2022-2 객체지향형 프로그래밍과 자료구조 기말고사 (Exam4A)

2022. 12. 16. (금) 13:00 ~ 17:00

담당교수 김영탁

4A. STL (Standard Template Library) set 및 map 응용 프로그램 (25점)

4A.1 STL Set

- Standard Template Library (STL) set은 중복을 허용하지 않는 유일한 원소들 만을 가지는 연관 컨테이너 (associative container)이며, 지정된 Key 자료형에 따라 정렬된 상태를 유지하고, 균형 이진 트리 또는 red-black tree로 구현되어 빠른 탐색 기능을 제공함
- STL set에 포함된 원소들을 정렬된 상태로 유지하기 위하여 지정된 Key 자료형에 비교연산자 '<'의 연산자 오버로딩이 구현되어 있어야 함
- STL set을 사용하기 위해서는 #include <set>이 필요하며, set<T> 형식으로 Key의 자료형 T를 설정하게됨
- STL set 객체의 insert() 멤버 함수를 사용하여 새로운 항목을 추가하며, find() 멤버함수를 사용하여 set에 지정된 항목이 포함되어 있는지를 확인. 만약 지정된 항목이 포함되어 있으면, 그 위치에 해당하는 interator가 반환됨. 만약 지정된 항목이 포함되어 있지 않으면 해당 set의 .end()가 반환됨.
- STL set에 포함된 항목들은 set의 iterator를 사용하여 차례대로 열거할 수 있음

4A.2 STL map

- STL map은 key와 value를 쌍 (pair)로 구성하여 1:1 매핑을 시켜줌. **make_pair**(key, value)함수를 사용하여 key와 value의 쌍을 만들 수 있음
- STL map의 **insert**() 함수에서는 **make_pair**(key, value)함수를 사용하여 생성된 key-value의 쌍을 새로운 항목으로 추가
- STL map의 .find(key) 멤버함수를 사용하여 key에 해당하는 value를 반환받음
- STL map에 포함된 항목들은 map의 iterator를 사용하여 차례대로 열거할 수 있음

4A.3 class CityPair, InterCityDist.h

- 클래스 CityPair 인스탄스는 2개의 도시 이름을 포함하며, 이를 STL map의 key로 사용할 수 있게 함
- class CityPair와 관련 멤버 함수를 InterCityDist.h 파일에 구현할 것

```
/* InterCityDist.h */
. . . . // 필요한 헤더파일 포함, 전처리기 선언
class CityPair
{
public:
    CityPair() {};
    CityPair(string c1, string c2) :city1_name(c1), city2_name(c2) {      };
    string getC1() const { return city1_name; }
    string getC2() const { return city2_name; }
    bool operator<(const CityPair other) const {
        if (city1_name < other.city1_name)
            return true;
        else if ((city1_name == other.city1_name) && (city2_name < other.city2_name))
            return true;
        else
            return false;
    }
private:
    string city1_name;
    string city2_name;
};
void fget_ICD(string fname, map<CityPair, double>* pMap_ICD, set<string>& city_names);
void print_set(set<string>& s);
void print_ICD_Map(map<CityPair, double>& map_ICD);
void print_ICD_Table(set<string>& city_names, map<CityPair, double>& map_ICD);
```

4A.4 InterCityDist (ICD) 관련 함수 (InterCityDist.cpp에 포함)

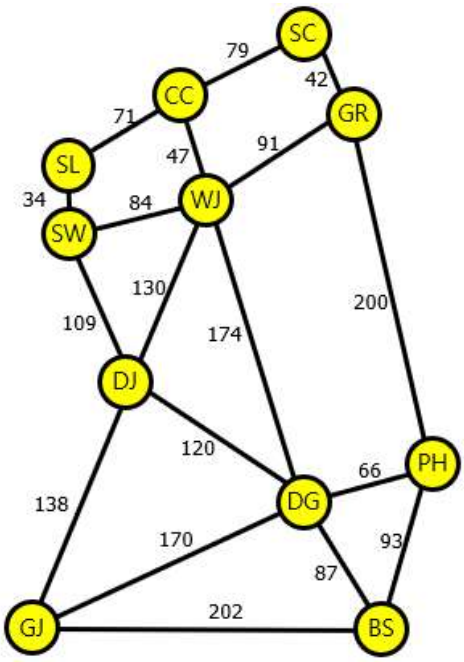
- 함수 fget_ICD(string fname, map<CityPair, double>* pMap_ICD, set<string>& city_names)는 인수로 전달된 파일이름 fname으로 지정된 파일로부터 도시간의 거리 정보를 (city1_nm, city2_nm, dist) 차례대로 읽고, 입력된 두도시를 CityPair로 구성한 후, 그 도시들간 거리를 make_pair() 함수를 사용하여 쌍으로 생성한 후, map에 저장하고, 도시 이름을 set인 city_names에 저장하여 주는 기능을 제공하도록 구현 하라. 도시의 이름은 string 자료형으로 도시 간의 거리는 double 자료형으로 관리한다.
- 함수 print_set()은 인수로 전달된 set의 항목들을 차례대로 출력하도록 구현하라.
- 함수 print_ICD_Map()은 인수로 전달된 ICD_Map의 항목들을 차례대로 출력하도록 구현하라.
- 함수 print_ICD_Table(set<string>& city_names, map<CityPair, double>& map_ICD)은 인수로 전달된 city_names 집합에 포함된 각 도시들 간의 거리를 표 형식으로 출력하도록 구현하라. 각 도시들간의 거리는 map_ICD로부터 찾도록 하며, 소숫점 이하 2자리 까지 출력하도록 하라. 만약 두 도시가 직접 연결되는 도로가 없는 경우 “+oo”를 출력할 것.
- 위 함수들의 함수 원형을 InterCityDist.h에 포함하라.

