

Introduzione al linguaggio C

Laboratorio di
Linguaggi di Programmazione
a.a. 2001/2002

dott.ssa Francesca A. Lisi
lisi@di.uniba.it

Sommario (II parte)

- Struttura di un programma C
- Funzioni
- La libreria standard
- I/O base
- Tipi derivati
 - puntatori
 - vettori
 - strutture

La struttura di programma

Pascal

C

```
program nome-prog  
    dich-tipi  
    dich-variabili  
    def-procedure-e-funzioni  
begin  
    istruzioni  
end
```

```
main() {  
    istruzioni  
}  
funzione-1  
funzione-2  
...  
funzione-n
```

La struttura di programma (II)

Pascal

- struttura nidificata
- dichiarazioni raccolte in apposita sezione
 - visibilità all'interno della procedura/funzione
 - leggibilità, controllo

C

- struttura appiattita
- dichiarazioni sparse
 - visibilità all'interno del blocco
 - efficienza in spazio

Le funzioni: prototipo

return-value-type function-name(parameter-list)

- *function-name*: qualsiasi identificatore valido
- *return-value-type*: tipo del valore di ritorno
 - **int** (tipo di default)
 - **void** (nessun valore di ritorno)
- *parameter-list*: lista di parametri separati da virgola
 - **int** (tipo di default)

- Esempio

```
int maximum( int, int, int );
```

Le funzioni: definizione

*return-value-type function-name (parameter-list) {
 declarations-and-statements
}*

- *declarations-and-statements* : corpo di funzione (blocco)
 - Dichiarazione di variabili
 - Impossibilità di annidare definizioni di funzione
- Restituzione del controllo
 - Se non viene restituito alcun valore
 - **return;**
 - o, fino al raggiungimento di }
 - Se viene restituito un valore
 - **return** *expression*;



Outline



1. Function prototype
(3 parameters)

2. Input values

2.1 Call function

- Function definition

Program Output

```
1  /* Fig. 5.4: fig05_04.c
2     Finding the maximum of three integers */
3  #include <stdio.h>
4
5  int maximum( int, int, int );    /* function prototype */
6
7  int main()
8  {
9     int a, b, c;
10
11     printf( "Enter three integers: " );
12     scanf( "%d%d%d", &a, &b, &c );
13     printf( "Maximum is: %d\n", maximum( a, b, c ) );
14
15     return 0;
16 }
17
18 /* Function maximum definition */
19 int maximum( int x, int y, int z )
20 {
21     int max = x;
22
23     if ( y > max )
24         max = y;
25
26     if ( z > max )
27         max = z;
28
29     return max;
30 }
```

```
Enter three integers: 22 85 17
Maximum is: 85
```

I file di intestazione

- Contengono prototipi di funzione
- Funzioni della libreria standard
 - Individua il file di libreria che contiene le funzioni di tuo interesse (Es. `<stdio.h>` , `<math.h>` , etc)
 - Caricale nei tuoi programmi con

```
#include <filename>
```
- Funzioni riusabili definite dall'utente
 - Crea funzioni
 - Salvale in un file `filename.h`
 - Caricale nei tuoi programmi con

```
#include "filename.h"
```

La libreria standard

- `<stdio.h>` (I/O)
- `<ctype.h>` (controllo sulla classe dei caratteri)
- `<string.h>` (manipolazione di stringhe)
- `<math.h>` (funzioni matematiche)
- `<stdlib.h>` (funzioni di utilità)
- `<assert.h>` (funzioni di diagnostica)
- `<stdarg.h>` (funzioni di scansione liste variabili di argomenti)
- `<setjmp.h>` (funzioni di gestione salti non locali)
- `<time.h>` (funzioni di manipolazione data e ora)
- `<limits.h>` (definizioni limiti implementativi)

Esempio: Un gioco d'azzardo

- Simulatore di lancio dei dadi
- Regole
 - Tira due dadi
 - il giocatore vince se ottiene 7 o 11 al primo lancio,
 - il giocatore perde se ottiene 2, 3, o 12 al primo lancio
 - Uno dei valori 4, 5, 6, 8, 9, 10 diventa il punteggio del giocatore
 - Il giocatore deve lanciare un numero pari al proprio punteggio prima di tirare 7 per vincere



