Puntatori a matrici, Dati astratti, Input-Output su File

Vitoantonio Bevilacqua

bevilacqua@poliba.it

Parole chiave: Puntatore a matrice, Struttura dati astratta, LIFO, FIFO, Grafo orientato, Flusso di informazioni, File binario, File formattato.

1 Puntatori a matrici

Analogamente ai puntatori a vettori, una volta dichiarata una matrice, si può avere la necessità di passare l'indirizzo di questa matrice ad una funzione che attraverso un puntatore deve lavorare sui suoi elementi. In C la semplice variabile A, così come per i vettori, corrisponde all'indirizzo della prima componente della matrice e questo significa quindi che `A=&A[0][0]` è il valore da dover passare come parametro attuale della funzione.

Ma poiché in C una matrice può essere pensata come un vettore colonna fatto di puntatori ciascuno dei quali punta ad un vettore riga, indipendentemente dal fatto che la matrice verrà utilizzata tutta, il parametro corretto per passare una matrice a una funzione è un argomento di tipo puntatore che specifica la dimensione massima delle colonne che si vogliono allocare; questo significa che la definizione di una funzione, ad esempio `leggi_matrice` che dovrà avere tra gli argomenti una matrice, va fatta nel seguente modo:

```
#include <stdio.h>
#define[1] MAXR 13 /*Numero massimo di righe*/
#define MAXC 13 /*Numero massimo di colonne*/

void leggi_matrice(int(*)[MAXC],int,int);
```

Questo argomento che a primo avviso sembra dover essere del tipo `int**` per come abbiamo considerato la matrice precedentemente, è più correttamente quello appena indicato nell'esempio, perché un argomento del tipo `int**` non riesce correttamente a far capire al compilatore qual'è il numero massimo di colonne per poter gestire la matrice, cioè non è in grado di far capire al compilatore di quanti byte è necessario spostarsi per passare da una riga alla successiva.

Tutto ciò significa che la funzione verrà dichiarata nel seguente modo, con i seguenti parametri formali:

---

[1] la direttiva "define" è utilizzata per associare una sequenza di caratteri ad un identificatore: in compile time all'occorrenza il compilatore sostituirà all'identificatore la sequenza di caratteri associata; la sintassi è: `#define nome_identificatore sequenza_caratteri`

```
void leggi_matrice(int(*A)[MAXC],int R,int C)
```

La variabile A viene interpretata come una variabile di tipo puntatore ad un vettore riga fatto di massimo MAXC elementi.

2 Costrutto `switch-case`

Le decisioni a più vie possono essere risolte utilizzando il costrutto `switch-case`, che consente di implementare decisioni multiple attraverso il confronto fra il valore di un'espressione (int o char, che può essere un valore dato da tastiera precedentemente) e un insieme di valori costanti, ciascuno dei quali fa eseguire il rispettivo corpo di istruzioni.

```
switch (espressione)
{
  case costante1:
        istruzioni
        break;
  case costante2:
        istruzioni
        break;
  ...
  case costanteN:
        istruzioni
        break;
  default:
        istruzioni
        break;
}
```

In fase di esecuzione, viene valutata `espressione` e il risultato viene confrontato con `costante1`: se i due valori sono uguali il controllo passa alla prima istruzione che segue i due punti corrispondenti, altrimenti si prosegue confrontando il valore di `espressione` con `costante2`, e così via.

Una volta che il controllo è trasferito a una certa istruzione vengono eseguite tutte le istruzioni presenti nel `case`, fino a che non si raggiunge l'istruzione `break`[2] che causa l'immediata uscita dallo `switch`.

Se `espressione` non è uguale a nessuna delle costanti, l'eventuale presenza del caso particolare `default` passa il controllo alla prima istruzione che segue i due punti corrispondenti.

---

[2] così come `break` permette l'immediata uscita da un ciclo, è anche possibile uscire dall'intero programma attraverso la funzione `exit()`, presente nella libreria `stdlib.h`, la cui sintassi è: `exit (valore_di_ritorno);` dove `valore_di_ritorno` è il valore restituito al sistema operativo, di solito 0 per indicare una normale uscita dal programma e 1 per indicare una situazione di errore

3 Strutture di dati astratte

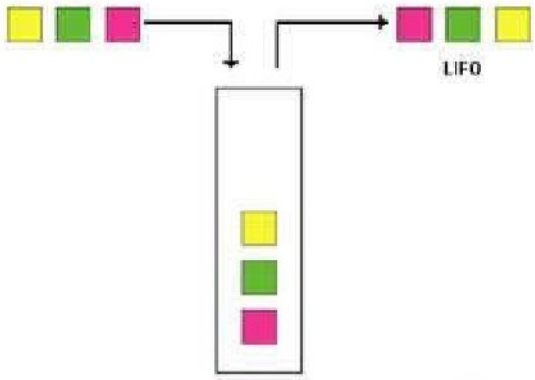
Finora abbiamo sempre operato con strutture di dati concrete, ossia strutture dati che in qualche maniera specificano la forma nella quale i dati vengono memorizzati in memoria (un esempio di struttura di dati concreta è fornito dai vettori, che hanno la proprietà per la quale i singoli elementi sono adiacenti in memoria).

Introduciamo ora le strutture dati di tipo astratto: sono strutture dati che non specificano il modo in cui i dati vengono memorizzati in memoria, ma ciascuna definisce un particolare comportamento di tipo logico nell'organizzazione dei dati. La rappresentazione di una struttura di dati astratta fatta mediante strutture dati concrete è detta "implementazione".

Le strutture astratte che qui di seguito definiremo e implementeremo sono la Pila, la Coda e il Grafo.

3.1 Pila

La "pila" o "stack" è una struttura di dati astratta, monodimensionale, costituita da elementi omogenei, ad esempio di tipo `int`, ordinati in una sequenza. La pila segue la logica di gestione dei dati LIFO (Last In First Out): le operazioni di inserimento ed eliminazione/estrazione di un elemento della pila sono effettuale dallo stesso estremo della sequenza, detto "testa della pila", dunque l'ultimo elemento inserito è quello che per primo viene estratto/eliminato.



Una pila può essere implementata o utilizzando i vettori o utilizzando le "liste", ma del secondo caso parleremo nella prossima dispensa.

