0x15. C - File I/O - 100% Complete Step by step

**0x15. C - File I/O**

**README.md**

**Project**

**main.h**

**#ifndef MAIN\_H**

**#define MAIN\_H**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**#include <unistd.h>**

**ssize\_t read\_textfile(const char \*filename, size\_t letters);**

**int create\_file(const char \*filename, char \*text\_content);**

**int append\_text\_to\_file(const char \*filename, char \*text\_content);**

**#endif**

**0-read\_textfile.c**

**#include "main.h"**

**#include <stdlib.h>**

**/\*\***

**\* read\_textfile- Read text file print to STDOUT.**

**\* @filename: text file being read**

**\* @letters: number of letters to be read**

**\* Return: w- actual number of bytes read and printed**

**\*        0 when function fails or filename is NULL.**

**\*/**

**ssize\_t read\_textfile(const char \*filename, size\_t letters)**

**{**

**char \*buf;**

**ssize\_t fd;**

**ssize\_t w;**

**ssize\_t t;**

**fd = open(filename, O\_RDONLY);**

**if (fd == -1)**

**return (0);**

**buf = malloc(sizeof(char) \* letters);**

**t = read(fd, buf, letters);**

**w = write(STDOUT\_FILENO, buf, t);**

**free(buf);**

**close(fd);**

**return (w);**

**}**

**1-create\_file.c**

**#include "main.h"**

**/\*\***

**\* create\_file - Creates a file.**

**\* @filename: A pointer to the name of the file to create.**

**\* @text\_content: A pointer to a string to write to the file.**

**\***

**\* Return: If the function fails - -1.**

**\*         Otherwise - 1.**

**\*/**

**int create\_file(const char \*filename, char \*text\_content)**

**{**

**int fd, w, len = 0;**

**if (filename == NULL)**

**return (-1);**

**if (text\_content != NULL)**

**{**

**for (len = 0; text\_content[len];)**

**len++;**

**}**

**fd = open(filename, O\_CREAT | O\_RDWR | O\_TRUNC, 0600);**

**w = write(fd, text\_content, len);**

**if (fd == -1 || w == -1)**

**return (-1);**

**close(fd);**

**return (1);**

**}**

**2-append\_text\_to\_file.c**

**#include "main.h"**

**/\*\***

**\* append\_text\_to\_file - Appends text at the end of a file.**

**\* @filename: A pointer to the name of the file.**

**\* @text\_content: The string to add to the end of the file.**

**\***

**\* Return: If the function fails or filename is NULL - -1.**

**\*         If the file does not exist the user lacks write permissions - -1.**

**\*         Otherwise - 1.