

1. 문학적 프로그래밍. 컴퓨터 프로그래머로서 T_EX에 관심있는 분이라면, “문학적 프로그래밍 (Literate programming)”¹이라는 말을 한 번쯤은 들어봤을 것입니다. T_EX 역시 문학적 프로그래밍의 한 도구인 WEB이라는 언어로 구현된 프로그램입니다. WEB은 조판 언어인 T_EX과 교육용 프로그래밍 언어로 알려진 Pascal 프로그래밍 언어로 이루어져 있습니다. 문학적 프로그래밍은 T_EX의 저자이기도 한 Knuth가 제창한 프로그래밍의 새로운 패러다임입니다. 다음은 Knuth가 한 말로, 문학적 프로그래밍의 기본 배경이 되는 개념이라고 할 수 있습니다.

I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title: “Literate Programming.”

*Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.*²

더불어 Knuth은 *Mathematical Writing*이라는 책에서 다음과 같은 재미있는 말을 했습니다.

Don says that a computer program is a piece of literature. (“I look forward to the day when a Pulitzer Prize will be given for the best computer program of the year.”)

심지어는 그의 홈페이지³에서는 아래와 같은 겁을 주기도 합니다.

“If you are in the software industry and do not use CWEB but your competitors do, your competitors will soon overtake you—and you’ll miss out on a lot of fun besides.”

이 프로그램은 T_EX과 가장 많이 사용되는 프로그래밍 언어 중의 하나인 C 프로그래밍 언어가 조합된 CWEB란 언어를 소개하기 위해 작성된 간단한 프로그램입니다. 이 프로그램은 CWEB의 소개에 목적을 두고 만들어진 프로그램이라, 프로그램의 완성도는 높지 않은 편입니다. 심지어는 입력에 대한 검사도 하지 않습니다. 너그려이 봐 주시기를 바랍니다. 하지만 매우 흥미로운 알고리즘을 구현한 것이기도 합니다. 또한 이 프로그램은 CWEB 프로그래밍이 얼마나 재미있고 흥분되는 것인가 하는 것을 소개하기 위한 것 이기도 합니다.

¹ <http://faq.ktug.or.kr/faq/LiterateProgramming>

² <http://www.literateprogramming.com>

³ <http://www-cs-faculty.stanford.edu/~knuth/cweb.html>

2. 배열 회전. 컴퓨터 프로그래머라면 한 권쯤 가지고 있을 법한 Jon Bentley가 저술한 “*Programming Pearls*”이란 책이 있습니다. 이 책의 두 번째 컬럼은 효율적인 알고리즘의 중요성과 그 알고리즘 개발의 흥미를 설명한 *Aha! Algorithms*이란 제목의 컬럼입니다. Jon은 여기서 세가지 재미있는 문제를 제시하는데, 여기서는 그 세 가지중 두 번째 문제인 배열의 회전에 관한 문제를 CWEB을 이용해서 구현해 보겠습니다. 문제는 다음과 같습니다.

“크기가 n 인 일차원 배열을 왼쪽으로 i 만큼 회전 이동 하라.”

예를 들면, $n = 8, i = 3$ 으로 주어졌을 때, 배열 *abcdefgh*는 *defghabc*로 변환됩니다. 잠깐 생각하여 떠오르는 방법은 배열 x 의 처음 i 개를 임시 장소에 저장하고, 나머지 $n-i$ 개의 원소들을 왼쪽으로 i 만큼 옮긴 다음에, 임시 장소에 저장되어 있던 i 개의 원소들을 뒤에 붙이는 방법입니다. 이 방법은 i 개 만큼의 원소들을 저장 할 수 있는 임시 장소가 필요한데, i 의 값이 크다면, 메모리 낭비가 심하다는 단점이 있습니다. 그래서 메모리를 절약하는 방법으로 생각 할 수 있는 방법이 일단 1개의 원소를 왼쪽으로 회전이동 하는 함수를 구현하고, 이 함수를 i 번 호출하는 것입니다. 이 방법은 메모리는 절약되지만, 실행시간이 너무 길어진다는 단점이 있습니다.

