자료 구조

[프로그래밍 과제 #2]

heap 생성 및 heap sort

제출일자 : 2017.11.15

담당교수 : 이경아 교수님

분반번호 : 01

학과 : 컴퓨터공학과

학번 : 1615051

이름 : 이영은

연락처 : 010-6729-0907

자료구조 레포트

1. 문제 기술

: 1부터 100까지의 정수를 random number 30개를 만들어서 heap을 생성하고, 이렇게 생성한 heap을 heap sort를 사용하여 최소 heap으로 나타내고 이것을 오름차순으로 30개의 수를 나타내 보여주는 문제이다.

2. 입출력의 예

: 예를 들어 random number 5개인 {64 23 77 1 5}가 주어졌다고 하겠다. 주어진 number로 heap을 생성하고 heap sort를 사용하면 그 결과 값은 오름차순으로 표현되어 {1 5 23 64 77}로 출력 될 것이다.

3. 문제풀이 방법(알고리즘)

: 먼저 main 함수에서 배열을 선언하고, HeapType \*h을 선언한다. 선언한 h를 초기화를 시켜준 뒤, index의 1부터 30까지 for을 이용하여 random number을 arr[index]에 넣는다. 넣은 arr[index]를 다시 h->heap[index].key에 넣어서 heap을 표현한다. 그리고 h->heap\_size를 1씩 더하여 뒤의 heap\_sort에서 delete\_min\_heap을 쓸 때 사용한다. 왜냐하면 delete\_min\_heap의 코드에서 heap\_size를 하나씩 줄여나가면서 값을 나타내는 코드가 있기 때문이다.

그 다음, print\_heap(h) 함수를 이용하여 heap 모양으로 출력되게 한다. 만약 i가 heap의 높이인 level 1과 같으면, 그 다음줄로 넘어가면서 level이 곱하기 2가 된다. 이러한 알고리즘으로 heap을 화면에 출력해낸다.

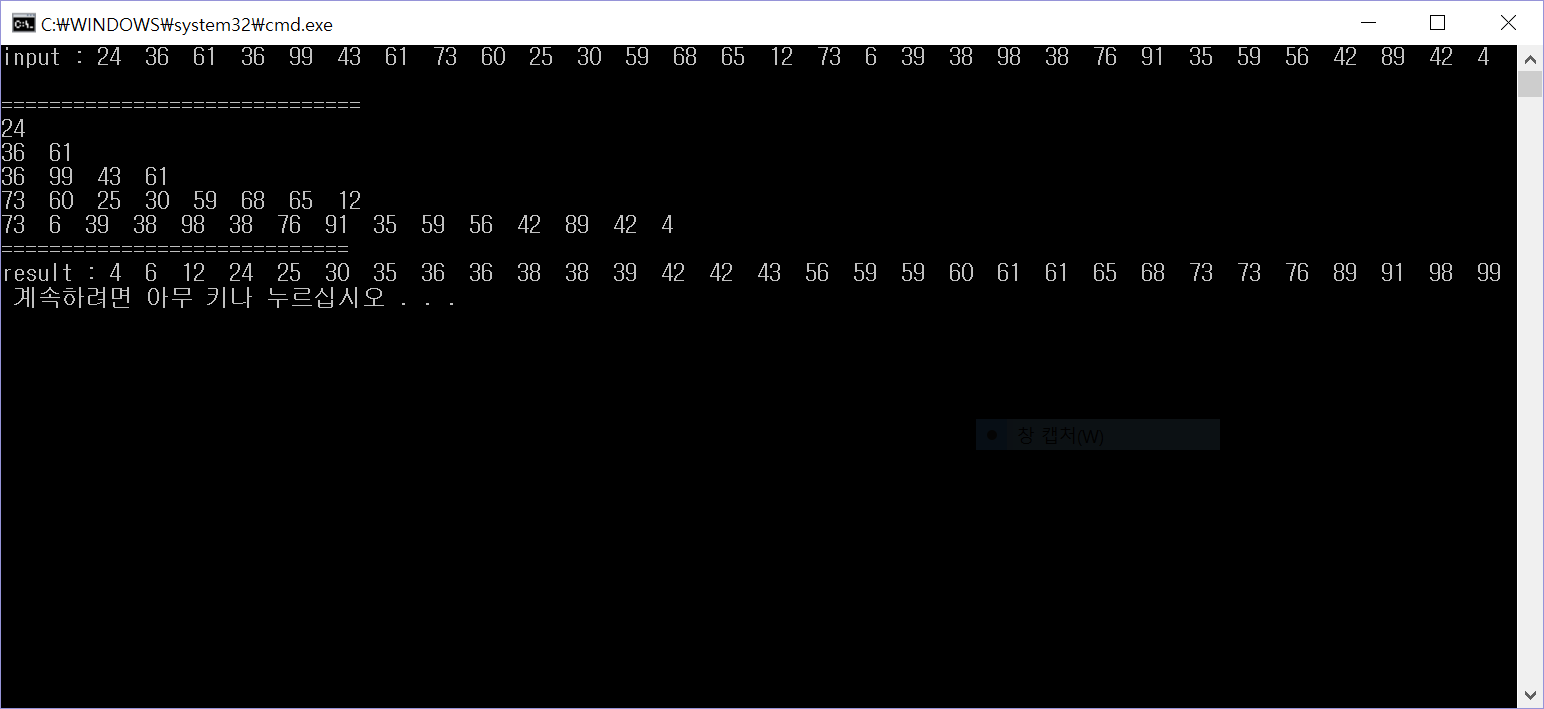
그리고 heap\_sort(h->heap,MAX) 함수를 이용하여 min heap 정렬을 한다. 먼저 HeapType인 h를 정의하고 이를 초기화한다. 그리고 min\_heap(HeapType \*h,element item) 함수를 이용하여 heap에 최소 heap으로 정렬한다. i는 1이 아니고 item.key이 부모 노드의 값보다 작으면 부모 노드는 자식 노드로 가고, heap\_size는 줄어든다. 이러한 while을 빠져나오면 item을 h->heap[i]에 넣는다. 이렇게 최소 heap으로 정렬된 이진 트리를 delete\_min\_heap을 이용해 하나씩 지워나가면서 배열에 다시 넣는 동작을 한다. item을 h->heap[i]로 정의하고 temp를 h->heap[(h->heap\_size)--]로 정의한다. while문을 통해서 뿌리의 왼쪽 자식부터 비교를 시작하는데 child가 heap\_size보다 작거나 같을 때 만약 자식노드가 heap\_size보다 작고 왼쪽 자식노드가 오른쪽 자식 노드보다 크면 child를 +1한다. 만약 temp.key가 h->heap[child].key 보다 작거나 같으면 break한다. 그리고 h->heap[parent]에 h->heap[child]를 넣고 parent는 child가 되고 child는 곱하기 2를 한다. 이렇게 while문이 끝나면 temp값은 h->heap[parent]로 넣고 item을 return 한다.

