



Programma al Maggio 2006

- Rappresentazione della informazione
- Algoritmi in pseudocodice e flow charts
- Dati semplici (tutto).... *no POINTER*
- Variabili ed Espressioni
- Dati strutturati (tutto)
- Strutture di programma selezioni e cicli
- Funzioni elementari e struttura del programma (dati globali, include, librerie, funzioni esterne ..)

Rappresentazione della informazione

- Solo cifre 1 e 0 in unità di memoria (byte, word)
- Quanti “pattern” diversi con n bit? (2^n)
- Associazione dell’informazione ai “pattern”
 - ◆ Algoritmi (numeri $x_1 * 2^{n-1} + x_2 * 2^{n-2} + \dots$)
 - ◆ Tabelle (caratteri - Codice Ascii)
- Scrittura dei valori ('X' "Abc" 32 032 0x3F)

Tabella ASCII

bit meno
significativi



bit più significativi

	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP ₃₂	0 ₄₈	@ ₆₄	P ₈₀	` ₉₆	p ₁₁₂
0001	SOH	DC1	! ₃₃	1 ₄₉	A ₆₅	Q ₈₁	a ₉₇	q ₁₁₃
0010	STX	DC2	« ₃₄	2 ₅₀	B ₆₆	R ₈₂	b ₉₈	r ₁₁₄
0011	ETX	DC3	# ₃₅	3 ₅₁	C ₆₇	S ₈₃	c ₉₉	s ₁₁₅
0100	EOT	DC4	\$ ₃₆	4 ₅₂	D ₆₈	T ₈₄	d ₁₀₀	t ₁₁₆
0101	ENQ	NAK	% ₃₇	5 ₅₃	E ₆₉	U ₈₅	e ₁₀₁	u ₁₁₇
0110	ACK	SYN	& ₃₈	6 ₅₄	F ₇₀	V ₈₆	f ₁₀₂	v ₁₁₈
0111	BEL	ETB	' ₃₉	7 ₅₅	G ₇₁	W ₈₇	g ₁₀₃	w ₁₁₉
1000	BS	CAN	(₄₀	8 ₅₆	H ₇₂	X ₈₈	h ₁₀₄	x ₁₂₀
1001	HT	EM) ₄₁	9 ₅₇	I ₇₃	Y ₈₉	i ₁₀₅	y ₁₂₁
1010	LF	SUB	* ₄₂	: ₅₈	J ₇₄	Z ₉₀	j ₁₀₆	z ₁₂₂
1011	VT	ESC	+ ₄₃	; ₅₉	K ₇₅	[₉₁	k ₁₀₇	{ ₁₂₃
1100	FF	FS	, ₄₄	< ₆₀	L ₇₆	\ ₉₂	l ₁₀₈	₁₂₄
1101	CR	GS	- ₄₅	= ₆₁	M ₇₇] ₉₃	m ₁₀₉	} ₁₂₅
1110	SO	RS	. ₄₆	> ₆₂	N ₇₈	^ ₉₄	n ₁₁₀	~ ₁₂₆
1111	SI	US	/ ₄₇	? ₆₃	O ₇₉	- ₉₅	o ₁₁₁	DEL

Scrittura dei valori

Costanti

Numeri interi

Rappresentano numeri relativi (quindi con segno):

	2 byte	4 byte
base decimale	12	70000, 12L
base ottale	014	0210560
base esadecimale	0xFF	0x11170

Numeri reali

Varie notazioni:

24.0 2.4E1 240.0E-1

Suffissi: l, L, u, U (interi-long, unsigned)
f, F (reali - floating)

Prefissi: 0 (ottale) 0x, 0X(esadecimale)

Dati semplici

- Una unità manipolabile dotata di TIPO
 - ◆ Codifica usata per rappresentare i valori
 - ◆ Quali valori sono validi
 - ◆ Quali operazioni sono lecite e come agiscono tali operazioni
- ◆ Es. `int` numeri compl a 2, -32.768 a +32.767
 - + - * / % -- ++ == != < > <= >=

Variabili

- Sono contenitori di informazioni
- Associano un nome simbolico a una locazione di memoria
`int A;`
- Associano un TIPO al nome

Rappresentazione in memoria della variabile **A**

indirizzo di A

A valore di A

- Con `&A` si indica l'indirizzo di A

Dati strutturati

- Composizioni di dati semplici
- Regole di composizione
 - ◆ **[n]** *n* oggetti uguali e dello stesso tipo ($0 \leftrightarrow n-1$)
 - ◆ `struct {` *dich di var;*
dich di var; `}`
- Regole di accesso
 - ◆ `A[12]` dodicesimo elemento di A
 - ◆ `A.B` componente B del dato A

