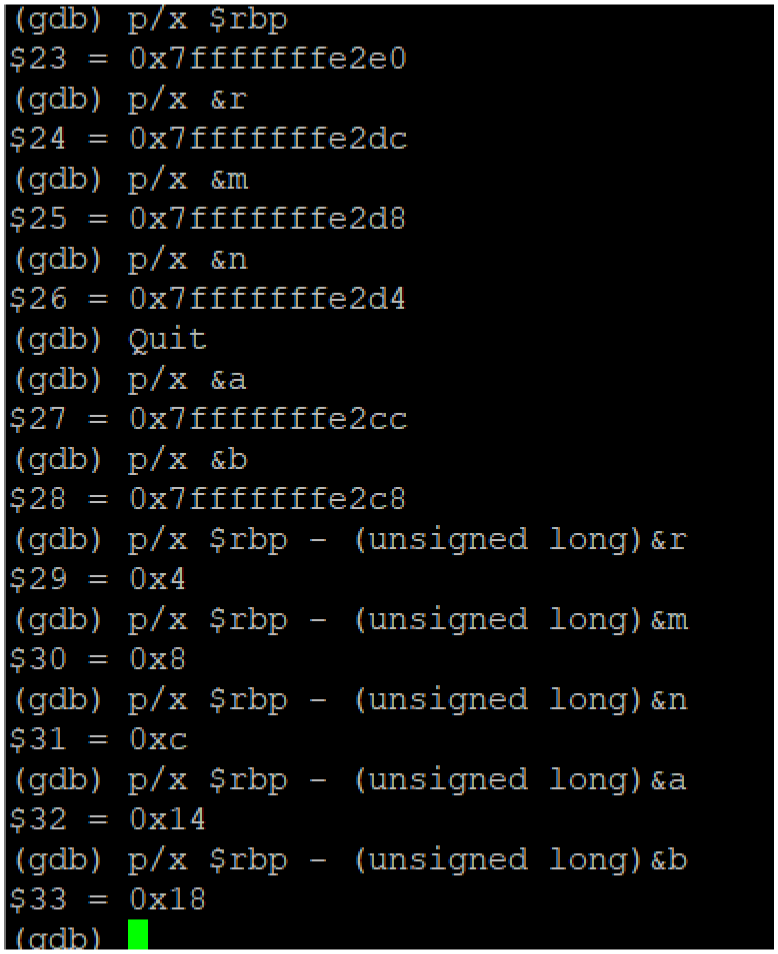
Offsets of parameter and locals from RBP in g (activation record layout).

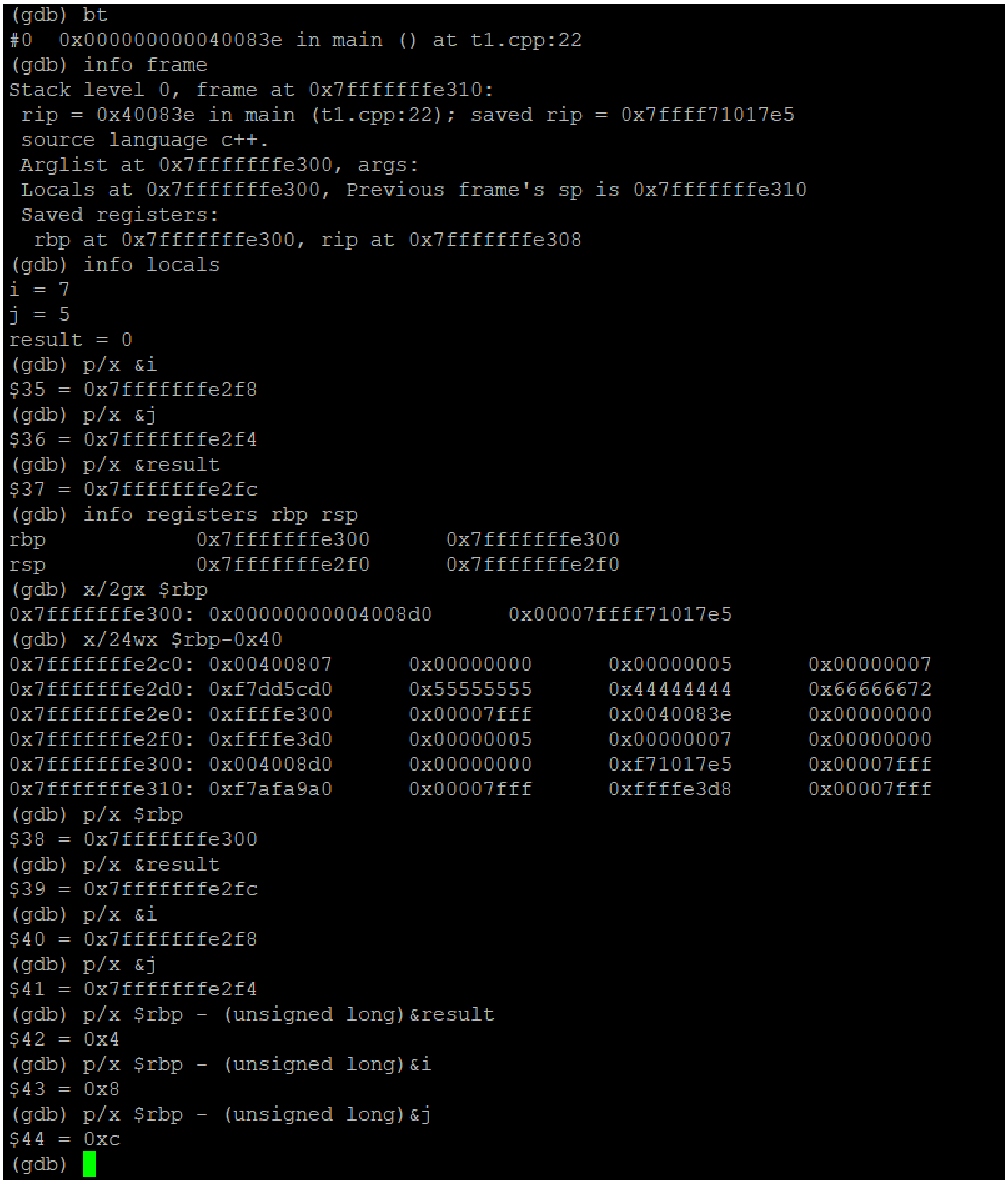


```
(gdb) finish
Run till exit from #0  g (q=12) at t1.cpp:9
0x0000000000400807 in f (a=7, b=5) at t1.cpp:15
15                int r = g(a + b);
Value returned is $17 = 1717986930
(gdb) bt
#0  0x0000000000400807 in f (a=7, b=5) at t1.cpp:15
#1  0x000000000040083e in main () at t1.cpp:22
(gdb) info frame
Stack level 0, frame at 0x7fffffffe2f0:
 rip = 0x400807 in f (t1.cpp:15); saved rip = 0x40083e
 called by frame at 0x7fffffffe310
 source language c++.
 Arglist at 0x7fffffffe2e0, args: a=7, b=5
 Locals at 0x7fffffffe2e0, Previous frame's sp is 0x7fffffffe2f0
 Saved registers:
  rbp at 0x7fffffffe2e0, rip at 0x7fffffffe2e8
(gdb)
```



