자료구조

김태원

2023년 10월 9일

제 1 장

C 리뷰

1. 입력으로 하나의 양의 정수를 받은 후 0이 될 때까지 연속적으로 2로 나눈 몫을 출력하는 프로그램을 작성하라.

```c
#include <stdio.h>
int main(){
  int n;
  scanf("%d", &n);
  for(int i=n; i!=0; i/=2)
    printf("%d ", i);
}
```

쉬운 문제다. `i/=2`는 그냥 멋내는 용이다.

2. 입력으로 하나의 양의 정수 n을 받은 후 다음의 합을 구하여 출력하는 프로그램을 작성하라. 단, 소수점 4자리까지만 출력하라.

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}$$

```c
#include <stdio.h>
int main(){
  int n;
  double sum = 0;
  for(double r=1; r<=n; r++)
    sum = sum + 1/r;
  printf("%.4f\n", sum);
}
```

5번 라인의 `for`문에서 `r`을 `double`로 선언했다. `int`로 선언하면 모든 입력에 대해 출력은 1이기 때문이다.

3. 입력으로 하나의 양의 정수 n을 받은 후 다음의 합을 구하여 출력하는 프로그램을 작성하라.

단, 소수점 4자리까지만 출력하라.

$$1+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}$$

```c
#include <stdio.h>
int main(){
  int n;
  double sum = 0;
  scanf("%d", &n);
  for(double i=1; i<=n; i++){
    double frac = 1;
    for(double j=i; j>0; j--)
      frac *= j;
    sum += 1/frac;
  }
  printf("%.4f\n", sum);
}
```

순서대로 해결했다. 9번 라인이 아니라 7번 라인에서 `frac`을 선언한 이유는 10번 라인에서 `frac`이 쓰이기 때문이다.

4. 먼저 입력될 정수의 개수 $n \leq 100$을 입력받고, 이어서 n개의 정수를 받아 평균과 표준편차를 계산하여 소수점 이하 4번째 자리까지 출력하는 프로그램을 작성하라. 표준편차는 다음과 같이 정의된다. 루트를 계산하기 위해서 `math.h`를 `include`하고 `sqrt`함수를 사용하라.

$$SD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})^2}$$

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main(){
  int n;
  int *arr;
  double avg = 0;
  double sum = 0;
  scanf("%d", &n);
  arr = (int *)malloc(sizeof(int)*n);
  for(int i=0; i<n; i++){
    int p;
    scanf("%d", &p);
```

```
14      arr[i] = p;
15      avg += arr[i];
16    }
17    for(int i=0; i<n; i++)
18      sum += pow(arr[i]-avg/n, 2.0);
19    printf("%.5g\n", sqrt(sum/n));
20    free(arr);
21  }
```

.5g라는 formant specifier를 사용했는데, 가령 4.5000...를 4.5로 잘라서 출력하기 위해서다. 참고로 `math.h` 라이브러리를 포함한 파일을 컴파일하려면 `-lm` 명령어를 덧붙여야 한다.

5. 먼저 입력될 정수의 개수 $2 \leq n \leq 100$을 입력받고, 이어서 n개의 정수를 입력받는다. 입력된 정수들 중에서 최소값과 두 번째로 작은 값을 찾아 출력하는 프로그램을 작성하라. 만약 최소값이 2개 이상 중복되어 존재하면 그 중 하나를 최소값으로, 다른 하나를 두 번째로 작은 값으로 간주한다.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(){
4    int n, key;
5    int *arr;
6    scanf("%d", &n);
7    arr = (int*)malloc(sizeof(int)*n);
8    for(int i=0; i<n; i++)
9      scanf("%d", &arr[i]);
10   for(int i=1; i<n; i++){
11     key = arr[i];
12     int j;
13     for(j=i-1; j>=0 && arr[j]>key; j--)
14       arr[j+1] = arr[j];
15     arr[j+1] = key;
16   }
17   printf("%d %d\n", arr[0], arr[1]);
18   free(arr);
19 }
```

10번 라인에서 시작되는 `for` 블록은 삽입정렬 알고리즘을 구현한 것이다. 삽입 정렬 알고리즘은 아

Algorithm 1: Insertion Sort

Input: A: Array
for $i \in \{2, \ldots, A.length\}$ **do**
  $key \leftarrow A[i]$
  $j \leftarrow i - 1$
  while $j > 0$ & $A[j] > key$ **do**
    $A[j+1] \leftarrow A[j]$
    $j \leftarrow j - 1$
  $A[j+1] \leftarrow key$

래와 같은 방식으로 작동한다.

$A = [7, 3, 1, 2, 4, 6]$ 첫 번째 for 루프: $7 = A[1] > key = A[2] = 3$

$\mapsto [3, 7, 1, 2, 4, 6]$

두 번째 for 루프: $7 = A[2] > key = A[3] = 1$

$\mapsto [3, 1, 7, 2, 4, 6]$

두 번째 for 루프: $3 = A[1] > key = A[3] = 1$

$\mapsto [1, 3, 7, 2, 4, 6]$

세 번째 for 루프: $7 = A[3] > key = A[4] = 2$

$\mapsto [1, 3, 2, 7, 4, 6]$

세 번째 for 루프: $3 = A[2] > key = A[4] = 2$

$\mapsto [1, 2, 3, 7, 4, 6]$

네 번째 for 루프: $7 = A[4] > key = A[5] = 4$

$\mapsto [1, 2, 3, 4, 7, 6]$

다섯 번째 for 루프: $7 = A[5] > key = A[6] = 6$

$\mapsto [1, 2, 3, 4, 6, 7]$

6. 수열에서 큰 값이 작은 값보다 앞서 나오는 경우 두 값을 역전된 쌍이라고 부른다. 예를 들어 수열 $4, 2, 1, 1, 3$에는 $(4, 2), (4, 1), (4, 1), (4, 3), (2, 1), (2, 1)$의 총 6개의 역전된 쌍이 있다. 수열을 입력으로 받아서 역전된 쌍의 개수를 카운트하여 출력하는 프로그램을 작성하라. 키보드로부터 먼저 정수의 개수 N을 입력받고, 이어서 N개의 정수를 입력 받는다.

