# Modern C++ - Parameters of main

## Parameters of the main function

The main function is declared in one of the following ways:

```
1 int main() { /* ... */ }
2 int main( int argc, char *argv[]) { /* ... */ }
3
4 // only when the host environment supports it (mainly UNIX):
5 int main( int argc, char *argv[], char *envp[]) { /* ... */ }
```

If **argc** is defined then **argv[argc]** is NULL pointer. If **argc** is greater than zero, then **argv[0]** is the name of the program, and **argv[1] ... argv[argc-1]** are the program parameters. The **argv[i]** parameters are pointers to NULL terminated character arrays.

In usual environments, **argc** is always greater than zero.

```
1 #include <iostream>
2 int main(int argc, char *argv[])
3 {
4   std::cout << "name of the program = " << argv[0] << std::endl;
5   for (int i = 1; i < argc; ++i)
6     std::cout << "argv[" << i << "] = " << argv[i] << std::endl;
7   return 0;
8 }
```

```
$ g++ -ansi -Wall -W -o mainpars mainpars.cpp
$ ./mainpars
name of the program = ./mainpars
$ ./mainpars first second third
name of the program = ./mainpars
argv[1] = first
argv[2] = second
argv[3] = third
```

In the following we implement a *grep*-like program that can read input line by line and print those lines to the standard output when there is matching with the patterns given as the first mandatory parameter.

We will implement only a small subset of grep, we will concentrate mainly on the handling of the command line parameters.

Here is a minimal version:

```
 1 /*
 2  * gr.cpp -- a minimal grep-like program
 3  * usage: gr pattern [files...]
 4  *
 5  */
 6 #include <iostream>     // for basic io
 7 #include <fstream>      // for files
 8 #include <string>       // for string class
 9 #include <cerrno>       // for errno
10 #include <cstring>      // for strerror()
11 #include <cstdlib>      // for exit() in usage()
12
13 //
14 // matching function called for all lines
15 //
16 void gr( std::istream& in, std::string pattern)
17 {
18   std::string  line;
19   while ( std::getline( in, line) )
20   {
21     if ( std::string::npos != line.find(pattern) )
22     {
23       std::cout << line << std::endl;
24     }
25   }
26 }
27
28 int main( int argc, char *argv[])
29 {
30   if ( argc  < 2 )  // missing pattern
31   {
32     std::cerr << "Usage: " << argv[0] << " pattern [files...]" <<
std::endl;
33     exit(1); // terminate program
34   }
35   else if ( argc < 3 ) // missing filenames
```

```
36   {
37     gr( std::cin, argv[1]);
38   }
39   else
40   {
41     for ( int i = 2; i < argc; ++i )  // start from i for file arguments
42     {
43       std::ifstream inpfile( argv[i], std::ios_base::in);   // constructor
44
45       if ( ! inpfile )  // converted to logical value
46       {
47         std::setlocale(LC_MESSAGES, "en_EN.utf8");
48         std::cerr << argv[i] << ": " << std::strerror(errno) << std::endl;
49       }
50       else
51       {
52         gr( inpfile, argv[1]);
53       }
54     }  // inpfile closes automatically -  inpfile destructor called
55   }
56   return 0;
57 }
```

Let start to explain the program from the main function.

```
1 int main( int argc, char *argv[])
2 {
3   if ( argc  < 2 )  // missing pattern
4   {
5     std::cerr << "Usage: " << argv[0] << " pattern [files...]" << std::endl;
6     exit(1); // terminate program
7   }
```

The **main** function declares the **argc** and **argv** parameters. As in **argv** the 0th element is the program name itself, and the 1st is the mandatory *pattern* parameter, if we have less argument, we emit error message - usually about the correct usage of the program - and finish the program. Here we use **argv[0]** as the program name, since the user can rename the binary which can be extracted only from here.

```
1 else if ( argc < 3 ) // missing filenames
2   {
```

```
3     gr( std::cin, argv[1]);
4   }
```

If there is further parmeter(s), we suppose they are filenames. In an extended version of the program here we should have handle other
program parameters, i.e. *flags*.

The actual work for a whole file happens in **gr** function. If there is no further parameter after the mandatory pattern, we read from standard input and write the results to standard output. The arguments of **gr** are the input stream and the pattern itself.

```
 1 else
 2   {
 3     for ( int i = 2; i < argc; ++i )  // start from i for file arguments
 4     {
 5       std::ifstream inpfile( argv[i], std::ios_base::in);   // constructor
 6
 7       if ( ! inpfile )  // converted to logical value
 8       {
 9         std::setlocale(LC_MESSAGES, "en_EN.utf8");
10         std::cerr << argv[i] << ": " << std::strerror(errno) << std::endl;
11       }
12       else
13       {
14         gr( inpfile, argv[1]);
15       }
16     }  // inpfile closes automatically -  inpfile destructor called
17   }
```

Otherwise we have to walk thru all filename parameters, and construct an **ifstream** object representing the input stream. The *constructor* of ifstream tries to open the file. If successful, the object has value **true** and we pass it to *gr*. We do not need to close the file, the *destructor* of ifstream class will close the file at the end of the loop cycle.

If we can not open the file, we emit error message and continue with the next file.

The actual work happens in the **gr** file. The first parameter is the input stream and the second is the pattern to find.

We organize a loop reading lines until end of file with the help of the standard library function **getline**. The second parameter **line** is passed by reference and here placed the actual line as string.

```
 1 void gr( std::istream& in, std::string pattern)
 2 {
 3   std::string  line;
 4   while ( std::getline( in, line) )
 5   {
 6     if ( std::string::npos != line.find(pattern) )
 7     {
 8       std::cout << line << std::endl;
 9     }
10   }
11 }
```

The **find(pattern)** call on **line** try to find **pattern** as a substring of **line**. If this is succesfull, the actual position returns, otherwise returns the **std::string::npos** constant.

If the return value is *not* **std::string::npos** we have a match and we print the line.