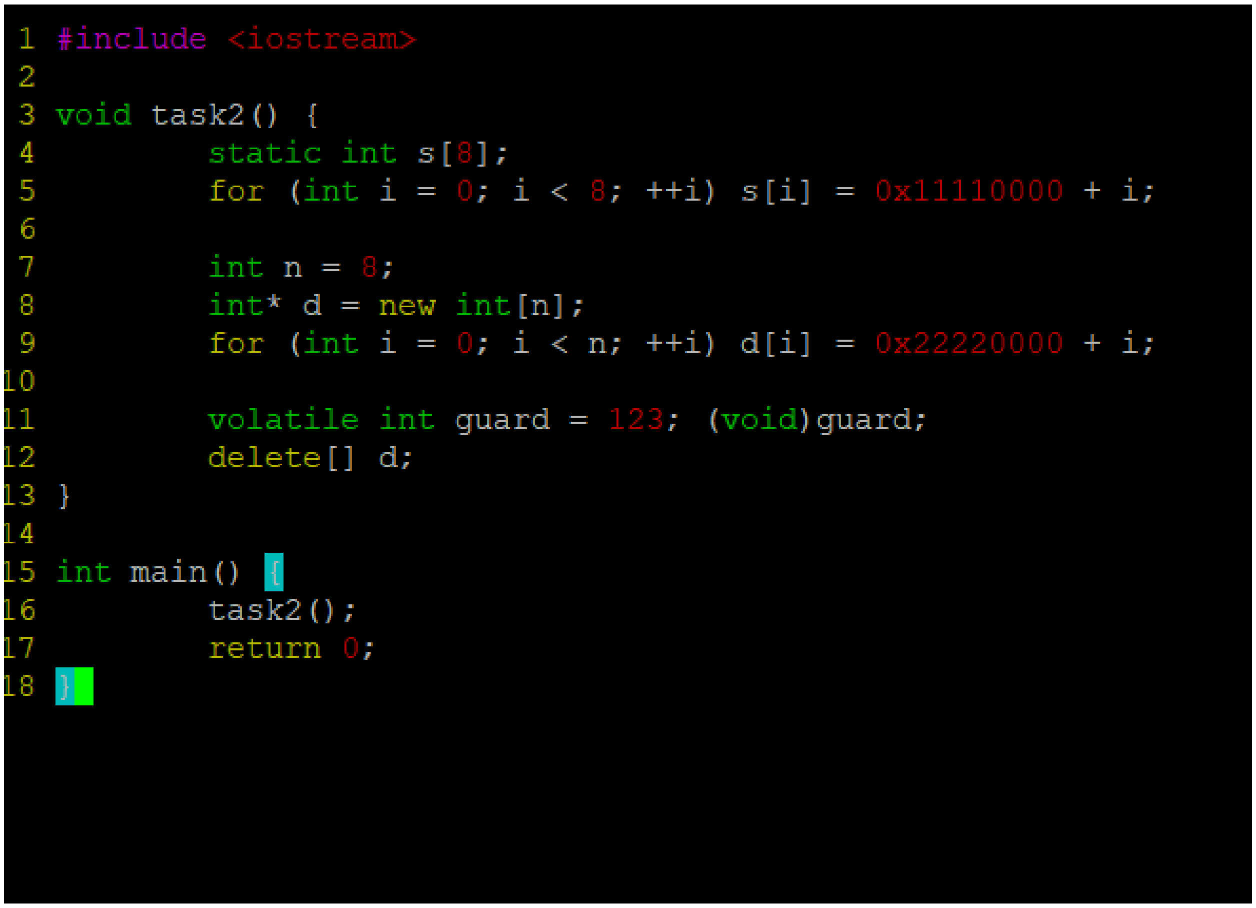
```
(gdb) info args
a = 7
b = 5
(gdb) info locals
m = 1145324612
n = 1431655765
r = 0
(gdb) p/x &a
$18 = 0x7fffffffe2cc
(gdb) p/x &b
$19 = 0x7fffffffe2c8
(gdb) p/x &m
$20 = 0x7fffffffe2d8
(gdb) p/x &n
$21 = 0x7fffffffe2d4
(gdb) p/x &r
$22 = 0x7fffffffe2dc
(gdb) info registers rbp rsp
rbp            0x7fffffffe2e0      0x7fffffffe2e0
rsp            0x7fffffffe2c8      0x7fffffffe2c8
(gdb)
```


```
(gdb) x/2gx $rbp
0x7fffffffe2e0: 0x00007fffffffe300      0x000000000040083e
(gdb) x/24wx $rbp-0x40
0x7fffffffe2a0: 0x0060ae08      0x0000000c      0x33333333      0x22222222
0x7fffffffe2b0: 0x11111111      0x66666672      0xffffe2e0      0x00007fff
0x7fffffffe2c0: 0x00400807      0x00000000      0x00000005      0x00000007
0x7fffffffe2d0: 0xf7dd5cd0      0x55555555      0x44444444      0x00000000
0x7fffffffe2e0: 0xffffe300      0x00007fff      0x0040083e      0x00000000
0x7fffffffe2f0: 0xffffe3d0      0x00000005      0x00000007      0x00000000
(gdb)
```

```
(gdb) p/x $rbp
$23 = 0x7fffffffe2e0
(gdb) p/x &r
$24 = 0x7fffffffe2dc
(gdb) p/x &m
$25 = 0x7fffffffe2d8
(gdb) p/x &n
$26 = 0x7fffffffe2d4
(gdb) Quit
(gdb) p/x &a
$27 = 0x7fffffffe2cc
(gdb) p/x &b
$28 = 0x7fffffffe2c8
(gdb) p/x $rbp - (unsigned long)&r
$29 = 0x4
(gdb) p/x $rbp - (unsigned long)&m
$30 = 0x8
(gdb) p/x $rbp - (unsigned long)&n
$31 = 0xc
(gdb) p/x $rbp - (unsigned long)&a
$32 = 0x14
(gdb) p/x $rbp - (unsigned long)&b
$33 = 0x18
(gdb)
```
Offsets of f’s parameters and locals from RBP



```
(gdb) bt
#0  0x000000000040083e in main () at t1.cpp:22
(gdb) info frame
Stack level 0, frame at 0x7fffffffe310:
 rip = 0x40083e in main (t1.cpp:22); saved rip = 0x7ffff71017e5
 source language c++.
 Arglist at 0x7fffffffe300, args:
 Locals at 0x7fffffffe300, Previous frame's sp is 0x7fffffffe310
 Saved registers:
  rbp at 0x7fffffffe300, rip at 0x7fffffffe308
(gdb) info locals
i = 7
j = 5
result = 0
(gdb) p/x &i
$35 = 0x7fffffffe2f8
(gdb) p/x &j
$36 = 0x7fffffffe2f4
(gdb) p/x &result
$37 = 0x7fffffffe2fc
(gdb) info registers rbp rsp
rbp            0x7fffffffe300      0x7fffffffe300
rsp            0x7fffffffe2f0      0x7fffffffe2f0
(gdb) x/2gx $rbp
0x7fffffffe300: 0x00000000004008d0      0x00007ffff71017e5
(gdb) x/24wx $rbp-0x40
0x7fffffffe2c0: 0x00400807      0x00000000      0x00000005      0x00000007
0x7fffffffe2d0: 0xf7dd5cd0      0x55555555      0x44444444      0x66666672
0x7fffffffe2e0: 0xffffe300      0x00007fff      0x0040083e      0x00000000
0x7fffffffe2f0: 0xffffe3d0      0x00000005      0x00000007      0x00000000
0x7fffffffe300: 0x004008d0      0x00000000      0xf71017e5      0x00007fff
0x7fffffffe310: 0xf7afa9a0      0x00007fff      0xffffe3d8      0x00007fff
(gdb) p/x $rbp
$38 = 0x7fffffffe300
(gdb) p/x &result
$39 = 0x7fffffffe2fc
(gdb) p/x &i
$40 = 0x7fffffffe2f8
(gdb) p/x &j
$41 = 0x7fffffffe2f4
(gdb) p/x $rbp - (unsigned long)&result
$42 = 0x4
(gdb) p/x $rbp - (unsigned long)&i
$43 = 0x8
(gdb) p/x $rbp - (unsigned long)&j
$44 = 0xc
(gdb) 
```

