# CS601: Software Development for Scientific Computing

Autumn 2024

Week7: Tools for profiling, debugging, and more..

# Valgrind

- Suite of tools for debugging and profiling
  - `memcheck` and `cachegrind` are popular ones
    - `cachegrind` is cache and branch-prediction profiler.
    - `memcheck` is a memory error detector.
- Demo of cachegrind tool with matmul
  - https://valgrind.org/docs/manual/cg-manual.html
- Demo of memcheck with matmul

# Steps to use `cachegrind`

- Example: `matmul.cpp`
    1. Compile with `–g` and create a target.
    2. Run as: `valgrind --tool=cachegrind ./matmul 2048`
    3. Output of `cachegrind` is dumped in a file that has the format `cachegrind.out.xxxxxx` where xxxxx is the process ID
    4. Use `cg_annotate` to get annotated output
        1. E.g. `cg_annotate cachegrind.out.12345`

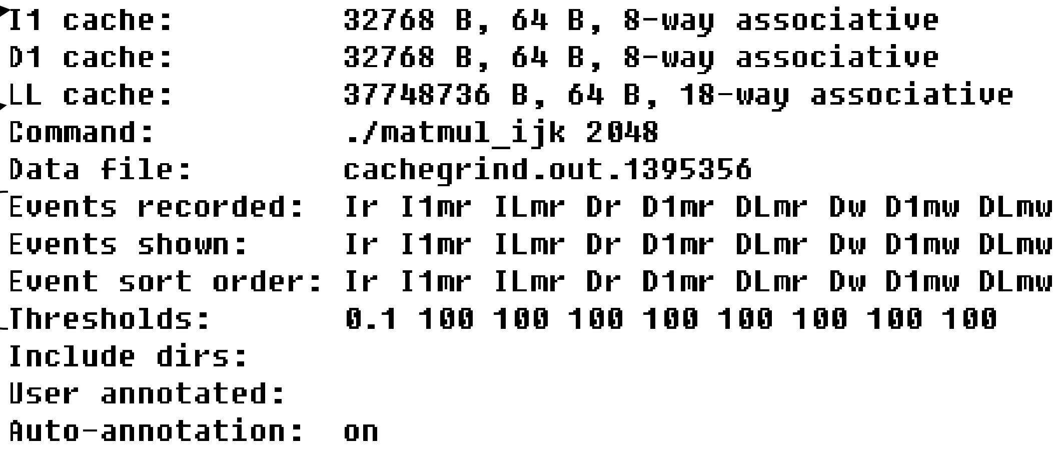
# cachegrind

- Visualizing cache transactions

L1 Instruction

L1 Data

Last layer



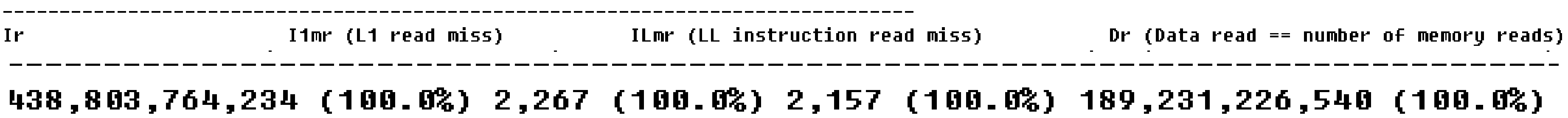
```
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         32768 B, 64 B, 8-way associative
LL cache:         37748736 B, 64 B, 18-way associative
Command:          ./matmul_ijk 2048
Data file:        cachegrind.out.1395356
Events recorded:  Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Events shown:     Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Event sort order: Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Thresholds:       0.1 100 100 100 100 100 100 100 100
Include dirs:
User annotated:
Auto-annotation:  on
```

- Instructions read
- L1 Instruction read misses
- Last layer instruction read misses
- Data reads (total memory reads)
- L1 data read misses
- Last layer data read misses
- Data writes (total memory writes)
- L1 data write misses
- Last layer data write misses

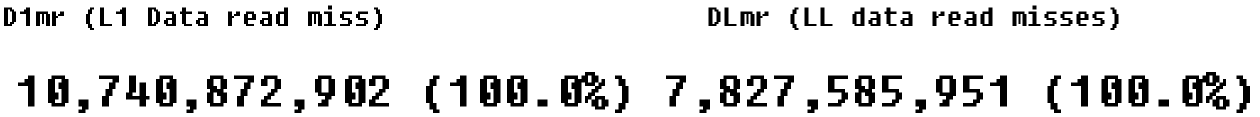
Total last layer misses = `ILmr + DLmr + DLmw`

# cachegrind

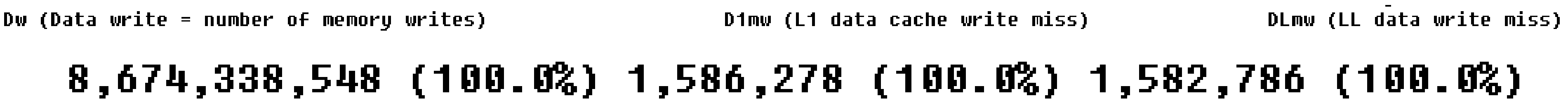
- Visualizing cache transactions (**ijk** loop ordering of matmul)



```
--------------------------------------------------------------------------------
Ir                        I1mr (L1 read miss)        ILmr (LL instruction read miss)          Dr (Data read == number of memory reads)
------------------------------------------------------------------------------------------------------------------------
438,803,764,234 (100.0%) 2,267 (100.0%) 2,157 (100.0%) 189,231,226,540 (100.0%)
```



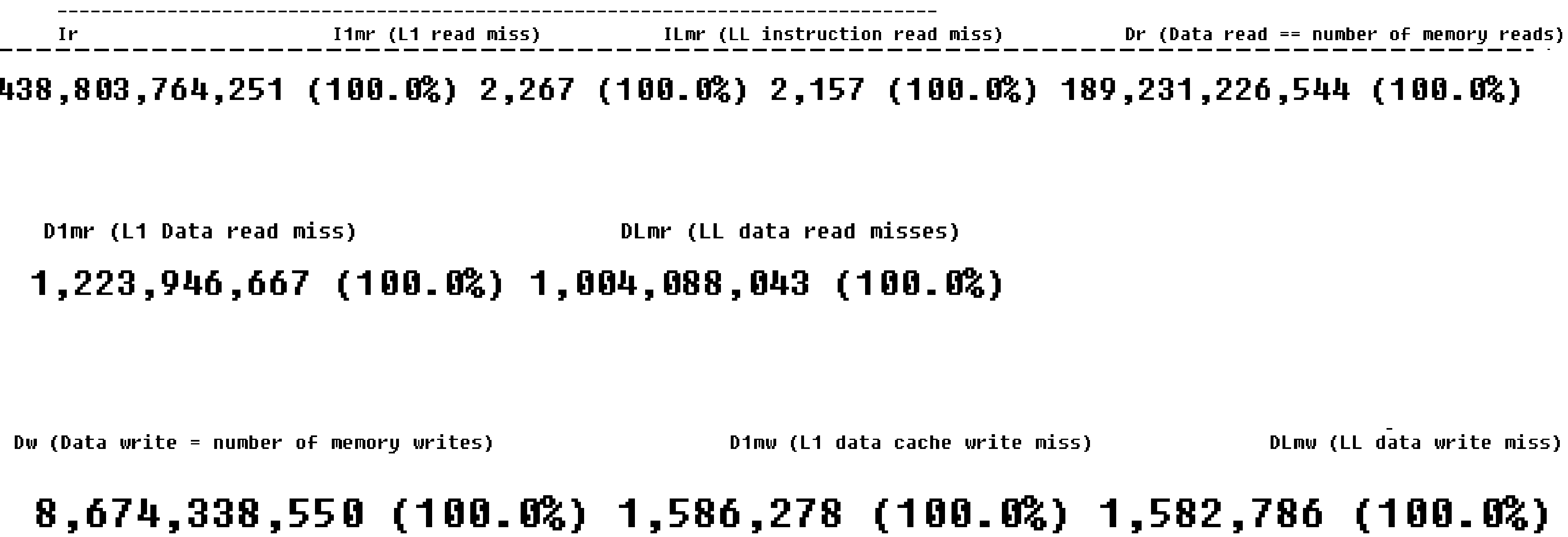
```
D1mr (L1 Data read miss)                        DLmr (LL data read misses)

10,740,872,902 (100.0%) 7,827,585,951 (100.0%)
```



```
Dw (Data write = number of memory writes)                    D1mw (L1 data cache write miss)             DLmw (LL data write miss)

8,674,338,548 (100.0%) 1,586,278 (100.0%) 1,582,786 (100.0%)
```