**

**\*/**

**int append\_text\_to\_file(const char \*filename, char \*text\_content)**

**{**

**int o, w, len = 0;**

**if (filename == NULL)**

**return (-1);**

**if (text\_content != NULL)**

**{**

**for (len = 0; text\_content[len];)**

**len++;**

**}**

**o = open(filename, O\_WRONLY | O\_APPEND);**

**w = write(o, text\_content, len);**

**if (o == -1 || w == -1)**

**return (-1);**

**close(o);**

**return (1);**

**}**

**3-cp.c**

**#include "main.h"**

**#include <stdio.h>**

**#include <stdlib.h>**

**char \*create\_buffer(char \*file);**

**void close\_file(int fd);**

**/\*\***

**\* creates\_buffer - Allocates 1024 bytes for a buffer.**

**\* @file: The name of the file buffer is storing chars for.**

**\***

**\* Return: A pointer to the newly-allocated buffer.**

**\*/**

**char \*creates\_buffer(char \*file)**

**{**

**char \*buffer;**

**buffer = malloc(sizeof(char) \* 1024);**

**if (buffer == NULL)**

**{**

**dprintf(STDERR\_FILENO,**

**"Error: Can't write to %s\n", file);**

**exit(99);**

**}**

**return (buffer);**

**}**

**/\*\***

**\* close\_file - Closes file descriptors.**

**\* @fd: The file descriptor to be closed.**

**\*/**

**void close\_file(int fd)**

**{**

**int c;**

**c = close(fd);**

**if (c == -1)**

**{**

**dprintf(STDERR\_FILENO, "Error: Can't close fd %d\n", fd);**

**exit(100);**

**}**

**}**

**/\*\***

**\* main - Copies the contents of a file to another file.**

**\* @argc: The number of arguments supplied to the program.**

**\* @argv: An array of pointers to the arguments.**

**\***

**\* Return: 0 on success.**

**\***

**\* Description: If the argument count is incorrect - exit code 97.**

**\* If file\_from does not exist or cannot be read - exit code 98.**

**\* If file\_to cannot be created or written to - exit code 99.**

**\* If file\_to or file\_from cannot be closed - exit code 100.**

**\*/**

**int main(int argc, char \*argv[])**

**{**

**int from, to, r, w;**

**char \*buffer;**

**if (argc != 3)**

**{**

**dprintf(STDERR\_FILENO, "Usage: cp file\_from file\_to\n");**

**exit(97);**

**}**

**buffer = create\_buffer(argv[2]);**

**from = open(argv[1], O\_RDONLY);**

**r = read(from, buffer, 1024);**

**to = open(argv[2], O\_CREAT | O\_WRONLY | O\_TRUNC, 0664);**

**do {**

**if (from == -1 || r == -1)**

**{**

**dprintf(STDERR\_FILENO,**

**"Error: Can't read from file %s\n", argv[1]);**

**free(buffer);**

**exit(98);**

**}**

**w = write(to, buffer, r);**

**if (to == -1 || w == -1)**

**{**

**dprintf(STDERR\_FILENO,**

**"Error: Can't write to %s\n", argv[2]);**

**free(buffer);**

**exit(99);**

**}**

**r = read(from, buffer, 1024);**

**to = open(argv[2], O\_WRONLY | O\_APPEND);**

**} while (r > 0);**

**free(buffer);**

**close\_file(from);**

**close\_file(to);**

**return (0);**

**}**

**100-elf\_header.c**

**#include <elf.h>**

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**#include <unistd.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**void check\_elf(unsigned char \*e\_ident);**