Outline



1. rollDice prototype

1.1 Initialize variables

1.2 Seed srand

2. Define switch statement for win/loss/continue

2.1 Loop

```
1  /* Fig. 5.10: fig05_10.c
2     Craps */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  int rollDice( void );
8
9  int main()
10 {
11     int gameStatus, sum, myPoint;
12
13     srand( time( NULL ) );
14     sum = rollDice();           /* first roll of the dice */
15
16     switch ( sum ) {
17         case 7: case 11:       /* win on first roll */
18             gameStatus = 1;
19             break;
20         case 2: case 3: case 12: /* lose on first roll */
21             gameStatus = 2;
22             break;
23         default:               /* remember point */
24             gameStatus = 0;
25             myPoint = sum;
26             printf( "Point is %d\n", myPoint );
27             break;
28     }
29
30     while ( gameStatus == 0 ) { /* keep rolling */
31         sum = rollDice();
32     }
```



Outline



2.2 Print win/loss

```
33     if ( sum == myPoint )           /* win by making point */
34         gameStatus = 1;
35     else
36         if ( sum == 7 )             /* lose by rolling 7 */
37             gameStatus = 2;
38 }
39
40 if ( gameStatus == 1 )
41     printf( "Player wins\n" );
42 else
43     printf( "Player loses\n" );
44
45 return 0;
46 }
47
48 int rollDice( void )
49 {
50     int die1, die2, workSum;
51
52     die1 = 1 + ( rand() % 6 );
53     die2 = 1 + ( rand() % 6 );
54     workSum = die1 + die2;
55     printf( "Player rolled %d + %d = %d\n", die1, die2, workSum );
56     return workSum;
57 }
```

```
Player rolled 6 + 5 = 11
Player wins
```

Program Output

```
Player rolled 6 + 6 = 12
Player loses
```



Outline



```
Player rolled 4 + 6 = 10
Point is 10
Player rolled 2 + 4 = 6
Player rolled 6 + 5 = 11
Player rolled 3 + 3 = 6
Player rolled 6 + 4 = 10
Player wins
```

Program Output

```
Player rolled 1 + 3 = 4
Point is 4
Player rolled 1 + 4 = 5
Player rolled 5 + 4 = 9
Player rolled 4 + 6 = 10
Player rolled 6 + 3 = 9
Player rolled 1 + 2 = 3
Player rolled 5 + 2 = 7
Player loses
```



Outline



1. Function prototype

1.1 Initialize variables

2. Input an integer

2.1 Call function fibonacci

2.2 Output results.

3. Define fibonacci recursively

Program Output

```
1  /* Fig. 5.15: fig05_15.c
2     Recursive fibonacci function */
3  #include <stdio.h>
4
5  long fibonacci( long );
6
7  int main()
8  {
9     long result, number;
10
11    printf( "Enter an integer: " );
12    scanf( "%ld", &number );
13    result = fibonacci( number );
14    printf( "Fibonacci( %ld ) = %ld\n", number, result );
15    return 0;
16 }
17
18 /* Recursive definition of function fibonacci */
19 long fibonacci( long n )
20 {
21    if ( n == 0 || n == 1 )
22        return n;
23    else
24        return fibonacci( n - 1 ) + fibonacci( n - 2 );
25 }
```

```
Enter an integer: 0
Fibonacci(0) = 0
```

```
Enter an integer: 1
Fibonacci(1) = 1
```



Outline



Program Output

```
Enter an integer: 2  
Fibonacci(2) = 1
```

```
Enter an integer: 3  
Fibonacci(3) = 2
```

```
Enter an integer: 4  
Fibonacci(4) = 3
```

```
Enter an integer: 5  
Fibonacci(5) = 5
```

```
Enter an integer: 6  
Fibonacci(6) = 8
```

```
Enter an integer: 10  
Fibonacci(10) = 55
```

```
Enter an integer: 20  
Fibonacci(20) = 6765
```

```
Enter an integer: 30  
Fibonacci(30) = 832040
```

```
Enter an integer: 35  
Fibonacci(35) = 9227465
```

Streams di I/O

- Sequenze di caratteri organizzati in righe
 - Terminazione con ritorno a capo
 - ANSI C deve supportare righe di almeno 254 caratteri
- Forniscono tutto ciò che occorre per I/O
- Possono essere ridirezionati
 - Standard input - tastiera
 - Standard output - schermo
 - Standard error - schermo

Formattare con `printf`

`printf` (*format-control-string*, *other-arguments*) ;