# cachegrind

- Visualizing cache transactions (**ikj** loop ordering of matmul)



```
    --------------------------------------------------------------------------------
    Ir                       I1mr (L1 read miss)          ILmr (LL instruction read miss)          Dr (Data read == number of memory reads)
--------------------------------------------------------------------------------------------------------------------------------------------
438,803,764,251 (100.0%) 2,267 (100.0%) 2,157 (100.0%) 189,231,226,544 (100.0%)
```

```
   D1mr (L1 Data read miss)                    DLmr (LL data read misses)

 1,223,946,667 (100.0%) 1,004,088,043 (100.0%)
```

```
 Dw (Data write = number of memory writes)                    D1mw (L1 data cache write miss)              DLmw (LL data write miss)

  8,674,338,550 (100.0%) 1,586,278 (100.0%) 1,582,786 (100.0%)
```

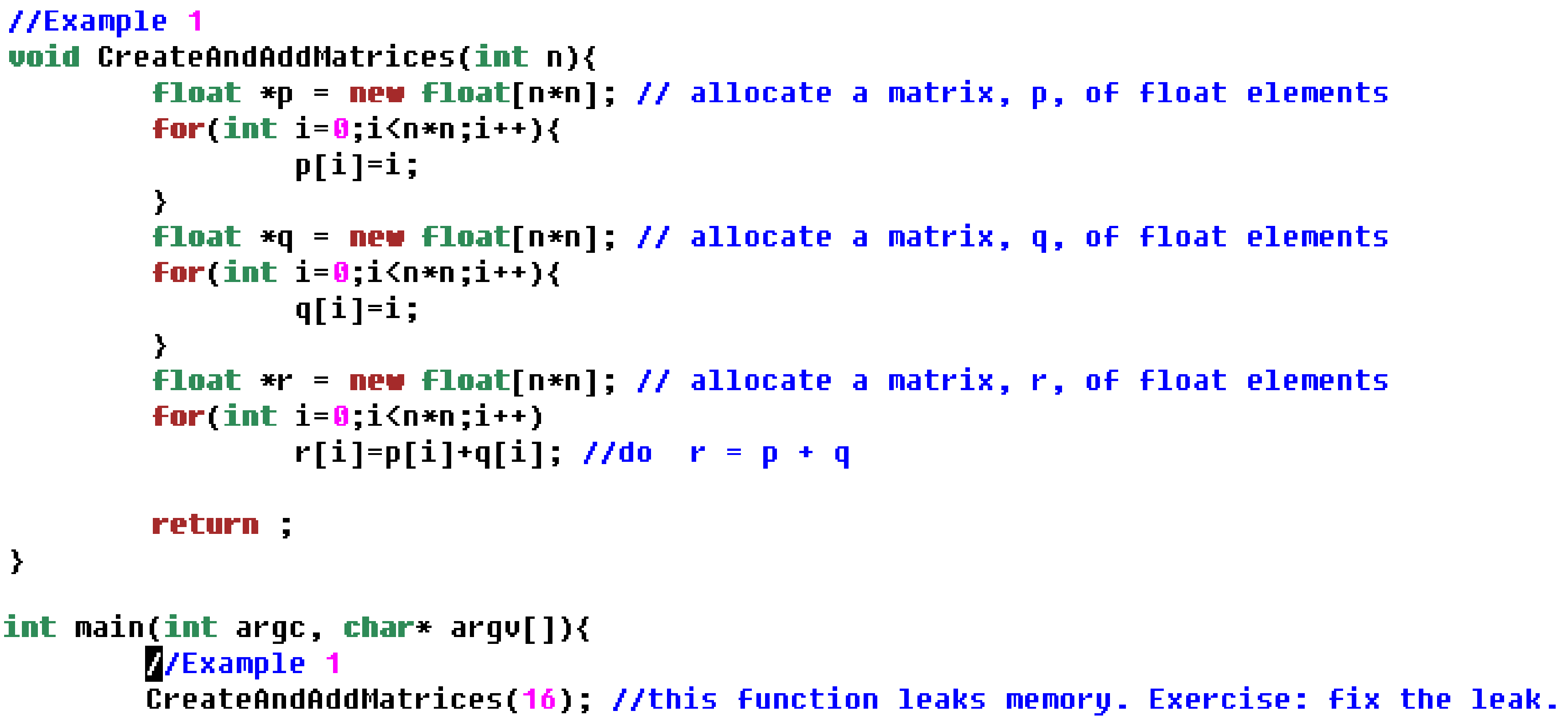
*Total last layer misses are much lesser than that in ijk loop!*

# GNU gprof

- Usage:
  - Compile your program with `–pg` flag
  - Execute your program as normal
    - A file `gmon.out` is generated
  - `gprof <yourexecutable>`

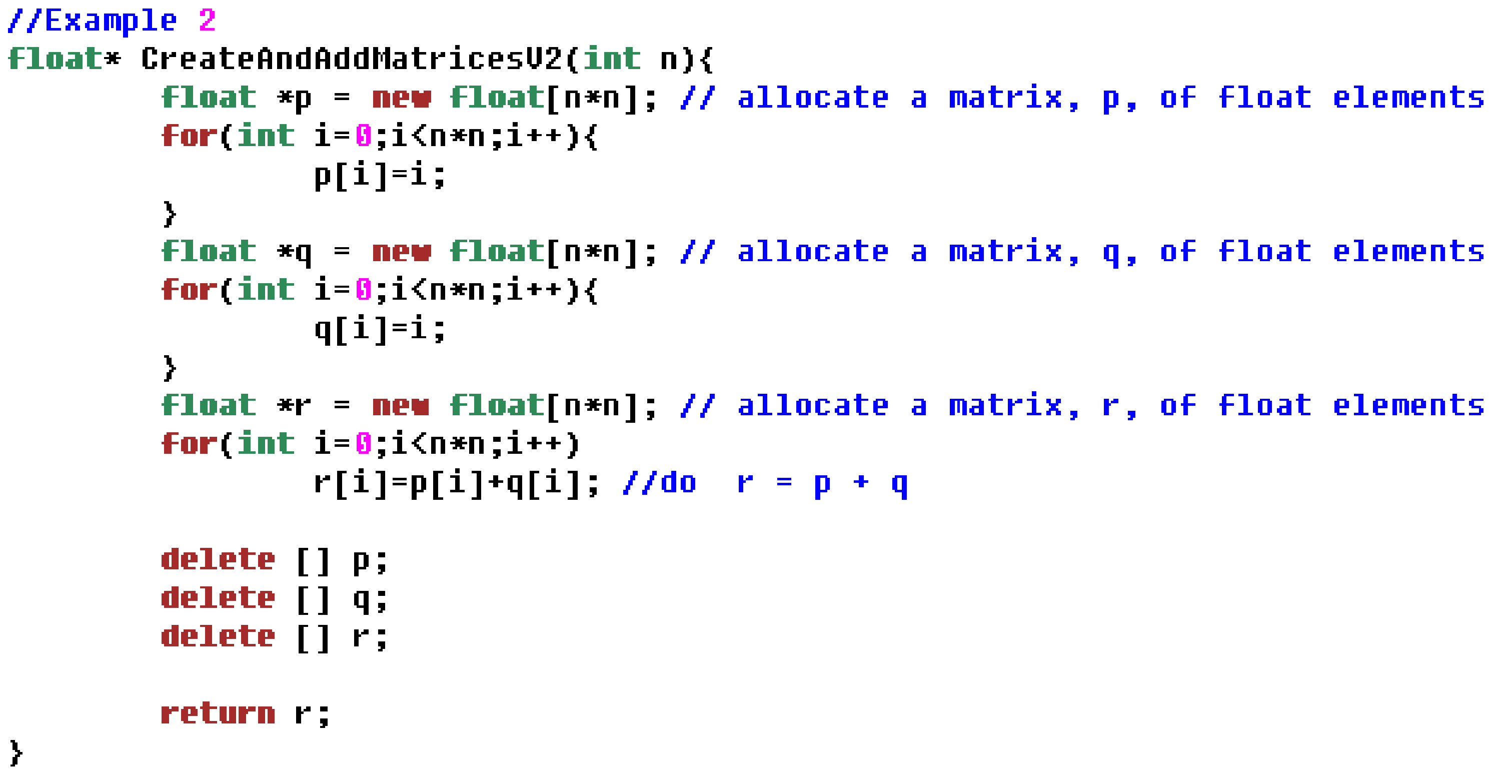
# Memcheck – ex1

- Used for detecting memory error that include memory leaks and invalid read/write to memory

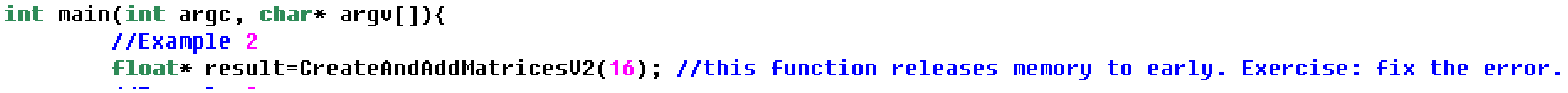


```
//Example 1
void CreateAndAddMatrices(int n){
        float *p = new float[n*n]; // allocate a matrix, p, of float elements
        for(int i=0;i<n*n;i++){
                p[i]=i;
        }
        float *q = new float[n*n]; // allocate a matrix, q, of float elements
        for(int i=0;i<n*n;i++){
                q[i]=i;
        }
        float *r = new float[n*n]; // allocate a matrix, r, of float elements
        for(int i=0;i<n*n;i++)
                r[i]=p[i]+q[i]; //do  r = p + q

        return ;
}

int main(int argc, char* argv[]){
        //Example 1
        CreateAndAddMatrices(16); //this function leaks memory. Exercise: fix the leak.
```

