

Lab 4

Data Structures
C++ for C Coders

한동대학교 김영섭 교수
idebtor@gmail.com

function pointer
bubble sort

Lab 04: Bubble sort

- Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. For example,
- 1st Pass:
 - (**5** **1** 4 2 8) \rightarrow (**1** **5** 4 2 8), It compares the first two elements, and swaps since $5 > 1$.
 - (1 **5** **4** 2 8) \rightarrow (1 **4** **5** 2 8), Swap since $5 > 4$
 - (1 4 **5** **2** 8) \rightarrow (1 4 **2** **5** 8), Swap since $5 > 2$
 - (1 4 2 **5** **8**) \rightarrow (1 4 2 **5** **8**), Now, since these elements are already in order ($8 > 5$), it does not swap them.

Lab 04: Bubble sort

- Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. For example,
- 1st Pass:
 - (**5** **1** 4 2 8) \rightarrow (**1** **5** 4 2 8), It compares the first two elements, and swaps since $5 > 1$.
 - (1 **5** **4** 2 8) \rightarrow (1 **4** **5** 2 8), Swap since $5 > 4$
 - (1 4 **5** **2** 8) \rightarrow (1 4 **2** **5** 8), Swap since $5 > 2$
 - (1 4 2 **5** **8**) \rightarrow (1 4 2 **5** **8**), Now, since these elements are already in order ($8 > 5$), it does not swap them.
- 2nd Pass:
 - (**1** **4** 2 5 8) \rightarrow (**1** **4** 2 5 8)
 - (1 **4** **2** 5 8) \rightarrow (1 **2** **4** 5 8), Swap since $4 > 2$
 - (1 2 **4** **5** 8) \rightarrow (1 2 **4** **5** 8)
 - (1 2 4 **5** **8**) \rightarrow (1 2 4 **5** **8**)
 - Now, the sequence is already sorted, but the algorithm does not know if it is completed. It needs one **whole** pass without **any** swap to know it is sorted.

Lab 04: Bubble sort

- 3rd Pass:
 - (**1** **2** 4 5 8) \rightarrow (**1** **2** 4 5 8)
 - (1 **2** **4** 5 8) \rightarrow (1 **2** **4** 5 8)
 - (1 2 **4** **5** 8) \rightarrow (1 2 **4** **5** 8)
 - (1 2 4 **5** **8**) \rightarrow (1 2 4 **5** **8**)
 - Sorting is over since no element is swapped.
- It can be implemented as shown without using a function pointer.

Lab 04: Bubble sort

- 3rd Pass:
 - (**1** **2** 4 5 8) → (**1** **2** 4 5 8)
 - (1 **2** **4** 5 8) → (1 **2** **4** 5 8)
 - (1 2 **4** **5** 8) → (1 2 **4** **5** 8)
 - (1 2 4 **5** **8**) → (1 2 4 **5** **8**)
 - Sorting is over since no element is swapped.
- It can be implemented as shown without using a function pointer.

```
void bubblesort(int *list, int n) {
    int i, j, temp;

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

```
void print_list(int *list, int N) {
    for (int i = 0; i < N; i++)
        cout << list[i] << " ";
    cout << endl;
}
```

Lab 04: Bubble sort

- The simple bubble sort implemented here sorts an int array in ascending order.
- Use the following code to test it.

```
int main() {  
    int list[] = { 2, 8, 1, 9, 5 };  
    int N = sizeof(list)/sizeof(list[0]);  
  
    cout << "UNSORTED: " << endl;  
    print_list(list, N);  
  
    bubblesort(list, N);  
    cout << "SORTED: " << endl;  
    print_list(list, N;  
  
}
```

```
void bubblesort(int *list, int n) {  
    int i, j, temp;  
  
    for (i = 0; i < n - 1; i++) {  
        for (j = 0; j < n - i - 1; j++)  
            if (list[j] > list[j + 1]) {  
                temp = list[j];  
                list[j] = list[j + 1];  
                list[j + 1] = temp;  
            }  
        }  
    }  
}
```

```
void print_list(int *list, int N) {  
    for (int i = 0; i < N; i++)  
        cout << list[i] << " ";  
    cout << endl;  
}
```

- Now, we want to modify `bubblesort()` such that it takes a comparator function pointer as an additional argument.
- Then, the user will be able to control the behaviors of `bubblesort()` to sort in either ascending or descending order.
- The comparator function that takes two ints and returns an int as shown below:

```
int (*compare) (int a, int b)
```
- Define the following two comparator functions as shown below:
 - `int ascend(int a, int b)`
 - `int descend(int a, int b)`

- `bubblesort()` function must work with `main()` function given here.
 - Note that the default comparator function is to sort the list in ascending order. This means that you must use the default argument featured in C++, but not in C.

```
int main() {  
    int list[] = { 2, 8, 1, 9, 5 };  
    int N = sizeof(list)/sizeof(list[0]);  
  
    cout << "UNSORTED: " << endl;  
    print_list(list, N);  
  
    bubblesort(list, N);  
    cout << "SORTED Up: " << endl;  
    print_list(list, N);  
  
    bubblesort(list, N, descend);  
    cout << "SORTED Dn: " << endl;  
    print_list(list, N);  
}
```


- Files to submit:
 - **fps.cpp, qsort.cpp**
 - no credit/point for these parts since most code presented in class
 - but required for your class attendance
 - and may get a penalty if the code does not work
 - **bubblesort.cpp – for credit**
- Due:
 - 11:55 pm, on the day of this lecture
- Grade:
 - 0.5

Lab 4

Data Structures
C++ for C Coders

한동대학교 김영섭 교수
idebtor@gmail.com

function pointer
bubble sort