Linguaggio C

Marta Capiluppi
marta.capiluppi@univr.it
Dipartimento di Informatica
Università di Verona

Struttura di un programma C

- Versione minima

```
Parte dichiarativa globale
main()
{
    Parte dichiarativa locale
    Parte esecutiva (istruzioni)
}
```

Struttura di un programma C

- Versione più generale:

```
Parte dichiarativa globale
main()
{
      Parte dichiarativa locale
      Parte esecutiva (istruzioni)
}
funzione1 ()
{
      Parte dichiarativa locale
      Parte esecutiva (istruzioni)
}
…
funzioneN ()
{
      Parte dichiarativa locale
      Parte esecutiva (istruzioni)
}
```

Struttura di un programma C

- Parte dichiarativa globale
  - Elenco dei dati usati in tutto il programma e delle loro caratteristiche (*tipo*)
    - numerici, non numerici
- Parte dichiarativa locale
  - Elenco dei dati usati dal **`main`** o dalle singole funzioni, con il relativo tipo

Il preprocessore C

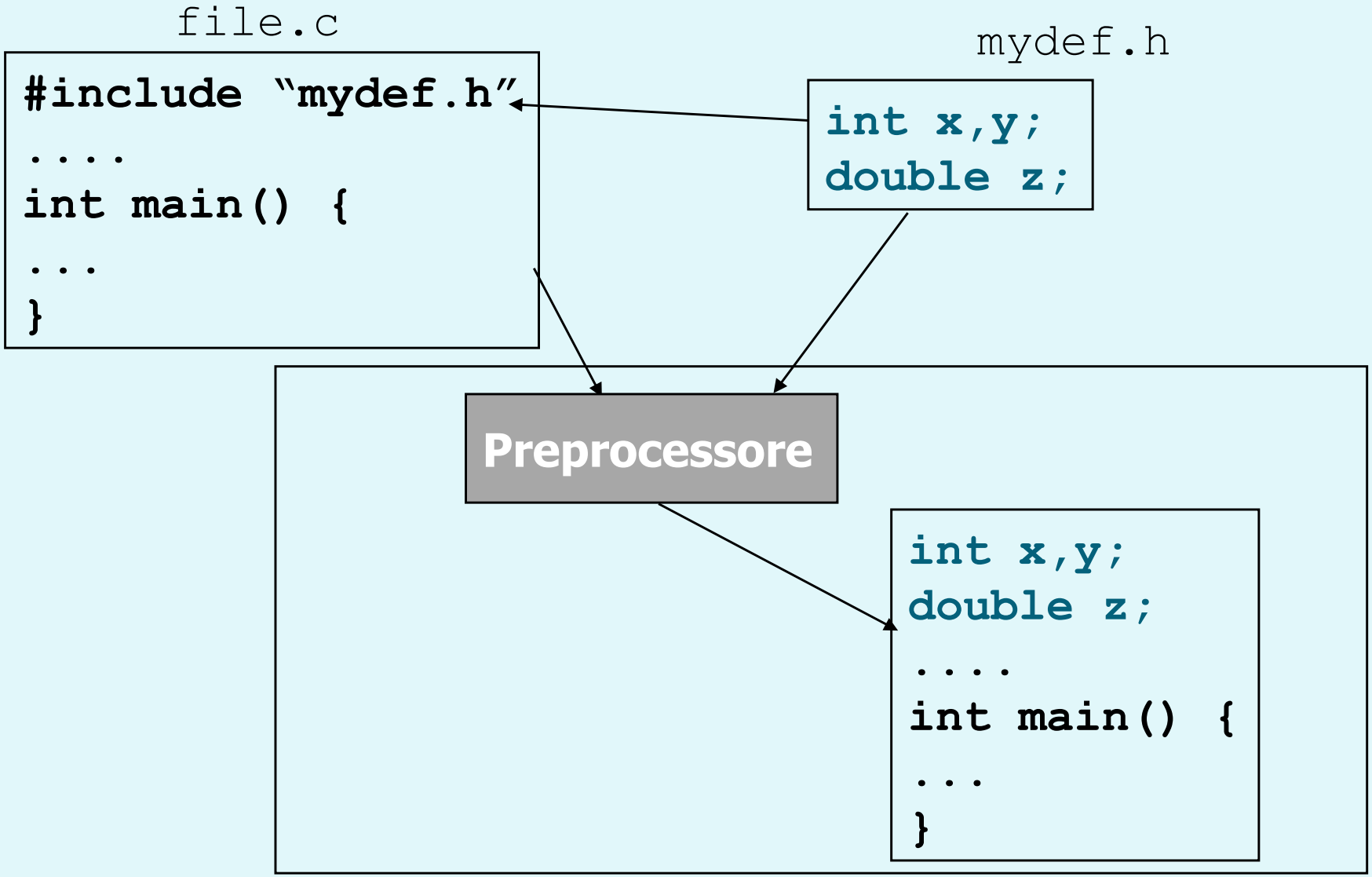
- La prima fase della compilazione (trasparente all'utente) consiste nell'invocazione del *preprocessore*
- Un programma C contiene specifiche direttive per il preprocessore
  - Inclusioni di file di definizioni (*header* file)
  - Definizioni di costanti
  - Altre direttive
- Individuate dal simbolo '**#**'

