POLITECNICO
DI TORINO
Dipartimento
di Automatica e Informatica

Il linguaggio C in breve

ASPETTI SINTATTICI, TIPI BASE E COSTRUTTI

Contenuti

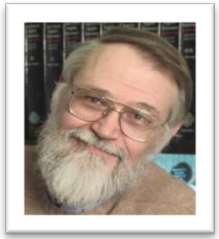
- Cenni storici
- Aspetti sintattici
  - **Come è organizzato un programma C**
- Tipi base e I/O
  - **tipi di dato primitivi (scalari),** costanti simboliche
  - **operazioni di I/O** (su stdin/stdout e su file testo)
- Costrutti di controllo
  - **costrutti condizionali e iterativi**
  - **funzioni** e passaggio parametri
- Dati aggregati
  - **vettori e matrici** (di interi, float e caratteri)
  - **stringhe** e vettori di stringhe
  - **strutture** (tipi aggregati)

Genesi del linguaggio C

- Sviluppato tra il 1969 ed il 1973 presso gli AT&T Bell Laboratories (Ken Thompson, B. Kernighan, Dennis Ritchie)
  - Per uso interno
  - Legato allo sviluppo del sistema operativo Unix
- Nel 1978 viene pubblicato “The C Programming Language”, prima specifica ufficiale del linguaggio
  - Detto “K&R”

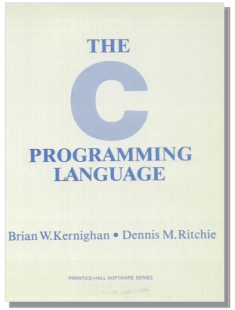
AT&T Bell Laboratories

Ken Thompson



Brian Kernighan

Dennis Ritchie



Storia

- Sviluppo
  - 1969-1973
  - Ken Thompson e Dennis Ritchie
  - AT&T Bell Labs
- Versioni del C e Standard
  - K&R (1978)
  - **C89 (ANSI X3.159:1989)**
  - C90 (ISO/IEC 9899:1990)
  - C99 (ANSI/ISO/IEC 9899:1999, INCITS/ISO/IEC 9899:1999)
  - C11
  - C17m C20
- Non tutti i compilatori sono standard!
  - GCC: *Quasi* C99, con alcune mancanze ed estensioni
  - Borland & Microsoft: *Abbastanza* C89/C90
  - CLion: supporta GCC (C99)

Diffusione attuale

- Il linguaggio C è tradizionalmente uno dei linguaggi più diffusi
- La sintassi del linguaggio C è ripresa da tutti gli altri linguaggi principali
- La sintassi dei principali linguaggi di programmazione (compresi Python e Java) è derivata dal C
- E' il linguaggio di elezione dei sistemi embedded
- Un ingegnere informatico non può non conoscere il C…

Un esempio

```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");

    return 0;
}
```

Caratteristiche generali del linguaggio C

- Il C è un linguaggio:
  - Imperativo ad alto livello
    - ... ma anche poco astratto (cioè si avvicina molto all'Hardware)
  - Strutturato
    - ... ma con eccezioni
  - Tipizzato
    - Ogni oggetto ha un tipo
  - Elementare
    - Poche *keyword*
  - *Case sensitive*
    - Maiuscolo diverso da minuscolo negli identificatori!
  - Portabile (con eccezioni)
  - Standard ANSI

Python e C

PYTHON

1. Linguaggio di alto livello, molto flessibile, privilegia comprensione/leggibilità a efficienza
2. Programmi interpretati
3. Blocchi basati su righe e spaziatura
4. Tipo di un dato (variabile, valore di ritorno di funzione) implicitamente dichiarato
5. La funzione main è una “buona norma”

C

1. Linguaggio molto meno versatile, di più basso (più vicino all’HW) livello, sacrifica aspetti di chiarezza all’efficienza (è di fatto il linguaggio dei sistemi operativi e dei sistemi embedded)
2. Programmi compilati
3. Blocchi basati su parentesi graffe e su parole chiave
4. Necessaria dichiarazione esplicita “sempre” (variabili, parametri formali e valore ritornato, …)
5. La funzione main è “obbligatoria”

