

Reliable Data Structures in C

The pointers, structures, and files of C allow data structures which are both general-purpose and efficient. Reliable use of these powerful tools requires strict attention to the defining properties of the data.

These techniques are illustrated with complete case-study programs for sorting, screen-handling, menu interaction, simulation, and record-file handling.

Thomas Plum

RELIABLE DATA STRUCTURES IN C

Thomas Plum

Library of Congress Cataloging in Publication Data

Plum, Thomas, 1943-
Reliable Data Structures in C.

Bibliography: p. 10-1.

Includes index.

1. C(Computer program language) 2. Data
structures (Computer science) I. Title.
QA76.73.C15P59 1985 005.13'3 85-45156
ISBN 0-911537-04-X

Acknowledgement of trademarks: UNIX is a trademark of AT&T Bell Laboratories; PDP-11 and VAX are trademarks of Digital Equipment Corporation; CP/M is a trademark of Digital Research Corporation; Venix is a trademark of VenturCom, Inc.; Xenix and MS-DOS are trademarks of Microsoft Corporation; Lattice is a trademark of Lattice, Inc.; Idris is a trademark of Whitesmiths, Ltd.; Tplus is a trademark of Textware International; Safe C is a trademark of Catalytix Corporation; MC68000 is a trademark of Motorola Inc. C is not a trademark, nor are the names of the software development commands such as cc.

Copyright © 1985 by Plum Hall Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

ISBN 0-911537-04-X

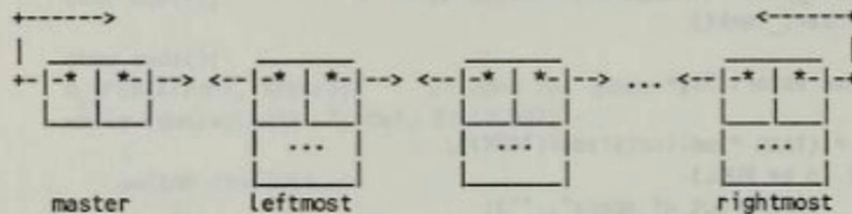
10 9

This book was set by the author, using a Hewlett-Packard LaserJet controlled by the Textware Tplus output driver, formatted by troff under the Venix version of the UNIX operating system on a PDP-11/23.

7.4 Deques: Double-Ended Queues

A "double-ended queue" or *deque* is a data structure in which each node has two pointers, one forward and one backward. Thus, it is also known as a "doubly-linked list."

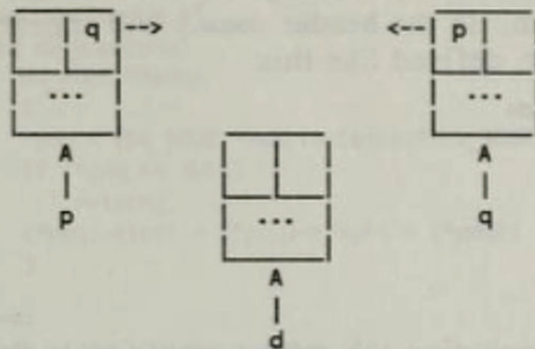
A deque is often implemented with an extra node which contains no data but serves to make the operations more uniform; we will call this node a *master node*. A picture may help:



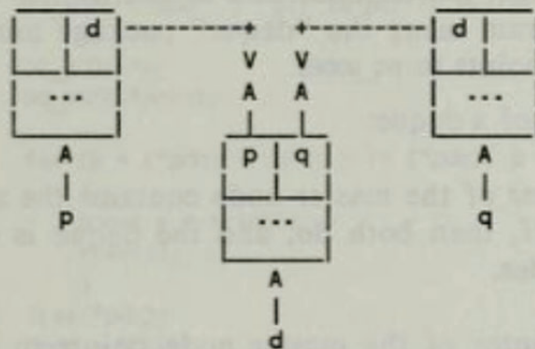
Given a pointer to any node, it is possible to reach all the other nodes by traversing pointers.

A node can be inserted to the right or to the left of any other node; all that is required is a pointer to the new node and a pointer to the insertion point. The following picture shows the insertion of the node *d* (i.e., the node pointed to by pointer *d*), to the right of node *p*, or to the left of node *q*. (Either *p* or *q* is adequate; either can be determined from the other.)

Before insertion:

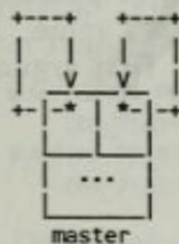


After insertion:



It is possible to remove a node from a deque given nothing more than a pointer to the node itself. Referring to the "after insertion" diagram above, given the pointer d , the nodes to the left and right can be located using information in the node $*d$. Thus, a deletion is just the exact reversal of an insertion.

An empty deque consists of one node, the master node, whose left and right pointers both point to itself:



Thus, even in the empty deque, the master node has a left and right node (itself). There are no special cases in the insertion logic. To implement the deletion logic, the only special case is that the master node should not be freed; the manipulation of left and right pointers is the same for the deletion of any node.

Having a master node makes the calling sequence of the deque operations much more uniform. In the header `deque.h` will appear a type definition for the name `DQ_NODE`, defined like this:

```
#define DQ_NODE struct dq_node
DQ_NODE
{
    DQ_NODE *left;
    DQ_NODE *right;
    /* ... */
};
```

All the nodes in the deque, including the master, conform to this template. Since the master node is not a data node, we can require that the calling program must refer to the master node as a `DQ_NODE`, rather than as a data node. Thus, any program using the "deque" package must provide a "head" pointer which points to `DQ_NODES`.

These are the properties of a deque:

0. If the left or right pointer of the master node contains the address of the master node itself, then both do, and the deque is empty, i.e., contains no data nodes.
- N. Otherwise, the right pointer of the master node points to a well-defined node; this is the "first" node in the deque. Some node, node-N, has a right pointer which points to the master node. The deque consists of the N nodes reachable via the chain of right-pointers, from the first node to node-N. For each node *a* in the deque, the node to the right of node *a* has a left-pointer that points back to *a*. Every node in the deque is well-defined, i.e., all its members are well-defined.

The operations `dq_open` and `dq_close` will alter the deque pointer and must be given its address, a "pointer to pointer to `DQ_NODE`" value. Most of the other operations will take a "pointer to `DQ_NODE`" first argument. Here are the `dq_open` and `dq_close` functions: