**HW2**

2014104272 이준영

**1. Makefile**

help:

@echo "make help"

@echo "make all"

ADT\_llist.o:ADT\_llist.c

g++ -c ADT\_llist.c

main.o:main.c

g++ -c main.c

all:main.o ADT\_llist.o

g++ -o path main.o ADT\_llist.o

run: all

./path

g++ -o create\_map create\_map.c

./create\_map

firefox gps.html

clean:

rm \*.o \*.in \*.swp \*.html path create\_map

./run.bat이 실행되지 않아서 make run 부분에 g++ ~ firefox를 넣어 create\_map.c를 컴파일하고

실행하도록 구성하였습니다.

**2. main.c**

#include "ADT\_llist.h"

#define SORT 0 //sort

#define D\_SIZE 8

int compare1(void\* x, void\* y) {

return \*((int\*)x) - \*((int\*)y);

}

void print1(void\* x){

int\* xp = (int\*)x;

printf(" - int data %d\n", \*xp);

}

void print2(void\* x){ **//double형 값을 출력하는 함수**

double\* xp = (double\*)x;

printf(" - double data %lf\n", \*xp);

}

typedef struct point{

double x;

double y;

} POINT;

void sort\_list(LLIST\* list,void\* data);

void sort\_list1(LLIST\* list, void\* data);

int main() {

**//생략**

#else

for(i=0;i<D\_SIZE;i++){

if(!add\_node\_at(list, list->count, &path[i]))

printf("data insertion failed on list!\n");

else

printf("data insertion ok on list!\n");

}

sort\_list(list,path); **// list의 node에 저장되어 있는 순서를 x좌표로 오름차순 정렬시킴**

//sort\_list1(list,path);

for(i = 0; i < D\_SIZE; i++) **{ // x좌표로 오름차순 정렬된 list에 저장된 좌표 값을 이용해 node로 생성**

POINT\* p = (POINT\*)get\_data\_at(list,i);

fprintf(fnode, "%lf %lf\n", p->x, p->y);

}

for(i = 0; i < D\_SIZE-1; i++) **{ // node와 다음 node를 이어주는 link를 생성**

POINT\* p1 = (POINT\*)get\_data\_at(list,i);

POINT\* p2 = (POINT\*)get\_data\_at(list,i+1);

fprintf(flink, "%lf %lf %lf %lf\n",p1->x, p1->y, p2->x, p2->y);

}

#endif

return 0;

}

void sort\_list(LLIST\* list,void\* data){

int i, j;

void\* standard; **//기준으로 사용되는 변수**

void\* compare; **//비교하기 위해 사용되는 변수**

void\* check; **//list안에 있는 주소 값과 sort\_list의 data의 주소 값이 일치하는지 확인하기 위한 변수**

POINT\* path = (POINT\*)data;

for(i=0;i<(list->count);i++){

check = get\_data\_at(list,i);

**//list의 i번째에 저장되어 있는 주소 값과 path의 i번째 주소 값이 일치하지 않으면 정렬X**

if(check != &path[i]){

printf("sort\_list Error: data mismatch\n");

break;

}

for(i=0;i<(list->count)-1;i++){

for(j = i+1; j<(list->count); j++){

standard = get\_data\_at(list,i); **//기준인 standard에 list의 i번째 값을 저장**

compare = get\_data\_at(list,j); **//비교대상인 compare에 list의 j번째 값을 저장**

**//standard의 x좌표값이 compare의 x좌표값보다 큰 경우 node의 위치를 바꿔 오름차순 정렬**

if(((POINT\*)standard)->x > ((POINT\*)compare)->x){

del\_node\_at(list,j);

add\_node\_at(list,j,standard);

del\_node\_at(list,i);

add\_node\_at(list,i,compare);

