Lezione 7 e 8

- Tipi di dato built-in
- Rappresentazione numerica
- Rappresentazione dei caratteri
- Vettori e matrici
- Stringhe

Fabio Scotti (2004-2009)

Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)

Laboratorio di programmazione

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 7 e 8

Rappresentazione dell'informazione:

Obiettivo :

- Conoscere approfonditamente l'uso dei tipi di dato nella programmazione in C.

Rappresentazione dell'informazione in C

- **Il C permette di definire e trattare vari tipi di dato ad esempio interi o caratteri.**
- **La dichiarazione del tipo di dato è presente in tutte le dichiarazione della variabili.**
- **Al nome di una variabile viene associato il suo tipo ad esempio:**
  - `int indice;`
- **Per tipo di dato si intende:**
  - un insieme di valori
  - un insieme di operazioni che possono essere applicate
- **Esempio: i numeri interi e le usuali operazioni aritmetiche come la somma, sottrazione, etc.**

3

Rappresentazione in memoria

- **Ogni tipo di dato ha una rappresentazione in memoria (codifica binaria) che utilizza un certo numero di celle di memoria.**
- **Possiamo trattare le informazioni in maniera astratta:**
  - se sommiamo due interi non ci preoccupiamo di come devono essere sommati i bit dei due addendi in memoria
  - indichiamo un segno di somma nell'istruzione in C
  - ad esempio scriviamo `c = a+b;`

4

Variabili con tipo

- **L'uso di variabili con tipo ha importanti conseguenze:**
  - per ogni variabile è possibile determinare a priori
    - **l'insieme dei valori ammissibili**
    - **l'insieme delle operazioni applicabili**
    - **la quantità di memoria necessaria**
  - è possibile **rilevare a priori** (a tempo di compilazione)
    - **errori nell'uso delle variabili**

5

Tipo di dato (1)

- **I tipi di dato si dividono in:**
  - tipi semplici
  - tipi strutturati
- **I tipi semplici consentono**
  - di rappresentare informazioni composte da un valore
  - esempio: una temperatura, una velocità
- **I tipi strutturati consentono**
  - di rappresentare informazioni costituite dall'aggregazione di varie componenti
  - esempio una data, una matrice, una cartella clinica

6

Tipo di dato (2)

- **Un valore di un tipo semplice è indivisibile**
- **Un valore di un tipo strutturato può essere scomposto nei valori delle sue componenti**
- **Esempio:**
  - un valore di tipo "data" è costituito da tre valori (semplici):
    - `int giorno;`
    - `int mese;`
    - `int anno;`

7

Tipo di dato built-in (1)

- **Il C mette a disposizione**
  - un insieme di tipi predefiniti (tipi built-in)
  - dei meccanismi per definire nuovi tipi (user-defined)
- **I tipi semplici che il C mette a disposizione sono:**
  - Interi
  - Reali
  - Caratteri

8

Tipo di dato built-in (2)

- **Il tipo deve essere scelto tenendo conto della variabile che intendiamo usare.**
- **Esempio:**
  - Per una variabile che deve assumere solo valori interi (es. un contatore di un ciclo)
    - possiamo dichiarare una variabile intera cont
    - `int cont;`
  - Se dobbiamo impiegare una variabile che può assumere dei valori reali (es. la diagonale di un generico rettangolo)
    - useremo un variabile reale diag
    - `float diag;`

9

Tipo di dato built-in (3)

- **Per ogni tipo analizziamo i seguenti aspetti:**
  - intervallo di definizione (se applicabile)
  - notazione per le costanti
  - operatori
  - predicati (operatori di confronto)
  - formati di ingresso/uscita
- **La dimensione dell'occupazione in memoria di una variabile o di un tipo si ottiene con:**
  - La funzione `sizeof()`
  - che restituisce lo spazio di memoria occupato in byte

10

`int`

Interi con segno (int) (1)

- **In C esistono 3 tipi di interi con segno:**
  - **short** (equivalente a: `short int`, `signed short`, `signed short int`)
  - **int** (equivalente a: `signed int`, `signed`)
  - **long** (equivalente a: `long int`, `signed long`, `signed long int`)
- **L' intervallo di definizione**
  - va da -2^{n-1} a $+2^{n-1}-1$
  - dove n dipende dal compilatore
  - e rappresenta il numero di bit impiegati per la loro rappresentazione

12

Interi con segno (int) (2)