Task 2:


```
#include <iostream>

void task2() {
        static int s[8];
        for (int i = 0; i < 8; ++i) s[i] = 0x11110000 + i;

        int n = 8;
        int* d = new int[n];
        for (int i = 0; i < n; ++i) d[i] = 0x22220000 + i;

        volatile int guard = 123; (void)guard;
        delete[] d;
}

int main() {
        task2();
        return 0;
}
```

Setup & Method: I wrote a C++ program that allocates (1) a local static array static int s[8] and (2) a dynamic array int* d = new int[n] with recognizable contents. I compiled with -g -O0 on delmar.umsl.edu and used GDB. I set a breakpoint on the guard line just before delete[] d so both arrays remained allocated. I then recorded addresses and examined nearby memory bytes/words.

Are the static and dynamic arrays adjacent? No.
In my run:
&s[0] = 0x6010a0, &s[7] = 0x6010bc; d = &d[0] = 0x613eb0.
The difference &d[0] - &s[7] = 0x12df4 (≈77 KB) and &s[0] - &d[7] = -0x12e2c show the regions are far apart. GDB also shows s in static storage (near .bss/.data symbols), while d is on the heap. Conclusion: the static and dynamic arrays are not next to each other in memory on this system.

Do the preceding bytes of the dynamic array store size metadata?
I examined memory around d with:
x/16bx (char*)d-32, x/8gx (char*)d-32, and probed likely header slots:

(unsigned long)(d-8) = 0x31, *(unsigned long*)(d-16) = 0x0, *(unsigned long*)(d-24) = 0x0.

