Command line arguments
Variable no.of arguments
Make File Utility

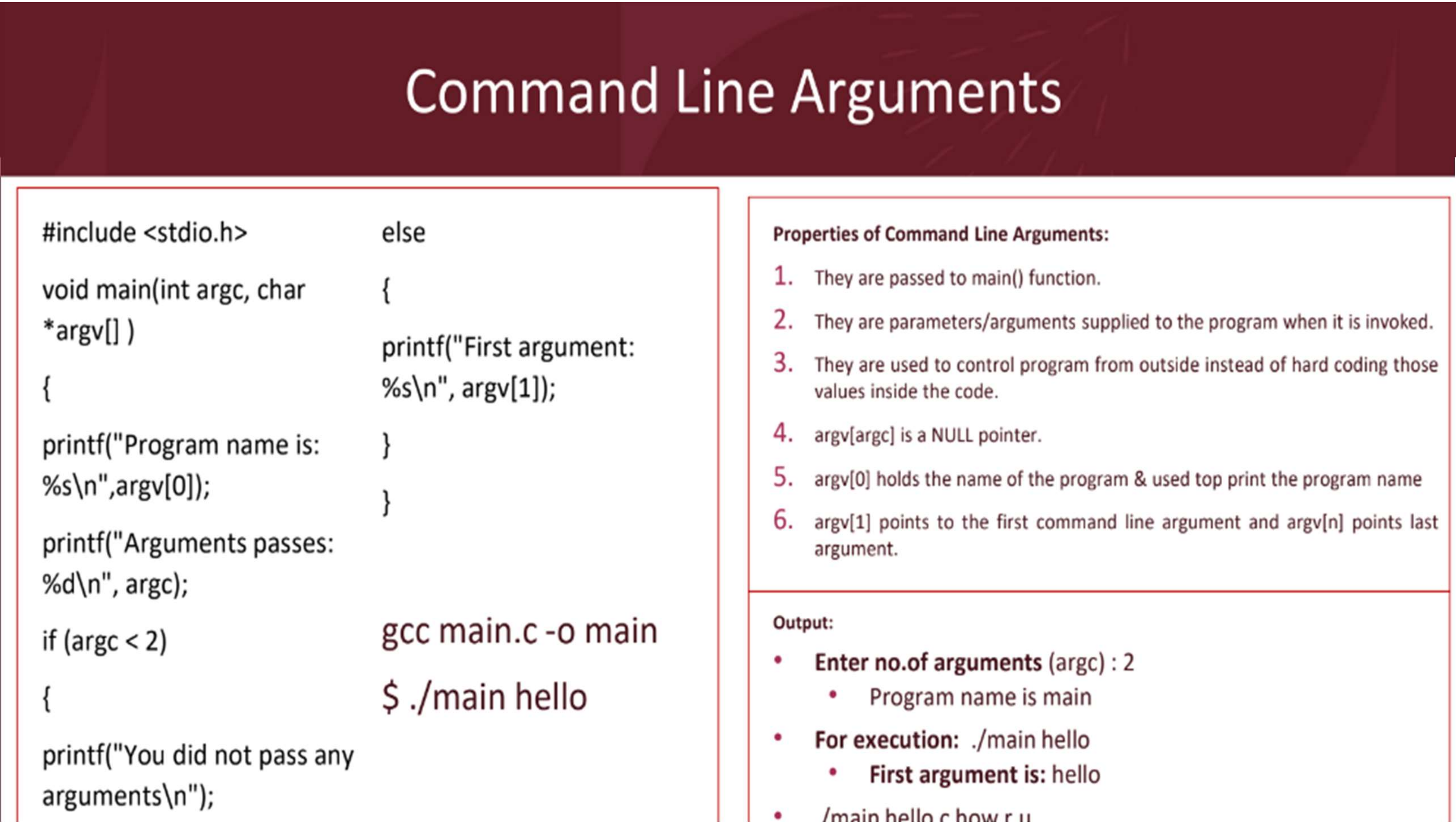
By

V Jaya Shravan

Command Line Arguments

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

- To pass command line arguments, we typically define main() with two arguments:
  - first argument is the number of command line arguments and
  - second is list of command-line arguments.
- argc is int and stores number of command-line arguments passed by the user including the name of the program. So, if we pass a value to a program, value of argc would be 2 (one for argument and one for program name)
- The value of argc should be nonnegative.
- argv is array of character pointers listing all the arguments.
- If argc is greater than zero, the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
- argv[0] is the name of the program . After that till argv[argc-1] every element is command -line arguments.



Command Line Arguments

```
#include <stdio.h>
void main(int argc, char
*argv[] )
{
printf("Program name is:
%s\n",argv[0]);
printf("Arguments passes:
%d\n", argc);
if (argc < 2)
{
printf("You did not pass any
arguments\n");
else
{
printf("First argument:
%s\n", argv[1]);
}
}
```

```
gcc main.c -o main
$ ./main hello
```

Properties of Command Line Arguments:

1. They are passed to main() function.
2. They are parameters/arguments supplied to the program when it is invoked.
3. They are used to control program from outside instead of hard coding those values inside the code.
4. argv[argc] is a NULL pointer.
5. argv[0] holds the name of the program & used top print the program name
6. argv[1] points to the first command line argument and argv[n] points last argument.

