DLL	**Description** Double Linked List 구현
InfixPostfix	**Description** 수식을 입력 받고 수식을 Postfix로 변환하여 변환된 식을 출력, 연산 **Algorithm** 스택을 이용해 Postfix로 변환한다. 입력값 중 '$' 입력되면 입력중단,반복문을 빠져나온다. **Function** postfix - infix form을 Postifix로 변환하는 함수 precedence get_token - 입력받은 토큰을 읽고 반환하는 함수 calc - 주어진 수식을 연산하는 함수
Lab1-2	**Description** 데이터 파일에서 한 단어씩 읽고 단어의 개수를 출력하는 프로그램 **Algorithm** 배열을 초기화 한 후 데이터를 배열에 저장, 저장된 배열에서 단어의 수를 Counting 하고 배열 I행 0열의 값이 null이면 counting 종료 **Function** Null_array - 배열을 NULL로 초기화 , 데이터를 배열에 저장시키는 함수 Count - 단어를 출력후 개행 , 단어의 갯수를 파악하는 함수
Lab2-1	**Description** 데이터 파일을 입력받고 내가 찾고자 하는 수를 입력, 어느 곳에 위치하는지 찾는다. **Algorithm** 이진탐색으로 입력받은 수의 위치를 찾는다 **Function** Binarysearch - 이진탐색을 구현한 함수
Lab2-2	**Description** 피보나치 수열값(n)을 입력받고 출력 **Algorithm** 피보나치 수열 알고리즘을 짜고 함수에 대입, 입력받은 수 만큼 for문을 돌려 출력한다. **Function** Fibo - 피보나치 수열을 구현하기 위한 함수. Num을 입력받고 이에 따른 피보나치 수열값을 반환하여 준다.

Lab3	**Description** 다항식의 덧셈과 뺄셈을 하고 출력을 한다. **Algorithm** 각 다항식은 구조체로 이루어져 있고 차수를 비교하여 다항식 간의 덧셈과 뺄셈을 한다. **Function** padd_puls - A(x) 와 B(x)를 더하여 D(x)를 생성한다. COMPARE - 두 변수를 비교하는 함수. attach - D(x)를 실제로 생성하는 함수. padd_sub - A(x) 와 B(x)를 빼서 D(x)를 생성한다.
Lab4	**Description** 스택을 구현한다. **Algorithm** top이라는 인덱스를 이용하여 스택을 구현한다. 이 때, 스택의 크기는 100으로 고정되어 있다. **Function** create_stack - 스택을 생성한다. isFull - 스택이 꽉 찼으면 1을 반환한다. 그렇지 않으면 0을 반환한다. isEmpty - 스택이 비었으면 1을 반환한다. 그렇지 않으면 0을 반환한다. push - 스택에 숫자를 푸쉬하는 함수이다. pop - top 하나를 제거하는 함수. displayStack - 현재 쌓여진 수를 보여주는 함수이다. 스택을 출력한다.
Lab5	**Description** 스택은 정적, 큐는 동적으로 구현을 한다. **Algorithm** **Function** QueueInit - 큐를 초기화한다. QIsEmpty - 큐의 front와 rear가 같으면 TRUE, 그렇지 않으면 FALSE를 반환. Enqueue - 큐에 값을 저장한다. 인덱스는 rear로 지정한다. Dequeue - 큐의 값을 출력하면서 front의 값을 올린다. QPeek - 큐의 값을 출력은 하나 front의 값은 변동없다. Menu - 스택을 생성한다. Push - 스택에 값을 넣는다. Pop - 스택의 값을 하나 출력한다. top이 하나 내려간다. Peek - 스택의 값을 출력하나 인덱스는 그대로이다. isEmpty - 스택이 비었는지 확인한다. isFull - 스택이 가득 차 있는지 확인한다.

Lab6 , Lab7	**Description** Linked List를 직접 구현한다. **Algorithm** Node라는 구조체를 만들고 포인터로 각 node를 연결한다. **Function** List - 리스트를 생성한다. insert - 노드를 생성한 후 마지막 노드에 이어 붙인다. 이 때, 노드는 정렬이 되어 리스트에 붙여지게 된다. append - 리스트의 꼬리에 노드를 붙인다. isEmpty - 리스트가 비어있는지 확인한다. search - 입력받은 number를 리스트에서 검색하는 함수이다. deleteNode - 모든 노드를 출력한다. menu - 메뉴를 선택한다.
Lab8	**Description** Doubly List를 구현한다. **Algorithm** Node라는 Class를 만들고 포인터로 각 node를 연결한다. **Function** init - list의 head를 초기화한다. insertList - 오름차순으로 값을 리스트에 넣는다. deleteList - 특정 숫자를 가진 노드를 제거한다. forwardList - 리스트를 head부터 출력한다. backwardList - 리스트를 꼬리부터 출력한다. isEmpty - 리스트가 empty인지 체크한다. insert_before - 헤드에 노드를 붙인다. insert_after - 리스트 꼬리에 노드를 붙인다. findn - 리스트에서 특정 숫자를 찾고 몇 번째 노드에 있는지 반환한다. searchList - 특정 값이 리스트에 있는지 확인한다.

