CSED101. Programming & Problem solving

Spring, 2017

Programming Assignment #4

(70 points)

김인한(kiminhan@postech.ac.kr)

- ***Due***: **2017.05.18 23:59**
- ***Development Environment:*** **Windows Visual Studio 2015**

- ***제출물***
  - **C Code files (mystring.h, mystring.c, assn4.c)**
    - 프로그램의 소스 코드를 이해하기 쉽도록 반드시 **주석**을 붙일 것.
  - **보고서 파일** (.doc(x) or .hwp) 예) assn4.doc(x) 또는 assn4.hwp
    - AssnReadMe.pdf 를 참조하여 작성할 것.
    - 보고서는 Problem2에 대해서만 작성할 것.
    - **각각의 기능에 대한 프로그램 실행 화면을 캡쳐하여 보고서에 포함시키고 간단히 설명할 것.**
    - **명예서약(Honor code)**: 보고서 표지에 명예서약이 없는 경우는 과제를 제출하지 않은 것으로 처리한다.
    - 소스코드와 보고서 파일을 LMS를 이용하여 제출한다.

- ***주의사항***
  - 각 문제에 해당하는 요구사항을 반드시 지킬 것.
  - 모든 문제의 출력 형식은 아래의 예시들과 동일해야 하며, 같지 않을 시는 감점이 된다.
  - 각 문제에 제시되어 있는 파일이름으로 제출 할 것. 그 외의 다른 이름으로 제출하면 감점 또는 0점 처리된다.
  - 컴파일 & 실행이 안되면 무조건 0점 처리된다.
  - **하루 late시 20%가 감점되며, 3일 이상 지나면 받지 않는다. (0점 처리)**
  - 부정행위에 관한 규정은 POSTECH 전자컴퓨터공학부 학부위원회의 'POSTECH 전자컴퓨터공학부 부정행위 정의'를 따른다. (LMS의 과목 공지사항의 제목 [document about cheating]의 첨부파일인 disciplinary.pdf를 참조할 것.)
  - 이번 과제에서는 추가 기능 구현에 대한 추가 점수는 없습니다.
  - **과제 작성시 전역변수는 사용할 수 없으며, 요구되는 기능을 사용자 정의 함수로 적절히 분리하여 코드를 작성하여야 합니다. (main() 함수만 사용하는 경우 감점이 됩니다.)**

■ Problem 1: 문자열 처리 함수 (5점)

[문제)

C 언어에서 제공되는 문자열 함수는 "string.h" 라이브러리에서 제공된다. 문자열 라이브러리에 포함되어 있는 함수 중 일부를 직접 작성해 보자.

(설명)

제공된 assn4.zip 의 압축을 풀면 **mystring.h** 와 **mystring.c** 가 주어진다.

mystring.h 는 아래와 같은 문자열 처리 함수의 선언을 포함하고 있다. (그대로 사용하고 변경하지 말 것)

```
#ifndef MY_STRING_H
#define MY_STRING_H

int mystrlen(char *str);
char *mystrcpy(char *toStr, char *fromStr);
int mystrcmp(char *str1, char *str2);
char *mystrcat(char *str1, char *str2);

#endif
```

위 함수의 구현부를 포함한 mystring.c 를 작성하라. 각 함수의 정의는 다음과 같다.

(1) int mystrlen(char *str)

NULL 문자를 제외한 문자열의 길이를 반환한다. 빈 문자열의 경우 0 을 반환한다.

예제)

```
printf("%d\n", mystrlen("cs101")); // 결과: 5
```

(2) char *mystrcpy(char *toStr, char *fromStr)

문자열 복사 함수로 NULL 문자를 포함한 문자열 fromStr 를 문자열 toStr 에 복사한 후, 문자열 toStr 의 시작 주소를 반환한다.

예제)

```
char str[256];
printf("%s\n", mystrcpy(str, "Good Day")); // 결과: Good Day
printf("%s\n", mystrcpy(str, "Hello"));    // 결과: Hello
```

(3) int mystrcmp(char *str1, char *str2)

문자열 str1 과 str2 의 대소를 비교한다(대소문자 구분). 비교기준은 아스키코드표의 값을 기준으로 한다.

각 문자열의 첫 번째 문자부터 비교를 시작한다. 만일 같다면 두 문자가 다를 때까지나 NULL 에 도달할 때까지 계속 비교를 수행한다.

- 비교 중 str1 의 문자가 작을 경우 -1, 클 경우 1 을 반환한다.
- 문자열이 길이가 같고 모든 문자가 같을 경우, 0 을 반환한다.
- 비교 중 하나의 문자열이 먼저 끝에 도달할 경우, 끝난 문자열을 작다고 판단한다.

예제)

```
printf("%d\n", mystrcmp("csed101", "csed103")); // 결과: -1
printf("%d\n", mystrcmp("csed", "Csed"));       // 결과: 1
printf("%d\n", mystrcmp("csed", "cse"));        // 결과: 1
printf("%d\n", mystrcmp("csed", "csed103"));    // 결과: -1
```

(4) char *mystrcat(char *str1, char *str2)

문자열 연결함수로 str1 의 끝에 str2 를 이어 붙인다. 즉, 문자열 str1 뒤의 NULL 문자는 str2 의 첫 번째 문자로 덮어 씌워지고 str2 의 NULL 문자는 남는다. 문자열 연결 후, 문자열 str1 의 시작 주소를 반환한다.

예제)

```
char str[256] = "Hello";
mystrcat(str, ", World");
printf("%s₩n", str); // 결과: Hello, World
```

■ Problem 2: 기본 이미지 처리(Image Processing) 기능 구현 (65점)

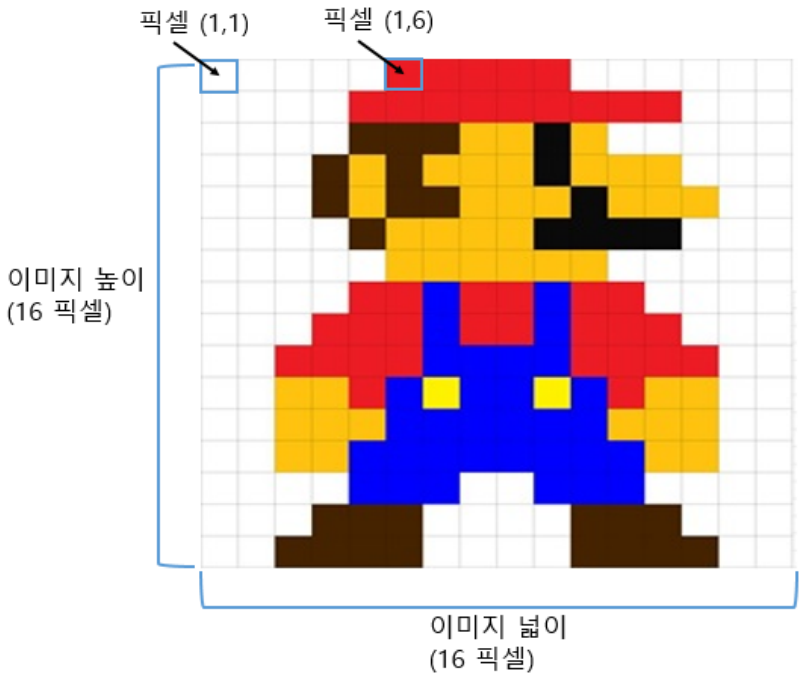
[목적]

- 이미지 처리에 사용되는 간단한 알고리즘들을 구현해보며 포인터의 사용법을 익힌다.

