Esercizio - Algebra di Boole, Aritmetica Binaria, Codifica delle Informazioni (2,5 punti)

(a) Si costruisca la tabella di verità della seguente espressione Booleana. Si badi alle precedenze! [***1*** *punto*]

A and (not B or not C) or B and (not A or not C) *con altra notazione:*
A(B+C) + B(A+C)

0	0	1	0	1	1	0	**0**	0	0	1	0	1	1	0
0	0	1	0	1	0	1	**0**	0	0	1	0	1	0	1
0	0	0	1	1	1	0	**1**	1	1	1	0	1	1	0
0	0	0	1	0	0	1	**1**	1	1	1	0	1	0	1
1	1	1	0	1	1	0	**1**	0	0	0	1	1	1	0
1	1	1	0	1	0	1	**1**	0	0	0	1	0	0	1
1	1	1	1	1	1	0	**1**	1	1	0	1	1	1	0
1	0	0	1	0	0	1	**0**	1	0	0	1	0	0	1

(b) Si stabilisca il minimo numero di bit sufficiente a rappresentare in complemento a due entrambi i numeri A = **49**$_{dec}$ e B = **−73**$_{dec}$, li si converta, se ne calcolino la somma (A + B) e la differenza (A − B) in complemento a due e si indichi se si genera riporto dalla colonna dei bit più significativi e se si verifica overflow. [***1,5*** *punti*]

Con 8 bit si rappresentano in C_2 i numeri da -128_{dec} a $+127_{dec}$; 7 bit non sono sufficienti per B.

Col metodo dei resti in binario assoluto, e poi in C_2:

$49_{dec} = 110001_{bin} = 00110001_{C2}$
$73_{dec} = 1001001_{bin} = 01001001_{C2}$
$-73_{dec} = 10110110 + 00000001 = 10110111_{C2}$

Applicando l'algoritmo di addizione, ricordando che 49-(-73) si calcola come 49+73, e indicando (riporto) e [overflow] con le rispettive parentesi:

		0 0 1 1 0 0 0 1	49_{dec}
+		1 0 1 1 0 1 1 1	-73_{dec}
=	(0) [0]	1 1 1 0 1 0 0 0	-24_{dec}
		0 0 1 1 0 0 0 1	49_{dec}
+		0 1 0 0 1 0 0 1	73_{dec}
=	(0) [0]	0 1 1 1 1 0 1 0	122_{dec}

In entrambi i casi non si genera riporto dalla colonna dei bit più significativi, né si genera overflow (il risultato non eccede mai l'intervallo di rappresentazione).

Esercizio - Struttura e allocazione della memoria, sintesi di codice - (7,5 punti)

La seguente dichiarazione definisce il catalogo prezzi nel sistema informativo di un piccolo supermercato, su una piattaforma in cui **`sizeof(long)=8, sizeof(int)=sizeof(float)=sizeof(void *) = 4`**

```
typedef char Descrizione[40];
typedef struct {      unsigned long barcode;
                  Descrizione   descr;
                  float         prezzo;
                  int           noPunti;
                  char        * descrEstesa; } Articolo;
typedef struct {     Articolo      list[1000];
                  int           numArticoli; } CatalogoGenerale;
CatalogoGenerale cat;
```

`noPunti` è un booleano che identifica gil articoli su cui per legge non si possono effettuare promozioni o fidelizzazioni; **`numArticoli`** indica quanti dei 1000 elementi di **`list`** sono effettivamente utilizzati.

