9

C 언어의 아파트 - 배열

안녕하세요, 여러분. 지난 강의에 내 주었던 마지막 문제는 모두 푸셨나요? 저는 기본적으로 자신이 내준 문제는 스스로 무슨 수를 써서라도 풀어야 한다라는 것이 원칙이지만, 댓글로 한에서 답안을 공개할 수 있으니 여러분의 답안을 댓글로 남겨주시면 고맙겠습니다. 댓글이 저를 귀찮게 한다는 생각 말고요, 과감하게 댓글을 남겨 주세요. 저는 오히려 여러분의 댓글을 기다리고 있는 사람입니다.

얼마전에 Psi 는 친구로 부터 프로그램을 하나 짜 달라는 요청을 받았습니다. 친절한 Psi 는 그 친구의 요청을 흔쾌히 승낙했죠. 그런데, 그 친구가 요청한 프로그램은 그다지 평범한 프로그램이 아니였습니다. 그 친구의 반에 30 명의 학생들이 있는데 각 학생들의 성적들을 입력받아서 평균보다 낮은 사람들의 번호 옆에 '불합격', 평균 이상의 사람들에게 '합격' 이라는 메세지 까지 출력하는 프로그램을 말입니다.

그래서 Psi 는 생각하였습니다.

'30 명의 학생들의 점수를 입력 받아서 평균 까지는 구할 수 있겠는데 말야. 각 학생의 점수들을 보관하기 위한 변수들이 필요하단 말이야. 학생이 4 명 이라면 편하겠지만 30 명이라면.. `a1, a2, a3, a4, a5, ..... a30` 까지 각 변수의 값들을 언제 다 입력 받지? 젠장할! 이거 완전히 '캐' 노가다 아닌가.

아무튼, 프로그래밍을 하다가 위와 같이 여러 개의 값을 동시해 보관할 필요성이 생기게 되었습니다. 이전의 경우 여러개의 값을 보관할 경우에, 그 개수 만큼의 변수가 필요했었지요. 하지만 위 경우처럼 이용해야 할 변수의 개수가 매우 많아지게 되면 어떨 까요?

만약 전국의 학생수에 해당하는 프로그램을 작성하려면 10만개 이상의 변수들이 필요하게 됩니다. 이는, 프로그래머가 아무리 'Ctrl+C, Ctrl+V' 신공이 뛰어나다고 해도 불가능한 일이겠지요.

따라서, C 언어에 **배열(Array)** 이라는 것이 등장하게 되었습니다. 배열은, 간단히 말하자면 변수들의 집합이라고 말할 수 있습니다. 예를 들어서 `int` 형 배열의 경우, `int` 형 변수들이 메모리 상에 여러개 할당 되어 있는 것이지요.

배열의 기초

```c
/* 배열 기초 */
#include <stdio.h>
int main() {
  int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

  printf("Array 3 번째 원소 : %d \n ", arr[2]);
  return 0;
}
```

성공적으로 컴파일 하였다면

실행 결과

```
Array 3 번째 원소 : 3
```

와 같이 나오게 됩니다.

일단 여러분은 새로운 것들이 나왔기 때문에 당황하는 분들이 계실 수 도 있습니다. 또한, 여태까지 배운 것들도 이해하기 힘든데 이제 더욱 어려운 것이 나타났구나! 라고 생각하시는 분들도 계실 것입니다. 하지만, 다행이도 여러분이 배우실 것들은 정말 쉬운 내용 들입니다. 그런 걱정 하시지 말고 차분히 읽어 보시면 됩니다.

앞서, 배열에 대해 잠깐 소개를 하였는데 배열은 말그대로 **특정한 형(Type) 의 변수들의 집합** 입니다. 변수를 정의할 때 에는

(변수의 형) (변수의 이름);

과 같이 정의했는데 배열은 그와 비슷하게도

(배열의 형) (배열의 이름)[원소 개수];

와 같이 해주면 됩니다. 위의 경우

```c
int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

와 같이 해주었으므로, *int 형의 10 개의 원소를 가지는 배열 arr!* 이라고 생각하시면 됩니다. 다시말해, 이 배열은 10 개의 int 형 변수들을 보관 할 수 있게 됩니다. 만약 char arr[10] 이라 한다면 각 원소들이 모두 char 형으로 선언 됩니다. 또한 위와 같이 중괄호로 감싸 주었을

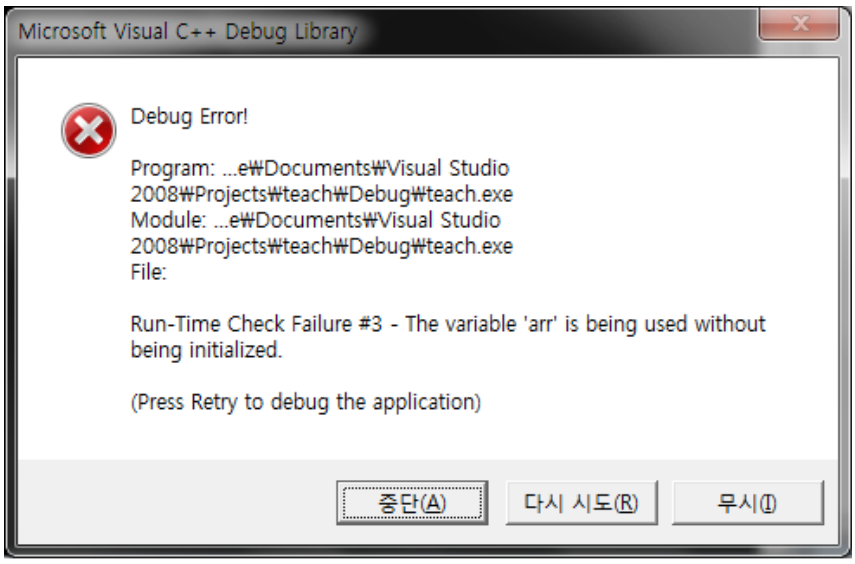
때, 배열의 각각의 원소에는 중괄호 속의 각 값들이 순차적으로 들어가게 됩니다. 배열의 첫 번째 원소에는 1, 두 번째 원소에는 2, ... 10 번째 원소에는 10 이 들어가게 됩니다.

그런데, 막연하게 배열을 정의해 놓고 보니 배열의 각각의 원소들에 접근하는 방법을 알려주지 않았군요. 사실 이는 간단합니다. 배열의 n 번째 원소의 접근하기 위해서는 arr[n-1] 와 같이 써주시면 됩니다. 즉, 대괄호[] 안에 접근하고자 하는 원소의 (번째수 - 1) 을 써주면 되죠. 예를 들어서

```
printf("Array 3 번째 원소 : %d \n ", arr[2]);
```

이라고 하면 배열의 3 번째 원소인 3 를 출력하게 됩니다. 많은 사람이 헷갈리는 부분인데, arr[2] 라고 하게되면 배열의 2 번째 원소인 2 를 출력하게 될 줄이라고 생각하지만 사실 3 번째 원소가 출력되게 됩니다. 즉, arr[0] 은 배열의 첫 번째 원소인 1 이 출력되고 arr[9] 는 배열의 10 번째 원소인 10 이 출력되게 됩니다.

만일 arr[10] 을 출력하려 한다면 무엇을 출력하게 될 까요? 한 번 해보세요. 아마도 아래와 같은 달콤한 애러 메세지를 볼 수 있게 될 것입니다.

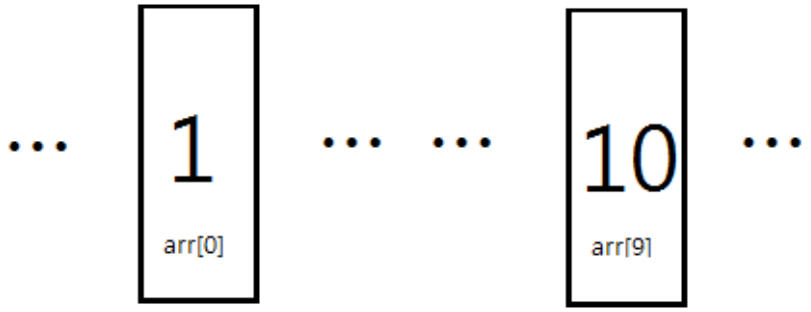


이는 배열의 존재하지도 않는 10 번째 원소를 참조하였기에 메모리 오류가 발생했다는 것 입니다. 사실, "배열의 10 번째 원소는 존재하지도 않으니, 우리가 참조한다면 0 과 같은 값들을 나타내면 되는데, 왜 위와 같이 껄렁한 오류 메세지를 출력하는 것인가?' 라고 생각할 것입니다. 그러기 위해선, 컴퓨터 상에 배열이라는 것이 어떻게 나타나게 되는이 알아야 합니다.

메모리 상에서의 배열

이미 3강에서도 다루었는데, 이 강의에서 저는 컴퓨터 메모리(RAM) 을 '직사각형 방이 쭉 나열되어 있는 형태' 로 표현하기로 했습니다. 사실 이는 다른 책들에서도 많이 이렇게 표현하므로 알아 두시는 것이 좋습니다. 이전 강의에서는 한 방의 크기가 1 바이트 였으므로 `int` 형의 경우 4 바이트를 차지하므로 배열의 한 원소를 표현하기 위해서는 4 칸을 차지해야 할 것입니다. 하지만, 각 4 개의 방을 모두 그리는 것이 힘들어서 메모리 한 칸을 그냥 4 바이트라고 합시다. 이 때, 위 배열 `arr` 의 경우 원소의 개수가 10 개 이므로 10 칸의 방을 차지하게 됩니다. (아쉽게도 그림 상에서는 모두 표현하지 못했으나 '…' 로 표현해 여러분의 상상력을 자극하고 있습니다 ㅎㅎ)

그런데, 메모리에 위 배열 하나만이 저장되어 있는 것일까요? 물론 아닙니다. 이 프로그램을 실행하는데 조차 수천개의 변수들의 메모리 상에 적재되어 이리저리 사라지고 저장되고 있습니다. 따라서 위 그림을 좀더 사실적으로 그리자면 아래와 같습니다.



즉, `arr[0]` 의 앞과 `arr[9]` 뒤에는 아무것도 없는 것이 아니라 다른 변수의 값들이 저장되고 있다는 사실 입니다. (물론 중간의 … … 에서는 `arr[1]` 부터 `arr[8]` 까지 '연속적' 으로 놓여있습니다.) 따라서, 훌륭한 운영체제라면 초짜 프로그래머가 다른 변수의 값들을 침범하는 것을 막기

위해 허락되지 않는 접근이 감지된다면 위와 같이 Run-Time 오류를 발생시키는 것이 정석입니다. 생각해 보세요. 당신이 스타를 하는데 미네랄이 갑자기 1000 에서 4 로 줄어든다면 기분이 어떻겠습니까?

배열 가지고 놀기

이제 본격적으로 배열을 가지고 놀아 보기로 합시다.

