과제

장유선

2023.08.11

1. 문제 정의

DBF 파일을 읽어서 화면에 출력하는 프로그램 제작

2. Python Code Hard Copy

|  |
| --- |
| def read\_dbf\_file(file\_path):      with open(file\_path, 'rb') as f:          version = f.read(1)          year = f.read(1)          month = f.read(1)          day = f.read(1)          record\_number = int.from\_bytes(f.read(4), byteorder='little')          header\_bytes = int.from\_bytes(f.read(2), byteorder='little')          record\_bytes = int.from\_bytes(f.read(2), byteorder='little')          f.read(20)          field\_count = (header\_bytes - 32 - 1) // 32          fields = []          for \_ in range(field\_count):              field\_info = f.read(11).decode('utf-8').rstrip('\x00')              field\_type = f.read(1).decode('utf-8')              f.read(4)              field\_length = int.from\_bytes(f.read(1), byteorder='little')              f.read(15)              fields.append((field\_info, field\_type, field\_length))          f.read(1)          print("필드 정보")          for field in fields:              print("- " + field[0])            for \_ in range(record\_number):              f.read(1)              record\_data = f.read(record\_bytes - 1)              offset = 0              print('\n')              print("Row", \_ + 1)              for field\_info, \_, field\_length in fields:                  field\_value = record\_data[offset:offset + field\_length].rstrip(b'\x00').decode('utf-8')                  print(f"- {field\_info}: {field\_value.strip()}")                  offset += field\_length    dbf\_file\_path = 'sample.dbf'  read\_dbf\_file(dbf\_file\_path) |

3. Code 설명

<참고 자료>

헤더 정보를 간단히 나열해 보면, 다음과 같다.

|  |
| --- |
| 1. 0 (길이:1바이트) : 정확한 dBase III+ 파일인지 여부(03h이면 메모파일이 없는 DBF파일이고, 83h이면 메모파일이 있는 DBF 파일) 2. 1-3 (3) : YYMMDD 형식의 최종 갱신 일자. 3. 4-7 (4) : 테이블 내의 전체 레코드 개수. 4. 8-9 (2) : 헤더의 바이트 수. 5. 10-11 (2) : 레코드의 바이트 수. 6. 12-14 (3) : 예약된 바이트(건너뛰면 됩니다). 7. 15-27 (13) : LAN을 위한 dBase III+를 위해 예약된 바이트(역시 건너뜁니다). 8. 28-31 (4) : 예약된 바이트(건너뛰면 됩니다). 9. 32-n (n\*32) : 각각의 컬럼들을 설명하기 위한 32바이트. 10. n+1 (1) : 헤더의 끝을 알리는 0Dh. |

1번에서 8번까지는 전체 DBF 파일에 대한 정보이고, 9번은 각 컬럼별 정보이다.

이것은 다음과 같다.

|  |
| --- |
| 1. 0-10 (11) : ASCII 형식의 컬럼명(이후는 00h로 채워집니다). 2. 11 (1) : ASCII 형식의 컬럼 타입(C/D/L/M/N 중의 하나입니다). 3. 12-15 (4) : 컬럼 항목의 주소(잘 쓰이지 않으므로 건너뜁니다). 4. 16 (1) : 컬럼값의 최대 길이. 5. 17 (1) : 컬럼의 개수라는데, 역시 건너뛰어도 무방합니다. 6. 18-19 (2) : LAN을 위한 dBase III+를 위해 예약된 바이트(건너뜁니다). 7. 20 (1) : 작업 영역의 ID(건너뜁니다). 8. 21-22 (2) : LAN을 위한 dBase III+를 위해 예약된 바이트(건너뜁니다). 9. 23 (1) : SET FIELDS 플래그(건너뜁니다). 10. 24-31 (8) : 예약된 바이트(건너뜁니다). |

헤더가 끝난 이후는, 헤더에 기록된 컬럼 순서와 길이대로 실제 컬럼값이 나열된다.다만 하나의 레코드 앞에는 1바이트가 더 붙고, 이 1바이트가 '공백(20h)'이면 해당 레코드가 존재한다는 뜻이고, '\*(2Ah)'이면 해당 레코드가 삭제되었다는 뜻이다.  
  