Primo esempio (python e C)

```
##
#  Demonstrate the print function

#  Print 7
print(3 + 4)

#  Print Hello World! on two lines
print("Hello")
print("World!")

#  Print multiple values
#  with a single print function call
print("My numbers are", 3 + 4, "and", 3 + 10)

#  Print messages with empty line in between
print("Goodbye")
print()
print("Hope to see you again")
```

```
//
//  Demonstrate the print function
#include <stdio.h>

int main (void) {

  printf("%d\n", 3 + 4); // Print 7

  //  Print Hello World! on two lines
  printf("Hello\n")
  printf("World!\n")

  //  Print multiple values
  //  with a single print function call
  printf("My numbers are %d and %d\n",
          3+4, 3+10); // can be on multiple lines

  //  Print messages with empty line in between
  print("Goodbye\n\nHope to see you again\n");
}
```

Secondo esempio (python e C)

```
##
# This program simulates an elevator panel
# that skips the 13th floor.
#

# Obtain floor number from the user as an integer
floor = int(input("Floor: "))

# Adjust floor if necessary.
if floor > 13 :
   actualFloor = floor - 1
else :
   actualFloor = floor

# Print the result.
print("The elevator will travel to the ”
      "actual floor", actualFloor)
```

```
/*
   This program simulates an elevator panel
   that skips the 13th floor. */
#include <stdio.h>

int main (void) {
  // declare/define variables
  int floor, actualfloor;

  // Obtain floor number from the user as an integer
  printf ("Floor: ”);
  scanf (“%d”, &floor);

  /* Adjust floor if necessary. */
  if (floor > 13)
    actualFloor = floor – 1;
  else
    actualFloor = floor;

  // Print the result.
  printf("The elevator will travel to the "
         "actual floor %d\n", actualFloor);
}
```

Secondo esempio (C: le righe non contano!)

```
/*
   This program simulates an elevator panel
   that skips the 13th floor. */
#include <stdio.h>

int main (void) {
  // declare/define variables
  int floor, actualfloor;

  // Obtain floor number from the user as an integer
  printf ("Floor: ”);
  scanf (“%d”, &floor);

  /* Adjust floor if necessary. */
  if (floor > 13)
    actualFloor = floor – 1;
  else
    actualFloor = floor;

  // Print the result.
  printf("The elevator will travel to the "
         "actual floor %d\n", actualFloor);
}
```

```
/*
   This program simulates an elevator panel
   that skips the 13th floor. */
#include <stdio.h>

int main (void) {
  // declare/define variables
  int floor,
      actualfloor;

  // Obtain floor number from the user as an integer
  printf ("Floor: ”); scanf (“%d”, &floor);

  /* Adjust floor if necessary. */
  if (floor > 13) actualFloor = floor – 1;
  else actualFloor = floor;

  // Print the result.
  printf("The elevator will travel to the ”);
  printf("actual floor %d\n", actualFloor);
}
```

Terzo esempio: stampa quadrato di *

```
#include <stdio.h>
int main (int argc, char* argv[])
{
  int n, i, j;

  n = 0;
  printf("Inserire un intero >= a 2: ");
  scanf("%d", &n);
  if (n < 2){
    printf("Errore: valore < 2\n");
    return -1;
  }

  for(i=0; i<n; i++)
    printf("*");
  printf("\n");

  for(i=2; i<n; i++){
    printf("*");
    for(j=2; j<n; j++)
      printf(" ");
    printf("*\n");
  }

  for(i=0; i<n; i++)
    printf("*");
  printf("\n");

  return 0;
}
```

Esecuzione

```
Insert an integer >= 2: 1
Error: value < 2
```

```
Insert an integer >= 2: 4
****
*  *
*  *
****
```

Quarto Esempio: verifica di ordinamento

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

const int MAXC=50;

int verificaOrdine (FILE *fp);

int main(void) {
  char nomein[MAXC+1];
  FILE *fin;

  printf("nome file in ingresso: ");
  scanf("%s", nomein);
  fin=fopen(nomein,"r");

  if (verificaOrdine(fin)==1)
    printf("Il file %s e' in ordine\n", nomein);
  else
    printf("Il file %s non e' in ordine\n",
           nomein);
  fclose(fin);
}
```

```
int verificaOrdine (FILE *fp) {
  char riga0[MAXC+1], riga1[MAXC+1];

  fgets(riga0,MAXC,fp);

  while (fgets(riga1,MAXC,fp)!=NULL) {
    if (strcmp(riga1,riga0)<0)
        return 0;
    strcpy(riga0,riga1);
  }
  return 1;
}
```

Elementi sintattici del linguaggio C

- **Parole riservate** (*keyword*)
  - Es. break, if, for, int, float
- **Identificatori**
  - liberi, predefiniti
- **Costanti letterali**
  - Numeri, caratteri, stringhe
- **Caratteri speciali** (simboli: *parentesi, segni di punteggiatura, operatori*)
  - Es. {} () [] , ; . ->

Esempio di programma C

