

Past Exam – TEXT EDITOR



Past Exam – Text Editor (1/10)



Similar to a notepad, implement a simple Text editor program that provides the following function.

- Input a character into the cursor location
- Input an enter (newline)
- Cursor movement



As shown in [Fig.1], there is one cursor and an 'input area' in the Text editor.

The cursor is the current location to input either a character or enter. (The red bar in [Fig.1])

The input area limits the cursor to move within that area. (The green area in [Fig. 1])

The initial input area and the initial cursor location are row 1 column 1.

Depending on the character or enter input, the input area is extended to different forms.

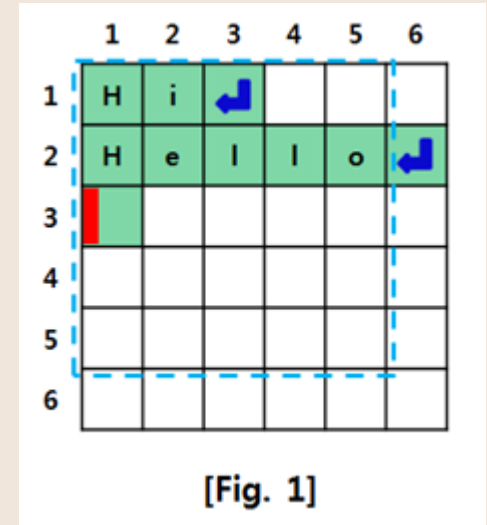
Since the Text editor has a limited capacity, up to row N can be inputted.

Also, the maximum number of characters that can be inputted into a row is N. (dotted blue line in [Fig.1], $N=5$)

If N characters are inputted into one row, the cursor can be positioned in column $N + 1$. (row 2 column 6 in [Fig.1])

Value N is given through the parameter of function `init()`.

In the problem, row R and column C are denoted as (R, C).





Past Exam – Text Editor (2/10)



Refer to the following API description and implement each function.

void init(int n)

Called in the beginning of each test case.

Initial cursor location is set as row 1 column 1 such as (1, 1).

Parameters

n : N value of the Text editor

void input_char(char in_char)

The character 'in_char' is inserted into the current cursor location.

After insertion, cursor moves one cell right.

If there are N characters inputted into the row where the current cursor is positioned, it is guaranteed that this function will not be called.

Parameters

in_char : character to insert into the cursor position



Past Exam – Text Editor (3/10)

void input_newline()

Enter is inputted for the current cursor position and newline is executed.

When enter is inputted for row R, the cursor is positioned in (R+1, 1).

If currently enter has been inputted N-1 times in total, it is guaranteed that this function will not be called.

void move_cursor(int direction)

Cursor is moved one cell up/down/left/right.

The cursor movement is only possible within the current input area. (refer to [Fig.3])

If function calls a direction to which it cannot move, call is ignored and the cursor remains at its current position.

Parameters

direction : Direction to which the cursor moves (0: up, 1: down, 2: left, 3: right)

char get_char(int row, int column)

Returns character of (row, column).

Guaranteed that only cells with a character inputted will be called. Enter is not a character.

Parameters

row : Row index

column : Column index



Returns

Character inputted in row and column



Past Exam – Text Editor (4/10)

[Cursor Movement]

