

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**

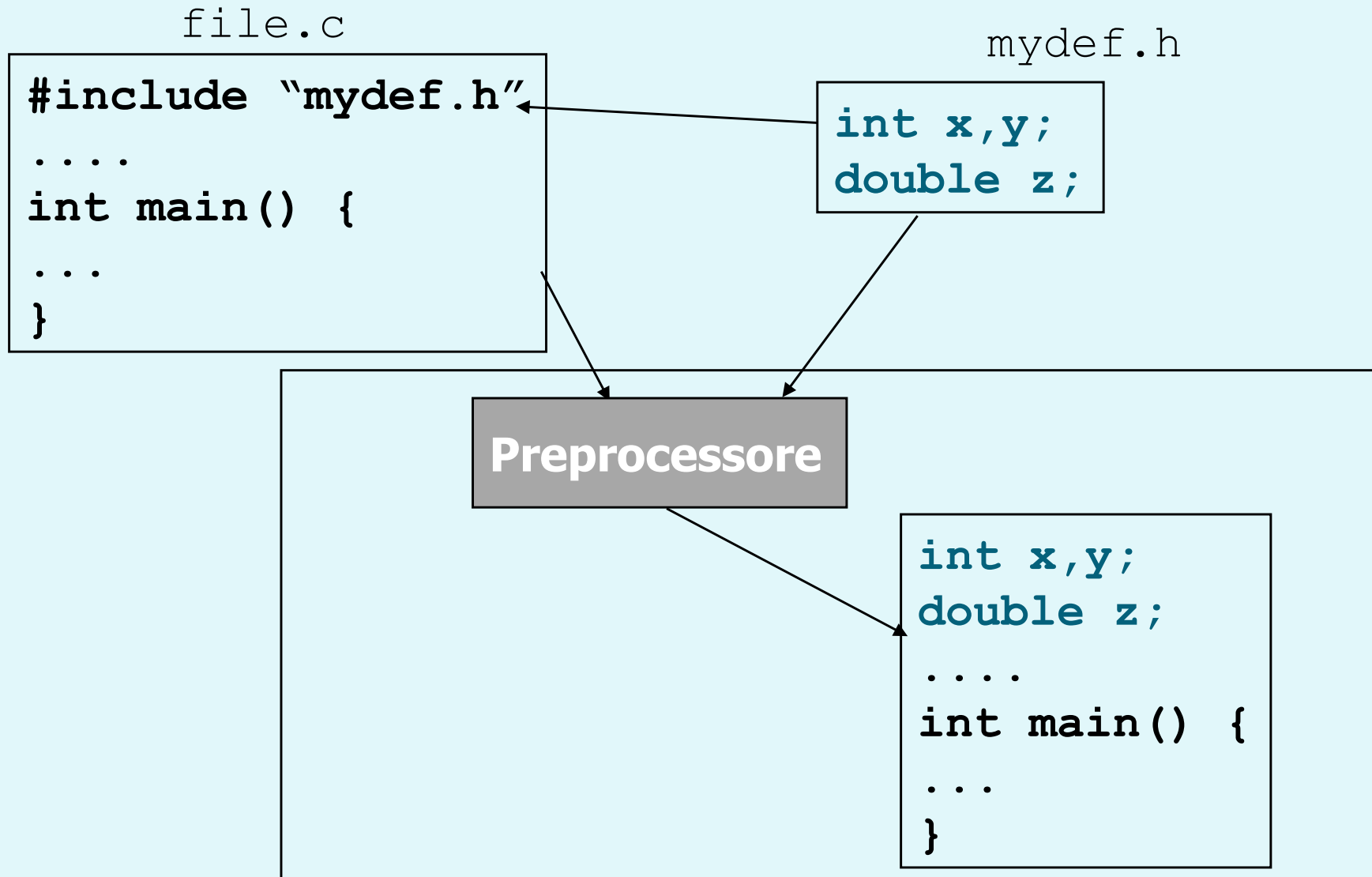
- Definizione di un valore costante
- Ogni riferimento alla costante viene espanso dal preprocessore al valore corrispondente

- Esempi:

- `#define FALSE 0`
- `#define SEPARATOR "-----"`

La direttiva #include

- Esempio:



Definizione di dati

- Tutti i dati devono essere definiti 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)
 - $|\text{char}| = 8$ 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 costanti 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>`

• <code>%d</code>	<code>intero</code>
• <code>%u</code>	<code>unsigned</code>
• <code>%s</code>	<code>stringa</code>
• <code>%c</code>	<code>carattere</code>
• <code>%x</code>	<code>esadecimale</code>
• <code>%o</code>	<code>ottale</code>
• <code>%f</code>	<code>float</code>
• <code>%g</code>	<code>double</code>

