Разбор командной строки

Практикум, 3 курс

Рассказывает:

Подымов Владислав Васильевич

Осень 2016

С чем имеем дело

```
terminal> ./prog -iou --time=forever --input some-file
```

С тем, как в исходном файле C/C++ разбирать такого вида аргументы командной строки

(*вычленять из неё **опции***)

Варианты:

- Делать это вручную с ноля (*изобретать велосипед*)

- Использовать что-нибудь готовое
  - getopt
  - Boost.Program_options
  - Или, может быть, знаете что-нибудь ещё?

В любом случае всё сводится к тому, чтобы сказать что-нибудь полезное про аргументы командной строки, имея вот это:

```
int main(int argc, char ** argv) ...
```

Что такое “опция”

Существует несколько стандартов определения “опции”,
но здесь попробуем описать это понятие содержательно, как оно обычно встречается:

“-x” – это (*короткая*) опция “x” (*“-W” лучше не писать – это зарезервированная опция*)

“-xarg” - это либо опция “x” с аргументом “arg”, либо четыре опции: “x”, “a”, “r”, “g”

“-x arg” - это либо опция “x” с аргументом “arg”,
либо опция “x” без агрумента, за которой следует “arg”,
не относящийся к опциям

“--opt” - это длинная опция “opt”

“--opt=val” - это длинная опция “opt” с аргументом “val”

“--opt val” - это либо длинная опция “opt” с аргументом “val”,
либо длинная опция “opt” без аргумента, за которой следует “val”,
не относящееся к опциям

Строка “-” (*без дополнительных символов*) никак не относится к опциям

“--” - это строка, обозначающая безусловный конец разбора опций

Есть аргумент у опции или нет, решается программой

getopt: общее описание

man7.org/linux/man-pages/man3/getopt.3.html

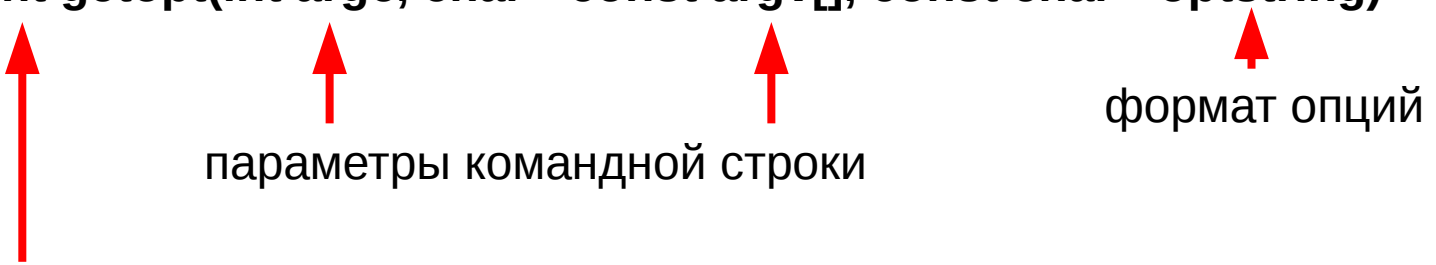
Когда говорят о getopt, имеют в виду три функции
и несколько внешних глобальных переменных:

Определение	Header	Пояснение
int getopt(int argc, char * const argv[], const char * optstring)	<unistd.h>	Разбирает одну за одной **короткие** опции согласно заданному формату и возвращает их
char * optarg	<unistd.h> <getopt.h>	Сюда записывается аргумент разобранной опции
int optopt	<unistd.h> <getopt.h>	В некоторых случаях сюда записывается короткая опция
int getopt_long(int argc, char * const argv[], const char * optstring, const struct option * longopts, int * longindex)	<getopt.h>	Как getopt, но способна разбирать и **длинные** опции
int getopt_long_only(int argc, char * const argv[], const char * optstring, const struct option * longopts, int * longindex)	<getopt.h>	Как getopt_long, но разбирает “-opt” как длинную опцию “opt”, если такая опция определена

(*есть ещё несколько глобальных переменных – они встретятся позже*)

Функция getopt

int getopt(int argc, char * const argv[], const char * optstring)