```c
/* 배열 출력하기 */
#include <stdio.h>
int main() {
  int arr[10] = {2, 10, 30, 21, 34, 23, 53, 21, 9, 1};
  int i;
  for (i = 0; i < 10; i++) {
    printf("배열의 %d 번째 원소 : %d \n", i + 1, arr[i]);
  }
  return 0;
}
```

성공적으로 컴파일 하였다면

실행 결과

```
배열의 1 번째 원소 : 2
배열의 2 번째 원소 : 10
배열의 3 번째 원소 : 30
배열의 4 번째 원소 : 21
배열의 5 번째 원소 : 34
배열의 6 번째 원소 : 23
배열의 7 번째 원소 : 53
배열의 8 번째 원소 : 21
배열의 9 번째 원소 : 9
배열의 10 번째 원소 : 1
```

아마, 여태까지 배운 내용을 잘 숙지하셨다면 위 소스를 어려움 없이 이해 하셨을 것이라고 생각합니다.

```c
int arr[10] = {2, 10, 30, 21, 34, 23, 53, 21, 9, 1};
```

일단, 배열의 정의 부분. arr 라는 원소를 10 개 가지는 int 형 배열을 선언한다라는 뜻이지요. 이 때, 각 원소의 값들은 중괄호 속에 있는 값들이 순차적으로 들어가므로 2, 10, 30 ~ 9, 1 까지 들어가게 됩니다.

```
for (i = 0; i < 10; i++) {
  printf("배열의 %d 번째 원소 : %d \n", i + 1, arr[i]);
}
```

이제, 배열의 원소들의 값들을 출력하는 부분 입니다. 잘 아시다싶이, for 문은 i = 0 부터 9 까지 1 씩 증가하면서 대입하는데 각 경우의 배열의 i 번째 원소를 출력하게 되므로 위 처럼 배열의 원소들의 값들이 출력되게 됩니다. 다시 한 번 기억하세요! 배열의 n 번째 원소를 참조하려면 arr[n-1] 로 입력해야 합니다!! arr[n] 이 '절대로' 아닙니다!

여기서 우리는 배열의 장점을 알 수 있습니다. 위 프로그램을 배열 없이 작성한다고 생각해보세요. 우리는 10 개의 서로 다른 변수를 만들어서 각각을 출력하는 작업을 해야 했을 것 입니다. 아래 (더러운) 코드는 위 예제와 동일한 내용을 10 개의 변수를 잡아서 직접 출력한 것을 보여 줍니다.

```
/* 더러운 코드 */
#include <stdio.h>
int main() {
  int a, b, c, d, e, f, g, h, i, j;
  a = 2;
  b = 10;
  c = 30;
  d = 21;
  e = 34;
  f = 23;
  g = 53;
  h = 21;
  i = 9;
  j = 1;

  printf("1 째 값 : %d \n", a);
  printf("2 째 값 : %d \n", b);
  printf("3 째 값 : %d \n", c);
  printf("4 째 값 : %d \n", d);
  printf("5 째 값 : %d \n", e);
  printf("6 째 값 : %d \n", f);
  printf("7 째 값 : %d \n", g);
  printf("8 째 값 : %d \n", h);
  printf("9 째 값 : %d \n", i);
  printf("10 째 값 : %d \n", j);

  return 0;
}
```

만일 여기에서 10 개의 변수의 값을 각각 입력받는 부분이라도 추가하라면 마우스라도 움켜 쥐고 울 것입니다. 하지만 배열을 이용하면 간단히 끝내 버릴 수 있습니다. 아래 예제 처럼 말이지요.

```c
/* 평균 구하기*/
#include <stdio.h>
int main() {
  int arr[5];  // 성적을 저장하는 배열
  int i, ave = 0;

  for (i = 0; i < 5; i++)  // 학생들의 성적을 입력받는 부분
  {
    printf("%d 번째 학생의 성적은? ", i + 1);
    scanf("%d", &arr[i]);
  }
  for (i = 0; i < 5; i++)  // 전체 학생 성적의 합을 구하는 부분
  {
    ave = ave + arr[i];
  }

  // 평균이므로 5 로 나누어 준다.
  printf("전체 학생의 평균은 : %d \n",  ave / 5);
  return 0;
}
```

성공적으로 컴파일 하였다면

실행 결과

```
1 번째 학생의 성적은? 100
2 번째 학생의 성적은? 90
3 번째 학생의 성적은? 80
4 번째 학생의 성적은? 70
5 번째 학생의 성적은? 60
전체 학생의 평균은 : 80
```

사실, 위 평균 구하는 프로그램은 앞에서도 한 번 만들어 보았는데 이번에는 배열을 이용하여 프로그램을 만들어 보았습니다. 이전보다 오히려 더 복잡해진 느낌이지만, 학생 개개인의 성적을 변수로 보관하기 때문에 더 많은 작업들을 할 수 있게 되죠.

```c
for (i = 0; i < 5; i++)  // 학생들의 성적을 입력받는 부분
{
  printf("%d 번째 학생의 성적은? ", i + 1);
  scanf("%d", &arr[i]);
}
```

위 소스에서 학생들의 성적을 입력받는 부분 입니다. 별다른 특징이 없습니다. 이전에 scanf 에서 &(변수명) 형태로 값을 입력 받았는데 배열도 마찬가지로 같습니다. (이 부분에 대해서는 나중에 좀 더 자세히 다루도록 하지요. 왜 &(arr[i]) 로 안해도 상관이 없는지...) 배열도, &arr[i] 로 쓰면 arr 배열의 (i+1) 번째 원소에 입력을 받게 됩니다. 이 때, arr 이 int 형 배열이기에 각 원소도 모두 int 형 이므로 %d 를 사용하게 되지요.

```
for (i = 0; i < 5; i++)  // 전체 학생 성적의 합을 구하는 부분
{
  ave = ave + arr[i];
}
```

이제 ave 에 배열의 각 원소들의 합을 구하는 부분 입니다. 배열도 변수의 모음이기에, 각 원소들은 모두 변수처럼 사용 가능합니다. 물론 배열의 각 원소들끼리의 연산도 가능합니다. 예를 들어서

```
ave[4] = ave[3] + ave[2] * i + ave[1]/i;
```

와 같이 해도 ave[4] 는 정확한 값이 들어가게 되지요. 마지막으로

```
printf("전체 학생의 평균은 : %d \n", ave / 5);
```

라 하면 전체 학생들의 평균을 구할 수 있게 됩니다.

자, 그렇다면 아까 위의 친구가 부탁했던 프로그램을 만들 수 있는 수준까지 도달할 수 있겠습니다. 한 번 여러분이 짜보고 제가 만든 소스코드와 비교해 보는 것도 좋은 방법 인 것 같습니다. (참고로 저는 학생을 30 명으로 하면 처음에 데이터 입력하기가 너무 힘들어서 그냥 10 명으로 했으니 여러분은 마음대로 하시기 바랍니다.)

