

# 게임 프로그래밍

C

# Game Programming

게임응용모듈



# 카드 표시

트럼프 카드는 4개의 모양(♠♦♥♣)으로 나누어져 있으며 각 모양은 13장의 카드로 구성되므로 모두 52장이 사용

A (1)	2	3	4	5	6	7	8	9	10	J (11)	Q (12)	K (13)
♠	♠	♠	♠	♠	♠	♠	♠	♠	♠	♠	♠	♠

숫자(또는 문자)와 모양 즉, 두 가지 데이터로 표시되므로 구조체로 표현

```
struct card
{
    char order;
    char number;
}
```

멤버 order는 카드의 우선순위  
멤버 number는 카드의 숫자(또는 문자)



순위는 스페이드(♠) 모양을 1순위(order=0), 다이아몬드(♦)는 2순위(order=1), 하트(♥)는 3순위(order=2) 그리고 클로버(♣)는 4순위(order=3)로 가정

```
struct trump
{
    char order;
    char shape[3];
    char number;
}

trump card[52];
```

[화면 표시의 예]

A	2	3	4	10	J	Q	K
♠	♦	♥	♣	♠	♦	♥	♣

[실제 저장된 멤버의 값]

number	1	2	3	4	10	11	12	13
shape	0	1	2	3	0	1	2	3

- 멤버 order는 카드의 우선순위
- 멤버 shape는 카드 모양(2 byte크기의 완성형 기호)
- 멤버 number는 카드의 숫자(또는 문자)

카드의 모양(♠♦♥♣)은 2차원 배열에 저장하여 구분  
반복문에 의해 카드의 순위를 멤버 order에 저장  
멤버 order의 값에 따라 카드의 모양을 멤버 shape에 저장하고,  
멤버 number에는 카드의 번호를 저장하되 switch-case문으로

```
int i, j;
char shape[4][3]={"♠", "♦", "♥", "♣"};
for(i=0;i<4;i++)
{
    for(j=i*13;j<i*13+13;j++)
    {
        m_card[j].order=i;
        strcpy(m_card[j].shape, shape[i]);
        m_card[j].number=j%13+1;
        switch(m_card[j].number)
        {
            m_card[j].number에 따라 A, J, Q, K를 저장
        }
    }
}
```

1일 경우는 number에 'A'를,  
11일 경우는 'J',  
12일 경우는 'Q',  
13일 경우는 'K'를 저장



## Game Programming

```
01 void make_card(trump m_card[])
02 {
03     int i, j;
04     char shape[4][3]={"♠", "♦", "♥", "♣"};
05     for(i=0;i<4;i++)
06     {
07         for(j=i*13;j<i*13+13;j++)
08         {
09             m_card[j].order=i;
10             strcpy(m_card[j].shape, shape[i]);
11             m_card[j].number=j%13+1;
```

```
12         switch(m_card[j].number)
13         {
14             case 1:
15                 m_card[j].number='A';
16                 break;
17             case 11:
18                 m_card[j].number='J';
19                 break;
20             case 12:
21                 m_card[j].number='Q';
22                 break;
23             case 13:
24                 m_card[j].number='K';
25                 break;
26         }
27     }
28 }
29 }
```



```
01 void display_card(trump m_card[])
02 {
03     int i, count=0;
04     for (i=0; i<52; i++)
05     {
06         printf("%s", m_card[i].shape);
07         if (10<m_card[i].number)
08             printf("%-2c  ", m_card[i].number);
09         else
10             printf("%-2d  ", m_card[i].number);
11         count++;
12         if (i%13+1==13)
13         {
14             printf("\n");
15             count=0;
16         }
17     }
18 }
```

number가 10보다 클 경우에는 문자형으로 출력.

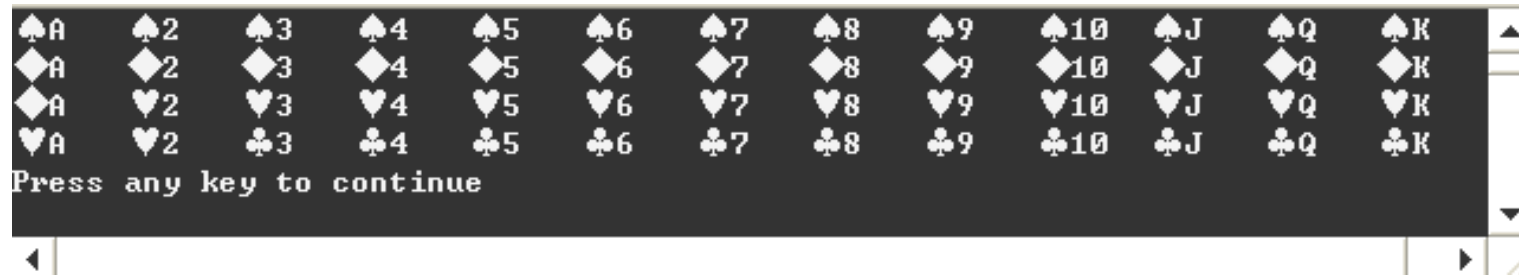
멤버 number는 숫자일 경우 최대 2자리까지 사용되지만, 문자가 저장된 경우에는 하나의 문자이므로 문자일 경우는 형식 지정자 "%-2c"(2자리 왼쪽 맞춤)를, 숫자일 경우는 "%-2d "를 사용