с каждым вызовом разбирается следующая короткая опция,
возвращается символ опции (или **‘?’** - см. ниже);
если больше нечего разбирать, возвращает **-1**

Функция **getopt** считывает из глобальных переменных **<unistd.h>** положение места, где, предположительно, находится следующая опция, и пытается её разобрать и вернуть

В аргументе **optstring** описывается то, какие опции являются **допустимыми**,
и у каких допустимых опций есть аргументы

Если опция разобрана и оказалась недопустимой, то возвращается символ **‘?’**,
а символ опции записывается в переменную **optopt**

Если разобрана опция с аргументом, то указатель на аргумент в массиве argv
кладётся в переменную **optarg**

Формат опций

Формат аргумента **optstring** можно описать вот так:

- строка символов описывает множество символов допустимых опций без аргументов

  (*кроме ‘:’ и ‘;’ - их лучше не делать опциями*)

  Пример: строка **“abc”** говорит, что допустимы опции ‘a’, ‘b’, ‘c’ без аргументов

- если хочется сказать “а эта опция имеет аргумент”, в строку формата после символа опции следует добавить ‘:’

  Пример: строка **“ab:c”** говорит, что допустимы опции ‘a’, ‘c’ без аргументов и опция ‘b’ с аргументом

- если хочется сказать “а эта опция может как иметь, так и не иметь аргумент”, то после символа опции ставится два двоеточия

  Примечание: аргумент в этом случае должен **не** отделяться пробелом от опции:

  - Формат: **“b::”**
  - **“-barg”** - опция **‘b’** с аргументом **“arg”**
  - **“-b arg”** - опция **‘b’** без аргумента, а **“arg”** не относится к опциям

Формат опций

Формат аргумента **optstring** можно описать вот так:

- при разборе недопустимой опции и невозможности разобрать опцию с аргументом **getopt** пишет сообщение в поток ошибок
- если в начале строки **optstring** стоит символ **‘:’**, то это **не** опция; что происходит:
  - сообщения об ошибках не выводятся
  - если обязательный аргумент опции не найден, то вместо **‘?’** возвращается **‘:’**
- ещё один способ отключить сообщения об ошибках – написать команду **“opterr = 0”**
- по умолчанию **getopt** игнорирует параметры **argv**, не относящиеся к опциям
- чтобы изменить такой порядок обхода, можно предварить строку формата символом
  - **‘+’**, если хочется иметь обход “до первого параметра, не относящегося к опциям”
  - **‘-’**, чтобы каждый параметр, не относящийся к опциям, расценивался как опция **1** (*не символ, а число*) с аргументом – этим параметром

Пример

```cpp
#include <iostream>
#include <unistd.h>

int main(int argc, char ** argv) {
  int i;
  while((i = getopt(argc, argv, ":ab:c::")) != -1) {
    std::cout << "option " << (char)i << ", ";
    if(optarg == 0) std::cout << "no argument";
    else std::cout << "argument: " << optarg;
    std::cout << std::endl;
  }
  return 0;
}
```

```
terminal> ./main -abc -adcb -aac -b
option a, no argument
option b, argument: c
option a, no argument
option ?, no argument
option c, argument: b
option a, no argument
option a, no argument
option c, no argument
option :, no argument
terminal>
```

Функция getopt_long

int getopt_long(int argc, char * const argv[], const char * optstring,
const struct option * longopts, int * longindex)

С короткими опциями **getopt_long** работает так же, как и **getopt**

Два дополнительных аргумента – **longopts** и **longindex** –
нужны для работы с длинными опциями

longopts – это описание допустимых длинных опций

Если разобрана длинная опция и **longindex != NULL**, то в ***longindex** записывается смещение разобранной опции в массиве **longopts**

Для длинной опции в аргументе **longopts** можно установить **синоним** - символ, расцениваемый как короткий аналог этой опции

При разборе длинной опции (*как правило, но не всегда; подробности далее*) функция **getopt_long** возвращает короткий синоним разобранной опции

При этом синоним разбирается так, как описано в аргументе **optstring**

(*в том числе он может быть недопустимым или иметь другой набор аргументов*)