```c
/* simple C program */
#include <stdio.h>
int main(void)
{
    int max, A, B;

    scanf("%d%d",&A,&B);
    if (A >= B)
        max = A;
    else
        max = B;
    printf("%d\n",max);
    return 0;
}
```

/* commenti */
parole chiave
identificatori liberi
predefiniti
costanti letterali
caratteri speciali

Parole riservate (keyword)

Vocaboli “riservati” per scopi sintattici e/o semantici

- Non possono essere usate per altri scopi
- Costituiscono i “mattoni” della sintassi del linguaggio

Nel C standard sono 32

`auto`	`double`	`int`	`struct`
`break`	`else`	`long`	`switch`
`case`	`enum`	`register`	`typedef`
`char`	`extern`	`return`	`union`
`const`	`float`	`short`	`unsigned`
`continue`	`for`	`signed`	`void`
`default`	`goto`	`sizeof`	`volatile`
`do`	`if`	`static`	`while`

Identificatori

- Si riferiscono ad una delle seguenti entità:
  - Costanti
  - Variabili
  - Tipi
  - Sottoprogrammi (funzioni)
- Regole:
  - Iniziano con carattere alfabetico o “_”
  - Contengono caratteri alfanumerici o “_”
  - Case sensitive
    (maiuscole e minuscole contano come caratteri diversi)

Costanti letterali e caratteri speciali (simboli)

- Costanti letterali
  - Servono a rappresentare valori numerici, caratteri e stringhe:
    - numeri interi: 10 -72 025 0x24
    - numeri reali: 3.14159 1.7E+12
    - caratteri: 'H' ';' '\0' '\n'
    - stringhe: "Hello World!\n"
- Caratteri speciali (simboli)
  - Servono a
    - raggruppare o separare parti del programma (es. parentesi, virgola, punto-e-virgola)
    - Indicare operazioni in espressioni (es. >, <, >=, <=, +, -, *, /, &, ++, --, …)

Commenti

- Testo libero inserito all'interno del programma
- Non viene considerato dal compilatore
- Serve al programmatore, non al sistema!
- Formato:
  - Racchiuso tra /* */
  - Non è possibile annidare un commento dentro a un altro
  - Da // fino alla fine della linea (originariamente C++, anche in C dal C99)
- Esempi:
  - /* Questo è un commento ! */
  - /* Questo /* risulterà in un */ errore */
  - // Questo è un altro commento

Struttura di un programma C

Un programma C è una sequenza di:

- Direttive al precompilatore
- Funzioni (tra cui main)
- Dati
  - variabili
  - Costanti
  - Espressioni
- Istruzioni
  - Dichiarative
  - Operative
- Commenti

Struttura di semplice programma C

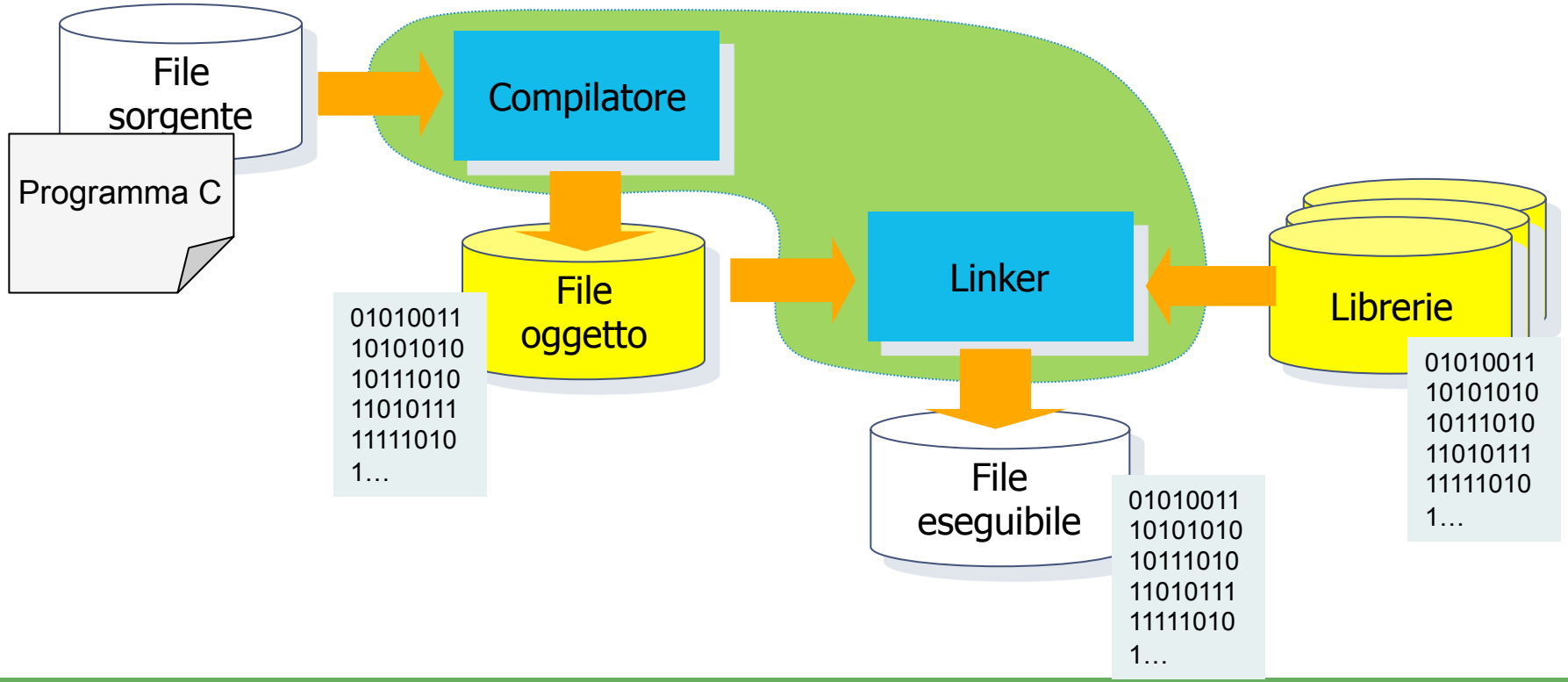
```
<direttive al precompilatore>
<eventuali dichiarazioni globali>

