그런즉 너희가 먹든지 마시든지 무엇을 하든지 다 하나님의 영광을 위하여 하라 (고전 10:31)

NOTE: The following materials have been compiled and adapted from the numerous sources including my own. Please help me to keep this tutorial up-to-date by reporting any issues or questions. Send any comments or criticisms to idebtor@gmail.com Your assistances and comments will be appreciated.

Midterm - Circular Queue

목차

제공되는 파일 목록

- cirque.pdf this file
- cirque1.cpp skeleton code for Step 1
- cirque1x.exe, cirque1x a solution for checking Step 1 & 2 for PC & MacOS
- driver.cpp tests this the circular queue code interactively for Step 3
- cirque.h include file
- cirquex.exe, cirquex a solution for checking for Step 3 for PC & MacOS

문제 목적과 진술

일반적인 Queue 구조의 문제는 front 는 줄어들기만 하고, rear (or back) 늘어나기만 하기 때문입니다. 이런 문제를 해결하기 도입한 것이 Circular Queue 입니다. Circular Queue 는 상당히 효율적으로 메모리를 사용하는 Queue 데이터 구조를 제공합니다. 여기서 우리의 목표는 기존하는 STL 이나 링크리스트를 사용하지 않고, Circular Queue 라를 동적 메모리 할당을 사용하여 O(1) 시간 복잡도의 알고리즘을 구현하고자 합니다.

Step 1: Static Circular Queue - cirque1.cpp (2 점)

우리는 강의 시간에 배운 Circular Queue 에 대한 충분한 이해가 있어야 합니다. 수업에서는 크기가 정해진 Circular Queue 를 가지고 코딩을 했습니다. Step 1 에서 수업에서 공부한 CircularQueue 데이터 구조와 같이 MAXLEN 을 사용합니다.

주어진 소스 파일(cirque1.cpp)은 magic number MAXLEN 을 사용하고 또한 부분적으로 미완성된 코드이지만, 컴파일해서 실행할 수 있습니다. 큐의 내용들은 모두 string type 데이터입니다. 테스트 코드와 기대하는 결과값이 주어졌습니다. 결과값은 cirque1x.exe 비교해볼 수 있습니다. 다음과 같은 함수를 완성하십시오.

- size()
- show_queue()
- dequeue()

이 프로그램을 빌드하기 위해서 다음과 같은 명령어로 할 수 있으며, 기대하는 결과값을 다음과 같습니다.

```
g++ cirque1.cpp -o cirque1
./cirque1
```

```
Front:back=[-1:-1] maxlen=4 size=0
Items:[ - - - - ]
Queue:[  ]

enqueued: a
enqueued: b
Front:back=[0:1] maxlen=4 size=2
Items:[ a b - - ]
Queue:[ a b ]

enqueued: c
enqueued: d
enqueued: e
enqueued: f
Front:back=[2:1] maxlen=4 size=4
Items:[ e f c d ]
Queue:[ c d e f ]

dequeued: c
Front:back=[3:1] maxlen=4 size=3
Items:[ e f - d ]
Queue:[ d e f ]

enqueued: g
Front:back=[3:2] maxlen=4 size=4
Items:[ e f g d ]
Queue:[ d e f g ]

enqueued: h
Front:back=[2:3] maxlen=4 size=2
Items:[ - - g h ]
Queue:[ g h ]
```

Step 2: Circular Queue - cirque2.cpp (1 점)

여기서는 newCircularQueue()함수를 C++ Constructor 로 대체 헙니다.

- cirque1.cpp 파일을 복사하여 cirque2.cpp 라 이름합니다.
- 아래의 코드에서 struct CircularQueue 안에 constructor 와 destructor 를 추가하고, newCircularQueue()함수를 삭제합니다.

```
// MAXLEN of circular queue, a magic number to get rid of
const int MAXLEN = 4;
struct CircularQueue {
  string items[MAXLEN];             // queue item storage
  int front, back;                  // set to -1 to begin with
};

using cirque = CircularQueue *;

cirque newCircularQueue(){
  cirque q = new CircularQueue;
  for (int i = 0; i < MAXLEN; i++)
    q->items[i] = '-';            // dash: a placeholder for empty slot
  q->front = -1;
  q->back = -1;
  return q;
}
```

