Command-Line Parameters A01. CL Parameters 1

Table of Contents

Access the Command Line A01. CL Parameters 2

To access the command-line environment under Windows, go to the Start Menu and select MS-DOS prompt (really a command shell, under NT).

Note the location you've saved your source code and change to that directory (aka folder), by using the commands:

```
d:          where d is the drive letter.
cd <path>   where <path> is the sequence
            of directories from the root
            of the drive to your source
            code.
```

Note that Visual C++ creates your executable in a subdirectory of the location of your source files, probably `debug`. You'll have to be in the same directory as your executable in order for the following instructions to work.

You can run your program by typing the name of the executable and then `<Enter>`.

If your program requires command-line parameters, type them after the name of the executable.

Program Parameters

`main(int argc, char* argv[])`

- When a program name is typed at the Operating System Line prompt, it is treated as a command. The name of the executable program file and any other strings typed on the line, (usually file names), are considered command line parameters.
- C/C++ compilers provide access to command line parameters.

`argc`

- gives the number of command line parameters (arguments) that the user typed on the command line.
- always equals at least 1 for the command itself

`argv[]`

- an array of pointers to C-style strings that holds the command line arguments.
- argv[0] holds the name of the program (command)
- may hold full pathname on some systems
- may be capitalized on DOS systems

Echo Command Parameters

Echo the command line parameters:

```
#include <iostream>
#include <cstdlib>     // for EXIT_SUCCESS
using namespace std;

int main (int argc, char* argv[]) {

   cout << "argc = " << argc << endl;

   for (int i =0 ; i < argc; i++) {
     cout << "argv[ " << i << " ] = "
          << argv[i] << endl;
   }

   return EXIT_SUCCESS;
}
```

number of parameters

array of (pointers to) parameters


```
// OpenFilesCL.cpp
// Illustrates use of command-line parameters.
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <cstring>

using namespace std;

void InvocationError();
void SplashScreen();

void main(int argc, char* argv[]) {

   // Check for correct number of command-line
   // arguments:
   if (argc != 2) {
      InvocationError();
      return;
   }

   // Check for help request:
   // (Remember the program name counts as one.)
   if ( (argc == 2) &&
        (strcmp(argv[1], "/?") == 0) ) {
      SplashScreen();
      return;
   }
```



```
// continued...
   char* fileName = argv[1];

   ifstream iFile;
   iFile.open(fileName);
   if (iFile.fail()) {
      cout << ""File: "
           << fileName
         << " not found in current directory.\n";
      return;
   }
   string Line;
   getline(iFile, Line);
   int LineNumber = 0;

   while (iFile) {

      cout << setw(5) << LineNumber << "  "
           << Line << endl;
      getline(iFile, Line);
      LineNumber++;
   }
   iFile.close();
}  // end main()
```



```
void SplashScreen() {

   string Splash = "Displays named file to "
            "screen with line numbers.\n\n";

   cout << Splash;
   InvocationError();
}

void InvocationError() {

   string Invocation = "Invocation:  OpenFilesCL"
          " <file name>\n"
          "where\n"
          "      <file name> must be the name of"
          " an existing file\n"
          "                  in the current"
          " directory\n\n";

   cout << Invocation;
}
```


Given the text file `sample.text`:

```
Command-line parameters provide a way
for a program to obtain user input
without using a prompt and reading
from cin.
```

