Bitwise Operators

- AND (&), OR (|), XOR(^), Complement (~)
- In all our examples sizeof(int) = 4 bytes.

Examples

- Extract the least significant byte of an integer
  - \( a \& 0x00000ff \)
  - \( a \& 0xff \)
- In general the above operation is called masking. One can mask any bit of an integer with appropriate mask.
Bitwise Operators

• Examples
  • Set $n^{th}$ lowest bit of an integer
    - $a \mid (0x1 \ll n)$
  • Compare the second bytes of two integers
    - $!(((a \& 0xff00) \^ (b \& 0xff00)))$
  • Toggle $n^{th}$ lowest bit of an integer
    - $a \^ (0x1 \ll n)$
Formatted Output & `printf`

- `printf` needs **format type & value** to print.
- Format types are available for all built-in data types: int, long, float, double, char, pointers.
  - `printf("%d", 20);`
  - `printf("%x", 20);`
  - `printf("%s", "Hello CS 261");`
  - `printf("%f", 22.0/7);`
  - `printf("%e", 22.0/7);`
  - `printf("%lf", area);`
Formatted Output & `printf`

- What happens if your format type and value type don't match.
  - `printf("%f", 20);`
  - *GCC* compiler gives a warning. So look out for warnings.
  - If you ignore the warning, your output may contain junk.
Formatted Output & `printf`

- Get complete information about `printf` using `man` command.
  - `man 3 printf`

- Width, and text justification
  - `printf("%-20d", sum);`
Formatted Input & `scanf`

- `scanf` needs format type & address of a variable.

- Examples:
  - `int count; char name[100];`
  - `int *ip = &count`
  - `scanf("%d", &count);`
  - `scanf("%s", name);`
  - `scanf("%d", ip);`
Formatted Input & `scanf`

- *Get more information on `scanf` using `man` command.*
  - `man 3 scanf`
Double Ended Queue (Deque)

- Data for N elements
- Operations:
  - append_last
  - append_first
  - remove_last
  - remove_first
  - is_empty
  - is_full
  - get_size
Dequeue – Using Arrays

• Why should we look at Dequeue
  • A few interesting observations in the design of data structures.
  • Its a typical example subtle compromises one have to make with implementation ADTs.

```
struct dequeue
{
    int *data;
    int max;
    int first, last;
};
```

- Pointer for 1-D array to hold dequeue elements.
- 1-D array size.
- First and last indices.
  Also called head & rear in some textbooks.
create_dequeue

• Create a dequeue structure in a consistent state.

```c
struct dequeue* create_dequeue(int queue_size)
{
    struct dequeue *q    = (struct dequeue*) malloc(sizeof(struct dequeue));
    q->data              = (int*) malloc(sizeof(int) * (queue_size + 1));
    q->max               = queue_size;
    q->first             = 0;
    q->last              = 0;

    return q;
}
```
create_dequeue

Queue Size = 5
Data Array Size = 6

q->first = q->last = 0

Empty in the created state.
is_empty

- Depends on implementation. We start with first = last in our implementation.

```c
int is_empty(struct dequeue *q) {
    return q->first == q->last;
}
```
append_last

Empty

append_last(q, 100)

append_last(q, 200)
int append_last(struct dequeue *q, int elt) {
    /* Check for full queue. */
    if (is_full(q)) {
        fprintf(stderr,
                "Queue is full. Cannot add more elements. Queue Size = %d",
                get_size(q));
        return 0;
    }

    q->data[q->last] = elt;
    q->last = (q->last + 1) % q->max;

    return 1;
}
append_first

Queue state from last slide.

append_first(q, 300)

append_first(q, 400)
int
append_first(struct dequeue *q, int elt)
{
    /* Check for full queue */
    if (is_full(q))
    {
        fprintf(stderr,
            "Queue is full. Cannot add more elements. Queue size = %d",
            get_size(q));
        return 0;
    }

    q->first = q->first > 0 ? q->first - 1 : q->max - 1;
    q->data[q->first] = elt;

    return 1;
}
is_full

Queue state from last slide.

append_last(q, 500)

No more elements can be appended. We designed $q->last == q->first$ as condition for is_empty. To distinguish between full and empty states, we compromise one array location.
is_full

int is_full(struct dequeue *q)
{
    return (q->last + 1) % q->max == q->first;
}
get_size

```
size = last - first = 3
```

```
size = last - first = -1
```
int get_size(struct dequeue *q) {
    int sz = q->last - q->first;
    return (sz >= 0 ? sz : q->max + sz);
}

max - first + last = 5
remove_first

```c
int
remove_first(struct dequeue *q)
{
    int r;
    if (is_empty(q))
    {
        printf("Queue is empty. Cannot remove elements."));
        return 0;
    }

    r = q->data[q->first];
    q->first = (q->first + 1) % q->max;
    return r;
}
```
remove_last

```c
int remove_last(struct dequeue *q)
{
    if (is_empty(q))
    {
        fprintf(stderr, "Queue is empty. Cannot remove elements.");
        return 0;
    }

    q->last = q->last > 0 ? q->last - 1 : q->max - 1;
    return q->data[q->last];
}
```
Linked Lists
**Single Linked Lists**

```c
struct node
{
    /*data can be anything.*/
    struct record *elt;
    /*pointer to next node.*/
    struct node *next;
};

struct node *list;
```
Adding a node - at the end

Diagram showing a linked list with a new node added at the end, pointed to by null.
Adding a node - in the middle
Double Linked Lists

Circularly Double Linked List
See you next week!