## Game Programming

```
01 #include <stdio.h>
02 #include <string.h>
03 struct trump {
04     int order;
05     char shape[3];
06     int number;
07 };
08
09 void make_card(trump m_card[]);
10 void display_card(trump m_card[]);
```

```
12 int main(void)
13 {
14     trump card[52];
15     make_card(card);
16     display_card(card);
17     return 0;
18 }
19 void make_card(trump m_card[])
20 {
21     앞의 함수 정의 부분 참고
22 }
23 void display_card(trump m_card[])
24 {
25     앞의 함수 정의 부분 참고
26 }
```



# 카드 섞기

- 카드를 섞는 것은 카드의 위치를 서로 교환하는 것(카드 섞기 방법 1)
- 1 단계 : 정수 난수 rnd를 발생하여 a[0]과 a[rnd]의 카드를 서로 바꾼다.
  - 2 단계 : 정수 난수 rnd를 발생하여 a[1]과 a[rnd]의 카드를 서로 바꾼다.
- 단계 반복

[카드 섞기 방법 1]	[카드 섞기 방법 2]
<pre>for (i=0; i&lt;10; i++) {     rnd=rand() %10;     temp=a[rnd];     a[rnd]=a[i];     a[i]=temp; }</pre>	<pre>for (i=0; i&lt;10; i++) {     do     {         rnd=rand() %10;     }while (rnd==i);     temp=a[rnd];     a[rnd]=a[i];     a[i]=temp; }</pre>

[카드 섞기 방법 1]에서는 발생된 난수(rnd)가 바꿀 카드의 위치 i와 같다면 카드의 위치 교환이 이루어지지 않기 때문에 [카드 섞기 방법 2]와 같이 수정

[카드 섞기 방법 2] 프로그램은 발생된 난수 rnd가 i와 같을 경우에는 다시 난수를 발생하도록 하여 위치교환이 이루어지지 않는 경우가 없도록 한다.





## 카드 섞는 함수

[카드 섞기 방법 1]

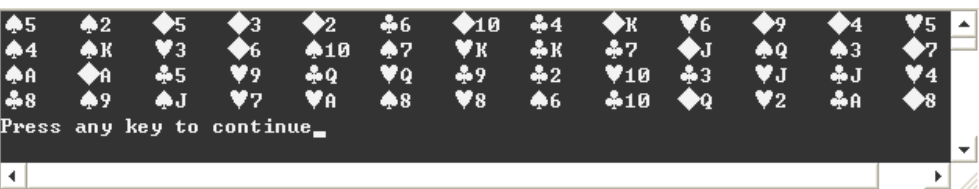
```
for (i=0; i<10; i++)  
{  
    rnd=rand() %10;  
    temp=a[rnd];  
    a[rnd]=a[i];  
    a[i]=temp;  
}
```

```
01 void shuffle_card(trump m_card[])  
02 {  
03     int i, rnd;  
04     trump temp;  
05     srand(time(NULL)); //난수의 초기화  
06     for (i=0; i<52; i++)  
07     {  
08         rnd=rand() %52;  
09         temp=m_card[rnd];  
10         m_card[rnd]=m_card[i];  
11         m_card[i]=temp;  
12     }  
13 }
```



카드 섞은 후 출력

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include <string.h>
04 #include <time.h>
05
06 struct trump {
07     int order;
08     char shape[3];
09     int number;
10 };
11
12 void make_card(trump m_card[]);
13 void display_card(trump m_card[]);
14 void shuffle_card(trump m_card[]);
```