- **Valgono le seguenti relazioni:**
  - `sizeof(short)` ≤ `sizeof(int)` ≤ `sizeof(long)`
  - `sizeof(short)` ≥ 2 **(ovvero, almeno 16 bit)**
  - `sizeof(long)` ≥ 4 **(ovvero, almeno 32 bit)**
- **Nel caso del compilatore gcc si ha :**
  - short: 16 bit
  - int: 32 bit
  - long: 32 bit

13

Interi con segno (int) (3)

- **I valori limite**
  - sono contenuti nel file limits.h,
  - sono definiti dalle costanti:
    - SHRT_MIN, SHRT_MAX,
    - INT_MIN, INT_MAX,
    - LONG_MIN, LONG_MAX
- **Possono servire**
  - per controllare se il dato immesso da un utente cade nell'intervallo di rappresentazione
  - per evitare un errore segnalando all'utente il problema

14

Interi con segno (int) (4)

- **Costanti:**
  - sono gli interi semplicemente in decimale,
  - ad esempio 0, 10, -10, . . .
- **Operatori:**

  `+, -, *, /, %,`

  `==, !=, <, >, <=, >=`
- **Specificatori di formato per funzioni ingresso/uscita (dove d indica "decimale"):**
  - `%hd` per short
  - `%d` per int
  - `%ld` per long (con `l` minuscola)
- **Es: se `cont` è un short** `printf("%hd", cont);`

15

Interi senza segno (unsigned) (1)

- **In C esistono 3 tipi di interi senza segno:**
  - **unsigned short** (**equivalente a**: `unsigned short int`)
  - **unsigned int**
  - **unsigned long** (**equivalente a:** `unsigned long int`)
- **L' intervallo di definizione**
  - va da 0 a $+2^n-1$
  - dove n dipende dal compilatore
  - e rappresenta il numero di bit impiegati per la loro rappresentazione

16

Interi senza segno (unsigned) (2)

- **I valori limite**
  - sono contenuti nel file limits.h,
  - sono definiti dalle costanti:
    - USHRT_MAX, UINT_MAX, ULONG_MAX
    - il minimo è sempre 0
- **Costanti:**
  - decimale: come per interi con segno
  - esadecimale: 0xA, 0x2F4B, . . .

17

Interi senza segno (unsigned) (3)

- **Ingresso/uscita: tramite printf e scanf, con i seguenti specificatori di formato:**
  - `%u` per numeri in decimale
  - `%o` per numeri in ottale
  - `%x` per numeri in esadecimale con cifre 0, . . . , 9, a, . . . , f
  - `%X` per numeri in esadecimale con cifre 0, . . . , 9, A, . . . , F
- **Per interi**
  - short : si antepone `h`
  - long : si antepone `l` (minuscola)

18

float

Tipi reali o floating point (float) (1)

- **In C i tipi reali vengono rappresentati con un punto (es: `3.56`) e ne esistono 3 tipi :**
  - **float** (dimensione in memoria 4 byte)
  - **double** (dimensione in memoria 8 byte)
  - **long double** (dimensione in memoria 12 byte)
- **In particolare questa notazione si chiama in virgola mobile e verrà presentata in altri corsi**

Tipi reali o floating point (float) (2)

- **Le grandezze e gli intervalli di questi tipi di dato**
  - sono scritte nel file float.h di libreria
  - dipendono dal compilatore
- **Costanti:**
  - semplicemente in decimale USANDO SEMPRE IL PUNTO
  - notazione scientifica
- **Esempi:**
  - double x, y, z, w;
  - x = 123.45; y = 0.0034; y = .0034;
  - z = 34.5e+20; z = 34.5E+20; w = 5.3e-12;

Tipi reali o floating point (float) (3)

- **Uscita: tramite printf con i seguenti specificatori di formato:**
  - `%f` per stampare il numero in virgola fissa
  - `%8.3f` per stampare 8 cifre complessive, di cui 3 cifre decimali
  - `%e` per stampare in forma esponenziale
- **Possiamo stampare anche in forma ottimizzata**
  - %g (oppure %G) sceglie la forma più compatta fra la notazione esponenziale (%e) e quella di %f
- **Ingresso: con scanf si usa indifferentemente**
  - `%f` o `%e`.

