Relazione progetto Sistemi Operativi

Swordx

Obbiettivi

Sviluppare una applicazione di sistema Unix/Linux chiamata `swordx` che sia in grado di leggere un insieme di file (di testo) da una o più sorgenti e che produca in output un tile di testo contenente la lista delle parole (ordinate in ordine alfabetico) che occorrono nei file letti con la relativa occorrenza. Per parola si intende ogni sequenza costituita da soli caratteri alfanumerici (caratteri o lettere). Le parole non sono sensibili al maiuscolo.

La sinossi del programma è:

`swordx [options] [inputs]`

Dove `[inputs]` è una sequenza di file e/o directory, mentre `[options]` sono una sequenza di parametri riportati sotto.

Dettagli

Feature 1

Il programma è in grado di trattare invocazioni del tipo:

`swordx path/file`

Il risultato dell'elaborazione è salvata nel file `swordx.out` salvato nella directory corrente.

Esempio: Ricevuto in input il file test.txt viene prodotto il seguente file swordx.out.

Feature 2

Come input può essere passata una directory:

`swordx path/directory`

In questo caso il file `swordx.out` conterrà le statistiche ottenute elaborando tutti i file (regolari) contenuti nella directory. Nell'analisi, quindi, vengono esclusi i link e le directory.

Feature 3

E' possibile richiamare `swordx` passando più input:

`swordx <input1> <input2> … <inputn>`

Viene generato un singolo file `swordx.out` contenente le statistiche collezionate.

Feature 4

E' possibile indicare il nome del file dove devono essere salvate le statistiche collezionate:

`swordx –output <file> [inputs]` `swordx -o <file> [inputs]`

Le statistiche collezionate vengono salvate nel file `<file>`.

Feature 5

E' possibile passare a `swordx` i seguenti parametri aggiuntivi:

1. `–help` e `-h`: stampa a video l'help del programma;

2. `--recursive` e `-r`: se viene passato questo argomento, nella collezione delle informazioni di una directory vengono seguite tutte le sottodirectory;
3. `--follow` e `-f`: nell'elaborazione di una directory vengono seguiti i link;
4. `--explude <file>` e `-e <file>`: `<file>` non viene considerato nell'elaborazione (questo parametro ha senso solo se si sta elaborando una directory);
5. `--alpha` e `-a`: vengono considerate nella statistica solo le parole contenenti caratteri alfabetici;
6. `--min <num>` e `-m <num>`: vengono considerate nella statistiche solo le parole con una lunghezza maggiore o uguale a `<num>`;
7. `--ignore <file>` e `-i <file>`: `<file>` è un elenco di parole (una per riga) che vengono ignorate nella statistica;
8. `--sortbyoccurrency` e `-s`: le parole vengono inserite nel file di output ordinate per numero di occorrenze.
9. `--log <file>` e `-l <file>`: genera un file di log (identificato da `<file>`) dove viene riportate viene riportata, per ogni file analizzato viene riportata una riga della forma `<name> cw iw time` dove
    a. `<name>` è il nome del file analizzato;
    b. `cw` è il numero delle parole registrate;
    c. `iw` è il numero delle parole ignorate;
    d. `time` è il tempo necessario all'elaborazione del file.
10. `--update <file>` se presente il file `swordx.out` (o il file indicato con l'opzione `-o`) scrive nel file `<file>`, con lo stesso formato del file di output, l'elenco delle parole con le rispettive variazioni rispetto all'esecuzione precedente del programma.

Feature 6

Consentire di specificare valori parametri di input usando espressioni regolari.

File

Il mio progetto è costituito da tre file c, rispettivamente Swordx.c Reader.c Structure.c

E due header file, Reader.h Structure.h

```c
#include <stdlib.h>
#include <ctype.h>
#include <dirent.h>
#include "Reader.h"

FILE *open_file(char *path)
{
    FILE *file = fopen(path, "r");
    return file;
}

FILE *create_file(char *path)
{
    FILE *file = fopen(path, "ab+");
    return file;
}

FILE *open_file_w_mode(char *path)
{
    FILE *file = fopen(path, "w+");
    return file;
}

DIR *open_dir(char *path)
{
    DIR *dir = open_dir(path);
    return dir;
}
```

Il file Reader.c contiene tutti i metodi per l'apertura, la creazione di file.txt o directory

La struttura è la medesima per ogni metodo, si differenziano principalmente per le "modalità". Le modalità sono quelle espressioni che consentono di capire in che modo il file o la directory deve essere aperta(Lettura-Scrittura ecc).

I metodi per la gestione dei file hanno come valore di ritorno il file che si sta utilizzando mentre per la gestione delle directory la directory corrente.

