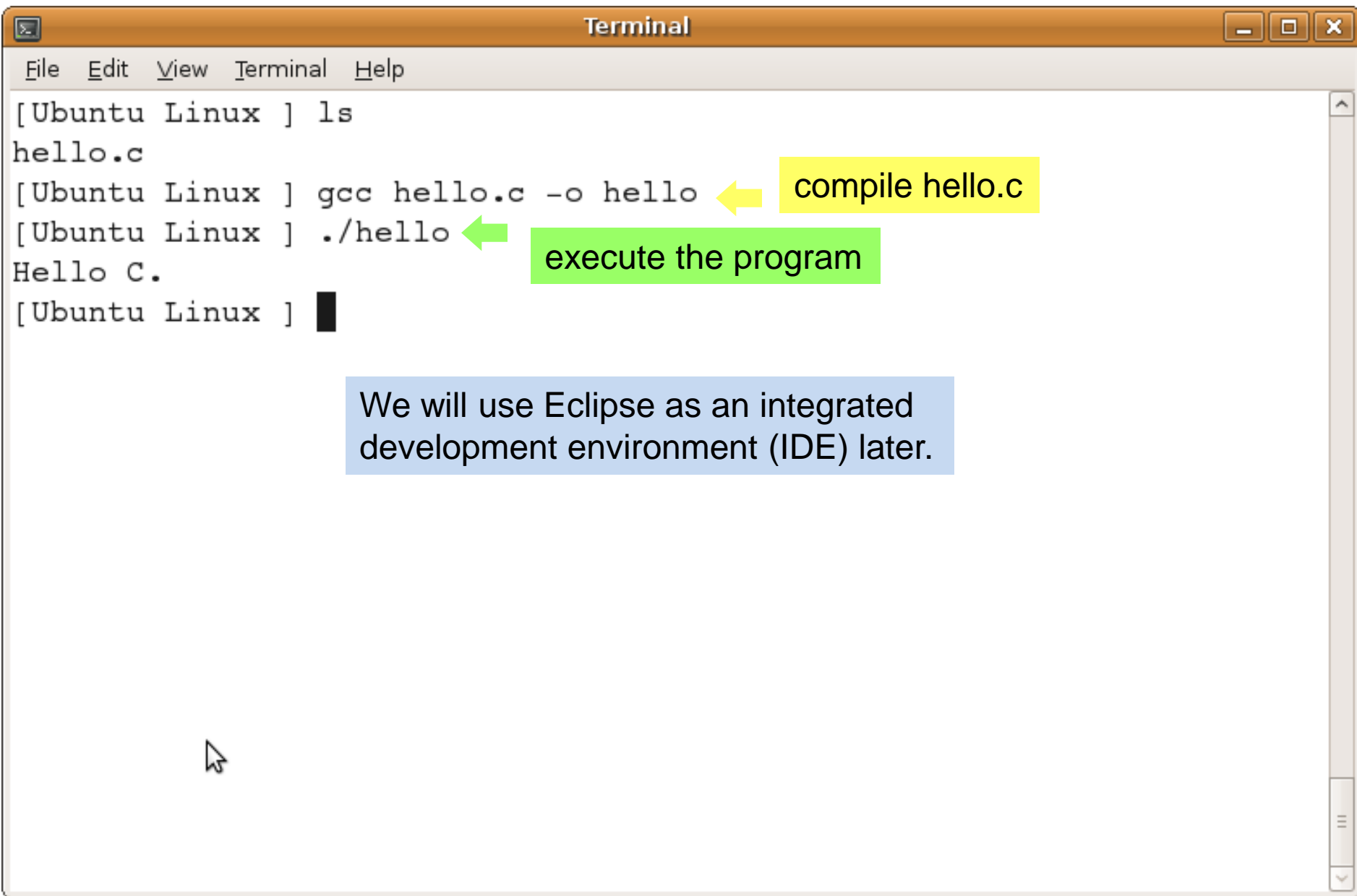


# **Simple Examples of C Programs**

**Yung-Hsiang Lu**

# hello.c





A terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Help) and standard window controls. The terminal shows the following sequence of commands and output:

```
[Ubuntu Linux ] ls  
hello.c  
[Ubuntu Linux ] gcc hello.c -o hello  
[Ubuntu Linux ] ./hello  
Hello C.  
[Ubuntu Linux ]
```

Annotations with arrows point to specific parts of the terminal output:

- A yellow box labeled "compile hello.c" with an arrow pointing to the command `gcc hello.c -o hello`.
- A green box labeled "execute the program" with an arrow pointing to the command `./hello`.