Tipi reali o floating point (float) (4)

- **Esempio:**
  - `float x = 123.45;`
    `printf("|%f| |%8.3f| |%-8.3f|\n", x, x, x);`
  - Otteniamo sul nostro output
    |123.449997| | 123.450| |123.450 |

Il risultato viene allineato tutto a sinistra

Ci sono 7 caratteri e ne devono essere rappresentati 8: viene aggiunto uno spazio a destra

- **Esempio:**
  - `double x = 123.45;`
    `printf("|%e| |%10.3e| |%-10.3e|\n", x, x, x);`
  - Otteniamo sul nostro output
    |1.234500e+02| | 1.234e+02| |1.234e+02 |

char

Tipi caratteri (char) (1)

- **Un codice associa ad ogni carattere un intero.**
- **Esempio il codice ASCII associa**
  - al carattere '#' il numero decimale 35
  - alle lettere '0'… '9' della nostra tastiera i numeri decimali dal 48 al 57
  - ai caratteri 'a'… 'z' della nostra tastiera sono stati associati i numeri decimali dal 97 al 122.
- **I caratteri in C possono essere usati come interi**
  - se ci ricordiamo la corrispondenza fra il carattere ed il suo valore nel codice (es: a→97)

25

Tipi caratteri (char) (2)

- **In C esistono 3 tipi di caratteri:**
  - **char**
  - **signed char** (rappresentato in complemento a 2)
  - **unsigned char**
- **I tipi signed ed unsigned char sono esattamente rappresentati come interi.**
- **Invece il tipo char**
  - dipende dal compilatore e
  - può essere stato definito come signed oppure unsigned

26

Tipi caratteri (char) (3)

- **Valgono le seguenti relazioni:**
  - `sizeof(char)` ≤ `sizeof(int)`
  - `sizeof(char)` vale 8 bit (1 byte)
- **Operatori come per gli interi:**

  `+, –, *, /, %,`

  `==, !=, <, >, <=, >=`
- **Esempi**
  - se x='a' e y='b' allora la proposizione (y>x) è vera
    il contenuto di x è 97 (ovvero 'a' nel codice ASCII)
    il contenuto di y è 98
  - se un char x rispetta la seguente proposizione ((x>96)&&(x<123)) allora è un carattere minuscolo

27

Tipi caratteri (char) (4)

- **Ingresso/uscita: tramite printf e scanf, con il seguenti specificatore di formato:**
  - `%c`
- **Esempio: visualizzare due interi sia con formato intero (%d) sia carattere (%c)**

```
int i, j;
    printf("Immetti due interi ---> \n");
scanf("%d%d", &i, &j);
printf("%d %d\n", i, j);
printf("%c %c\n", i, j);
```

Output alle due printf:

```
35 48
#  a
```

28

Esempio studio allocazione memoria (1)

- **Esempio**
  - definiamo 4 variabili di tipo carattere, 4 di tipo int, 4 di tipo long e 4 di tipo double.
  - Otteniamo la loro occupazione in memoria.
  - calcoliamo l'indirizzo delle variabile i1 che è ottenibile scrivendo &i1
  - lo stampiamo in due modi: in formato esadecimale (%p nella printf) e decimale (%d nella printf)
  - nella printf scriviamo sempre 0x davanti ad un numero che verrà stampato in modo esadecimale per esplicitarlo all'utente.

29

Esempio (2)

```
// Occupazione di memoria dei tipi di dato
#include <stdio.h>
int main()
{
    char c1, c2, c3, c4;
    int i1, i2, i3,i4;
    long l1, l2, l3, l4;
    double d1, d2, d3, d4;
    // SOLO per stavolta non inizializziamo le celle di memoria
    printf("\n*******************************************\n");
    printf("* Occupazione in memoria \n");
```

```
**************************************************
* Occupazione in memoria
**************************************************

il char c1 si trova  0x 0022FF6F (2293615)
il char c2 si trova  0x 0022FF6E (2293614)
il char c3 si trova  0x 0022FF6D (2293613)
il char c4 si trova  0x 0022FF6C (2293612)
l' int  i1 si trova  0x 0022FF68 (2293608)
l' int  i1 si trova  0x 0022FF64 (2293604)
l' int  i1 si trova  0x 0022FF60 (2293600)
l' int  i1 si trova  0x 0022FF5C (2293596)
il long l1 si trova  0x 0022FF58 (2293592)
il long l2 si trova  0x 0022FF54 (2293588)
il long l3 si trova  0x 0022FF50 (2293584)
il long l4 si trova  0x 0022FF4C (2293580)
il double d1 si trova  0x 0022FF40 (2293568)
il double d2 si trova  0x 0022FF38 (2293560)
il double d3 si trova  0x 0022FF30 (2293552)
il double d4 si trova  0x 0022FF28 (2293544)
```

```
    ************************\n\n");
    in 0x %p (%d) \n", &c1 , &c1);
    in 0x %p (%d) \n", &c2 , &c2 );
    in 0x %p (%d) \n", &c3 , &c3 );
    in 0x %p (%d) \n", &c4 , &c4 );
    in 0x %p (%d) \n", &i1 , &i1 );
    in 0x %p (%d) \n", &i2 , &i2 );
    in 0x %p (%d) \n", &i3 , &i3 );
    in 0x %p (%d) \n", &i4 , &i4 );
    in 0x %p (%d) \n", &l1 , &l1);
    in 0x %p (%d) \n", &l2 , &l2 );
    in 0x %p (%d) \n", &l3 , &l3 );
    in 0x %p (%d) \n", &l4 , &l4 );
va in 0x %p (%d) \n", &d1 , &d1);
va in 0x %p (%d) \n", &d2 , &d2 );
va in 0x %p (%d) \n", &d3 , &d3 );
va in 0x %p (%d) \n", &d4 , &d4 );
    getchar();
}
```

