



UNLaM

Dto. Ingeniería e Investigaciones Tecnológicas

Argumentos a main

Programación 1110

Objetivo

Comprender la utilidad de los argumentos en la llamada a una aplicación y poder implementar la solución en lenguaje C.

Introducción

Cuando hablamos de “Argumentos a main” o “Argumentos en la línea de comandos” nos referimos a la capacidad de enviar parámetros de inicio a la ejecución de un programa. El objetivo de esta capacidad es poder hacer nuestros programas más versátiles, que sea posible establecer criterios de operación, archivos sobre los que trabajar, modalidades u opciones en general.

En nuestro caso utilizaremos la línea de comandos de nuestro sistema operativo o las facilidades de nuestro IDE de desarrollo para enviar información que condicione la ejecución del programa. Es importante aclarar que la funcionalidad no se circunscribe a este escenario, los argumentos al main son perfectamente válidos y útiles en una llamada entre programas. Veremos un ejemplo de esto oportunamente.

En C, el primer paso para que nuestro programa acepte argumentos es, si nuestro entorno no lo hace automáticamente, declarar los parámetros en el main como indica la **Figura 1**:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[])
5  {
6      //
7      return 0;
8  }
9
```

Figura 1 - Argumentos

Los parámetros se denominan por convención **argv** y **argc** aunque esto no es obligatorio claro. **argc** determina la cantidad de argumentos almacenados en **argv**, de esto deducimos que la cantidad de parámetros es variable. El argumento **argv** es un array de punteros a cadenas de caracteres (Puedes encontrarlo declarado como

`char **argv`), la cantidad de cadenas dependerá de los parámetros pasados al programa en su ejecución y `argc` nos ayuda a conocer este número.

Para que quede más claro, desarrollemos un programa, al que llamaremos “Copiador” que por ahora solo nos muestre los parámetros pasados a nuestra aplicación. La **Figura 2** nos da una aproximación al código:

```
#include <stdio.h>
#include <stdlib.h>

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

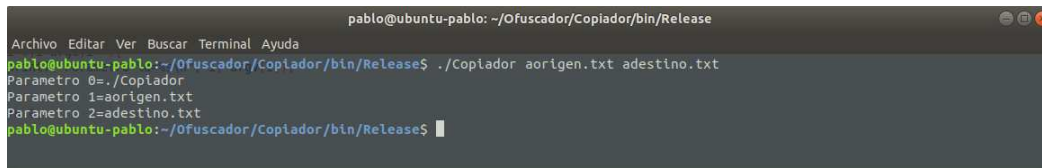
    for(i=0;i<argc;i++){
        printf("Parametro %d=%s\n", i, argv[i]);
    }

    return 0;
}
```

Figura 2 – Se muestra todos los argumentos

El programa itera en un ciclo `for`, definido por `argc` y muestra todos los parámetros recibidos.

Para probar nuestro programa, nos posicionamos con el shell de línea de comandos de nuestro sistema operativo en la ubicación física del archivo ejecutable y procedemos a realizar la llamada como muestra la **Figura 3**.



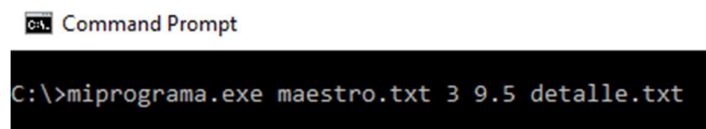
```
pablo@ubuntu-pablo: ~/Ofuscador/Copiador/bin/Release
Archivo Editar Ver Buscar Terminal Ayuda
pablo@ubuntu-pablo:~/Ofuscador/Copiador/bin/Release$ ./Copiador aorigen.txt adestino.txt
Parametro 0=./Copiador
Parametro 1=aorigen.txt
Parametro 2=adestino.txt
pablo@ubuntu-pablo:~/Ofuscador/Copiador/bin/Release$
```

Figura 3 – Ejecución por consola/shell

Como muestra la **Figura 3**, a pesar de escribir 2 parámetros (`aorigen.txt` y `adestino.txt`) el programa finalmente muestra 3 parámetros recibidos. Es claro que se agrega un parámetro más, y este parámetro es siempre el nombre (y en algunos casos la ruta completa) del archivo ejecutado.

Como habrás observado, puedes pasar tantos parámetros como quieras, los límites no están explícitamente definidos y dependen de la implementación particular del compilador y las limitaciones de plataforma, en cualquier caso, para aplicaciones prácticas encontraras que los límites exceden las necesidades.

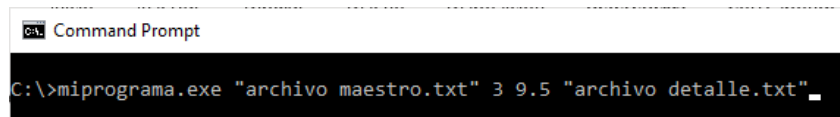
Los parámetros pueden ser de cualquier tipo ya que en definitiva siempre serán una cadena de texto, la línea de comandos con argumentos al `main` de la **Figura 4**, combinando enteros, flotantes y cadenas de texto es perfectamente válida