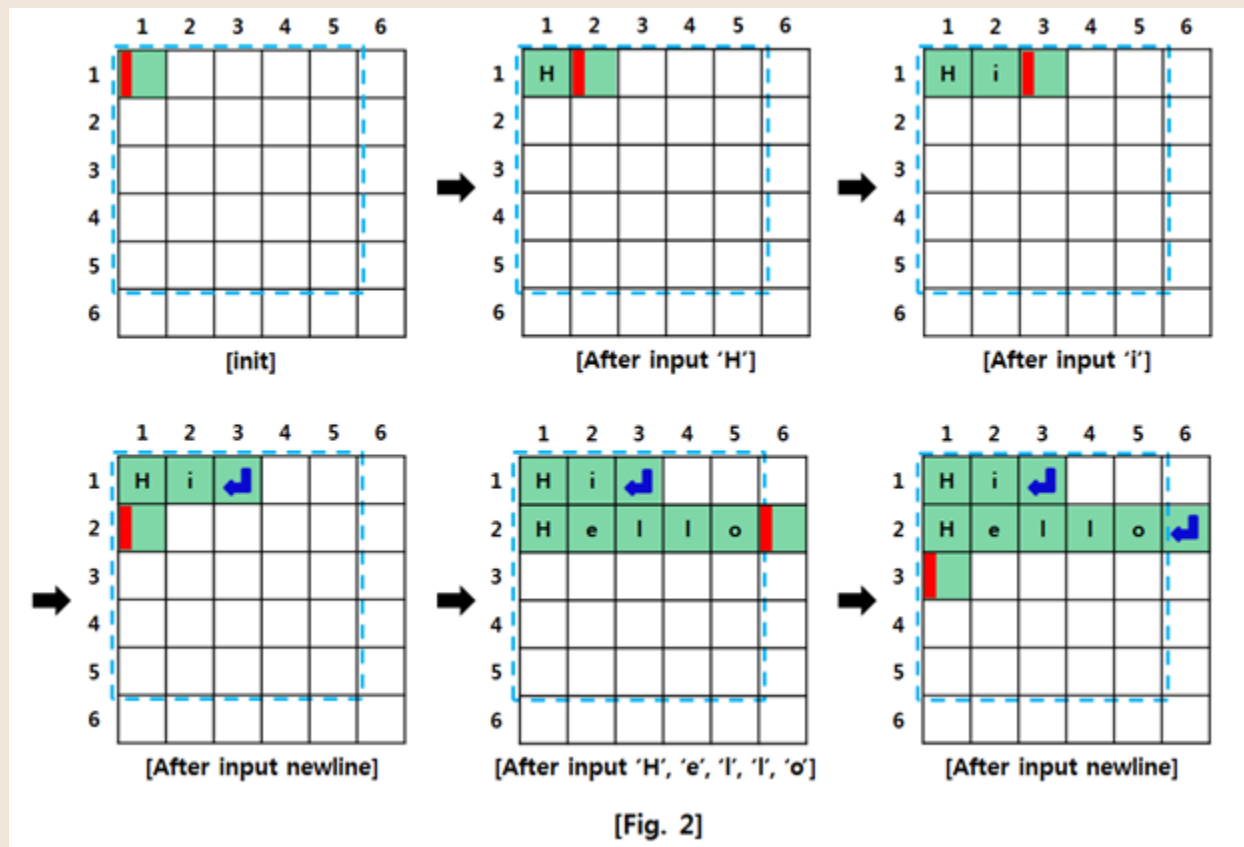
-  The cursor can always only move within the input area.
-  There are exceptional cases for the cursor depending on the current position and the direction to move such as the following:
 - a. When moving up from row 1
 - : remains at its current position
 - b. When moving up from (R, C) and the length of row R-1 of the input area is less than C
 - : Within the input area, moves to the last column of row R-1
 - c. When moving down from the last row within the input area
 - : Remains at its current position
 - d. When moving down from (R, C) and the length of row R+ 1 of the input area is less than C
 - : Within the input area, moves to the last column of row R+ 1 of the input area
 - e. When moving to the left from (1, 1)
 - : Remains at its current position
 - f. When moving to the left from (R, 1)
 - : Within the input area, moves to the last column of row R-1
 - g. When moving to the right from the last column and last row within the input area
 - : Remains at its current position
 - h. When moving to the right from the last column of row R within the input area



Past Exam – Text Editor (5/10)

✍ In case of [Fig.2], N is 5 and it denotes an example of the character and enter input.

