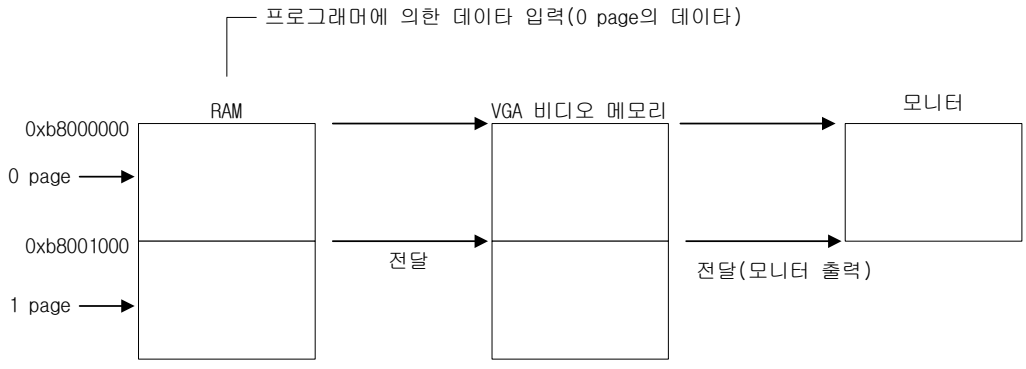
프로그래밍 과제 (텍스트 에디터 작성)

이 과제는 '텍스트 에디터' 프로그램을 작성하는 것으로 이 과제를 통해서, 비디오 메모리를 이용한 문자의 화면 출력, 문자의 속성, 인터럽트 함수, 저수준 파일 입출력함수, 포인트를 이용한 double linked-list, 특수키의 처리 등을 공부하게 된다. 이 과제는 다음과 같은 8개의 모듈로 나뉘어져 한 학기를 통해서 5개의 과제로 단계적으로 작성된다.

1. 창 만들기
2. 메뉴 만들기
3. 커서 제어하기
4. 텍스트 입력
5. 메뉴 실행하기
6. 텍스트의 삽입과 삭제
7. 줄 단위 스크롤
8. 화면 스크롤

<참고사항>

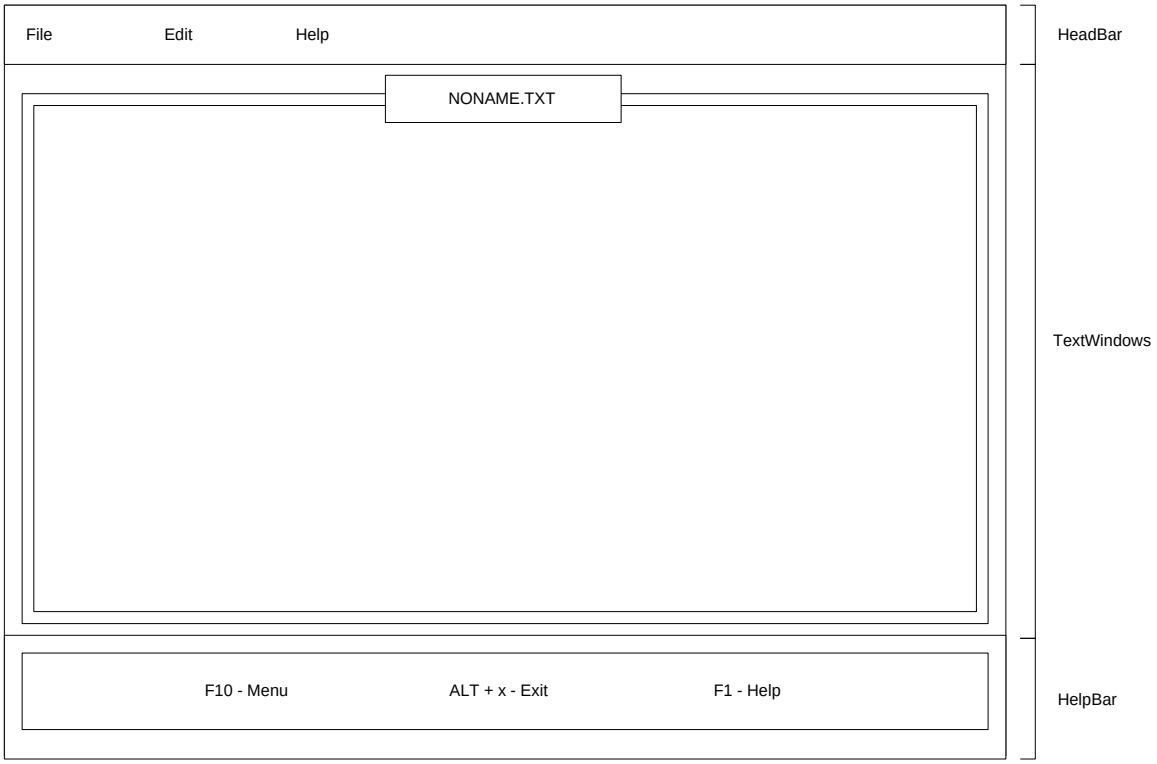
프로그램에서 화면에 문자를 출력하는 메커니즘은 다음과 같다. 상세한 부분은 필요에 따라 참고서적을 살펴보기 바란다. 화면에 출력하려는 데이터(문자)는 VGA 카드에 있는 '비디오 메모리'에 저장되어 있다. 프로그램은 화면에 출력하는 문자를 비디오 메모리에 기록함으로 VGA에 의해서 화면에 출력된다. VGA 비디오 메모리의 영역은 RAM의 0xa0000000 ~ 0xb000ffff 까지의 128Kbyte 크기로 할당되어 있으며, 실제 프로그램에서는 VGA 표준 Text 모드에서는 RAM의 0xb8000000 ~ 0xb000ffff까지의 32Kbyte 크기의 영역을 사용하고 있다. 이 영역들의 데이터들은 디바이스 드라이브(VGA 카드)에 의하여 그대로 VGA 비디오 메모리로 매핑에 의해 전달된다. 그러므로 우리는 VGA 표준 Text 모드에서 화면 출력을 하고자 할 때, RAM의 0xb8000000 ~ 0xb000ffff까지의 32Kbyte 크기의 영역에 출력하려는 데이터를 적어주기만 하면 된다. 그러면 이 데이터가 VGA 비디오 메모리로 그대로 전달되고, 그에 따라 모니터의 출력이 자동적으로 이루어진다. 이것은 아래 그림과 같다.



