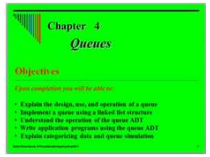


CS 1037
Computer Science Fundamentals II

Part Ten:
Queues

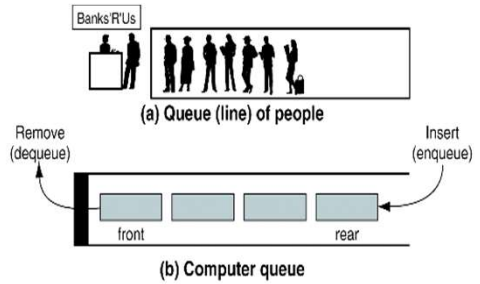


FIGURE 4-1 Queue Concept

4-1 Queue Operations

This section discusses the four basic queue operations. Using diagrammatic figures, it shows how each of them work. It concludes with a comprehensive example that demonstrates each operation.

- Enqueue
- Dequeue
- Queue Front
- Queue Rear
- Queue Example

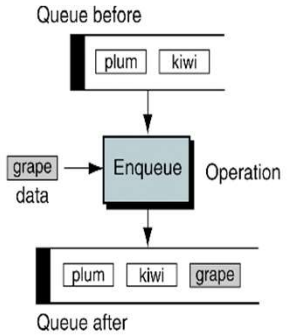


FIGURE 4-2 Enqueue

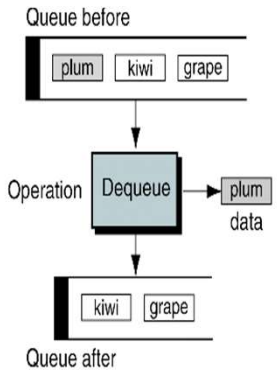


FIGURE 4-3 Dequeue

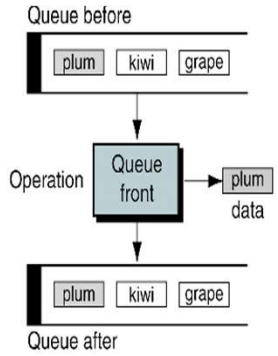


FIGURE 4-4 Queue Front

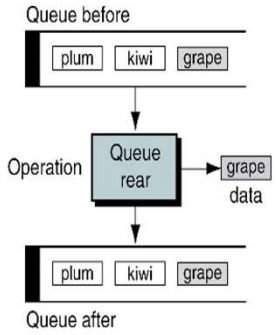


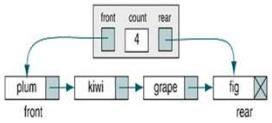
FIGURE 4-5 Queue Rear

4-3 Queue ADT

This section develops the data structures and C code to implement a Queue ADT. The first program contains the data structure declarations and a list of the prototypes for all of the functions. We then develop the C code for the algorithms discussed in Section 4.2

- Queue Structure
- Queue ADT Algorithms

```
#include <stdio.h>
#include <stdlib.h>
#include "queues.h"

int main (void)
{
// Local Definitions
   char* dataPtr ;

   ...

   return 0;
}        // main
```



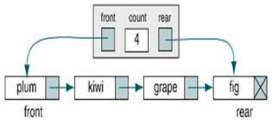
queues.h

```
#include "P4-01.h"                      /* Queue ADT Data Structures */

//           Prototype Declarations
  QUEUE* createQueue  (void);
  bool  dequeue    (QUEUE* queue, void** itemPtr);
  bool  enqueue    (QUEUE* queue, void*  itemPtr);
  bool  emptyQueue (QUEUE* queue);
  int   queueCount (QUEUE* queue);
  void printQueue (QUEUE* queue);
  bool  queueFront (QUEUE* queue, void** itemPtr);
  bool  queueRear  (QUEUE* queue, void** itemPtr);
  int   queueCount (QUEUE* queue);
  QUEUE* destroyQueue (QUEUE* queue);
  bool  fullQueue  (QUEUE* queue);

#include "P4-02.h"                      /* Create Queue */
#include "P4-03.h"                      /* Enqueue */
#include "P4-04.h"                      /* Dequeue */
#include "P4-05.h"                      /* Queue Front */
#include "P4-06.h"                      /* Queue Rear */
#include "P4-07.h"                      /* Empty Queue */
#include "P4-08.h"                      /* Full Queue */
#include "P4-09.h"                      /* Queue Count */
#include "P4-10.h"                      /* Destroy Queue */
#include "P4-14a.h"                  /* Print Queue */
```