3.2 Implementazione di una pila mediante vettori

Dovendo utilizzare i vettori, è palese che sarà il vettore stesso a contenere la sequenza delle informazioni di una pila; si deve inoltre

- stimare il numero massimo di elementi che la pila può contenere (la "profondità" di una pila), ciò verrà fatto mediante le seguenti linee di codice:

```
#define PROF_PILA 5
int pila[PROF_PILA];
```

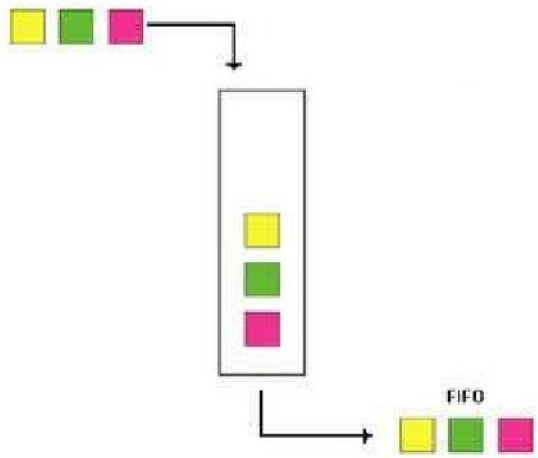
- utilizzare una variabile di tipo intero, `punt_testa`, detta "stack-pointer" come riferimento alla testa della pila, ovvero all'elemento del vettore dove effettuare il successivo inserimento: supponendo che la profondità della pila valga 5, `punt_testa` potrà avere valore compreso tra 0 e 4 (se vale 0 indica che la pila è vuota, se vale 5 vuol dire che la pila è piena)
- dichiarare e poi definire le funzioni che si occuperanno dell'inserimento di un elemento, della eliminazione di un elemento e della visualizzazione della pila.

Nel `main` vi è il costrutto `switch` con al quale in runtime di volta in volta viene chiesta quale operazione eseguire (inserimento, eliminazione/estrazione, visualizzazione, uscita) e all'interno di ogni `case` vi è la chiamata delle opportune funzioni regolata dal controllo sul valore `punt_testa`.

Il codice di implementazione di una pila mediante vettore è presente in Appendice.

3.3 Coda

La "coda" o "queue" è, come la pila, una struttura di dati astratta, monodimensionale, costituita da elementi omogenei ordinati in una sequenza. La coda però segue la logica di gestione dei dati FIFO (First In First Out): il primo elemento inserito è quello che per primo viene estratto; dunque l'inserimento di un elemento avviene dopo l'ultimo elemento inserito in sequenza, come per la pila, mentre l'eliminazione è effettuata sul primo elemento della sequenza.



Analogamente a quanto detto per la pila, anche una coda può essere implementata o utilizzando i vettori o utilizzando le "liste", ma del secondo caso parleremo nella prossima dispensa.

3.4 Implementazione di una coda mediante vettori

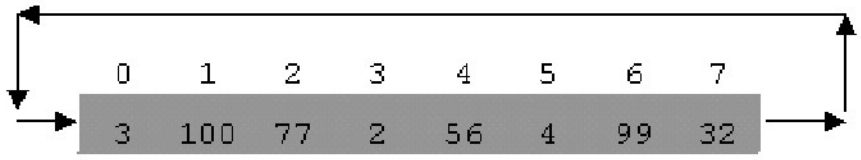
La gestione di una struttura astratta di tipo coda viene implementata in modo sostanzialmente analogo a quanto avvenuto per la pila, ad eccezione delle funzioni di inserimento e di eliminazione, poiché queste operazioni vengono effettuate agli estremi opposti della sequenza.

Il comportamento di tipo FIFO per poter essere implementato ha bisogno di due indici: un puntatore di inserimento, `punt_ins`, e un puntatore di estrazione, `punt_el`, che all'inizio punteranno entrambi sul primo elemento utile della coda, ovvero varranno entrambi 0.

Successivamente, ogni volta che si deve inserire un elemento, il puntatore di inserimento viene incrementato di una unità, invece ogni volta che si estrae viene incrementato il puntatore di estrazione.

Quando il puntatore di inserimento raggiunge il valore di lunghezza massima della coda, vuol dire che la coda è piena, mentre quando il puntatore di estrazione raggiunge il puntatore di inserimento (ossia hanno lo stesso valore, indipendentemente dall'elemento cui fanno riferimento) vuol dire che la coda è stata completamente gestita.

Si può tuttavia presentare un problema: quando il puntatore di inserimento ha raggiunto il valore massimo si è tenuti a pensare che la coda sia piena ma se almeno un elemento della coda è stato gestito, il puntatore di estrazione è stato incrementato, quindi tutte le componenti del vettore che si trovano prima del puntatore di estrazione sono, nella logica FIFO, tornate libere, dunque la coda in realtà non è più piena. Per poter risolvere questo problema il vettore viene gestito ciclicamente, ossia quando il puntatore di inserimento arriva alla fine del vettore, lo si riporta all'inizio.



Per poter implementare la gestione ciclica del vettore, si utilizza la seguente istruzione che ricalcola il valore di `punt_ins` subito dopo l'inserimento di un nuovo elemento:

```
*punt_ins=((*punt_ins+1)%LUNG_CODA);
```

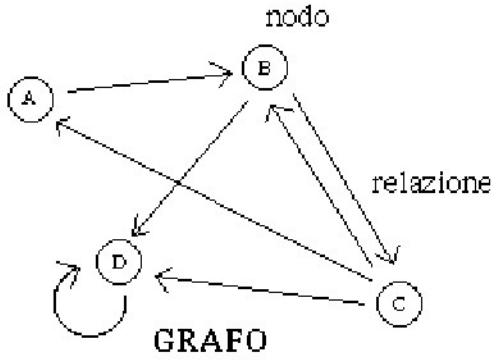
Il codice di implementazione di una coda mediante vettore è presente in Appendice.

3.5 Grafi

Un grafo è una struttura dati di tipo astratto non lineare.

Un grafo non orientato è caratterizzato da un insieme di nodi, ciascuno con la propria etichetta, e da un insieme di percorsi/archi/relazioni ciascuno dei quali unisce una coppia di nodi, in modo che non ci sia più di un percorso tra una coppia di nodi.

Se invece il grafo è orientato, è ammesso che tra due nodi vi siano due archi, purché di verso opposto. È anche possibile che un arco ricada sullo stesso nodo.



