# 동적 메모리 할당

## 동적 메모리 할당 및 반환

- **정적 할당**
  - 변수 선언을 통해 필요한 메모리 할당
    - 많은 양의 메모리는 배열 선언을 통해 할당
- **동적 할당**
  - 필요한 양이 예측되지 않는 경우. 프로그램 작성시 할당 받을 수 없음
  - 실행 중에 운영체제로부터 할당 받음
    - 힙(heap)으로부터 할당
      - 힙은 운영체제가 소유하고 관리하는 메모리. 모든 프로세스가 공유할 수 있는 메모리
- **C 언어의 동적 메모리 할당 : malloc()/free() 라이브러리 함수 사용**
- **Java 의 동적 메모리 할당/반환**
  - new 연산자
    - 기본 타입 메모리 할당, 배열 할당, 객체 할당, 객체 배열 할당
    - 객체의 동적 생성
    - 객체 할당 시 생성자 호출
  - 반환
    - JVM이 GC 과정을 통해 반환

## malloc/free 함수

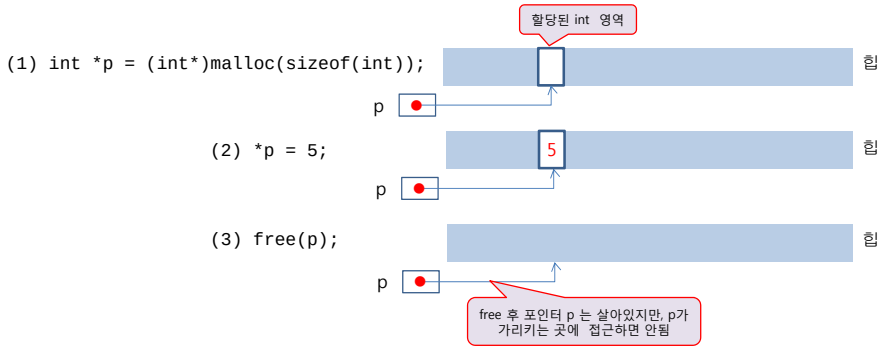
- **stdlib.h 을 include해야 함**
- **사용 형식**

```
데이터타입 *포인터변수 = (데이터 타입*) malloc( 메모리 크기);

free( 포인터변수 );
```

- **사용 예제**

```
int *pInt = (int*) malloc (sizeof(int)); // int 타입의 메모리 동적 할당
char *pChar = (char*) malloc (sizeof(char)); // char 타입의 메모리 동적 할당

free(pInt); // 할당 받은 정수 공간 반환
free( pChar); // 할당 받은 문자 공간 반환
```

## 1) 기본 타입의 메모리 동적 할당 및 반환



## 예제 4-5 정수형 공간의 동적 할당 및 반환 예

```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int *p; int n;

   p = (int*) malloc(sizeof(int));
   if(!p) {
      printf("메모리를 할당할 수 없습니다.");
      return 0;
   }

   *p = 5; // 할당 받은 정수 공간에 5 삽입
   n = *p;
   printf ("*p = %d₩n", *p);
   printf( "n = %d₩n", n);

   free (p);
}
```

int 타입 1개 할당

p 가 NULL이면, 메모리 할당 실패

할당 받은 메모리 반환

```
*p = 5
n = 5
```

## free 사용 시 주의 사항

- **적절치 못한 포인터로 free하면 실행 시간 오류 발생**
  - 동적으로 할당 받지 않는 메모리 반환 – 오류

```
int n;
int *p = &n;
free( p ); // 실행 시간 오류
// 포인터 p가 가리키는 메모리는 동적으로 할당 받은 것이 아님
```

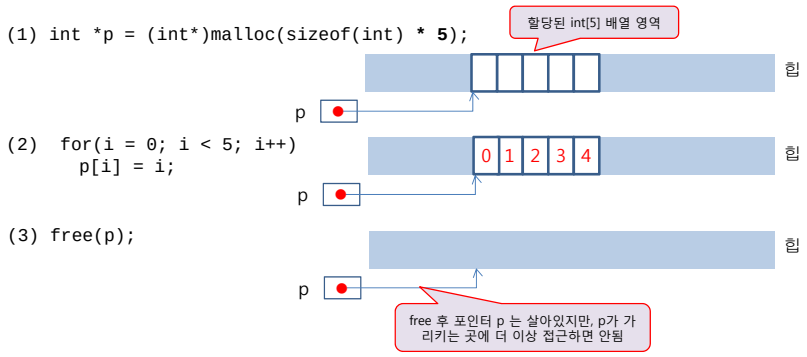
  - 동일한 메모리 두 번 반환 - 오류

```
int *p = (int*) malloc(sizeof(int));
free (p); // 정상적인 메모리 반환
free (p); // 실행 시간 오류. 이미 반환한 메모리를 중복 반환할 수 없음
```

