Data Structures & Programming

A Reminder on C++ Part 2

Golnar Sheikhshab

Sorting

An inefficient sorting algorithm: insertion sort

Idea: Read items one at a time and insert each item in its rightful place thus far

Illustration: Input : 64 90 12 51 3

Step	Sorted_until_now	Item Read	Sorted_until_after_item_read
1	[]	64	[64]
2	[64]	90	[64 90]
3	[64 90]	12	[12 64 90]
4	[12 64 90]	51	[12 51 64 90]
5	[12 51 64 90]	3	[3 12 51 64 90]

Sorting Program

- Let's keep the sorted list in an array

```
const int max_num_items = 100;
int sorted[max_num_items];
int n=0;
```

- Read positive integers from standard input and update the array

```
int x=-100;
while ((n < max_num_items) && (x != -1)){
        cin >> x;
        if (x > 0 )
                update(sorted, n, x);
        cin.clear();
        cin.ignore(1000, '\n');
}
```

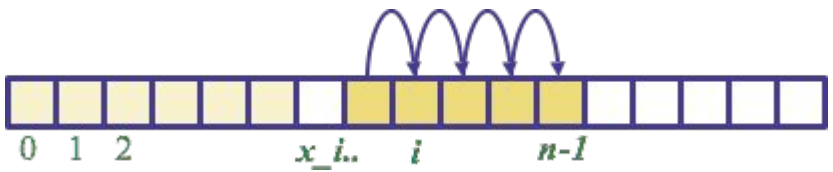
Sorting Program - Updating the Array

```
void update(int* sorted_array, int & n, int x )
{
	// finding the rightful place of x
	int x_index = right_place(sorted_array, n, x);

	n++;

	// shift everything starting from x_index to the right to open a space for x
	for (int i= n-2; i>= x_index; i--)
		sorted_array[i+1] = sorted_array[i];

	sorted_array[x_index] = x;
}
```



Sorting Program - Finding the Rightful Place of x

```
int right_place(int* sorted_array, int & n, int x){
        int x_index = n;
        for (int i = 0; i<n; i++){
                if (sorted_array[i] >= x){
                        x_index = i;
                        break;
                }
        }
        return x_index;
}
```

Sorting Program - Seeing the output

To see the effect of each update we make it print the content of the array as well

```
void print(int* sorted_array, int n){
        for (int i=0; i<n; i++)
                cout << sorted_array[i] << " ";
        cout << endl;
}

void update(int* sorted_array, int & n, int x )
{
       // same as before ....
       print(sorted_array, n);
}
```

Sorting Program - The output

g++ -o insertion_sort.o insertion_sort_of_integers_by_array.cpp

./insertion_sort.o

5 // input
5
43 // input
5 43
21 // input
5 21 43
10 // input
5 10 21 43
-1 // input -- stop

What if we want a generic sorting function?

A function that can work with different types? use c++ templates

```
template<typename Type>
int right_place(Type* sorted_array, int & n, Type x){
	int x_index = n;
	for (int i = 0; i<n; i++){
		if (sorted_array[i] >= x){
			x_index = i;
			break;
		}
	}
	return x_index;
}
```

Sorting program - generic update

```
template<typename Type>
void update(Type* sorted_array, int & n, Type x )
{
        // finding the rightful place of x
        int x_index = right_place(sorted_array, n, x);

        n++;

        // shift everything starting from x_index to the right to open a space for x
        for (int i= n-2; i>= x_index; i--)
                sorted_array[i+1] = sorted_array[i];

        sorted_array[x_index] = x;
       print (sorted_array, n);
}
```

Sorting program - generic - works with strings

```
void sort_string(){
        string sorted[max_num_items];
        int n=0;

        string x = "<start>";
        while ((n < max_num_items) && (x != "<stop>")){
                cin >> x;
                update(sorted, n, x);
        }
}
```

Sorting program - generic - works with integers

```
void sort_positive_int(){
        int sorted[max_num_items];
        int n=0;
        int x=-100;
        while ((n < max_num_items) && (x != -1)){
                cin >> x;
                if (x > 0 ){
                        update(sorted, n, x);
                }
                cin.clear();
                cin.ignore(1000, '\n');
        }
}
```

Does our generic sort work for any type now?

