프로그래밍을 이용한 문제 해결
1일차 – 문제 풀이의 개념

강영민
동명대학교 게임공학과

창의적 소프트웨어 융합 전문 인력 양성 사업단
2016년 1월 – 소프트웨어 역량 강화 프로그램

문제를 푼다는 것

- 문제를 푼다는 것
  - 해법을 찾는 것
- 컴퓨터를 이용하여 문제를 푼다는 것
  - 해법을 찾는 것
    - 알고리즘을 구하기
  - 해법을 컴퓨터가 수행하도록 하는 것
    - 프로그래밍

쉬운 문제

- 해법을 그대로 적용할 수 있는 것
  - 프로그래밍 언어의 문법만 이해하면 됨
- 문제
  - 화면에 “hello world”를 출력하라
- 해법
  - 화면에 문자를 출력하는 함수에 “hello world”를 파라미터로 제공한다.

```
int main(void) {
   printf(“hello world\n”);
   return 0;
}
```

어려운 문제

- 해법이 알려지지 않은 문제
  - 해법부터 찾아야
- 효율적인 해법이 알려지지 않은 문제
  - 많은 문제는 효율적으로 풀 수 있는 방법이 알려져 있지 않다.
    - 가능한 모든 해를 다 찾아 보는 방법
    - 문제가 커지면 찾아야 할 공간이 너무 커진다.
- 효율적인 해법이 복잡할 경우
  - 쉽게 짜면 빨리 만들 수 있지만
  - 큰 문제에는 적용할 수 없다.
  - 큰 문제를 다룰 때를 대비해서 힘들게 짜야 한다.

어려운 문제를 푼다는 것

- 프로그래밍 언어를 습득하는 것 이상의 일
- 효율적인 해법을 고안할 수 있는 능력
  - 누구나 말을 하지만, 누군가는 토론에서 이기고 누군가는 토론에서 진다.
  - 토론은 문법을 안다고 이기는 것이 아니다.
  - 말하는 방법은 문법책으로 익힐 수 없다.
- 글을 쓰고, 말을 해야 단련이 된다.
- 프로그래밍도 훈련되어야 한다.

여러분이 알고 있어야 할 내용

- 프로그래밍 언어
  - C, C++, C#, Java, JavaScript, Swift, Go, Fortran, Cobol, SmallTalk, … 중 어느 것이든 **단 하나.**
- 반드시 알아야 하는 문법
  - 변수 선언
  - 구조체 (둘 이상의 기본형 데이터가 든 복합 자료형)
  - 배열
  - if, switch, for, while
  - 함수 사용

쉬운 문제의 예 – 3n+1

- 3n+1
  - 어떤 수 n이 주어졌을 때, 이 수가 짝수이면 2로 나누고, 홀수이면 3을 곱한 뒤에 1을 더한다.
  - 이렇게 얻은 수에 앞의 단계를 계속 반복한다.
  - 입력된 수는 계속 변하다가 결국 1이 된다.
    - 신기하게도 그렇다고 한다.
  - 1이 되기 위해 첫 번째 단계를 몇 번이나 적용해야 할까?

3n+1: 반복문으로 풀기

- 문제
  - https://github.com/dknife/ProgrammingChallenges/wiki/2016SWPCDay1Prob1
- 정답

```c
#include <stdio.h>

int main(int argc, char **argv) {
    unsigned int i, j;
    printf("input two numbers : ");
    scanf("%u", &i);
    scanf("%u", &j);

    if(i>j) { unsigned int t = i; i=j; j=t; }

    unsigned int maxCycle=1;

    for(int k=i; k<=j; k++) {
        unsigned int number = k;
        unsigned int cycle = 1;
        while (number != 1) {
            if(number%2) number=3*number+1;
            else number=number/2;
            cycle++;
        }
        if(cycle>maxCycle) maxCycle=cycle;
    }

    printf("%u %u %u\n", i, j, maxCycle);
}
```

3n+1: 재귀호출을 이용한 풀이

```c
#include <stdio.h>

// n/2 or 3n + 1
int half_Or_TriplePlusOne(int level, unsigned int n) {
    if(n<0) return 0;
    printf("[%u]", n);
    if(n==1) return level;
    if(n%2==0) n>>=1;
    else n=3*n+1;
    return half_Or_TriplePlusOne(level+1, n);
}
int main(int argc, char **argv) {
    unsigned int n;
    while(1) {
        printf("input a number: ");
        scanf("%u", &n);
        if(n<=0) return 0;
        int len = half_Or_TriplePlusOne(1, n);
        printf("\nlen = %d\n", len);
    }
}
```