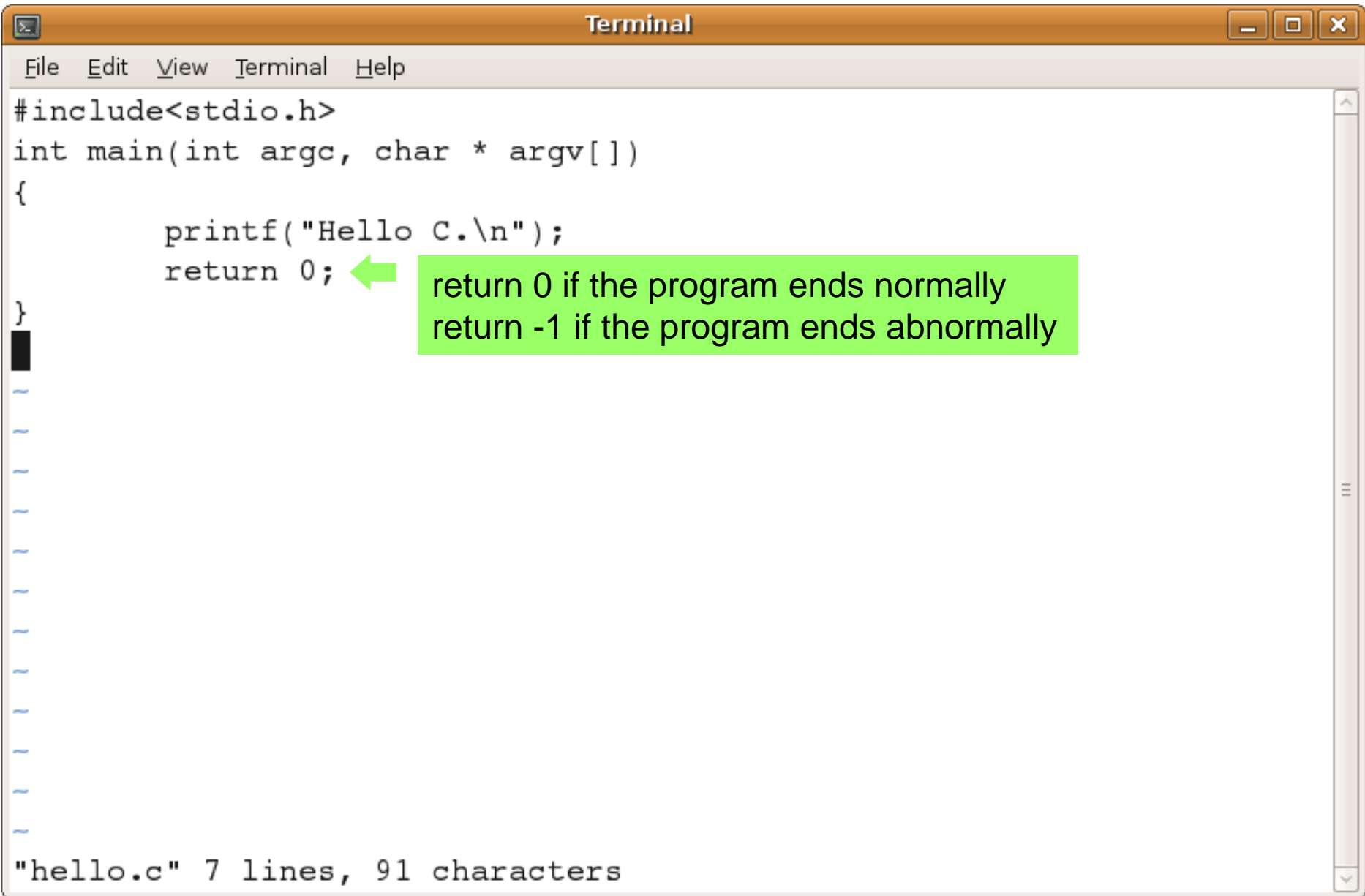
A blue box containing the text "We will use Eclipse as an integrated development environment (IDE) later." is positioned below the terminal output.

```
#include<stdio.h>
int main(int argc, char * argv[])
{
    printf("Hello C.\n");
    return 0;
}
```

print "Hello C."  
\n means new line  
printf is a library function

C programs are case-sensitive.  
Printf, printf, printF are **different**.