Output:

- **Enter no.of arguments** (argc) : 2
  - Program name is main
- **For execution:** ./main hello
  - **First argument is:** hello
- ./main hello c how r u

Variable no.of Arguments

- Variadic functions are functions that can take a variable number of arguments. Its adds flexibility to the program.
- It takes one fixed argument and then any number of arguments can be passed. The variadic function consists of at least one fixed variable and then an ellipsis(...) as the last parameter.
- **Syntax:** int function_name(data_type variable_name, ...);
- **Library: stdarg.h**

```
#include<stdarg.h>

#include<stdio.h>

int sum(int, ...);


int main(void)

{

  printf("Sum of 10, 20 and 30 =
  %d\n",  sum(3, 10, 20, 30) );

   return 0;

}
```

```
int sum(int num_args, ...)

{

  int val = 0;

  va_list ap;

  int i;

  va_start(ap, num_args);

  for(i = 0; i < num_args; i++)

{

    val += va_arg(ap, int);

  }

  va_end(ap);

  return val;

}
```

Variable no.of Arguments

Methods	Description
va_start(va_list ap, argN)	This enables access to variadic function arguments. where *va_list* will be the pointer to the last fixed argument in the variadic function *argN* is the last fixed argument in the variadic function. From the above variadic function (function_name (data_type variable_name, …);), variable_name is the last fixed argument making it the argN. Whereas *va_list ap* will be a pointer to argN (variable_name)
va_arg(va_list ap, type)	This one accesses the next variadic function argument.
va_copy(va_list dest, va_list src)	This makes a copy of the variadic function arguments.
va_end(va_list ap)	This ends the traversal of the variadic function arguments.

Variable no.of arguments

- The standard library in C provides several functions that use variable number of arguments, such as printf() and scanf().
- These functions allow us to pass a varying number of arguments to them, which makes them very versatile in handling different types of input and output.
- The most commonly used macros are va_start(), va_arg(), and va_end(), which allow us to initialize the argument list, access the arguments, and clean up the argument list, respectively.

Make File Utility

Makefile using variables and clean target:

Makefile content:

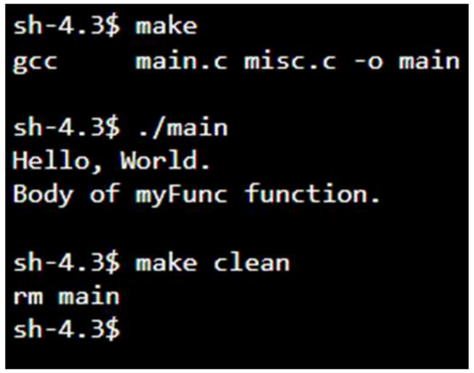
```
CC=gcc  #compiler
TARGET=main #target file name


all:
    $(CC) main.c misc.c -o $(TARGET)


clean:
    rm $(TARGET)
```

To compile makefile type make

To clean type make clean



```
sh-4.3$ make
gcc      main.c misc.c -o main

sh-4.3$ ./main
Hello, World.
Body of myFunc function.

sh-4.3$ make clean
rm main
sh-4.3$
```

Make File Utility

misc.c

```
#include <stdio.h>
#include "misc.h"
/*function definition*/
void myFunc(void)
{
   printf("Body of myFunc function.\n");
}
```

misc.h

```
#ifndef MISC_H
   #define MISC_H
   /*function declaration.*/
   void myFunc(void);
#endif
```

main.c

```
#include <stdio.h>
#include "misc.h"
int main()
{
   printf("Hello, World.\n");
   myFunc();
    return 0;
}
```

Makefile Content:

```
all:   #target name
   gcc main.c misc.c -o main
```

Save file with name **"Makefile".**

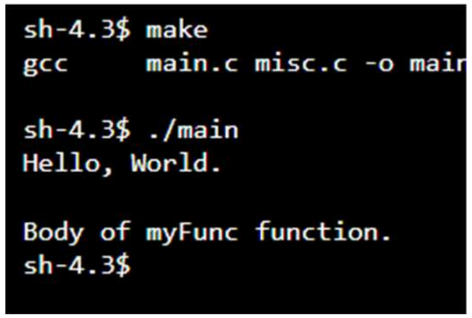
Insert comment followed by # character.

all is a target name, insert : after target name.

gcc is compiler name, **main.c, misc.c** source file names, **-o** is linker flag and **main** is binary file name.

Command **make** is used to compile program through **Makefile**

- make all



```
sh-4.3$ make
gcc      main.c misc.c -o mai

sh-4.3$ ./main
Hello, World.

Body of myFunc function.
sh-4.3$
```