(a) Si calcoli quanto vale **`sizeof(cat)`** [***1** punto*]

sizeof(Descrizione) = 40 /* i char occupano un byte in base alla standardizzazione ANSI */
sizeof(Articolo) = 8 + 40 + 4 + 4 + 4 = 60 byte (= 15 celle)
sizeof(cat) = 1000 x 60 + 4 byte = 600004 byte = (15001 celle)

(b) Sapendo che le parole di memoria sono di 32 bit e che **`cat`** è allocato a partire dall'indirizzo 2348, si calcoli l'indirizzo della cella contenente la variabile **`cat.list[20].noPunti`** [***0,5** punti*]

`&(cat.list[20].noPunti) =`
2348 + (20 x sizeof(Articolo) + sizeof(long) + sizeof(Descrizione) + sizeof(float)) / 4 celle/byte =
2348 + (20 x 60 + 8 + 40 + 4) / 4 = 2348 + 1252 / 4 = 2348 + 313 = 2661

(c) Si implementi una funzione C **`...stampa(...)`** che riceve come parametro un **`Articolo`** (passato per indirizzo) e stampa su stdout una riga così come deve apparire sullo scontrino (idealmente, in modo che righe consecutive abbiano un corretto incolonnamento di descrizioni brevi e prezzi). [***1** punto*]

```
int stampa( Articolo *a ) {
  if( a == NULL )
    return -1;
  printf( "%40s%3.2f\n", a->descr, a->prezzo );
  return 0;
}
```

(d) Si consideri che: (a) i cataloghi contengono gli articoli in ordine di codice a barre crescente, e (b) non possono esservi due articoli con lo stesso codice a barre. Si implementi una funzione C **`...trova(...)`** che riceve come parametri un catalogo di articoli (passato per indirizzo) e un codice a barre, e restituisce al chiamante l'indirizzo dell'articolo identificato dal codice, o NULL se il codice non corrisponde a nessun articolo in catalogo. [***2** punti*]

La soluzione più semplice è probabilmente:

```
Articolo * trova( CatalogoGenerale *cg, unsigned long cb ) {
  Articolo * a = NULL;
  int i = 0;
  while( a == NULL && i < cg->numArticoli && cb <= cg->list[i].barcode ) {
    if( cg->list[i].barcode == bc )
      a = &( cg->list[i]);
    ++i;
  }
  return a;
}
```

Se si continua sul retro di qualche foglio, indicare quale

Si noti che, in virtù del terzo congiunto nella condizione del ciclo while, se il codice a barre del prodotto i-esimo supera quello del prodotto cercato la ricerca si interrompe, evitando di cercare nella parte restante del catalogo. Questo permette, sfruttando l'ordinemanto del vettore `cb->list`, di scandire, in media, solo metà dei prodotti a catalogo. La complessità computazionale, però, in questo caso è ancora lineare.

Per sfruttare appieno l'ordinamento del vettore e ottenere una complessità logaritmica occorre, ad esempio, dimezzare ad ogni controllo la dimensione dello spazio di ricerca (cioè della porzione di vettore ancora da esplorare). Si può tener traccia degli indici "low" e "high" procedendo in modo iterativo, oppure procedere ricorsivamente (re-invocando la funzione di ricerca di volta in volta su metà del vettore). Vediamo come si può fare con una soluzione ricorsiva:

```
Articolo * trova( CatalogoGenerale *cg, unsigned long cb ) {
  return dicotofind(cg,cb,0,cg->numArticoli-1);
}
```

La prima chiamata serve a "mascherare" il fatto che la nostra ricerca dicotomica ricorsiva necessita anche dei due parametri che indicano l'inizio e la fine della porzione di vettore in cui effettuare la ricerca. La soluzione ricorsiva vera e propria è la seguente

```
Articolo * dicotofind( CatalogoGenerale *cg, unsigned long cb, int da, int a ) {
  int mezzo = (da+a)/2;
  if( da > a )                          // gli indici si sono incontrati e superati
    return NULL;
  if( cg->list[mezzo].barcode == cb )  // abbiamo trovato l'articolo
    return &(cg->list[mezzo]);
  if( cg->list[mezzo].barcode < cb )    // proseguiamo nella seconda metà
    return dicotofind(cg,cb,mezzo+1,a);
  return dicotofind(cg,cb,da,mezzo-1);   // proseguiamo nella prima metà
}
```

(e) Si implementi [***2** punti*] una funzione **`...scan(...)`** che legge da stdin una sequenza di interi positivi (di lunghezza ignota e a priori illimitata), terminata dal valore 0, che rappresentano i codici a barre nell'ordine in cui sono letti dallo scanner laser di una delle casse del supermercato. La funzione riceve come parametro il catalogo del supermercato (passato per indirizzo), e:

- Restituisce al chiamante l'importo totale da pagare.
- Stampa su stdout lo scontrino (*attenzione: non si perda tempo a organizzare intestazioni elaborate!*).
- Stampa su stdout i punti fedeltà conseguiti dal cliente (si ha diritto a un punto fedeltà per ogni euro di spesa oltre i primi 5€, senza considerare i prodotti "non promozionabili", come ad esempio giornali e medicinali, per i quali l'attributo **`noPunti`** ha valore 1).

```
float scan( CatalogoGenerale *cg ) {      // assumiamo che cg non sia NULL
  unsigned long cb;
  float tot = nonop = 0.0;    // tot. dello scontrino e tot. per il calcolo punti
  Articolo *a;
  scanf("%d",&cb);
  printf("Benvenuti al PoliMarket, ecco la vostra spesa:\n\n");
  while( cb > 0 ) {           // se non è la "sentinella" (la cassiera dice stop)
    a = trova(cg,cb);         // ricerca dell'articolo nel catalogo
    if( a != NULL ) {         // se non c'è.... non facciamo niente
      stampa(a);              // relativa riga dello scontrino
      tot += a->prezzo;       // aggiorno il totale
      if( ! a->noPunti )      // se è un articolo "promozionabile"
        nonop += a->prezzo;   // aggiorno anche il "totale per i punti"
    }
    scanf("%d",&cb);          // lettura del prossimo articolo
  }
  printf("Totale della spesa:   € %f4.2\n\n", tot);
  printf("Punti di oggi:        %d\n", (nonop>5.0 ? (int)nonop-5 : 0) );
  return tot;
}
```

Se si continua sul retro di qualche foglio, indicare quale

(f) Si spieghi brevemente [***1** punto*] perché è particolarmente opportuno che **`trova()`** e **`scan()`** ricevano il catalogo per indirizzo.

`trova()` deve restituire al chiamante l'indirizzo dell'articolo trovato, quindi deve necessariamente avere a disposizione il catalogo originale e non una sua copia. **`scan()`**, invece, potrebbe ricevere il catalogo per copia.

Il catalogo, inoltre, è una variabile di grandi dimensioni, la cui variabilità nel tempo è anche assai limitata, almeno rispetto alla frequenza con cui è ragionevole che siano invocate entrambe le funzioni (che devono usarlo ogni volta per un tempo piuttosto breve). Benché non debbano modificarlo, quindi, è comunque opportuno che le funzioni ricevano solo un puntatore, invece di sobbarcarsi la copia (onerosa!) dell'intera struttura ad ogni invocazione.

Esercizio - Analisi di codice: ricorsione e record di attivazione - (3 punti)

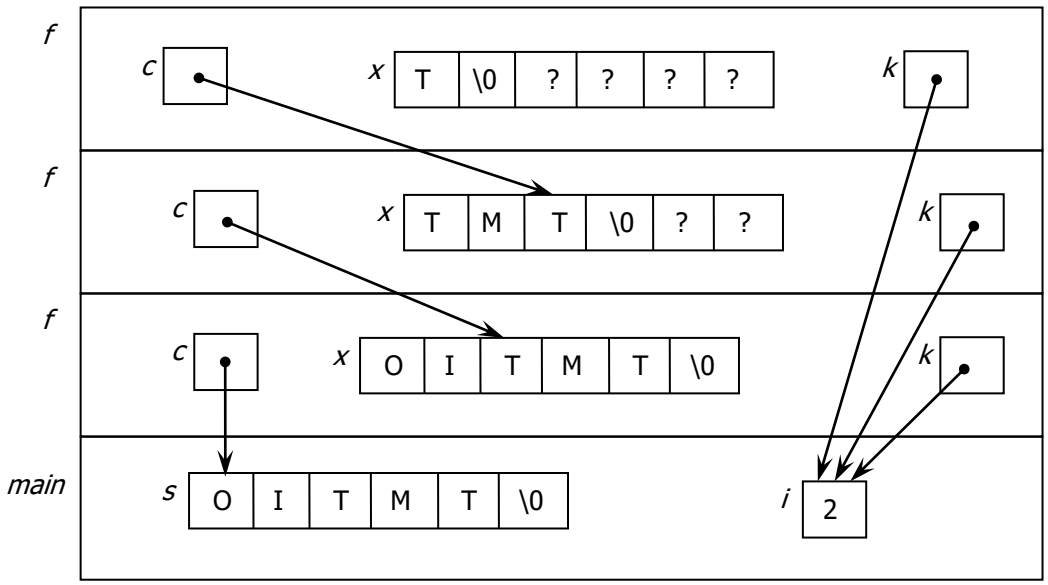
Si consideri il seguente programma:

```
void f(char * c, int *k) {
  char x[6];
  strcpy(x,c);
  printf("%c",*x);
  if( *k > 0 && strlen(x)>2 )
    f(x+2,k);
  return;
}
```

```
int main() {
  char s[] = "OITMT";
  int i;
  for(i=2;i>=0;i--)
    f(s+(i%2),&i);
  return 0;
}
```