```
/* 친구의 부탁 */
#include <stdio.h>
int main() {
  int arr[10];
  int i, ave = 0;
  for (i = 0; i < 10; i++) {
    printf("%d 번째 학생의 성적은? ", i + 1);
    scanf("%d", &arr[i]);
  }
  for (i = 0; i < 10; i++) {
    ave = ave + arr[i];
  }
  ave = ave / 10;
  printf("전체 학생의 평균은 : %d \n", ave);
  for (i = 0; i < 10; i++) {
    printf("학생 %d : ", i + 1);
    if (arr[i] >= ave)
```

```
      printf("합격 \n");
    else
      printf("불합격 \n");
  }
  return 0;
}
```

성공적으로 컴파일 하였다면 아래와 같은 화면을 볼 수 있을 것입니다.

실행 결과

```
1 번째 학생의 성적은? 30
2 번째 학생의 성적은? 90
3 번째 학생의 성적은? 80
4 번째 학생의 성적은? 68
5 번째 학생의 성적은? 99
6 번째 학생의 성적은? 100
7 번째 학생의 성적은? 78
8 번째 학생의 성적은? 23
9 번째 학생의 성적은? 85
10 번째 학생의 성적은? 49
전체 학생의 평균은 : 70
학생 1 : 불합격
학생 2 : 합격
학생 3 : 합격
학생 4 : 불합격
학생 5 : 합격
학생 6 : 합격
학생 7 : 합격
학생 8 : 불합격
학생 9 : 합격
학생 10 : 불합격
```

유후! 어때요. 잘 출력되는 지요.

```
for (i = 0; i < 10; i++) {
  printf("학생 %d : ", i + 1);
  if (arr[i] >= ave)
    printf("합격 \n");
  else
```

```
    printf("불합격 \n");
}
```

사실, 위 소스는 앞에서 보았던 우리의 평균 구하는 소스에 약간 더해서 만든 것 이므로 위 부분만 살펴 보면 되겠습니다. i 가 0 부터 9 까지 가면서 배열의 각 원소들을 ave 와 비교하고 있습니다. 만약, ave 이상이라면 합격, 그렇지 않다면 불합격을 출력하게 말이지요. 솔직히 이 정도 수준의 프로그램 소스는 이제 더이상 설명해 줄 필요가 없어진 것 같습니다. (저만 그런가요? 만약 그렇지 않다면 이전의 강의들을 다시 한 번 정독하기를 강력하게 권합니다)

소수 찾는 프로그램

이번에는 배열을 활용한 프로그램을 하나 더 살펴 보겠습니다. 이번 프로그램은 '배열' 을 활용한 소수 찾는 프로그램 입니다. **소수(prime number)**는 1 과 자신을 제외한 약수가 하나도 없는 수를 일컫습니다.

예를 들어, 2 와 3 은 소수 이지만 4 는 2 가 약수 이므로 소수가 아니지요. 또한 1 도 소수가 아닙니다. 아무튼, 소수를 찾는데 배열을 활용한다는 것은 이전에 찾은 소수들을 배열에 저장하여, 어떠한 수가 소수인지 판별하기 위해 그 수 이하의 소수들로 나누어 본다는 뜻입니다.

만일, 그 수 이하의 모든 소수들로 나누었는데 나누어 떨어지는 것이 없다면 그 수는 소수가 됩니다. 또한, 짝수 소수는 2 가 유일하므로 홀수들에 대해서만 계산하도록 합니다 . 또한, 짝수 소수는 2 가 유일하므로 홀수들에 대해서만 계산하도록 합니다.

이러한 아이디어를 바탕으로 프로그램을 짜 보겠습니다. 여러분은 아래 제가 구현한 코드를 보지 말고 한 번 스스로 해보시기 바랍니다. 참고로 저의 프로그램은 소수를 1000 개 만 찾습니다.

```
/* 소수 프로그램 */
#include <stdio.h>
int main() {
  /* 우리가 소수인지 판별하고 있는 수 */
  int guess = 5; /* 소수의 배열 */
  int prime[1000]; /*     현재까지 찾은 (소수의 개수 - 1)      아래 두 개의 소수를
                        미리 찾았으므로      초기값은 1 이 된다.      */
  int index = 1; /* for 문 변수 */
  int i;         /* 소수인지 판별위해 쓰이는 변수*/
  int ok;        /* 처음 두 소수는 특별한 경우로 친다 */
  prime[0] = 2;
  prime[1] = 3;
  for (;;) {
    ok = 0;
    for (i = 0; i <= index; i++) {
      if (guess % prime[i] != 0) {
        ok++;
      } else {
```

```
        break;
      }
    }
    if (ok == (index + 1)) {
      index++;
      prime[index] = guess;
      printf("소수 : %d \n", prime[index]);
      if (index == 999) break;
    }
    guess += 2;
  }
  return 0;
}
```

성공적으로 컴파일 했다면

실행 결과

```
... (생략) ...
소수 : 7727
소수 : 7741
소수 : 7753
소수 : 7757
소수 : 7759
소수 : 7789
소수 : 7793
소수 : 7817
소수 : 7823
소수 : 7829
소수 : 7841
소수 : 7853
소수 : 7867
소수 : 7873
소수 : 7877
소수 : 7879
소수 : 7883
소수 : 7901
소수 : 7907
소수 : 7919
```

와 같이 소수가 쭉 나오는 것을 볼 수 있습니다. 일단 위 소스코드의 핵심적인 부분만 설명해 보도록 하겠습니다.

```
for (i = 0; i <= index; i++) {
  if (guess % prime[i] != 0) {
    ok++;
  } else {
    break;
  }
}
```

위 부분은 guess 이하의 모든 소수들로 나누어 보고 있는 작업 입니다. index 는 (배열에 저장된 소수의 개수 - 1) 인데 prime[i] 로 접근하고 있으므로 배열의 모든 소수들로 나누어 보게 됩니다.

만일 guess 가 prime[i] 로 나누어 떨어지지 않는다면 ok 를 1 증가 시킵니다. 그리고 나누어 떨어진다면 소수가 아니므로 바로 break 되서 루프를 빠져 나가게 됩니다. 만일 ok 가 prime 배열에 저장된 소수의 개수, 즉 (index + 1)과 같다면 자기 자신 미만의 모든 소수들로도 안 나누어 떨어진다는 뜻이 되므로 소수가 됩니다.

이 때, 주의해야 할 점은 한 개의 수를 검사할 때 마다 ok 가 0 으로 리셋되어야 합니다. 그렇지 않다면 정확한 결과를 얻을 수 없겠죠?

```
if (ok == (index + 1)) {
  index++;
  prime[index] = guess;
  printf("소수 : %d \n", prime[index]);
  if (index == 999) break;
}
```

따라서, 위와 같이 index 를 하나 더 증가시킨 후 prime[index] 에 guess 를 추가 시켜 줍니다. 만일 index 가 999 가 된다면 배열이 꽉 찼단 뜻이 되므로 break 를 해서 for(;;) 를 빠져 나가게 됩니다. 어때요? 배열을 이용하여 정말 많은 일을 할 수 있지요?

배열의 중요한 특징

만약 똑똑한 사람이라면 다음과 같이 생각할 수 있을 것 입니다.

> 처음에 배열의 원소의 수를 숫자로 지정하지 않고 변수로 지정해도 될까? 예를 들어 arr[i] 라던지 말이야. 그렇게 된다면 위 프로그램에서 반의 총 학생 수를 입력 받아서 딱 필요한 데이터만 집어 넣으면 되잖아?

그렇다면, 그의 아이디어를 빌려서 프로그램을 만들어 봅시다.

```c
/* 과연 될까? */
#include <stdio.h>
int main() {
  int total;
  printf("전체 학생수 : ");
  scanf("%d", &total);
  int arr[total];
  int i, ave = 0;

  for (i = 0; i < total; i++) {
    printf("%d 번째 학생의 성적은? ", i + 1);
    scanf("%d", &arr[i]);
  }
  for (i = 0; i < total; i++) {
    ave = ave + arr[i];
  }

  ave = ave / total;
  printf("전체 학생의 평균은 : %d \n", ave);

  for (i = 0; i < total; i++) {
    printf("학생 %d : ", i + 1);
    if (arr[i] >= ave)
      printf("합격 \n");
    else
      printf("불합격 \n");
  }

  return 0;
}
```

만약 컴파일 한다면 아래에 수많은 오류가 쏟아져 나오는 것을 볼 수 있습니다.