# memcheck – ex2

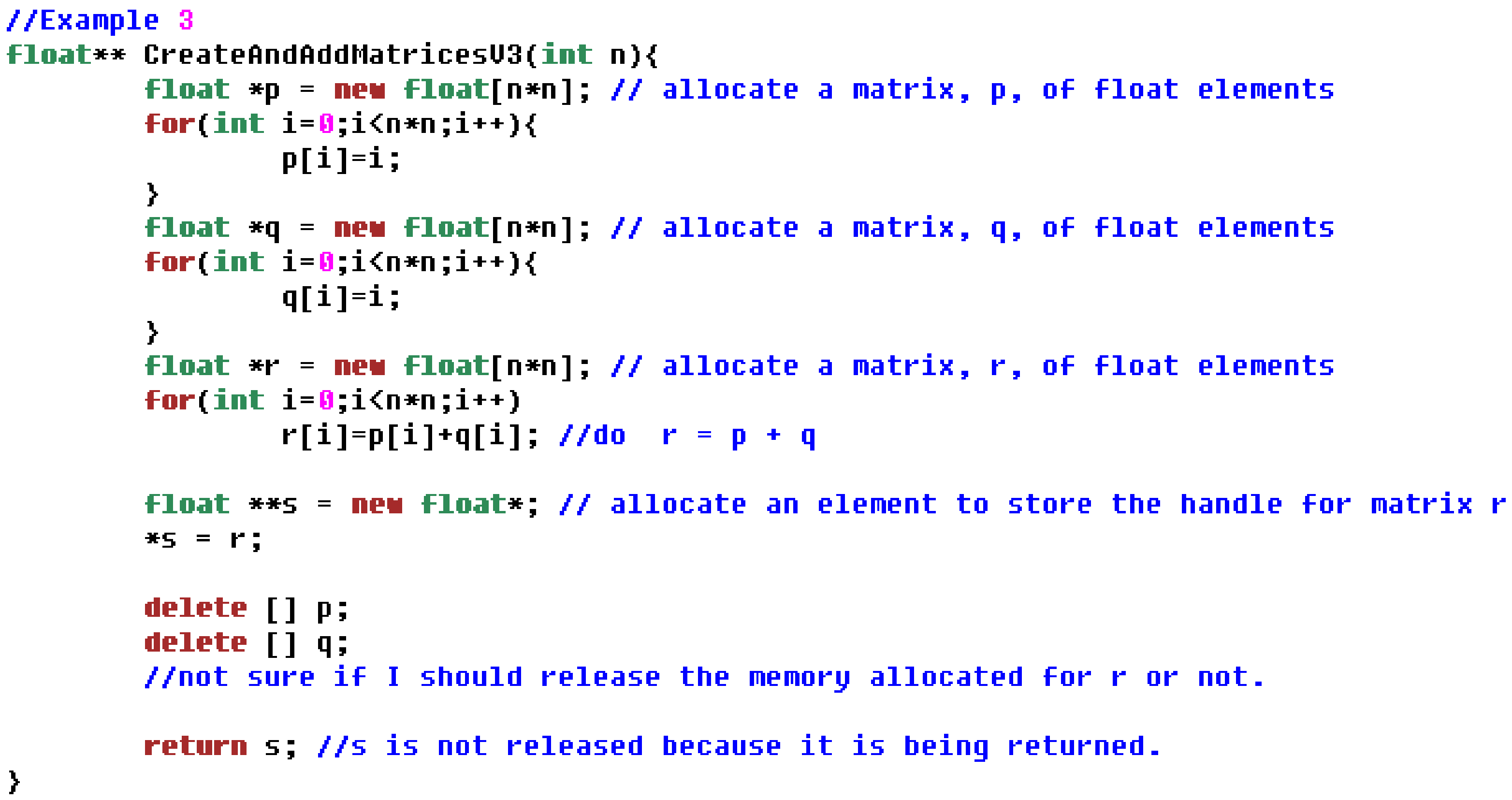


```
//Example 2
float* CreateAndAddMatricesV2(int n){
	float *p = new float[n*n]; // allocate a matrix, p, of float elements
	for(int i=0;i<n*n;i++){
		p[i]=i;
	}
	float *q = new float[n*n]; // allocate a matrix, q, of float elements
	for(int i=0;i<n*n;i++){
		q[i]=i;
	}
	float *r = new float[n*n]; // allocate a matrix, r, of float elements
	for(int i=0;i<n*n;i++)
		r[i]=p[i]+q[i]; //do  r = p + q

	delete [] p;
	delete [] q;
	delete [] r;

	return r;
}
```

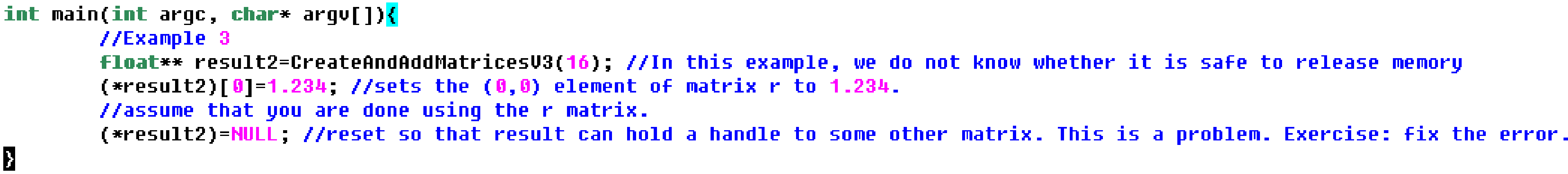


```
int main(int argc, char* argv[]){
	//Example 2
	float* result=CreateAndAddMatricesV2(16); //this function releases memory to early. Exercise: fix the error.
```

# memcheck – ex3



```
//Example 3
float** CreateAndAddMatricesV3(int n){
        float *p = new float[n*n]; // allocate a matrix, p, of float elements
        for(int i=0;i<n*n;i++){
                p[i]=i;
        }
        float *q = new float[n*n]; // allocate a matrix, q, of float elements
        for(int i=0;i<n*n;i++){
                q[i]=i;
        }
        float *r = new float[n*n]; // allocate a matrix, r, of float elements
        for(int i=0;i<n*n;i++)
                r[i]=p[i]+q[i]; //do  r = p + q

        float **s = new float*; // allocate an element to store the handle for matrix r
        *s = r;

        delete [] p;
        delete [] q;
        //not sure if I should release the memory allocated for r or not.

        return s; //s is not released because it is being returned.
}
```



```
int main(int argc, char* argv[]){
        //Example 3
        float** result2=CreateAndAddMatricesV3(16); //In this example, we do not know whether it is safe to release memory
        (*result2)[0]=1.234; //sets the (0,0) element of matrix r to 1.234.
        //assume that you are done using the r matrix.
        (*result2)=NULL; //reset so that result can hold a handle to some other matrix. This is a problem. Exercise: fix the error.
}
```

# memcheck - Usage

- Compile with –g option and create a target
- Execute with valgrind

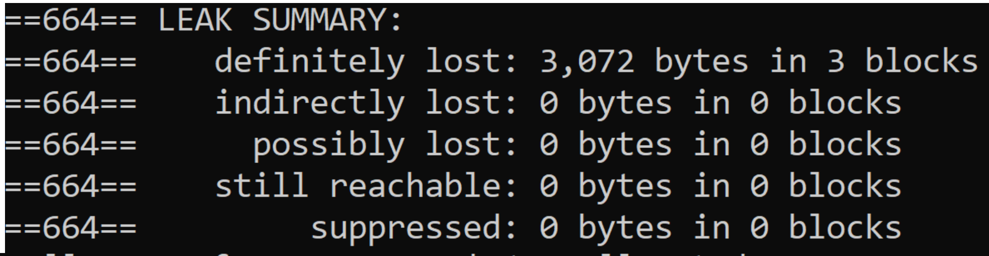
```
valgrind –tool=memcheck –leak-check=full mytarget
```