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

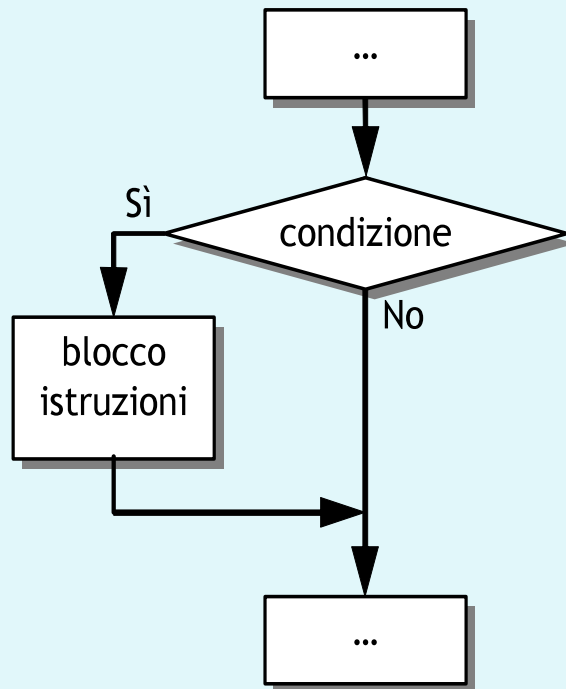
Operazioni su float

=	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

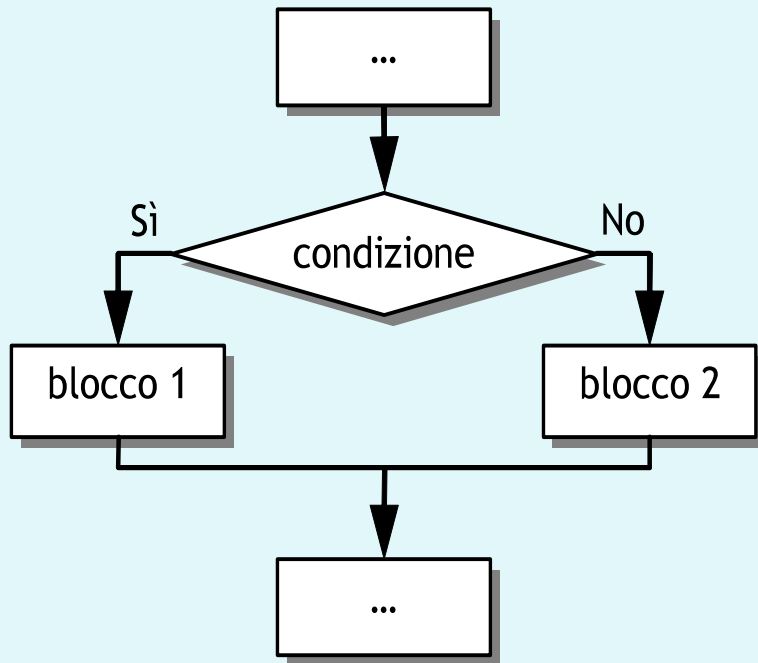
- Istruzioni di ingresso/uscita
- Istruzioni aritmetico-logiche
- Istruzioni di controllo

