프로그램 기능 및 구현 방법

File Manager 는 다음과 같은 기능을 갖는다.

1) 폴더 관리 기능

- **폴더 새로 만들기**
  : 모든 폴더는 하위 폴더 리스트를 갖고, 폴더를 새로 만들면 그 하위 리스트로 들어간다. 사용자가 폴더 이름을 입력하면, 자동으로 현재 폴더 위치에 따른 폴더 경로가 지정되며 현재 시간이 만든 날짜로 지정된 폴더가 생성된다. 모든 폴더는 상위 폴더의 포인터를 변수로 가진다.
- **폴더 삭제**
  : 사용자가 폴더 이름을 입력하면, 해당 폴더가 하위 폴더 리스트에서 없어진다.
- **폴더 열기**
  : 사용자가 요청한 폴더의 포인터를 가져와 현재 폴더 포인터를 변경해준다.
- **폴더 이름 바꾸기**
  : 이름을 바꾸면 해당 폴더의 경로, 최근 수정한 날짜가 변경되며 해당 폴더의 모든 Sub components 들의 경로, 최근 수정한 날짜 역시 자동으로 변경된다.
- **폴더 복사/붙여넣기**
  : 폴더를 복사하는 경우, 임시 폴더에 원래 폴더의 정보를 저장한다. 폴더 붙여넣기의 경우, 임시 폴더를 폴더리스트에 추가한다. 만약 해당 위치에 같은 이름을 가진 폴더가 있는 경우, “copy”가 이름에 붙게 된다.
- **폴더 복제**
  : 원래 폴더 이름에 “_copy”가 붙은 폴더 이름을 가진 폴더가 복제된다. 만약 똑같은 폴더를 n 번 복제하는 경우, 해당 폴더 리스트에서 자동으로 중복을 체크하여 n 번째 복제 폴더에는 “_copy”가 n 번 이름에 붙게 된다.
- **폴더 속성 보기**
  : 사용자가 요청한 폴더의 이름, 크기, 위치, 파일 수, 서브 폴더 수, 만든 날짜, 최근 수정한 날짜, 최근 열어본 날짜를 보여준다.