30

Esempio (3)

```
**********************************************
* Occupazione in memoria
**********************************************

il char c1 si trova  0x 0022FF6F (2293615)
il char c2 si trova  0x 0022FF6E (2293614)
il char c3 si trova  0x 0022FF6D (2293613)
il char c4 si trova  0x 0022FF6C (2293612)
l' int  i1 si trova  0x 0022FF68 (2293608)
l' int  i1 si trova  0x 0022FF64 (2293604)
l' int  i1 si trova  0x 0022FF60 (2293600)
l' int  i1 si trova  0x 0022FF5C (2293596)
il long l1 si trova  0x 0022FF58 (2293592)
il long l2 si trova  0x 0022FF54 (2293588)
il long l3 si trova  0x 0022FF50 (2293584)
il long l4 si trova  0x 0022FF4C (2293580)
il double d1 si trova  0x 0022FF40 (2293568)
il double d2 si trova  0x 0022FF38 (2293560)
il double d3 si trova  0x 0022FF30 (2293552)
il double d4 si trova  0x 0022FF28 (2293544)
```

- **Le variabili sono state allocate rispettando l'ordine di dichiarazione (c1, c2, c3, c4, i1,…).**
- **La memoria libera è quindi verso i numeri bassi in quanto le celle di memoria sono state allocate dalla …615 verso la …544**
- **i caratteri occupano un solo byte, gli interi 4 byte, i long occupano 4 byte e i double 8 byte.**

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 7 e 8

Vettori matrici e stringhe:

Obiettivo :

- Essere in grado di utilizzare adeguatamente vettori, matrici e stringhe.

Array (1)

- **L'array è una struttura dati**
  - costituita da un insieme di variabili dello stesso tipo di dato (intero, carattere, reale ecc.),
  - a cui è possibile accedere tramite un nome, e referenziare uno specifico elemento attraverso un indice.
- **in C gli elementi di un array sono allocati in memoria in celle adiacenti, in cui**
  - l'indirizzo più basso corrisponde al primo elemento dell' array
  - l'indirizzo più alto all'ultimo elemento dell'array

33

Array (2)

- **Gli array possono essere definiti**
  - ad una dimensione (vettori),
  - a due dimensioni (matrici)
  - ad n dimensioni (array multidimensionali)
- **Le stringhe sono array di caratteri**

34

Vettori (1)

- **Un vettore è**
  - un insieme contiguo monodimensionale di elementi dello stesso tipo (vengono allocati in memoria in un blocco unico)
- **E' possibile accedere ad un vettore tramite un nome**
- **Ogni specifico elemento si referenzia attraverso un indice**

35

Vettori (2)

- **Dichiarazione di un vettore:**
  - il tipo di elemento del vettore (per esempio int)
  - spazio
  - il nome del vettore (ad esempio x)
  - fra parentesi quadre si dichiara il numero di elementi che il vettore dovrà contenere.
  - **Esempio:** `int x[100];`
- **Come si accede agli elementi del vettore?**
  - usando il nome del vettore e scrivendo fra parentesi quadre l'indice dell'elemento a cui vogliamo accedere
  - **Esempio:** `x[0]= 1;`

36

Vettori (3)

- **Attenzione: il C numera gli elementi di un vettore a partire da 0.**
- **Esempio:**
  - vettore x di 100 elementi interi
  - assegnare il valore 1 al primo elemento:

```
int x[100];
x[0]=1;
```

  - assegnare all'ultimo elemento il valore 100:

```
x[99]=100;
```

37

Vettori (4)

