

086180 - INFORMATICA APPLICATA

A.A. 2010-11

2° semestre

# M06<sub>A</sub> – RIEPILOGO

# Programma al Maggio 2011

- ⦿ Rappresentazione della informazione
- ⦿ Algoritmi
- ⦿ Dati semplici (tutto)....
- ⦿ Variabili ed Espressioni
- ⦿ Strutture di programma selezioni e cicli
- ⦿ Funzioni elementari e struttura del programma (dati globali, include, librerie,..)

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

# Espressioni

- Variabili e Costanti cui si applicano Operatori secondo la loro precedenza

- `SCO + B / 015 - V + k`
- `((i < SCO) && (i > SNO - SCO))`
- 

Precedenze fra Operatori	Operato	Associativita'
() [] ->	.	-> da sinistra a destra
! ~ ++ -- + - & *		
(tipo) sizeof		<- da destra a sinistra
* / %		-> da sinistra a destra
+ -		-> da sinistra a destra
<< >>		-> da sinistra a destra
< <= > >=		-> da sinistra a destra
== !=		-> da sinistra a destra
&		-> da sinistra a destra
^		-> da sinistra a destra
		-> da sinistra a destra
&&		-> da sinistra a destra
		-> da sinistra a destra
?:		<- da destra a sinistra
= += -= *= /= %=		
&= ^=  = <<= >>=		<- da destra a sinistra
,		-> da sinistra a destra

# Assegnamento

⦿ E' la frase principale con cui si fanno cambiare i valori delle variabili

- VAR = <espressione>;

⦿ 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.

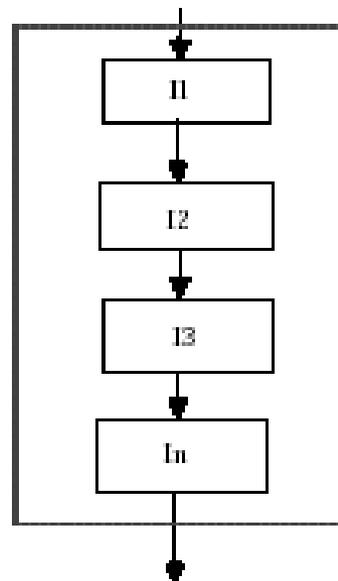
# Istruzione composta { }

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).