```
16 int main(void)
17 {
18     trump card[52];
19     make_card(card);
20     shuffle_card(card);
21     display_card(card);
22     return 0;
23 }
24
25 void make_card(trump m_card[])
26 {
27     함수 정의 부분 참고
28 }
29 void display_card(trump m_card[])
30 {
31     함수 정의 부분 참고
32 }
33 void shuffle_card(trump m_card[])
34 {
35     함수 정의 부분 참고
36 }
```



# 음계와 피아노 건반

MP3와 같이 저장된 음원을 재생하는 방법이 아닌 컴퓨터상에서 기계적인 음을 표현하는 방법에 대해서 설명  
기계적인 음은 주파수를 이용하여 구분하고 표현하는데 음파의 주파수가 높으면 고음을, 낮으면 저음을 낸다.

음계(musical scale) : 음을 높이 순서로 차례로 늘어놓은 것  
음 높이의 차이(음정, tone)는 절대음정을 기준으로, 음과 음 사이의 주파수 비율을 동일하게 하여 얻어짐.  
절대음정은 음정의 기준이며, 피아노 건반의 정 중앙에 위치한 라(A)를 기준으로 하고, 라(A)의 주파수는 440Hz로 정해져 있다.  
따라서 솔이나 시의 음정은 라(A)를 기준으로 계산한 주파수의 값으로 나타낸다.

한 옥타브(octave)를 똑같은 비율로 나눈 음률(tune)을  
평균율이라고 하며, 한 옥타브를 그림과 같이  
12개의 반음정(semitone)으로 나눈 12평균율을  
가장 많이 사용한다

옥타브는 1:2의 음정비를 가지므로 그림에서  
도②의 주파수는 도① 주파수의 두 배가 된다.



12평균율의 음계는 1:2의 음정비를 갖는 한 옥타브를 12개로 균등하게 나눈 것으로서, 모든 음계에 동일한 주파수의 비,  $\sqrt[12]{2}:1$  (약 1.0594:1)을 사용 따라서 각 음계의 주파수는 라(440Hz)를 기준으로 표와 같이 계산한다

음계	음의 주파수 (Hz)	한 옥타브 아래	한 옥타브 위
도	$277.313/1.0594=261.764$	$261.764/2$	$261.764\times 2$
도#	$293.786/1.0594=277.313$	$277.313/2$	$277.313\times 2$
레	$311.237/1.0594=293.786$	$293.786/2$	$293.786\times 2$
레#	$329.724/1.0594=311.237$	$311.237/2$	$311.237\times 2$
미	$349.310/1.0594=329.724$	$329.724/2$	$329.724\times 2$
파	$370.059/1.0594=349.310$	$349.310/2$	$349.310\times 2$
파#	$392.041/1.0594=370.059$	$370.059/2$	$370.059\times 2$
솔 (G)	$415.329/1.0594=392.041$	$392.041/2$	$392.041\times 2$
솔# (G#)	$440/1.0594 = 415.329$	$415.329/2$	$415.329\times 2$
라 (A)	440 (기준)	$440/2$	$440\times 2$
라# (A#)	$440\times 1.0594 = 466.136$	$466.136/2$	$466.136\times 2$
시 (B)	$466.136\times 1.0594=493.824$	$466.136/2$	$466.136\times 2$
도 (C)	$493.824\times 1.0594=523.157$	$493.923/2$	$493.923\times 2$



앞의 표에서 사용한 계산방법으로 5개의 음계에 대한 주파수 값에 대해 소수 이하 첫째 자리 반올림을 하여 얻은 결과는 아래 표와 같다.

음계 octave	도 (C)	C#	레 (D)	D#	미 (E)	파 (F)	F#	솔 (G)	G#	라 (A)	A#	시 (B)
1	33	35	37	39	41	44	46	49	52	55	58	62
2	65	69	73	78	82	87	92	98	104	110	117	123
3	131	139	147	156	165	175	185	196	208	220	233	247
4	262	277	294	311	330	349	370	392	415	440	466	494
5	523	554	587	622	659	698	740	784	831	880	932	988
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976

표에서 옥타브 1의 도(C)는 소수 첫째 자리에서 반올림하여 33이지만 정확한 주파수의 값은 32.7032이며, 이 값을 음계 1의 도(C)로 사용.  
옥타브는 1:2의 음정 비를 가지므로 옥타브 2의 도(C)는 옥타브 1의 도(C)×2(=65.4064)로 계산



## 옥타브에 따른 음계 별 주파수를 출력하는 함수

```
01 void print_frequency(int octave)
02 {
03     double do_scale=32.7032;
04     double ratio=pow(2., 1/12.), temp;
05     int i;
06     temp=do_scale*pow(2, octave-1);
07     printf("%d옥타브 : ", octave);
08     for(i=0; i<12; i++)
09     {
10         printf("%4ld ", (unsigned long) (temp+0.5));
11         temp*=ratio;
12     }
13     printf("\n");
14 }
```



## 음계의 주파수 출력