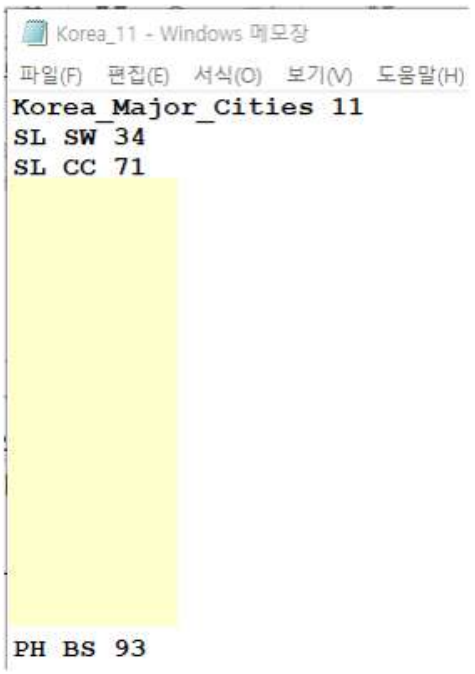
4A.5 STL Map 기능 시험 프로그램

- 11개의 한국 주요 도시간 거리 정보를 포함한 텍스트 파일 Korea_11.txt를 준비하라. (주요 도시간의 거리는 아래의 그래프에서 제공)
- 입력 텍스트 파일의 첫 줄에는 거리 정보 그래프의 이름과 도시 개수를 설정 (예: Korea_Major_Cities 11)
- map_KR_icd와 major_KR_cities를 준비
- fget_ICD() 함수를 사용하여 Korea_11.txt로부터 주요 도시간 거리 정보를 읽고, map_KR_icd에 추가. 각 도시 이름은 major_KR_cities에 추가
- print_set() 함수를 사용하여 major_KR_cities에 포함된 도시들을 출력
- print_ICD_map() 함수를 사용하여 map_KR_icd에 포함된 정보를 출력
- print_ICD_Table() 함수를 사용하여 major_KR_cities에 포함된 도시들 간의 거리를 표 형식으로 출력. 두 도시간의 거리 (distance) 정보는 map_KR_icd에서 도시 이름 쌍 (CityPair)으로 검색할 수 있도록 구현.

```
/* InterCityDist_Map.cpp */

#include <iostream>
#include <map>
#include <set>
#include "InterCityDist.h"
using namespace std;

int main()
{
	map<CityPair, double> map_KR_icd;
	set<string> major_KR_cities;

	cout << "2022-2 객체지향프로그래밍과 자료구조 학번:00000000 이름: 홍00" << endl;
	fget_ICD("Korea_11.txt", &map_KR_icd, major_KR_cities);

	cout << "Korean Major Cities = ";
	print_set(major_KR_cities);
	print_ICD_Map(map_KR_icd);
	print_ICD_Table(major_KR_cities, map_KR_icd);

	return 0;
}
```

