im4u 봄방학 캠프
DAY 1; introduction & 간보기

구종만
jongman@gmail.com

강사소개

- 구종만
- jongman@gmail.com
- 010-4153-1547
- 연세대학교 컴퓨터과학과

캠프에서 무엇을 할까?

- 책 보고 알고리즘 배우는 것은 언제나 할 수 있고
- 학원 선생님한테 수업이야 일 년 내내 듣는 거고
- 문제야 일 년 내내 푸는 거고……
- 포인트는 '집중'
  - 짧은 시간에 많은 문제를 푸는 것의 효과
  - 문제를 푸는 데 필요한 '깨달음' 을 체화할 수 있다~!
  - 짧은 시간에 많은 코드를 짜 봄으로써 손에 완전히 익힐 수 있다~!
  - 그러니까 놀지말고 이주일만 달려봅시다

문제 해결 개관

- 모델링
- 알고리즘 설계 및 정당성 확인
- 적절한 도구 (자료구조, 알고리즘) 의 사용
- 코드 작성 및 효율성 확인

- 네개만 잘하면 초고수 +_+

문제 모델링하기

- 현실 세계의 문제를 적절한 전산학적/수학적 구조로 바꿔서 표현하기
  - 문제의 재귀적인 특성 찾아내기
  - 적절한 수학적/전산학적 모델로 표현하기

알고리즘 설계 및 정당성 확인

- 알고리즘 설계 패러다임
  - 분할 정복 / 동적 계획법
  - 완전 탐색 / 기타 탐색 기법들 (IDA*, …)
  - 탐욕법
- 알고리즘 설계 방식들
  - 중복을 무시하기
  - 최적화 문제를 결정 문제로 변환하기
  - 순서를 무시하기: 스위핑 알고리즘들

알고리즘의 증명

- 증명
  - 증명을 많이 따라 해봐야 나중에 자기도 할 수 있다
  - 연습 때는 가능한 한 전부 증명, 대회 때는 맘대로
- 증명에는 일정한 패턴이 있다
  - 루프 불변 조건을 이용한 귀납법
  - 귀류법
- 오류 없는 코드를 작성하는 데 큰 도움이 된다

적절한 도구의 사용

- 다양한 탐색 자료 구조 및 알고리즘의 사용
  - Fenwick Tree (aka Binary Indexed Tree)
  - 구간 트리 (Segment Tree)
  - 최소 공통 조상 (Least Common Ancestor)
  - 행렬
  - Suffix Tree / Suffix Array
  - KMP String Match
  - Aho-Corasick String Match
  - … (무지 많음) …

코딩

- 가장 무시되기 쉬운 부분
  - 같은 문제를 같은 시간에 해결하는 코드라고 해서 다 같은 것이 아니다
- 이해가 쉽고 간결한 코드를 짜는 버릇을 들이자
  - 이런 코드는 디버깅하기가 쉽고
  - 심지어는 (버그 없이) 짜기도 더 쉽다
  - 인생에도 도움이 많이 될 걸?
- 오늘의 메인 주제

키워드는 '습관' '직관'

- 짧은 시간에 많은 일을 해야 하는 대회 환경에서는 직관이 가장 중요하다

- 직관은 경험으로만 얻어질 수 있다
  - 맞는 코드를 짜는 습관이 들어 있어야 어떻게 짜야 하는지 알 수 있다
  - 알고리즘을 따라 증명해 봐야 내 접근 방법이 맞은지도 증명할 수 있다
  - 알고리즘을 완전히 이해하고 있어야 그 변형도 정당한 것인지 알 수 있다

코딩 향상을 위한 조언들

- 변수와 함수에 대한 적절한 명명법을 사용하라
  - f(y, x) 는 무슨 답을 반환하는가?
  - isGood(y, x) 는 무슨 답을 반환하는가?
  - 동사+명사 명명법까지 쓰면 금상첨화겠지만..
- 적절하고 일관된 포매팅을 사용하라
  - (제발 좀) 공백에 인색해지지 마라
- 변수의 범위를 좁혀라
  - for 문에 쓰는 변수는 for 안에서 선언
  - 다 쓰고 난 변수는 scope 안에서 사라져야 한다
  - 전역 변수의 개수를 최소화하라