Reader.h

```c
#include <stdio.h>
#include <ctype.h>
#include <dirent.h>

FILE *open_file(char *path);
FILE *create_file(char *path);
DIR *open_dir(char *path);
FILE *open_file_w_mode(char *path);
```

Reader.c è affiancato da un header file chiamato Reader.h , questo file contiene i prototipi delle funzioni utilizzate all'interno del file c

Structure.c

```c
int compare(const void *a, const void *b)
{
    wfstruct *ap = (wfstruct *)a;
    wfstruct *bp = (wfstruct *)b;
    return (strcmp(ap->seen, bp->seen));
}
/* qsort compare funciton by frequency */
int compare_by_frequency(wfstruct *a, wfstruct *b)
{
    if (a->freq < b->freq)
        return +1;
    if (a->freq > b->freq)
        return -1;
    return 0;
}
```

```c
int get_word_freq(wfstruct *words, size_t *idx, FILE *file)
{
    char word[MAXC] = {0};
    size_t i;

    /* read first word into array, update index 'idx' */
    if (*idx == 0)
    {

        if (fscanf(file, " %32[^ ,.\t\n]%*c%*[!?-_'><@#*:;°]", word) == 1)
        {

            strcpy(words[*idx].seen, word);
            words[*idx].freq++;
            (*idx)++;
        }
        else
        {
            fprintf(stderr, "error: file read error.\n");
            return 1;
        }
    }
```

Questo file rappresenta il core di Swordx in quanto contiene tutti i metodi per la scansione delle parole dei file che si vuole analizzare lo storage dei caratteri e molto altro.

Il metodo get_word_freq utilizza una struttura wfstruct presente all'interno di Structure.h (analizzeremo i n seguito).

Utilizzando un indice (idx) legge la prima parola e la posiziona all'interno di un array aggiornando l'indice, fscanf effettua la scansione del file e tramite i parametri aggiuntivi ([!?,;.]ecc.)comprende i delimitatori di un file e può così andare avanti e analizzare altre parole senza considerare i delimitatori come parte della parola che sta leggendo.

Dopo che ha effettuato lo storage di tutte le parole le confronta all'interno di quelle presenti e conta le occorrenze di ogni parola facendo un upgrade di frequenza ogni qualvolta che la parola viene ripetuta.

L'espressione goto skipdup: è una jump-statement questa etichetta indirizza a un controllo per il numero massimo di parole che possono essere presenti nei file da analizzare.

Seguono poi i metodi compare e compare_by_frequency il primo ha il compito di ordinare le parole in modo alfabetico tramite lo string comprare.

Comprare_by_frequency funziona in modo molto simile solo che invece di fare lo string compare effettua un controllo del valore integer delle varie frequenze ordinandole.

Structure.h

All'interno possiamo trovare la dichiarazione della struttura chiamata wfstruct.

Questa struttura comprende un char seen con all'interno MAXC ovvero il numero massimo di caratteri che può contenere una parola ossia(32) questo perché nel dizionario della lingua "ITALIANA" (potrebbe variare in caso di un'altra lingua) la parola più lunga è di 32 caratteri e un size_t freq.

Size_t è uno standard e viene usato per la rappresentazione della dimensione di un oggetto, nel mio caso per la rappresentazione della frequenza.

Siamo arrivati al main del programma Swordx ovvero Swordx.c

```c
int main(int argc,char *argv[])
{
clock_t start = clock();
char array[1000][1000];
char filename[1000];
struct dirent *in_file;
wfstruct words[MAXW] = {{{0}, 0}};
size_t i, idx = 0;
int c;
int j;
int num;
char *filename_to_exclude;
DIR *dir;
FILE *file;
const char *short_options = "hr:f:e:a:m:i:l:s:u";
const struct option long_options[] =
    {
        {"help", no_argument, 0, 'h'},
        {"recursive", required_argument, 0, 'r'},
        {"follow", no_argument, 0, 'f'},
        {"alpha", no_argument, 0, 'a'},
        {"exclude", required_argument, 0, 'e'},
        {"min", required_argument, 0, 'm'},
        {"ignore", required_argument, 0, 'i'},
        {"sortbyoccurences", no_argument, 0, 's'},
        {"log", no_argument, 0, 'm'},
        {"upgrade", required_argument, 0, 'u'},
        {NULL, 0, NULL, 0},
};
```

Qui possiamo trovare tutte le dichiarazioni delle variabili quali(array,filename,i,j,start,dir,file ecc).

Per svolgere questo progetto, dato che il programma includeva la possibilità di richieste aggiuntive ho utilizzato le short_option.