Direttive del preprocessore

- **`#include`**
  - Inclusione di un file di inclusione (tipicamente con estensione **`.h`**
  - Esempi:
    - **`#include <stdio.h>`** <- ***dalle directory di sistema***
    - **`#include "myheader.h"`** <- ***dalla directory corrente***
- **`#define`**
  - Definizione di un valore costante
  - Ogni riferimento alla costante viene espanso dal preprocessore al valore corrispondente
  - Esempi:
    - **`#define FALSE 0`**
    - **`#define SEPARATOR "--------------------------"`**

La direttiva #include

- Esempio:



Definizione di dati

- Tutti i dati devono essere definizione prima di essere usati
- Definizione di un dato:
  - riserva spazio in memoria
  - assegna un nome
- Richiede l'indicazione di:
  - tipo
  - modalità di accesso (variabili/costanti)
  - nome (identificatore)

Tipi base (primitivi)

- Sono quelli forniti direttamente dal C
- Identificati da parole chiave
  - `char` caratteri ASCII
  - `int` interi (complemento a 2)
  - `float` reali (floating point singola precisione)
  - `double` reali (floating point doppia precisione)
- La dimensione precisa di questi tipi dipende dall'architettura (non definita dal linguaggio)
  - |`char`| = 8 bit sempre

Modificatori dei tipi base

- Sono previsti dei modificatori, identificati da parole chiave da premettere ai tipi base
- **`signed`/`unsigned`**
  - Applicabili ai tipi **`char`** e **`int`**
    - **`signed`**: valore numerico con segno
    - **`unsigned`**: valore numerico senza segno
- **`short`/`long`**
  - Applicabili al tipo **`int`**
  - Utilizzabili anche senza specificare **`int`**

Definizione di variabili

- Sintassi:
  - *<tipo> <variabile>;*
- Sintassi alternativa (definizioni multiple)
  - *<tipo> <lista di variabili>;*

  - *<variabile>*: l'identificatore che rappresenta il nome della variabile
  - *<lista di variabili>*: lista di identificatori separati da ','

Definizione di dati

- Esempi:
  - `int x;`
  - `char ch;`
  - `long int x1,x2,x3;`
  - `double pi;`
  - `short int stipendio;`
  - `long y,z;`

Definizione di costanti

- Sintassi:

  [`const`] <***tipo***> <***variabile***> [= <***valore***>] `;`

- Esempi:
  - `const double pigreco =  3.14159;`
  - `const char  separatore = '$';`
  - `const float aliquota = 0.2;`
- Convenzione:
  - Identificatori delle constanti tipicamente in MAIUSCOLO
    - `const double PIGRECO = 3.14159`

Costanti speciali

- Caratteri ASCII non stampabili e/o “speciali”
- Ottenibili tramite “*sequenze di escape*”
  - \<codice ASCII ottale su tre cifre>
- Esempi:
  - `'\007'`
  - `'\013'`
- Caratteri “predefiniti”
  - **‘\b’ backspace**
  - **‘\f’ form feed**
  - **‘\n’ line feed**
  - **‘\t’ tab**

Stringhe

- Definizione:
  - sequenza di caratteri terminata dal carattere `NULL` (‘`\0`’)
- Non è un tipo di base del C
- Costanti stringa:

  “<***sequenza di caratteri***>“

  - Es:
    - **“Ciao!”**
    - **“abcdefg\n”**

Visibilità delle variabili

- Ogni variabile è definita all'interno di un preciso *ambiente di visibilità (scope)*
- *Variabili globali*
  - Definite all'esterno al **`main()`**
- Variabili *locali*
  - Definite all'interno del main
  - Più in generale, definite all'interno di un blocco

Visibilità delle variabili - Esempio

```
int n;
double x;
main() {
   int a,b,c;
   double y;
   {
      int d;
      double z;
   }
}
```

- `n,x`: visibili in tutto il file
- `a,b,c,y`: visibili in tutto il main
- `d,z`: visibili nel blocco

Le istruzioni

- Istruzioni di ingresso/uscita
- Istruzioni aritmetico–logiche
- Istruzioni di controllo