4A.6 한국의 11개 도시간 거리 정보, 입력데이터 파일 및 화면 출력 (예시)






```
2022-2 객체지향프로그래밍과 자료구조 학번:00000000 이름: 홍00
Korean Major Cities = [BS, CC, DG, DJ, GJ, GR, PH, SC, SL, SW, WJ]

InterCityDistance Map =
(BS, DG) : 87.00
(BS, GJ) : 202.00
(BS, PH) : 93.00
(CC, SC) : 79.00
(CC, SL) : 71.00
(CC, WJ) : 47.00
(DG, BS) : 87.00
(DG, DJ) : 120.00
(DG, GJ) : 170.00
(DG, PH) : 66.00
(DG, WJ) : 174.00
(DJ, DG) : 120.00
(DJ, GJ) : 138.00
(DJ, SW) : 109.00
(DJ, WJ) : 130.00
(GJ, BS) : 202.00
(GJ, DG) : 170.00
(GJ, DJ) : 138.00
(GR, PH) : 200.00
(GR, SC) : 42.00
(GR, WJ) : 91.00
(PH, BS) : 93.00
(PH, DG) : 66.00
(PH, GR) : 200.00
(SC, CC) : 79.00
(SC, GR) : 42.00
(SL, CC) : 71.00
(SL, SW) : 34.00
(SW, DJ) : 109.00
(SW, SL) : 34.00
(SW, WJ) : 84.00
(WJ, CC) : 47.00
(WJ, DG) : 174.00
(WJ, DJ) : 130.00
(WJ, GR) : 91.00
(WJ, SW) : 84.00

InterCityDistance Table =
          |      BS      CC      DG      DJ      GJ      GR      PH      SC      SL      SW      WJ
----------+-----------------------------------------------------------------------------------------
       BS |    0.00    +oo   87.00    +oo  202.00    +oo   93.00    +oo    +oo    +oo    +oo
       CC |     +oo    0.00    +oo    +oo    +oo    +oo    +oo   79.00   71.00    +oo   47.00
       DG |   87.00    +oo    0.00  120.00  170.00    +oo   66.00    +oo    +oo    +oo  174.00
       DJ |     +oo    +oo  120.00    0.00  138.00    +oo    +oo    +oo    +oo  109.00  130.00
       GJ |  202.00    +oo  170.00  138.00    0.00    +oo    +oo    +oo    +oo    +oo    +oo
       GR |     +oo    +oo    +oo    +oo    +oo    0.00  200.00   42.00    +oo    +oo   91.00
       PH |   93.00    +oo   66.00    +oo    +oo  200.00    0.00    +oo    +oo    +oo    +oo
       SC |     +oo   79.00    +oo    +oo    +oo   42.00    +oo    0.00    +oo    +oo    +oo
       SL |     +oo   71.00    +oo    +oo    +oo    +oo    +oo    +oo    0.00   34.00    +oo
       SW |     +oo    +oo    +oo  109.00    +oo    +oo    +oo    +oo   34.00    0.00   84.00
       WJ |     +oo   47.00  174.00  130.00    +oo   91.00    +oo    +oo    +oo   84.00    0.00
```

4A.7 결과물 제출

- 바탕화면의 Exam4 폴더를 생성 후 Exam4A 프로젝트를 생성
- 압축 파일 내에 포함사항 : 작성한 Visual Studio 프로젝트, 입력 데이터 파일 (Korea_11.txt), 실행결과 Capture(채점 시 정확한 실행 유무를 확인하기 위함)
- 실행 화면 캡쳐파일은 각 시험 섹션별로 프로젝트 폴더 내에 저장 후 시험 섹션별 폴더별로 압축
- 제출시 .vs 폴더는 삭제 후 문제별 폴더를 압축하여 제출

2022-2 객체지향형 프로그래밍과 자료구조 기말고사 (Exam4B)

2022. 12. 16. (금)
담당교수 김영탁

4B. 예측 구문 (predictive text) 탐색을 위한 trie 자료구조 구현 (25점)

4B.1 class TrieNode_pStr

- 클래스인 class TrieNode_pStr은 데이터 멤버로 문자형 keyChar와 string pointer의 value를 가지며, trie 구조를 구성하기 위한 4개의 포인터를 가짐

```
class TrieNode_pStr
{
public:
    TrieNode() {} // default constructor
    TrieNode(char k, string* v) {  . . . . .  }
    void setKeyChar(char k) {  . . . . .  }
    void setValue(string* v) {  . . . . .  }
    void setNext(TrieNode_pStr *nxt) {  . . . . .  }
    void setPrev(TrieNode_pStr *pv) {  . . . . .  }
    void setParent(TrieNode_pStr *pr) {  . . . . .  }
    void setChild(TrieNode_pStr *chld) {  . . . . .  }
    char getKeyChar() {  . . . . .  }
    string* getValue() {  . . . . .  }
    TrieNode_pStr *getPrev() { return prev; }
    TrieNode_pStr *getNext() { return next; }
    TrieNode_pStr *getParent() { return parent; }
    TrieNode_pStr *getChild() { return child; }
    void _fprint(ostream& fout, TrieNode_pStr *pTN, int indent);
private:
    char keyChar;
    string* value;
    TrieNode_pStr *prev;
    TrieNode_pStr *next;
    TrieNode_pStr *parent;
    TrieNode_pStr *child;
};
```

4B.2 class Trie_pStr

- class Trie_pStr은 root TrieNode를 가리키는 포인터 _root, 전체 키워드 개수인 num_keyWords, 문자열 자료형인 trie_name을 데이터 멤버로 가짐
- class Trie_pStr은 다음과 같은 멤버함수들을 가짐 (deleteKeyStr()과 eraseTrie()는 구현대상이 아님)

```
class Trie_pStr
{
public:
    Trie(string name); // constructor
    int size() { return num_keys; }
    string getName() { return trie_name; }
    void insert(string keyStr, string* value);
    void insertExternalTN(TrieNode_pStr *pTN, string keyStr, string* value);
    TrieNode_pStr *find(string keyStr);
    void findPrefixMatch(string prefix, List_pStr& predictVocas);
    //void deleteKeyStr(string keyStr);
    //void eraseTrie();
    void fprintTrie(ostream& fout);
protected:
    TrieNode_pStr *  find(string keyStr, SearchMode sm=FULL_MATCH);
    void _traverse(TrieNode_pStr *pTN, List_pStr& list_keywords);
private:
    TrieNode_pStr *_root; // _root trie node
    int num_keys;
    string trie_name;
};
```