(a) Si <u>disegni lo stack dei record di attivazione</u> nel momento in cui la sua dimensione è **massima**. Si rappresentino tutti i record (incluso quello del main), mostrando **tutte** le variabili e adottando le solite convenzioni (vettori: blocchi contigui; puntatori: frecce; valori indefiniti: punti interrogativi). [***2** punti*]

Dall'analisi del codice si evince che nella prima iterazione del ciclo for f() si attiva 3 volte, e nelle successive si attiva rispettivamente due volte e poi una sola volta. Poiché al termine di ogni iterazione i record di f() vengono deallocati fino a tornare al main, l'altezza dello stack è massima proprio al termine della prima iterazione.



Si noti che il vettore x è riallocato ad ogni chiamata, e la funzione strcpy copia porzioni di s nelle varie versioni di x presenti nei diversi record di attivazione. La printf stampa di volta in volta il primo carattere di questi vettori x.

(b) Si indichi l'output stampato dal programma su stdout. [***1** punto*]

L'output del programma lo calcoliamo (naturalmente) considerando tutte le iteraioni del ciclo for. Nella seconda iterazione i vale 1, quindi i%2 vale a sua volta 1, e si stamperanno i caratteri in posizione dispari della stringa s (I e M). Nella terza iterazione i vale 0, quindi la funzione f non richiama se stessa (*k è 0) e stampa solo il primo carattere di s. IL risultato finale è:

OTTIMO

Esercizio - Analisi di codice offuscato - (1 punto)

Si dica, giustificando molto brevemente la risposta, quali proprietà della stringa `p` sono verificate dalle funzioni f() e g().

```
int ff( char *p, char *u ) {
  return  p>=u || *p==*u && ff(++p,--u);
}
int gg( char *p, char *u ) {
  return !(*u) || *p==*u && gg(++p,++u);
}

int f( char *p ) {
  int q = strlen(p);
  return ff(p,p+q-1);
}
int g( char *p ) {
  int q = strlen(p);
  return !(q%2) && gg(p,p+q/2);
}
```

f() passa a ff() due puntatori, al primo e all'ultimo carattere di p. ff() restituisce 1 se [caso base] i due puntatori coincidono o si sono scambiati di posizione (primo successivo a ultimo), oppure se [passo induttivo] i due caratteri puntati sono uguali e la funzione restituisce 1 ricorsivamente sulla sottostringa ottenuta senza considerare gli estremi. Questa è la definizione ricorsiva di una parola **palindroma**.

g() passa a gg() due puntatori ai primi caratteri delle due metà della stringa p se la sua lunghezza è pari, o restituisce immediatamente 0 se la lunghezza è dispari. gg() scandisce in parallelo e "in avanti" le due metà della stringa, verificando che siano uguali e interrompendosi nonappena incontra due caratteri diversi (restituendo 0) o la fine dela stringa (restituendo però 1). La funzione quindi verifica che la parola sia **doppia**, cioè costituita da due sottostringhe consecutive uguali (esempi: PEPE, PAPA, CANCAN, TOMTOM, COUSCOUS, RICARICA, RESTERESTE, e altre parole che siete tutti invitati a trovare (non è facilissimo!!)...).