그리고, 지정된 컬럼값 미만의 길이를 가질 때는 나머지는 공백문자('20h')로 채우게 되며, 컬럼별 구분자는 없다.  
  
파일의 끝은 '1Ah'로 끝나게 된다.

출처 [<https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=best_today&logNo=90019151568>]

def read\_dbf\_file(file\_path):

    with open(file\_path, 'rb') as f:

        version = f.read(1)

        year = f.read(1)

        month = f.read(1)

        day = f.read(1)

        record\_number = int.from\_bytes(f.read(4), byteorder='little')

        header\_bytes = int.from\_bytes(f.read(2), byteorder='little')

        record\_bytes = int.from\_bytes(f.read(2), byteorder='little')

        f.read(20)

#필드 정보 하나당 32바이트를 사용하므로 필드 정보 바이트를 제외한 부분을

    #필드 정보 하나의 바이트 수로 나누어 필드 개수를 계산

        field\_count = (header\_bytes - 32 - 1) // 32

        fields = []

* 필드 정보의 크기와 내용을 파일에서 읽어와서 field\_info, field\_type, field\_length 변수에 저장한다.
* field\_info를 utf-8 형식으로 디코딩하고, 끝에 있는 NULL 바이트('\x00')를 제거한다.

- 필드 정보 하나를 읽을 때마다 32바이트를 읽어 건너뛴다.

for \_ in range(field\_count):

            field\_info = f.read(11).decode('utf-8').rstrip('\x00')

            field\_type = f.read(1).decode('utf-8')

            f.read(4)

            field\_length = int.from\_bytes(f.read(1), byteorder='little')

            f.read(15)

            fields.append((field\_info, field\_type, field\_length))

        f.read(1) #terminator 건너뛰기

        print("필드 정보")

        for field in fields:

            print("- " + field[0])

        # 데이터 레코드 출력

        for \_ in range(record\_number):

            f.read(1)  # Skip the record marker

            record\_data = f.read(record\_bytes - 1)

#레코드 마커 바이트를 제외한 나머지 데이터 읽기

            offset = 0 #레코드 데이터에서 각 필드의 시작 위치를 나타내는 변수 초기화

            print('\n')

            print("Row", \_ + 1) #현재 처리중인 레코드 번호 출력

            for field\_info, \_, field\_length in fields:

                field\_value = record\_data[offset:offset + field\_length].rstrip(b'\x00').decode('utf-8')

                print(f"- {field\_info}: {field\_value.strip()}")

                offset += field\_length

- record\_data[offset:offset + field\_length]는 레코드 데이터에서 현재 필드의 데이터를 읽는다. offset부터 offset + field\_length까지의 데이터를 가져온다.

- .rstrip(b'\x00')은 읽어온 데이터 뒤에 있는 NULL 바이트를 제거한다.

- .decode('utf-8')는 바이트 데이터를 UTF-8 문자열로 변환한다.

- print(f"- {field\_info}: {field\_value.strip()}")은 현재 필드의 정보와 읽어온 데이터를 출력한다. .strip()을 사용하여 문자열 앞뒤 공백을 제거한다.

- offset += field\_length은 다음 필드의 시작 위치를 업데이트하여 다음 필드를 읽어오기 위한 변수이다.

# DBF 파일 경로 설정

dbf\_file\_path = 'sample.dbf'

# DBF 파일 읽기 및 필드 정보 및 레코드 출력

read\_dbf\_file(dbf\_file\_path)

4. 결과

|  |
| --- |
| 필드 정보  - id  - NAME  Row 1  - id: 1  - NAME: AAAA  Row 2  - id: 2  - NAME: BBBB  Row 3  - id: 3  - NAME: CCcc  Row 4  - id: 4  - NAME: DDDD  Row 5  - id: 5  - NAME: EEEE  Row 6  - id: 6  - NAME: FFFF  Row 7  - id: 7  - NAME: GGGG  Row 8  - id: 8  - NAME: HHHH  Row 9  - id: 9  - NAME: IIII  Row 10  - id: 10  - NAME: JJJJ  Row 11  - id: 11  - NAME: KKKK  Row 12  - id: 12  - NAME: asdf  Row 13  - id: 13  - NAME: qwer |

5. 결과 화면

 