4B.3 main 함수

- main() 함수는 클래스 Trie_pStr를 사용하여 trie_PredictText를 생성함
- main() 함수의 첫 번째 for-loop에서는 미리 생성된 sampleTexts[] 배열의 문자열을 trie_PredictText에 차례로 insert시킨 후, fprintTrie() 멤버 함수를 사용하여 trie_PredictText의 내용을 출력시킴
- main() 함수는 testPrefix[] 배열에 포함된 접두어 (prefix)로 시작하는 단어를 findPrefixMatch() 멤버함수를 사용하여 trie_PredictText로부터 탐색하고, 그 결과를 출력함

```
/* main_trie.cpp */

#include <iostream>
#include <fstream>
#include <list>
#include <string>
#include "Trie.h"
#include "TrieNode.h"

using namespace std;
#define NUM_SAMPLE_TEXTS 100
#define NUM_TEST_PREFIX 3

void main()
{
	Trie_pStr trie_PredictText("Trie_PredictiveText");
	TrieNode_pStr *pTN;
	int word_count;
	string keyStr, prefixStr;
	List_pStr predictTexts;
	List_pStr_Iter itr;

	/* Testing Basic Operation in trie */
	string sample_texts[] =
	{
		"abcd", "abc", "ab", "a", "alpha", "WXYZ", "WXY", "WX", "W", "bcd", "bc", "binary", "bit", "bravo",
		"-1"
	};
	cout << "Inserting keywords (texts) into " << trie_PredictText.getName() << " ..... " << endl;


	for (int i=0; i<NUM_SAMPLE_TEXTS; i++)
	{
		keyStr = sample_texts[i];
		if (keyStr == "-1")
			break;
		trie_PredictText.insert(keyStr, &sample_texts[i]);
	}
	trie_PredictText.fprintTrie(cout);

	/* testing keyWord search in trie */
	string testPrefix[NUM_TEST_PREFIX] = { "W", "a", "b" };
	for (int i = 0; i < NUM_TEST_PREFIX; i++)
	{
		predictTexts.clear();
		prefixStr = testPrefix[i];
		trie_PredictText.findPrefixMatch(prefixStr, predictTexts);
		cout << "Predictive wors that starts with with (" << prefixStr << ") : " ;
		itr = predictTexts.begin();
		for (int i = 0; i < predictTexts.size(); i++)
		{
			keyStr = *(*itr);
			cout << keyStr << ", ";
			++itr;
		}
		cout << endl;
	}
}
```

4B.4 main() 함수의 실행 결과

(1) trie에 입력이 완료된 결과:

```
2022-2 객체지향 프로그래밍과 자료구조 기말고사 학번:00000000 성명:홍00
Inserting keywords (texts) into Trie_PredictiveText .....
trie ( Trie_PredictiveText) with 14 trie_nodes

W
 X
  Y
   Z
a
 b
  c
   d
  lpha
bc
  d
 inary
  t
 ravo
```

(2) 주어진 prefix로 시작하는 예측구문의 탐색 결과 (20점)

```
Predictive words that starts with with (W) : W, WX, WXY, WXYZ,
Predictive words that starts with with (a) : a, ab, abc, abcd, alpha,
Predictive words that starts with with (b) : bc, bcd, binary, bit, bravo,
```

4B.5 결과물 제출

- 바탕화면의 Exam4 폴더를 생성 후 Exam4B 프로젝트를 생성
- 압축 파일 내에 포함사항 : 작성한 프로젝트, 실행결과 Capture(채점 시 정확한 실행 유무를 확인하기 위함)
- 실행 화면 캡쳐파일은 각 시험 섹션별로 프로젝트 폴더 내에 저장 후 시험 섹션별 폴더별로 압축
- 제출시 .vs 폴더는 삭제 후 문제별 폴더를 압축하여 제출

2022-2 객체지향형 프로그래밍과 자료구조 기말고사 (Exam4C)

2022. 12. 16. (금)
담당교수 김영탁

4C. 그래프 자료구조 (25점)

