Capitolo 8

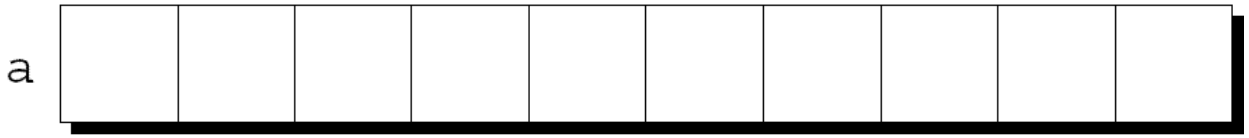
Array

Variabili scalari e variabili aggregate

- Finora abbiamo usato solo variabili ***scalari:*** possono contenere un solo elemento
- Il C permette di usare anche variabili ***aggregate***, che possono contenere insiemi di valori
- Esistono due tipi di variabili aggregate: i vettori (array) e le strutture (structure).

Array monodimensionali

- Un ***array*** è una struttura dati che contiene un certo numero di valori dello stesso tipo
- Questi valori, detti ***elementi dell'array,*** possono essere selezionati individualmente usando la loro *posizione* nell'array
- Gli elementi di un array monodimensionale `a` sono organizzati uno dopo l'altro in quella che possiamo pensare come una riga (o una colonna);



Array monodimensionali

- Per dichiarare un array dobbiamo specificare il *tipo* e il *numero* di elementi:

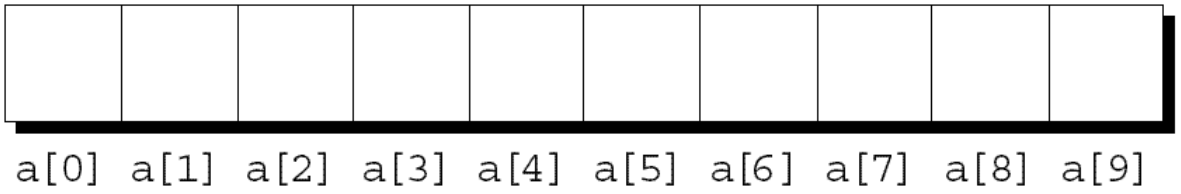
```
int a[10];
```

- Il tipo può essere un tipo qualsiasi mentre il numero può essere specificato da una espressione intera costante.
- Usare una macro per definire la lunghezza di un array è una buona pratica:

```
#define N 10
…
int a[N];
```

Indicizzazione

- Per accedere ad un singolo elemento dell'array si può usare l'**indice** dell'elemento
- Gli elementi di un array di lunghezza n sono indicizzati da 0 ad $n - 1$.
- Se `a` è un array di lunghezza 10, i suoi elementi sono `a[0]`, `a[1]`, ..., `a[9]`:



Indicizzazione

- Espressioni del tipo `a[i]` sono *lvalues*, quindi possono essere usate come normali variabili:

```
a[0] = 1;
printf("%d\n", a[5]);
++a[i];
```

- In generale, se un array contiene elementi di tipo *T*, allora ogni elemento dell'array viene trattato come una variabile di tipo *T*

Indicizzazione

- Molti programmi usano dei cicli `for` nel cui corpo viene eseguita un'operazione su un elemento dell'array: ad ogni iterazione si accede ad un elemento
- Esempi con un array `a` di lunghezza `N`:

```
for (i = 0; i < N; i++)
  a[i] = 0;                    /* clears a */

for (i = 0; i < N; i++)
  scanf("%d", &a[i]);          /* reads data into a */

for (i = 0; i < N; i++)
  sum += a[i];                 /* sums the elements of a */
```

Indicizzazione

- Il compilatore non fa nessun controllo sugli indici utilizzati; se usiamo un indice che va al di fuori dei limiti, il comportamento è impredicibile (spesso disastroso)
- Un errore comune è quello di dimenticare che gli indici vanno da 0 a $n - 1$, non da 1 a n:

```
int a[10], i;

for (i = 1; i <= 10; i++)
  a[i] = 0;
```