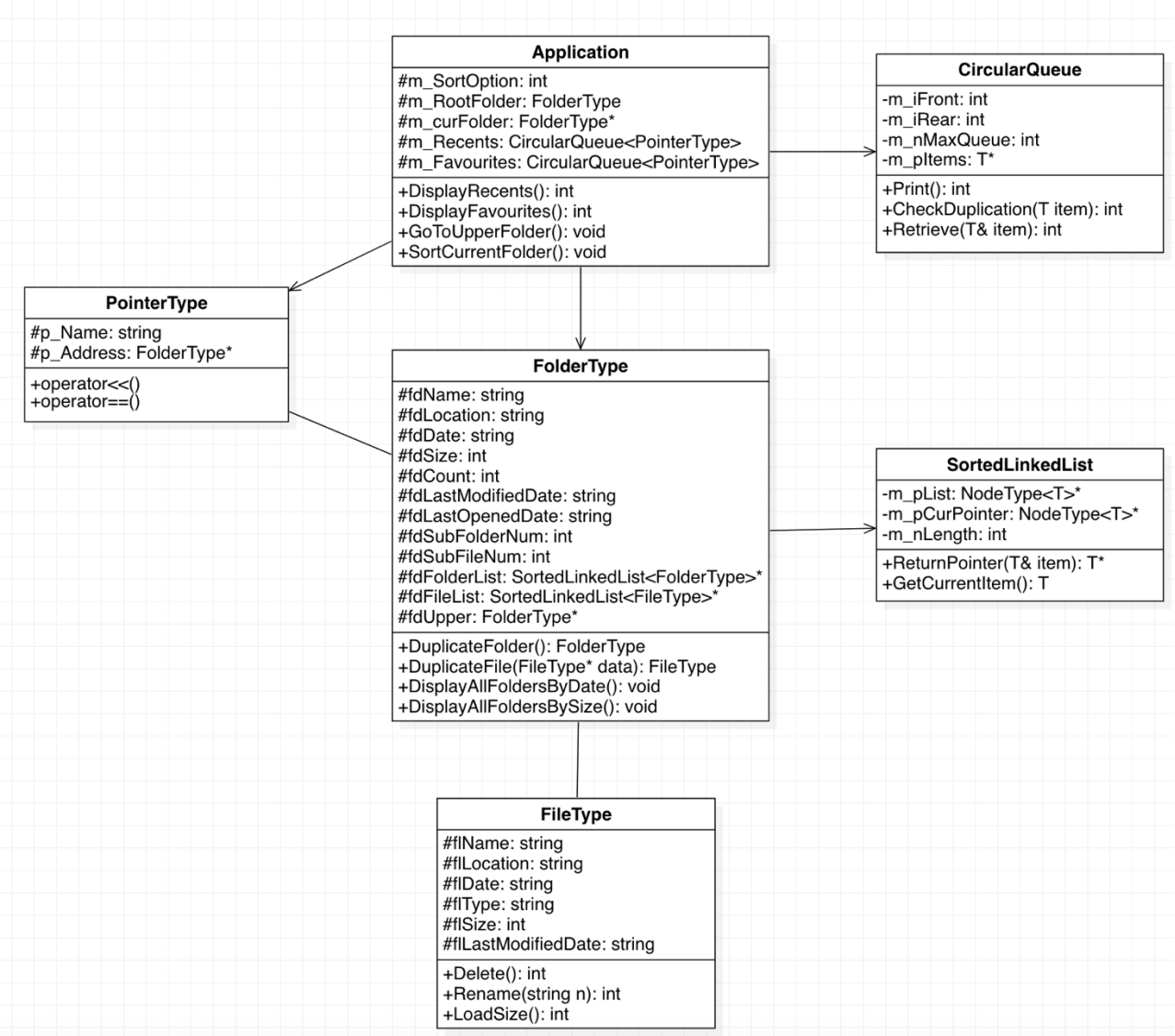
2) 파일 관리 기능

- **파일 새로 만들기**
  : 모든 폴더는 하위 파일 리스트를 갖고, 파일을 새로 만들면 그 하위 리스트로 들어간다. 사용자가 파일 형식, 파일 이름을 입력하면, 자동으로 현재 폴더 위치에 따른 파일 경로가 지정되며 현재 시간이 만든 날짜로 지정된 파일이 생성된다. 현재 파일 형식은 텍스트(.txt), 음악(.wav), 이미지(.jpg)이다. 텍스트 파일의 경우 실제로 프로젝트 폴더에 프로그램에서 입력한 내용을 가진 파일이 생성된다. 실제로 생성된 파일의 진짜 파일 크기를 불러와 프로그램 내 파일 크기로 설정한다. 파일이 추가될 경우, 해당 파일의 모든 상위 폴더의 사이즈를 해당 파일 크기 사이즈 만큼 늘려준다.
- **파일 삭제**
  : 사용자가 파일 이름을 입력하면, 해당 파일이 하위 폴더 리스트에서 없어진다. 실제로 프로젝트 폴더에서 삭제된다.
- **파일 열기**
  : 사용자가 요청한 파일을 실행한다. 텍스트 파일(.txt)의 경우, 파일 내용을 보여준다. 음악파일(.wav)의 경우, 실제로 프로젝트 폴더에 해당 이름을 가진 파일이 있을 경우, 음악이 재생된다. 사진 파일(.jpg)의 경우, 현재는 그냥 파일이 열렸다는 문구만 뜨게 설정했다.
- **파일 이름 바꾸기**
  : 이름을 바꾸면 해당 파일의 경로, 최근 수정한 날짜가 변경된다. 실제로 프로젝트 폴더에서 이름이 변경된다.
- **파일 복사/붙여넣기**
  : 파일을 복사하는 경우, 임시 폴더에 원래 폴더의 정보를 저장한다. 파일 붙여넣기의 경우, 임시 파일을 파일리스트에 추가한다. 만약 해당 위치에 같은 이름을 가진 파일이 있는 경우, “copy”가 이름에 붙게 된다.
- **파일 복제**
  : 원래 파일 이름에 “_copy”가 붙은 폴더 이름을 가진 폴더가 복제된다. 만약 똑같은 파일을 n 번 복제하는 경우, 해당 파일 리스트에서 자동으로 중복을 체크하여 n 번째 복제 파일에는 “_copy”가 n 번 이름에 붙게 된다.
- **파일 속성 보기**
  : 사용자가 요청한 파일의 이름, 크기, 위치, 만든 날짜, 수정 날짜, 최근 열어본 날짜를 보여준다.