여러분은 최소한의 임시 공간을 사용하며 n 에 비례하는 실행 시간만을 이용해서 해결 할 수 있습니까? 즉, 이 문제에는 시간과 공간의 제약 사항이 있는데, 이 제약 사항을 좀 유식하게 표현하면,

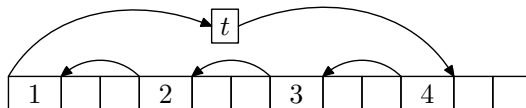
$O(n)$ 시간, $O(1)$ 추가 공간

쯤 됩니다. 물론, 이러한 제약 사항 아래서 문제를 해결하려면, 해결 방법이 복잡해 지는 것은 당연합니다. Jon은 이 문제에 대해서 세 가지 매우 흥미로운 알고리즘을 제시합니다. 여기서는 그 세가지 각각의 알고리즘을 알아보고, 구현해 보겠습니다.

이 프로그램의 전체적인 구조는 다음과 같습니다.

```
#include <stdio.h>
#include <stdlib.h>    /* atoi() */
< 필요한 서브루틴들 4 >
< 세 가지 Vector rotation 알고리즘 3 >
main(argc, argv)
    int argc;
    char *argv[];
{
    int i, n, d, which;
    char *vector;    /* 프로그램에서 사용되는 일차원 배열입니다 */
    < Command line parameter 설정 14 >;
    < 일차원 배열 생성 및 초기화 15 >;
    < 주어진 알고리즘 실행 16 >;
    free(vector);
    return 0;
}
```

3. Juggling 알고리즘. Jon이 소개한 세 가지 알고리즘 중 첫번째는 저글링 알고리즘이라 불리는 방법입니다. 먼저 $x[0]$ 에 있는 원소를 임의의 공간 t 로 옮깁니다. 그리고 나서 $x[i]$ 는 $x[0]$ 로, $x[2i]$ 는 $x[i]$ 로 옮깁니다. 이때 옮기는 모든 인덱스에 대해서 $\text{mod } n$ 연산을 취합니다. 이유는 회전 이동이 모듈라 연산과 관계가 있기 때문입니다. 이러한 식으로 계속 원소들을 옮기다가 $x[0]$ 위치의 원소를 옮길 경우가 발생하면, $x[0]$ 의 원소는 이미 $x[i]$ 에서 옮겨왔기 때문에 또 옮길 필요가 없고, 임시 저장 장소 t 에 있는 원소를 $x[0]$ 를 옮길 위치로 옮기고 여기서 이 과정을 여기서 멈춥니다. i 가 3이고 n 이 12일때, 앞에 설명한 대로 옮기는 과정은 아래 그림과 같습니다.



위 과정에서 모든 원소들이 이동되지 않았으면, $x[1]$ 을 시작으로 해서 위와 같은 과정을 거칩니다. 이런 식으로 모든 원소들을 옮길 때까지 계속합니다.

〈 세 가지 Vector rotation 알고리즘 3 〉 ≡

```
void juggling(x, n, rotdist)
    char *x;    /* 배열 */
    int n;      /* 배열의 크기 */
    int rotdist; /* 문제에서 i에 해당합니다. */
{
    int i, j, k, t;
    int cycles; /* rotdist와 n의 최대공약수, 즉, 루프의 횟수 */
    cycles = gcd(rotdist, n);
    for (i = 0; i < cycles; i++) {
        t = x[i]; /* 시작 원소를 임시 공간에 둡니다 */
        j = i;
        〈 저글링을 실행하라 5 〉
        x[j] = t; /* 임시 저장 장소의 값을 배열의 빈 공간으로 옮긴다 */
    }
}
```

9, 13번 섹션 들도 살펴보세요.

이 코드는 2번 섹션에서 사용됩니다.

4. 함수 *juggling*을 보면, *rotdist* 과 *n*의 최대 공약수 만큼 루프를 도는데, 이 두수의 최대 공약수를 현대 대수학에서는 ‘이 회전으로 야기된 permutation 그룹의 cosets의 수’라고 한다네요. 무슨 말인지 모르겠습니다. 그냥 알아두세요. 두 수의 최대 공약수를 구하는데는 잘 알려진 유클리드의 알고리즘을 이용했습니다.

〈필요한 서브루틴들 4〉 ≡

```
int gcd(int i, int j)
{
    while (i ≠ j) {
        if (i > j) i -= j;
        else j -= i;
    }
    return i;
}
```

7, 8, 12번 섹션 들도 살펴보세요.

이 코드는 2번 섹션에서 사용됩니다.

5. 위 그림과 설명에서 언급했듯이, 모든 인덱스에 대해서 mod *n* 연산을 취하면서 저글링하는 방법은 다음과 같습니다.

