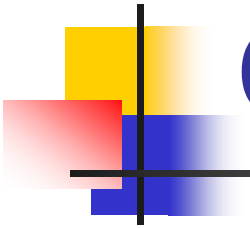
CS 2213
Advanced Programming
Ch 9 – Efficiency and Abstract Data Types (ADT)

Turgay Korkmaz

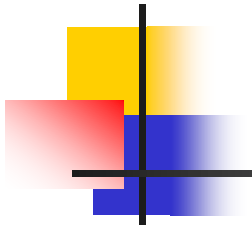
Office: SB 4.01.13
Phone: (210) 458-7346
Fax: (210) 458-4437
e-mail: korkmaz@cs.utsa.edu
web: www.cs.utsa.edu/~korkmaz

Thanks to Eric S. Roberts, the author of our textbook, for providing some slides/figures/programs.



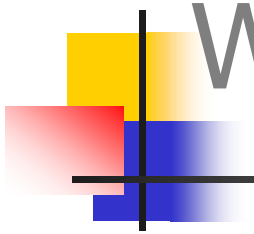
Objectives

- To learn that **different strategies for representing** data can have a **profound effect** on the efficiency of your code.
- To be able to compare the **computational complexity** of different representation strategies.
- To understand the **concept of a linked list** and how it can be used to provide a basis for **efficient insertion and deletion** operations.
- To appreciate that representations which are efficient in terms of their **execution time** may be inefficient in their use of **memory**.



As an application,
we will consider a simple
vi-like **text editor**

Why We Look at Old Editors?



- We are going to look at an ancient (at least in the sense of Internet time) editor technology, which is largely based on the TECO (Text Editor and Corrector) (*vi has similar ideas*)
- Many students greet this idea with skepticism. Why should we look at something so old that doesn't meet even the most requirements we would insist on in an editor.
- Because
  - It's absolutely the best example of how using **different data representations** can have a profound effect on performance.
  - No modern editor is simple enough to understand in its entirety.
  - TECO is historically important as the first extensible editor. It was the first platform for EMACS, which is still in use today.

WYSIWYG vs. Command Line

- Most editors today follow the WYSIWYG principle, which is an acronym for "what you see is what you get" that keeps the screen updated so that it shows the current document.
- TECO was a command-based editor. To edit a document, you enter commands that consist of a letter, possibly along with some additional data. Rather than showing the contents of the editor all the time, command-line editors showed the contents only when the user asked for them.
- Most histories of computer science date the first WYSIWYG editor to Xerox PARC in 1974.

The PDP-1 Computer

- A great deal of early graphics development was done on the Digital Equipment Corporation's PDP-1 computer, which was released in 1959. Most PDP-1s came with a Type 30 display, which was a 1024x1024 pixel display. Splat! ran on this display, but so did Spacewar!

skip

A B C D E
^

The Editor Commands

- Our minimal version of TECO has the following commands: (similar to vi, right?)

`I`*text*	**Inserts** the characters following the `I` into the buffer.
`J`	Moves the cursor to the **beginning** of the buffer.
`Z`	Moves the cursor to the **end** of the buffer.
`F`	Moves the cursor **forward** one character.
`B`	Moves the cursor **backward** one character.
`D`	**Deletes** the character **after** the cursor.
`Q`	**Exits** from the editor.

Sample operation of the editor

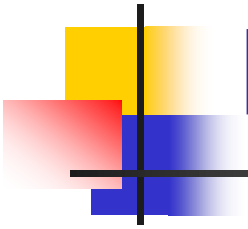
```
*Iaxc
 a x c
       ^
```
This command inserts the three characters 'a', 'x', and 'c', leaving the cursor at the end of the buffer.

```
*J
 a x c
^
```
This command moves the cursor to the beginning of the buffer.

```
*F
 a x c
   ^
```
This command moves the cursor forward one character.

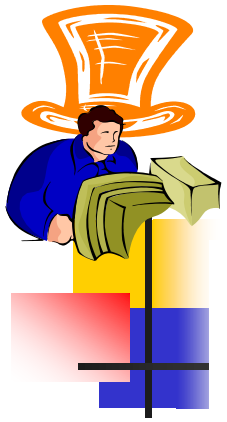
```
*D
 a c
   ^
```
This command deletes the character after the cursor.

```
*Ib
 a b c
     ^
*
```
This command inserts the character 'b' .



Defining buffer abstraction

- To create the said editor, we need to **first design a data structure** that maintains the state of the editor buffer
- The structure should keep track of
  - Characters in the buffer and
  - The position of the cursor
- It should also update the buffer contents when an editing operation is given
- We need to first develop `buffer.h` interface exporting the types and functions



buffer.h Interface

ElementType???

```
#ifndef _buffer_h
#define _buffer_h

#include "genlib.h"

typedef struct bufferCDT *bufferADT;

bufferADT NewBuffer(void);

void FreeBuffer(            bufferADT buffer);
void MoveCursorForward(     bufferADT buffer);
void MoveCursorBackward(    bufferADT buffer);
void MoveCursorToStart(     bufferADT buffer);
void MoveCursorToEnd(       bufferADT buffer);
void InsertCharacter(       bufferADT buffer, char ch);
void DeleteCharacter(       bufferADT buffer);
void DisplayBuffer(         bufferADT buffer);

#endif
```

Implementing the editor.c

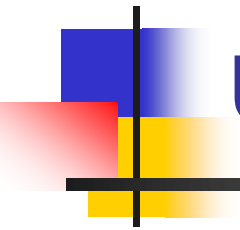
```
#include "buffer.h"
/* other libraries …*/
main()
{
     bufferADT buffer;
     string line;
     char fch;
     buffer = NewBuffer();
     do {
       printf("*"); line = GetLine();
       ExecuteCommand(buffer, line );
       fch=line[0]; FreeBlock(line);
       DisplayBuffer(buffer);
     }while (toupper(fch)!='Q');
     FreeBuffer(buffer);
}
```

```
static void HelpCommand(void)
{
    printf("Use the following commands to edit the buffer:\n");
    printf("  I...   Inserts text up to the end of the line\n");
    printf("  F      Moves forward a character\n");
    printf("  B      Moves backward a character\n");
    printf("  J      Jumps to the beginning of the buffer\n");
    printf("  E      Jumps to the end of the buffer\n");
    printf("  D      Deletes the next character\n");
    printf("  H      Generates a help message\n");
    printf("  Q      Quits the program\n");
}
```

```
void ExecuteCommand(bufferADT buffer, string line)
{
     int i;

     switch (toupper(line[0])) {
       case 'I':
           for (i = 1; line[i] != '\0'; i++) {
               InsertCharacter(buffer, line[i]);
            }
           break;
       case 'D':
           DeleteCharacter(buffer); break;
       case 'F':
           MoveCursorForward(buffer); break;
       case 'B':
           MoveCursorBackward(buffer); break;
       case 'J':
           MoveCursorToStart(buffer); break;
       case 'E':
           MoveCursorToEnd(buffer); break;
       case 'H':
           HelpCommand(); break;
       case 'Q':
           return;
       default:
           printf("Illegal command\n"); break;
     }
}
```

Where Do We Go From Here?