```
1>c:\Users\lee\Documents\Visual Studio 2008\Projects\teach\teach\a.c(13) : error C2109: 첨자는 배열 또는 포인터 형식을 사용해야 합니다.
1>c:\Users\lee\Documents\Visual Studio 2008\Projects\teach\teach\a.c(15) : error C2065: 'i' : 선언되지 않은 식별자입니다.
1>c:\Users\lee\Documents\Visual Studio 2008\Projects\teach\teach\a.c(15) : error C2065: 'i' : 선언되지 않은 식별자입니다.
1>c:\Users\lee\Documents\Visual Studio 2008\Projects\teach\teach\a.c(15) : error C2065: 'i' : 선언되지 않은 식별자입니다.
1>c:\Users\lee\Documents\Visual Studio 2008\Projects\teach\teach\a.c(17) : error C2065: 'ave' : 선언되지 않은 식별자입니다.
1>c:\Users\lee\Documents\Visual Studio 2008\Projects\teach\teach\a.c(17) : error C2065: 'ave' : 선언되지 않은 식별자입니다.
1>c:\Users\lee\Documents\Visual Studio 2008\Projects\teach\teach\a.c(17) : error C2065: 'arr' : 선언되지 않은 식별자입니다.
1>c:\Users\lee\Documents\Visual Studio 2008\Projects\teach\teach\a.c(17) : error C2065: 'i' : 선언되지 않은 식별자입니다.
```

왜 오류가 나올까? 라고 고민한다면 3강 맨 아래 부분을 다시 보시길 바랍니다.[1)]

왜냐하면 변수는 무조건 최상단에 선언되야 되기 때문입니다! 위와 같이 배열 `arr` 과 변수 `i, ave` 가 변수 선언문이 아닌 다른 문장 다음에 나타났으므로 C 컴파일러는 무조건 오류로 처리하게 됩니다. (물론 C++ 에서는 가능합니다)

아아. 애초에 사람이 입력하는 대로 배열의 크기를 임의로 정할 수 는 없는 것이였군요. 그렇다면, 그냥 변수 크기 지정시 특정한 값이 들어있는 변수가 가능한지 살펴 봅시다.

1) 2018년 현재, 더이상 변수를 최상단에 선언하지 않아도 됩니다. 아마 오류가 발생하지 않을 것입니다.

```c
/* 설마 이것도? */
#include <stdio.h>
int main() {
  int total = 3;
  int arr[total];
  int i, ave = 0;

  for (i = 0; i < total; i++) {
    printf("%d 번째 학생의 성적은? ", i + 1);
    scanf("%d", &arr[i]);
  }
  for (i = 0; i < total; i++) {
    ave = ave + arr[i];
  }

  ave = ave / total;
  printf("전체 학생의 평균은 : %d \n", ave);

  for (i = 0; i < total; i++) {
    printf("학생 %d : ", i + 1);
    if (arr[i] >= ave)
      printf("합격 \n");
    else
      printf("불합격 \n");
  }

  return 0;
}
```

과연 성공할까요?

```
1>컴파일하고 있습니다...
1>8.c
1>c:₩users₩lee₩documents₩visual studio 2008₩projects₩teach₩teach₩8.c(5) : error C2057: 상수 식이 필요합니다.
1>c:₩users₩lee₩documents₩visual studio 2008₩projects₩teach₩teach₩8.c(5) : error C2466: 상수 크기 0의 배열을 할당할 수 없습니다.
1>c:₩users₩lee₩documents₩visual studio 2008₩projects₩teach₩teach₩8.c(5) : error C2133: 'arr' : 알 수 없는 크기입니다.
1>빌드 로그가 "file://c:₩Users₩Lee₩Documents₩Visual Studio 2008₩Projects₩teach₩teach₩Debug₩BuildLog.htm"에 저장되었습니다.
1>teach - 오류: 3개, 경고: 0개
========== 빌드: 성공 0, 실패 1, 최신 0, 생략 0 ==========
```

아아. 역시 우리의 기대를 처절하게 저버리고 오류를 내뿜는 컴파일러.

이는 C 언어에 처음에 배열의 크기를 변수를 통해 정의할 수 없게 규정하고 있기 때문입니다. (사실, '동적 할당' 이라는 방법으로 억지로 해서 정의할 수 있으나 이 부분에 대한 이야기는 나중에 다루도록 합시다.) 왜냐하면 처음에 컴파일러가 배열을 처리할 대 메모리 상에 공간을 잡아야 하는데 이 때, 잡아야 되는 공간의 크기가 반드시 상수로 주어져야 하기 때문입니다. 지금 수준에서 깊게 설명하는 것은 너무 무리인 것 같으니 그냥 '배열의 크기는 변수로 지정할 수 없다' 정도로 넘어가도록 합시다.

상수 (Constant)

상수는 변수의 정반대로 처음 정의시 그 값이 바로 주어지고, 그 값이 영원히 바뀌지 않습니다.

```
/* 상수 */
#include <stdio.h>
int main() {
  const int a = 3;

  printf("%d", a);
  return 0;
}
```

만약 성공적으로 컴파일 하였다면

```
3
```

와 같이 나오게 됩니다. 상수는 아래와 같이 정의합니다.

const (상수의 형) (상수 이름) = (상수의 값);

위 소스의 경우, a 라는 이름의 int 형 상수이고 그 값은 3 이라는 것을 나타내고 있습니다.

```
const int a = 3;
```

상수라고 해서 꼭 특별한 것이 있는 것은 아닙니다. 단지, 처음에 한 번 저장된 값은 '절대로' 변하지 않는 다는 점일 뿐이지요. 그렇기 때문에 처음 상수를 정의시 값을 정의해 주지 않는다면

```
#include <stdio.h>
int main() {
  const int a;

  printf("%d", a);
  return 0;
}
```

와 같이 컴파일시 아래와 같이 나타나게 됩니다.

```
1>컴파일하고 있습니다...
1>a.c
1>c:₩users₩lee₩documents₩visual studio 2008₩projects₩teach₩teach₩a.c(6) : warning C4700: 초기화되지 않은 'a' 지역 변수를 사용했습니다.
1>링크하고 있습니다...
1>매니페스트를 포함하고 있습니다...
1>빌드 로그가 "file://c:₩Users₩Lee₩Documents₩Visual Studio 2008₩Projects₩teach₩teach₩Debug₩BuildLog.htm"에 저장되었습니다.
1>teach - 오류: 0개, 경고: 1개
========== 빌드: 성공 1, 실패 0, 최신 0, 생략 0 ==========
```

상수는 또한 그 특성 답게 그 값 자체를 바꿀 수 없습니다. 예를 들어서

```c
#include <stdio.h>
int main() {
  const int a = 2;

  a = a + 3;
  printf("%d", a);
  return 0;
}
```

를 한다면,

```
1>------ 빌드 시작: 프로젝트: teach, 구성: Debug Win32 ------
1>컴파일하고 있습니다...
1>a.c
1>c:₩users₩lee₩documents₩visual studio 2008₩projects₩teach₩teach₩a.c(6) : error C2166: l-value가 const 개체를 지정합니다.
1>빌드 로그가 "file://c:₩Users₩Lee₩Documents₩Visual Studio 2008₩Projects₩teach₩teach₩Debug₩BuildLog.htm"에 저장되었습니다.
1>teach - 오류: 1개, 경고: 0개
========== 빌드: 성공 0, 실패 1, 최신 0, 생략 0 ==========
```

와 같은 오류가 발생하게 됩니다. 즉, 상수는 어떠한 짓으로도 값을 변경할 수 없는 불멸의 데이터입니다. 이러한 특성 때문에 여러분은 아마 '배열의 크기를 상수로 지정할 수 는 없을까?' 라는 생각을 하게 됩니다. 하지만, 아래와 같은 코드를 보면 이러한 생각이 깨지게 되죠.

```c
#include <stdio.h>
int main() {
  int b = 3;
  const int a = b;
  char c[a];
  return 0;
}
```

즉, 이는 상수 a 로 배열의 크기가 할당이 가능하게 된다면 다시 말해 변수 b 의 크기로 배열의 크기를 지정할 수 있다는 말이 되기 때문에 이전에 '변수로 배열의 크기를 지정할 수 없다' 라는 사실에 모순됩니다.

컴퓨터 프로그래밍을 하다 보면 상수를 사용할 일이 꽤나 많습니다. 예를 들어서 계산기 프로그램을 만들 때 에는 Pi 의 값을 상수로 지정해 놓게 된다면 변수로 지정할 때 보다 훨씬 안전해지게 됩니다. 왜냐하면 변수로 Pi 의 값을 지정했을 때, 프로그래머가 코딩상의 실수로 그 값을 바꾼다면 찾아낼 도리가 없지만, 상수로 지정시에 그 값을 실수로 바꾸도록 코딩을 해도 애초에 컴파일러가 오류를 뿜기 때문에 오류를 미연에 방지할 수 있습니다.

초기화 되지 않은 값

우리가 변수의 값을 초기화 하지 않는다면 그 변수는 무슨 값을 가질 까? 라는 생각을 한 분들이 많을 것 같습니다. 0 을가질까요? 아닙니다. 0 도 값 이지 않습니까? 0 을 가진다면 0 이라는

값을 가진다는 것 이지요? 그렇다면 한 번 해보지요.값이 대입되지 않은 변수의 값을 출력해보는 프로그램을 짜보면 아래와 같습니다.

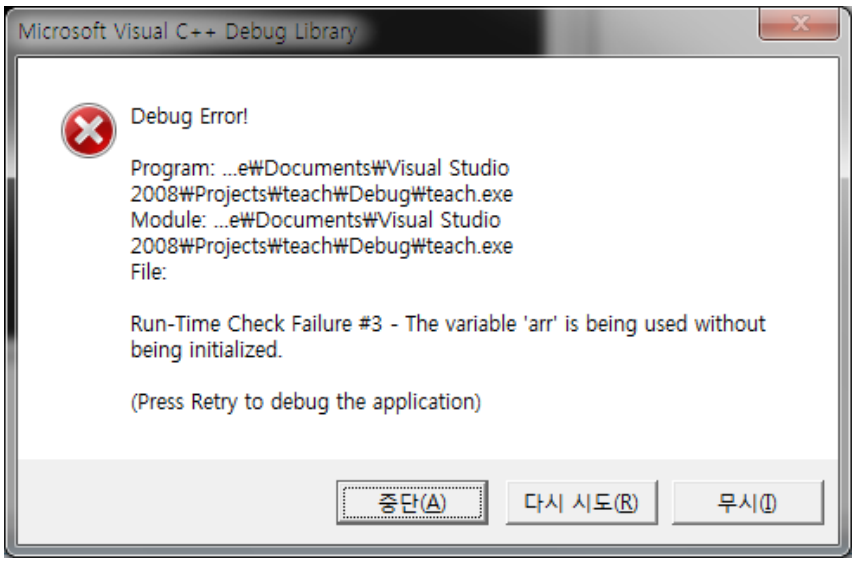
```
#include <stdio.h>
int main() {
  int arr;