표준 Text 모드가 4K byte로 한 화면의 내용을 저장할 수 있다는 것을 감안할 때, 32Kbyte는 8개의 화면 내용을 저장할 수 있다. 이 과제에서는 8개의 화면 중 첫 번째 화면만을 사용한다.

PROJECT #1 텍스트 에디터 창 만들기

첫 번째 과제는 그림과 같은 텍스트 에디터의 초기 화면을 만드는 것이다.



화면 구성은 HeadBar, TextWindows, HelpBar의 3부분으로 나뉘어 진다. HeadBar에는 작업을 위한 메뉴가 표시되며, TextWindows에는 텍스트의 본문이 나타난다. HelpBar는 기본적으로 사용하는 키에 대한 설명이 들어있으며 최소한 'F10 - Menu', 'ALT + X - Exit', 'F1 - Help'가 있어야 한다.

화면의 창을 그릴 때는 아스키 코드 179번에서 218번까지의 문자를 사용해서 그린다. 문자 하나를 화면에 출력하는 데에는 2byte 크기의 정보가 필요하다. 아래 그림과 같이 하위 1byte에는 문자의 아스키 코드 값이, 상위 1byte에는 문자의 속성 값이 들어간다.

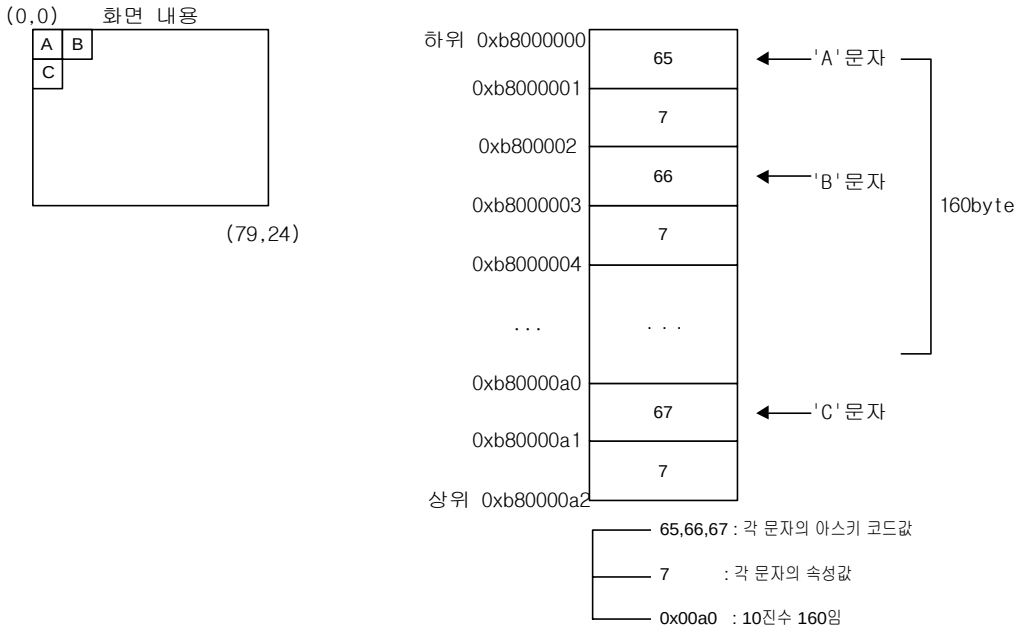
2byte의 문자의 정보

상위 1byte	하위 1byte

