3. An electric car that runs on batteries must be periodically recharged for a certain number of hours. The battery technology in the car requires that the charge time not be interrupted.

   The cost for charging is based on the hour(s) during which the charging occurs. A rate table lists the 24 one-hour periods, numbered from 0 to 23, and the corresponding hourly cost for each period. The same rate table is used for each day. Each hourly cost is a positive integer. A sample rate table is given below.

Hour	Cost
0	50
1	60
2	160
3	60
4	80
5	100
6	100
7	120

Hour	Cost
8	150
9	150
10	150
11	200
12	40
13	240
14	220
15	220

Hour	Cost
16	200
17	200
18	180
19	180
20	140
21	100
22	80
23	60

The class `BatteryCharger` below uses a rate table to determine the most economic time to charge the battery. You will write two of the methods for the `BatteryCharger` class.

```
public class BatteryCharger
{
  /** rateTable has 24 entries representing the charging costs for hours 0 through 23. */
  private int[] rateTable;

  /** Determines the total cost to charge the battery starting at the beginning of startHour.
   *  @param startHour the hour at which the charge period begins
   *         Precondition: 0 ≤ startHour ≤ 23
   *  @param chargeTime the number of hours the battery needs to be charged
   *         Precondition: chargeTime > 0
   *  @return the total cost to charge the battery
   */
  private int getChargingCost(int startHour, int chargeTime)
  {  /* to be implemented in part (a) */  }

  /** Determines start time to charge the battery at the lowest cost for the given charge time.
   *  @param chargeTime the number of hours the battery needs to be charged
   *         Precondition: chargeTime > 0
   *  @return an optimal start time, with 0 ≤ returned value ≤ 23
   */
  public int getChargeStartTime(int chargeTime)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour the hour at which the charge period begins
 *         Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
```

(b) Write the `BatteryCharger` method `getChargeStartTime` that returns the start time that will allow the battery to be charged at minimal cost. If there is more than one possible start time that produces the minimal cost, any of those start times can be returned.

For example, using the rate table given at the beginning of the question, the following table shows the resulting minimal costs and optimal starting hour of several possible charges.

Hours of Charge Time	Minimum Cost	Start Hour of Charge	Last Hour of Charge
1	40	12	12
2	110	0 **or** 23	1 **or** 0 (the next day)
7	550	22	4 (the next day)
30	3,710	22	3 (two days later)

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
```

2009 A Question 3: Battery Charger — Assessment Rubric

Part A: `getChargingCost` 5 pts

+1½ access array elements
- **+½** accesses any element of `rateTable`
- **+½** accesses an element of `rateTable` using an index derived from `startHour`
- **+½** accesses multiple elements of `rateTable` with no out of bounds access potential

+2½ accumulate values
- **+½** declares and initializes an accumulator
- **+½** accumulates values from elements of `rateTable`
- **+½** selects values from `rateTable` using an index derived from `startHour` and `chargeTime`
- **+1** determines correct sum of values from `rateTable` based on `startHour` and `chargeTime`

+1 value returned
- **+½** returns any non-constant (derived) value
- **+½** returns accumulated value

Part B: `getChargeStartTime` 4 pts

+½ invokes `getChargingCost` or replicates functionality with no errors

+1 determine charging cost
- **+½** considers **all** potential start times; must include at least 0 … 23
- **+½** determines charging cost for potential start times

Note: No penalty here for parameter passed to `getChargingCost` *that violates its preconditions (e.g., 24)*

+1 compares charging costs for two different start times

+1 determines minimum charging cost based on potential start times

Note: Penalty here for using result of call to `getChargingCost` *that violates its preconditions (e.g., 24)*

+½ returns start time for minimum charging cost

2009 A Question 3: Battery Charger — Canonical Solution

PART A:

```
/** Determines the total cost to charge the battery starting
 *    at the beginning of startHour.
 *  @param startHour the hour at which the charge period begins
 *         Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime) {
  int cost = 0;
  for (int x = 0; x < chargeTime; x++) {
    cost += this.rateTable[(startHour + x) % 24];
  }
  return cost;
}
```

PART B:

```
/** Determines start time to charge the battery at the lowest
 *    cost for the given charge time.
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime) {
  int startTime = 0;
  for (int i = 1; i < 24; i++) {
    if (this.getChargingCost(i, chargeTime)
        < this.getChargingCost(startTime, chargeTime)) {
      startTime = i;
    }
  }
  return startTime;
}
```

2009 A Question 3 (Battery Charger) Folder # Reader #

Part A (5 pts) getChargingCost									Part B (4 pts) getChargeStartTime								
access array			accumulate values				value ret			det cost							
any ½	index start hour ½	oob multi ½	dec/ init ½	accum ½	sel ½	deter sum 1	any deriv ½	accum val ½	get Charg ½	cons all ½	det ½	compares costs 1	deter min cost 1	ret start ½	**Usage Max -3** part/reason	**Booklet Number**	**Score**

Wed Thu Fri Sat Sun Mon Tue

am pm

6/13/2009 15:14

2009 A Question 3 (Battery Charger) Folder # Reader #

Part A (5 pts) getChargingCost									Part B (4 pts) getChargeStartTime								
access array			accumulate values				value ret			det cost					Usage Max -3	Booklet	
any ½	index start hour ½	oob multi ½	dec/ init ½	accum ½	sel ½	deter sum 1	any deriv ½	accum val ½	get Charg ½	cons all ½	det ½	compares costs 1	deter min cost 1	ret start ½	part/reason	Number	Score
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		A3 A	9
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		A3 B	9
✓	✓	✓	0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	0		A3 C	8
✓	✓	0	✓	✓	✓	0	✓	✓	✓	0	✓	✓	0	✓		A3 D	6
✓	✓	0	0	✓	✓	0	✓	✓	✓	0	✓	✓	0	0		A3 E	5
✓	✓	✓	✓	✓	✓	0	✓	✓	0	0	0	0	0	0		A3 F	4
✓	✓	0	0	0	0	0	0	0	✓	0	✓	✓	0	0		A3 G	3
✓	✓	0	0	0	0	0	✓	0	0	✓	0	0	0	0		A3 H	2
✓	✓	0	0	0	0	0	✓	0	0	0	0	0	0	0		A3 I	2

Wed Thu Fri Sat Sun Mon Tue
am pm

6/13/09 15:54

AP® SUMMER INSTITUTE SCORING NOTES
2009 AP COMPUTER SCIENCE A

Question 3

Sample Identifier: A3 A
Score: 9

- part a: correct
- part b: correct

Sample Identifier: A3 B
Score: 9

- part a: correct
- part b: correct

Sample Identifier: A3 C
Score: 8

- part a: loses 1/2 (dec/init). The accumulator is not initialized.
- part b: loses 1/2 (return start). The local variable `optimalStart` is not initialized and is not guaranteed to have a value when the `return` is executed.

Sample Identifier: A3 D
Score: 6

- part a: loses 1/2 (oob/multi). There is no attempt to wrap around when the loop control variable exceeds 23. This results in an out of bounds error.
- part a: loses 1 point (det sum). The correct sum will not be determined in the case where a wrap around is necessary and there is no attempt to wrap around.
- part b: loses 1/2 point (cons all). Only elements 0..22 are considered.
- part b: loses 1 point (deter min cost). The local variable `lowPrice` is initialized to a constant. There is no guarantee that all charging costs will be lower than this constant.

AP® SUMMER INSTITUTE SCORING NOTES
2009 AP COMPUTER SCIENCE A

Sample Identifier: A3 E
Score: 5

- part a: loses 1/2 point (oob/multi). There is no attempt to wrap around when the loop control variable exceeds 23. This results in an out of bounds error.
- part a: loses 1/2 point (dec/init). The accumulator is not initialized.
- part a: loses 1 point (deter sum). In the loop, the <= should be a < and there is no attempt to wrap around.
- part b: loses 1/2 point (cons all). Only elements 0..22 are considered.
- part b: loses 1 point (deter min cost). The local variable `lowest` is set to a constant. There is no guarantee that all charging costs will be lower than this constant. In fact, in this case, no charging cost will be less than `lowest`.
- part b: loses 1/2 point (ret start). No value is returned.

Sample Identifier: A3F
Score: 4

- part a: loses 1 point (deter sum). In the loop, the <= should be a < . (There is no deduction for confusing the mathematical ≤ with <=.)
- part b: loses 1/2 point (getCharg). Does not invoke `getChargingCost.`
- part b: loses 1/2 point (cons all). Does not consider any start times.
- part b: loses 1/2 point (det). Does not determine any charging cost.
- part b: loses 1 point (compare costs). Never compares charging costs.
- part b: loses 1 point (deter min cost). No attempt to determine the minimum cost.
- part b: loses 1/2 point (ret start). Does not return a start time.

Question 2

Sample Identifier: A3 G
Score: 3

- part a: loses 1/2 point (oob/multi). `cTime` may be greater than 23 which would result in an out of bounds error.
- part a: loses all *accumulates values* points because there is no accumulator.
- part a: loses *value ret* points because no value is returned.
- part b: loses 1/2 point (cons all). Only elements 0..22 are considered.
- part b: loses 1 point (deter min cost). The local variable `big` is not initialized.
- part b: loses 1/2 point (ret start). The start time is not returned.

Sample Identifier: A3 H
Score: 2

- part a: loses 1/2 point (oob/multi). The upper bound of the loop is a value in `rateTable` and will cause an out of bounds error.
- part a: loses all *accumulates values* points because there is no accumulator.
- part a: loses the last 1/2 point (accum val) . The value returned is not an accumulated value.
- part b: loses 1/2 point (getCharg). Does not invoke `getChargingCost.`

- part b: loses 1/2 point (det). Does not determine any charging cost.
- part b: loses 1 point (compare costs). Never compares charging costs.
- part b: loses 1 point (deter min cost). No attempt to determine the minimum cost.
- part b: loses 1/2 point (ret start). Does not return a start time.

Sample Identifier: A3 I
Score: 2

- part a: loses 1/2 (oob/multi). There is no attempt to wrap around when `startHour` exceeds 23. This results in an out of bounds error.
- part a: loses all *accumulates values* points because there is no accumulator.
- part a: loses 1/2 (accum val). The value returned is not an accumulated value.
- part b: loses 1/2 point (getCharg). Does not invoke `getChargingCost` correctly (`time` does not have a value).
- part b: loses 1/2 point (cons all). Does not consider any start times.
- part b: loses 1/2 point (det). Does not determine any charging cost.
- part b: loses 1 point (compare costs). Never compares costs.
- part b: loses 1 point (deter min cost). No attempt to determine the minimum cost.
- part b: loses 1/2 point (ret start). Does not return a start time.

This problem scores a 1 1/2 which is rounded up to a 2.

A3 A

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour the hour at which the charge period begins
 *         Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{
    int cost = 0;
    for (int x = startHour; x < startHour + chargetime ; x ++)
    {
        cost += rateTable[x % 24];
    }
    return cost;
}
```