Lab9	**Description** 수식트리를 입력받고 전위, 후위, 중위 연산을 거친 수식을 출력한다. **Algorithm** 최대 길이 50의 수식을 입력받을 수 있으며 연산을 거친 결과의 값을 알 수 있다. 수식은 링크드리스트로 연결되어 있으며 각 연산에 따라 노드의 탐색 방법을 달리하게 된다. **Function** Bulid - 연산자의 우선순위를 결정하고, 수식을 입력받는다. Operand - 정수라면 노드를 오른쪽에 연결시킨다. Operator - 연산자라면 우선순위를 비교해서 트리를 재구성한다. inorder - 중위연산. postorder - 후위연산. preorder - 전위연산. evaluate - 루트를 기준으로 왼쪽과 오른쪽의 서브트리를 순회해서 수식의 값을 차례로 올려 보내 최종 결과를 반환한다.
Lab10	**Description** BST(Binary Search Tree)를 Linked List를 통해 구현하며 트리는 반 시계 방향 90도로 회전시킨 것 만큼 출력을 할 수 있다. 또한 특정 값을 insert 혹은 delete 할 수 있으며 중위, 후위, 전위 순회 등의 트리 순회를 선택 하여 노드를 탐색 할 수 있다. **Algorithm** BST를 Linked List를 통해 구현. 트리에 값을 추가 할 때는 루트 노드의 키 값과 비교하여 작을 경우 왼쪽, 클 경우 오른쪽 서브트리에 추가하며 모든 경우에 대하여 서브트리의 루트 키 값 > 왼쪽 노드 , 루트 키 값 < 오른쪽 노드를 만족해야 한다. **Function** drawBSTree - tree를 출력한다. insertTree - tree에 값을 추가한다. 이 때, 추가되는 노드는 알맞은 자리를 찾아간다. deleteBSTree - 특정 값을 가진 노드를 찾아 삭제를 하고 tree를 재구성한다. searchTree - 해당 Tree에서 특정 값을 찾는다. findmin - 해당 Tree에서 최소값을 찾는다.

Lab11&12	**Description** DFS , BFS를 구현한다. **Algorithm** 인접행렬을 이용하여 그래프를 구성 한 후, DFS혹은 BFS탐색을 할 수 있다. 이 때 방문여부는 vistied라는 이름의 배열을 이용하여 알 수 있다. dfs와 bfs의 함수는 재귀함수로 구현되어 있다. **Function** initGraph - 정점의 개수를 입력받고 graph를 초기화하는 함수. insertGraph - 파라미터로 u와 v를 받으며 u와 v를 연결하는 edge를 추가한다. displayGraph - 그래프에 연결된 각 정점을 출력한다. dfs - 깊이우선탐색을 실시한다. bfs - 너비우선탐색을 실시한다.
Lab13	**Description** PRIM알고리즘을 구현한다. 이 때 트리는 중간 과정을 출력을 하며 최종적으로 PRIM알고리즘에 따라 신장된 Tree를 보여주게 된다. 또한 Shortest Path를 구할 수 있다. **Algorithm** 하나의 정점을 선택하고, 선택한 정점에서 갈 수 있는 모든 정점 중에 최소 비중의 간선으로 이어지는 정점을 선택한다. 단, 싸이클이 없으며 비중이 같을 경우 먼저 탐색한 간선을 택한다. **Function** init - grpah의 초기화, 인자는 정점의 개수를 받는다. Shortestpath - 인자로 정점을 받고 해당 정점에서 모든 정점을 방문하기 위한 최단 경로를 그린다. 이 때, Dijkstra의 알고리즘을 이용한다. getcost - 주어진 인접행렬에서 비중을 구한다. choose - 비중을 기준으로 탐색할 정점을 선택하는 함수. Prim - Prim알고리즘을 실행한다.