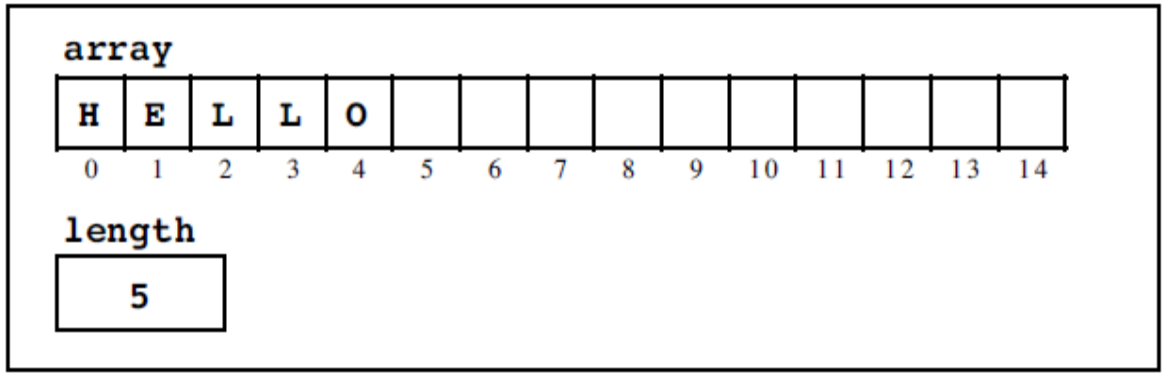
- Our strategy for the next few classes is
  - to implement the **Buffer ADT** in **three** different ways and
  - to compare the algorithmic efficiency of the various options. Those three representations are:
    1. A simple array model.
    2. A two-stack model that uses a pair of character stacks.
    3. A linked-list model that uses pointers to indicate the order. (single and double linked list)
- For each model, we'll calculate the complexity of each of the fundamental functions in the Buffer ADT. Some operations will be more efficient with one model, others will be more efficient with a different underlying representation.



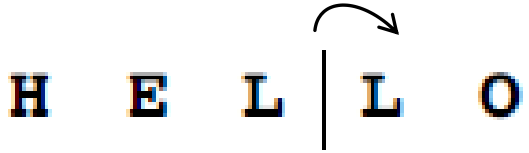
Implementing the buffer ADT using **ARRAYS**

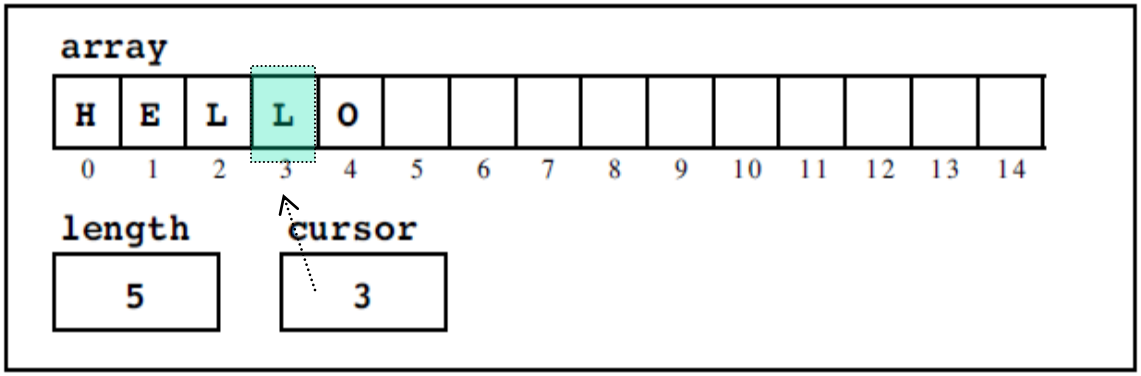
```
typedef struct bufferCDT *bufferADT;
```

Define concrete representation




```
#define MaxBuffer 100

struct bufferCDT {
    char array[MaxBuffer];
    int length;
    int cursor;
};
```





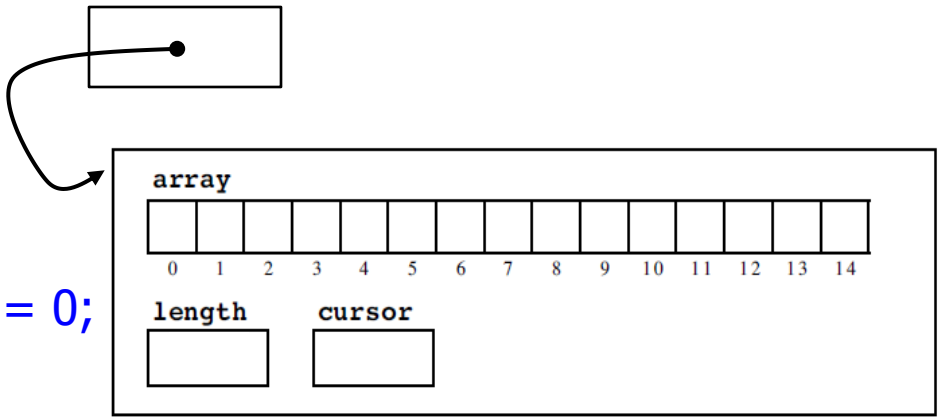
cursor field records the index position of the character that follows the cursor. So,

- cursor==0 means it is at the beginning of the buffer
- cursor==length means it is at the end of the buffer

Implementing arraybuf.c

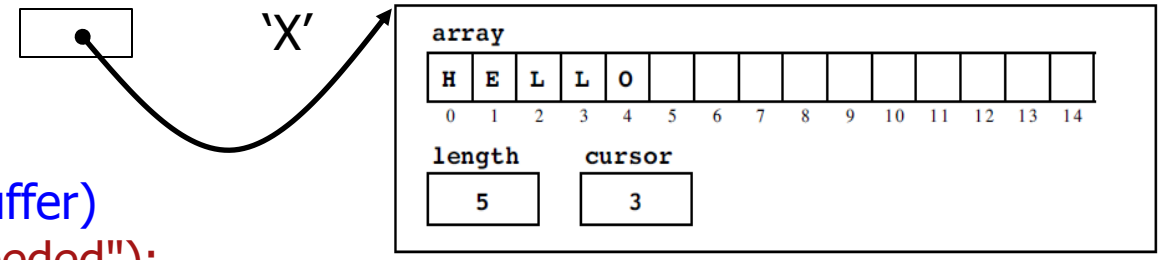
```
#include <stdio.h>
#include "genlib.h"
#include "buffer.h"

#define MaxBuffer 100

struct bufferCDT {
    char array[MaxBuffer];
    int length;
    int cursor;
};

bufferADT NewBuffer(void)
{
    bufferADT buffer;

    buffer = New(bufferADT);
    buffer->length = buffer->cursor = 0;
    return (buffer);
}
```

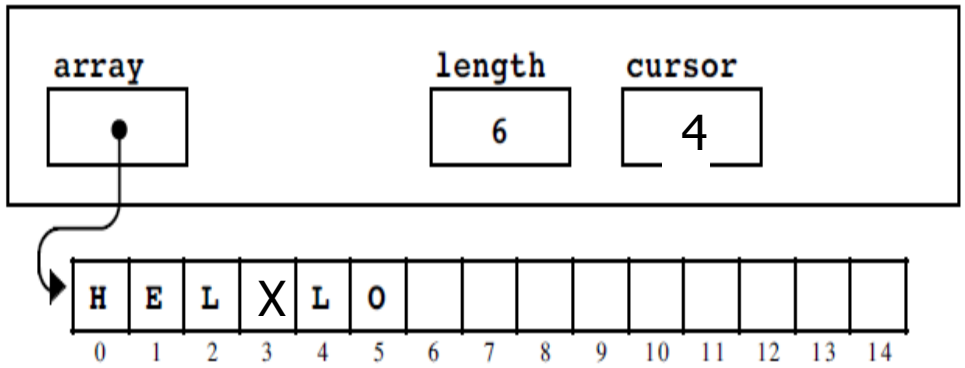
```
typedef struct bufferCDT *bufferADT;
```