Immaginiamo che A sia il nodo iniziale di un particolare percorso informatico. Partendo dallo stato iniziale, se si verifica una particolare condizione, è possibile transitare sul nodo B e di qui, a seconda della situazione che si verifica, è possibile transitare su C o su D, e così via. Quindi è verosimile pensare che lo stato A è lo stato iniziale, lo stato D è lo stato finale, mentre B e C sono stati intermedi.

Un grafo può anche dirsi pesato se ciascun arco che lo compone ha un suo "costo" di transito; una struttura dati di questo tipo è particolarmente utile nel calcolare, tra tutti i percorsi possibili, quelli con costo minimo, massimo, ecc.; le macchine che si occupano di ciò sono dette "macchine a stati finiti".

Un grafo può essere implementato o utilizzando le matrici o utilizzando le "liste", ma del secondo caso parleremo nella prossima dispensa.

3.6 Implementazione di un grafo mediante matrice

Un modo per memorizzare i grafi orientati è quello di utilizzare una matrice detta "matrice delle adiacenze", in cui ogni riga e ogni colonna rappresentano un nodo.

	A	B	C	D
A	0	1	0	0
B	0	0	1	1
C	1	1	0	1
D	0	0	0	1

Nella matrice appare un 1 se esiste un arco orientato dal nodo di riga al nodo di colonna, uno 0 se non esiste. In particolare, si osservi il nodo D, che ha un arco orientato su se stesso.

Se il grafo è pesato, al posto di ogni 1 appare il costo del rispettivo arco.

In Appendice è presente un programma che memorizza e gestisce un grafo attraverso una matrice di adiacenze.

4 File

Finora abbiamo operato con meccanismi di input/output che coinvolgevano soltanto la tastiera come dispositivo di input e lo schermo come dispositivo di output.

Ora introduciamo alcune funzioni contenute nella libreria `stdio.h` per l'input/output che operano sui "files". Tutto ciò permetterà di accedere a files e di lavorare su questi come se fossero delle periferiche di entrata e uscita.

Il sistema di input/output del linguaggio C fornisce un'interfaccia indipendente dal particolare dispositivo, detto "file" (es: file su disco, terminale, stampante, ecc.), su cui viene eseguito l'accesso, poiché questo è associato ad un flusso di informazioni (stream). Tutti i flussi si comportano poi nello stesso modo. Esistono due diversi tipi di flussi, di testo e binari:

- un flusso di testo è una sequenza di caratteri;
- un flusso binario è una sequenza di byte con corrispondenza uno a uno con quelli della periferica esterna.

4.1 Apertura di un file

L'operazione di apertura di un file consente di accedervi e di associare ad esso un flusso specifico; questo viene poi associato ad una struttura di controllo di tipo `FILE`.

Prima di poter aprire un file, è necessario dichiarare quindi un "puntatore a file" (file-pointer) di tipo `FILE`, il cui scopo è quello di identificare uno specifico file su disco e viene usato dal flusso associato per svolgere le operazioni di i/o.

```
FILE *fp;
```

Per poter aprire un file si utilizza la funzione `fopen` che serve ad aprire un file su disco in una modalità specifica con cui sarà utilizzato e ad associargli la variabile puntatore `fp`. La sintassi della funzione è:

```
fp = fopen ("nome_file","modo_apertura");
```

Le modalità con cui aprire un file sono:

Modo	Significato
"r"	"read": apre un file di testo in modalità di sola lettura; se si verifica un errore (es: il file non esiste) la funzione `fopen` ritorna il codice di errore `NULL`
"w"	"write": apre un file di testo in modalità di sola scrittura. Se il file esiste, verrà sovrascritto. Se il file non esiste, verrà creato automaticamente.
"a"	"append": apre un file di testo con la possibilità di sola scrittura a fine file. Se il file non esiste, verrà creato automaticamente.
"rb"	apre un file binario in modalità di sola lettura; se si verifica un errore (es: il file non esiste) la funzione `fopen` ritorna il codice di errore `NULL`
"wb"	apre un file binario in modalità di sola scrittura. Se il file esiste, verrà sovrascritto. Se il file non esiste, verrà creato automaticamente.
"ab"	apre un file binario con la possibilità di sola scrittura a fine file. Se il file non esiste, verrà creato automaticamente.

Esempio:

```
fp = ("testo.txt","r");
```

Alla notazione di ciascuno di questi modi è possibile aggiungere un "+" per indicare che accanto alla funzione di scrittura è abilitata la funzione di lettura e viceversa, sempre rispettando le specifiche modalità di apertura del file. Esempio:

```
fp = ("testo.txt","r+");
```

Nel caso in cui si verifichi un errore, ad esempio perché il file non esiste, la funzione `fopen` in modalità `r` o `rb` ritorna il valore della macro `NULL`. È quindi possibile operare un controllo al momento di apertura del file:

```
fp=fopen("testo.txt","r");
if(fp==NULL)
          printf("file inesistente");
```

Se il file che si intende aprire non si trova nella cartella in cui è presente ed è stato compilato il codice sorgente, insieme al nome del file occorre anche specificare il suo percorso. Esempio:

```
fp=fopen("C:\Documents and Settings\testo.txt","w");
```

4.2 Lettura e scrittura formattata

Una volta aperto il file con la `fopen`, vi si può accedere attraverso due funzioni principali: la `fprintf` per scrivere e la `fscanf` per leggere. Queste due funzioni trattano i dati in modo formattato, ossia in formato ASCII (come apparirebbero sullo schermo).

La `fprintf`, simile alla `printf`, consente di scrivere su un file di testo; la sua sintassi è:

```
fprintf (fp, "stringa_da_scrivere", argomenti);
```

dove `fp` rappresenta il file sul quale si desidera scrivere.

La `fscanf`, simile alla `scanf`, consente di acquisire un dato da un file di testo e di memorizzarlo in una variabile o vettore in memoria; la sua sintassi è:

```
fscanf (fp, "%specificatore_di_tipo", argomenti);
```

dove `fp` rappresenta il file dal quale si desidera leggere.

