Karadeniz Teknik Üniversitesi
Bilgisayar Mühendisliği Bölümü
Öğr.Gör. Ömer ÇAKIR

BIL 205 Veri Yapıları
1. Arasınav, 23.11.2012, 15:00, D-2, D-9
Süre : **100** Dakika

CEVAPLAR

```
void insertOrdered(const string& e, const int& i)
{
  DoublyNode* newNode  = new DoublyNode;
  newNode->elem        = e;    newNode->score = i;
  newNode->next        = NULL; newNode->prev  = NULL;

  if(header->next == trailer)
  {//BLOCK 1
    newNode->next      = trailer;
    newNode->prev      = header;
    header->next       = newNode;
    trailer->prev      = newNode;
    return;
  }

  DoublyNode* current  = header->next;
  if(newNode->score < current->score)
  {//BLOCK 2
    newNode->next      = current;
    newNode->prev      = current->prev;
    current->prev      = newNode;
    current->prev->next= newNode;
    return;
  }

  DoublyNode* founded  = NULL;
  while (current != trailer)
  {
    if(newNode->score>=current->score) founded=current;
    else break;
    current = current->next;
  }
  //BLOCK 3
  newNode->next        = founded->next;
  newNode->prev        = founded;
  founded->next        = newNode;
  founded->next->prev  = newNode;
}
```

1. Elemanları çift yönlü bağlı listeye sıralı ekleyen yukarıdaki `insertOrdered()` adlı fonksiyonda **koyu** yazılmış kod bloklarının birinde veya birkaçında satırların yerleri değiştirildiği için program koştuğunda hata vermektedir. Bu blok veya blokların doğrusunu aşağıya yazınız. **(20P)**

BLOCK 1 CORRECT SEQUENCE	
BLOCK 2 CORRECT SEQUENCE	
newNode->next	= current;
newNode->prev	= current->prev;
current->prev->next	= newNode;
current->prev	= newNode;
BLOCK 3 CORRECT SEQUENCE	
newNode->next	= founded->next;
newNode->prev	= founded;
founded->next->prev	= newNode;
founded->next	= newNode;

```
CircularlyLinkedList C;

void CircularlyLinkedQueue::enqueue(const string& e)
{
        // Kodu buraya yazınız !
        C.add(e);
        C.advance();

        n++;
}

void CircularlyLinkedQueue::dequeue()
{
        if (empty())
        {
                cout << "dequeue of empty queue" << endl;
                return;
        }
        // Kodu buraya yazınız !
        C.remove();

        n--;
}
```

2. Kuyruk veri yapısına ait `enqueue()` ve `dequeue()` fonksiyonlarını, C dairesel bağlı listesinin `add()`, `advance()` ve `remove()` fonksiyonları ile gerçeklemek üzere yukarıya gerekli kodları yazınız. Yazdığınız kodları açıklayınız. **(20P)**

Dairesel bağlı listede cursor listenin sonuna (back) işaret ediyordu. Hem ekleme (add) hem de silme (remove) işlemi cursor->next ile yapılıyordu. Yani cursor'ın peşine ekleniyor veya peşine gelen siliniyordu. Bu bilgiler ışığında :

`enqueue()` implementation using C circularly linked list functions :
Kuyruk veri yapısında veri ekleme (enqueue) kuyruk sonuna yapıldığından dairesel bağlı listenin add fonksiyonunun peşine, cursor yeni eklenene son eleman olarak işaret etsin diye advance de çağrılmıştır.
`dequeue()` implementation using C circularly linked list functions :
Kuyruk veri yapısında veri silme (dequeue) kuyruk başından yapıldığından dairesel bağlı listenin remove fonksiyonu ile birebir örtüşür. Çünkü dairesel listede de cursor->next yani ilk eleman silinmektedir.

```
int tripleSum(int A[], int i, int n)
{
   if (n == 1) return A[i];
   else
   {
        int Sum =     tripleSum(A, i + 2*n/3, n/3 )
                    + tripleSum(A, i + n/3, n/3 )
                    + tripleSum(A, i, n/3 );
        cout << "Sum = " << Sum << endl;
        return Sum;
   }
}

void main()
{
   int A[27] = { 1,2,3,4,5,6,7,8,9,
                 10,11,12,13,14,15,16,17,18,
                 19,20,21,22,23,24,25,26,27 };

   tripleSum(A, 0, 27);
}
```

3. Yukarıdaki programın çıktısı nedir? **(30P)**

İpucu→ Program ekrana **13** kere `Sum = ...` yazar.
[1..27] arası sayıların toplamı **378**'dir.
Toplarken işlem hatası yapmamaya dikkat ediniz !

Sum = 78
Sum = 69
Sum = 60
Sum = 207
Sum = 51
Sum = 42
Sum = 33
Sum = 126
Sum = 24
Sum = 15
Sum = 6
Sum = 45
Sum = 378

```
void reverseList(DoublyNode* hNext,DoublyNode* tPrev)
{
   if ( hNext == tPrev ) return;

   if(hNext->next == tPrev)
   {
      hNext->next = tPrev->next;
      tPrev->next->prev = hNext;

      tPrev->prev = hNext->prev;
      hNext->prev->next = tPrev;

      hNext->prev = tPrev;
      tPrev->next = hNext;

      return;
   }
   else
   {
      DoublyNode* hNextNext = hNext->next;
      DoublyNode* hNextPrev = hNext->prev;

      hNext->next = tPrev->next;
      hNext->prev = tPrev->prev;
      tPrev->prev->next = hNext;
      tPrev->next->prev = hNext;

      tPrev->next = hNextNext;
      tPrev->prev = hNextPrev;
      hNextPrev->next = tPrev;
      hNextNext->prev = tPrev;

      return reverseList( tPrev->next, hNext->prev);
   }
}
```

4.

a) Yukarıda verilen **rList()** fonksiyonu main() fonksiyonunda **rList(header->next, trailer->prev)** parametreleri ile çağrıldığında ne iş yapar? **(15P)**

Fonksiyon liste elemanlarını reverse yapar. Yani ters sırada dizer.

b) **rList()** fonksiyonu kendini recursive olarak neden **rList(tPrev->next, hNext->prev)** parametreleri ile çağırmıştır? **(15P)**

İlk recursive çağrı öncesi listenin ilk ve son elemanları yer değiştirildikten sonra baştan ve sondan ikinciler ile recursive çağrı yapılmalıdır. İlk elemana işaret eden hNext artık son elemana; son elemana işaret eden tPrev de ilk elemana işaret etmektedir. O yüzden recursive çağrıda ilk parametre listenin baştan 2. elemanını göstermek üzere tPrev->next, ikinci parametre de listenin sondan 2. elemanını göstermek üzere hNext->prev olmuştur.