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
  int n, gncount;
  int* arr;
  scanf("%d", &n);
  arr = (int*)malloc(sizeof(int)*n);
```

```
  for(int i=0; i<n; i++)
    scanf("%d", &arr[i]);
  for(int i=0; i<n; i++){
    int gn = arr[i];
    for(int j=i; j<n; j++){
      if(gn > arr[j])
        gncount++;
    }
  }
  free(arr);
  printf("%d\n", gncount);
}
```

for문을 돌면서 각 성분에 대해 그 성분보다 큰 것들을 세는 단순무식한 코드다.

7. 키보드로부터 연속해서 음이 아닌 정수들을 입력받는다. 정수가 하나씩 입력될 때마다 현재까지 입력된 정수들을 오름차순으로 정렬하여 화면에 출력한다. 단, 새로 입력된 정수가 이미 이전에 입력된 정수라면 duplicate라고 출력하고 저장하지 않고 버린다. 사용자가 -1을 입력하면 프로그램을 종료한다. 입력되는 정수의 개수는 100개를 넘지 않는다.

```
#include <stdio.h>
int check(int* arr, int size, int n);
void insertionSort(int* arr, int size);
int main(){
  int arr[100];
  scanf("%d", &arr[0]);
  printf("%d\n\n", arr[0]);
  for(int i=1; i<100; i++){
    int n;
    scanf("%d", &n);
    if(n==-1)
      break;
    if(check(arr, i, n)){
      arr[i] = n;
      insertionSort(arr, i);
      for(int j=0; j<=i; j++)
        printf("%d ", arr[j]);
    }
    else{
      i--;
      printf("duplicate");
    }
```

```
    printf("\n\n");
  }
}
int check(int* arr, int size, int n){
  int result;
  for(int i=0; i<size; i++){
    if(arr[i]==n){
      result = 0;
      break;
    }
    else
      result = 1;
  }
  return result;
}
void insertionSort(int* arr, int size){
  for(int i=1; i<=size; i++){
    int key = arr[i];
    int j;
    for(j=i-1; j>=0 && arr[j]>key; j--)
      arr[j+1] = arr[j];
    arr[j+1] = key
  }
}
```

8. 먼저 입력될 정수의 개수 $n \leq 100$을 입력받고, 이어서 n개의 정수를 받아 순서대로 배열에 저장한다. 그런 다음 키보드로부터 다시 하나의 정수 k를 입력받은 후 n개의 정수들 중에서 k에 가장 가까운, 즉 k와의 차이의 절대값이 가장 작은 정수를 찾아 출력하는 프로그램을 작성하라.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
  int n, k, diff;
  int* arr;
  scanf("%d", &n);
  arr = (int*)malloc(sizeof(int)*n);
  for(int i=0; i<n; i++)
    scanf("%d", &arr[i]);
  scanf("%d", &k);
  diff = arr[0];
```

```
  for(int i=0; i<n; i++){
    if(abs(k-diff) > abs(k-arr[i]))
      diff = arr[i];
  }
  free(arr);
  printf("%d\n", diff);
}
```

abs 함수를 사용한 단순무식한 방법이다.

9. 사용자로부터 $n < 100$개의 정수를 입력받아 크기순으로 정렬한 후 중복된 수를 제거하는 프로그램을 작성하라. 입력 형식은 먼저 n의 값이 주어지고 이어서 n개의 정수들이 주어진다. 예를 들어 $n = 8$이고 입력된 정수들이 $4, 7, 4, 12, 410, 9, 7$이라면 중복을 제거하고 남은 정수들은 $4, 7, 9, 10, 12$이다. 그러면 먼저 남은 정수의 개수 5를 출력하고 콜론을 출력한 후 남은 정수들을 오름차순으로 출력한다.

```
#include <stdio.h>
#include <stdlib.h>
void insertionSort(int* arr, int size);
int main(){
  int n;
  int count = 0;
  int index = 0;
  int *temp, *arr;
  scanf("%d", &n);
  temp = (int*)malloc(sizeof(int)*n);
  arr = (int*)malloc(sizeof(int)*count);
  for(int i=0; i<n; i++)
    scanf("%d", &temp[i]);
  for(int i=0; i<n; i++){
    int check = 0;
    for(int j=0; j<i; j++){
      if(temp[i]==temp[j]){
        check = 1;
        break;
      }
    }
    if(check == 0){
      count++;
      arr[index] = temp[i];
      index++;
    }
```

```
  }
  free(temp);
  insertionSort(arr, count);
  printf("%d: ", count);
  for(int i=0; i<count; i++)
    printf("%d ", arr[i]);
  printf("\n");
  free(arr);
}
void insertionSort(int* arr, int size){
  for(int i=1; i<size; i++){
    int key = arr[i];
    int j;
    for(j=i-1; j>=0 && arr[j]>key; j--)
      arr[j+1] = arr[j];
    arr[j+1] = key;
  }
}
```

다양한 불변항을 사용했다.

10. 정렬을 하는 가장 간단한 방법 중의 하나는 다음과 같이 하는 것이다. 배열 data에 data[0]에서 data[n-1]까지 n개의 정수가 저장되어 있다. 먼저 data[0]와 data[n-1] 사이의 정수들 중에서 가장 큰 정수를 찾는다. 그것을 data[k]라고 가정해보자. 그러면 data[k]와 data[n-1]을 교환한다. 이제 가장 큰 정수가 data[n-1], 즉 맨 마지막 위치에 저장되었으므로 그 값에 대해서는 더 이상 생각할 필요가 없다. 이제 data[0] - data[n-2] 중에서 최대값을 찾는다. 그 값을 data[p]라고 하자. 그러면 다시 data[p]와 data[n-2]를 교환하고 data[n-2]에 대해서는 잊어버려도 된다. 이런 식으로 계속하면 마지막에는 data[0]과 data[1] 중에 최대값을 data[1]과 교환하면 전체의 정렬이 완료된다. 이 알고리즘을 구현하라. 입력은 먼저 정렬할 개수 $n \leq 100$이 주어지고 이어서 n개의 정수들이 주어진다.

```
#include <stdio.h>
#include <stdlib.h>
void selectionSort(int* arr, int size);
void swap(int* a, int* b);
int main(){
  int n;
  int* data;
  scanf("%d", &n);
  data = (int*)malloc(sizeof(int)*n);
  for(int i=0; i<n; i++)
    scanf("%d", &data[i]);
```

```
  selectionSort(data, n);
  for(int i=0; i<n; i++)
    printf("%d ", data[i]);
  printf("\n");
  free(data);
}
void selectionSort(int* arr, int size){
  int i, j, max;
  for(i=size-1; i>0; i--){
    max = i;
    for(j=i-1; j>=0; j--)
      if(arr[j] > arr[max])
        max=j;
      swap(&arr[max], &arr[i]);
  }
}
void swap(int* a, int* b){
  int temp = *a;
  *a = *b;
  *b = temp;
}
```

선택정렬 알고리즘에 대한 문제다. 배열 성분 자체를 최대값으로 두지 않고 최대값의 성분에 대응하는 인덱스로 정렬한다는 점에 유의하라. 보통 선택 정렬은 최대값이 아니라 최소값을 기준으로 한다. 불변항만 살짝 바꾸면 된다.

