9

1

C++ Exercises
Set 8-64

64

Author(s): Olivier Gelling, Leon Lan, Mohammad Habibi
Previously rated by Frank
13:33

October 30, 2025

64

Exercise 64:

This exercise was a whole lot of work! We have a mostly functional version of Arg going, but it has a few (at times glaring) flaws:

We can now use long variants of a short option to set a value in both of their arrays. Check. We cannot do the opposite of setting the value of a long option using its short counterpart.

When we hand option() a string pointer that already points to an earlier assigned string, it is not properly clearing this value. We are not sure why this is. We have added an extra checks to our Arg::option()functions to clear, value, but it seems something is going wrong enough for the ArgOption and ArgLongOption size() functions to not be able to properly distinguish these cases, as those functions *should* already be able to clear the pointed at string in the case no return value is present.

Something else that is worth criticism is the large size of our second arg constructor. It makes sense for this member to be comprehensive, but it can definitely do with some further compactifying! We will leave it as is for now.

An example is the following output

```
64$ tmp/bin/binary — filenames files
f 1: files
filenames 1: files

64$ tmp/bin/binary  f files
f 1: files
filenames 0:
```

We see that calling filenames triggers both option() calls, while calling f leaves the long option's value blank. Before adding our somewhat crude value check it would also even show:
filenames 0: files
which is of course not right.

We will also add a small comment to the graders: I (Olivier) took a look at Frank's Tom Poes eh I mean Bobcat library, and have made a decent bit of use of copilot for this exercise, especially to get started and see some structure arising from the chaos. Every line of code was, of course, written by me, and the main fix I have been working on these last few days was not inspired by peeking at bobcat's Arg__, although I will say it helped to see my solution mostly verified! As such we (well, I, I am not sure my teammates would like me to be as overly honest as I tend to be!) understand entirely if a penalty is in order. I would already be happy with feedback and a working class!

Listing 1: arg/arg.h

```
#ifndef INCLUDED_ARG_
#define INCLUDED_ARG_

// include support classes

#include <string>
#include <cstddef> // for size_t
```

SF once a std header file is included

```
// forward declaring to reduce dependencies
class ArgOption;
class ArgLongOption;
class OptStructArray;

class Arg
{
    static Arg *s_instance;
```

this leaks. How to solve it?

```
    ArgOption *d_option;
    ArgLongOption *d_longOption;
    OptStructArray *d_optStructArray;
```

pointers are OK, but also OC: let the objects do what they need to do, and avoid needless allocations.

```
    std::string d_basename; // = ""; // ?
```

SF, already empty

```
    int d_argc;
    char **d_argv;
    size_t d_nArgs;

    public:
        enum Type
        {
            None,
            Required,
            Optional
        };

        // Nested class LongOption
        class LongOption
        {
            std::string d_name;
            Type d_type;
            int d_optionChar;

            public:
                LongOption(char const *name, Type type = None);
                LongOption(char const *name, int optionChar);

                // Accessors
                std::string const &name() const;
                Type type() const;
                int optionChar() const;
        };

        // Arg's own member functions:

        static Arg &initialise(char const *optstring, int argc, char **argv);
        static Arg &initialise
        (
            char const *optstring,
            LongOption const *const begin,
            LongOption const *const end,
            int argc, char **argv
        );

        Arg &instance();

        char const *arg(unsigned idx) const;
        std::string const &basename() const;
        size_t nArgs() const;
        size_t nOptions() const;
        size_t option(int opt) const;
        size_t option(std::string const &options) const;
        size_t option(std::string *value, int option) const;
        size_t option(std::string *value, char const *longOption) const;
    private:
        // private constructors since singleton
        Arg(char const *optstring, int argc, char **argv);
        Arg(char const *optstring, LongOption const *begin,
                                   LongOption const *end, int argc, char **argv);

        Arg() = delete;
        Arg(Arg const &) = delete;
        Arg &operator=(Arg const &other) = delete; // Probably already done

        static std::string setBaseName(char *argv0);
        static std::string makeOptStr(char const *optstring);
        static void buildLongOptArray
        (
            std::string const &optstring,
            LongOption const *begin,
            LongOption const *end,
            struct option *options
        );
        static int setArgType(Arg::LongOption thisOption,
                                                    std::string const &optstr);
};
```

implied

there are no public constructors

correct. implied by the deleted CC.

put this in the public section

flaw: No destructor

```
// inline accessors:
inline size_t Arg::nArgs() const
{
    return d_nArgs;
}
inline std::string const &Arg::basename() const
{
    return d_basename;
}
//

inline std::string const &Arg::LongOption::name() const
{
    return d_name;
}
inline Arg::Type Arg::LongOption::type() const
{
    return d_type;
}
inline int Arg::LongOption::optionChar() const
{
    return d_optionChar;
}



#endif
```