4C.1 Class Graph with class Vertex and class Edge

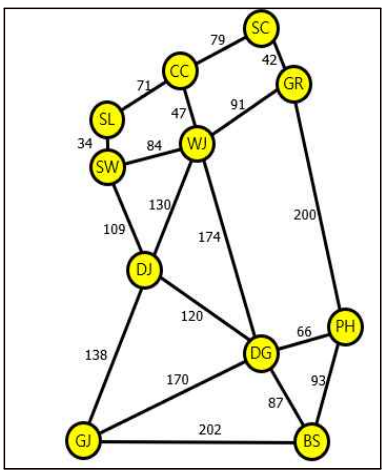
- class Graph는 그래프 자료구조를 구현하며, 내부적으로 class Vertex와 class Edge를 포함

```
/** Graph.h */
#include <list>
#include <iostream>
. . . . . // 기타 필요한 헤더파일 포함, 전처리기 설정
using namespace std;

#define PLUS_INF INT_MAX/2
enum VrtxStatus { UN_VISITED, VISITED };
enum EdgeStatus { DISCOVERY, BACK, CROSS, EDGE_UN_VISITED, EDGE_VISITED };

class Graph   // Graph based on Adjacency Matrix
{
public:
    class Vertex;
    class Edge;
    typedef std::list<Graph::Vertex> VrtxList;
    typedef std::list<Graph::Edge> EdgeList;
    typedef std::list<Graph::Vertex>::iterator VrtxItor;
    typedef std::list<Graph::Edge>::iterator EdgeItor;
public:
    class Vertex   // Graph::Vertex
    {
        friend ostream& operator<<(ostream& fout, Vertex v)
        {  . . . .      }  // Vertex (vertex_name) 형식으로 출력
    public:
        Vertex() : name(), ID(-1) {}
        Vertex(string n, int id) : name(n), ID(id) { }
        Vertex(int id) : ID(id) {}
        string getName() { return name; }
        void setName(string c_name) { name = c_name; }
        int getID() { return ID; }
        void setID(int id) { ID = id; }
        void setVrtxStatus(VrtxStatus vs) { vrtxStatus = vs; }
        VrtxStatus getvrtxStatus() { return vrtxStatus; }
        bool operator==(Vertex v) { return ((ID == v.getID()) && (name == v.getName())); }
        bool operator!=(Vertex v) { return ((ID != v.getID()) || (name != v.getName())); }
    private:
        string name;
        int ID;
        VrtxStatus vrtxStatus;
    };  // end class Vertex
public:
    class Edge   // Graph::Edge implementing weighted edge
    {
        friend ostream& operator<<(ostream& fout, Edge& e)
        {  . . . . .  } // Edge(nm1 -> nm2 : dist) 형식으로 출력
    public:
        Edge() : pVrtx_1(NULL), pVrtx_2(NULL), distance(PLUS_INF) {}
        Edge(Vertex& v1, Vertex& v2, int d) {  . . . .  }
        void endVertices(VrtxList& vrtxLst) {  . . . . . }
        Vertex opposite(Vertex v)  {  . . . . .  }
        Vertex* getpVrtx_1() { return pVrtx_1; }
        Vertex* getpVrtx_2() { return pVrtx_2; }
        int getDistance() {  . . . . .  }
        void setpVrtx_1(Vertex* pV) { pVrtx_1 = pV; }
        void setpVrtx_2(Vertex* pV) { pVrtx_2 = pV; }
        void setDistance(int d) {  . . . . .  }
```

```
        bool operator!=(Edge e) {   . . . . .   }
        bool operator==(Edge e)   { .. . . .   }
        void setEdgeStatus(EdgeStatus es) { edgeStatus = es; }
        EdgeStatus getEdgeStatus() { return edgeStatus; }
    private:
        Vertex* pVrtx_1;
        Vertex* pVrtx_2;
        double distance;
        EdgeStatus edgeStatus;
    }; // end class Edge
public:
    Graph() : name(""), pVrtxArray(NULL), pAdjLstArray(NULL) {}    // default constructor
    Graph(string nm, int num_nodes) : name (nm), pVrtxArray(NULL), pAdjLstArray(NULL)
    {   . . . . .   }
    string getName() { return name; }
    void initGraph(string nm, int num_nodes);
    void vertices(VrtxList& vrtxLst);
    void edges(EdgeList&);
    bool isAdjacentTo(Vertex v, Vertex w);
    Vertex* insertVertex(Vertex& v);
    Vertex* getVertex(string vName);
    void insertEdge(Edge& e);
    int getNumVertices() { return num_vertices; }
    void incidentEdges(Vertex v, EdgeList& edges);
    Vertex* getpVrtxArray() { return pVrtxArray; }
    EdgeList* getpAdjLstArray() { return pAdjLstArray; }
    double** getppDistMtrx();
    void initDistMtrx();    // initialize distance matrix
    void fprintDistMtrx(ostream& fout); // print distance matrix in table format
    void fprintGraph(ofstream& fout);
    bool isValidvID(int vid);
    . . . . // 기타 필요한 멤버함수는 직접 선언 및 구현하여 사용할 것
private:
    string name;
    Vertex* pVrtxArray;
    EdgeList* pAdjLstArray;
    int num_vertices;
    double** ppDistMtrx;
};

typedef Graph::Vertex Vertex;
typedef Graph::Edge Edge;
typedef Graph::VrtxList VrtxList;
typedef Graph::EdgeList EdgeList;
typedef Graph::VrtxItor VrtxItor;
typedef Graph::EdgeItor EdgeItor;

void fgetGraph(Graph* g, string fin_name);
void printPath(Vertex* pStart, Vertex* pEnd, VrtxList path);
```

