File

파일 입출력

파일을 열고 닫기

- 1) 파일 입출력 변수는 FILE 형식의 포인터 변수로 선언합니다.
- 2) 파일을 열 때는 fopen() 함수를 이용합니다.
- 3) 파일을 닫을 때는 fclose() 함수를 이용합니다.

```
FILE *fp;
fp = fopen(파일 경로, 접근 방식);
// 파일 관련 처리
fclose(fp);
```

파일을 열고 닫기

- 1) 파일 열기 함수인 fopen() 함수에는 파일 경로와 접근 방식을 설정할 수 있습니다.
- 2) 기본 경로는 현재 프로그램의 경로입니다.
- 3) 가장 많이 사용되는 접근 방식은 다음과 같습니다.

r	파일에 접근하여 데이터를 읽습니다.
W	파일에 접근하여 데이터를 기록합니다. (파일이 이미 존재하면 덮어쓰기)
а	파일에 접근하여 데이터를 뒤에서부터 기록합니다.

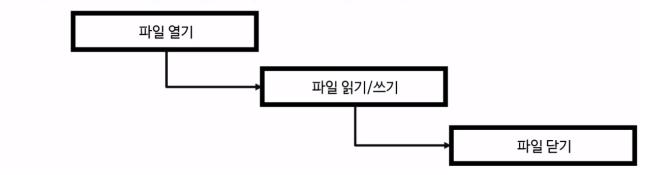
파일 입출력 함수

- 1) 기본적인 입출력을 위해서 printf()와 scanf() 함수를 사용하곤 했습니다.
- 2) 파일 입출력에서는 그 대신에 fprintf()와 fscanf()가 사용됩니다.

```
fprintf(파일 포인터, 서식, 형식지정자);
fscanf(파일 포인터, 서식, 형식지정자);
```

파일 입출력의 과정

- 1) 파일 입출력은 열고, 읽고/쓰고, 닫기의 과정을 철저히 따라야 합니다.
- 2) 파일을 열 때는 파일 포인터가 사용되며, 이는 동적으로 할당된 것입니다.
- 3) 따라서 파일 처리 이후에 파일을 닫아주지 않으면 메모리 누수가 발생합니다.



```
#define _CRT_SECURE_NO_WARNINGS
#include \( \stdio.h \)

int main(void) {
   char s[20] = "Hello World";
   FILE *fp;
   fp = fopen("temp.txt", "w");
   fprintf(fp, "%s\n", s);
   fclose(fp);
   return 0;
}
```

● 파일 입출력 또한 입력으로 보기 때문에, 컴퓨터 시스템 에

안좋은 영향을 줄 수 있는 공격으로 볼 수 있다..

File 경로 가져오기

```
std::tstring GetFilePathFromHandle(HANDLE hFile)
{
        TCHAR tszFilename[MAX_PATH + 1];
        HANDLE hFileMap;
        // Get the file size.
        DWORD dwFileSizeHi = 0;
        DWORD dwFileSizeLo = GetFileSize(hFile, &dwFileSizeH
i);
        std::tstring strRet;
        if (dwFileSizeLo == 0 && dwFileSizeHi == 0)
        {
                return FALSE;
        }
        // Create a file mapping object.
        hFileMap = ::CreateFileMappingA(hFile,
                NULL,
                PAGE_READONLY,
                0,
                1,
                NULL);
        if (hFileMap)
        {
                // Create a file mapping to get the file name.
                void* pMem = ::MapViewOfFile(hFileMap, FILE_MA
P_READ, 0, 0, 1);
```

```
if (pMem)
                 {
                         if (GetMappedFileName(::GetCurrentProc
ess(),
                                  pMem,
                                  tszFilename,
                                  MAX_PATH))
                         {
                                  // Translate path with device
name to drive letters.
                                  TCHAR szTemp[MAX_PATH];
                                  szTemp[0] = ' \setminus 0';
                                  if (GetLogicalDriveStrings(MAX
_PATH - 1, szTemp))
                                  {
                                          TCHAR szName[MAX_PAT
H];
                                          TCHAR szDrive[3] = TEX
T(":");
                                          BOOL bFound = FALSE;
                                          TCHAR* p = szTemp;
                                          do
                                          {
                                                  // Copy the dr
ive letter to the template string
                                                   *szDrive = *p;
```

```
// Look up eac
h device name
                                                  if (QueryDosDe
vice(szDrive, szName, MAX_PATH))
                                                  {
                                                           size_t
uNameLen = _tcslen(szName);
                                                          if (uN
ameLen < MAX_PATH)</pre>
                                                           {
  bFound = _tcsnicmp(tszFilename, szName, uNameLen) == 0
          && *(tszFilename + uNameLen) == _T('\\');
  if (bFound)
          strRet = core::Format(L"%s%s", szDrive, tszFilename
+ uNameLen);
                                                          }
                                                  }
                                                  // Go to the n
ext NULL character.
                                                  while (*p++);
                                          } while (!bFound && *
p); // end of string
                                 }
                         }
```

```
::UnmapViewOfFile(pMem);
}

CloseHandle(hFileMap);
}

return strRet;
}
```

File 경로 분할

```
std::tstring GetFileNameFromPath(LPCTSTR pszPath)
{
         TCHAR tszDrive[MAX_PATH];
         TCHAR tszDir[MAX_PATH];
         TCHAR tszFile[MAX_PATH];
         TCHAR tszExt[MAX_PATH];
         __wsplitpath_s(pszPath, tszDrive, MAX_PATH, tszDir, MAX_PATH, tszFile, MAX_PATH, tszExt, MAX_PATH);
         return strRet;
}
```