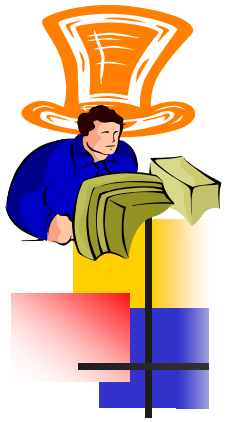


H E L L O

```
void InsertCharacter(bufferADT buffer, char ch)
{
    int i;

    if (buffer->length == MaxBuffer)
            Error("Buffer size exceeded");
    for (i = buffer->length; i > buffer->cursor; i--) {
        buffer->array[i] = buffer->array[i - 1];
    }
    buffer->array[buffer->cursor] = ch;
    buffer->length++;
    buffer->cursor++;
}
void DeleteCharacter(bufferADT buffer)
{
    int i;

    if (buffer->cursor < buffer->length) {
        for (i = buffer->cursor+1; i < buffer->length; i++) {
            buffer->array[i - 1] = buffer->array[i];
        }
        buffer->length--;
    }
}
```







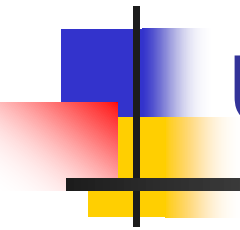
```
void FreeBuffer(bufferADT buffer)
{
   FreeBlock(buffer);
}
void MoveCursorForward(bufferADT buffer)
{
   if (buffer->cursor < buffer->length) buffer->cursor++;
}
void MoveCursorBackward(bufferADT buffer)
{
   if (buffer->cursor > 0) buffer->cursor--;
}
void MoveCursorToStart(bufferADT buffer)
{
   buffer->cursor = 0;
}
void MoveCursorToEnd(bufferADT buffer)
{
   buffer->cursor = buffer->length;
}
```

End of arraybuf.c

```
void DisplayBuffer(bufferADT buffer)
{
    int i;

    for (i = 0; i < buffer->length; i++) {
        printf(" %c", buffer->array[i]);
    }
    printf("\n");
    for (i = 0; i < buffer->cursor; i++) {
        printf("  ");
    }
    printf("^\n");
}
```

Complexity of the editor operations *(array representation)*

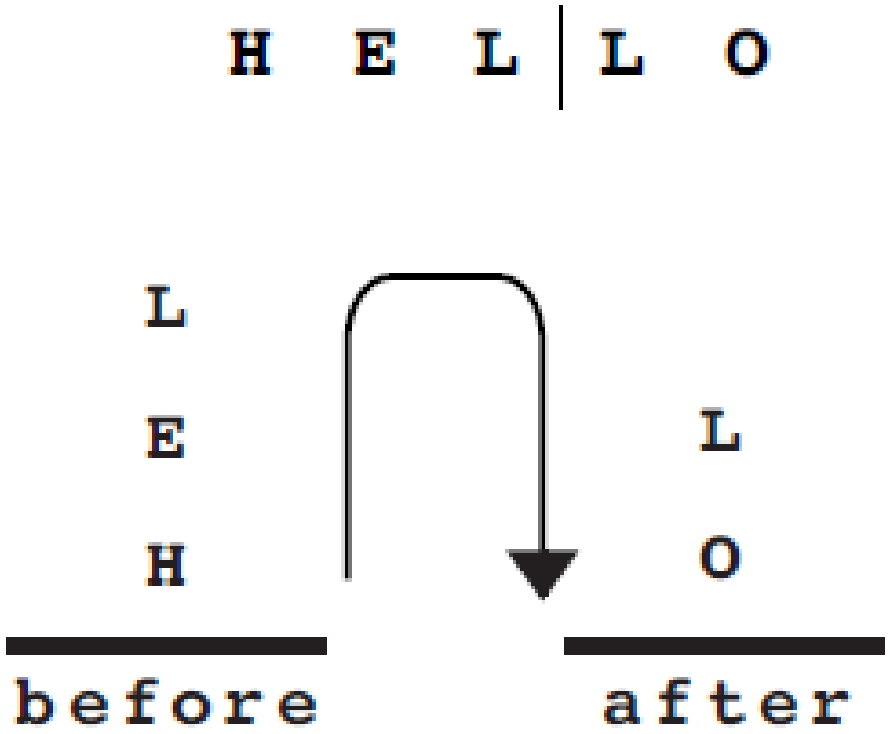
Function	**Complexity**
MoveCursorForward	*O(1)*
MoveCursorBackward	*O(1)*
MoveCursorToStart	*O(1)*
MoveCursorToEnd	*O(1)*
InsertCharacter	***O(N)***
DeleteCharacter	***O(N)***



Implementing the buffer ADT using **STACKS**

```
typedef struct bufferCDT *bufferADT;
```

Define concrete representation



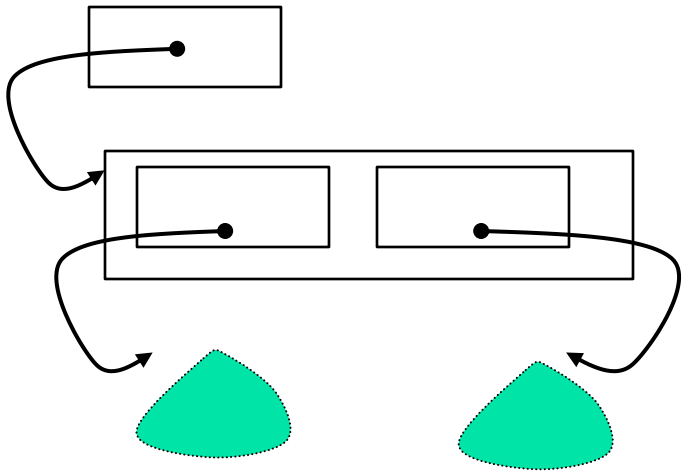

```
struct bufferCDT {
    stackADT before;
    stackADT after;
};
```

Implementing stackbuf.c

```
#include <stdio.h>
#include "genlib.h"
#include "buffer.h"
#include "stack.h"

struct bufferCDT {
   stackADT before;
   stackADT after;
};
```

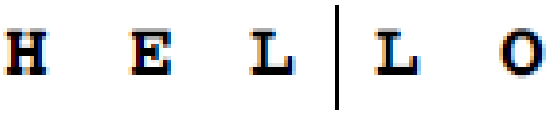
```
typedef struct bufferCDT *bufferADT;
/*  suppose you edited stack.h such that
    typedef char StackElementT; */
```