출력되는 문자의 아스키 코드 값을 변수 ch에 속성은 변수 attr에 저장한다고 가정할 때, 문자 'A'(아스키 코드값 65)를 검정 바탕에 하얀 글씨로 화면에 출력할 때는 다음과 같은 코드가 삽입된다.

```
char ch = 65;
char attr = 7;   //문자의 속성 값으로 보통 C 언어 책에 나와있음
```

이러한 문자 정보들은 RAM의 0xb8000000 위치에서 시작하여 순차적으로 저장되고, 이 값은 VGA 비디오 메모리로 그대로 전달되어 화면에 그려진다. 한 문자 당 2 byte가 소요되므로 80×25 크기의 텍스트 화면에서 한 줄을 표현하는 데에는 80 * 2 = 160 byte가 필요하게 된다. 화면의 우측 상단의 세 문자의 출력을 위한 메모리의 저장내용은 아래의 그림과 같다. 이때 화면의 좌표는 좌측상단이 (0,0)이고 우측하단이 (79,24)이다.



이와 같이 한 줄을 표현하는 데에는 160 byte가 소요되고, 한 문자를 출력하는 데에는 2byte가 소요되므로, 화면에서 임의의 (x, y) 위치에 문자를 출력하기 위해서는 다음과 같은 공식으로 문자가 출력되는 RAM의 주소를 알아낼 수 있다.

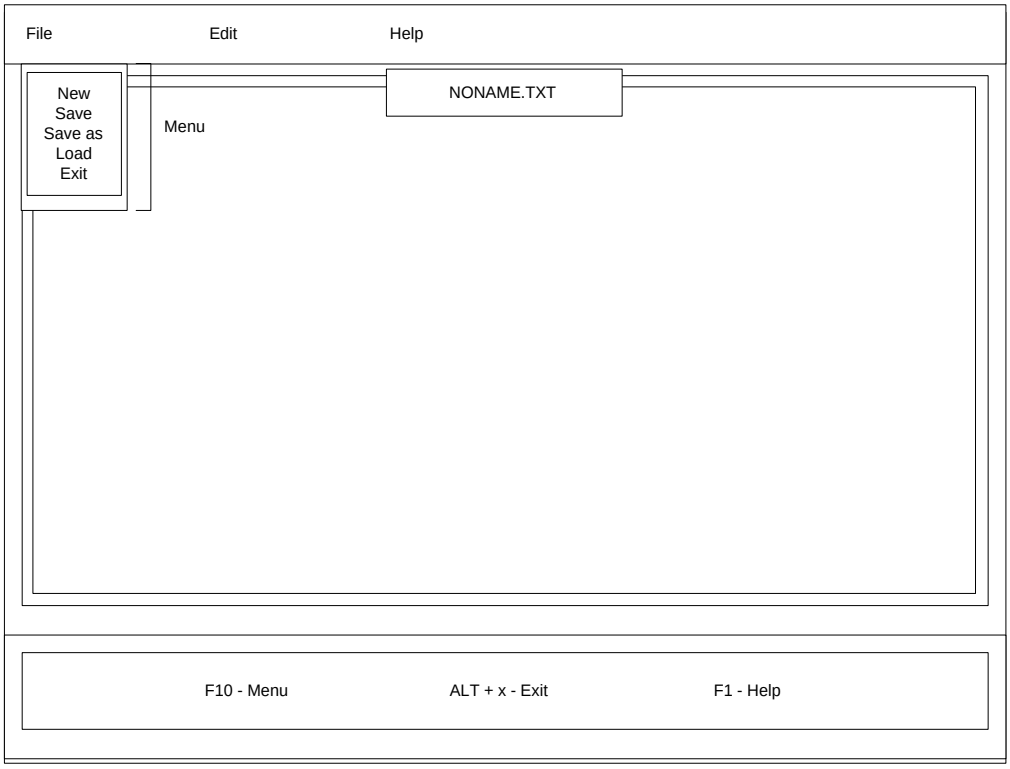
RAM의 주소 = 비디오 메모리의 시작주소(=0xb8000000) + y*160 + x*2 (x, y는 0부터 시작)