[배경 지식 및 환경 설명]

1. 이미지의 구성 형태

이미지는 픽셀들로 이루어지며, 각 픽셀의 값들은 이미지의 색 정보를 담고 있다. 이미지의 높이와 넓이는 픽셀의 개수로 정해진다. 큰 이미지일수록 이미지를 구성하는 픽셀들의 개수가 많고, 따라서 이미지의 용량도 커진다. 이미지의 총 픽셀 개수는 (이미지 높이 X 이미지 넓이) 가 된다.



<그림 1> 16x16 픽셀로 구성된 예시 이미지

위 이미지는 높이 16 픽셀, 넓이 16 픽셀로 구성된 간단한 이미지이다. 각각의 픽셀 값에 따라 다른 색을 나타낸다.

이미지는 2 차원 행렬(Matrix) 구조로 구성되기 때문에, 각 픽셀은 (행, 열)로 부를 수 있다. 예를 들어, <그림 1>에서 (1,1) 픽셀의 색은 하얀색, (1,6) 픽셀의 색은 빨간색이라 말할 수 있다.

2. 제공 txt 파일 형식

Assignment 를 위해 제공되는 "이미지명.txt" 파일은 "이미지명.png" 파일의 픽셀 값을 text 파일로 나타낸 것이다. "이미지명"은 띄어쓰기 없이 영어, 숫자, 기호(_)의 조합으로 이루어져 있으며 최대 길이는 30 글자를 넘지 않는다고 가정한다.

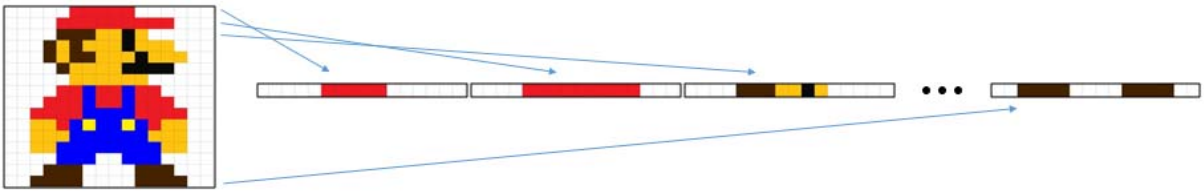
파일의 첫 번째 줄에는 이미지의 기본 정보가 기록되어있다. 띄어쓰기로 구분되는 세 개의 숫자 값이 적혀 있으며, 첫 번째 값은 이미지의 높이, 두 번째 값은 이미지의 넓이, 세 번째 값은 이미지의 압축 여부(1: 압축된 이미지, 0: 압축되지 않은 이미지)이다.

예) "kkobu2.txt"의 첫 번째 줄은 아래와 같으며,

80 80 0

이미지 kkobu2 는 80x80 이미지며, 압축되지 않았음을 의미한다.

파일의 두 번째 줄에는 이미지의 첫 번째 행부터 마지막 행까지의 픽셀 값이 띄어쓰기로 구분해 나열돼 있다.



<그림 2> "이미지명.txt" 파일의 두 번째 줄에 나열된 픽셀 값 나열 방법

즉, 이미지를 구성하는 픽셀 값들이 "이미지명.txt" 파일의 두 번째 줄에 <그림 2>에서 나타나는 순서대로 나열되어 있다. 이미지를 구성하는 픽셀 수가 (이미지 높이 X 이미지 넓이)이므로, 두 번째 줄에 나열된 픽셀 값의 수도 (이미지 높이 X 이미지 넓이)와 동일하다. 제공되는 이미지 파일의 두 번째 줄에 나열되어 있는 픽셀 값의 개수는 항상 첫 번째 줄에 적혀 있는 (이미지 높이 X 이미지 넓이)의 결과 값과 같다.

예) "kkobu2.txt"의 두 번째 줄에는 80 x 80 = 1600 개의 픽셀 값이 띄어쓰기로 구분되어 나열되어 있다.

156 152 158 155 159 154 ... 41 44 151 157 (총 1600 개의 값)

3. 콘솔 화면에 픽셀 색 출력 및 환경 설정

콘솔에 색을 출력하기 위해서는 "Windows.h"에서 구현되어 있는 "SetConsoleTextAttribute()" 함수를 사용해야 한다.

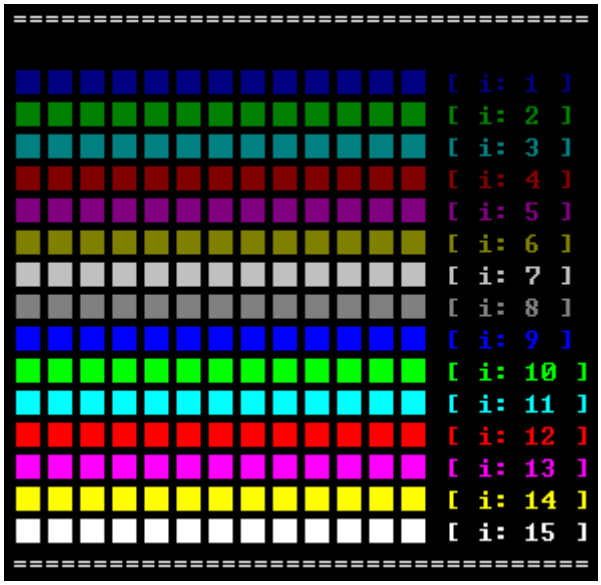
"SetconsoleTextAttribute()" 함수를 호출 하면, 그 뒤의 모든 글자는 지정된 색으로 콘솔에 출력이 된다.

함수를 호출 하는 방법은

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 색 인덱스);

인데, 마지막 파라미터인 "색 인덱스"에 입력되는 값에 따라 다른 색이 출력된다. 입력 범위는 0 ~ 15 이다. 아래의 표는 "색 인덱스" 값에 따라 어떤 색의 글자가 출력 되는지 보여주는 예시코드이며, <그림 3>은 예시 코드의 결과를 보여준다.

```
#include <stdio.h>
#include <Windows.h>
int main()
{
	int i;
	printf("====================================\n");
	for (i = 0; i <= 15; i++)
	{
		SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), i);
		printf("■■■■■■■■■■■■■■■ [ i: %d ]\n", i);	//"■" 13개 출력
	}
	printf("====================================\n");
}
```



<그림 3> 예시 코드 실행 결과

"색 인덱스"가 "0"일때는 검정색의 글자 색을 출력하는데, 콘솔의 배경과 동일한 색이라 보이지 않음을 유의한다.

콘솔에 픽셀을 출력하는 기호는 "■"로 한다. (옆의 네모 기호를 복사하여 printf 함수에 사용)

또한, 입력되는 이미지의 크기는 최대 80 x 80 으로 가정 할 것이며, 최대 크기의 이미지를 출력할 수 있는 적절한 크기로 콘솔 크기를 설정 하여야 한다. 이를 위해 main 함수 첫 번째 줄에 아래와 같이 "system()" 함수를 추가하자.

```
int main()
{
        system("mode con:cols=170 lines=95");
```

4. 양자화(Quantization) 및 픽셀 값에 따른 색

이미지 처리에서 특정 범위의 픽셀 값을 하나의 값으로 간주하는 것을 양자화라고 한다. 제공되는 "이미지명.txt" 파일의 두 번째 줄에는 이미지의 픽셀 값들이 나열되어 있는데, 이 범위는 0 ~ 159 이다.

각 픽셀 값은 아래의 표와 같이 양자화 하여 "SetconsoleTextAttribute()" 함수의 "색 인덱스"로 사용하도록 한다.