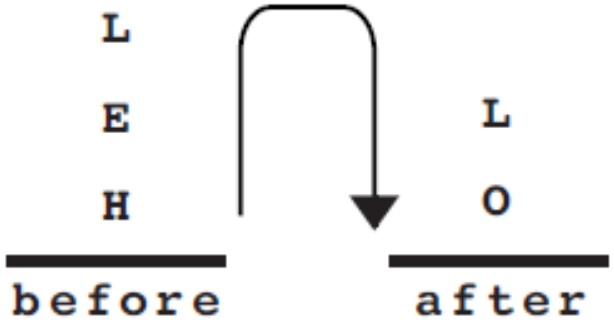
```
bufferADT NewBuffer(void)
{
   bufferADT buffer;

   buffer = New(bufferADT);
   buffer->before = NewStack();
   buffer->after = NewStack();
   return (buffer);
}
```



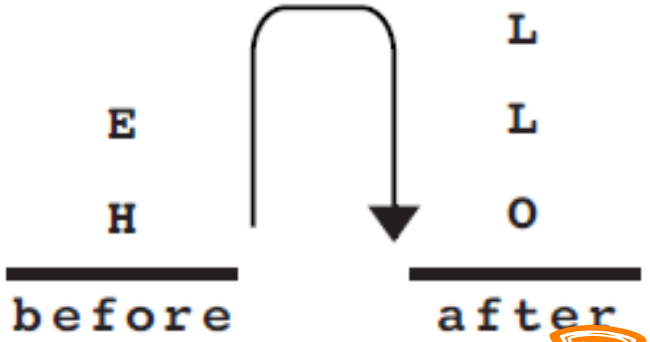
```
void InsertCharacter(bufferADT buffer, char ch)
{
   Push(buffer->before, ch);
}

void DeleteCharacter(bufferADT buffer)
{
   if (!StackIsEmpty(buffer->after)) {
      (void) Pop(buffer->after);
   }
}

void FreeBuffer(bufferADT buffer)
{
   FreeStack(buffer->before);
   FreeStack(buffer->after);
   FreeBlock(buffer);
}
```

```
void MoveCursorBackward(bufferADT buffer)
{
    if (!StackIsEmpty(buffer->before)) {
        Push(buffer->after, Pop(buffer->before));
    }
}

void MoveCursorForward(bufferADT buffer)
{
   if (!StackIsEmpty(buffer->after)) {
      Push(buffer->before, Pop(buffer->after));
   }
}
```






```
void MoveCursorToStart(bufferADT buffer)
{
   while (!StackIsEmpty(buffer->before)) {
      Push(buffer->after, Pop(buffer->before));
   }
}
void MoveCursorToEnd(bufferADT buffer)
{
   while (!StackIsEmpty(buffer->after)) {
      Push(buffer->before, Pop(buffer->after));
   }
}
```

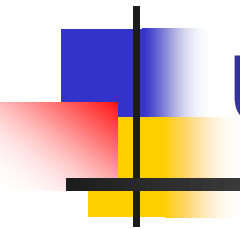


End of stackbuf.c

```
void DisplayBuffer(bufferADT buffer)
{
    int i;

    for (i = StackDepth(buffer->before) - 1; i >= 0; i--) {
        printf(" %c", GetStackElement(buffer->before, i));
    }
    for (i = 0; i < StackDepth(buffer->after); i++) {
        printf(" %c", GetStackElement(buffer->after, i));
    }
    printf("\n");
    for (i = 0; i < StackDepth(buffer->before); i++) {
        printf("  ");
    }
    printf("^\n");
}
```

Complexity of the editor operations *(array and stack)*

Function	**Array**	**Stack**
MoveCursorForward	*O(1)*	*O(1)*
MoveCursorBackward	*O(1)*	*O(1)*
MoveCursorToStart	*O(1)*	***O(N)***
MoveCursorToEnd	*O(1)*	***O(N)***
InsertCharacter	***O(N)***	*O(1)*
DeleteCharacter	***O(N)***	*O(1)*



Implementing the buffer ADT using **LINKED LIST**

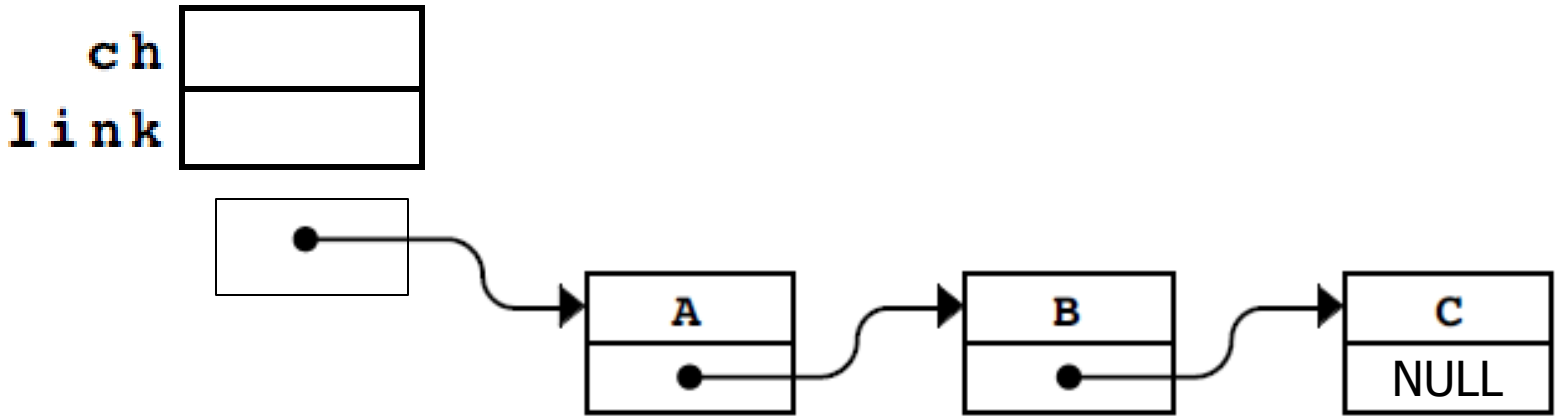
Linked List Concept

A C D E F G H I J K L M N O P Q R S T U V W X Y Z

B
A C D E F G H I J K L M N O P Q R S T U V W X Y Z
^

A→C→D→E→F→G→H→I→J→K→L→M→N→O→P→Q→R→S→T→U→V→W→X→Y→Z

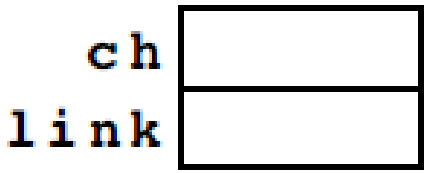
B
A C→D→E→F→G→H→I→J→K→L→M→N→O→P→Q→R→S→T→U→V→W→X→Y→Z



Linked List Concept (cont'd)

- How can we define cell structure in C?

```
typedef struct {
    char ch;
    cellT *link;
} cellT;
```