Algorithm 2: Selection Sort

Input: A: Array
for $i \in \{1, \ldots, A.length - 1\}$ **do**
  $min \leftarrow i$
  for $j \in \{i+1, A.length - 1$ **do**
    if $A[j] < A[min]$ **then**
      $min \leftarrow j$
  if $min \neq i$ **then**
    swap$(A[i], A[min])$

11. 입력으로 하나의 문자열을 받은 후 뒤집어서 출력하는 프로그램을 작성하라. 예를 들어 `hello`를 입력하면 `olleh`가 출력된다.

```
#include <stdio.h>
#include <string.h>
void revstr(char *str);
int main(){
```

```
  char* str;
  scanf("%s", str);
  revstr(str);
  printf("%s\n", str);
  return 0;
}
void revstr(char *str){
  int len = strlen(str);
  for(int i=0; i<len/2; i++){
    int temp = str[i];
    str[i] = str[len-i-1];
    str[len-i-1] = temp;
  }
}
```

단순하게 swap하여 해결했다.

12. 영문 소문자로 구성된 하나의 문자열을 입력받은 후 문자열을 구성하는 문자들을 알파벳 순으로 정렬하여 만들어지는 문자열을 출력하라. 예를 들어 hello가 입력되면 ehllo를 출력한다.

```
#include <stdio.h>
#include <string.h>
void alphabetOrder();
int main(){
  char *ch;
  scanf("%s", ch);
  alphabetOrder(ch);
  puts(ch);
}
void alphabetOrder(char *ch){
  char temp;
  int i, j, length = strlen(ch);
  for(i=0; i<length; i++){
    for(j=i+1; j<length; j++){
      if(ch[i] > ch[j]){
        temp = ch[i];
        ch[i] = ch[j];
        ch[j] = temp;
      }
    }
  }
```

```
}
```

기본적으로 선택정렬 알고리즘의 응용이다.

13. 아나그램이란 문자들의 순서를 재배열하여 동일하게 만들 수 있는 문자열을 말한다. 대소문자는 구분하지 않는다. 예를 들어서 Silent와 Listen은 아나그램이다. 입력으로 두 문자열을 받아서 아나그램인지 판단하는 프로그램을 작성하라.

```c
#include <stdio.h>
#include <string.h>
void cnvt_lwr(char *str);
int anagram(char *str1, char *str2);
int main(){
  char *words[2];
  for(int i=0; i<2; i++){
    char buf[100];
    scanf("%s", buf);
    words[i] = strdup(buf);
    cnvt_lwr(words[i]);
  }
  if(anagram(words[0], words[1]))
    puts("yes");
  else
    puts("no");
}
void cnvt_lwr(char *str){
  for(int i=0; i<strlen(str); i++){
    if(str[i] >= 65 && str[i] <= 90)
      str[i] = str[i]+32;
  }
}
int anagram(char *str1, char *str2){
  if(strlen(str1)!=strlen(str2))
    return 0;
  int count = 0;
  for(int i=0; i<strlen(str1); i++){
    for(int j=0; j<strlen(str1); j++){
      if(str1[i]==str2[j]){
        count++;
        break;
      }
    }
```

```
  }
  if(count==check)
    return 1;
  else
    return 0;
}
```

main에서 중요한 것은 buf와 strdup을 이용한 입력 이다. anagram은 단순무식한 논리를 break로 구현한 것이고 cnvt_lwr는 ASCII 값을 활용하는 함수이므로 그냥 외워두는 편이 나을 것 같다.

14. 영문 소문자로 구성된 2개의 단어를 입력받은 후 두 단어가 동일한 문자집합으로 구성되었는지 검사하여 yes 혹은 no를 출력하는 프로그램을 작성하라. 예를 들어 ababc와 cba는 문자집합 {a,b,c}로 구성되었으므로 yes다. 입력 단어의 길이는 20이하다.

```
#include <stdio.h>
#include <string.h>
int alphSet(int* wordA, int* wordB);
int main(){
  char* words[2];
  char alphabet[26];
  int firstAlphCount[26], secAlphCount[26];
  for(int i=0; i<2; i++){
    char buf[20];
    scanf("%s", buf);
    words[i] = strdup(buf);
  }
  for(int i=0; i<26; i++){
    alphabet[i] = 'a'+i;
    firstAlphCount[i] = 0;
    secAlphCount[i] = 0;
  }
  for(int i=0;i<strlen(words[0]);i++){
    for(int j=0;j<26;j++){
      if(words[0][i] == alphabet[j])
        firstAlphCount[i] = 1;
    }
  }
  for(int i=0;i<strlen(words[1]);i++){
    for(int j=0;j<26;j++){
      if(words[1][i] == alphabet[j])
        secAlphCount[i] = 1;
    }
  }
```

```
30    if(alphSet(firstAlphCount, secAlphCount))
31      puts("yes");
32    else
33      puts("no");
34  }
35  int alphSet(int* wordA, int* wordB){
36    for(int i=0; i<26; i++){
37      if(wordsA[i] != wordsB[i])
38        return 0;
39    }
40    return 1;
41  }
```

단순무식하지만 뭘 하려는지 잘 보인다. 아무튼 14번 라인인 `alphabet[i] = 'a'+i`가 핵심이다.