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

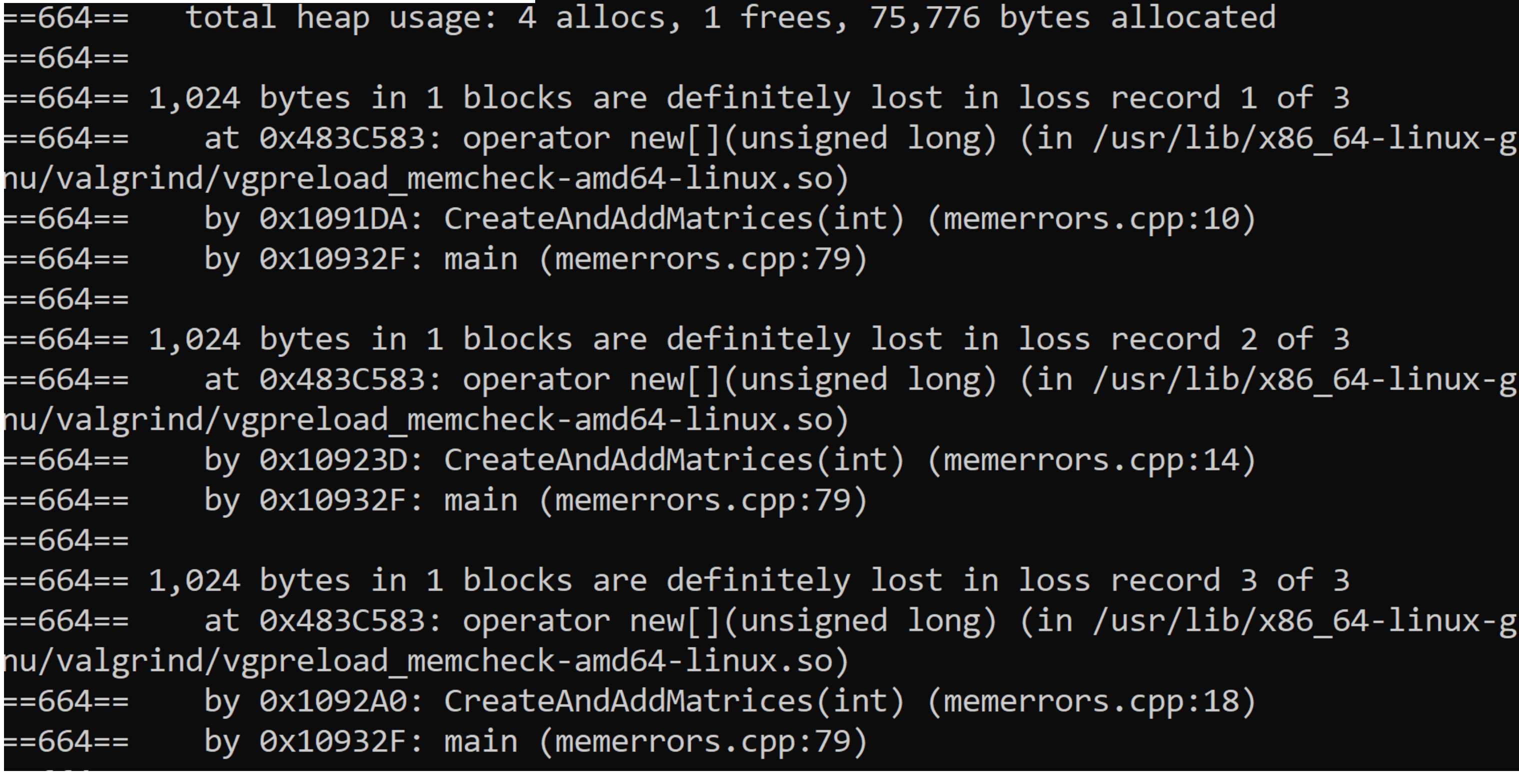
# memcheck - Demo

- From week7 code samples, run:

```
make –f memchkMakefile example1
```



```
==664== LEAK SUMMARY:
==664==    definitely lost: 3,072 bytes in 3 blocks
==664==    indirectly lost: 0 bytes in 0 blocks
==664==      possibly lost: 0 bytes in 0 blocks
==664==    still reachable: 0 bytes in 0 blocks
==664==         suppressed: 0 bytes in 0 blocks
```

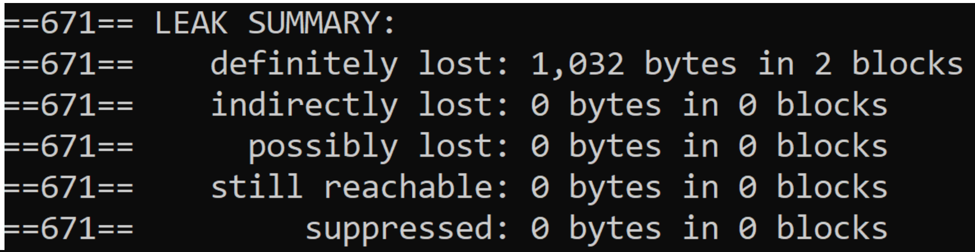


```
==664==   total heap usage: 4 allocs, 1 frees, 75,776 bytes allocated
==664==
==664== 1,024 bytes in 1 blocks are definitely lost in loss record 1 of 3
==664==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-g
nu/valgrind/vgpreload_memcheck-amd64-linux.so)
==664==    by 0x1091DA: CreateAndAddMatrices(int) (memerrors.cpp:10)
==664==    by 0x10932F: main (memerrors.cpp:79)
==664==
==664== 1,024 bytes in 1 blocks are definitely lost in loss record 2 of 3
==664==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-g
nu/valgrind/vgpreload_memcheck-amd64-linux.so)
==664==    by 0x10923D: CreateAndAddMatrices(int) (memerrors.cpp:14)
==664==    by 0x10932F: main (memerrors.cpp:79)
==664==
==664== 1,024 bytes in 1 blocks are definitely lost in loss record 3 of 3
==664==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-g
nu/valgrind/vgpreload_memcheck-amd64-linux.so)
==664==    by 0x1092A0: CreateAndAddMatrices(int) (memerrors.cpp:18)
==664==    by 0x10932F: main (memerrors.cpp:79)
```

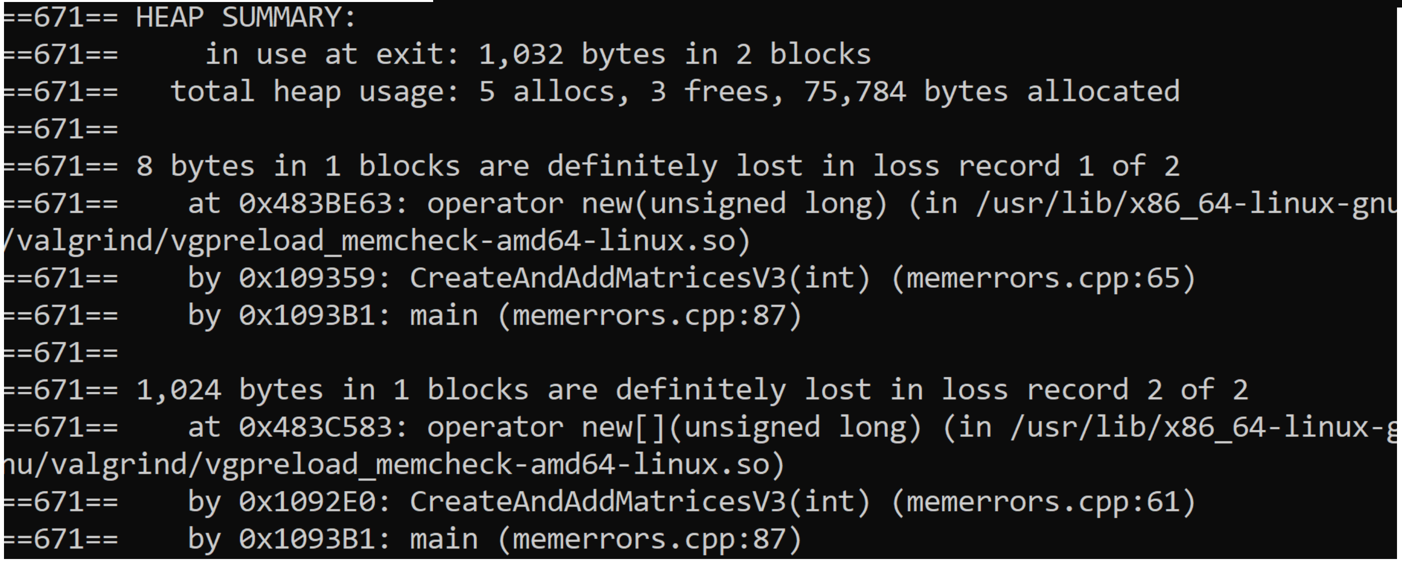
# memcheck - Demo

- From week7 code samples, run:

```
make –f memchkMakefile example3
```



```
==671== LEAK SUMMARY:
==671==    definitely lost: 1,032 bytes in 2 blocks
==671==    indirectly lost: 0 bytes in 0 blocks
==671==      possibly lost: 0 bytes in 0 blocks
==671==    still reachable: 0 bytes in 0 blocks
==671==         suppressed: 0 bytes in 0 blocks
```