〈저글링을 실행하라 5〉 ≡

```
for ( ; ; ) {
    k = (j + rotdist) % n;    /* mod n 연산을 취한다 */
    if (k ≡ i) break;
    x[j] = x[k];
    j = k;
}
```

이 코드는 3번 섹션에서 사용됩니다.

6. Block swap 알고리즘. 두 번째로 살펴볼 알고리즘은 *Block swap* 알고리즘이라 방법입니다. 주어진 배열의 회전에 관한 문제를 다른 시각에서 보면, 배열을 회전하는 문제는 두개의 블록으로 구성된 배열 xy 에서 블록 x 와 블록 y 의 위치를 바꾸어 배열 yx 로 만드는 문제와 동일합니다. 이 글 서두에 제시된 문제에서 블록 x 는 abc 에 해당하고, 블록 y 는 $defgh$ 에 해당합니다.

설명의 편의상 블록 x 의 크기가 블록 y 의 크기보다 작다고 가정합니다. (만약 x 의 크기가 더 크다면, x 와 y 를 바꾸어서 생각하면 그만입니다.) 그리고나서 y 를 다시 두개의 블록 y_l 과 y_r 로 나누는데, y_r 의 크기가 x 의 크기와 같도록 합니다. 이때 x 와 y_r 의 위치를 바꾸면, 처음의 $xy_l y_r$ 은 $y_r y_l x$ 가 됩니다. 이 상황이 되고나면, x 의 위치는 주어진 문제를 해결했을 때의 위치가 됩니다. 즉, 우리가 최종적으로 원하는 모습은 $y_l y_r x$ 인데, $y_r y_l x$ 가 되었으니, 우리는 이제 x 에 대한 신경을 끄고, 나머지 $y_r y_l$ 을 $y_l y_r$ 로 바꾸는데만 집중하면 됩니다.

여기서 뭔가 머리를 확 스치고 지나가지 않나요? 그렇습니다. $y_r y_l$ 을 $y_l y_r$ 로 바꾸는 문제는 우리가 처음 해결하고자 했던 xy 를 yx 로 바꾸는 문제와 똑같아 졌습니다. 즉 문제가 재귀적인 성격을 갖고 있다는 말입니다. 이처럼 원래의 배열을 두 개의 블록으로 나누어서 두 블록의 위치를 바꿔서 해결한다고 해서 이 방법을 *Block swap* 알고리즘 이라고 부르나 봅니다.

7. 문제와 본격적인 전투에 앞서 먼저 총알을 준비하도록 합시다. 앞으로 문제를 해결하다 보면, 배열 내의 서로다른 두 위치내의 원소를 바꾸어 주어야 할 경우가 자주 발생합니다. 이러한 일을 하는 함수를 *swap* 이라고 합시다.

〈필요한 서브루틴들 4〉 +=

```
void swap(char *s, char *t)
{
    char temp;
    temp = *s;
    *s = *t;
    *t = temp;
}
```

8. 다음으로 위의 *swap* 함수를 이용해서 크기가 같은 두 블록을 바꾸는 함수 *blkswap*을 만들어 보겠습니다. 예를 들어, 배열 v 가 $abcdefgh$ 일때, $blkswap(v, 3, 5)$ 는 맨 앞의 3개 abc 와 인덱스 5번 위치부터 3개 fgh 를 바꾸어 원래의 배열을 $fghdeabc$ 로 만듭니다.

〈필요한 서브루틴들 4〉 +=

```
void blkswap(v, n, t)
{
    char *v;
    int n, t;
    while (--n >= 0)
        swap(v + n, v + t + n);
}
```

9. Block swap 알고리즘을 구현한 *block_swaping* 함수는 위의 설명대로 재귀적(recursive) 함수로 구현됩니다. 이 함수는 세개의 파라미터를 필요로 하는데, *v*는 우리가 회전하고자 하는 일차원 배열을 나타내고, *n*은 배열 *v*의 크기, *d*는 회전하고자 하는 원소의 갯수는 나타냅니다.