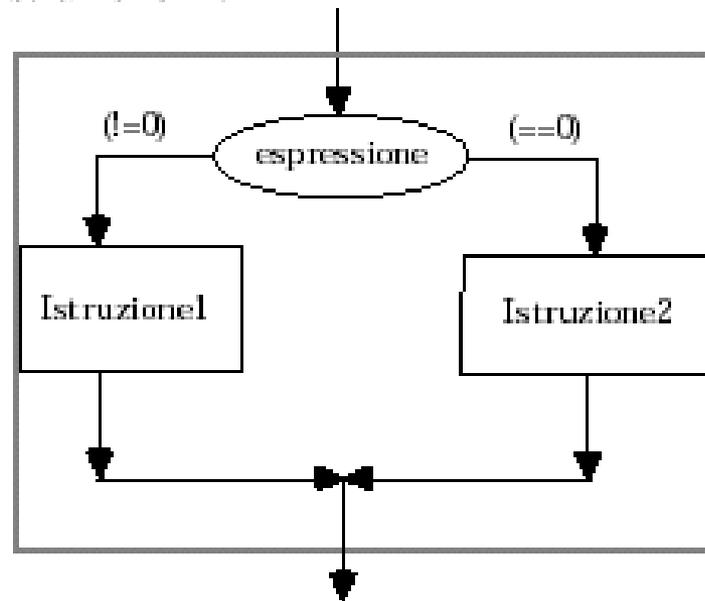
# Istruzioni di alternativa

## 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;
}
```

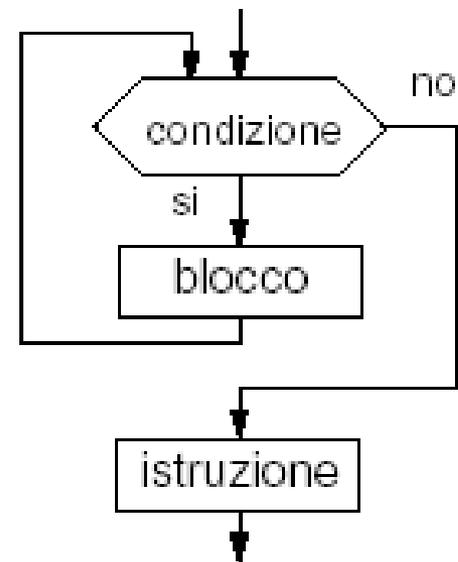
# Iteration

- 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

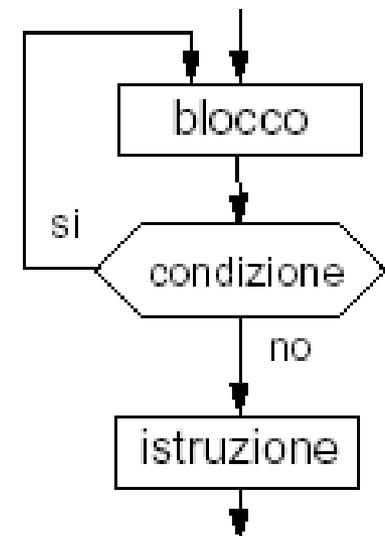
```
while (<condizione>)  
{  
    <blocco>  
}  
<istruzione>
```



## CICLO A CONDIZIONE FINALE

### Sintassi

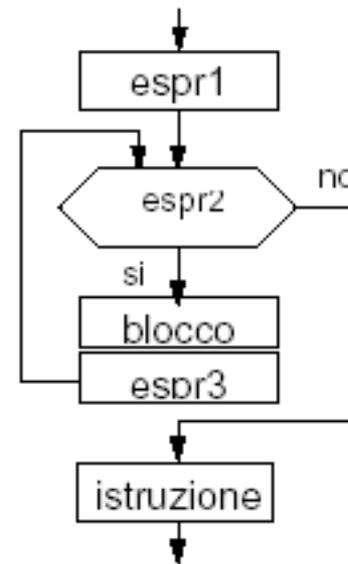
```
do  
{  
    blocco  
} while (<condizione>);  
<istruzione>
```



# 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

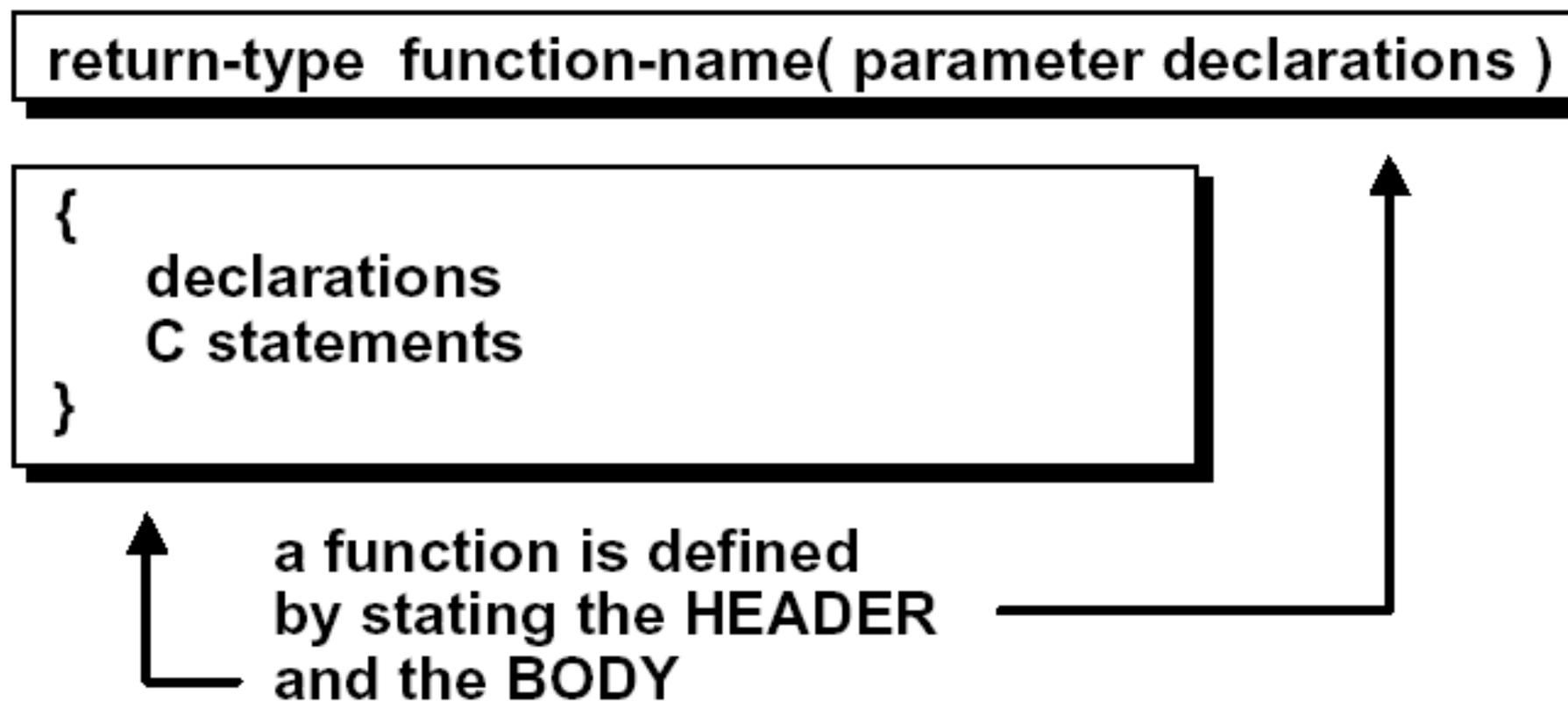
- ⦿ Le funzioni sono un modo per comporre le frasi assegnando un NOME alla operazione complessa (funzione) eseguita
- ⦿ L'operazione complessa può essere eseguita in modo parametrico (rispetto ad una lista di parametri formali)
- ⦿ L'operazione complessa può ritornare un valore (come una  $f(x)$  )

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);
}
```