4. 소스 프로그램

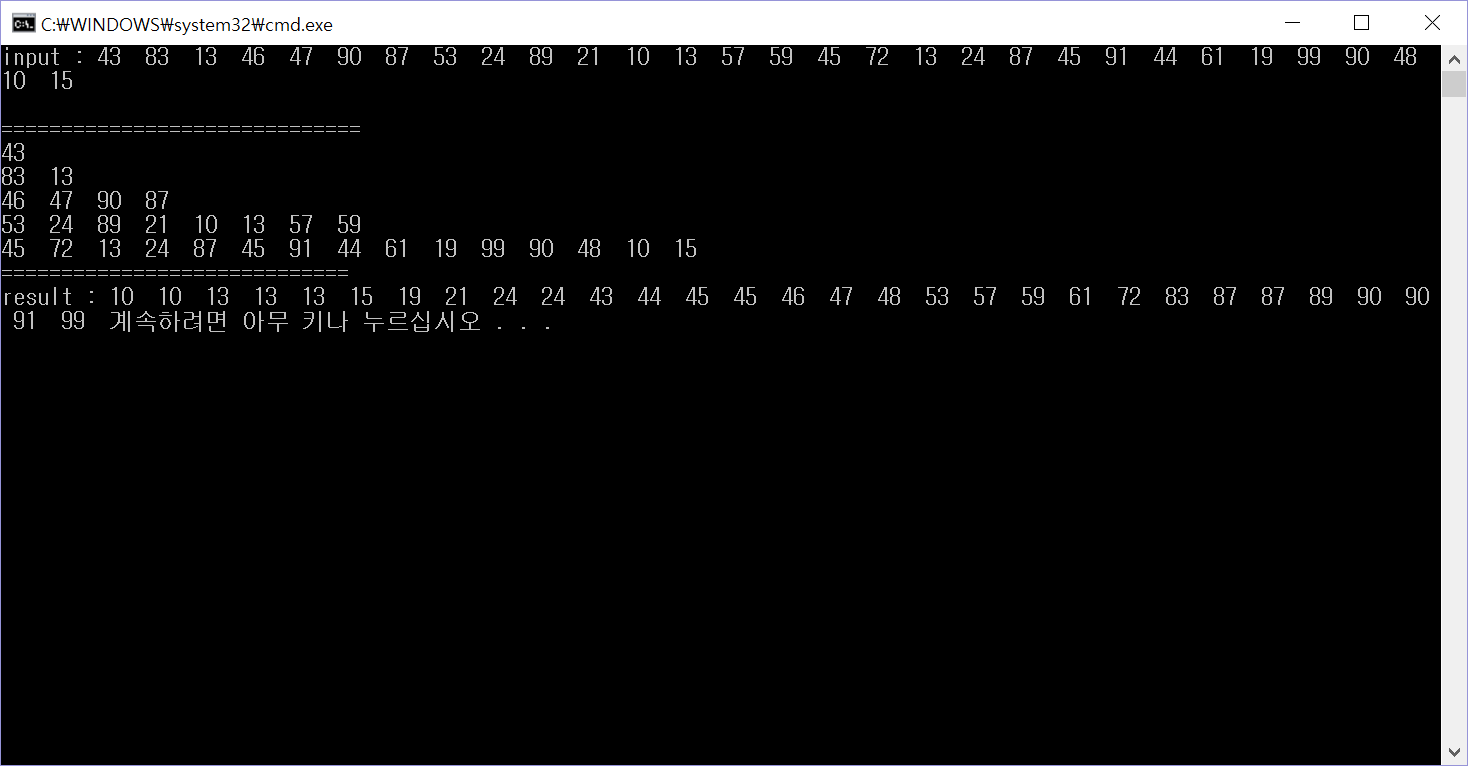
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #define MAX 31  #define MAX\_ELEMENT 200  void min\_heap();  typedef struct{  int key;  }element;  typedef struct{  element heap[MAX\_ELEMENT];  int heap\_size;  }HeapType;//히프를 구현  //힙 초기화 함수  void init(HeapType \*h)  {  h->heap\_size=0;  }  void min\_heap(HeapType \*h, element item)  {  int i;  i = ++(h->heap\_size);  while((i != 1) && (item.key < h->heap[i/2].key)) {  h->heap[i] = h->heap[i/2];  i /= 2;  }  h->heap[i] = item;    }  element delete\_min\_heap(HeapType \*h)  {  int parent, child;  element item, temp;  item = h->heap[1];  temp = h->heap[(h->heap\_size)--];  parent = 1;  child = 2;  while( child <= h->heap\_size ){  // 현재 노드의 자식노드중 더 작은 자식노드를 찾는다.  if( ( child < h->heap\_size ) &&  (h->heap[child].key) > h->heap[child+1].key)  child++;  if( temp.key <= h->heap[child].key ) break;  // 한단계 아래로 이동  h->heap[parent] = h->heap[child];  parent = child;  child \*= 2;  }  h->heap[parent] = temp;  return item;  }  void heap\_sort(element arr[], int n)  {  int i;  HeapType h;  init(&h);  for(i=0;i<n;i++){  min\_heap(&h,arr[i]);    }  for(i=(n-1);i>=0;i--){  arr[i]= delete\_min\_heap(&h);  }  printf("\nresult : ");  