**작품 요약서**

* **Basic Data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **작품명** | Ez Project | | | **지원분야** | Software |
| **개발인원** | 1명 | **본인역할** | PL | **개발기간** | 2013.09  ~ 2013.12 |
| **개발환경** | Microsoft Windows 7 Professional K  .Net Framework 4.0 Client Profile | | | | |
| **개발언어** | C# | | | | |
| **개발툴** | Microsoft Visual Studio 2010 | | | | |
| **작품소개**  **(요약)** | 컴퓨터로 작업했던 대부분의 사람들은 수집한 자료들의 관리에 한번씩 어려움을 느껴봤을 것입니다. 보관하는 자료가 웹 브라우저 즐겨찾기에 내용이 추측 불가능하게 추가되어있고, 해당 폴더에 특정기준 없이 수집된 파일들, 바탕화면 쪽지 프로그램에 무분별하게 기록된 수십 개의 쪽지들, 간단하게 기록한다며 켜둔 여러 개의 메모장 등 구분되지 않고 자료들이 계속 생기며, 다른 작업과의 병행을 방해합니다. 윈도우 파일 구조를 이용하여 체계적으로 정리를 할 수도 있지만, 관리하는 자료가 늘어남에 따라 정리하는 시간도 비례하여 늘어나며 자신이 원하는 자료가 어디에 위치하는지, 어떤 파일이름인지 암기를 해야 하는 상황도 발생합니다.  여러 가지 파일과 폴더 관리 프로그램을 찾아봤지만 윈도우 탐색기를 개량하거나 윈도우의 기능을 좀더 쉽게 접근하는 기능뿐이었으며, 프로젝트 관리 프로그램은 프로젝트의 주기나 모델링 기능이 있었고 관련된 데이터를 정리하는 기능은 없었습니다. 그래서 위에 나왔던 불편한 점을 묶어서 관리하도록 생각했습니다. 인터넷을 검색하다 빠르게 해당 자료를 저장하거나 링크를 걸고, 관련 있는 사진 파일이나 동영상파일과 개인적인 생각, 태그 등을 정의하여 하나로 통합하여 관리하며 수집된 자료를 빠르고 쉽게 찾을 수 있도록 프로그램을 제작하게 되었습니다. | | | | |
| **작품내용** | 성공한 사람들의 대부분 공통점은 자주 메모를 하고 정리하였으며 필요한 순간마다 정리한 메모를 사용한 것입니다. 메모의 기술 7가지를 확인하면 언제 어디서든 메모를 하며, 중요사항 체크, 정리방법, 활용방법 등의 방법이 나옵니다. 위의 메모방식을 크게 나누어 보면 빠르게 메모를 하고, 메모를 체계적으로 정리를 해야 하며, 필요한 순간 원하는 메모를 찾는 것입니다. 프로그래밍 초점을 여기에 맞춰서 개발을 했습니다. 메모를 입력할 때는 여러 가지 데이터(제목, URL, 파일경로, 태그, 관련날짜)를 통합적으로 간편하게 입력할 수 있도록 UI를 구성하고 접근이 쉽도록 System Tray나 투명도, Drag & Drop으로 다른 작업과의 병행능력을 높였습니다. 그리고 메모를 Tree형식으로 정리하여 메모간의 상하관계와 관련링크 및 그룹화, 중요도 설정기능으로 자료를 쉽게 분별할 수 있으며, 원하는 메모를 검색할 경우엔 프로그램에 맞도록 Trie자료구조와 검색알고리즘을 수정하여 자신이 원하는 메모에 빠르게 접근하도록 구현하였습니다. | | | | |
| **개발내용**  **(본인 구현부분)** | **<<변경한 Trie자료구조>>**  저장된 데이터를 파일로 입출력 경우엔 Trie자료구조 형식을 사용합니다. 파일로부터 데이터를 읽을 경우 Trie자료구조를 통해 데이터의 유무를 빠르게 확인할 수 있으며, 자료를 순차적으로 불러올 경우(파일 입력) 순차적으로 특정 인덱스를 부여합니다. 데이터의 수정 및 변경, 저장 시 Trie구조로 관리하기가 쉬우며 저장된 Trie구조를 바탕으로 객체들간의 관계를 표시합니다.  **<<변경한 Trie자료구조에 대한 검색알고리즘>>**  객체에 데이터가 할당될 때 순차적으로 인덱스 값을 설정합니다. 순차적으로 인덱스 값을 설정할 경우 DFS(Depth First Search)방식 순서로 인덱스가 정해집니다. 이 인덱스 값은 특정 객체의 번호를 중심으로 검색하거나 접근할 때 사용되며 BST(Binary Search Tree)에서 착안하여 만들어졌습니다. 