L'istruzione `printf()`

- Sintassi

  `printf``(<`***stringa formato***`>,<`***arg1***`>,...,<`***argn***`>);`

  - **<*stringa formato*>**: stringa che determina il formato di stampa di ognuno dei vari argomenti
    - Può contenere:
      - Caratteri (stampati come appaiono)
      - Direttive di formato nella forma `%`<carattere>
        - **`%d`** **`intero`**
        - **`%u`** **`unsigned`**
        - **`%s`** **`stringa`**
        - **`%c`** **`carattere`**
        - **`%x`** **`esadecimale`**
        - **`%o`** **`ottale`**
        - **`%f`** **`float`**
        - **`%g`** **`double`**

L'istruzione `printf()`

- <***arg1***>,...,<***argn***>: le quantità (espressioni) che si vogliono stampare
  - **Associati alle direttive di formato nello stesso ordine!**

- Esempi

```
int x=2;
float z=0.5;
char c='a';

printf("%d %f  %c\n",x,z,c);
```

output

```
2  0.5   a
```

```
printf("%f***%c***%d\n",z,c,x);
```

output

```
0.5***a***2
```

L'istruzione `scanf()`

- Sintassi

  `scanf``(<`***stringa formato***`>,<`***arg1***`>,...,<`***argn***`>);`

  - **<*stringa formato*>**: come per **`printf`**
  - **<*arg1*>,...,<*argn*>**: le variabili cui si vogliono assegnare valori
    - **IMPORTANTE: i nomi delle variabili vanno precedute dall'operatore & che indica l'indirizzo della variabile**

- Esempio:

```
int x;
float z;
scanf("%d %f", &x, &z);
```

I/O a caratteri

- Acquisizione/stampa di un carattere alla volta
- Istruzioni:
  - **`getchar()`**
    - Legge un carattere da tastiera
    - Il carattere viene fornito come “risultato” di **`getchar`**
    - (valore intero)
    - In caso di errore il risultato è la costante **`EOF`** (definita in **`stdio.h`**)
  - **`putchar`**(<*carattere*>)
    - Stampa <*carattere*> su schermo
    - <*carattere*>*:* una dato di tipo **`char`**

I/O a caratteri - Esempio

```
#include <stdio.h>
main()
{
  int tasto;
  printf("Premi un tasto...\n");
  tasto = getchar();
  if (tasto != EOF)    /* errore ? */
  {
      printf("Hai premuto %c\n", tasto);
      printf("Codice ASCII = %d\n", tasto);
  }
}
```

I/O a righe

- Acquisizione/stampa di una riga alla volta
  - Riga = serie di caratteri terminata da '\n'
- Istruzioni:
  - `gets` (<*variabile stringa*>)
    - **Legge una riga da tastiera (fino al '`\n`')**
    - **La riga viene fornita come stringa (<*stringa*>), senza il carattere '`\n`'**
    - **In caso di errore il risultato è la costante `NULL` (definita in `stdio.h`)**
  - `puts` (**<***stringa***>**)
    - **Stampa <*stringa*> su schermo**
    - **Aggiunge sempre '\n' alla stringa**

Le istruzioni

- Istruzioni di ingresso/uscita
- Istruzioni aritmetico–logiche
- Istruzioni di controllo

Operazioni su int

=	Assegnamento
+	Somma
-	Sottrazione
*	Moltiplicazione
/	Divisione con troncamento della parte frazionaria
%	Resto della divisione intera
==	Relazione di uguaglianza
!=	Relazione di diversità
<	Minore di…
>	Maggiore di…
<=	Minore o uguale
>=	Maggiore o uguale

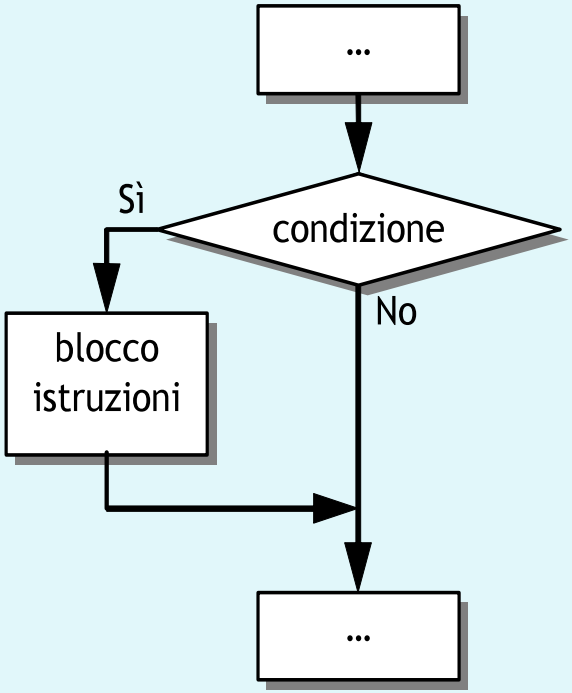
Operazioni su float

=	Assegnamento
+	Somma
-	Sottrazione
*	Moltiplicazione
/	Divisione a risultato reale
==	Relazione di uguaglianza
!=	Relazione di diversità
<	Minore di…
>	Maggiore di…
<=	Minore o uguale
>=	Maggiore o uguale

