CSE 410/510 Special Topics: Software Security

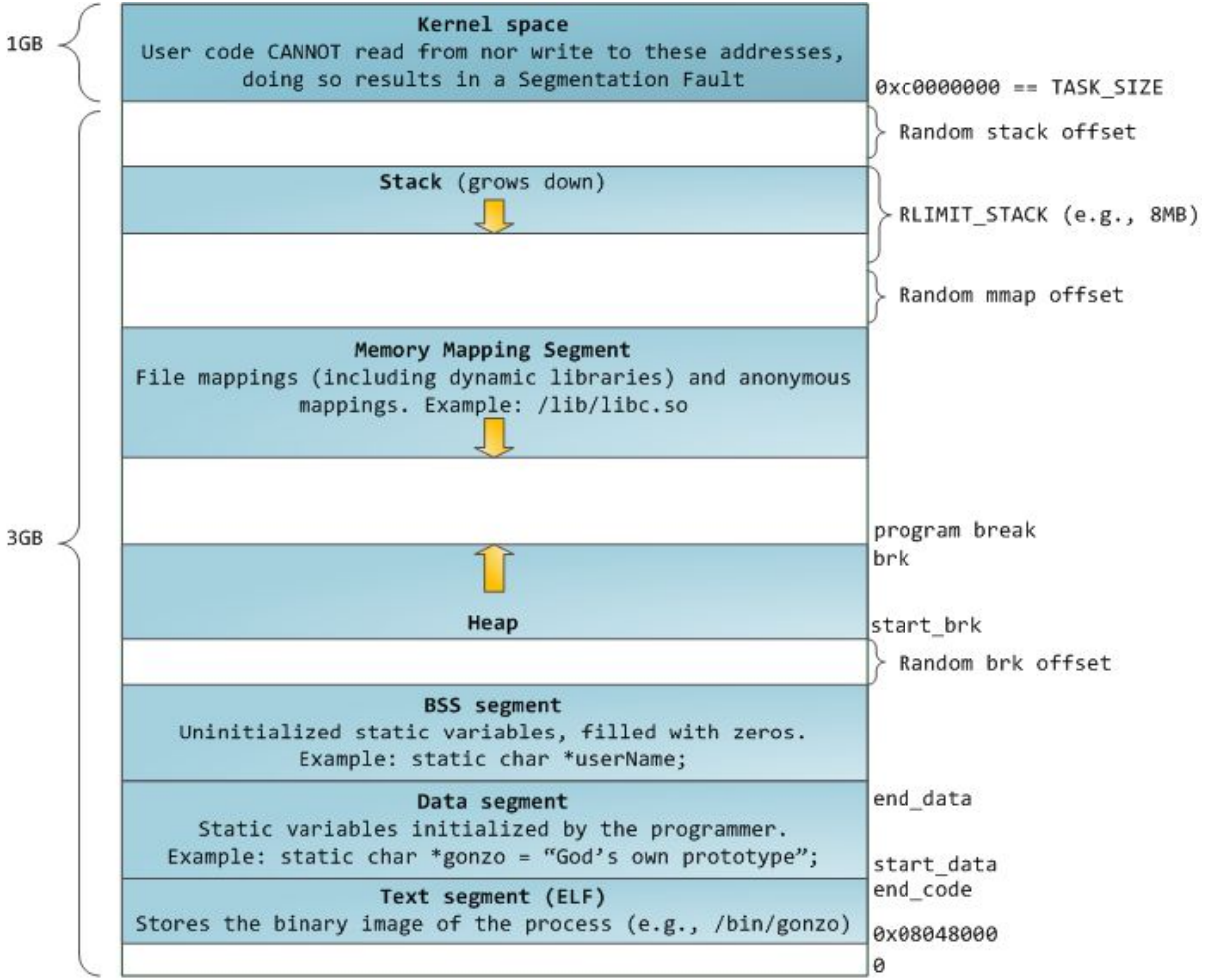
Instructor: Dr. Ziming Zhao

Location: Obrian 109
Time: Monday, Wednesday 5:00PM-6:20PM

Today

1. Heap and heap exploitation

Memory Map of Linux Process (32 bit system)



https://manybutfinite.com/post/anatomy-of-a-program-in-memory/

The Heap

The heap is pool of memory used for dynamic allocations at runtime

– **malloc**() grabs memory on the heap
– **free**() releases memory on the heap

Both are standard C library interfaces. Neither of them directly mapps to a system call.

