2018

Cyclone

INFORMATICA III – MODULO A

LORENZO MILESI – 1030232
Università degli studi di Bergamo

Introduzione

Obiettivo del progetto è sviluppare un piccolo algoritmo in Cyclone che cifri una stringa in ingresso secondo un modello predefinito. Per comodità di visione il programma stampa schermo anche la riconversione della stringa, di modo che l'utente capisca che il processo è reversibile.

Costrutti propri di Cyclone utilizzati

Qualificatore @notnull (puntatore @)

Cyclone permette l'utilizzo dei normali puntatori `*` con alcune modifiche rispetto a C, in particolare un controllo se il puntatore è nullo ad ogni de-reference dello stesso (per prevenire *segmentation fault*). Tuttavia, questo processo potrebbe essere dispendioso. Con il qualificatore `@notnull` è possibile effettuare il controllo all'assegnamento del valore ed evitarlo al suo utilizzo. È uno dei più utili ed utilizzati, al pari di `@fat`. Può essere espresso sia con `* @notnull` che, più semplicemente, con `@`.

Nel nostro caso è stato utilizzato per il caricamento del file contenente la stringa da cifrare:

```
FILE *@notnull test = (FILE@)fopen("test.txt", "r");
```

Qualificatore @fat (puntatore ?)

Questo qualificatore impone che il puntatore su cui viene applicato mantenga anche l'informazione sul numero degli elementi dell'array. È quindi particolarmente utile per poter conoscere la dimensione delle stringhe senza dover ricorrere a `strlen()`. Può essere espresso sia con `* @fat` che, più semplicemente con `?`.

Nel nostro caso è stato utilizzato per dichiarare gli array ospitanti le stringhe usate e allocarne lo spazio necessario:

```
char *@fat crypt = calloc(MAX,sizeof(char));
char *@fat dec = calloc(MAX,sizeof(char));
char *@fat str = calloc(MAX,sizeof(char));
```

Garbage Collector

È importante notare che tutte le variabili allocato nello heap non sono liberate attraverso delle chiamate `free()`, ma gestite in modo automatico dal Garbage Collector, che si occupa appunto di liberare la memoria quando le variabili non sono più referenziate da alcun puntatore.

Logica

La funzione di cifratura è svolta da un semplice ciclo for, che scorre l'array in ingresso e assegna un valore al corrispondente slot dell'array in uscita, nel modo seguente:

```
     for (int i=0;str[i]!='\0';i++){
          if(str[i]==' ' || str[i]==',' || str[i]=='.')
               crypt[i] = str[i];
          else
               crypt[i] = (str[i] - 64 + DELAY+(i+1)) % 27+64;
          printf("%c",crypt[i]);
     }
```

L'idea alla base della codifica è quella di applicare uno slittamento di N posizioni nell'alfabeto latino a tutti i caratteri che compongono la stringa, tale che N = (i+1) + DELAY, dove:

- DELAY = costante predefinita, in questo caso con il valore di 5.
- i+1 = posizione che il carattere assume all'interno della stringa.

I caratteri corrispondenti a spazio, virgola e punto non vengono ignorati ma replicati tali e quali, quindi non codificati.

Output

Esempio di output:

```
## PROGRAMMA DI CIFRATURA ##
Il programma riceve una stringa in input e ne restituisce una cifratura
ottenuta con
un delay di N posizioni lungo l'alfabeto latino (26 lettere) con N = n +
i, dove:
- n è una costante predefinita (5);
- i è la posizione del carattere all'interno della stringa;
NB: Spazi, virgole e punti sono, per scelta, ignorati nella cifratura.

Stringa in ingresso: HI, MY NAME IS LORENZO.
Stringa cifrata: NP, WI @OAU @K FJNBLYO.
Stringa decifrata: HI, MY NAME IS LORENZO.
```