}

}

}

}

//print\_all(list,list->front); **//코드를 짜면서 정렬이 되었는지 확인하기 위해 사용**

}

**//처음에 짜본 코드로 path에 저장된 값 자체를 x좌표로 비교해서 정렬시키는 방식으로 만듦**

**//list에 저장되어 있는 값은 path의 주소 값이기 때문에 path의 값 자체를 바꾸면**

**//자동으로 list안에 있는 값들도 정렬되는 포인터 성질을 이용**

void sort\_list1(LLIST\* list,void\* data){

int i,j;

POINT temp;

POINT\* path = (POINT\*)data;

for(i=0;i<(list->count)-1;i++){

for(j = i+1; j<(list->count); j++){

**//i번째 x좌표값이 j번째 x좌표값 보다 크면 값을 바꿔 오름차순 정렬시킴**

if(path[i].x > path[j].x){

temp = path[i];

path[i] = path[j];

path[j] = temp;

}

}

}

//print\_all(list,list->front**); //코드를 짜면서 정렬이 되었는지 확인하기 위해 사용**

}

**<추가설명>**

처음에 짜본 void sort\_list1(LLIST\* list, void\* data) 함수는 path 배열의 값 자체를 x좌표의 오름차순으로 정렬시키면 list에 저장되어 있는 값은 path 배열을 가리키는 주소 값이기 때문에 수정없이 자동으로 정렬되는 포인터 성질을 이용해서 구성해봤습니다.

하지만, POINT path에 저장되어 있는 값의 순서 자체가 정렬되기 때문에 만약 기존 순서의 path를 사용해야하는 경우 문제가 발생하기 때문에 좀 더 일반화된 경우를 위해 void sort\_list(LLIST\* list, void\* data) 함수를 만들었습니다.

void sort\_list(LLIST\* list, void\* data) 함수의 경우 처음에는 아래와 같이 코드를 구성했습니다.

void sort\_list(LLIST\* list){

int i, j;

void\* standard; //기준으로 사용되는 변수

void\* compare;

for(i=0;i<(list->count)-1;i++){

for(j = i+1; j<(list->count); j++){

standard = get\_data\_at(list,i);

compare = get\_data\_at(list,j);

if(((POINT\*)standard)->x > ((POINT\*)compare)->x){

del\_node\_at(list,j);

add\_node\_at(list,j,standard);

del\_node\_at(list,i);

add\_node\_at(list,i,compare);