```
01 #include <stdio.h>
02 #include <math.h>
03 void print_frequency(int octave);
04 int main(void)
05 {
06     char *scale[]={ "도", "도#", "레", "레#", "미",
07                     "파", "파#", "솔", "솔#", "라", "라#", "시", "도" };
08     int i, octave, count=0;
09     printf("음계와 주파수\n\n음계\t");
10     for(i=0;i<12;i++)
11         printf("%-5s",scale[i]);
12     printf("\n");
13     for(i=0;i<=70;i++)
14         printf("-");
15     printf("\n");
16     for(octave=1;octave<7;octave++)
17         print_frequency(octave);
18     return 0;
19 }
```



# 주파수를 이용한 음의 출력

DOS 환경의 Turbo C에는 주파수를 이용하여 컴퓨터 스피커로 소리는 sound 라는 함수가 있었고, Visual C++의 경우에는 ANSI C 표준함수는 아니지만 Win API 함수로서 주파수의 값과 지속시간을 이용하여 소리를 출력하는 라이브러리 함수 Beep를 이용. 함수 Beep는 헤더파일 <windows.h> 필요

Beep	함수원형	BOOL WINAPI Beep( __in  DWORD dwFreq, __in  DWORD dwDuration );	
	함수인자	DWORD dwFreq	소리의 주파수로서 37부터 32,767 사이의 값만을 사용한다.
		DWORD dwDuration	소리의 지속시간을 의미하며 밀리초 (milliseconds)로 나타낸다.
	반환 값	정상적으로 실행되면 0이 아닌 값을 반환한다.	





옥타브 4에 대해 반음을 제외하고, 지연시간을 0.5초로 가정하여  
함수 Beep를 사용한 예

옥타브 4에 대한 함수 Beep의 사용방법

음계	주파수 (옥타브 4)	함수 Beep
도	262	Beep (262, 500) ;
레	294	Beep (294, 500) ;
미	330	Beep (330, 500) ;
파	349	Beep (349, 500) ;
솔	392	Beep (392, 500) ;
라	440	Beep (440, 500) ;
시	494	Beep (494, 500) ;
도	523	Beep (523, 500) ;



앞의 표에서 옥타브 4에 대한 주파수를 계산하기 위해서 정수형 주파수를 반환하는 함수 `calc_frequency`로 수정하여 사용

함수 `calc_frequency`의 첫 번째 매개변수는 옥타브를 의미하며, 두 번째 매개변수는 계산할 음계의 `index`를 의미.

음계의 `index`를 정의하는 다음의 배열 `index[]`는 도#, 레#, 파#, 솔#, 라#을 제외한 나머지 음계에 대한 `index`를 의미하며 다음과 같이 정의하여 사용. 함수 `calc_frequency`의 호출방법은 표와 같다.

```
int index[]={0, 2, 4, 5, 7, 9, 11, 12}
```

음계	index[]	함수 calc_frequency
도	index[0]=0	calc_frequency(4, index[0])
레	index[1]=2	calc_frequency(4, index[1])
미	index[2]=4	calc_frequency(4, index[2])
파	index[3]=5	calc_frequency(4, index[3])
솔	index[4]=7	calc_frequency(4, index[4])
라	index[5]=9	calc_frequency(4, index[5])
시	index[6]=11	calc_frequency(4, index[6])
도	index[7]=12	calc_frequency(4, index[7])

## 옥타브에 따른 음계별 주파수를 반환하는 함수

```
01 int calc_frequency(int octave, int inx)
02 {
03     double do_scale=32.7032;
04     double ratio=pow(2., 1/12.), temp;
05     int i;
06     temp=do_scale*pow(2, octave-1);
07     for (i=0; i<inx; i++)
08     {
09         temp=(int) (temp+0.5);
10         temp*=ratio;
11     }
12     return (int) temp;
13 }
```



## 스피커로 소리내기