In alcuni compilatori, questo `for` causa un ciclo infinito

Indicizzazione

- L'indice può essere un'espressione:

```
a[i+j*10] = 0;
```

- L'espressione può avere effetti collaterali:

```
i = 0;
while (i < N)
  a[i++] = 0;
```

Indicizzazione

- Fare attenzione ai casi in cui l'indice è un'espressione con effetti collaterali:

```
i = 0;
while (i < N)
  a[i] = b[i++];
```

- L'espressione `a[i] = b[i++]` usa il valore di `i` ma lo modifica anche, ed il risultato non è definito
- Il problema può essere evitato spostando in un altro punto l'incremento:

```
for (i = 0; i < N; i++)
  a[i] = b[i];
```

Programma: Invertire una lista di numeri

- Il programma `reverse.c` chiede all'utente di inserire una lista di numeri, e poi la scrive in ordine inverso:

```
Enter 10 numbers: 34 82 49 102 7 94 23 11 50 31
In reverse order: 31 50 11 23 94 7 102 49 82 34
```

- Il programma memorizza i numeri in un array nell'ordine in cui li riceve e poi accede agli elementi dell'array partendo da quello con indice più grande

reverse.c

```
/* Reverses a series of numbers */

#include <stdio.h>

#define N 10

int main(void)
{
  int a[N], i;

  printf("Enter %d numbers: ", N);
  for (i = 0; i < N; i++)
    scanf("%d", &a[i]);

  printf("In reverse order:");
  for (i = N - 1; i >= 0; i--)
    printf(" %d", a[i]);
  printf("\n");

  return 0;
}
```

Inizializzazione

- Un array, come una qualsiasi altra variabile, può essere inizializzato quando viene dichiarato
- La forma più comune per un ***inizializzatore di array*** è una lista di espressioni costanti fra parentesi graffe e separate dalla virgola:

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Inizializzazione

- Se l'inizializzatore è più piccolo dell'array, ai rimanenti elementi verrà assegnato il valore 0:

```
int a[10] = {1, 2, 3, 4, 5, 6};
/* initial value of a is {1, 2, 3, 4, 5, 6, 0, 0, 0, 0} */
```

- Possiamo sfruttare questa caratteristica per inizializzare a 0 l'intero array scrivendo un solo 0:

```
int a[10] = {0};
/* initial value of a is {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} */
```

- Non è permesso usare un inizializzatore più lungo dell'array

Inizializzazione

- Se c'è l'inizializzatore la lunghezza può essere omessa:

```
int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

- Ovviamente il compilatore considererà la lunghezza dell'inizializzatore come lunghezza dell'array

Inizializzatori designati (C99)

- Spesso è necessario inizializzare pochi elementi di un array; agli altri elementi possiamo dare dei valori di default
- Un esempio:

```
int a[15] =
  {0, 0, 29, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 48};
```

- Se l'array è molto grande questo metodo diventa scomodo e soggetto ad errori

Inizializzatori designati (C99)

- Nel C99 esistono gli ***inizializzatori designati*** che possono essere usati per risolvere questo problema

- Ecco un esempio di come funzionano:

```
int a[15] = {[2] = 29, [9] = 7, [14] = 48};
```

- I numeri fra parentesi quadre nell'inizializzatore sono detti ***designatori***

Inizializzatori designati (C99)

- Gli inizializzatori designati sono concisi e facili da leggere (almeno per alcuni array)

- Inoltre, l'ordine in cui specifichiamo gli elementi non ha più importanza

- Ecco un altro modo per avere lo stesso effetto dell'esempio precedente:

```
int a[15] = {[14] = 48, [9] = 7, [2] = 29};
```

Inizializzatori designati (C99)

- I designatori devono essere espressioni intere costanti
- Se l'array ha lunghezza *n*, ogni designatore deve essere un numero fra 0 e *n* – 1.
- Se la lunghezza dell'array viene omessa, un designatore può essere un qualsiasi intero nonnegativo
  - La lunghezza dell'array sarà data dal designatore più grande
- Il seguente array ha 24 elementi:

```
int b[] = {[5] = 10, [23] = 13, [11] = 36, [15] = 29};
```

Inizializzatori designati (C99)

- Un inizializzatore può usare anche un misto fra i due metodi:

```
int c[10] = {5, 1, 9, [4] = 3, 7, 2, [8] = 6};
```

- Gli elementi per i quali non è specificato un valore saranno inizializzati a 0.

Programma: cifre ripetute

- Il programma `repdigit.c` controlla se un numero ha delle cifre ripetute
- Dopo aver fatto inserire un numero il programma stamperà `Repeated digit` oppure `No repeated digit`:

```
Enter a number: 28212
Repeated digit
```

- In questo esempio 28212 ha una cifra ripetuta (il 2); il numero 9357 non ha cifre ripetute

Programma: cifre ripetute

- Il programma usa un array di lunghezza 10 contenente valori booleani
- Inizialmente ogni elemento dell'array `digit_seen` contiene il valore `false`
- Quando viene dato in input il numero `n`, il programma esamina ogni singola cifra e ne memorizza il valore nella variabile `digit`.
  - Se `digit_seen[digit]` è già vero, allora la cifra `digit` è già apparsa e quindi sappiamo che appare almeno 2 volte
  - Se `digit_seen[digit]` vale `false`, allora la stiamo vedendo per la prima volta e quindi il programma cambia `digit_seen[digit]` al valore `true` e procede

repdigit.c

```
/* Checks numbers for repeated digits */

#include <stdbool.h>    /* C99 only */
#include <stdio.h>

int main(void)
{
  bool digit_seen[10]  = {false};
  int digit;
  long n;

  printf("Enter  a number: ");
  scanf("%ld",  &n);
  while  (n > 0)  {
    digit  = n  %  10;
    if  (digit_seen[digit])
      break;
    digit_seen[digit]  = true;
    n  /= 10;
  }
```

```
  if (n > 0)
    printf("Repeated digit\n");
  else
    printf("No repeated digit\n");

  return 0;
}
```

L'operatore `sizeof` con gli array

- L'operatore `sizeof` può essere usato per determinare la lunghezza in byte di un array
- Se `a` è un array di 10 interi ed ogni intero è lungo 4 byte, `sizeof(a)` è 40
- Se usiamo `sizeof` su un singolo elemento dell'array, come `a[0]`, otteniamo la grandezza dell'elemento
- Pertanto l'espressione:

  ```
  sizeof(a) / sizeof(a[0])
  ```

  fornisce la lunghezza dell'array in termini di numero di elementi

L'operatore `sizeof` con gli array

- Alcuni programmatori usano questa espressione quando serve sapere la lunghezza dell'array
- Ecco un ciclo che azzera l'array `a`:

```
for (i = 0; i < sizeof(a) / sizeof(a[0]); i++)
  a[i] = 0;
```

Si noti che il codice non deve essere modificato se si cambia la lunghezza dell'array

L'operatore `sizeof` con gli array

- Alcuni compilatori producono un messaggio di avvertimento per l'espressione `i < sizeof(a) / sizeof(a[0])`.
- Con alta probabilità la variabile `i` è un `int` (che è con segno), mentre `sizeof` restituisce un valore di tipo `size_t` (che è senza segno)
- Confrontare un intero con segno con un intero senza segno può essere pericoloso, ma in questo caso non crea problemi

L'operatore `sizeof` con gli array

- Per evitare il messaggio di avvertimento possiamo aggiungere un cast di `sizeof(a) / sizeof(a[0])` ad un intero con segno:

```
for (i = 0; i < (int) (sizeof(a) / sizeof(a[0])); i++)
  a[i] = 0;
```

- Può essere utile una macro:

```
#define SIZE ((int) (sizeof(a) / sizeof(a[0])))

for (i = 0; i < SIZE; i++)
  a[i] = 0;
```

Programma: calcolare gli interessi

- Il programma `interest.c` stampa una tabella che mostra gli interessi che derivano dalla somma di $100 investiti a vari tassi e per vari periodi
- L'utente inserirà un tasso di interesse ed un numero di anni
- La tabella mostrerà il valore della somma iniziale dopo un numero di anni che varia da 1 al numero di anni specificato dall'utente e per vari tassi di interesse (assumendo che la capitalizzazione avvenga ad ogni anno)

Programma: calcolare gli interessi

- Ecco un esempio di input e di output del programma:

```
Enter interest rate: 6
Enter number of years: 5

Years      6%      7%      8%      9%     10%
  1     106.00  107.00  108.00  109.00  110.00
  2     112.36  114.49  116.64  118.81  121.00
  3     119.10  122.50  125.97  129.50  133.10
  4     126.25  131.08  136.05  141.16  146.41
  5     133.82  140.26  146.93  153.86  161.05
```

Programma: calcolare gli interessi

- I numeri nella seconda riga dipendono dai numeri nella prima riga, quindi è ragionevole memorizzare la prima riga in un array
  - I valori dell'array verrano usati per calcolare i valori nella seconda riga
  - Questo processo può essere ripetuto per le righe successive
- Il programma usa cicli `for` annidati
  - Il ciclo esterno conta da 1 al numero di anni richiesti (quindi è sulle righe della tabella)
  - Mentre il ciclo interno incrementa il tasso di interesse dal valore minimo al valore massimo (quindi è sulle colonne della tabella)

interest.c

```
/* Prints a table of compound interest */

#include <stdio.h>

#define NUM_RATES ((int) (sizeof(value) / sizeof(value[0])))
#define INITIAL_BALANCE 100.00

int main(void)
{
  int i, low_rate, num_years, year;
  double value[5];

  printf("Enter interest rate: ");
  scanf("%d", &low_rate);
  printf("Enter number of years: ");
  scanf("%d", &num_years);
```

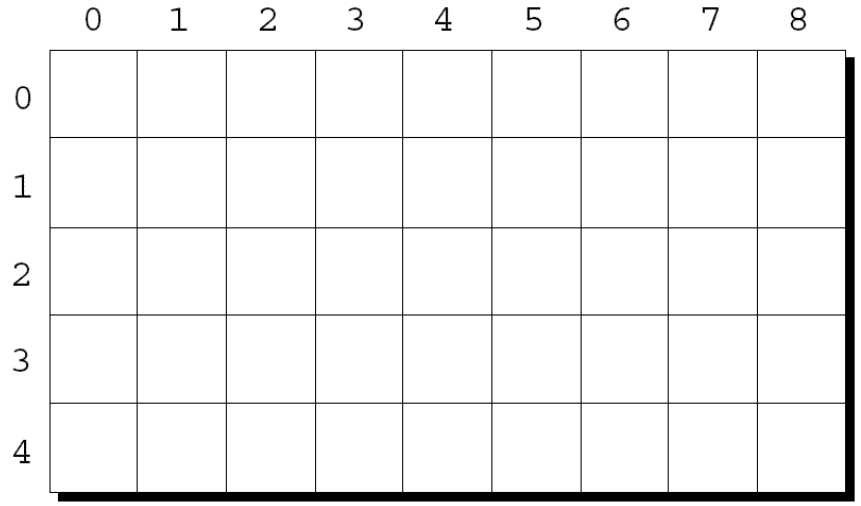
```
    printf("\nYears");
    for (i = 0; i < NUM_RATES; i++) {
      printf("%6d%%", low_rate + i);
      value[i] = INITIAL_BALANCE;
    }
    printf("\n");

    for (year = 1; year <= num_years; year++) {
      printf("%3d     ", year);
      for (i = 0; i < NUM_RATES; i++) {
        value[i] += (low_rate + i) / 100.0 * value[i];
        printf("%7.2f", value[i]);
      }
      printf("\n");
    }

    return 0;
}
```

Array a più dimensioni

- Un array può avere un numero qualsiasi di dimensioni
- La seguente dichiarazione specifica un array bidimensionale (cioè una *matrice*):