화면의 우측상단에 문자를 출력하기 위해서는 위의 공식에서 x와 y의 값에 0를 대입하면 된다. 화면상의 임의의 위치에 문자를 출력할 때는 대응되는 메모리의 위치를 계산한 다음, far 포인트를 이용하여 해당위치에 문자 값을 부여한다 (왜 far포인트를 사용해야 되는지 알아보라). 출력되는 문자를 변수 ch에 속성은 변수 attr에 있다고 가정하자. 한 문자를 화면의 특정위치에 출력하는 코드는 다음과 같다.

```
char far *location;   // location은 위의 공식에 의해서 값을 갖는다.
location = 0xb8000000 + y * 160 + x * 2 ;
*location++ = ch;
*location  = attr;
```

PROJECT #2 메뉴 만들기

이 과제에서는 앞에서 작성한 창에 다음 그림과 같은 메뉴를 만든다.

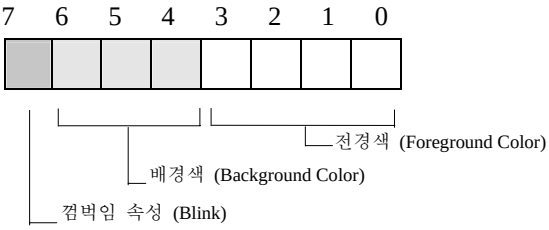


메뉴는 Pull-Down 방식으로 주메뉴 항목은 화면 상단에 나열되어 있으며, 이를 선택하면 그에 해당하는 부메뉴(Pop-Up 방식)가 나타난다. 주메뉴는 그림에서와 같이 File, Edit, Help의 세 종류로 구성된다. 주메뉴 중 File의 부메뉴는 New, Load, Save, Save as, Exit의 5종류로 규성된다. 과제에서는 프로그램을 종료시키는 exit 명령만을 실행하고 나머지 명령은 다음 과제에서 실행한다. 메뉴의 작동은 F10 키를 누르면 커서는 주메뉴로 이동한다. 일반 편집기의 기능에서와 같이 화살표키와 엔터키(선택시)를 사용한다. F10 키를 누른 후 메뉴를 지우기 위해서 Esc 키를 누르면 메뉴가 뜨기전의 화면으로 돌아가게 한다. 그러기 위해서는 화면을 저장하는 함수 gettext()와 저장된 화면을 출력하는 puttext() 함수를 사용한다. 이 두 함수에서의 화면의 좌표의 범위는 (1,1) 부터 (80, 25)이다. 라이브러리를 찾아보기 바란다.

메뉴를 그릴 때는 앞의 과제에서와 같은 방법을 사용한다. 메뉴 상에서의 커서는 역상바로 표시된다. 역상바란 선택된 문자열이나 항목을 표시하기 위해서, 전경색(글자색)과 배경색을 바꾸어 출력하여 마치 막대기가 문자열 위에 출력된 듯이 표현하는 것을 말한다. 프로그램은 F10 키를 눌렀을 경우 첫메뉴인 NEW에 커서가 위치하여 역상바로 나타난다.

역상바를 위한 속성 변경 함수

역상바를 만들기 위해 속성을 바꾸려면 전경색 속성과 배경색 속성을 서로 뒤바꾸어 주면 된다. 문자의 속성을 저장하는 1byte크기의 비트 구조는 다음과 같다.



상위 4bit(비트 4, 5, 6, 7)는 배경색을 표현하기 위해 사용하고, 하위 4bit(비트 0, 1, 2, 3)는 전경색을 표현하기 위해 사용한다. 엄밀히 따지자면 최상위 비트인 비트 7은 껌벅임 속성을 위해 사용하는 비트이지만, 프로그래밍의 편의상 배경색을 표현하는 비트로 간주한다.

이 상위 4bit와 하위 4bit를 서로 뒤바꾸어 줌으로서 우리는 역상바를 표현할 수 있다. 상위 4bit와 하위 4bit를 뒤바꾸는 VGA_inverse_attrib()함수는 다음과 같다.