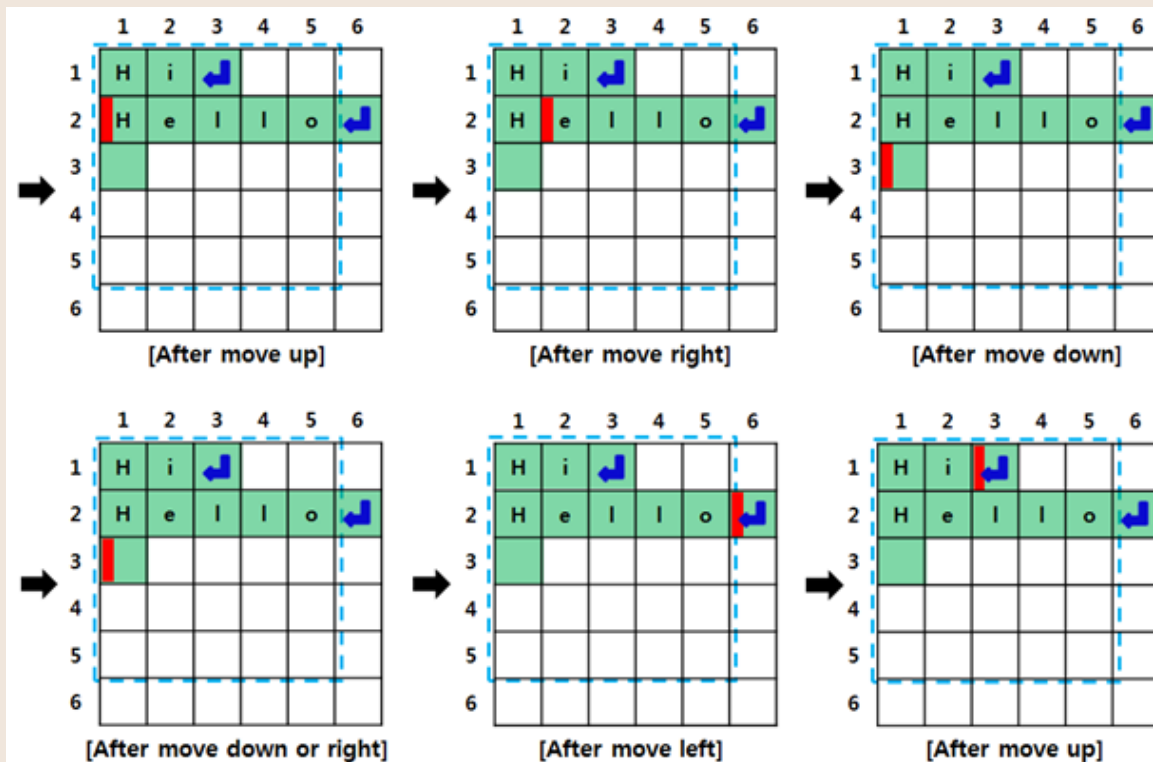
The green shadowed part denotes the input area, the red bar denotes the cursor location and the blue angled arrow denotes the enter input.





Past Exam – Text Editor (6/10)

[Fig.3] is an example showing that shows the continued cursor movement of [Fig.2] after the last step. When the cursor moves up from (3, 1), it will be positioned in (2, 1). When it moves from (2, 1) to the right, it will be positioned in (2, 2). When the cursor in (2, 2) moves down, it will be positioned in (3, 1). That is because of the inputted enter in row 2. So the cursor can move to row 3. The cursor in (3, 1) cannot move down or to the left and remains at its current position. When the cursor in (3, 1) moves to the left, it will be positioned in (2, 6). When the cursor in (2, 6) moves up, it will be positioned in (1, 3).



