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 1
문제 목적과 진술 1
Step 1: Static Circular Queue - quecir1.cpp 1
Step 2: Dynamic Circular Queue - quecir2.cpp 4
Step 3: Circular Queue - quecir3.cpp, driver.cpp & quecir.h 5
Step 4: Circular Queue - quecir4.cpp, driver.cpp & quecir.h 8
Step 5 [Optional]: Circular Queue - quecir5.cpp ..... 오류! 책갈피가 정의되어 있지 않습니다.
과제 제출 9
제출 파일 목록 9
마감 기한 & 배점 9

제공되는 파일 목록

- quecir.pdf this file
- driver.cpp tests this the circular queue code interactively and rigorously.
- quecir.h include file
- quecirx.exe a solution for checking

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 - quecir1.cpp

우리는 강의 시간에 배운 Circular Queue 에 대한 충분한 이해가 있어야 합니다. 수업에서는 크기가 정해진 Circular Queue 를 가지고 코딩을 했는데, 아래 주어진 코드와 아주 유사할 것입니다. 여기서 유의할 것은

CircularQueue 데이터 구조의 Static 하며 동적할당을 사용하지 않는다는 것입니다. 큐의 내용들은 모두 string type 데이터입니다. 테스트 코드와 기대하는 결과값이 주어졌습니다.

- 다음 코드를 copy and paste 하여 quecir1.cpp 라고 이름하십시오.
- 함수 display()에 있는 버그를 수정하십시오.
- 함수 assert( )구문의 괄호 안에 logical expression 을 넣어 코드를 완성하십시오.
  함수 assert() 에 주어진 논리식이 true 이면, 코드가 자연스럽게 다음 줄로 넘어갑니다. 그러나 만약, 이 논리식이 false 가 되면, 실행파일이 멈추고, assert 에러 메시지를 출력하며 파일이름과 줄번호를 출력합니다.

```
#include <iostream>
#include <sstream>
#include <string>
#include <cassert>

using namespace std;

// size of circular queue, a magic number
const int SIZE = 5;
struct CircularQueue {
  string items[SIZE];
  int front;
  int back;
};

using quecir = CircularQueue *;

quecir newCircularQueue(){
  quecir q = new CircularQueue;
  q->front = -1;
  q->back = -1;
  return q;
}

bool full(quecir q){
  if (q->front == 0 && q->back == SIZE - 1) return true;
  if (q->front == q->back + 1) return true;
  return false;
}

bool empty(quecir q){
  return q->front == -1;
}

void enqueue(quecir q, string item){
  if(full(q)){
    cout << "Queue is full" << endl;
    return;
  }

  if(q->front == -1) q->front = 0;
  q->back = (q->back + 1) % SIZE;
  q->items[q->back] = item;
  cout << "enqueued: " << item << endl;
}

string dequeue(quecir q){
    if (empty(q)){
        cout << "Queue is empty" << endl;
        return "";
```

```
    }

    String item = q->items[q->front];
    if(q->front == q->back){
        q->front = -1;
        q->back = -1;
    } // q has only one item, we reset the q after deleting it.
    else {
        q->front=(q->front + 1) % SIZE;
    }
    return item;
}

void display(quecir q) { // display quecir status
    int i;
    if(empty(q))
        cout << endl << "Empty Queue" << endl;
    else {
        cout << "Front:Back = [" << q->front << ":" << q->back << "]\n";
        cout << "Items:[ ";
        for(i = q->front; i != q->back; i = i + 1)
            cout << q->items[i] << ' ';
        cout << q->items[i];
        cout << " ]\n";
    }
}
```

