How to Implement a Queue in C

[Jacob Sorber](https://www.youtube.com/channel/UCwd5VFu4KoJNjkWJZMFJGHQ)

<https://www.youtube.com/watch?v=FcIubL92gaI>

|  |  |
| --- | --- |
| so what is a queue and how do you implement it in C that's our topic for today  so I recently did a video about stacks  one of the first data structures that you learn anytime you start a real study of data structures and today I want to talk about its cousin the queue so quick recap stacks were basically just a list of elements where you can only access them at one end the top you can push stuff on to the stack you can pop stuff off of the stack if this isn't making any sense you should go watch that video  and then come back it may make today's video make more sense but queues are also basically a list there are linear data structure but instead of just accessing at one end we access at both ends but in a very controlled way we have two ends a head and a tail we add new elements to the tail and we remove things from the head so this is a data structure that basically works like a queue in a grocery store or really any scenario where you're waiting in line basically you're adding things to the back of the list and you're taking things off the front of the list and things get serviced in a first-come first-serve order just like with stacks we have two operations instead of push and pop we have n Q and D Q and Q is where we actually add something to the queue DQ is when we remove something from the queue and you're gonna find that we really use cues really everywhere in software anytime you have stuff arriving this can be work this can be network connections whatever anytime you have something that is arriving that you want to process in the order it  arrived you want a queue and like a stack we can implement a queue with an array or with a linked list today I'm a little short on time so I'm going to focus on the linked list implementation and we'll pick up the array base queue in a future video and of course if you've never seen linked lists before and that's a little confusing check out my linked list videos just to catch up again link in the description also as always source code is available through patreon check the description for more information on how to get access but now let's jump into the code if you saw the snacks video this code should look familiar to begin with I just took that code replaced stack with Q and removed the code from push and pop so now we  02:01  have this empty NQ and DQ we have our  02:03  two operations and we have main that  02:06  calls those functions  02:08  but which of course won't work right now  02:09  because our code doesn't do anything yet  02:11  and like before we're working with intz  02:13  just because they're simple and we could  02:15  easily use doubles float structs  02:17  pointers anything we want really  02:19  but insert quick and easy so that's what  02:21  we're going to use today now the first  02:22  thing I'm going to change is the  02:24  definition of what we want to call a  02:26  queue here with a stack everything  02:28  happens at one end all pushing and  02:30  popping happens at that one end at the  02:32  top so it was enough to just store a  02:34  single pointer and call that our stack  02:36  but with my queue I want to keep track  02:39  of two pointers I need to keep track of  02:40  the head and the tail which will put  02:43  together in another struct and we're  02:45  going to call that struct a queue this  02:47  is basically just all the data that  02:49  we're going to need to represent our  02:50  queue and I'm gonna follow the advice  02:52  that I recommended to myself in my  02:53  stacks video and make an initialization  02:56  function that sets up our queue in an  02:58  object-oriented language this would be  03:00  in your constructor but here it's just a  03:02  function that's going to take a pointer  03:04  to our struct that's so we can modify it  03:06  and then we're going to set both the  03:08  head and the tail to null meaning that  03:10  our queue starts out empty and saving  03:18  some typing let's rename these my queues  03:20  to just queue I like that a bit better  03:22  now in NQ we're just going to create a  03:25  new node by calling malloc we set its  03:33  value to the value we want to add to the  03:35  cube and we set its next pointer to the  03:39  goal that's because this is going at the  03:41  end of the list so it's not going to  03:43  have anything after it now we need to  03:45  connect it to our list so we'll check to  03:47  see if we have a tail on this list if we  03:51  do we just connect it to the next  03:53  pointer to the new node we just created  03:56  then we set the tail to be the new node  03:59  so it is now the new tail now one last  04:04  thing if the list was empty and we just  04:06  added the first element then head is  04:08  going to still be null so let's just  04:10  check to see if that's the case and if  04:12  it is we can just set the head to point  04:14  to that new node as well so just to  04:17  recap we created a new node if there's a  04:20  tail we connect the old  04:21  up to this new tail and then we make  04:30  sure that the head still make sense  04:33  and that's really it of course we still  04:35  need to return true if we were  04:37  successful and let's come back up and  04:40  return false if Malik failed basically  04:42  just meaning that we couldn't get any  04:44  more memory because that's really this  04:45  functions only failure case unless of  04:47  course someone passes in a bad pointer  04:49  now let's look at D Q now anytime you're  04:52  writing a function it's useful to think  04:54  about its error cases what can I check  04:56  now to see if this operation that was  04:59  requested is even possible now with D Q  05:01  the main check we want to do up front is  05:03  the empty case if the queue