Le istruzioni

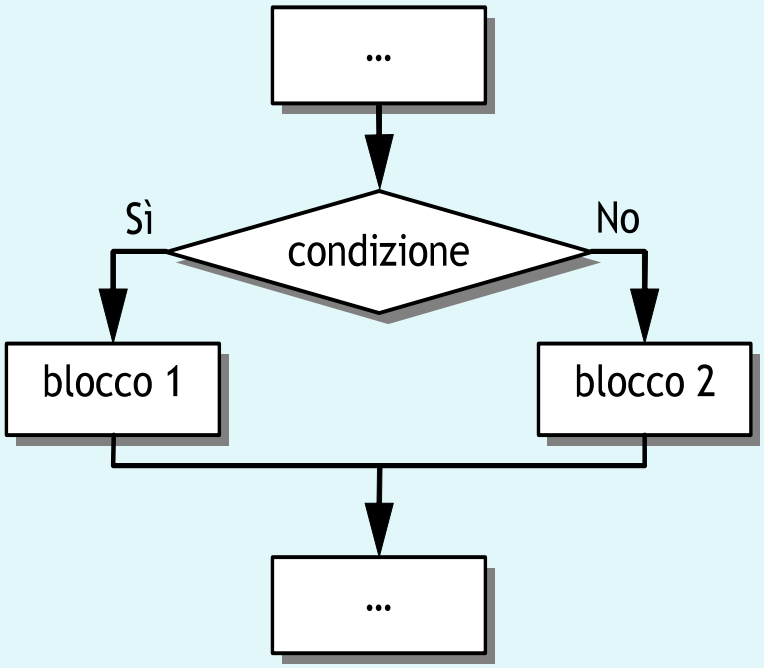
- Istruzioni di ingresso/uscita
- Istruzioni aritmetico–logiche
- Istruzioni di controllo

Selezione semplice



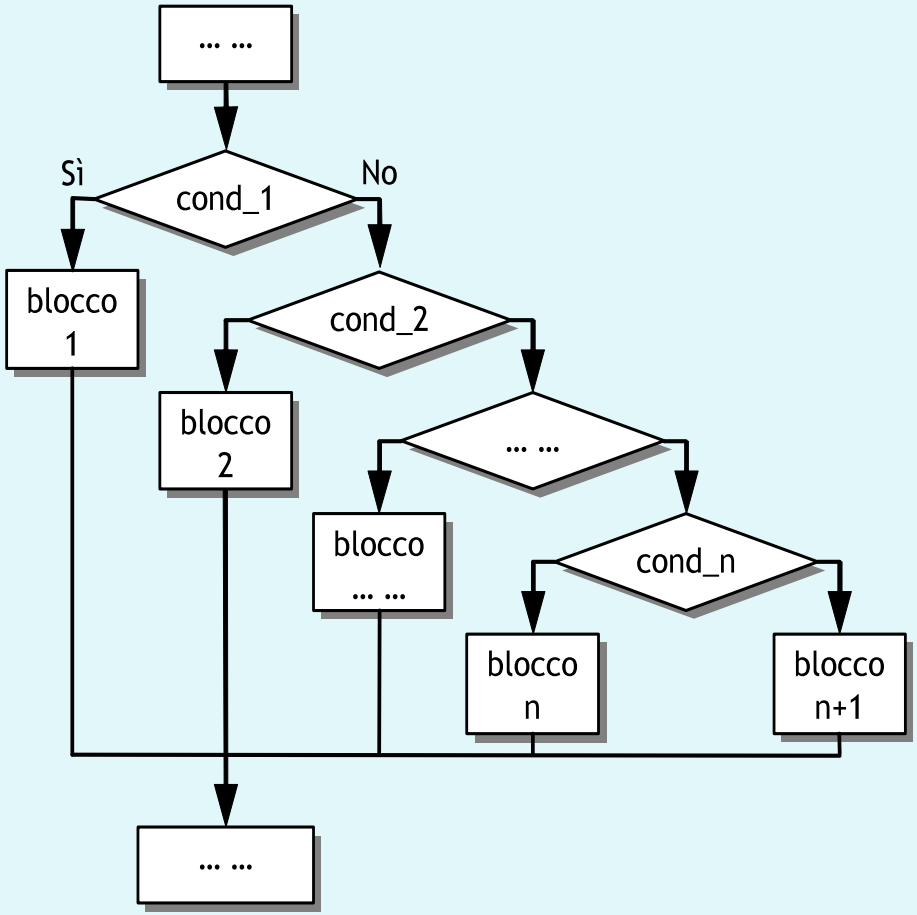

```
main()
{ …
  /* selezione semplice */
  if (condizione) {
    /* blocco istruzioni
       eseguito solo se
       condizione è true */
   …
  }
  …
}
```