코딩 향상을 위한 조언들

- 함수가 하는 일을 명확하게 정의하라
  - 입력과 출력이 명확하게 정의되어야 한다
  - 다섯 가지 일을 하는 doit()
  - input()/process()/output() 패턴
  - 각각 한 가지 일을 하는 다섯 개의 함수

- 같은 코드가 세 번 반복되면 (주저하지 말고) 별도의 함수를 만들어라
  - Copy-and-paste 는 재앙을 불러온다
  - (Debugging Hell)

코딩 향상을 위한 조언들

- 디버깅에 너무 의존하지 말아라
  - F5 부터 누르는 습관을 버려라: 디버거는 5천줄 넘는 프로그램을 짜는 사람들을 위한 물건이다
  - (프로그램이 어디서 죽는지 알아낼 때만 빼고)
- 매번 다른 방식으로 코드를 작성하는 버릇을 버려라
  - 매번 코드를 검증하고 있을 여유 따위는 없다
  - 스스로에게 가장 잘 맞고 가장 안정적으로 짤 수 있는 '한 가지 방법' 을 찾아낸 뒤, 그 방법으로만 짜라

코딩 향상을 위한 조언들

- 언어와 표준 라이브러리에 익숙해져라
  - 매크로, 표준 라이브러리, 그리고 STL 을 적극 이용하라
  - 코드를 훨씬 짧고 간결하게 만들어 줄 수 있다
- 간결한 코드에 집착해라
  - 인간은 실수가 많은 동물이라 무얼 해도 실수할 수 있다
  - 더 적은 바이트 수와 더 적은 라인 수는 더 적은 버그를 의미한다
- 문제를 해결하고 나서도 계속 배우라
  - 두 번째 짠 코드는 첫 번째 짠 코드와 다르다
  - 어떻게 하면 더 나은 코드를 짤 수 있을까?
  - 잘 짠 남의 코드를 구경하라

제발 쓰지 말아야 할 것들

- 메모리 관리 직접 하지 마라
  - 몇천명 동접 받는 게임 서버에서도 안 한다, 왜 너네가 하냐 ㅜㅜ
  - int* x = new int[10]; => int x[10];
  - Int* x = new int[n]; => vector<int> x(10, 0);
- 자료 구조 직접 구현하지 마라
  - heap 직접 구현하고 디버깅할 시간에 문제 하나 더 풀 수 있다
  - 심지어 queue 를 구현하는 것도 시간낭비 (라고 주장 중)

C++ In A Nutshell

- 변수들의 유효 범위 (scope)
- 레퍼런스 자료형 (reference)
- 오퍼레이터 오버로딩
- STL
  - pair, vector, queue, priority_queue, map, set, …
  - lower_bound, upper_bound, kth_element, next_permutation, …

Discovering C++

```
const int MONTHS = 12;
MONTHS = 13;
cout << MONTHS << endl;
```

- ?

```
const int MONTHS = 12;
MONTHS = 13;
cout << MONTHS << endl;
```

- COMPILE ERROR!

Discovering C++

```
const string name("BoA");
name[2] = 'C';
cout << name << endl;
```

- ?

```
const string name("BoA");
name[2] = 'C';
cout << name << endl;
```

- COMPILE ERROR!

Discovering C++

```
double a = 3/4;
double b = double(3) / double(4);
cout << a << " " << b << endl;
```

- ?

Discovering C++

```
double a = 3/4;
double b = double(3) / double(4);
cout << a << " " << b << endl;
```

- 0 0.75

Discovering C++

```
long long a = 2147483647LL;
int b = 10;
int c = 2147483647;
cout << a * b << " " << b * c << endl;
```

- ?

Discovering C++

```
long long a = 2147483647LL;
int b = 10;
int c = 2147483647;
cout << a * b << " " << b * c << endl;
```

- **21474836470 -10**

Discovering C++

```
int * pt = new int; // allocate space for an int
*pt = 1001; // store a value there

cout << "int ";
cout << "value = " << *pt << ": location = " << pt << " ";

double * pd = new double; // allocate space for a double
*pd = 10000001.0; // store a double there

cout << "double ";
cout << "value = " << *pd << ": location = " << pd << " ";
cout << "size of pt = " << sizeof pt;
cout << ": size of *pt = " << sizeof *pt << " ";
cout << "size of pd = " << sizeof pd;
cout << ": size of *pd = " << sizeof *pd << " ";

delete pt;
delete pd;
```

■ ?