queues.h

```
#include "P4-01.h"                      /* Queue ADT Data Structures */

//       Prototype Declarations
  QUEUE* createQueue  (void);
  bool  dequeue    (QUEUE* queue, void** itemPtr);
  bool  enqueue    (QUEUE* queue, void*  itemPtr);
  bool  emptyQueue (QUEUE* queue);
  int   queueCount (QUEUE* queue);
  void printQueue (QUEUE* queue);
  bool  queueFront (QUEUE* queue, void** itemPtr);
  bool  queueRear  (QUEUE* queue, void** itemPtr);
  int   queueCount (QUEUE* queue);
  QUEUE* destroyQueue (QUEUE* queue);
  bool  fullQueue  (QUEUE* queue);

#include "P4-02.h"                      /* Create Queue */
#include "P4-03.h"                      /* Enqueue */
#include "P4-04.h"                      /* Dequeue */
#include "P4-05.h"                      /* Queue Front */
#include "P4-06.h"                      /* Queue Rear */
#include "P4-07.h"                      /* Empty Queue */
#include "P4-08.h"                      /* Full Queue */
#include "P4-09.h"                      /* Queue Count */
#include "P4-10.h"                      /* Destroy Queue */
#include "P4-14a.h"                      /* Print Queue */
```

```
#include <stdio.h>
#include <stdlib.h>
#include "queues.h"

int main (void)
{
// Local Definitions
   char* dataPtr ;

   QUEUE*  ourQ ;

   ...

   return 0;
}        // main
```



P4-01.h

```
// Queue ADT Type Defintions
   typedef struct node
   {
      void*          dataPtr;
      struct node*   next;
   } QUEUE_NODE;

   typedef struct
   {
      QUEUE_NODE*   front;
      QUEUE_NODE*   rear;
      int           count;
   } QUEUE;
```

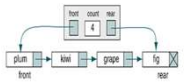
```
#include <stdio.h>
#include <stdlib.h>
#include "stacksADT.h"

int main (void)
{
// Local Definitions
   int* dataPtr ;
   QUEUE* ourQ;

   ourQ = createQueue();
   ...
   return 0;
}        // main
```

P4-02.h

```
QUEUE* createQueue (void)
{
   QUEUE* queue;
   queue = (QUEUE*) malloc (sizeof (QUEUE));
         if (queue)
            {
             queue->front  = NULL;
             queue->rear   = NULL;
             queue->count  = 0;
            } // if
         return queue;
}        // createQueue
```

Label	Address	Value
dataPtr	326 - 329	
ourQ	400 - 403	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	

```
   ...
for (int i = 1; i<=3; i++)
   {
     dataPtr = (char*) malloc (sizeof(char));
     *dataPtr = 64 + i;
     enqueue (ourQ, dataPtr);
   }
   ...
}        // main
```



P4-03.h

```
bool enqueue (QUEUE* queue, void* itemPtr)
{
   QUEUE_NODE* newPtr;
   if (!(newPtr =
      (QUEUE_NODE*)malloc(sizeof(QUEUE_NODE))))
     return false;

   newPtr->dataPtr = itemPtr;
   newPtr->next    = NULL;

   if (queue->count == 0)
   // Inserting into null queue
      queue->front  = newPtr;
   else
      queue->rear->next = newPtr;

   (queue->count)++;
   queue->rear = newPtr;
   return true;
}        // enqueue
```

ALGORITHM 4-4 Retrieve Data at Front of Queue

```
Algorithm queueFront (queue, dataOut)
Retrieves data at the front of the queue without changing
queue contents.
  Pre    queue is a metadata structure
         dataOut is a reference to calling algorithm variable
  Post   data passed back to caller
  Return true if successful, false if underflow
1 if (queue empty)
  1 return false
2 end if
3 move data at front of queue to dataOut
4 return true
end queueFront
```



```
   ...
queueFront (ourQ, (void*)&dataPtr);
printf("Value at Front: %c \n", *dataPtr);

   ...
}        // main
```

P4-05.h

```
bool queueFront (QUEUE* queue, void** itemPtr)
{
// Statements
   if (!queue->count)
      return false;
   else
   {
      *itemPtr = queue->front->dataPtr;
      return true;
   } // else
}     // queueFront
```

P4-09.h

```
int queueCount(QUEUE* queue)
{
//        Statements
          return queue->count;
}         // queueCount
```