Espressioni

- Variabili e Costanti cui si applicano Operatori secondo la loro precedenza
- Se le variabili sono strutturate si deve accedere ai componenti per cui sono definiti gli operatori
 - ◆ `(V<<1 | 7>>1 & 0xB)`
 - ◆ `S.CO[2] + B / 015 - V + S.Index`
 - ◆ `((i<S.CO[5]) && (i > S.NO[5] - S.CO[4]))`

Assegnamento

- E' la frase principale con cui si fanno cambiare i valori delle variabili
 - ◆ $\text{VAR} = \langle \text{espressione} \rangle;$
- Ma anche in uso le varianti
 - $+=$ $-=$ $/=$ ecc.

Struttura del programma

```
main ()
```

```
{
```

```
  parte dichiarativa  
  locale
```

```
  parte esecutiva
```

```
}
```

parola riservata (identificatore di funzione)
appare una e una sola volta nel programma
definisce l'inizio dell'esecuzione
è (formalmente) una funzione

definisce l'insieme di «oggetti» usati dal
programma principale per l'esecuzione.
sono oggetti visibili (locali) a main.

insieme di istruzioni che costituiscono il
programma principale



Controllo della esecuzione

- Le istruzioni del programma possono essere eseguite
 - ◆ In sequenza
 - ◆ Alternativamente
 - ◆ Iterativamente
- Ogni frase del C, anche se composta di più frasi, è equivalente ad una sola frase.

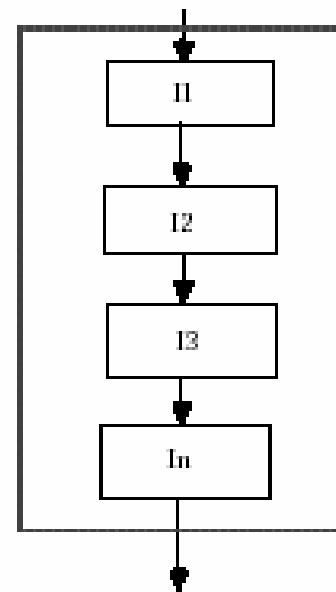
Istruzione composta { }

Determina l'esecuzione nell'ordine testuale delle istruzioni componenti.

Sintassi:

```
{  
  <Dichiarazioni e Definizioni>  
  <Sequenza di Istruzioni>  
}
```

```
<sequenza-istruzioni> ::=  
  <istruzione> { ; <istruzione> }
```



Sintatticamente equivalente a una singola istruzione (strutturata).

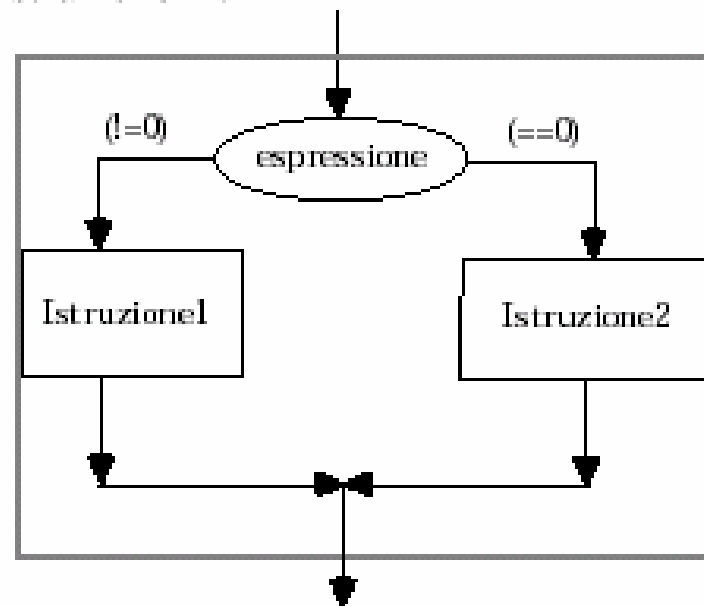
Istruzioni di alternativa

Istruzione if:

seleziona l'esecuzione di una sola tra le istruzioni componenti in base al risultato di un'espressione logica (detta selettore).

```
if (<espressione>
    <istruzione1>
[else
    <istruzione2>]
```

se il risultato di <espressione> è *vero* (cioè, diverso da zero) viene eseguita <istruzione1>, altrimenti viene eseguita <istruzione2>.



