C++ for Coders and Data Structures
Lecture Notes by idebtor@gmail.com, Handong Global University

Lab: Build & Use Static Library

A library is a collection of pre-compiled object files that can be linked into your programs via the linker. Examples are the system functions such as printf() and sqrt(). There are two types of external libraries: *static library* and *shared library*.

- **A static library** has file extension of **".a"** (archive file) in Unixes or **".lib"** (library) in Windows. When your program is linked against a static library, the machine code of external functions used in your program is copied into the executable. A static library can be created via the archive program `ar.exe`.
- **A shared library** has file extension of **".so"** (shared objects) in Unixes or **".dll"** (dynamic link library) in Windows. When your program is linked against a shared library, only a small table is created in the executable. Before the executable starts running, the operating system loads the machine code needed for the external functions - a process known as dynamic linking. Dynamic linking makes executable files smaller and saves disk space, because one copy of a library can be shared between multiple programs. Furthermore, most operating systems allows one copy of a shared library in memory to be used by all running programs, thus, saving memory. The shared library codes can be upgraded without the need to recompile your program.

Getting Started

We want to build static libraries **`libsort.a,`** **`librand.a`** and **`libnowic.a`** for Windows and **`libsort_mac.a,`** **`librand_mac.a`** and **`libnowic_mac.a`** for macOS. They contain several sorting functions and input/output functions, respectively.
Files provided:

- `staticlib.pdf` – this file
- `sortDriver.cpp` – invokes functions defined in `~.a` for testing
- `include/sort.h` - Do not change this file.
  `include/nowic.h` - Do not change this file.
- `src/rand.cpp` – random functions source file
- `src/nowic.cpp` – nowic source file
- `nowicDriver.cpp` – invokes functions defined in `~.a` for testing

Step 2. Prepare source files for libsort.a

1. Copy all the sort function files (`bubblesort.cpp ~ selectionsort.cpp`) you have implemented into this folder.
2. Make sure those function prototypes matches ones in sort.h as shown below:

```
// declare the sort function prototypes
```

```
void bubblesort(int *list, int n, bool (*comp)(int, int) = ::less);
void insertionsort(int *list, int n, bool (*comp)(int, int) = ::less);
void mergesort(int *a, int N, bool (*comp)(int, int) = ::less);
void quicksort(int *a, int n, bool (*comp)(int, int) = ::less);
void selectionsort(int *list, int n, bool (*comp)(int, int) = ::less);
```

3. Turn off `#if 0` and `#endif` in sort function files that defined `main()`, `less()` and `more()` such that it prevents from defining those function more than once.

```
#if 0
// two comparator functions
// The value returned indicates whether the element passed as first argument
// is considered to go before the second in the specific ordering.
// more() and less() are equivalent to greater<int>() and less<int>() in STL.
bool more(int x, int y) { return x > y; }   // for descending order
bool less(int x, int y) { return x < y; }   // for ascending order
...
...
void main()
...
#endif
```

4. Now, create `comparator.cpp` that implements two comparator functions, `less()` and `more()`. Two comparator prototypes are already declared in `sort.h`.

```
// file: comparator.cpp
// two comparator functions
// The value returned indicates whether the element passed as first argument
// is considered to go before the second in the specific ordering.
// more() and less() are equivalent to greater<int>() and less<int>() in STL.
bool more(int x, int y) { return x > y; }   // for descending order
bool less(int x, int y) { return x < y; }   // for ascending order
```

Step 3. Building static libraries

Build the following static libraries or macOS equivalents:

- `libsort.a` - Copy source files from the previous lab, modify, create as needed
- `libnowic.a` – Use `nowic.cpp` provided
- `librand.a` – Use `rand.cpp` provided

`libsort.a` is supposed to contain the following sort functions.
(Refer to `include/sort.h` for detail. If it does not exist, create one.)

```
// comparator function prototypes
bool less(int *list, int i, int j);     // ascending order
bool more(int *list, int i, int j);     // descending order

// sort function prototypes
void bubblesort(int *list, int n, bool (*comp)(int, int) = ::less);
void insertionsort(int *list, int n, bool (*comp)(int, int) = ::less);
void mergesort(int *a, int N, bool (*comp)(int, int) = ::less);
void quicksort(int *a, int n, bool (*comp)(int, int) = ::less);
void selectionsort(int *list, int n, bool (*comp)(int, int) = ::less);

void printlist(int *list, int n, int show_n = 20, int per_line = 10);
```

`libnowic.a` is supposed to contain the following I/O functions.
(Refer to `include/nowic.h` for detail. If it does not exist, create one.)

```
int GetInt(string prompt = "Enter an integer:");
float GetFloat(string prompt = "Enter a floating point number: ");
double GetDouble(string prompt = "Enter a floating point number: ");
char GetChar(string prompt = "Enter a character: ");
string GetString(string prompt = "Enter a string: ");
```

We use **ar** comand to maintain **archive** libraries in C/C++. The archive library is a collection of files, typically object files. Using **ar**, you can create a new library, add, delete, extract members from an existing library, and print a table of contents.