특정 인덱스를 비교하여 검색하기 때문에 최저 비교횟수는 1번, 최대 비교횟수는 (Tree의 높이\*n)만큼 비교를 하기 때문에 검색속도가 빠릅니다.  **<<Tree의 하위 node에 대한 일괄작업>>**  특정 데이터 객체를 삭제하거나 Tree의 특정한 위치로 옮기게 될 경우, 특정 데이터 객체의 하위 node를 일괄적으로 작업합니다. 연관된 링크가 설정되어있는 경우 삭제 시 링크를 제거해버리고, 옮기게 될 경우엔 링크를 재 조정하는 작업을 하게 됩니다.  **<<프로젝트 구분>>**  개인 사용자가 여러 가지 작업을 구분하도록 ‘주제’라는 단어로 프로젝트 및 작업을 구분합니다. 프로젝트에 데이터 객체를 추가, 삭제, 편집, 검색 가능합니다. Tree구조의 Root부분에 해당합니다.  **<<Drag & Drop>>**  바탕화면이나 특정 폴더에 들어있는 아이콘을 프로그램의 데이터 객체로 Drag하면 Drag한 해당 프로그램의 정보가 Drag당한 데이터 객체의 하위 개념의 데이터 객체로 저장이 됩니다. 데이터 객체가 아닌 컨텐츠에 Drag를 할 경우 아무 이벤트가 일어나지 않습니다.  **<<IE 실행중인 웹 주소 참조>>**  Shdocvw.dll의 ShellWindows()함수에서 실행중인 Internet Explorer의 리스트를 받아옵니다. 그리고 리스트를 URL과 URL이름으로 출력한 다음 선택된 리스트 항목의 URL을 정해진 항목에 넣습니다.  **<<프로그램 자동실행>>**  프로그램을 자동실행 하도록 설정을 변경하면 Microsoft.Win32.Registry클래스를 이용하여HKEY\_CURRENT\_USER\Software\Microsoft \Windows\CurrentVersion\Run 에 레지스터리 키 값이 추가가 됩니다. 설정을 해제할 경우 해당 위치에 추가한 키 값을 삭제합니다.  **<<투명도 설정>>**  작업화면에서 투명도를 0%~80%까지 수정할 수 있습니다. 0%~80%까지 5단계로 구분하여 20%씩 조정이 가능하며, 투명도 설정 후 프로그램에서 새로운 폼이 열리면 설정한 투명도를 그대로 따라갑니다. | | | | |
| **기타** | 이 프로그램은 파일 입출력 시 DB를사용하지 않고 텍스트 파일을 이용합니다. 텍스트 파일로 입출력을 하는 이유는 DB파일은 고정적으로 용량을 할당합니다. 그러나 해당 프로그램은 몇몇 개의 특정 필드의 데이터가 가변적이라 단일테이블로 구성을 할 경우 저장된 데이터에 비해 용량 낭비가 심해집니다. 용량 낭비를 줄이기 위해 테이블을 여러 개 생성하여 파일 입출력을 하게 되면 테이블끼리의 조인에 오버헤드가 생기며, 텍스트 파일의 입출력 속도보다 느립니다. 그리고 이 프로그램은 개인사용자를 대상으로 만들어 졌기 때문에 다중사용자에 장점이 많은 DB파일은 부적합하다고 생각합니다. 그런 이유로 텍스트 파일로 입출력하며 따로 암호화를 설정하였습니다.  차후 개발 방향은 개인사용자를 다중사용자로 바꾸면서 서버를 구축해 운영을 해볼 생각입니다. 다중사용자로 초점을 맞추면서 그룹으로 프로젝트를 관리하거나 등급에 따라 권한이 나눠지는 네트워크를 기반으로 개발할 예정입니다. 이 경우엔 DB로 입출력하며 Cloud로 개발할 생각입니다. | | | | |

* **System Architecture**

|  |
| --- |
| C:\Users\NC\Desktop\1.png |
| **그림 1 System Architecture** |
| 사용자가 저장한 데이터는 sav파일에 있습니다. 저장된 데이터를 WorkSpace로 불러오기 위해 sav파일을 복호화(DES.decrypt)한 후, File I/O와 Parsing을 거치게 됩니다. Parsing된 데이터는 Memo클래스에 동적으로 할당됩니다. 메모리에 적재된 Memo객체는 WorkSpace에서의 작업데이터가 됩니다. 작업이 완료된 Memo객체는 파일에 입력 후 암호화(DES.Encrypt)를 진행합니다. 암호화 방식은 DES 알고리즘을 사용하였으며, 사용자가 관리하는 패스워드 하나와 프로그램에 입력되어있는 패스워드로 암호화합니다. MHOL(Microsoft Html Object Library)와 MIC(Microsoft Internet Controls)는 현재 열린 IE 탐색기의 주소 값들을 추출하기 위해 사용합니다. |