"hello.c" 7 lines, 91 characters



A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", and "Help". The main area contains a C program snippet:

```
#include<stdio.h>
int main(int argc, char * argv[])
{
    printf("Hello C.\n");
    return 0;
}
```

A green callout box with a green arrow pointing to the `return 0;` line contains the text:

return 0 if the program ends normally  
return -1 if the program ends abnormally

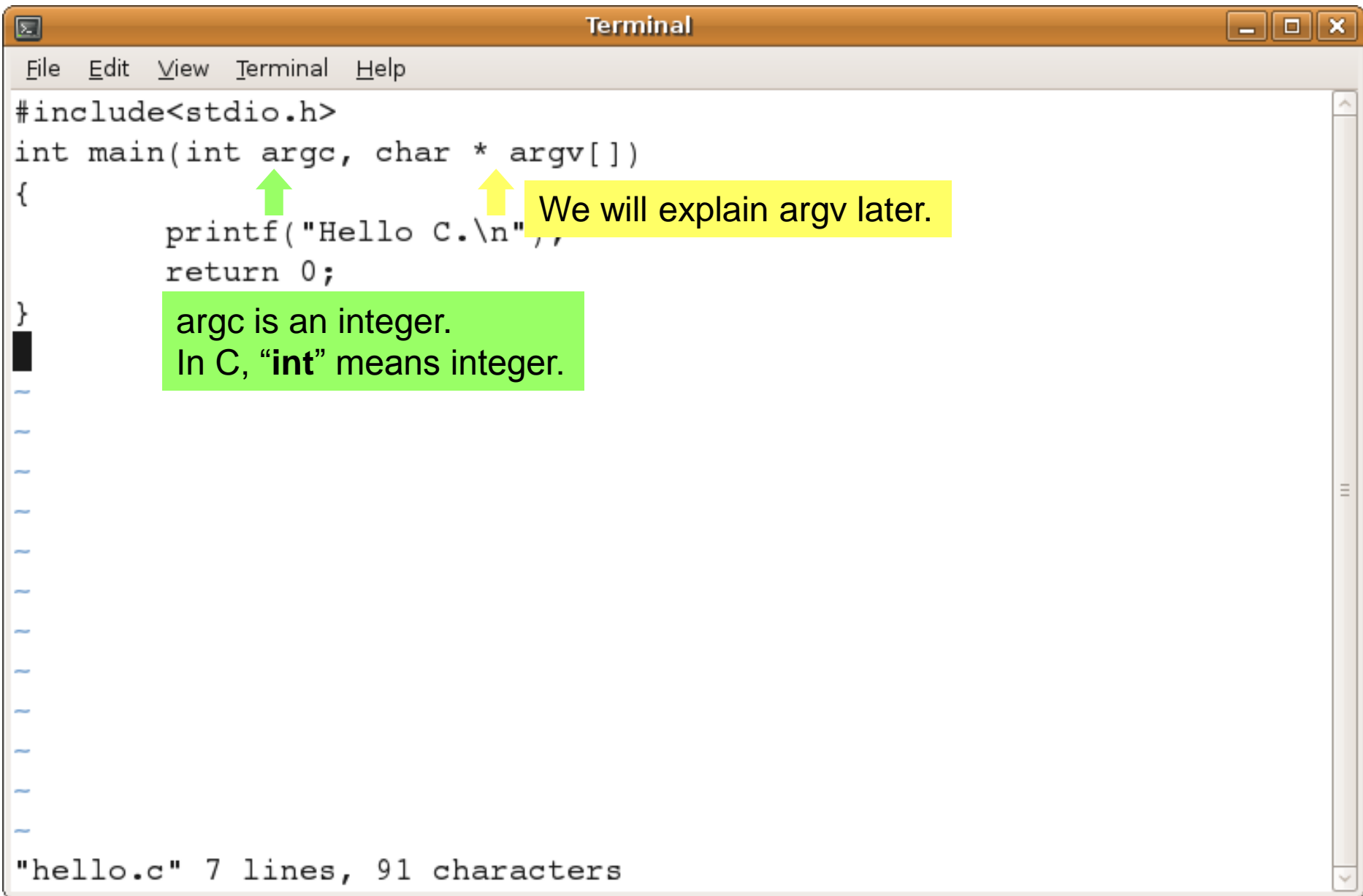
Below the code, the terminal shows the file statistics:

```
"hello.c" 7 lines, 91 characters
```

```
Terminal
File Edit View Terminal Help
#include<stdio.h>
int main(int argc, char * argv[])
{
    printf("Hello C.\n");
    return 0;
}
```

**"main" is a special function and  
it is the starting point of the program**

"hello.c" 7 lines, 91 characters



A terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Help). The code inside is a C program for "hello.c". Annotations include a green arrow pointing to 'argc' with a green box stating "argc is an integer. In C, 'int' means integer." and a yellow arrow pointing to 'argv' with a yellow box stating "We will explain argv later." The status bar at the bottom shows "hello.c" 7 lines, 91 characters.