- *format-control-string*: stringa che descrive il formato di output
 - ogni specifica comincia con un segno % e termina con lo specificatore di conversione (flags, field widths, precisions, etc.)
 - Può effettuare arrotondamenti, allineamento di colonne, giustificazione sinistra/destra, inserzione di caratteri literal, formato esponenziale, formato esadecimale, e ampiezza e precisione fisse
- *other-arguments*: un argomento per ciascuna specifica di conversione nella stringa *format-control-string*

Formattare con scanf

scanf (*format-control-string*, *other-arguments*);

- *format-control-string*: stringa che descrive il formato di input
 - Acquisisce tutti i tipi di dati
 - Acquisisce specifici caratteri
 - Salta specifici caratteri
- *other-arguments*: puntatori a variabili dove i valori di input verranno memorizzati
- può includere ampiezze di campo per leggere un numero specifico di caratteri dal flusso di input



Outline



1. Initialize variables

2. Input

3. Print

Program Output

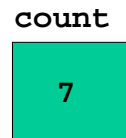
```
1  /* Fig 9.24: fig09_24.c */
2  /* Reading and discarding characters from the input stream */
3  #include <stdio.h>
4
5  int main()
6  {
7      int month1, day1, year1, month2, day2, year2;
8
9      printf( "Enter a date in the form mm-dd-yyyy: " );
10     scanf( "%d%c%d%c%d", &month1, &day1, &year1 );
11     printf( "month = %d  day = %d  year = %d\n\n",
12           month1, day1, year1 );
13     printf( "Enter a date in the form mm/dd/yyyy: " );
14     scanf( "%d%c%d%c%d", &month2, &day2, &year2 );
15     printf( "month = %d  day = %d  year = %d\n",
16           month2, day2, year2 );
17
18     return 0;
19 }
```

```
Enter a date in the form mm-dd-yyyy: 11-18-2000
month = 11  day = 18  year = 2000
```

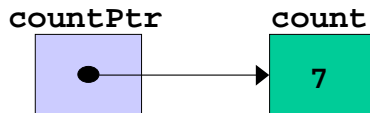
```
Enter a date in the form mm/dd/yyyy: 11/18/2000
month = 11  day = 18  year = 2000
```

I puntatori

- Le variabili “normali” contengono un valore (riferimento diretto)



- Le variabili “puntatore” contengono l’indirizzo di un’altra variabile (riferimento indiretto)



*tipo *var*

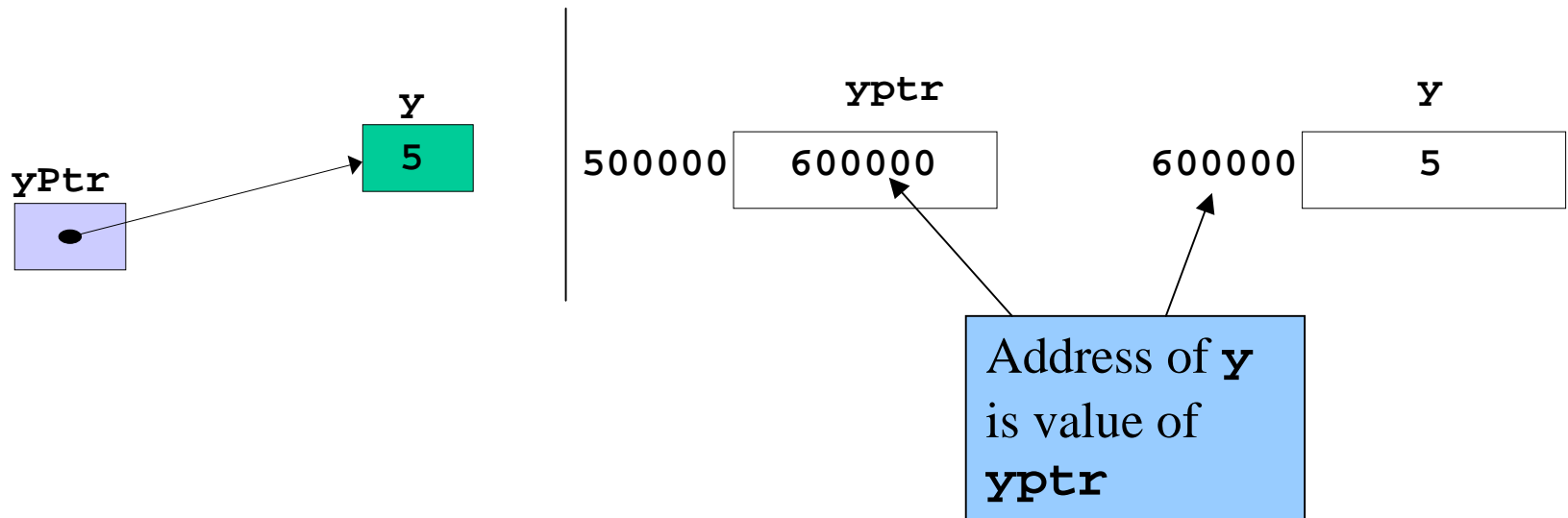
I puntatori (II)