4C.2 Graph 관련 함수 작성

- 주어진 파일 이름(“Korea_11.txt”)의 입력 데이터 파일로부터 그래프의 vertex와 edge 정보를 입력 받아 Graph 객체 인스탄스에 저장하는 함수 fgetGraph(Graph* g, string fin_name)를 작성하라.
- 주어진 start vertex와 end vertex간의 경로 path를 출력하는 함수 printPath(Vertex* pStart, Vertex* pEnd, VrtxList path)를 작성하라.



4C.3 Graph topology (Korean 11 cities, 18 bi-directional edges) 및 입력 데이터 파일 준비

- 다음 그래프의 vertex 및 edge 정보를 제공하는 입력 데이터 파일 (Korea_11.txt)을 준비하라
- 입력 데이터 파일의 첫줄에는 그래프 이름 (문자열)과 vertex 개수 (정수)를 제공하라. (예: Korea_Major_Cities 11)
- 다음 줄 부터는 “도시이름1 도시이름2 도시간 거리“ 의 양식으로 그래프 연결 정보를 제공할 것

4C.3 main() 함수

- main() 함수는 주어진 입력 데이터 파일 (Korea_11.txt)의 그래프 topology에 따라 vertex 배열과 edge 배열을 구성하고, 이를 그래프 객체 graph에 차례로 추가함
- graph에 포함된 그래프 노드(정점)들과 간선들을 출력하여 그래프 구성이 올바르게 수행된 것을 확인하며, fprintGraph() 함수를 사용하여 그래프의 adjacency list를 출력함.

```
/** Exam4C.cpp */

#include <iostream>
#include <fstream>
#include <string>
#include "Graph.h“
using namespace std;

void main()
{
	Graph graph;

	fgetGraph(&graph, "Korea_11.txt");

	graph.fprintGraph(cout);
	graph.initDistMtrx();
	graph.fprintDistMtrx(cout);
	cout << endl;
}
```

4C.4 main() 함수의 실행 결과

```
Initializing graph (Korea) from input data file(Korea_11.txt) ...
insert edge (   SL ->    SW :    34)
insert edge (   SL ->    CC :    71)
insert edge (   CC ->    SC :    79)
insert edge (   CC ->    WJ :    47)
insert edge (   SC ->    GR :    42)
insert edge (   SW ->    WJ :    84)
insert edge (   SW ->    DJ :   109)
insert edge (   WJ ->    GR :    91)
insert edge (   DJ ->    WJ :   130)
insert edge (   DJ ->    GJ :   138)
insert edge (   DJ ->    DG :   120)
insert edge (   WJ ->    DG :   174)
insert edge (   DG ->    PH :    66)
insert edge (   GR ->    PH :   200)
insert edge (   GJ ->    DG :   170)
insert edge (   GJ ->    BS :   202)
insert edge (   DG ->    BS :    87)
insert edge (   PH ->    BS :    93)
Total 11 vertices and 18 bi-directional edges were inserted to graph.
Korea with 11 vertices has following adjacency lists :
 vertex ( SL) :  Edge( SL ->  SW :    34)  Edge( SL ->  CC :    71)
 vertex ( SW) :  Edge( SW ->  SL :    34)  Edge( SW ->  WJ :    84)  Edge( SW ->  DJ :   109)
 vertex ( CC) :  Edge( CC ->  SL :    71)  Edge( CC ->  SC :    79)  Edge( CC ->  WJ :    47)
 vertex ( SC) :  Edge( SC ->  CC :    79)  Edge( SC ->  GR :    42)
 vertex ( WJ) :  Edge( WJ ->  CC :    47)  Edge( WJ ->  SW :    84)  Edge( WJ ->  GR :    91)  Edge( WJ ->  DJ :   130)  Edge( WJ ->  DG :   174)
 vertex ( GR) :  Edge( GR ->  SC :    42)  Edge( GR ->  WJ :    91)  Edge( GR ->  PH :   200)
 vertex ( DJ) :  Edge( DJ ->  SW :   109)  Edge( DJ ->  WJ :   130)  Edge( DJ ->  GJ :   138)  Edge( DJ ->  DG :   120)
 vertex ( GJ) :  Edge( GJ ->  DJ :   138)  Edge( GJ ->  DG :   170)  Edge( GJ ->  BS :   202)
 vertex ( DG) :  Edge( DG ->  DJ :   120)  Edge( DG ->  WJ :   174)  Edge( DG ->  PH :    66)  Edge( DG ->  GJ :   170)  Edge( DG ->  BS :    87)
 vertex ( PH) :  Edge( PH ->  DG :    66)  Edge( PH ->  GR :   200)  Edge( PH ->  BS :    93)
 vertex ( BS) :  Edge( BS ->  GJ :   202)  Edge( BS ->  DG :    87)  Edge( BS ->  PH :    93)

Distance Matrix of Graph (Korea) :
      |   SL   SW   CC   SC   WJ   GR   DJ   GJ   DG   PH   BS
------+-----------------------------------------------------------
   SL |    0   34   71  +oo  +oo  +oo  +oo  +oo  +oo  +oo  +oo
   SW |   34    0  +oo  +oo   84  +oo  109  +oo  +oo  +oo  +oo
   CC |   71  +oo    0   79   47  +oo  +oo  +oo  +oo  +oo  +oo
   SC |  +oo  +oo   79    0  +oo   42  +oo  +oo  +oo  +oo  +oo
   WJ |  +oo   84   47  +oo    0   91  130  +oo  174  +oo  +oo
   GR |  +oo  +oo  +oo   42   91    0  +oo  +oo  +oo  200  +oo
   DJ |  +oo  109  +oo  +oo  130  +oo    0  138  120  +oo  +oo
   GJ |  +oo  +oo  +oo  +oo  +oo  +oo  138    0  170  +oo  202
   DG |  +oo  +oo  +oo  +oo  174  +oo  120  170    0   66   87
   PH |  +oo  +oo  +oo  +oo  +oo  200  +oo  +oo   66    0   93
   BS |  +oo  +oo  +oo  +oo  +oo  +oo  +oo  202   87   93    0
```