15. 입력으로 $n < 100$개의 영문 문자열을 받는다. 각 문자열의 길이는 20이하다. 이 문자열들을 문자열의 길이가 짧은 것부터 긴 것 순서로 정렬하여 출력하라. 단, 길이가 동일한 문자열들은 그들끼리 사전식 순서로 정렬해야 한다. 입력 형식은 먼저 문자열의 개수 n이 주어지고, 이어서 n개의 문자열이 한 줄에 하나씩 주어진다.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4   int main(){
5     int n;
6     char **words;
7     scanf("%d", &n);
8     words = (char **)malloc(sizeof(char[101])*n);
9     for(int i=0; i<n; i++){
10      char buf[101];
11      scanf("%s", buf);
12      words[i] = strdup(buf);
13    }
14    for(int i=0; i<n; i++){
15      for(int j=0; j<n; j++){
16        if(strlen(words[j])>strlen(words[i])){
17          char* temp1;
18          temp1 = words[i];
19          words[i] = words[j];
20          words[j] = temp1;
21        }
22        else if(strlen(words[j])==strlen(words[i])){
```

```
23          if(words[j][0] > words[i][0]){
24            char* temp2;
25            temp2 = words[i];
26            words[i] = words[j];
27            words[j] = temp2;
28          }
29        }
30      }
31    }
32    printf("\n");
33    for(int i=0; i<n; i++)
34      printf("%s\n", words[i]);
35    free(words);
36  }
```

8번 라인의 동적 할당에서 쓰인 이중 포인터를 이해하는 것이 관건이다. 다시 말해 14번 라인 이하의 for 블록에서 swap되는 words[]는 문자가 아니라 문자열 자체다.

16. 입력으로 텍스트 파일 harry.txt를 읽어서 이 파일에 등장하는 모든 단어의 목록을 중복된 단어 없이 사전식 순서로 정렬한다. 이제 새로운 단어 하나를 키보드로부터 입력 받는다. 저장된 단어들 중에서 이 새로운 단어를 접두어(prefix)로 하는 모든 단어를 찾아서 한 줄에 하나씩 출력하는 프로그램을 작성하라. 마지막으로 찾아진 단어의 개수를 출력하라. 어떤 단어가 다른 단어의 접두어인지는 표준 라이브러리가 제공하는 strncmp 함수를 이용하여 검사할 수 있다. 이 함수의 사용 방법은 검색하여 알아보라.

```
1  #include <stdio.h>
2  #include <string.h>
3  int prefix(char *pre, char *words);
4  int main(){
5    char *words[100000];
6    char buf[100];
7    int n = 0;
8    FILE *fp = fopen("harry.txt", "r");
9    while(fscanf(fp, "%s", buf)!=EOF){
10     int i = 0;
11     for(;i<n;i++){
12       if(strcmp(buf,words[i])==0)
13         break;
14     }
15     if(i==n)
16       words[n++] = strdup(buf);
17   }
```

```
  fclose(fp);
  for(int i=0; words[i]; i++){
    for(int j=0; words[j]; j++){
      if(strcmp(words[i], words[j])<0){
        char *tmp = words[i];
        words[i] = words[j];
        words[j] = tmp;
      }
    }
  }
  scanf("%s", buf);
  printf("\n");
  int count = 0;
  for(int i=0; words[i]; i++){
    if(prefix(buf, words[i])){
      printf("%s\n", words[i]);
      count++;
    }
  }
  printf("%d\n", count);
  return 0;
}
int prefix(char *pre, char *words){
  return strncmp(pre, words, strlen(pre))==0;
}
```

코드가 꽤 복잡하다. 우선

```
char *words[100000];
```

은 문자열(`char *`) 100000개로 구성된 배열을 만든다.

```
while(fscanf(fp, "%s", buf)!=EOF)
```

는 입력된 파일이 파일의 끝(`EOF`)에 이를 때까지 문자열을 한 줄 씩 읽는다. 이 `while`문 전체를 보자.

```
while(fscanf(fp, "%s", buf)!=EOF){
  int i = 0;
  for(; i<n; i++){
    if(strcmp(buf, words[i])==0)
      break;
  }
  if(i==n)
    words[n++] = strdup(buf);
}
```

앞서 선언한 문자열 배열 words의 인덱스는 n으로, 버퍼에서 한 번씩 복제(strdup)할 때마다 n이 증가한다. 또한 문자열을 한 줄씩 읽는 while문의 인덱스는 i인데, while문 내부의 for문도 i를 사용한다. 문자열 배열상의 문자열 개수 n번 동안 버퍼와 i번째 문자열 배열을 비교(strcmp)하여 일치하면 for문에서 break한다. 이때 i는 for문 외부에서 선언되었기에 증가된 i 혹은 중복이 발생한 i의 값이 유지된다. 이때 i와 n이 일치하면 문자열 배열에 버퍼의 문자열을 복제하고 n을 증가하는 것이다. 좀 꼬였는데, 중복 없이 입력하기 위한 것이다.

prefix 함수에 사용된 strncmp는 길이를 지정하여 strcmp하는 함수다.

17. 입력으로 텍스트 파일 harry.txt를 읽는다. 이 텍스트 파일은 오직 영문 소문자만으로 구성되어 있다. 이 파일에 등장하는 길이가 6이상인 모든 단어의 목록과 각 단어의 등장 빈도를 구하여 화면으로 출력하는 프로그램을 작성하라. 단어들은 사전식 순서로 정렬되어 출력되어야 한다. 출력 파일의 각 줄에 하나의 단어와 그 단어의 등장 빈도를 콜론으로 구분하여 출력하라. 동일한 단어가 중복해서 출력되어서는 안 된다. 출력의 마지막에는 전체 단어의 개수를 출력하라. 아래는 올바른 출력의 시작 부분과 끝 부분을 보여준다.

```c
#include <stdio.h>
#include <string.h>
int wordCount(char* file, char* word);
int main(){
  char *words[100000];
  int count, n = 0;
  FILE *fp = fopen("harry.txt", "r");
  char buf[100];
  while(fscanf(fp, "%s", buf)!=EOF){
    if(strlen(buf)>=6){
      int i = 0;
      for(;i<n;i++){
        if(strcmp(buf,words[i])==0)
          break;
      }
      if(i==n)
        words[n++] = strdup(buf);
    }
  }
  fclose(fp);
  for(int i=0; words[i]; i++){
    for(int j=0; words[j]; j++){
      if(strcmp(words[i],words[j])<0){
        char *temp = words[i];
        words[i] = words[j];
        words[j] = temp;
```

```
      }
    }
  }
  for(count=0; count<n; count++)
    printf("%s: %d\n", words[count], wordCount("harry.txt",words[
   count]));
  printf("%d\n", count);
}
int wordCount(char *file, char *word){
  int wc = 0;
  FILE *txt = fopen(file, "r");
  char buf[100];
  while((fgets(buf, 100, txt))!=NULL){
    if((strstr(buf, word))!=NULL)
      wc++;
  }
  fclose(txt);
  return wc;
}
```

while문은 앞선 문제와 일치한다. wordCount에서 쓴 strstr 함수는 file에 word가 나타나지 않으면 NULL을 반환한다. 즉 file에 word가 이미 있으면 wc를 하나 증가시키는 함수가 wordCount다.

19. 하나의 영문 소문자로 된 문자열이 입력으로 주어진다. 이 문자열에서 자음이 가장 여러 번 연속해서 등장하는 부분을 찾아서 그 부분을 출력하는 프로그램을 작성하라. 예를 들어 문자열 "nietzsche"에서는 "tzsch"가 가장 긴 연속된 자음이다. 입력 문자열의 길이는 100이하이고, 'a', 'e', 'i', 'o', 'u'를 제외한 모든 알파벳은 자음으로 간주한다.

