SCCC 알고리즘 스터디

<STL>

STL 이란?

- 표준 C++ 라이브러리 (Standard Template Library)
- Templete를 이용해 기존 기능을 간단하게 사용
  - 이를 이용해 PS문제를 편리하게 풀수 있다.
- 구성 요소
  - 컨테이너 : stack, vector, queue 등
  - 알고리즘 : 정렬, 삭제, 이분탐색 등
  - 반복자 : iterator
  - 함수자

헤더 추가 방법

- C 헤더

```c
/* cos example */
#include <stdio.h>
#include <math.h>

```

- C++ 헤더

```cpp
// vector::push_back
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector;
  int myint;

```

using namespace std 로 std:: 를 생략할 수 있다.

내용 순서

1. 기타

   memory.h(memset), math.h, iostream,

2. 컨테이너

   stack, queue, deque, vector, priority_queue, pair, tuple, string, set, map,

3. 알고리즘

   swap, fill, sort, unique, binary_search, lower_bound

memset

- 변수, 배열, 문자열등의 주소를 통해 값을 지정
- string.h, memory.h를 이용해 사용가능

function

memset

```
void * memset ( void * ptr, int value, size_t num );
```

- 예)

```
1  /* memset example */
2  #include <stdio.h>
3  #include <string.h>
4
5  int main ()
6  {
7    char str[] = "almost every programmer should know memset!";
8    memset (str,'-',6);
9    puts (str);
10   return 0;
11 }
```

Output:

```
------ every programmer should know memset!
```

memset

- 대입되는 값은 모두 바이트 단위
  - int, long과 같은 자료형에는 예상치 못한 값이 대입될 수 있음

- 주로 사용 되는 방법
  - int,long 배열 원소를 모두 0으로 만들기 : memset(arr,0,sizeof(arr));
  - int,long 배열 원소를 모두 -1으로 만들기 : memset(arr,-1,sizeof(arr));

math.h

- Hypot

```
     double hypot  (double x      , double y);
      float hypotf (float x       , float y);
long double hypotl (long double x, long double y);
```

$\sqrt{x^2 + y^2}$ 를 반환
좌표계에서 원점과의 거리, 두 점과의 거리를 구하는데 편리

- atan

```
     double atan  (double x);
      float atanf (float x);
long double atanl (long double x);
```

tan함수의 역함수
직선의 기울기를 구하는데 유용

iostream

- C++ 입출력 헤더
  - cin : 표준 입력 스트림
    - >> 연산자로 변수에 입력값을 대입
    - 똑똑해서 어디 변수에 어떤 값을 대입해야하는지 알아서 판단
    - scanf보다 속도가 느릴 수 있음
  - cout 표준 출력 스트림
    - << 연산자로 변수값을 화면에 출력
  - printf, scanf와 사용시 예측 불가능한 순서로 입력, 출력될수 있다.
    - tie, sync_with_stdio(false)로 스트림 지정, 표준 입출력과 동기화 on/off

컨테이너

- 데이터를 저장하는 자료구조

pair	vector	string
stack	priority_queue	map
queue	set	deque

등등

- 하나씩 배우면서 백준 문제를 풀어보자

Stack

- 후입선출(Last In First Out) 컨테이너
- 사용법

```cpp
#include <iostream>
#include <stack>          // Stack 헤더추가
using namespace std;

int main ()
{
    stack<int> S;   // Stack 선언
    S.push(1);      // 원소 삽입
    S.top();        //맨 위 원소 반환
    S.pop();        //맨 위 원소 삭제
    S.size();       //스택 원소 개수 반환

    return 0;
}
```

Stack

- 풀어봅시다
  - 10828 - 스택
  - 10799 - 쇠막대기

Queue

- 선입선출(First In First Out) 컨테이너
- 사용법

```cpp
#include <iostream>
#include <queue>          // queue 헤더추가
using namespace std;

int main ()
{
    queue<int> Q;   // Queue 선언
    Q.push(1);      // 원소 삽입
    Q.front();      //맨 앞 원소 반환
    Q.back();       //맨 뒤 원소 반환
    Q.pop();        //맨 앞 원소 삭제
    Q.size();       //스택 원소 개수 반환

    return 0;
}
```