for(i=(n-2);i>=0;i--){  printf("%d",arr[i]);  printf(" ");  }  }  void print\_heap(HeapType \*h)  {  int i;  int level = 1;  printf("\n==============================");  for (i = 1; i <= h->heap\_size; i++) {  if (i == level) {  printf("\n");  level \*= 2;  }  printf("%d", h->heap[i].key);  printf(" ");  }  printf("\n=============================");  }  int main(void)  {    int arr[MAX];  int index=0;  int i=0;  HeapType \*h;  h=(HeapType\*)malloc(sizeof(HeapType));  init(h);  srand(time(NULL));  printf("input : ");  for(index=1;index<MAX;index++){  arr[index]=(rand()%100)+1;  h->heap[index].key=arr[index];  printf("%d",h->heap[index]);  printf(" ");  (h->heap\_size)++;  }  printf("\n");    print\_heap(h);    heap\_sort(h->heap,MAX);  return 0;  } |

5. 수행 결과

1)



2)



3)



6. 결과 분석 및 토의

:

random number을 문제 없이 heap에 넣어서 이것을 heap 모양으로 문제없이 출력하였고, heap에 넣은 random number을 heap\_sort를 사용하여 문제 없이 오름차순으로 출력하였다. 그리고 따로 printf(“ “);를 추가하여 숫자들의 출력값을 보기 좋게 하였다. random number을 정의하는 방법을 몰랐는데, 이번 과제를 통해서 random number을 사용하는 방법을 알았다. 또한 rand()%100은 0부터 100까지의 random number이고, (rand()%100)+1은 1부터 100까지의 random number을 나타내는 것이라는 흥미로운 사실을 알게 되었다. 그리고 구조체 선언을 통해서 어떻게 heap을 구현하고 구조체에서 선언한 heap\_size는 heap 안에 저장된 요소의 개수라는 의미를 새롭게 알게 되었다.