

Testo TEMA A

Introduzione

Nella Settimana Enigmistica, accanto a schemi di parole crociate, a sciarade, a zeppe sillabiche, c'è un gioco molto divertente, chiamato "Il bersaglio": dato un insieme di parole, delle quali sono indicate una iniziale ed una finale, bisogna metterle in sequenza in modo che a partire dalla parola iniziale si arrivi a quella finale (il bersaglio) applicando ad ogni passo una delle regole seguenti (noi consideriamo solo quelle piu' semplici):

- La parola può essere un anagramma della parola precedente (es, Galere --> Regale)
- La si può ottenere aggiungendo una lettera (es, Calco :: Calcio)
- La si può ottenere togliendo una lettera (es, Penosa :: Pensa)
- La si può ottenere cambiando una lettera (es. Creta :: Crepa)

ESEMPIO

Ecco un esempio di bersaglio risolto (che da "Tino" arriva a "Cerca"): Tino :: Rino :: Riso :: Roso :: Orso :: Dorso :: Sordo :: Sardo :: Saldo :: Caldo :: Calco :: Calcio :: Falcio :: Falcia :: Faccia :: Caccia :: Ciccia :: Riccia :: Ricca :: Circa :: Cerca.

Le strutture dati da utilizzare sono le seguenti (la sequenza la possiamo rappresentare come lista di parole)

```
#define N 20 /* lunghezza massima delle parole */

typedef char Parola[N];
/* un valore di tipo parola è una stringa di lunghezza minore di N-2 */

typedef struct elem{
    Parola val;
    struct elem * next;
} Elem;

typedef Elem* BersaglioLista;
/* un valore di tipo BersaglioLista e' una lista semplice di parole*/
```

Esercizio 1:

Implementare la seguente funzione:

```
int anagramma(Parola prima, Parola seconda);
```

La funzione, avendo in input due parole deve restituire 1 se sono una anagramma dell'altra, e restituire 0 altrimenti

Soluzione di Claudio Borsato (per un'altra soluzione: vedi Tema B)

```
void ordina(char p[])
{
    int i,j;
    char app;

    for(i=0; i<(strlen(p))-1; i++)
    {
        for(j=i+1; j<strlen(p); j++)
        {
            if(p[i]>p[j])
            {
                app=p[i];
                p[i]=p[j];
                p[j]=app;
            }
        }
    }
}

int anagramma(Parola prima, Parola seconda)
{
    int flag;
    char *pcopy, *scopy;

    if(strlen(prima)!=strlen(seconda)) return 0;

    pcopy=calloc(strlen(prima)+1,sizeof(char));
    scopy=calloc(strlen(seconda)+1,sizeof(char));
    //ricopio le due stringhe in due altre stringhe provvisorie
    strcpy(pcopy,prima);
    strcpy(scopy,seconda);
    //ordino le due stringhe provvisorie
    ordina(pcopy);
    ordina(scopy);
    //se le due stringhe sono uguali, allora ritorno 1, altrimenti 0
    flag=strcmp(pcopy,scopy);
    free(scopy);
    free(pcopy);

    if(flag==0) return 1;
    else return 0;
}
```

Esercizio 2:

Implementare la seguente funzione:

```
int modifica(Parola prima, Parola seconda);
```

La funzione deve restituire 1 se la seconda parola è ottenibile dalla prima aggiungendo, togliendo o modificando una lettera, e deve restituire 0 altrimenti.

Soluzione di Francesco Cagnin (per una soluzione piu' semplice vedi Tema B)

```
int parolaContenuta(Parola parola, Parola contenuta) {
    //parola >= contenuta
    int *caratteriUsatiParola = (int *) calloc(sizeof(int), strlen(parola));
    char* auxParola;

    while (*contenuta) {
```

```

    int trovato = 0;
    auxParola = parola;
    int i = 0;
    while (*auxParola && !trovato) {
        if ((*auxParola==*contenuta) && caratteriUsatiParola[i]==0) {
            trovato = 1;
            caratteriUsatiParola[i] = 1;
        }
        i++;
        auxParola++;
    }
    if (!trovato) {
        return 0;
    }
    contenuta++;
}
return 1;
}

```

```

int modifica(Parola prima, Parola seconda) {
    if (prima == NULL || seconda == NULL) return 0;

    int lunghezzaPrima = strlen(prima);
    int lunghezzaSeconda = strlen(seconda);

    if (lunghezzaPrima == 0 || lunghezzaSeconda == 0) return 0;
    switch (lunghezzaPrima - lunghezzaSeconda) {
        case 1: {
            // tolta lettera
            return parolaContenuta(prima, seconda);
        }
        case 0: {
            // modificata lettera
            int numErrori = 0;
            while (*prima) {
                if (*prima != *seconda) {
                    numErrori++;
                }
                prima++;
                seconda++;
            }
            return numErrori == 1;
        }
        case -1: {
            // aggiunta lettera
            return parolaContenuta(seconda, prima);
        }
        default:
            return 0;
            // le parole hanno più di un carattere di differenza
    }
}

```

Esercizio 3:

Implementare la seguente funzione:

```
int diverse(BersaglioLista bersaglio);
```

La funzione, data una lista di tipo Bersaglio, deve restituire 1 se le parole nella lista sono tutte diverse, e restituire 0 nel caso invece ci siano parole ripetute.

