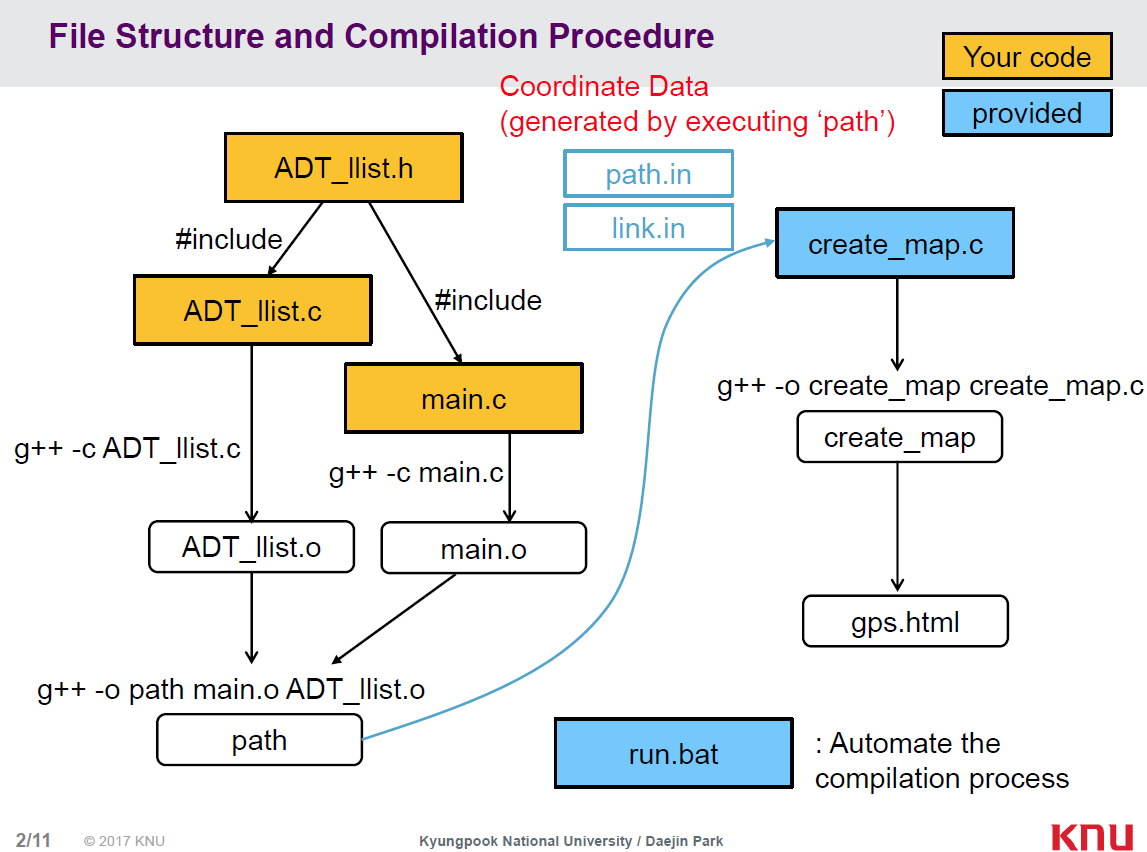
1. **HW2 Report**
2. **-Sorting with linked list-**

**2013104352 한진희**

**File Structure and Compilation Procedure**



|  |  |  |
| --- | --- | --- |
| 현재 결과 |  | 목표 결과 |
|  | ⤳  sort |  |

**1. Makefile (양식에 맞게 수정)**

|  |
| --- |
| 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: ADT\_llist.o main.o  g++ -o path main.o ADT\_llist.o  run: all  ./path  clean:  rm \*.o \*.in \*.swp \*.html path create\_map |