Discovering C++

```
int * pt = new int; // allocate space for an int
*pt = 1001; // store a value there

cout << "int ";
cout << "value = " << *pt << ": location = " << pt << " ";

double * pd = new double; // allocate space for a double
*pd = 10000001.0; // store a double there

cout << "double ";
cout << "value = " << *pd << ": location = " << pd << " ";
cout << "size of pt = " << sizeof pt;
cout << ": size of *pt = " << sizeof *pt << " ";
cout << "size of pd = " << sizeof pd;
cout << ": size of *pd = " << sizeof *pd << " ";

delete pt;
delete pd;
```

- **int value = 1001: location = 0x004301a8**
- **double value = 1e+07: location = 0x004301d8**
- **size of pt = 4: size of *pt = 4**
- **size of pd = 4: size of *pd = 8**

Discovering C++

```
int rats = 101;
int & rodents = rats; // rodents is a reference

cout << "rats = " << rats;
cout << ", rodents = " << rodents << "₩ n";
rodents++;
cout << "rats = " << rats;
cout << ", rodents = " << rodents << "₩ n";
```

- ?

Discovering C++

```
int rats = 101;
int & rodents = rats; // rodents is a reference

cout << "rats = " << rats;
cout << ", rodents = " << rodents << "₩ n";
rodents++;
cout << "rats = " << rats;
cout << ", rodents = " << rodents << "₩ n";
```

- **rats = 101, rodents = 101**
- **rats = 102, rodents = 102**

Discovering C++

```
int rats = 101;
int & rodents = rats; // rodents is a reference

cout << "rats = " << rats;
cout << ", rodents = " << rodents << "₩ n";

int bunnies = 50;
rodents = bunnies; // can we change the reference?
cout << "bunnies = " << bunnies;
cout << ", rats = " << rats;
cout << ", rodents = " << rodents << "₩ n";
```

- ?

Discovering C++

```
int rats = 101;
int & rodents = rats; // rodents is a reference

cout << "rats = " << rats;
cout << ", rodents = " << rodents << "₩ n";

int bunnies = 50;
rodents = bunnies; // can we change the reference?
cout << "bunnies = " << bunnies;
cout << ", rats = " << rats;
cout << ", rodents = " << rodents << "₩ n";
```

- **rats = 101, rodents = 101**
- **bunnies = 50, rats = 50, rodents = 50**

Discovering C++

```
void swapr(int & a, int & b) // use references
{
int temp;
temp = a;
a = b;
b = temp;
}

void swapp(int * p, int * q) // use pointers
{
int temp;
temp = *p;
*p = *q;
*q = temp;
}
```

```
void swapv(int a, int b) // try using values
{
int temp;
temp = a;
a = b;
b = temp;
}
```

- ?

Discovering C++

```
void swapr(int & a, int & b) // use references
{
int temp;
temp = a;
a = b;
b = temp;
}

void swapp(int * p, int * q) // use pointers
{
int temp;
temp = *p;
*p = *q;
*q = temp;
}
```

```
void swapv(int a, int b) // try using values
{
int temp;
temp = a;
a = b;
b = temp;
}
```

- !

Discovering C++

```
int harpo(int n, int m = 4, int j = 5);
int chico(int n, int m = 6, int j);
int groucho(int k = 1, int m = 2, int n = 3);
```

- ?

Discovering C++

```
int harpo(int n, int m = 4, int j = 5);
int chico(int n, int m = 6, int j);
int groucho(int k = 1, int m = 2, int n = 3);
```

- !

Discovering C++

```
int teledeli = 5;
{
cout << "Hello ";
int websight = -2;
int teledeli = 7;
cout << websight << ' ' << teledeli << endl;
}
cout << teledeli << endl;
```

- ?

Discovering C++

```
int teledeli = 5;
{
cout << "Hello ";
int websight = -2;
int teledeli = 7;
cout << websight << ' ' << teledeli << endl;
}
cout << teledeli << endl;
```

- **Hello -2 7**
- **5**

That's It For Today

- ㅅㄱ염