자료구조 4주차 과제

제출일: 2022-09-29
제출자: 2016121150 윤준영

문1) 첨부의 namecardfix.c 파일에 대하여 물음에 답하시오.
1) 교재에 없는 erase_nl 함수를 사용한 이유는 무엇인지 gets와 fgets의 차이를 반영하여 설명하시오.
gets 함수는 문자열을 입력받을 때 문자열의 길이를 지정해주지 않아도 되지만 문자열의 길이가 저장할 공간보다 길 경우에도 저장을 계속하게 된다. 따라서 저장하지 않아야 될 공간에도 저장을 계속하게 되는 버퍼 오버플로우를 일으킬 수 있다. 이는 악용될 소지가 있기 때문에 표준에서는 gets 함수를 사용하지 않는 것을 권장하거나 최근에는 gets 함수를 금지하고 있으며 fgets 함수가 이를 대체하게 된다.
반면 fgets함수는 입력받을 문자열의 길이를 지정하여야 한다. 또, 문자열 입력을 받을 때 줄 바꿈('\n')까지 같이 읽기 때문에 그대로 문자열을 출력하게 되면 namecard를 읽을 때와 같은 상황에서 원하지 않게 줄 바꿈을 하게 된다. 따라서 erase_nl 함수를 통하여 줄 바꿈('\n')을 NULL('\0')로 바꿔주어 문자열의 끝을 나타내 준다.

2) fopen_s와 fopen 함수의 차이에 대하여 설명하시오.
fopen_s함수는 fopen함수에서 SDL 검사가 추가되어 보안 기능이 강화된 함수이다. fopen_s함수는 file pointer인자를 추가로 받아 file pointer에 직접 파일을 넣어 읽는 구조이다.

3) save와 load 시에 파일 이름을 입력받아 사용하도록 프로그램을 수정하시오.

```
5   #define NAME_SIZE  21
6   #define CORP_SIZE  31
7   #define TEL_SIZE   16
8   #define REC_SIZE   (NAME_SIZE + CORP_SIZE + TEL_SIZE)
9   #define FILE_SIZE  64
```

먼저 입력받을 문자열의 크기를 define을 이용하여 정의해 주었다.

```
174 void main(void)
175 {
176     //char* fname = "NAMECARD3.DAT";
177     char fname[FILE_SIZE];
178     char name[NAME_SIZE];
```

그 후 main함수에서 fname 변수를 FILE_SIZE 크기로 할당하여 준 후, 아래의 switch case 문을 fgets를 이용하여 파일명을 입력받을 수 있게 수정하였다.

```
206         case 4: printf("\n    Input file name to load -> ");
207             erase_nl(fgets(fname, FILE_SIZE, stdin));
208             load_cards(fname);
209             break;
210         case 5: printf("\n    Input file name to save -> ");
211             erase_nl(fgets(fname, FILE_SIZE, stdin));
212             save_cards(fname);
213             break;
```

```
NAMECARD Manager
----------------------------
1. Input  Namecard
2. Delete Namecard
3. Search Namecard
4. Load   Namecard
5. Save   Namecard
6. List   Namecard
7. Print  Namecard
8. End    Program

   : select operation -> 5

    Input file name to save -> NAMECARD4.DAT
```

NAMECARD4.DAT으로 파일을 저장한 후 프로그램을 재실행시켜 NAMECARD4.DAT를 불러와 namecard가 제대로 저장되어 있는지 확인했다.

```
    : select operation -> 4

     Input file name to load -> NAMECARD4.DAT


NAMECARD Manager
----------------------------
1. Input  Namecard
2. Delete Namecard
3. Search Namecard
4. Load   Namecard
5. Save   Namecard
6. List   Namecard
7. Print  Namecard
8. End    Program

   : select operation -> 6

Name                 Corporation                  Telephone number
-------------------  ---------------------------  ----------------
Kim                  KH                            83982
Park                 Con                           9183
```

입력한 namecard가 제대로 저장되어 있는 것을 확인할 수 있다.