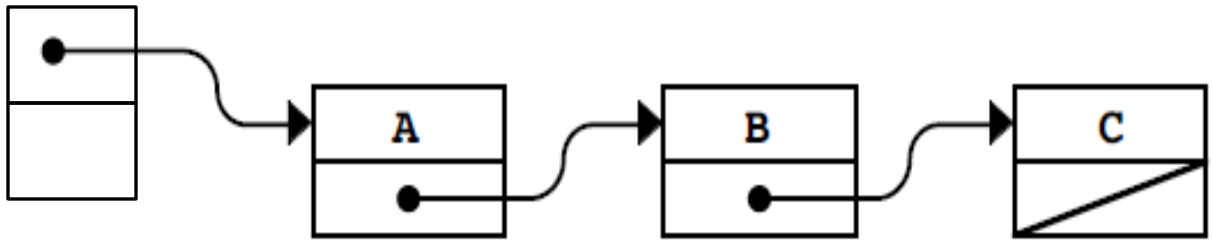


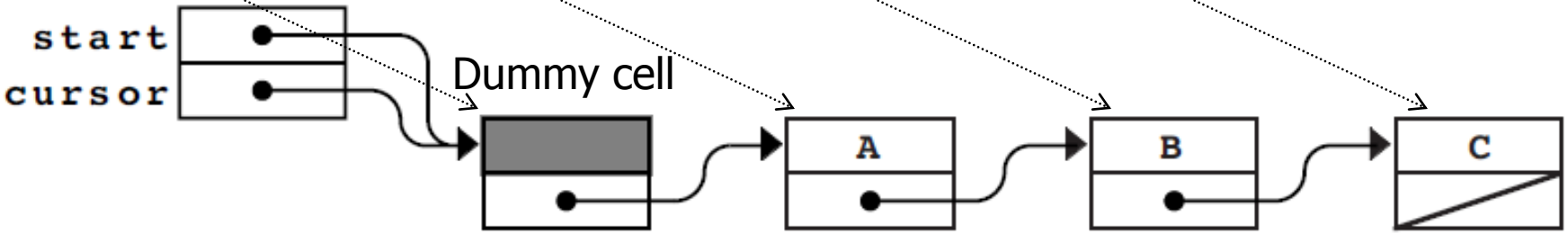
- What can we do?

```
typedef struct cell {
    char ch;
    struct cell *link;
} cellT;
```

```
typedef struct cell cellT;
struct cell {
    char ch;
    cellT *link;
}
```

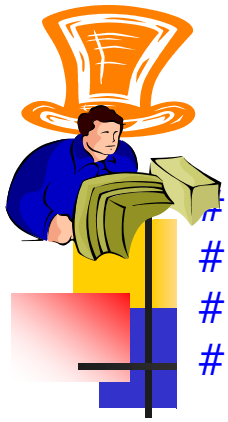
Define concrete representation for buffer using linked list





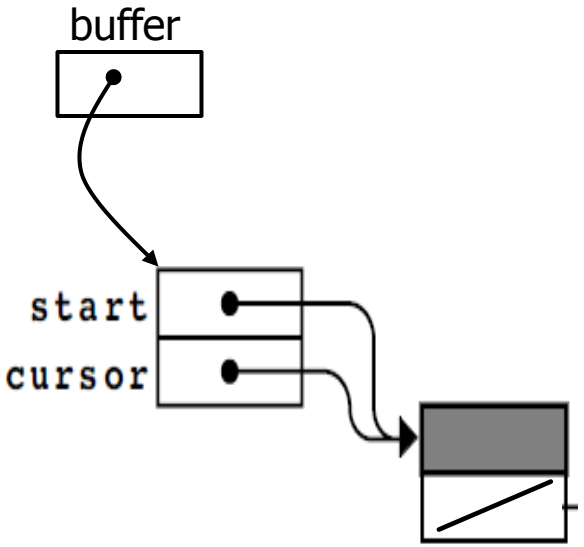

```
typedef struct cellT {
    char ch;
    struct cellT *link;
} cellT;

struct bufferCDT {
    cellT *start;
    cellT *cursor;
};
```

cursor points to the cell immediately before the logical insertion point.



Implementing listbuf.c

```
#include <stdio.h>
#include "genlib.h"
#include "strlib.h"
#include "buffer.h"

typedef struct cellT {
    char ch;
    struct cellT *link;
} cellT;

struct bufferCDT {
    cellT *start;
    cellT *cursor;
};

bufferADT NewBuffer(void)
{
    bufferADT buffer;

    buffer = New(bufferADT);
    buffer->start = buffer->cursor = New(cellT *);
    buffer->start->link = NULL;
    return (buffer);
}
```



void InsertCharacter(bufferADT buffer, char ch)

1. Allocate space for a new cell, and store the pointer to this cell in the local/temporary variable **cp.**

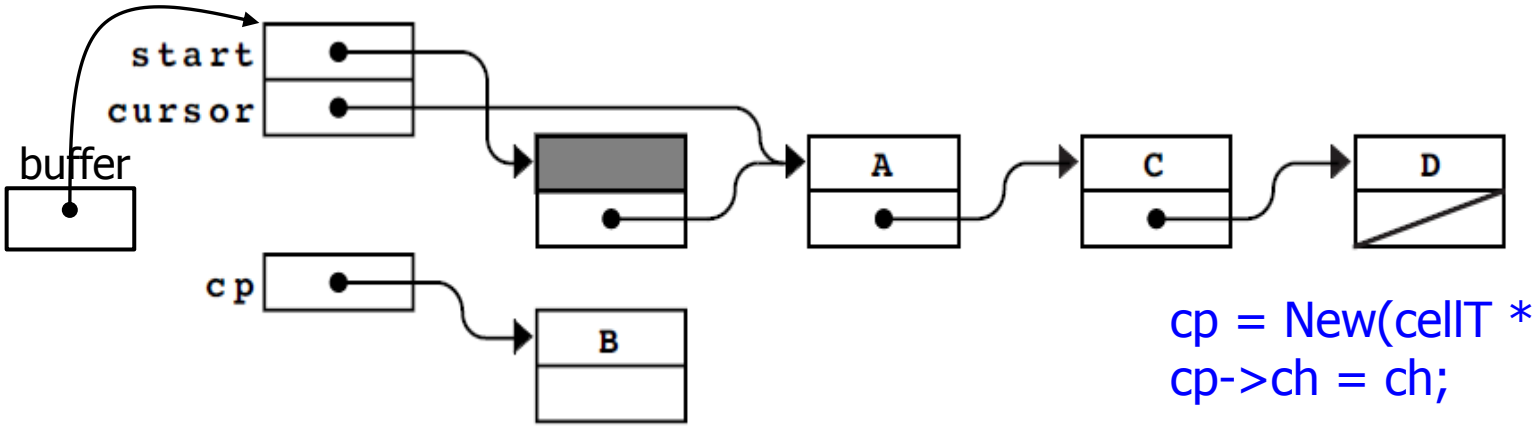
2. Copy the character to be inserted into the **ch** field of the new cell **(cp->ch).**

3. Go to the cell indicated by the **cursor** field of the buffer and copy its link field (**buffer->cursor->link)** to the link field of the new cell **(cp->link).** This operation makes sure that you don't lose the characters after the current cursor position.

4. Change the **link** field in the cell addressed by the cursor **(buffer->cursor->link)** so that it points to the new cell.

5. Change the **cursor field in the buffer so that it also points to the new cell.** This operation ensures that the next character will be inserted after this one in repeated insertion operations.