Si noti come le precedenze tra gli operatori e il fatto che le condizioni sono valutate solo fino a determinarne il valore di verità garantiscono la correttezza e la terminazione delle due funzioni ricorsive ff e gg.

Esercizio - Algebra di Boole, Aritmetica Binaria, Codifica delle Informazioni (2,5 punti)

(c) Si costruisca la tabella di verità della seguente espressione Booleana. Si badi alle precedenze! [***1*** *punto*]
(A or (not B and C) and B) and (A or not (B or C)) *con altra notazione:*
(A+(BC)B)(A+(B+C))

(d) Si stabilisca il minimo numero di bit sufficiente a rappresentare in complemento a due entrambi i numeri A = $\mathbf{54}_{dec}$ e B = $\mathbf{-78}_{dec}$, li si converta, se ne calcolino la somma (A + B) e la differenza (A – B) in complemento a due e si indichi se si genera riporto dalla colonna dei bit più significativi e se si verifica overflow. [***1,5*** *punti*]

Esercizio - Struttura e allocazione della memoria, sintesi di codice - (7 punti)

Il cinema multisala PoliPictures ha NS sale, tutte uguali, ognuna dotata di NP posti, in cui proietta, a beneficio esclusivo degli studenti e di loro invitati, pellicole di carattere didattico. NS e NP sono costanti definite a livello globale. Le seguenti dichiarazioni definiscono la struttura dati per la gestione dei posti e degli spettacoli, e il formato di un biglietto. Nella piattaforma **`sizeof(int)= sizeof(void *) = 4`** e **`sizeof(long int) = 8`**

```
typedef long int Matricola;
typedef struct {      char   titolo[32], regista[22], attori[42];
                      int    durataInMinuti; } Film;
typedef Film elencoFilm[100];
typedef struct {      char   titolo[32], oraInizio[6], nomesala[6];
                      int    numeroPosto; } Biglietto;
typedef struct {      Film  * film;
                      char  oraInizio[6];
                      Matricola postiassegnati[NP]; } Spettacolo;
typedef struct {      char  nome[6];
                      int numeroProiezioni;
                      Spettacolo proiezioni[10]; } Sala;
typedef Sala PoliPictures[NS];
```

In vettore elencoFilm contiene la descrizione di tutti i cento film disponibili in cineteca, da classici come "Sangue e Arena" ai più recenti "Interseption" e "Bastardi senza Lode".

Le sale contengono, ognuna in un vettore, gli spettacoli del giorno previsti per la sala, e ogni spettacolo ha al suo interno un vettore che tiene traccia dell'assegnamento dei posti (che sono numerati da 0 a NP-1) agli studenti che li hanno prenotati (tramite il numero di matricola). I posti non assegnati sono marcati con una matricola pari a zero. Il numero di proiezioni giornaliere previste in ogni sala è dato da numeroProiezioni, e il vettore proiezioni è spesso usato solo in parte.

(a) Si calcoli quanto vale **`sizeof(PoliPictures)`**, espresso in funzione di NS e NP. [***1*** *punto*]

Sizeof(PoliPictures) = NS x sizeof(Sala) = NS x (6 + 4 + 10 x sizeof(Spettacolo)) =
= NS x (10 + 10 x (4 + 6 + sizeof(postiassegnati))) =
= NS x (10 + 10 x (10 + NP x sizeof(Matricola))) =
= NS x (10 + 10 x (10 + NP x 8)) = 10NS (11 + 8NP)

(b) Si implementi <u>una funzione</u> C **`...stampa(...)`** che riceve come parametro un **`Biglietto`** e lo stampa in modo ordinato e leggibile, per essere esibito al rettore in caso di un controllo [***1*** *punto*]