Queue

- BFS에서 자주 사용
- 풀어봅시다
  - 10845 - 큐

Deque

- 양방향입출이 가능한(**d**ouble-**e**nded **que**ue) 컨테이너
- 사용법

```cpp
#include <iostream>
#include <deque// deque 헤더추가
using namespace std;

int main ()
{
    deque<int> D;        // Deque 선언
    D.push_front(1);     // 맨 앞에 원소 추가
    D.push_back(2);      //맨 뒤에 원소 추가
    D.front();           //맨 앞 원소 조회
    D.back();            //맨 뒤 원소 조회
    D.pop_front();       //맨 앞 원소 삭제
    D.pop_back();        //맨 뒤 원소 삭제
    return 0;
}
```

Deque

- 풀어봅시다
  - 10866 -덱

Vector

- 대표적인 시퀀스 컨테이너
  - 매우 많이 사용
  - Iterator를 이용해 정렬, 이분탐색 등 사용가능
- 사용법

```cpp
#include <iostream>
#include <vector> // vector 헤더추가
using namespace std;

int main ()
{
    vector<int> V;              //vector 생성
    for(int i=0;i<10;i++){
        V.push_back(i);         //맨 뒤에 원소 추가
    }
    cout << V[2];               // 임위 위치의 원소 조회
    V[3] = 4;                   // 원소의 값 수정 가능
    for(V.size()>0){            // 원소의 개수 조회
        V.pop_back();           // 맨 뒤의 원소 삭제
    }
    return 0;
}
```

priority_queue

- 우선순위큐
- 맨 위의 원소를 비교(함수)를 통한 최상위원소를 맨앞에 배치하는 컨테이너
- **힙 구현 가능**

priority_queue 구현

- 두 자료형간의 비교 방식을 정해야한다.
  1. 해당 객체의 비교조건자가 있다면 이를 이용해 비교 구현
     기본 자료형은 이미 비교조건자가 있다. int,long, string,pair, tuple 등등
  2. 비교 템플릿을 이용한다.
  3. 객체의 비교조건자를 구현한다.

priority_queue 구현

- 사용법

```
#include <iostream>
#include <vector> // vector 헤더 추가
#include <queue> // queue 헤더추가
#include <functional> // 비교템플릿를 위한 헤더 추가
using namespace std;
struct CompareRequestP
{
    bool operator() (const Request* req1, const Request* req2) const // 비교 조건자 구현
    {
        return req1->track > req2->track;
    }
};
int main ()
{
    priority_queue<int,vector<int>,greater<int> > Q1; // 비교 연산자가 있기에 세번째 인자는 무시할수 있다.
    //주의 구 C++컴파일러는 >>가 붙어있으면 컴파일 오류를 줄수도 있음
    priority_queue<CompareRequestP,vector<CompareRequestP>,CompareRequestP> Q2 // 비교 조건자를 이용
    Q1.size();      // 원소 개수 반환
    Q1.push(1);     // 원소 추가
    Q1.top();       // 원소 맨 위(비교 조건자를 통한 제일 최상위 원소) 반환
    Q1.pop();       // 원소 맨 위 삭제
    return 0;
}
```

priority_queue

- 풀어봅시다
  - 1927 – 최소 힙
  - 11279 – 최대 힙
  - 2750 – 수 정렬하기

pair, tuple

- 여러 자료형을 묶어주는 자료형
- pair : 두개의 자료형을 쌍으로 묶는다
  - 첫 원소는 first, 두번째 원소는 second
- tuple : 두개 이상의 자료형을 묶는다
  - 사용이 번거롭기에 struct를 사용하는게 편할때가 있다.