* **Software Architecture**

|  |
| --- |
| C:\Users\NC\Desktop\2.png |
| **그림 2 software architecture(Parsing)** |
| 다음은 복호화된 파일을 Parsing하는 과정입니다. 복호화된 파일이 입력되면 정의된 ReadFile 클래스에서 데이터를 Parsing합니다. Parsing된 데이터는 크게 두 가지로 TrieIndex와 Data로 이루어 집니다. 하나의 데이터가 Parsing이 되면 Counter의 값이 1 증가하며, Memo객체에 TrieIndex, CountIndex, Data를 입력 받으며 생성합니다. |
| C:\Users\NC\Desktop\3.png |
| **그림 3 software architecture(trie identify and data input)** |
| 다음으로 생성된 Memo객체가 바르게 생성되었는지 무결성(integrity)을 확인하고 해당위치에 연결하는 구조입니다. Parsing을 거쳐 생성된 Memo객체는 링크를 걸어서 Tree구조로 만들어 줘야 하며 Trie\_Identify()함수를 통해 해당 Memo객체가 제대로 Parsing되었는지, 상위 node가 어디에 있는지 확인하고 링크를 생성합니다. 링크가 생성될 경우, Memo[]에 순차적으로 new Memo의 위치가 들어갑니다. 상위 node가 없을 경우엔 Error처리를 하며, 다음으로 생성된 Memo를 확인합니다. |
| **C:\Users\NC\Desktop\4.png** |
| **그림 4 software architecture(workspace method)** |
| 다음은 WorkSpace(메인화면)에서 사용하는 주요 메서드들의 구조와 특정 메서드의 루틴입니다. Parsing이 완료된 최상위 Memo객체(root)를 이용하여 작업을 시작합니다.  PaintGorup은 WorkSpace에 Memo객체의 내용을 Tree형식으로 모니터에 출력하는데 필요한 함수들의 그룹입니다.  FunctionGroup의 Add(), Delete(), Move()가 실행될 경우 countIndex(이후 Memo번호)값이 변경이 생기며 재조정하는 작업을 하며 DFS방식으로 재조정을 합니다. 재조정 작업 후 모니터에 데이터들을 새로 출력해줍니다. Search()의 경우 search\_Routine()을 실행하여 값을 검색하며, Memo번호를 이용하여 검색을 합니다.  save()는 파일에 입력하기 위해 그 동안 수정되었던 Tree를 재조정하여 Trie 형식으로 변경합니다. |
| **C:\Users\NC\Desktop\5.png** |
| **그림 5 software architecture(search algorithm)** |
| 특정 메모를 빠르게 찾기 위한 알고리즘입니다. 이 알고리즘은 countIndex(메모번호)의 값이 DFS(Depth First Search)방식으로 할당되어야 합니다. Memo배열의 길이를 중심으로 -1씩 감소하며 검사합니다. 검색하는 메모번호가 Memo[i]의 countIndex보다 클 경우 Memo[i-1]의 count를 검사하며, 검색 count가 클 경우엔 Memo[i]의 내부 변수인 Memo배열을 받아와 재귀로 호출합니다. 검색하는 메모번호와 Memo[i].countIndex의 비교 값이 Memo[i].count가 클 경우가 적중률이 높아 먼저 비교를 하며, 적중률 순으로 검색하는 메모번호 값이 클 경우, 해당하는 검색count이 Memo[i].count인지 확인하는 순으로 설정하여 오버헤드를 줄였습니다. |
| **C:\Users\NC\Desktop\6.png** |
| **그림 6 software architecture(another method)** |