- **I vettori di solito vengono gestiti mediante cicli `for`.**
- **Il `for` deve essere usato quando si conosce il numero di cicli da eseguire.**
  - il numero di elementi di un vettore è fissato fin dalla dichiarazione
- **Esempio:**
  - assegnare al vettore **x**, i numeri da 1 a 100:

```
int x[100];
for(i=0; i<100; i++)
     x[i]=i+1;
```

38

Inizializzazione di un vettore

- **L'inizializzazione di un vettore può essere effettuata nel seguente modo:**

```
int v[5]={1,67,22,6,34};
```

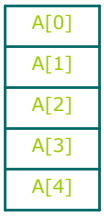
- **Oppure con un ciclo `for`:**

```
int v[5];
int i;
for (i=0; i<5; i++)
      v[i]= 0 ;
```

39

Vettori in memoria (1)

- `int A[5];`



Attenzione:
`A[-1]` e `A[5]`
NON ESISTONO!

- **Esempio:**

```
A[3] = 7;  //7 assegnato alla quarta componente di A
```

- **Il vettore può essere gestito con espressioni aritmetiche :**

```
j=2;
A[j+2] = 5; // 5 assegnato ad A[4]
```

40

Vettori in memoria (2)

1001 byte A[0]
1002 byte
1003 byte
1004 byte
1005 byte A[1]
1006 byte
1007 byte
1008 byte
1009 byte A[2]
1010 byte
1011 byte
1012 byte
1013 byte A[3]
1014 byte
1015 byte
1016 byte

- **Calcolare l'occupazione di memoria del vettore dichiarato come `int A[4];`**

```
printf("La memoria occupata dal A e'%d\n",
  (void *)&A[3]+sizeof(int)-(void *)&A[0]);
```

- **(void *) serve per effettuare il calcolo tra locazioni di memoria e non semplicemente tra due numeri che indicano la locazione finale (x[99]) e quella iniziale (x[0])**
- **1013 + 4 - 1001 = 16 byte**

41

Errori comuni

- **I vettori non possono essere assegnati a vettori!**

```
int x[5];
int y[5];
x=y ;    // NON E' CORRETTA !!
```

- **Si copia un vettore in un altro con un `for`**
- **Il C non effettua nessun controllo sull'effettiva esistenza della componente indirizzata.**
- **Esempio**

```
int x[5];
x[12] = 9; //passa senza errori di compilazione
```

Il 9 verrà memorizzato nella tredicesima locazione intera a partire dall'inizio dell'array, anche se questa cella NON appartiene all'array!

42

Matrici

- **Una matrice in C è un array bidimensionale**
- **Dichiarazione di una matrice:**
  - il tipo di elemento della matrice (per esempio int)
  - spazio
  - il nome della matrice (ad esempio M)
  - doppie parentesi quadre contenenti il numero di righe e colonne della matrice
- **Esempio:**
  - `int M[10][8];  //matrice con 10 righe //e 8 colonne(80 interi)`

Esempio gestione delle matrici

- **Problema: dichiarare una matrice con 6 righe e 4 colonne e inizializzare tutti i suoi elementi a 0.**

```
int main()
{
   int  matrix[6][4];
   int i,j;
   for  (i=0; i<6; i++)
   {
     for  (j=0; j<4;   j++)
     {
       matrix[i][j]= 0 ;
     }
   }
}
```

Righe e colonne di una matrice

	Secondo indice (j)			
Primo indice (i)	0 , 0	0 , 1	0 , 2	0 , 3
	1 , 0	1 , 1	1 , 2	1 , 3
	2 , 0	2 ,1	2 , 2	2 , 3
	3 , 0	3 , 1	3 ,2	3 , 3
	4 , 0	4 , 1	4 ,2	4 , 3
	5 , 0	5 , 1	5 ,2	5 , 3

- **La matrice è memorizzata diversamente in memoria:**
  - gli elementi della matrice vengono memorizzati sequenzialmente.

45

Inizializzazione di una matrice

- **L'inizializzazione di una matrice può essere effettuata nel seguente modo:**

```
int matrix[3][3]={{1,2,4},{8,16,32},{64,128,256}};
```

- **Oppure con due cicli `for`:**

```
int matrix[3][4];
int i,j;
for  (i=0; i<3; i++)
  for  (j=0; j<4; j++)
    matrix[i][j]= 0 ;
```

46

Errori comuni

- **Le matrici non possono essere assegnate ad altre matrici!**

```
int M1[5][5];
int M2[5][5];
…
M1 = M2 ;  // NON E' CORRETTA !!
```

- **Il C non effettua nessun controllo sull'effettiva esistenza dell'elemento indirizzato:**

```
x[12][3] = 9;
```

passa senza errori la compilazione anche se la matrice ha solo 10 righe!