테스트 코드는 다음과 같습니다.

```
int main() {
    quecir q = newCircularQueue();
    dequeue(q);
    enqueue(q, "a");
    enqueue(q, "b");
    enqueue(q, "c");
    enqueue(q, "d");
    enqueue(q, "e");
    enqueue(q, "f");
    display(q);
    string item = dequeue(q);
    cout << "dequeued: " << item << endl;
    display(q);
    enqueue(q, "g");
    display(q);
    enqueue(q, "h");
    dequeue(q);
    dequeue(q);
    display(q);
    return 0;
}
```

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

```
g++ quecir1.cpp -o quecir1
./quecir1
```

```
Queue is empty
enqueued: a
enqueued: b
enqueued: c
```

```
enqueued: d
enqueued: e
Queue is full
Front:Back = [0:4]
Items:[ a b c d e ]
dequeued: a
Front:Back = [1:4]
Items:[ b c d e ]
enqueued: g
Front:Back = [1:0]
Items:[ b c d e g ]
Queue is full
Front:Back = [3:0]
Items:[ d e g ]
```

Step 2: Dynamic Circular Queue - quecir2.cpp

아래와 같이 CircularQueue 구조의 items 의 크기를 SIZE = 5 로 고정하는 magic number 를 사용하지 않기 위하여, 큐의 크기가 배열의 크기와 일치하여 full 상태에서 또 다른 큐가 입력이 되면, 자동적으로 items 배열의 크기(capa)를 2 배로 확장하는 코딩을 수행합니다.

또한 dequeue 를 한 직후, 큐의 크기(size)가 체크하여 그 크기가 capa 의 1/4(one quarter full)로 되었다면, items 의 배열의 크기를 반으로 줄여야 합니다. 이것은 Step3 에서 코딩합니다.

```
// size of circular queue, a magic number
// const int SIZE = 5;
```

- quecir1.cpp 파일을 복사하여 quecir2.cpp 라 이름합니다.
- 이를 위하여 아래와 같이 items 를 포인터로 수정하고, items[] 배열의 크기를 나타내는 capa 를 추가하십시오.

```
. . .

// size of circular queue, a magic number
// const int SIZE = 5;
struct CircularQueue {
  string *items;
  int front, int back;
  int capa;               // total capacity of the queue
};

using quecir = CircularQueue *;
. . .
```

- **Size() & resize() 함수 구현:**
  이제 enqueue()함수를 수정하여 Queue 가 Full 상태가 되면, items[]배열의 크기를 조정할 수 있도록 resize()함수를 구현하십시오.
  또 이를 현재 queue 의 items 배열 크기를 동적으로 구하는 size()함수도 구현하십시오.

  주의할 것은 size()함수는 front 와 back 을 사용하여 계산하여 반환하는 함수이어야 합니다. 함수 resize()는 stack 관련 수업 시간에 배우고, 제공된 resize() 함수를 활용하면 됩니다.

```
int size(quecir q) {
  if (q->front == -1) return 0;
  ...
}

void resize(quecir q, int new_capacity) {
  string *copied = new string[new_capacity];
  int qsize = size(q);                    // current q size. . .
  ...
```

다음은 quecir1.cpp 에서 사용한 간단한 테스트를 quecir2.cpp 에 적용하여 실행한 경우입니다. 더 이상 “Queue is full” 메시지가 나오지 않는 것을 관찰할 수 있습니다.

```
int main() {
    quecir q = newCircularQueue();
    dequeue(q);
    enqueue(q, "a");
    enqueue(q, "b");
    enqueue(q, "c");
    enqueue(q, "d");
    enqueue(q, "e");
    enqueue(q, "f");
    display(q);
    string item = dequeue(q);
    cout << "dequeued: " << item << endl;
    display(q);
    enqueue(q, "g");
    display(q);
    enqueue(q, "h");
    dequeue(q);
    dequeue(q);
    display(q);
    return 0;
}
```

```
Queue is empty
Front:Back = [0:5]
Items:[ a b c d e f ]
dequeued: a
Front:Back = [1:5]
Items:[ b c d e f ]
Front:Back = [1:6]
Items:[ b c d e f g ]
Front:Back = [3:7]
Items:[ d e f g h ]
```

Step 3: Circular Queue - quecir3.cpp, driver.cpp & quecir.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 봅니다. 학우들은 이미 만들어 놓은 include 파일을 활용하면 됩니다.