| Ez Project에서 사용되는 함수입니다. Change\_Opactity()은 프로그램 작업화면의 투명도를 조절하는 함수이며, change\_ViewOption()는 Memo객체를 한번에 확인할 수 있는 개수를 설정합니다. 두 함수는 실행 후 WorkSpace에 Memo객체의 내용을 출력합니다.  Ie\_getData()함수는 windows detector의 활성화된 프로그램에 관련된 주소를 가져오는 함수입니다. Shdocvw.dll파일이 있어야 하며, 현재 접속한 URL주소나 실행중인 프로그램의 파일경로를 얻습니다.  File\_DragEv()는 windows icon을 프로그램으로 드래그 시 이벤트가 발생하며 해당 icon에 연결된 프로그램의 정보들을 얻습니다. systemTray()는 windows form에서 close()이벤트가 실행될 때, 시스템트레이로 숨기는 함수입니다.  autorun()은 컴퓨터가 부팅될 때 프로그램이 자동실행 유무를 설정합니다. 설정할 경우 상단의 레지스터리 경로에 필요한 프로그램 정보들을 입력합니다. |

**Function Explanation (Applicant)**

|  |  |  |
| --- | --- | --- |
| C:\Users\NC\Desktop\1.png | | |
| **그림 7 Category** | | |
| 자신의 프로젝트를 분류해 놓은 주제선택창입니다. 주제를 클릭하면 오른쪽에 주제와 관련된 간략한 정보들이 나오며, 새로운 주제를 추가하거나 필요 없는 주제를 삭제합니다. 하나의 주제를 누른 후 주제이동 버튼을 누르면 WorkSpace창으로 넘어갑니다. | | |
| C:\Users\NC\Desktop\8.png | | |
| **그림 8 System Tray** | | |
| Ez Project 프로그램이 Close()이벤트가 발생했을 때 System Tray로 들어갑니다. 이후 System Tray에서 작업 중이었던 프로그램을 열거나, 종료를 할 수 있습니다. | | |
| C:\Users\NC\Desktop\2.png | | |
| **그림 9 WorkSpace(메인 화면)** | | |
| Ez Project의 주기능이 있는 WorkSpace(메인 화면)입니다. Memo객체의 데이터들이 Trie트리형식으로 출력되어 있으며, 오른편으로 추가, 삭제, 옮기기, 이동, 검색 등 기본메뉴가 있습니다.  Memo객체의 왼쪽 상단의 숫자는 countIndex(메모번호)이며, 해당 프로그램이 데이터를 구분, 빠르게 찾기 위해 사용합니다.  Memo객체의 아래에 F, U, D의 뜻은 File path, URL, Detail Memo의 약자입니다. 해당 데이터의 존재여부를 알려줍니다.  Memo객체의 아래왼편에 있는 별표는 데이터의 중요도를 나타낸 것입니다. 0개에서 5개까지 표현할 수 있습니다.  Memo객체의 오른편에 있는 화살표를 클릭 시 해당 Memo객체를 중심으로 Tree가 출력됩니다. Tree의 높이를 조절하거나 상위 node의 데이터를 확인할 경우 오른편의 화살표 버튼을 이용하면 됩니다. | | |
| C:\Users\NC\Desktop\3.png | |
| **그림 10 DataView(세부 정보)** | |
| Memo객체를 클릭하면 나타나는 세부정보 창입니다. 관련된 세부정보를 추가 및 삭제할 수 있으며, 저장된 파일경로나 URL주소를 등록하여 실행할 수 있습니다. 그리고 Memo객체끼리 서로 링크를 걸 수 있습니다. 이동 버튼을 누르면 WorkSpace에 해당 메모를 중심으로 Tree가 출력됩니다. 현재 화면은 88번 Memo 장이론(field theory)에 대한 세부정보를 확인하는 것입니다. | |
| C:\Users\NC\Desktop\4.pngC:\Users\NC\Desktop\5.png | |
| **그림 11 DataView – URL search** | |
| 그림 10의 세부정보 창에서 IE에서 찾기 버튼을 누르면 그림 11의 리스트가 나타납니다. 리스트의 내용은 현재 실행중인 IE의 웹 이름과 URL주소를 확인할 수 있습니다. 리스트 중 하나를 더블클릭 시 그림 10 세부정보창의 URL의 빈칸에 선택된 URL의 주소가 자동으로 입력됩니다. | |
| C:\Users\NC\Desktop\6.png | |
| **그림 12 Opacity(투명도)** | |