Part (b) begins on page 12.

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *                     Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    int cheapestHour = 0;

    for (int x = 1; x <= 23; x++)
    {
        if ( getChargingCost(cheapestHour, chargeTime) > getChargingCost(x, chargeTime))
            cheapestHour = x;
    }
    return cheapestHour;
}
```

A3B

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour the hour at which the charge period begins
 *         Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{
  int sum = 0;
  int hour = startHour;
  for(int x = 0; x < chargeTime; x++)
    {
      if (hour == 24)
          hour -= 24;
      sum += rateTable[hour];
      hour++;
    }
  return sum;
}
```

Part (b) begins on page 12.

A3 B

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *                     Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    int low = getChargingCost(0, chargeTime);
    int start = 0;
    for(int x = 1; x < 24; x++)
        if(getChargingCost(x, chargeTime) < low)
        {
            start = x;
            low = getChargingCost(x, chargeTime);
        }
    return start;
}
```

A3C

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour  the hour at which the charge period begins
 *                    Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *                    Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{  int result;
   for (int i = 0; i < chargeTime; i++)
   {
      result += rateTable[(i + startHour) % 24];
   }

   return result;
}
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

A3C

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    int optimalStart;
    int lowCost = getChargingCost(0, chargeTime);
    for (int i=0; i<=23; i++)
    {
        if ( lowCost > getChargingCost (i, chargeTime)
        {
            optimalStart = i;
            lowCost = getChargingCost(i, chargeTime);
        }
    }

    return optimalStart;

}
```

A3 D

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour the hour at which the charge period begins
 *         Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{
    int price = 0;

    for(int i = 0; i < chargeTime; i++)
    {
        price += rateTable[startHour + i];
    }

    return price;

}
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

A3 D

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *                     Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    int startHour = 0;
    int lowPrice = 500000;
    for(int x = 0; x < 23; x++)
    {
        if( lowPrice > getChargingCost(x, chargeTime))
        {
            lowPrice = getChargingCost(x, chargeTime);
            startHour = x;
        }

    }
    return startHour;
}
```

GO ON TO THE NEXT PAGE.

A3E

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour  the hour at which the charge period begins
 *           Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime  the number of hours the battery needs to be charged
 *           Precondition: chargeTime > 0
 *  @return  the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{ int cost;
  for(int i=0; i<=chargeTime; i++)
  { cost += rateTable[startHour+i];
  }
  return cost;
}
```

Part (b) begins on page 12.