int main (void) {
  <istruzioni dichiarative>
  <istruzioni operative>
}
```

```
#include <stdio.h>
/* mancano dichiarazini globali */
int main(void) {
   int max, A, B;
   scanf("%d%d",&A,&B);
   if (A >= B)
     max = A;
   else
     max = B;
   printf("%d\n",max);
}
```


22

Traduzione di un programma



File oggetto e librerie

- I file oggetto non contengono comandi (o istruzioni) per interfacciarsi all'hardware e al sistema operativo
- Tali comandi sono contenuti nelle librerie di sistema
  - Contengono parti di uso comune in formato già compilato
- Il linker collega queste due informazioni per creare un file (eseguibile) che è completo delle informazioni per l'esecuzione

I dati

- Tipi
- Dichiarazione di variabili e costanti
- Assegnazione
- Espressioni
- Cast

Dichiarazione di dati

- **In C, tutti i dati devono essere dichiarati prima di essere utilizzati!**
- La dichiarazione (definizione) di un dato consiste in:
  - L'**allocazione** di uno spazio in memoria atto a contenere (collocarvi) il dato
  - L'**attribuzione** di un nome a tale spazio in memoria
- In particolare, occorre specificare:
  - **Nome** (identificatore)
  - **Tipo**
  - **Modalità di accesso** (variabile o costante)

Tipi di dato primitivi(scalari)

- Sono quelli forniti direttamente dal C
- Identificati da parole chiave!
  - char caratteri ASCII
  - int interi (complemento a 2)
  - float reali (floating point singola precisione)
  - Nel C99: _Bool booleano (vero o falso)
- La dimensione precisa di questi tipi dipende dall'architettura (non definita dal linguaggio)
  - |char| = 8 bit = 1 Byte sempre

Tipi di dato primitivi (scalari)

- Tipi:
  - Base: int, char, float
  - Varianti: (unsigned/signed, short, long, double)
- Costanti (valori):
  - letterali:
    - 10 -72 0250x24
    - 3.14159 1.7E+12
    - 'H' ';' '\0' '\n' "Ciao Mondo!\n"
  - Simboliche (identificatore associato a valore):

