Laboratorio di Elementi di Architetture e Sistemi Operativi

Esercizi del 18 Aprile 2012

Esercizio 1. *Modificare l'Esercizio 3 della lezione scorsa (occorrenze di ogni lettera dell'alfabeto) in modo che legga l'input da un file anziché dallo standard input. Il nome del file viene passato come primo argomento del* `main` *dalla linea di comando. Si assuma per semplicità che il nome del file non contenga spazi, e si gestiscano correttamente gli errori.*

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  FILE *input;
  int f[26];
  int c;

  if(argc != 2) {
    fprintf(stderr, "Uso: %s nomefile\n", argv[0]);
    return 1;
  }

  input = fopen(argv[1], "r");
  if(input == NULL) {
    fprintf(stderr, "Errore nell'apertura del file %s\n", argv[1]);
    return 1;
  }

  for(c = 0; c < 26; c++)
    f[c] = 0;

  for(c = getc(input); c != EOF; c = getc(input)) {
    c = tolower(c);
    if(c >= 'a' && c <= 'z')
      f[c-'a']++;
  }

  if(fclose(input) != 0) {
    fprintf(stderr, "Errore nella chiusura del file %s\n", argv[1]);
    return 1;
  }

  for(c = 'a'; c <= 'z'; c++)
    printf("%c: %d\n", c, f[c-'a']);

  return 0;
}
```

Esercizio 2.

1. *Scrivere una funzione con il seguente prototipo:*
   `int strindex{char *s, char *t);`
   che restituisce la posizione (indice del vettore) di inizio della prima occorrenza della stringa `t` *in* `s`*, oppure* `-1` *se* `t` *non compare in* `s`*.*

2. *Scrivere una versione semplificata del comando `grep` che prenda come argomenti una stringa e due nomi di file, e scriva tutte le linee del primo file contenenti almeno un'occorrenza della stringa nel secondo file. Si assuma per semplicità che che la stringa ed i nomi dei file non contengano spazi, che le righe siano lunghe al massimo `80` caratteri, e si gestiscano correttamente gli errori.*

```
#include <stdio.h>

#define MAXCHAR 81

int strindex(char *, char *);

int main(int argc, char *argv[])
{
  FILE *input, *output;
  char str[MAXCHAR];
  char *res;

  if(argc != 4) {
    fprintf(stderr, "Uso: %s stringa nomefile nomefile\n", argv[0]);
    return 1;
  }

  input = fopen(argv[2], "r");
  if(input == NULL) {
    fprintf(stderr, "Errore nell'apertura del file %s\n", argv[2]);
    return 1;
  }

  output = fopen(argv[3], "w");
  if(output == NULL) {
    fprintf(stderr, "Errore nell'apertura del file %s\n", argv[3]);
    return 1;
  }

  for(res = fgets(str, MAXCHAR, input); res != NULL; res = fgets(str, MAXCHAR, input)) {
    if(strindex(str, argv[1]) >= 0) {
      fputs(str, output);
    }
  }

  if(fclose(input) != 0) {
    fprintf(stderr, "Errore nella chiusura del file %s\n", argv[2]);
    return 1;
  }

  if(fclose(output) != 0) {
    fprintf(stderr, "Errore nella chiusura del file %s\n", argv[3]);
    return 1;
  }

  return 0;
}

int strindex(char *s, char *t) {
  int i, j;
```

```
  for(i = 0; s[i] != '\0'; i++) {
    for(j = 0; s[i+j] == t[j] && t[j] != '\0'; j++)
      ;
    if(t[j] == '\0') return i;
  }

  return -1;
}
```

Esercizio 3.

1. *Realizzare un insieme di funzioni per gestire una pila (stack) di valori floating point. In particolare, si implementino le operazioni di inserimento (`pop`) ed estrazione (`push`) di un valore nella pila. Si assuma che la pila possa contenere al massimo 10 valori.*
2. *Utilizzare le funzioni di gestione della pila per implementare una calcolatrice che esegua le quattro operazioni aritmetiche (+, −, ⋆, /) usando la notazione polacca inversa (RPN). La calcolatrice riceve le operazioni da eseguire dalla tastiera, leggendo un operatore o operando per ogni riga. Dopo aver letto un numero o un operatore (ed aver eseguito l'operazione) mostra i valori contenuti nello stack. Il programma termina quando l'utente inserisce `q`.*

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

#define NUM 0
#define OP 1
#define Q 2
#define ERR -1

int getop(char *str);
float getnum(char *str);
void esegui(char c);

void push(float f);
float pop();
void printstack();

main()
{
    char str[20], *res;
    int op;

    printf("Calcolatrice Polacca inversa\n");
    op = NUM;
    res = gets(str);
    while(res != NULL) {
        op = getop(str);
        switch(op) {
          case NUM:
            push(getnum(str));
            break;
          case OP:
            esegui(str[0]);
            break;
          case Q:
            printf("Arrivederci!\n");
            return;
            break;
```

```
          default:
            printf("Istruzione non valida!\n");
            break;
        }
        printstack();
        res = gets(str);
   }
}


// Gestione dello stack
float stack[10];
int cima = -1;

void push(float f) {
  cima++;
  stack[cima] = f;
}

float pop() {
  if(cima >= 0) {
    cima--;
    return stack[cima+1];
  } else {
    printf("Errore! Lo stack è vuoto!\n");
    return 0;
  }
}

void printstack() {
    int i;
    printf("Stack: ");
    for(i = 0; i <= cima; i++)
        printf("%f ",stack[i]);
    printf("\n");
}

// funzione che parserizza l'input
int getop(char *str) {
    int n = strlen(str);
    float f;

    if(n == 1 && (str[0] == '+' || str[0] == '-' || str[0] == '*' || str[0] == '/'))
        return OP;
    if(n == 1 && str[0] == 'q')
        return Q;
    n = sscanf(str, "%f", &f);
    if(n == 1)
        return NUM;
    return ERR;
}

float getnum(char *str) {
    float f;
    sscanf(str, "%f", &f);
```

```
    return f;
}

void esegui(char c) {
    float a,b;

    if(cima < 1) {
        printf("ERRORE: numero insufficiente di operandi nello stack!\n");
        return;
    }
    b = pop();
    a = pop();
    switch(c) {
      case '+':
        push(a+b);
        break;
      case '-':
        push(a-b);
        break;
      case '*':
        push(a*b);
        break;
      case '/':
        push(a/b);
        break;
      default:
        printf("Operazione sconosciuta!\n");
        push(a);
        push(b);
        break;
    }
}
```