**2. ADT\_llist.h, c (기존 수업시간에 배운대로)**

|  |
| --- |
| ADT\_llist.h |
| #ifndef ADT\_LLIST  #define ADT\_LLIST  #include <stdio.h>  #include <stdlib.h>  typedef struct node {  void\* data\_ptr;  struct node\* next;  }NODE;  typedef struct{  int count;  NODE\* front;  NODE\* rear;  NODE\* pos;  }LLIST;  LLIST\* create\_list();  bool add\_node\_at(LLIST\* list,unsigned int index,void\* in);  bool del\_node\_at(LLIST\* list,unsigned int index);  void\* get\_data\_at(LLIST\* list,unsigned int index);  int find\_data(LLIST\* ptr\_list, void\* search\_data);  #endif |
| ADT\_llist.c |
| #include "ADT\_llist.h"  LLIST\* create\_list(){  LLIST\* list =(LLIST\*)malloc(sizeof(LLIST));  if(!list)  return NULL;  list->count =0;  list->front =NULL;  list->rear =NULL;  list->pos = NULL;  return list;  }  bool add\_node\_at(LLIST\* list,unsigned int index,void\* in){  if(index>(list->count))  return false;  NODE\* node =(NODE\*)malloc(sizeof(NODE));  if(node == NULL)  return false;  node->data\_ptr =in;  node->next = NULL;    if(list->count==0){  list->front =node;  list->rear =node;  (list->count)++;  return true;  }    int iter\_i=0;    if(index==0){  node->next=list->front;  list->front=node;  (list->count)++;  return true;  }    iter\_i++;  list->pos=list->front;    while(iter\_i!=index){  list->pos=list->pos->next;  iter\_i++;  }    if(iter\_i==list->count){  list->pos->next=node;  list->rear=node;  (list->count)++;  return true;  }  else{  node->next=list->pos->next;  list->pos->next=node;  (list->count)++;  return true;  }  }  bool del\_node\_at(LLIST\* list,unsigned int index){  if(list->count==0)  return false;  else if(index>=list->count)  return false;  if(list->count==1){  free(list->front);  list->front=NULL;  list->rear=NULL;  list->count=0;  return true;  }  int iter\_i=0;  list->pos = list->front;  NODE\* pre = NULL;  while(iter\_i!=index){  pre=list->pos;  list->pos=list->pos->next;  iter\_i++;  }  if(index==0){  list->front=list->pos->next;  free(list->pos);  list->pos=NULL;  (list->count)--;  }  if(index==(list->count-1)){  list->rear=pre;  pre->next=NULL;  free(list->pos);  list->pos = NULL;  (list->count)--;  return true;  }  else{  pre->next=list->pos->next;  free(list->pos);  list->pos=NULL;  (list->count)--;  return true;  }  return false;  }  void\* get\_data\_at(LLIST\* list,unsigned int index){  if(index>=list->count)  return NULL;  list->pos=list->front;  int iter\_i=0;  while(list->pos!=NULL){  if(iter\_i==index)  return list->pos->data\_ptr;  list->pos=list->pos->next;  iter\_i++;  }  return NULL;  }  int find\_data(LLIST\* list, void\* search\_data){  list->pos = list->front;  int cmp\_result, left, right;  int iter\_i=0;  while(list->pos!=NULL){  left=\*(int\*)(list->pos->data\_ptr);  right=\*(int\*)search\_data;  cmp\_result=left-right;  if(cmp\_result==0)  return iter\_i;    list->pos=list->pos->next;  iter\_i++;  }  return -1;  } |

**3-1. 기존의 main.c**

|  |  |
| --- | --- |
| <실행결과> | <해석> |
|  | 좌표 8개를 사용하기 위해 D\_SIZE = 8 정의  2개의 데이터를 정수값으로 비교하여 차를 나타내는 함수 compare1  데이터를 정수값으로 출력하는 함수print1  실수 x, y 2개의 변수를 가지는 POINT형 선언  리스트의 데이터를 이용하여 정렬하는 sort\_list함수 선언  ------------------------------------main------------------------------------  지도에서 좌표를 나타내는 fnode, 좌표사이 의 길을 나타내는 flink 선언  리스트를 하나 생성  POINT형인 path[8]을 생성하여 8개의 (x, y) 데이터를 넣음.  -----------------변수 SORT를 선언하지 않으면 실행하는 코드-----------------  for문을 이용한 (path[1].x,path[1])부터 (path[8].x,path[8])까지 fnode, 좌표사이의 flink를 지도에 표시 |

**3-2. 기존의 main.c 분석**

|  |
| --- |
|  |
| <시험 출력> |

|  |  |
| --- | --- |
| <실행 결과> | |
| (35.88,128.60) | (35.88,128.61) |
|  |  |
| (35.89,128.60) | (35.89,128.61) |
|  |  |

x가 커질수록 지도상에선 북쪽으로 움직이고 y가 커질수록 지도상에선 동쪽으로 움직임을 알 수 있다.