```
#define N 100
#define PIGRECO 3.14159
#define cancelletto '#‘
```

```
const int N = 100;
const float PIGRECO = 3.14;
const char cancelletto ='#';
```

Modificatori dei tipi base

Sono previsti dei modificatori, identificati da parole chiave da premettere ai tipi base

- Segno: signed/unsigned
  - Applicabili ai tipi char e int
  - signed: Valore numerico con segno
  - unsigned: Valore numerico senza segno
- Dimensione: short/long
  - Applicabili al tipo int
  - Utilizzabili anche senza specificare int
- Nel C99: Numeri complessi / parte immaginaria:
  - _Complex
  - _Imaginary

Varianti dei tipi primitivi

- Interi
  - [signed/unsigned] short [int]
  - [signed/unsigned] int
  - [signed/unsigned] long [int]
  - [signed/unsigned] long long [int] (nel C99)

- Reali
  - float
  - double
  - long double (nel C99)
  - float _Complex (nel C99)
  - double _Complex (nel C99)
  - long double _Complex (nel C99)

Dichiarazione di Variabili

- Esempi:
  - int x;
  - char ch;
  - long int x1, x2, x3;
  - double pi;
  - short stipendio;
  - long y, z;
- Usiamo nomi significativi!
  - Esempi:
    - int x0a11; /* NO */
    - int valore; /* SI */
    - float raggio; /* SI */

Dichiarazione di Costanti (simboliche)

- Identificatore associato a valore (non modificabile)
- Due possibilità
  - Variabile “non modificabile”
    - const <tipo> <costante> = <valore> ;
  - Esempi:
    - const double PIGRECO = 3.14159;
    - const char SEPARATORE = ‘$’;
    - const float ALIQUOTA = 0.22;
- Direttiva al precompilatore (#define), che associa un identificatore a una costante letterale (stile più “vecchio”, meno rigoroso, ma ancora molto usato)
  - Esempi:
    - #define PIGRECO 3.14159
    - #define SEPARATORE ‘$’
    - #define ALIQUOTA 0.22

Costanti (esempi)

- Esempi di valori attribuibili ad una costante:
  - Costanti di tipo char:
    - ‘f’
  - Costanti di tipo int, short, long
    - 26
    - 0x1a,0X1a
    - 26L
    - 26u
    - 26UL
  - Costanti di tipo float, double
    - -212.6
    - -2.126e2, -2.126E2, -212.6f

Costanti carattere speciali

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

Visibilità delle variabili

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

Struttura a blocchi

- In C, è possibile raccogliere istruzioni in blocchi racchiudendole tra parentesi graffe
- Significato: Delimitazione di un ambiente di visibilità di "dati" (variabili, costanti)
- Corrispondente ad una "sequenza" di istruzioni
- Esempio:

```
{
   int a=2;
   int b;
   …
}
```

`a` e `b` sono definite solo all'interno del blocco!

Visibilità delle variabili: Esempio

```
/* questo programma è parziale, si vedono solo
   istruzioni dichiarative, mancano le operative */
int n;
double x;
int main()
{
   int a,b,c;
   double y;
   {
      int d;
      double z;
   }
}
```

Assegnazione

- **Significato:** assegnare = attribuire un valore a una variabile.
- **Utilizzo** tipico: modificare il valore di una variabile
- **Sintassi:** `<identificatore> = <valore>;`
  - parte sinistra: variabile da modificare,
  - il carattere uguale (=): indica assegnazione
  - parte destra: espressione che genera il valore da assegnare.
- **Tipo:** il valore deve essere ***compatibile*** con il tipo della variabile.
  - Non è possibile, ad esempio, assegnare a una variabile di tipo `char` un valore di tipo `float`.
  - Ma è possibile, ad esempio, assegnare a una variabile di tipo `float` un valore `int` (o viceversa), a patto di conoscere le regole di conversione (cast)

Assegnazione

- Il valore può essere:
  - una costante (letterale o simbolica)
  - una variabile
  - un'espressione con costanti, variabili, operatori ed eventuali chiamate a funzione.
- Esempi
  - `a = 42;`
  - `c = NUM;`
  - `b = a;`
  - `d = a + b * divisione (a, NUM) + 5;`

Espressioni

- **Significato:** formule che rappresentano valutazioni, in genere di tipo aritmetico o logico, con operandi e operatori
- **Operandi** elementari delle espressioni:
  - costanti (letterali e/o simboliche)
  - variabili, di cui si usa il valore associato (memorizzato, al momento della valutazione dell'espressione).
  - Chiamate di funzione
- **Valutazione e risultato:**
  - Se un'espressione contiene solo valori costanti, il risultato è unico ed indipendente dal momento della valutazione,
  - Se sono presenti variabili, il rusultato dipende dai dati presenti nelle variabili al momento dell'esecuzione.

Espressioni (esempi)

```
5 – 10
3 * 3.14
a – 10
b * 3.14
(x + 5) * (x - y)
2*(a + b) - (a*a - b*b)
(2 * PIGRECO * r)