4C.5 결과물 제출

- 바탕화면의 Exam4 폴더를 생성 후 Exam4C 프로젝트를 생성
- 압축 파일 내에 포함사항 : 작성한 프로젝트, 실행결과 Capture(채점 시 정확한 실행 유무를 확인하기 위함)
- 실행 화면 캡쳐파일은 각 시험 섹션별로 프로젝트 폴더 내에 저장 후 시험 섹션별 폴더별로 압축
- 제출시 .vs 폴더는 삭제 후 문제별 폴더를 압축하여 제출

2022-2 객체지향형 프로그래밍과 자료구조 기말고사 (Exam4D)

2022. 12. 16. (금)
담당교수 김영탁

4D. class BreadthFirstSearch with Dijkstra (25점)

4D.1 class BreadthFirstSearch()

- BreadthFirstSearch() 클래스는 내부에 최단 거리 경로를 탐색하여 반환하는 DijkstraShortestPath() 멤버함수를 구현

```
/** BFS_Dijkstra.h */
#include <fstream>
. . . . .
using namespace std;

class BreadthFirstSearch
{
protected:
      Graph& graph;
      bool done;       // flag of search done
protected:
      void initialize();
      bool isValidvID(int vid)   {   . . . .   }
      int getNumVertices()   {   . . . .   }
      Graph& getGraph() { return graph; }
      . . . . // 기타 필요한 멤버함수는 직접 선언 및 구현하여 사용할 것
public:
      BreadthFirstSearch(Graph& g) :graph(g)   {   . . . .   }
      void DijkstraShortestPath(ofstream& fout, Vertex& s, Vertex& t, VrtxList& path);
};
```

4D.2 main() 함수 (문제 Exam4C로부터 확장)

- test_DijkstraShortestPathSearch(Graph* pG) 함수에서 BreadthFirstSearch 클래스의 객체인 bfsGraph를 생성
- 최단 거리 경로 탐색의 시작 노드와 끝 노드의 이름을 입력 받음
- BreadthFirstSearch 클래스의 맴버함수인 DijkstraShortestPath() 멤버함수를 사용하여 지정된 시작 노드 (start node)로부터 지정된 목적지 노드 (end node)까지의 최단 거리 경로를 찾고, 이를 출력함.
- 알고리즘을 사용하여 탐색된 경로를 출력하기 위한 print_path() 함수를 사용하여 출력

```
/** main.cpp */

#include <iostream>
#include <fstream>
#include <string>
#include "Graph.h"
#include "BFS_Dijkstra.h"
using namespace std;

void test_DijkstraShortestPathSearch(Graph* pG);
void main()
{
      . . . .   // Exam4C에서 작성되었던 main() 함수의 그래프 준비 내용을 사용


      /* ========================================*/
      test_DijkstraShortestPathSearch(&graph);
}

void test_DijkstraShortestPathSearch(Graph* pG)
{
      VrtxList path;
      BreadthFirstSearch bfsGraph(*pG);
```

```
    cout << "\nTesting Breadth First Search with Dijkstra Algorithm" << endl;

    path.clear();
    string start_nm, end_nm;
    Vertex* pStart, * pEnd;
    while (1)
    {
        cout << "Input start and end of path to search shortest path (. . to quit) : ";
        cin >> start_nm >> end_nm;
        if (start_nm == ".")
            break;
        pStart = pG->getVertex(start_nm);
        pEnd = pG->getVertex(end_nm);
        if (pStart == NULL || pEnd == NULL)
        {
            cout << "Error in start or end vertex name !!" << endl;
            return;
        }
        cout << "Dijkstra Shortest Path Finding from " << pStart->getName() << " to ";
        cout << pEnd->getName() << " .... " << endl;
        bfsGraph.DijkstraShortestPath(cout, *pStart, *pEnd, path);
        cout << "Path found by DijkstraShortestPath from " << *pStart << " to " << *pEnd << " : ";
        printPath(pStart, pEnd, path);

        pEnd = pG->getVertex(start_nm);
        pStart = pG->getVertex(end_nm);
        cout << "Dijkstra Shortest Path Finding from " << pStart->getName() << " to ";
        cout << pEnd->getName() << " .... " << endl;
        bfsGraph.DijkstraShortestPath(cout, *pStart, *pEnd, path);
        cout << "Path found by DijkstraShortestPath from " << *pStart << " to " << *pEnd << " : ";
        printPath(pStart, pEnd, path);
    }
}
```