| 오른쪽 메뉴의 하단에 있는 Bar를 이용하여 WorkSpace의 투명도를 조절할 수 있습니다. 투명도는 0%에서 80%까지 조절할 수 있으며 투명도를 조절한 후 Memo객체의 세부정보를 확인하거나 다른 창을 열게 되면 설정된 투명도가 그대로 적용되어 나타납니다. | |
| **C:\Users\NC\Desktop\7.png** | |
| **그림 13 Option** | |
| WorkSpace에서 한번에 확인할 수 있는 Tree의 높이를 설정할 수 있습니다. 그리고 윈도우 실행 시 자동실행에 체크를 할 경우, 컴퓨터 부팅 시 해당 프로그램이 자동으로 실행됩니다. | |
| C:\Users\NC\Desktop\9.png | |
| **그림 14 Memo Link** | |
| 상단의 그림을 보면 빨간 표시가 된 Memo객체가 배경색만 다른 상태로 같은 이름과 같은 countIndex(메모번호)를 가지고 있습니다. 해당 기능은 Memo객체를 링크를 건 것입니다. 링크를 걸어서 Memo의 중복입력을 피하고, 관련된 Memo와의 관계를 유기적으로 표현할 수 있습니다. 링크로 연결된 Memo는 점선으로 연결되며, 초록색 배경으로 표시됩니다. | |
| C:\Users\NC\Desktop\10.png | |
| **그림 15 Memo Move** | |
| 입력된 Memo에 대하여 위치에 변동이 생길 경우 Memo 옮기기를 할 수 있습니다. Memo 옮기기를 할 경우 해당 메모 countIndex(메모번호)와 옮겨질 위치의 상위 메모 countIndex를 입력하면 해당 Memo와 하위 Memo들이 옮겨질 위치로 모두 이동 됩니다. 그리고 메모번호가 재배치됩니다.  그림 15는 메모리 계층구조(countIndex:49)가 chapter 02 운영체제 개요(countIndex:55)의 하위로 이동한 결과입니다. | |
| **C:\Users\NC\Desktop\11.pngC:\Users\NC\Desktop\12.png** | |
| **그림 16 Memo Search** | |
| Memo객체를 검색할 경우엔 countIndex(메모번호)를 이용하여 검색합니다. countIndex를 기준으로 범위를 정하며, 해당하는 문자열이 있으면 결과를 출력합니다. 검색결과는 countIndex를 기준으로 결과가 나타나며, 리스트의 항목을 더블클릭 시 세부정보가 나옵니다. 검색 결과의 출력물로 다시 재 검색을 할 수 있습니다. | |
| C:\Users\NC\Desktop\13.png**C:\Users\NC\Desktop\14.png** | C:\Users\NC\Desktop\15.png |
| **그림 17 icon Drag** | |
| 바탕화면이나 폴더에서 아이콘을 Drag하여 WorkSpace의 Memo객체에 놓으면 Drag한 프로그램의 정보들이 Memo객체로 입력되기 직전의 상태로 보여줍니다. 입력될 위치와 파일 생성날짜, 작업날짜, 메모제목, 확장자, 용량, 해당 파일의 경로가 자동으로 기입되며, 메모 입력을 누르면 오른편의 그림처럼 결과가 출력됩니다. | |
| C:\Users\NC\Desktop\enc.png | |
| **그림 16 Data Encryption** | |
| 프로그램 sav파일이 암호화된 모습입니다. DES알고리즘을 이용하였습니다. 파일이 로드로드 사용될 때 복호화를 거쳐서 사용됩니다. | |
| C:\Users\NC\Desktop\점근적분석.png | |
| **그림 17 Algorithm Time Complesity** | |
| Trie구조에 인덱스의 유무에 따라 알고리즘의 점화식을 확인하고 반복대치 방식으로 속도차이를 비교해 보겠습니다. 특정한 기준인 Index가 있을 경우 하나의 node에서 비교 후 다음 하위 node를 검색하기 때문에 T(n)=kn이라는 값이 나옵니다. 그러나 기준이 없는 Trie구조는 전체를 검색해야 하기 때문에 T(n)=n^k라는 값이 나옵니다. 대략적으로 n의 K승 만큼, Tree의 길이가 깊어질수록 속도차이가 많이 납니다.  사전 검색 시 사용되는 Trie구조는 기준을 문자로 사용합니다. 그러나 저의 프로그램에선 문자로 Index를 줄 경우 Index가 중복될 수 있고, 시스템에서 문자를 비교하는 속도보다 정수를 비교 하는 수행속도가 더 빠르기 때문에 Trie자료구조에 정수형 Index를 추가하여 사용합니다. | |