Typing: `OpenFilesCL sample.text`:

produces the output:

```
0  Command-line parameters provide a way
1  for a program to obtain user input
2  without using a prompt and reading
3  from cin.
```



```
// CalculatorCL.cpp
// Illustrates use of numerical command-line
// parameters.
//
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

enum ArgStatus {BadOperand1, BadOperator, BadOperand2,
                AllOK};

void      InvocationError();
void      ArgumentsError(ArgStatus clStatus);
ArgStatus CheckCLArguments(char* argv[]);
int       Calc(int Operand1, char Operator,
               int Operand2);
void      PrintExpression(int Operand1, char Operator,
                          int Operand2, int Result);

void main(int argc, char* argv[]) {

   // Check for correct number of command-line arguments:
   // (Remember the program name counts as one.)
   if (argc != 4) {
      InvocationError();
      return;
   }

   // continues...
```



```
// ... continued
   // Check whether command-line parameters are of
   // correct types:
   ArgStatus clStatus = CheckCLArguments(argv);

   if ( clStatus != AllOK) {
      ArgumentsError(clStatus);
      return;
   }

   // convert command-line arguments to proper types:
   // (atoi converts ascii string to an integer)
   int  Operand1 = atoi(argv[1]);
   char Operator = argv[2][0];
   int  Operand2 = atoi(argv[3]);

   int  Result = Calc(Operand1, Operator, Operand2);

   PrintExpression(Operand1, Operator, Operand2, Result);
}
```



```
ArgStatus CheckCLArguments(char* argv[]) {

   bool OperandOK  = true, OperatorOK = true;

   OperatorOK = (argv[2][1] == '\0') &&  // check
                                         // operator
                                         // length
                ( (argv[2][0] == '+') || // check
                  (argv[2][0] == '-') || // operator
                  (argv[2][0] == '*') || // symbol
                  (argv[2][0] == '/') );

   if (!OperatorOK) return BadOperator;

   // Check if first operand is a string of digits:
   for (int Idx = 0; argv[1][Idx] != '\0'; Idx++) {

      if ( !isdigit(argv[1][Idx]) ) {
         OperandOK = false;
         break;
      }
   }

   if (!OperandOK)  return BadOperand1;

   // continues...
```



```
 // ... continued
   // Check if second operand is a string of digits:
   for (Idx = 0; argv[3][Idx] != '\0'; Idx++) {

      if ( !isdigit(argv[3][Idx]) ) {
         OperandOK = false;
         break;
      }
   }

   if (!OperandOK)  return BadOperand2;

   // No errors detected if we get this far:
   return (AllOK);
}

int Calc(int Operand1, char Operator, int Operand2) {

   switch (Operator) {
   case '+': return (Operand1 + Operand2);
   case '-': return (Operand1 - Operand2);
   case '*': return (Operand1 * Operand2);
   case '/': if (Operand2 == 0)
                return 0;
             else
                return (Operand1 / Operand2);
   default:  return 0;
   }
}
```



```
void PrintExpression(int Operand1, char Operator, int
   Operand2, int Result) {

   cout << Operand1 << ' '
        << Operator << ' '
        << Operand2 << " = "
        << Result << endl;
}

void InvocationError() {

   string Invocation =
       "Invocation:  CalculatorCS "
       "<integer> <operator> <integer>\n"
       "where\n"
       "      <integer> must be an integer value\n"
       "      <operator> must be one of: '+', '-', "
       "'*' or '/'\n\n"
       "and the expression must be well-formed.\n";

   cout << Invocation;
}
```



```
void ArgumentsError(ArgStatus clStatus) {

   string PleaseFix      =
          "Please correct the problem.\n\n";
   string BadOp1Msg      =
          "The first operand is not an integer.\n";
   string BadOp2Msg      =
          "The second operand is not an integer.\n";
   string BadOperatorMsg =
          "The operand is not + or - or * or /.\n";
   string AllOKMsg       =
          "The command-line arguments are all proper.\n";
   string BadStatusMsg   =
          "Unrecognized command-line status.\n"
          "Please notify program vendor.\n\n";

   switch (clStatus) {
   case BadOperand1:  cout << BadOp1Msg << PleaseFix;
                      break;
   case BadOperand2:  cout << BadOp2Msg << PleaseFix;
                      break;
   case BadOperator:  cout << BadOperatorMsg
                           << PleaseFix;
                      break;
   case AllOK:        cout << AllOKMsg;
                      break;
   default:           cout << BadStatusMsg;
   }
}
```