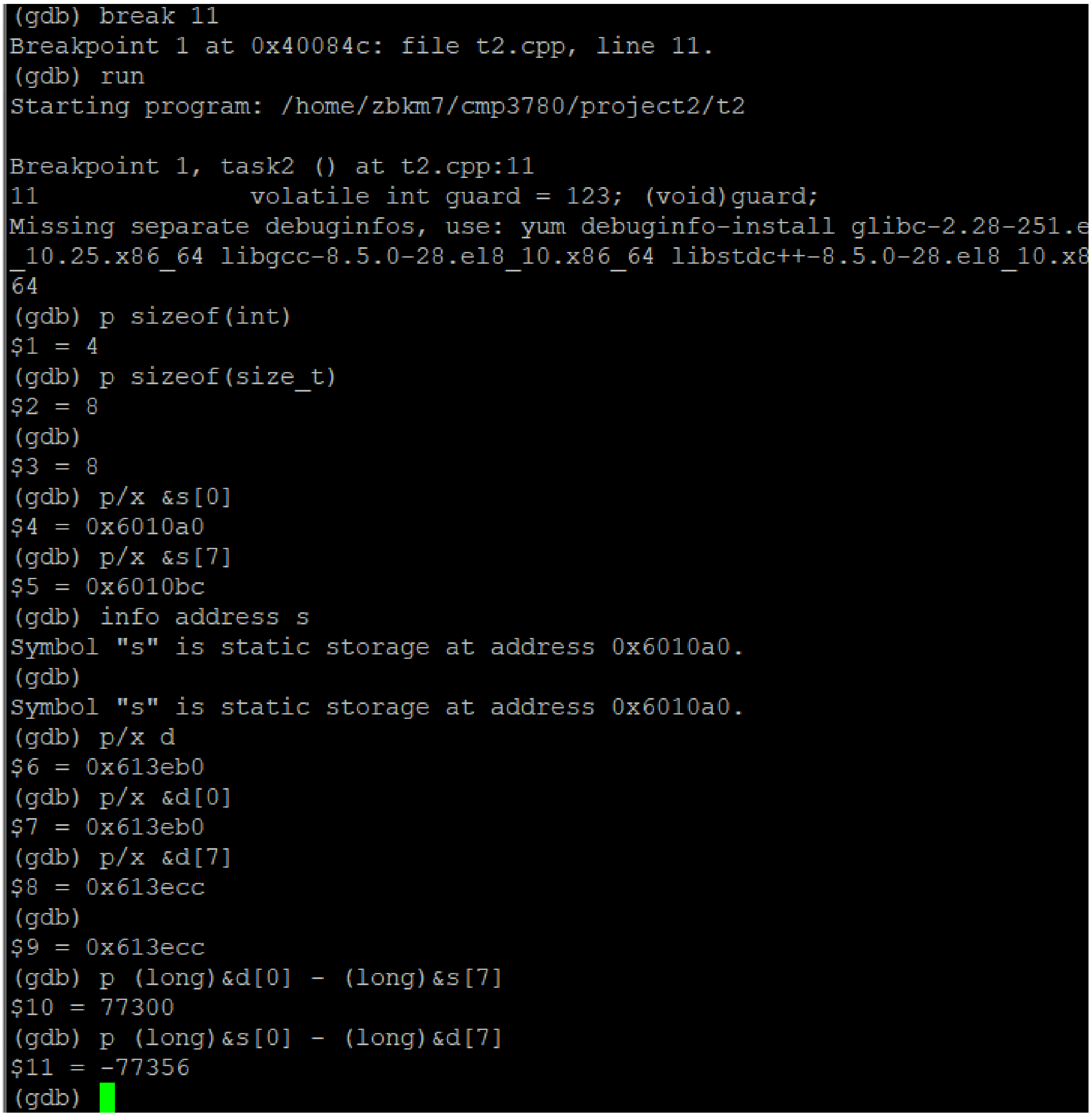
I did observe a non-zero word 8 bytes before the block (0x31), which is plausibly allocator/cookie metadata, but it does not equal n (8) or n*sizeof(int) (32). This aligns with the instruction that results vary by implementation and may not expose a simple, portable header format.

Does a similar header exist for the local static array? I examined bytes before &s[0] and saw no heap-style header—just zeros and other static data/symbols (consistent with static storage). Conclusion: no allocator size header like the one sometimes seen with dynamic allocations.

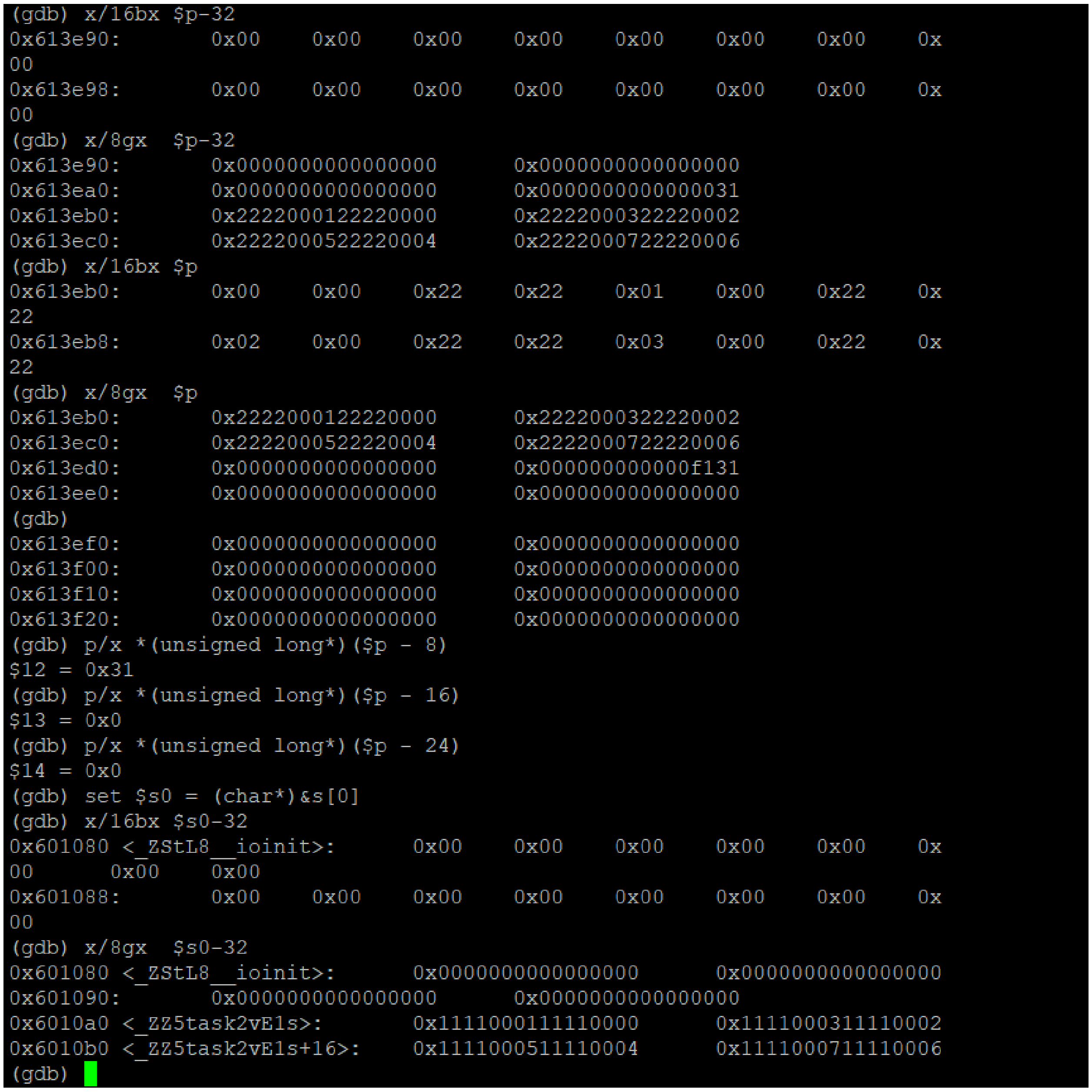
Screenshots include:

- Addresses, sections, and distance calculations for s and d.
- Memory inspections (“preceding bytes”) for d and s.

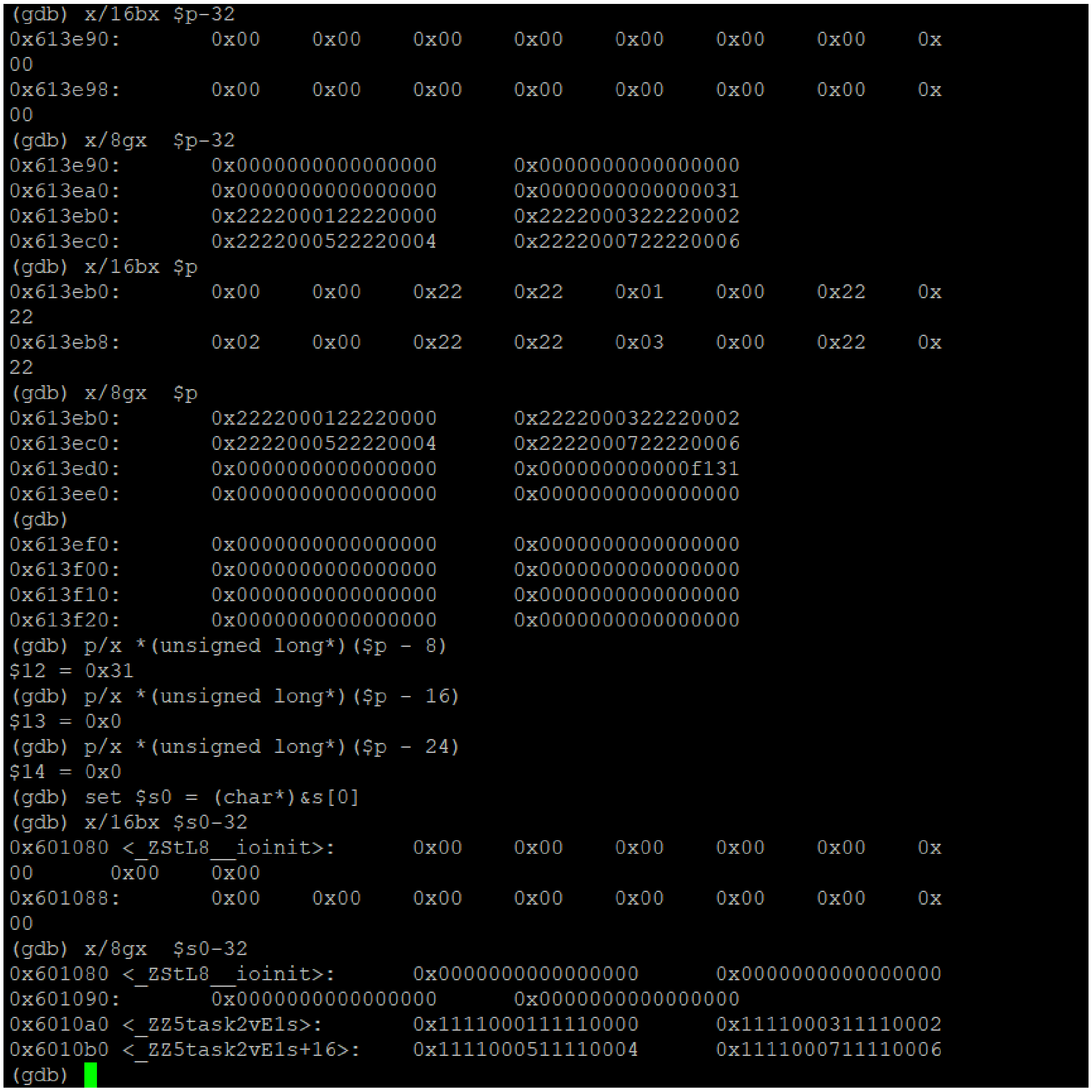
Analysis: On delmar for this build, static and dynamic arrays live in different regions (static storage vs. heap) and are not adjacent. The heap block had a non-zero word at d-8, suggesting internal metadata, but its value did not directly encode the element count or byte size. The static array did not show comparable header data. These observations demonstrate debugger-based memory inspection rather than any single expected outcome.