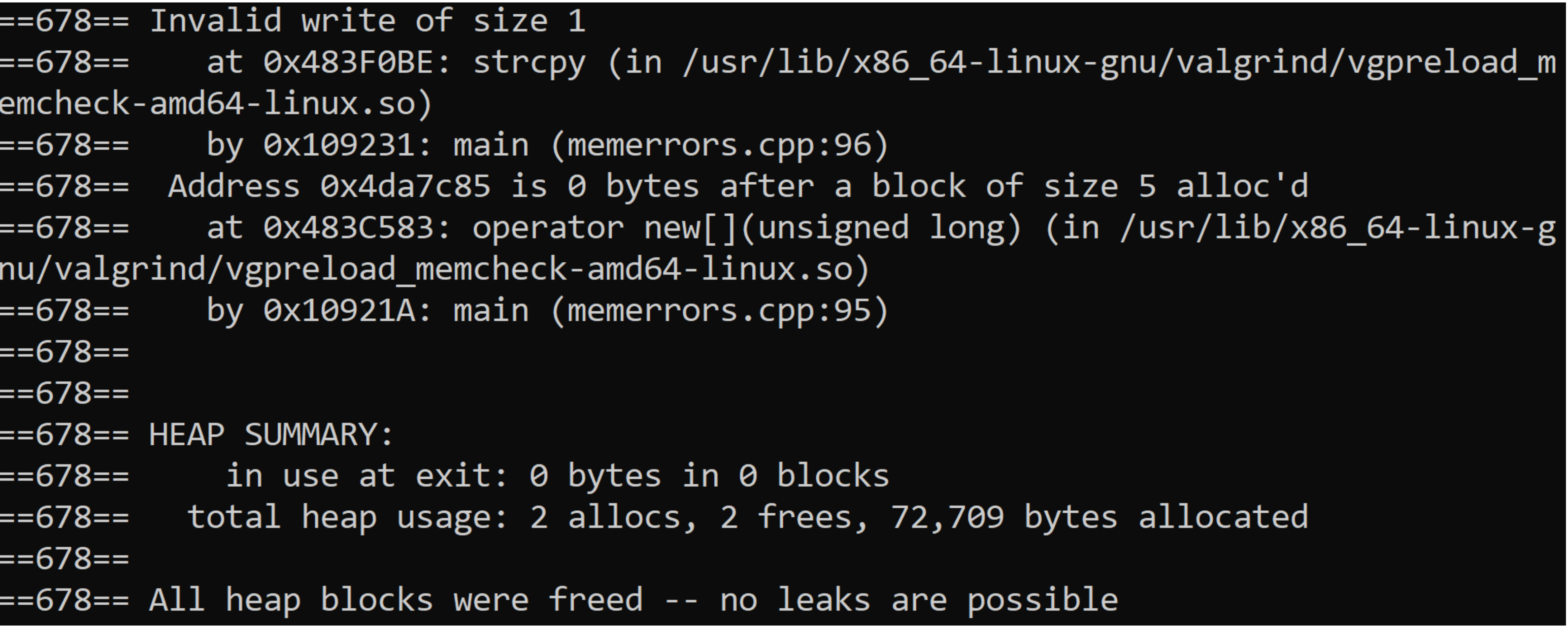


```
==671== HEAP SUMMARY:
==671==     in use at exit: 1,032 bytes in 2 blocks
==671==   total heap usage: 5 allocs, 3 frees, 75,784 bytes allocated
==671==
==671== 8 bytes in 1 blocks are definitely lost in loss record 1 of 2
==671==    at 0x483BE63: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu
/valgrind/vgpreload_memcheck-amd64-linux.so)
==671==    by 0x109359: CreateAndAddMatricesV3(int) (memerrors.cpp:65)
==671==    by 0x1093B1: main (memerrors.cpp:87)
==671==
==671== 1,024 bytes in 1 blocks are definitely lost in loss record 2 of 2
==671==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-g
nu/valgrind/vgpreload_memcheck-amd64-linux.so)
==671==    by 0x1092E0: CreateAndAddMatricesV3(int) (memerrors.cpp:61)
==671==    by 0x1093B1: main (memerrors.cpp:87)
```

# memcheck - Demo

- From week7 code samples, run:

```
make –f memchkMakefile example4
```

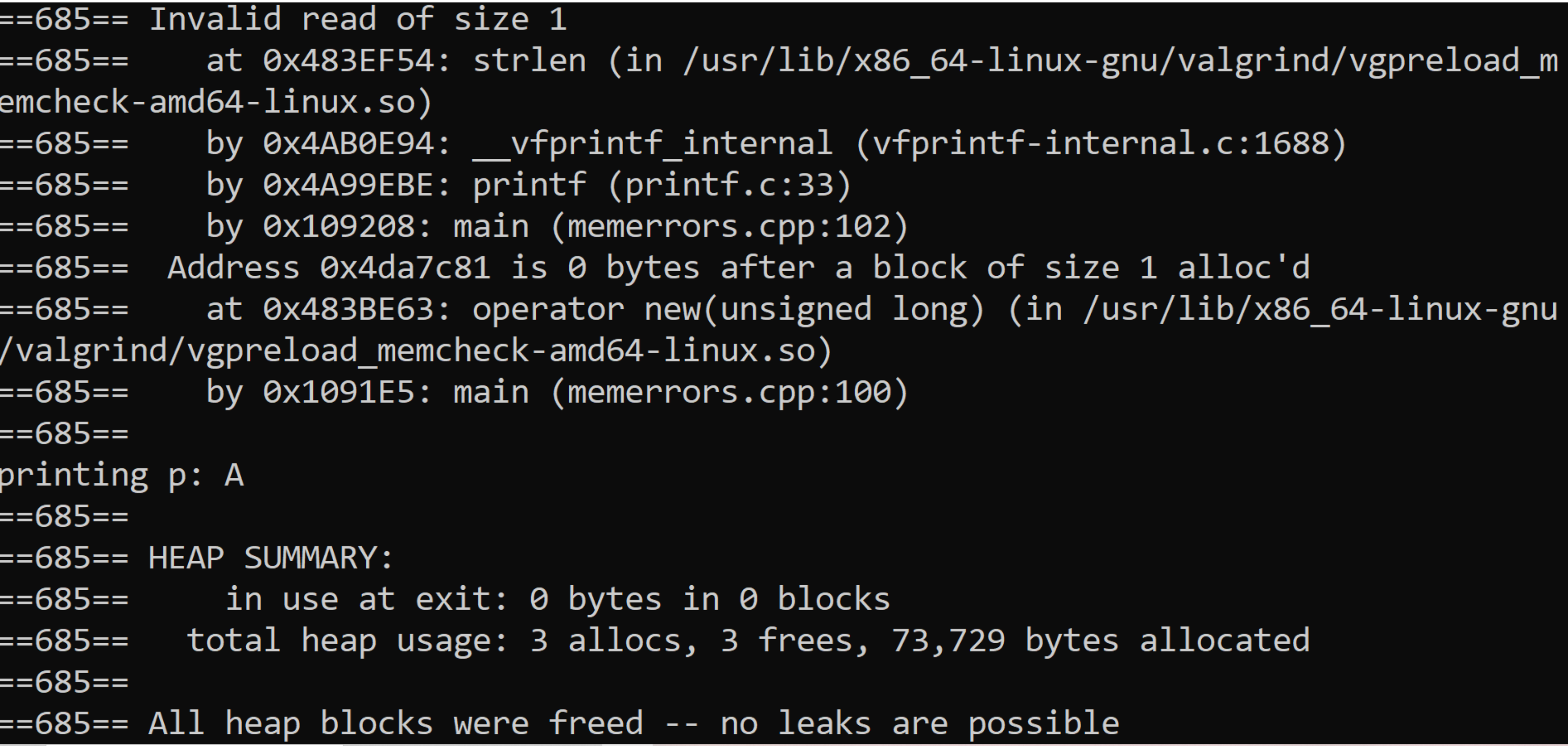


```
==678== Invalid write of size 1
==678==    at 0x483F0BE: strcpy (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_m
emcheck-amd64-linux.so)
==678==    by 0x109231: main (memerrors.cpp:96)
==678==  Address 0x4da7c85 is 0 bytes after a block of size 5 alloc'd
==678==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-g
nu/valgrind/vgpreload_memcheck-amd64-linux.so)
==678==    by 0x10921A: main (memerrors.cpp:95)
==678==
==678==
==678== HEAP SUMMARY:
==678==     in use at exit: 0 bytes in 0 blocks
==678==   total heap usage: 2 allocs, 2 frees, 72,709 bytes allocated
==678==
==678== All heap blocks were freed -- no leaks are possible
```

# memcheck - Demo

- From week7 code samples, run:

```
make –f memchkMakefile example5
```



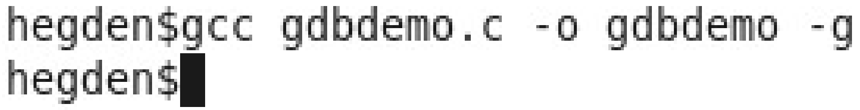
```
==685== Invalid read of size 1
==685==    at 0x483EF54: strlen (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_m
emcheck-amd64-linux.so)
==685==    by 0x4AB0E94: __vfprintf_internal (vfprintf-internal.c:1688)
==685==    by 0x4A99EBE: printf (printf.c:33)
==685==    by 0x109208: main (memerrors.cpp:102)
==685==  Address 0x4da7c81 is 0 bytes after a block of size 1 alloc'd
==685==    at 0x483BE63: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu
/valgrind/vgpreload_memcheck-amd64-linux.so)
==685==    by 0x1091E5: main (memerrors.cpp:100)
==685==
printing p: A
==685==
==685== HEAP SUMMARY:
==685==     in use at exit: 0 bytes in 0 blocks
==685==   total heap usage: 3 allocs, 3 frees, 73,729 bytes allocated
==685==
==685== All heap blocks were freed -- no leaks are possible
```