Selezione a due vie



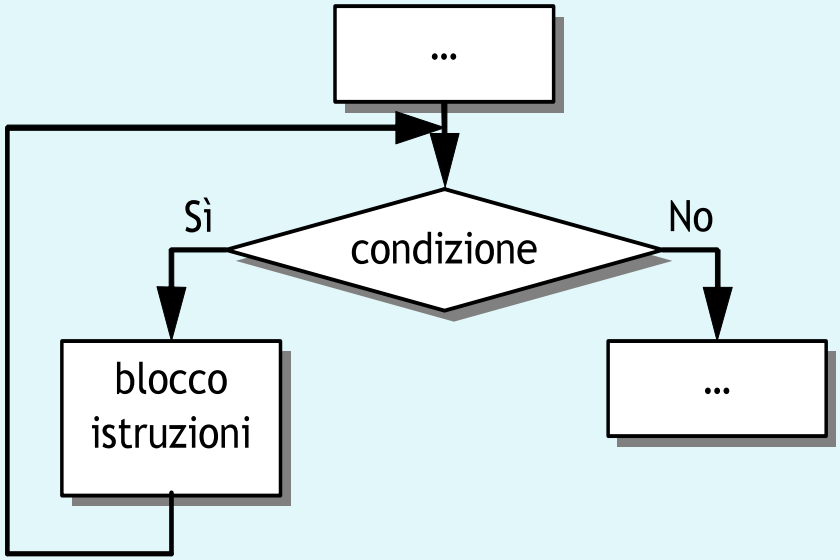

```
main()
{  …
    /* selezione a due vie */
   if (condizione) {
     … … /* blocco 1 */
   }
   else {
     … … /* blocco 2 */
   }
   …
}
```

Selezione a più vie



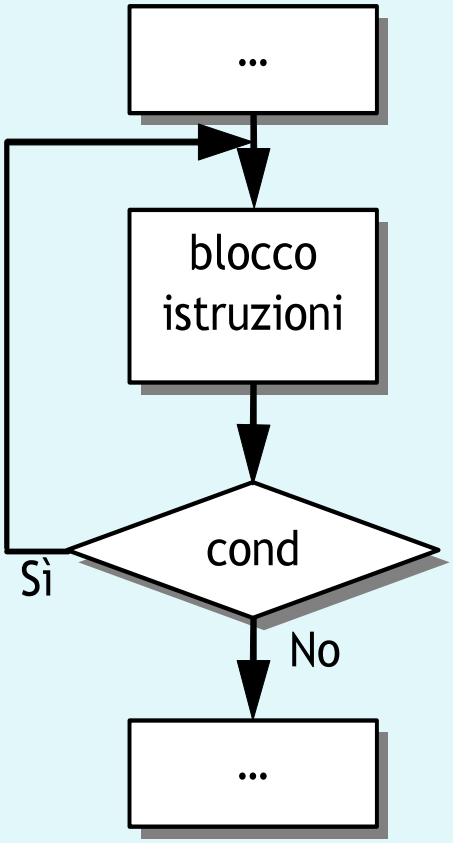

```
main()
{  … /* sel a più vie */
 if (cond_1) {
    …  /* blocco_1 */
 }
else if (cond_2) {
    … /* blocco_2 */
 }
else if (…  …) {
    … /* blocco_… */
 }
else if (cond_n) {
    … /* blocco_n */
 }
else {
    … /* blocco_n+1*/
 }
 …
}
```

Ciclo a condizione iniziale




```
main()
{ … …
/* ciclo a condizione iniziale */
  while (condizione) {
    … … /* blocco istruzioni */
    /* ripetuto finché condizione è true */
  }
  … /* eseguito quando condizione è
false */
}
```

Ciclo a condizione finale




```
main()
{ …
/* ciclo a condizione finale */
  do {
    … … /* blocco istruzioni */
    /* eseguito una volta e
       ripetuto se cond è true */
  } while (cond)
  … … /* eseguito se cond è
false */
}
```

Il ciclo for

- Può essere usato per sostituire il ciclo while quando si vuole eseguire il ciclo un numero finito di volte
- Utilizzato quando si devono utilizzare contatori

```
main()
{ … …
/* ciclo a condizione iniziale */
  for (espr1;cond2;espr3) {
    … … /* blocco istruzioni */
    /* ripetuto finché cond2 è true */
  }
}
```

Tipi strutturati: array

- In C non esiste il tipo array (vettore), ma è possibile definirlo utilizzando il costruttore di tipo array
- La sintassi di specifica di un array è

  <tipo> <variabile>[<dimensione>]
- Gli elementi sono ordinati e accessibili tramite un indice di posizione
- Gli elementi sono di tipo omogeneo
- Si può definire un tipo array usando l'istruzione **`typedef`**

  `typedef` <tipo> <variabile>[<dimensione>]

EX:

```
typedef int array[20];
array vett1, vett2;
```

Riempire un array

Algoritmo

Dati
  n = 100 intero
  f[ ] vettore di interi
  i intero positivo
Risoluzione
  ...
  i ← 1

  finché (i ≤ n) ripeti
    f[i] ← 0
    i ← i + 1
  fine ciclo
  ...