Selezione semplice



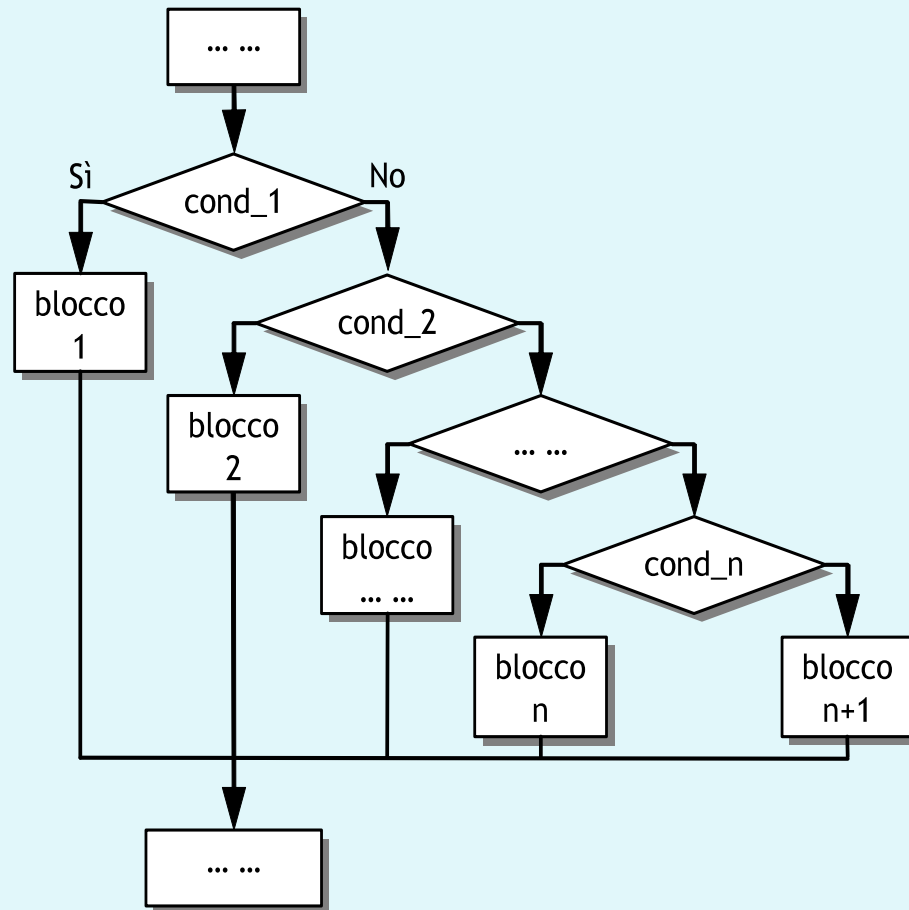
```
main()
{ ...
  /* selezione semplice */
  if (condizione) {
    /* blocco istruzioni
    eseguito solo se
    condizione è true */
    ...
  }
  ...
}
```

Selezione a due vie



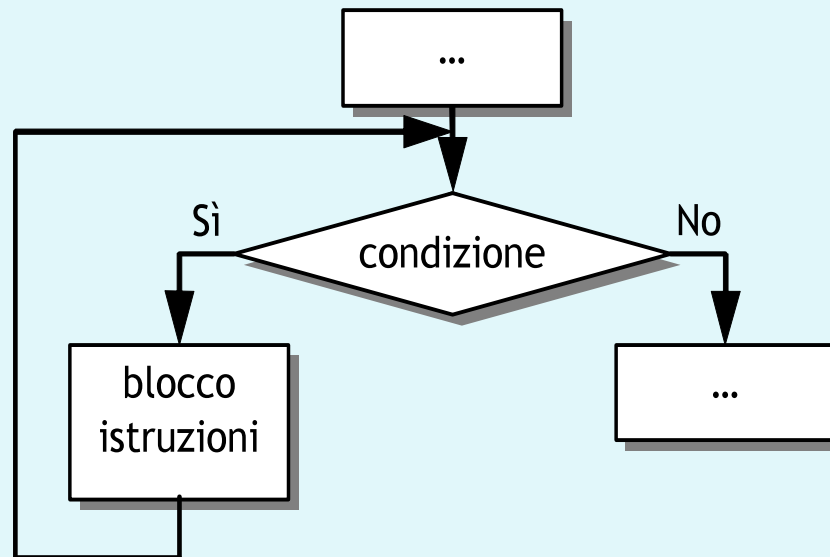
```
main()
{ ...
  /* selezione a due vie */
  if (condizione) {
    ... /* blocco 1 */
  }
  else {
    ... /* blocco 2 */
  }
  ...
}
```

Selezione a più vie



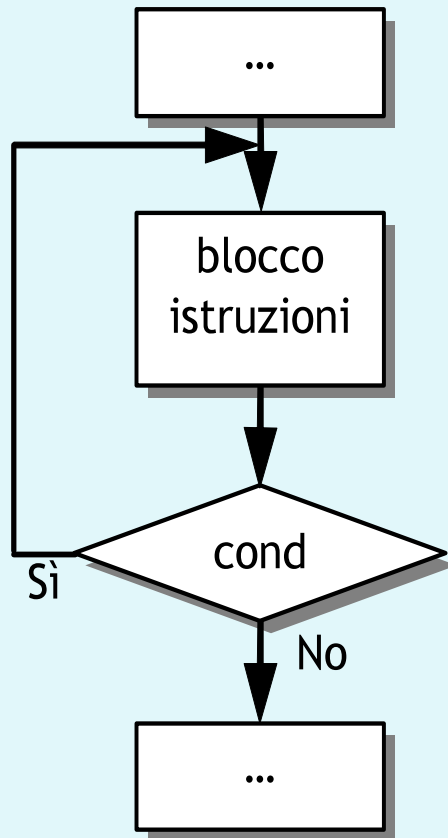
```
main()
{ ... /* sel a più vie */
  if (cond_1) {
    ... /* blocco_1 */
  }
  else if (cond_2) {
    ... /* blocco_2 */
  }
  else if (... ..) {
    ... /* blocco_... */
  }
  else if (cond_n) {
    ... /* blocco_n */
  }
  else {
    ... /* blocco_n+1*/
  }
  ...
}
```