[예제 프로그램]

```
void VAG_inverse_attrib(unsigned char far *attrib)
{
        unsigned char origin_attrib;

        origin_attrib = *attrib;                /* 원래 속성 값을 저장 */
        *attrib >>= 4;                          /* *attrib의 상위 4bit를 하위 4bit로 옮김 */
        *attrib = *attrib & 0x0f;               /* 0000 1111. *attrib의 상위 4bit를 0으로 만듦 */
        origin_attrib <<= 4;          /*  origin_attrib의 하위 4bit를 상위 4bit로 옮김*/
        *attrib = *attrib | origin_attrib;   /* *attrib의 상위 4bit를 origin_attrib의 상위 4 bit의 값으로 바꿈*/
}                                                       ↑
                                        origin_attrib의 상위 4bit의 값은 원래 *attrib의 하위 4bit의
값임
```

역상바를 만드는 함수[VGA_inverse_bar()]

실질적으로 역상바를 출력하려면, 우리는 화면의 얼마만큼 영역을 역상으로 할 것인가를 지정해 주어야 한다.

앞의 VGA_inverse_attrib() 함수는 1개의 문자 속성만을 역상으로 만들어 주는 것이므로, 우리는 이 함수의 문자열 길이인 length만큼 호출하여 특정 영역의 문자열을 역상으로 만들 수 있다.

이 함수의 원형을 다음과 같이 만들어 보자.

```
void VGA_inverse_bar( int x, int y, int length);
```

여기서 인수들은 다음과 같은 값을 입력받는다.

```
x       : 역상바가 시작하는 x 좌표
y       : 역상바가 시작하는 y 좌표
length: 역상바가 그려질 길이(x 좌표의 길이)
```

문자의 속성값이 저장되는 시작 메모리 시작 주소는 0xb8000000L이었지만, 문자의 속성값이 시작되는 비디오 메모리 시작 주소는 그보다 1byte 상위인 0xb8000001L이다. 이에 주의하기 바란다.

[예제 프로그램]

```
void VGA_inverse_bar(int x, int y, int length)
{
        int i = 0;
        unsigned char far *attr_,memory = (unsigned char far *) 0xb8000001L;

        attr_memory = attr_memory + y * 160 + x * 2;    /* 문자열의 속성값이 저장되어 있는 RAM 주
        for(i = 0; i < length; I++){                    /* length만큼 반복 */
                VGA_inverse_attrib(attr_memory);                /* 문자의 속성을 역상으로 만듦*/
                attr_memory += 2;
        }
}
```

PROJECT #3 커서 제어하기

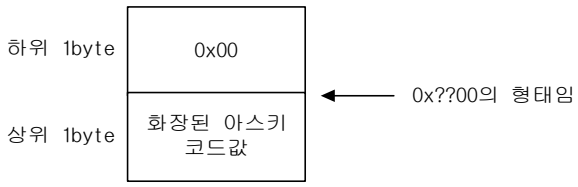
이 과제에서는 커서를 제어하여 이동시키고, 특수키 중 방향키(, , ,)와 엔터키( )를 실행한다. 커서는 이 프로그램에서 항상 사용됨으로 현재의 커서의 위치를 가지고 있는 변수를 만들어서 계속해서 추적해야 한다. 커서의 제어는 인터럽트 함수를 이용한다. 화면좌표 (x, y)에 커서가 깜빡거리는 코드는 다음과 같다. (인터럽트 함수에 대해서는 각자가 공부하기 바람)

```
void move_cursor (int page, int x, int y)  //page는 비디오메모리 페이지번호로 여기서는 0
 {      union REGS regs;
        regs.h.ah =2;
        regs.h.dh = y;
        regs.h.dl= x;
        regs.h.bh = page;       // 이 프로그램에서는 첫페이지만 사용함으로 0
        int86(0x10, &regs, &regs);      //함수에 대해서 잘 공부할 것
 }
```

이 함수를 이용하여 커서를 만든 후 자신이 원하는 위치에 커서를 이동시켜보아라. 방향키들은 화면에서 커서를 움직이는데 사용된다. 방향키가 입력되었을 경우 화면상의 커서의 위치를 바꾸어 주면 된다. 입력된 방향키의 종류에 따라 새로운 커서의 위치를 계산하

여 move_cursor() 함수를 호출한다. left나 right 방향키를 누르는 경우는 현재의 커서의 위치에서 x 좌표를 증가시키거나 감소시킨 다음 move_cursor() 함수를 호출한다. up이나 down 방향키를 누를 경우에는 처리하는 경우는 현재 행의 X 좌표의 위치와 새로운 행의 X 좌표의 위치를 비슷하게 하여 자연스럽게 보이도록 한다. 엔터키가 입력되었을 경우 커서는 현재 행 아래 행의 시작좌표(좌측)에 위치한다.