- Quecir2.cpp 를 복사하여 quecir3.cpp 를 만드십시오.
  nowic/include/quecir.h 를 quecir3.cpp 포함하십시오. 그리고, 이미 ~.h 에 정의된 부분을 quecir3.cpp 에서 제거 하십시오.

newCircularQueue()의 디폴트 파라미터도 ~.h 에 정의된 것을 사용해야 하므로, capa 디폴트 값을 제거하고, 나중에 사용할 int shown 인자도 추가하십시오.

- 함수 main()을 quecir3.cpp 에서 제거하십시오.
- 함수 display()함수를 show_queue()로 이름을 수정하고, 그 안에 "Items: [" 를 "\tQueue:["로 **대체하십시오.**
- 그리고, 다음과 같은 명령어(각자의 개발환경에 따라 path 가 다를 수 있음)로 컴파일을 하면 **에러 메시지가 출력됩니다**..

```
g++ quecir3.cpp driver.cpp -I../../include -o quecir
```

이를 관찰해보면 다음과 같은 함수들이 undefined 되었다는 것을 알 수 있습니다.

```
undefined reference to `show_setup(CircularQueue*)
undefined reference to `show_items(CircularQueue*)
undefined reference to `clear(CircularQueue*)
undefined reference to `free(CircularQueue*)
```

- 위의 함수들을 quecir3.cpp 에 추가하여 실행 파일 quecir.exe 를 만들어 실행하면, 다음과 같은 결과가 나타날 것입니다. 메뉴에 있는 명령어들 중에서 clear, profile, show n items 를 제외하고 작동하거나 부분적으로 작동할 것입니다.

```
PS C:\GitHub\nowicx\psets\pset-midterm> g++ cqueue3.cpp cqueuedriver.cpp -I../../include -o cqueue3
PS C:\GitHub\nowicx\psets\pset-midterm> ./cqueue3


Empty Queue

        e - enqueue ...     O(1)        p - profile
        E - enqueue n       O(n)        r - resize method
        d - dequeue         O(1)        R - reset  queue
        D - dequeue n       O(n)
        c - clear           O(n)        s - show n items

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

아래와 같이 기존의 CircularQueue 구조에 추가적으로 도입된 멤버들이 있습니다.