Malloc and Free Prototype

```
void* malloc(size_t size);
```

Allocates size bytes of uninitialized storage. If allocation succeeds, returns a pointer that is suitably aligned for any object type with fundamental alignment.

```
void free(void* ptr);
```

Deallocates the space previously allocated by malloc(), etc.

How to use malloc() and free()

```
int main()
{
        char * buffer = NULL;

        /* allocate a 0x100 byte buffer */
        buffer = malloc(0x100);

        /* read input and print it */
        fgets(stdin, buffer, 0x100);
        printf("Hello %s!\n", buffer);

        /* destroy our dynamically allocated buffer */
        free(buffer);
        return 0;
}
```

Heap vs. Stack

Heap

- Dynamic memory allocations at runtime
- Objects, big buffers, structs, persistence, larger things

Slower, Manual
– Done by the programmer
– malloc/calloc/recalloc/free
– new/delete

Stack

- Fixed memory allocations known at compile time
- Local variables, return addresses, function args

Fast, Automatic; Done by the compiler
– Abstracts away any concept of allocating/de-allocating

Heap Implementations

dlmalloc. Default native version of malloc in some old distributions of Linux (http://gee.cs.oswego.edu/dl/html/malloc.html)

ptmalloc. ptmalloc is based on dlmalloc and was extended for use with multiple threads. On Linux systems, ptmalloc has been put to work for years as part of the GNU C library.

tcmalloc. Google's customized implementation of C's malloc() and C++'s operator new (https://github.com/google/tcmalloc)

jemalloc. jemalloc is a general purpose malloc(3) implementation that emphasizes fragmentation avoidance and scalable concurrency support.

The **Hoard** memory allocator. UMass Amherst CS Professor Emery Berger

Which implementation on my laptop?

ldd --version

GLIBC 2.31

Ptmalloc2

https://elixir.bootlin.com/glibc/glibc-2.31/source/malloc/malloc.c

```
➜ heapfrees ldd --version
ldd (Ubuntu GLIBC 2.31-0ubuntu9.2) 2.31
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.
```

Malloc Trivia

How many bytes on the heap are your ***malloc chunks*** really taking up?

