최종 보고서

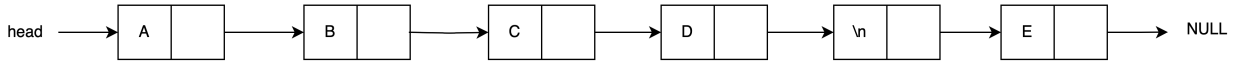
고급자료구조 visual text editor(vite) 만들기 과제

통계학과 20192084학번 박성은

1. 자료구조

- **single linked list**

  single linked list의 각 노드는 하나의 character와 다음을 가리키는 포인터를 가지는 구조이다. 다른 자료구조에 비해 시간이 오래 걸리고 단순하지만 기능 구현을 우선으로 하여 간단한 자료구조를 선택했다.




```
struct Node_{
  char data;
    struct Node_* ptr;
};
typedef struct Node_ Node;
```

- **single linked list의 함수**

  ① Node* InsertAtHeadl(Node* head, char x);

  기존의 파일을 불러오는 경우에만 사용한다. 파일의 내용을 불러와, linked list의 Head에 삽입하여 저장한다. 파일을 불러올 때 걸리는 시간을 줄이 기 위해 Tail에 삽입(InsertAtTail)하지 않고 Head에 삽입(InsertAtHead)한 뒤 뒤집는(Reverse) 방식을 채택했다.

```
Node* InsertAtHead(Node* head, char x){
  Node* tmp = (Node*)malloc(sizeof(Node));
  tmp -> data = x;
  tmp -> ptr = NULL;
  if(head != NULL) tmp -> ptr = head;
  head = tmp;
  return head;
}
```

  ② Node* Reverse(Node* head);

  기존의 파일을 불러오는 경우에만 사용한다. `InsertAtHead()` 함수를 통해 기존의 파일을 linked list에 저장한 뒤, `Reverse()` 를 통해 뒤집는다.

```
Node* Reverse(Node* head){
  Node *prev, *cur, *next;
  prev = NULL;
  cur = head;
  while(cur != NULL){
    next = cur -> ptr;
    cur -> ptr = prev;
    prev = cur;
    cur = next;
  }
  head = prev;
  return head;
}
```

③ Node* InsertNode(Node* head, char x, int n);

기존의 파일을 연 후 텍스트를 수정하거나 새로운 파 일을 생성하여 텍스트를 입력할 때 작동한다. 인자로 노드 번호를 의미하는 int n을 받아, n번 째 노드 뒤에 새로운 노드를 생성하여 삽입한다.

```
Node* InsertNode(Node* head, int n, char x){
  Node* new = (Node*)malloc(sizeof(char));
  new -> data = x;
  new -> ptr = NULL;

  if(head == NULL){//create new node
    head = new;
  }else if(n == 0){ //insert at head
    Node* tmp = head;
    head = new;
    new -> ptr = tmp;
  }else{
    Node* tmp = head;
    for(int i = 0; i < n-1; i++) tmp = tmp -> ptr;
    Node* tmp1 = tmp -> ptr;
    tmp -> ptr  = new;
    new -> ptr = tmp1;
  }
  return head;
}
```

④ Node* Delete(Node* head, int n);

인자로 노드 번호를 의미하는 int n을 받아, n번째 노드를 삭제(free)한다. backspace 키가 입력되었을 때만 작동한다.

```
Node* Delete(Node* head, int n){
```

```
  if(head == NULL) return head;

  Node* tmp = head;
  if(n == 1){
    head = tmp -> ptr;
    free(tmp);
    return head;
  }
  for(int i = 0; i < n-2; i++) tmp = tmp -> ptr;
  Node* tmp1 = tmp -> ptr;
  tmp -> ptr = tmp1 -> ptr;
  free(tmp1);

  return head;
}
```

⑤ void DeleteAll(Node* head);

vi editor를 나갈 때, malloc으로 생성되어 있는 모든 노드를 삭제(free)하기 위한 함수다. ctrl-s와 ctrl-q가 연달아 입력되거나 ctrl-q가 2회 연달 아 입력되면 해당 함수를 호출한다.

```
void DeleteAll(Node* head){
  Node* tmp = head;
  while(tmp != NULL){
    head = tmp -> ptr;
    free(tmp);
    tmp = head;
  }
}
```

2. UI

3가지 window 창 (editor, status bar, message bar) 으로 구성된다. curses.h 라이브러리의 window로 선언한다.

```
WINDOW *editor, *statusbar, *messagebar;
```