pair, tuple

- 사용법 (pair)

```cpp
#include <iostream>
#include <pair>

using namespace std;

int main ()
{
   pair<int,int> P; // pair 생성
   P.first = 1;     // pair의 첫 원소에 대입
   P.second = 2;     // pair의 두번째 원소에 대입
    return 0;
}
```

pair, tuple

- 사용법 (tuple)

```cpp
#include <iostream>
#include <tuple>        // tuple 헤더 추가
using namespace std;
int main ()
{
  tuple<int,char> foo (10,'x');                          // 생성자함수를 이용한 생성
  auto bar = make_tuple ("test", 3.1, 14, 'y');          // make_tuple 함수를 이용한 생성
  // auto를 통해 자료형을 축약할 수 있다.

  get<2>(bar) = 100;                                     // 2번째 원소 조회및 대입

  int myint; char mychar;

  tie (myint, mychar) = foo;                             // myint,mychar변수에 해당 tuple 값을 대입

  return 0;
}
```

string

- 문자열 배열의 상위호환 :)
- 내장된 함수로 문자열 취급이 편리
- 사용법

```cpp
#include <iostream>
#include <sring>        // string 헤더 추가
using namespace std;
int main ()
{
    string s1 = "123";

    string s2 ="456"
    string s3 = s1+ s2; // == 123456
    s3[3] = 'a';
    s3.size();
    s3.substr(0,3); // 0번째 인덱스부터 3개를 자름
    s3.push_back('a');
    s3.pop_back();


  return 0;
}
```

string

- 풀어봅시다
  - 10809 – 알파벳 찾기
  - 1157 – 단어 공부

set

- 집합 컨테이너
- 총 4가지 집합의 종류가 있다.
  - Set : 중복을 허용하지 않는 집합 (오름차순으로 정리)
  - Multiset : 중복을 허용하는 집합
  - Unordered_set : hash를 이용한 set
  - Unordered_multiset : hash를 이용한 multimap

set

- 사용법

```
#include <iostream>
#include <set>        // set 헤더 추가
using namespace std;
int main ()
{
   set<int> S;
   S.insert(1);      // set에 원소 추가
   S.insert(2);
   S.insert(1);      // 중복된 값을 넣어도 같은 원소는 1개
   S.size();         // 원소의 개수 반환
   set<int>::iterator it = S.begin(); //iterator를 이용해 원소를 조회
    for (it; it != S.end(); it++){
        printf("%d ", *it);
    }


  return 0;
}
```

map

- Key,value 형식으로 저장하는 컨테이너
- 총 4가지 종류의 map이 있다
  - Map : <key, value> 형식으로 저장
  - Multimap : 중복 key가 가능한 map
  - Unordered_map : hash를 이용한 map
  - Unordered_multimap : hash를 이용한 multimap

map

- 사용법

```cpp
#include <iostream>
#include <string>
#include <map>        // map 헤더 추가
using namespace std;
int main ()
{
  map<char,string> mymap;

  mymap['a']="an element";      //[]연산자를 이용해 원소 추가
  mymap['b']="another element";
  mymap['c']=mymap['b'];        //[] 연산자로 대입가능
  mymap.insert(make_pair('d',"asdf")); // insert함수를 이용한 원소 추가, pair를 이용
  return 0;
}
```

map

- 풀어봅시다
  - 1764 – 듣보잡
  - 1157 – 단어 공부

알고리즘

- STL 컨테이너를 활용한 알고리즘이 내장
- 검색, 정렬 등 기능을 손쉽게 사용가능
- 잘 활용하면 PS 문제를 매우 빠르게 풀 수 있다.
  - 너무 자주 사용하면 실제 구조를 잊어버릴수도...
- 대부분 <algorithm> 헤더에 내장

swap

- 같은 자료형의 두 변수의 값을 교체
  - Tmp변수를 선언 안하고 사용할수 있어 편리
- 사용법

```
#include <iostream>
#include <algrithm> // algorithm 헤더 추가
using namespace std;
int main ()
{
  int x=10, y=20;                // x:10 y:20
  swap(x,y);                  // x:20 y:10
  return 0;
}
```

fill

- 컨테이너의 원소들의 값을 변경
- memset과는 달리 byte단위로 동작하지 않는다.
- 사용법

```cpp
#include <iostream>
#include <algrithm> // algorithm 헤더 추가
#include <vector>

using namespace std;
int main ()
{
  std::vector<int> myvector (8);                       // myvector: 0 0 0 0 0 0 0 0
  std::fill (myvector.begin(),myvector.begin()+4,5);   // myvector: 5 5 5 5 0 0 0 0
  std::fill (myvector.begin()+3,myvector.end()-2,8);   // myvector: 5 5 5 8 8 8 0 0
  return 0;
}
```

