601.220 Intermediate Programming

Spring 2023, Day 16 (Feb 27th)

Today's agenda

- Exercise 15 review
- Linked lists
- Exercise 16

Reminders/Announcements

- HW4: due Friday (Mar 3rd) no late subs
- Midterm project
  - Will be posted on course website Wednesday (Mar 1st)
  - Due Friday, Mar 17th (no late submissions)
  - Expect an email soon if you have not registered a team (you will be assigned a partner)
- Midterm exam
  - In class Friday, March 10th
  - Review materials are available on course website

Exercise 15 review

```
(gdb) break endian.c:21
Breakpoint 1 at 0x1243: file endian.c, line 21.
(gdb) run
[...output omitted...]
Breakpoint 1, main () at endian.c:21
21    printf("%u\n", *p);
(gdb) print/x ((unsigned char *)p)[0]
$1 = 0x83
(gdb) print/x ((unsigned char *)p)[1]
$2 = 0x7e
(gdb) print/x ((unsigned char *)p)[2]
$3 = 0xa3
(gdb) print/x ((unsigned char *)p)[3]
$4 = 0x38
```

In base-16, 950238851 is 38A37E83. Since we're seeing the bytes in order from least to most significant, the ugrad machines are *little endian*.

Exercise 15 review

To negate a two's complement value:

- Invert all of the bits (the ~ operator is useful for this)
- Add 1

Exercise 15 review

(1U << 31)

Note that 0x80000000U is the unsigned int value with only the most significant bit set to 1. This is the sign bit, and values with this bit set are negative.

```
unsigned int magnitude(unsigned int value) {
  if ((value & 0x80000000U) == 0U) {
    return value; // value is non-negative
  }

  // value is negative, so invert bits and add 1
  value = ~value;  // invert bits
  value += 1U;     // add 1
  return value;
}
```

Exercise 15 review

Generating a uniformly distributed pseudo-random integer in the range 0 (inclusive) to `max_num` (exclusive):

```
int gen_uniform(int max_num) {
  return rand() % max_num;
}
```

Generating 500 random values in range 0 (inclusive) to `max_range` (exclusive) and tallying them in the `hist` array:

```
for (int i = 0; i < 500; i++) {
  hist[gen_uniform(max_range)]++;
}
```

Exercise 15 review

Generating normally-distributed integer values in the range 0 (inclusive) to `max_range` (exclusive):

```c
int normal_rand(int max_num) {
  int result = 0;
  for (int i = 1; i < max_num; i++) {
    if ((rand() & 1) == 1) {
      result++;
    }
  }
  return result;
}
```

This is basically flipping a coin `max_num-1` times and counting how many times it's heads.

Exercise 15 review

Generating 500 normally-distributed values in the range 0 (inclusive) to `max_range` (exclusive) and tallying them in the `hist` array:

```
for (int i = 0; i < 500; i++) {
  hist[normal_rand(max_range)]++;
}
```

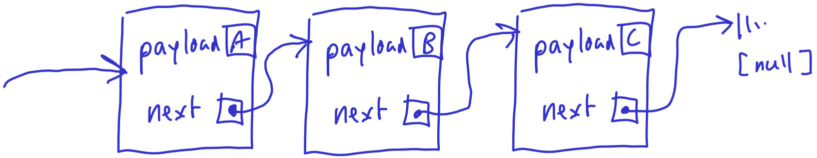
Day 17 recap questions

1. Describe the linked list structure by a diagram.
2. Compare arrays and linked lists. Write down their pros and cons.
3. What is a linked list's head? How is it different from a node? Explain.
4. How do you calculate `length` of a linked list?
5. How do you implement `add_after` on a singly linked list?

