Command Line Parameters

CSE 1320

Command Line Parameters

Command line parameters are a sequence of strings used to pass information to a C program at execution time.

They appear as strings on the command line after the name of the program when it is executed.

These strings are separated by blanks, tabs, or other whitespace.

Command line parameters are available throughout the time that the program is executing.

Command Line Parameters

Command line parameters are accessed via arguments to `main()`

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

`argc` and `argv` are traditional names but can be anything

`argc` contains the count of parameters on the command line. The name of the program is the first command line parameter and it is part of the count so `argc` is always at least one.

Command Line Parameters

`argv` is an array of pointers to `chars`

the pointers point to the strings that appear on the command line

the array is indexed by `0` to `argc - 1` and terminated with a `NULL` pointer

Command Line Parameters

Running a program with command line parameters

```
a.out clp1 clp2 clp3
```

Running a program in debug with command line parameters

```
gdb --args a.out clp1 clp2 clp3
```

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

   printf("argc = %d\n", argc);

   for (i = 0; i < argc; i++)
   {
      printf("argv %d - %s\n", i, argv[i]);
   }

   return 0;
}
```

```
a.out clp1 clp2

argc = 3
argv 0 - a.out
argv 1 - clp1
argv 2 - clp2
```

```
a.out Monday,Tuesday
argc = 2
argv 0 - a.out
argv 1 - Monday,Tuesday
```

```
a.out
argc = 1
argv 0 - a.out
```

```
a.out frog TOAD elePhant candy
argc = 5
argv 0 - a.out
argv 1 - frog
argv 2 - TOAD
argv 3 - elePhant
argv 4 - candy
```

```
a.out What day is today?
argc = 5
argv 0 - a.out
argv 1 - What
argv 2 - day
argv 3 - is
argv 4 - today?
```

```
a.out trick-or-treat
argc = 2
argv 0 - a.out
argv 1 - trick-or-treat
```

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
   char filename1[20] = {};
   char filename2[20] = {};

   if (argc == 3)
   {
      strcpy(filename1, argv[1]);
      strcpy(filename2, argv[2]);

      printf("filename1 is %s and filename2 is %s\n",
      filename1, filename2);
   }
   else
   {
      printf("Need 2 command line parameters\n");
   }

   return 0;
}
```

What output would this program produce given certain command line parameters?

```
[frenchdm@omega ~]$ argcargv4Demo.e
Need 2 command line parameters
[frenchdm@omega ~]$ argcargv4Demo.e lion
Need 2 command line parameters
[frenchdm@omega ~]$ argcargv4Demo.e lion tiger
filename1 is lion and filename2 is tiger
[frenchdm@omega ~]$ argcargv4Demo.e lion tiger bear
Need 2 command line parameters
[frenchdm@omega ~]$ argcargv4Demo.e lion tiger-bear
filename1 is lion and filename2 is tiger-bear
[frenchdm@omega ~]$ argcargv4Demo.e lion, tiger-bear
filename1 is lion, and filename2 is tiger-bear
[frenchdm@omega ~]$ argcargv4Demo.e lion,tiger-bear
Need 2 command line parameters
[frenchdm@omega ~]$ argcargv4Demo.e lion,tiger bear
filename1 is lion,tiger and filename2 is bear
```

Command line parameters in debug.

```
gdb --args argcargv4Demo.e inputfile outputfile
…
Reading symbols from /home/f/fr/frenchdm/argcargv4Demo.e...done.
(gdb) break main
Breakpoint 1 at 0x400537: file argcargv4Demo.c, line 8.
(gdb) run
Starting program: /home/f/fr/frenchdm/argcargv4Demo.e inputfile outputfile
warning: no loadable sections found in added symbol-file system-supplied DSO at
0x2aaaaaaab000
Breakpoint 1, main (argc=3, argv=0x7fffffffe7e8) at argcargv4Demo.c:8
8                 char filename1[20] = {};
(gdb) p argv
$1 = (char **) 0x7fffffffe7e8
(gdb) p *argv@argc
$2 = {0x7fffffffea2a "/home/f/fr/frenchdm/argcargv4Demo.e",
  0x7fffffffea4e "inputfile", 0x7fffffffea58 "outputfile"}
(gdb) p argc
$3 = 3
(gdb) p argv[0]
$4 = 0x7fffffffea2a "/home/f/fr/frenchdm/argcargv4Demo.e"
(gdb) p argv[1]
$5 = 0x7fffffffea4e "inputfile"
(gdb) p argv[2]
$6 = 0x7fffffffea58 "outputfile"
```

Write a C program that sums command line parameters. Assume the command line parameters are integers. Make no assumptions about how many parameters are on the command line – your program should be able to handle 0 to many command line parameters.

Sample runs

```
student@cse1325:/media/sf_VM2320$ ./a.out
Total = 0
student@cse1325:/media/sf_VM2320$ ./a.out 1
Total = 1
student@cse1325:/media/sf_VM2320$ ./a.out 1 2
Total = 3
student@cse1325:/media/sf_VM2320$ ./a.out 1 22
Total = 23
student@cse1325:/media/sf_VM2320$ ./a.out 1 22 4
Total = 27
student@cse1325:/media/sf_VM2320$ ./a.out 1 22 4 3
Total = 30
student@cse1325:/media/sf_VM2320$ ./a.out 1 22 4 32
Total = 59
```

```c
// Sum command line parameters
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
	int total = 0;
	int i = 0;

	for (i = 1; i < argc; i++)
	{
		total += atoi(argv[i]);
	}

	printf("Total = %d\n", total);

	return 0;
}
```