```
#include <stdio.h>
#include <string.h>
int isVowel(char c);
int main(){
  char input[101];
  scafn("%s", input);
  int maxConVow = 0;
  int curConVow = 0;
  int index = 0;
  for(int i=0; i<strlen(input); i++){
    if(isVowel(input[i]))
      curConVow++;
    else{
      if(curConVow > maxConVow){
```

```
15          maxConVow = curConVow;
16          index = i- maxConVow;
17        }
18        curConVow = 0;
19      }
20    }
21    if(curConVow > maxConVow){
22      maxConVow = curConVow;
23      index = length - maxConVow;
24    }
25    if(maxConVow > 0){
26      for(int i=index; i<index+maxConVow; i++)
27        printf("%c", input[i]);
28      printf("\n");
29    }
30  }
31  int isVowel(char c){
32    return(c != 'a' && c!= 'e' && c != 'i' && c != 'o' && c != 'u');
33  }
```

제 2 장

문자열

1. 하나의 구간(interval)은 시작점과 끝점의 좌표로 정의된다. 구간의 시작점과 끝점은 정수이고, 끝점은 항상 시작점보다 크거나 같다. 먼저 구간의 개수 $n < 100$이 주어지고, 이어서 n개의 구간이 입력으로 주어진다. 그런 다음 다시 추가로 하나의 구간이 주어진다. 추가로 주어진 구간에 완전히 포함되면서 가장 긴 구간을 찾아 그 길이를 출력하는 프로그램을 작성하라.

```c
#include <stdio.h>
#include <stdlib.h>
int eval(int (*arr)[2], int (*arr)[2], int size);
int main(){
  int n;
  scanf("%d", &n);
  int (*interval)[2] = malloc(sizeof(int[n][2]));
  for(int i=0; i<n; i++){
    for(int j=0; j<2; j++)
      scanf("%d", &interval[i][j]);
  }
  int newval[1][2];
  for(int i=0; i<2; i++)
    scanf("%d", &newval[0][i]);
  printf("%d\n", eval(interval, newval, n));
  free(interval);
}
int eval(int (*arr1)[2], int (*arr2)[2], int size){
  int j=0;
  int length[100];
  for(int i=0; i<size; i++){
    if(arr2[0][0] <= arr1[i][0] && arr2[0][1] >= arr1[i][1]){
      length[j] = arr1[i][1]-arr1[i][0];
      j++;
    }
  }
  int maxLength = 0;
  for(int i=0; i<j; i++){
    if(maxLength <= length[i])
```

```
30        maxLength = length[i];
31    }
32    return maxLength;
33 }
```

우선 `int (*arr)[2]`은 정수형 포인터 배열을 나타낸다. `int (*interval)[2]`를 선언한 다음, `int[n][2]`의 크기로 `malloc`한다. 즉 2차원 배열이다.

2. 입력으로 $n < 100$개의 구간이 주어진다. 각 구간은 구간의 시작점과 끝점으로 표현된다. 각 구간의 시작점과 끝점은 정수이고, 끝점은 항상 시작점보다 크거나 같다. 이 구간들을 시작점이 빠른 순서대로 정렬하여 출력하는 프로그램을 작성하라. 시작점이 같은 경우 끝점이 빠른 것을 먼저 출력한다. 입력 형식은 먼저 n의 값이 주어지고, 이어서 각 구간의 시작점과 끝점이 차례대로 주어진다.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  void psuedoInsertionSort(int *arr1, int *arr2, int size);
4  int main(){
5    int n;
6    scanf("%d", &n);
7    int *first = (int*)malloc(sizeof(int)*n);
8    int *second = (int*)malloc(sizeof(int)*n);
9    for(int i=0; i<n; i++){
10     int num1, num2;
11     scanf("%d", &num1);
12     scanf("%d", &num2);
13     first[i] = num1;
14     second[i] = num2;
15   }
16   psuedoInsertionSort(first, second, n);
17   printf("\n");
18   for(int i=0; i<n; i++)
19     printf("%d %d\n", first[i], second[i]);
20   free(first);
21   free(second);
22 }
23 void psuedoInsertionSort(int *arr1, int *arr2, int size){
24   int i, j, key1, key2;
25   for(i=1; i<size; i++){
26     key1 = arr1[i];
27     key2 = arr2[i];
```

```
    for(j=i-1; j>=0 && (arr1[j]>key1 || (arr1[j]==key1&&arr2[j]>
   key2)); j--){
      arr1[j+1] = arr1[j];
      arr2[j+1] = arr2[j];
    }
    arr1[j+1] = key1;
    arr2[j+1] = key2;
  }
}
```

앞선 문제와 다르게 이차원 배열을 그냥 배열 두 개로 구현할 수도 있다.

3. 키보드로부터 여러 개의 단어로 이루어진 한 라인의 텍스트를 입력받은 후 단어와 단어 사이에 있는 하나의 공백 문자를 제외한 모든 공백 문자를 제거하고 출력하는 프로그램을 작성하라. 또한 마지막에 출력된 문자열의 길이를 출력한다.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>

int main(){
  char in[1000];
  fgets(in, sizeof(in), stdin);
  char *out;
  out = (char*)malloc(sizeof(char)*length);
  int index = 0;
  int space = 1;
  for(int i=0; i<strlen(in); i++){
    if(isspace(in[i])){
      if(!space){
        out[index++] = ' ';
        space = 1;
      }
    }else{
      out[index++] = in[i];
      space = 0;
    }
  }
  if(index > 0 && out[index-1] == ' ')
    index--;
  out[index] = '\0';
```

```
  printf("%s:%d\n", out, strlen(out));
  free(out);
}
```

scanf가 아니라 fgets를 사용하는 이유는 공백 문자를 포함하여 입력하기 위한 것이다. 또한 isspace 함수는 ctype.h 헤더에 포함되어 있으며, 어떤 문자가 공백 문자인지 판별한다.

4. 사전 파일 shuffled_dict.txt을 읽는다. 이 파일에는 각 줄마다 하나의 "단어"그 단어에 대한 "설명"이 저장되어 있다. "단어"와 그 단어에 대한 "설명"은 하나의 탭 문자로 구분되어 있다. "단어"는 하나의 영문 소문자 문자열이며, "설명"은 여러 개의 단어로 구성된 문장이다. 이 사전 파일에서 단어들은 사전식 순서로 정렬되어 있지 않다. 이 파일을 읽은 후 단어들을 사전식 순서로 정렬하여 "sorted_dict.txt"라는 이름의 새로운 파일을 생성하는 프로그램을 작성하라. 저장된 파일에서는 한 줄에 하나의 단어와 그 단어에 대한 설명을 탭 문자로 구분하여 저장해야 한다.