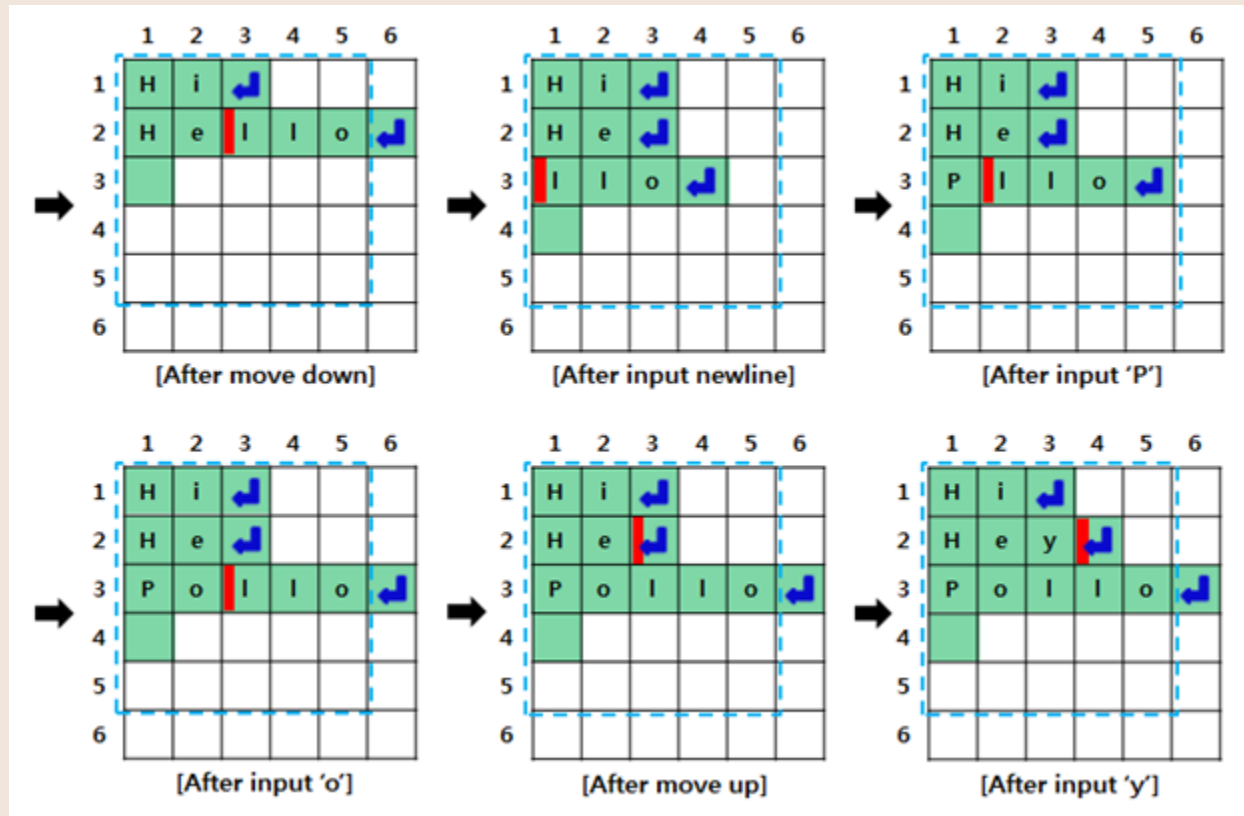
[Fig. 3]



Past Exam – Text Editor (7/10)