Label	Address	Value
dataPtr	326 - 329	17800
ourQ	400 - 403	10100
	...	
	...	
	...	
	...	
front	10100 - 10103	10400
rear	10104 - 10107	21400
count	10108 - 10111	3
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
next	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
next	12564 - 12567	24100
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
next	21404- 21407	NULL
	...	
	...	
	...	
	...	
	...	


```
   ...
queueFront (ourQ, (void*)&dataPtr);
printf("Value at Front: %c \n", *dataPtr);

   ...
}        // main
```

P4-05.h

```
bool queueFront (QUEUE* queue, void** itemPtr)
{
// Statements
   if (!queue->count)
      return false;
   else
   {
      *itemPtr = queue->front->dataPtr;
      return true;
   } // else
}     // queueFront
```

P4-09.h

```
int queueCount(QUEUE* queue)
{
//        Statements
          return queue->count;
}         // queueCount
```

Label	Address	Value
dataPtr	326 - 329	17800
ourQ	400 - 403	10100
	...	
	...	
	...	
	...	
front	10100 - 10103	10400
rear	10104 - 10107	21400
count	10108 - 10111	3
{ DM }	10210 - 10213	A
dataPtr	10400 - 10403	10210
next	10404 - 10407	12560
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
next	12564 - 12567	24100
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
next	21404- 21407	NULL
	...	
	...	
	...	
	...	
	...	

(void*)&dataPtr
- why ?
- what is going on ...

ALGORITHM 4-3 Delete Data from Queue

```
Algorithm dequeue (queue, item)
This algorithm deletes a node from a queue.
  Pre    queue is a metadata structure
         item is a reference to calling algorithm variable
  Post   data at queue front returned to user through item
         and front element deleted
  Return true if successful, false if underflow
1 if (queue empty)
  1 return false
2 end if
3 move front data to item
4 if (only 1 node in queue)
  Deleting only item in queue
  1 set queue rear to null
5 end if
6 set queue front to queue front next
7 decrement queue count
8 return true
end dequeue
```



```
   ...
   dequeue (ourQ, (void*)&dataPtr);
   printf("Value at Front: %c \n", *dataPtr);
   ...
}        // main
```

output: Value at Front: A

P4-05.h

```
bool dequeue (QUEUE* queue, void** itemPtr)
{
   QUEUE_NODE* deleteLoc;

   if (!queue->count)
      return false;

   *itemPtr  = queue->front->dataPtr;
   deleteLoc = queue->front;
   if (queue->count == 1)
      // Deleting only item in queue
      queue->rear  = queue->front = NULL;
   else
      queue->front = queue->front->next;

   (queue->count)--;
   free (deleteLoc);

   return true;
}          // dequeue
```

Label	Address	Value
dataPtr	326 - 329	10210
ourQ	400 - 403	10100
	...	
	...	
	...	
	...	
front	10100 - 10103	12560
rear	10104 - 10107	21400
count	10108 - 10111	2
{ DM }	10210 - 10213	A
	...	
	...	
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
next	12564 - 12567	24100
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
next	21404- 21407	NULL
	...	
	...	
	...	
	...	
	...	

SOLUTION ONE:

```
   ...
   dequeue (ourQ, (void*)&dataPtr);
   printf("Value at Front: %c \n", *dataPtr);
   free(dataPtr);
   ...
}        // main
```

P4-05.h

```
bool dequeue (QUEUE* queue, void** itemPtr)
{
   QUEUE_NODE* deleteLoc;

   if (!queue->count)
      return false;

   *itemPtr  = queue->front->dataPtr;
   deleteLoc = queue->front;
   if (queue->count == 1)
      // Deleting only item in queue
      queue->rear  = queue->front = NULL;
   else
      queue->front = queue->front->next;

   (queue->count)--;
   free (deleteLoc);

   return true;
}          // dequeue
```

Label	Address	Value
dataPtr	326 - 329	10210
ourQ	400 - 403	10100
	...	
	...	
	...	
	...	
front	10100 - 10103	12560
rear	10104 - 10107	21400
count	10108 - 10111	2
{ DM }	10210 - 10213	A
	...	
	...	
{ DM }	12300 - 12303	B
dataPtr	12560 - 12563	12300
next	12564 - 12567	24100
{ DM }	17800 - 17803	C
dataPtr	21400 - 21403	17800
next	21404- 21407	NULL
	...	
	...	
	...	
	...	
	...	

```
   ...
// Now print numbers in reverse
  printf ("\n\nThe list of numbers reversed:\n");
  while (!emptyStack (stack))
  {
    dataPtr = (int*)popStack (stack);
    printf ("%3d\n", *dataPtr);
    free (dataPtr);
  } // while
  ...
  return 0;
}        // main
```