1. Describe the linked list structure by a diagram.

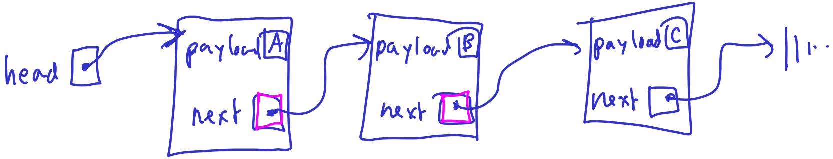
`struct Node` type:

```
struct Node {
  char payload; // payload could be any data type
  struct Node *next;
};
```

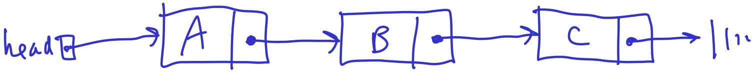


Example linked list

```
// code creating a linked list
struct Node *head = malloc(sizeof(struct Node));
head->payload = 'A';
head->next = malloc(sizeof(struct Node));
head->next->payload = 'B';
head->next->next = malloc(sizeof(struct Node));
head->next->next->payload = 'C';
head->next->next->next = NULL;
```



A more concise representation



2. Compare $\underbrace{\text{arrays and linked lists}}_{\text{Sequences}}$. Write down their pros and cons.

Arrays:

- Pro: O(1) access to arbitrary element "random access"
- Con: O(N) to insert or remove element at arbitrary position
- Pro: better locality (fewer cache misses when iterating)
- Pro: more compact
- Con: fixed size, to reallocate must allocate new array and copy existing data

Linked list pros and cons

Linked list:

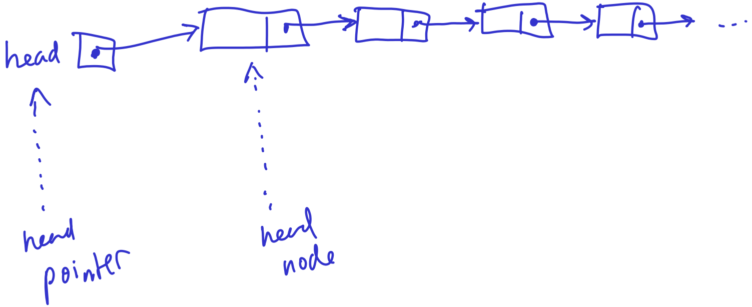
"sequential access"

- Con: O(N) access to arbitrary element
- Pro: O(1) to remove element at arbitrary position
- Con: worse locality (more cache misses when iterating)
- Con: less compact (next pointers require space)
- Pro: can grow incrementally, nodes are allocated one at a time

3. What is a linked list's head? How is it different from a node? Explain.

Contrast: *head pointer* vs. *head node*. The head pointer is a pointer variable storing a pointer to the first node. The head node *is* the first node in the linked list.

Picture:



4. How do you calculate `length` of a linked list?

A loop is required:

advance

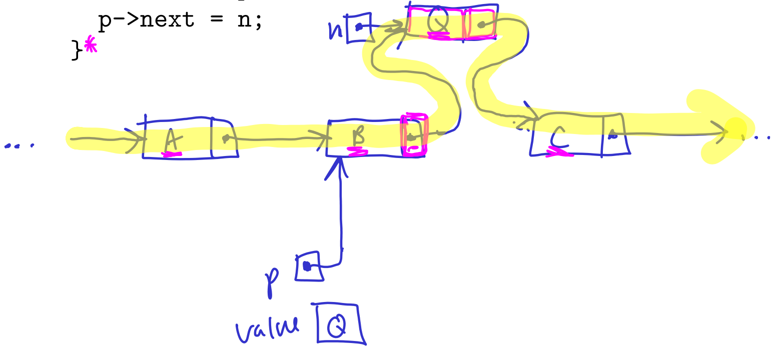
```
struct Node *head = /* points to first node */;
int count = 0;

for (struct Node *cur = head; cur != NULL; cur = cur->next) {
  count++;
}
```

5. How do you implement add_after on a singly linked list?

points to some node in list

a = b;

```
void add_after(struct Node *p, char value) {
  struct Node *n = malloc(sizeof(struct Node));
  n->payload = value;
  n->next = p->next;
  p->next = n;
}*
```



Exercise 17

- Basic linked list functions
- Drawing pictures to reason about how linked lists operations should work is very helpful!
- Note that `reverse_print` is most easily implemented using recursion
- Talk to us if you have questions!

Notes

Notes

Notes

Notes

Notes