문2) 첨부의 namecardfix.c 프로그램을 수정하여, 입력한 주소가 이름 순서대로 리스트에 저장되도록 프로그램을 수정하시오. (편의상 이름은 모두 알파벳 소문자로 가정한다.)
input_card 함수를 수정하여 입력받은 이름을 알파벳 순서대로 리스트에 저장되도록 하였다.

```
35  void input_card(void)
36  {
37      card* t;
38      card* s;
39      t = (card*)malloc(sizeof(card));
40      s = (card*)malloc(sizeof(card));
41      s = head;
```

먼저 차례로 참조할 namecard를 사용하기 위하여 s라는 card 포인터를 할당하고 head로 초기화하였다.

```
55      int i = 0;
56      while (s->next!=tail){
57          if (s->next->name[i] == t->name[i]) i+=1;
58          else if (s->next->name[i] < t->name[i]){
59              s = s->next;
60              i = 0;
61          }
62          else break;
63      }
64      t->next = s->next;  /* insert at s */
65      s->next = t;
```

그 후 input_card의 마지막부분을 수정하여 s와 t를 비교하여 s->next->name[0]의 값이 t->name[0]보다 작을 경우 s=s->next의 값으로, 만약 같을 경우 i를 증가시켜 차례로 다음 문자까지 비교한다. 만약 s->next->name의 값이 t->name보다 클 경우 t는 s의 다음 원소로 들어오게 된다.
컴파일하여 실행해 본 결과는 다음과 같다. 먼저 apple, bear, beem, case, ziz를 차례로 등록해 본 결과 알파벳 순서대로 list된 것을 확인할 수 있었다.

```
: select operation -> 6

Name                 Corporation                    Telephone number
-------------------  -----------------------------  ----------------
apple                casd                           1242
bear                 kdsu                           2423
beem                 sdi                            324
case                 kdu                            389
ziz                  kij                            083
```

다음으로 beaz를 추가한 화면이다. bear과 4번째 문자까지 비교하여 r<z이기 때문에 세 번째에 추가된 것을 확인할 수 있다.

```
: select operation -> 6

Name                 Corporation                    Telephone number
-------------------  -----------------------------  ----------------
apple                casd                           1242
bear                 kdsu                           2423
beaz                 qwd                            135
beem                 sdi                            324
case                 kdu                            389
ziz                  kij                            083
```

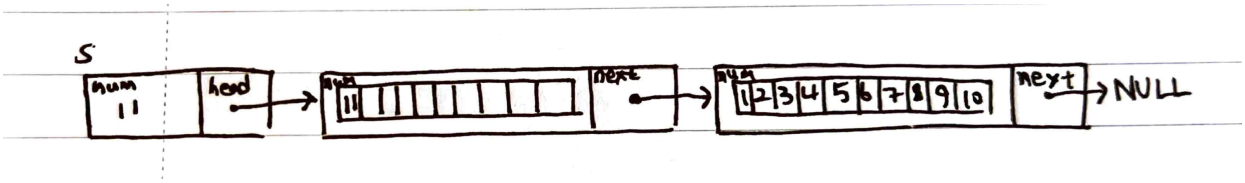
다음은 bee를 추가한 결과이다. beem과 4번째 문자까지 비교하게 되는데 bee는 3글자이기 때문에 마지막 4번째 문자는 null('\0')을 나타내게 된다. null 문자는 ASCII코드가 0이기 때문에

항상 모든 알파벳보다 작으므로 null<m이므로 4번째에 추가된 것을 확인할 수 있었다.

```
    : select operation -> 6

Name                 Corporation                  Telephone number
-------------------  ---------------------------  ----------------
apple                casd                          1242
bear                 kdsu                          2423
beaz                 qwd                           135
bee                  321                           351
beem                 sdi                           324
case                 kdu                           389
ziz                  kij                           083
```

문3) 주어진 stackPush.c 프로그램의 마지막 printStack 실행 전 스택의 상태를 그림으로 나타내시오. (연결구조 및 구조체 내부의 값들을 표시)



문4) 다음 inorder 표현식에 대한 후위표기법 및 전위표기법 식을 작성하시오.

2*(2+3*(3−4))+5

= 2*(2+3*(−1))+5 = 2*(2+(−3))+5 = 2*(−1)+5 = (−2)+5 = 3

후위 표기법: 22334−*+*5+

= (2(2(3(34−)*)+)*)5+ = (2(2(3(−1)*)+)*)5+ = (2(2(−3)+)*)5+ = (2(−1)*)5+ = (−2)5+ = 3

전위 표기법: +*2+2*3−345

= +(*2(+2(*3(−34))))5 = +(*2(+2(*3(−1))))5 = +(*2(+2(−3)))5 = +(*2(−1))5 = +(−2)5 = 3