3) 전체 관리 기능

- **상위 폴더로 이동**
  : 현재 폴더가 가진 상위 폴더 포인터를 현재 폴더 포인터로 지정해 현재 폴더의 상위 폴더를 오픈한다.
- **이름, 만든 날짜, 폴더 크기를 기준으로 폴더 및 파일 정렬**
  : 사용자 요청 시 만든 날짜, 폴더 크기를 기준으로 폴더와 파일을 정렬한다. 정렬 후 다시 기본값인 이름으로 설정된다.
- **폴더 및 파일 검색**
  : 현재 폴더에서 사용자가 입력한 키워드를 포함한 이름을 가지고 있는 모든 sub components, 파일과 폴더의 이름을 보여준다. 우선 현재 폴더의 파일 리스트를 검색한다. 그 후 폴더리스트를 검색하고, 폴더 리스트의 폴더가 또 서브 폴더, 파일을 가진 경우 recursion 을 이용해 검색한다.
- **자주 사용한 폴더, 최근 열어본 폴더 열람 및 선택 폴더로 이동**
  : 폴더 오픈을 하면 최근 열어본 폴더에, 폴더 오픈 횟수가 3 번을 넘게 되면 자주 사용한 폴더에 들어간다. 해당 리스트에 있는 폴더의 이름을 입력하면 바로 해당 폴더로 이동한다.

Class Diagram



Class Specification

해당 클래스가 갖는 특정 멤버 변수에 접근하지 않는 이상, 기능은 최대한 자료형 함수를 그냥 불러오는 식으로 프로그래밍 했다. Application Class 에서 AddFile()은, FolderType Class 의 AddFile()을 호출하고, FolderType Class 의 AddFile 은 SortedLinkedList Class 의 Add()를 호출한다.

1) Application Class

<멤버 변수>

int m_SortOption; // 사용자 정렬 옵션 (1 이름순, 2 최근 열어본 날짜순, 3 크기순)

FolderType tmpFolder; // 복사/붙여넣기용 임시 폴더

FileType tmpFile; // 복사/붙여넣기용 임시 파일

FolderType m_RootFolder; // 최상위 루트 폴더

FolderType* m_curFolder; // 현재 폴더를 가르키는 포인터

CircularQueue<PointerType> m_Recents; // 최근 열어본 폴더 큐

CircularQueue<PointerType> m_Favourites; // 자주 열어본 폴더 큐

<주요 멤버 함수>

int DisplayRecents();

// 큐의 Print() 함수를 이용해 최근 열어본 폴더 큐가 갖고 있는 PointerType 의 변수, 모든 폴더 이름 p_Name 을 보여준다. 그리고, 사용자가 폴더 오픈을 요청하면, 큐에서 해당 폴더를 검색한다. 찾으면 PointerType 의 변수, 해당 폴더의 주소 p_Address 를 불러와 현재 폴더 포인터로 할당해준다.

// DisplayFavourites()도 위와 같은 방법이다.

void GoToUpperFolder();

// 상위 폴더로 이동한다. 해당 폴더가 가진 변수인 상위 폴더 포인터로 현재 포인터를 옮겨준다.

2) PointerType Class

큐 사용을 위해 만든 클래스이다. 최근 열어본 폴더 큐에 폴더 메모리 주소를 담아야 해당 폴더 선택시 폴더 이동이 가능한데, 출력할 때에는 이름을 출력해야 한다. 따라서, string 형식의 폴더 이름과 FolderType* 형식의 폴더 주소를 변수로 갖는 PointerType Class 를 만들었다.

<멤버 변수>

string p_Name; // 폴더의 이름

FolderType* p_Address; // 폴더의 주소

<주요 멤버 함수>

friend ostream& operator<<(ostream& os, const PointerType& data);

// << 연산자를 오버로딩 할 때, PointerType 의 변수 중 이름을 출력하게 해 궁극적으로 CircularQueue Class 의 Print() 함수를 이용한다.