La `fscanf` restituisce un valore di tipo intero; se si è arrivati alla fine del file, ritorna il valore della macro `EOF` (End Of File); è quindi possibile operare un controllo come nell'esempio:

```
  j=0;
while(fscanf(fp,"%d",&i)!=EOF)
  {
```

```
        j++;
}
if(j==0)
        printf("File vuoto\n");
```

Esistono inoltre due funzioni per leggere e scrivere un singolo carattere da file:
- la funzione `fgetc` legge un singolo carattere dal file `fp` e lo memorizza nella variabile char c; la sua sintassi è:

```
c = fgetc (fp);
```

- la funzione `fputc` scrive il carattere contenuto in c sul file `fp`; la sua sintassi è:

```
fputc(c,fp);
```

4.3 Lettura e scrittura in binario

Qualora siano richieste particolari caratteristiche di velocità ed efficienza, si dovrà fare ricorso alle funzioni `fread` e `fwrite`, per eseguire rispettivamente le operazioni di lettura e scrittura, che trattano i dati non in modo formattato ma in modo binario.

La funzione `fread` ha la seguente sintassi:

```
n = fread (buf, dimensione, elementi, fp);
```

dove `buf` è il vettore in cui sono memorizzate le informazioni lette dal file, `elementi` indica il numero di elementi da leggere, `dimensione` rappresenta la dimensione in byte di un singolo elemento da leggere, `fp` è il file da leggere. Il valore di ritorno della `fread` è un intero e indica il numero di elementi letti dal file (si noti che tale valore può non coincidere con `elementi` nel caso in cui, ad esempio, il file sia vuoto o contenga un numero di elementi inferiore a quello di `buf`); se tale valore è negativo, vuol dire che è stato commesso qualche errore.

Quando tutto il contenuto del file è stato letto, la funzione `fread` ritorna, nella logica di quanto detto in precedenza, il valore 0 per indicare che l'indicatore di posizione è ormai posizionato a fine file.

La funzione `fread` può essere utile in quanto può leggere il file con una sola chiamata o può essere invocata più volte leggendo a ogni chiamata soltanto una limitata porzione di file. Le operazioni di lettura accedono al file in maniera sequenziale e mantengono traccia del punto in cui si è arrivati nella lettura del file. Esempio:

```
fread (buf, sizeof(int), 100, fp);
```

equivale a:

```
for(i=0;i<100;i++)
          fread (&buf[i], sizeof(int), 1, fp);
```

La funzione `fwrite` ha, analogamente a `fread`, la seguente sintassi:

```
n = fwrite (buf, dimesione, elementi, fp);
```

dove `buf` è il vettore che contiene i dati che devono essere memorizzati nel file, `elementi` indica il numero di elementi da memorizzare, `dimensione` rappresenta la dimensione in byte di un singolo elemento da memorizzare, `fp` è il file dove devono essere memorizzati i dati. Il valore di ritorno della `fwrite` è un intero e indica il numero di elementi che sono stati memorizzati nel file (si noti che tale valore può non coincidere con `elementi` nel caso in cui, ad esempio, il file abbia raggiunto la massima dimensione ammessa); se tale valore è negativo, vuol dire che è stato commesso qualche errore.

4.4 Indicatore di posizione

Il C consente di operare sui file di byte oltre che in modo strettamente sequenziale, anche in modo casuale (random).

L'apertura del file inizializza un indicatore di posizione, posto all'inizio del file, che per ogni lettura o scrittura viene incrementato.

La funzione `fseek` consente di muovere l'indicatore di posizione all'interno del file; la sua sintassi è:

```
err = fseek (fp, n, mode);
```

dove `fp` è il puntatore a file, `n` indica di quanti byte l'indicatore di posizione deve essere spostato (se negativo, lo spostamento è all'indietro), `mode` indica a partire da quale posizione muovere l'indicatore (0 indica l'inizio del file, 1 la posizione corrente, 2 la fine del file). Il valore di ritorno della `fseek` è un intero; se tale valore è negativo, vuol dire che è stato commesso qualche errore.

La funzione `ftell` serve invece per conoscere la posizione corrente dell'indicatore di posizione; la sua sintassi è la seguente:

```
n = ftell (fp);
```

dove `fp` è il file-pointer; la funzione ritorna un valore intero indicante la posizione corrente dell'indicatore di posizione all'interno del file. Se si verifica un errore, ad esempio se il file non è stato aperto, tale valore è negativo.

Infine la funzione `rewind` ripristina l'indicatore di posizione all'inizio del file specificato, ossia riavvolge il file; la sua sintassi è:

```
rewind (fp);
```

dove `fp` è il puntatore a file.

4.5 Chiusura di un file

L'operazione di chiusura dissocia il file dal flusso specifico e, in seguito all'operazione di chiusura, l'eventuale contenuto del flusso a esso associato viene scritto nel dispositivo esterno (in modalità scrittura, le modifiche ad un file non sono apportate finché il file non viene chiuso).

Sebbene quando il programma termina, tutti i files sono chiusi automaticamente, una chiusura inaspettata del programma in runtime dovuta a un errore o il voler riutilizzare lo stesso file-pointer per più files, portano a voler chiudere durante l'esecuzione del programma un file. La funzione che permette di fare ciò è la `fclose` e la sua sintassi è la seguente:

```
fclose (fp);
```

dove `fp` è il puntatore al file da chiudere.

4.6 Cancellazione e rinomina di un file

Per cancellare un file si utilizza la funzione `remove`; la sua sintassi è la seguente:

```
n = remove (fp);
```

dove `fp` è il puntatore al file da eliminare.

Per rinominare un file si utilizza la funzione `rename`, la cui sintassi è:

```
n = rename ("vecchio_nome","nuovo_nome");
```

dove `vecchio_nome` è il nome del file da rinominare, `nuovo_nome` è il nuovo nome del file.

Entrambe le funzioni `remove` e `rename` restituiscono un valore intero diverso da 0 quando viene rilevato un errore, 0 se l'operazione è stata eseguita correttamente.

Ringraziamenti. Il presente capitolo è stato scritto anche grazie al prezioso contributo degli studenti Donato Mancuso, Giuseppe Ragno,
Flavio Palmieri, Pasquale Bonasia.