Oppure

switch

```
switch ( integer expression )
{
    case int-const1: statement1;
                    statement2;
                    break;

    case int-const2: statement3;
                    statement4;
                    break;

    :
    default: statement5;
            statement6;
            break;
}
```

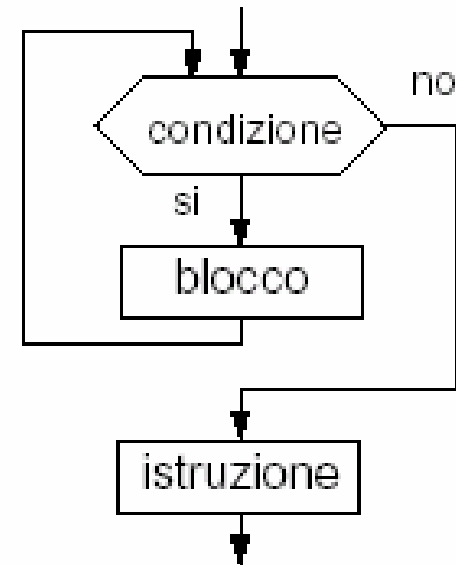
Iteration

- 2 Basic types of repetition control
- Counter controlled
 - loop is done until counter reaches a predetermined ending value
 - needs: NAME of counter, INITIAL VALUE, INCREMENT amount, and FINAL VALUE
- Sentinel Controlled
 - looping continues until some event occurs or some value is encountered

CICLO A CONDIZIONE INIZIALE (**Sentinel Controlled**)

Sintassi

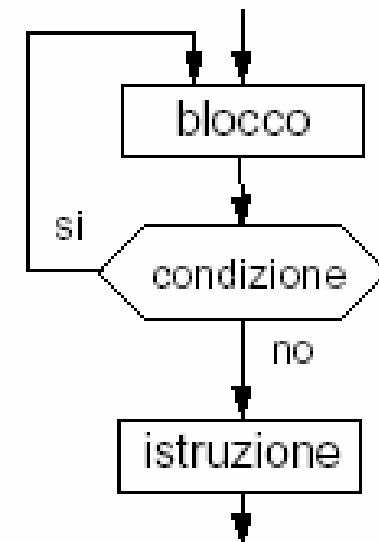
```
while (<condizione>)  
{  
    <blocco>  
}  
<istruzione>
```



CICLO A CONDIZIONE FINALE

Sintassi

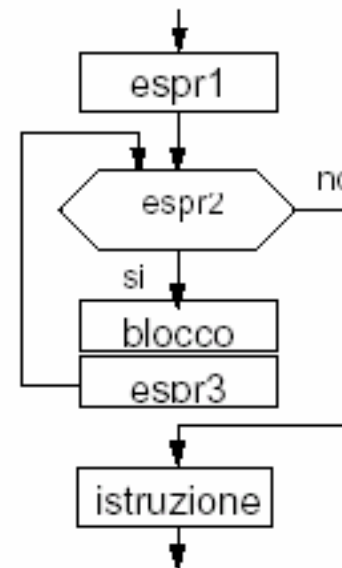
```
do  
{  
    blocco  
} while (<condizione>);  
<istruzione>
```



Ciclo con contatore

Il linguaggio C prevede anche un costrutto di ciclo che racchiude **inizializzazione**, **test** e **modifica** di una variabile.

```
for (espr1; espr2; espr3)
{
    <istr del ciclo>
}
<istruzione>
```



espressione1: deve definire il valore iniziale della variabile di conteggio

espressione2: deve definire la condizione sul valore finale della variabile di conteggio