FolderType* GetAddress()

// PointerType 의 변수 중 폴더 주소를 가져온다.

friend bool operator==(const PointerType& data1, const PointerType& data2)

// == 연산자를 오버로딩 할 때, PointerType 의 변수 중 이름으로 비교해 해당 이름을 가진 폴더가 큐에 들어있는지 확인한다.

3) FolderType Class

<멤버 변수>

string fdName; // 폴더 이름

string fdLocation; // 폴더 위치

string fdDate; // 생성 날짜

int fdSize; // 폴더 사이즈

int fdCount; // 오픈한 횟수

string fdLastModifiedDate; // 최근 수정한 날짜

string fdLastOpenedDate; // 최근 열어본 날짜

int fdSubFolderNum; // 하위 폴더 개수

int fdSubFileNum; // 하위 파일 개수

SortedLinkedList<FolderType>* fdFolderList; // 서브 폴더 리스트

SortedLinkedList<FileType>* fdFileList; // 서브 파일 리스트

FolderType* fdUpper; // 상위 폴더 주소값을 갖는 포인터

<주요 멤버 함수>

FolderType DupilcateFolder(); // 폴더의 복제본을 만들어 반환한다.

FileType DuplicateFile (FileType *data); // 파일 data 의 복제본을 만들어 반환한다.

void DisplayAllFoldersByDate();

// 서브 폴더 리스트에 들어간 폴더들의 fdDate, fdName 을 각각 임시 array 에 순서대로 넣는다. 그 후 날짜 array 를 선택 정렬해주는데, 날짜 대소 비교에 따라 해당 순서로 이름 array 도 정렬해준다. 그 후 이름 array 의 item 들을 차례로 출력해 sorting 한다.

// DisplayAllFoldersBySize()도 위와 같은 방법이다. fdDate 대신 fdSize 로 정렬한다.

4) FileType Class

<멤버 변수>

string flName; // 파일 이름

string flLocation; // 파일 위치

string flDate; // 생성 날짜

string flType; // 파일 타입 (txt, jpg, wav)

int flSize; // 파일 크기

string flLastModifiedDate; // 최근 수정한 날짜

<주요 멤버 함수>

int Delete();

int Rename(string n);

// 실제로 프로젝트 폴더에 있는 파일을 삭제하고, 이름을 변경할 수 있게 했다.

int LoadSize();

// 프로젝트 폴더에 있는 파일의 실제 크기를 불러와 flSize 에 할당한다.

자료 구조 선택 이유

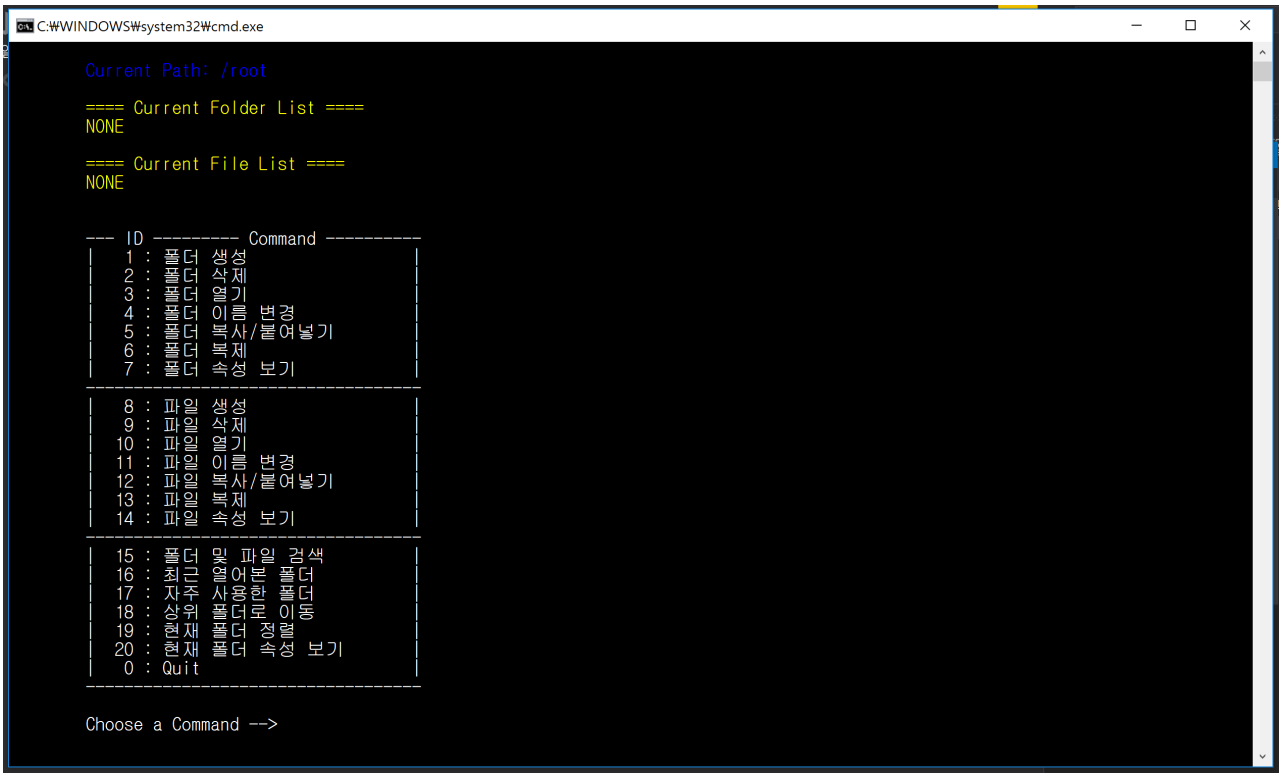
1) Application Class 의 m_Recents, m_Favourites 를 Queue 로 구현한 이유