**void print\_magic(unsigned char \*e\_ident);**

**void print\_class(unsigned char \*e\_ident);**

**void print\_data(unsigned char \*e\_ident);**

**void print\_version(unsigned char \*e\_ident);**

**void print\_abi(unsigned char \*e\_ident);**

**void print\_osabi(unsigned char \*e\_ident);**

**void print\_type(unsigned int e\_type, unsigned char \*e\_ident);**

**void print\_entry(unsigned long int e\_entry, unsigned char \*e\_ident);**

**void close\_elf(int elf);**

**/\*\***

**\* check\_elf - Checks if a file is an ELF file.**

**\* @e\_ident: A pointer to an array containing the ELF magic numbers.**

**\***

**\* Description: If the file is not an ELF file - exit code 98.**

**\*/**

**void check\_elf(unsigned char \*e\_ident)**

**{**

**int index;**

**for (index = 0; index < 4; index++)**

**{**

**if (e\_ident[index] != 127 &&**

**e\_ident[index] != 'E' &&**

**e\_ident[index] != 'L' &&**

**e\_ident[index] != 'F')**

**{**

**dprintf(STDERR\_FILENO, "Error: Not an ELF file\n");**

**exit(98);**

**}**

**}**

**}**

**/\*\***

**\* print\_magic - Prints the magic numbers of an ELF header.**

**\* @e\_ident: A pointer to an array containing the ELF magic numbers.**

**\***

**\* Description: Magic numbers are separated by spaces.**

**\*/**

**void print\_magic(unsigned char \*e\_ident)**

**{**

**int index;**

**printf(" Magic: ");**

**for (index = 0; index < EI\_NIDENT; index++)**

**{**

**printf("%02x", e\_ident[index]);**

**if (index == EI\_NIDENT - 1)**

**printf("\n");**

**else**

**printf(" ");**

**}**

**}**

**/\*\***

**\* print\_class - Prints the class of an ELF header.**

**\* @e\_ident: A pointer to an array containing the ELF class.**

**\*/**

**void print\_class(unsigned char \*e\_ident)**

**{**

**printf(" Class: ");**

**switch (e\_ident[EI\_CLASS])**

**{**

**case ELFCLASSNONE:**

**printf("none\n");**

**break;**

**case ELFCLASS32:**

**printf("ELF32\n");**

**break;**

**case ELFCLASS64:**

**printf("ELF64\n");**

**break;**

**default:**

**printf("<unknown: %x>\n", e\_ident[EI\_CLASS]);**

**}**

**}**

**/\*\***

**\* print\_data - Prints the data of an ELF header.**

**\* @e\_ident: A pointer to an array containing the ELF class.**

**\*/**

**void print\_data(unsigned char \*e\_ident)**

**{**

**printf(" Data: ");**

**switch (e\_ident[EI\_DATA])**

**{**

**case ELFDATANONE:**

**printf("none\n");**

**break;**

**case ELFDATA2LSB:**

**printf("2's complement, little endian\n");**

**break;**

**case ELFDATA2MSB:**

**printf("2's complement, big endian\n");**

**break;**

**default:**

**printf("<unknown: %x>\n", e\_ident[EI\_CLASS]);**

**}**

**}**

**/\*\***

**\*  \* print\_version - Prints the version of an ELF header.**

**\*   \* @e\_ident: A pointer to an array containing the ELF version.**

**\*    \*/**

**void print\_version(unsigned char \*e\_ident)**

**{**

**printf(" Version: %d",**

**e\_ident[EI\_VERSION]);**

**switch (e\_ident[EI\_VERSION])**

**{**

**case EV\_CURRENT:**

**printf(" (current)\n");**

**break;**

**default:**

**printf("\n");**

**break;**

**}**

**}**

**/\*\***

**\* print\_osabi - Prints the OS/ABI of an ELF header.**

**\* @e\_ident: A pointer to an array containing the ELF version.**

**\*/**

**void print\_osabi(unsigned char \*e\_ident)**

**{**

**printf(" OS/ABI: ");**

**switch (e\_ident[EI\_OSABI])**

**{**

**case ELFOSABI\_NONE:**

**printf("UNIX - System V\n");**

**break;**

**case ELFOSABI\_HPUX:**

**printf("UNIX - HP-UX\n");**

**break;**

**case ELFOSABI\_NETBSD:**

**printf("UNIX - NetBSD\n");**

**break;**

**case ELFOSABI\_LINUX:**

**printf("UNIX - Linux\n");**

**break;**

**case ELFOSABI\_SOLARIS:**

**printf("UNIX - Solaris\n");**

**break;**

**case ELFOSABI\_IRIX:**

**printf("UNIX - IRIX\n");**

**break;**

**case ELFOSABI\_FREEBSD:**

**printf("UNIX - FreeBSD\n");**

**break;**

**case ELFOSABI\_TRU64:**

**printf("UNIX - TRU64\n");**

**break;**

**case ELFOSABI\_ARM:**

**printf("ARM\n");**

**break;**

**case ELFOSABI\_STANDALONE:**

**printf("Standalone App\n");**

**break;**

**default:**

**printf("<unknown: %x>\n", e\_ident[EI\_OSABI]);**

**}**

**}**

**/\*\***

**\* print\_abi - Prints the ABI version of an ELF header.