키보드에서 키를 입력하면 키의 아스키 코드값이 '키보드 버퍼'에 저장된다. 프로그램에서는 입력 함수들을 통해서 '키보드 버퍼'에 있는 값을 얻는다. 특수키의 경우는 2 byte 크기를 갖는다. 프로그램 작성을 위해서 필요한 특수키의 값을 알아야 한다 (책을 통해서 조사하기 바람). 이 경우에는 getch() 함수를 두번 호출하여 그 값을 1 byte 씩 읽어 들인다. 처음에는 하위 1 byte를 읽고, 두 번째는 상위 1 byte를 읽는다. 2 byte 크기의 아스키 코드값의 구조는 하위 1 byte의 아스키 코드값이 '0'이다. 이것을 이용하여 2 byte 크기의 키인지를 구별한다. 이것은 구조는 아래의 그림과 같다.



다음은 2 byte 크기의 아스키 코드값을 갖는 키들이다.

, ,

만약 키보드 버퍼에 1 byte 이상의 값들이 저장되어 있다면, getch() 함수를 여러번 호출해서 이 값들을 읽어서 처리한다. 이 때에는 키보드 버퍼에 먼저 저장한 값부터 차례대로 읽어 들인다. 또한 입력 함수에 의해서 읽어 들인 값들은 시스템에 의해서 키보드 버퍼로부터 자동적으로 사라진다. 키보드버퍼는 getch()함수가 접근하므로 프로그래머가 직접 다루지 않는다. 일반 문자키는 1 바이트로 구성되어 있고, 특수키(화살표키 등)는 2 바이트로 구성되어 있다. 만약 1 byte 크기의 키를 누른 경우에는 1byte 크기씩 키보드 버퍼에 저장되고, 2 byte 크기의 키를 누른 경우에는 2 byte 크기씩 키보드 버퍼에 저장된다. 하지만 읽어 들일 때에는 둘 다 1 byte 크기씩 읽어 들인다.

PROJECT #4 텍스트 입력

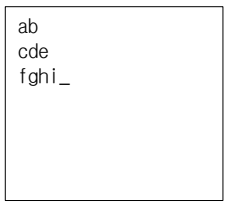
이 과제에서는 이중 연결 리스트를 이용하여 텍스트 입력을 실행한다. 텍스트 창에 입력되는 텍스트를 처리하기 위하여 구조체 line_tag와 char_tag을 이용한다.

```
struct line_tag {
  int line_number;              // 화면상 각 행의 일련번호, 문장의 번호가 아님
  int line_count;               // 화면상 각 행의 문자의 갯수
  struct line_tag *down;        // 다음 line의 구조체를 가리키는 포인터
  struct line_tag *up;          // 앞 line의 구조체를 가리키는 포인터
  struct char_tag *char_point;  // 화면상의 실제 문자를 가리키는 포인트
```

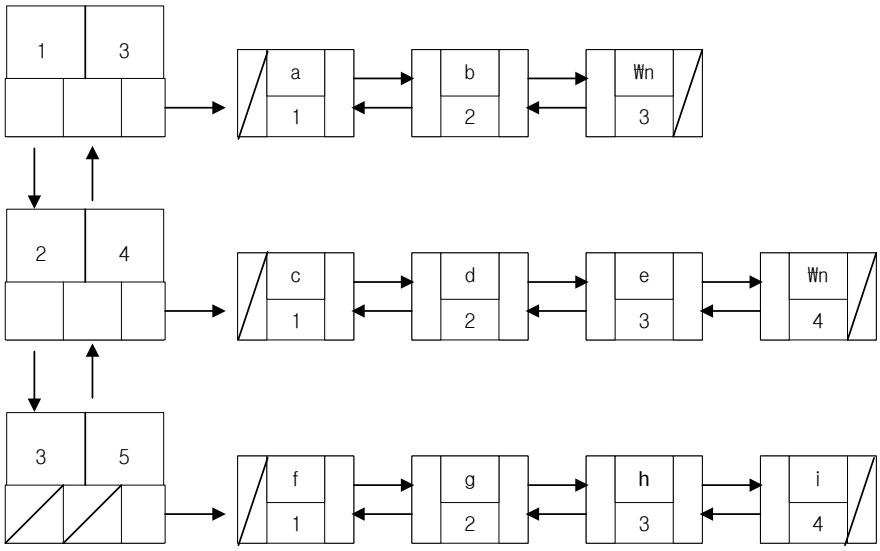
```
};

struct char_tag {
  char character;          // 입력받은 하나의 문자
  int char_number;         // 각 행에서 문자의 일련번호
  struct char_tag *prev;   // 앞의 char_tag 구조체(앞의 문자)를 가리키는 포인터
  struct char_tag *next;   // 다음 char_tag구조체(다음 문자)를 가리키는 포인터
};
```