## 2) 배열의 동적 할당 및 반환

- **연산자의 사용 형식**

```
데이터타입 *포인터변수 = (데이터타입*) malloc (sizeof(데이터타입) * [배열의 크기]); // 동적 배열 할당
free( 포인터변수); // 배열 반환
```




## 정수형 배열의 동적 할당 및 반환

사용자로부터 입력할 정수의 개수를 입력 받아 배열을 동적 할당 받고,

하나씩 정수를 입력 받은 후

합을 출력하는 프로그램을 작성하라.

```
입력할 정수의 개수는?4
1번째 정수: 4
2번째 정수: 20
3번째 정수: -5
4번째 정수: 9
평균 = 7
```

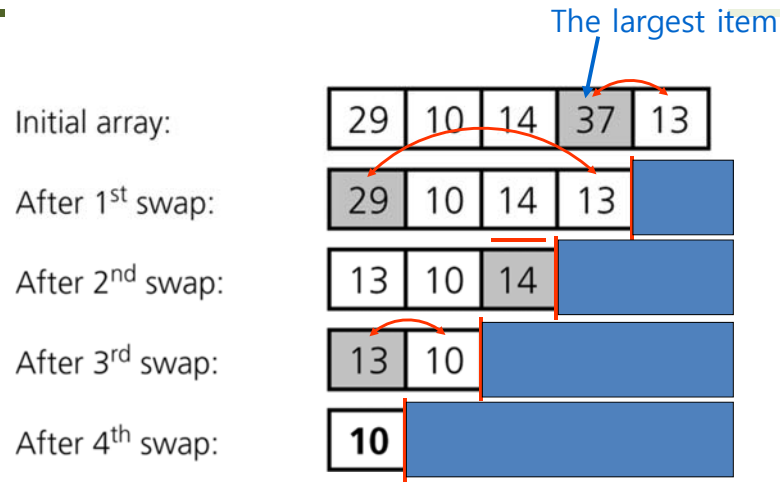
```
#include <stdio.h>
#include <stdlib.h>

int main() {
  int n,sum,i;
  int *p;
  scanf("%d", &n); // 정수의 개수 입력
  if(n <= 0) return 0;
  p = (int*) malloc(sizeof(int) * n); // n 개의 정수배열 동적 할당

  if(!p) {
    printf("메모리를 할당할 수 없습니다.");
    return 0;
  }

  for(i=0; i<n; i++) {
    scanf("%d", &p[i]); // 키보드로부터 정수 입력
  }

  sum = 0;
  for(i=0; i<n; i++)
    sum += p[i];
  printf  ("평균 = %d\n", sum/n );

  free (p); // 배열 메모리 반환
}
```

# 정렬Sorting

## 학습목표

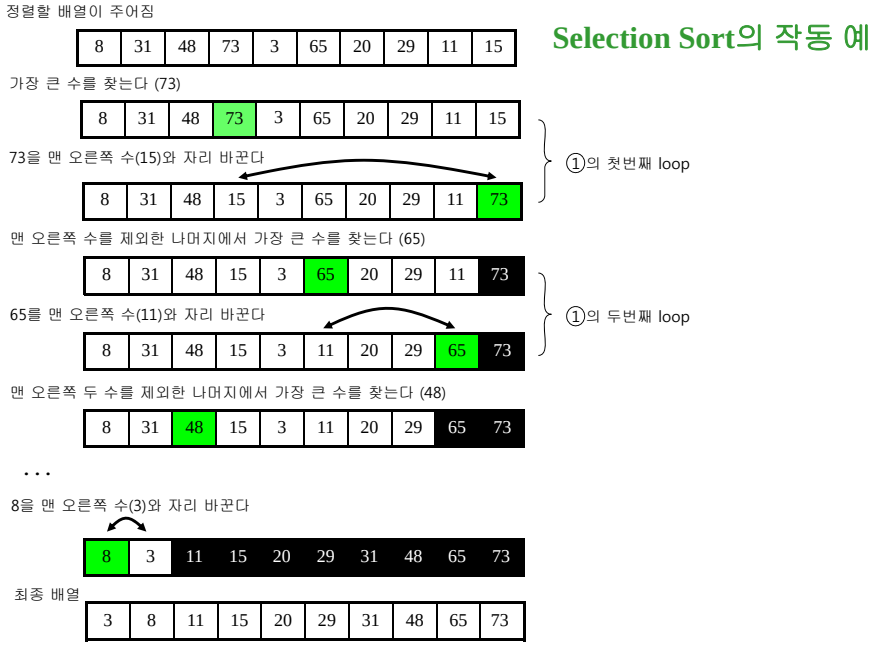
- **Computer Science에서 가장 많이 접하게 되는 문제 중의 하나인 Sorting (정렬)에 대해서 학습.**
  - **Selection Sort**
  - **Bubble Sort**
  - **Insertion Sort**
  - **Merge Sort**
  - **Quick Sort**
  - Heap Sort
  - Radix Sort
  - Counting Sort
  - ...