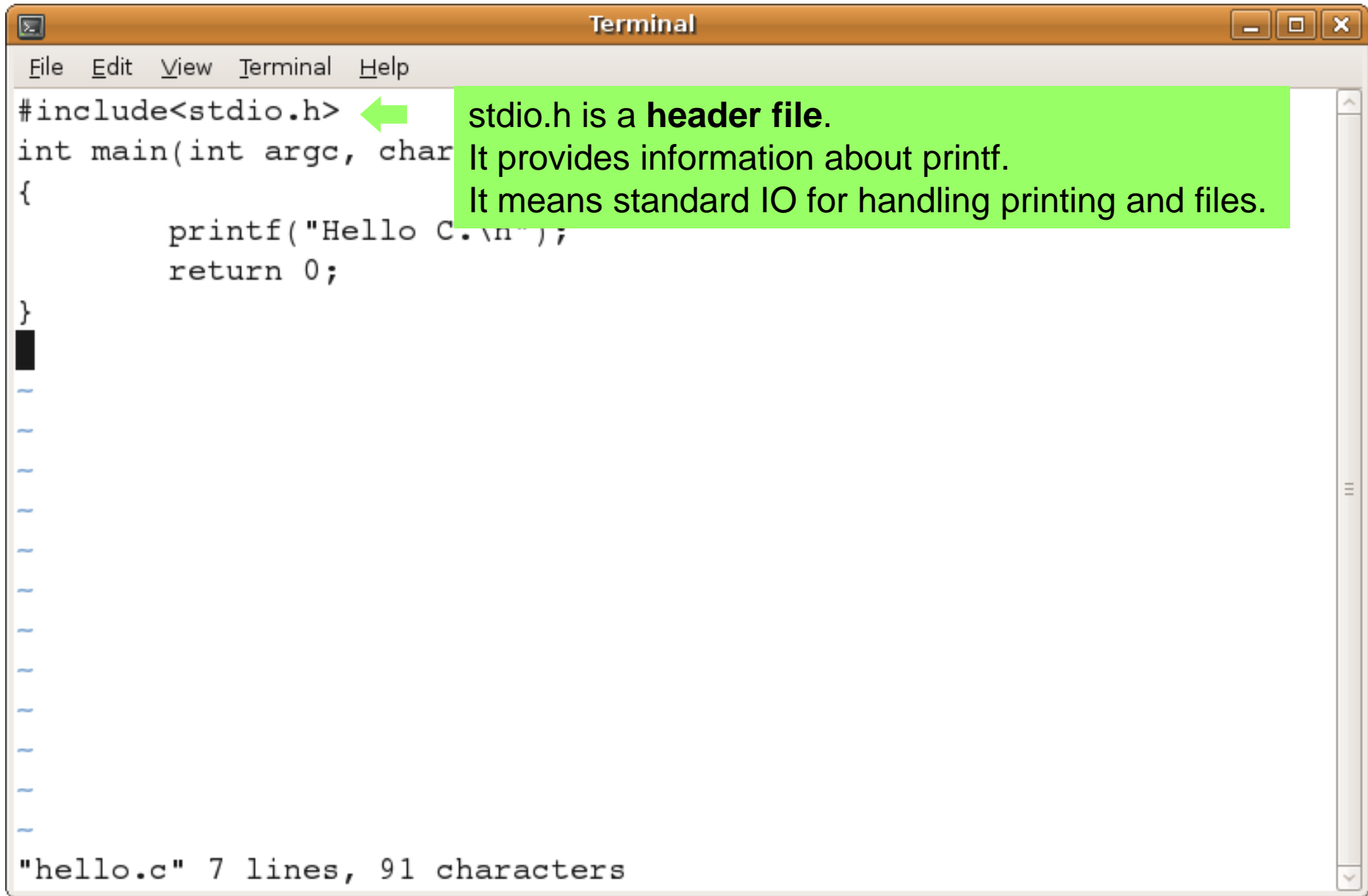
```
#include<stdio.h>
int main(int argc, char * argv[])
{
    printf("Hello C.\n");
    return 0;
}
```

argc is an integer.  
In C, "int" means integer.

We will explain argv later.