구조체 line_tag는 텍스트의 각 라인에 대한 정보를 가지고 있으며, 구조체 char_tag는 한 라인에서 각각의 문자에 대한 정보를 가지고 있다. 화면의 내용이 그림(가)와 같을 때 텍스트의 각 문자가 저장되는 구조는 그림(나)와 같다. 구조체의 마지막 노드에 있는 '/n'은 화면에 이상한 형태의 문자로 출력된다. 이때 '/0'로 출력하면 된다. (다른 방법을 알면 다르게 해결해도 무방함) 각행의 마지막 문자는 반드시 엔터키가 되는 것은 아니라는 점을 명심하기 바란다. 입력문자가 한행에서 78문자를 넘으면 엔터키 없이 자동으로 다음 줄로 넘어가야 한다.

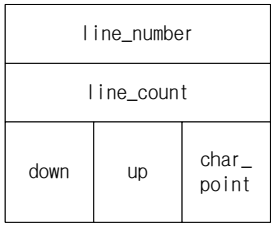
화면내용



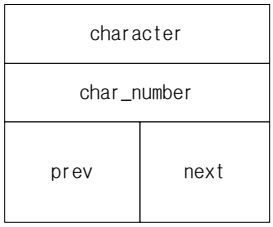
(가)



(나)



line_tag의 구조



char_tag의 구조

사용자가 문자를 입력할 때에는 getch() 함수를 호출하여 입력값을 읽는다. 이 함수는 키보드 버퍼에 저장되어 있는 값들을 1 byte 크기씩 읽어 들여

1) 새로 읽은 문자를 구조체 char_tag에 저장하여 적당한 위치에 삽입한다.
2) RAM의 적정위치에 문자값과 속성값을 기록하여 화면에 뿌려준다.
3) 커서를 다음 위치로 이동한다.

이 과제에서는 일반 문자키만 다루고, 특수키는 다음 과제에서 다루기로 한다. 화면에 문자를 출력할 때는 처음 과제와 같은 방법으로 한다.

PROJECT #5 메뉴 실행하기

이 과제에서는 주메뉴 File의 부메뉴(New, Save, Save as, Load)를 실행한다. 메뉴에서의 New는 새로운 text를 입력받을 수 있도록 텍스트 창을 깨끗이 지워주고, 커서를 좌측상단에 위치시키고 사용자의 입력을 기다리게 해준다. 이때 사용중인 text가 저장이 안되었을 때에는 저장을 하고 난 뒤 실행되도록 한다. Save는 double linked-list내에 저장된 텍스트를 디스크에 저장하는 것이다. Save할 때 여태까지 없었던 새로운 화일의 경우에는 create() 함수를 사용하여 새로운 화일을 만든다. 이때 화일을 다시 close()한 다음 다시 open() 함수를 호출해야 한다. 이것은 처음 화일을 생성한 후, 화일의 속성을 저장하기 위해서이다. file create할 때의 속성은 'S_IWRITE'이고, open할 때의 속성은 'O_WRONLY | O_TEXT'이다. 텍스트의 내용을 디스크에 넣을 때는 write() 함수를 쓴다. Save 명령의 수행에서는 한글96에서와 같이 화일의 이름을 입력받는 대화창이 생성되어 사용자의 입력을 받아야 한다. 대화창의 생성은 여태까지와 같이 상자 그리는 방법으로 만들 수 있다. 여기서는 화일을 저장할 때, 새로운 화일에 저장할 때와 기존의 화일에 저장할 때의 경우를 생각해서 프로그램을 작성하기 바란다. Save as는 Save메뉴와 비슷하나 사용중인 text를 다른 이름으로 저장하는 것이다. Load 명령은 디스크에 있는 화일을 앞의 과제에서 정의한 구조체를 이용하여 double linked-list로 구현하는 것이다. Load 명령을 수행할 때는 먼저 대화창을 통하여 사용자로부터 화일의 이름을 입력받아야 된다. 화일을 열때는 open() 함수의 속성은 'O_RDWR | O_TEXT'이다. 디스크의 내용을 double linked-list로 구현할 때 read() 함수를 쓴다. Load 명령의 수행시 화면에는 해당 화일의 내용이 출력되어야 하며, 커서는 화면의 좌측 상단에 위치하여야 한다. 화일에 텍스트를 저장하기 위해서 가능하면 저수준 입출력 함수를 이용하기 바란다. 공부해야 될 함수는 create(), close(), open(), read(), write() 등이다. 이 과제에서는 New 명령을 실행하고, 텍스트를 입력하고 저장하는 기능을 가진다. 입력된 텍스트의 추가나 삭제는 다음 과제에서 수행한다.