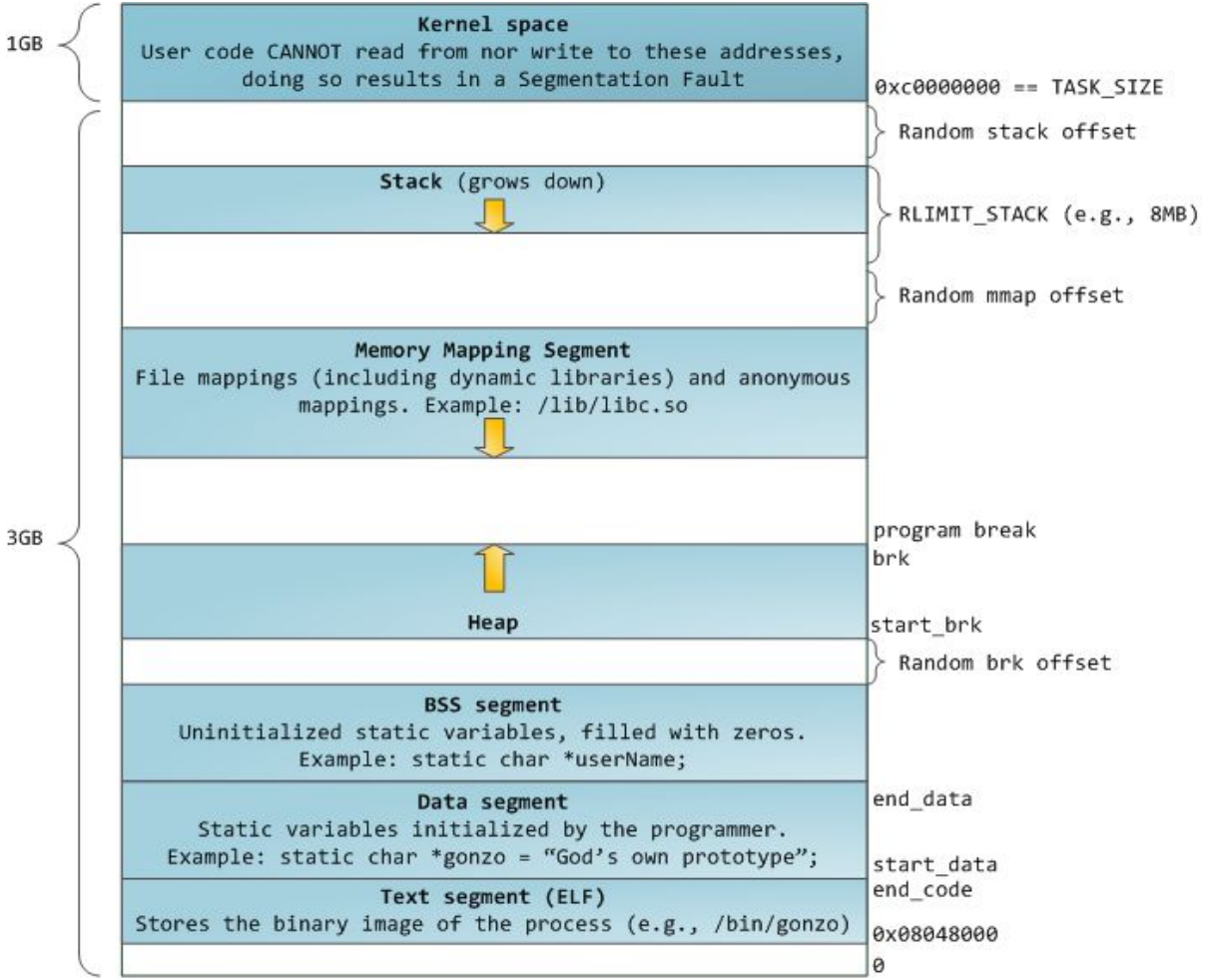
- malloc(32);
- malloc(4);
- malloc(20);
- malloc(0);

code/heapsizes

```
int main()
{
   unsigned int lengths[] = {32, 4, 20, 0, 64, 32, 32, 32, 32, 32};
   unsigned int * ptr[10];
   int i;

   for(i = 0; i < 10; i++)
      ptr[i] = malloc(lengths[i]);

   for(i = 0; i < 9; i++)
      printf("malloc(%2d) is at 0x%08x, %3d bytes to the next pointer\n",
            lengths[i],
            (unsigned int)ptr[i],
            (ptr[i+1]-ptr[i])*sizeof(unsigned int));

  return 0;}
```

https://github.com/RPISEC/MBE/blob/master/src/lecture/heap/sizes.c

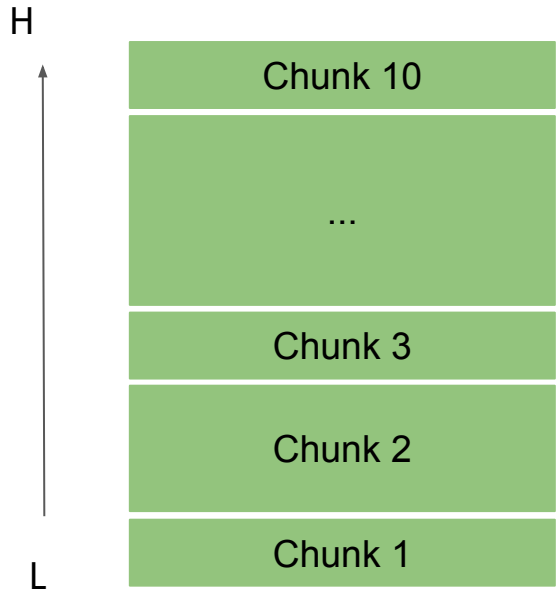
Heap goes from low address to high address



