제법 쓸만한 수준의 힙 구현

◆ 프로그래머가 힙의 우선순위 판단 기준을 설정할 수 있어야 한다!

```
typedef struct _heapElem
{
     Priority pr;
     HData data;
} HeapElem;

typedef struct _heap
{
     int numOfData;
     HeapElem heapArr[HEAP_LEN];
} Heap;
```

구조체의 변경!

typedef int PriorityComp(HData d1, HData d2);

```
typedef struct _heap
{
     PriorityComp * comp;
     int numOfData;
     HData heapArr[HEAP_LEN];
} Heap;
```

```
void HeapInit(Heap * ph, PriorityComp pc)
{
     ph->numOfData = 0;
     ph->comp = pc;
}
```

구조체의 변경에 따른 초기화 함수의 변경!

제법 쓸만한 수준의 힙 구현
(PriorityComp형 함수의 정의 기준)

- 첫 번째 인자의 우선순위가 높다면, 0보다 큰 값 반환!
- 두 번째 인자의 우선순위가 높다면, 0보다 작은 값 반환!
- 첫 번째, 두 번째 인자의 우선순위가 동일하다면, 0이 반환!

```
void HInsert(Heap * ph, HData data, Priority pr);
```

우선 순위 정보를 별도로 받지 않는다.

```
void HInsert(Heap * ph, HData data);
```

PriorityComp형 함수를 이용하여 우선순위 판단!

제법 쓸만한 수준의 힙 구현
(구조체 타입 선언)

```
#include <stdio.h>
#define TRUE   1
#define FALSE  0
#define HEAP_LEN      100

typedef char HData;
typedef int PriorityComp(HData d1, HData d2);
typedef struct _heap {
        PriorityComp * comp;
        int numOfData;
        HData heapArr[HEAP_LEN];
} Heap;

int DataPriorityComp(HData ch1, HData ch2) {
        return ch2 - ch1;
        //return ch1-ch2;
}
```

d1의 우선순위가 높다면 0보다 큰 값
d2의 우선순위가 높다면 0보다 작은 값
d1과 d2의 우선순위가 같다면 0을 반환

int (*comp)(HData d1, HData d2);

363-364p

제법 쓸만한 수준의 힙 구현
(초기화 함수와 Helper 함수들)

```
void HeapInit(Heap * ph, PriorityComp pc) {
	ph->numOfData = 0;
	ph->comp = pc;
}

int HIsEmpty(Heap * ph) {
	if (ph->numOfData == 0)
		return TRUE;
	else
		return FALSE;
}

int GetParentIDX(int idx) { return idx / 2; }

int GetLChildIDX(int idx) { return idx * 2; }

int GetRChildIDX(int idx) { return GetLChildIDX(idx) + 1; }
```

제법 쓸만한 수준의 힙 구현
(초기화 함수와 Helper 함수들)

```
int GetHiPriChildIDX(Heap * ph, int idx) {
    int leftIndex = GetLChildIDX(idx);
    int rightIndex = GetRChildIDX(idx);
    if (leftIndex > ph->numOfData)
        return 0;
    else if (leftIndex == ph->numOfData)
        return leftIndex;
    else {
        //if(ph->heapArr[leftIndex].pr > ph->heapArr[rightIndex].pr)
        if (ph->comp(ph->heapArr[leftIndex],
            ph->heapArr[rightIndex]) < 0)
            return rightIndex;
        else
            return leftIndex;
    }
}
```

제법 쓸만한 수준의 힙 구현
(HInsert 함수)

```
void HInsert(Heap * ph, HData data) {
    int idx = ph->numOfData + 1;

    while (idx != 1) {
        //if(pr < (ph->heapArr[GetParentIDX(idx)].pr))
        if (ph->comp(data, ph->heapArr[GetParentIDX(idx)]) > 0) {
            ph->heapArr[idx] = ph->heapArr[GetParentIDX(idx)];
            idx = GetParentIDX(idx);
        }
        else
            break;
    }

    ph->heapArr[idx] = data;
    ph->numOfData += 1;
}
```

제법 쓸만한 수준의 힙 구현
(HDelete 함수)

```
HData HDelete(Heap * ph) {
    HData retData = ph->heapArr[1];
    HData lastElem = ph->heapArr[ph->numOfData];
    int parentIdx = 1;
    int childIdx;

    while (childIdx = GetHiPriChildIDX(ph, parentIdx)) {
        //if(lastElem.pr <= ph->heapArr[childIdx].pr)
        if (ph->comp(lastElem, ph->heapArr[childIdx]) >= 0)
            break;

        ph->heapArr[parentIdx] = ph->heapArr[childIdx];
        parentIdx = childIdx;
    }
    ph->heapArr[parentIdx] = lastElem;
    ph->numOfData -= 1;
    return retData;
}
```

365-366p

제법 쓸만한 수준의 힙 구현
(main 함수 : 이전에서 수정)

```
int main(void) {
	Heap heap;
	HeapInit(&heap, DataPriorityComp);

	HInsert(&heap, 'A');
	HInsert(&heap, 'B');
	HInsert(&heap, 'C');
	printf("%c ₩n", HDelete(&heap));

	HInsert(&heap, 'A');
	HInsert(&heap, 'B');
	HInsert(&heap, 'C');
	printf("%c ₩n", HDelete(&heap));

	while (!HIsEmpty(&heap))
		printf("%c ₩n", HDelete(&heap));
	return 0;
}
```

제법 쓸만한 수준의 힙을 이용한 우선순위 큐의 구현 (이전 소스에 추가)

```
typedef Heap PQueue;
typedef HData PQData;

void PQueueInit(PQueue * ppq, PriorityComp pc) {
	HeapInit(ppq, pc);
}

int PQIsEmpty(PQueue * ppq) {
	return HIsEmpty(ppq);
}

void PEnqueue(PQueue * ppq, PQData data) {
	HInsert(ppq, data);
}

PQData PDequeue(PQueue * ppq) {
	return HDelete(ppq);
}
```

제법 쓸만한 수준의 힙을 이용한 우선순위 큐의 구현 (이전 소스 수정)

```
int main(void) {
	PQueue pq;
	PQueueInit(&pq, DataPriorityComp);

	PEnqueue(&pq, 'A');
	PEnqueue(&pq, 'B');
	PEnqueue(&pq, 'C');
	printf("%c ₩n", PDequeue(&pq));

	PEnqueue(&pq, 'A');
	PEnqueue(&pq, 'B');
	PEnqueue(&pq, 'C');
	printf("%c ₩n", PDequeue(&pq));

	while (!PQIsEmpty(&pq))
		printf("%c ₩n", PDequeue(&pq));
	return 0;
}
```

학생 정보의 우선순위 큐

- 기존에는 구현을 간단히 하기 위해 int 타입을 대상으로 함.
- 실제 활용 예로써 학생 정보를 대상으로 하도록 수정해 보자.
  - 우선순위 : 점수가 높을수록 우선순위가 높다.
- 현재 소스의 어느 부분을 어떤 순서로 수정해야 할지 구상하기!

```
typedef struct {
        int id;
        char name[20];
        float score;
} HData;

int DataPriorityComp(HData ch1, HData ch2) {
        return (ch1.score - ch2.score);     // 점수 높은 순서
}
```

학생 정보의 우선순위 큐

```
int main(void) {
	Heap heap;
	HeapInit(&heap, DataPriorityComp);
	HData data[3] = {
	{ 2, "Park", 79.5}, { 1, "Lee", 75.5}, { 3, "Kim", 88}, };
	HData get;

	for (int i = 0;i < 3;i++)
		HInsert(&heap, data[i]);
	get = HDelete(&heap);
	printf("%d %s %.2f₩n", get.id, get.name, get.score);

	for (int i = 0;i < 3;i++)
		HInsert(&heap, data[i]);
	get = HDelete(&heap);
	printf("%d %s %.2f₩n", get.id, get.name, get.score);

	while (!HIsEmpty(&heap)) {
		get = HDelete(&heap);
		printf("%d %s %.2f₩n", get.id, get.name, get.score);
	}
	return 0;
}
```

학생 정보의 우선순위 큐

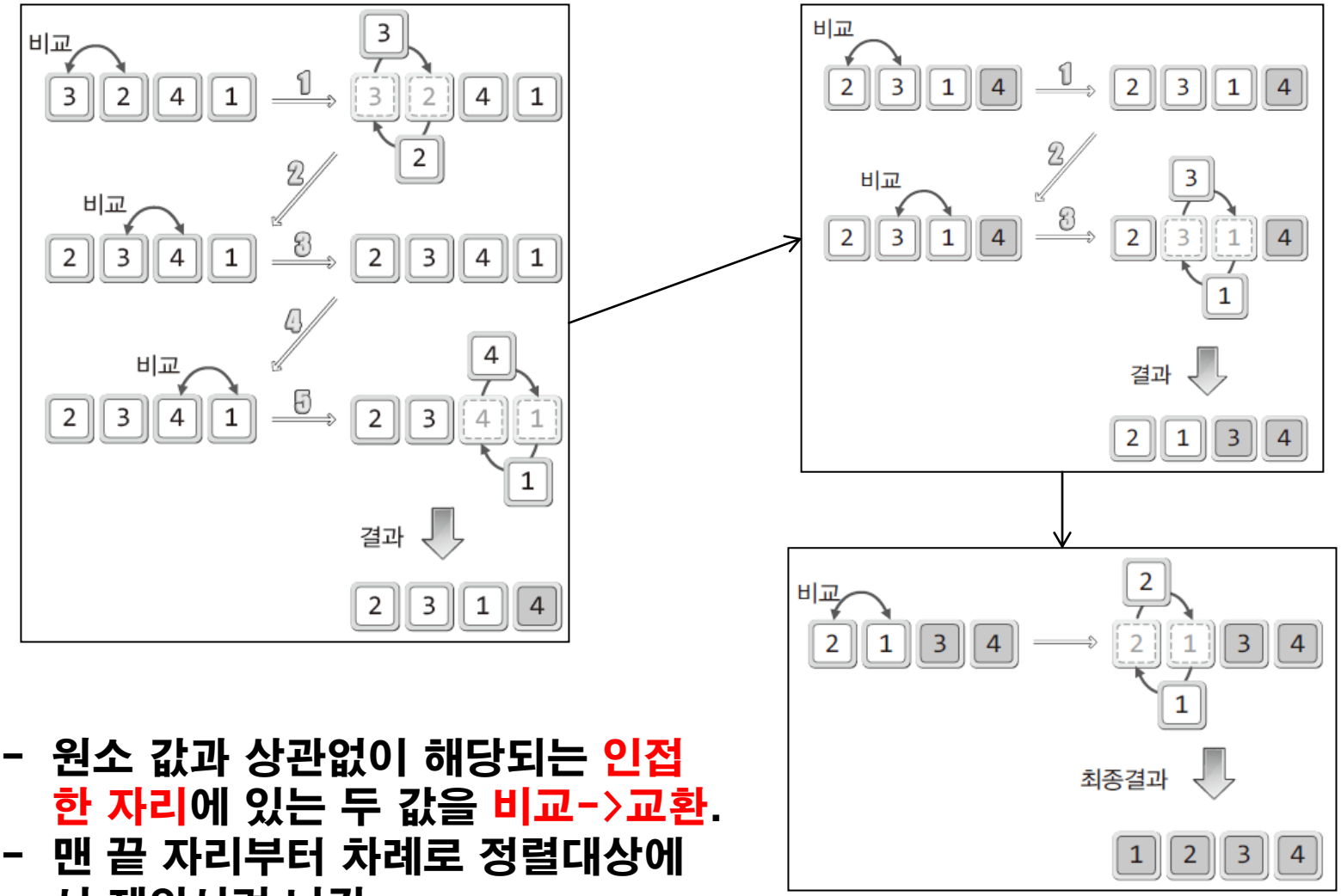
- **우선순위 변경**
  - **id 값이 작을수록 우선순위가 높다.**

```
int DataPriorityComp(HData ch1, HData ch2) {
        return (ch2.id - ch1.id);
}
```

- **우선순위 변경**
  - **이름을 가나다순으로 나오게 하기.**

```
int DataPriorityComp(HData ch1, HData ch2) {
        return strcmp(ch2.name, ch1.name);
}
```

버블 정렬
(이해)



- 원소 값과 상관없이 해당되는 **인접 한 자리**에 있는 두 값을 **비교->교환**.
- 맨 끝 자리부터 차례로 정렬대상에서 제외시켜 나감.

버블 정렬
(구현)

```
#include <stdio.h>

void BubbleSort(int arr[], int n) {
	int i, j;
	int temp;

	for (i = 0; i < n - 1; i++) {
		for (j = 0; j < (n - i) - 1; j++) {
			if (arr[j] > arr[j + 1]) {
				temp = arr[j];
				arr[j] = arr[j + 1];
				arr[j + 1] = temp;
			}
		}
	}
}
```

버블 정렬
(구현)

```
int main(void)
{
	int arr[4] = { 3, 2, 4, 1 };
	int i;

	BubbleSort(arr, sizeof(arr) / sizeof(int));

	for (i = 0; i < 4; i++)
		printf("%d ", arr[i]);

	printf("₩n");
	return 0;
}
```