Riferimenti

1. Bevilacqua, V.: Dispense Linguaggio C In: http://www.vitoantoniobevilacqua.it
2. http://it.wikipedia.org/wiki/Stack
3. http://it.wikipedia.org/wiki/Coda_(informatica)
4. http://it.wikipedia.org/wiki/Grafo
5. http://it.wikipedia.org/wiki/Input/output
6. http://it.wikipedia.org/wiki/Stdio.h

Appendice: Codice in linguaggio C

```
// ALGORITMO DI COMPRESSIONE DI UNA MATRICE
#include <stdio.h>
#define MAXR 13
#define MAXC 13

void leggi_matrice(int(*)[MAXC],int,int);
void binarizza(int(*)[MAXC],int,int);
void stampa_matrice(int(*)[MAXC],int,int);
int comprimi(int(*)[MAXC],int,int,int*);
void stampa_compressione(int*,int);
void decomprimi(int*,int,int(*)[MAXC],int,int);


void main()
{
  int A[MAXR][MAXC],B[MAXR][MAXC];
  int R,C,y;
  int v[MAXR*MAXC];

  do
  {
        printf("inserisci il numero di righe R =");
        scanf("%d",&R);
  }
  while(R<2 || R>MAXR);
  do
  {
```

```
        printf("inserisci il numero di colonne C=");
        scanf("%d",&C);
  }
  while(C<2 || C>MAXC);

  leggi_matrice(A,R,C);
  binarizza(A,R,C);
  printf("la matrice binarizzata risulta essere :\n");
  stampa_matrice(A,R,C);
  y=comprimi(A,R,C,v);
  printf("\nla compressione risulta essere :\n");
  stampa_compressione(v,y);
  decomprimi(v,y,B,R,C);
  printf("\nla decompressione risulta essere :\n");
  stampa_matrice(B,R,C);
}

void leggi_matrice(int(*A)[MAXC],int R,int C)
{
  int i,j;
  for(i=0;i<R;i++)
        for(j=0;j<C;j++)
        {
              do
              {
                    printf("inserisci A(%d,%d)\n",i,j);
                    scanf("%d",&A[i][j]);
              }
              while(A[i][j]<0 || A[i][j]>255);
        }
}

void binarizza(int(*A)[MAXC],int R,int C)
{
  int i,j;
  for(i=0;i<R;i++)
        for(j=0;j<C;j++)
        {
              if(A[i][j]<=127)
                    A[i][j]=0;
              else
                    A[i][j]=1;
        }
}

void stampa_matrice(int(*A)[MAXC],int R,int C)
{
  int i,j;
```

```
  for(i=0;i<R;i++)
        for(j=0;j<C;j++)
              printf("A(%d,%d) =%d\n",i,j,A[i][j]);
}

int comprimi(int(*A)[MAXC],int R,int C,int*v)
{
  int i,j,k,y,z;
  int w[MAXR*MAXC];
  k=0;
  z=1;
  for(i=0;i<R;i++)
        for(j=0;j<C;j++)
        {
              w[k]=A[i][j];
              k=k+1;
        }
  y=0;
  for(i=0;i<k;i++)
  {
        if(w[i]==w[i+1])
              z=z+1;
        else
        {
              v[y]=z;
              y++;
              v[y]=w[i];
              y++;
              z=1;
        }
  }
  return y;
}

void stampa_compressione(int*v,int y)
{
  int i;
  for(i=0;i<y;i++)
        printf("%d",v[i]);
}

void decomprimi(int*v,int y,int(*B)[MAXC],int R,int C)
{
  int w[MAXR*MAXC];
  int i,j,k,z;
  z=0;
  for(i=0;i<y;i+=2)
  {
```

```
        for(k=0;k<v[i];k++)
        {
                w[z]=v[i+1];
                z++;
        }
  }
  z=0;
  for(i=0;i<R;i++)
        for(j=0;j<C;j++)
        {
                B[i][j]=w[z];
                z++;
        }
}


  //ALGORITMO GESTIONE PILA MEDIANTE VETTORE
  #include <stdio.h>
  #define PROF_PILA 5

  void inserisci(int*,int,int*);
  void estrai(int*,int*,int*);
  void visualizza(int*,int);

  void main()
  {
        int pila[PROF_PILA];
        int ele, scelta=-1, punt_testa=0;
        do
        {
        printf("\n1. Inserisci elemento\n");
        printf("2. Estrai elemento\n");
        printf("3. Visualizza pila\n");
        printf("0. Esci\n");
        scanf("%d",&scelta);
        switch (scelta)
        {
                case 1:
                        if(punt_testa>=PROF_PILA)
                                printf("pila piena\n");
                      else
                      {
                              printf("elemento da inserire = ");
                              scanf("%d",&ele);
                              inserisci(pila,ele,&punt_testa);
                      }
                      break;
                case 2:
```

```
                        if(punt_testa==0)
                              printf("pila vuota\n");
                        else
                        {
                              estrai(pila,&ele,&punt_testa);
                              printf("elemento estratto: %3d\n",ele);
                        }
                        break;
                  case 3:
                        if(punt_testa==0)
                              printf("pila vuota\n");
                        else
                              visualizza(pila,punt_testa);
                        break;
                default:
                      printf("non ci sono altre operazioni
                definite su una pila\n");
                      break;
        }
        }
        while(scelta!=0);
}

void inserisci(int*a,int b,int*c)
{
              a[*c]=b;
              *c=*c+1;

}

void estrai(int*a,int*b,int*c)
{
        *c=*c-1;
        *b=a[*c];
}

void visualizza(int*a,int c)
{
        int i=c-1;
        printf("testa della pila --> ");
        for(;i>=0;i--)
              printf("%3d",a[i]);
}


//ALGORITMO GESTIONE CODA MEDIANTE VETTORE
#include<stdio.h>
#define LUNG_CODA 5
```

```
void inserisci(int *, int *, int, int *);
void estrai(int *, int *, int *, int *);
void visualizza(int *, int, int );

void main()
{
	int ele,punt_ins=0,punt_el=0,presenti=0;
	int scelta = -1;
	int coda[LUNG_CODA];
	while(scelta != 0)
	{
	printf("\n1. Inserisci elemento\n");
	printf("2. Estrai elemento\n");
	printf("3. Visualizza elemento\n");
	printf("0. Esci\n");
	scanf("%d", &scelta);
	switch (scelta)
	{
		case 1:
			if (presenti==LUNG_CODA)
				printf("coda piena\n");
			else
			{
				printf("elemento da inserire = ");
				scanf("%d",&ele);
				inserisci(coda,&punt_ins,ele,&presenti);
			}
			break;
		case 2 :
			if (presenti==0 )
			{
				punt_ins=punt_el=0;	/*condizione
di robustezza dell'algoritmo*/
				printf("coda vuota\n");
			}
			else
			{
				estrai(coda,&punt_el,&ele,&presenti);
				printf("elemento	estratto	=	%d",
ele);
			}
			break;
		case 3:
			if (presenti==0 )
			{
				punt_ins=punt_el=0;