- `&` (operatore di indirizzamento)
 - Restituisce l'indirizzo di un operando

```
int y = 5;
```

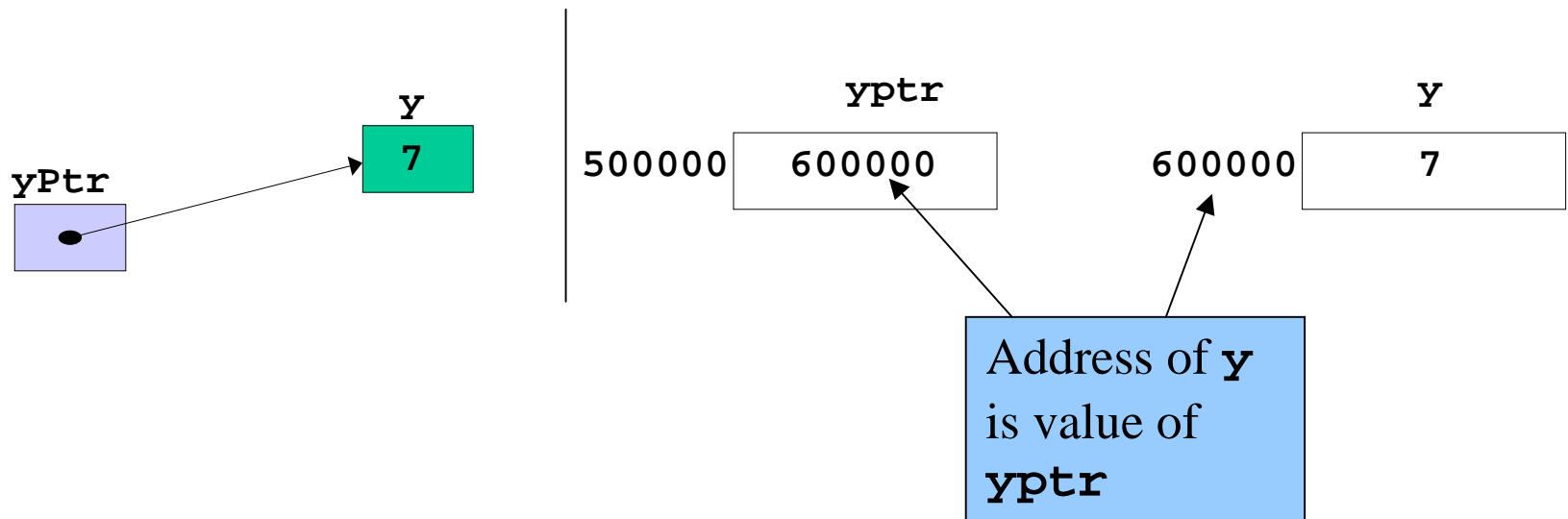
```
int *yPtr; //yPtr points to y
```

```
yPtr = &y; //yPtr gets address of y
```



I puntatori (III)

- * (operatore di dereferenziazione)
 - Restituisce un alias della variabile “puntata”
***yPtr** restituisce **y** (perché **yPtr** punta a **y**)
 - Dereferenziazione può essere usato per l’assegnamento di nuovi valori alla variabile “puntata”
***yPtr = 7; // changes y to 7**





Outline



1. Function prototype - takes a pointer to an int.
 - 1.1 Initialize variables
2. Call function
3. Define function

Notice how the address of **number** is given - **cubeByReference** expects a pointer (an address of a variable).

Inside **cubeByReference**, ***nPtr** is used (***nPtr** is **number**).