PROJECT #6　텍스트의 삽입과 삭제

이 과제에서는 text의 삽입과 삭제를 실행한다. text의 삽입을 3번 과제의 그림 (가)와 (나)의 예로 살펴보자. 두 번째 라인의 'd' 다음에 'q'가 삽입되는 경우

1) line_tag의 line_number가 2인 노드로 찾아간다.
2) char_tag의 character가 'c'인 노드로 찾아간다.
3) char_tag의 구조로 노드를 하나 생성하여 character와 char_number에 적당한 값을 넣는다.
4) 새로운 노드를 'c'에 해당하는 노드 다음에 삽입한다.
5) 'q'에 해당하는 노드 다음부터 라인의 마지막 노드까지 char_number를 갱신한다.

text의 삭제을 3번 과제의 그림 (가)와 (나)의 예로 살펴보자. 세번째 라인의 'g'를 삭제하는 경우

1) line_tag의 line_number가 3인 노드로 찾아간다.
2) char_tag의 character가 'g'인 노드로 찾아간다.
3) 'g'인 노드를 참고하여 앞 노드('f')의 next를 현재노드('g') next로 포인터한다.
4) 현재노드('g')의 next의 prev를 앞노드('f')로 포인터한다.
5) 현재노드('g')의 다음부터 라인의 마지막 노드까지 char_number를 갱신한다.

PROJECT #7　줄 단위 스크롤

이 과제에서는 한 줄 단위의 스크롤과 특수키인(　　,　　)을 실행한다. 커서가 Textwindows의 가장 마지막 줄이거나 첫줄에 위치할 경우, up 화살표나 down 화살표키를 누르면 화면이 스크롤 되어야 한다.

16줄만 나타낼 수 있는 공간에서 17줄 이상을 표시할 경우, 16줄이 넘어 가면 첫 번째 줄이 아니라 두 번째 줄을 탑 포인터로 잡은 다음 탑포인터부터 16줄을 혹은 NULL일 때까지 화면에 출력해 주기만 하면 스크롤을 하는 것과 같은 효과를 얻을 수 있다.

탑 포인터를 잡는 방법은 순환문을 이용하여 현재 커서위치의 라인 넘버로부터 y의 좌표가 1일 때까지 포인터를 이동해 주면 된다.

다음은 탑포인터로부터 화면에서 표시 가능한 마지막 줄까지 출력하는 예시 코드이다. 참조하기 바란다.

```
while((i < 16) && (lineprint != NULL))
{
	po = 1;
	if(print->ch == 'Wn')
		print = NULL;
	while(print != NULL)
	{
		xyadd(po, linepo+3, print->ch, 0x1f);
		po++;
```

```
                print = print->next;

                if(print->ch == '₩n')
                        print = NULL;
        }
        linepo++;
        lineprint = lineprint->down;
                if(lineprint != NULL)
                print = lineprint->charpoint;
        i++;
}
```

그리고 커서가 범위를 넘어서 스크롤 되어야 할 때 위의 코드가 실행되면, 화면상으로는 스크롤이 되고 있는 것처럼 보이게 될 것이다.

특수키인 과 은 앞의 3번 과제에서 한 방향키와 같은 구조를 가진다. 키는 현재 커서가 위치한 문자를 삭제하고, 는 커서가 위치한 앞의 문자를 삭제해 나간다. 또한 화면의 마지막 줄에서의 키를 눌렀을 때와 화면의 제일 처음에서 키를 눌렀을 경우에 줄 단위의 스크롤이 일어 나야만 한다.

PROJECT #8 화면 스크롤

이 과제에서는 한 화면 단위의 스크롤을 실행한다. 앞의 과제와 비슷한 방법으로 뿌려줄 때 16줄만 뿌려주면 스크롤과 같은 기능을 할 수 있다. 화면을 스크롤 하는데 사용되는 특수키는 , 키(page up, page down)를 사용한다.

예를 들어 (page up)키를 눌렀을 때, 탑 포인터를 Loop 문을 이용하여 16번을 위로 올려 주고 탑 포인터부터 시작하여 16줄 혹은 NULL일 때까지 출력하여 주면 스크롤하고 같은 기능을 할 수 있다.