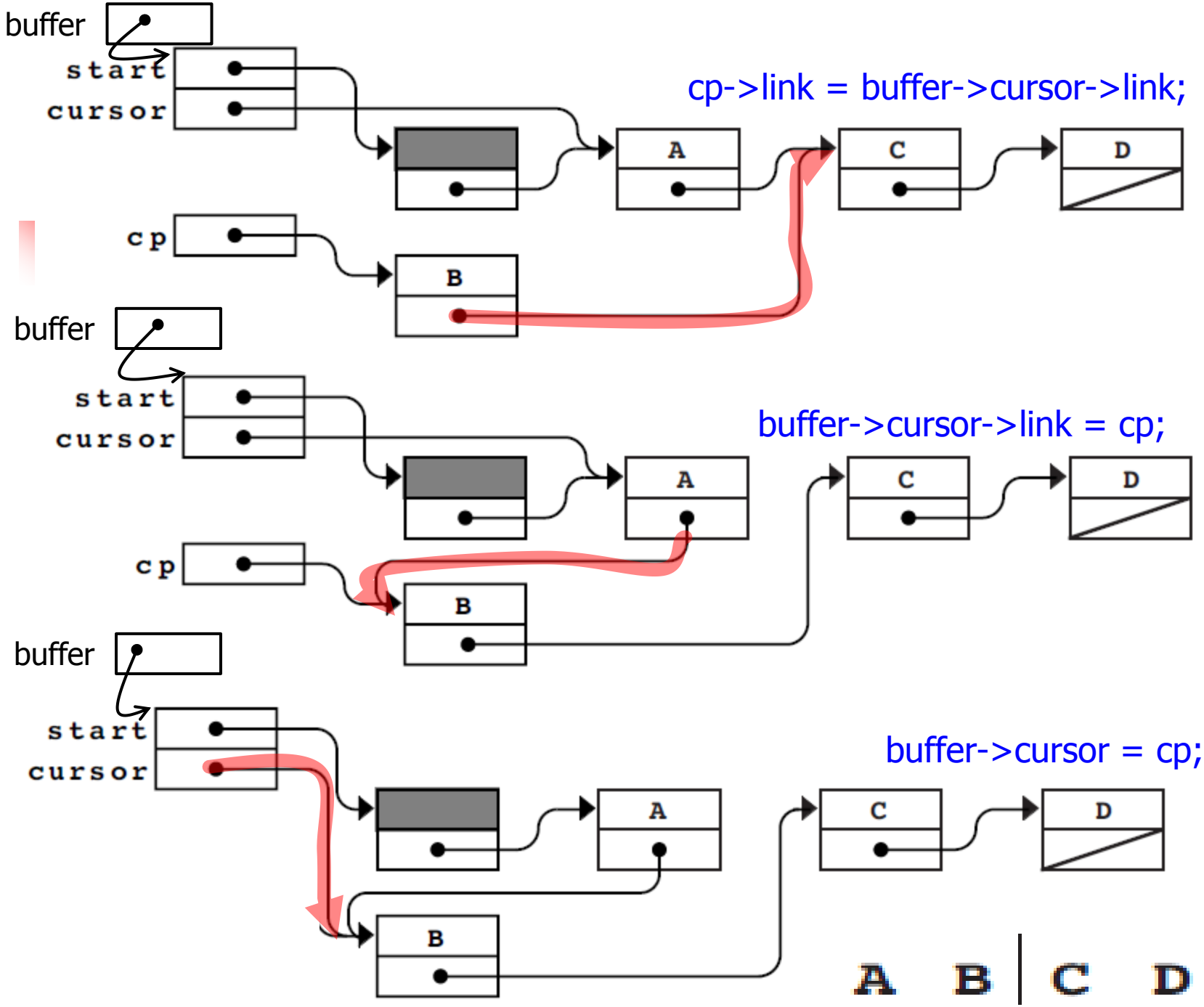
Suppose you want to insert the letter **B** into a buffer that currently contains **A|C D**
The situation prior to the insertion looks like this:

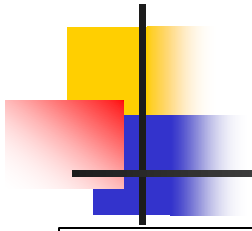


Steps

1 cp = New(cellT *);

2 cp->ch = ch;


Steps
3
cp->link = buffer->cursor->link;
buffer
start
cursor
cp
A
C
D
B
4
buffer->cursor->link = cp;
5
buffer->cursor = cp;
A B C D

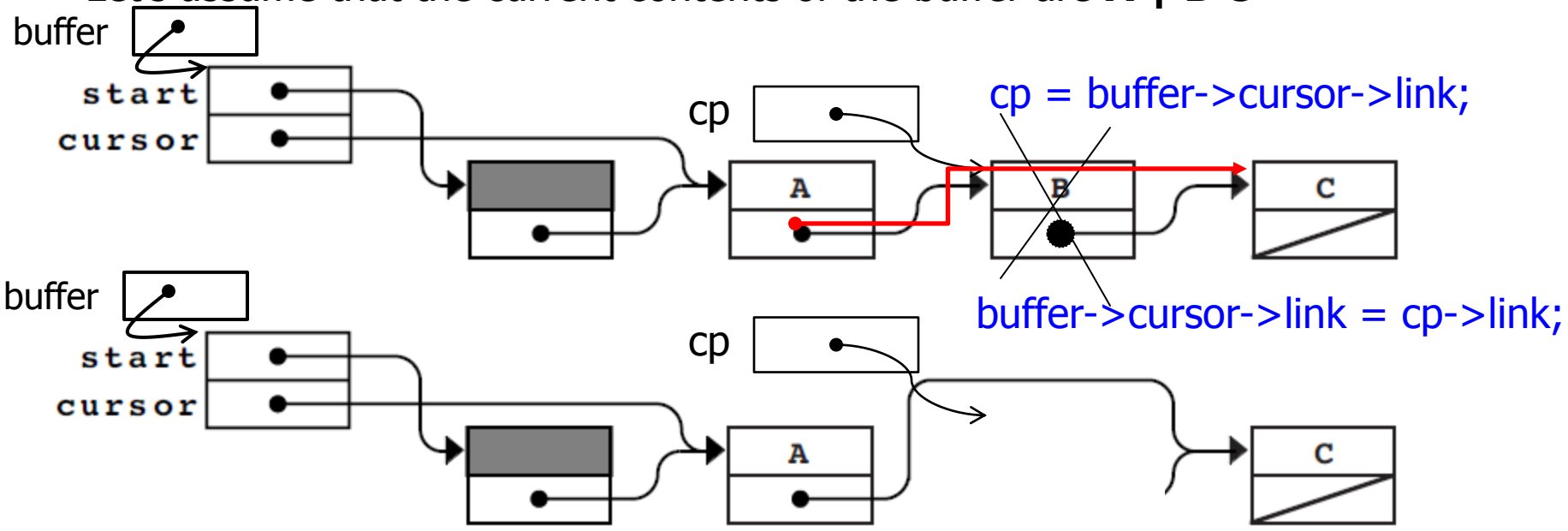


```
void InsertCharacter( bufferADT buffer, char ch )
{
    cellT *cp;

    cp = New(cellT *);
    cp->ch = ch;
    cp->link = buffer->cursor->link;
    buffer->cursor->link = cp;
    buffer->cursor = cp;
}
```

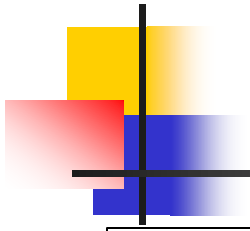
void DeleteCharacter(bufferADT buffer)

To delete a cell in a linked list, just remove the one after the cursor from the pointer chain.

