# ALX LESSON

# 0x0A C - argc, argv

C - Programming

# TABLE OF CONTENTS


01 Overview topics

02 Learning Objectives

03 Quiz questions

04 hands on lab practice

# 01

# OVERVIEW topics

# Topics


argc

argv

How to use arguments passed to your program

What are two prototypes of main that you know of, and which case do you use one or the other

How to use __attribute__((unused)) or (void) to compile functions with unused variables or parameters

# Slides On Telegram

https://t.me/alx_2023

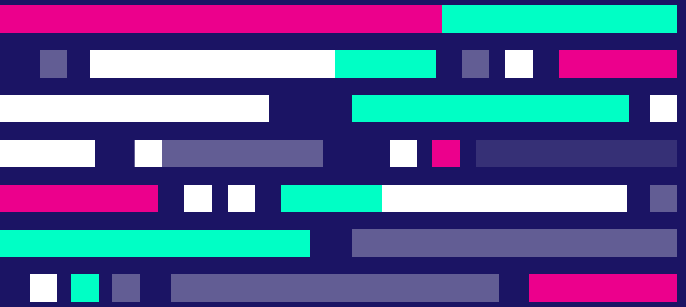
# C Programming

Topics

# 02

# Learning Objectives

# Arguments to main

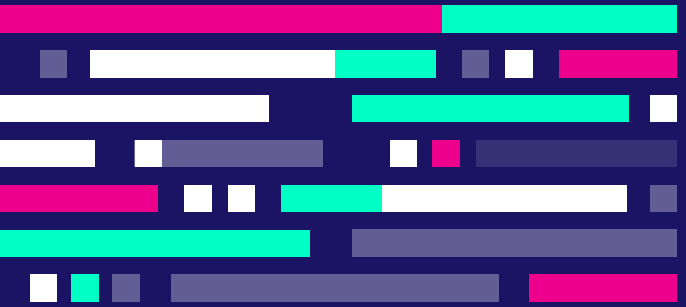


There are at least two arguments to main: argc and argv.

The first of these is a count of the arguments supplied to the program and the second is an array of pointers to the strings which are those arguments—its type is (almost) 'array of pointer to char'.

These arguments are passed to the program by the host system's command line interpreter or job control language.

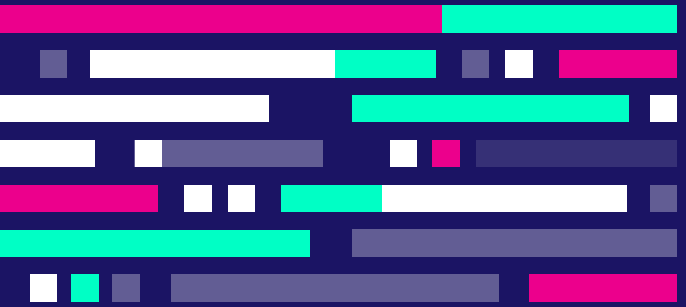
# Arguments to main



The declaration of the argv argument is often a novice programmer's first encounter with pointers to arrays of pointers and can prove intimidating. However, it is really quite simple to understand. Since argv is used to refer to an array of strings, its declaration will look like this:

```
char *argv[]
```

# Arguments to main



```
char *argv[]
```

Remember too that when it is passed to a function, the name of an array is converted to the address of its first element. This means that we can also declare argv as `char **argv`; the two declarations are equivalent in this context.

Indeed, you will often see the declaration of main expressed in these terms. This declaration is exactly equivalent to that shown above:

```
int main(int argc, char **argv);
```

# Arguments to main

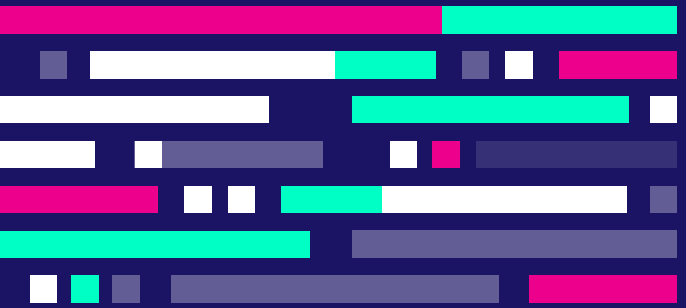
When a program starts, the arguments to main will have been initialized to meet the following conditions:

argc is greater than zero.
argv[argc] is a null pointer.

argv[0] through to argv[argc-1] are pointers to strings whose meaning will be determined by the program.

argv[0] will be a string containing the program's name or a null string if that is not available. Remaining elements of argv represent the arguments supplied to the program. In cases where there is only support for single-case characters, the contents of these strings will be supplied to the program in lower-case.