Questa funzionalità mi è stata molto utile in quando definisce le specifiche aggiuntive che si possono dare in pasto al programma.7

All'interno presentano

1. Il nome della specifica
2. L'argument che richiede, se lo richiede
3. Valore
4. la lettera associata alla short_option

```
if (argc < 2)
{
    get_word_freq(words, &idx, stdin);
}
else
{
    //read each file given on command line
    for (i = 1; i < (size_t)argc; i++)
    {

        // open file for reading
        if (!(file = open_file(argv[i])))
        {
            fprintf(stderr, "error: file open failed '%s'.\n", argv[i]);
            continue;
        }
        // check 'idx' against MAXW
        if (idx == MAXW)
            break;

        get_word_freq(words, &idx, file);
    }
}

//sort words alphabetically
qsort(words, idx, sizeof *words, compare);
printf("\nthe occurrence of words are:\n\n");
for (i = 0; i < idx; i++) {
    printf("  %-28s : %zu\n", words[i].seen, words[i].freq);
```

Questo controllo serve nel caso l'utente non voglia utilizzare le funzionalità aggiuntive ma si voglia semplicemente limitare a analizzare i file.

Tramite un controllo (<2) o (>=2) possiamo sapere se l'utente ha inserito un solo file da analizzare o più file da analizzare contemporaneamente.

Se il file è singolo il programma ottiene la lista di parole e la loro relativa frequenza, se i file sono molteplici il programma comprende tutte le parole che possono essere presenti nei file e produce un solo file di output contenente la relazione di “tutti” i file.

```
case 'r':
    dir = open_dir(argv[2]);
    while ((in_file = readdir(dir)) != NULL)
    {
        printf("%s\n", in_file->d_name);
        sprintf(filename, "%s/%s", argv[2], in_file->d_name);
        file = open_file(filename);
        get_word_freq(words, &idx, file);
        qsort(words, idx, sizeof *words, compare);
        printf("\nthe occurrence of words are:\n\n");
        for (i = 0; i < idx; i++)
        {
            printf("  %-28s : %zu\n", words[i].seen, words[i].freq);
        }

    }
    closedir(dir);
    return 0;
case 'm':
    num = atoi(argv[2]);
    file = open_file(argv[3]);
    get_word_freq(words, &idx, file);
    qsort(words, idx, sizeof *words, compare);
    printf("\nthe occurrence of words are:\n\n");
    for (i = 0; i < idx; i++)
    {
        if (strlen(words[i].seen) >= num)
            printf("  %s : %zu\n", words[i].seen, words[i].freq);
    }
    return 0;
```

```
case 's':
    file = open_file(argv[2]);
    get_word_freq(words, &idx, file);
    qsort(words, idx, sizeof *words, (int (*)(const void *, const void *))compare_by_frequency);
    printf("\nthe occurrence of words are:\n\n");
    for (i = 0; i < idx; i++)
        printf("  %s : %zu\n", words[i].seen, words[i].freq);

    return 0;
case 'e':

    filename_to_exclude = (argv[5]);
    break;
case 'a':
    file = open_file(argv[2]);
    get_word_freq_alpha(words, &idx, file);
    qsort(words, idx, sizeof *words, compare);
    printf("\nthe occurrence of words are:\n\n");
    for (i = 0; i < idx; i++)
    {
        printf("  %s : %zu\n", words[i].seen, words[i].freq);
    }
    return 0;
```

```
case 'l':

    file = open_file(argv[2]);
    get_word_freq_alpha(words, &idx, file);
    qsort(words, idx, sizeof *words, compare);
    file = open_file_w_mode("logFile.log");
    clock_t end = clock();
    double cpu_timing = (double)(end - start) / CLOCKS_PER_SEC;
    fprintf(file, "%f\n%s\n%ld\n%d", cpu_timing, argv[2], idx, j);
    fclose(file);
    return 0;
case 'u':
    file = open_file(argv[2]);
    get_word_freq_alpha(words, &idx, file);
    qsort(words, idx, sizeof *words, compare);
    file = open_file(argv[3]);
    for (i = 0; i < idx; i++)
    {
        fprintf(file, "%s:%zu\n", words[i].seen, words[i].freq);
        fclose(file);
    }
    return 0;
```

```c
case 'i':
    file = open_file(argv[2]);
    while (fscanf(file, "%1000c", array[j]) != EOF)
    {
        j++;
    }
    for (int p = 0; p < j; p++)
    {
        printf("%s\n", array[p]);
    }
    fclose(file);
    file = open_file(argv[3]);
    get_word_freq_alpha(words, &idx, file);
    qsort(words, idx, sizeof *words, compare);
    printf("\nthe occurrence of words are:\n\n");
    for (i = 0; i < idx; i++)
    {
        for (int p = 0; p < j; p++)
            if (strcmp(words[i].seen, array[p]) == 0)
                words[i].seen == "\0";
    }
    for (i = 0; i < idx; i++)
    {
        printf("  %s : %zu\n", words[i].seen, words[i].freq);
    }
    return 0;
```

Questi sono tutti quanti i case relativi alle short_option.