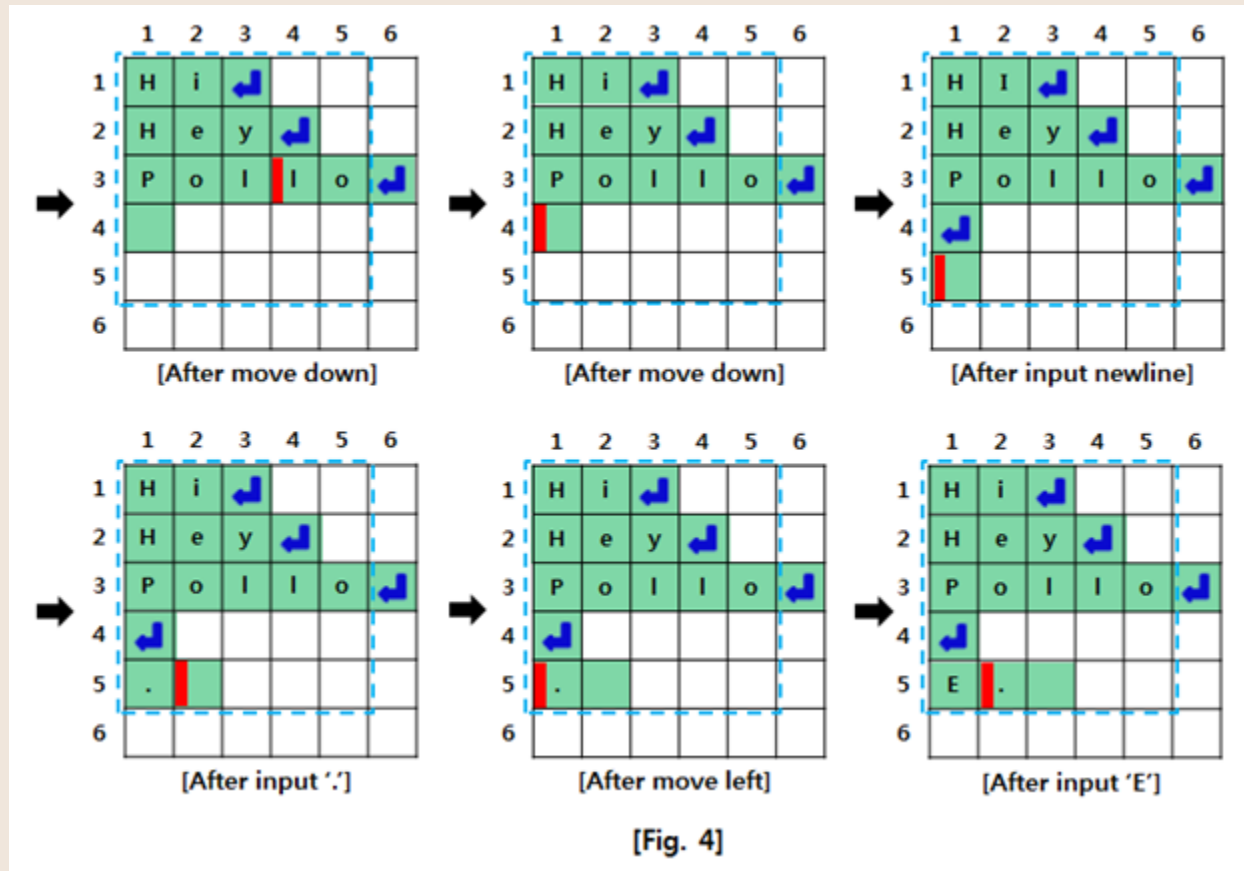


[Fig.4] is an example showing that shows different movements of [Fig.3] after the last step.





Past Exam – Text Editor (8/10)





Past Exam – Text Editor (9/10)



[Constraints]

1. The maximum size of characters that can be inputted into the Text editor is N rows, N columns.
2. The maximum size of the Text editor N is given as an integer that is greater than or equal to 5 and less than or equal to 700. ($5 \leq N \leq 700$)
3. Up to N characters can be inputted into one row.
4. Enter is not a character. In other words, it is possible to input N characters and enter into one row such as row 2 in [Fig.3]
5. Enter is inputted maximum N-1 times.
6. The initial input area and the initial cursor location are given as row 1 column 1.
7. The cursor cannot be positioned more to the right than the cell for which enter has been inputted. Thus, a character cannot be inputted to the right of enter.
8. The type of input character can be one of the three such as lowercase letter, uppercase letter or a period.
 - Example: 'a' ~ 'z', 'A' ~ 'Z', '.'
9. The number of total API calls for each test case is up to N by N ($N \times N$) times. (However, if $N < 10$, up to 100 times).
10. The given input satisfies the constraint at any time.



Past Exam - Text Editor (10/10)



[Input/Output]



Input/Output are processed in the provided main part code. Thus, in the user code part, input/output are not separately processed.

The correct output result for the sample input will be shown as follows:

```
#1 1000
#2 1000
#3 1000
#4 1000
#5 1000
Total Score : 5000
```

C++ Solution code

C++ : Solution code using Array (1/5)

```
#define MAX_N 700

struct Line {
    char str[MAX_N];
    int len;
};

Line* LineArr[MAX_N];
int LineCnt;
int CurR, CurC;

void init(int n) {
    for (register int i = 0; i < LineCnt; ++i) {
        if (LineArr[i] != 0) {
            delete LineArr[i];
            LineArr[i] = 0;
        }
    }
    CurR = CurC = 0;
    LineArr[0] = new Line;
    LineArr[0]->len = 0;
    LineCnt = 1;
}
```



C++ : Solution code using Array (2/5)

```
void input_char(char in_char) {
    if (CurC < LineArr[CurR]->len) {
        for (register int i = LineArr[CurR]->len; i > CurC; --i)
            LineArr[CurR]->str[i] = LineArr[CurR]->str[i - 1];
    }

    LineArr[CurR]->str[CurC++] = in_char;
    LineArr[CurR]->len++;
}

char get_char(int row, int column) {
    return LineArr[row - 1]->str[column - 1];
}
```