〈 세 가지 Vector rotation 알고리즘 3 〉 +=

```
void block_swaping(v, n, rotdist)
    char *v;    /* 일차원 배열 */
    int n;      /* 배열의 크기 */
    int rotdist; /* 회전하고자 하는 원소의 갯수 */
{
    int i, j;
    〈종료 조건 10〉;
    i = rotdist; /* 블록 x의 크기 */
    j = n - rotdist; /* 블록 y의 크기 */
    if (i > j) { /* x가 y보다 큰 경우: x를  $x_l x_r$ 로 나눕니다. */
        blkswap(v, j, i); /*  $x_l$ 과 y를 교환하라 */
        block_swaping(v + j, i, i - j); /*  $x_r$ 과  $x_l$ 를 교환하라 */
    } else { /* y가 x보다 큰 경우: y를  $y_l y_r$ 로 나눕니다. */
        blkswap(v, i, j); /* x와  $y_r$ 를 교환하라 */
        block_swaping(v, j, i); /*  $y_r$ 과  $y_l$ 를 교환하라 */
    }
}
```

10. 함수 *block_swaping*은 재귀함수 이므로 반드시 종료 조건을 가져야 합니다. 이 함수의 경우는 *n*과 *d*가 같거나 *d*의 값이 0인 경우가 바로 그때입니다. 즉 이 두 경우는 회전을 해도 배열일 변하지 않는 경우입니다.

〈 종료 조건 10 〉 ≡

```
if (rotdist ≡ n ∨ rotdist ≡ 0)
    return;
```

이 코드는 9번 섹션에서 사용됩니다.

11. Reversal 알고리즘. 앞의 두 알고리즘은 이해하기에 다소 어려운 면이 있었습니다. 지금부터 살펴 볼 알고리즘은 너무도 간단하면서도 참으로 기발한 방법입니다. 아마도 무릎을 치시며 “아하!” 하실지도 모릅니다. 문제를 다시 정의해 보면, 배열 xy 를 yx 로 변환하는 것입니다. 먼저, x^r 을 x 의 역순으로 된 배열이라고 합시다. 예를 들어 x 가 $abcd$ 라면, a^r 은 $dcba$ 가 됩니다. 그러면 Reversal 알고리즘은 다음과 같이 진행됩니다.

$$xy \longrightarrow x^r y \longrightarrow x^r y^r \longrightarrow yx$$

먼저 x 를 역순으로 하고, 그 결과에서 y 를 역순으로 하고, 마찬가지로 그 결과로 나온 $x^r y^r$ 을 역순으로 하면 yx 를 얻을 수 있습니다. 즉, $(x^r)^r = x$ 이고, $(xy)^r = y^r x^r$ 이므로, $(x^r y^r)^r = yx$ 입니다.

12. 함수 *reverse*는 주어진 배열의 일부분을 역순으로 만드는 함수입니다. 예를 들어, *reverse*($v, 2, 5$)는 배열 v 에서 2번 인덱스의 원소 부터 5번 인덱스의 원소까지를 역순으로 만듭니다.

〈필요한 서브루틴들 4〉 +=

```
void reverse(v, s, t)
    char *v;    /* 배열 */
    int s;      /* 바꾸고자 하는 블록의 시작 위치 */
    int t;      /* 바꾸고자 하는 블록의 끝 위치 */
{
    while (s < t)
        swap(v + (s++), v + (t--));
}
```

13. Reversal 알고리즘을 구현한 함수 *reversal*은 *reverse* 함수를 이용해서 매우 간단히 구현됩니다.

〈세 가지 Vector rotation 알고리즘 3〉 +=

```
void reversal(char *v, int n, int rotdist)
{
    reverse(v, 0, rotdist - 1);    /* x를 역순 */
    reverse(v, rotdist, n - 1);    /* y를 역순 */
    reverse(v, 0, n - 1);          /* x^r y^r를 역순 */
}
```

14. 테스트 프로그램 실행. 지금까지 살펴본 세가지 알고리즘을 테스트하는 프로그램을 만들어 봅시다. 프로그램은 다음과 같이 실행됩니다.

```
myhome% vrotate 1 8 5
abcdefgh
fghabcde

myhome% vrotate 2 26 13
abcdefghijklmnoqrstuvwxyz
nopqrstuvwxyzabcdefghijklm

myhome% vrotate 3 10 3
abcdefghij
defghijabc
```

맨 처음 인자 1는 저글링 알고리즘을 나타냅니다. 2는 Block swap 알고리즘을, 3은 Reversal 알고리즘을 나타냅니다. 두번째 인자 8은 배열의 크기이고, 마지막 인자 5는 회전할 원소의 갯수입니다. 이 프로그램은 간단한 테스트 프로그램으로써 구현한 알고리즘의 정상 작동 여부를 파악하기 위한 것으로, 에러 처리에 대한 상당 부분은 생략했습니다.

```
#define JUGGLING 1
#define BLOCK_SWAPING 2
#define REVERSAL 3
```

〈Command line parameter 설정 14〉 ≡

```
if (argc ≠ 4) {
    fprintf(stderr, "usage: _vrotate_ _algo#_ _n_ _d_ \n algo#: \n"
        "\t1: _Juggling_ \n \t2: _Block_swaping_ \n \t3: _Reversal_ \n");
    exit(1);
}
which = atoi(*++argv);
n = atoi(*++argv);
d = atoi(*++argv);
```

이 코드는 2번 섹션에서 사용됩니다.