  printf("니 값은 모니 : %d", arr);
  return 0;
}
```

컴파일 해보면 아래와 같은 경고를 볼 수 있습니다.

컴파일 오류

```
warning C4700: 초기화되지 않은 'arr' 지역 변수를 사용했습니다.
```

아마, 심상치 않지만 그래도 오류가 없으니 실행은 해볼 수 있겠군요. 아마 실행해 보면 아래와 같은 모습을 보실 수 있을 것 입니다.



아니 이럴수가! 변수 arr 의 값을 보기위해 값을 출력하려고 했더니만, 런타임 오류(프로그램 실행 중에 발생하는 오류)가발생하군요. 운영체제는 초기화 되지 않은 변수에 대한 접근 자체를 불허하고 있습니다. 이 때문에 우리는 이 변수에 들어있는 값을영영 보지 못하게 됩니다.

```
/* 초기화 되지 않은 값 */
#include <stdio.h>
int main() {
  int arr[3];
  arr[0] = 1;
```

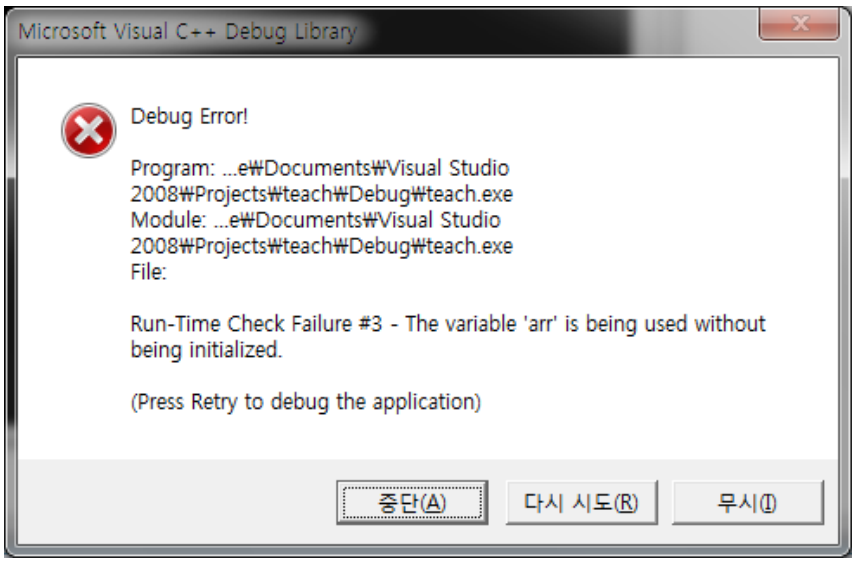
```
  printf("니 값은 모니 : %d", arr[1]);  // arr[0] 이 아닌 arr[1] 을 출력

  return 0;
}
```

이번에는 배열의 경우를 살펴 봅시다. 상콤한 기분으로 컴파일을 했으나,

컴파일 오류

`warning C4700: 초기화되지 않은 'arr' 지역 변수를 사용했습니다.`

와 같은 경고가 발생합니다. 심지어 앞 예제에서 발생했던 경고와 번호(C7400)와 동일하군요. 불안감에 사로잡혀 실행해보면.. 아니나 다를까 아래와 같은 오류 메세지 창을 보게 됩니다.



역시, `arr[1]` 의 값은 정의되어 있지 않기 때문에 이 값을 출력할 생각은 꿈도 꾸지 말라는 것이라는 것이죠. 참으로 야속한 컴퓨터 입니다.

```
/* 초기화 되지 않은 값 */
#include <stdio.h>
int main() {
  int arr[3] = {1};
  printf("니 값은 모니 : %d", arr[1]);

  return 0;
}
```

이번에는 마지막으로 거의 자포자기 한 심정으로 위 소스를 컴파일 해 봅시다.

실행 결과

```
니 값은 모니 : 0
```

오잉! 경고도, 오류도 나타나지 않습니다. 더군다나, 0 이라는 값이 출력되었습니다! 놀랍군요. 우리는 위 문장에서 `arr[1]` 에 0 을 집어 넣는 다는 말은 한 번이라도 하지 않았습니다. 사실 여러분은 위와 같이 배열을 정의한데에 놀랄 수도 있습니다. 배열의 원소는 3 개나 있는데 값은 오직 1 개 밖에 없기 때문이죠. 하지만 여러분들은 위 문장이 다음과 같은 역할을 한다는 것은 직감적으로 알 수 있습니다.

int arr[3]; arr[0] = 1;

물론 맞는 말 입니다. `printf` 부분을 `arr[0]` 출력으로 바꾸어 보면 여러분이 생각했던 대로 1 이 출력됩니다. 하지만, `arr[1]` 이 도대체 왜 오류가 나지 않는 것일까요? 아까전에 `int arr[3]; arr[0] = 1;` 방법으로 해서 끔찍한 오류가 발생되는 것을 여러분이 두 눈으로 톡톡히 보셨지 않습니까? 그 이유는

```
int arr[3] = {1};
```

와 같이 정의한다면 컴파일러가 내부적으로 아래와 같이 생각하기 때문입니다.

```
int arr[3] = {1, 0, 0};
```

따라서, 자동적으로 우리가 특별히 초기화 하지 않은 원소들에는 0 이 들어가게 됩니다.

그렇다면