P3-11.h

```
bool emptyStack (STACK* stack)
{
// Statements
   return (stack->count == 0);
}  // emptyStack
```

Label	Address	Value
dataPtr	326 - 329	17800
stack	400 - 403	10100
i	404 - 407	3
	...	
	...	
	...	
count	10100 - 10103	3
top	10104 - 10107	21400
{ DM }	10210 - 10213	1
dataPtr	10400 - 10403	10210
link	10404 - 10407	NULL
{ DM }	12300 - 12303	2
dataPtr	12560 - 12563	12300
link	12564 - 12567	10400
{ DM }	17800 - 17803	3
dataPtr	21400 - 21403	17800
link	21404- 21407	12560
	...	

```
  ...
// Now print numbers in reverse
  printf ("\n\nThe list of numbers reversed:\n"
  while (!emptyStack (stack))
  {
    dataPtr = (int*)popStack (stack);
    printf ("%3d\n", *dataPtr);
    free (dataPtr);
  } // while
  ...
  return 0;
}       // main
```

P3-09.h

```
void* popStack (STACK* stack)
{
   void*          dataOutPtr;
   STACK_NODE*    temp;

   if (stack->count == 0)
      dataOutPtr = NULL;
   else
   {
      temp       = stack->top;
      dataOutPtr = stack->top->dataPtr;
      stack->top = stack->top->link;
      free (temp);
      (stack->count)--;
   } // else
   return dataOutPtr;
}       // popStack
```

Label	Address	Value
dataPtr	326 - 329	17800
stack	400 - 403	10100
i	404 - 407	3
	...	
	...	
	...	
count	10100 - 10103	3
top	10104 - 10107	21400
{ DM }	10210 - 10213	1
dataPtr	10400 - 10403	10210
link	10404 - 10407	NULL
{ DM }	12300 - 12303	2
dataPtr	12560 - 12563	12300
link	12564 - 12567	10400
{ DM }	17800 - 17803	3
dataPtr	21400 - 21403	17800
link	21404- 21407	12560
	...	

```
  ...
// Now print numbers in reverse
  printf ("\n\nThe list of numbers reversed:\n"
  while (!emptyStack (stack))
  {
    dataPtr = (int*)popStack (stack);
    printf ("%3d\n", *dataPtr);
    free (dataPtr);
  } // while
  ...
  return 0;
}       // main
```

P3-09.h

```
void* popStack (STACK* stack)
{
   void*          dataOutPtr;
   STACK_NODE*    temp;

   if (stack->count == 0)
      dataOutPtr = NULL;
   else
   {
      temp       = stack->top;
      dataOutPtr = stack->top->dataPtr;
      stack->top = stack->top->link;
      free (temp);
      (stack->count)--;
   } // else
   return dataOutPtr;
}       // popStack
```

Label	Address	Value
dataPtr	326 - 329	17800
stack	400 - 403	10100
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	

ALGORITHM 3-8 Destroy Stack

```
Algorithm destroyStack (stack)
This algorithm releases all nodes back to the dynamic memory.
   Pre    stack passed by reference
   Post   stack empty and all nodes deleted
1  if (stack not empty)
```

continued

```
   1  loop (stack not empty)
      1  delete top node
   2  end loop
2  end if
3  delete stack head
end destroyStack
```



```
int main (void)
{
   ...
// Destroying Stack
   destroyStack(stack);

   return 0;
}       // main
```

P3-14.h

```
STACK* destroyStack (STACK* stack)
{
   STACK_NODE* temp;

   if (stack)
   {
      while (stack->top != NULL)
      {
        free (stack->top->dataPtr);
        temp = stack->top;
        stack->top = stack->top->link;
        free (temp);
       } // while

       free (stack);
   } // if stack
   return NULL;
}       // destroyStack
```

Label	Address	Value
dataPtr	326 - 329	17800
stack	400 - 403	10100
i	404 - 407	3
stack	508 - 511	10100
	...	
	...	
count	10100 - 10103	3
top	10104 - 10107	21400
{ DM }	10210 - 10213	1
dataPtr	10400 - 10403	10210
link	10404 - 10407	NULL
{ DM }	12300 - 12303	2
dataPtr	12560 - 12563	12300
link	12564 - 12567	10400
{ DM }	17800 - 17803	3
dataPtr	21400 - 21403	17800
link	21404- 21407	12560
	...