Example:
Create a static library, **`libsort.a`**, that contains **`bubblesort() ~ selectionsort(), less(), more() and printlist()`** functions which are defined in each files(~~.cpp), respectively.
Let's suppose that you have several source files (**`bubble.cpp, insertion.cpp, merge.cpp, quick.cpp, selection.cpp, comparator.cpp, printlist.cpp`**) to turn it into a static library (**`nowic/lib/libsort.a`**).

```
> g++ -c comparator.cpp
> g++ -c bubble.cpp
> g++ -c merge.cpp

> ar rcs libsort.a comparator.o bubble.o merge.o    # use libsort_mac.a for macOS
> ar                                                # list all the options available
> ar t libsort.a                                    # list ~.o files archived
> ar x libsort.a insertion.o                        # extract ~.o files archived
> ar d libsort.a insertion.o                        # delete ~.o files archived

> cp libsort.a ../../lib                            # copy it in nowic/lib folder
```

```
ar flags:
   r: replace or insert files into archive (with replacement).
   c: create an archive file
   d: delete files in archive file
   s: regenerate the external symbol table
   t: display contents of archive
```

NOTE: It is important that you recognize that the GCC compiler requires that you prefix your static library with the keyword **lib** and suffix **.a**, like **lib**sort**.a**. The lib prefix is required by the linker to find the static library.

Step 3: Searching for Header Files and Libraries (`-I`, `-L` and `-l`)

It is very critical that you understand the meaning of each option flags and search mechanism of files by gcc compiler.
When compiling the program, the compiler needs the header files to compile the source codes; the linker needs the libraries to resolve external references from other

object files or libraries. The compiler and linker will not find the headers/libraries unless you set the appropriate options, which is not obvious for first-time user.

For each of the headers used in your source (via #include directives), the compiler searches the so-called include-paths for these headers. The include-paths are specified via `` `-Ipath` `` option (uppercase `` `I` `` followed by the path). Since the header's filename is known (e.g., `` `iostream.h` ``, `` `stdio.h` ``), the compiler only needs the directories.
The linker searches the so-called library-paths for libraries needed to link the program into an executable. The library-path is specified via `` `-Lpath` `` option (uppercase `` `L` `` followed by the path). In addition, you also have to specify the library name. In Unixes, the library `` `libxxx.a` `` is specified via `` `-lxxx` `` option (lowercase letter `` `l` ``, without the prefix `` `lib` `` and `` `.a` `` extension). In Windows, provide the full name such as `` `-lxxx.lib` ``. The linker needs to know both the directories as well as the library names. Hence, two options need to be specified.

Step 4. Using a static library

Recall the following flags:

```
g++ flags:
-I: Specifies the folder name where the header files (~.h) exist.
-L: Specifies the folder name where the referenced library exists.
-l: Specifies the library name you want to attach (without 'lib' in its name)
  For example, use -lnowic if its file name is libnowic.a
```

Example 1. Build an executable, sort.exe, with sortDriver.cpp and libsort.a
If you are present at ~nowic/labs/lab8 and you have **not** moved **libsort.a** into nowic/lib folder yet, then you can create and run an executable 'sort.exe' as shown below: (Use `-lsort_mac` instead of `-lsort` on macOS)

```
> g++ sortDriver.cpp  -I../../include -L./  -lsort  -o sortDriver
> ./sortDriver
```

If you are present at ~nowic/labs/lab8 and you have copied or moved **libsort.a** into nowic/lib folder, then you can create an executable 'sort.exe' as shown below:

```
> g++ sortDriver.cpp  -I../../include -L../../lib -lsort  -o sortDriver
> ./sortDriver
```

If you are additionally using some functions in **nowic.a** which exists in ~nowic/lib, you may add it at the end as shown below

```
> g++ sortDriver.cpp  -I../../include -L../../lib -lsort -lnowic -o sortDriver
```

Example 2. Build an executable, nowic.exe, with nowicDriver.cpp and libnowic.a
If you are present at ~nowic/labs/lab8 and libnowic.a exists in nowic/lib folder, then you can create an executable 'nowic.exe' as shown below:

```
> g++ nowicDriver.cpp -I../../include -L../../lib -lnowic -o nowicDriver
> ./nowicDriver
```

Files to submit

Use `sortDriver.cpp` and `nowicDriver.cpp` to build the following executables, respectively. You must use the static libraries you built in this lab.

- `sortDriver.exe`
- `nowicDriver.exe`
- `libnowic.a  or libnowic_mac.a`
- `libsort.a or libsort_mac.a`
- `librand.a or librand_mac.a`

Due and Grade

- Due: 11:55 pm
- Grade: 2 points

One thing I know, I was blind but now I see. John 9:25