# GDB

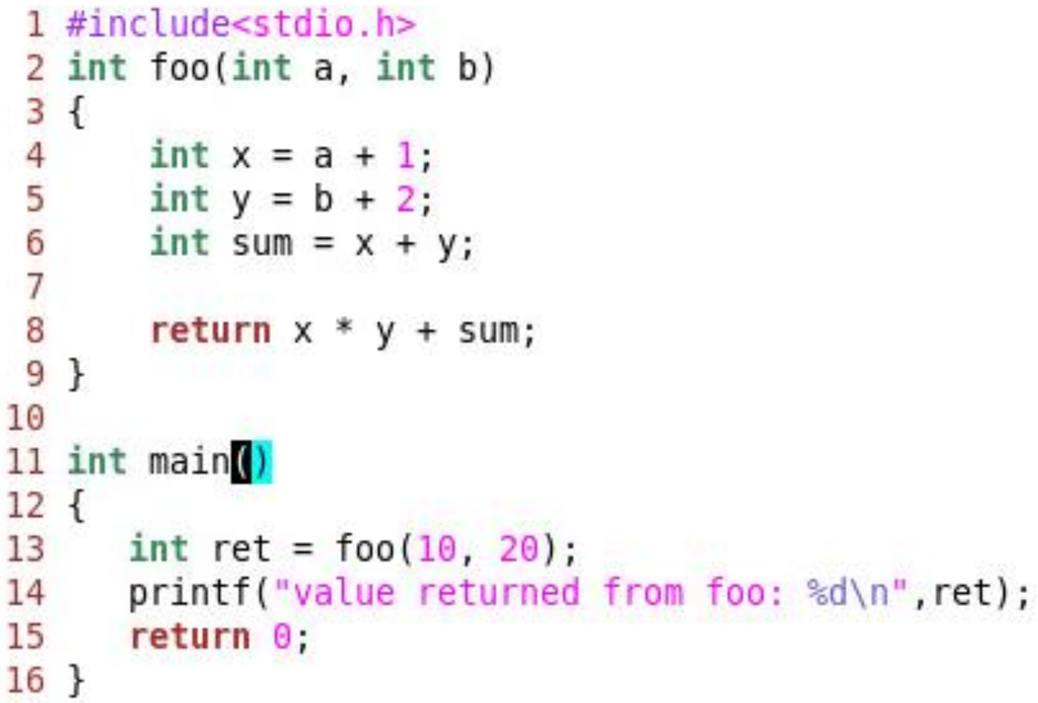
- – GNU Debugger – A tool for inspecting your C/C++ programs
  - How to begin inspecting a program using gdb?
  - How to control the execution?
  - How to display, interpret, and alter memory contents of a program using gdb?
  - Misc – displaying stack frames, visualizing assembler code.

# GDB

– Compile your programs with –g option



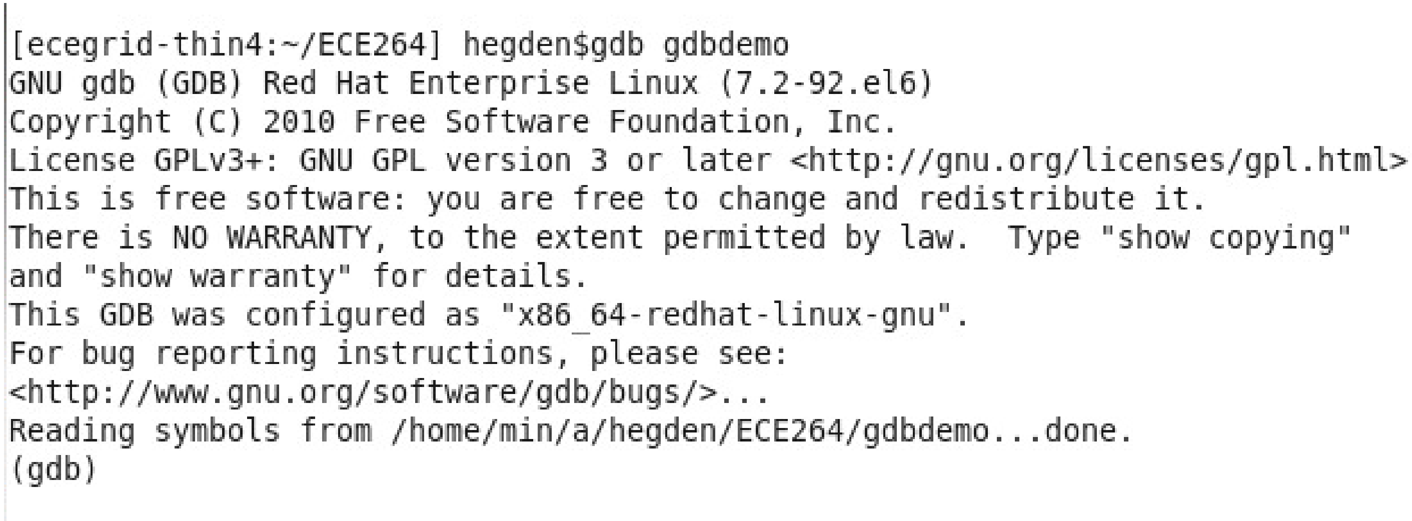
```
hegden$gcc gdbdemo.c -o gdbdemo -g
hegden$
```



```c
#include<stdio.h>
int foo(int a, int b)
{
    int x = a + 1;
    int y = b + 2;
    int sum = x + y;

    return x * y + sum;
}

int main()
{
   int ret = foo(10, 20);
   printf("value returned from foo: %d\n",ret);
   return 0;
}
```

# GDB – Start Debug

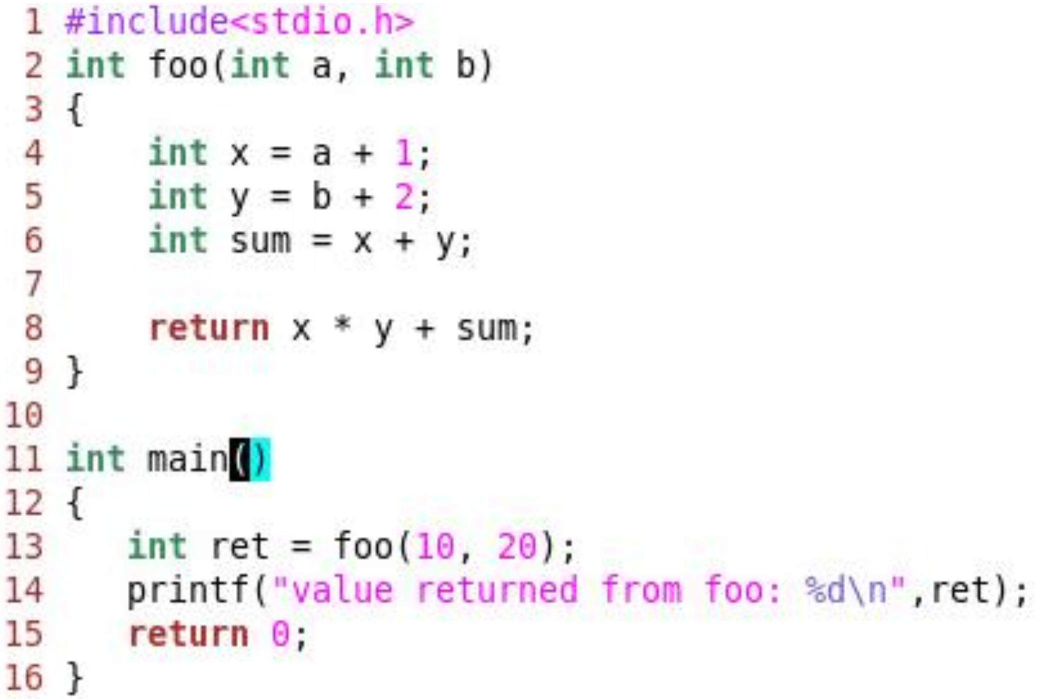
- Start debug mode (`gdb gdbdemo`)
  - Note the executable on first line (not .c files)
  - Note the last line before `(gdb)` prompt:
    - if –g option is not used while compiling, you will see “`(no debugging symbols found)`”



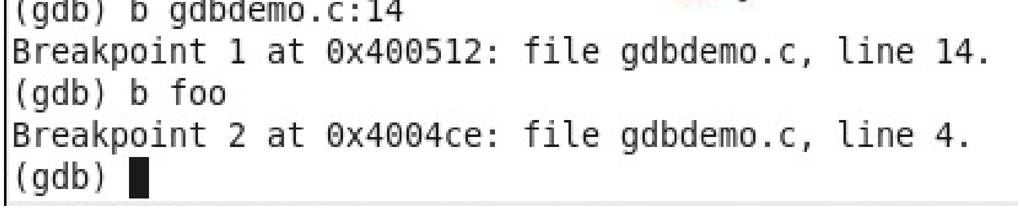
```
[ecegrid-thin4:~/ECE264] hegden$gdb gdbdemo
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-92.el6)
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/min/a/hegden/ECE264/gdbdemo...done.
(gdb)
```