- editor

  status bar와 message bar가 하단의 2줄을 차지하므로, 전체 창 크기(LINES)에서 3줄 위까지가 editor 창의 범위이다.

  ```
  editor = newwin(LINES, COLS-2, 0, 0); //from 0 to LINES-3
  ```

- status bar(상태바)

  아래에서 두번째에 위치하는 한 행으로, 전체 창 크기에서 2줄 위에 있다. 파일 이름과 총 라인 수, 현재 커서가 위치한 라인을 표시한다. 색상 반전은 `wbkgd()`를 이용한다.

  ```
  statusbar = newwin(1, COLS, LINES-2, 0);// from LINES-2 to LINES-1

  char status[100];
  sprintf(status, "%s%s%d%s", filename, " - ", total_linenum, " lines");
  mvwprintw(statusbar, 0, 1, status);
  status[0] = '\0';
  sprintf(status, "%s%d%s%d", "no ft | ", row, "/", total_linenum);
  mvwprintw(statusbar, 0, COLS - strlen(status) , status);

  //invert background color
  start_color();
  init_pair(1, 0, 7);
  wbkgd(statusbar, COLOR_PAIR(1));
  wrefresh(statusbar);
  ```

- message bar(메세지바)

  마지막 한 행으로, 전체 창 크기에서 1줄 위에 위치해있다. 초기에는 저장, 탐색, 나가기 방법을 표시한다. Ctrl - s 입력 시, 저장 여부 문구를 표시한다.

  - 기존의 파일은 연 경우, "SAVED" 문구를 표시한다.
  - 새로운 파일을 연 경우, "SAVE MODE, put the filename: " 문구를 표시하며 파일명을 입력받을 수 있다.

```
messagebar = newwin(1, COLS, LINES-1, 0); //from LINES-1 to LINES

mvwprintw(messagebar, 0, 1, "HELP: Ctrl-S = save | Ctrl-Q = quit | Ctrl-F =
find");
wrefresh(messagebar);
```

4. 기능

- **새로운 파일 생성과 기존의 파일 열기**

  main의 argument로 int argc와 char* argv[]를 받는다. argc의 값에 따라 두 경우를 구분하고 경우에 따라 다른 과정을 거친다.

새로운 파일 생성	기존의 파일 열기
argc가 1인 경우	argc가 2인 경우
x	파일명인 argv를 확인 fopen을 통해 파일 열기 파일의 내용을 읽어 한 문자씩 linked list에 저장

  아래의 두 코드는 기존의 파일을 여는 경우이다. 먼저, 파일의 내용을 읽은 후 한 문자씩 linked list에 저장하는 코드로, `InsertAtHead()` 함수와 `Reverse()` 함수를 사용한다.

```
if( argc == 2 ){ //argument exists
  filename[0] = '\0';
  strcpy(filename, argv[1]);
  FILE *fo = fopen(filename, "r");
  if(fo != NULL){ // file exists
    char buffer[1024] = {0, };
    while(feof(fo) == 0){ //check the end of file
      int count = fread(buffer, 1, sizeof(buffer), fo);
      if(count > 0){
        for(int i = 0; i < strlen(buffer); i++) {
          head = InsertAtHead(head, buffer[i]); //using InsertAtHead function
          if(buffer[i] == '\n') total_linenum++;
        }
        memset(buffer, 0, sizeof(buffer));
      }
    }
    total_linenum++;
  }
  fclose(fo);
}
if(head != NULL) head = Reverse(head); //using Reverse function
```

  이와 같이 파일의 내용을 저장하였다면, 화면에 파일의 내용을 표시한다. 최대 라인 수가 창의 크기보다 큰 경우에는 한 페이지 즉, editor 창의 크기인 LINES - 3만큼의 라인만 표시한다. 화면에 뿌려진 라인에 대해서만 한 행의 최대 열 위치인 maxColumInaline을 계산하여 저장한다.

```
if( head != NULL ){ //when file existed, spray on the screen for the frist time
  int tmprow = 0;
  int tmpcol = 0;
  Node* p = head;
  while(p != NULL){
```

```
    tmpline[0] = '\0';
    sprintf(tmpline, "%c", p -> data);
    mvwprintw(editor, tmprow, tmpcol++, tmpline);
    if(p -> data == '\n'){
      maxColumnInaline[tmprow] = tmpcol - 1;
      tmpcol = 0;
      wmove(editor, ++tmprow, tmpcol);
      if(tmprow == LINES -2) break; // spray only first page contents
    }
    p = p -> ptr;
  }
  maxColumnInaline[tmprow] = tmpcol;
}
```

새로운 파일을 생성하는 경우는 위와 같은 과정 없이 바로 빈 에디터 창을 띄우도록 한다.