픽셀 값	색 인덱스	픽셀 값	색 인덱스
0 ~ 9	0	80 ~ 89	8
10 ~ 19	1	90 ~ 99	9
20 ~ 29	2	100 ~ 109	10

30 ~ 39	3	110 ~ 119	11
40 ~ 49	4	120 ~ 129	12
50 ~ 59	5	130 ~ 139	13
60 ~ 69	6	140 ~ 149	14
70 ~ 79	7	150 ~ 159	15

예) "kkobu2.tx"t 의 (1,1) 값은 156 이다. 이 값은 15 로 양자화 되며, "SetConsoleTextAttribute()" 함수의 색 인덱스로 15 가 사용된다. 이 후, 콘솔에 출력되는 글자는 하얀색이 된다.

[구현 기능 설명]

1. 메뉴 출력 및 입력 기능

프로그램이 실행 되면 아래의 메뉴가 출력되며, 하나의 기능을 완료하면 메뉴는 다시 출력되도록 한다. (메뉴명의 띄어쓰기나 "="의 개수가 <그림 4>와 달라도 감점되지 않는다.)

<그림 4> 메뉴 출력 및 입력 화면

메뉴가 출력 된 뒤에는, 사용자가 원하는 메뉴번호를 입력할 수 있도록 한다.

2. 이미지 로드 기능

"2"를 입력하여 "이미지 로드" 기능을 선택할 경우, 이미지 이름을 입력 받고, 이미지가 유효할 경우 아래의 구조체에 형식에 맞게 로드 한다.

```
typedef struct
{
        int width;      // 이미지의 넓이
        int height;     // 이미지의 높이
        int compFlag;   // 압축을 하였는가 0: No, 1: Yes
        char *imgName; // 이미지의 이름
        int **imgValue;// 이미지의 픽셀 값들
} IMGMAT;
```

구조체의 width, height, compFlag 에 들어갈 값은 "배경 지식 및 환경 설명"의 "2. 제공 txt 파일 형식"을 참고한다.

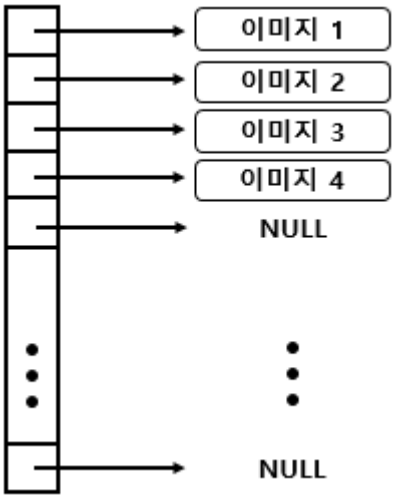
imgName 은 확장자(".txt")를 포함한 이미지의 이름을 저장하고, 이미지마다 이름의 길이가 다르므로 이미지의 이름만큼만 메모리를 동적으로 할당하여 메모리의 낭비가 없도록 한다. 이미지의 이름은 띄어쓰기 없이 영어, 숫자, 기호(_)의 조합으로 이루어져 있으며 최대 길이는 30 글자를 넘지 않는다고 가정한다.

imgValue 는 이미지의 픽셀 값들을 저장할 포인터 변수이다. 이미지마다 크기가 다르므로, 이미지의 높이랑 넓이를 기반으로 하여 2 차원 배열을 동적으로 할당하여 메모리의 낭비가 없도록 한다.

또한, 로드 된 이미지들은 하나의 포인터 배열을 이용해 관리한다.

IMGMAT 구조체를 이용하여 포인터 배열을 선언하고, 이 포인터 배열을 이용해 이미지들을 관리한다. 구조체 포인터 배열은 사용 될 때 메모리를 동적으로 할당 받아 사용 하도록 한다.

이 구조체 포인터 배열의 최대 크기는 10 으로 하며, 로드한 이미지들은 낮은 배열의 인덱스 순서부터 저장한다 <그림 5>.



<그림 5> 구조체 포인터 배열

<그림 6>은 "kkobu1.txt"를 로드 한 예를 보여준다.

<그림 6> 이미지 로드 화면 예

포인터 배열의 최대 크기가 10 이기 때문에, 모든 포인터에 이미지가 할당되면 이미지가 제거되기 전 까지 더 이상 로드 할 수 없도록 한다. (제거 기능은 "5. 이미지 제거 기능"에서 설명)

또한, 동일한 이미지는 중복으로 로드 될 수 없으며, 압축된 이미지를 로드 할 때는 압축을 풀어서 로드 하도록 한다. (압축 기능은 "6. 이미지 압축 기능"에서 설명)

3. 이미지 목록 보기 기능

"1"을 입력 하면, "이미지 로드"기능을 이용해 로드 한 이미지들의 이름을 모두 출력한다. 로드 되는 순서대로 이미지에 번호가 할당 되며, 1 번부터 시작한다.

```
메뉴 번호 입력: 1
[1]번 이미지 이름: pica1.txt
[2]번 이미지 이름: kkobu1.txt
```

<그림 7> 이미지 목록 보기 화면 예

4. 이미지 출력 기능

"3"을 입력하면 출력할 이미지 번호를 입력 받도록 한다.

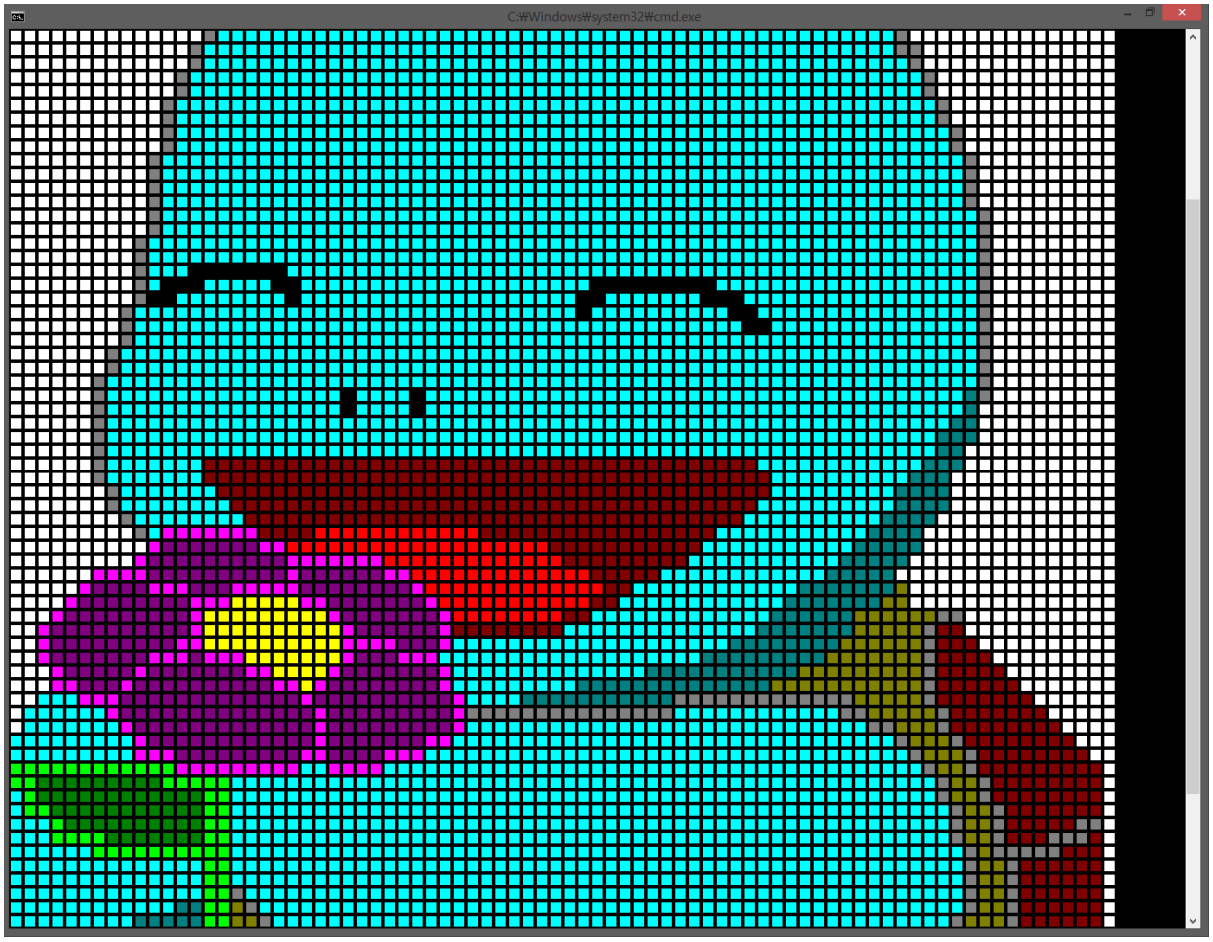
```
메뉴 번호 입력: 3
출력할 이미지 번호나 범위: (ex: 2 or 2-4)
```