(c) Si implementi una funzione C `...contaposti(...)` che conta il numero di posti assegnati per la giornata in tutto il cinema per la visione di un dato film, identificato in base al titolo passato come parametro. Attenzione: il film potrebbe essere in proiezione in più sale e in orari diversi anche più volte nella stessa sala. [***2,5** punti*]

(d) Si implementi una funzione C `...assegnaposti(...)` che assegna **n** posti consecutivi allo studente con matricola **mat** per il film di titolo **tit**. Se non ci sono posti disponibili restituisce -1, altrimenti assegna i posti e restituisce 0. [***2,5** punti*]

Esercizio - Ricorsione - (1,5 punti)

La funzione discreta f ha dominio e codominio definiti sui numeri interi, e definisce una serie in cui ogni valore è pari alla differenza dei valori precedenti nella serie, e in particolare:

$f(n) = f(n-1) - f(n-2)$ per $n > 1$

Inotre si ha, per definizione della funzione stessa:

$f(n) = n$ per $n = 0$ e per $n = 1$

(a) Si definisca una funzione C **ricorsiva** che restituisce i numeri della serie in funzione di n.

Applichiamo direttamente la definizione!

```
int funz( int n ) {
  if( n == 0 || n == 1 )
    return n;
  return funz( n-1 ) - funz( n-2 );
}
```

(b) Si noti che f(0) = f(6) = 0, f(1) = f(7) = 1, etc... cioè i valori 0, 1 e -1 si ripetono periodicamente e indefinitamente nella serie, con un periodo di sei elementi. Questo causa problemi di terminazione alla funzione definita prima? Perché?

Non c'è nessun problema di terminazione, perché il caso base è definito per valori minimi di n, e il parametro viene decrementato ad ogni invocazione ricorsiva. I valori restituiti non intervengono in alcun modo nella determinazione di quali chiamate vengono effettuate.

(c) Sfruttando il fatto che i numeri si ripetono periodicamente nella serie, come osservato al punto (b), si proponga una soluzione che non faccia uso **né di iterazione né di ricorsione** ma sia equivalente a quella definita al punto (a), pur eseguendo in un tempo costante.

Il fatto che l'output sia periodico suggerisce di ragionare in base al resto della divisione di n per opportuni divisori, ad esempio come segue:

```
int funz( int n ) {
  if( n % 3 == 0 )
    return 0;
  if( n % 6 < 3 )
    return 1;
  return -1;
}
```

Esercizio - Analisi di codice: ricorsione e record di attivazione

Si consideri il seguente programma. Attenzione! Il parametro c è un puntatore passato per indirizzo :

```
void f(char ** c, int x) {
  void *z = &z;
  if( strlen(*c) < 2 )
    return;
  printf( "%c", (**c)+x );
  (*c)++;
  f( c, --x );
  return;
}
```

```
int main() {
  char *p, s[6] = "DBPAY";
  int a = 3;
  p = s;
  printf("%c", *(p++));
  f( &p, a );
  printf("%c !!", *p);
  return 0;
}
```

(c) Si disegni lo stack dei record di attivazione nel momento in cui la sua dimensione è **massima**. Si rappresentino tutti i record (incluso quello del main), mostrando **tutte** le variabili e adottando le solite convenzioni (vettori: blocchi contigui; puntatori: frecce; valori indefiniti: punti interrogativi).

(d) Si indichi la linea stampata dal programma su stdout.

Esercizio

Dire cosa stampa il seguente codice:

```
int inv(int val, char* a, char* b){
 int start=val, i, i1, i2, tot=0;
 char ch;
 while(a[i1]!='\0') i1++;
 while(b[i2]!='\0') i2++;

 while(start<i1 && start<i2){
  for(i=start;i<i1;i++){
   ch=*(a+i);
   *(a+i)=*(b+i2-i-1);
   *(b+i2-i-1)=ch;
   tot++;
  }
  start=start+2;
 }
 return tot;
}

int main(){
 char c,s1[]="abcde",s2[]="12345678";
 int tot=inv(1,s1,s2);
 printf("%s %s %d\n", s1, s2, tot);
 return 0;
}
```

Soluzione

Le variabili i1 e i2 non sono inizializzate.
Il programma ha quindi un comportamento non prevedibile.