Let's assume that the current contents of the buffer are **A | B C**

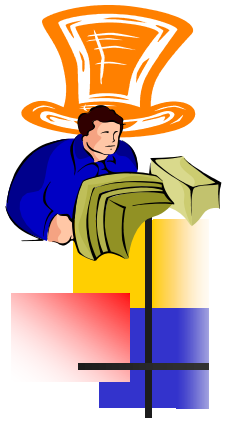


Instead of cp and two statements, can I simply use: buffer->cursor->link = buffer->cursor->link->link



```
void DeleteCharacter(bufferADT buffer)
{
    cellT *cp;

    if (buffer->cursor->link != NULL) {
        cp = buffer->cursor->link;
        buffer->cursor->link = cp->link;
        FreeBlock(cp);
    }
}
```

```
void FreeBuffer(bufferADT buffer)
{
    cellT *cp, *next;

    cp = buffer->start;
    while (cp != NULL) {
        next = cp->link;
        FreeBlock(cp);
        cp = next;
    }
    FreeBlock(buffer);
}

void MoveCursorForward(bufferADT buffer)
{
    if (buffer->cursor->link != NULL) {
        buffer->cursor = buffer->cursor->link;
    }
}
```



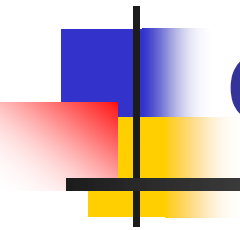
```
void MoveCursorBackward(bufferADT buffer)
{
   cellT *cp;

   if (buffer->cursor != buffer->start) {
      cp = buffer->start;
      while (cp->link != buffer->cursor) {
         cp = cp->link;
      }
      buffer->cursor = cp;
   }
}

void MoveCursorToStart(bufferADT buffer)
{
   buffer->cursor = buffer->start;
}

void MoveCursorToEnd(bufferADT buffer)
{
   while (buffer->cursor->link != NULL) {
      MoveCursorForward(buffer);
   }
}
```

End of listbuf.c

```
void DisplayBuffer(bufferADT buffer)
{
    cellT *cp;

    for (cp = buffer->start->link; cp != NULL; cp = cp->link) {
        printf(" %c", cp->ch);
    }
    printf("\n");
    for (cp = buffer->start; cp != buffer->cursor; cp = cp->link) {
        printf("  ");
    }
    printf("^\n");
}
```

Complexity of the editor operations *(array, stack and list)*

Function	**Array**	**Stack**	**LIST**
MoveCursorForward	*O(1)*	*O(1)*	*O(1)*
MoveCursorBackward	*O(1)*	*O(1)*	***O(N)***
MoveCursorToStart	*O(1)*	***O(N)***	*O(1)*
MoveCursorToEnd	*O(1)*	***O(N)***	***O(N)***
InsertCharacter	***O(N)***	*O(1)*	*O(1)*
DeleteCharacter	***O(N)***	*O(1)*	*O(1)*

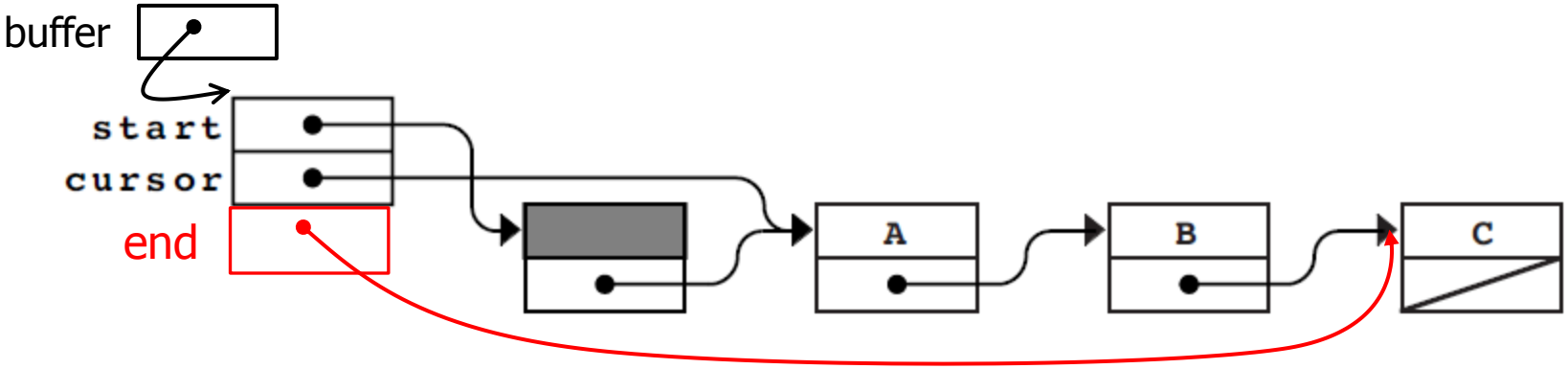


Can we get around ***O(N)*** in case of Linked List solution?

Same Single Link List
with a minor modification

```
struct bufferCDT {
   cellT *start;
   cellT *cursor;
   cellT *end;
};
```

- Maintain and update an **end** field in bufferCDT



- MoveCursorToEnd can be done in *O(1)*
  - How about InsertCharacter, DeleteCharacter (exercise)
- But, MoveCursorBackward is still ***O(N)***
  - can we make this in *O(1)*

Exercise: Change the followings to incorporate **end** field

```
bufferADT NewBuffer(void)
{
   bufferADT buffer;
   buffer = New(bufferADT);
   buffer->start = buffer->cursor = New(cellT *);
   buffer->start->link = NULL;
   return (buffer);
}
```

```
void MoveCursorToEnd( bufferADT buffer)
{
   while (buffer->cursor->link) {
      MoveCursorForward(buffer);
   }
}
```

```
void InsertCharacter(
   bufferADT buffer,  char ch )
{
   cellT *cp;
   cp = New(cellT *);
   cp->ch = ch;
   cp->link = buffer->cursor->link;
   buffer->cursor->link = cp;
   buffer->cursor = cp;
}
```

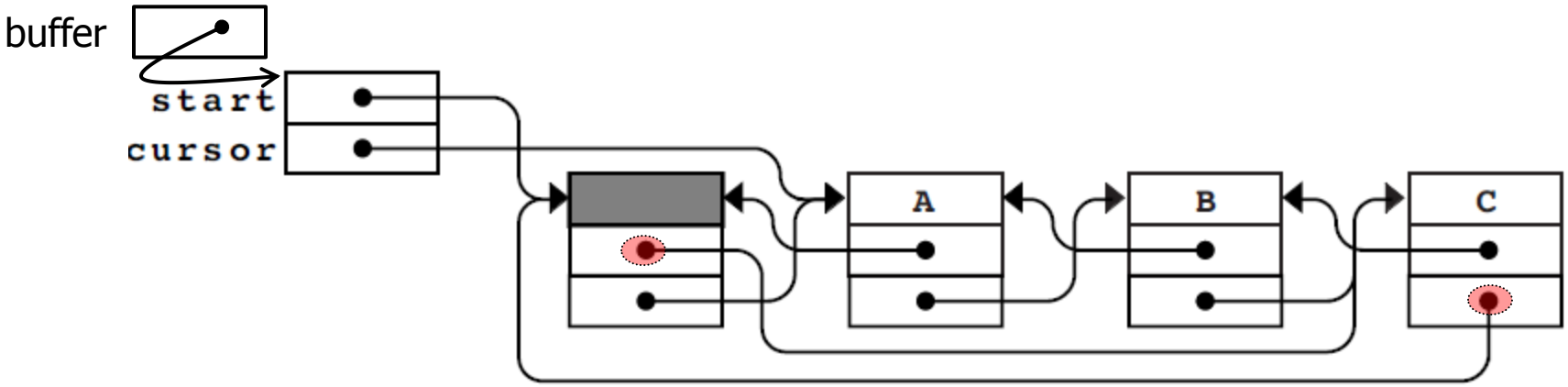
```
void DeleteCharacter( bufferADT buffer)
{
   cellT *cp;
   if (buffer->cursor->link != NULL) {
      cp = buffer->cursor->link;
      buffer->cursor->link = cp->link;
      FreeBlock(cp);
   }
}
```