문제의 조건에 맞는 재귀호출 사용

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

int half_Or_TriplePlusOne(int level, unsigned int n) {
    if(n==1) return level;
    n = (n%2)?3*n+1:n>>1;
    return half_Or_TriplePlusOne(level+1, n);
}
int main(int argc, char **argv) {
    unsigned int i, j;
    printf("input two numbers : ");
    scanf("%u", &i);
    scanf("%u", &j);

    if(i>j) { unsigned int t = i; i=j; j=t; }

    unsigned int maxCycle=1;

    for(int k=i; k<=j; k++) {

        unsigned int cycle = half_Or_TriplePlusOne(1, k);
        if(cycle>maxCycle) maxCycle=cycle;
    }

    printf("%u %u %u\n", i, j, maxCycle);
}
```

재귀호출은 강력하다

- 이유 1. 프로그램을 간단하게 만든다.
- 유명한 문제
  - 누승(factorial)
    - n! = n(n-1)(n-2)(n-3)...1
    - 3! = 3 * 2 * 1 = 6
    - 5! = 5 * 4 * 3 * 2 * 1 = 120
  - 프로그램으로 구현하라

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

unsigned int fact(unsigned int n) {
    return n>1?fact(n-1)*n:1;
}

int main(int argc, char **argv) {
    unsigned int n = atoi(argv[1]);
    printf("%u!=%u\n", n, fact(n));
}

```

재귀호출은 강력하다

- 이유 2: 문제를 빠르게 해결할 수 있다.
  - 문제가 "분할정복"으로 해결될 경우
- 관련 문제
  - Day1 문제 3. power(b, e);
    - b^e 를 구하기

이런 코드는 곤란하다

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

double power(double base, unsigned int exponent) {
    double result=1.0;
    for(unsigned int n=0; n<exponent; n++) result *= base;
    return result;
}

int main(int argc, char **argv) {
    double base = atof(argv[1]);
    unsigned int exponent = atoi(argv[2]);
    printf("%lf^%u = %lf\n", base, exponent, power(base, exponent));
}
```

빠른 코드

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

double power(double base, unsigned int exponent) {
    if(!exponent) return 1.0;
    double val = power(base, exponent>>1);
    return (exponent&1)? base*val*val : val*val;
}

int main(int argc, char **argv) {
    double base = atof(argv[1]);
    unsigned int exponent = atoi(argv[2]);
    printf("%lf^%u = %lf\n", base, exponent, power(base, exponent));
}

```

이것은 왜 빠를까?

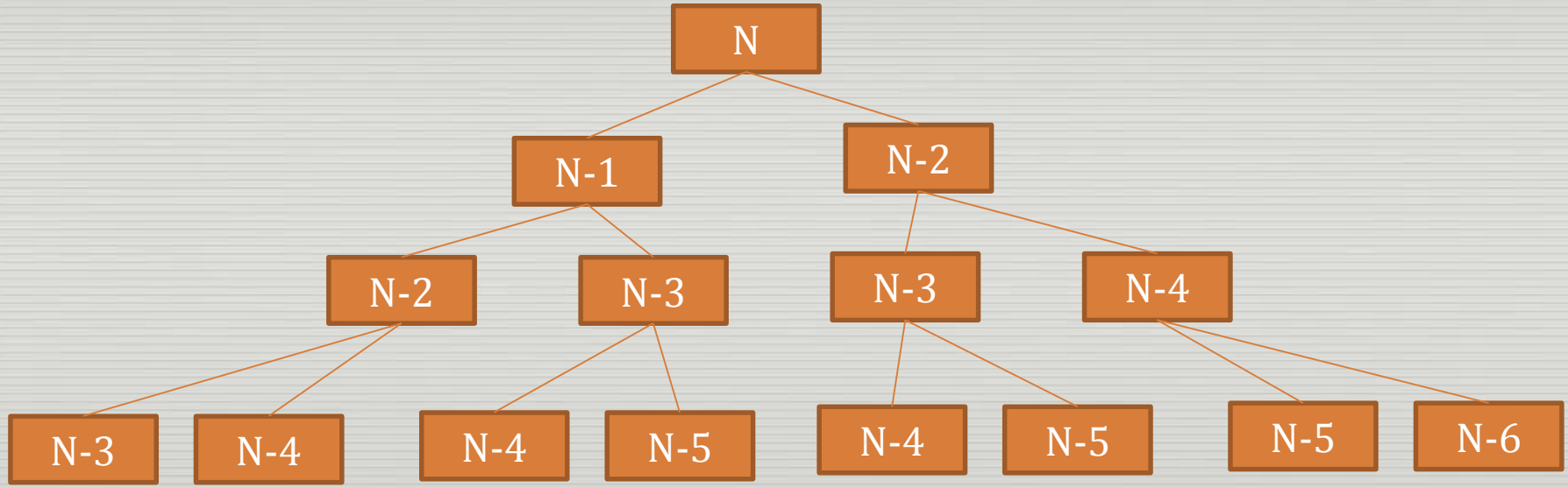
재귀호출은 만능인가?

- 피보나치 수열을 구해보자
  - 피보나치 수열
    - a(0) = 1
    - a(1) = 1
    - a(n) = a(n-1) + a(n-2)
- 재귀호출로 짜기 좋은 문제로 '보인다'

```
Fibo(n) [
    if(n<=1) return 1;
    return Fibo(n-1)+Fibo(n-2);
]
```

왜 문제인가

- Fibo(n)은 몇 번의 함수 호출을 할까?