- **파일 저장 및 나가기**

  파일 저장이 되었는지 여부를 확인하기 위해 message bar에 문구를 표시한다. 새로운 파일을 생성하는 경우에는 파일명이 지정되어 있지 않으므로 파일명을 지정할 수 있도록 추가적인 입력을 받는다. 새로운 파일을 생성하는 것인지 기존의 파일을 여는 것인지를 구별하는 것은 파일명이 "[No Name]"인지의 여부에 따라 결정된다.

	새로운 파일 생성	기존의 파일 열기
파일명	"[No Name]"	main의 argument로 받은 argv
ctrl-s 입력 후의 동작	"SAVE MODE, put the filename" 문구 표시 파일명 입력 받기	"SAVED" 문구 표시

ctrl - s가 입력된 이후로, 연달아 ctrl - q가 입력되는지를 확인한다. 연달아 ctrl - q가 입력되었을 때, file을 w모드로 열어 텍스트 내용을 저장하고 vi editor를 나간다. 다음은 새로운 파일을 생성한 경우의 저장 및 나가기 과정을 간략화한 코드이다.

```
if( ch == 19 ){ //ctrl - s
  if(strcmp(filename, "[No Name]") == 0) { //case of a new file created

      ...중략...

      if(ch_tmp[i] == 17){ //when ctrl - q is entered
        FILE *fo = fopen(filename, "w"); //file open with w mode
        Node* p = head;
        while( p != NULL){
          fprintf(fo,"%c", p -> data);
          p = p -> ptr;
        }
        fclose(fo);
        break;
      }
}
```

ctrl - s없이 ctrl - q가 연달아 2번 입력될 경우에 저장 없이 나가기 동작한다. vi editor를 나갈 때에는, `DeleteAll()` 함수를 통해 생성되어 있는 모든 노드를 삭제한다.

```
if( ch == 17 ) { //ctrl - q
  char quitch = getch(); //ctrl - q is entered 2 times continuously
  if (quitch == 17) break;
}
```

- **backspace를 통한 삭제**

  특정 위치의 노드를 삭제하는 것이므로, 커서의 위치를 통해 노드 번호를 찾아내는 함수`FindingNode()`를 만든다. 찾아낸 노드 번호를 통해 해당 노드만을 삭제(free)하고 전후 노드를 이어붙이는 linked list의 `Delete()`함수를 이용한다.

  커서의 위치인 row, column과 head를 인자로 받는 `FindingNode()`함수는 다음과 같이 정의한다.

```
int FindingPlace(Node* head, int row, int column){
    if(head == NULL) return 0;
    int rowcnt = 0;
    int colcnt = 0;
    int nodecnt = 0;
    Node* p = head;
    while(p != NULL){
        if(rowcnt == row){
            if(column == colcnt) break;
            colcnt ++;
        }
        nodecnt++;
        if(p -> data == '\n') rowcnt ++;
        p = p -> ptr;
    }
    return nodecnt;
}
```

  아래는 backspace가 입력되었을 때 동작하는 과정 중 일부를 발췌한 것이다. 갱신된 linked list를 이용하여 한 페이지 분량의 내용을 새로 표시하며maxColumnInaline 또한 갱신한다.

```
else if(ch == 127) {//backspace
  if((row == 0)&(column == 0)&(startlinenum==0)) continue;
  int nodenum = 0;
  nodenum = FindingPlace(head, startlinenum + row , column);
  head = Delete(head, nodenum);
}
```

- **텍스트 입력**

  특정 위치에 노드를 삽입하는 것이므로, `FindingNode()` 함수를 사용하여 노드 위치를 찾는다. 찾아낸 노드 번호를 통해 해당 노드 뒤에 새로운 노드를 삽입하고 전후 노드를 새로운 노드와 이어붙이는 linked list의 `InsertNode()` 함수를 이용한다. 새로 갱신된 linked list를 한 페이지 분량 다시 뿌리는 것으로 마무리한다.

```
else{
  int nodenum = FindingPlace(head, startlinenum + row, column); //using
FindingPlace funtion
  if(nodenum == 0) total_linenum++;
  head = InsertNode(head, nodenum, ch); //using InsertNode function

  ...중략...

  //spray
  Node* p = head;
  int cnt = 0;
  int tmprow = 0;
  int tmpcol = 0;
  while( p != NULL ){

    ...중략...

  }

  status[0] = '\0'; ///changing line number in status bar
  sprintf(status, "%s%s%d%s", filename, " - ", total_linenum, " lines");
  mvwprintw(statusbar, 0, 1, status);
  status[0] = '\0';
  sprintf(status, "%s%d%s%d", "no ft | ", startlinenum + row + 1, "/",
total_linenum);
  mvwprintw(statusbar, 0, COLS - strlen(status) , status);
  wrefresh(statusbar);

  wmove(editor, row, column);
}
```