Listing 2: arg/arg.ih

```
#include "arg.h"
#include "../argoption/argoption.h"
#include "../arglongoption/arglongoption.h"
#include "../optstructarray/optstructarray.h"

#include <iostream>
#include <unistd.h>
#include <getopt.h>
#include <libgen.h>
#include <cstring>

using namespace std;
```

Listing 3: arg/data.cc

```
#include "arg.ih"
```

```
    // by

Arg *Arg::s_instance = nullptr;
```

Listing 4: arg/longoption1.cc

```
#include "arg.ih"

    // by

Arg::LongOption::LongOption(char const *name, Type type)
:
    d_name(name),
    d_type(type),
    d_optionChar(0)
{}
```

Listing 5: arg/longoption2.cc

```
#include "arg.ih"

    // by

Arg::LongOption::LongOption(char const *name, int optionChar)
:
    d_name(name),
    d_type(None),
    d_optionChar(optionChar)
{}
```

N

Listing 6: arg/initialise1.cc

S

C

```
#include "arg.ih"

    // by

Arg &Arg::initialise(char const *optstring, int argc, char **argv)
{
    if (s_instance)
    {
        cerr << "initialise called repeatedly\n";
        exit(1);
    }

    s_instance = new Arg(optstring, argc, argv);

    return *s_instance;
}
```

Listing 7: arg/initialise2.cc

```
#include "arg.ih"

    // by

Arg &Arg::initialise
(
    char const *optstring,
    LongOption const *const begin,
    LongOption const *const end,
    int argc, char **argv
)
{
    if (s_instance)
    {
        cerr << "initialise called repeatedly\n";
        exit(1);
    }
    s_instance = new Arg(optstring, begin, end, argc, argv);

    return *s_instance;
}
```

Listing 8: arg/instance.cc

```
#include "arg.ih"

    // by

Arg &Arg::instance()
{
    if (!s_instance)
    {
        cerr << "Not initialised.\n";
        exit(1);
    }

    return *s_instance;
}
```

Listing 9: arg/argidx.cc

```
#include "arg.ih"

    // by

char const *Arg::arg(unsigned idx) const
{
    return (idx >= d_nArgs ? nullptr : d_argv[optind + idx]);
}
```

Listing 10: arg/setbasename.cc

```
#include "arg.ih"

    // by

string Arg::setBaseName(char *argv0)
{
    // set basename using a copy of argv[0]
    char progName[strlen(argv0) + 1];
    strcpy(progName, argv0);
    return string(basename(progName));
}
```

* The basename is the name of the program called without any path specifications.

*- this implementation is too C-ish

Listing 11: arg/noptions.cc

```
#include "arg.ih"

    // by

size_t Arg::nOptions() const
{
    return d_option ? d_option->size() : 0;
}
```

Listing 12: arg/option1.cc

```
#include "arg.ih"

    // by

size_t Arg::option(int opt) const
{
    return d_option ? d_option->size(opt) : 0;
}
```

6

Listing 13: arg/option2.cc

```
#include "arg.ih"

    // by

size_t Arg::option(string const &options) const
{
    size_t count = 0;
    for (size_t index = 0; index != options.size(); ++index)
        count += option(options[index]);
    return count;
}
```

Listing 14: arg/option3.cc

N

S

C

```
#include "arg.ih"

    // by

size_t Arg::option(string *value, int option) const
{
    size_t temp = d_option ? d_option->size(value, option) : 0;

    if (not temp)
        *value = "";

    return temp;
}
```

Listing 15: arg/option4.cc

```
#include "arg.ih"

    // by

size_t Arg::option(string *value, char const *longOption) const
{
    size_t temp = d_longOption ? d_longOption->size(value, longOption) : 0;

    if (not temp)
        *value = "";

    return temp;
}
```

Listing 16: arg/arg1.cc

```
#include "arg.ih"

    // by

Arg::Arg(char const *optstring, int argc, char **argv)
:
    d_basename(setBaseName(argv[0])),
    d_argc(argc),
    d_argv(argv)
{
    string optstr = makeOptStr(optstring);
                                    // adds ":" to start of string
    d_option = new ArgOption();
    opterr = 0;

    int opt;
    //int old_optind = optind;
    while ((opt = getopt(argc, argv, optstr.c_str())) != -1)
    {
        if (opt == '?' || opt == ':')
            continue;
        d_option->add(opt);
```