```
#include <stdio.h>
#include <string.h>
int main(){
  char *words[100000];
  char buf[1000];
  int n = 0;
  FILE *txt = fopen("shuffled_dict.txt", "r");
  FILE *ntxt = fopen("sorted_dict.txt", "w");
  while(fgets(buf, 1000, txt)!=NULL)
    words[n++] = strdup(buf);
  for(int i=0; words[i]; i++){
    for(int j=0; words[j]; j++){
      if(strcmp(words[i],words[j])<0){
        char *tmp = words[i];
        words[i] = words[j];
        words[j] = tmp;
      }
    }
  }
  fclose(txt);
  for(int i=0; i<n; i++){
    char *r = strchr(words[i], '\t');
    *r = '\t';
    fprintf(ntxt, "%s\n", words[i]);
  }
  fclose(ntxt);
}
```

`strchr` 함수는 문자열에서 특정 문자를 찾은 다음 그 주소를 리턴한다. 사실 `shuffled_dict.txt` 파일 자체가 탭 문자로 구분되어 있기에 정렬 말고는 할 일이 없다.

5. `table.txt`를 읽어서 `output.txt` 파일에 테이블 형태로 출력하는 프로그램을 작성하라. 입력 파일의 첫 줄에는 테이블 행의 개수 $m \leq 10$과 열의 개수 $n \leq 10$이 주어진다. 이어진 m 줄에는 각 줄마다 테이블의 한 행에 들어갈 n개의 내용이 문자 `&`로 구분되어 주어진다. 테이블의 어떤 칸은 빈 칸일 수도 있다는 것을 유념하라. 빈 칸의 경우에도 하나 이상의 공백 문자로 표현되어 있다. 불필요한 공백 문자들을 제거하여 최대한 깔끔하게 출력되도록 하라.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_ROWS 100
#define MAX_COLS 100

int main(){
  int rows, cols;
  int width[MAX_COLS] = {0};
  char buf[256];
  char table[MAX_ROWS][MAX_COLS][256];
  FILE *input, *output;
  input = fopen("table.txt", "r");
  output = fopen("output.txt", "w");
  if(fgets(buf, sizeof(buf), input)!=NULL)
    sscanf(buf, "%d %d", &rows, &cols);
  for(int row=0; row<rows; row++){
    if(fgets(buf, sizeof(buf), input)==NULL)
      break;
    char *token = strtok(buf, "&");
    int col = 0;
    while(token != NULL && col < cols){
      strcpy(table[row][col], token);
      int tokLen = strlen(table[row][col]);
      if(tokLen > width[col])
        width[col] = tokLen;
      token = strtok(NULL, "&");
      col++;
    }
  }
  for(int row=0; row<rows; row++){
```

```
    for(int col=0; col<cols; col++){
      fprintf(output, "%-*s", width[col], table[row][col]);
      if(col <cols-1)
        fprintf(output, " ");
    }
    fprintf(output, "\n");
  }
  fclose(input);
  fclose(output);
}
```

strtok의 용례(https://blockdmask.tistory.com/382)와 더불어 fprintf의 용례(https://jhnyang.tistory.com/314)를 이해하면 수월한 문제다.

6. 프로그램을 시작하면 먼저 텍스트 파일 data.mat을 읽는다. 이 파일의 첫 줄에는 양의 정수 $N \leq 100$이 주어지고, 이어진 N 줄에는 각 줄마다 N개의 정수가 주어진다. 즉 하나의 $N \times N$ 정수 행렬이 주어진다. 그런 다음 화면에 프롬프트를 출력하고 일련의 사용자 명령을 처리하는 프로그램을 작성하라.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 100
#define BUF_SIZE 256

void show_mat(int matrix[MAX_SIZE][MAX_SIZE], int size);
void colmax_mat(int matrix[MAX_SIZE][MAX_SIZE], int size);
void colmin_mat(int matrix[MAX_SIZE][MAX_SIZE], int size);
void rowmax_mat(int matrix[MAX_SIZE][MAX_SIZE], int size);
void rowmin_mat(int matrix[MAX_SIZE][MAX_SIZE], int size);
void slice_mat(int matrix[MAX_SIZE][MAX_SIZE], int size, int x, int
    p, int y, int q);

int main(){
  const char* exit = "exit";
  const char* show = "show";
  const char* colmax = "colmax";
  const char* colmin = "colmin";
  const char* rowmax = "rowmax";
  const char* rowmin = "rowmin";
  const char* slice = "slice";
```

```
23  char matrix[MAX_SIZE][MAX_SIZE][BUF_SIZE];
24  char buf[BUF_SIZE];
25  int n;
26  int width[MAX_SIZE] = {0};
27  int numMat[MAX_SIZE][MAX_SIZE];
28  FILE *input;
29  input = fopen("data.mat", "r");
30  if(fgets(buf, sizeof(buf), input)!=NULL)
31    sscanf(buf, "%d", &n);
32  for(int row=0; row<n; row++){
33    if(fgets(buf, sizeof(buf),input)==NULL)
34      break;
35    char *token = strtok(buf, "\t");
36    int col = 0;
37    while(token != NULL && col<n){
38      strcpy(matrix[row][col], token);
39      int tokLen = strlen(matrix[row][col]);
40      if(tokLen > width[col])
41        width[col] = tokLen;
42      token = strtok(NULL, "\t");
43      col++;
44    }
45    char* temp = strdup(matrix[row][0]);
46    token = strtok(temp, " ");
47    col = 0;
48    while(token != NULL){
49      int number = atoi(token);
50      numMat[row][col]=number;
51      token = strtok(NULL, " ");
52      col++;
53    }
54  }
55  while(1){
56    char arg[10];
57    printf("\n$ ");
58    scanf("%s", arg);
59    if(strcmp(arg, exit)==0)
60      break;
61    else if(strcmp(arg, show)==0)
62      show_mat(numMat, n);
63    else if(strcmp(arg, colmax)==0)
```