<그림 8> 출력 이미지 입력 번호 입력 화면 예

출력할 이미지 번호를 입력받는 방법은 두 가지이다.

1) **단일 번호 입력**: 하나의 이미지 번호만을 입력 받고, 해당하는 번호의 이미지를 출력한다.
2) **범위 번호 입력**: 범위 형식으로 번호를 입력 받고, 해당되는 범위의 이미지를 모두 출력한다. 입력 형식은 "시작번호-끝번호"이며, 공백은 없다고 가정한다. 해당 범위에 이미지가 모두 있을 경우만 출력하도록 한다.

   예를 들어, 이미지를 3 개만 로드 하여서 1~3 번까지밖에 이미지가 없는데, "2-4"를 입력하면 4 번 이미지가 없기 때문에, 하나도 출력되지 않게 한다.

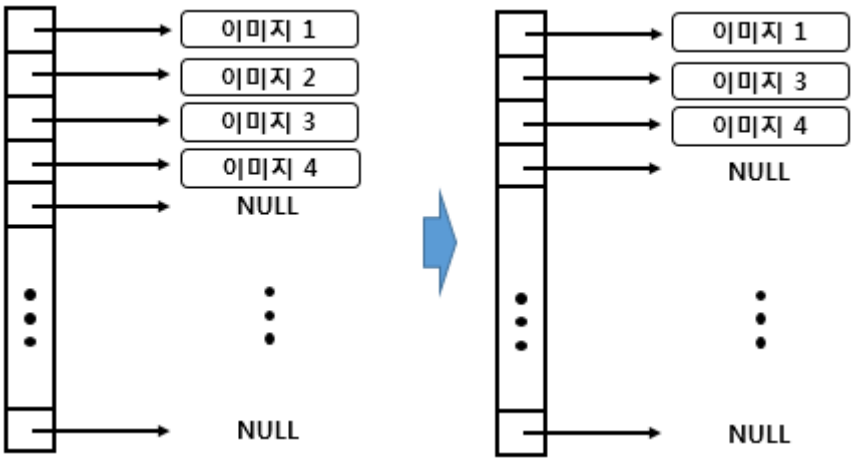


<그림 9> kkobu2.txt 이미지를 출력한 예

5. 이미지 제거 기능

"4"를 입력하면, "이미지 제거"기능을 이용해 목록에 있는 이미지를 제거할 수 있다. 원하는 이미지 번호를 입력하면 제거되게 한다.

만약, 제거하고자 하는 번호 뒤에 이미지들이 있으면, 앞 번호로 이동시킨다.



<그림 10> 이미지 2 를 삭제 하기 전(왼쪽)과 후(오른쪽)의 배열 구조

예) "pica1.txt", "pica2.txt", "pica3.txt", "pica4.txt" 순서로 네 개의 이미지를 로드 했을 경우 목록은 <그림 11 - 1>과 같다.

```
메뉴 번호 입력: 1
[1]번 이미지 이름: pica1.txt
[2]번 이미지 이름: pica2.txt
[3]번 이미지 이름: pica3.txt
[4]번 이미지 이름: pica4.txt
```

<그림 11 - 1> 네 개의 이미지가 로드 된 목록 화면 예

이 때, "4"를 메뉴에서 선택하고, "pica2.txt"를 지우기 위해 "2"를 입력 한다면, 결과는 <그림 11 - 2>과 같아야 한다.

```
메뉴 번호 입력: 4
삭제할 이미지 번호: 2
성공적으로 삭제하였습니다.
========================================
[1] 이미지 목록 보기
[2] 이미지 로드
[3] 이미지 출력
[4] 이미지 제거
[5] 이미지 압축
[6] 이미지 축소
[0] 종료
========================================
메뉴 번호 입력: 1
[1]번 이미지 이름: pica1.txt
[2]번 이미지 이름: pica3.txt
[3]번 이미지 이름: pica4.txt
```

<그림 11 - 2> "2"번 이미지를 제거한 결과 화면 예

이미지 제거 시, 동적으로 할당 받은 메모리들은 모두 할당 해제 해야 한다.

6. 이미지 압축 기능

"5"를 입력하면, 압축할 이미지 번호를 입력 받을 수 있고, 입력 받은 이미지를 압축한다.

이미지 파일의 용량을 줄이는 방법 중 하나는 압축을 하는 것이다. 이미지 압축 방법 중 하나인 "Run-Length Encoding" 방법을 구현하자.

이미지는 보통 유사한 색이 반복되어 특정 부분에 나타난다. 이 특징을 이용하여 압축을 하면, 이미지 파일 용량을 줄일 수 있다.

이미지의 픽셀 값들을 양자화 하였을 때, 같은 색을 나타내는 픽셀 값들은 같은 "색 인덱스" 값으로 양자화 된다.

예) "kkobu2.txt" 이미지의 (1,1) ~ (1, 36) 범위의 픽셀 값들은 모두 150 ~ 159 사이의 값들로 구성되어 있다. 이 값들은 각각 다르지만, "15"로 동일하게 양자화 된다. 즉, "색 인덱스" 15 에 해당되는 값이 36 개가 나열되어 있다는 의미이다.

"Run-Length Encoding" 방법은 특정 값이 몇 개나 연속으로 나타나는지를 encoding 하는 알고리즘이다. 연속으로 동일하게 나타나는 값이 있을 때, "값"과 "연속되어 나타나는 수" 두 개의 값으로 연속되는 수들을 표현하는 것이다.