**

**\* @e\_ident: A pointer to an array containing the ELF ABI version.**

**\*/**

**void print\_abi(unsigned char \*e\_ident)**

**{**

**printf(" ABI Version: %d\n",**

**e\_ident[EI\_ABIVERSION]);**

**}**

**/\*\***

**\* print\_type - Prints the type of an ELF header.**

**\* @e\_type: The ELF type.**

**\* @e\_ident: A pointer to an array containing the ELF class.**

**\*/**

**void print\_type(unsigned int e\_type, unsigned char \*e\_ident)**

**{**

**if (e\_ident[EI\_DATA] == ELFDATA2MSB)**

**e\_type >>= 8;**

**printf(" Type: ");**

**switch (e\_type)**

**{**

**case ET\_NONE:**

**printf("NONE (None)\n");**

**break;**

**case ET\_REL:**

**printf("REL (Relocatable file)\n");**

**break;**

**case ET\_EXEC:**

**printf("EXEC (Executable file)\n");**

**break;**

**case ET\_DYN:**

**printf("DYN (Shared object file)\n");**

**break;**

**case ET\_CORE:**

**printf("CORE (Core file)\n");**

**break;**

**default:**

**printf("<unknown: %x>\n", e\_type);**

**}**

**}**

**/\*\***

**\* print\_entry - Prints the entry point of an ELF header.**

**\* @e\_entry: The address of the ELF entry point.**

**\* @e\_ident: A pointer to an array containing the ELF class.**

**\*/**

**void print\_entry(unsigned long int e\_entry, unsigned char \*e\_ident)**

**{**

**printf(" Entry point address: ");**

**if (e\_ident[EI\_DATA] == ELFDATA2MSB)**

**{**

**e\_entry = ((e\_entry << 8) & 0xFF00FF00) |**

**((e\_entry >> 8) & 0xFF00FF);**

**e\_entry = (e\_entry << 16) | (e\_entry >> 16);**

**}**

**if (e\_ident[EI\_CLASS] == ELFCLASS32)**

**printf("%#x\n", (unsigned int)e\_entry);**

**else**

**printf("%#lx\n", e\_entry);**

**}**

**/\*\***

**\* close\_elf - Closes an ELF file.**

**\* @elf: The file descriptor of the ELF file.**

**\***

**\* Description: If the file cannot be closed - exit code 98.**

**\*/**

**void close\_elf(int elf)**

**{**

**if (close(elf) == -1)**

**{**

**dprintf(STDERR\_FILENO,**

**"Error: Can't close fd %d\n", elf);**

**exit(98);**

**}**

**}**

**/\*\***

**\* main - Displays the information contained in the**

**\* ELF header at the start of an ELF file.**

**\* @argc: The number of arguments supplied to the program.**

**\* @argv: An array of pointers to the arguments.**

**\***

**\* Return: 0 on success.**

**\***

**\* Description: If the file is not an ELF File or**

**\* the function fails - exit code 98.**

**\*/**

**int main(int \_\_attribute\_\_((\_\_unused\_\_)) argc, char \*argv[])**

**{**

**Elf64\_Ehdr \*header;**

**int o, r;**

**o = open(argv[1], O\_RDONLY);**

**if (o == -1)**

**{**

**dprintf(STDERR\_FILENO, "Error: Can't read file %s\n", argv[1]);**

**exit(98);**

**}**

**header = malloc(sizeof(Elf64\_Ehdr));**

**if (header == NULL)**

**{**

**close\_elf(o);**

**dprintf(STDERR\_FILENO, "Error: Can't read file %s\n", argv[1]);**

**exit(98);**

**}**

**r = read(o, header, sizeof(Elf64\_Ehdr));**

**if (r == -1)**

**{**

**free(header);**

**close\_elf(o);**

**dprintf(STDERR\_FILENO, "Error: `%s`: No such file\n", argv[1]);**

**exit(98);**

**}**

**check\_elf(header->e\_ident);**

**printf("ELF Header:\n");**

**print\_magic(header->e\_ident);**

**print\_class(header->e\_ident);**

**print\_data(header->e\_ident);**

**print\_version(header->e\_ident);**

**print\_osabi(header->e\_ident);**

**print\_abi(header->e\_ident);**

**print\_type(header->e\_type, header->e\_ident);**

**print\_entry(header->e\_entry, header->e\_ident);**

**free(header);**

**close\_elf(o);**

**return (0);**

**}**