espressione3: deve definire la modifica della variabile di conteggio



Funzioni elementari

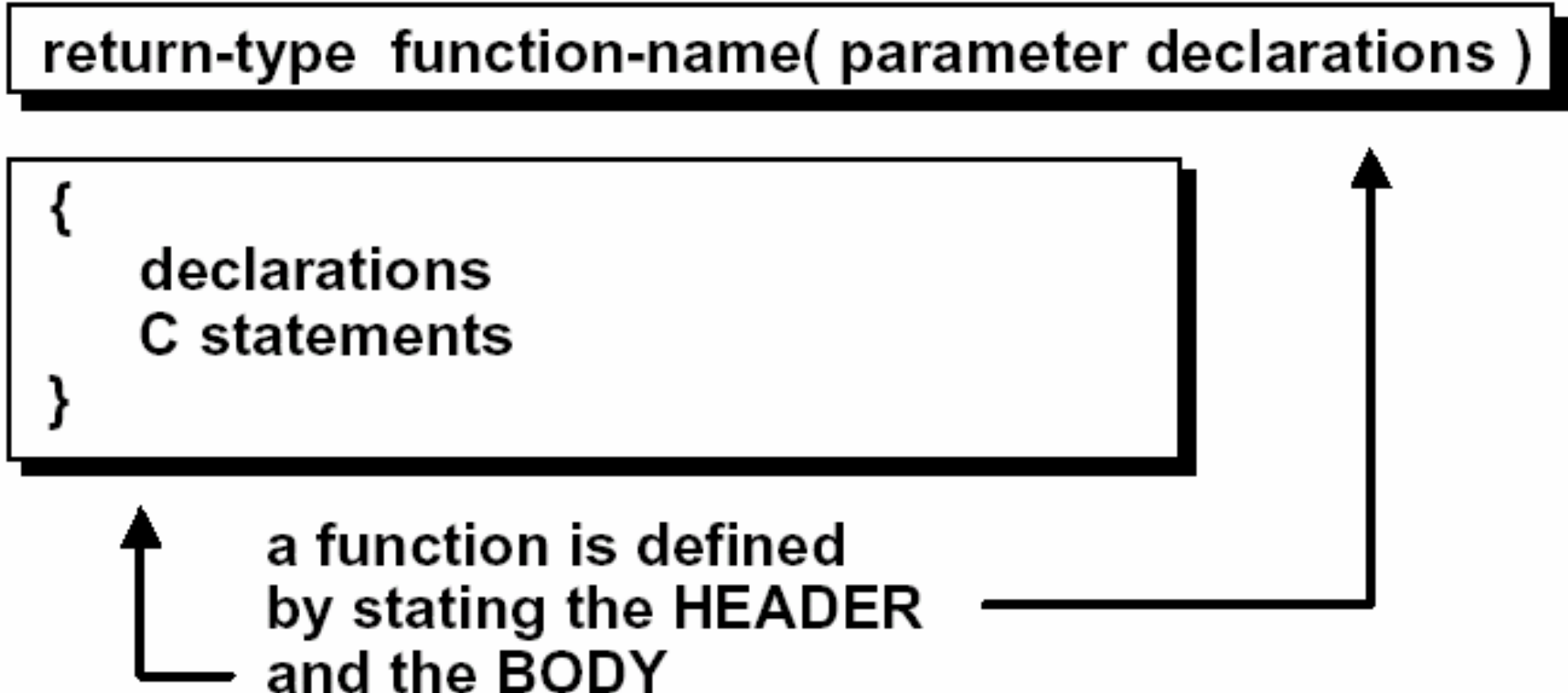
- Le funzioni sono un modo per comporre le frasi assegnando un NOME alla operazione complessa (funzione) eseguita
- L'operazione complessa può essere eseguita in modo parametrico (rispetto ad una lista di parametri formali)
- L'operazione complessa può ritornare un valore (come una $f(x)$)

