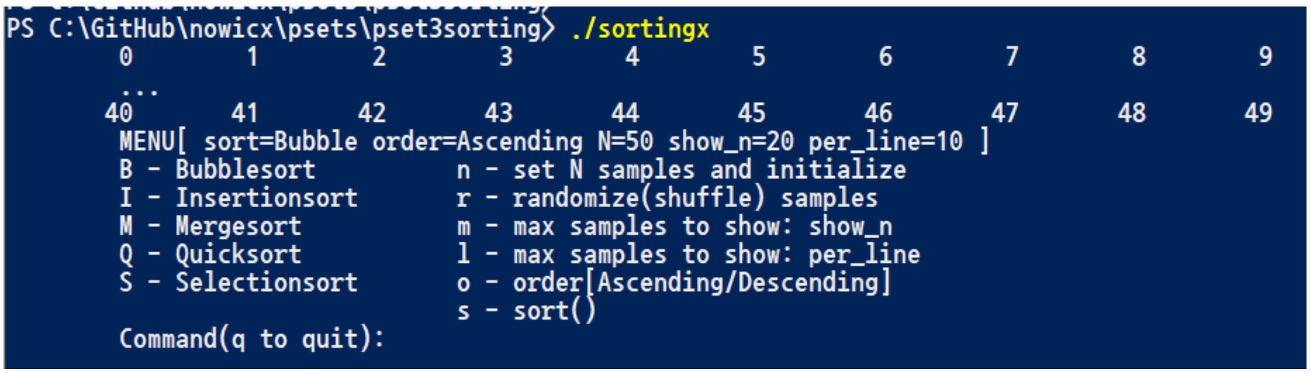
The following materials have been collected from the numerous sources such as Stanford CS106 and Harvard CS50 including my own and my students over the years of teaching and experiences of programming. Please help me to keep this tutorial up-to-date by reporting any issues or questions. Please send any comments or criticisms to idebtor@gmail.com. Your assistances and comments will be appreciated.

PSet – Sorting

Table of Contents

Getting Started

We would like to write a program such that users can test some sorting algorithms interactively as shown below:


```
PS C:\GitHub\nowicx\psets\pset3sorting> ./sortingx
       0        1        2        3        4        5        6        7        8        9
       ...
      40       41       42       43       44       45       46       47       48       49
       MENU[ sort=Bubble order=Ascending N=50 show_n=20 per_line=10 ]
       B - Bubblesort           n - set N samples and initialize
       I - Insertionsort        r - randomize(shuffle) samples
       M - Mergesort            m - max samples to show: show_n
       Q - Quicksort            l - max samples to show: per_line
       S - Selectionsort        o - order[Ascending/Descending]
                                s - sort()
       Command(q to quit):
```

While you follow the instructions step by step, you will eventually complete the specifications of this pset. During implementing the program, we are going to deal with the following subjects.

- Sort algorithms: bubble, insertion, mergesort, quicksort, selection sort
- Shuffle/randomize algorithms
- Using static libraries, header files
- Using function pointers as a first class object

- Using **`rand(), srand(), %, new, delete, nothrow, assert()`**
- Using C++ string stream **`<sstream>`**
- Using STL **`map`** container, **`pair, make_pair`** in STL

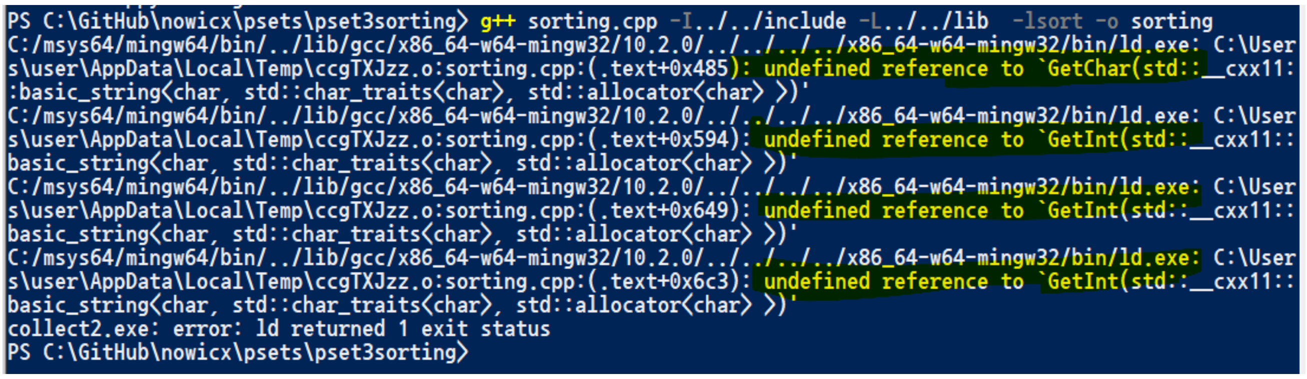
First of all, get **nowic/psets/pset3**, **nowic/include**, **nowic/lib from github** and you may see the following files:

- **pset3.pdf** – this file
- **random.pdf** – reading material about random number generation.
- **sorting.cpp** – Skeleton code that runs various sorting algorithms interactively.
- **sortingx.exe**, **sortingx –** A solution provided to compare it with your work.
- **include/nowic.h** – I/O functions: **`GetInt(), GetChar(), GetString()`** etc.
- **include/sort.h –** sorting function prototypes defined
- **include/rand.h –** random number function prototypes defined
- **lib/libnowic.a**, **libsort.a**, **librand.a** - an user-defined static library for Windows. **lib/libnowic_mac.a**, **libsort_mac.a**, **librand_mac.a** – It is for MacOS.

Using static libraries

Since you studied about the static library and build process, we will use them here. Take a look at both `nowic.h`, `sort.h` and `rand.h` in `nowic/include` folder. For example, you may understand which one to fix if you encounter the following error message during the compilation.

```
$ g++ sorting.cpp -I../../include -L../../lib  -lsort -o sorting             # error
```



```
PS C:\GitHub\nowicx\psets\pset3sorting> g++ sorting.cpp -I../../include -L../../lib  -lsort -o sorting
C:/msys64/mingw64/bin/../lib/gcc/x86_64-w64-mingw32/10.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe: C:\User
s\user\AppData\Local\Temp\ccgTXJzz.o:sorting.cpp:(.text+0x485): undefined reference to `GetChar(std::__cxx11:
:basic_string<char, std::char_traits<char>, std::allocator<char> >)'
C:/msys64/mingw64/bin/../lib/gcc/x86_64-w64-mingw32/10.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe: C:\User
s\user\AppData\Local\Temp\ccgTXJzz.o:sorting.cpp:(.text+0x594): undefined reference to `GetInt(std::__cxx11::
basic_string<char, std::char_traits<char>, std::allocator<char> >)'
C:/msys64/mingw64/bin/../lib/gcc/x86_64-w64-mingw32/10.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe: C:\User
s\user\AppData\Local\Temp\ccgTXJzz.o:sorting.cpp:(.text+0x649): undefined reference to `GetInt(std::__cxx11::
basic_string<char, std::char_traits<char>, std::allocator<char> >)'
C:/msys64/mingw64/bin/../lib/gcc/x86_64-w64-mingw32/10.2.0/../../../../x86_64-w64-mingw32/bin/ld.exe: C:\User
s\user\AppData\Local\Temp\ccgTXJzz.o:sorting.cpp:(.text+0x6c3): undefined reference to `GetInt(std::__cxx11::
basic_string<char, std::char_traits<char>, std::allocator<char> >)'
collect2.exe: error: ld returned 1 exit status
PS C:\GitHub\nowicx\psets\pset3sorting>
```

The error message like
`"...:undefined reference to 'GetChar(std:...) collect2.exe: ld returned 1 exit status`
means that the linking(ld) cannot find `GetChar()` to build the executable.