4D.3 Dijkstra's Shortest Path Search 실행 결과

```
Testing Breadth First Search with Dijkstra Algorithm
Input start and end of path to search shortest path (. . to quit) : GJ SC
Dijkstra Shortest Path Finding from GJ to SC ....
Dijkstra::Least Cost from Vertex (GJ) at each round :
           |   SL   SW   CC   SC   WJ   GR   DJ   GJ   DG   PH   BS
-----------+-----------------------------------------------------
round [ 1] |  +oo  247  +oo  +oo  268  +oo  138    0  170  +oo  202  ==> selected vertex : DJ
round [ 2] |  +oo  247  +oo  +oo  268  +oo  138    0  170  236  202  ==> selected vertex : DG
round [ 3] |  +oo  247  +oo  +oo  268  +oo  138    0  170  236  202  ==> selected vertex : BS
round [ 4] |  +oo  247  +oo  +oo  268  436  138    0  170  236  202  ==> selected vertex : PH
round [ 5] |  281  247  +oo  +oo  268  436  138    0  170  236  202  ==> selected vertex : SW
round [ 6] |  281  247  315  +oo  268  359  138    0  170  236  202  ==> selected vertex : WJ
round [ 7] |  281  247  315  +oo  268  359  138    0  170  236  202  ==> selected vertex : SL
round [ 8] |  281  247  315  394  268  359  138    0  170  236  202  ==> selected vertex : CC
round [ 9] |  281  247  315  394  268  359  138    0  170  236  202  ==> selected vertex : GR
round [10] |
reached to the target node (SC) at Least Cost = 394
Path found by DijkstraShortestPath from GJ to SC : Path found (GJ => SC) : GJ -> DJ -> WJ -> CC -> SC
Dijkstra Shortest Path Finding from SC to GJ ....
Dijkstra::Least Cost from Vertex (SC) at each round :
           |   SL   SW   CC   SC   WJ   GR   DJ   GJ   DG   PH   BS
-----------+-----------------------------------------------------
round [ 1] |  +oo  +oo   79    0  133   42  +oo  +oo  +oo  242  +oo  ==> selected vertex : GR
round [ 2] |  150  +oo   79    0  126   42  +oo  +oo  +oo  242  +oo  ==> selected vertex : CC
round [ 3] |  150  210   79    0  126   42  256  +oo  300  242  +oo  ==> selected vertex : WJ
round [ 4] |  150  184   79    0  126   42  256  +oo  300  242  +oo  ==> selected vertex : SL
round [ 5] |  150  184   79    0  126   42  256  +oo  300  242  +oo  ==> selected vertex : SW
round [ 6] |  150  184   79    0  126   42  256  +oo  300  242  335  ==> selected vertex : PH
round [ 7] |  150  184   79    0  126   42  256  394  300  242  335  ==> selected vertex : DJ
round [ 8] |  150  184   79    0  126   42  256  394  300  242  335  ==> selected vertex : DG
round [ 9] |  150  184   79    0  126   42  256  394  300  242  335  ==> selected vertex : BS
round [10] |
reached to the target node (GJ) at Least Cost = 394
Path found by DijkstraShortestPath from SC to GJ : Path found (SC => GJ) : SC -> CC -> WJ -> DJ -> GJ
```

4D.4 결과물 제출

- 바탕화면의 Exam4 폴더를 생성 후 Exam4D 프로젝트를 생성
- 압축 파일 내에 포함사항 : 작성한 프로젝트, 실행결과 Capture(채점 시 정확한 실행 유무를 확인하기 위함)
- 실행 화면 캡쳐파일은 각 시험 섹션별로 프로젝트 폴더 내에 저장 후 시험 섹션별 폴더별로 압축
- 제출시 .vs 폴더는 삭제 후 문제별 폴더를 압축하여 제출