

**PROVA SCRITTA DEL MODULO DI
ELEMENTI DI INFORMATICA
CORSO DI LAUREA IN INGEGNERIA BIOMEDICA
26 giugno 2019**

MOTIVARE IN MANIERA CHIARA LE SOLUZIONI PROPOSTE A CIASCUNO DEGLI ESERCIZI SVOLTI

ESERCIZIO 1 (4 punti)

E' possibile rappresentare il valore 1135 con 10 bit? Motivando la risposta, rappresentarlo con il minimo numero di bit possibile.

ESERCIZIO 2 (4 punti)

Descrivere in modo chiaro e sintetico il gestore della memoria di un moderno sistema operativo.

ESERCIZIO 3 (25 punti)

(5 punti) Avete deciso di dotare il vostro robot domestico dell'abilità di interpretare le espressioni facciali di chi ha di fronte in base ai lineamenti del viso, in modo da dotarlo di un comportamento adeguato per ogni occasione. Il robot, tramite due videocamere poste in corrispondenza degli occhi, osserva la persona per 1 secondo, durante il quale scatta 1000 foto della persona. Le 1000 foto ottenute vengono elaborate velocemente dal robot che scrive per ogni riga del file "espressioni.txt" i valori di intensità espressiva di ogni foto, associati, nell'ordine, ad otto espressioni-base: neutrale, felicità, disprezzo, disgusto, rabbia, tristezza, paura, sorpresa. Ciascun valore di intensità è un numero reale compreso tra 0 e 1.

Un esempio di file "espressioni.txt" è dato dal seguente:

```
0.1 0.5 0.234 0.124 0.8 0.8 0.2 0.15
0.441 0.3 0.34 0.924 0.38 0.18 0.21 0.115
0.12 0.53 0.2134 0.1524 0.83 0.1 0.7 0.15
```

...

L'esempio riporta solo le prime tre righe del file, ma va considerato il file intero.

A voi si richiede di sviluppare il modulo in grado di elaborare il file restituendo un nuovo file "risultati.txt" nel quale verrà scritta l'espressione con intensità complessiva maggiore ed il relativo valore di intensità. Il file "risultati" dovrà quindi prevedere in una riga il nome dell'espressione più intensa, seguita dal corrispondente valore di intensità complessiva. Per intensità complessiva si intende la somma delle intensità relative a quell'espressione riportate nel file "espressioni.txt".

Per far ciò decidete di memorizzare ciascuna riga del file "espressioni.txt" in una lista concatenata di vettori di *floating point*, tutti di dimensione pari al numero di espressioni prese in considerazione. Implementate quindi le seguenti funzioni:

- (4 punti) `leggiEspressioni(nomefile)`: riceve in ingresso il nome del file da aprire e restituisce la lista concatenata organizzata come descritto;
- (6 punti) `calcolaIntensita(lista, espr)`: riceve in ingresso la lista organizzata come descritto ed una stringa `espr` che indica appunto l'espressione della quale si intende calcolare l'intensità complessiva. Restituisce il corrispondente valore di intensità complessiva;
- (8 punti) `estraiEspressione(vettore_misure)`: riceve in ingresso un vettore di intensità associate alle espressioni, nell'ordine indicato all'inizio del testo, e l'indice associato alla massima intensità (es. 0 → "neutrale", 1 → "felicità", ..., 7 → "sorpresa").
- (2 punti) `scriviRisultati(vettore_misure, indice_max_intensita)`: scrive in un file "risultati.txt" l'espressione ed il relativo valore di intensità associate al parametro numerico `indice_max_intensita`, che rappresenta appunto l'indice numerico a sua volta corrispondente all'espressione nell'ordine indicato all'inizio del testo.

Nota: chi riuscirà ad individuare e scrivere ulteriori funzioni per modularizzare e semplificare la lettura del codice, riceverà un bonus fino a 6 punti. Si presuma inoltre che le funzioni HEAD, CONS, TAIL, ISEMPY siano già state implementate.



Soluzione dell'esercizio 1.

Poiché $2^{10}=1024<1135$, non è possibile rappresentare il numero dato con soli 10 bit. D'altra parte $2^{11}=2048>1135$, quindi 11 bit saranno sufficienti.

La conversione si effettua con l'algoritmo delle divisioni successive e si lascia allo studente.

Soluzione dell'esercizio 2.

Si vedano le dispense del corso.

