Insertion Sort Program in C

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it is to be inserted there. Hence the name insertion sort.

Implementation in C

Live Demo

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

#define MAX 7

int intArray[MAX] = {4,6,3,2,1,9,7};

void printline(int count) {
   int i;

   for(i = 0;i < count-1;i++) {
      printf("=");
   }

   printf("=\n");
}

void display() {
   int i;
   printf("[");

   // navigate through all items
   for(i = 0;i < MAX;i++) {
      printf("%d ",intArray[i]);
   }

   printf("]\n");
}

void insertionSort() {
   int valueToInsert;
   int holePosition;
   int i;

   // loop through all numbers
   for(i = 1; i < MAX; i++) {

      // select a value to be inserted.
      valueToInsert = intArray[i];

      // select the hole position where number is to be inserted
      holePosition = i;

      // check if previous no. is larger than value to be inserted
      while (holePosition > 0 && intArray[holePosition-1] > valueToInsert) {
         intArray[holePosition] = intArray[holePosition-1];
         holePosition--;
         printf(" item moved : %d\n" , intArray[holePosition]);
      }

      if(holePosition != i) {
         printf(" item inserted : %d, at position : %d\n" , valueToInsert,holePositio
         // insert the number at hole position
         intArray[holePosition] = valueToInsert;
      }

      printf("Iteration %d#:",i);
      display();

   }
}

void main() {
   printf("Input Array: ");
   display();
   printline(50);
   insertionSort();
   printf("Output Array: ");
   display();
   printline(50);
}
```

If we compile and run the above program, it will produce the following result −

Output

```
Input Array: [4 6 3 2 1 9 7 ]
==================================================
Iteration 1#:[4 6 3 2 1 9 7 ]
 item moved : 6
 item moved : 4
 item inserted : 3, at position : 0
Iteration 2#:[3 4 6 2 1 9 7 ]
 item moved : 6
 item moved : 4
 item moved : 3
 item inserted : 2, at position : 0
Iteration 3#:[2 3 4 6 1 9 7 ]
 item moved : 6
 item moved : 4
 item moved : 3
 item moved : 2
 item inserted : 1, at position : 0
Iteration 4#:[1 2 3 4 6 9 7 ]
Iteration 5#:[1 2 3 4 6 9 7 ]
 item moved : 9
 item inserted : 7, at position : 5
Iteration 6#:[1 2 3 4 6 7 9 ]
Output Array: [1 2 3 4 6 7 9 ]
==================================================
```