

# Software libraries: cxxopts

Victor Eijkhout, Susan Lindsey

Fall 2023

last formatted: September 19, 2023

# 1. Don't reinvent the wheel: use a library

Many things you want to program have been thought of before:  
see if there is a library for it.

Library: 'program without main':  
you supply the main, functionality comes from library

## 2. External libraries: usage

Suppose the 'fancy' library does what you need.

1. Include a header file
2. Then use the functions defined there.

```
#include "fancylib.h"
```

```
int main() {  
    x = fancyfunction(y);  
}
```

### 3. External libraries: compile

1. Compiler needs to know where the header is:

```
icpc -c yourprogram.cpp -I/usr/include/fancylib
```

2. You may need to link a library file:

```
icpc -o yourprogram yourprogram.o \  
-L/usr/lib/fancylib -lfancy
```

(not for 'header only' libraries)

## 4. Where to find libraries

Search ...There is a lot of stuff on github.

# Commandline arguments

## 5. Traditional commandline parsing

Use:

```
int main( int argc, char **argv ) { // stuff };
```

then

Code:

```
1 // args/argcv.cpp
2 cout << "Program name: "
3     << argv[0] << '\n';
4 for (int iarg=1; iarg<argc;
5     ++iarg)
6     cout << "arg: " << iarg
7         << argv[iarg] << " => "
8         << atoi( argv[iarg] ) <<
9         '\n';
```

Output:

```
./argcv 5 12
Program name: ./argcv
arg 1: 5 => 5
arg 2: 12 => 12
./argcv abc 3.14 foo
Program name: ./argcv
arg 1: abc => 0
arg 2: 3.14 => 3
arg 3: foo => 0
```

Major headache dealing with general arguments:

## 6. Example: cxxopts

`https://github.com/jarro2783/cxxopts`

Find the 2.2.1 release or newer.

Use `wget` or `curl` to download straight to the class machine.

`wget https://github.com/jarro2783/cxxopts/archive/refs/tags`

Unpack it:

`tar fxv v3.0.0.tar.gz`



## 7. Cmake based installation

The cxxopts-2.2.1 directory has a file CMakeLists.txt

```
cd cxxopts-2.2.1
mkdir build
cd build
cmake -D CMAKE_INSTALL_PREFIX:PATH=${HOME}/mylibs/cxxopts \
..
make
make install
```

(This is an 'in-source' build. I don't like it: prefer to have the build directory elsewhere to keep the source untouched.)

## 8. Let's use this library

```
#include "cxxopts.hpp"

// in the main program:
cxxopts::Options options
("cxxopts",
 "Commandline options demo");
```

Compile

```
icpc -o program source.cpp \
    -I/path/to/cxxopts/installdir/include
```

Can you compile and run this?

## 9. Help option

You want your program to document its own usage:

```
options.add_options()
    ("h,help","usage information")
    ;
    /* ... */
auto result = options.parse(argc, argv);
if (result.count("help")>0) {
    cout << options.help() << '\n';
    return 0;
}
```

Use:

```
./myprogram -h
```

## 10. Numerical options

```
// define '-n 567' option:
options.add_options()
  ("n,ntimes","number of times",
   cxxopts::value<int>()
   ->default_value("37")
  )
;
/* ... */
// read out '-n' option and use:
auto number_of_times = result["ntimes"].as<int>();
cout << "Using number of times: " << number_of_times << '\n';
```

# 11. Array options

```
//define '-a 1,2,5,7' option:
options.add_options()
    ("a,array","array of values",
     cxxopts::value< vector<int> >()->default_value("1,2,3")
    )
;
/* ... */
auto array = result["array"].as<vector<int>>();
cout << "Array: " ;
for ( auto a : array ) cout << a << ", ";
cout << '\n';
```

## 12. Positional arguments

```
// define `positional argument' option:
options.add_options()
    ("keyword","whatever keyword",
     cxxopts::value<string>())
    ;
options.parse_positional({"keyword"});
/* ... */
// read out keyword option and use:
auto keyword = result["keyword"].as<string>();
cout << "Found keyword: " << keyword << '\n';
```

## 13. Put it all to the test

Now make your program do something with the inputs:

```
./myprogram -n 10 whatever
```