La funzionalità “r” serve per la ricorsione ovvero la lettura di tutti file all’interno di un una directory.

Si controlla prima di tutto l’esistenza della cartella stessa, dopo di che la presenza di file da analizzare, se queste condizioni sono rispettate avverrà l’analisi come da programma e al termine si chiuderà la directory.

La funzionalità “m” utilizza (atoi) per determinare la lunghezza delle parole e trasformarle in un intero, così facendo all’interno dell’analisi non verranno considerate le parole che sono minori del numero che l’utente ha inserito.

La funzionalità “s” utilizza il metodo compre_by_frequency definito in precedenza per ordinare le parole non più i ordine alfabetico bensì in ordine di occorrenza.

La funzionalità “e” prende come input il nome del file che si vuole escludere dall’analisi, così facendo quando si lavorerà all’interno di una cartella, se è presente un file con lo stesso nome dell’Exclude questo verrà ignorato.

La funzionalità ”a” serve per considerare solo caratteri alfabetici, tramite il metodo

get_word_freq_alpha.

Questo metodo è simile a quello convenzionale ma, all’interno dei parametri che aggetta, esclude tutti quanti i numeri tramite ([1,2,3,4,5,6,7,8,9,0]).

La funzionalità ”l” serve per generare un file.log che contiene all’interno il nome del file, il numero di parole analizzate e il numero di parole che sono state ignorate.

Per definire il tempo esiste una libreria apposita che utilizza il clock, tramite questo metodo si può sapere quando ci mette la macchina a processare i file, il risultato ovviamente sarà maggiore nel momento in cui si analizzano più file.

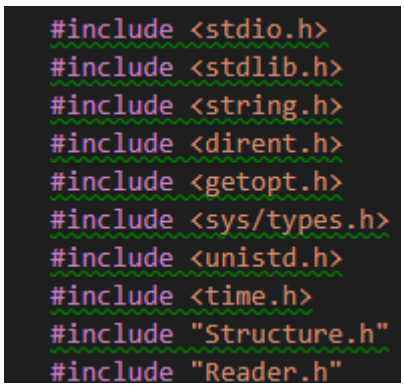
La funzionalità ”u” serve per fare l’update di un file che era stato analizzato in precedenza.

Bisogna dare in input la produzione che era già stata fatta e affianco il nuovo file che si vuole analizzare, il programma sostituisce i valori dell’analisi passata con quella corrente.

La funzionalità “i” serve per ignorare determinate parole che non si vogliono contare nell’analisi del file.

Il programma analizza il file e salva tutte le parole presenti all’interno, dopo di che apre il nuovo file da analizzare e tramite le parole salvate in precedenza esegue uno skip ogni qualvolta si ritrova una di quelle parole.

All’interno è presente un doppio ciclo for con uno string compare che ha appunto il compito di comprare la parola salvata con la parola che è appena stata letta.



Queste sono rispettivamente le librerie che mi sono state utili al fine del completamento della consegna.

<stdio.h> contiene tutte le marco per la gestione input output

<stdlib.h> l’ho utilizzata per eseguire le conversioni come (atoi)

<string.h> per la manipolazione delle stringhe e della memoria

<dirent.h> per la manipolazione delle directory

<getopt.h> per la funzionalità delle short_option

<sys/types.h> per la gestione del size_t

<unistd.h> per le chiamate di sistema

<time.h> per la gestione del clock

Infine Structure.h e Reader.h per l’associazione degli header file già visti in precedenza.

Considerazioni personali

Per la correttezza del codice sono stati effettuati vari testi, molti di questi grazie all'ausilio del debugger presente in Visual studio code.

Utilizzando diversi file.txt contenti parole numeri ecc è stato possibile verificare che tutte le funzionalità da me sviluppate fossero esatte.

Il tema del progetto è stato molto interessante in quanto soprattutto per quanto riguarda la completezza delle funzionalità.

Tramite Swordx mi è stato possibile apprendere un tipo di programmazione che per me era ancora sconosciuta, sebbene sia edotto che il programma poteva essere ottimizzato in maniera ancora più esemplare.

La difficoltà maggiore credo sia stata nella realizzazione della struttura, infatti questa rappresenta il core del programma tramite il quale si basano tutti i metodi e tutte le funzionalità delle short_option

File di output

Mentre lavoravo a Swordx ho appreso una funzionalità di Gnu che ancora non conoscevo ovvero la possibilità di generare un file di output tramite l'utilizzo del carattere(>) posto al fine della dichiarazione a linea di comando e seguito da il nome del file che si vuole produrre come output.

All'interno di Swordx è presente il metodo create_file infatti se si vuole si può anche decidere di produrre un file di output però c'è da considerare che questa funzionalità l'ho trovata molto interessante e pertanto ho deciso di utilizzarla.