- **커서 이동**

  커서의 위치를 통해 노드 번호를 알아내어 특정 위치의 노드를 삽입, 삭제하므로 커서 위치 제한이 중요하다. 빈 화면에서 커서가 움직이지 않도록 반드시 문자가 있는 위치에만 커서가 이동할 수 있도록 한다. 커서는 ① 화살표 키; ② 텍스트 입력; ③ 백스페이스; 에 의해 이동하며 curses.h의 `wmove()`를 이용한다. 전체 파일의 내용이 한 페이지를 넘어가는 경우, 화면에 뿌리는 라인의 위치를 확인하기 위해 int 타입의 startlinenum을 선언하여 이용한다. 이 startlinenum은 화면에 뿌려지는 가장 첫 라인 수를 기록한다.

  ① 화살표 키

  vi를 참고하여 화살표 키의 움직임은 아래와 같이 설계한다.

  - KEY_RIGHT: 행의 마지막 문자 위치까지만 이동할 수 있다. 커서가 행의 마지막 열에 위치해있을 땐 아무런 동작을 하지 않는다.
  - KEY_LEFT: 행의 첫 글자 위치까지만 이동할 수 있다. 커서가 행의 첫 열에 위치에 있을 땐 아무런 동작을 하지 않는다.
  - KEY_UP: 현재 커서의 위치(x, y)에서 KEY_UP에 의해 한 행 위로 올라갈 때, x-1행의 최대 열(k)이 y보다 작은 경우, 커서의 위치는 (x-1, k)이 된다.
  - KEY_DOWN: 파일의 최대 라인 수 이상으로 동작하지 않도록 한다. 또한, 현재 커서의 위치(x, y)에서 KEY_DOWN에 의해 한 행 아래로 내려갈 때, x+1행의 최대 열(k)이 y보다 작은 경우, 커서의 위치는 (x+1, k)이 된다.

  파일의 내용이 한 페이지보다 많을 때는 한 행씩 위 또는 아래로 이동해야 한다. 이에 따라 KEY_UP과 KEY_DOWN는 가장 끝 행에 위치해 있을 경우와 아닌 경우를 나누어 구현했다. 다음은 KEY_LEFT, KEY_RIGHT, KEY_UP의 코드이다.

```
else if (ch == KEY_LEFT){
   if(column != 0) wmove(editor, row, --column);
}
else if (ch == KEY_RIGHT) {
   if(maxColumnInaline[row] > column) wmove(editor, row, ++column);
}
else if (ch == KEY_UP) {
  if(row > 0){ // row is not at the top
    --row;
    if(maxColumnInaline[row] < column) column = maxColumnInaline[row];
    wmove(editor, row ,column);
  }
  else if(row == 0){ //row is at the top
    if(startlinenum == 0) continue;
    else{     //there are more upper rows
      --startlinenum;
      //spray on window
      Node* p = head;
      int cnt = 0;
      int tmprow = 0;
      int tmpcol = 0;
      while( p != NULL ){
        if(cnt < startlinenum){
          if(p -> data == '\n') cnt++;
          p = p -> ptr;
```

```
            continue;
          }
          if((tmprow == LINES - 3)&(p -> data == '\n')) break;
          tmpline[0] = '\0';
          sprintf(tmpline, "%c", p -> data);
          mvwprintw(editor, tmprow, tmpcol++, tmpline);
          if(p -> data == '\n'){
            maxColumnInaline[tmprow] = tmpcol - 1;
            tmpcol = 0;
            wmove(editor, ++tmprow, tmpcol);
            tmpline[0] = '\n';
            mvwprintw(editor, tmprow, tmpcol, tmpline);
          }
          p = p -> ptr;
        }
        maxColumnInaline[tmprow] = tmpcol;
      }
    }
    //changing the current line on status bar
    status[0] = '\0';
    sprintf(status, "%s%d%s%d", "no ft | ", startlinenum + row + 1, "/",
total_linenum);
    mvwprintw(statusbar, 0, COLS - strlen(status) , status);
    wrefresh(statusbar);
    wmove(editor, row , column);
}
```

KEY_RIGHT, KEY_UP, KEY_DOWN의 동작에서는, 모든 행의 마지막 열이 어느 위치인지에 대한 정보를 필요로 한다. 이는 maxColumnInaline이라는 int 배열을 선언하여 행의 가장 마지막 열의 위치를 저장하고 해당 배열은 텍스트가 입력되거나 백스페이스가 입력될 때마다 업데이트한다.

