UMEÅ UNIVERSITET
Institutionen för Datavetenskap
Rapport obligatorisk uppgift

Datastrukturer och algoritmer (C) 7.5 p
5DV149

Obligatorisk uppgift nr 2

Namn	Jonas Sjödin
E-post	josj0105@student.umu.se
Datum	6 februari 2017
Handledare	Daniel Harr, Jakob Lindqvist, Jakob Vesterlind, Sebastian Sandberg

UMEÅ UNIVERSITET
Institutionen för Datavetenskap
Rapport obligatorisk uppgift

Innehåll

UMEÅ UNIVERSITET
Institutionen för Datavetenskap
Rapport obligatorisk uppgift

Syfte

Syftet med denna uppgift är att du ska:

- bekanta dig med en given implementation av datatypen Kö i enlighet med gränsytan i kursboken,
- lära dig att testa en datatyp för att övertyga dig (och andra) om att den fungerar korrekt.

De FSR (lärmål) som examineras (helt eller delvis) med denna uppgift är:

- implementera lösningen i form av ett program i programspråket C,
- tillämpa dynamisk minnesallokering,
- visa att en komponent i ett program fungerar korrekt med hjälp av testning.

Inledning

I denna laboration har alla funktionerna i filen queue.c testats för att kunna uppfylla de olika kriterierna som står i den formella specifikationen av kö (Janlert Lars-Erik och Wibergs Torbjörn. *Datatyper och Algoritmer*. 2011. sid 156). Med hjälp av 6 axiomer har 6 olika tester utförts för att kunna bevisa om en anonym kod uppfyller kriterierna för att kallas kö. Testerna görs i C.

Ax 1 $Isempty(Empty)$

Ax 2 $\neg(Isempty(Enqueue(v, q)) = q$

Ax 3 $Isempty(q) \Rightarrow Dequeue(Enqueue(v, q)) = q$

Ax 4 $\neg(Isempty(q)) \Rightarrow Dequeue(Enqueue(v, q)) = Enqueue(v, , Dequeue(q))$

Ax 5 $Isempty(q) \Rightarrow Front(Enqueue(v, q)) = v$

Ax 6 $\neg(Isempty(q)) \Rightarrow Front(Enqueue(v, q)) = Front(q)$

C-koden

I denna del kommer alla sex axiomtest att presenteras med sin kod i C. Jag kommer även att presentera beskrivningar till hur funktionerna fungerar. De funktioner i queue.c som jag har använt mig av är:

- queue_setMemHandler: allokerar minne dynamiskt för en kö.
- queue_free: frigör det allokerade minnet för en kö.
- queue_empty: Skapar en tom kö.
- queue_isEmpty: Kollar om en kö är tom och returnerar TRUE/FALSE
- queue_enqueue: Lägger till ett värde sist i kön
- queue_dequeue: Tar bort det första värdet i en kö.
- queue_front: Returnerar det första värdet i kön.

Axiom 1 $Isempty(Empty)$

Axiom 1 säger att varje gång en ny kö skapas så är den tom. För att kontrollera detta har jag först skapat en kö med queue_empty och sedan använder jag mig av en if-sats som använder funktionen queue_isEmpty för att kolla om kön är tom.

```
1  void axiom1(){
2    queue *queue1 = queue_empty();
3    queue_setMemHandler(queue1, free);
4
5    //If the queue is empty the test has suceeded.
6    if (queue_isEmpty(queue1)){
7      printf("Axiom 1. Successful!\n");
8    }
9    else{
10     queue_free(queue1);
11     printf("Axiom 1. Failed!\n");
12     printf("The queue isn't created empty. Either doesn't queue_empty or ");
13     printf("queue_isEmpty work correctly\n");
14
15     exit(1);
16   }
17
18   queue_free(queue1);
19   printf("\n");
```

Resultat Axiom 1

```
Axiom 1. Successful!
```

Axiom 2 $\neg(Isempty(Enqueue(v, q)) = q$

Axiom 2 säger att om man lägger till ett värde först i en tom kö så kommer kön inte vara tom. För att kolla detta använder jag queue_enqueue för att lägga till ett värde först i den tomma kön och sedan queue_isEmpty för att kolla så att kön inte är tom längre.

```
1  void axiom2(){
2    queue *queue1 = queue_empty();
3    queue_setMemHandler(queue1, free);
4
5    int *value1 = malloc(sizeof(int));
6
7    queue_enqueue(queue1, value1);
8
9    //If the queue is not empty the queue has succeeded.
10   if (queue_isEmpty(queue1)){
11     printf("Axiom 2. Failed!\n");
12     printf("Can't add a value. ");
13     printf("The queue_enqueue function does not work.\n");
14
15     exit(1);
16   }
17   else{
```

```
18      printf("Axiom 2. Successful!\n");
19    }
20
21    queue_free(queue1);
22    printf("\n");
23  }
```