10 * 20  + 1 // = 200 + 1 =  201
10 * (20 + 1) // = 10 * 21 =  210
```

Cast (problema)

- espressioni e assegnazioni ***richiedono di regola*** variabili e costanti di tipo compatibile:
  - un'operazione aritmetica richiede numeri dello stesso tipo (es. tutti `int` o tutti `float`)
  - un'assegnazione vuole, nella parte destra, un'espressione, variabile o costante dello stesso tipo della variabile a sinistra.
- Esempio: date due variabili x e y di tipo **`float`**, **sarebbero (attenzione: SAREBBERO)** scorrette le istruzioni:
  - `x = 1;`
  - `y = x * 2;`

  Perchè le costanti utilizzate non sono di tipo float} ma int.

Cast (problema)

- espressioni e ass… …i di tipo compatib…
  - un'operazione ar… …o tutti `float`)
  - un'assegnazione … costante dello stesso tipo … a sinistra.

```
Andrebbero riscritte così:
x = 1.0;
y = x * 2.0;
```

- Esempio: date due … x e y di tipo **`float`**, **sarebbero (attenzione: SAR…BERO)** scorrette le istruzioni:
  - `x = 1;`
  - `y = x * 2;`

  Perchè le costanti utilizzate non sono di tipo float} ma int.

Cast (soluzione)

- La compatibilitò stretta tra variabili e/o costanti è
  - formalmente ineccepibile,
  - in pratica troppo vincolante.
- Si accetta quindi un COMPROMESSO tra rigore e praticità
  - CAST: operazione di conversione di tipo
  - genera un valore di un tipo, a partire da un dato di un altro tipo
- implicito: applicato in modo automatico
- esplicito: consiste nel premettere a un dato (variabile o espressione) il tipo, tra parentesi tonde, a cui lo si vuole convertire

Cast (regole)

- gerarchia tra i tipi di dato, basata su con regole di corrispondenza e inclusione:
  - Es. {interi} $\subset$ {reali}, {char come interi} $\subset$ {interi}
  - Promozione: tra due dati, il meno generale viene trasformato nel pù generale (es. un int viene convertito nel float corrispondente)
- CAST implicito: promozione ogni volta che possibile
  - Es. operazione aritmetica tra int e float: l'int viene convertito a float, prima di eseguire l'operazione: 6/1.2 diventa 6.0/1.2
- CAST esplicito: si può sia promuovere che andare in senso opposto , ma perdendo informazione
  - Es. da float a int si tronca: (int) 7.8 diventa 7

Cast (esempi)

```
/* esempi di cast IMPLICITO */
  float x, y;
  ...
  x = 10; // trasformato in  x = 10.0;
  y = x * (2/3.0); // trasformato in y = x * (2.0/3.0);
  ...
  int a_ascii, n;
  ...
  /* Alla variabile a_ascii} viene assegnato il numero 97 (codice ASCII di 'a'
     considerato come intero), alla variabile n viene assegnato 26, differenza tra
     i codici di 'z' e 'a' (visti come interi)
  a_ascii = 'a';
  n = 'z' - 'a' + 1;
```

Cast (esempi 2)

```
/* A x viene assegnato il valore 1.75 ma l'assegnazione i = x
  produce una conversione a intero con troncamento: i riceve il valore 1.
  float x; int i;
  ...
  x = 7.0/4;
  i = x;

/* cast espliciti */
  float x, y, z, t;
  ...
  x = 10.5;
  y = (float)((int)x * (3/2)); // y = 10.0
  w = x * (float)(3/2); // w = 10.5
  z = x * (float)3/(float)2;
```

Riassumendo

Definizione/dichiarazione:

```
int numero;
char a, b, c;
float num_reale;
```

Espressioni:

```
5 - 10
3 * 3.14
a - 10
b * 3.14
(x + 5) * (x - y)
2*(a + b) - (a*a - b*b)
(2 * PIGRECO * r)
```

Riassumendo

Assegnazione:

```
tmp = a;  // assegnazioni standard
a = b;
b = a;
// assegnazioni con cast esplicito
```

Inizializzazione:

```
int a = 1, b = 2;
// definizione con inizializzazione
```

Cast (conversione di tipo):

```
x = 10.5;
y = (float)((int)x * (3/2));  // y = 10.0
w = x * (float)(3/2);  // w = 10.5
z = x * (float)3/(float)2;
```

Costrutti di controllo

- costrutti condizionali
  - if (con o senza parte else)
  - switch-case
    - selettore multiplo, ma solo per valori interi o char
- costrutti iterativi
  - while, do-while
    - consigliati per condizioni di controllo generali (espressioni logiche)
  - for
    - consigliato per iterazioni numerate (con conteggio)

Costrutti condizionali

- Espressioni logiche:
  - condizioni per abilitare/disabilitare o selezionare il blocco da eseguire
- if