You know that this kind of `'GetChar()'` I/O function is defined in `libnowic.a.` You check whether or not the library exists, or your command line is correct. In this example, the command line should be:

```
$g++ sorting.cpp -I../../include -L../../lib  -lsort -lnowic -o sorting
```

Using map and function pointer

In this problem set, we are going to use map and function pointers extensively. If you are not familiar with them, please review these topics first and come back here.

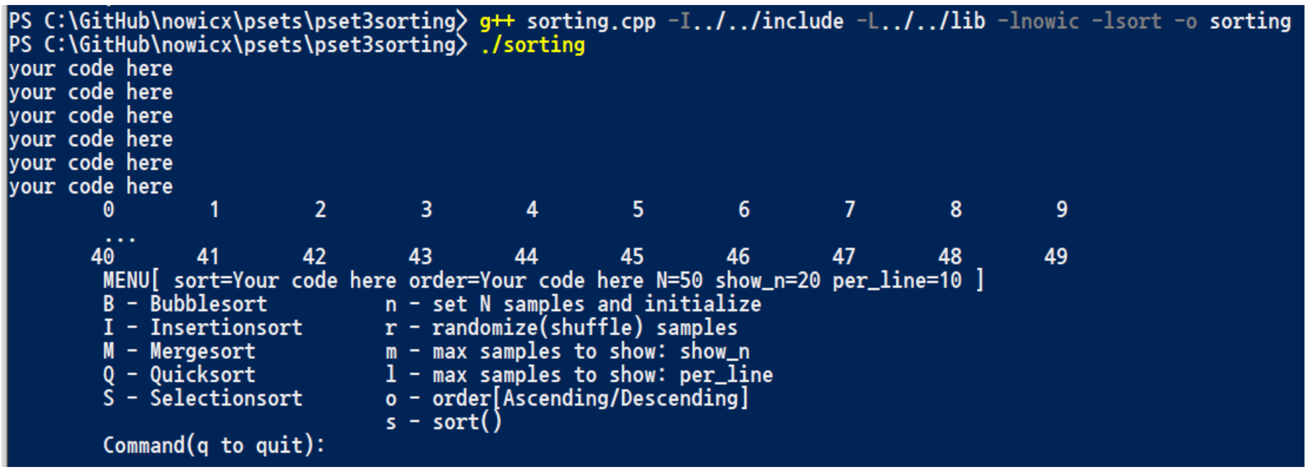
- `map` – an associative container in STL(Standard Template Library)
- `pair, make_pair()` – pair object construct and function in STL
- function pointer, array of function pointers as the first class object

Implementing menu options and status

Firstly, we want to sort an arbitrary number of samples and display its result.

- Follow the skeleton code, **sorting.cpp. You may follow the instructions in this file as well as ones in the skeleton file as you code sorting.cpp.** You can build and run the skeleton code, but its functionality is very limited as shown below:

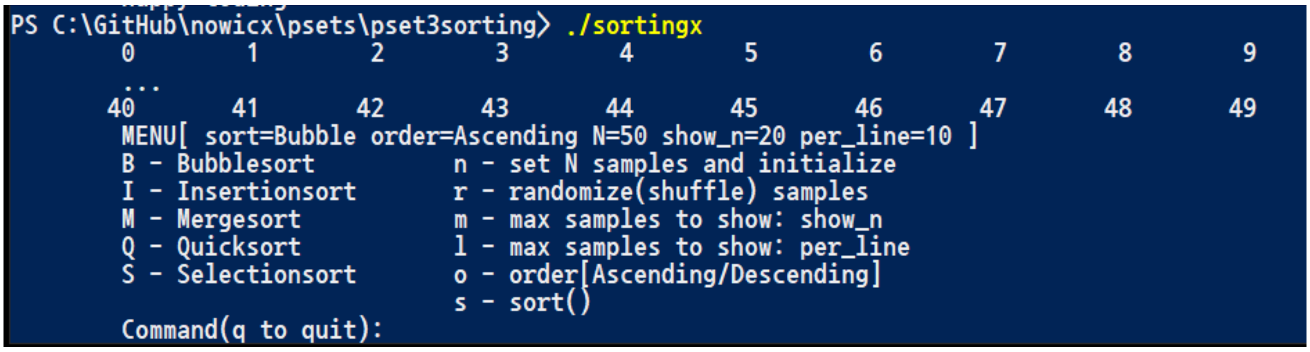
Sample Run: sorting.cpp



```
PS C:\GitHub\nowicx\psets\pset3sorting> g++ sorting.cpp -I../../include -L../../lib -lnowic -lsort -o sorting
PS C:\GitHub\nowicx\psets\pset3sorting> ./sorting
your code here
your code here
your code here
your code here
your code here
your code here
        0        1        2        3        4        5        6        7        8        9
        ...
       40       41       42       43       44       45       46       47       48       49
        MENU[ sort=Your code here order=Your code here N=50 show_n=20 per_line=10 ]
        B - Bubblesort            n - set N samples and initialize
        I - Insertionsort         r - randomize(shuffle) samples
        M - Mergesort             m - max samples to show: show_n
        Q - Quicksort             l - max samples to show: per_line
        S - Selectionsort         o - order[Ascending/Descending]
                                  s - sort()
        Command(q to quit):
```

- Complete the code **in sorting.cpp** that works like **sortingx.exe** provided as a guideline.

Sample Run: sortingx.exe



```
PS C:\GitHub\nowicx\psets\pset3sorting> ./sortingx
        0        1        2        3        4        5        6        7        8        9
        ...
       40       41       42       43       44       45       46       47       48       49
        MENU[ sort=Bubble order=Ascending N=50 show_n=20 per_line=10 ]
        B - Bubblesort            n - set N samples and initialize
        I - Insertionsort         r - randomize(shuffle) samples
        M - Mergesort             m - max samples to show: show_n
        Q - Quicksort             l - max samples to show: per_line
        S - Selectionsort         o - order[Ascending/Descending]
                                  s - sort()
        Command(q to quit):
```

The code shows sorting options such that the user can set them up interactively. The line of the MENU shows the current status of sorting.

- `sort:` algorithm selected

- `N:` the total number of samples to sort
- `order:` the current sort ordering, either ascending or descending
- `show_n:` the maximum samples to display after operation
- `per_line:` the maximum samples to display per line

Step 1: Option B/I/M/Q/S – sort algorithm selection

Provide the sorting algorithms such as Bubble, Insertion, Merge, Quicksort, and Selection sort.

- Define a function pointer to a sorting function called `sort_fp` with `bubblesort` defined in **`sort.h`**.

```
// Define q sort function pointer variable 'sort_fp' variable and initialize it
// with the bubblesort function, 'bubblesort'.

cout << "your code here\n";                              //set currunt sort_fp
```

- When the user selects a new sort algorithm, reset it accordingly as shown below:

```
     case 'M':
        cout << "your code here\n";
        break;
```

Step 2: Option n - set number of samples N and initialize

Use `GetInt()` in `libnowic.a` to get an integer input from the user. If user's input is less than 1, display an error message and go to the menu.
If user's input is valid, do the following.

1. set `N`, the number of samples, with the new value that the user entered.
2. Before allocating the new list, deallocate the old list.
3. Allocate memory for new data samples
4. Fill the list with numbers from 0 to `N - 1` and

Note: Don't use **malloc()** and **free()**, but use **new** and **delete** since we are learning C++ from now on. Before exiting the program, make sure that you **deallocate the memory** that you dynamically allocated during the session.

Step 3: Option r

If the list is newly created or sorted, the list is filled with sorted numbers. Therefore, it is common that the list elements needs to be shuffled before sorting.

Do this coding in two ways as shown in the skeleton code.

```
     case 'r':
        // your code here
        #if 1
        randomize_bruteforce(list, N);
        #else
        randomize(list, N);
        #endif
        break;
```

Method 1: randomize_bruteforce()

For every sample, starting from the first element in the list, it is swapped with the element randomly selected by the index generated by a 'real' (not pseudo) random number out of from 0 to N-1.