# GDB – Set breakpoints

- Set breakpoints (b)
  - At line 14
  - Beginning of foo



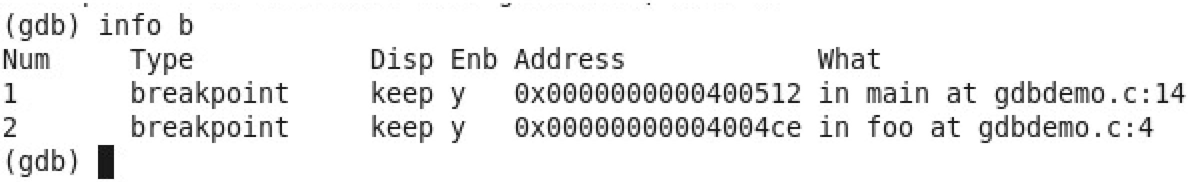
```
 1 #include<stdio.h>
 2 int foo(int a, int b)
 3 {
 4     int x = a + 1;
 5     int y = b + 2;
 6     int sum = x + y;
 7 
 8     return x * y + sum;
 9 }
10 
11 int main()
12 {
13    int ret = foo(10, 20);
14    printf("value returned from foo: %d\n",ret);
15    return 0;
16 }
```



```
(gdb) b gdbdemo.c:14
Breakpoint 1 at 0x400512: file gdbdemo.c, line 14.
(gdb) b foo
Breakpoint 2 at 0x4004ce: file gdbdemo.c, line 4.
(gdb) 
```

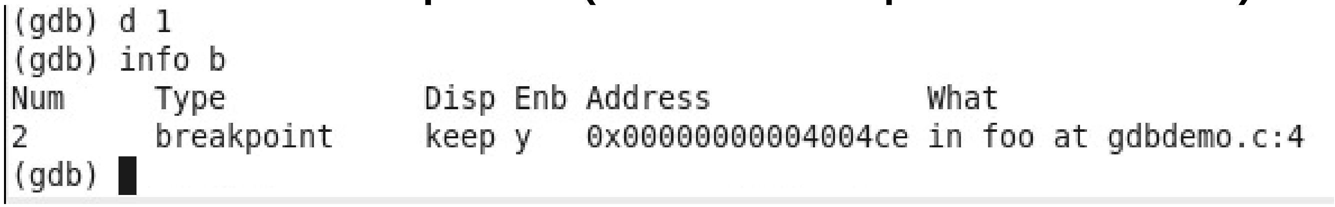
# GDB – Manage breakpoints

- Display all breakpoints set (`info b`)



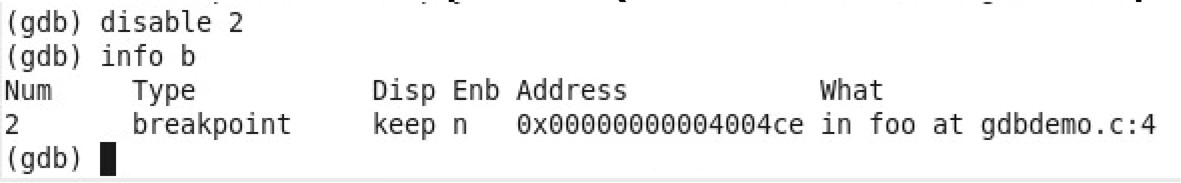
```
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000400512 in main at gdbdemo.c:14
2       breakpoint     keep y   0x00000000004004ce in foo at gdbdemo.c:4
(gdb) █
```

- Delete a breakpoint (`d <breakpoint num>`)



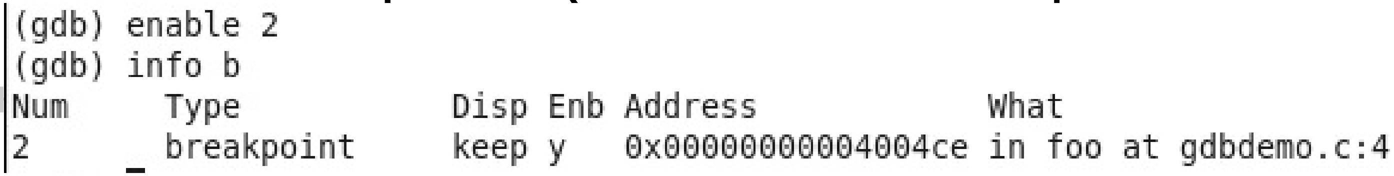
```
(gdb) d 1
(gdb) info b
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x00000000004004ce in foo at gdbdemo.c:4
(gdb) █
```

- Disable a breakpoint (`disable <breakpoint num>`)



```
(gdb) disable 2
(gdb) info b
Num     Type           Disp Enb Address            What
2       breakpoint     keep n   0x00000000004004ce in foo at gdbdemo.c:4
(gdb) █
```

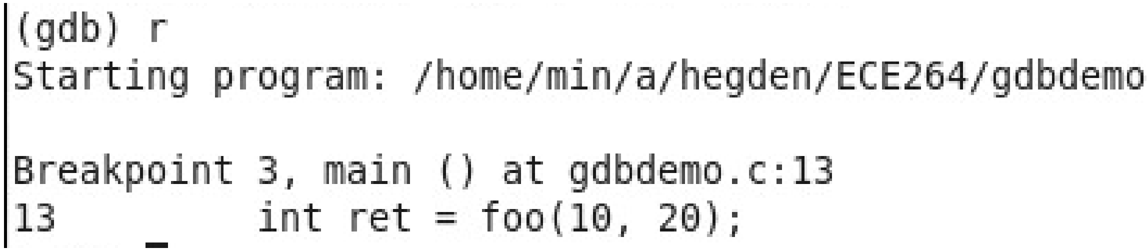
- Enable breakpoint (`enable <breakpoint num>`)



```
(gdb) enable 2
(gdb) info b
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x00000000004004ce in foo at gdbdemo.c:4
```

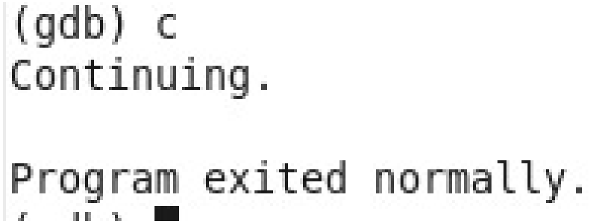
# GDB – Start execution

- Start execution (`r <command-line arguments>`)
  - Execution stops at the first breakpoint encountered



```
(gdb) r
Starting program: /home/min/a/hegden/ECE264/gdbdemo

Breakpoint 3, main () at gdbdemo.c:13
13          int ret = foo(10, 20);
```

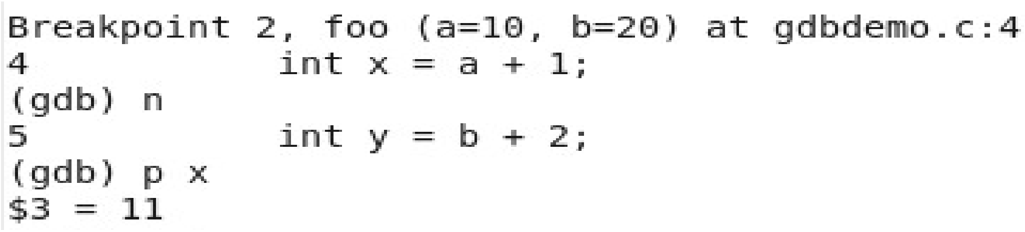
  - Continue execution (`c`)



```
(gdb) c
Continuing.

Program exited normally.
(gdb) █
```

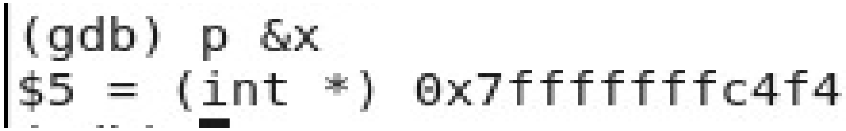
# GDB – Printing

– Printing variable values (`p <variable_name>`)



```
Breakpoint 2, foo (a=10, b=20) at gdbdemo.c:4
4           int x = a + 1;
(gdb) n
5           int y = b + 2;
(gdb) p x
$3 = 11
```

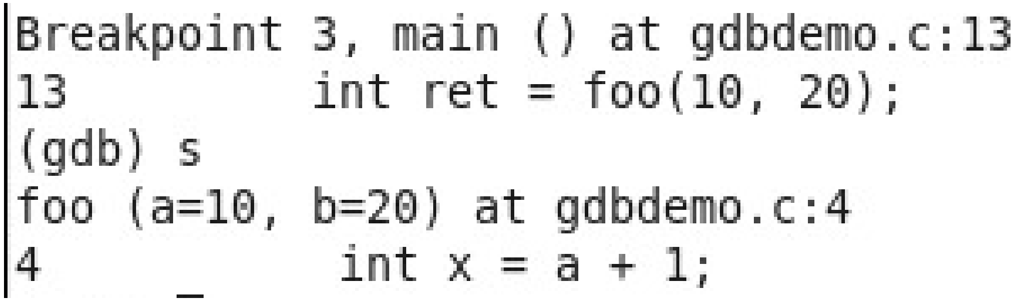
– Printing addresses (`p &<variable_name>`)



```
(gdb) p &x
$5 = (int *) 0x7fffffffc4f4
```