"hello.c" 7 lines, 91 characters





The image shows a terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Help). The terminal contains the following C code:

```
#include<stdio.h>
int main(int argc, char
{
    printf("Hello C.\n");
    return 0;
}
```

A green callout box points to the `#include<stdio.h>` line with the text:

**stdio.h is a header file.**  
It provides information about printf.  
It means standard IO for handling printing and files.

At the bottom of the terminal, it says: `"hello.c" 7 lines, 91 characters`

# How to Write (Simple) C Programs?

- Use the main function as the starting point.
- Select appropriate library functions (such as printf).
  - ⇒ How to know which library functions are available?  
It is similar to learning vocabulary in English.  
**The more you know, the better.**
- Include necessary header files (such as stdio.h).
  - ⇒ This can be found by using manual (man) pages.
- Return 0 if the program ends normally.

man printf(3) - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://www.manpagez.com/man/3/printf/ Google

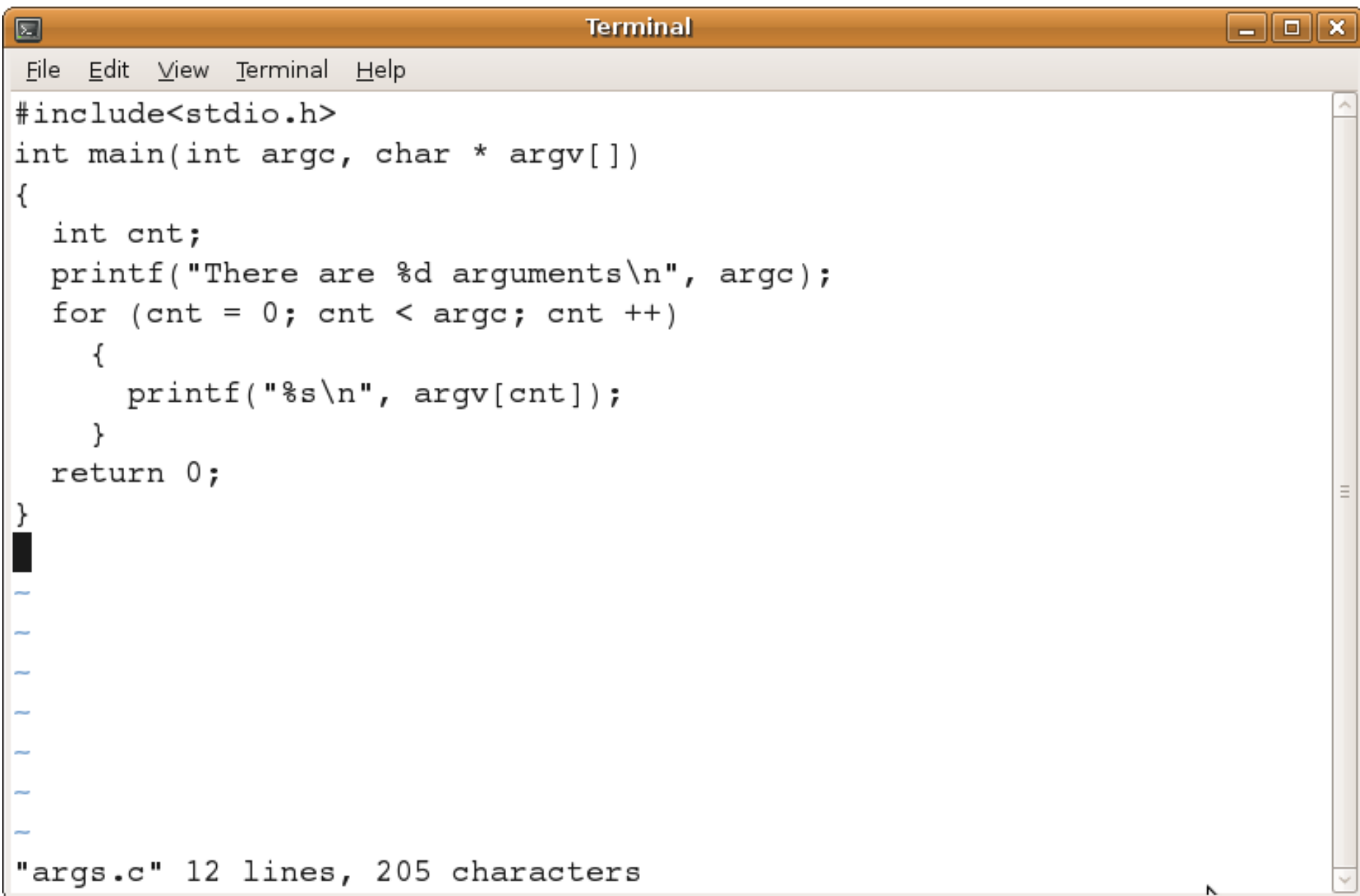
## LIBRARY

Standard C Library (libc, -lc)