https://manybutfinite.com/post/anatomy-of-a-program-in-memory/

code/heapsizes

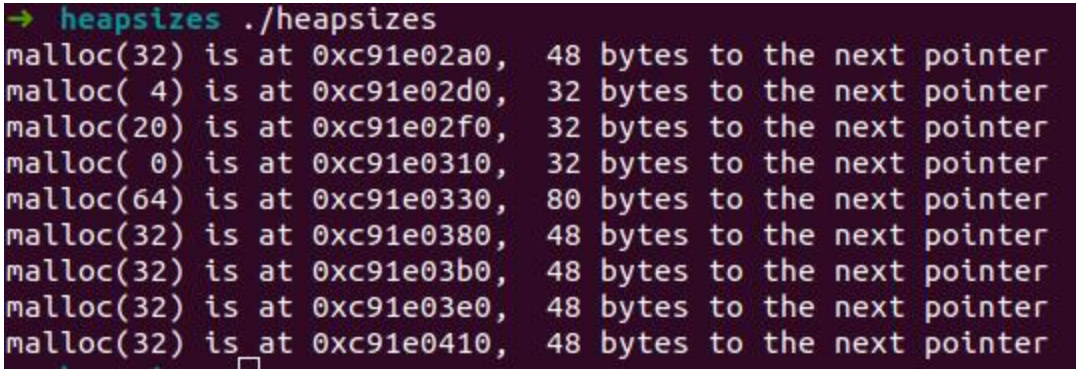
```
int main()
{
   unsigned int lengths[] = {32, 4, 20, 0, 64, 32, 32, 32, 32, 32};
   unsigned int * ptr[10];
   int i;

   for(i = 0; i < 10; i++)
      ptr[i] = malloc(lengths[i]);

   for(i = 0; i < 9; i++)
      printf("malloc(%2d) is at 0x%08x, %3d bytes to the next pointer\n",
            lengths[i],
            (unsigned int)ptr[i],
            (ptr[i+1]-ptr[i])*sizeof(unsigned int));

  return 0;}
```



code/heapsizes 32bit

```
→ heapsizes ./heapsizes32
malloc(32) is at 0x5695b1a0,  48 bytes to the next pointer
malloc( 4) is at 0x5695b1d0,  16 bytes to the next pointer
malloc(20) is at 0x5695b1e0,  32 bytes to the next pointer
malloc( 0) is at 0x5695b200,  16 bytes to the next pointer
malloc(64) is at 0x5695b210,  80 bytes to the next pointer
malloc(32) is at 0x5695b260,  48 bytes to the next pointer
malloc(32) is at 0x5695b290,  48 bytes to the next pointer
malloc(32) is at 0x5695b2c0,  48 bytes to the next pointer
malloc(32) is at 0x5695b2f0,  48 bytes to the next pointer
```

code/heapsizes 64bit



Malloc Trivia

How many bytes on the heap are your ***malloc chunks*** really taking up?

- malloc(32); 48 bytes (32bit/64bit)
- malloc(4); 16 bytes (32bit) / 32 bytes (64bit)
- malloc(20); 32 bytes (32bit/64bit)
- malloc(0); 16 bytes (32bit) / 32 bytes (64bit)