## Selection Sort(선택 정렬)



## Selection Sort

- 각 루프마다
  - 최대 원소를 찾는다
  - 최대 원소와 맨 오른쪽 원소를 교환한다
  - 맨 오른쪽 원소를 제외한다
- 하나의 원소만 남을 때까지 위의 루프를 반복




```
selectionSort(int* A, n)        ▷ 배열 A를 정렬한다
{
  for (int i = 0; i < n-1; i++ {        // n-1 반복
    A[0] ... A[_____] 중 가장 큰 수 A[max_idx]를 찾는다;
    A[max_idx] ↔ A[_____];  //두 배열 원소를 교환
  }
}
```

```
selectionSort(int* A, n) // 배열 A를 정렬한다
{
    for (i = 0; i< n-1; i++ {     // n-1 반복
        A[0] ... A[n-1-i] 중 가장 큰 수 A[max_idx]를 찾는다;
        A[max_index] ↔ A[n-1-i]; //두 배열 원소를 교환
    }
}
```

*version1*

**배열에서 가장 큰 수 찾는 것도 loop**

// loop가 끝나면 배열에서 가장 큰 수와 그 index를 알게 됨.

가장 큰 수를 찾는다 (73)

| 8 | 31 | 48 | 73 | 3 | 65 | 20 | 29 | 11 | 15 |
|---|---|---|---|---|---|---|---|---|---|

①의 첫번째 loop

맨 오른쪽 수를 제외한 나머지에서 가장 큰 수를 찾는다 (65)

| 8 | 31 | 48 | 15 | 3 | 65 | 20 | 29 | 11 | 73 |
|---|---|---|---|---|---|---|---|---|---|

①의 두번째 loop

맨 오른쪽 두 수를 제외한 나머지에서 가장 큰 수를 찾는다 (48)

| 8 | 31 | 48 | 15 | 3 | 11 | 20 | 29 | 65 | 73 |
|---|---|---|---|---|---|---|---|---|---|

...

*version1*

n, n-1, n-2, n-3 ... 3, 2

```
for (i = 0; i < n-1; i++) {
  max = A[0]; max_idx=0;
  for (j = 1; j < n-i; j++) {
    if(max < A[j] ) {
      max = A[j];
      max_idx = j;
    }
  }
  // loop가 끝나면 배열에서 가장 큰 수와 그 index를 알게 됨.
```

가장 큰 수를 찾는다 (73)

| 8 | 31 | 48 | 73 | 3 | 65 | 20 | 29 | 11 | 15 |
|---|---|---|---|---|---|---|---|---|---|

①의 첫번째 loop

맨 오른쪽 수를 제외한 나머지에서 가장 큰 수를 찾는다 (65)

| 8 | 31 | 48 | 15 | 3 | 65 | 20 | 29 | 11 | 73 |
|---|---|---|---|---|---|---|---|---|---|

①의 두번째 loop

맨 오른쪽 두 수를 제외한 나머지에서 가장 큰 수를 찾는다 (48)

| 8 | 31 | 48 | 15 | 3 | 11 | 20 | 29 | 65 | 73 |
|---|---|---|---|---|---|---|---|---|---|

...