- MAXLEN 을 삭제하고, structure 의 한 멤버 변수(property)로 maxlen 을 추가합니다.
- 큐 요소들을 저장할 items[]는 pointer 로 정의하고, constructor 안에서 default 혹은 사용자가 전달하는 크기(capa = 4) 혹은 default 로 설정하십시오.
- 큐 요소가 가 저장되지 않은 부분을 빈칸이 아니라 하나의 mark 를 표시하기 위하여 a placeholder 를 위한 멤버변수(dash)를 추가하고, "-"를 저장하여 사용하십시오.
- Constructor 를 사용하면서, new 로 메모리를 할당했으므로, Destructor 를 한번은 사용하십시오.

```
struct CircularQueue {
  string *items;                    // queue item storage
  int front, back;                  // set to -1 to begin with
  string dash;                      // dash: a placeholder for empty items
  int maxlen;                       // maxlen: total capacity of the queue

  // your code here: constructor and destructor

}
. . .

int main() {
    cirque q = new CircularQueue(6);
    show_qstat(q); show_items(q); show_queue(q); cout << endl;
    . . .
    . . .
```

그러면, main()함수에서 CircularQueue 생성 방법만 다를 뿐이며, 모든 결과는 Step 1 과 같게 됩니다.

Step 3: Circular Queue - cirque.cpp, driver.cpp & cirque.h

이제부터 본격적인 개발 과정에 진입합니다. 개발하는 코드를 집중적으로 테스트할 수 있도록 driver.cpp 코드를 개발하여 사용합니다. 또한 사용자와 개발자를 구별하고, 개발의 편의성을 위하여 include 파일을 만들어 제공합니다. 학우들은 이미 만들어 놓은 include 파일(cirque.h)을 활용합니다.

- 이제 여러분에 제공된 nowic/include/cirque.h 파일과 driver.cpp 파일을 읽어 보십시오. 전체 내용을 파악하십시오. CircularQueue 구조 안에 shown 이 추가된 것 외에는 변화가 없을 것입니다.
- 소스코드 cirque2.cpp 를 복사하여 cirque.cpp 를 만드십시오.
  nowic/include/cirque.h 를 cirque.cpp 포함(include)하십시오.
  그리고, 이미 cirque.h 에 정의된 부분을 cirque.cpp 에서 제거 하십시오.
- 함수 main()을 cirque.cpp 에서 제거하십시오.
  왜냐면? 이제 driver.cpp 에 정의된 main()을 사용하기 때문입니다
- 그리고, 다음과 같은 명령어(각자의 개발환경에 따라 path 가 다를 수 있음)로 컴파일을 하면 **에러 메시지가 출력됩니다**..

```
g++ cirque.cpp driver.cpp -I../../include -o cirque
```

이를 관찰해보면 다음과 같은 함수들이 undefined 되었다는 것을 알 수 있습니다. 그 이유는 driver.cpp 에서 이 함수들을 호출하지만, 컴파일할 때는 cirque.h 에서 찾아 컴파일할 수 있었지만, 실행파일을 만들기 위한 링크 단계에서는 발견할 수 없어서 오류가 발생한 것입니다.

```
PS C:\GitHub\nowicx\psets\pset-mid-cirque> g++ cirque.cpp driver.cpp -I../../include -o cirque
C:/msys64/mingw64/bin/../lib/gcc/x86_64-w64-mingw32/13.2.0/../../../../x86_64-w64-mingw32/bin/l
d.exe: C:\Users\User\AppData\Local\Temp\ccp8mykx.o:driver.cpp:(.text+0x795): undefined referenc
e to `resize(CircularQueue*, int)'
C:/msys64/mingw64/bin/../lib/gcc/x86_64-w64-mingw32/13.2.0/../../../../x86_64-w64-mingw32/bin/l
d.exe: C:\Users\User\AppData\Local\Temp\ccp8mykx.o:driver.cpp:(.text+0x7c7): undefined referenc
e to `clear(CircularQueue*)'
collect2.exe: error: ld returned 1 exit status
PS C:\GitHub\nowicx\psets\pset-mid-cirque>
```

- cirque.cpp 에 위에서 언급한 정의되지 않은 함수들을 skeleton 형식으로 일단 추가하고, 나중에 코딩을 하십시오, 그러면, 실행 파일 cirque.exe 를 만들어 실행할 수 있고, 그러면, 다음과 같은 결과가 나타날 것입니다. 메뉴에 있는 명령어들 중에서 resize(), clear()을 제외하고 작동하거나 부분적으로 작동할 것입니다.

```
PS C:\GitHub\nowicx\psets\pset-mid-cirque> ./cirquex

        Qstat: front:back=[-1:-1], size=0, maxlen=2, show n=32
        Items: [ - - ]
        Queue: [  ]

        e - enqueue(s)    O(1)          r - resize queue/maxlen
        E - enqueue n     O(n)          c - clear queue
        d - dequeue       O(1)          s - show n items
        D - dequeue n     O(n)

        Enter a command[q to quit]:
```

함수 resize() 구현하기 (4 점)

함수 resize()는 “resize queue/maxlen" 메뉴에서 사용하고, 큐의 크기 즉 maxlen 을 재설정함으로, queue item 을 저장하는 items[]를 확장 혹은 축소합니다.