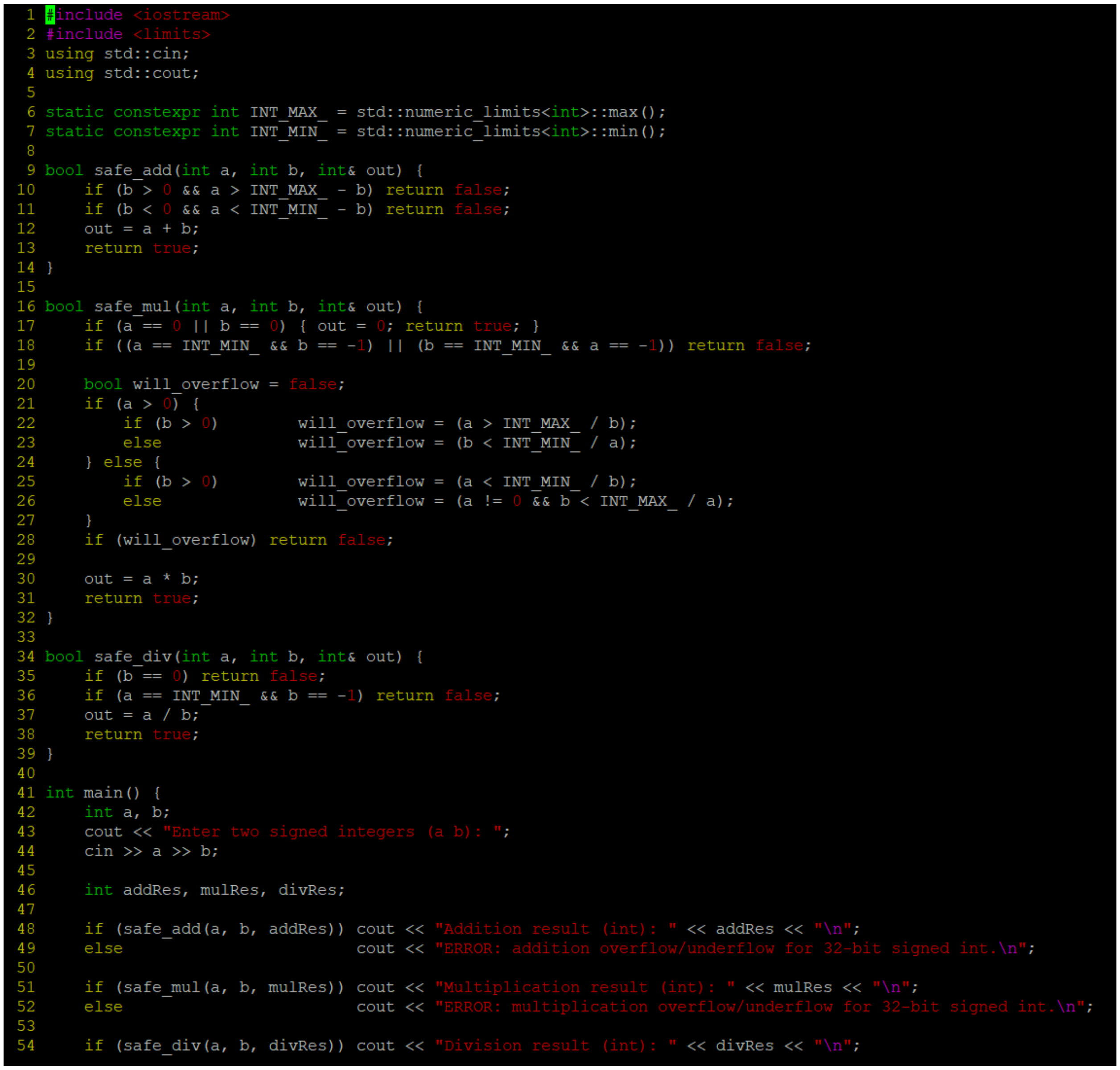
```
(gdb) break 11
Breakpoint 1 at 0x40084c: file t2.cpp, line 11.
(gdb) run
Starting program: /home/zbkm7/cmp3780/project2/t2

Breakpoint 1, task2 () at t2.cpp:11
11              volatile int guard = 123; (void)guard;
Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-251.e
_10.25.x86_64 libgcc-8.5.0-28.el8_10.x86_64 libstdc++-8.5.0-28.el8_10.x8
64
(gdb) p sizeof(int)
$1 = 4
(gdb) p sizeof(size_t)
$2 = 8
(gdb)
$3 = 8
(gdb) p/x &s[0]
$4 = 0x6010a0
(gdb) p/x &s[7]
$5 = 0x6010bc
(gdb) info address s
Symbol "s" is static storage at address 0x6010a0.
(gdb)
Symbol "s" is static storage at address 0x6010a0.
(gdb) p/x d
$6 = 0x613eb0
(gdb) p/x &d[0]
$7 = 0x613eb0
(gdb) p/x &d[7]
$8 = 0x613ecc
(gdb)
$9 = 0x613ecc
(gdb) p (long)&d[0] - (long)&s[7]
$10 = 77300
(gdb) p (long)&s[0] - (long)&d[7]
$11 = -77356
(gdb) 
```



