Un programma deve essere scritto come collezione di tante piccole funzioni perché:

- è più facile scrivere correttamente una funzione breve che abbia un unico compito: in questo modo sia la scrittura che la verifica di correttezza risultano più facili.

- è più facile mantenere o modificare un programma così costruito: è possibile cambiare facilmente solo l'insieme delle funzioni che devono essere riscritte, sapendo che il resto del codice funziona correttamente.

- l'impiego di funzioni brevi aumenta la leggibilità e fa sì che il programma si documenti da solo.

Un'utile regola pratica per la scrittura di buoni programmi consiste nel fare in modo che il codice di ciascuna funzione non superi una singola pagina.

Istruzione return

```
return ;
return ++a;
return (a * b) ;
```

- Quando viene raggiunta un'istruzione return, l'esecuzione della funzione si conclude e il controllo torna all'ambiente chiamante.
- Se l'istruzione return contiene un'espressione, allora il relativo valore viene restituito all'ambiente chiamante.
- L'espressione che deve essere restituita può essere scritta tra parentesi tonde, ma ciò non è obbligatorio.
- Se necessario, tale valore viene convertito al tipo della funzione come specificato nella definizione di funzione.

```
float f(int a, int b) {
        return a * b;        /* il valore viene convertito a float */
}
```

Una funzione può contenere zero o più istruzioni return;

in loro assenza il controllo torna all'ambiente chiamante quando viene raggiunta la parentesi graffa che chiude corpo della funzione.

```
double assoluto( double x ) {
        if ( x  >=  0.0 ) return x ;
        else return -x;
}
```

Il valore restituito da una funzione non deve necessariamente essere utilizzato.

```
main() {
        int a;

        scanf("%d", &a);
```

Prototipi di funzione

Le funzioni devono essere dichiarate prima di essere utilizzate.
In C è possibile dichiarare il ***prototipo*** di ogni funzione che informa il compilatore del numero e del tipo dei parametri che devono essere passati alla funzione e sul tipo del valore da essa restituito.

```
double    assoluto( double );
```

segnala al compilatore che assoluto ( ) è una funzione che riceve un unico parametro di tipo double e restituisce un valore di tipo double.
La forma generale di un prototipo di funzione è

```
type function_name (lista-del-tipo-dei-parametri) ;
```

la lista del tipo dei parametri è tipicamente un elenco di tipi separati da virgole.
Gli identificatori sono opzionali e non influiscono sul prototipo.
Per esempio, il prototipo di funzione

```
void f(char e, int i);
```

è equivalente a:

```
void f(char, int);
```

- Se la funzione è priva di argomenti si utilizza la parola chiave void;
- Se la funzione non restituisce alcun valore si utilizza la parola chiave void;
- I prototipi di funzione permettono al compilatore di verificare il codice in modo più approfondito;
- i valori passati alla funzione vengono, quando possibile, convertiti opportunamente

Chiamata di funzione

- Un programma è formato da una o più definizioni di funzione, una delle quali di nome main(), da cui ha sempre inizio l'esecuzione del programma.
- Quando il controllo del programma incontra un nome di funzione, tale funzione viene chiamata, o invocata, vale a dire che il controllo passa ad essa.
- Al termine della funzione il controllo del programma ritorna all'ambiente chiamante.
- Per chiamare una funzione si scrive il nome seguito da un elenco di zero o più parametri, racchiusi tra parentesi tonde.
- Di solito questi parametri corrispondono in numero e in tipo (o tipo compatibile).
- Tutti i parametri vengono passati per valore; al momento della chiamata tutti i parametri vengono valutati, e il valore ottenuto viene utilizzato dalla funzione.
- Se a una funzione viene passata una variabile, il valore memorizzato nella variabile nell'ambiente chiamante non viene modificato, ma viene inviato alla funzione per operare.

```
void modifica( int i ) {
        i +=3;
}

main() {
        int i = 3;
        printf(“%d”, i);
        modifica( i );
        printf(“%d”, i);
}
```

Il valore di i nella funzione main non viene cambiato dalla funzione modifica. Modifica acquisisce il valore di i e lo incrementa di 3.

```
int leggi( void ) {
        int x;
        do {
                printf("Inserisci un numero. [-1 per finire]");
                scanf("%d",&x);
        } while (x < -1);
        return x;
}

int fattoriale( int n ) {
        int prodotto = 1;

        for(; n>1; n--)
                prodotto *= n;
        return prodotto;
}

main() {
        int n;

        while( (n=leggi()) != -1 )
                printf("Il fattoriale di %6d è %10d\n", n, fattoriale(n));
}
```

Il valore della variabile n nel main() non viene modificato dalla funzione fattoriale.