```
int arr2[5] = {1, 2, 3};
```

은 어떻게 될까요? 역시, 해보면

```
int arr2[5] = {1, 2, 3, 0, 0}
```

과 같이 한 것과 똑같이 됩니다.

이상으로, 배열에 대한 첫 번째 강의를 마치도록 하겠습니다. 아직 여러분은 배열에 대해 모든것을 다 알고 계신 것은 아닙니다. 다음 강의에서는 더욱 놀라운 배열의 기능을 알아 보도록 합시다.

생각해 볼 문제

문제 1

위 입력받는 학생들의 성적을 높은 순으로 정렬하는 프로그램을 만들어 보세요.

문제 2

입력받은 학생들의 성적을 막대 그래프로 나타내는 프로그램을 만들어 보세요.

다차원 배열

안녕하세요, 여러분. 아마 이쯤 되면 여태까지 공부하셨던 사람들 중 일부는 아, 내 머리는 컴퓨터 언어를 배우기에 최적화 되어 있지 않나 보다 하고 포기하는 사람들과 오오... 내 맘대로 프로그램을 만들고 주물럭 주물럭 거릴 수 있는 것이 신기한데?? 하는 두 가지 부류의 사람들로 나뉘게 됩니다.

사실, 여기 까지 온 것 만으로도 정말 대단하다고 말 할 수 있습니다. 왜냐하면 인터넷을 통해 무언가 짬짬히 보아서 공부해 나가는 것은 쉬운 일이 아니기 때문이죠. 여러 게임의 유혹도 있고, 내가 이걸 배우는 시간에 채팅이나 하면 좋을 것을.. 와 같은 생각도 들기 때문이죠.

하지만, 저는 여러분이 조금만 더 힘을 내어 이러한 유혹을 이겨내고 C 언어의 끝에 도달하시기 바랍니다. 사실 배열과 앞으로 나오는 포인터 부분만 넘어간다면 더이상 어려울 부분이 없기 때문이죠. 그 부분이 넘어가 C 언어 끝에 도달하게 되면, 윈도우 API 를 공부하여 윈도우 앱도 만들고, Direct X 나 OpenGL 등을 공부해서 3D 게임 까지. 뿐만 아니라 소켓 프로그래밍을 공부한다면 친구들과 채팅할 수 있는 프로그램도 만들 수 있고 게임 서버들도 만들 수 있습니다.

C 언어를 배움으로써 얻을 수 있는 위 많은 것들을 쉽게 포기하실 것 입니까? 물론, 나가실 분은 조용히 뒤로가기를 누르셔도 상관 없습니다. 다만, 이 순간의 클릭이 당신의 앞날을 좌우 할 지도 모른다는 사실을 잊지 마세요. 그리고 참, 이전 내용이 기억이 나지 않는다고 머리를 쥐어 뜯지 마세요. 그냥 이전 강좌를 다시 보면 됩니다. 이전 강좌의 내용이 잘 기억이 나지 않는 것은 '지극히 정상' 이니 다시 한 번 읽어 보므로써 기억을 강화시키도록 하세요 :)

그럼 잡담을 끝내고 본론으로 들어가도록 합시다. 이전 강좌에서 배열은 변수 들의 모임이라고 했습니다. 이 때 `arr` 이라는 배열의 `i` 번째 원소를 참조 하기 위해선 `arr[i]` 라고 써야 한다는 것도 알았습니다. 그런데 똑똑한 사람이라면 이 아이디어를 확장해서 다음과 같은 생각을 할 수 도 있을 것 입니다.

> 배열의 배열을 만들면 어떨까?

정말로 놀라운 생각 입니다. 일단 여기서 우리는 **배열의 배열**의 의미를 좀 더 명확하게 해야 겠습니다. *int* 형의 배열 이란 말은 배열의 각 원소가 *int* 형 변수 인 것*!* 입니다.

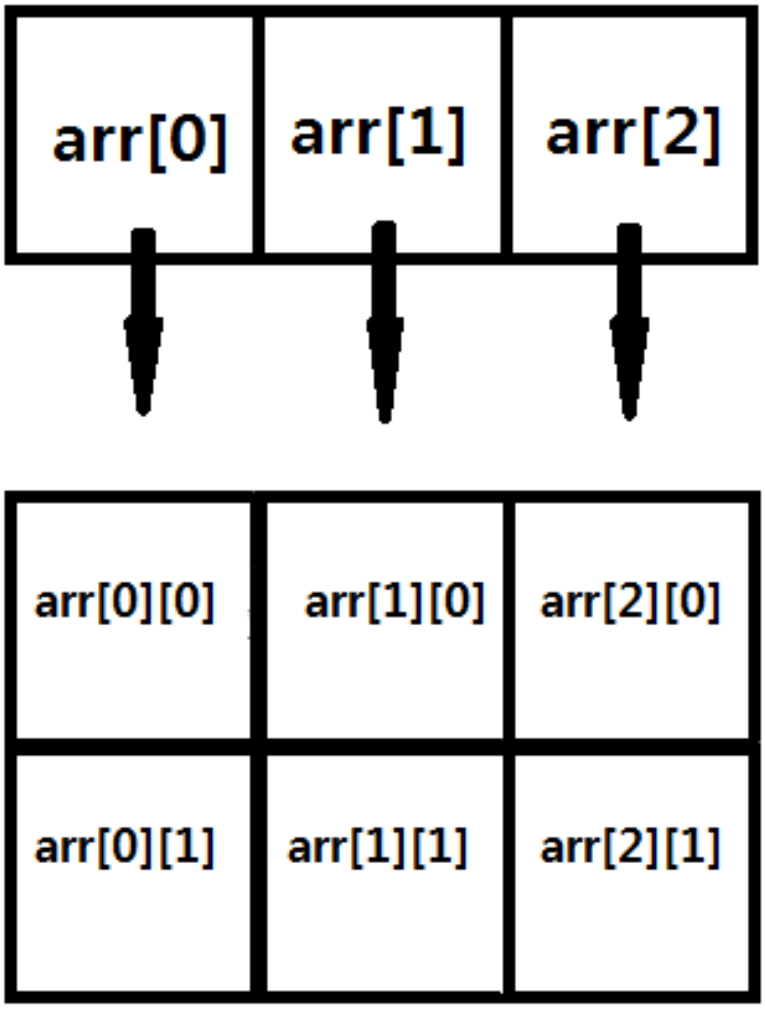
그렇다면 **int 형의 배열의 배열** 이란 말은 무엇일까요? 이 말은, **배열의 각 원소가 int 형의 배열인 것** 을 말합니다. C 언어 에서 이러한 형태의 배열을 정의하기 위해선 아래와 같이 하면 써주면 됩니다.

```
// 위 경우 (배열의 형) 부분에 int 가 오면 된다. ? 에는 배열의 크기
(배열의 형) (배열의 이름)[?][?];
```

이전에는 `(배열의 이름)[?]` 였지만 2 차원 배열은 옆에 `[?]` 하나가 더 붙었지요.먼저 배열의 배열이 무엇인지 확실하게 알기 위해서 다음과 같은 배열을 정의해 봅시다.

```
int arr[3][2];
```

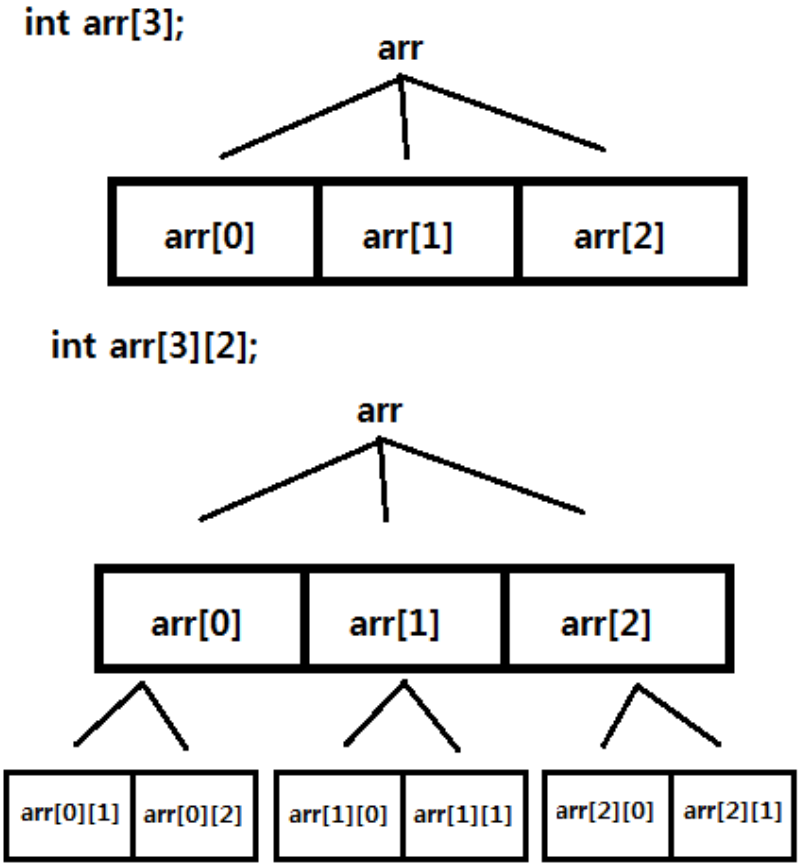
앞에서 말했듯이 위 배열은 '배열의 각 원소 3 개가 원소를 2 개 가지는 int 형의 배열이고 이름은 arr 이다.' 을 의미 합니다. 따라서,



가 됩니다.

즉, arr[0] 이라 하면 int 형의 원소를 2 개 가지는 배열을 말하는 것이며 그 배열의 원소는 각각 arr[0][0], arr[0][1] 이 되겠지요.

일차원 배열과 이차원 배열을 한 눈에 비교하자면 아래와 같습니다.



어때요, 간단하죠? 따라서, `arr[m][n];` 과 같이 배열을 선언한다면 (`m` 과 `n` 은 임의의 정수값), $m \times n$ 개의 변수를 가지는 배열을 선언한 것이 됩니다.

그렇다면, 2 차원 배열을 가지고 무슨 짓을 할 수 있을 까요? 사실, 여러분도 이미 예상한 바 있지만 오히려 일차원 배열에 비해 활용도가 훨씬 높아지게 됩니다. 예를 들면, 33 명의 학생에 대해 국어, 수학, 영어, 과학 점수를 보관하는 배열을 만든다 (이전의 배열 하나만 이용해선 한 과목의 점수 밖에 보관할 수 없었죠) 도서 입출 관리 프로그램에서 개개의 도서에 대해서 이 도서를 빌려간 날짜, 반납한 날짜 등을 보관하는 배열을 만든다 등등이 있습니다.

아! 그런데 아직 왜 이러한 배열을 '2차원' 배열이라고 말하는지 이야기 하지 않았군요. 사실, 메모리에는 모든 배열이 일차원 배열과 다름없이 들어갑니다. 그런데, 아래 예제를 보고 나면 왜 '2차원' 배열이라 이야기 하는지 감이 확 올 것 입니다.

