Command-line arguments in the C language

It is possible to pass some values from the command line to your C programs when they are executed. These values are called **command line arguments** and many times they are important for your program specially when you want to control your program from outside instead of hard coding those values inside the code.

The C language provides a method to pass parameters to the main() function. This is typically accomplished by specifying arguments on the operating system command line (console).
The prototype for main() looks like:

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

There are two parameters passed to main(). The first parameter is the number of items on the command line (int argc). Each argument on the command line is separated by one or more spaces, and the operating system places each argument directly into its own null-terminated string. The second parameter passed to main() is an array of pointers to the character strings containing each argument (char *argv[]).
For example, at the command prompt:

```
test _prog  1  apple  orange  4096.0
```

There are 5 items on the command line, so the operating system will set argc=5 . The parameter argv is a pointer to an array of pointers to strings of characters, such that:

argv[0] is a pointer to the string "test_prog"
argv[1] is a pointer to the string "1"
argv[2] is a pointer to the string "apple"
argv[3] is a pointer to the string "orange"

and

argv[4] is a pointer to the string "4096.0"

Notes

- The main() routine can check argc to see how many arguments the user specified.
- The minimum count for argc is 1: the command line just contained the name of the invoked program with no arguments.
- The program can find out its own name as it was invoked: it is stored in the argv[0] string! Some operating systems don't provide this feature, however.
- The arguments from the command line are *not* automatically converted: the characters are just copied into the argv strings.

• If an argument on the command line is to be interpreted as a numerical constant, such as argv[1] and argv[4] in this example, it can be converted using a string conversion.

```
int  int_val;
float float_val;
int_val=atoi (argv[1]);
float_val=atof (argv[4]);

printf("The 1st  and 4th items on the command line are %d and %f\n", argv[1], argv[4]);
```

// The C library function **int atoi(const char *str)** converts the string argument **str** to an integer (type int).

//The C library function **double atof(const char *str)** converts the string argument **str** to a floating-point number (type double).

and

```
printf("The 3rd and 4th items on the command line are %s and %s\n", argv[2], argv[3]);
```

results in:
The 2nd and 3rd items on the command line are apple and orange

The command line arguments are handled using main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program. Following is a simple example which checks if there is any argument supplied from the command line and take action accordingly:

```
#include <stdio.h>

int main( int argc, char *argv[] )
{
  if( argc = = 2 )
  {
    printf("The argument supplied is %s\n", argv[1]);
  }
  else if( argc > 2 )
  {
    printf("Too many arguments supplied.\n");
  }
  else
  {
    printf("One argument expected.\n");
  }
}
```

When the above code is compiled and executed with a single argument, it produces the following result.

```
C:\tc> testing testing1
The argument supplied is testing1
```

When the above code is compiled and executed with a two arguments, it produces the following result.

```
C:\tc>testing  testing1 testing2
Too many arguments supplied.
```

When the above code is compiled and executed without passing any argument, it produces the following result.

```
C:\tc>testing
One argument expected
```

It should be noted that argv[0] holds the name of the program itself and argv[1] is a pointer to the first command line argument supplied, and *argv[n] is the last argument. If no arguments are supplied, argc will be one, otherwise and if you pass one argument then argc is set at 2.

You pass all the command line arguments separated by a space, but if argument itself has a space then you can pass such arguments by putting them inside double quotes "" or single quotes ''. Let us re-write above example once again where we will print program name and we also pass a command line argument by putting inside double quotes:

```
#include <stdio.h>

int main( int argc, char *argv[] )
{
  printf("Program name %s\n", argv[0]);

  if( argc == 2 )
  {
    printf("The argument supplied is %s\n", argv[1]);
  }
  else if( argc > 2 )
  {
    printf("Too many arguments supplied.\n");
  }
  else
  {
    printf("One argument expected.\n");
  }
}
```

When the above code is compiled and executed with a single argument separated by space but inside double quotes, it produces the following result.

```
C:\tc> testing "testing1 testing2"

Progranm name testing
The argument supplied is testing1 testing2
```

```
            #include<stdio.h>
#include<stdlib.h>
int main(int argc, char * argv[]) {
            int  i;

            if (argc<0) {
                        printf("You have forgot to specify arguemnts");
                        exit(1);
            }
             for(i=0;i<argc;i++)
                        printf("\n%s",argv[i]);
            return 0;

}
```

Output:-

```
gcc  cmdex.c
.a/out 22 33 44 55

.a/out 22 33 44 55
```

Program to add two number by using the command line argument

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char * argv[]) {
            int  sum = 0;

            if (argc != 3) {
                        printf("You have forgot to specify two numbers.");
                        exit(1);
            }
            printf("The sum is : ");

            sum= atoi(argv[1])+atoi(argv[2]);

            printf("%d", sum);
            return 0;

}
```

Output:-

```
gcc  cmdex.c
.a/out 22 33
The sum is: 55
```