Regole di visibilità

- Gli identificatori sono accessibili solo all'interno del blocco nel quale sono stati dichiarati.
- Ogni blocco ha un proprio elenco di identificatori. Un nome del blocco esterno è valido a meno che un blocco interno non lo ridefinisca.
- Se due blocchi sono paralleli, cioè si trovano uno dietro all'altro, gli identificatori definiti in un blocco non sono riconosciuti dall'altro blocco.
- Le funzioni possono essere viste come particolari blocchi che hanno un nome ed istruzioni di rientro.

```
{
        int a = 2;
        printf("%d", a);
        {
                int a = 5;
                printf("%d", a);
        }
        printf("%d", a);
}


{
        {
                int x = 4;
                ….
        }
        ….
        {
                int y = 5;
                …..
        }
}
```

La variabile x è disponibile nel blocco in cui è definita, non nel blocco che segue.

Vettore

Nella soluzione di problemi sono spesso necessarie delle variabili in grado di contenere più dati. Una di queste è il vettore.

Le caratteristiche del vettore sono:

- le componenti sono tutte dello stesso tipo;
- ogni elemento occupa una posizione del vettore; la prima è la posizione 0;
- il numero degli elementi del vettore deve essere dichiarato esplicitamente;

La dichiarazione del vettore avviene indicando il numero e il tipo delle singole componenti. Ad esempio la dichiarazione

```
int vet[10] ;
```

indica un vettore di 10 elementi interi consecutivi chiamati vet[0], vet[1], vet[2], ..., vet[9]. Ogni elemento è di tipo int. Il primo ha come indice 0;
l'ultimo 9.
Ogni vettore occupa spazio consecutivo in memoria; l'indirizzo di partenza è indicato da una costante identificata dal nome stesso del vettore. Nell'esempio precedente la costante vet indica l'indirizzo di partenza del vettore, che coincide con l'indirizzo del primo elemento. In sostanza abbiamo che

```
vet = = &vet[0]
```

```
/*
leggere un vettore di n interi e stamparlo
*/
```

Vettori e funzioni

Per utilizzare un vettore bisogna indicare alla funzione il tipo dei singoli elementi, l'indirizzo di partenza, e il numero di elementi consecutivi utilizzabili.

```
int massimo( int vet[], int n )
```

- int vet[] segnala alla funzione che il primo parametro in ingresso è l'indirizzo di un vettore di interi;
- int n indica il numero di celle del vettore effettivamente utilizzate;

La chiamata alla funzione massimo avviene indicando l'indirizzo di partenza del vettore e il numero di elementi da considerare. Se

```
int a[100];          /* vettore */
int n;               /* numero elementi */
int max;
```

la chiamata alla funzione massimo è

```
max=massimo( a, n );
```

dove a è l'indirizzo del vettore, n il numero di elementi. Si otterrebbe lo stesso risultato scrivendo

```
max=massimo( &a[0], n );
```

mentre è sbagliato scrivere

```
max=massimo( a[0], n ); /* errato */
```

In questo ultimo caso si invia alla funzione massimo il primo elemento, non l'indirizzo del primo elemento.

Ogni funzione che deve operare su di un vettore ha in ingresso l'indirizzo di partenza del vettore, il tipo degli elementi ed eventualmente il numero di elementi da considerare. In pratica la funzione opera sugli ***elementi originali*** del vettore non su copie.

```
/*
        funzione di lettura di un vettore
Soluzione 1:
*/
int leggi( int a[ ], int max ) {
        int i, n;

        do {
                printf("Numero elementi da inserire");
                scanf("%d",&n);
        } while( (n <= 0) || (n > max));
        for(i=0;i < n; ++i)
                scanf("%d",&a[i]);
        return n;
}
```

/* La funzione dopo aver inserito i dati in un vettore, restituisce il numero di dati effettivamente letti. Tale numero non può essere superiore alla dimensione massima.

Se le variabili sono

```
int a[100];
int n;
```

La chiamata sarà:

```
n=leggi( a, 100 );
```

*/

/*Soluzione 2 Poiché la funzione scanf restituisce il numero di dati effettivamente letti si può modificare la funzione in questo modo:

*/

```
int leggi( int a[ ], int max ) {
        int i=0;

        printf("Inserire gli elementi \n ");
        while((i < max) && (scanf("%d",&a[i]) ==1) )
                i++;
        return i;
}
```

/*

Per interrompere l'inserimento basta inserire il carattere di EOF, che in dos è ctrl^ z mentre in Unix è ctrl^ d.

Quando si raggiunge la dimensione massima del vettore (i == max), la prima condizione indicata nell'istruzione while risulterà falsa e non verrà effettuata la funzione scanf.

*/