47

Esempio (1)

- **Problema:**
  - Costruire una matrice A che contiene in ogni cella A[i][j] il prodotto dei suoi indici (i*j)
  - Stampare tutti gli elementi di A

48

Esempio (2)

```
#include<stdio.h>
int main()
{
   int  A[5][4]; // dichiarazione della matrice
   int  i , j ; // dichiarazione indici di riga e di colonna
   for (i=0; i<5; i++)
   {
     for (j=0; j<4; j++)
      {
       A[i][j] = i * j ;
     }
   }
   for (i=0; i<5; i++)
   {
     for (j=0; j<4; j++)
     {
       printf ("%3d ", A[i][j]) ;
     }
     printf ("\n") ; //a capo, ad ogni riga della matrice
   }
   getchar();
   return 0;
}
```

49

Stringhe

- **Le stringhe in C sono array monodimensionali di caratteri**
- **Una stringa termina SEMPRE con il carattere terminatore di stringa:**
  - '\0' (una cella tutta a zero).
- **Non si considera il terminatore come facente parte della stringa:**
  - non lo si conteggiare fra i caratteri
  - ma DEVE essere presente.

50

Dichiarazione di stringhe

- **Dichiarazione di una stringa:**
  - il tipo char (obbligatorio, una stringa contiene solo char)
  - spazio
  - il nome della stringa (ad esempio s)
  - parentesi quadre contenti il numero di elementi della stringa
- **Esempio:**

```
char s[10];
```


Costanti stringhe

- **Se dobbiamo assegnare stringhe più lunghe esiste un metodo più pratico.**
- **Immaginiamo di volere inizializzare una stringa s con la valore della seguente stringa di caratteri**
  - "Laboratorio di programmazione per la sicurezza".
- **Il codice per farlo è il seguente:**

```
char s[]="Laboratorio di programmazione per la sicurezza";
```

- **In C una costante stringa si racchiude tra virgolette doppie, come "Crema".**
- **Il terminatore di stringa viene aggiunto automaticamente dal compilatore.**

Caratteri e stringhe

- **Le stringhe sono diverse dai caratteri:**
  - ‘m’ è un carattere
  - “m” è la stringa che contiene DUE caratteri: ‘m’ e ‘\0’!
- **Questo errore produce malfunzionamenti difficili da trovare in programmi complessi!**

Come si accede agli elementi della stringa?(1)

- **Immaginiamo di dichiarare la stringa s:**
  - `char s[]="mamma";`
- **Dal punto di vista del programma la stringa conta cinque caratteri**
  - 'm'+'a'+'m'+'m'+'a'
- **Dal punto di vista della memoria sono stati allocate sei celle per poter fare spazio anche al terminatore**
  - 'm'+'a'+'m'+'m'+'a'+'\0'

ma nei nostri programmi di questo fatto non ce ne accorgiamo

Come si accede agli elementi della stringa? (2)

- **Ogni carattere allocato per la stringa occupa un byte in memoria**
- **La stringa è array di caratteri quindi**
  - accediamo ad essa come in un array
- **Esempio**

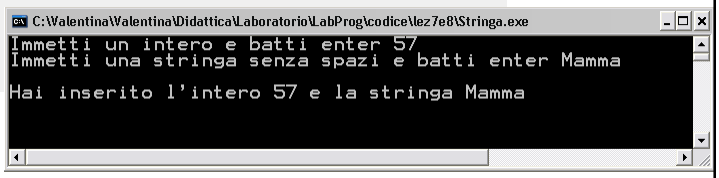
```
int  v[5];
char s[]="mamma";
…
v[0]= 1  ; // assegno 1 alla prima cella di v
s[0]= 'x'; // assegno il carattere 'x' alla prima
           // cella di s, non uso "x"!
```

55

Input/output con le stringhe (1)

- **Una stringa viene acquisita mediante**
  - **"%s"** senza & prima della variabile: il nome di un array è già l'indirizzo del primo elemento!

```
#include<stdio.h>
int main()
{
   int  conto;
   char s[128];

   printf("Immetti un intero e batti enter ");
   scanf ( "%d" , &conto );

   printf("Immetti una stringa senza spazi e batti enter ");
   scanf ( "%s" , s ); // senza &!!!!!

   printf("\nHai inserito l'intero %d e la stringa %s",conto,s);

   fflush(stdin);
   getchar();
   return 0;
}
```