```
C:\>miprograma.exe maestro.txt 3 9.5 detalle.txt
```

Figura 4 - Argumentos de distintos tipos

Otro aspecto para tener en cuenta es la forma de operar cuando mi cadena de texto parámetro tiene espacios. Imaginemos que los archivos de la **Figura 4** en realidad se llaman “archivo maestro.txt” y “archivo detalle.txt”. El espacio entre archivo y maestro o entre archivo y detalle generará parámetros adicionales, tomando a “archivo” como un parámetro y a “maestro.txt” como otro. Para evitar esta situación en la línea de comandos la cadena debe ser encerrada entre “”, en este caso será tomada como un único parámetro. La **Figura 5** ilustra la situación.



```
C:\>miprograma.exe "archivo maestro.txt" 3 9.5 "archivo detalle.txt"
```

Figura 5 - Parámetros cadena con espacios

Aplicación práctica

Para mostrar un caso de aplicación, si ya has cubierto las unidades de archivos, desarrollemos programa que copie archivos de manera binaria, el primer parámetro será el archivo origen, es decir el archivo a copiar y el segundo parámetro será el archivo destino, que en caso de existir será remplazado (**Figura 6**).

```

#include <stdio.h>
#include <stdlib.h>
#define PARAMETROS_TOTAL 3
#define PARAMETRO_EJECUTABLE 0
#define PARAMETRO_ORIGEN 1
#define PARAMETRO_DESTINO 2
int main(int argc, char *argv[])
{
    FILE* fo;
    FILE* fd;
    char buff;

    if(argc!=PARAMETROS_TOTAL){
        printf("La cantidad de parametros no es la esperada, se aborta proceso");
        return 1;
    }
    printf("Ejecutando %s \n", argv[PARAMETRO_EJECUTABLE]);
    //Selecciono el primero ya que el parametro 0 es el ejecutable
    //Archivo origen, se abre solo para lectura
    fo = fopen(argv[PARAMETRO_ORIGEN], "rb");
    if(!fo){
        printf("Ocurrio un problema al intentar abrir archivo %s", argv[PARAMETRO_ORIGEN]);
        return 1;
    }
    //Archivo destino se abre para escritura
    fd = fopen(argv[PARAMETRO_DESTINO], "wb");
    if(!fd){
        printf("Ocurrio un problema al intentar crear/sobreescribir archivo %s", argv[PARAMETRO_DESTINO]);
        return 1;
    }

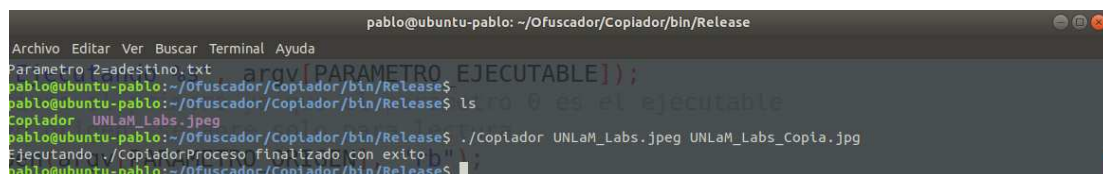
    fread(&buff, sizeof(buff), 1, fo);
    while(!feof(fo)){
        fwrite(&buff, sizeof(buff), 1, fd);
        fread(&buff, sizeof(buff), 1, fo);
    }

    fclose(fo);
    fclose(fd);
    puts("Proceso finalizado con exito");
    return 0;
}

```

Figura 6 - Programa de copia

Para probar el ejemplo utilizaremos un archivo de imagen de los laboratorios de la UNLaM denominado UNLaM_Labs.jpeg. Puedes conseguir tu propia imagen para realizar la prueba. La Figura 7 muestra la llamada del programa mediante el Shell linux.