Resultat Axiom 2

```
Axiom 2:
Data is added to queue. Test OK!
```

Axiom 3

Axiom 3 säger att om vi har en tom kö där vi sedan lägger till ett värde längst fram och därefter tar bort värdet så bör kön vara oförändrad. För att kontrollera axiom 3 skapar jag en tom kö med queue_empty. Lägger till ett värde i den skapade kön med queue_enqueue. Sedan tar jag bort ett värde med queue_dequeue. Genom en if- sats med villkoret queue_isEmpty kontrollerar jag sedan om kön är oförändrad, dvs tom.

```
1   //If the queue is not empty the queue has succeeded.
2   if (queue_isEmpty(queue1)){
3     printf("Axiom 2. Failed!\n");
4     printf("Can't add a value. ");
5     printf("The queue_enqueue function does not work.\n");
6
7     exit(1);
8   }
9   else{
10    printf("Axiom 2. Successful!\n");
11  }
12
13  queue_free(queue1);
14  printf("\n");
15 }
16
17 /*
18  * Function: axiom3()
19  *
20  * Description: Tests if the queue is empty if a value is added to an empty
21  * queue and then the first value of the queue is deleted.
22  *
23  * Input: NONE.
```

Resultat test 3

```
Axiom 3:
Queue is unchanged and empty. Test OK!
```

Test 4

Axiom 4 säger att så länge kön inte är tom så spelar ordningen på operationerna ingen roll. För att testa det så skapar jag två köer som jag båda lägger till samma värde i. Sedan lägger jag till ännu ett värde i kön den första kön och tar sedan bort det. I den andra kön tar jag först bort ett värde och lägger sedan till ett värde. För att axiom 4 nu ska stämma borde värdena i de två olika köerna ha samma värde. Med en if- sats med queue_front som villkor kontrollerar jag nu om värdena är lika.

```
1
2     queue_enqueue(queue1, value1);
3     queue_dequeue(queue1);
4
5     //If the queue is empty the test has succeeded.
6     if (queue_isEmpty(queue1)){
7       printf("Axiom 3. Successful!\n");
8
9     }
10    else{
11      printf("Axiom 3. Failed!\n");
12      printf("Can't delete a value. ");
13      printf("The queue_dequeue function does not work.\n");
14
15      exit(1);
16    }
17
18    queue_free(queue1);
19    printf("\n");
20
21  }
22
23  /*
24   * Function: axiom4()
25   *
26   * Description: If you have two identical NONempty queues. If you add a
27   * value to one of the queues and then deletes the first value in the queue.
28   * Then you delete the first value in the queue and then add the same value as
29   * above. If both queues are the same now then the test has suceeded.
30   *
31   * Input: NONE.
32   * Output: Prints info if test failed or succeeded.
33   */
34  void axiom4(){
35    queue *queue1 = queue_empty();
36    queue *queue2 = queue_empty();
```

Resultat test 4

```
Axiom 4:
The operations doesn't matter which order. Test OK!
```

Test 5

Axiom 5 säger att en tom kö med ett tillagt värde v, är densamma som det första element i kön, dvs v. För att kontrollera axiom 5 så skapar jag en kö och lägger till ett värde. Med en if- sats så kollar jag sedan om det första värdet i kön är lika med det värde jag la till i kön.

```
1   *value1 = 1;
2   *value2 = 9;
3   *value3 = 1;
4   *value4 = 9;
5
6   queue_enqueue(queue1, value1);
7   queue_enqueue(queue2, value3);
8
9   queue_enqueue(queue1, value2);
10  queue_dequeue(queue1);
11
12  queue_dequeue(queue2);
13  queue_enqueue(queue2, value4);
14
15  //If the queues are the same the test has succeeded.
16  if (*(int*)queue_front(queue1) == *(int*)queue_front(queue2)){
17    printf("Axiom 4. Successful!\n");
18  }
19
20  else{
21    printf("Axiom 4. Failed!\n");
22    printf("One of the queue_enqueue and/or queue_dequeue does not work");
```

Resultat test 5

```
Axiom 5:
Values are the same. Test OK!
```

Test 6

Axiom 6 säger att om vi lägger till ett element i en icke tom kö, så skall det första elementet i den nya kön vara detsamma som det första elementet i den gamla. För att kontrollera axiom 6 så skapar jag två köer som jag sedan lägger till ett värde i. Sedan lägger jag till ett till värde i den ena kön. Om axiom 6 nu ska stämma så ska de första värdena i köerna vara samma. Detta kontrollerar jag med en if- sats.

```
1     exit(1);
2   }
3
4   queue_free(queue1);
5   queue_free(queue2);
6   printf("\n");
7
8 }
9
10 /*
11  * Function: axiom5()
```

```
12   *
13   * Description: If you have an empty queue which you add a value to. The test
14   * succeeds if the first value of the que is the same as the one that was
15   * recently added.
16   *
17   * Input: NONE.
18   * Output: Prints info if test failed or succeeded.
19   */
20  void axiom5(){
21    queue *queue1 = queue_empty();
22    queue_setMemHandler(queue1, free);
23
24    int *value1 = malloc(sizeof(int));
25    *value1 = 1;
26
27    queue_enqueue(queue1, value1);
28
29    //If the queues first value is the same as value1 the test has succeeded.
```

Resultat test 6

```
Axiom 6:
Correct order. Test OK!
```