A3E

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *                     Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    int time = 0;
    int lowest = 0;
    for (int i = 0; i < 23; i++)
    {
        if(getChargeCost(i, chargeTime) < lowest);
            lowest = getChargeCost(i, chargeTime);
            time = i;
    }
}
```

A3 F

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour  the hour at which the charge period begins
 *                    Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *                    Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
```

```
int total = 0;
for(int k=0; k ≤ chargeTime; k++) {
total = total + rateTable[(startHour+k) % 24];
}
return total;
}
```

Part (b) begins on page 12.

A3 F

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
```

```
if(chargeTime == 1){
  return 12;
}else if(chargeTime == 2){
  return 0;
}else if(chargeTime = 3){
  return 23;
}else if(chargeTime > 3){
  return 22;
}else{
  return -1;
}
```

GO ON TO THE NEXT PAGE.

A3 G

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour  the hour at which the charge period begins
 *                    Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *                    Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
```

```
    int start = startHour;
    start = rateTable[start];
    int cTime = chargeTime;
    cTime = rateTable[cTime];
    charge = start - 24;
    return // values of rateTable[] inclusive
           // from cTime to charge, sum of all
           // values is charging cost.
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a). A3 G

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    int CTIME = ChargeTIME;
    int big;          int temp=0;
    for(int k=0; k< 23; k++)
{
temp= getChargingCost(k, CTIME)
    if(temp<big)
        temp=big;
}
return temp;
```

A3 H

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour the hour at which the charge period begins
 *         Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{
    if ((startHour <= 23 && startHour >= 0) &&
                                    chargeTime > 0)
    {
        startCost = rateTable[startHour];
        for (int i = 0; i < rateTable[startHour + 1]; i++)
        {
    return startCost * rateTable[startHour];
        }
    }
}
```

Part (b) begins on page 12.

A3 H

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    for(int i =0; i < rateTable.length; i++)
    {
        if(chargeTime == 1)
        {
            return rateTable[1];
        }
        else if(chargeTime == 2)
            return rateTable[0];
    }
}
```

A3 I

(a) Write the `BatteryCharger` method `getChargingCost` that returns the total cost to charge a battery given the hour at which the charging process will start and the number of hours the battery needs to be charged.

For example, using the rate table given at the beginning of the question, the following table shows the resulting costs of several possible charges.

Start Hour of Charge	Hours of Charge Time	Last Hour of Charge	Total Cost
12	1	12	40
0	2	1	110
22	7	4 (the next day)	550
22	30	3 (two days later)	3,710

Note that a charge period consists of consecutive hours that may extend over more than one day.

Complete method `getChargingCost` below.

```
/** Determines the total cost to charge the battery starting at the beginning of startHour.
 *  @param startHour  the hour at which the charge period begins
 *                    Precondition: 0 ≤ startHour ≤ 23
 *  @param chargeTime the number of hours the battery needs to be charged
 *                    Precondition: chargeTime > 0
 *  @return the total cost to charge the battery
 */
private int getChargingCost(int startHour, int chargeTime)
{
    int total = 0;
    while (startHour <= chargeTime)
    {
        total = rateTable[startHour] + rateTable[startHour+1];
        startHour++;
    }
    return total;
}
```

Part (b) begins on page 12.

A 3 I

Assume that `getChargingCost` works as specified, regardless of what you wrote in part (a).

Complete method `getChargeStartTime` below.

```
/** Determines start time to charge the battery at the lowest cost for the given charge time.
 *  @param chargeTime  the number of hours the battery needs to be charged
 *         Precondition: chargeTime > 0
 *  @return an optimal start time, with 0 ≤ returned value ≤ 23
 */
public int getChargeStartTime(int chargeTime)
{
    int total = getChargingCost(time, chargeTime);
    int time = 0;
    while (total > 0)
    {
        total = total - rateTable[time];
        time++;
    }
    return time;
}
```

GO ON TO THE NEXT PAGE.