We need to generate n distinct random numbers from 0 to n – 1 and store them in an array. Here is a simple way – a brute force algorithm.
For example, let us suppose that we have an int array **a[]** and **n = 100**.

```
void randomize_bruteforce(int list[], int n) {
  Set a[0] = 0, a[1] = 1, … , a[99] = 99.
  For loop from i = 0 to 99:
     Get a random number r.
     Swap two values: a[i] and a[r]
```

Additionally, you may refer to **rand()** and **srand()** functions explained **random.pdf.**

Method 2: randomize()

There is a well-known algorithm for shuffling explained here. It is so-called Fisher-Yates shuffle algorithms. It is define in **nowic/include/rand.h** and **available** from **librand.a** and After you implemented **randomize_bruteforice(),** you also test your code with **randomize()** function.

Step 4: Option m and Option l

When we have a **very long list**, we want to show some in the front part of the list and some in the rear part of the list. The **show_n** specifies the total number of samples to show.

- If **show_n** is less than the total number of samples **N**, elements in the middle of the list may not be shown. Otherwise, all the elements will be shown.
- The **per_line** determines how many samples to show per line.
- Currently, **show_n** and **per_line** are set to 20 and 10 to begin with, respectively.

These options are already implemented through **printlist()** defined in **sort.h.**
You just get an input value and set to show_n and per_line as necessary.

```
void printlist(int *list, int N, int show_n, int per_line);
```

Step 5: Option o

This option toggles the sort ordering function. If the current comparator function pointer, **comp_fp** is for ascending order, set it to descending one, and vice versa.

```
    case 'o': // use comp_fp, ::less, more and a ternary operator
      cout << "o: your code here\n";      // one-line code, use
      break;
```

Step 6: Option s

This option executes the algorithm which is already set previously. Execute sorting using the function pointer **`sort_fp`**. This is just one line of coding.

```
      case 's':
         begin = clock();
         cout << "s: your code here\n";   // one-line code, use sort_fp, comp_fp
         bubblesort(list, N);             // remove this line
         show_timeit(begin);
         break;
```

Step 7: MENU status

To display MENU status properly for "sort" and "order", you use `comp_fp` and `sort_fp` as keys to access and get the mapped string from `comp_map[]` and `sort_map[]`, respectively.

```
   do {
      ...
      stringstream ss;
      ss << "\tMENU[ sort=" << "Your code here" << " order=" << "Your code here";
      ss << " N=" << N << " show_n=" << show_n << " per_line=" << per_line << " ]";
     ...
```

For example, `comp_map` has a comparator function pointer as a key, its description as a value such that the following code prints "`Descending`".

```
cout << comp_map[more];
```

For example, `sort_map` has a sort function pointer as a key, its description as a value such that the following code prints "`Selection`".

```
cout << comp_map[selectionsort];
```

Submitting your solution

- Include the following line at the top of your every source file with your name signed.
  On my honor, I pledge that I have neither received nor provided improper assistance in the completion of this assignment.
  Signed: ______________ **Student Number:** __________________
- Make sure your code **compiles** and **runs** right before you submit it. Every semester, we get dozens of submissions that don't even compile. Don't make "a tiny last-minute change" and assume your code still compiles. You will not get sympathy for code that "almost" works.
- If you only manage to work out the Project problem partially before the deadline, you still need to turn it in. However, don't turn it in if it does not compile and run.
- Place your source files in the folder you and I are sharing.
- After submitting, if you realize one of your programs is flawed, you may fix it and submit again as long as it is **before the deadline**. You will have to resubmit any related files together, even if you only change one. You may submit as often as you like. **Only the last version** you submit before the deadline will be graded.

Files to submit, Due and Grade points

Files to submit: upload the following files in a piazza folder

- sorting.cpp – 2 points

Due: 11:55 pm