this produces another memory leak

```
    }

    d_nArgs = argc - optind;
}
```

Listing 17: arg/arg2.cc

```
#include "arg.ih"

    // by

Arg::Arg
(
    char const *optstring,
    LongOption const *begin,
    LongOption const *end,
    int argc, char **argv
)
:
    d_basename(setBaseName(argv[0])),
    d_argc(argc),
    d_argv(argv)
{
    string optstr = makeOptStr(optstring);
                                    // adds : to start of option string
    d_option = new ArgOption();
    d_longOption = new ArgLongOption();

    size_t nLongOpts = end - begin;
    d_optStructArray = new OptStructArray(nLongOpts + 1);
                                            // we build the struct and make a
                                            // pointer to it for ease of use
    struct option *options = d_optStructArray->get();
    buildLongOptArray(optstr, begin, end, options);

    opterr = 0;
    int opt;
    int longIdx = -1;
    while ((opt = getopt_long(
                    argc, argv, optstr.c_str(), options, &longIdx)) != -1)
    {
        switch (opt)
        {
            case '?':
            case ':':
            continue;
            case 0:                         // exclusively long
                d_longOption->add(options[longIdx].name);
            break;
            default:
                d_option->add(opt);         // adding short
                if (longIdx != -1 && options[longIdx].val)// == opt)
                    d_longOption->add(options[longIdx].name);
            break;                          // long with short counterpart
        }       // only triggers for long. Should also work for counterpart
    }
    d_nArgs = argc - optind;
}
```

Leaks

Listing 18: arg/setbasename.cc

= listing 10 ?

```
#include "arg.ih"

    // by

string Arg::setBaseName(char *argv0)
{
    // set basename using a copy of argv[0]
    char progName[strlen(argv0) + 1];
    strcpy(progName, argv0);
    return string(basename(progName));
}
```

Listing 19: arg/makeoptstr.cc

? why

JJSC

```
#include "arg.ih"

    // by

string Arg::makeOptStr(char const *optstring)
{
    string optstr = (optstring[0] == ':') ?
                optstring : string(":") + optstring;
    return optstr;
}
```

Listing 20: arg/buildlongoptarray.cc

```
#include "arg.ih"

    // by

void Arg::buildLongOptArray
(
    string const &optstring,
    LongOption const *begin,
    LongOption const *end,
    struct option *options
)
{
    size_t nLongOpts = end - begin;
    for (size_t index = 0; index != nLongOpts; ++index)
    {
        options[index].name = begin[index].name().c_str();
        options[index].has_arg = setArgType(begin[index], optstring);
        options[index].val = begin[index].optionChar() ?
                                            begin[index].optionChar() : 0;
    }
}
```

Listing 21: arg/setargtype.cc

```
#include "arg.ih"

    // by

int Arg::setArgType(Arg::LongOption thisOption, string const &optstr)
{
    size_t optStrIndex = optstr.find_first_of(thisOption.optionChar());

    if (optStrIndex != string::npos)
        return (optstr[optStrIndex + 1] != ':' ? None :
            (optstr[optStrIndex + 2] != ':' ? Required
                : Optional));

    return thisOption.type();
}
```

Listing 22: main.ih

```
#include "arg/arg.h"
#include <iostream>

using namespace std;
```

Listing 23: main.cc

```
#include "main.ih"
```

```
namespace
{
    Arg::LongOption longOptions[] =
    {
        Arg::LongOption{"debug", Arg::Required},
        Arg::LongOption{"filenames", 'f'},
        Arg::LongOption{"help", 'h'},
        Arg::LongOption{"version", 'v'},
        Arg::LongOption{"only", Arg::Required},
        Arg::LongOption{"long", Arg::Required},
    };
    auto longEnd = longOptions + std::size(longOptions);
}

~~int~~ main(int argc, char **argv)
try
{
    //Arg &arg = Arg::initialise("f:h:v:qx:y:", argc, argv);
    Arg &arg = Arg::initialise("f:h:v:qx:y:",
                    longOptions, longEnd, argc, argv);
    // code using arg, etc.
    string *value = new string;

    cerr << "x " << arg.option(value, 'x') << ": " << *value << '\n';
    cerr << "y " << arg.option(value, 'y') << ": " << *value << '\n';
    cerr << "q " << arg.option(value, 'q') << ": " << *value << '\n';
    cerr << "only " << arg.option(value, "only") << ": "
        << *value << '\n';
    cerr << "f " << arg.option(value, 'f') << ": " << *value << '\n'
        << "filenames " << arg.option(value, "filenames") << ": "
        << *value << '\n';
    cerr << "v " << arg.option(value, 'v') << ": " << *value << '\n'
        << "version " << arg.option(value, "version") << ": "
        << *value << '\n';

}
catch (...)
{}
```

help and version don't need arguments: NC !!

Leaks

CTR

you should have removed this: NC...

don't pass a pointer, but pass the address of a local string to avoid memory leaks.