```
01 #include <stdio.h>
02 #include <math.h>
03 #include <windows.h>
04 int calc_frequency(int octave, int inx);
05 int main(void)
06 {
07     int index[]={0, 2, 4, 5, 7, 9, 11, 12};
08     int freq[8];
09     int i;
10     for(i=0;i<8;i++)
11         freq[i]=calc_frequency(4, index[i]);
12     for(i=0;i<=7;i++)
13         Beep(freq[i],500);
14     sleep(1000); //음간의 지연시간 1초
15     for(i=7;i>=0;i--)
16         Beep(freq[i],500);
17     return 0;
18 }
19 int calc_frequency(int octave, int inx)
20 {
21
22 }
```

실행하면 화면상에는 아무것도 나타나지 않는 상태에서 "도레미파솔라시도"의 음계를 스피커로 출력하고, 이어서 역으로 "도시라솔파미레도"를 출력



## 숫자 키에 따라 해당 음을 출력

[숫자 키와 해당 음계의 계산]

숫자 키	음계	함수 calc_frequency
1	도	calc_frequency(4, index[0])
2	레	calc_frequency(4, index[1])
3	미	calc_frequency(4, index[2])
4	파	calc_frequency(4, index[3])
5	솔	calc_frequency(4, index[4])
6	라	calc_frequency(4, index[5])
7	시	calc_frequency(4, index[6])
8	도	calc_frequency(4, index[7])



숫자 키에 따라 음계의 주파수를 스피커로 출력 함수

practice\_piano

```
01 void practice_piano(void)
02 {
03     int index[]={0, 2, 4, 5, 7, 9, 11, 12};
04     int freq[8], code, i;
05     for(i=0;i<8;i++)
06         freq[i]=calc_frequency(4, index[i]);
07     do
08     {
09         code=getch();
10         if ('1'<=code && code<='8')
11         {
12             code-=49;
13             Beep(freq[code],300);
14         }
15     }while (code!=27);
16 }
```

키보드의 숫자 키 입력에 있어서 scanf를 사용한다면 숫자 입력 후에 Enter키를 눌러야 하므로 음의 연결이 자연스럽지 못하기 때문에 Enter키가 필요 없는 getch를 이용.

그러나 getch는 숫자 키를 문자로 입력받기 때문에 문자 '1'을 입력했을 경우 숫자 0 (배열의 첫 번째 첨자)으로 만들어주기 위해 원래의 값에서 49를 빼 주어야 한다.