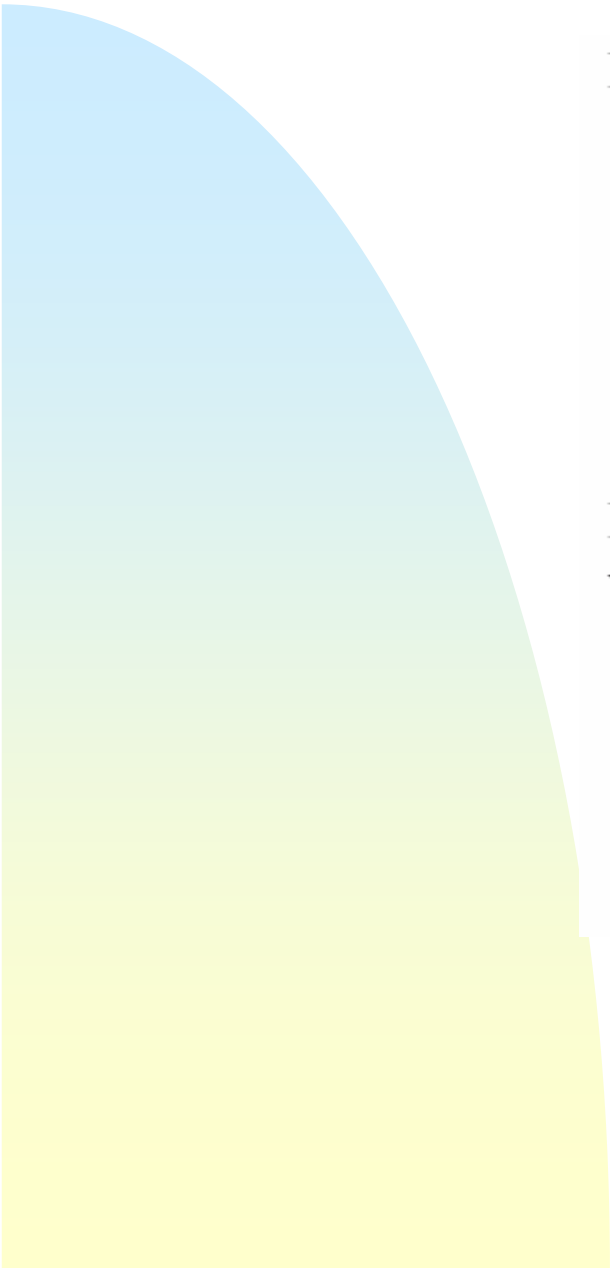
Function definition general form:

```
return-type function-name( parameter declarations )
```

```
{  
  declarations  
  C statements  
}
```

a function is defined
by stating the **HEADER**
and the **BODY**





Es. funzione che somma due valori di tipo int e restituisce un int:

```
int somma(int a, int b)
{
    int sum;
    sum = a+b;
    return(sum);
}
```

La chiamata della funzione viene fatta così:

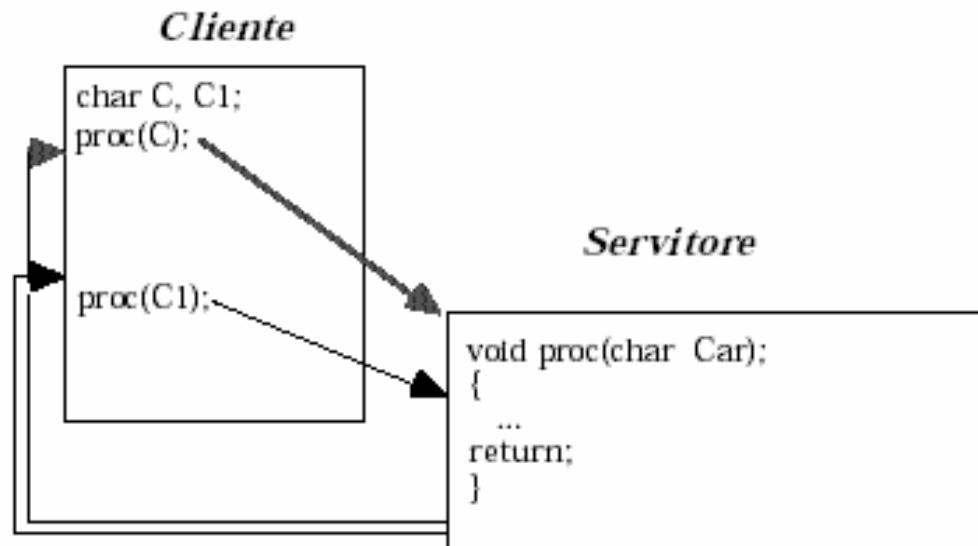
```
void main(void)
{
    int A=23; int B=-31; int risultato;
    risultato = somma(A,B);
    printf("somma= %d\n", risultato);
}
```

I

Passaggio dei parametri

Modello Cliente-Servitore

- Il VALORE del parametro ATTUALE viene COPIATO nel parametro FORMALE
- È come se
 - ◆ `Car=C`
 - ◆ `Car=C1`



Per proseguire

- E' necessario introdurre un nuovo Tipo di variabili
- Un tipo semplice per costruire tipi complessi

👉 **II** **POINTER**

I costruttori

- Tipi strutturati: **struct**
- Considerazioni sui metodi di strutturazione
 - ◆ Sequenza: *struct*
 - ◆ Iterazione: *array*
 - ◆ Selezione: *union*
- Costruttore di tipi POINTER:
 - ☞ fuori dagli schemi
- Interazione tra *array*, *struct* e *Pointer*