```
int m[5][9];
```

- `m` ha 5 righe e 9 colonne. Sia le righe che le colonne sono indicizzate a partire da 0:

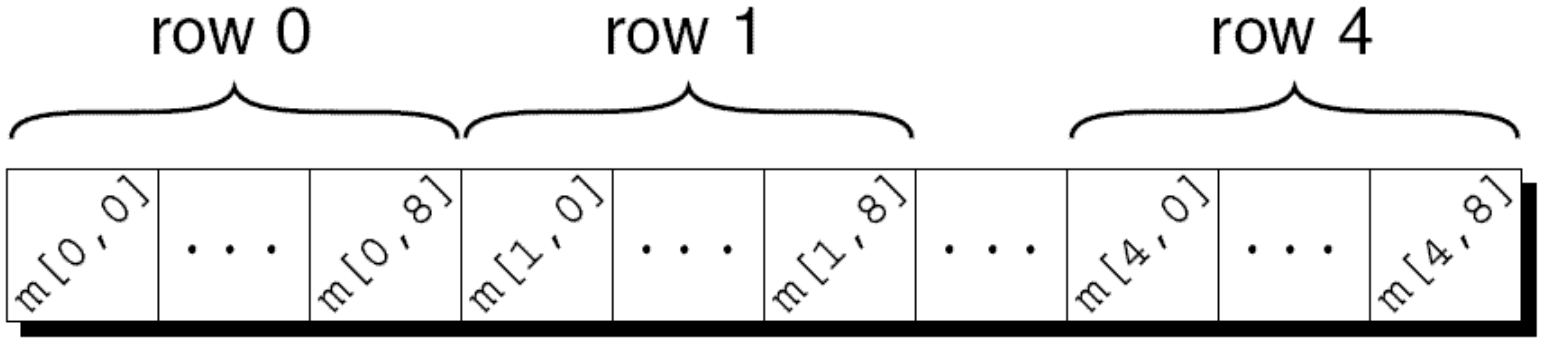


Array a più dimensioni

- Per accedere all'elemento di `m` nella riga `i`, colonna `j`, dobbiamo scrivere `m[i][j]`
- L'espressione `m[i]` indica la riga `i` di `m`, mentre `m[i][j]` indica l'elemento in posizione `j` nella riga `m[i]`
- Non scrivere `m[i,j]` al posto di `m[i][j]`
- Il C tratta la virgola come un operatore e quindi l'espressione `m[i,j]` è uguale a `m[j]`

Array a più dimensioni

- Sebbene visualizziamo una matrice come una tabella, questo non è il modo in cui i dati vengono memorizzati nella memoria del computer
- La memoria è una "array" monodimensionale di celle di memorie
- Il C memorizza una matrice in ordine di righe, cioè tutte le righe in sequenza partendo dalla riga 0
- Ecco come `m` viene memorizzata:



Array a più dimensioni

- Cicli `for` annidati sono utili per lavorare con gli array a più dimensioni
- Si consideri il problema di inizializzare una matrice per usarla come matrice identità
- Una coppia di cicli `for` annidati è perfetta:

```
#define N 10

double ident[N][N];
int row, col;

for (row = 0; row < N; row++)
  for (col = 0; col < N; col++)
    if (row == col)
      ident[row][col] = 1.0;
    else
      ident[row][col] = 0.0;
```

Inizializzazione di array multidimensionali

- Possiamo creare un inizializzatore nello stesso modo in cui inizializziamo gli array monodimensioanli:

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1, 0},
               {0, 1, 0, 1, 1, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 0, 1, 0},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

- Esistono vari altri modi più concisi per inizializzare un array multidimensionale

Inizializzazione di array multidimensionali

- Se un inizializzatore è più piccolo della dimensione dell'array, gli elementi rimanenti vengono inizializzati con il valore 0

- Il seguente inizializzatore specifica dei valori solo per le prime 3 righe di `m`; le ultime due righe conterranno 0:

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1, 0},
               {0, 1, 0, 1, 1, 0, 0, 1, 0}};
```