*version2*

n, n-1, n-2, n-3 ... 3, 2

```
for (i = 0; i < n-1; i++ {
for (i = n; i > 1; i--) {
  max = A[0]; max_idx=0;
  for (j = 1; j < i; j++) {
    if( max < A[j] ) {
      max = A[j];
      max_idx = j;
    }
  }
  // loop가 끝나면 배열에서 가장 큰 수와 그 index를 알게 됨.
```

IT COOKBOOK

- 오름차순으로의 정렬을 위해 선택 정렬을 사용할 때
  - **큰 값을 찾아** 오른쪽으로 보내는 방법(우리가 조금 전 살펴본)과
  - **작은 값을 찾아** 왼쪽으로 보내는 방법도 가능하다. 다음 슬라이드는 두 번째 방법을 보여준다.

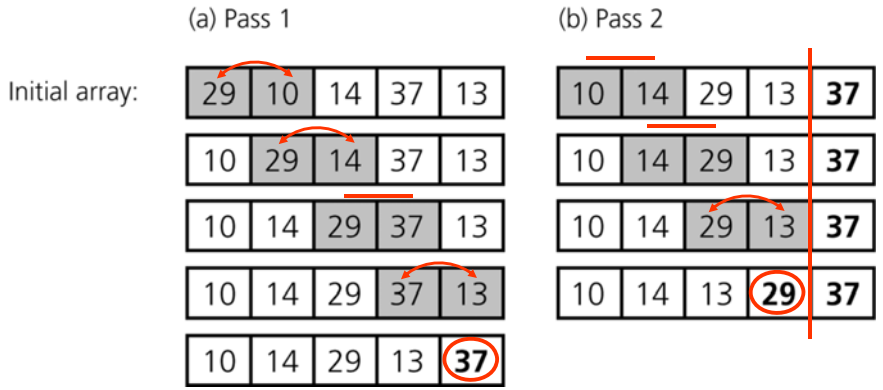
## 선택정렬(selection sort)


선택정렬 유사코드

```
selection_sort(A, n)

for i←0 to n-2 do
        least ← A[i], A[i+1],..., A[n-1] 중에서 가장 작은 값의 인덱스;
        A[i]와 A[least]의 교환;
        i++;
```

# Bubble Sort(버블 정렬)

## Bubble Sort




```
bubbleSort(int *A, n)     ▷ 배열 A를 정렬한다
{
  int i;
  for (i = 0; i < n-1; i++ {       // n-1 반복
     인접한 두 숫자의 SUB 배열의 끝까지 순서만 바꿈.
}
```

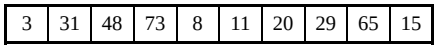
```
bubbleSort(int *A, n)   ▷ 배열 A를 정렬한다
{
    for (int i = 0; i < n-1; i++ {     // n-1 반복
        인접한 두 숫자의 SUB 배열의 끝까지 순서만 바꿈.
}
```

이것도 loop

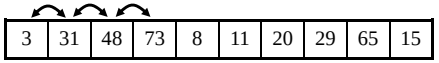
```
int j;
for (j = 0; j < n-1-i; j++ {
  if( A[j] > A[j+1] ) {
     A[j]와 A[j+1]을 SWAP
  }
}
// loop가 끝나면 SUB 배열에서 가장 큰 수가 제일 뒤에
```

## Bubble Sort의 작동 예

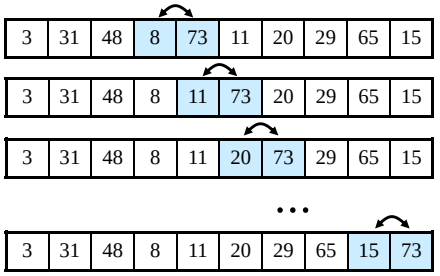
정렬할 배열이 주어짐



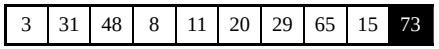

왼쪽부터 시작해 이웃한 쌍들을 비교해간다




순서대로 되어 있지 않으면 자리 바꾼다

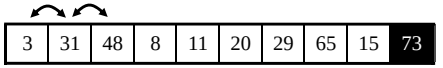



맨 오른쪽 수(73)를 대상에서 제외한다

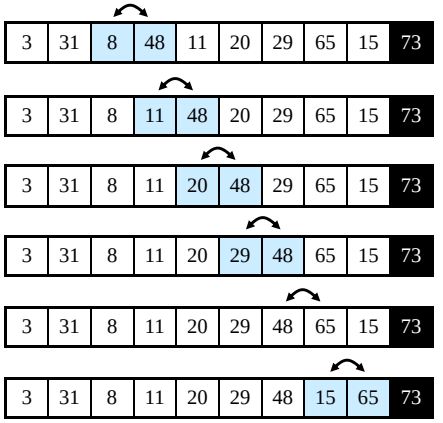



*Pass 1*

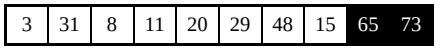
왼쪽부터 시작해 이웃한 쌍들을 비교해간다




순서대로 되어 있지 않은 경우에는 자리 바꾼다



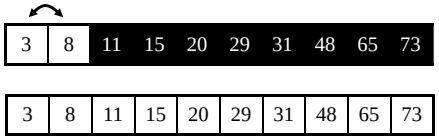

맨 오른쪽 수(65)를 대상에서 제외한다



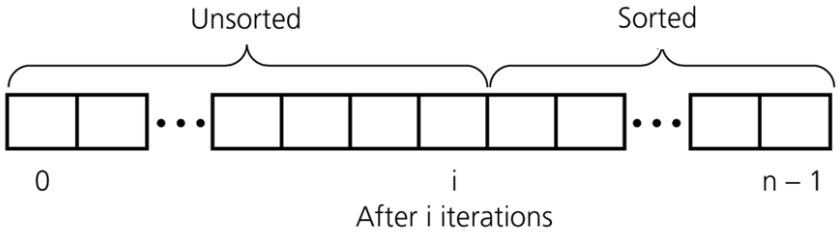

*Pass 2*

앞의 작업을 반복하면서 계속 제외해 나간다


두개짜리 배열의 처리를 끝으로 정렬이 완료된다



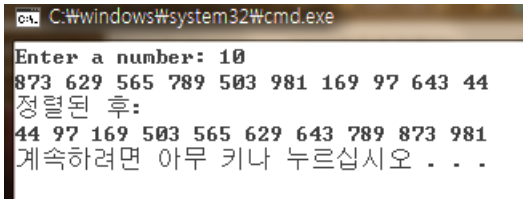

*Pass n-1*



# 동적 할당
# 정렬(selection, bubble)

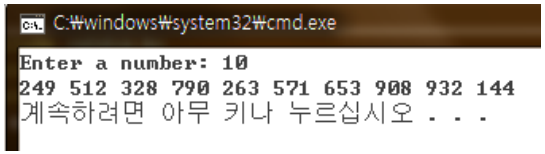
## Lab(selection)

- 임의의 정수 n을 입력 받아 n개의 정수 (0에서 999까지)의 난수를 만들어 이를 오름차순으로 정렬하는 프로그램을 작성하시오. 정렬하는 부분은 함수로. **(selection sort)**
  - 동적 할당 : n개의 정수를 저장할 수 있는 배열 생성
  - 배열에는 랜덤 정수를 저장
    - srand(time(NULL), rand()함수 사용
  - Selection sort로 정렬.
  - 프로그램 종료 전에 배열 반환

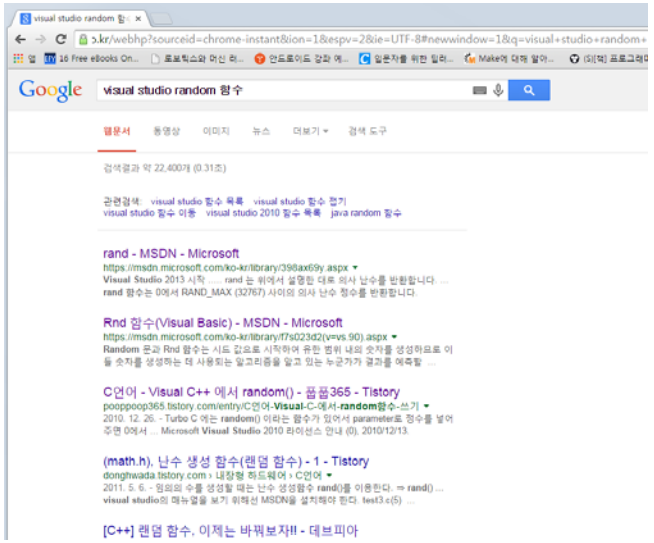



## Lab(1차원동적할당)

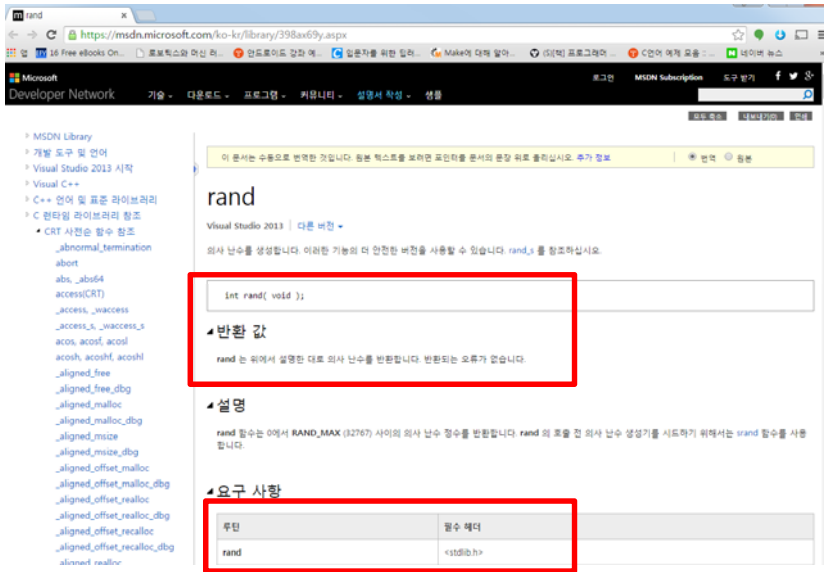
- 임의의 정수 n을 입력 받아 n개의 정수 (0에서 999까지)의 난수를 만들어 이를 출력하는 프로그램을 작성하라
  - 동적 할당 : n개의 정수를 저장할 수 있는 배열 생성
  - 배열에는 랜덤 정수를 저장
  - 프로그램 종료 전에 배열 반환




## Random (검색)



# MSDN




# Seed (Random)

- 역시 google 검색 후 MSDN

# srand

Visual Studio 2013 | 다른 버전

의사 난수 생성기에 대한 시작 시드 값을 설정합니다.

```
void srand(
   unsigned int seed
);
```

### 매개 변수

*seed*
의사 난수 생성에 대한 시드

### 설명

**srand** 함수는 현재 스레드의 일련의 의사 난수 정수를 생성 하기 위한 시작점을 설정 합니다. 호출 시퀀스가 동일한 결과 만드는 생성자를 초기화 하기 위해, **srand** 작동을 호출하고 *seed* 인수와 똑같은 이름을 다시 사용합니다. *seed* 에 대한 다른 값은 생성자를 난수 시퀀스의 다른 시작 지점으로 설정합니다. **rand** 생성 된 난수를 검색 합니다. **rand** 를 **srand** 가 **srand** 와 1로 전달된 *seed* 를 호출하는 같은 시퀀서를 호출하기 전에 호출합니다.

### 요구 사항

| 루틴 | 필수 헤더 |
|---|---|
| **srand** | <stdlib.h> |

- Seed를 매번 다르게 하기 위해서는 현재 시간을 srand의 파라미터로..
- 역시 google 검색 후 MSDN

시스템 시간을 가져옵니다.

```
time_t time(
   time_t *timer
);
__time32_t _time32(
   __time32_t *timer
);
__time64_t _time64(
   __time64_t *timer
);
```

### ◢매개 변수

*timer*
시간에 대한 저장소 위치에 대한 포인터입니다.

### ◢반환 값

오류 발생 시-1 또는 1970 년 1 월 1 일 자정 이후 경과 된 초 시간을 반환 합니다.

### ◢설명

**time** 함수는 자정 (00:00), 1970, 1 월 1 일 UTC (협정 세계시) 이후 경과 된 시간을 초 단위로 반환합니다. 반환 값은 *timer* 가 제공한 위치에 저장 됩니다. 이 매개 변수는 반환 값이 저장되지 않는 경우 **NULL** 일 수 있습니다.

**time** 는 **_time64** 및 **time_t** 에 대한 래퍼이며 기본적으로 **__time64_t** 와 같습니다. 컴파일러가 이전 32 비트 **time_t** 로 **time_t**를 해석 할 수 있도록 하는 경우, **_USE_32BIT_TIME_T** 를 정의할 수 있습니다. 응용 프로그램이 2038년 1월 18일 후 실패 할 수 있기 때문에 사용하지 않는 것이 좋습니다 이 매크로의 사용은 64 비트 플랫폼에서 사용할 수 없습니다.