Input/output con le stringhe (2)

- **La scanf() memorizza correttamente anche il terminatore di stringa '\0'**
- **Non è noto a priori il numero di caratteri che l'utente immetterà da tastiera**
  - occorre dichiarare una stringa abbastanza grande per soddisfare le nostre esigenze
- **Se ad esempio compiliamo:**

```
char s[5];   // allocato spazio per 5 caratteri
printf("Immetti una stringa senza spazi e batti enter");
scanf ( "%s" , s );
```

e l'utente immette "mamma" abbiamo:

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]
'm'	'a'	'm'	'm'	'a'	'\0'

57

Input/output con le stringhe (2)

- **Se l'utente immette "zio" la situazione in memoria è:**

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]
'z'	'i'	'o'	'\0'	??	??

- **Le ultime due celle rimarranno allocate ma non vi sarà nessun valore corretto memorizzato**
- **Più precisamente, dei numeri vi sono sicuramente memorizzati**
- **Naturalmente non hanno alcun senso per il nostro programma**

58

Input/output con le stringhe (3)

- **Se l'utente immette "unastringalunghissima" la situazione in memoria è la seguente:**

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]
'u'	'n'	'a'	's'	't'	'r'

- **La stringa**
  - viene memorizzata nello spazio che abbiamo riservato
  - e poi scrive su posizioni di memoria adiacenti!

59

Input/output con le stringhe (4)

- **Se l'utente immette "uso gli spazi" la situazione in memoria è la seguente:**

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]
'u'	's'	'o'	\0'	??	??

- **La memorizzazione dei caratteri immessi da tastiera si ferma se si incontra uno spazio.**
- **Dove rimangono i caratteri non memorizzati?**
  - Nel buffer di tastiera.

60

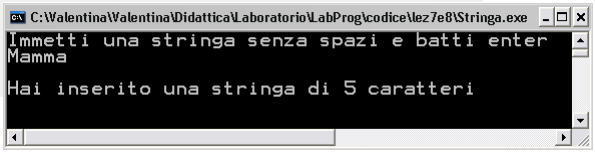
Conteggio dei caratteri e stampa delle stringhe

- **Se non sappiamo a priori quanti caratteri immetterà da tastiera l'utente**
  - come possiamo contare i caratteri immessi?

```c
#include <stdio.h>
int main()
{
   char s[128];
   int lung;

   printf("Immetti una stringa senza spazi e batti enter\n");
   scanf ( "%s" , s ); // senza &!!!!!

   lung = strlen(s);

   printf("\nHai inserito una stringa di %d caratteri", lung);

   fflush(stdin);
   getchar();
   return 0;
}
```

Esercizio da fare a casa: contare il numero di caratteri di s senza usare la funzione strlen()

Numero di caratteri della stringa s (senza contare '\0')



61

Copia di una stringa

- **La libreria standard che contiene già molte funzioni per manipolare le stringhe**
- **Ad esempio, per copiare una stringa in un'altra usa strcpy()**

```c
#include <stdio.h>
main()
{
    char s[128];

    // strcpy(s_to,s_from) copia la stringa s_from in s_to
    // fino a quando trova un terminatore di stringa

    strcpy(s,"Laboratorio di Informatica applicata");

    printf("%s",s);
    getchar();
}
```

62

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 7 e 8

Esempio di programma: il calcolo degli interessi

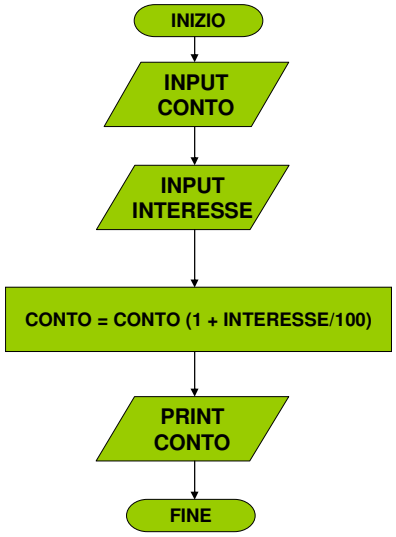
Obiettivo :

- Conoscere gli effetti sui risultati della scelta del tipo di dato delle variabili.

Oggetto del programma

- **Calcolare la capitalizzazione degli interessi su un conto corrente.**
- **Si supponga che l'interesse venga capitalizzato una sola volta all'anno.**
- **Sono utilizzate due variabili:**
  - **conto**;
  - **interesse**.