Programma in C

```
main() {
  int f[100];
  int i;
  …
  i = 0;

  while (i ≤ 99) {
    f[i] = 0;
    i = i + 1;
  }
…
}
```

```
main() {
  int f[100];
  int i;
  …

  for (i=0;i ≤ 99;i++) {
    f[i] = 0;
}
…
}
```

Tipi strutturati: matrici

- Una matrice è un array di array

Ex:

```
typedef int array[20];
array matrice[20];
```

oppure

```
typedef int matrice[20][20];
matrice mat1,mat2;
```

- La sintassi può essere

<tipo> <variabile>[<dimRighe>][<dimColonne>]

Tipi strutturati: record

- In C è possibile definire dati composti da elementi eterogenei (*record*), aggregandoli in una singola variabile individuata dalla keyword **`struct`**
- Sintassi

```
struct <identificatore> {
  campi
};
```

I **`campi`** sono nel formato

<*tipo*> <*nome campo*>;

struct - Esempio

```
struct complex {
   double re;
   double im;
}

struct identity {
   char nome[30];
   char cognome[30];
   char codicefiscale[15];
   int altezza;
   char statocivile;
}
```

struct

- **`struct`** è un costruttore di tipo
- Si può anche definire con **`typedef`**

Ex:

```
struct complex {
  double re;
  double im;
}
...
struct complex num1, num2;
```

Accesso ai campi

- Una struttura permette di accedere ai singoli campi tramite l'operatore '.', applicato a variabili del corrispondente tipo struct

  <*variabile***>.<***campo***>**

- Esempio:

```
struct complex {
     double re;
     double im;
}
...
struct complex num1, num2;
num1.re = 0.33; num1.im = -0.43943;
num2.re = -0.133; num2.im = -0.49;
```

Definizione di struct come tipi

- E' possibile definire un nuovo tipo a partire da una **`struct`** tramite **`typedef`**

Ex:

```
typedef struct complex {
  double re;
  double im;
} compl;
compl z1,z2;
```

Funzioni

- Un programma C consiste di una o più funzioni
  - Almeno **`main()`**
- Definizione delle funzioni
  - Dopo la definizione di **`main()`**
  - Prima della definizione di **`main()`** ➡ necessario premettere in testa al file il *prototipo* della funzione
    - Nome
    - Argomenti

Funzioni e prototipi: esempio

```
double f(int x)
{
  …
}
int main ()
{
  …
}
```

```
double f(int);
```

prototipo

```
int main ()
{
  …
}
double f(int x)
{
  …
}
```

Funzioni di libreria

- Il C prevede numerose funzioni predefinite per scopi diversi
- Particolarmente utili sono:
  - Funzioni matematiche
  - Funzioni di utilità
- Definite in specifiche *librerie*

Funzioni matematiche

- Utilizzabili con `#include <math.h>`

funzione	*definizione*
`double sin (double x)`	
`double cos (double x)`	
`double tan (double x)`	
`double asin (double x)`	
`double acos (double x)`	
`double atan (double x)`	
`double atan2 (double y, double x)`	atan (y / x)
`double sinh (double x)`	
`double cosh (double x)`	
`double tanh (double x)`	

Funzioni matematiche

`funzione`	*definizione*
`double pow (double x, double y)`	x^Y
`double sqrt (double x)`	radice quadrata
`double log (double x)`	logaritmo naturale
`double log10 (double x)`	logaritmo decimale
`double exp (double x)`	e^X
`double ceil (double x)`	ceiling(x)
`double floor (double x)`	floor(x)
`double fabs (double x)`	valore assoluto
`double fmod (double x, double y)`	modulo

Funzioni di utilità

- Varie categorie
  - Classificazione caratteri

```
#include <ctype.h>
```

  - Funzioni matematiche intere

```
#include <stdlib.h>
```

  - Stringhe

```
#include <string.h>
```

Funzioni di utilita'

- Classificazione caratteri

funzione	*definizione*
`int isalnum (char c)`	**Se c è lettera o cifra**
`int isalpha (char c)`	**Se c è lettera**
`int isascii(char c)`	**Se c è lettera o cifra**
`int isdigit (char c)`	**Se c è una cifra**
`int islower(char c)`	**Se c è minuscola**
`int isupper (char c)`	**Se c è maiuscola**
`int isspace(char c)`	**Se c è spazio,tab,\n**
`int iscntrl(char c)`	**Se c è di controllo**
`int isgraph(char c)`	**Se c è stampabile, non spazio**
`int isprint(char c)`	**Se c è di stampabile**
`int ispunct(char c)`	**Se c è di interpunzione**