### ◢요구 사항

| 루틴 | 필수 헤더 |
|---|---|
| time | <time.h> |

**코드의 예:**

```
int a,b;
srand( time(NULL) );

a = rand();
b = rand();
...
```

# Lab(bubble)

- 이제는.. Bubble sort로
  - 동적 할당 : n개의 정수를 저장할 수 있는 배열 생성
  - 배열에는 랜덤 정수를 저장
  - bubble sort로 함수를 정렬.
  - 프로그램 종료 전에 배열 반환

# HW(selection)

- 학생의 성적을 정렬하는 프로그램 작성.
  - 임의의 학생의 수 n을 입력 받음.
  - 학생은 학번, 영어, 수학, 국어 성적을 attribute(member)로 가진다 ➔ struct
    - 모든 attribute는 int 형
    - 학번은 1번부터 시작. 성적은 0~100점 사이의 랜덤 정수.
  - 국어 성적 기준으로 내림 차순으로 정렬
  - Selection sort로 정렬 (함수)

```
struct Student
{
	int id; //학번. 1번 부터 부여 ..
	int korean, english, math;
};

int main()
{
	//n 입력 받음
	//Student 구조체 배열을 동적으로 할당

	// 학번 부여
	// random으로 성적 저장

	// 학생 정보(학번, 성적들) 출력

	//국어 성적 기준으로 내림 차순 정렬 → selection sort 함수 호출

	// (정렬된) 학생 정보(학번, 성적들) 출력

	//동적으로 할당 받은 구조체 배열 반환
};
```

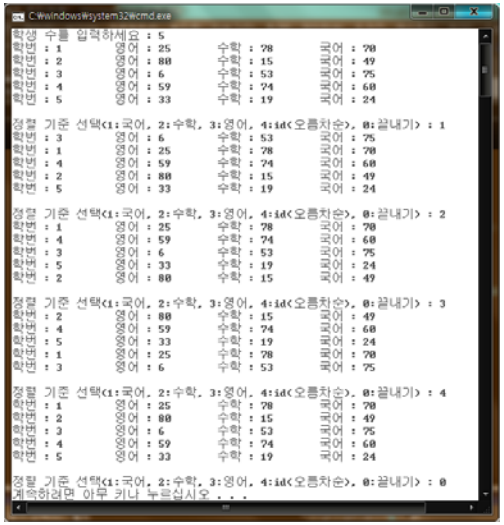
# HW(bubble)

- 이번엔 bubble sort로.. 단, 국어 성적으로 내림 차순 정렬한 후 다시 id로 오름차순 정렬(즉, 원본으로)한다. 각각의 정렬 결과를 출력한다.
- 두 개의 함수를 사용한다.
  - bubbleSortDescendingByKorean
  - bubbleSortAscendingById

# HW(bubble**2**)

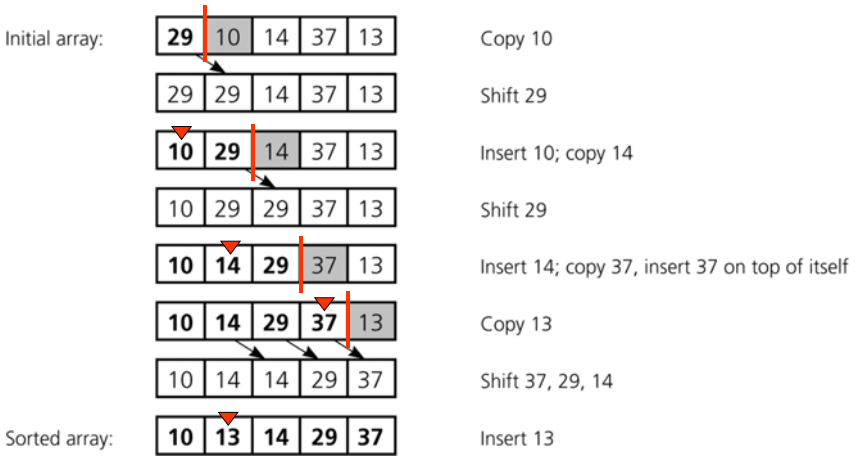
원하는 기준(국어/영어/수학/id)를 main에서 선택 후 그 기준에 따라 정렬하는 버전으로 작성하라.

- 하나의 bubbleSort 사용
- 그 외 다른 함수(swap?)
- id는 오름차순으로, 성적으로 정렬 시는 내림차순
- 원할 때까지 반복적으로 기준을 선택해 정렬한다.