- 사용자로부터 새로운 큐의 크기를 입력 받음으로 시작합니다.
  기존의 큐의 배열 items 를 free 하고 새로운 크기(maxlen)의 items 를 생성합니다.
- 새로운 큐의 크기가 더 커질 경우, 모든 queue item 들은 모두 확장된 queue 에 모두 옮겨질 때 items[0]부터 채워집니다. Front = 0 이 됩니다.
- 새로운 큐의 크기가 더 작아질 경우, 모든 queue item 들이 축소된 queue 에 모두 옮겨질 수 없을 정도로 줄어든다면, 가장 최근에 enqueue 된 item 들을 유지하고 오래된 item 들은 버려야 합니다.
- 빈 큐에는 dash 문자를 사용하여 채워 놓습니다. Hard coding("-")을 사용하지 말고, CircularQueue 객체에 저장되어 있는 dash 를 사용하십시오.

함수 clear() 구현하기 (1 점)

함수 clear()는 clear queue 메뉴에서 사용하며, 큐의 내용을 모두 삭제하고, 시작하는 상태로 설정합니다

- 기존의 큐의 maxlen 를 그대로 유지하면서, 큐에 입력된 item 들을 모두 지웁니다.
  큐의 크기가 0 가 되므로, front = back = -1 이 되어야 합니다.
- 빈 큐에는 dash 문자를 사용하여 채워 놓습니다. Hard coding("-")을 사용하지 마십시오,

두 함수, show_items() & show_queue() 수정하기 (2 점)

“show n items”은 사용자가 최대 몇 개의 queue items 들을 화면에 보여줄 수 있는지 조정할 수 있습니다. 때때로, 사용자가 백만개의 요소를 가진 큐를 테스트할 때 유용할 것입니다.

show_queue(), show_items()에서 queue 의 크기 즉 items 의 크기가 지나치게 클 때, 즉 shown 의 값에 따라, 최대로 출력할 수 있는 queue items 의 개수를 조정할 수 있습니다.

- 예를 들면, maxlen=50 이고, shown=20 이면,
- 큐 items[]에서 최대 20 개를 보여줍니다. Items[] 배열의 앞부분 10 개와 뒷부분 10 개를 출력하고, 그 중간에 생략 부호를 출력합니다.
- 큐 items[]에서 queue size 가 shown 보다 많다면, 여기 역시 큐의 앞부분(front)에서 10 개 뒤부분(back)에서 최대 10 개를 출력합니다.

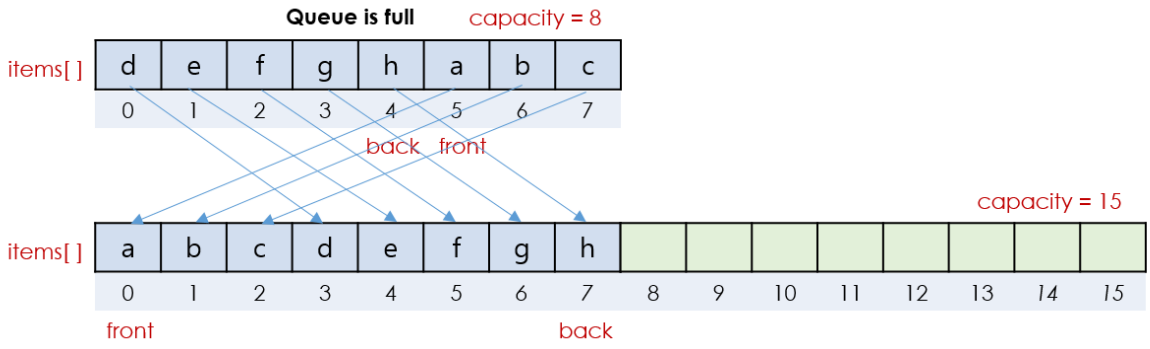
```
Qstat: front:back=[43:4], size=12, maxlen=50, show n=20
Items: [ p g g x r - - - - - - ... - - - r t k j p r e ]
Queue: [ r t k j p r e p g g x r ]

e - enqueue(s)    O(1)          r - resize queue/maxlen
E - enqueue n     O(n)          c - clear queue
d - dequeue       O(1)          s - show n items
D - dequeue n     O(n)

Enter a command[q to quit]:
```

```
Qstat: front:back=[43:16], size=24, maxlen=50, show n=20
Items: [ p g g x r a b c d e ... - - - r t k j p r e ]
Queue: [ r t k j p r e p g g ... c d e f g h i j k l ]

e - enqueue(s)    O(1)          r - resize queue/maxlen
E - enqueue n     O(n)          c - clear queue
d - dequeue       O(1)          s - show n items
D - dequeue n     O(n)

Enter a command[q to quit]:
```