C++ : Solution code using Array (3/5)

```
void input_newline() {
    if (CurR < LineCnt - 1) {
        for (register int i = LineCnt; i > CurR + 1; --i)
            LineArr[i] = LineArr[i - 1];
    }

    LineArr[CurR + 1] = new Line;
    LineArr[CurR + 1]->len = 0;
    LineCnt++;

    if (CurC < LineArr[CurR]->len) {
        register int i;
        for (i = 0; CurC + i < LineArr[CurR]->len; ++i) {
            LineArr[CurR + 1]->str[i] = LineArr[CurR]->str[CurC + i];
        }
        LineArr[CurR + 1]->len = i;
        LineArr[CurR]->len = CurC;
    }

    ++CurR;
    CurC = 0;
}
```



C++ : Solution code using Array (4/5)

```
void move_cursor(int direction) { // 0:Up, 1:Down, 2:Left, 3:Right
    switch (direction) {
        case 0:
            if (CurR > 0) {
                --CurR;

                if (CurC > LineArr[CurR]->len)
                    CurC = LineArr[CurR]->len;
            }
            break;
        case 1:
            if (CurR < LineCnt - 1) {
                ++CurR;

                if (CurC > LineArr[CurR]->len)
                    CurC = LineArr[CurR]->len;
            }
            break;
        ...
    }
}
```


C++ : Solution code using Array (5/5)

```
case 2:
    if (CurC > 0) {
        --CurC;
    }
    else {
        if (CurR > 0) {
            --CurR;
            CurC = LineArr[CurR]->len;
        }
    }
    break;
case 3:
    if (CurC < LineArr[CurR]->len) {
        ++CurC;
    }
    else {
        if (CurR < LineCnt - 1) {
            ++CurR;
            CurC = 0;
        }
    }
    break;
}
```

C++ : Solution code using Array – Improvement

```
Line Pool[MAX_N];
int PoolCnt;

...
void init(int n) {
    PoolCnt = 0;

    CurR = CurC = 0;
    LineArr[0] = &Pool[PoolCnt++];
    LineArr[0]->len = 0;
    LineCnt = 1;
}

void input_newline() {
    if (CurR < LineCnt - 1) {
        for (register int i = LineCnt; i > CurR + 1; --i)
            LineArr[i] = LineArr[i - 1];
    }

    LineArr[CurR + 1] = &Pool[PoolCnt++];
    LineArr[CurR + 1]->len = 0;
    LineCnt++;
    ...
}
```

C++ : Solution code using Linked List (1/5)

```
#define MAX_N 700
struct Line {
    char str[MAX_N];
    int len;
    Line* prev;
    Line* next;
};
Line* Head = 0;
Line* CurLine;
int CurC;
void init(int n){
    Line* pLine = Head;
    while (pLine) {
        Line* tmp = pLine;
        pLine = pLine->next;
        delete tmp;
    }
    Head = new Line;
    Head->prev = Head->next = 0;
    Head->len = 0;
    CurLine = Head;
    CurC = 0;
}
```

C++ : Solution code using Linked List (2/5)

```
void input_char(char in_char){
    if (CurC < CurLine->len) {
        for (register int i = CurLine->len; i > CurC; --i)
            CurLine->str[i] = CurLine->str[i - 1];
    }

    CurLine->str[CurC++] = in_char;
    CurLine->len++;
}

char get_char(int row, int column){
    register Line* pLine = Head;
    for (register int i = 0; i < row - 1; ++i)
        pLine = pLine->next;

    return pLine->str[column - 1];
}
```

C++ : Solution code using Linked List (3/5)

```
void input_newline(){
    register Line* newline = new Line;

    Line* tmp = CurLine->next;
    CurLine->next = newline;
    newline->prev = CurLine;
    newline->next = tmp;
    newline->len = 0;
    if (tmp)
        tmp->prev = newline;

    if (CurC < CurLine->len) {
        register int i;
        for (i = 0; CurC + i < CurLine->len; ++i) {
            newline->str[i] = CurLine->str[CurC + i];
        }
        newline->len = i;
        CurLine->len = CurC;
    }

    CurLine = newline;
    CurC = 0;
}
```