sort

- 컨테이너의 원소들을 정렬
  - 퀵 정렬기법을 이용 : 평균 시간복잡도 O(NlogN)
- 비교 연산자 정의 or 비교 함수 정의가 필요

sort

```cpp
#include <iostream>
#include <algrithm> // algorithm 헤더 추가
#include <vector>
#include <functional>

using namespace std;
int comp(int a,int b){
    return a < b; // 비교 함수 구현
}
int main ()
{
    vector<int> V;
    for(int i=10;i>=0;i--)
        V.push_back(i);          // [10,9,8,7,6,5,4,3,2,1,0]


    sort(V.begin(), V.end(), greater<int>());  //비교 템플릿을 활용
    sort(V.begin(),V.end(),comp);               //비교 함수를 활용
    for (int i : V){
        printf("%d ", i);
    }

  return 0;
}
```

sort

- 풀어봅시다
  - 11651 – 수 정렬하기 2
  - 10814 – 나이순 정렬
  - 11650 – 좌표 정렬하기

unique

- 컨테이너의 중복된 원소를 1개로 만들때 사용
  - 중복된 원소가 있다면 고유 원소들 개수+1번째부터 모여진다
- 정렬되어 있는 컨테이너에서 사용
- 좌표압축을 위해 많이 사용한다

unique

- 사용법

```
#include <iostream>
#include <algrithm> // algorithm 헤더 추가
#include <vector>
#include <functional>

using namespace std;
int comp(int a,int b){
    return a < b; // 비교 함수 구현
}
int main ()
{
    vector<int> V;
    for(int i=0;i<10;i++)
        V.push_back(i%3);          // [0,1,2,0,1,2,0,1,2,0]
    sort(V.begin(),V.end(),less<int>());
    auro it = unique(V.begin(),V.end());  //[0,1,2,?,?,?,?,?,?,?], 첫 중복 iterator를 반환
    V.erase(it,V.end());     // [0,1,2]
  return 0;
}
```

binary_search, lower_bound, upper_bound

- 이분탐색 알고리즘 O(logN) -> 정렬 후 사용가능
  - binary_serach : 컨테이너에서 원소가 있는지 없는지 true/false
  - lower_bound : 컨테이너에서 해당 원소와 같거나 큰 위치 반환
    - 없다면 컨테이너의 마지막 위치 반환
  - upper_bound : 컨테이너에서 해당 원소보다 첫번째로 큰 위치 반환
    - 없다면 컨테이너의 마지막 위치 반환

binary_search, lower_bound, upper_bound

- 사용법

```cpp
#include <iostream>
#include <algrithm> // algorithm 헤더 추가
#include <vector>
#include <functional>

using namespace std;
int comp(int a,int b){
    return a < b; // 비교 함수 구현
}
int main ()
{

  int myints[] = {1,2,3,4,5,4,3,2,1};
  std::vector<int> v(myints,myints+9);                         // 1 2 3 4 5 4 3 2 1

  // using default comparison:
  std::sort (v.begin(), v.end());                               // 1 1 2 2 3 3 4 4 5

  if (std::binary_search (v.begin(), v.end(), 3))
    std::cout << "found!\n"; else std::cout << "not found.\n"; // 원소가 3이 있는지 확인

  auto lb = lower_bound(v.begin(),v.end(),3);       // 원소가 3인 주소 반환 == v[4]
  auto ub = upper_bound(v.begin(),v.end(),3);       // 원소가 3보다 큰 첫 주소 반환 == v[6]

}
```

binary_search, lower_bound, upper_bound

- 풀어봅시다
  - 1920 – 수 찾기

끝

- 대부분의 대회에서는 reference 참고를 허용하기에 참조하는것이 좋다.
- 대표 레퍼런스 주소:
  - http://www.cplusplus.com
  - https://en.cppreference.com/w/
  - etc