

Richard Litt

Scan QR code to add me as a friend.

Week 9 Lecture

XMUT-NWEN 241 - 2023 T2

Systems Programming

Richard Littauer

School of Engineering and Computer Science

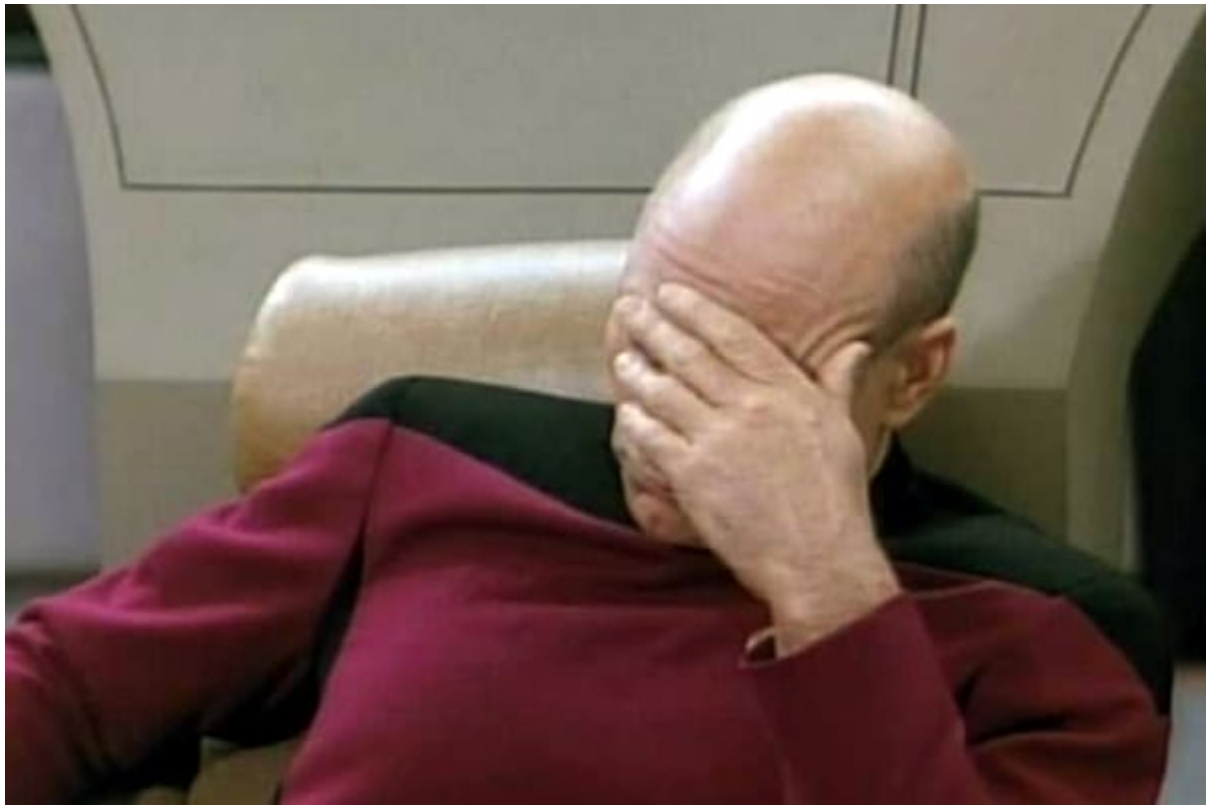
Victoria University of Wellington

richard.littauer@vuw.ac.nz

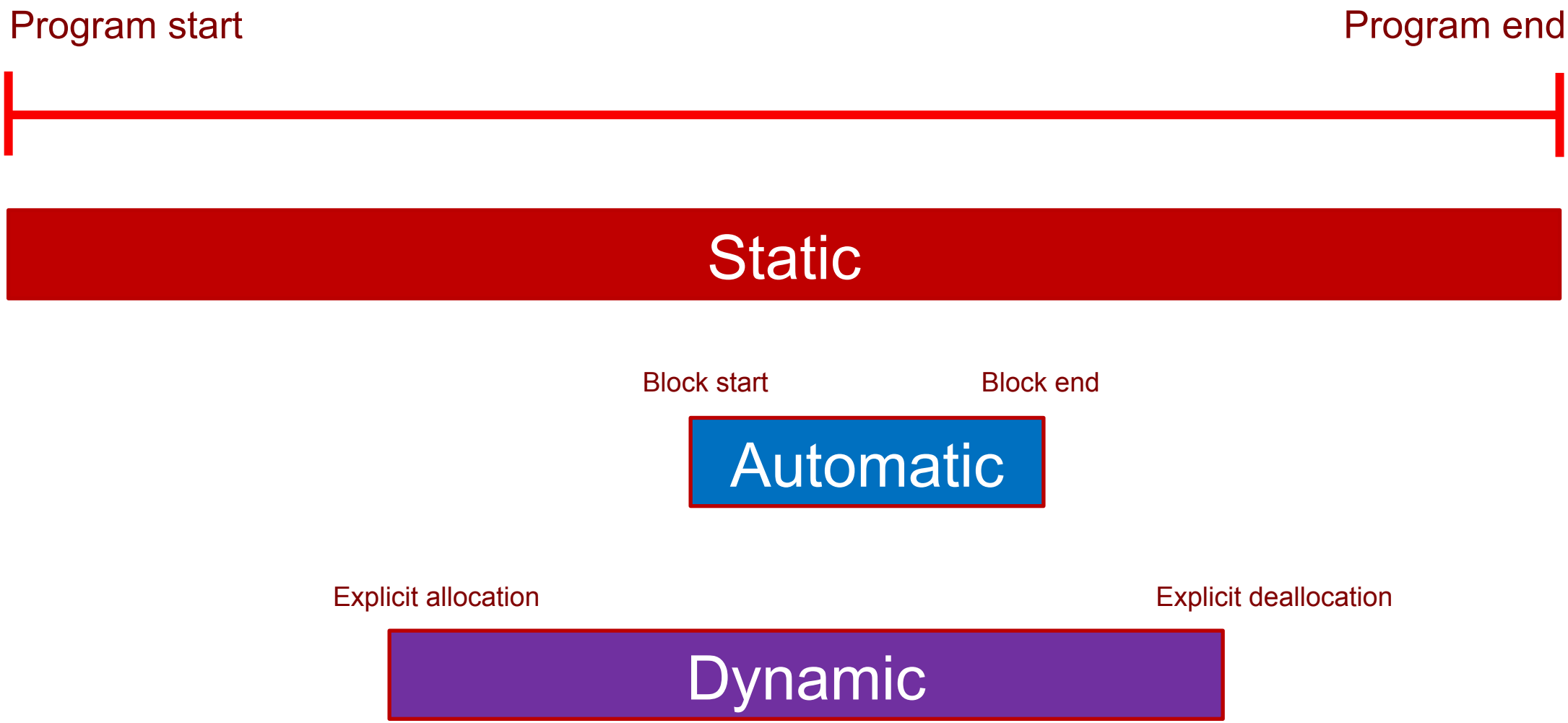
Questions from last time

- How do you look up the % strings without memorizing them?
  - $ man 3 printf
- How do you parse difficult expressions?
  - https://cdecl.org/
- Does realloc(pointer, 0) basically do the same as free()?
  - No; more on this later.

Common Issues With Dynamic Memory



Lifetime


Program start
Program end
Static
Block start
Block end
Automatic
Explicit allocation
Explicit deallocation
Dynamic

Questions

- What happens to an OS, a phone, or a browser if you don't call `free`?
- Java and Python never make you call `free`. What do they do instead, and what does that cost?
- What frees stack memory, and what frees heap memory?
- Why is "it ran fine" not proof that memory code is correct?

Questions

- Which things live on the stack and which on the heap?

```
{
   int a;
   int *p = malloc(sizeof(int));
}
```

Issue #1

- Returning a pointer to an automatic variable

```
int *foo(void)
{
      int x;

      …

      return &x;
      /* x does not exist outside the function */
      /* Returning its address will result in unknown
behaviour */
}
```

Issue #1

- What happens if you run this?

```
int *foo(void)
{
  int x; x = 42; return &x;
}

int main(void)
{
  int *p;
  p = foo();
  printf("%d\n", *p);
  return 0;
}
```

- Will it print 42, a garbage number, or crash?
- Is it OK to `return x;` (the value) when it isn't OK to `return &x;`? Why?

Issue #1

- What if we reset *p afterwards in the code?

```
int *foo(void)
{
  int x;
  x = 42;
  return &x;
}

void bar(void)
{
  volatile int y;
  y = 999;
}
```

```
int main(void)
{
  int *p;
  int before;
  int after;
  p = foo();
  before = *p;
  bar();
  after = *p;
  printf("before bar(): %d\n",
before);
  printf("after bar():  %d\n",
after);
  return 0;
}
```

Issue #1: Fixes

- Store what you want in the heap

```
int *foo_heap(void)
{
  int *x = (int *) malloc(sizeof(int));

  if (x == NULL)
    return NULL;

  *x = 42;
  return x;
}
```

Issue #1: Fixes

- Make it static

```
int *foo_static(void)
{
  static int x;

  x = 42;
  return &x;
}
```

Issue #1: Fixes

- Have the caller control memory

```
void foo_fill(int *out)
{
  *out = 42;
}
```

Issue #2

- Heap block overrun: similar to array going out of bounds

```
void foo(void)
{
     int *x = (int *) malloc(10 * sizeof(int));

     x[10] = 10;
     /* Allocated memory is only up to x[9] */

     …

     free(x);
}
```

Issue #2

- Heap block overrun: similar to array going out of bounds

```
#define N 16

int main(void)
{
  int *a = (int *) malloc(N * sizeof(int));
  int *b = (int *) malloc(N * sizeof(int));
  int i;

  if (a == NULL || b == NULL)
    return 1;

  b[0] = 7;
  printf("a starts at %p\n", (void *) a);
  printf("b starts at %p\n", (void *) b);
  printf("b[0] before: %d\n", b[0]);
```

```
for (i = 0; i <= N; i++)
    a[i] = -1;

  printf("b[0] after:  %d\n", b[0]);

  free(a);
  free(b);
  return 0;
}
```

Issue #2

- What's the bug here?

```
int main(void)
{
  int *x = (int *) malloc(10);
  int i;

  if (x == NULL)
    return 1;

  for (i = 0; i < 10; i++)
    x[i] = i;

  printf("x[9] = %d\n", x[9]);

  free(x);
  return 0;
}
```

Issue #2

- Fixes:

- Allocate `N * sizeof(*x)` bytes. `sizeof(*x)` means "the size of whatever x points to", so it stays correct even if x's type changes later.
- Check that `malloc` did not return `NULL`.
- Loop with `i < N`, and use the same N for the allocation and the loop.

Issue #2

- Fixes:

```
int main(void)
{
  int *x = malloc(N * sizeof(*x));
  int i;

  if (x == NULL) {
    fprintf(stderr, "out of memory\n");
    return 1;
  }

  /* over there ---> */
```

```
  for (i = 0; i < N; i++)
    x[i] = i * i;

  printf("last valid element x[%d] =
%d\n", N - 1, x[N - 1]);

  free(x);

  return 0;
}
```

Questions

```
int *x = malloc(10);
for (i = 1; i <= 10; i++)
   x[i] = 0;
```

- How many bugs are in this loop?
- Java throws an exception for an out-of-bounds index. Why doesn't C check?
- How many bytes does malloc(40 * sizeof(int)) allocate on a typical machine?
- Why prefer `sizeof(*x)` over `sizeof(int)`?

Issue #3

- Memory leak: loss of pointer to allocated memory

```
int *pi;

void foo(void)
{
      pi = (int*) malloc(8*sizeof(int));
      /* Leaked the old memory pointed to by pi */
      …
      free(pi); /* foo() is done with pi, so free it */
}

int main(void)
{
      pi = (int*) malloc(4*sizeof(int));
      foo();
}
```

Issue #3

- Fix: Make the block have its own local pointer

```
void foo(void)
{
  int *mine = (int *)
malloc(8 * sizeof(int));
/* foo's own block */

  if (mine == NULL)
    return;
  /* … */
  free(mine);
}
```

```
int main(void)
{
  int *pi = (int *) malloc(4 *
sizeof(int));

  if (pi == NULL)
    return 1;

  foo();

  free(pi); /* every malloc has exactly
one matching free */
  return 0;
}
```