```c
/* 2 차원 배열 */
#include <stdio.h>
int main() {
  int arr[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

  printf("arr 배열의 2 행 3 열의 수를 출력 : %d \n", arr[1][2]);
  printf("arr 배열의 1 행 2 열의 수를 출력 : %d \n", arr[0][1]);
  return 0;
```

```
}
```

성공적으로 컴파일 하였으면

실행 결과

```
arr 배열의 2 행 3 열의 수를 출력 : 6
arr 배열의 1 행 2 열의 수를 출력 : 2
```

와 같이 나옵니다. 처음에 2 차원 배열을 정의 할 때 부터 확 와닿는 느낌이 듭니다. 왜냐하면 정말로 2 차원 상에 배열된 배열이라고 생각할 수 있고, 배열의 선언 자체가 2 차원 적으로 정의 되었기 때문이죠

```
int arr[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
```

위에서 2 차원 배열을 정의 하였습니다. 그런데 사실 아래와 같이 모두 한 줄에

```
int arr[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
```

다 써도 큰 문제는 없지만 보기 좋기 위해 위와 같이 나열한 것 입니다.왜냐하면 2 차원 배열을 아래와 같은 모습으로 존재한다고 상상 할 수 있기 때문이죠.

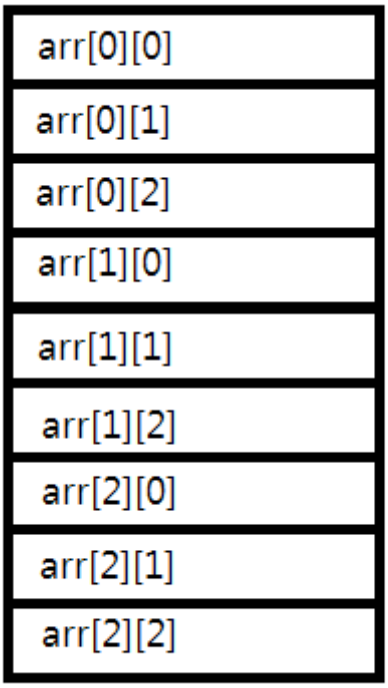
arr[0][0] 1	arr[0][1] 2	arr[0][2] 3
arr[1][0] 4	arr[1][1] 5	arr[1][2] 6
arr[2][0] 7	arr[2][1] 8	arr[2][2] 9

마치 우리가 배열을 정의했던 것 처럼 이차원 상에 배열이 있다고 생각해서 그려볼 수 있습니다. 이 때, `arr[x][y]` 라고 한다면 `x` 는 몇 번째 줄에 있는지, `y` 는 몇 번째 열에 있는지를 나타냅니다. 예를 들어서 `arr[0][1]` 은 0 번째 줄, 1 번째 열에 있으므로 2 가 되겠지요. 결과적으로 말하자면 '일차원 배열은 한 개의 값(x)으로 원소에 접근하는 것이고, 이차원 배열은 두 개의 값(x,y)으로원소에 접근하는 것이다!' 라고 생각할 수 있게 됩니다.

한가지 눈 여겨 볼 점은 arr 이 '배열의 배열' 이라는 것이 맞다는 것 입니다. 왜냐하면 arr[0] 을 하나의 배열의 이름이라고 생각해 보면 3 개의 원소 arr[0][0], arr[0][1], arr[0][2] 를 가진다고 볼 수 있다는 것 이지요. 마찬가지로 arr[1], arr[2] 도 하나의 배열이라고 생각할 수 있습니다. 그런데 arr 이 배열이므로 우리는 2 차원 배열을 '배열의 배열' 이라고 볼 수 있지요.

기존의 배열과 마찬가지로 arr[3] 이라 하면 arr[0] ~ arr[2] 까지 사용 가능했듯이 arr[3][3] 이라 하면 arr[0][0] ~ arr[0][2], arr[1][0] ~ arr[1][2], arr[2][0] ~ arr[2][2] 까지 사용 가능 합니다.

그런데 한 가지 지적해야 할 부분은 컴퓨터 메모리 상에선 절대로 이차원 적으로 만들어 지지 않는다는 것 입니다. 사실, 컴퓨터 메모리 상에선 2차원 이라는 것이 존재할 수 가 없습니다. 단지 선형으로 된 데이터들의 나열일 뿐이지요. 위 배열의 경우 컴퓨터 메모리 상에 다음과 같이 존재합니다.



하지만 우리가 이렇게 메모리 상에 선형으로 배열되어 있음에도 불구하고 '이차원 배열' 이라고 부르는 이유는 위 처럼 메모리 상에 2 차원으로 배열되어 있다고 생각하면 정말로 간편하기 때문입니다. 예를 들어서 arr[2][1] 은 메모리 상에서 배열의 시작 부분 (arr[0][0]) 에서 부터 3*2 + 1 = 7 번째에 있는 값 이라고 생각해야 되지만 사실 이 데이터를 2 차원 상에 배열해 놓고 2 행, 1 열의 값 이라고 생각하면 훨씬 편하기 때문입니다. (물론 컴퓨터는 전자의 경우로 계산하게 됩니다)

```c
/* 학생 점수 입력 받기 */
#include <stdio.h>
int main() {
  int score[3][2];
  int i, j;
```

```
  for (i = 0; i < 3; i++)  // 총 3 명의 학생의 데이터를 받는다
  {
    for (j = 0; j < 2; j++) {
      if (j == 0) {
        printf("%d 번째 학생의 국어 점수 : ", i + 1);
        scanf("%d", &score[i][j]);
      } else if (j == 1) {
        printf("%d 번째 학생의 수학 점수 : ", i + 1);
        scanf("%d", &score[i][j]);
      }
    }
  }

  for (i = 0; i < 3; i++) {
    printf("%d 번째 학생의 국어 점수 : %d, 수학 점수 : %d \n", i + 1,
           score[i][0], score[i][1]);
  }

  return 0;
}
```

성공적으로 컴파일 하였다면

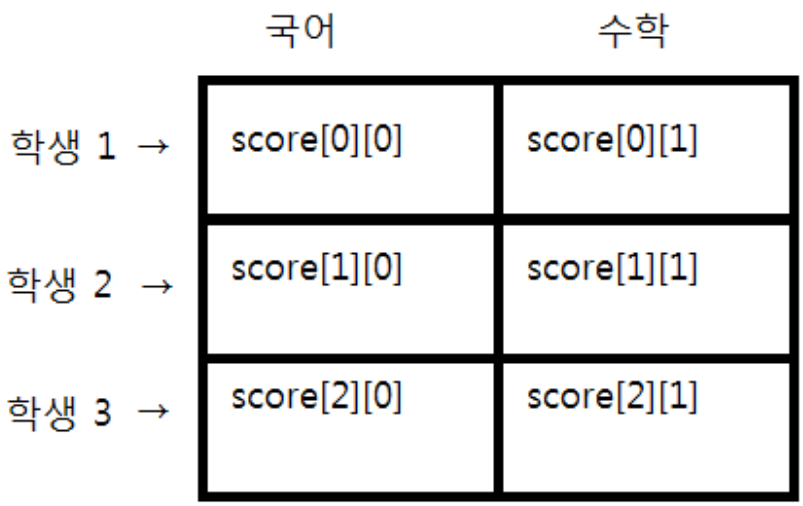
실행 결과

```
1 번째 학생의 국어 점수 : 30
1 번째 학생의 수학 점수 : 60
2 번째 학생의 국어 점수 : 90
2 번째 학생의 수학 점수 : 100
3 번째 학생의 국어 점수 : 70
3 번째 학생의 수학 점수 : 80
1 번째 학생의 국어 점수 : 30, 수학 점수 : 60
2 번째 학생의 국어 점수 : 90, 수학 점수 : 100
3 번째 학생의 국어 점수 : 70, 수학 점수 : 80
```

와 같이 됩니다. 사실 작동 원리는 간단 합니다.

```
int score[3][2];
```

일단 위 구문을 통해 3 행, 2 열의 크기를 가지는 2 차원 배열 score 을 선언 하였습니다. 사실 우리가 프로그래밍 하고자 하는 목표에 따라 해석해 보면 '3 명의 학생의 2 과목의 데이터를 보관하는 score 2 차원 배열' 이라고 볼 수 도 있습니다. 이를 그림으로 나타내면



꼴로 보면 됩니다.

```
for (i = 0; i < 3; i++)  // 총 3 명의 학생의 데이터를 받는다
{
  for (j = 0; j < 2; j++) {
    if (j == 0) {
      printf("%d 번째 학생의 국어 점수 : ", i + 1);
      scanf("%d", &score[i][j]);
    } else if (j == 1) {
      printf("%d 번째 학생의 수학 점수 : ", i + 1);
      scanf("%d", &score[i][j]);
    }
  }
}
```

이제 for 문을 통해서 3 명의 학생의 데이터를 입력 받게 됩니다. 일단 오래간만에 두 개의 for 문이 같이 돌아가는데 어떠한 형식으로 작동되는 지는 알고 있겠지요? i = 0 일 때, j = 0 ~ 1, i = 1 일 때, j = 0 ~ 1, i = 2 일 때, j = 0 ~ 1 로 돌아가게 됩니다. 즉 위 부분을 통해 2 차원 score 배열의 값을 집어 넣게 되는 것 이지요.