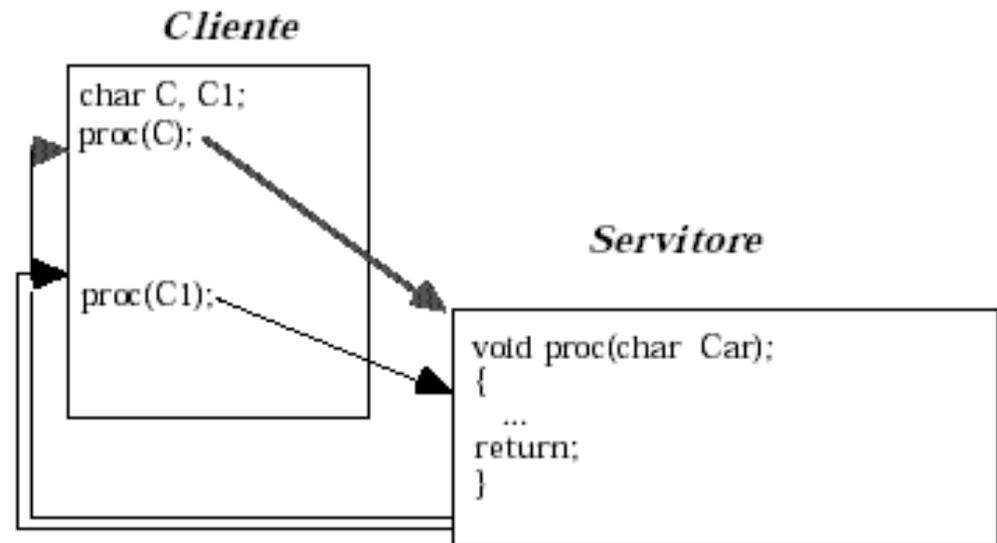
La chiamata della funzione viene fatta così:

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

# Passaggio dei parametri

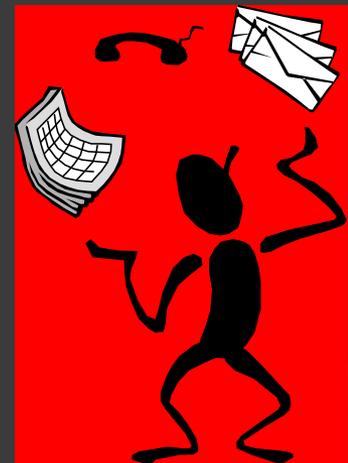
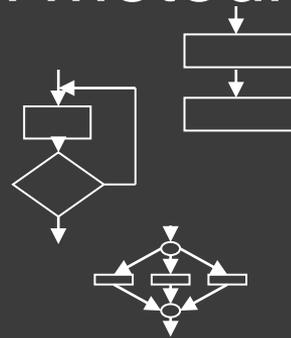
- Il VALORE del parametro ATTUALE viene COPIATO nel parametro FORMALE
- È come se
  - Car=C .....
  - Car=C1

## Modello Cliente-Servitore



# SEGUE ..... I costruttori

- Tipi strutturati: **array** [x] e **struct**
- Considerazioni sui metodi di strutturazione
  - Sequenza: *struct*
  - Iterazione: *array*
  - Selezione: *union*



- Interazione tra *array* e *struct*