파일 패킹 < C 언어 >

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>


//-------------------------------------------------------------
//   패킹파일 헤더
//
//  Type         4 Byte          : 0x99886655
//  FileNum      4 Byte          : 내부 파일 개수
//
// ---- 내부 파일 이름 및 사이즈 ( FileNum 만큼 반복 ) ----------
//
//      Size            4 Byte          : 파일 사이즈
//      char            128 Byte        : 파일 이름
//
//-------------------------------------------------------------
//  파일 데이터 FileNum * Size
//

//---------------------------------
// 패킹 파일 가장 앞 해더
//---------------------------------
typedef struct st_PACK_HEADER_
{
        unsigned int iType;             // 0x99886655 이 들어감.
        int iFileNum;

} stPACK_HEADER;



//---------------------------------
// 내부 파일 정보
//---------------------------------
typedef struct st_PACK_FILEINFO_
{
        int iFileSize;
        char szFileName[128];


} stPACK_FILEINFO;
```



```
//---------------------------------
// 실제 파일 패킹 처리 작업
//---------------------------------
void FilePacking(char *szPackingFile, int iElementFileNum, char **szElementNameArray)
{
        FILE *pPackingFile;
        FILE **pElementFile;

        stPACK_HEADER PackHeader;
        stPACK_FILEINFO FileInfo;

        char *pBuffer = NULL;

        int iCnt;
        int iMaxFile = 0;
        int iFileSize;

        //-----------------------------------------------
        // 패킹파일 헤더 셋팅
        //-----------------------------------------------
        PackHeader.iType = 0x99886655;
        PackHeader.iFileNum = iElementFileNum;



        //-----------------------------------------------
        // 패킹 대상 파일 오픈
        //-----------------------------------------------
        pPackingFile = fopen(szPackingFile, "w+b");
        if (pPackingFile == NULL)
        {
                printf("패킹파일 %s 을 열 수 없습니다", szPackingFile);
                return;
        }

        //-----------------------------------------------
        // 요소파일용 FILE 포인터 배열 생성
        //-----------------------------------------------
        pElementFile = (FILE **)malloc(sizeof(FILE *) * iElementFileNum);

        //-----------------------------------------------
        // 요소파일을 모두 오픈
        //-----------------------------------------------
        for (iCnt = 0; iCnt < iElementFileNum; iCnt++)
        {
                pElementFile[iCnt] = fopen(szElementNameArray[iCnt], "r+b");
                if (pElementFile[iCnt] == NULL)
                {
                        printf("요소파일 %s 을 열 수 없습니다", szElementNameArray[iCnt]);
                        fclose(pPackingFile);
                        free(pElementFile);
                        return;
                }
        }
```

```
//-------------------------------------------------
// 먼저 패킹파일 헤더 저장
//-------------------------------------------------
fwrite(&PackHeader, sizeof(stPACK_HEADER), 1, pPackingFile);

//-------------------------------------------------
// 각 파일들 사이즈 구해서 파일정보 저장
//-------------------------------------------------
for (iCnt = 0; iCnt < iElementFileNum; iCnt++)
{
        fseek(pElementFile[iCnt], 0, SEEK_END);
        FileInfo.iFileSize = ftell(pElementFile[iCnt]);
        strcpy(FileInfo.szFileName, szElementNameArray[iCnt]);

        fwrite(&FileInfo, sizeof(FileInfo), 1, pPackingFile);

        if (iMaxFile < FileInfo.iFileSize)
                iMaxFile = FileInfo.iFileSize;
}


//-------------------------------------------------
// 각 파일들 내용을 읽어서 패킹파일에 저장
//-------------------------------------------------
pBuffer = malloc(iMaxFile);

for (iCnt = 0; iCnt < iElementFileNum; iCnt++)
{
        fseek(pElementFile[iCnt], 0, SEEK_END);
        iFileSize = ftell(pElementFile[iCnt]);
        rewind(pElementFile[iCnt]);

        fread(pBuffer, iFileSize, 1, pElementFile[iCnt]);
        fwrite(pBuffer, iFileSize, 1, pPackingFile);
        fclose(pElementFile[iCnt]);
}


fclose(pPackingFile);

free(pBuffer);
free(pElementFile);
}
```



```
//-------------------------------
// 실제 파일 분리 작업
//-------------------------------
void FileUnpacking(char *szFile)
{
        FILE *pPackingFile;
        FILE *pSplitFile;
        stPACK_HEADER PackHeader;
        stPACK_FILEINFO *FileInfo;

        char *pBuffer = NULL;
        int iCnt;
        int iRead = 0;
        int iMaxFile = 0;

        //-------------------------------------------------
        // 패킹 파일 오픈
        //-------------------------------------------------
        pPackingFile = fopen(szFile, "r+b");
        if (pPackingFile == NULL)
        {
                printf("패킹파일 %s 을 열 수 없습니다", szFile);
                return;
        }

        //-------------------------------------------------
        // 패킹파일 헤더 읽기.
        //-------------------------------------------------
        iRead = fread(&PackHeader, sizeof(stPACK_HEADER), 1, pPackingFile);
        if (iRead != 1)
        {
                printf("패킹파일 헤더를 읽을 수 없습니다");
                fclose(pPackingFile);
                return;
        }

        //-------------------------------------------------
        // 헤더 타입 체크
        //-------------------------------------------------
        if (PackHeader.iType != 0x99886655)
        {
                printf("패킹파일 헤더를 읽을 수 없습니다");
                fclose(pPackingFile);
                return;
        }