Inizializzazione di array multidimensionali

- Se una lista relativa ad una singola riga specifica meno elementi, i restanti varranno 0:

```
int m[5][9] = {{1, 1, 1, 1, 1, 0, 1, 1, 1},
               {0, 1, 0, 1, 0, 1, 0, 1},
               {0, 1, 0, 1, 1, 0, 0, 1},
               {1, 1, 0, 1, 0, 0, 0, 1},
               {1, 1, 0, 1, 0, 0, 1, 1, 1}};
```

Inizializzazione di array multidimensionali

- Possiamo anche omettere le parentesi che specificano le righe:

```
int m[5][9] = {1, 1, 1, 1, 1, 0, 1, 1, 1,
               0, 1, 0, 1, 0, 1, 0, 1, 0,
               0, 1, 0, 1, 1, 0, 0, 1, 0,
               1, 1, 0, 1, 0, 0, 0, 1, 0,
               1, 1, 0, 1, 0, 0, 1, 1, 1};
```

  Il compilatore "consumerà" tanti valori quanti ne servono per inizializzare ogni riga prima di procedere alla prossima riga

- Omettere le parentesi graffe per le righe è rischioso: un elemento in più o uno mancante modificherà il resto dell'array

Inizializzazione di array multidimensionali

- Nel C99 si possono usare gli inizializzatori designati anche per gli array multidimensionali
- Ecco un esempio:

```
double ident[2][2] = {[0][0] = 1.0, [1][1] = 1.0};
```

  Come prima, gli elementi per i quali non si specifica un valore varranno 0

Array costanti

- Un array può diventare "costante" se nella dichiarazione si specifica la parola `const`:

```
const char hex_chars[] =
  {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
   'A', 'B', 'C', 'D', 'E', 'F'};
```

- Un array costante non può essere modificato

Array costanti

- Vantaggi dell'uso di array `const`:
  - Rende chiaro che il programma non modificherà l'array
  - Aiuta il compilatore ad individuare errori

- L'uso di `const` non è limitato agli array, ma è particolarmente utile per gli array

Programma: dare una mano di carte

- Il programma `deal.c` illustra l'uso di array bidimensionali e di array costanti
- Il programma dà una mano di carte da poker
- Ogni carta ha un *seme* (fiori, quadri, cuori o picche) e *valore* (2,3,4,5,6,7,8,9,J,Q,K,A)

- Nota: i semi in Inglese sono: *club* (fiori), *diamond* (quadri), *heart* (cuori) e *spades* (picche)

Programma: dare una mano di carte

- L'utente specifica quante carte bisogna dare in una mano:

```
Enter number of cards in hand: 5
Your hand: 7c 2s 5d as 2h
```

- Problemi da risolvere:
  - Come selezioniamo le carte in modo **casuale** dal mazzo di carte?
  - Come facciamo ad evitare di selezionare **due volte** la stessa carta?

Programma: dare una mano di carte

- Per scegliere una carta utilizzeremo varie funzioni della libreria standard del C:
  - `time` (da `<time.h>`) – restituisce l'ora corrente, codificata in un numero (il numero di secondi dal 1/1/1970).
  - `srand` (da `<stdlib.h>`) – initializza il generatore di numeri casuali
  - `rand` (da `<stdlib.h>`) – produce un numero pseudocasuale ogni volta che viene chiamata
- Usando l'operatore `%`, possiamo "scalare" il numero casuale dal valore restituito dalla funzione `rand` in modo che sia compreso fra 0 e 3 (per il seme) oppure fra 0 e 12 (per il valore)