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>`

<i>funzione</i>	<i>definizione</i>
<code>double sin (double x)</code>	
<code>double cos (double x)</code>	
<code>double tan (double x)</code>	
<code>double asin (double x)</code>	
<code>double acos (double x)</code>	
<code>double atan (double x)</code>	
<code>double atan2 (double y, double x)</code>	<code>atan (y / x)</code>
<code>double sinh (double x)</code>	
<code>double cosh (double x)</code>	
<code>double tanh (double x)</code>	

Funzioni matematiche

funzione	definizione
<code>double pow (double x, double y)</code>	x^y
<code>double sqrt (double x)</code>	radice quadrata
<code>double log (double x)</code>	logaritmo naturale
<code>double log10 (double x)</code>	logaritmo decimale
<code>double exp (double x)</code>	e^x
<code>double ceil (double x)</code>	ceiling(x)
<code>double floor (double x)</code>	floor(x)
<code>double fabs (double x)</code>	valore assoluto
<code>double fmod (double x, double y)</code>	modulo

Funzioni di utilità

- Varie categorie

- Classificazione caratteri

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

- Funzioni matematiche intere

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

- Stringhe

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

# Funzioni di utilità

- Classificazione caratteri

| <i>funzione</i>                   | <i>definizione</i>               |
|-----------------------------------|----------------------------------|
| <code>int isalnum (char c)</code> | Se c è lettera o cifra           |
| <code>int isalpha (char c)</code> | Se c è lettera                   |
| <code>int isascii(char c)</code>  | Se c è lettera o cifra           |
| <code>int isdigit (char c)</code> | Se c è una cifra                 |
| <code>int islower(char c)</code>  | Se c è minuscola                 |
| <code>int isupper (char c)</code> | Se c è maiuscola                 |
| <code>int isspace(char c)</code>  | Se c è spazio,tab,\n             |
| <code>int iscntrl(char c)</code>  | Se c è di controllo              |
| <code>int isgraph(char c)</code>  | Se c è stampabile,<br>non spazio |
| <code>int isprint(char c)</code>  | Se c è di stampabile             |
| <code>int ispunct(char c)</code>  | Se c è di interpunzione          |

# Funzioni di utilita'

- Funzioni matematiche intere

| <i>funzione</i>                                       | <i>definizione</i>                   |
|-------------------------------------------------------|--------------------------------------|
| <code>int abs (int n)</code>                          | <b>valore assoluto</b>               |
| <code>long labs (long n)</code>                       | <b>valore assoluto</b>               |
| <code>div_t div (int numer, int denom)</code>         | quoto e resto della divisione intera |
| <code>ldiv_t ldiv<br/>(long numer, long denom)</code> | 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

| <i>funzione</i>                                             | <i>definizione</i>              |
|-------------------------------------------------------------|---------------------------------|
| <code>char* strcat (char* s1, char* s2) ;</code>            | <i>concatenazione<br/>s1+s2</i> |
| <code>char* strchr (char* s, int c) ;</code>                |                                 |
| <code>int strcmp (char* s1, char* s2) ;</code>              | <i>confronto</i>                |
| <code>char* strcpy (char* s1, char* s2) ;</code>            | <i>s1 &lt;= s2</i>              |
| <code>int strlen (char* s) ;</code>                         | <i>lunghezza di s</i>           |
| <code>char* strncat (char* s1, char* s2, int n) ;</code>    | <i>concat. n car. max</i>       |
| <code>char* strncpy (char* s1, char* s2, int n) ;</code>    | <i>copia n car. max</i>         |
| <code>char* strncmp (char* dest, char* src, int n) ;</code> | <i>fr. n car. max</i>           |

# Compilatore C

- **gcc prova.c**
  - Genera a.out come eseguibile
- **gcc -g prova.c**
  - Genera a.out con info di debugging
- **gcc -o prova prova.c**
  - Genera un eseguibile con il nome **prova**

**Per eseguire il file ottenuto:**

**./prova**