```
struct CircularQueue {
  string *items;
  int front, back;
  int capa;           // total capacity of the queue
  string dash;        // dash - a place holder string
  bool doubling;      // if true, increase capacity by 2x, otherwise by 10
  int incr;           // if doubling is false, increase capacity by incr
  int shown;          // the max number of items to show
};
```

- CircularQueue 구조에서 추가된 멤버들은 **다음 Step 에서** 적극적으로 구현합니다.
  - dash 는 items 배열에 queue 가 저장되지 않은 것을 표시하는 한 글자이며, "-"를 사용합니다.
  - doubling 은 디폴트로 true 이며, 이 값이 true 이면, items[] 를 resize()에서 재조정할 때 successive doubling 알고리즘을 사용하고, false 일 때는 incr 에 저장되어 있는 1, 10, 100, 1000 등의 값을 사용합니다.

doubling 이 false 일 때, capacity 를 줄이는 방법은 incr * 2 로 빈 공간이 생길 때, capacity 를 incr 만큼 줄입니다.

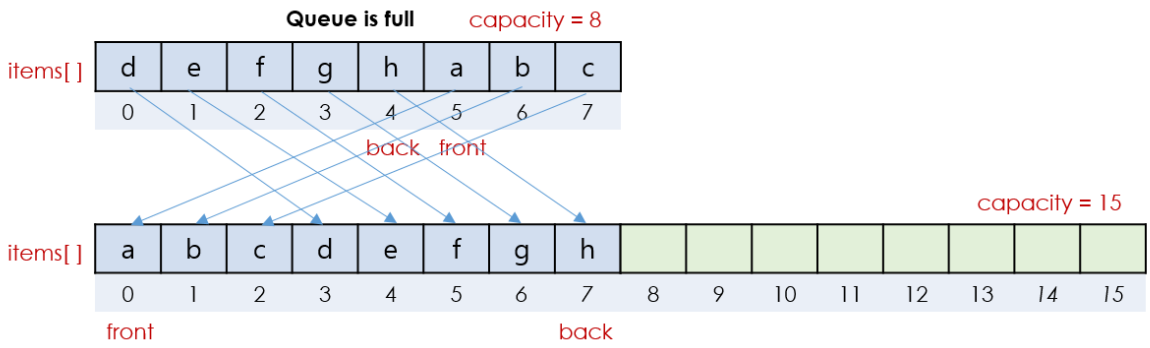
- incr 는 doubling 이 false 이며, items[]를 resize 할 때 사용하는 increment 입니다.
- shown 은 show_queue(), show_items()에서 queue 의 크기 즉 items 의 크기가 지나치게 클 때, 사용자의 요청에 따라 최대로 출력할 수 있는 queue 의 개수를 의미합니다.

- clear()함수를 작성하십시오.
  - 이 함수는 clear 명령에서 사용합니다.
  - 기존의 큐의 capacity 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.
- reset()함수를 작성하십시오.
  - reset queue 명령에서 사용합니다
  - 기존의 큐의 배열 items 를 free 하고 새로운 크기(capacity)의 items 를 생성합니다.
  - 물론, 큐의 크기가 0 가 되므로, front = back = -1 이 되어야 합니다.
  - 빈 큐에는 dash 문자를 사용하여 채워 놓습니다.
- free()함수를 작성하십시오.
  - 이 함수 사용자가 q 명령어를 사용하여 프로그램을 종료할 때 한번 사용됩니다.
  - 동적으로 할당된 모두 메모리를 free 합니다.
- **resize()함수를 완성합니다.**
  - **Step 3 에서 가장 중요한 코딩 부분입니다.**
  - dequeue 를 한 직후, 큐의 크기(size)가 체크하여 그 크기가 capa 의 1/4(one quarter full)로 되었다면, items 의 배열의 크기를 반으로 줄이는 코딩을 완성하고, 테스트하십시오.

참고사항:

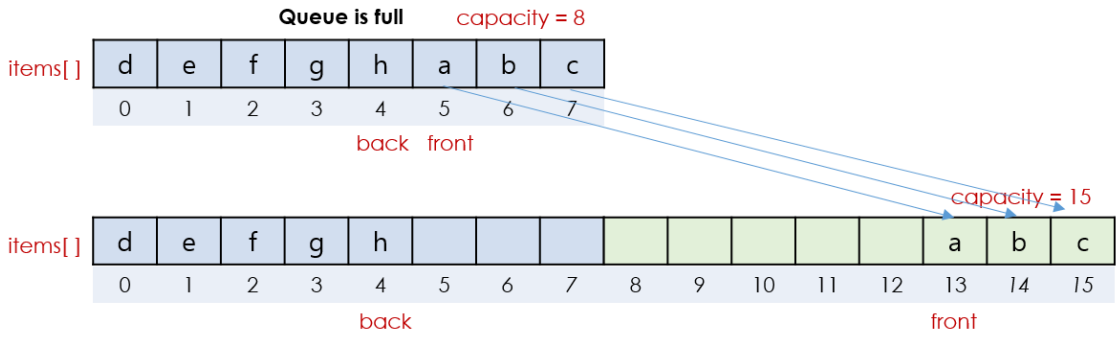
큐 배열(items)을 Successive doubling 으로 확장할 때, 두 가지 방법으로 배열을 조정할 수 있습니다. 첫째 방법은 기존의 큐를 원소들을 모두 앞으로 재정렬하여 모든 원소들을 재조정하는 방법입니다. 제공되는 실행파일에서는 Case 1 의 방법으로 구현되어 있습니다.

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.



Step 4: Circular Queue - quecir4.cpp, driver.cpp & quecir.h

지금까지 구현한 것을 Circular Queue 를 테스트하려면 어려운 것은 아니지만 쉽지도 않습니다. 지루한 작업입니다. 테스트를 돕기 위해 몇가지 기능을 추가하여 현재 Queue 의 상태를 보여주고자 합니다.

지금까지 show_queue()함수에서 시행하는 디스플레이 기능을 세 부분을 나누고자 합니다. 하나는 show_setup()이고, 다른 하나는 show_items()입니다. 이를 사용하여 궁극적으로는 아래와 같은 일종의 대시보드를 보여주고 싶은 것입니다.