Malloc_chunk (ptmalloc2 in glibc2.31)

```
struct malloc_chunk {

  INTERNAL_SIZE_T      mchunk_prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      mchunk_size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;         /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```

INTERNAL_SIZE_T is the same as size_t. 8 bytes in 64 bit;
4 bytes in 32 bits machine.
Pointer is 8/4 bytes on a 64/32 bit machine, respectively.

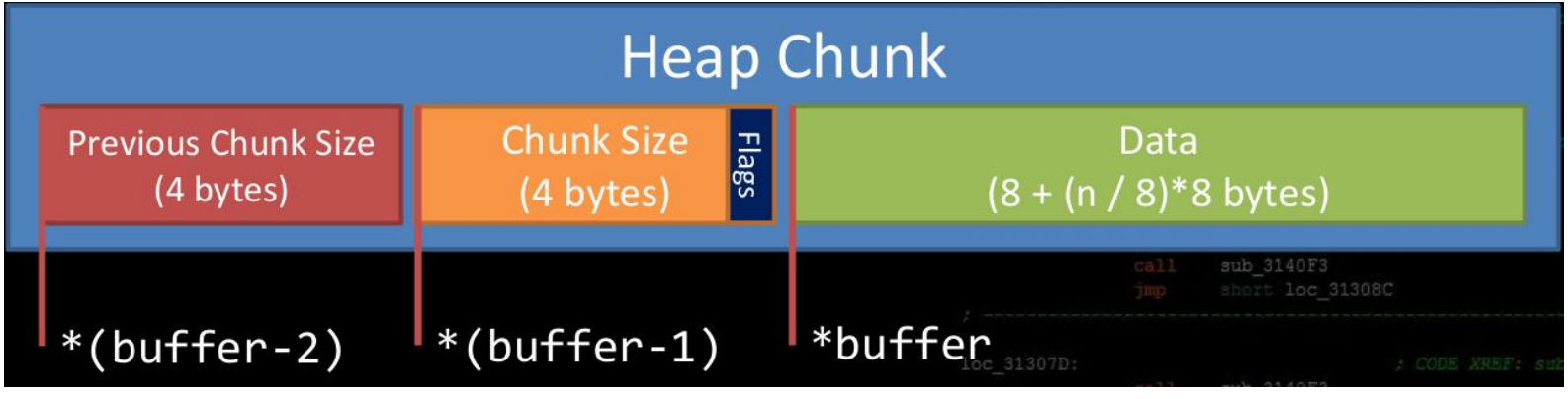
https://elixir.bootlin.com/glibc/glibc-2.31/source/malloc/malloc.c

Heap Chunks (figures in 32 bit)

buffer = malloc(0x100);

//Out comes a heap chunk

Previous Chunk Size: Size of previous chunk (if prev chunk is free)
Chunk Size: Size of entire chunk including overhead
Data: Your newly allocated memory / ptr returned by malloc
Flags: Because of byte alignment, the lower 3 bits of the chunk size field would always be zero. Instead they are used for flag bits.
0x01 PREV_INUSE – set when previous chunk is in use
0x02 IS_MMAPPED – set if chunk was obtained with mmap()
0x04 NON_MAIN_ARENA – set if chunk belongs to a thread arena



code/heapchunks

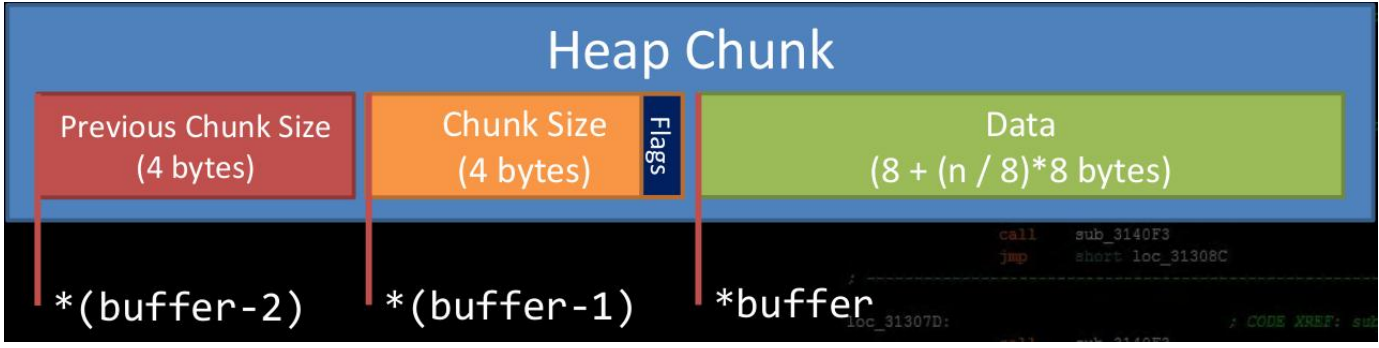
```
void print_chunk(size_t * ptr, unsigned int len)
{
  printf("[ prev - 0x%08x ][ size - 0x%08x ][ data buffer (0x%08x) -------> ... ] - from
malloc(%d)\n", *(ptr-2), *(ptr-1), (unsigned int)ptr, len);}