Funzioni di utilita'

- Funzioni matematiche intere

funzione	*definizione*
`int abs (int n)`	**valore assoluto**
`long labs (long n)`	**valore assoluto**
`div_t div (int numer, int denom)`	quoto e resto della divisione intera
`ldiv_t ldiv (long numer, long denom)`	quoto e resto della divisione intera

Nota: div_t e ldiv_t sono di un tipo aggregato particolare fatto di due campi (int o long a seconda della funzione usata):

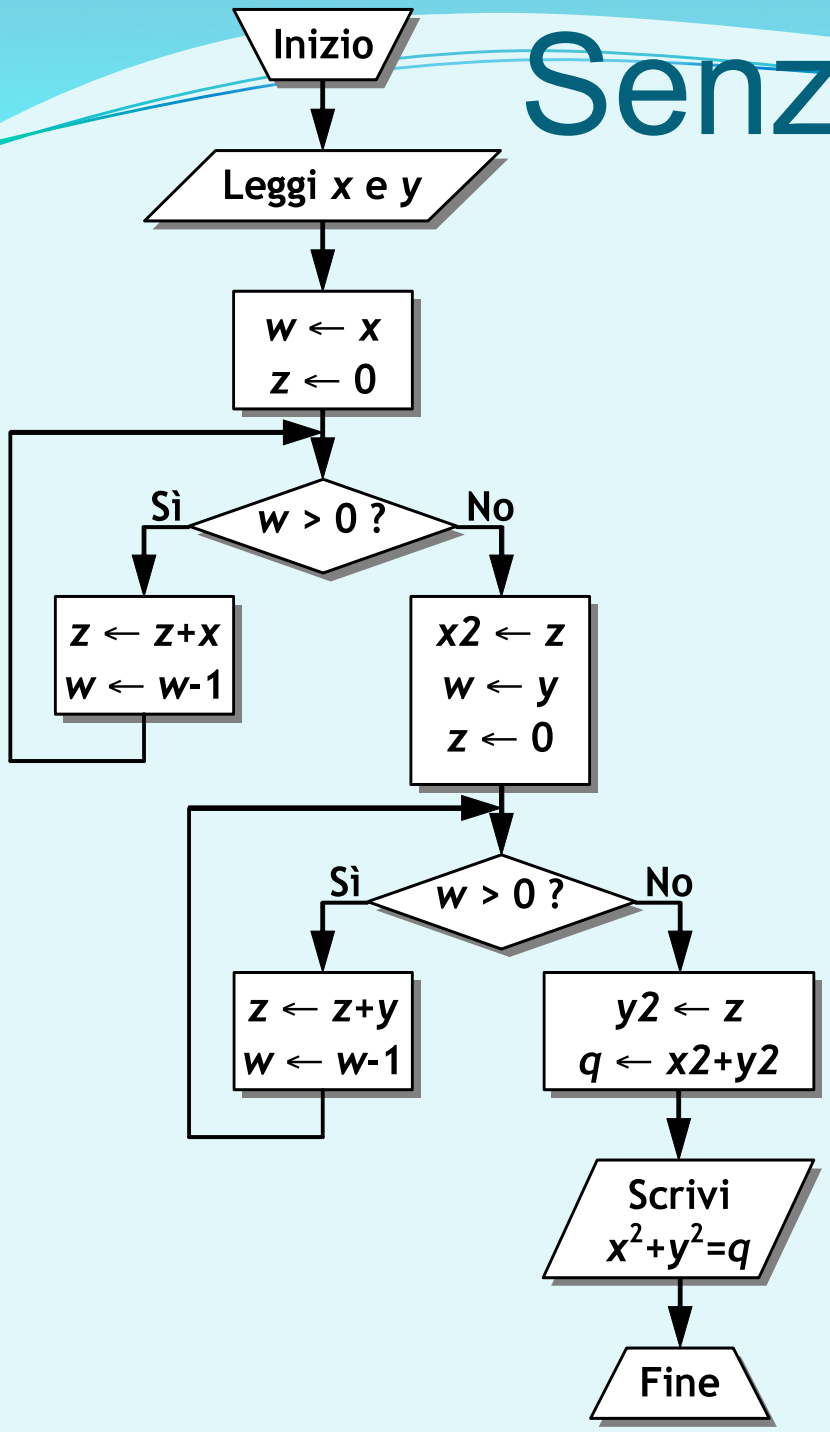
```
quot /* quoziente */
rem /* resto */
```