15. 배열의 크기는 명령행 인수로 부터 동적 생성되고, 배열의 원소는 알파벳 *a*부터 시작하여 배열의 크기 만큼 생깁니다.

〈일차원 배열 생성 및 초기화 15〉 ≡

```
if ((vector = (char *) calloc(sizeof(char), n)) ≡ Λ) {
    fprintf(stderr, "not _enough_memory_. \n");
    exit(2);
}
for (i = 0; i < n; i++)
    vector[i] = 'a' + i;
```

이 코드는 2번 섹션에서 사용됩니다.

16. 주어진 알고리즘을 실행하기 전에 먼저 설정된 배열을 출력하고, 알고리즘 실행 후에 변경된 배열을 출력하여 결과를 확인합니다.

〈 주어진 알고리즘 실행 16 〉 ≡

```
printf ("%s\n", vector);  
if (which ≡ JUGGLING)  
    juggling(vector, n, d);  
else if (which ≡ BLOCK_SWAPING)  
    block_swaping(vector, n, d);  
else if (which ≡ REVERSAL)  
    reversal(vector, n, d);  
printf ("%s\n", vector);
```

이 코드는 2번 섹션에서 사용됩니다.

17. 색인.*argc*: [2](#), [14](#).*argv*: [2](#), [14](#).*atoi*: [2](#), [14](#).*blkswap*: [8](#), [9](#).*block_swaping*: [9](#), [10](#), [16](#).BLOCK_SWAPING: [14](#), [16](#).*calloc*: [15](#).*cycles*: [3](#).*d*: [2](#).*exit*: [14](#), [15](#).*fprintf*: [14](#), [15](#).*free*: [2](#).*gcd*: [3](#), [4](#).*i*: [2](#), [3](#), [4](#), [9](#).*j*: [3](#), [4](#), [9](#).*juggling*: [3](#), [4](#), [16](#).JUGGLING: [14](#), [16](#).*k*: [3](#).*main*: [2](#).*n*: [2](#), [3](#), [8](#), [9](#), [13](#).*printf*: [16](#).*reversal*: [13](#), [16](#).REVERSAL: [14](#), [16](#).*reverse*: [12](#), [13](#).*rotdist*: [3](#), [4](#), [5](#), [9](#), [10](#), [13](#).*s*: [7](#), [12](#).*stderr*: [14](#), [15](#).*swap*: [7](#), [8](#), [12](#).*t*: [3](#), [7](#), [8](#), [12](#).*temp*: [7](#).*v*: [8](#), [9](#), [12](#), [13](#).*vector*: [2](#), [15](#), [16](#).*which*: [2](#), [14](#), [16](#).*x*: [3](#).

- 〈 세 가지 Vector rotation 알고리즘 3, 9, 13 〉 2번 섹션에서 사용됩니다.
- 〈 일차원 배열 생성 및 초기화 15 〉 2번 섹션에서 사용됩니다.
- 〈 저글링을 실행하라 5 〉 3번 섹션에서 사용됩니다.
- 〈 종료 조건 10 〉 9번 섹션에서 사용됩니다.
- 〈 주어진 알고리즘 실행 16 〉 2번 섹션에서 사용됩니다.
- 〈 필요한 서브루틴들 4, 7, 8, 12 〉 2번 섹션에서 사용됩니다.
- 〈 Command line parameter 설정 14 〉 2번 섹션에서 사용됩니다.

배열 회전이동 Algorithms

	섹션 쪽번호	
문학적 프로그래밍	1	1
배열 회전	2	2
Juggling 알고리즘	3	3
Block swap 알고리즘	6	5
Reversal 알고리즘	11	7
테스트 프로그램 실행	14	8
색인	17	10

한글 cweb 프로그래밍에서 한글 북마크가 가능하도록 해주신 김도현님께 깊은 감사를 드립니다.