```
if (condizione) {
  // istruzioni caso VERO
}
else {
  // istruzioni caso FALSO
}
```

- Condizioni multiple e costrutti if annidati

Switch: selezione multipla

```
switch (selettore) {
  case 0: printf("Scelta n. 0\n"); break;
  case 1: printf("Scelta n. 1\n"); break;
   ...
  default: printf("Scelta non valida\n");}
}
```

Costrutti iterativi

```
while           while (condizione_per_continuare) {
                    ...
                }

do … while      do {
                  printf("scrivi numero positivo");
                  scanf("%d", &x);
                } while (x<=0);

for             for(i = 0; i < 10; i++) {
                  printf("Inserisci intero:  ");
                  scanf("%d", &vet[i]);
                }
```

Input Output

- IO su file testo (di caratteri). Esiste I/O di dati binari (fread(fwrite), ma NON VIENE SVOLTO.
  - stdin, stdout, stderr (automaticamente aperti, sempre)
  - fopen/fclose per altri file
- operazioni di I/O e file testo
  - formattato: (f)printf e (f)scanf (%d, %c, %f, %s)
  - righe/stringhe: (f)gets, (f)puts ((f)printf ("%s"))
  - caratteri: (f)getc/getchar, (f)putc ((f)printf ("%c"))
- Output (più facile)
- Input
  - Semplice con formato fisso
  - Più complicato se formato più libero

Input/output (compresi file)

- Apertura/chiusura file:
  - FILE *fp;
  - fp=fopen("myfile.txt","r");
  - …
  - fclose(fp);
- Tipi di I/O:
  - formattato (include quasi totalmente gli altri):
    - fscanf, fprintf, scanf, printf, (sscanf)
  - stringhe:
    - fgets, fputs, gets, puts
  - caratteri:
    - fgetc, fputc, getchar, putchar

Input/output per caratteri (esempi)

```
// Esempi: uso di getc, fgetc, getchar
  char a, b, c;
  FILE *fp;
  fp=fopen("myfile.txt","r");
  ...
  // legge un carattere da file
  a = getc(fp);
  // equivale alla precedente cambia solo
  // l'implementazione interna
  b = fgetc(fp);
  // acquisisce un carattere da tastiera
  // equivale a c = getc(stdin);
  c = getchar();
  ...
  fclose(fp);
```

```
// Esempi: uso di putc, fputc, putchar
  char a = 'x', b = 'y', c = 'z';
  FILE *fp;
  ...
  fp=fopen("myfile.txt","w");
  // scrive un carattere su file
  putc(a, fp);
  // equivale alla precedente cambia solo
  // l'implementazione interna
  fputc(b, fp);
  // scrive un carattere su stdout (video) –
  // equivale a putc(c, stdout);
  putchar(c);
  ...
  fclose(fp);
```

Input/output per stringhe (esempi)

```
// Esempio di uso di gets

  char mystring[5];
  // legge da stdin (tastiera) a mystring
  gets(mystring);

// Esempio di uso di fgets

  char str[50];
  FILE *fp;
  fp=fopen("myfile.txt","r");
  // legge da file a str
  fgets(str,10,fp);
```

```
// Esempio di uso di puts

  char mystring[5]="ciao";
  // scrive su stdout la stringa in mystring
  puts(mystring);

// Esempio di uso di fputs

  FILE *fp;
  char mystring[5]="ciao";
  fp=fopen("myfile.txt","w");
  // l'output va nel file
  fputs (mystring,fp);
  fclose(fp);
```

Input/output per stringhe (esempi)

```
// Esempio di uso di gets

  char mystring[5];
  // legge da stdin (tastiera) a mystring
  gets(mystring);

// Esempio di uso di gets

  char str[50];
  FILE *fp;
  fp=fopen("myfile.txt","r");
  // legge da file a str
  fgets(str,10,fp);
```

```
// Esempio di uso di puts

  char mystring[5]="ciao";
  // scrive su stdout la stringa in mystring
  puts(mystring);
```

NOTA: gets legge tanti caratteri quanti ne vengono incontrati fino al raggiungimento di a-capo o EOF.
RISCHIA DI USCIRE DA mystring.
si aggiunge automaticamente \0 in fondo

Input/output per stringhe (esempi)

```
// Esempio di uso di gets

  char mystring[5];
  // legge da stdin (tastiera) a
  gets(mystring);

// Esempio di uso di fgets

  char str[50];
  FILE *fp;
  fp=fopen("myfile.txt","r");
  // legge da file a str
  fgets(str,10,fp);
```

Preleva al massimo 9 (10-1) caratteri dal file alla stringa str e termina la stringa con \0. Permette quindi di "proteggere" str nel caso di input di "troppi" caratteri.