Soluzione di Diego Cornello

```
int controllapresente (Parola a, BersaglioLista l){
    if (l==NULL){return 0;}
    if(strcmp(a,l->val)==0){return 1;}
    return (controllapresente(a, l->next));
}

int diverse(BersaglioLista bersaglio){
    if(bersaglio==NULL){return 1;}
    if(bersaglio->next==NULL){return 1;}
    if(controllapresente(bersaglio->val,bersaglio->next)==1){return 0;}
    return (diverse(bersaglio->next));
}
```

Testo Tema B

Introduzione

Nella Settimana Enigmistica, accanto a schemi di parole crociate, a sciarade, a zeppe sillabiche, c'è un gioco molto divertente, chiamato "Il bersaglio": dato un insieme di parole, delle quali sono indicate una iniziale ed una finale, bisogna metterle in sequenza in modo che a partire dalla parola iniziale si arrivi a quella finale (il bersaglio) applicando ad ogni passo una delle regole seguenti (noi consideriamo solo quelle più semplici):

- La parola può essere un anagramma della parola precedente (es, Galere --> Regale)
- La si può ottenere aggiungendo una lettera (es, Calco :: Calcio)
- La si può ottenere togliendo una lettera (es, Penosa :: Pensa)
- La si può ottenere cambiando una lettera (es. Creta :: Crepa)

ESEMPIO

Ecco un esempio di bersaglio risolto (che da "Tino" arriva a "Cerca"): Tino :: Rino :: Riso :: Roso :: Orso :: Dorso :: Sordo :: Sardo :: Saldo :: Caldo :: Calco :: Calcio :: Falcio :: Falcia :: Faccia :: Caccia :: Ciccia :: Riccia :: Ricca :: Circa :: Cerca.

Le strutture dati da utilizzare sono le seguenti (la sequenza la possiamo rappresentare come array di parole)

```
#define M 20 /* lunghezza massima della sequenza di parole */

typedef char* Stringa;
/* un valore di tipo Stringa è una stringa di caratteri */

typedef Stringa BersaglioArray[M];
/* un valore di tipo BersaglioArray e' un array di (al piu') M parole*/
```

Esercizio 1:

Implementare la seguente funzione:

```
int anagram(Stringa prima, Stringa seconda);
```

La funzione, avendo in input due parole deve restituire 1 se sono una anagramma dell'altra, e restituire 0 altrimenti

Soluzione di Davide Moro

```
int count (Stringa str, char c) {
    int i;
    int tot = 0;
    int size = strlen (str);
    for (i = 0; i < size; i++)
        // confronto i caratteri della stringa con il carattere cercato
        if (str[i] == c)
            tot++;
    return tot;
}

int anagram (Stringa prima, Stringa seconda) {
    int i;
    int l = strlen (prima);
```

```

// la seconda stringa non può avere più caratteri della prima
if (l != strlen (seconda)) return 0;
// scorro la prima lista verificando che ogni suo carattere
//sia anche nella seconda nella stessa q.tà
for (i = 0; i < l; i++)
    if (count (prima, prima[i]) != count (seconda, prima[i])) return 0;
return 1; // stesso numero di caratteri, stessa q.tà = anagramma
}

```

Esercizio 2:

Implementare la seguente funzione:

```
int modified(Stringa prima, Stringa seconda);
```

La funzione deve restituire 1 se la seconda parola è ottenibile dalla prima aggiungendo, togliendo o modificando una lettera, e deve restituire 0 altrimenti.

Soluzione di Marco Gasparini

```

int modified(Stringa prima, Stringa seconda){

    //scorre la parte uguale iniziale
    while(*prima != 0 && *prima == *seconda) {
        prima++;
        seconda++;
    }

    //caso tolta una lettera
    if(*prima && strcmp(prima+1, seconda) == 0) {
        return 1;
    }

    //caso aggiunta una lettera
    if(*seconda && strcmp(prima, seconda+1) == 0) {
        return 1;
    }

    //caso modificata una lettera
    if(*seconda && *prima && strcmp(prima+1, seconda+1) == 0) {
        return 1;
    }

    return 0;
}

```

Esercizio 3:

Implementare la seguente funzione:

```
int verifica(BersaglioArray bersaglio, int dim);
```

La funzione, dato un array di tipo Bersaglio, di dim elementi, deve restituire 1 se tutte le parole contenute sono diverse fra loro, e restituire 0 altrimenti.