Описание длинной опции

```
struct option {
  const char * name;
  int has_arg;
  int * flag;
  int val;
}
```

name – имя опции

has_arg = 0 – без аргумента

has_arg = 1 – обязательный аргумент

has_arg = 2 – опциональный аргумент

val – синоним длинной опции

flag = 0 – значение **val** возвращается функцией **getopt_long** (*при разборе опции*)

flag != 0 – значение **val** записывается в ***flag**; функция **getopt_long** возвращает **0**

Функция getopt_long

int getopt_long(int argc, char * const argv[], const char * optstring, const struct option * longopts, int * longindex)

Аргумент **longopts** – это массив описаний длинных опций

Этот массив обязательно должен оканчиваться такой опцией **opt**:

- **opt.name = 0**
- **opt.has_arg = 0**
- **opt.flag = 0**
- **opt.val = 0**

Пример

```cpp
#include <iostream>
#include <getopt.h>

int main(int argc, char ** argv) {
  option o[4];
  o[0].name = "long1";
  o[0].has_arg = 0;
  o[0].flag = 0;
  o[0].val = '1';
  o[1].name = "long2";
  o[1].has_arg = 1;
  o[1].flag = 0;
  o[1].val = '2';
  o[2].name = "long3";
  o[2].has_arg = 2;
  int val;
  o[2].flag = &val;
  o[2].val = '3';
  o[3].name = 0;
  o[3].has_arg = 0;
  o[3].flag = 0;
  o[3].val = 0;
  int i;
  while((i = getopt_long(argc, argv, "2", o, NULL)) != -1) {
    std::cout << "option " << (char)i << ", ";
    if(optarg == 0) std::cout << "no argument";
    else std::cout << "argument: " << optarg;
    std::cout << std::endl;
  }
  return 0;
}
```

```
terminal> ./main --long1 --long3=a \
> --long3 --long2 --long1 -2 -1
option 1, no argument
option , argument: a
option , no argument
option 2, argument: --long1
option 2, no argument
./main: invalid option -- '1'
option ?, no argument
terminal>
```

Boost.Program_options

http://www.boost.org/doc/libs/1_58_0/doc/html/program_options.html

Boost – это набор библиотек, содержащих **всё** стандартное, что можно придумать

Разбор параметров командной строки – это стандартная задача,
и **Boost** имеет отдельную библиотеку для её решения:

Program_options

Как и почти для всего, что есть в Boost, для понимания этой библиотеки можно

- долго, медленно и вдумчиво читать документацию, или
- взять готовый пример и, примерно осознавая, что в нём происходит,
  переделать под себя

Поэтому перейдём сразу к примеру и посмотрим, как примерно это работает

Пример

```cpp
#include <boost/program_options.hpp>
namespace po = boost::program_options;

int main(int argc, char ** argv) {
  po::options_description desc("Short description");
  desc.add_options()
    ("help,h","help me")
    (",o","short, no arguments")
    ("opt", po::value<std::string>(),"long, string argument")
  ;
  po::variables_map vm;
  try {
    po::store(po::parse_command_line(argc, argv, desc), vm);
    po::notify(vm);
  }
  catch(po::error& e) {
    std::cout << e.what() << std::endl;
    std::cout << desc << std::endl;
    return 1;
  }
  if(vm.count("help") ) {
    std::cout << desc << std::endl;
    return 1;
  }
  if(vm.count("opt")) {
    std::cout << "option opt, argument: "
      << vm["opt"].as<std::string>() << std::endl;
  }
  return 0;
}
```

```
terminal> g++ main.cpp -lboost_program_options
terminal> ./a.out --help
Short description:
  -h [ --help ]          help me
  -o                     short, no arguments
  --opt arg              long, string argument

terminal> ./a.out --opt=AAA
option opt, argument: AAA
terminal>
```

Всё выглядит как-то непонятно?