Se si usa un compilatore che invece inizializza gli interi a 0 l'output è
a76de 12345cb8 6

Esercizio

Si dica cosa stampa il seguente codice e si spieghi cosa fa la funzione f

```
void f( int a, int b);
int h( int a, int * b, int c );
int g( int *a, int b);

int main() {
  char c;int i, v[4] = { 8, 21, 27, 4 };
  for( i=0; i<4; i++ ) {
    printf("\n%d: ", v[i]); f(v[i],2);
  }
  return 0;
}

void f( int a, int b) {
  int c = g(&a,b);
  if( c > 1 )
    printf("%d ", c);
  if( a > 1 )
    f(a,b);
}

int h( int a, int * b, int c ) {
  if( *b % a != 0 )
    return h( a+c, b, c );
  *b = *b / a;
  return a;
}

int g( int *a, int b) {
  if( *a <= 0 )
    exit(1);
  if( *a == 1 )
    return 1;
  return h(b,a,1);
}
```

Esercizio

SimpleScrabble è una versione del gioco *Scrabble* dove il valore delle parole dipende solo dalle lettere che le compongono. Ogni lettera ha un valore (da 1 a 10) ed è disponibile in un certo numero di esemplari (da 1 a 12). Il valore di una parola è la somma dei valori delle sue lettere. Gli array **`valore`** e **`numero`**, definiti nell'ambiente globale come segue, indicano il valore e il numero di esemplari disponibili di ognuna delle 26 lettere, in ordine alfabetico. Consideriamo per semplicità solo le lettere maiuscole.

```
int numero[26] = {9,2,2,4,12,2,3,2,9,1,1,4,2,6,8,2, 1,6,4,6,4,2,2,1,2, 1};
int valore[26] = {1,2,3,2, 1,4,2,4,1,8,5,1,3,1,1,3,10,1,1,1,1,4,4,8,4,10};
```

Ci sono quindi 9 `A` da 1 punto, 2 `B` da 2 pt, 2 `C` da 3 pt, 4 `D` da 2 pt, 12 `E` da 1 pt, …, 1 `Z` da 10 pt.

Una funzione **`punteggio(…)`** calcola e restituisce il valore della parola ricevuta come parametro, assegnando 0 alle parole *non valide*. Una parola è valida se è di almeno 2 lettere ed è ottenibile con le lettere a disposizione (`CAB` vale 3+1+2=6 pt, `KARAOKE` non è valida, perché disponiamo di una sola `K`).

Si analizzino le funzioni **`punteggio(…)e valida(…)`** date sotto e si correggano gli errori contenuti nelle funzioni (gli errori totali sono quattro, due per ognuna delle due funzioni)

```
int punteggio ( char parola[] ) {
  int p = 0, l;
  l = strlen(parola);
  if ( l >= 2 && valida(parola) )
    for ( ; l >= 0 ; l++ )
      p += valore[parola[l]-'A'];
  return p;
}

int valida ( char parola[] ) {
  int i, p, k=0;
  for ( i=0 ; i<=26 ; i++ ) {
    for( p=0 ; p<strlen(parola) ; p++ )
      if ( parola[p] == 'A'+i )
        k++;
    if ( k > numero[i] )
      return 0;
  }
  return 1;
}
```

Esercizio

```
typedef struct node { char c[8]; struct n *neat; } elem;
elem * f( elem * a, char * s ) {
  elem * n;
  if( strlen(s) > 0 ) {
    n = (elem *) malloc(sizeof(elem));
    n->neat = a;
    strcpy( n->c, s++ );
    n->c[1] = '\0';
    *(n->c) += strlen(s);
    return f( n, s );
  }
  return a;
}

void printr( elem * r ) {
  if( r ) {
    printf("%s", r->c);
    printr( r->neat );
  }
}

int main() {
  char p[] = "AMNG" ;
  printr( f( NULL, p ) );
  return 0;
}
```

Si indichi la linea stampata dal programma sullo standard output.