C++ : Solution code using Linked List (4/5)

```
void move_cursor(int direction){ // 0:Up, 1:Down, 2:Left, 3:Right
    switch (direction) {
        case 0:
            if (CurLine->prev) {
                CurLine = CurLine->prev;
                if (CurC > CurLine->len)
                    CurC = CurLine->len;
            }
            break;
        case 1:
            if (CurLine->next) {
                CurLine = CurLine->next;
                if (CurC > CurLine->len)
                    CurC = CurLine->len;
            }
            break;
        ...
    }
```

C++ : Solution code using Linked List (5/5)

```
...
case 2:
    if (CurC > 0) {
        --CurC;
    }
    else {
        if (CurLine->prev) {
            CurLine = CurLine->prev;
            CurC = CurLine->len;
        }
    }
    break;
case 3:
    if (CurC < CurLine->len) {
        ++CurC;
    }
    else {
        if (CurLine->next) {
            CurLine = CurLine->next;
            CurC = 0;
        }
    }
    break;
}
}
```



C++ : Solution code using Linked List -

```
Line Pool[MAX_N];
int PoolCnt;

...
void init(int n){
    PoolCnt = 0;

    Head = &Pool[PoolCnt++];
    Head->prev = Head->next = 0;
    Head->len = 0;
    CurLine = Head;
    CurC = 0;
}

void input_newline(){
    register Line* newline = &Pool[PoolCnt++];
    Line* tmp = CurLine->next;
    CurLine->next = newline;
    newline->prev = CurLine;
    newline->next = tmp;
    newline->len = 0;
    if (tmp)
        tmp->prev = newline;
    ...
}
```


JAVA Solution code



Java : Solution code using Array (1/5)

```
public class UserSolution {
    private static final int MAX_N = 700;

    class Line {
        char str[] = new char[MAX_N];
        int len;
    }

    Line LineArr[] = new Line[MAX_N];
    int LineCnt = 0;
    int CurR, CurC;

    public void init(int n){
        CurR = CurC = 0;
        LineArr[0] = new Line();
        LineArr[0].len = 0;
        LineCnt = 1;
    }
}
```



Java : Solution code using Array (2/5)

```
public void input_char(char in_char){
    if (CurC < LineArr[CurR].len) {
        for (int i = LineArr[CurR].len; i > CurC; --i)
            LineArr[CurR].str[i] = LineArr[CurR].str[i - 1];
    }

    LineArr[CurR].str[CurC++] = in_char;
    LineArr[CurR].len++;
}

public char get_char(int row, int column){
    return LineArr[row - 1].str[column - 1];
}
```



Java : Solution code using Array (3/5)

```
public void input_newline(){
    if (CurR < LineCnt - 1) {
        for (int i = LineCnt; i > CurR + 1; --i)
            LineArr[i] = LineArr[i - 1];
    }

    LineArr[CurR + 1] = new Line();
    LineArr[CurR + 1].len = 0;
    LineCnt++;

    if (CurC < LineArr[CurR].len) {
        int i;
        for (i = 0; CurC + i < LineArr[CurR].len; ++i) {
            LineArr[CurR + 1].str[i] = LineArr[CurR].str[CurC + i];
        }
        LineArr[CurR + 1].len = i;
        LineArr[CurR].len = CurC;
    }

    ++CurR;
    CurC = 0;
}
```



Java : Solution code using Array (4/5)

```
public void move_cursor(int direction){ //0:Up,1:Down,2:Left,3:Right
    switch (direction) {
        case 0:
            if (CurR > 0) {
                --CurR;

                if (CurC > LineArr[CurR].len)
                    CurC = LineArr[CurR].len;
            }
            break;
        case 1:
            if (CurR < LineCnt - 1) {
                ++CurR;

                if (CurC > LineArr[CurR].len)
                    CurC = LineArr[CurR].len;
            }
            break;
        ...
    }
}
```