Stack overflow!

합리적인 해법

- 이때는 점화식을 그대로 재귀호출로 옮기는 방식은 적합하지 않다.
- 반복문이 낫다

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

int main(int argc, char **argv) {
    unsigned long int p=1, pp=1;
    unsigned long int res;
    int n = atoi(argv[1]);

    for(int i=2; i<=n; i++) {
        res = p+pp;
        pp=p; p=res;
    }
    printf("fibo[%lu] = %lu\n", n, res);
}

```

유쾌한 점프

- 문제
  - https://github.com/dknife/ProgrammingChallenges/wiki/2016SWPCDay1Prob5
- 배열을 이용하여 푼다
- 배열
  - 지금까지 변수는 하나씩 독립적으로 존재
  - 배열은
    - 같은 자료형(data type)의 변수가 메모리 공간에 연속적으로 존재
    - 각각의 개별 데이터는 색인(index)로 접근
  - 배열은 다양한 문제에서 유용한 자료구조
  - 특히... 동적 계획법(dynamic programming)에서 유용
    - 이건 나중에 다룰 것

유쾌한 점퍼

- 풀이법을 생각해 보자
  - N 개의 데이터가 들어오면
  - 인접한 수들의 차가 모두 서로 달라야 하며
  - 그 값은 1, 2, … N-1이어야 함
  - N-1개의 원소를 가진 배열 array을 생성
  - 입력되는 수들의 차가 k이면
    - 배열 array[k-1]을 1로
    - 입력이 모두 끝나면 array의 모든 원소가 1이어야 함
    - 그렇지 않으면 “not jolly”

문제풀이 – 정적 배열

```c
#include <stdio.h>


int main(int argc, char **argv) {
    unsigned int n;
    int seq[3000];
    unsigned int diff[3000];
    printf("input your sequence (n a1 a2 ... an): ");
    scanf("%u", &n);

    for(int i=1;i<=n-1;i++) diff[i] = 0;

    for(int i=0;i<n;i++) scanf("%d", &seq[i]);

    for(int i=0;i<n-1;i++) {
        int difference = seq[i]-seq[i+1];
        if(difference<0) difference*=-1;
        if(difference<1 || difference>n-1) { printf("Not Jolly\n"); return 0; }
        if(diff[difference]==0) diff[difference] = 1;
        else { printf("Not Jolly\n", difference); return 0; }
    }
    printf("Jolly\n");
}
```

문제풀이 - 동적할당

```
#include <stdio.h>
#define abs(x) ((x)<0? (-(x)):(x))

int main(int argc, char **argv) {
    unsigned int n;
    int prev; int cur;
    unsigned int *diff;
    printf("input your sequence (n a1 a2 ... an): ");
    scanf("%u %d", &n, &prev);
    diff = new unsigned int[n-1];

    for(int i=0;i<n;i++) diff[i] = 0;
    for(int i=0;i<n-1;i++) {
        scanf("%d", &cur);
        int difference = abs(cur-prev);
        if(difference<1 || difference>n-1) { printf("Not Jolly\n"); return 0; }
        if(diff[difference-1]==0) diff[difference-1] = 1;
        else { printf("Not Jolly\n", difference); return 0; }
        prev = cur;
    }
    printf("Jolly\n");
}
```

암호깨기 문제

- 알려진 평문 공격법(known plain text attack)
- 문제
  - https://github.com/dknife/ProgrammingChallenges/wiki/2016SWPCDay1Prob6

풀이

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

// coding and decoding array for 26 letters
static char coding[26], decoding[26];
bool matchCheck(char *known, char *input);

int main(int argc, char **argv) {
    static char knownText[] = "the quick brown fox jumps over the lazy dog";
    static char text[100][80];

    int numLines;
    scanf("%d", &numLines);
    fgets(text[0], 80, stdin);

    bool matchFound = false;
    for(int i=0;i<numLines;i++) {
        fgets(text[i], 80, stdin);
        if(!matchFound) matchFound = matchCheck(knownText,text[i]);
    }

    if(!matchFound) {
        printf("No Solution"); return 0;
    }

    for(int i=0;i<numLines;i++) {
        for(int j=0; j<strlen(text[i]); j++) {
            putchar(decoding[text[i][j]-'a']);
        } printf("\n");
    }
}
```

```c
bool matchCheck(char *known, char *input) {
    int len = strlen(input)-1; // removing one invisible character
    if(strlen(known)!=len) return false;

    for(int i=0; i<26; i++) { coding[i] = decoding[i] = '\0'; }

    for(int i=0; i<len; i++) {
        char A = known[i];
        char B = input[i];
        int IdxA = A -'a';
        int IdxB = B -'a';

        if(coding[IdxA]=='\0') coding[IdxA]=B;
        else if(coding[IdxA]!=B) return false;

        if(decoding[IdxB]=='\0') decoding[IdxB]=A;
        else if(decoding[IdxB]!=A) return false;
    }
    return true;
}
```