int main()
{
   void * ptr[LEN];
   unsigned int lengths[] = {0, 4, 8, 16, 24, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384};
   int i;

   printf("mallocing...\n");

   for(i = 0; i < LEN; i++)
      ptr[i] = malloc(lengths[i]);

   for(i = 0; i < LEN; i++)
      print_chunk(ptr[i], lengths[i]);
   return 0;}
```

Extended from https://github.com/RPISEC/MBE/blob/master/src/lecture/heap/heap_chunks.c

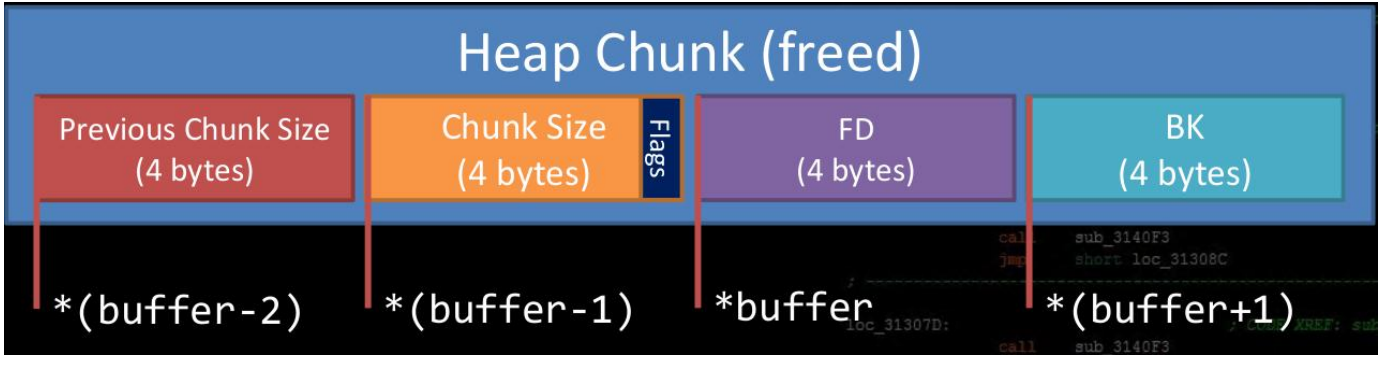
```
➜ heapchunks ./heapchunks32
mallocing...
[ prev - 0x00000000 ][ size - 0x00000011 ][ data buffer (0x57b665b0) -------> ... ] - from malloc(0)
[ prev - 0x00000000 ][ size - 0x00000011 ][ data buffer (0x57b665c0) -------> ... ] - from malloc(4)
[ prev - 0x00000000 ][ size - 0x00000011 ][ data buffer (0x57b665d0) -------> ... ] - from malloc(8)
[ prev - 0x00000000 ][ size - 0x00000021 ][ data buffer (0x57b665e0) -------> ... ] - from malloc(16)
[ prev - 0x00000000 ][ size - 0x00000021 ][ data buffer (0x57b66600) -------> ... ] - from malloc(24)
[ prev - 0x00000000 ][ size - 0x00000031 ][ data buffer (0x57b66620) -------> ... ] - from malloc(32)
[ prev - 0x00000000 ][ size - 0x00000051 ][ data buffer (0x57b66650) -------> ... ] - from malloc(64)
[ prev - 0x00000000 ][ size - 0x00000091 ][ data buffer (0x57b666a0) -------> ... ] - from malloc(128)
[ prev - 0x00000000 ][ size - 0x00000111 ][ data buffer (0x57b66730) -------> ... ] - from malloc(256)
[ prev - 0x00000000 ][ size - 0x00000211 ][ data buffer (0x57b66840) -------> ... ] - from malloc(512)
[ prev - 0x00000000 ][ size - 0x00000411 ][ data buffer (0x57b66a50) -------> ... ] - from malloc(1024)
[ prev - 0x00000000 ][ size - 0x00000811 ][ data buffer (0x57b66e60) -------> ... ] - from malloc(2048)
[ prev - 0x00000000 ][ size - 0x00001011 ][ data buffer (0x57b67670) -------> ... ] - from malloc(4096)
[ prev - 0x00000000 ][ size - 0x00002011 ][ data buffer (0x57b68680) -------> ... ] - from malloc(8192)
[ prev - 0x00000000 ][ size - 0x00004011 ][ data buffer (0x57b6a690) -------> ... ] - from malloc(16384)
➜ heapchunks ./heapchunks
mallocing...
[ prev - 0x00000000 ][ size - 0x00000021 ][ data buffer (0x665046b0) -------> ... ] - from malloc(0)
[ prev - 0x00000000 ][ size - 0x00000021 ][ data buffer (0x665046d0) -------> ... ] - from malloc(4)
[ prev - 0x00000000 ][ size - 0x00000021 ][ data buffer (0x665046f0) -------> ... ] - from malloc(8)
[ prev - 0x00000000 ][ size - 0x00000021 ][ data buffer (0x66504710) -------> ... ] - from malloc(16)
[ prev - 0x00000000 ][ size - 0x00000021 ][ data buffer (0x66504730) -------> ... ] - from malloc(24)
[ prev - 0x00000000 ][ size - 0x00000031 ][ data buffer (0x66504750) -------> ... ] - from malloc(32)
[ prev - 0x00000000 ][ size - 0x00000051 ][ data buffer (0x66504780) -------> ... ] - from malloc(64)
[ prev - 0x00000000 ][ size - 0x00000091 ][ data buffer (0x665047d0) -------> ... ] - from malloc(128)
[ prev - 0x00000000 ][ size - 0x00000111 ][ data buffer (0x66504860) -------> ... ] - from malloc(256)
[ prev - 0x00000000 ][ size - 0x00000211 ][ data buffer (0x66504970) -------> ... ] - from malloc(512)
[ prev - 0x00000000 ][ size - 0x00000411 ][ data buffer (0x66504b80) -------> ... ] - from malloc(1024)
[ prev - 0x00000000 ][ size - 0x00000811 ][ data buffer (0x66504f90) -------> ... ] - from malloc(2048)
[ prev - 0x00000000 ][ size - 0x00001011 ][ data buffer (0x665057a0) -------> ... ] - from malloc(4096)
[ prev - 0x00000000 ][ size - 0x00002011 ][ data buffer (0x665067b0) -------> ... ] - from malloc(8192)
[ prev - 0x00000000 ][ size - 0x00004011 ][ data buffer (0x665087c0) -------> ... ] - from malloc(16384)
```

Heap Chunks – Two states (figures in 32 bit)

Heap chunks exist in two states
– in use (malloc'd)



– free'd.
Forward Pointer: A pointer to the next freed chunk
Backwards Pointer: A pointer to the previous freed chunk
Implementation-defined.



code/heapfrees