```
01 #include <stdio.h>
02 #include <math.h>
03 #include <conio.h>
04 #include <windows.h>
05
06
07 int calc_frequency(int octave, int inx);
08 void practice_piano(void);
09
10 int main(void)
11 {
12     printf("1부터 8까지 숫자 키를 누르면\n");
13     printf("각 음의 소리가 출력됩니다.\n\n");
14     printf("1:도 2:레 3:미 4:파 5:솔 6:라 7:시 8:도\n");
15     printf("프로그램 종료는 Esc키 \n");
16     practice_piano();
17     return 0;
18 }
20 void practice_piano(void)
21 {
22
23 }
24 int calc_frequency(int octave, int inx)
25 {
26     -
27 }
```



## 자료 구조

### 자료 구조(data structure)란

컴퓨터상에서 정보를 표현하기 위하여 데이터와 이들 데이터간의 상호 관계를 정의하는 것.  
즉, 컴퓨터상에서 데이터를 효율적으로 탐색하고, 추가하거나 삭제하는데 있어서 어떻게 데이터를 저장하거나 연결해야 하는가 다루는 분야

자료 구조는 데이터 간의 관계를 어떻게 나타낼 것인가에 따라 선형 구조와 비선형 구조로 구분

선형 구조 : 연결 리스트(linked list), 스택(stack), 큐(queue), 데크(dequeue) 등

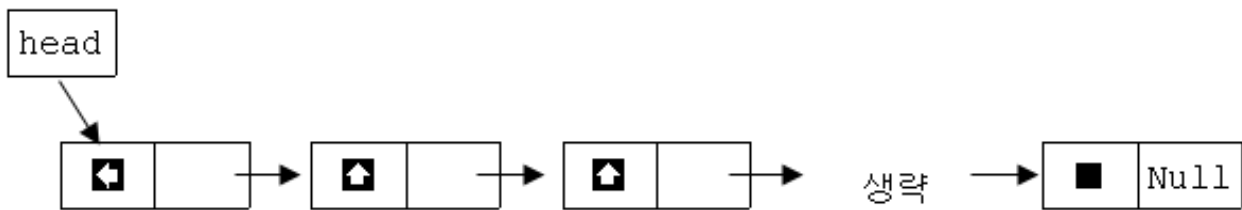
비선형 구조 : 트리(tree)와 그래프(graph) 등

자료 구조 중에서 선형 구조인 연결 리스트와 스택에 관련된 알고리즘들이 생활 속에서 어떻게 응용되는지에 대해 살펴본다.

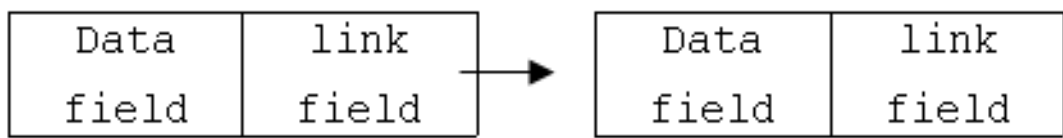




Game에서의 replay는 연결 리스트를 활용하는 예



연결 리스트에서 데이터는 노드(node)에 저장. 각 노드들은 데이터를 저장하는 데이터 필드(data field)와 다음 노드를 가리키는 링크 필드(link field)로 구성



Game 중에 발생하는 데이터를 어떻게 저장할 것인가?  
버튼의 조작이 있을 때마다 필요한 만큼 메모리를 확보하여 데이터를 연결 리스트에 순서대로 저장한다면 필요한 만큼의 메모리만 사용하게 될 것이고, 연결 리스트가 시작되는 위치(head)만 기억하고 있다면 저장된 버튼 데이터를 순서대로 불러들여 게임을 재연할 수 있다.

찜질방에서 볼 수 있는 새 수건 더미는 스택의 좋은 예  
새 수건은 항상 위쪽에 올려놓을(추가) 것이고, 가져가는 사람들 역시 항상 제일 위에 있는 수건부터 한 장씩 차례로 가져(삭제)가는 구조

스택에 데이터를 저장할 경우에는 항상 위(top)에서 데이터가 추가되고, 데이터를 불러올 경우에도 역시 위(top)에서부터 차례로 이루어지므로 스택은 데이터의 추가와 삭제가 한쪽에서만 이루어지는 자료구조

데이터 추가		데이터 삭제	
top	↓	top	
	D		↑
	C		C
	B		B
bottom	A	bottom	A



배열을 사용할 때 가장 큰 문제는 배열의 크기를 미리 정해야 한다는 것

미리 정한 배열의 크기보다 적게 사용된다면 메모리를 낭비하게 된다.  
따라서 메모리를 효율적으로 사용하기 위해서는 프로그램의 실행에 필요한 만큼의 기억 공간을 확보하는 방법이 필요

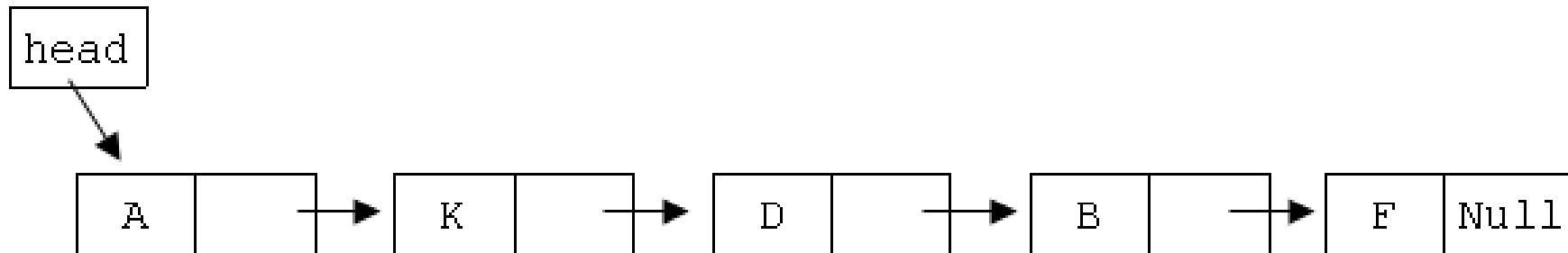
해결방법 : 정보가 생성될 때마다 입력된 정보에 맞는 데이터 형으로 메모리에 저장하고, 만약 그 정보가 필요 없게 되었다면 다른 정보를 저장할 수 있는 메모리 공간으로 돌려주는 것.

이러한 방법을 동적 메모리 관리(dynamic memory management)라 하며, 동적 할당(dynamic allocation)과 반환(deallocation)을 이용

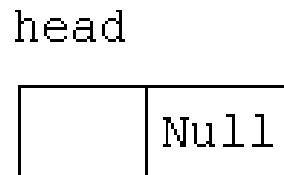


# 연결 리스트

연결 리스트(linked list)는 각 노드가 한 줄로 연결되는 방식으로 데이터를 저장하는 자료 구조로서 임의의 위치에서 데이터의 삽입과 삭제가 용이함.  
 각 노드들이 하나의 링크 필드를 갖고, 마지막 노드의 링크 필드가 Null인 연결 리스트를 단순 연결 리스트(singly linked list)



단순 연결 리스트를 생성하기 위해 먼저 head를 생성합니다. head의 데이터 노드에는 아무것도 없고, 첫 번째 만들어진 노드이므로 head의 링크 필드에는 Null을 저장



단순 연결리스트에서 데이터의 삽입

단계	설명	연결 리스트 구조
단계 1	리스트의 마지막으로 이동	head <div><div></div><div>Null</div></div>
단계 2	새 노드를 생성 새 노드의 data← 'A' 새 노드의 link←마지막 노드의 link	<div><div>A</div><div>Null</div></div>
단계 3	마지막 노드의 link←새 노드의 주소	head <div><div></div><div></div> → <div><div>A</div><div>Null</div></div></div>