# Arguments to main

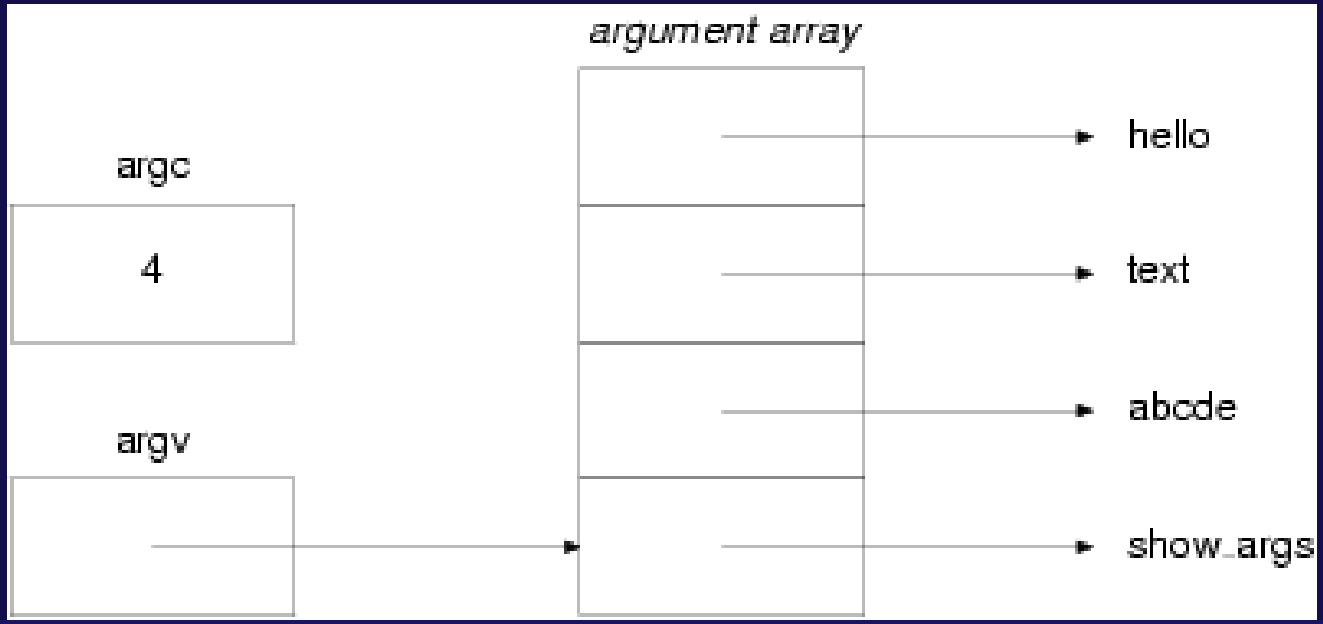


```c
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("%d \n",argc);
    printf("%s \n",argv[argc]);
    printf("%s \n",argv[argc-1]);
    printf("%s \n",argv[0]);
    return 0;
}
```

# Arguments to main

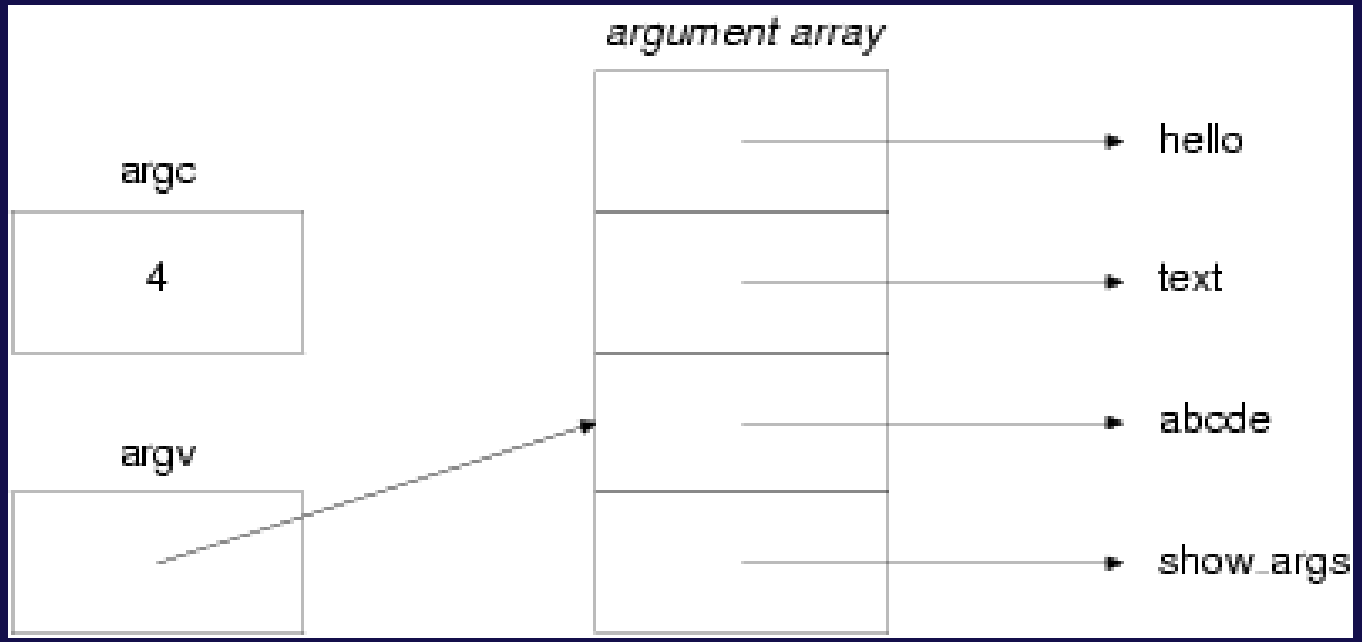
If the program name is show_args and it has arguments abcde, text, and hello

when it is run, the state of the arguments and the value of argv can be illustrated like this:

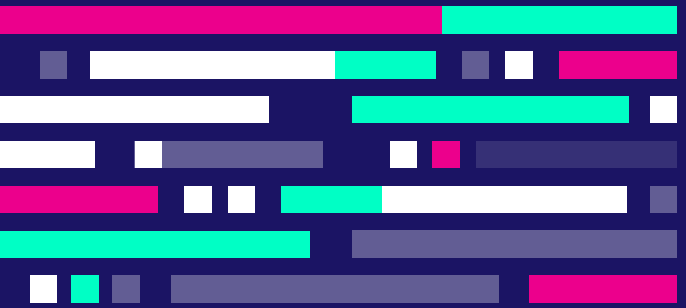


# Arguments to main

Each time that argv is incremented, it is stepped one item further along the array of arguments. Thus after the first iteration of the loop, argv will point to the pointer which in turn points to the abcde argument.



# Arguments to main



```
$ ./show_args abcde text hello

the output is :

./show_args
abcde
text
hello
```

# How to use __attribute__((unused)) or (void) to compile functions with unused variables or parameters

Declaring variables or parameters as __attribute__((unused)) or using (void) with them is useful in situations where you want to avoid warnings from the compiler about unused variables or parameters.

```c
#include <stdio.h>

void foo(int x) {
  printf("Hello from foo()\n");
}

int main() {
  int x = 10;
  foo(x);
  return 0;
}
```

```
unused.c: In function 'main':
unused.c:7:6: warning: unused variable 'x' [-Wunused-variable]
   int x = 10;
       ^
```

# How to use __attribute__((unused)) or (void) to compile functions with unused variables or parameters

Declaring variables or parameters as __attribute__((unused)) or using (void) with them is useful in situations where you want to avoid warnings from the compiler about unused variables or parameters.

Solution:

```
#include <stdio.h>

void foo(int x __attribute__((unused))) {
  printf("Hello from foo()\n");
}

int main() {
  int x = 10;
  foo(x);
  return 0;
}
```

# How to use __attribute__((unused)) or (void) to compile functions with unused variables or parameters

Declaring variables or parameters as __attribute__((unused)) or using (void) with them is useful in situations where you want to avoid warnings from the compiler about unused variables or parameters.

Solution:

```
#include <stdio.h>

void foo(int x) {
  printf("Hello from foo()\n");
  (void)x; // unused parameter
}

int main() {
  int x = 10;
  foo(x);
  return 0;
}
```

# What is the ASCII character set

```
cook@pop-os:~$ ascii -d
   0 NUL    16 DLE    32      48 0    64 @    80 P    96 `   112 p
   1 SOH    17 DC1    33 !    49 1    65 A    81 Q    97 a   113 q
   2 STX    18 DC2    34 "    50 2    66 B    82 R    98 b   114 r
   3 ETX    19 DC3    35 #    51 3    67 C    83 S    99 c   115 s
   4 EOT    20 DC4    36 $    52 4    68 D    84 T   100 d   116 t
   5 ENQ    21 NAK    37 %    53 5    69 E    85 U   101 e   117 u
   6 ACK    22 SYN    38 &    54 6    70 F    86 V   102 f   118 v
   7 BEL    23 ETB    39 '    55 7    71 G    87 W   103 g   119 w
   8 BS     24 CAN    40 (    56 8    72 H    88 X   104 h   120 x
   9 HT     25 EM     41 )    57 9    73 I    89 Y   105 i   121 y
  10 LF     26 SUB    42 *    58 :    74 J    90 Z   106 j   122 z
  11 VT     27 ESC    43 +    59 ;    75 K    91 [   107 k   123 {
  12 FF     28 FS     44 ,    60 <    76 L    92 \   108 l   124 |
  13 CR     29 GS     45 -    61 =    77 M    93 ]   109 m   125 }
  14 SO     30 RS     46 .    62 >    78 N    94 ^   110 n   126 ~
  15 SI     31 US     47 /    63 ?    79 O    95 _   111 o   127 DEL
```

# Hexadecimal Numbering System

| Decimal | Binary | Hexadecimal |
|---|---|---|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |

# 04

# Hands on lab Practice


Have a Question
Leave a Comment!

## Share

To let the others know more

## Subscribe

To stay updated with latest videos

# Thanks