예) 아래와 같이 20 개의 픽셀 값이 있다고 가정하자. (총 20 개의 수)

152 158 154 157 153 155 16 13 15 16 19 11 150 155 152 159 42 48 49 77

이 픽셀 값은 순서대로 아래와 같이 양자화 된다. (총 20 개의 수)

15 15 15 15 15 15 1 1 1 1 1 1 15 15 15 15 4 4 4 7

위의 양자화된 값들을 encoding 하면, 아래와 같다. (총 10 개의 수)

15 6 1 6 15 4 4 3 7 1

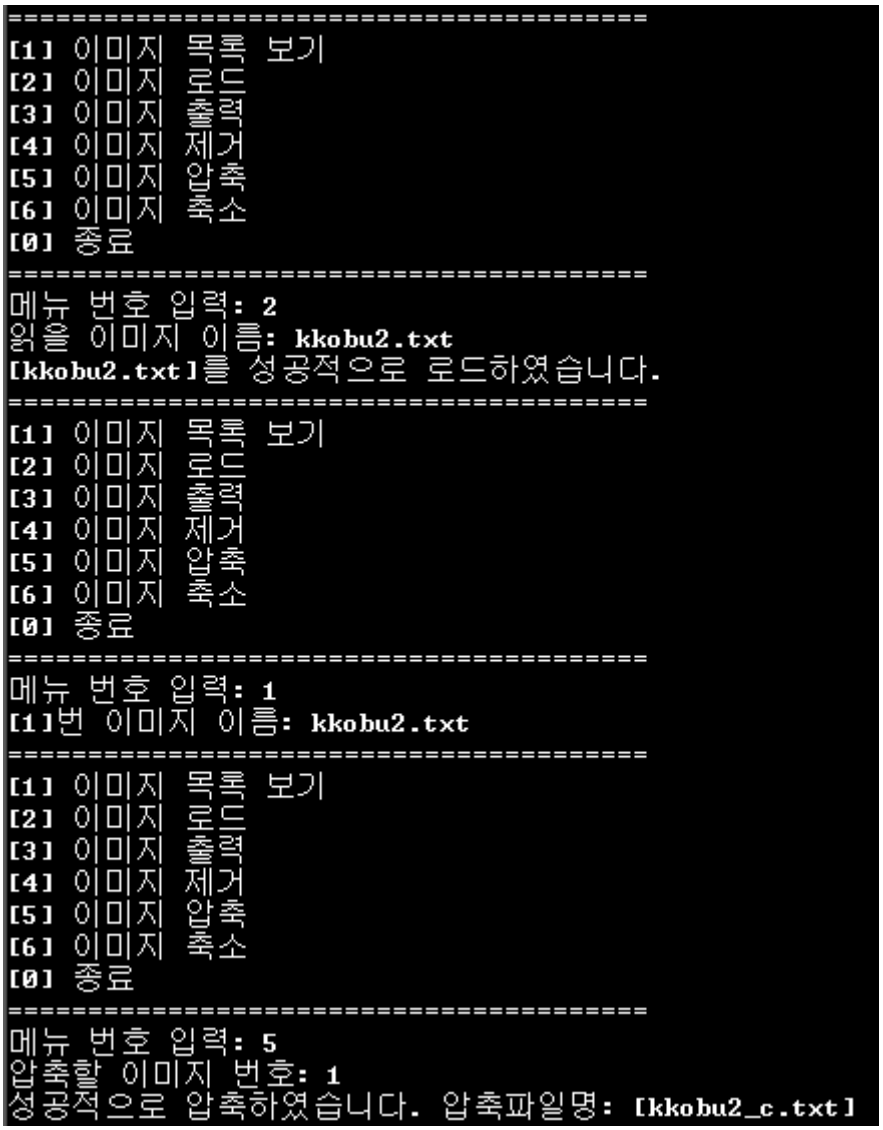
15 가 연속으로 6 번, 1 이 연속으로 6 번, 15 가 연속으로 4 번, 4 가 연속으로 3 번, 7 이 연속으로 1 번 나타남을 의미한다. 이 예에서는, 20 개의 픽셀을 표현하기 위해서 10 개의 값들만 사용되었기 때문에, 크기가 반으로 줄어든 것이다.

이미지를 압축 할 경우, 이미지 이름 뒤에 "_c"가 붙은 text 파일을 생성하여 저장한다. (확장자는 ".txt"를 항상 붙인다.) 기존에 동일한 이미지를 압축 했던 적이 있어, "이미지명_c.txt" 파일이 존재할 경우 덮어쓴다.

압축 이미지 파일도 압축 되지 않은 이미지 파일과 유사한 형식을 유지한다. 즉, 첫 번째 줄에는 "이미지 높이" "이미지 넓이" "1(압축 되었으므로)" 이 순서대로 작성되며, 두 번째 줄에는 encoding 된 값들이 순서대로 나열된다.

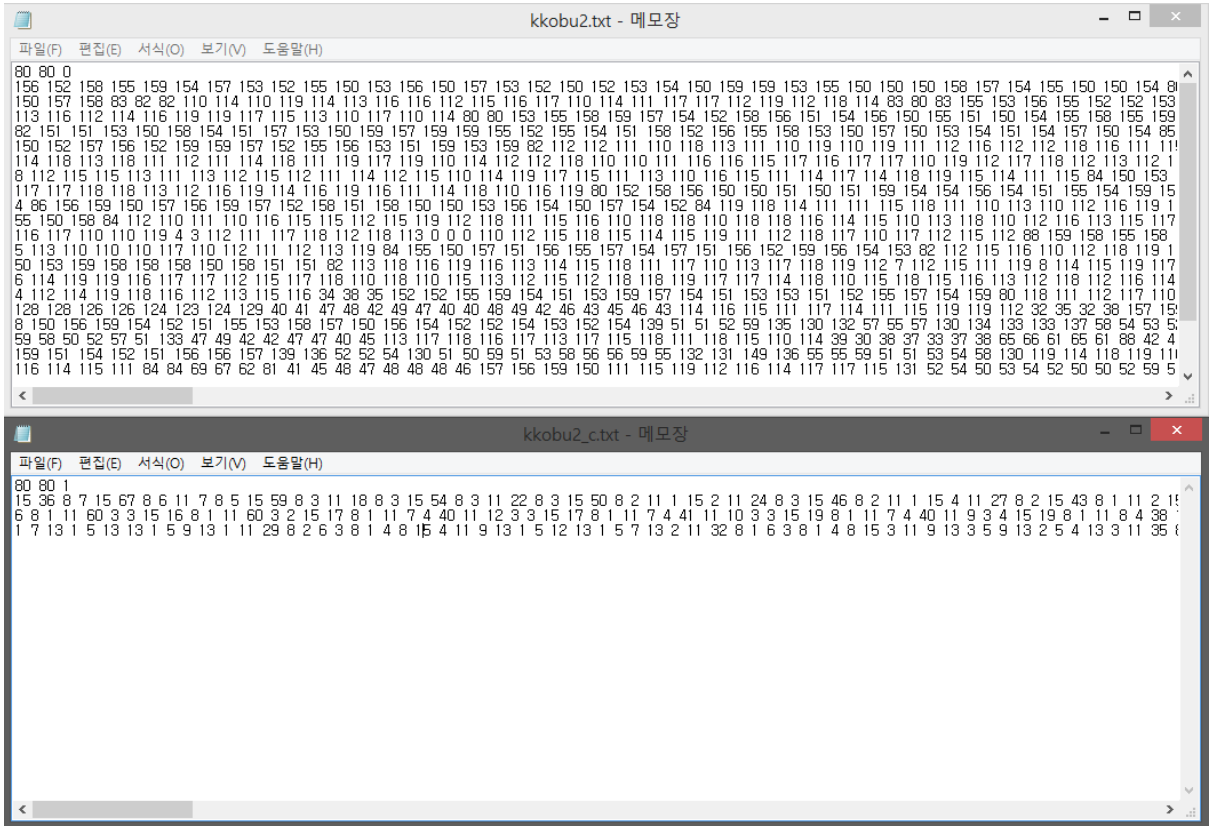
단, 이미 압축된 이미지는 다시 압축할 수 없다.