```
struct linked_list
{
    char data;
    struct linked_list *link;
};
```

C 언어에서 연결 리스트는 자기 참조 구조체(self-referential structure)와 동적 할당을 통해 구현연결 리스트는 리스트에 데이터를 추가하거나 삭제할 경우 나머지 데이터에 대해서 데이터의 이동이 필요 없는 자료 구조

### 연결 리스트에서 데이터를 삽입(추가)하는 함수

연결 리스트에서 데이터를 삽입 (추가) 하는 함수 add\_node

```
01 void add_node(char data)
02 {
03     struct linked_list *new_node, *last;
04     last=head;
05     while(last->link != NULL)
06         last=last->link;
07     new_node = (struct linked_list*)malloc(sizeof(struct linked_list));
08     new_node->data=data;
09     new_node->link=last->link;
10     last->link=new_node;
11 }
```



연결 리스트에 데이터를 삽입(추가) 및 출력

키보드를 누를 때마다 동적 할당을 통해 새 노드를 만들고 입력된 키값을 단순 연결 리스트에 삽입. 키보드 입력의 종료는 Esc 키를 사용  
Esc키를 누르면 현재까지 입력된 키값을 순서대로 출력하고, 이어서 역순으로 출력하는 방법은 재귀적인 방법을 이용

사용자 정의 함수	기능
add_node;	단순 연결 리스트에 데이터 삽입 (추가)
print_linked_list;	단순 연결 리스트에 저장된 데이터를 입력된 순서대로 출력
print_reverse_linked_list	단순 연결 리스트에 저장된 데이터를 역순으로 출력 (재귀적 방법)



# Game Programming

```
16 int main(void)
17 {
18     int count=0;
19     char key;
20     head = (struct linked_list*)malloc(sizeof(struct linked_list));
21     head->link=NULL;
22     do
23     {
24         count++;
25         printf("%2d번 문자입력 >",count);
26         key=getch();
27         add_node(key);
28         printf("%c \n", key);
29     }while(key!=27);
30
31     printf("\n\n");
32     printf("입력된 순서 : ");
33     print_linked_list(head->link);
34     printf("\n");
35     printf("입력의 역순 : ");
36     print_reverse_linked_list(head->link);
37     printf("\n");
38     return 0;
39 }
```

```
40 void add_node(char data)
41 {
42
43 }
44
45 void print_linked_list(struct linked_list *now)
46 {
47     while(now!=NULL)
48     {
49         printf("%c ", now->data);
50         now=now->link;
51     }
52     printf("\n");
53 }
54
55 void print_reverse_linked_list(struct linked_list *now)
56 {
57     if (now->link!=NULL)
58     {
59         print_reverse_linked_list(now->link);
60         printf("%c ", now->data);
61     }
62     else
63         printf("%c ", now->data);
64 }
```














스택은 데이터의 추가와 삭제가 한쪽에서만 이루어지는 자료 구조  
스택에 데이터를 저장할 항상 위(top)에서 데이터가 추가되고, 데이터를 불러올 경우에도  
역시 위(top)에서부터 차례로 이루어진다.

데이터 추가		데이터 삭제	
top	↓	top	
	D		↑
	C		C
	B		B
bottom	A	bottom	A

원 카드(one card)게임에서 같은 무늬나 숫자를 가지고 있어서 카드를 내려놓은 경우, 버리는 카드의 스택 위에서부터 데이터의 추가가 일어나고, 같은 무늬나 숫자가 없어서 카드를 가져가는 경우에는 가져가는 카드의 위에서부터 차례로 데이터의 삭제가 일어난다.



스택에서 데이터의 삽입(추가)

단계	설명	연결 리스트 구조						
단계 1	top 노드 생성	top <table border="1"><tr><td></td><td>Null</td></tr></table>		Null				
	Null							
단계 2	새 노드 생성 새 노드의 data←'A' 새 노드의 link←top top← 새 노드의 주소	top <table border="1"><tr><td>A</td><td></td></tr></table> <table border="1"><tr><td></td><td>Null</td></tr></table>	A			Null		
A								
	Null							
단계 3	새 노드 생성 새 노드의 data←'B' 새 노드의 link←top top← 새 노드의 주소	top <table border="1"><tr><td>B</td><td></td></tr></table> <table border="1"><tr><td>A</td><td></td></tr></table> <table border="1"><tr><td></td><td>Null</td></tr></table>	B		A			Null
B								
A								
	Null							

