20.5.8 How do I handle linked lists?

In general, you usually don't need a linked list in Perl, since with regular arrays, you can push and pop or shift and unshift at either end, or you can use splice to add and/or remove arbitrary number of elements at arbitrary points. Both pop and shift are both O(1) operations on Perl's dynamic arrays. In the absence of shifts and pops, push in general needs to reallocate on the order every log(N) times, and unshift will need to copy pointers each time.

If you really, really wanted, you could use structures as described in *perldsc* or *perltoot* and do just what the algorithm book tells you to do. For example, imagine a list node like this:

```
$node = {
    VALUE => 42,
    LINK  => undef,
};
```

You could walk the list this way:

```
print "List: ";
for ($node = $head;  $node; $node = $node->{LINK}) {
    print $node->{VALUE}, " ";
}
print "\n";
```

You could add to the list this way:

```
my ($head, $tail);
$tail = append($head, 1);       # grow a new head
for $value ( 2 .. 10 ) {
    $tail = append($tail, $value);
}

sub append {
    my($list, $value) = @_;
    my $node = { VALUE => $value };
    if ($list) {
        $node->{LINK} = $list->{LINK};
        $list->{LINK} = $node;
    } else {
        $_[0] = $node;      # replace caller's version
    }
    return $node;
}
```

But again, Perl's built-in are virtually always good enough.

20.5.9 How do I handle circular lists?

Circular lists could be handled in the traditional fashion with linked lists, or you could just do something like this with an array:

```
unshift(@array, pop(@array));  # the last shall be first
push(@array, shift(@array));   # and vice versa
```