**4. 계획 및 결과**

랜덤으로 주어진 지도들의 좌표들 중에 북문에서 정문으로 가는 최적경로는 좌표의 x 성분을 기준으로 내림차순으로 정렬하여 나타내야 한다.

|  |  |
| --- | --- |
| 변수 SORT를 정의했을 때 실행하는 코드 | |
| #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);  for(i = 0; i < D\_SIZE; i++) {  fprintf(fnode, "%lf %lf\n", (\*(POINT\*)get\_data\_at(list,i)).x, (\*(POINT\*)get\_data\_at(list,i)).y);  }  for(i = 0; i < D\_SIZE-1; i++) {  fprintf(flink, "%lf %lf %lf %lf\n",  (\*(POINT\*)get\_data\_at(list,i)).x,(\*(POINT\*)get\_data\_at(list,i)).y,(\*(POINT\*)get\_data\_at(list,i+1)).x,(\*(POINT\*)get\_data\_at(list,i+1)).y);  }  for(i = 0; i < D\_SIZE; i++) {  printf("%lf %lf\n", (\*(POINT\*)get\_data\_at(list,i)).x,(\*(POINT\*)get\_data\_at(list,i)).y);  }    #endif  return 0;  }  void sort\_list(LLIST\* list, void\* data) {  int i,j;  void\* stand;  void\* walk;  void\* tmp;  POINT\* path = (POINT\*) data;  for(i = 0; i < (list->count)-1; i++){  for(j=i+1; j<(list->count); j++){  stand = get\_data\_at(list,i);  walk = get\_data\_at(list,j);  if((\*(POINT\*)stand).x < (\*(POINT\*)walk).x){  tmp = walk;  add\_node\_at(list,i,tmp);  del\_node\_at(list,j+1);  tmp = stand;  add\_node\_at(list,j+1,tmp);  del\_node\_at(list,i+1);  }  }  }  } | |
| <해석> | |
| 기존에 있던 path성분들을 순서 상관없이 list에 넣는다.(add\_node\_at 이용)  (add\_node\_at 함수 오류 시 오류출력)  sort\_list 함수 사용(list의 data가 x성분을 기준으로 내림차순이 되게 정렬)  for문을 이용하여 list의 정렬된 data들을 차례대로 꺼내서 좌표로 지도에 출력(fprint, fnode, get\_data\_at 사용)  for문을 이용하여 list의 정렬된 data들을 차례대로 꺼내서 경로로 지도에 출력(fprint, flink, get\_data\_at 사용)  for문을 이용하여 list의 정렬된 data들을 차례대로 꺼내어 터미널에 출력(print, get\_data\_at 사용)  -------------------------------------------sort\_list함수 선언-------------------------------------------  리스트와 데이터를 입력으로 받는 sort\_list 함수  for문을 이용하기 위해 미리 정수 I, j를 선언  list의 node들을 재배치하기 위한 그릇(?) stand, walk, tmp 선언  입력 data를 path로 옮김  [과제자료에 올려진 ‘selection sort concept in array’를 참고하여,  첫 번째 노드를 기준으로 나머지 노드들과 x성분을 비교하여 나머지 노드들 중 x성분이 더 크면 자리를 바꾸는 방법으로 맨 끝 노드까지 기준이 옮겨가서 이 과정이 끝나면 x성분에 대하여 내림차순으로 정렬되게 설계한다.]  I=0부터 (list->count)-1까지, j=i+1부터 (list->count)까지 중첩으로 반복문을 만들어 기준 노드(stand)와 기준 다음에 있는 노드(walk)들과 x성분을 비교하여 기준 노드의 x성분이 작으면 walk 노드를 tmp 노드로 넣고 이 노드를 stand 노드 위치(i번째)에 넣고(add) 원래 있던 walk 노드를 제거한다.(j+1번째)(del). 이로써 walk 노드가 I번째로 이동했다. 또한 stand 노드를 walk 노드의 자리로 이동시키기 위해 stand 노드를 tmp 노드에 넣고 원래 walk 노드 자리(j+1)에 넣는다. 또한 원래 있던 stand 노드(i+1)를 제거한다. 이 과정을 반복하여 정렬한다. | |
| <시험 출력> | <실행 결과> | |
|  |  | |
| path의 성분을 x의 내림차순으로 정렬 | x의 내림차순 기준으로 한 list를 fprint를 이용하여 지도에 출력 | |

**5. 고찰**

처음에 ADT\_llist.c, h 파일을 작성하고 실행을 하였는데 run.bat 파일이 실행이 안돼서 코드에서 오류를 찾았었는데 알고 보니 run.bat 파일이 잠겨(?) 있었습니다.

공지사항을 보고 해결한 뒤 실행했는데 처음에 C언어를 이용하여 지도에 좌표랑 경로를 표시도 할 수 있다는 것을 보고 놀랐습니다.

sort\_list 함수를 정의하는 부분에서 조금 어려웠지만 제시해 주신 힌트를 이용하여, 수업시간에 배운 list의 함수와 특성을 이해하고 코딩을 해보니 생각보다 쉽게 작성할 수 있었습니다.

과제를 하면서 list를 직접 사용해 원하던 결과가 나와 보람도 있고 오래 기억될 것 같습니다.