예) <그림 12>는 "kkobu2.txt" 파일을 로드 하여, 압축하는 과정을 보여준다.



<그림 12> "kkobu2.txt"를 압축하는 과정 예

<그림 13>은 원본 이미지 파일 "kkobu2.txt"와 압축된 이미지 파일 "kkobu2_c.txt"를 메모장에서 불러온 화면이다. (두 번째 줄의 값들은 너무 많은 수가 나열되어 있어 여러 줄처럼 보이나, 실제로는 한 줄로 나열되어 있다.)

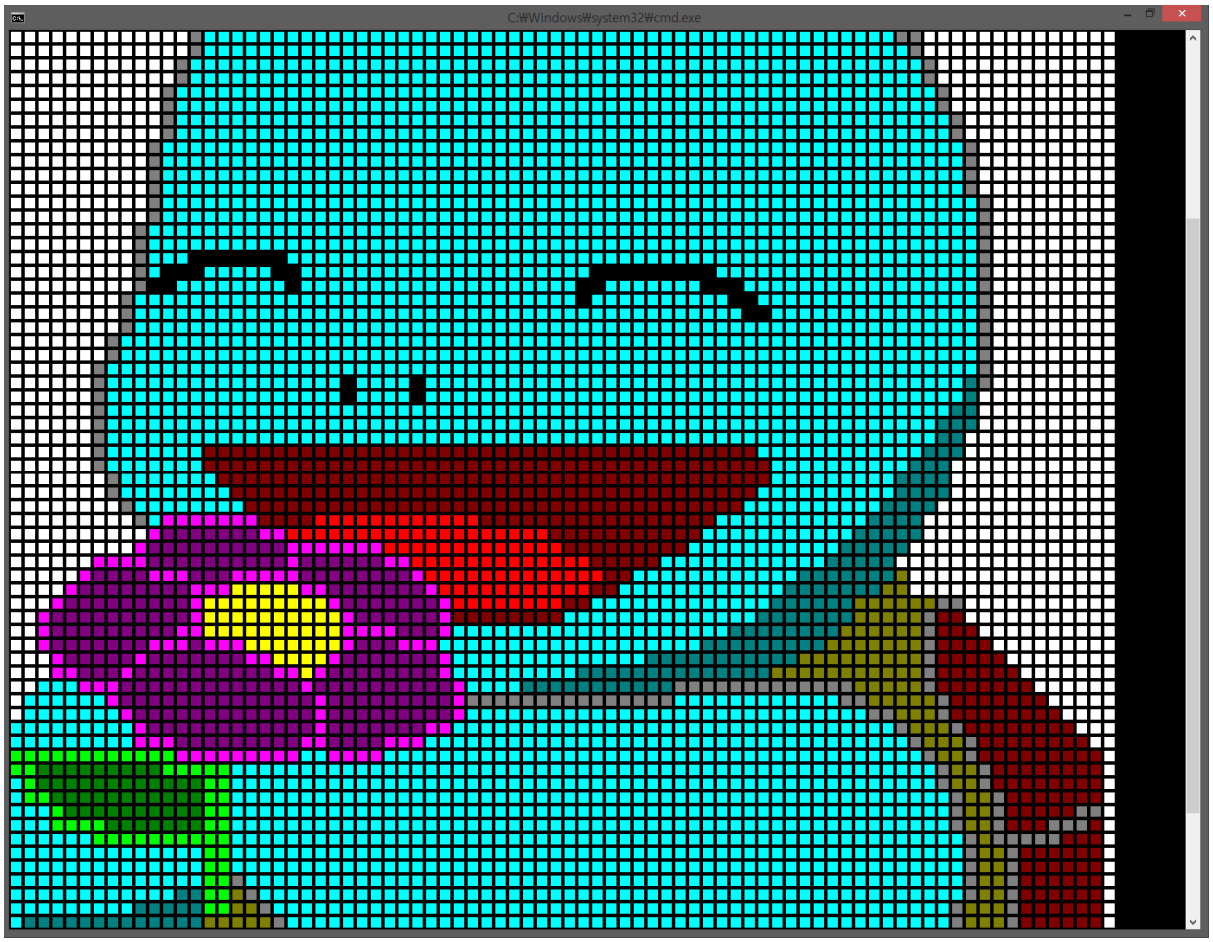


<그림 13> "kkobu2.txt"(위)를 메모장에서 불러온 결과와 "kkobu2_c.txt"(아래)를 메모장에서 불러온 결과의 예

또한, <그림 14>는 "kkobu2.txt"와 "kkobu2_c.txt"의 용량 차이를 보여주기 위해 폴더내의 파일 정보를 캡쳐한 것이다.

<그림 14> 원본 이미지 파일과 압축된 이미지 파일의 용량 차이의 예

마지막으로, <그림 15>는 "kkobu2_c.txt" 이미지 파일을 로드 한 뒤, 출력한 결과이다. 이미지의 압축 및 로드 단계에서의 압축 해제가 정상적으로 이루어졌다면, "kkobu2.txt" 이미지 파일을 로드 한 뒤, 출력한 결과와 동일한 결과가 나온다.



<그림 15> kkobu2_c.txt 이미지를 출력한 예

7. 이미지 축소 기능

"6"를 입력하면, 축소할 이미지 번호를 입력 받을 수 있고, 입력 받은 이미지의 크기를 축소한다. 축소를 원하는 이미지 번호를 입력하면, 해당하는 이미지의 높이와 넓이는 각각 절반으로 축소된다. (80x80 이미지를 축소하면 40x40 이미지가 됨)

이미지를 축소할 경우 이미지 이름 뒤에 "_r"을 붙은 text 파일을 생성하여 저장한다. 기존에 축소를 한 적이 있어, 동일 파일명이 있을 경우 덮어쓴다.

만약, 축소할 이미지가 압축 된 이미지를 로드한 이미지 일 경우, 축소한 결과를 다시 압축하여 저장한다. 즉, 축소할 파일의 원본이 압축이미지이므로, 축소하더라도 원본의 형식을 따라 파일의 내용은 압축파일형식(특정값이 몇 개나 연속되었는지로 표현된 압축파일형식)으로 저장되어야 한다. (제공된 kkobu2_c_r.txt 는 압축된 kkobu2_c.txt 를 축소한 예제이니 참고하시오.)

예를 들어, 압축된 파일을 로드 하였다면 이미지 목록에는 "이미지명_c.txt"이 압축이 풀린 상태로 로드 될 것이며, 이 이미지를 축소할 경우에 축소된 결과를 다시 압축하여 "이미지명_c_r.txt" 이름으로 파일을 저장해야 한다.

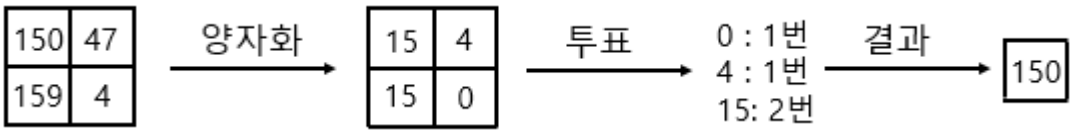
추가적으로, "이미지명_c_r.txt"를 다시 로드하여 축소 할 경우에 생성되는 파일의 이름은 "이미지명_c_r_r.txt"가 된다.

이미지를 축소할 때, 축소된 이미지의 높이나 넓이 중 하나라도 20 픽셀 보다 작으면 축소할 수 없다. 즉, 최소 이미지의 크기는 20x20 이다.
축소된 파일의 형식은 축소 전의 이미지 파일과 동일하다.