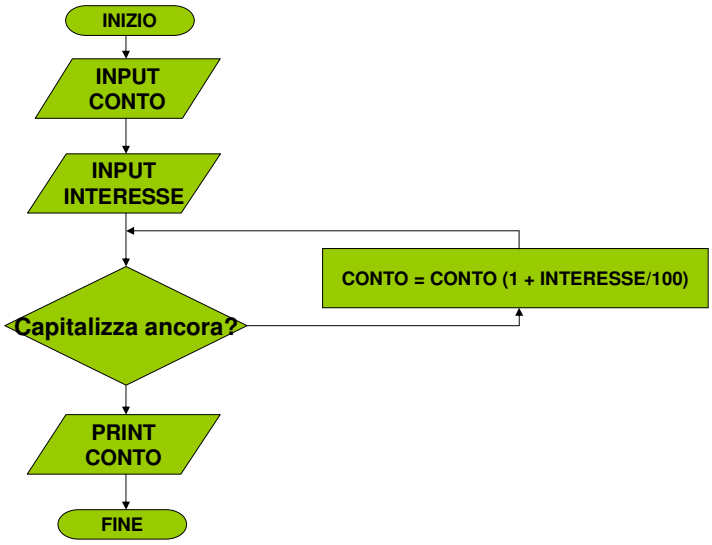
Capitalizzazione annuale



65

Fabio Scotti e Valentina Ciriani - Università degli Studi di Milano

Capitalizzazione periodica



66

Fabio Scotti e Valentina Ciriani - Università degli Studi di Milano

Programma (1)

```
// Capitalizzazione Annua fatta con interi

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int conto;
    int interesse;
```


Programma (2)

```
printf("\n Valore del conto all'inizio dell'anno: \n");
scanf( "%d", &conto );

printf("\n Inserire l'interesse della banca annuale: \n");
scanf( "%d", &interesse );

//calcolo il risultato in capitalizzazione annua
conto = conto * (1 + interesse/100 );

printf("\nValore del conto alla FINE dell'anno (CAP. ANNUA): %d \n", conto );
```

Analisi tipo di dato (1)

```
int conto;                  // 1000
int interesse;              //  10

...
conto = conto * (1 + interesse / 100 );
```

10/100

~~0.1~~

0

69

Analisi tipo di dato (2)

```
int conto;                  // 1000
int interesse;              //  10

...
conto = conto * (1 + (float) interesse / 100 );
```

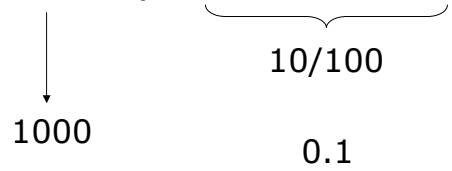
10/100

0.1

70

Analisi tipo di dato (3)

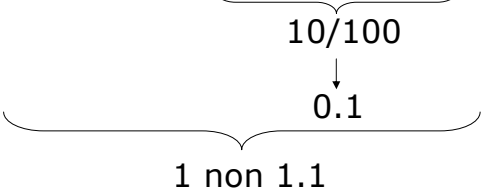
```
int conto;                  // 1000
float interesse;            //  10
…
scanf( "%d", &conto );
…
scanf( "%f", &interesse );

conto = conto * (1 + interesse/100 );
```



71

Analisi tipo di dato (4)

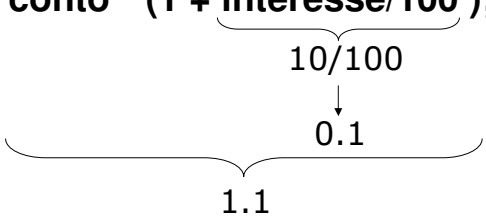
```
int conto;                  // 1
float interesse;            //  10
…
scanf( "%d", &conto );
…
scanf( "%f", &interesse );

conto = conto * (1 + interesse/100 );
```



72

Analisi tipo di dato (5)

```
float conto;                    // 1
float interesse;                //  10
...
scanf( "%f", &conto );
...
scanf( "%f", &interesse );

conto = conto * (1 + interesse/100 );
```



73

In sintesi …

- Occorre **progettare** il **tipo di dato** delle variabili dichiarate per non generare errori difficili da eliminare con il debugging.
- Ogni tipo di dato deve essere **letto e scritto correttamente** (printf , scanf con i corretti parametri **%...**).

74