Programming in Modern C++

Module M04: Sorting and Searching

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

All url's in this module have been accessed in September, 2021 and found to be functional

Module Recap

- Working with variable sized arrays is more flexible with `vector`s in C++
- String operations are easier with C++ standard library

Module Objectives

- Implementation of Sorting and Searching in C and C++

Module Outline

1. Sorting in C and C++
   - Bubble Sort
   - Using Standard Library
2. Searching in C and C++
   - Using Standard Library
3. STL: algorithm - The `algorithm` Library
4. Module Summary

Sorting in C and C++

Sorting in C and C++

Program 04.01: Bubble Sort

C Program

```
#include <stdio.h>

int main() { int data[] = {32, 71, 12, 45, 26};
    int i, step, n = 5, temp;
    for(step = 0; step < n - 1; ++step)
        for(i = 0; i < n-step-1; ++i) {
            if(data[i] > data[i+1]) {
                temp = data[i];
                data[i] = data[i+1];
                data[i+1] = temp;
            }
        }

    for(i = 0; i < n; ++i)
        printf("%d  ", data[i]);
}
```

```
12 26 32 45 71
```

C++ Program

```
#include <iostream>
using namespace std;
int main() { int data[] = {32, 71, 12, 45, 26};
    int n = 5, temp;
    for(int step = 0; step < n - 1; ++step)
        for(int i = 0;i < n-step-1; ++i) {
            if (data[i] > data[i+1]) {
                temp = data[i];
                data[i] = data[i+1];
                data[i+1] = temp;
            }
        }

     for(int i = 0; i < n; ++i)
         cout << data[i] << " ";
}
```

```
12 26 32 45 71
```

- Implementation is same in both C and C++ apart from differences in header files, I/O functions explained in Module 02

Program 04.02: Using sort from standard library

C Program (Desc order)

```
#include <stdio.h>
#include <stdlib.h> // qsort function

// compare Function Pointer
int compare(
    const void *a, const void *b) { // Type unsafe
    return (*(int*)a < *(int*)b);   // Cast needed
}
int main () { int data[] = {32, 71, 12, 45, 26};
    // Start ptr., # elements, size, func. ptr.

    qsort(data, 5, sizeof(int), compare);

    for(int i = 0; i < 5; i++)
        printf ("%d ", data[i]);
}
```

```
71 45 32 26 12
```

- sizeof(int) and compare function passed to qsort
- compare function is type unsafe & needs complicated cast

C++ Program (Desc order)

```
#include <iostream>
#include <algorithm> // sort function
using namespace std;
// compare Function Pointer
bool compare(
    int i, int j) { // Type safe
    return (i > j); // No cast needed
}
int main() { int data[] = {32, 71, 12, 45, 26};
    // Start ptr., end ptr., func. ptr.

    sort(data, data+5, compare);

    for (int i = 0; i < 5; i++)
        cout << data[i] << " ";
}
```

```
71 45 32 26 12
```

- Only compare passed to sort. No size is needed
- Only Size is inferred from the type int of data
- compare function is type safe & simple with no cast

Program 04.03: Using default sort of algorithm

C++ Program (Asc Order)

```
// sort.cpp
#include <iostream>
#include <algorithm> // sort function
using namespace std;

int main () {
    int data[] = {32, 71, 12, 45, 26};

    sort(data, data+5);

    for (int i = 0; i < 5; i++)
        cout << data[i] << " ";

    return 0;
}
```

```
12 26 32 45 71
```

- Sort using the default sort function of algorithm library which does the sorting in ascending order only
- No compare function is needed

Searching in C and C++

Searching in C and C++

Program 04.04: Binary Search

C Program

```
#include <stdio.h>
#include <stdlib.h> // bsearch function

// compare Function Pointer
int compare(
    const void * a, const void * b) { // Type unsafe
    if (*(int*)a<*(int*)b) return -1; // Cast needed
    if (*(int*)a==*(int*)b) return 0; // Cast needed
    if (*(int*)a>*(int*)b) return 1;  // Cast needed
}
int main () { int data[] = {1,2,3,4,5}, k = 3;

    if (bsearch(&k, data, 5, sizeof(int), compare))
        printf("found!\n");
    else printf("not found\n");
}
```

```
found!
```

- compare function is type unsafe & needs complicated cast

C++ Program

```
#include <iostream>
#include <algorithm> // binary_search function
using namespace std;

int main() { int data[] = {1,2,3,4,5}, k = 3;

    if (binary_search(data, data+5, k))
        cout << "found!\n";
    else cout << "not found\n";
}
```

```
found!
```

- No compare function needed

STL: algorithm - The `algorithm` Library

STL: algorithm - The `algorithm` Library

The algorithm Library

The algorithm library of c++ helps us to easily implement commonly used complex functions. We discussed the functions for sort and search. Let us look at some more useful functions.

- Replace element in an array
- Rotates the order of the elements

Program 04.05: replace and rotate functions

Replace

```
// Replace.cpp
#include <iostream>
#include <algorithm> // replace function
using namespace std;

int main() {
    int data[] = {1, 2, 3, 4, 5};

    replace(data, data+5, 3, 2);

    for(int i = 0; i < 5; ++i)
        cout << data[i] << " ";

    return 0;
}
```

```
1 2 2 4 5
```

- 3^{rd} element replaced with 2

Rotate

```
// Rotate.cpp
#include <iostream>
#include <algorithm> // rotate function
using namespace std;

int main() {
    int data[] = {1, 2, 3, 4, 5};

    rotate(data, data+2, data+5);

    for(int i = 0; i < 5; ++i)
        cout << data[i] << " ";

    return 0;
}
```

```
3 4 5 1 2
```

- Array circular shifted around 3^{rd} element

Module Summary

- Flexibility of defining *customised* sort algorithms to be passed as parameter to sort and search functions defined in the `algorithm` library
- Predefined optimised versions of these sort and search functions can also be used
- There are a number of useful functions like `rotate`, `replace`, `merge`, `swap`, `remove` etc. in `algorithm` library