이미지를 축소하는 방법은 아래와 같다.

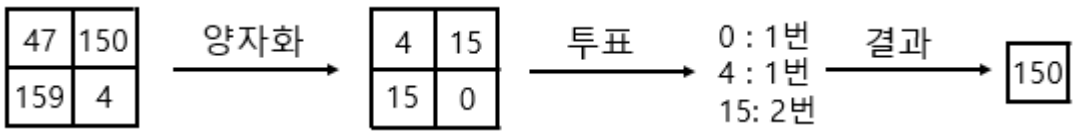
1. 이미지의 i 행, j 열 픽셀 값을 (i, j)라고 하였을 때, (i, j), (i, j+1), (i+1, j), (i+1, j+1) 값이 비교 대상 픽셀 값들이다. 이 4 개의 값을 대표 할 수 있는 하나의 픽셀 값을 찾아 하나의 픽셀로 저장한다. 즉, 크기는 1/4 로 줄어드는 것이다.
2. 위의 4 픽셀 값들의 양자화 값을 계산한다.
3. 양자화된 값들 중 가장 많이 나타난 값의 수를 센다. (일종의 투표)
4. 가장 많이 나타난 양자화 값을 만든 픽셀 값들 중 우선순위가 가장 높은 위치의 값이 (i, j)의 값이 된다.
   우선순위: (i, j) > (i, j+1) > (i+1, j) > (i+1, j+1)
5. 만약, 가장 많이 나타난 양자화 값이 하나가 아니면, 가장 많이 나타난 양자화 값을 만든 픽셀 중 우선순위가 가장 높은 위치의 값이 (i, j)의 값이 된다.

예) 2x2 이미지를 축소하여 1x1 이미지를 만든다고 하자. 이때의 i 와 j 값은 둘 다 1 이다.



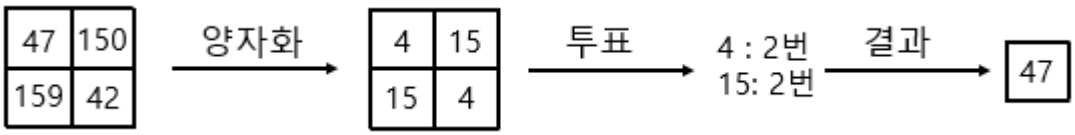
<그림 16> 축소의 예 1

<그림 16>의 예에서는 2x2 이미지를 양자화 하였을 때, 15 가 가장 많이 나타났으며, 압축된 이미지의 (1,1) 픽셀 값은 15 로 양자화된 150, 159 중 우선순위가 높은 150 이 된다.



<그림 17> 축소의 예 2

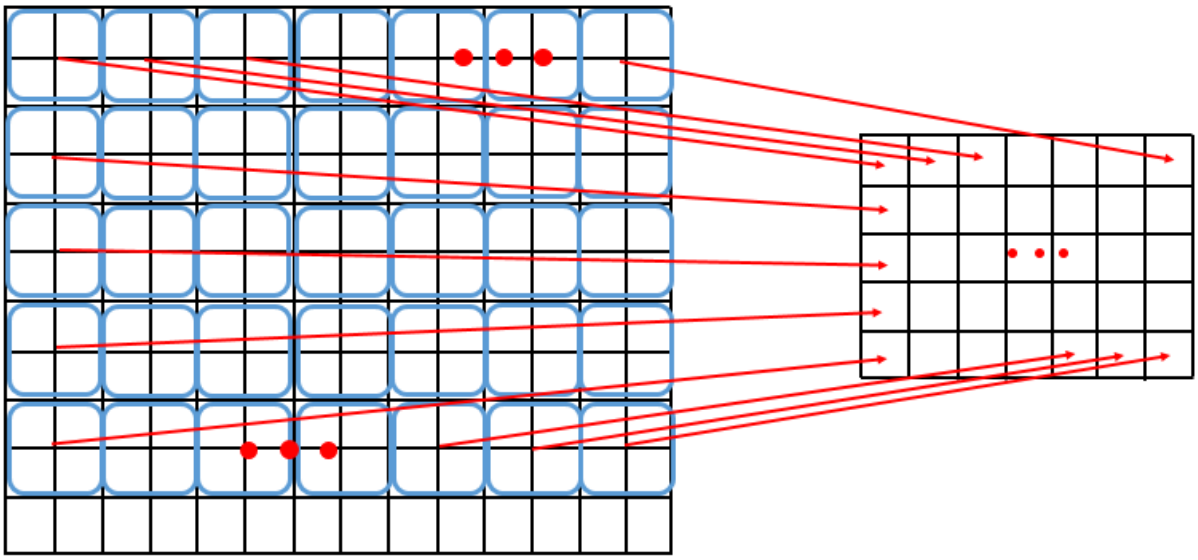
<그림 17>의 예에서는 <그림 16>의 예와 달리 150 과 47 의 위치가 바뀌었으나, 양자화된 결과가 15 가 가장 많고 150 과 159 중에 150 이 우선순위가 더 높기 때문에, 압축된 이미지의 (1, 1) 픽셀 값은 150 이 된다.



<그림 18> 축소의 예 3

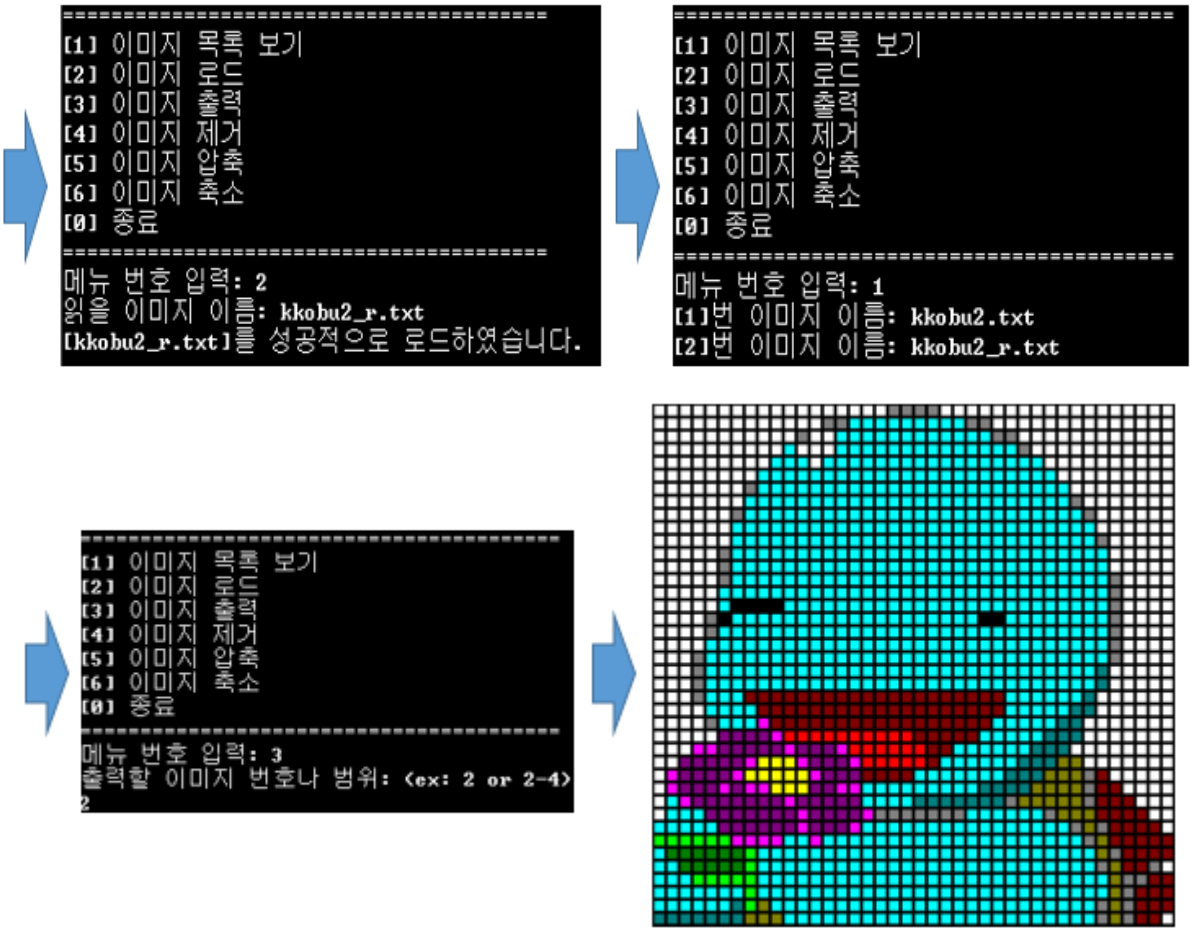
<그림 18>의 예에서는 4 가 2 번, 15 가 2 번 나와서 가장 많이 나타난 양자화 값의 수가 두 개 이다. 이럴 경우 우선순위가 가장 높은 위치에 있는 값인 47 이 결과가 된다.