```
void print_inuse_chunk(unsigned int * ptr)
{
	printf("[ prev - 0x%08x ][ size - 0x%08x ][ data buffer
(0x%08x) ----> ... ] - Chunk 0x%08x - In use\n", \
		*(ptr-2),
		*(ptr-1),
		(unsigned int)ptr,
		(unsigned int)(ptr-2));
}

void print_freed_chunk(unsigned int * ptr)
{
	printf("[ prev - 0x%08x ][ size - 0x%08x ][ fd - 0x%08x ][ bk -
0x%08x ] - Chunk 0x%08x - Freed\n", \
		*(ptr-2),
		*(ptr-1),
		*ptr,
		*(ptr+1),
		(unsigned int)(ptr-2));
}
```

```
int main()
{
   unsigned int * ptr[LEN];
   unsigned int lengths[] = {32, 32, 32, 32, 32}; int i;

   printf("mallocing...\n");
   for(i = 0; i < LEN; i++)
      ptr[i] = malloc(lengths[i]);

   for(i = 0; i < LEN; i++)
      print_inuse_chunk(ptr[i]);

   printf("\nfreeing all chunks...\n");
   for(i = 0; i < LEN; i++)
      free(ptr[i]);

   for(i = 0; i < LEN; i++)
      print_freed_chunk(ptr[i]);

   return 0;}
```