```
PS C:\GitHub\nowicx\psets\pset-midterm> g++ quecirx.cpp driver.cpp -I../../include -o quecirx
PS C:\GitHub\nowicx\psets\pset-midterm> ./quecirx

        Setup: front:back=[-1:-1], size=0, capacity=4, resize by 2x, show n=32
        Queue: [  ]
        Items: [ - - - - ]

        e - enqueue(s)    O(1)          r - resize(2x or 1,10,100,...)
        E - enqueue n     O(n)          R - reset queue
        d - dequeue       O(1)          s - show n items
        D - dequeue n     O(n)          c - clear

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

```
Setup: front:back=[2:4], size=3, capacity=8, resize by 2x, show n=32
Queue: [ q g h ]
Items: [ - - q g h - - - ]

e - enqueue(s)    O(1)          r - resize(2x or 1,10,100,...)
E - enqueue n     O(n)          R - reset queue
d - dequeue       O(1)          s - show n items
D - dequeue n     O(n)          c - clear

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

여기서 show_setup()은 첫 줄에 나타나며, 현재 큐의 상태를 보여줍니다.

“resize(2x or 1, 10, 100 …)”는 Queue 를 크기를 확장하거나 줄이는 토글링(toggling)방법을 선택합니다. 즉 현재 successive doubling(2x) 상태에서, 사용자가 resize 명령을 선택하면, 사용자는 [1,

10, 100, …] 같은 고정된 resize 의 크기를 입력할 수 있고, 현재 고정된 resize 의 크기가 설정되어 있다면, 자동적으로 successive doubling(2x)로 설정하게 됩니다.

“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.

이들을 구현하고, 특별히 resize, show n items 를 테스트 해보십시오. 그러면, 자동적으로 enqueue/dequeue 도 자주 사용하게 되면서 집중적인 테스트를 하게 됩니다. 예를 들면, 아래 화면은 100 개 원소를 무작위로 enqueue 하고, 첫 다섯개를 dequeue 한 경우입니다.

```
Enter a command[q to quit]: D
Enter a number of items: 5

Setup: front:back=[5:99], size=95, capacity=128, resize by 2x, show n=32
Queue: [ l n l f d x f i r c v s c x g g ... w s r e n z k y c x f x t l s g ]
Items: [ - - - - - l n l f d x f i r c v ... - - - - - - - - - - - - - - - ]

e - enqueue(s)    O(1)          r - resize(2x or 1,10,100,...)
E - enqueue n     O(n)          R - reset queue
d - dequeue       O(1)          s - show n items
D - dequeue n     O(n)          c - clear

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

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: quecir1.cpp 1 점**
- **Step 2: quecir2.cpp 1 점**
- **Step 3: quecir3.cpp 2 점**
- **Step 4: quecir4.cpp 2 점**

마감 기한 & 배점

- 마감 기한: 11:55 pm