list_sorting	**Description** Num와 Nmae을 입력받아 오름차순으로 정렬하여 LIST에 insert 후 delete, search, forward, backwoard의 기능을 수행한다. **Algorithm** Data를 입력받는다. 입력받은 Data를 List->data와 비교한다. 이 때, List가 비어있으며 추가된 노드가 head가 된다. 비어있지 않다면, 기존 data와 비교 후, 입력받은 data보다 더 큰 값이 나올 때 까지 노드를 찾고 이를 q와 p사이라고 지정한다. q와 p 사이에 노드를 삽입하고 next와 prev를 적절하게 링크를 시켜준다. **Function** init - List를 생성 후 초기화 하는 함수이다. head를 0으로 만든다. insertList - Num과 Name을 입력받은 뒤 List에 오름차순으로 정렬하여 node를 붙인다. deleteList - 입력받은 Num을 List에서 찾아 삭제한다. forwardList - 오름차순으로 List의 항목을 출력한다. backwardList - 내림차순으로 List의 항목을 출력한다. searchList - 입력받은 Num을 List내에서 검색한다.
Magicsquare	**Description** 사각형의 크기를 입력 받아 행, 열, 대각선의 합이 같은 사각형을 출력한다. (마방진을 말한다.) **Algorithm** 1부터 시작해 n까지 수가 사각형에 들어간다. 기준점을 1로 잡고 위로 한칸, 왼쪽으로 한칸 옮겨 그 점을 2라고 잡고 이를 n까지 반복한다. 단, 매직스퀘어의 크기는 홀수x홀수로 이루어진 사각형이다. n이 홀수일 때에는 마방진을 간단한 방법으로 만들 수 있다. 첫 번째 행의 한가운데 열에 1을 넣는다. 이어서 다음과 같은 규칙으로 숫자를 채운다. 즉, 다음 숫자를 대각선 방향으로 오른쪽 위 칸에 넣는다. 이때 해당하는 칸이 마방진의 위쪽으로 벗어난 경우에는 반대로 가장 아래쪽의 칸으로, 마방진의 오른쪽으로 벗어나는 경우는 가장 왼쪽의 칸으로 각각 한번더 이동한다. 오른쪽인 동시에 위쪽으로 벗어나는 경우 및 오른쪽 위에 다른 숫자가 이미 있는 경우에는 오른쪽위 대신 원래 칸의 한 칸 아래에 넣는다. **Function** size_check - 원하는 크기의 정사각형을 입력받고 범위가 올바른지를 검사하는 함수 Magicsquare - 입력받은 크기만큼 매직스퀘어를 생성, 출력하는 함수

ShortestPath	**Description** Cost Matrix에 기초하여 원하는 정점에서의 최단거리를 구한다. **Algorithm** Shortestpath(int v) Algorithm v는 정점을 의미하며 v에서의 각 정점에 이르는 최소 cost를 구하는 것이 목적이다. 함수가 호출되면 Cost Matrix에서 v의 행의 cost를 distance에 입력한 후에 choose 함수를 호출하여 방문자가 false이면 distance에 입력된 cost 비교를 한다. 더해진 cost가 적은 것이 그 정점에서의 새로운 cost가 된다. choose() Algorithm 입력된 vertex의 cost를 distance에 입력받으면(조건) choose함수 호출 시 distance의 원소를 비교하여 최소값을 갖는 원소의 번호를 반환한다. 여기서 원소의 번호는 vertex를 의미한다. **Function** init - graph를 초기화한다. Shortestpath - cost에 기초하여 최단거리를 비교, 출력한다. choose - 방문하지않은 정점에 대하여 최단거리를 구하고 반환한다. getcost - 주어진 Cost Matrix를 graph에 입력한다.
MaxHeap	**Description** 데이터를 입력받아 HEAP의 내용을 출력한다. **Algorithm** MaxHeap은 루트에 항상 가장 큰 킷값의 노드가 위치하고 있으며 부모 노드의 키 값이 자식 노드의 키 값보다 작지 않은 완전 이진트리를 뜻한다. 이를 토대로 프로그램을 구성했다. **Function** insert_maxheap - Data를 입력받아 내림차순 정렬삽입한다. root아래로 트리가 만들어진다. delete_maxheap - 가장 큰 Data 삭제 (root 삭제) print_maxheap - MAXHEAP을 root부터 출력한다. HEAP_FULL - HEAP이 풀인지를 검사한다. HEAP_EMPTY - HEAP이 비었는지를 검사한다. creat_maxheap - 입력데이터 8 6 4 2 5 3을 프로그램내에 생성한다.