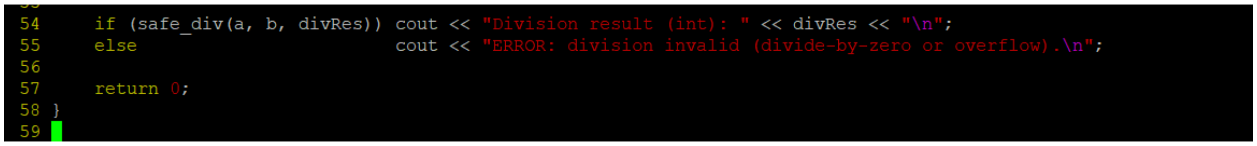
```
(gdb) x/16bx $p-32
0x613e90:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x
00
0x613e98:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x
00
(gdb) x/8gx  $p-32
0x613e90:       0x0000000000000000      0x0000000000000000
0x613ea0:       0x0000000000000000      0x0000000000000031
0x613eb0:       0x2222000122220000      0x2222000322220002
0x613ec0:       0x2222000522220004      0x2222000722220006
(gdb) x/16bx $p
0x613eb0:       0x00    0x00    0x22    0x22    0x01    0x00    0x22    0x
22
0x613eb8:       0x02    0x00    0x22    0x22    0x03    0x00    0x22    0x
22
(gdb) x/8gx  $p
0x613eb0:       0x2222000122220000      0x2222000322220002
0x613ec0:       0x2222000522220004      0x2222000722220006
0x613ed0:       0x0000000000000000      0x000000000000f131
0x613ee0:       0x0000000000000000      0x0000000000000000
(gdb)
0x613ef0:       0x0000000000000000      0x0000000000000000
0x613f00:       0x0000000000000000      0x0000000000000000
0x613f10:       0x0000000000000000      0x0000000000000000
0x613f20:       0x0000000000000000      0x0000000000000000
(gdb) p/x *(unsigned long*)($p - 8)
$12 = 0x31
(gdb) p/x *(unsigned long*)($p - 16)
$13 = 0x0
(gdb) p/x *(unsigned long*)($p - 24)
$14 = 0x0
(gdb) set $s0 = (char*)&s[0]
(gdb) x/16bx $s0-32
0x601080 <_ZStL8__ioinit>:      0x00    0x00    0x00    0x00    0x00    0x
00      0x00    0x00
0x601088:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x
00
(gdb) x/8gx  $s0-32
0x601080 <_ZStL8__ioinit>:      0x0000000000000000      0x0000000000000000
0x601090:       0x0000000000000000      0x0000000000000000
0x6010a0 <_ZZ5task2vE1s>:       0x1111000111110000      0x1111000311110002
0x6010b0 <_ZZ5task2vE1s+16>:    0x1111000511110004      0x1111000711110006
(gdb) 
```


```
(gdb) x/16bx $p-32
0x613e90:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x
00
0x613e98:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x
00
(gdb) x/8gx  $p-32
0x613e90:       0x0000000000000000      0x0000000000000000
0x613ea0:       0x0000000000000000      0x0000000000000031
0x613eb0:       0x2222000122220000      0x2222000322220002
0x613ec0:       0x2222000522220004      0x2222000722220006
(gdb) x/16bx $p
0x613eb0:       0x00    0x00    0x22    0x22    0x01    0x00    0x22    0x
22
0x613eb8:       0x02    0x00    0x22    0x22    0x03    0x00    0x22    0x
22
(gdb) x/8gx  $p
0x613eb0:       0x2222000122220000      0x2222000322220002
0x613ec0:       0x2222000522220004      0x2222000722220006
0x613ed0:       0x0000000000000000      0x000000000000f131
0x613ee0:       0x0000000000000000      0x0000000000000000
(gdb)
0x613ef0:       0x0000000000000000      0x0000000000000000
0x613f00:       0x0000000000000000      0x0000000000000000
0x613f10:       0x0000000000000000      0x0000000000000000
0x613f20:       0x0000000000000000      0x0000000000000000
(gdb) p/x *(unsigned long*)($p - 8)
$12 = 0x31
(gdb) p/x *(unsigned long*)($p - 16)
$13 = 0x0
(gdb) p/x *(unsigned long*)($p - 24)
$14 = 0x0
(gdb) set $s0 = (char*)&s[0]
(gdb) x/16bx $s0-32
0x601080 <_ZStL8__ioinit>:      0x00    0x00    0x00    0x00    0x00    0x
00      0x00    0x00
0x601088:       0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x
00
(gdb) x/8gx  $s0-32
0x601080 <_ZStL8__ioinit>:      0x0000000000000000      0x0000000000000000
0x601090:       0x0000000000000000      0x0000000000000000
0x6010a0 <_ZZ5task2vE1s>:       0x1111000111110000      0x1111000311110002
0x6010b0 <_ZZ5task2vE1s+16>:    0x1111000511110004      0x1111000711110006
(gdb)
```

Task 3:



```cpp
#include <iostream>
#include <limits>
using std::cin;
using std::cout;

static constexpr int INT_MAX_ = std::numeric_limits<int>::max();
static constexpr int INT_MIN_ = std::numeric_limits<int>::min();

bool safe_add(int a, int b, int& out) {
    if (b > 0 && a > INT_MAX_ - b) return false;
    if (b < 0 && a < INT_MIN_ - b) return false;
    out = a + b;
    return true;
}

bool safe_mul(int a, int b, int& out) {
    if (a == 0 || b == 0) { out = 0; return true; }
    if ((a == INT_MIN_ && b == -1) || (b == INT_MIN_ && a == -1)) return false;

    bool will_overflow = false;
    if (a > 0) {
        if (b > 0)          will_overflow = (a > INT_MAX_ / b);
        else                will_overflow = (b < INT_MIN_ / a);
    } else {
        if (b > 0)          will_overflow = (a < INT_MIN_ / b);
        else                will_overflow = (a != 0 && b < INT_MAX_ / a);
    }
    if (will_overflow) return false;

    out = a * b;
    return true;
}

bool safe_div(int a, int b, int& out) {
    if (b == 0) return false;
    if (a == INT_MIN_ && b == -1) return false;
    out = a / b;
    return true;
}

int main() {
    int a, b;
    cout << "Enter two signed integers (a b): ";
    cin >> a >> b;

    int addRes, mulRes, divRes;

    if (safe_add(a, b, addRes)) cout << "Addition result (int): " << addRes << "\n";
    else                        cout << "ERROR: addition overflow/underflow for 32-bit signed int.\n";

    if (safe_mul(a, b, mulRes)) cout << "Multiplication result (int): " << mulRes << "\n";
    else                        cout << "ERROR: multiplication overflow/underflow for 32-bit signed int.\n";

    if (safe_div(a, b, divRes)) cout << "Division result (int): " << divRes << "\n";
```



```cpp
    if (safe_div(a, b, divRes)) cout << "Division result (int): " << divRes << "\n";
    else                        cout << "ERROR: division invalid (divide-by-zero or overflow).\n";

    return 0;
}

```

Method: I wrote a C++ program that reads two signed 32-bit integers and performs addition, multiplication, and division. Before doing the operation, I check boundary conditions using std::numeric_limits<int> so detection does not rely on the overflow itself:

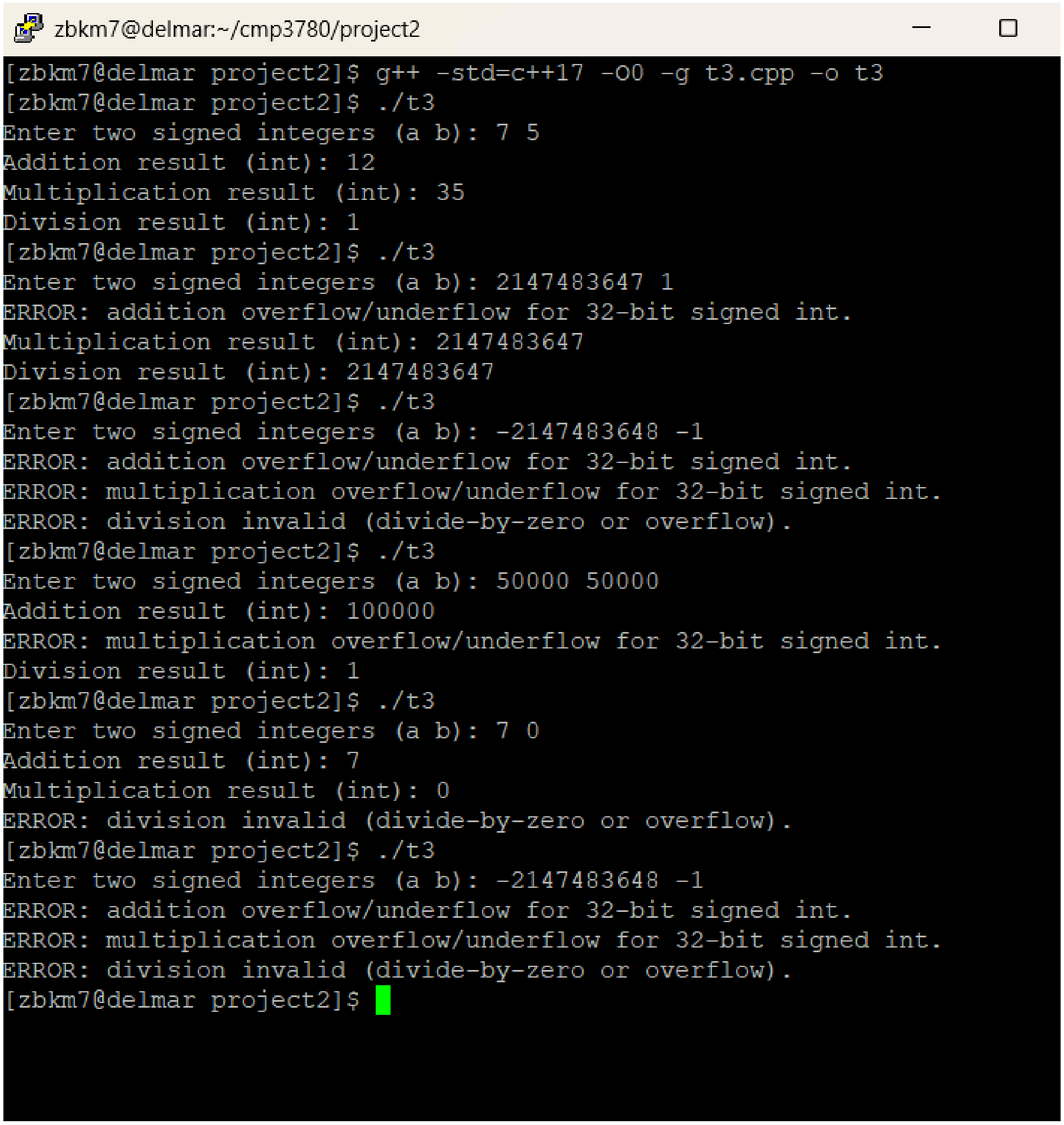
Addition:
overflow if b > 0 && a > INT_MAX - b; underflow if b < 0 && a < INT_MIN - b.

Multiplication:
handle 0 fast; reject INT_MIN * -1.
Otherwise use division pre-checks, e.g., for a > 0, b > 0: overflow if a > INT_MAX / b; analogous cases cover other sign combinations**.**

Division:
reject b == 0 and INT_MIN / -1.

Evidence (screenshot included):
Normal case 7 5 shows valid results.
2147483647 1 → addition overflow flagged.
-2147483648 -1 → addition underflow, multiplication overflow, and division invalid flagged.
50000 50000 → multiplication overflow flagged.
7 0 → divide-by-zero flagged.

Conclusion: The program safely detects overflow/underflow/invalid operations for 32-bit signed integers using pre-checks and reports an error message instead of producing incorrect results.



```
[zbkm7@delmar project2]$ g++ -std=c++17 -O0 -g t3.cpp -o t3
[zbkm7@delmar project2]$ ./t3
Enter two signed integers (a b): 7 5
Addition result (int): 12
Multiplication result (int): 35
Division result (int): 1
[zbkm7@delmar project2]$ ./t3
Enter two signed integers (a b): 2147483647 1
ERROR: addition overflow/underflow for 32-bit signed int.
Multiplication result (int): 2147483647
Division result (int): 2147483647
[zbkm7@delmar project2]$ ./t3
Enter two signed integers (a b): -2147483648 -1
ERROR: addition overflow/underflow for 32-bit signed int.
ERROR: multiplication overflow/underflow for 32-bit signed int.
ERROR: division invalid (divide-by-zero or overflow).
[zbkm7@delmar project2]$ ./t3
Enter two signed integers (a b): 50000 50000
Addition result (int): 100000
ERROR: multiplication overflow/underflow for 32-bit signed int.
Division result (int): 1
[zbkm7@delmar project2]$ ./t3
Enter two signed integers (a b): 7 0
Addition result (int): 7
Multiplication result (int): 0
ERROR: division invalid (divide-by-zero or overflow).
[zbkm7@delmar project2]$ ./t3
Enter two signed integers (a b): -2147483648 -1
ERROR: addition overflow/underflow for 32-bit signed int.
ERROR: multiplication overflow/underflow for 32-bit signed int.
ERROR: division invalid (divide-by-zero or overflow).
[zbkm7@delmar project2]$
```