```

pablo@ubuntu-pablo: ~/Ofuscador/Copiador/bin/Release
Archivo Editar Ver Buscar Terminal Ayuda
Parametro 2=adestino.txt
pablo@ubuntu-pablo:~/Ofuscador/Copiador/bin/Release$ ls
pablo@ubuntu-pablo:~/Ofuscador/Copiador/bin/Release$ ls
Copiador UNLaM_Labs.jpeg
pablo@ubuntu-pablo:~/Ofuscador/Copiador/bin/Release$ ./Copiador UNLaM_Labs.jpeg UNLaM_Labs_Copia.jpg
Ejecutando ./CopiadorProceso finalizado con exito
pablo@ubuntu-pablo:~/Ofuscador/Copiador/bin/Release$

```

Figura 7 - Programa de copia, test por consola

La **Figura 8** muestra el resultado de la ejecución del programa con los archivos efectivamente copiados.

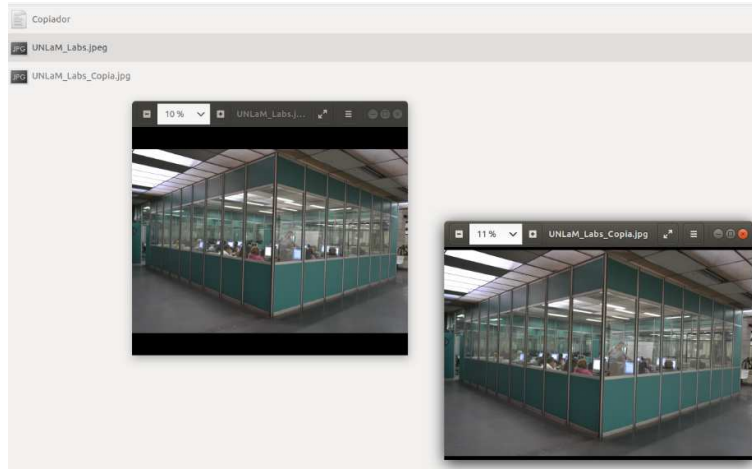


Figura 8 - Archivos copiados

Si utilizas code::blocks como entorno de desarrollo puedes probar los argumentos al main de una forma más cómoda mediante la opción en el menu project->set programs' argument... (Figura 9 y Figura 10).

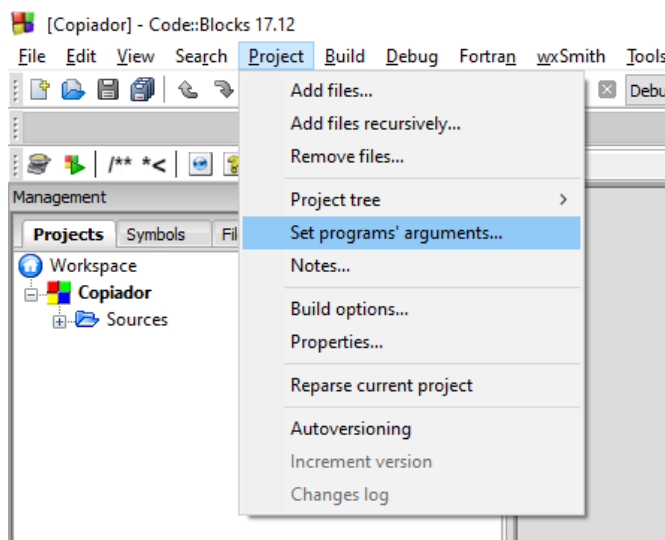


Figura 9 - Menú configuración de argumentos al main

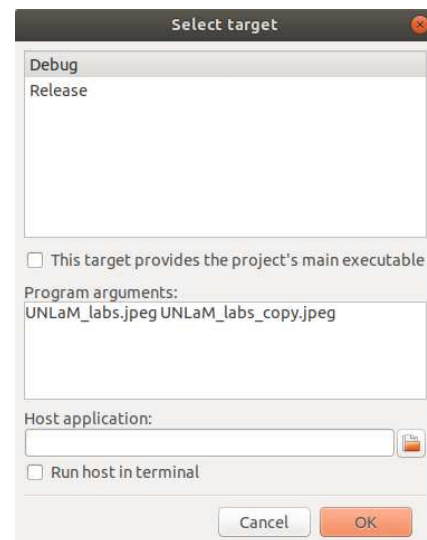


Figura 10 - Configuración de argumentos

Otros entornos de desarrollo disfrutan de la misma característica. Si optas por esta opción y no envías una ruta absoluta del archivo debes tener en cuenta que el IDE forzara como carpeta actual a la carpeta del proyecto y no a la carpeta donde se encuentra el ejecutable.

Ejecución desde terceros programas

Como se anticipó, la capacidad de enviar parámetros a una aplicación no se circunscribe a la ejecución del programa por un Shell. Es posible ejecutar nuestros programas C parametrizados desde aplicaciones de terceros. El siguiente fragmento de código Python hace uso de nuestro “Copiador.exe” sin que el programa sea explícitamente ejecutado desde el shell. El ultimo parámetro “shell=False” ayuda a que el proceso no despliegue una consola.

```
1 import subprocess
2 subprocess.call(["C:\\temp\\Copiador.exe", "UNLaM_labs.jpeg", "UNLaM_labs_copia.jpeg"], shell=False)
```

Para investigar

Si utilizas code::blocks y no lo has notado aun, observa que el ejecutable de tu programa se genera en una subcarpeta Debug o Release dentro de la carpeta bin, esto es dependiente de la opción de compilación activa que se muestra en la Figura 11. ¿Entiendes la diferencia y el significado de estas opciones?



Figura 11 - Compilacion Debug vs Release

Recursos adicionales

En el siguiente enlace encontrarás un video explicando estos conceptos directamente sobre la herramienta de desarrollo (**code::blocks**):

Link: <https://youtu.be/rI0q-hNovzo>

Actividades propuestas

1. Desarrolla, o modifica si ya lo has desarrollado, el ejercicio 2 de la practica 1 de la guía de trabajos prácticos, tal que m y n sean argumentos pasados por la línea de comando. El resultado deberá ser mostrado por consola.
2. Desarrolla, o modifica si ya lo has desarrollado, el ejercicio 6 de la practica 1 de la guía de trabajos prácticos, tal que X y TOL sean argumentos pasados por la línea de comando. Agrega un parámetro opcional que indique si X esta expresado en grados sexagesimales o radianes, en caso de no estar este parámetro se tomara en radianes. El resultado deberá ser mostrado por consola.
3. Escriba un programa que retorne cuantas veces se encuentra un sub-cadena dentro de una cadena. La cadena y la sub-cadena serán pasadas por línea de comando y el resultado será expresado por la salida de consola. Verifique probando con cadenas que tengas espacios.

Tabla de figuras

Figura 1 - Argumentos.....	1
Figura 2 – Se muestra todos los argumentos.....	2
Figura 3 – Ejecución por consola/shell.....	2
Figura 4 - Argumentos de distintos tipos.....	3
Figura 5 - Parámetros cadena con espacios.....	3
Figura 6 - Programa de copia.....	4
Figura 7 - Programa de copia, test por consola	4
Figura 8 - Archivos copiados.....	5
Figura 9 - Menú configuración de argumentos al main	5
Figura 10 - Configuración de argumentos.....	5
Figura 11 - Compilacion Debug vs Release.....	6

Referencias

- [1] B. W. Kernighan y D. M. Ritchie, El lenguaje de programación C, Pearson Educación, 1991.
- [2] H. M. Deitel y P. J. Deitel, Cómo programar en C/C+, Pearson Educación, 1995.