- For example for the following user-defined class?

```
class patient{
public:
        patient(string name, int severity): name_(name), severity_(severity){}
private:
        string name_;
        int severity_;
};
```

Let's try and see ...

- But first, let's read the input from file
- patients.txt contains patients, one per line:

```
p1      1
p100    100
p73     73
p21     21
p2      2
p10     10
p73     73
```

Let's try and see ... (2)

```
#include <fstream>
using namespace std;

        ifstream inf;
         inf.open("patients.txt");
         string line;
         while ((n<max_num_items) && (getline(inf,line))){
                int tab_index = line.find("\t");
                string name = line.substr(0, tab_index);
                int severity = atoi(line.substr(tab_index).c_str());
                // use the name and severity
         }
         inf.close();
```

Let's try and see ... (3)

```
void sort_patients(){
	patient sorted[max_num_items];
	int n=0;
	ifstream inf;
	inf.open("patients.txt");
	string line;
	while ((n<max_num_items) && (getline(inf,line))){
		int tab_index = line.find("\t");
		string name = line.substr(0, tab_index);
		int severity = atoi(line.substr(tab_index).c_str());
		update(sorted, n, patient(name, severity));
	}
	inf.close();
}
```

Let's try and see ... (4)

g++ -o isort_gen_with_patients_first_try.o insertion_sort..._first_try.cpp

insertion_sort_..._first_try.cpp:86:10: error: no matching constructor for initialization of 'patient [100]'
    patient sorted[max_num_items];
            ^
insertion_sort_..._first_try.cpp:9:7: note: candidate constructor (the implicit copy constructor) not viable: requires 1 argument, but 0 were provided
class patient{
  ^
insertion_sort_..._first_try.cpp:11:2: note: candidate constructor not viable: requires 2 arguments, but 0 were provided
    patient(string name, int severity): name_(name), severity_(severity){}
    ^
1 error generated.

Looks like default constructor is needed

We add patient(){}

and try again:

Lots of errors ...

one is

insertion_sort_..._second_try.cpp:24:37: error: invalid operands to binary expression ('patient' and 'patient')

```
        if (sorted_array[i] >= x){
            ~~~~~~~~~~~~^  ~
```

Operator >= was not understood

```
bool operator< (const patient& rhs) const{ return (severity_ > rhs.severity_); }

bool operator== (const patient& rhs) const{ return (severity_ == rhs.severity_); }

bool operator> (const patient& rhs) const{ return !(((*this) < rhs) || ((*this == rhs))); }

bool operator>= (const patient& rhs) const{ return !((*this) < rhs); }

bool operator<= (const patient& rhs) const{ return !((*this) > rhs); }

bool operator!= (const patient& rhs) const{ return !((*this) == rhs); }
```

Let's try again ...

g++ -o insertion_sort_..._third_try.o insertion_sort_..._third_try.cpp

Again Lots of errors but one is

insertion_sort_..._third_try.cpp:78:22: error: invalid operands to binary expression ('ostream' (aka 'basic_ostream<char>') and 'patient')

```
        cout << sorted_array[i] << " ";
        ~~~~ ^  ~~~~~~~~~~~~~~~
```

Looks like the << operator was not understood

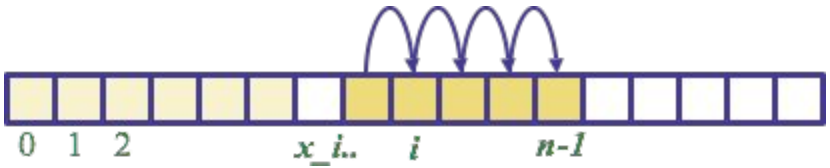
Overriding operator<<

```
friend ostream &operator<<( ostream &output, const patient &p ) {
        output << p.name_;
        return output;
    }
```

And finally, it looks like it compiles and runs.

Using arrays for sorting

- The arrays need compile-time known capacity (max_num_items)
- Shifting items is time consuming



- How is linked list?