# GDB – Step in

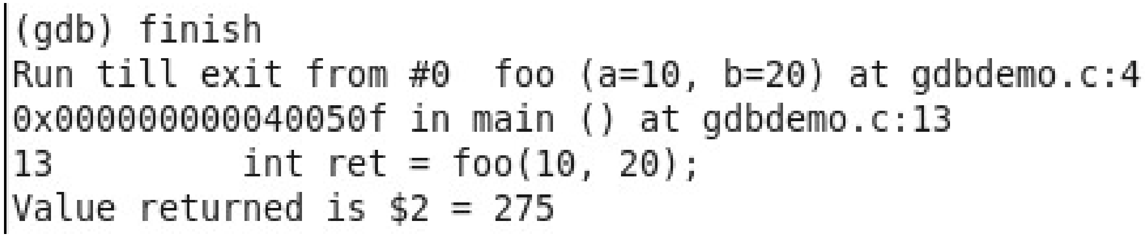
– Steps inside a function call (`s`)



```
Breakpoint 3, main () at gdbdemo.c:13
13          int ret = foo(10, 20);
(gdb) s
foo (a=10, b=20) at gdbdemo.c:4
4            int x = a + 1;
```

# GDB – Step out

– Jump to return address (`finish`)

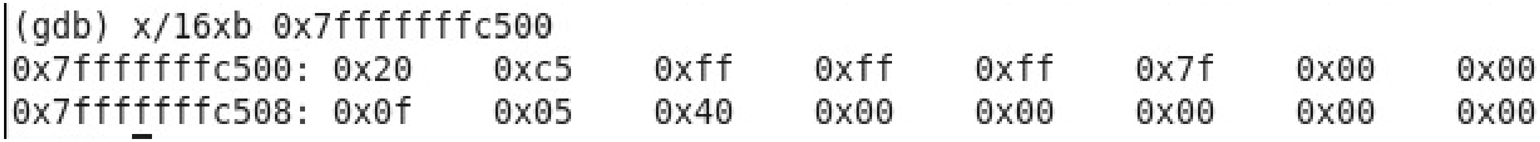


```
(gdb) finish
Run till exit from #0  foo (a=10, b=20) at gdbdemo.c:4
0x000000000040050f in main () at gdbdemo.c:13
13          int ret = foo(10, 20);
Value returned is $2 = 275
```

# GDB – Memory dump

– Printing memory content (`x/nfu <address>`)

- n = repetition (number of bytes to display)
- f = format (‘x’ – hexadecimal, ‘d’-decimal, etc.)
- u = unit (‘b’ – byte, ‘h’ – halfword/2 bytes, ‘w’ – word/4 bytes, ‘g’ – giga word/8 bytes)
- `E.g. x/16xb 0x7fffffffc500` (display the values of 16 bytes stored from starting address

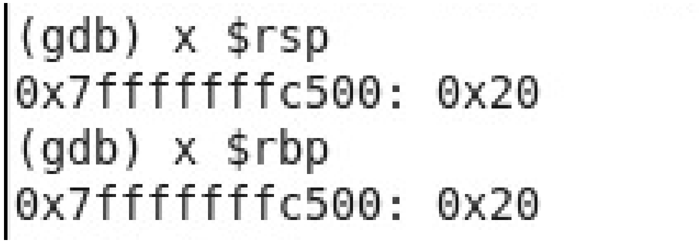


```
(gdb) x/16xb 0x7fffffffc500
0x7fffffffc500: 0x20    0xc5    0xff    0xff    0xff    0x7f    0x00    0x00
0x7fffffffc508: 0x0f    0x05    0x40    0x00    0x00    0x00    0x00    0x00
```

# GDB – Printing addresses

– Registers ($rsp, $rbp)

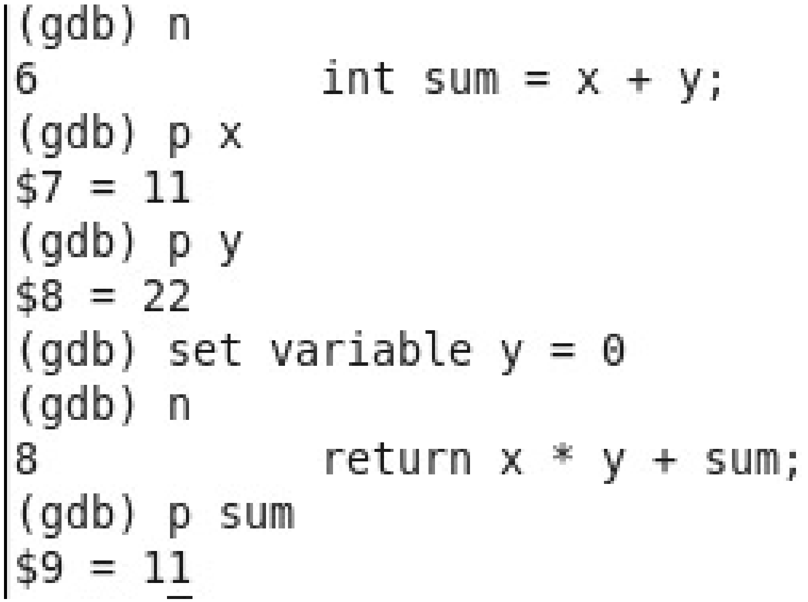
- Note that we use the ‘x’ command and not the ‘p’ command.



```
(gdb) x $rsp
0x7fffffffc500: 0x20
(gdb) x $rbp
0x7fffffffc500: 0x20
```

# GDB – Altering memory content

- Set command (`set variable <name> = value`)



```
(gdb) n
6           int sum = x + y;
(gdb) p x
$7 = 11
(gdb) p y
$8 = 22
(gdb) set variable y = 0
(gdb) n
8           return x * y + sum;
(gdb) p sum
$9 = 11
```

- Set command (`set *(<type *>addr) = value`)

# GDB Demo

- Refer to the demo example

# Doxygen

- Usage
  - Install Doxygen
  - Goto week7_codesamples
  - Create and edit a config file, `<Doxyfile>`, if required
  - Execute `doxygen <Doxyfile>`
    - Documentation corresponding to `matmulprof.cpp and memerrors` is automatically generated (in the doc folder)

# Matrix Data and Efficiency

- Sparse Matrices
  - E.g. banded matrices
  - Diagonal
  - Tridiagonal etc.
- Symmetric Matrices

$\xrightarrow{\textit{Admit optimizations w.r.t.}}$

- Storage
- Computation

# Sparse Matrices - Motivation

- Matrix Multiplication with Upper Triangular Matrices (C=C+AB)

$$\underset{A}{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix}} \underset{B}{\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ 0 & b_{22} & b_{23} \\ 0 & 0 & b_{33} \end{bmatrix}} =$$

$$\underset{A*B}{\begin{bmatrix} a_{11}b_{11} & a_{11}b_{12}+a_{12}\,b_{22} & a_{11}b_{13}+a_{12}\,b_{23} + a_{13}\,b_{13} \\ 0 & a_{22}b_{22} & a_{22}b_{23}+a_{23}\,b_{33} \\ 0 & 0 & a_{33}b_{33} \end{bmatrix}}$$

The result, A*B, is also upper triangular.

The non-zero elements appear to be like the result of *inner-product*

# Sparse Matrices - Motivation

- C=C+AB when A, B, C are upper triangular, pseudocode:

```
for i=
    for j=
        for k=
            C[i][j] = C[i][j] + A[i][k]*B[k][j]
```

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix}}_{A} * \underbrace{\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ 0 & b_{22} & b_{23} \\ 0 & 0 & b_{33} \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} a_{11}b_{11} & a_{11}b_{12}+a_{12}\, b_{22} & a_{11}b_{13}+a_{12}\, b_{23} + a_{13}\, b_{13} \\ 0 & a_{22}b_{22} & a_{22}b_{23}+a_{23}\, b_{33} \\ 0 & 0 & a_{33}b_{33} \end{bmatrix}}_{A*B}$$

# Sparse Matrices - Motivation

- C=C+AB when A, B, C are upper triangular, pseudocode:

```
for i=1 to N
    for j=i to N
        for k=i to j
            C[i][j] = C[i][j] + A[i][k]*B[k][j]
```

- Cost = $\Sigma_{i=1}^{N}\Sigma_{j=i}^{N}2(j-i+1)$ flops (why 2?)
- Using $\Sigma_{i=1}^{N}i \approx \frac{n^2}{2}$ and $\Sigma_{i=1}^{N}i^2 \approx \frac{n^3}{3}$
- $\Sigma_{i=1}^{N}\Sigma_{j=i}^{N}2(j-i+1) \approx \frac{n^3}{3}$, 1/3rd the number of flops required for dense matrix-matrix multiplication