Program Output

```
1  /* Fig. 7.7: fig07_07.c
2     Cube a variable using call-by-reference
3     with a pointer argument */
4
5  #include <stdio.h>
6
7  void cubeByReference( int * ); /*
8
9  int main()
10 {
11     int number = 5;
12
13     printf( "The original value of number is %d", number );
14     cubeByReference( &number );
15     printf( "\nThe new value of number is %d\n", number );
16
17     return 0;
18 }
19
20 void cubeByReference( int *nPtr )
21 {
22     *nPtr = *nPtr * *nPtr * *nPtr; /* cube number in main */
23 }
```

```
The original value of number is 5
The new value of number is 125
```

I vettori

Pascal

var:array [tipo-indice-1, .., tipo-indice-n] of tipo-elemento

C

tipo-elemento var[dim-1], .., [dim-1]

Name of array (Note that all elements of this array have the same name, **c**)

| | |
|-------|------|
| c[0] | -45 |
| c[1] | 6 |
| c[2] | 0 |
| c[3] | 72 |
| c[4] | 1543 |
| c[5] | -89 |
| c[6] | 0 |
| c[7] | 62 |
| c[8] | -3 |
| c[9] | 1 |
| c[10] | 6453 |
| c[11] | 78 |

Position number of the element within array **c**



Outline



1. Initialize array

2. Loop

3. Print

```
1  /* Fig. 6.8: fig06_08.c
2     Histogram printing program */
3  #include <stdio.h>
4  #define SIZE 10
5
6  int main()
7  {
8     int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
9     int i, j;
10
11    printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );
12
13    for ( i = 0; i <= SIZE - 1; i++ ) {
14        printf( "%7d%13d", i, n[ i ] ) ;
15
16        for ( j = 1; j <= n[ i ]; j++ ) /* print one bar */
17            printf( "%c", '*' );
18
19        printf( "\n" );
20    }
21
22    return 0;
23 }
```



Outline



Program Output

| Element | Value | Histogram |
|---------|-------|-----------|
| 0 | 19 | ***** |
| 1 | 3 | *** |
| 2 | 15 | ***** |
| 3 | 7 | ***** |
| 4 | 11 | ***** |
| 5 | 9 | ***** |
| 6 | 13 | ***** |
| 7 | 5 | ***** |
| 8 | 17 | ***** |
| 9 | 1 | * |



Outline



1. Initialize strings

2. Print strings

2.1 Define loop

2.2 Print characters individually

2.3 Input string

3. Print string

```
1  /* Fig. 6.10: fig06_10.c
2     Treating character arrays as strings */
3  #include <stdio.h>
4
5  int main()
6  {
7     char string1[ 20 ], string2[] = "string literal";
8     int i;
9
10    printf(" Enter a string: ");
11    scanf( "%s", string1 );
12    printf( "string1 is: %s\nstring2: is %s\n"
13           "string1 with spaces between characters is:\n",
14           string1, string2 );
15
16    for ( i = 0; string1[ i ] != '\0'; i++ )
17        printf( "%c ", string1[ i ] );
18
19    printf( "\n" );
20    return 0;
21 }
```

```
Enter a string: Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

Program Output



1. Function definitions

2. Pass array to a function

2.1 Pass array element to a function

```
1  /* Fig. 6.13: fig06_13.c
2     Passing arrays and individual array elements to functions */
3  #include <stdio.h>
4  #define SIZE 5
5
6  void modifyArray( int [], int ); /* appears strange */
7  void modifyElement( int );
8
9  int main()
10 {
11     int a[ SIZE ] = { 0, 1, 2, 3, 4 }, i;
12
13     printf( "Effects of passing entire array call "
14            "by reference:\n\nThe values of the "
15            "original array are:\n" );
16
17     for ( i = 0; i <= SIZE - 1; i++ )
18         printf( "%3d", a[ i ] );
19
20     printf( "\n" );
21     modifyArray( a, SIZE ); /* passed call by reference */
22     printf( "The values of the modified array are:\n" );
23
24     for ( i = 0; i <= SIZE - 1; i++ )
25         printf( "%3d", a[ i ] );
26
27     printf( "\n\nEffects of passing array element call "
28            "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
29     modifyElement( a[ 3 ] );
30     printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
31     return 0;
32 }
```

Entire arrays passed call-by-reference, and can be modified

Array elements passed call-by-value, and cannot be modified



3.1 Function definitions