참고사항:

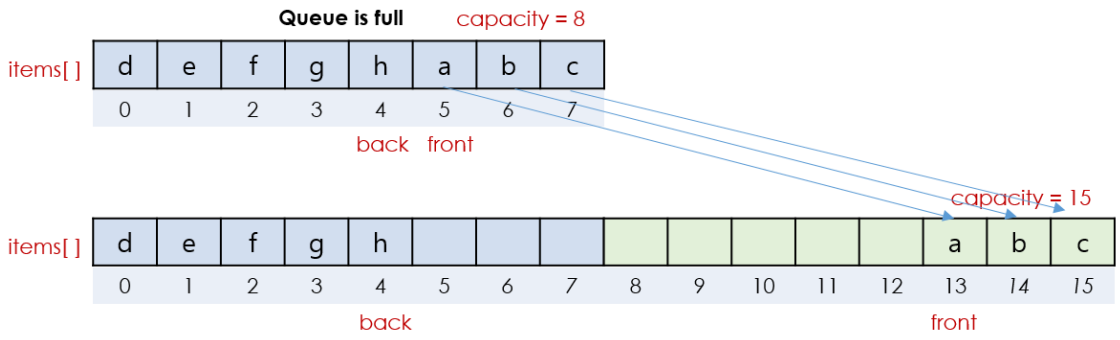
큐 배열(items)을 확장할 때, 두 가지 방법으로 배열을 조정할 수 있습니다. 첫째 방법은 기존의 큐를 원소들을 모두 앞으로 재정렬하여 모든 원소들을 재조정하는 방법입니다. 제공되는 실행파일에서는 Case 1 의 방법으로 구현되어 있습니다.

CASE 1:



Case II: back 과 front 사이에 공간이 생기며, 데이터 움직임을 최소로 하는 방법입니다. 두번째 방법은 back 과 front 사이 공간이 생기도록 front 를 새로 할당된 뒷부분으로 재조정하는 방법입니다.

다음 그림은 capacity = 8 인 큐가 포화상태가 되어 큐의 크기를 2 배로 조정하고 front 부터 배열의 끝까지 원소들을 찾아 새로 할당된 뒷부분에 위치하게 됩니다. Items 를 새로 정렬할 수도 있지만, 시간을 절약하기 위해, front 부터 끝까지 부분의 items 를 새로 생성한 배열의 끝에 복사/이동하는 방법으로 구현합니다.



과제 제출

- **On my honour, I pledge that I have neither received nor provided improper assistance in the completion of this assignment.**
  서명: _____ 학번: _____
- 제출하기 전에 코드가 제대로 컴파일이 되고 실행되는지 확인하세요. 제출 직전에 급하게 코드를 수정한 후 코드가 컴파일이 될 거라고 짐작하지 않는 게 좋습니다. "거의" 작동하는 코드도 틀린 것입니다.
- 과제가 컴파일 및 실행된다면, 마감 기한 전까지 과제의 일부만 완성했더라도 제출하기 바랍니다. 마감 시간 이후 24 시간 이내 제출하면, 만점에서 25% 감점하고 채점합니다. 그 이상 늦은 것은 채점하지 않으며, 0 점 처리합니다.
- 제출 후, **마감 기한 전까지** 수정 및 재제출이 가능합니다. 파일 하나만 수정하더라도 해당 파일과 관련된 파일들을 모두 재제출해야 합니다. 재제출 횟수는 제한 없습니다. 마감 기한 전에 **가장 마지막으로** 제출된 파일을 채점할 것입니다.

제출 파일 목록

아래에 명시된 파일을 Piazza 폴더에 제출하세요.

- **Step 1: cirque1.cpp 2 점**
- **Step 2: cirque2.cpp 1 점**
- **Step 3: cirque.cpp 7 점**

마감 기한 & 배점

- 마감 기한: 11:55 pm