```
64        colmax_mat(numMat, n);
65      else if(strcmp(arg, colmin)==0)
66        colmin_mat(numMat, n);
67      else if(strcmp(arg, rowmax)==0)
68        rowmax_mat(numMat, n);
69      else if(strcmp(arg, rowmin)==0)
70        rowmin_mat(numMat, n);
71      else if(strcmp(arg, slice)==0){
72        int x, p, y, q;
73        scanf("%d %d %d %d", &x, &p, &y, &q);
74        slice_mat(numMat, n, x, p, y, q);
75      }
76    }
77    fclose(input);
78  }
79  void show_mat(int matrix[MAX_SIZE][MAX_SIZE], int size){
80    for(int row=0; row<size; row++){
81      for(int col=0; col<size; col++)
82        printf("%d\t", matrix[row][col]);
83      printf("\n");
84    }
85  }
86  void colmax_mat(int matrix[MAX_SIZE][MAX_SIZE], int size){
87    int max[MAX_SIZE];
88    for(int i=0;i<size;i++)
89      max[i] = matrix[0][i];
90    for(int row=0;row<size;row++){
91      for(int col=0;col<size;col++){
92        if(max[col]< matrix[row][col])
93          max[col] = matrix[row][col];
94      }
95    }
96    for(int row=0;row<size;row++)
97      printf("%d\t", max[row]);
98    printf("\n");
99  }
100 void colmin_mat(int matrix[MAX_SIZE][MAX_SIZE], int size){
101   int min[MAX_SIZE];
102   for(int i=0;i<size;i++)
103     min[i] = matrix[0][i];
104   for(int row=0;row<size;row++){
```

```
105     for(int col=0;col<size;col++){
106       if(min[col]>matrix[row][col])
107         min[col] = matrix[row][col];
108     }
109   }
110   for(int row=0;row<size;row++)
111     printf("%d\t", min[row]);
112   printf("\n");
113 }
114 void rowmax_mat(int matrix[MAX_SIZE][MAX_SIZE], int size){
115   int max[MAX_SIZE];
116   for(int i=0;i<size;i++)
117     max[i] = matrix[i][0];
118   for(int row=0;row<size;row++){
119     for(int col=0;col<size;col++){
120       if(max[col]<matrix[col][row])
121         max[col] = matrix[col][row];
122     }
123   }
124   for(int row=0;row<size;row++)
125     printf("%d\t", max[row]);
126   printf("\n");
127 }
128 void rowmin_mat(int matrix[MAX_SIZE][MAX_SIZE], int size){
129   int min[MAX_SIZE];
130   for(int i=0;i<size;i++)
131     min[i] = matrix[i][0];
132   for(int row=0;row<size;row++){
133     for(int col=0;col<size;col++){
134       if(min[col]>matrix[col][row])
135         min[col] = matrix[col][row];
136     }
137   }
138   for(int row=0;row<size;row++)
139     printf("%d\t", min[row]);
140   printf("\n");
141 }
142 void slice_mat(int matrix[MAX_SIZE][MAX_SIZE], int size, int x, int
      p, int y, int q){
143   for(int row=x; row<size; row=row+p){
144     for(int col=y; col<size; col=col+q)
```

```
145         printf("%d\t", matrix[row][col]);
146     printf("\n");
147     }
148 }
```

그냥 엄청 귀찮은 문제다.

7. 강아지가 $N \times N$ 크기의 2차원 배열의 가운데 위치에서 출발한다. N은 홀수다. 상, 하, 좌, 우 4방향으로 인접한 셀들 중에서 방문한 적이 없는 한 셀을 동일한 확률로 랜덤하게 선택하여 한 칸 이동한다. 배열의 가장자리 셀에 도착하면 탈출에 성공한 것이다. 하지만 아무 곳으로도 이동할 수 없는 상태에 처하면 탈출에 실패한 것이다. 입력으로 하나의 홀수 $N \leq 100$을 받아서 강아지가 탈출에 성공할 확률을 시뮬레이션으로 계산하는 프로그램을 작성하라. 실험 횟수는 10,000번으로 하라.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  int escape(int N);
5  int main() {
6    int N;
7    scanf("%d", &N);
8    int success = escape(N);
9    double prob = (double)success/10000.0;
10   printf("%.3lf\n", prob);
11   return 0;
12 }
13 int escape(int N) {
14   int count = 0;
15   int x, y;
16   srand(time(NULL));
17   for (int i = 0; i < 10000; i++) {
18     x = N/2;
19     y = N/2;
20     int visited[N][N];
21     for(int row = 0; row < N; row++){
22       for(int col = 0; col < N; col++)
23         visited[row][col] = 0;
24     }
25     while (x > 0 && x < N-1 && y > 0 && y < N-1){
26       visited[x][y] = 1;
27       int dir[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
28       int possDir[4][2];
```

```
      int possDirCnt = 0;
      for (int i=0; i<4; i++) {
        int nX = x + dir[i][0];
        int nY = y + dir[i][1];
          if (!visited[nX][nY]){
            possDir[possDirCnt][0] = nX;
            possDir[possDirCnt][1] = nY;
            possDirCnt++;
          }
        }
      if(possDirCnt == 0)
        break;
      int random = rand() % possDirCnt;
        x = possDir[random][0];
        y = possDir[random][1];
    }
    if(x == 0 || x == N - 1 || y == 0 || y == N-1)
      count++;
  }
  return count;
}
```

time을 통한 유사난수의 구현과 다중 조건 문장이 핵심이다.

8. 입력 파일 board.txt에 오목판의 상태가 주어진다. 파일의 첫 줄에는 바둑판의 크기 $N \leq 19$가 주어지고, 이어진 N줄에는 각 줄마다 N개의 정수 0, 1 혹은 2가 주어진다. 0은 빈자리를 표시하고 1은 검은 돌, 2는 흰 돌을 표시한다. 주어진 상태가 검은 돌이 이긴 상태인지, 흰 돌이 이긴 상태인지, 혹은 아직 아무도 못 이긴 상태인지 검사하여 Black, White, 혹은 Not Finished라고 출력하는 프로그램을 작성하라. 둘 다 이긴 상태는 없다고 가정한다.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_N 19
int check(int board[MAX_N][MAX_N], int N);
int main(){
  int N;
  FILE *inputFile;
  inputFile = fopen("board.txt", "r");
  fscanf(inputFile, "%d", &N);
  int board[MAX_N][MAX_N];
  for(int i=0; i<N; i++) {
    for(int j=0; j<N; j++)
```

