# Software libraries: cxxopts

Victor Eijkhout, Susan Lindsey

Fall 2025

last formatted: October 14, 2025

# 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.

```
1 #include "fancylib.h"
2
3 int main() {
4   x = fancyfunction(y);
5 }
```

# 3. External libraries: compile

1. Compiler needs to know where the header is:

```
1 icpx -c yourprogram.cpp -I/usr/include/fancylib
```

2. You may need to link a library file:

```
1 icpx -o yourprogram yourprogram.o \
2     -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:

```
1     int main( int argc,char **argv ) { // stuff };
2
```

then

**Code:**

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

**Output:**

```
1 ./argcv 5 12
2 Program name: ./argcv
3 arg 1: 5 => 5
4 arg 2: 12 => 12
5 ./argcv abc 3.14 foo
6 Program name: ./argcv
7 arg 1: abc => 0
8 arg 2: 3.14 => 3
9 arg 3: foo => 0
```

Major headache dealing with general arguments:

```
1 myprog --debug -f "input.txt" -n 10 20 40
```

# 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/v3.0.0.tar.gz`*

Unpack it:

*`tar fxv v3.0.0.tar.gz`*

# 7. Cmake based installation

The `cxxopts-2.2.1` directory has a file `CMakeLists.txt`

```
1 cd cxxopts-2.2.1
2 mkdir build
3 cd build
4 cmake -D CMAKE_INSTALL_PREFIX:PATH=${HOME}/mylibs/cxxopts \
5     ..
6 make
7 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"
```

```
1 // args/cxxopts.cpp
2 // in the main program:
3 cxxopts::Options options
4   ("cxxopts",
5    "Commandline options demo");
```

Compile

```
1 icpx -o program source.cpp \
2    -I/path/to/cxxopts/installdir/include
```

Can you compile and run this?

# 9. Compilation with cmake

```
1 find_package( PkgConfig REQUIRED )
2 pkg_check_modules( OPTS REQUIRED cxxopts )
3 message( STATUS "Cxxopts includes: ${OPTS_INCLUDE_DIRS}" )
4 target_include_directories(
5     myprogram PUBLIC
6     ${OPTS_INCLUDE_DIRS}
7     )
```

# 10. Help option

You want your program to document its own usage:

```
1 // args/cxxopts.cpp
2 options.add_options()
3   ("h,help","usage information")
4   ;
5   /* ... */
6 auto result = options.parse(argc, argv);
7 if (result.count("help")>0) {
8   cout << options.help() << '\n';
9   return 0;
10 }
```

Use:

```
./myprogram -h
```

# 11. Numerical options

```
1  // args/cxxopts.cpp
2  // define '—n 567' option:
3  options.add_options()
4    ("n,ntimes","number of times",
5     cxxopts::value<int>()
6     ->default_value("37")
7     )
8    ;
9    /* ... */
10 // read out '—n' option and use:
11 auto number_of_times = result["ntimes"].as<int>();
12 cout << "Using number of times: " << number_of_times << '\n';
```

# 12. Array options

```
1 // args/cxxopts.cpp
2 //define '—a 1,2,5,7' option:
3 options.add_options()
4   ("a,array","array of values",
5    cxxopts::value< vector<int> >()->default_value("1,2,3")
6    )
7   ;
8   /* ... */
9 auto array = result["array"].as<vector<int>>();
10 cout << "Array: " ;
11 for ( auto a : array ) cout << a << ", ";
12 cout << '\n';
```

# 13. Positional arguments

```
1 // args/cxxopts.cpp
2 // define 'positional argument' option:
3 options.add_options()
4   ("keyword","whatever keyword",
5    cxxopts::value<string>())
6   ;
7 options.parse_positional({"keyword"});
8   /* ... */
9 // read out keyword option and use:
10 auto keyword = result["keyword"].as<string>();
11 cout << "Found keyword: " << keyword << '\n';
```

# 14. Put it all to the test

Now make your program do something with the inputs:

`./myprogram -n 10 whatever`