## SYNOPSIS

```
#include <stdio.h>
int
asprintf(char **ret, const char *format, ...);
int
fprintf(FILE *restrict stream, const char *restrict format, ...);
int
printf(const char *restrict format, ...);
int
snprintf(char *restrict s, size_t n, const char *restrict format, ...);
int
sprintf(char *restrict s, const char *restrict format, ...);
#include <stdarg.h>
#include <stdio.h>
int
```

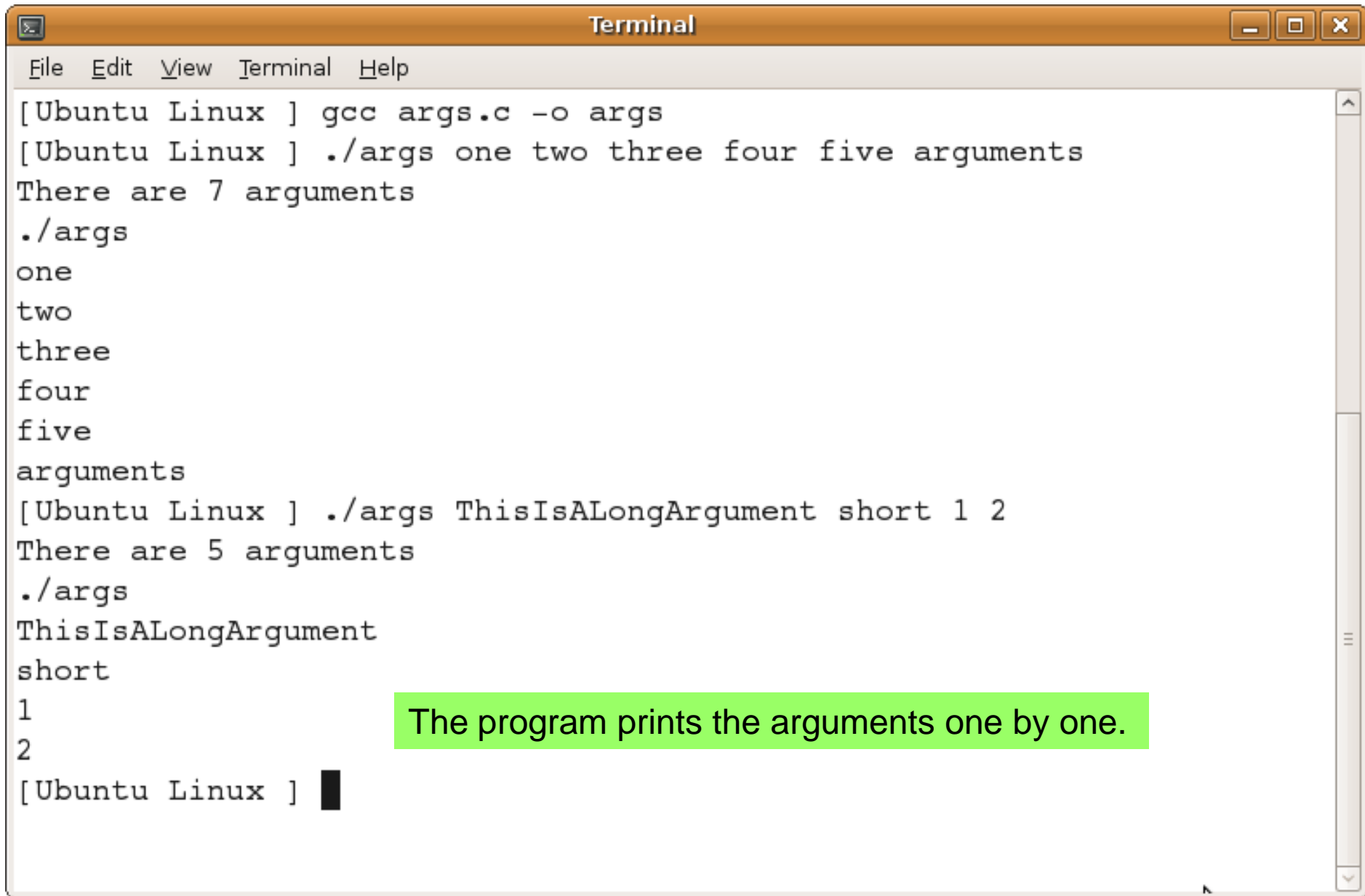
# args.c



A terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Help) and standard window controls. The window contains C code for counting command-line arguments. The code is as follows:

```
#include<stdio.h>
int main(int argc, char * argv[])
{
    int cnt;
    printf("There are %d arguments\n", argc);
    for (cnt = 0; cnt < argc; cnt ++)
    {
        printf("%s\n", argv[cnt]);
    }
    return 0;
}
```