Soluzione di Mara Pistellato

```

int verifica(BersaglioArray bersaglio, int dim){

    int i, j;

```

```
if((dim == 0) || (dim == 1))
    return 1;

for(i=0; i< dim-1; i++)
    for(j= i+1; j<dim; j++)
        if(!(strcmp(bersaglio[i], bersaglio[j])))
            return 0;

return 1;
}
```

Testo Tema C

Introduzione

Nella Settimana Enigmistica, accanto a schemi di parole crociate, a sciarade, a zeppe sillabiche, ci sono giochi meno conosciuti, come "il lucchetto" e "la cerniera".

Nel lucchetto si trovano due parole o frasi in cui l'ultima parte della prima è uguale alla prima parte della seconda. Eliminando le parti in comune e leggendo il restante si ottiene la terza parola o frase.

Esempio:

CANE/NERO = CARO;
CRISTALLO/STALLONE = CRINE;
RISCHIO/SCHIOCCO = RICCO;
MAGGIO/GIOIA = MAGIA;
CARATO/ATOMI = CARMI;
CAST/TERME = CASERME;
MATCH/CHADOR = MATADOR.

La cerniera è un gioco che consiste nel trovare due parole o frasi che hanno alcune lettere, agli estremi opposti, uguali; la terza parola o frase si ottiene con la scomparsa delle parti uguali e con l'accostamento delle parti restanti.

Per esempio: MANICO/STAMANI = COSTA; GLI OSTI/MAGLIO = STIMA.

Le strutture dati da utilizzare sono le seguenti:

```
typedef char Stringa[20];  
/* un valore di tipo Stringa è una stringa di caratteri */  
  
typedef struct elem{  
    Stringa val;  
    struct elem * next;  
} Elem;  
  
typedef Elem* ListaParole  
  
/* un valore di tipo ListaParole e' una lista semplice di parole*/
```


Esercizio 1:

Implementare la seguente funzione:

```
int lucchetto (Stringa prima, Stringa seconda);
```

La funzione, avendo in input due parole deve restituire 1 se esiste un suffisso della prima parola ed un prefisso della seconda parola che sono uguali, ovvero se da queste due parole si puo' formare un lucchetto (e restituire 0 altrimenti).

Soluzione di Sergiu Groza

```
int is_prefisso(char* s1,char* s2){
    int i;
    int dim = strlen(s1);
    int dim2 = strlen(s2);
    if (dim==0) return 0;
    if (dim2==0) return 0;
    if(dim>dim2){
        return 0;
    }
    for(i=0;i<dim;i++){
        if(s1[i]!=s2[i]) return 0;
    }
    return 1;
}

int lucchetto(Stringa prima, Stringa seconda){
    if(strlen(prima)==0) return 0;
    if(strlen(seconda)==0) return 0;
    if(!is_prefisso(prima,seconda)){
        return lucchetto(prima+1,seconda);
    }
    return 1;
}
```

Esercizio 2:

Implementare la seguente funzione:

```
int cerniera(Stringa prima, Stringa seconda, Stringa terza);
```

La funzione, avendo in input tre parole deve restituire 1 se costituiscono effettivamente una cerniera, e restituire 0 altrimenti

Soluzione

```
int cerniera(Stringa prima, Stringa seconda, Stringa terza){
    Stringa aux;
    int i,j;

    int lunprima = strlen(prima);
    int lunseconda = strlen(seconda);
    int lunterza = strlen(terza);
    int luntolto = (lunprima + lunseconda - lunterza)/2;

    if ((luntolto > lunprima) || (luntolto > lunseconda)) return 0;

    // verifico la parte comune
    strcpy(aux, seconda);
    aux[luntolto]='\0';
```

```

    if (strcmp(prima+(lunprima - luntolto), aux) !=0 ) return 0;

    // verifico che la terza sia concatenazione senza la parte comune
    for (i=0; i< lunprima - luntolto; i++)
        if (prima[i] != terza[i]) return 0;

    for (j= luntolto; j<lunseconda; j++, i++)
        if (seconda[j] != terza[i]) return 0;

    return 1;
}

```

Esercizio 3:

Implementare la seguente funzione:

```
int verifica(ListaParole lista);
```

La funzione, data una lista di tipo ListaParole, deve restituire 1 se la lista ha esattamente tre elementi e se questi, nell'ordine, costituiscono una cerniera (se si riutilizzano funzioni ausiliarie definite negli esercizi precedenti, devono essere reinserite).

Soluzione

```

int verifica(ListaParole lista){
    if (lista == NULL) || (lista->next == NULL) || (lista->next->next== NULL)
        ||(lista->next->next->next != NULL)
        return 0;
    return cerniera(lista->val, lista->next->val, lista->next->next->val);
}

```