```
if (j == 0) {
  printf("%d 번째 학생의 국어 점수 : ", i + 1);
  scanf("%d", &score[i][j]);
} else if (j == 1) {
  printf("%d 번째 학생의 수학 점수 : ", i + 1);
```

```
  scanf("%d", &score[i][j]);
}
```

위 for 문 안의 위 부분을 살펴 보면 j 가 0 이면 국어점수를 입력해라, j 가 1 이면 수학 점수를 입력해라 라고 물어 보는 것이 달라 집니다.

마지막으로

```
for (i = 0; i < 3; i++) {
  printf("%d 번째 학생의 국어 점수 : %d, 수학 점수 : %d \n", i + 1, score[i][0],
         score[i][1]);
}
```

를 통해 입력 받은 값을 깔끔하게 보여주게 됩니다.

2 차원 배열 정의하기

앞선 예제에서

```
int arr[2][3] = {1, 2, 3, 4, 5, 6};
```

와 같이 2 차원 배열을 선언하였습니다. 그런데, 프로그래밍시 줄 수를 절약하고 싶은 사람들은 아래와 같이 해도 큰 문제는 없습니다.

```
int arr[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

위 처럼 정의하는 것은 우리가 앞서 써 왔던 방법과 전혀 차이가 없으니 그냥 알아 두시면 됩니다. 여러분들 께서 원하는 방법으로 정의하시면 됩니다.

참고로, 이전 강의에서 이야기를 하지 않았는데, 다음과 같이 배열을 정의할 수 도 있습니다.

```
int arr[] = {1, 2, 3, 4};
```

음... 무언가 이상하다는 느낌이 드나요? 사실, 알고보면 단순합니다. 위와 같이 정의할 수 있는 이유는 컴파일러가 원소의 개수를 정확하게 알기 때문입니다. 컴파일러는 우리가 배열을 정의한 것을 보고 '아, 이 사람이 원소를 4 개 가지는 int 배열을 정의하였구나!' 라고 알아서 대괄호 안에 자동적으로 4 를 집어 넣어서 생각하게 됩니다. 따라서 위와 같이 정의하나

```
int arr[4] = {1, 2, 3, 4};
```

로 정의하나 같은 말이 되겠지요.하지만 아래와 같이 정의하는 것은 안됩니다.

```
int arr[];
```

이 것은 왜 안될까요? 아마, 이전 강의를 잘 보신 분들게서는 단박에 알아 차릴 수 있을 것 입니다. 그 이유는 '배열의 크기는 임의로 정해지지 않기 때문입니다. 즉, 위와 같이 배열을 정의한다면 컴파일러는 우리가 어떠한 크기의 배열을 정의하고 싶은지 모릅니다. 따라서 아래와 같은 오류를 내뿜게 됩니다.

컴파일 오류

```
error C2133: 'arr' : 알 수 없는 크기입니다.
```

이 아이디어를 2 차원 배열에도 그대로 적용 시킬 수 있습니다.일단, 아래와 같은 2 차원 배열의 정의를 살펴 보도록 합시다.

```
int arr[][3] = {{4, 5, 6}, {7, 8, 9}};
```

위 정의를 보면 여러분은 비어있는 대괄호 안에 무슨 값이 들어갈지 맞출 수 있을 것입니다. 무엇이냐고요? 바로 2 이지요. 왜냐하면 {4,5,6} 를 가지는 arr[3] 배열 하나와, {7,8,9} 를 가지는 또다른 arr[3] 배열을 정의하여 총 2 개의 arr[3] 배열을 정의하였기에, int arr[2][3] 이 되어야 하는 것이지요.

그렇다면 아래와 같은 문장이 유효한지 살펴보세요.

```
int arr[][2] = {{1, 2}, {3, 4}, {5, 6}, {7}};
```

어! 이상하네요. 마지막에 그냥 {7} 이라고 되어 있잖아요? 아마 여러분들 중 대다수는 이 것을 보고 위 문장이 틀렸고 성공적으로 컴파일 되지 않으리라 생각할 것입니다. 하지만, 위 2 차원 배열은 배열 정의시 arr[][2] 라고 하였기 때문에 무조건 원소가 2 인 1 차원 배열들이 생기게 됩니다. 즉, 7 이 속한 1 차원 배열에는 원소가 한 개인 것이 아니라 마치 arr[3] = {1} 고 해도 상관없는 것 처럼 8 번째 원소가 들어갈 자리를 비워놓게 됩니다. 따라서, 위 문장은 틀린 것이 아닙니다. 그렇다면 아래 문장을 봐보세요.

```
int arr[2][] = {{4, 5, 6}, {7, 8, 9}};
```

과연 될까요? 아마 여러분들 중 대다수는 될 것이라 생각하고 있을 것 입니다. 하지만 놀랍게도 컴파일해보면

컴파일 오류

```
error C2087: 'arr' : 첨자가 없습니다.
error C2078: 이니셜라이저가 너무 많습니다.
```

와 같은 오류들을 만나게 됩니다. **C 에서는 다차원 배열의 경우 맨 앞의 크기를 제외한 나머지 크기들을 정확히 지정해줘야 오류가 발생하지 않습니다.**

3 차원, 그 이후 차원의 배열들

2 차원 배열을 잘 이해하였다면 3 차원 배열을 이해하는 것은 그리 어려운 것이 아니라 생각됩니다. 사실, 보통의 프로그래밍에서 3 차원 배열을 쓰는 경우는 그렇게 많지 않습니다. (물론 제가 만들어본 프로그램들에 한해서...) 그렇지만 쓸 수 도 있기에 간단하게 집고 넘어가기만 합시다.

3 차원의 배열의 정의는 2 차원 배열과 거의 동일합니다. (그 이후의 차원들도 마찬가지)

```
(배열의 형)(배열의 이름)[x][y][z];  // 여기서 x,y,z 는 배열의 크기를 말합니다.
```

이제, 머리속으로 상상의 나래를 펼쳐 봅시다. 제가 그림판으로 3 차원 적인 그림을 그릴 수는 없으므로 여러분의 지능을 믿겠습니다! 일단, 아래의 배열을 머리에 그려 봅시다.

```
int arr[3][4];
```

이는 가로 길이가 4 이고 세로 길이가 3 인 평면위에 int 변수들이 하나씩 놀고 있는 것을 상상하면 됩니다. 그렇다면 이제 아래 배열을 머리에 그려 봅시다.

```
int brr[2][3][4];
```

아아악! 모르겠다고요? 아니요, 어렵지 않습니다. 위에서 상상한 평면 위에 동일한 평면이 한 층 더 있다고 생각하면 됩니다. 즉, 위에서 생각했던 평면이 2 개의 층으로 생겼다고 하면 됩니다. 어때요, 간단하죠?

하지만, 문제는 4차원 배열 부터 입니다. 뭐 우리는 3 차원 적인 세상에서 살고 있기 때문에 4 차원에 무엇인지 몸에 와닿기는 힘듭니다. (사실, 우리는 3차원 상의 공간에 시간의 축이 더해진 4차원 세상에서 살고 있다고 합니다) 그렇기에 4 차원 배열, 그리고 그 보다 더 놓은 차원의 배열이 무엇인지 머리속으로 그려보기란 고역이 아닐 수 없습니다.

그런데 말이죠. 제가 아까 굵은 글씨로 써 놓았던 것이 기억나시나요?

일차원 배열은 한 개의 값(x)으로 원소에 접근하는 것이고, 이차원 배열은 두 개의 값(x,y)으로 원소에 접근하는 것이다!

이를 확장해서 생각해 보면 삼차원 배열은 세 개의 값 `(x,y,z)` 을 통해서 원소에 접근하는 것입니다. 네, 맞아요. 우리가 원소가 몇 번째 층에 있고 (x), 그 층에 해당하는 평면에 몇 행(y), 그리고 몇 열(z) 를 알면 `int` 변수에 정확하게 접근할 수 있지 않습니까?

4 차원도 같습니다. 4 차원 배열은 4 개의 값 `(x,y,z,w)` 을 통해서 원소에 접근할 수 있습니다. 마찬가지로 5 차원은 5 개, `n` 차원은 `n` 개의 값을 통해서 원소에 접근하게 되는 것이지요. 이 아이디어를 적용시키면 어떠한 차원의 배열이 실제 프로그래밍 상에 필요하다고 하더라도 문제 없이 해결할 수 있으리라 생각합니다.

그렇다면 이번 강좌는 여기에서 마치도록 하겠습니다.

생각해 보기

문제 1

제 강좌 제목에서 배열이 왜 C 언어의 아파트 인지 설명해 보세요. 즉, 동 의 개념, 층 의 개념, 호 의 개념이 어떠한 배열을 형상화 하고 있는 지도 생각해 보세요. (난이도 : 下)