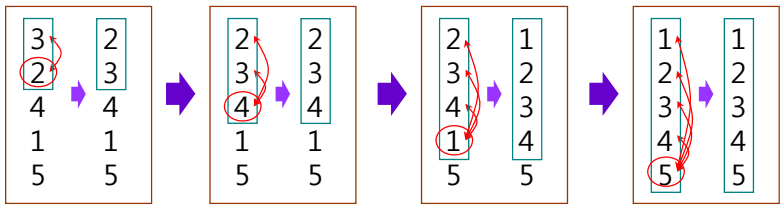


# Insertion Sort(삽입 정렬)

## Insertion Sort




## Insertion Sort



```
insertionSort(int* A, n)      ▷ 배열 A을 정렬한다
{
      int i;
      for (i  = 1 ; i < n ; i++ )
            A[0]...A[i-1]의 적당한 자리에 A[i]를 삽입한다;
}
```

```
insertionSort(int* A, n)      ▷ 배열 A을 정렬한다
{
    int i;
    for ( i = 1 ; i < n ; i++ )
        A[0]...A[i-1]의 적당한 자리에 A[i]를 삽입한다;
}
```

**A[i]가 들어갈 적당한 자리부터...**

**A[i]가 들어갈 적당한 자리부터...**

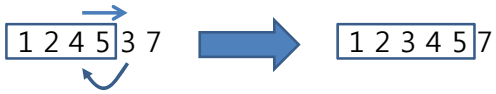
```
for (j = 0; j < i; j++)
  if( A[j] > A[i] ) break;

// loop가 끝나면 A[j] 위치에 A[i]가 들어가야 함을 알 수 있음
```

1 2 4 5 3 7

i=4, A[i] = 3; 인 상황

```
for(j=0 ; j < 4 ; j++ ) // 비교를 최대 A[3]까지 함.
        if ( A[j] > A[4] ) break;
```

이 부분을 수행하면
j ← 2 이고 A[4]는 A[2]에 들어가야 한다는 뜻
A[2], A[3]은 뒤로 이동하고
.




i=4, A[i] = 3; 인 상황., j = 2

먼저 A[4]를 적당한 변수 temp에 저장; `temp = A[i];`

A[3] → A[4] ⇒ 1 2 4 5 5 7

A[2] → A[3] ⇒ 1 2 4 4 5 7

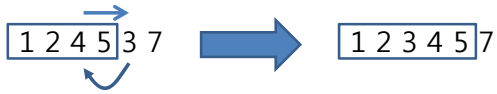
의 순서로 뒤로 밀어야 함.

그리고 A[2]에 m을 대입; `A[2] = temp;` ⇒ 1 2 3 4 5 7

만약 순서를 이상하게 하면 이상한 일이 발생.

A[2] → A[3] ⇒ 1 2 4 4 3 7

A[3] → A[4] ⇒ 1 2 4 4 4 7 ✗




i=4, A[i] = 3; 인 상황., j = 2

먼저 A[4]를 적당한 변수 temp에 저장; `temp = A[i];`

A[3] → A[4] ⇒ 1 2 4 5 5 7

A[2] → A[3] ⇒ 1 2 4 4 5 7

```
//오른쪽으로 옮기기


A[j] = temp;
```

```
insertionSort(int* A, n)        ▷ 배열 A을 정렬한다
{
        int i;
       for ( int i  = 1 ; i < n ; i++ )
              A[0]...A[i-1]의 적당한 자리에 A[i]를 삽입한다;
}
```

```
insertionSort(int* A, n) //배열 A을 정렬한다
{
    int i;
    int j, k, temp;

    for (i = 1 ; i < n ; i++)
    {
        for (j = 0; j < i; j++)
            if (A[j] > A[i]) break;

        temp = A[i];
        for (k = i; k > j; k--)
            A[k] = A[k-1];
        A[j] = temp;
    }
}
```

# Inductive Verification of Insertion Sort

- 배열 **A[0]**만 놓고 보면
  - 정렬되어 있음
- 배열 **A[0 … k]**까지 정렬되어 있다면
  - ➔ ② 행의 삽입에 의해 A[0 … k+1]까지 정렬된다

✓고등학교에서 배운 수학적 귀납법과 다를 바 없음

# Lab(insertionSort)

- 임의의 정수 n을 입력 받아 n개의 정수 (10만 이하)의 난수를 만들어 이를 정렬하는 프로그램을 작성하시오. 정렬하는 부분은 함수로. **(insertion sort)**
  - 동적 할당 : n개의 정수를 저장할 수 있는 배열 생성
  - 배열에는 랜덤 정수를 저장
  - insertion sort로 정렬.
  - 프로그램 종료 전에 배열 반환