}

}

}

}

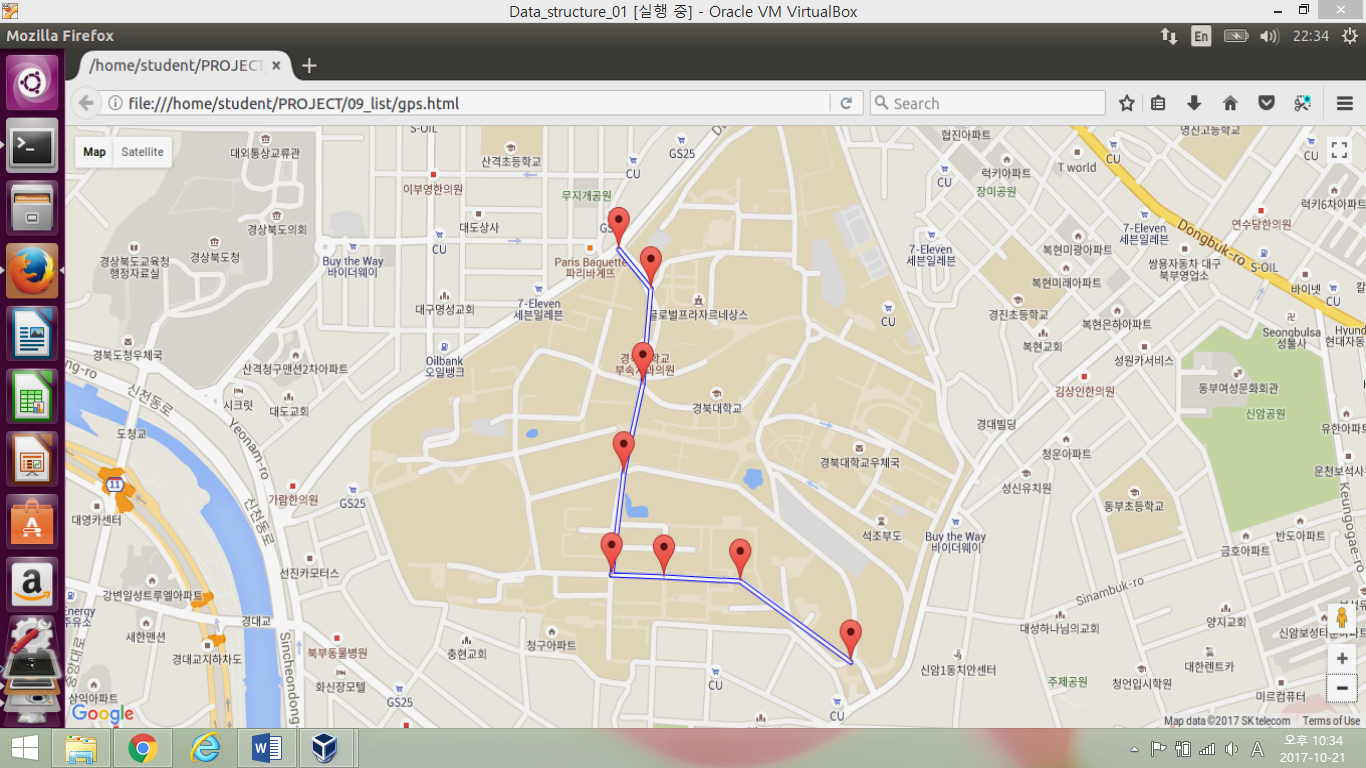
이 함수의 경우 main문에서 list에 이미 path의 주소 값을 저장해 놔서 void\* data 변수가 필요 없다고 생각하여 제거하고 list의 node에 저장되어 있는 path의 x좌표값을 비교해서 오름차순 정렬되도록 구성 했습니다.

그러다가 list와 data를 둘 다 써서 좀 더 일반화된 코드를 구성하기 위해 고민한 결과 본문에 짠 코드와 같이 check를 선언하여 list의 node에 저장되어 있는 주소 값들이 sort\_list 함수에 넣은 data의 주소 값이 일치하는지 확인하는 과정을 추가하였습니다.

if문을 이용하여 두 주소 값이 일치하지 않는 경우 break를 통해 for문에서 빠져나와 정렬이 되지않고 "sort\_list Error: data mismatch"가 출력되도록 설정했습니다.

일치하는 경우에는 standard에 list의 i번째 노드의 값을 넣어주고 compare에 j번째 노드의 값을 넣은 다음 x좌표 값을 비교해서 standard의 x좌표가 compare의 x좌표보다 큰 경우 i번째 노드와 j번째 노드를 바꿔주어 오름차순 정렬시켰습니다.

**3. 결과**



**4. 느낀점**

맨 처음 주어진 코드를 보면서 손으로 구조도 그려보고 흐름을 적으면서 구조를 파악하는데 시간이 오래 걸렸습니다. 특히 POINT path가 배열형태여서 path의 특정 위치에 있는 좌표의 주소와 x값을 얻고 비교하는 것이 익숙하지않아서 어려움을 겪었지만, 이 과정 덕분에 포인터에 대한 공부가 많이 되었습니다.

또한, sort\_list 함수를 구성할 때 2가지 방법으로 짜보았는데 list와 data를 각각 하나씩만 활용하였기 때문에 교수님이 보여준 예시처럼 list와 data를 둘 다 활용하는 코드를 짜고 싶어서 시간을 많이 투자했습니다. 이때 코드를 여러가지 짜보면서 고민한 것이 자료구조 공부에 많은 도움이 되었습니다.