Double Linked List (DLL)

```
typedef struct DcellT {
    char ch;
    struct DcellT *prev;
    struct DcellT *link;
} DcellT;
```

```
struct bufferCDT {
   DcellT *start;
   DcellT *cursor;
};
```

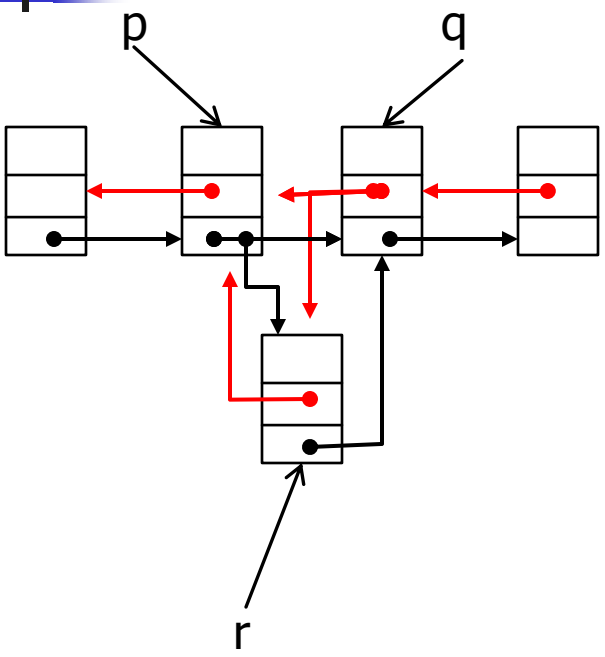
- Add a **prev** field to cell and update
- For **A | B C**, DLL looks like this



- All operations can be done in *O(1)*
- But there is a problem here
  - Time-space tradeoffs !!!

DLL: Insert a new cell

```
typedef struct DcellT {
    char ch;
    struct DcellT *prev;
    struct DcellT *link;
} DcellT;
```



Suppose you have three pointers p, q, r as shown on the figure.

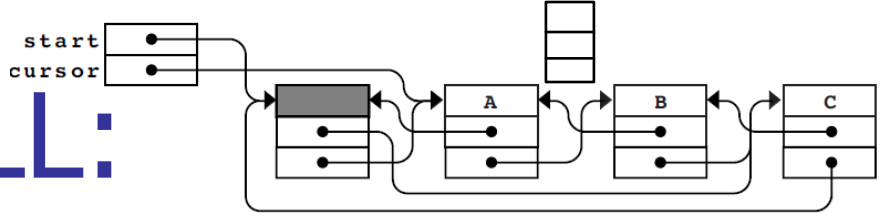
Write the necessary statements to insert the cell pointed by r between the cells pointed by p and q.

```
r->prev = p;
r->next = q;
p->next = r;
q->prev = r;
```

What if you have only p and r?

Write the necessary statements to insert the cell pointed by r between the cells pointed by p and q.

```
r->prev = p;
r->next = p->next;
p->next->prev = r;
p->next = r;
```



DLL: Insert a character

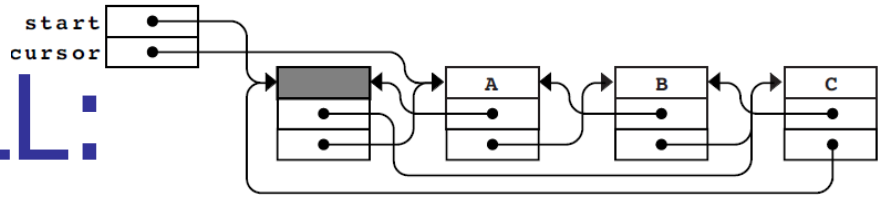
```
typedef struct DcellT {
    char ch;
    struct DcellT *prev;
    struct DcellT *link;
} DcellT;
```

```
void InsertCharacter(
   bufferADT buffer,
   char ch )
{
    cellT *cp;

    cp = New(cellT *);
    cp->ch = ch;
    cp->link = buffer->cursor->link;




    buffer->cursor->link = cp;
    buffer->cursor = cp;
}
```

```
void DLL_InsertCharacter(   /* DLL version */
   bufferADT buffer,
   char ch )
{
    DcellT *cp;

    cp = New(DcellT *);
    cp->ch = ch;
    cp->link = buffer->cursor->link;

    cp->prev = buffer->cursor;
    buffer->cursor->link->prev = cp;

    buffer->cursor->link= cp;
    buffer->cursor = cp;
}
```

/* check if this solution works when the cursor is at the beginning and at the end of the buffer. If needed, fix it */

DLL: Delete a character




```
typedef struct DcellT {
    char ch;
    struct DcellT *prev;
    struct DcellT *link;
} DcellT;
```

```
void DeleteCharacter(
bufferADT buffer)
{
    cellT *cp;

    if (buffer->cursor->link != NULL) {
        cp = buffer->cursor->link;
        buffer->cursor->link = cp->link;


        FreeBlock(cp);
    }
}
```

```
void DeleteCharacter(  /* DLL version */
bufferADT buffer)
{
    DcellT *cp;

    if (buffer->cursor->link!= buffer->start) {
        cp = buffer->cursor->link;
        buffer->cursor->link= cp->link;
        cp->link->prev = buffer->cursor;

        FreeBlock(cp);
    }
}
```

/* check if this solution works when the cursor is at the beginning and at the end of the buffer. If needed, fix it */

Exercise: DLL Versions of the following functions

```
void MoveCursorBackward(bufferADT buffer)
{
   cellT *cp;

   if (buffer->cursor != buffer->start) {
      cp = buffer->start;
      while (cp->link != buffer->cursor) {
         cp = cp->link;
      }
      buffer->cursor = cp;
   }
}

void MoveCursorToEnd(bufferADT buffer)
{
   while (buffer->cursor->link != NULL) {
      MoveCursorForward(buffer);
   }
}
```

```
bufferADT NewBuffer(void)
{
   bufferADT buffer;
   buffer = New(bufferADT);
   buffer->start = buffer->cursor =
      New(cellT *);
   buffer->start->link = NULL;
   return (buffer);
}
```

Exercise:
For each representation, implement
Bool SearchBuffer(bufferADT buffer, char ch);

- When this function is called, it should start searching from the current **cursor** position, looking for the next occurrence of the character **ch** in the rest of the buffer.

A little bit difficult version is to search a string in buffer

  - If it finds it, search should leave the **cursor** after the found character and return the value TRUE
  - If **ch** does not occur between the **cursor** and the end of the buffer, then search should leave the **cursor** unchanged and return FALSE
- What will be the complexity under each representations?

  Array Stack Single Linked List Double Linked list

- What if the characters in buffer were sorted?
  - Can you do binary search? How?

Home exercise