				printf("coda vuota\n");
```

```
                    }
                    else
                          visualizza(coda,punt_el,
presenti);
                    break;
      }
      }
}

void inserisci(int *vett,int *p_ins, int ele, int *pres)
{
      vett[*p_ins]=ele;
      *p_ins=((*p_ins+1)%LUNG_CODA);
      *pres=*pres+1;
}

void estrai(int *vett, int *p_el, int *ele, int *pres)
{
      *ele=vett[*p_el];
      *p_el=(*p_el+1)%LUNG_CODA;
      *pres=*pres-1;
}

void visualizza (int *coda, int p, int presenti)
{
      int i;
      printf("ci sono %d elementi nella coda \n", presenti);
      for(i=0;i<presenti;i++)
            printf("%d \n", coda[(p+i)%LUNG_CODA]);
      printf ("\n");
}


//ALGORITMO GESTIONE GRAFO    ORIENTATO   PESATO   MEDIANTE
MATRICE DELLE ADIACENZE
#include <stdio.h>
#include <conio.h>
#define MAX 20


void inserisci_grafo(int(*)[MAX],int);
int percorsi(int,int,int(*)[MAX]);
int acquisisci_nodo();

void main()
{
      int ma[MAX][MAX];
      int n,i,j,k,costo,scelta=-1;
```

```
	while(scelta!=0)
	{
	printf("1. Inserisci grafo\n");
	printf("2. Percorri grafo\n");
	printf("0. Esci\n");
	scanf("%d",&scelta);
	switch (scelta)
	{
	case 1:
		do
		{
			printf("inserisci numero    di   nodi   n=
\n");
			scanf("%d",&n);
		}
		while(n>MAX);
		inserisci_grafo(ma,n);
		printf("\n");
		break;
	case 2:
		costo=0;
		printf("inserire nodo da cui partire =\n");
		i=acquisisci_nodo();
		do
		{
			k=percorsi(i,n,ma);
			if(k!=0)
			{
				do
				{
					printf("\nnodo su cui transitare      =
\n");
					j=acquisisci_nodo();
				}
				while(ma[i][j]==0);
				costo=costo+ma[i][j];
				i=j;
				printf("\ncosto      del     percorso
effettuato finora = %d\n",costo);
				printf("continuare ? (s/n)\n");
				if(getch()=='n')
					break;
			}
		}
		while(k!=0);
	    printf("\n");
		break;
	}
```

```
	}
}

void inserisci_grafo(int(*A)[MAX],int N)
{
	int i,j;
	for(i=0;i<N;i++)
		for(j=0;j<N;j++)
		{
			printf("inserire costo per passare da %c
a %c\n",'A'+i,'A'+j);
			scanf("%d",&A[i][j]);
		}
}

int percorsi(int i, int N, int(*A)[MAX])
{
	int j,k=0;
	for(j=0;j<N;j++)
		if(A[i][j]!=0)
			k++;
	if(k==0)
			printf("il	nodo	%c	e'	lo	stato
finale\n",'A'+i);
	else
	{
		printf("dal nodo %c si puo' andare su\n",'A'+i);
		for(j=0;j<N;j++)
			if(A[i][j]!=0)
				printf("%c		con		costo
%d\n",'A'+j,A[i][j]);
	}
	return k;
}

int acquisisci_nodo()
{
	char c;
	c=getch();
	if(c>96)

		return (c-'a');
	else
		return (c-'A');
}


//ALGORITMO DI ACQUISIZIONE MATRICE DA FILE
```

```
/*la matrice quadrata di dimensione massima 10 è
memorizzata    in    un    file    formattato    "matrice.txt"
sottoforma di unico vettore; al termine vi è un numero
che indica la dimensione della matrice. L'algoritmo deve
leggere dal file, acquisire il vettore, trasformarlo in
matrice e stampare la matrice a video*/
#include <stdio.h>
int dimensione(FILE *);
void acquisisci_matrice(FILE *,int,unsigned char(*)[10]);
void visualizza_matrice(unsigned char(*)[10],int);
void main()
{
	FILE *fp;
	int dim;
	unsigned char A[10][10];
	fp=fopen("matrice.txt","r");
	if(fp==NULL)
		printf("File non esistente\n");
	else
	{
		dim=dimensione(fp);
		fclose(fp);
		if(dim==0)
			printf("Il file e' vuoto\n");
		else
		{
			fp=fopen("matrice.txt","r");
			printf("La     dimensione     della     matrice     e'
%d\n",dim);
			acquisisci_matrice(fp,dim,A);
			visualizza_matrice(A,dim);
			fclose(fp);
		}
	}
}


int dimensione(FILE *fp)
{
	int DIM=0,V;
	while(fscanf(fp,"%d",&V)!=EOF)
		DIM=V;
	return DIM;
}
void acquisisci_matrice(FILE *fp,int DIM,unsigned char(*matrice)[10])
{
	int i,j,k;
	unsigned char vett[101],v;
```

```
	i=0;
	while(fscanf(fp,"%s",&v) != EOF)
	{
		vett[i]=v; i++;
	}
	i=0;
/*passaggio da vettore a matrice*/
	for(j=0;j<DIM;j++)
	{
		for(k=0;k<DIM;k++)
		{
			matrice[j][k]=vett[i];
			i++;
		}
	}
}
void visualizza_matrice(unsigned char(*matrix)[10],int DIM)
{
	int i,j;
	for(i=0;i<DIM;i++)
	{
		for(j=0;j<DIM;j++)
			printf("%c ",matrix[i][j]);
		printf("\n");
	}
}


// UTILIZZO DI FILES e modalità "r", "w", "a"
/* lettura e scrittura utilizzando files e funzioni */
#include <stdio.h>
#include <conio.h>
#define N_MAX_VETT 9
void leggiVet (FILE *, int, int *); /* dichiarazione */
void azzeraSottoMedia (int, int *); /* dichiarazione */
void stampaVet (int, int *); /* dichiarazione */
void scriviVet (FILE *, int,int *); /* dichiarazione */
void main ()
{
	FILE *fileinput, *fileoutput;
	int vet[N_MAX_VETT],N;
	char fileingresso[12], fileuscita[12];
	do
	{
		do
		{
			printf("\n");
```

```
                  printf ("nome file di dati di ingresso:
");
                  scanf ("%s", &fileingresso);
                  fileinput                                    =
fopen(fileingresso,"r");/*creazione file di ingresso in
lettura*/
                  if (fileinput == NULL)
                        printf("Il file %s non esiste \n",
fileingresso);
            }
            while (fileinput == NULL);
            printf ("nome file di dati di uscita: ");
            scanf ("%s", &fileuscita);
            fileoutput      =fopen      (fileuscita,      "a");
/*creazione file di uscita in scrittura*/
            do
            {
                  printf("Inserisci       dimensione       del
vettore N=");
                  scanf("%d",&N);
            }
            while(N<2||N>N_MAX_VETT);
            leggiVet (fileinput, N, vet); /* legge il
vettore dal file */
            printf ("Vettore iniziale:\n");
            stampaVet (N, vet);
/* QUI E' CHIAMATA LA FUNZIONE, USANDO N E vet COME
PARAMETRI ATTUALI */
            azzeraSottoMedia (N, vet);
            printf ("Vettore con componenti sotto la
media azzerate:\n");
            stampaVet (N, vet);
            scriviVet (fileoutput, N, vet);
            fclose (fileinput);
            fclose (fileoutput);
            printf("Vuoi rieseguire? s/n ");
      }
      while(getch()!= 'n');
}

void azzeraSottoMedia (int n, int *v) /* lettura e stampa
di vettore */
{
      int i;
      float media = 0;
      for (i=0; i<n; i++)
            media += v[i];