is empty  05:05  there's nothing we can do I can't return  05:08  an integer from an empty queue so we're  05:09  going to first check to see if the queue  05:11  is empty and if it is we just returned  05:15  after that we're going to save a pointer  05:18  to the head and grab the integer value  05:22  in the head then we just move the  05:26  current head to point to the node after  05:28  it this is the same thing we did with  05:30  our stack and then of course if we run  05:33  out of nodes we need to make sure to let  05:35  the tail know and set it to null as well  05:39  we definitely don't want to leave our  05:41  tail dangling there and then of course  05:43  we return our saved result okay so to  05:46  recap we check for an empty queue just  05:49  to make sure that we can do what we're  05:50  being asked to do and then we save the  05:52  head of the queue oh wait I forgot to  05:56  free my memory that's not good  05:58  that would have been a memory leak if I  06:00  had forgotten to do that so now we save  06:02  the result and you're going to return  06:03  and then we finally remove it from the  06:05  actual list and that looks pretty good  06:10  let's make sure it works down here in  06:12  main we need to clean up some stuff  06:14  mostly just the initialization now let's  06:16  call our init Q function and these  06:21  queues are still called s1 s2 and s3 I  06:24  told you I just copied the stack code  06:25  over as a starter so that's going to bug  06:28  me so let's change these to make them q1  06:29  q2 and q3 and we initialize the last two  06:34  Q's  06:38  okay so now let's see if it works so  06:40  into the terminal we go we compile it  06:45  okay nice I'm actually pleasantly  06:47  surprised I didn't mess anything up  06:49  there  06:49  and we run it and it works it's just  06:51  like our stack example except that the  06:53  order is reversed so we're getting  06:54  values out in first in first out or FIFO  06:57  order rather than in last in first out  06:59  order or LIFO order which is what we got  07:02  from our stack so hopefully at this  07:03  point you can see that queues aren't  07:05  really complicated they're really not  07:06  and they're used everywhere as I  07:08  mentioned you can build a queue with an  07:10  array as well as a linked list it's not  07:12  complicated except we do have to be a  07:13  little bit careful about our array  07:15  indexes and how they wrap around this is  07:17  usually using the modulus operator but  07:19  we'll talk about that in a future video  07:20  also I know a lot of you have been  07:22  waiting for more information about the  07:23  upcoming course that I'm offering in  07:24  July I'm happy to be able to tell you  07:26  today that the course page is now up at  07:28  the following URL you can now sign up  07:30  for the course space is of course  07:31  limited because my time is limited and  07:33  this is going to be a hybrid video and  07:35  live interactive course check out the  07:37  link for more information and I look  07:38  forward to seeing you there if on the  07:40  other hand you're interested in more  07:41  free content tips and tutorials check  07:43  out my other videos like these subscribe  07:45  to the channel so you don't miss new  07:47  videos on data structures and other  07:48  topics that you care about and until  07:51  next time everyone I'll see you later | Итак что же такое очередь и как вы ее реализуете на языке Си это наша сегодняшняя тема  поэтому я недавно сделал видео о стеках  одна из первых структур данных которые вы изучаете в любое время когда вы начинаете реальное изучение структур данных и сегодня я хочу поговорить о ее двоюродном брате очереди так что быстрые стеки резюме были в основном просто списком элементов где вы можете получить к ним доступ только с одного конца сверху вы можете протолкнуть материал в стек вы можете вытащить материал из стека если это не имеет никакого смысла вы должны пойти посмотреть это видео  а потом вернитесь это может сделать сегодняшнее видео более осмысленным но очереди также в основном представляют собой список есть линейная структура данных но вместо того чтобы просто получать доступ на одном конце мы получаем доступ на обоих концах но очень контролируемым образом у нас есть два конца голова и хвост мы добавляем новые элементы в хвост и удаляем вещи из головы так что это структура данных которая в основном работает как очередь в продуктовом магазине или действительно любой сценарий где вы ждете в очереди в основном вы добавляете вещи в заднюю часть списка и снимаете вещи с передней части списка и вещи обслуживаются в как и в случае со стеками, у нас есть две операции вместо push и pop, у нас есть n Q и D Q и Q-это когда мы на самом деле добавляем что-то в очередь DQ-это когда мы удаляем что-то из очереди, и вы обнаружите, что мы действительно используем сигналы действительно везде в программном обеспечении в любое время, когда у вас есть вещи, прибывающие это может быть работа это может быть сетевое подключение что угодно в любое время, когда у вас есть что-то, что прибывает, что вы хотите обработать в порядке  если вы хотите очередь и как стек мы можем реализовать очередь с массивом или со связанным списком сегодня мне немного не хватает времени поэтому я собираюсь сосредоточиться на реализации связанного списка и мы возьмем базовую очередь массива в будущем видео и конечно если вы никогда не видели связанные списки раньше и это немного сбивает с толку проверьте мои видео со связанным списком просто чтобы наверстать упущенное ссылка в описании также как всегда исходный код доступен через patreon проверьте описание для получения дополнительной информации о том как получить доступ но теперь давайте перейдем к коду если вы видели видео с закусками это код должен выглядеть знакомым для начала я просто взял этот код заменил стек на Q и удалил код из push и pop так что теперь мы |