```
13        fscanf(inputFile, "%d", &board[i][j]);
14    }
15    int result = check(board, N);
16    if(result == 1)
17      printf("Black\n");
18    else if(result == 2)
19      printf("White\n");
20    else
21      printf("Not Finished\n");
22    fclose(inputFile);
23    return 0;
24  }
25
26  int check(int board[MAX_N][MAX_N], int N){
27    int i, j;
28    for (i=0; i<N; i++){
29      for (j=0; j<=N-5; j++){
30        if (board[i][j] == 1 && board[i][j + 1] == 1 && board[i][j +
      2] == 1 && board[i][j + 3] == 1 && board[i][j + 4] == 1)
31          return 1;
32        if (board[i][j] == 2 && board[i][j + 1] == 2 && board[i][j +
      2] == 2 && board[i][j + 3] == 2 && board[i][j + 4] == 2)
33          return 2;
34      }
35    }
36    for (i=0; i<=N-5; i++){
37      for (j=0; j<N; j++){
38        if (board[i][j] == 1 && board[i + 1][j] == 1 && board[i + 2][
      j] == 1 && board[i + 3][j] == 1 && board[i + 4][j] == 1)
39          return 1;
40        if (board[i][j] == 2 && board[i + 1][j] == 2 && board[i + 2][
      j] == 2 && board[i + 3][j] == 2 && board[i + 4][j] == 2)
41          return 2;
42      }
43    }
44    for(i=0; i<=N-5; i++){
45      for(j=0; j<=N-5; j++){
46        if (board[i][j] == 1 && board[i + 1][j + 1] == 1 && board[i +
      2][j + 2] == 1 && board[i + 3][j + 3] == 1 && board[i + 4][j +
      4] == 1)
47          return 1;
```

```
48        if (board[i][j] == 2 && board[i + 1][j + 1] == 2 && board[i +
      2][j + 2] == 2 && board[i + 3][j + 3] == 2 && board[i + 4][j +
     4] == 2)
49          return 2;
50      }
51    }
52    for(i= 0; i<=N-5; i++){
53      for (j=N-1; j>=4; j--){
54        if (board[i][j] == 1 && board[i + 1][j - 1] == 1 && board[i +
      2][j - 2] == 1 && board[i + 3][j - 3] == 1 && board[i + 4][j -
     4] == 1)
55          return 1;
56        if (board[i][j] == 2 && board[i + 1][j - 1] == 2 && board[i +
      2][j - 2] == 2 && board[i + 3][j - 3] == 2 && board[i + 4][j -
     4] == 2)
57          return 2;
58        }
59    }
60    return 0;
61 }
```

다중 조건문을 활용하면 되는 문제다.

9. `sample.html` 파일을 읽어서 파일에 등장하는 모든 HTML 태그들을 제거한 후 `sample.txt` 라는 이름의 파일로 저장하는 프로그램을 작성하라. 단 HTML 파일에서 태그를 제거하는 것 말고 다른 부분은 그대로 유지되어야 한다.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4
5  int main() {
6    FILE *input = fopen("sample.html", "r");
7    FILE *output = fopen("sample.txt", "w");
8    char c;
9    bool in = false;
10   while((c = fgetc(input)) != EOF){
11     if (c == '<')
12       in = true;
13     else if (c == '>')
14       in = false;
15     else if (!in)
```

```
        fputc(c, output);
    }
    fclose(input);
    fclose(output);
    return 0;
}
```

10. 입력으로 텍스트파일 input.txt를 읽어서 왼쪽 정렬하여 output.txt 파일로 출력하는 프로그램을 작성하라. 출력 파일의 한 줄은 80 문자를 초과해서는 안되며, 단어를 자르지 않는 한도 내에서 가능한 한 최대한 80 문자에 가깝도록 맞춘다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 80
void align(FILE *input, FILE *output);

int main(){
    FILE *input = fopen("input.txt", "r");
    FILE *output = fopen("output.txt", "w");
    align(input, output);
    fclose(input);
    fclose(output);
    return 0;
}
void align(FILE *input, FILE *output){
    char line[MAX+1];
    int currLength = 0;
    while(fgets(line, sizeof(line), input) != NULL){
        int length = strlen(line);
        if(length > 0 && line[length-1] == '\n'){
            line[length-1] = '\0';
            length--;
        }
        int i = 0;
        while(i<length){
            int rem = MAX - currLength;
            if (rem >= length - i){
                fprintf(output, "%s", &line[i]);
                currLength += length-i;
```

```
30          break;
31        } else{
32        fprintf(output, "%.*s\n", rem, &line[i]);
33        currLength = 0;
34        i += rem;
35        }
36      }
37    }
38  }
```

제 3 장

연결리스트 *1*

리스트란 순서 관계가 존재하는 집합이다. 기본 연산으로는 삽입(insert), 삭제(remove), 검색(search) 등이 있으며 리스트를 구현하는 대표적인 두 가지 방법으로는 배열과 연결리스트가 존재한다. 배열의 단점은 크기가 고정된다는 사실이다. 따라서 reallocation이 필요하다. 또한 리스트 중간에 원소를 삽입하거나 삭제할 경우 다수의 데이터를 옮겨야 한다. 반면 연결리스트는 다른 데이터의 이동 없이 중간에 삽입이나 삭제가 가능하며, 길이의 제한이 없다. 하지만 랜덤 액세스가 불가능하다. 다시 말해 임의의 n번째 entry에 접근할 때 배열과 다르게 추가적인 시간이 요구된다.

이는 연결리스트가 노드를 사용하기 때문이다. 각각의 노드는 데이터 필드와 하나 이상의 링크 필드로 구성되며 링크 필드에는 다음 노드의 주소가 저장된다. 참고로 첫 번째 노드의 주소 head는 따로 저장해야 하며 마지막 노드의 next 필드에는 NULL을 저장하여 연결리스트의 끝을 표시해야 한다.

```
struct node {
  char *data;
  struct node *next;  // self reference
}
typedef struct node Node;
Node *head = NULL;    // first node pointer
```

연결 리스트에서 하나의 노드를 표현하기 위한 구조체다.

```
int main(){
  head = (Node*)malloc(sizeof(Node));
  head->data = "Tuesday";
  heda->next = NULL;

  Node *q = (Node*)malloc(sizeof(Node));
  q->data = "Thursday";
  q->next = NULL;
  head->next = q;

  q = (Node*)malloc(sizeof(Node));
  q->data = "Monday";
  q->next = head;
  head = q;

  Node *p - head;
  while(p!=NULL){
```

```
18      printf("%s\n", p->data);
19      p = p->next;
20    }
21  }
```