최근 사용한 폴더 리스트의 경우, queue 가 max size 에 도달했을 때, 그중 가장 오래 전에 사용했던 first item 이 삭제되는 것이 합리적이다. 자주 사용한 폴더의 경우에도, 특정 사용 횟수를 넘은 항목들에 대해서는 자주 사용했지만, 최근에는 사용하지 않은 가장 먼저 들어온 item 이 삭제되는 것이 맞다고 생각했다. 그리고 sorting 을 그냥 들어온 순서대로 하면 되기 때문에, queue 가 적합하다고 판단했다. 또한, '최근에', '자주' 열어본 폴더를 담는 자료 구조는 일종의 캐시이므로 크기가 한정적인 것이 오히려 메모리를 과하게 차지를 하지 않게 되는 장점이 있다.

2) FolderType Class 의 fdFolderList, fdFileList 를 Sorted Linked List 로 구현한 이유

사용자가 원하는 대로 폴더와 파일을 계속 추가해주어야 한다. Linked List 의 경우, 동적 할당이 필요가 없으므로 이 폴더 관리 시스템에 더 적합하다고 판단했다. 그리고 폴더 시스템의 검색 기능은 특정 item 하나를 찾아주는 것이 아니라 keyword 를 포함한 모든 폴더를 검색하는 것이다. 결국 모든 item 들을 확인해야 한다. 따라서, Linked List 의 단점인 Binary Search 가 불가능하다는 점에서 타격을 입지 않는다.

콘솔 실행 예시 (* 동영상 별첨)



가독성을 위해 콘솔 글씨 색상을 변경했다.

GUI 구현

1) 사용한 프로그램, 작업 환경

C++을 기반으로 하는(* Syntax 가 아예 같지는 않다.) QT5 툴킷으로 QT Creator 로 Mac OS 환경에서 작업했다. 빌드 후 .app 맥 애플리케이션 파일을 만들 수 있었다. Reference 를 참고해 함수들을 사용했고, 내장된 Tree data structure 을 사용했다. 현재 폴더와 파일 목록을 보여주는 Tree Widget, 사용자의 입력을 받는 Line Edit, 타입을 선택하는 Combo Box, 사용자가 클릭해 기능을 실행하는 Push Button 등을 이용해 GUI 를 꾸몄다.

2) 지원하는 기능

각 아이템은 이름, 타입, 날짜를 가진다. 타입에 맞는 아이콘을 지원한다. Root 폴더 추가, 하위 폴더 / 파일 추가, 삭제, 모든 폴더 및 파일 삭제, 이름순 정렬, 타입순 정렬, 날짜순 정렬의 기능을 지원한다.

GUI 실행 예시 (* 실행 동영상, 작업 화면 동영상 별첨)