Questions

- Does the OS get the leaked memory back eventually?
- A server leaks 1 KB on every request and handles 100 requests per second. How long until it has leaked 1 GB?
- In this code, which block is leaked: the 4-int one or the 8-int one?

```
void foo(void)
{
      pi = (int*) malloc(8*sizeof(int));     free(pi);
}
int main(void)
{
      pi = (int*) malloc(4*sizeof(int));     foo();
}
```

Issue #4

- Potential memory leak
  - Loss of pointer to beginning of memory block
  - May still recover through pointer arithmetic

```
int *ip = NULL;

void foo(void)
{
     ip = (int *) malloc(2 * sizeof(int));
     …
     ip++;
     /* ip is not pointing to the start of the block
anymore */
}
```

Questions

- `ip` is an `int *` holding `0x1000`, pointing at a block of 2 ints. What is `ip + 1`? What is `ip + 2`, and can you read `*(ip + 2)`?
- Write two ways to read the second int in the block without changing `ip`.
- Why does C allow you to do `int++` at all, if it causes memory leaks?

Issue #5

- Freeing non-heap or unallocated memory

```
void foo(void)
{
    int fnh = 0;
    free(&fnh); /* Freeing stack memory */
}

void bar(void)
{
    int *fum = (int *) malloc(4 * sizeof(int));
    free(fum+1); /* fum+1 points to middle of block */
    free(fum);
    free(fum); /* Freeing already freed memory */
}
```

Issue #6

- Erasing realloc's pointer

```
p = realloc(p, n); /* leaks if realloc fails */

/* Fix */

void *tmp = realloc(p, n);
if (tmp == NULL) {
    /* handle error; p is still valid */
} else {
    p = tmp;
}
```

Earlier question

- What does this do?

```
realloc(p, 0);
```

- Nothing good. In C89 it worked, but since C99 it has basically been undefined behavior or implementation-defined.

Detecting Memory Leaks and Other Issues

Valgrind

- Valgrind is an open-source tool for detecting memory management and threading bugs
- For more information: http://valgrind.org/

Review

- What is wrong with this function? Give two different fixes.

```
char *greeting(void)
{
  char msg[] = "hello";
  return msg;
}
```

- Is this function safe?

```
int *same(int *p)
{
  return p;
}
```

Review

- Where is the bug in this?

```
int *a = malloc(5 * sizeof(int));
for (i = 0; i < 5; i++)
  a[i + 1] = i;
```

- How many bytes does `strcpy` write here, and how many were allocated?

```
char *s = malloc(3);
strcpy(s, "abc");
```

Review

- How many blocks are leaked here?

```
char *p;
for (i = 0; i < 3; i++)
  p = malloc(100);
free(p);
```

- Does this program leak? Where?

```
int *make(void)
{
  return malloc(sizeof(int));
}

int main(void)
{
  make();
  return 0;
}
```

Review

- `p` is an `int *` holding address `0x2000`. What address does it hold after `p += 3`?

Review

- Which of these free calls is valid?

```
int a[4];
int *b = malloc(4 *
sizeof(int));
int *c = b;
int *d = NULL;

free(a);
free(b);
free(c);
free(d);
```

- Add one line that makes the second free harmless.

```
int *p = malloc(sizeof(int));
free(p);
/* your line here */
free(p);
```

Review

- What doe sthis print?

```
int *p =
malloc(sizeof(int));
*p = 5;
free(p);
int *q =
malloc(sizeof(int));
*q = 7;
printf("%d\n", *p);
```

Review

- What is wrong, and which function would fix it?

```
int *a = malloc(3 * sizeof(int));
a[0] = 1;
printf("%d\n", a[0] + a[1]);
```

Review

- Fix this line:

```
long *v = malloc(n * sizeof(v));
```

- What can go wrong here? Write a safe version.

```
p = realloc(p, 2 * n * sizeof(int));
```

Review

- Safe:

```
int *tmp = realloc(p, 2 * n * sizeof(*p));
if (tmp == NULL) {
  /* handle the error; p is still valid */
} else {
  p = tmp;
}
```