6. i = 1, j = 1 부터 시작하며, i, j 의 값은 2 씩 증가한다. i 값이 먼저 2 씩 증가하며, i 혹은 i+1 값이 이미지의 넓이를 넘어가면 i 는 1 로 초기화 되고 j 가 2 증가한다. j 혹은 j+1 값이 이미지의 높이를 넘어가면 종료한다.(<그림 19> 참고)
7. i, j 값이 바뀔 때마다 1 ~ 5 번의 방법을 반복하며, 각 결과 값들을 "이미지명_r.txt"파일에 적어준다.



<그림 19> 11x14 크기의 이미지에 축소 하여 5x7 크기의 이미지를 생성. 빨간 화살표는 축소 전 이미지의 영역이 축소 후 할당되는 이미지의 위치를 나타냄. 원본 이미지의 마지막 행을 계산하려면 이미지의 범위를 넘어 가기 때문에 버려짐.

<그림 20>은 "kkobu2.txt"를 축소하여 생성된 "kkobu2_r.txt"를 출력하는 과정과 결과를 보여준다.



<그림 20> 이미지의 축소 과정과 그 결과

이미지를 축소하면, 4 개의 픽셀 값 중 하나의 값만 남겨지고 나머지는 버려지기 때문에 정보의 손실이 발생한다. <그림 9>의 원본 이미지와 <그림 20>의 마지막의 축소된 이미지를 출력한 결과에서 차이를 비교해 보자.

8. 종료 기능

"0"을 입력하여 종료를 할 경우, 동적으로 할당 받았던 모든 메모리를 해제하고 프로그램을 종료한다.

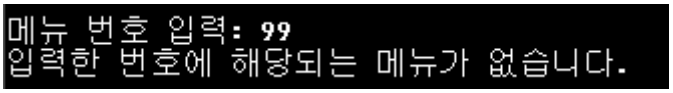
[주의 사항]

- 파일 이름은 "assn4.c"로 저장 할 것
- 보고서는 "assn4.doc" or "assn4.hwp"로 저장 할 것
- 표준 헤더 파일 <string.h>를 include 하여 사용할 수 있다.
- 출력은 위의 그림들과 동일하게 하되, 띄어쓰기나 "="의 개수는 달라도 무관
- 메뉴 번호를 설명과 동일하게 할 것
- 각 기능을 수행하는 함수를 별도로 만들 것
- main() 함수에 최대 크기 10 의 구조체 포인터 배열을 선언하고, 다른 함수에서 필요할 경우 파라미터로 넘겨서 사용 할 것. 아래는 구조체 포인터 배열의 선언 예를 보여줌

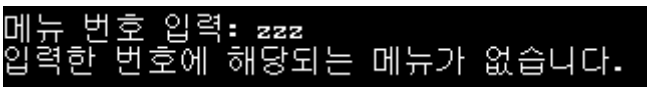
```
#define MAX_LOAD_IMG_SIZE 10
... // 생략
int main()
{
        ... // 생략
        IMGMAT *matArrP[MAX_LOAD_IMG_SIZE] = { NULL };
         ... // 생략
}
```

- 연결리스트(linked list)와 같은 다른 방법을 사용할 경우, 감점 처리한다.
- 위의 설명에서 동적으로 할당 받으라고 명시되어 있는 포인터는 필히 동적으로 할당 받아 사용하며, 더 이상 사용되지 않을 경우(예: 이미지 제거 기능) free 를 이용해 할당 해제할 것
- 프로그램이 수행 될 때, 발생할 수 있는 아래의 예외상황들을 고려하여 구현할 것 (예외상황 발생 시, 예외상황임을 알릴 수 있는 적절한 문장을 출력하도록 한다.)
  1) 숫자만 입력 가능한 경우에, 문자가 입력이 되면 예외상황 발생 (메뉴 번호 입력을 할 때, 문자를 입력할 경우 등)
  2) 유효하지 않은 값을 입력할 경우 예외상황 발생 (없는 메뉴 번호를 입력 하는 경우, 로드 되지 않은 이미지의 번호를 입력 하는 경우, 10 보다 큰 이미지 목록 번호를 입력 할 경우 등)
  3) "이미지 출력" 기능의 범위 번호 입력을 이용한 여러 장의 이미지를 출력 할 때, 비정상적인 범위 및 다른 구분자를 입력한 경우 예외상황 발생 (정상적인 경우: "시작번호-끝번호")
  4) 문제에서 제시한 조건을 어기는 경우 예외상황 발생 (20x20 보다 작은 크기로 축소를 하려고 할 경우, 이미 압축된 이미지를 압축 하려고 할 경우 등)
  5) 로드할 파일이 없을 경우 예외상황 발생

예 1) 없는 메뉴를 입력 하였을 경우

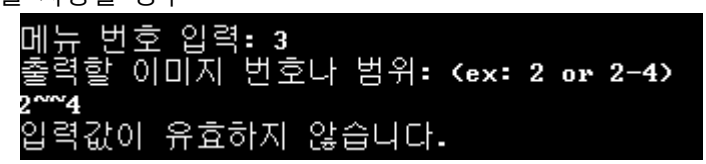


<그림 21> 없는 메뉴를 입력 하였을 경우 결과 화면 1



<그림 22> 없는 메뉴를 입력 하였을 경우 결과 화면 2

예 2) 범위 번호를 입력하여 여러 장의 이미지를 출력할 때, 두 범위의 구분자가 "-"가 아닌 다른 기호를 사용할 경우



<그림 23> 잘못된 구분자 기호를 입력 하였을 경우 결과 화면

[평가 조건]

- 설명된 기능을 이해하고 정확히 구현하였는가?
- "압축", "축소" 기능을 통해 만들어지는 결과가 맞는가?
- 기능별 함수를 기능 단위로 적절히 잘 만들었는가?
- "주의 사항"을 잘 지켰는가?
- 메모리 동적 할당 및 해제를 구현하였는가?
- 예외상황을 충분히 고려 하였는가?
- 지정된 구조체를 기반한 구조체 포인터 배열을 사용하였는가?
- 오류 없이 기능이 수행 되는가?