② 텍스트 입력

현재 커서 위치를 `(row, column)`이라고 가정하면 다음과 같다.

- 개행 문자 입력: 다음 행의 첫번째 열로 이동한다. `(row + 1 , 0)`
- 개행 문자 외의 다른 문자 입력: 해당 행의 다음 열로 이동한다. `(row, column + 1)`

```
//when a new character inserts
if(ch == '\n') {
  row++;
  column = 0;
  total_linenum++;
}
else column ++;
wmove(editor, row, column);
```

③ 백스페이스

현재 커서 위치를 `(row, column)`이라고 가정하면 다음과 같다.

- 개행 문자 삭제: 이전 행의 마지막 열로 이동한다. `(row - 1, maxcolumn)`
- 개행 문자 외의 다른 문자 삭제: 해당 행의 이전 열로 이동한다. `(row, column - 1)`

```
//when backspace is entered
if(column == 0){
  --row;
  column = maxColumnInaline[row];
  total_linenum --;
}
else --column;
wmove(editor, row, column);
```

- **page up/down**

  현재의 startlinenum에 대해 editor창 크기인 LINES - 3을 더하거나 빼어 startlinenum을 갱신한다. 갱신된 startlinenum으로부터 editor 창 크기만큼의 라인을 화면에 뿌리는 방식이다. 아래의 코드는 Page up을 구현한 부분이다.

```
else if(ch == KEY_PPAGE){ // page up
  if(startlinenum != 0){ // when there is no more upper line
    startlinenum -= LINES-2; // startlinenum - size of editor window
    if(startlinenum < 0) startlinenum = 0;

    //spray
    for(int i = 0; i<= LINES - 3; i++) {
      tmpline[0] = '\n';
      mvwprintw(editor, i, 0, tmpline);
    }
    Node* p = head;
    int cnt = 0;
    int tmprow = 0;
    int tmpcol = 0;
    while( p != NULL ){
      if(cnt < startlinenum){ //pass through every text that goes startlinenum
        if(p -> data == '\n') cnt++;
        p = p -> ptr;
        continue;
      }
      if((tmprow == LINES - 3)&(p -> data == '\n')) break;
      tmpline[0] = '\0';
      sprintf(tmpline, "%c", p -> data);
      mvwprintw(editor, tmprow, tmpcol++, tmpline);
      if(p -> data == '\n'){
        maxColumnInaline[tmprow] = tmpcol - 1;
        tmpcol = 0;
        wmove(editor, ++tmprow, tmpcol);
        tmpline[0] = '\n';
        mvwprintw(editor, tmprow, tmpcol, tmpline);
      }
      p = p -> ptr;
    }
    maxColumnInaline[tmprow] = tmpcol;
    wmove(editor, row, column);

    status[0] = '\0'; //change the current line num on status bar
    sprintf(status, "%s%d%s%d", "no ft | ", startlinenum + row, "/",
total_linenum);
    mvwprintw(statusbar, 0, COLS - strlen(status) , status);
    wrefresh(statusbar);
  }
}
```

5. Makefile

운영체제별로 conditional compile하기 위해 운영체제를 확인하여, 링크할 라이브러리를 정하는 LIBS 매크로를 다르게 설정하였다. Windows의 경우 pdcurses 라이브러리를 인식하지 못하는 문제가 있어, 링크할 라이브러리를 로컬에 두어 해결했다. 다시 말해, pdcurses 라이브러리를 가져왔고 로컬 의 pdcurses 라이브러리를 인식할 수 있도록 LIB_DIRS 매크로에 로컬로 정의했다.

	Windows	Mac OS	Linux
LIBS	-lpdcurses	-lncurses	-lncurses
LIB_DIRS	-L.	x	x

```
ifeq ($(OS), Windows_NT) //if os is Windows
    LIBS = -L. -lpdcurses
else //if os is macos or linux
    LIBS = -lncurses
endif
```