      media = media/n;
```

```
	for (i=0; i<n; i++)
		if (v[i] < media)
			v[i] = 0;
}

void leggiVet (FILE *f, int n, int *v)/* n il numero di
interi da leggere da f*/
{
	int i;
	for (i=0; i<n; i++)
		fscanf (f, "%d", &v[i]);
}

void scriviVet (FILE *f, int n, int *v)/* n il num. di
interi da scrivere in f*/
{
	int i;
	fprintf (f, "se leggi %d elementi del vettore
\n",n);
	for (i=0; i<n; i++)
		fprintf (f, "%d", v[i]);
	fprintf (f, "\n");
}

void stampaVet (int n, int *v)
{
	int i;
	for (i=0; i<n; i++)
		printf ("%d ", v[i]);
	printf ("\n");
}


//CODICE BATTAGLIA AEREA
#include <stdio.h>
#include <string.h>
#include <conio.h>
struct giocatore
{
	char username[10];
	char password[8];
};
void crea_mappa(FILE *);
void mappa0(int(*)[7]);
int confronta_mappe(int(*)[7],int(*)[7],int*);
void visualizza_mappa(int(*)[7]);
void acquisisci_mappa(FILE *,int(*)[7]);
void main()
```

```
{
      int i,j,k,m;
      FILE *fp,*ma;
      struct giocatore classe[100],newplayer,player,giocatori[2];
      int    scelta=-1,matrix[7][7],matrix2[7][7],campo1[7][7],campo2[7]
[7];
      printf(" _____________________\n");
      printf("|                      |\n");
      printf("|          TOP         |\n");
      printf("|          GUN         |\n");
      printf("|         -----        |\n");
      printf("|       Sfida all'     |\n");

      printf("|     ultimo aereo! |\n");
      printf("|_____________________|\n");
      printf("\n");
      do
      {
            printf("1 Crea nuovo giocatore e mappa aerea\
n2 Login e modifica mappa\n3 Login e bombardamento\n0
Esci\n");
            scanf("%d",&scelta);
            switch (scelta)
            {
            case 1:
                  {
                        fp=fopen("user.txt","r");
                        i=0;
                        while(fscanf(fp,"%s
%s",classe[i].username,classe[i].password) !=EOF)
                              i++;
                        do
                        {
                              printf("Inserisci        nuovo
username e password\n");
                              scanf("%s
%s",newplayer.username,newplayer.password);
                              k=i+1;
                              for(j=0;j<i;j++)


if((strcmp(newplayer.username,classe[j].username))==0     ||
(strcmp(newplayer.password,classe[j].password))==0)
                                          k=j;
                              if(k<i)
                                    printf("Username       o
password gia' in uso; si prega di reinserirli\n");
                        }
                        while(k<i);
```

```
                        printf("Username: %s      Password:
%s\n",newplayer.username,newplayer.password);
                        fclose(fp);
                        fp=fopen("user.txt","a");
                        fprintf(fp,"%s
%s\n",newplayer.username,newplayer.password);
                        fclose(fp);

                        ma=fopen(newplayer.username,"a");
                        crea_mappa(ma);
                        fclose(ma);
                  }
                  break;
            case 2:
                  {
                        fp=fopen("user.txt","r");
                        i=0;
                        while(fscanf(fp,"%s
%s",classe[i].username,classe[i].password) !=EOF)
                        {
                              i++;
                        }
                        if(i==0)
                        {
                              printf("Prima     di    giocare
devi creare i giocatori\n");
                              break;
                        }
                        do
                        {
                              printf("Inserisci username e
password\n");
                              scanf("%s
%s",player.username,player.password);
                              k=i+1;
                              for(j=0;j<i;j++)

      if((strcmp(player.username,classe[j].username))==0
&& (strcmp(player.password,classe[j].password))==0)
                                          k=i-1;
                              if(k>i)
                                    printf("Username         o
password errati; si prega di reinserirli\n");
                        }
                        while(k>i);
                        fp=fopen(player.username,"r");
                        acquisisci_mappa(fp,matrix);
                        printf("La tua attuale mappa aerea
e' la seguente\n");
```

```
                        visualizza_mappa(matrix);
                        printf("Vuoi    riposizionare    gli
aerei? s/n\n");
                        if(getch()=='s')
                        {

      fp=fopen(player.username,"w");
                                crea_mappa(fp);
                                fclose(fp);
                        }
                  }
                  break;
            case 3:
                  {
                        do
                        {
                                fp=fopen("user.txt","r");
                                i=0;
                                while(fscanf(fp,"%s
%s",classe[i].username,classe[i].password) !=EOF)
                                {
                                      i++;
                                }
                                if(i<2)
                                {
                                      printf("Prima          di
giocare devi creare almeno 2 giocatori\n");
                                      break;
                                }
                                do
                                {

                                      printf("Giocatore 1 :
inserisci username e password\n");
                                      scanf("%s
%s",giocatori[0].username,giocatori[0].password);
                                      k=i+1;
                                      for(j=0;j<i;j++)


if((strcmp(giocatori[0].username,classe[j].username))==0
&& (strcmp(giocatori[0].password,classe[j].password))==0)
                                                  k=i-1;