Soluzione dell'esercizio 3.

```
#include <stdio.h>
#include <stdlib.h>
#define N 1000      //Numero righe nel file
#define nEspr 8     //Numero di espressioni considerate

//Tipi di dato necessari per l'implementazione del programma
typedef struct tipo_Nodo
{
    float *valori;
    struct tipo_Nodo* successivo;
} listaEspressioni;

//Funzioni per la gestione della lista concatenata
//Queste funzioni non sono da implementare secondo il testo
listaEspressioni* CONS(listaEspressioni *l, float *m)
{
    listaEspressioni* lista;
    lista=(listaEspressioni*)malloc(sizeof(listaEspressioni));

    lista->valori=m;
    lista->successivo=l;

    return lista;
}

float* HEAD(listaEspressioni *l)
{
    return l->valori;
}

listaEspressioni* TAIL(listaEspressioni *l) //non dealloca i contenuti della
{                                           //testa
    return l->successivo;
}

listaEspressioni* CUT(listaEspressioni *l) //taglia la testa deallocandone
{                                           //i contenuti
    listaEspressioni *t;

    t=l->successivo;
    free(l->valori);
    free(l);

    return t;
}
```

```

int ISEMPY(listaEspressioni* l)
{
    return l==NULL;
}

//Funzioni richieste ed aggiunte per risolvere l'esercizio
float* leggiMisure(FILE *f) //non richiesta ma utile per alleggerire
{
    //la leggibilità del codice (modularizzazione)
    float *v;
    int j;

    v=(float*)malloc(sizeof(float)*nEspr);

    for(j=0; j<nEspr; j++)
        fscanf(f,"%f",&v[j]);

    return v;
}

listaEspressioni* leggiEspressioni(char* nomefile)
{
    FILE *f;
    int i,j;
    float *v;

    listaEspressioni* l=NULL;

    f=fopen(nomefile,"r");
    for(i=0; i<N; i++)
    {
        v=leggiMisure(f);
        l=CONS(l,v);
    }

    fclose(f);

    return l;
}

//Ordine delle espressioni da testo:
//neutrale, felicità, disprezzo, disgusto, rabbia, tristezza, paura, sorpresa
int codificaEspressione(char *espr) #non richiesta ma utile per
{
    #modularizzazione
    if (!strcmp(espr,"neutrale"))
        return 0;
    if (!strcmp(espr,"felicità"))
        return 1;
    if (!strcmp(espr,"disprezzo"))
        return 2;
    if (!strcmp(espr,"disgusto"))
        return 3;
    if (!strcmp(espr,"rabbia"))
        return 4;
    if (!strcmp(espr,"tristezza"))
        return 5;
    if (!strcmp(espr,"paura"))
        return 6;
    if (!strcmp(espr,"sorpresa"))
        return 7;
}

```

```

float calcolaIntensita(listaEspressioni *l, char* espr)
{
    int i;
    float *m;
    float s=0.0;
    listaEspressioni *t=l;

    i=codificaEspressione(espr);

    while (!ISEMPTY(t))
    {
        m=HEAD(t);
        s=s+m[i];
        t=TAIL(t);
    }

    return s;
}

```

```

int estraiEspressione(float* m)
{
    int i, imax;
    float vmax;

    imax=0;
    vmax=m[0];

    for (i=1; i<nEspr; i++)
        if(m[i]>vmax)
        {
            vmax=m[i];
            imax=i;
        }

    return imax;
}

```

```

char* codificaIndice(int i)
//non richiesta ma migliora la leggibilità del codice
{
    switch(i)
    {
        case 0: return "neutrale";
        case 1: return "felicità";
        case 2: return "disprezzo";
        case 3: return "disgusto";
        case 4: return "rabbia";
        case 5: return "tristezza";
        case 6: return "paura";
    }
    return "sorpresa";
}

```

```

void scriviRisultati(float *m, int imax)
{
    FILE *f;
    char* espr;

    espr=codificaIndice(imax);

    f=fopen("risultati.txt","w");
    fprintf(f,"%s %f\n",espr,m[imax]);
    fclose(f);
}

float* intensitaComplessive(listaEspressioni *l)
//non richiesta ma utile per modularizzazione
{
    float *m=(float*)malloc(sizeof(float)*nEspr);
    char* espr;
    int i;

    for (i=0; i<nEspr; i++)
    {
        espr=codificaIndice(i); //richiama un'ulteriore funzione aggiunta
        m[i]=calcolaIntensita(l,espr);
    }

    return m;
}

listaEspressioni* svuota(listaEspressioni *l)
{
    while (!ISEMPTY(l))
        l=CUT(l);
    return l;
}

int main() //con le funzioni aggiunte la main è più semplice da leggere
{
    listaEspressioni *l;
    float *m;
    int imax;

    l=leggiEspressioni("espressioni.txt");
    m=intensitaComplessive(l); //migliora la leggibilità e modularizza il codice
    imax=estraiEspressione(m);
    scriviRisultati(m,imax);

    l=svuota(l); //dealloco la lista: non necessaria ma permette lo svuotamento
                //totale della struttura sotto il mio controllo

    free(m);
    return 0;
}

```