```
33
34 void modifyArray( int b[], int size )
35 {
36     int j;
37
38     for ( j = 0; j <= size - 1; j++ )
39         b[ j ] *= 2;
40 }
41
42 void modifyElement( int e )
43 {
44     printf( "Value in modifyElement is %d\n", e *= 2 );
45 }
```

Program Output

Effects of passing entire array call by reference:

The values of the original array are:

0 1 2 3 4

The values of the modified array are:

0 2 4 6 8

Effects of passing array element call by value:

The value of a[3] is 6

Value in modifyElement is 12

The value of a[3] is 6

Le strutture

Pascal

```
tipo-record=  
record  
    campo-1;  
    ..;  
    campo-n  
end  
tipo-record var;
```

C

```
struct nome-struttura  
{  
    campo-1;  
    ..;  
    campo-n;} var;
```



Outline



1. Load headers

1.1 Define struct

1.2 Function prototypes

1.3 Initialize deck [] and face []

1.4 Initialize suit []

```
1  /* Fig. 10.3: fig10_03.c
2     The card shuffling and dealing program using structures */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  struct card {
8     const char *face;
9     const char *suit;
10 };
11
12 typedef struct card Card;
13
14 void fillDeck( Card * const, const char *[],
15              const char *[] );
16 void shuffle( Card * const );
17 void deal( const Card * const );
18
19 int main()
20 {
21     Card deck[ 52 ];
22     const char *face[] = { "Ace", "Deuce", "Three",
23                          "Four", "Five",
24                          "Six", "Seven", "Eight",
25                          "Nine", "Ten",
26                          "Jack", "Queen", "King"};
27     const char *suit[] = { "Hearts", "Diamonds",
28                          "Clubs", "Spades"};
29
30     srand( time( NULL ) );
```

2. Randomize

2. fillDeck

2.2 deal

3. Function definitions

Put all 52 cards in the deck.
face and **suit** determined by
remainder (modulus).

Select random number between 0 and 51.
Swap element **i** with that element.

```
31
32 fillDeck( deck, face, suit );
33 shuffle( deck );
34 deal( deck );
35 return 0;
36 }
37
38 void fillDeck( Card * const wDeck, const char * wFace[],
39              const char * wSuit[] )
40 {
41     int i;
42
43     for ( i = 0; i <= 51; i++ ) {
44         wDeck[ i ].face = wFace[ i % 13 ];
45         wDeck[ i ].suit = wSuit[ i / 13 ];
46     }
47 }
48
49 void shuffle( Card * const wDeck )
50 {
51     int i, j;
52     Card temp;
53
54     for ( i = 0; i <= 51; i++ ) {
55         j = rand() % 52;
56         temp = wDeck[ i ];
57         wDeck[ i ] = wDeck[ j ];
58         wDeck[ j ] = temp;
59     }
60 }
```



Cycle through array and print out data.

```
61
62 void deal( const Card * const wDeck )
63 {
64     int i;
65
66     for ( i = 0; i <= 51; i++ )
67         printf( "%5s of %-8s%c", wDeck[ i ].face,
68                 wDeck[ i ].suit,
69                 ( i + 1 ) % 2 ? '\t' : '\n' );
70 }
```



Outline



Program Output

| | |
|-------------------|-------------------|
| Eight of Diamonds | Ace of Hearts |
| Eight of Clubs | Five of Spades |
| Seven of Hearts | Deuce of Diamonds |
| Ace of Clubs | Ten of Diamonds |
| Deuce of Spades | Six of Diamonds |
| Seven of Spades | Deuce of Clubs |
| Jack of Clubs | Ten of Spades |
| King of Hearts | Jack of Diamonds |
| Three of Hearts | Three of Diamonds |
| Three of Clubs | Nine of Clubs |
| Ten of Hearts | Deuce of Hearts |
| Ten of Clubs | Seven of Diamonds |
| Six of Clubs | Queen of Spades |
| Six of Hearts | Three of Spades |
| Nine of Diamonds | Ace of Diamonds |
| Jack of Spades | Five of Clubs |
| King of Diamonds | Seven of Clubs |
| Nine of Spades | Four of Hearts |
| Six of Spades | Eight of Spades |
| Queen of Diamonds | Five of Diamonds |
| Ace of Spades | Nine of Hearts |
| King of Clubs | Five of Hearts |
| King of Spades | Four of Diamonds |
| Queen of Hearts | Eight of Hearts |
| Four of Spades | Jack of Hearts |
| Four of Clubs | Queen of Clubs |