Programma: dare una mano di carte

- L'array `in_hand` viene usato per memorizzare le carte già scelte
- L'array ha 4 righe e 13 colonne: un elemento per ognuno delle 52 carte del mazzo
- Tutti gli elementi sono inizializzati a 0 (falso)
- Ogni volta che scegliamo una carta (in modo casuale) controlliamo se l'elemento corrispondente in `in_hand` è vero o falso
  - Se è vero vuol dire che la carta è stata già scelta, e quindi dovremo scegliere una nuova carta
  - Se è falso, allora la carta è valida. Memorizzeremo `true` per le successive scelte

Programma: dare una mano di carte

- Dopo avere verificato che la carta non era già stata scelta, dobbiamo trasformare i numeri casuali nel seme e nel valore corrispondenti
- Per fare questo useremo due array—uno per il seme ed uno per il valore—e poi selezioneremo gli elementi usando i numeri casuali scelti
- Questi array non cambiano durante l'esecuzione del programma pertanto possono essere dichiarati `const`

deal.c

```
/* Deals a random hand of cards */

#include <stdbool.h>    /* C99 only */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NUM_SUITS 4
#define NUM_RANKS 13

int main(void)
{
  bool in_hand[NUM_SUITS][NUM_RANKS]  = {false};
  int num_cards, rank, suit;
  const char rank_code[] = {'2','3','4','5','6','7','8',
                            '9','t','j','q','k','a'};
  const char suit_code[] = {'c','d','h','s'};
```

```
  srand((unsigned)  time(NULL));

  printf("Enter  number  of  cards  in  hand:  ");
  scanf("%d",  &num_cards);

  printf("Your  hand:");
  while  (num_cards  >  0)  {
    suit  =  rand()  %  NUM_SUITS;      /*  picks  a  random  suit  */
    rank  =  rand()  %  NUM_RANKS;      /*  picks  a  random  rank  */
    if  (!in_hand[suit][rank])  {
      in_hand[suit][rank]  =  true;
      num_cards--;
      printf("  %c%c",  rank_code[rank],  suit_code[suit]);
    }
  }
  printf("\n");

  return  0;
}
```

Array a lunghezza variabile (C99)

- Nel C89, la lunghezza dell'array deve essere una espressione costante
- Nel C99, è possibile utilizzare espressioni che *non* sono costanti
- Il programma `reverse2.c`—una versione modificata di `reverse.c`—illustra questa possibilità

reverse2.c

```
/* Reverses a series of numbers using a variable-length
   array - C99 only */

#include <stdio.h>

int main(void)
{
  int i, n;

  printf("How many numbers do you want to reverse? ");
  scanf("%d", &n);

  int a[n];   /* C99 only - length of array depends on n */

  printf("Enter %d numbers: ", n);
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
```

```
  printf("In reverse order:");
  for (i = n - 1; i >= 0; i--)
    printf(" %d", a[i]);
  printf("\n");

  return 0;
}
```

Array a lunghezza variabile (C99)

- L'array `a` nel programma `reverse2.c` è un esempio di ***array a lunghezza variabile*** (ALV, per brevità)
- La lunghezza di un ALV viene calcolata dal programma quando lo si esegue
  - Il vantaggio principale di un ALV è che il programma calcola esattamente quanti elementi servono.
- Se deve scegliere il programmatore la lunghezza dell'array rischia di essere più grande del necessario (spreco di memoria). Se non lo è si rischia che sia troppo piccola (e questo può essere disastroso)

Array a lunghezza variabile (C99)

- La lunghezza di un ALV può essere un'espressione:

```
int a[3*i+5];
int b[j+k];
```

- Anche gli array multidimensionali possono essere a lunghezza variabile:

```
int c[m][n];
```

- Restrizioni:
  - Non si possono usare variabili statiche (Ne discuteremo nel Capitolo 18)
  - Non si possono usare gli inizializzatori

… arrivederci alla prossima lezione