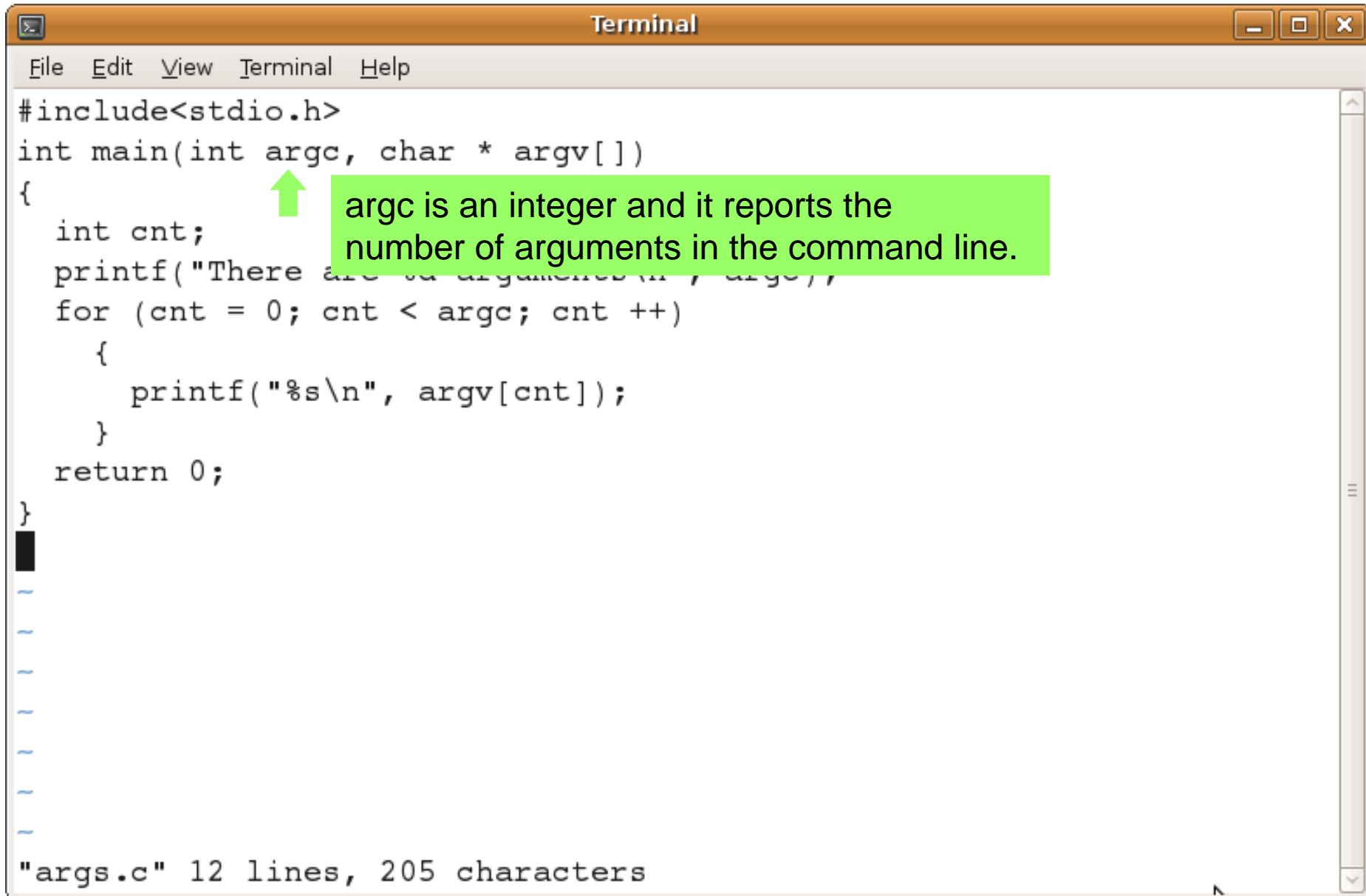
Below the code, there are several blue tilde (~) characters representing command prompts. At the bottom of the terminal, a status bar displays the text: "args.c" 12 lines, 205 characters.



A terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Help) and standard window controls. The terminal shows the compilation and execution of a program named 'args.c'. The first execution uses seven arguments, and the second uses five arguments, including a long one. The program prints each argument on a new line. A green text box highlights the output of the first execution.

```
[Ubuntu Linux ] gcc args.c -o args
[Ubuntu Linux ] ./args one two three four five arguments
There are 7 arguments
./args
one
two
three
four
five
arguments
[Ubuntu Linux ] ./args ThisIsALongArgument short 1 2
There are 5 arguments
./args
ThisIsALongArgument
short
1
2
[Ubuntu Linux ]
```

The program prints the arguments one by one.





A terminal window titled "Terminal" with a menu bar (File, Edit, View, Terminal, Help). The window contains C code for a program that counts command-line arguments. A green arrow points to the variable `argc` in the function signature, and a green box highlights an explanatory text.

```
#include<stdio.h>
int main(int argc, char * argv[])
{
    int cnt;
    printf("There are %d arguments\n", argc);
    for (cnt = 0; cnt < argc; cnt++)
    {
        printf("%s\n", argv[cnt]);
    }
    return 0;
}
```

"args.c" 12 lines, 205 characters

argc is an integer and it reports the number of arguments in the command line.

```
#include<stdio.h>
int main(int argc, char * argv[])
{
    int cnt;
    printf("There are %d arguments\n", argc);
    for (cnt = 0; cnt < argc; cnt++)
    {
        printf("%s\n", argv[cnt]);
    }
    return 0;
}
```



**printf ... %d ... prints an integer.**

"args.c" 12 lines, 205 characters



```
Terminal
File Edit View Terminal Help
[Ubuntu Linux ] gc 1 2 3 4 5 6 7
[Ubuntu Linux ] ./args one two three four five arguments
There are 7 arguments
./args
one
two
three
four
five
arguments 1 2 3 4 5
[Ubuntu Linux ] ./args ThisIsALongArgument short 1 2
There are 5 arguments
./args
ThisIsALongArgument
short
1
2
[Ubuntu Linux ]
```

`./args ThisIsAnArgument 1 2 AB`



One (or more) space to separate arguments

`argc` `5`



The index starts from **zero**, not one.

<code>argv[0]</code>	<code>./args</code>
<code>argv[1]</code>	<code>ThisIsAnArgument</code>
<code>argv[2]</code>	<code>1</code>
<code>argv[3]</code>	<code>2</code>
<code>argv[4]</code>	<code>AB</code>

The operating system (Linux) assigns the values of `argc` and `argv`.  
(To be more precise, it is the shell that assigns the values.)

```
#include<stdio.h>
int main(int argc, char * argv[])
{
    int cnt;
    printf("There are %d arguments\n", argc);
    for (cnt = 0; cnt < argc; cnt++)
    {
        printf("%s\n", argv[cnt]);
    }
    return 0;
}
```

← an integer counter

← cnt starts at zero,  
increases by one each time  
until it is no longer smaller than argc

↑ printf ... %s ... prints a string.

"args.c" 12 lines, 205 characters

```
#include<stdio.h>
int main(int argc, char * argv[])
{
    int cnt;
    printf("There are %d arguments\n", argc);
    for (cnt = 0; cnt < argc; cnt++)
    {
        printf("%s\n", argv[cnt]);
    }
    return 0;
}
```

argc and argv are the conventional names. It is better to keep them.

You can change the names of these **variables**.

"args.c" 12 lines, 205 characters

# **Congratulations.**

# What flag is used to specify the name of gcc's output file?

- ☐ A) -a
- ☐ B) -c
- ☒ C) -o
- ☐ D) -g

Correct - Click anywhere to continue

Incorrect - Click anywhere to continue

Your answer:

You did not answer this question

You must answer the question before continuing

Submit

Clear

# Starting point of a program?

The  function is the starting point of a program.

Correct - Click anywhere to continue

Incorrect - Click anywhere to continue

Your answer:

You did not answer this question

You must answer the question before continuing

Submit

Clear

# In C programs, indexes starting from

- ☐ A) -1.
- ☒ B) 0.
- ☐ C) 1.
- ☐ D) Specified by programmers.

Correct - Click anywhere to continue

Incorrect - Click anywhere to continue

Your answer:

You did not answer this question

You must answer the question before continuing

Submit

Clear



# Simple C Examples

<b>Your Score</b>	{score}
<b>Max Score</b>	{max-score}
<b>Number of Quiz Attempts</b>	{total-attempts}

Question Feedback/Review Information Will Appear Here

Continue

Review Quiz