```
struct stack_node
{
    char data;
    struct stack_node *link;
};
```



스택에 데이터를 삽입 (push) 하는 사용자 정의 함수 push

```
01 void push(char data)
02 {
03     struct stack_node *new_node;
04     new_node = (struct stack_node*)malloc(sizeof(struct stack_node));
05     new_node->data=data;
06     new_node->link=top;
07     top=new_node;
08 }
```



## 스택에서 데이터 삭제

현재 top에 있는 데이터를 반환하고 동적으로 할당된 메모리를 다른 프로그램이 사용할 수 있는 메모리로 해제하기 위해 함수 free를 사용  
top에 있는 데이터가 삭제되므로 이전 top의 링크 노드가 가리키는 주소를 현재의 top으로 전환

스택에서 데이터를 삭제 (pop) 하는 함수 pop

```
01 char pop(void)
02 {
03     struct stack_node *temp;
04     char data;
05     data=top->data;
06     temp=top;
07     top=top->link;
08     free(temp);
09     return data;
10 }
```



# Game Programming

```
17 int main(void)
18 {
19     int count=0;
20     char key;
21     do
22     {
23         count++;
24         printf("%2d 번 문자입력 >",count);
25         key=getch();
26         push(key);
27         printf("%c \n", key);
28     }while(key!=27);
29     printf("\n\n");
30     printf("데이터 pop과정 \n\n");
31     while (top != NULL)
32     {
33         key = pop();
34         printf("%c ",key);
35     }
36     printf("\n");
37     return 0;
38 }
```

```
39 void push(char data)
40 {
41
42 }
43
44 char pop(void)
45 {
46
47 }
```



## 데이터 정렬

텍스트 파일로부터 컴퓨터 용어를 읽어 들인 다음 검색을 하기 전에 알파벳 순서로 정렬  
대문자와 소문자가 혼용된 경우에 주의

대문자 'A'는 10진수 ASCII 코드로 65이고, 소문자 'a'는 97이므로 같은 문자라 하더라도 비교에서  
차이가 나기 때문에 대문자가 포함된 문자열에 대해 정렬하면 잘못된 결과를 얻게 된다.

방법은 문자열에 대문자가 포함된 경우 모두 소문자로 변환한 다음 비교해야 하며, 함수  
`upper_to_lower`를 이용

대문자가 포함된 원본 문자열은 그대로 둔 상태에서 소문자로만 비교한 다음 알파벳 순서로 정렬하  
여 배열에 저장하는데 일시적으로 소문자로 변환

<https://visualgo.net/>



## Reference

- ✓ <https://yeolco.tistory.com/64>
- ✓ <https://jaimemin.tistory.com/212>
- ✓ <https://manniz.tistory.com/entry/C%EC%96%B8%EC%96%B4-%EB%A3%A8%ED%8A%B8%EC%A0%9C%EA%B3%B1%EA%B7%BC-%EC%9D%84-%EA%B5%AC%ED%95%98%EB%8A%94-2%EA%B0%80%EC%A7%80-%EB%B0%A9%EB%B2%95sqrt-pow-%EC%86%8C%EC%8A%A4-%EC%BD%94%EB%93%9C>
- ✓ [https://e-koreatech.step.or.kr/page/lms/?m1=course%25&m2=course\\_list%25&search\\_keyword=%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0%25&startCount=0%25&filter\\_list=query%3D%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0%26sort%3Dcreation\\_day%2FDESC%25](https://e-koreatech.step.or.kr/page/lms/?m1=course%25&m2=course_list%25&search_keyword=%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0%25&startCount=0%25&filter_list=query%3D%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0%26sort%3Dcreation_day%2FDESC%25)
- ✓ <https://dojang.io/mod/page/view.php?id=285>
- ✓ <https://claudu.tistory.com/72>
- ✓ 명품 C언어 프로젝트, 생능출판, 안기수
- ✓ <https://visualgo.net/>