Java : Solution code using Array (5/5)

```
case 2:
    if (CurC > 0) {
        --CurC;
    }
    else {
        if (CurR > 0) {
            --CurR;
            CurC = LineArr[CurR].len;
        }
    }
    break;
case 3:
    if (CurC < LineArr[CurR].len) {
        ++CurC;
    }
    else {
        if (CurR < LineCnt - 1) {
            ++CurR;
            CurC = 0;
        }
    }
    break;
}
}
```

Java : Solution code using Array – Improvement

```
Line Pool[] = new Line[MAX_N];  
int PoolCnt;
```

```
...
```

```
UserSolution() {  
    for (int i = 0; i < Pool.length; ++i)  
        Pool[i] = new Line();  
}
```

```
public void init(int n){  
    PoolCnt = 0;  
    CurR = CurC = 0;  
    LineArr[0] = Pool[PoolCnt++];  
    LineArr[0].len = 0;  
    LineCnt = 1;  
}
```

```
public void input_newline(){  
    if (CurR < LineCnt - 1) {  
        for (int i = LineCnt; i > CurR + 1; --i)  
            LineArr[i] = LineArr[i - 1];  
    }  
    LineArr[CurR + 1] = Pool[PoolCnt++];  
    ...  
}
```



Java : Solution code using Linked List (1/5)

```
public class UserSolution {
    private static final int MAX_N = 700;

    class Line {
        char str[] = new char[MAX_N];
        int len;
        Line prev;
        Line next;
    }

    Line Head;
    Line CurLine;
    int CurC;

    public void init(int n){
        Head = new Line();
        Head.prev = Head.next = null;
        Head.len = 0;
        CurLine = Head;
        CurC = 0;
    }
}
```




Java : Solution code using Linked List (2/5)

```
public void input_char(char in_char){
    if (CurC < CurLine.len) {
        for (int i = CurLine.len; i > CurC; --i)
            CurLine.str[i] = CurLine.str[i - 1];
    }

    CurLine.str[CurC++] = in_char;
    CurLine.len++;
}

public char get_char(int row, int column){
    Line line = Head;
    for (int i = 0; i < row - 1; ++i)
        line = line.next;

    return line.str[column - 1];
}
```



Java : Solution code using Linked List (3/5)

```
public void input_newline(){
    Line newline = new Line();

    Line tmp = CurLine.next;
    CurLine.next = newline;
    newline.prev = CurLine;
    newline.next = tmp;
    newline.len = 0;
    if (tmp != null)
        tmp.prev = newline;

    if (CurC < CurLine.len) {
        int i;
        for (i = 0; CurC + i < CurLine.len; ++i) {
            newline.str[i] = CurLine.str[CurC + i];
        }
        newline.len = i;
        CurLine.len = CurC;
    }

    CurLine = newline;
    CurC = 0;
}
```



Java : Solution code using Linked List (4/5)

```
public void move_cursor(int direction){ //0:Up,1:Down,2:Left,3:Right
    switch (direction) {
        case 0:
            if (CurLine.prev != null) {
                CurLine = CurLine.prev;
                if (CurC > CurLine.len)
                    CurC = CurLine.len;
            }
            break;
        case 1:
            if (CurLine.next != null) {
                CurLine = CurLine.next;
                if (CurC > CurLine.len)
                    CurC = CurLine.len;
            }
            break;
        ...
    }
```



Java : Solution code using Linked List (5/5)

```
...
case 2:
    if (CurC > 0) {
        --CurC;
    }
    else {
        if (CurLine.prev != null) {
            CurLine = CurLine.prev;
            CurC = CurLine.len;
        }
    }
    break;
case 3:
    if (CurC < CurLine.len) {
        ++CurC;
    }
    else {
        if (CurLine.next != null) {
            CurLine = CurLine.next;
            CurC = 0;
        }
    }
    break;
}
}
```



Java : Solution code using Linked List -

```
Line Pool[] = new Line[MAX_N];
int PoolCnt;

...
UserSolution() {
    for (int i = 0; i < Pool.length; ++i)
        Pool[i] = new Line();
}

public void init(int n){
    PoolCnt = 0;

    Head = Pool[PoolCnt++];
    Head.prev = Head.next = null;
    Head.len = 0;
    CurLine = Head;
    CurC = 0;
}

public void input_newline(){
    Line newline = Pool[PoolCnt++];
    ...
}
```