        //-------------------------------------------------
        // 요소파일 개수만큼 파일정보 읽기.
        //-------------------------------------------------
        iRead = sizeof(stPACK_FILEINFO) * PackHeader.iFileNum;
        FileInfo = malloc(iRead);
        memset(FileInfo, 0, iRead);

        iRead = fread(FileInfo, iRead, 1, pPackingFile);
```

```c
	if (iRead != 1)
	{
		printf("패킹파일 파일 정보를 읽을 수 없습니다");
		fclose(pPackingFile);
		free(FileInfo);
		return;
	}

	//--------------------------------------------------------
	// 요소파일의 최대 사이즈 구하기.  가장 큰 파일 기준으로 메모리 버퍼를 잡기 위해서
	//--------------------------------------------------------
	for (iCnt = 0; iCnt < PackHeader.iFileNum; iCnt++)
	{
		if (iMaxFile < FileInfo[iCnt].iFileSize)
			iMaxFile = FileInfo[iCnt].iFileSize;
	}
	pBuffer = malloc(iMaxFile);

	//--------------------------------------------------------
	// 읽혀진 파일 정보 개수대로 돌면서 파일 분리.
	//--------------------------------------------------------
	for (iCnt = 0; iCnt < PackHeader.iFileNum; iCnt++)
	{
		printf("%s 파일 | 사이즈 : %d Wn", FileInfo[iCnt].szFileName, FileInfo[iCnt].iFileSize);

		pSplitFile = fopen(FileInfo[iCnt].szFileName, "wb");
		fread(pBuffer, FileInfo[iCnt].iFileSize, 1, pPackingFile);
		fwrite(pBuffer, FileInfo[iCnt].iFileSize, 1, pSplitFile);
		fclose(pSplitFile);
	}

	free(pBuffer);
	free(FileInfo);
	fclose(pPackingFile);

}


//----------------------------------------
// 파일 패킹모드
//----------------------------------------
void Mode_Packing(void)
{
	int iFileNum;
	int iCnt;
	char **szNameArray;
	char szPackingFile[128] = "";

	//------------------------------------------------
	// 총 패킹 파일의 개수 입력
	//------------------------------------------------
	printf("패킹 할 파일 개수를 입력 해주세요 : ");
	scanf("%d", &iFileNum);
	rewind(stdin);
```



```c
	//------------------------------------------------
	// 파일 개수에 맞게끔 파일이름 입력받을 버퍼 생성.
	//
	// char [128] 을 FileNum 만큼 생성.
	//
	// 방법 하나 . 지금처럼 최대 개수를 입력받아 그만큼 동적생성
	// 방법 둘 . 파일 이름을 링크드 리스트로 관리
	// 방법 셋 . 최대치를 고정하여 정적 생성
	//------------------------------------------------

	//------------------------------------------------
	// char[128] 을 여러개 가지기 위한 char ** 배열 생성.
	//
	// szNameArray[0] - char *
	// szNameArray[1] - char *
	// szNameArray[2] - char *
	// szNameArray[3] - char *
	// szNameArray[4] - char *
	//
	// 이런 식으로 각 배열 요소에 char * 가 들어간다.
	//------------------------------------------------
	szNameArray = (char **)malloc(sizeof(char *) * iFileNum);

	for (iCnt = 0; iCnt < iFileNum; iCnt++)
	{
		//------------------------------------------------
		// szNameArray[X] 가 char * 였으므로 여기에 char[128] 을 만들어준다.
		// 그리고 거기다 파일 이름을 받는다.
		//------------------------------------------------
		szNameArray[iCnt] = (char *)malloc(128);
		memset(szNameArray[iCnt], 0, 128);

		printf("%d 번 파일 이름 : ", iCnt);
		scanf("%s", szNameArray[iCnt]);
		rewind(stdin);

		printf("Wn");
	}

	//------------------------------------------------
	// 요소 파일들 이름 받기는 끝 !
	//
	// 이 파일들을 모아서 묶음파일 이름 받기
	//------------------------------------------------
	printf("패킹될 파일 이름 : ");
	scanf("%s", szPackingFile);
	rewind(stdin);

	//------------------------------------------------
	// 패킹 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
	//------------------------------------------------
	FilePacking(szPackingFile, iFileNum, szNameArray);
```

```
	//-------------------------------------------------------------------
	// 위에서 파일 이름을 받기위해 동적할당 했던 메모리들 해제
	//-------------------------------------------------------------------
	for (iCnt = 0; iCnt < iFileNum; iCnt++)
	{
		free(szNameArray[iCnt]);
	}
	free(szNameArray);
}


void Mode_Unpacking(void)
{
	char szPackingFile[128] = "";

	//-----------------------------------------
	// 총 패킹 파일의 개수 입력
	//-----------------------------------------
	printf("분리 할 패킹 파일 이름을 입력 해주세요 : ");
	scanf("%s", szPackingFile);
	rewind(stdin);

	//-----------------------------------------
	// 언패킹 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
	//-----------------------------------------
	FileUnpacking(szPackingFile);
}


void main(void)
{
	int iMode;

	//-----------------------------------------
	// 파일 패킹모드, 풀기모드 선택
	//-----------------------------------------
	printf("1.패킹하기 Wn2.패킹풀기Wn:");
	scanf("%d", &iMode);

	printf("Wn");

	switch (iMode)
	{
	case 1:
		Mode_Packing();
		break;

	case 2:
		Mode_Unpacking();
		break;
	}

}
```