string.h

- Funzioni per Stringhe

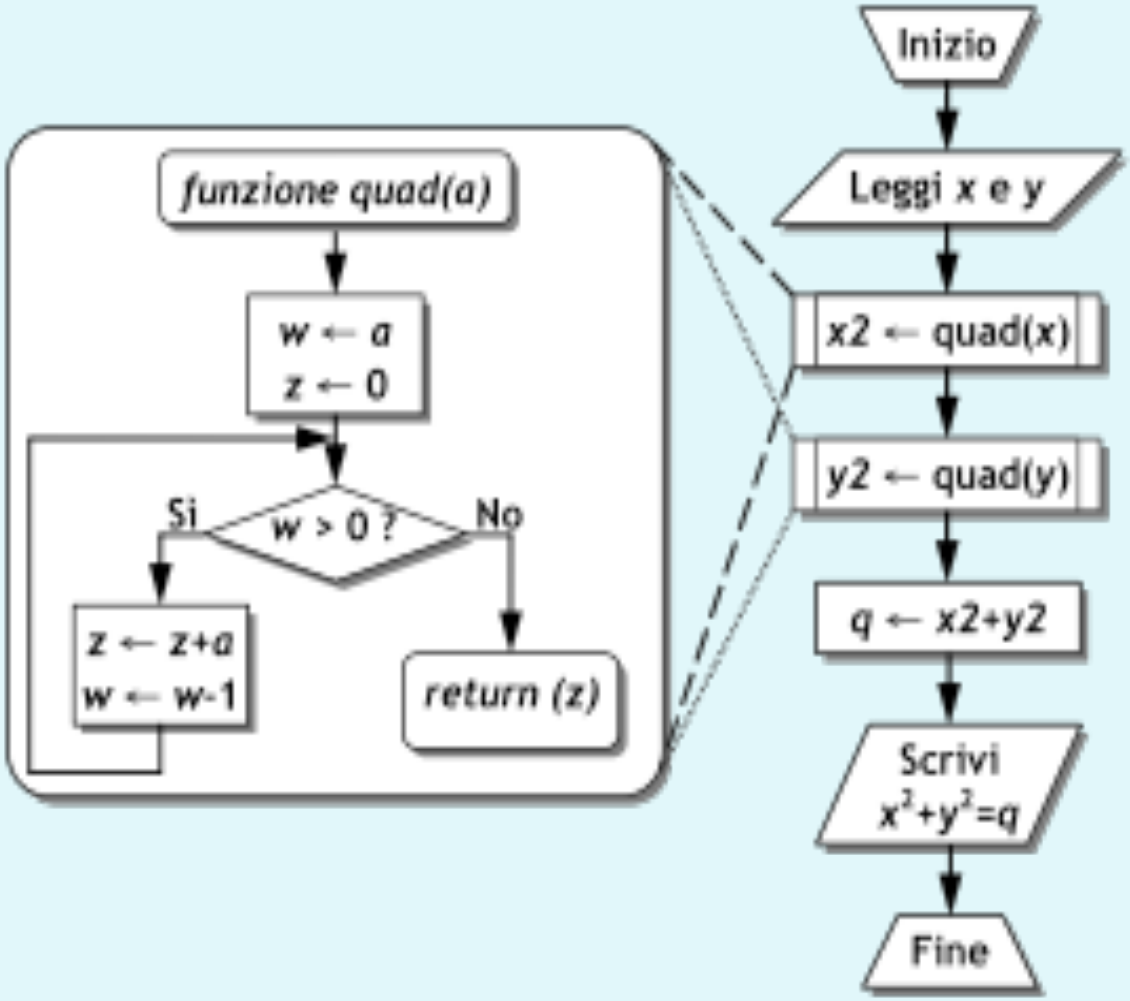
funzione	*definizione*
`char* strcat (char* s1, char* s2);`	*concatenazione s1+s2*
`char* strchr (char* s, int c);`	
`int strcmp (char* s1, char* s2);`	*confronto*
`char* strcpy (char* s1, char* s2);`	*s1 <= s2*
`int strlen (char* s);`	*lunghezza di s*
`char* strncat (char* s1,char* s2,int n);`	*concat. n car. max*
`char* strncpy (char* s1,char* s2,int n);`	*copia n car. max*
`char* strncmp(char* dest,char* src,int n);`	*confr. n car. max*

Senza sottoprogrammi




```
main()  /* q = x2 + y2 */
{ int x,y,x2,y2,q,w,z;
  scanf("%d %d",&x,&y);
  w = x;
  z = 0;
  while (w > 0) {
     z = z + x;
     w = w – 1; }
  x2 = z;
  w = y;
  z = 0;
  while (w > 0) {
     z = z + y;
     w = w – 1; }
  y2 = z;
  q = x2+y2;
  printf("%d", q);
}
```

Con sottoprogrammi




```
int quad (int a) {
   /* restituisce a2 /
   int w, z;
   w = a;  z = 0;
   while (w > 0) {
      z = z + a;
      w = w – 1;  }
   return (z);
}

main() /* q = x2 + y2 */
{ int x,y,x2,y2,q;
  scanf("%d %d",&x,&y);
  x2 = quad(x);
  y2 = quad(y);
  q = x2+y2;
  printf("%d", q);
}
```