                                      if(k>i)
                                            printf("Username
o password errati; si prega di reinserirli\n");
                                }
                                while(k>i);
                                do
```

```
                                {
                                        printf("Giocatore 2 :
inserisci username e password\n");
                                        scanf("%s
%s",giocatori[1].username,giocatori[1].password);
                                        k=i+1;
                                        for(j=0;j<i;j++)

      if((strcmp(giocatori[1].username,classe[j].username))==0
&&    (strcmp(giocatori[1].password,classe[j].password))==0      &&
(strcmp(giocatori[0].username,giocatori[1].username))!=0)
                                                        k=i-1;
                                        if(k>i)
                                              printf("Username
gia' loggato oppure username e password errati ; si prega
di reinserirli\n");
                                }
                                while(k>i);

      fp=fopen(giocatori[0].username,"r");
                                acquisisci_mappa(fp,matrix);

      ma=fopen(giocatori[1].username,"r");

      acquisisci_mappa(ma,matrix2);
                                printf("Il bombardamento   ha
inizio !!!\n");
                                printf("\nGiocatore     1    :
inserisci le coordinate dei punti da bombardare. Hai     a
disposizione 5 missili !!!\n");
                                mappa0(campo1);
                                printf("\nGiocatore     2    :
inserisci le coordinate dei punti da bombardare. Hai     a
disposizione 5 missili !!!\n");
                                mappa0(campo2);

      i=confronta_mappe(campo1,matrix2,&k);

      j=confronta_mappe(campo2,matrix,&m);
                                printf("Giocatore     1   :\n-
Bersagli colpiti = %d\n- Aerei abbattuti = %d/3\n",i,k);
                                printf("Mappa aerea dopo la
battaglia\n");
                                visualizza_mappa(matrix);
                                printf("\n");
                                printf("Giocatore     2   :\n-
Bersagli colpiti = %d\n- Aerei abbattuti = %d/3\n",j,m);
```

```
                                    printf("Mappa aerea dopo la
battaglia\n");
                                    visualizza_mappa(matrix2);
                                    if(k>m)
                                          printf("The       Winner
is : GIOCATORE 1 con %d aerei abbattuti. ",k);
                                    else if(k<m)
                                          printf("The       Winner
is : GIOCATORE 2 con %d aerei abbattuti. ",m);
                                    else

                                          printf("PARITA'. ");
                                    printf("Congratulazioni\n");
                                    printf("Vuoi     giocare     di
nuovo ? s/n\n");
                              }
                              while(getch()=='s');
                        }
                        break;
                  }
            }
            while(scelta!=0);
}
void crea_mappa(FILE *ma)
{
      int mappa[7][7],i,j,rig,col;
      char riga;
      for(i=0;i<7;i++)
            for(j=0;j<7;j++)
                  mappa[i][j]=0;
      printf("Inserisci 3 aerei\n");
      for(i=0;i<3;i++)
      {
            do
            {
                  printf("Coordinate %d aereo : riga,
colonna\n",i+1);
                  scanf("%s %d",&riga,&col);
                  col=col-1; rig=riga-65;
            }
            while(rig<1 || rig>5|| col<1 || col>5 ||
mappa[rig][col]==1       ||      mappa[rig-1][col]==1       ||
mappa[rig+1][col]==1     ||      mappa[rig][col-1]==1       ||
mappa[rig][col+1]==1);
            mappa[rig][col]=1;
            mappa[rig-1][col]=1;
            mappa[rig+1][col]=1;
            mappa[rig][col-1]=1;
            mappa[rig][col+1]=1;
```

```
	}
	printf("La tua mappa aerea e' la seguente\n");
	visualizza_mappa(mappa);
	for(i=0;i<7;i++)
	{
		for(j=0;j<7;j++)
			fprintf(ma,"%d ",mappa[i][j]);
	}
}
void mappa0(int(*campo)[7])
{
	int i,j,col,rig;
	char riga;
	for(i=0;i<7;i++)
		for(j=0;j<7;j++)
			campo[i][j]=0;
	for(i=0;i<5;i++)
	{
		do
		{
			printf("Inserisci le coordinate (riga,
colonna) in cui lanciare il %d missile\n",i+1);
			scanf("%s %d",&riga,&col);
			col=col-1; rig=riga-65;
		}
		while(rig<0 || rig>6 || col<0 || col>6);
		campo[rig][col]=1;
		printf("Fase attuale dell'attacco\n");
		visualizza_mappa(campo);
	}
}
int		confronta_mappe(int(*campo)[7],int(*matrix)
[7],int*abbattuti)
{
	int i,j,colpiti=0;
	*abbattuti=0;
	for(i=1;i<6;i++)
		for(j=1;j<6;j++)
			if(campo[i][j]==matrix[i][j]		&&
matrix[i][j]==1 && matrix[i-1][j]==1 && matrix[i+1][j]==1
&& matrix[i][j-1]==1 && matrix[i][j+1]==1)
			{
				colpiti++;
				*abbattuti=colpiti;
				matrix[i][j]=0;
				matrix[i-1][j]=0;
				matrix[i+1][j]=0;
				matrix[i][j-1]=0;
```

```
                        matrix[i][j+1]=0;
                  }
      colpiti=0;
      for(i=0;i<7;i++)
            for(j=0;j<7;j++)
                  if(campo[i][j]==matrix[i][j]            &&
matrix[i][j]==1)
                  {
                        colpiti++;
                        matrix[i][j]=0;
                  }
      return colpiti;
}

void visualizza_mappa(int(*MA)[7])
{
      int i,j;
      printf(" 1 2 3 4 5 6 7\n");
      for(i=0;i<7;i++)
      {
            printf("%c ",'A'+i);
            for(j=0;j<7;j++)
                  printf("%d ",MA[i][j]);
            printf("\n");
      }
}
void acquisisci_mappa(FILE *ma,int(*mappa)[7])
{
      int i,j,k,vett[49];
      i=0;
      while(fscanf(ma,"%d",&j) != EOF)
      {
            vett[i]=j; i++;
      }
      i=0;
      for(j=0;j<7;j++)
      {
            for(k=0;k<7;k++)
            {
                  mappa[j][k]=vett[i];
                  i++;
            }
      }
}
```