Это обычная ситуация для **Boost**, так что надо пояснить каждую строчку

Подключение

```
#include <boost/program_options.hpp>
namespace po = boost::program_options;
```

Для использования программных опций нужен header **program_options.hpp**, лежащий в одной из стандартных папок в подпапке boost

(*если библиотеки **Boost** устанавливались в стандартные папки*)

Всё, что определено **Boost**'ом, находится в пространстве имён **boost**

Всё, что относится к разбору параметров командной строки, находится в подпространстве **boost::program_options**

Здесь для этого подпространства для удобства заведено сокращение **"po"**

Описание опций

```
po::options_description desc("Short description");
desc.add_options()
  ("help,h","help me")
  (",o","short, no arguments")
  ("opt", po::value<std::string>(),"long, string argument")
;
```

Класс **options_description** хранит (*как бы это ни было очевидно*) описание опций

В числе прочего здесь хранится описание опций, которое будет выводиться по запросу наподобие **“--help”**

Это описание состоит из описания для каждой опции и того, что предваряет все эти описания (**“Short description”**)

При определении опции первый аргумент может выглядеть так:

- **“name”**: *name* - имя длинной опции
- **“name,n”**: *name* - имя длинной опции, *n* - синоним
- **“,n”**: *n* – имя короткой опции

Описание опций

```
po::options_description desc("Short description");
desc.add_options()
  ("help,h","help me")
  (",o","short, no arguments")
  ("opt", po::value<std::string>(),"long, string argument")
;
```

Если при определении опции используются два аргумента, то второй –
описание опции при вызове чего-нибудь наподобие **--help**

Если используются три аргумента, то второй может указывать на то,
в какой тип (*из естественных строково-числовых*) будет преобразовываться аргумент

Например, аргумент **opt** будет иметь тип **std::string** и инициализироваться естественно

Если аргумента, описывающего тип, нет, то (*обычно*) это означает,
что опция не имеет аргументов

Разбор строки

```
po::variables_map vm;
try {
  po::store(po::parse_command_line(argc, argv, desc), vm);
  po::notify(vm);
}
```

variables_map – класс, в котором будет собираться вся информация об опциях

parse_command_line возвращает особый объект (*неважно какой*),
который потом записывается в переменной,
собирающей информацию об опциях, с помощью функции **store**

notify(*vm*) – это магия, не задумывайтесь об этом, просто пишите
(*чтобы понять, зачем нужно делать **notify**, нужно неплохо разобраться в **Boost***)

Разбор строки

```
catch(po::error& e) {
    std::cout << e.what() << std::endl;
    std::cout << desc << std::endl;
    return 1;
}
```

Если параметры по какой-то причине не получилось разобрать,

то **parse_command_line** выкинет исключение типа **error**

При желании можно обернуть разбор строки в **try**-блок и узнать, что пошло не так:

- поймать исключение po::error
- спросить у пойманного исключения: **what?**

  (*стандартная вещь для исключений*)

- а затем вывести **desc** – *об этом позже*

Обработка значений опций

```
if(vm.count("help") ) {
  std::cout << desc << std::endl;
  return 1;
}
```

variables_map::count(*name*) говорит, встречалась ли опция *name* в параметрах строки

Здесь написан стандартный способ обработки опции **--help** (она же **-h**)

Для класса описания опций перегружена функция вывода в поток

Она красиво выводит краткое описание и описание всех опций (*см. ранее*)

Обработка значений опций

```
if(vm.count("opt")) {
  std::cout << "option opt, argument: "
    << vm["opt"].as<std::string>() << std::endl;
}
```

Объект класса **variables_map** может работать как **map** из имени опции
в особую структуру с информацией об опции

Функция **.as<Type>()**, применённая к этой структуре, возвращает значение
аргумента опции, расценивая информацию для значения как объект типа **Type**

Компиляция

```
terminal> g++ main.cpp -lboost_program_options
terminal> ./a.out --help
Short description:
  -h [ --help ]         help me
  -o                    short, no arguments
  --opt arg             long, string argument

terminal> ./a.out --opt=AAA
option opt, argument: AAA
terminal>
```

Boost.Program_options – это **нестандартная библиотека**,
и при компиляции нужно уметь подключать

В Linux всё просто: если Boost ставился менеджером пакетов
(*или неважно кем, но по стандартным путям*),
то достаточно при компиляции в **g++** добавить флаг **“-lboost-program-options”**