```
FILE *fp;
char mystring[5]="ciao";
fp=fopen("myfile.txt","w");
// l'output va nel file
fputs (mystring,fp);
fclose(fp);
```

Input/output formattato

- Permette di specificare una stringa che
  - (output) deve essere inviata in output,

oppure

  - (input) corrispondere all'input.

- La stringa può includere delle direttive formato, che indicano come trattare i caratteri in input o output per rappresentare un dato numerico o testuale. Le direttive di formato principali sono:
  - %c per singoli caratteri,
  - %s per stringhe,
  - %d per numeri interi,
  - %f per il tipo float

Input/output formattato di testi

- L’I/O formattato comprende la possibilità di leggere o scrivere
  - singoli caratteri (direttiva di formato "%c")
  - stringhe (direttiva di formato "%s").
- C'è ridondanza nella scelta di IO per caratteri o stringhe. In pratica,
  - Singoli caratteri possono essere letti/scritti
    - con l’I/O formattato (direttiva "%c")
    - con le funzioni getc, fgetc, putc, fputc, getchar, putchar (che potrebbero quindi essere evitate).
  - Le stringhe possono essere lette/scritte
    - con la direttiva di formato "%s"
    - con le funzioni fgets, fputs, gets, puts.

Input/output formattato di testi

fputs e puts possono essere completamente sostituite da output formattato (fprintf, printf) con formato "%s"

- possibilità di leggere o scrivere
  - to "%c")
  -).
- O per caratteri o stringhe. In pratica,
  - caratteri possono essere letti/scritti
    - formattato (direttiva "%c")
    - con nzioni getc, fgetc, putc, fputc, getchar, putchar (che potrebbero quindi essere evitate).
  - Le stringhe possono essere lette/scritte
    - con la direttiva di formato "%s"
    - con le funzioni fgets, fputs, gets, puts.

Input/output formattato di testi

- possibilità di leggere o scrivere
  - to "%c")
  - ').
  - O
  - caratteri possono essere letti
    - formattato (direttiva "%c")
    - con nzioni getc, fgetc, putc, fputc, getcha
  - Le string he possono essere lette/scr
    - con la direttiva di formato "%s"
    - con le funzioni fgets, fputs, gets, puts.

fputs e puts possono essere completamente sostituite da output formattato (fprintf, printf) con formato "%s"

l'input formattato mediante fscanf e scanf, con formato "%s" non equivale completamente a fgets e gets in quanto l'input con %s utilizza gli spazi come separatori, mentre fgets e gets leggono intere righe di testo (compresi eventuali spazi in esse presenti).

Input/output formattato (esempi)

```
// Esempio: uso di scanf.
  int n;
  scanf("%d",&n); // legge da stdin un intero
// Esempio: uso di fscanf.
  FILE *fp;
  int n;
  fp=fopen("myfile.txt","r");
  ...
  fscanf(fp,"%d",&n); legge da file un intero

// Esempio: effetto degli spazi in lettura.
  char str1[50], str2[50]; FILE *fp;
  fp=fopen("persone.txt","r");
  fscanf(fp, "%s", str1);
  rewind(fp); // ritorna all'inizio del file
  fgets(str2, 50, fp); // rilegge in altro modo
```

```
// Esempio: uso di printf.
  int n=5;
  printf("%d",n); // stampa un intero
// Esempio: uso di fprintf.
  FILE *fp;
  int n=5;
  fp=fopen("myfile.txt","w");
  ...
  fprintf(fp,"%d",n); // scrive un intero su file
```

Input/output formattato (esempi)

```
// Esempio: uso di scanf.
  int n;
  scanf("%d",&n); // legge da stdin un intero
// Esempio: uso di fscanf.
  FILE *fp;
  int n;
  fp=fopen("myfile.txt","r");
  ...
  fscanf(fp,"%d",&n); legge da file un intero

// Esempio: effetto degli spazi in lettura.
  char str1[50], str2[50]; FILE *fp;
  fp=fopen("persone.txt","r");
  fscanf(fp, "%s", str1);
  rewind(fp); // ritorna all'inizio del file
  fgets(str2, 50, fp); // rilegge in altro modo
```

Se la prima riga del file persone.txt contiene la riga
questo e' un esempio di lettura

fscanf (str1, ...) legge solo questo mentre
fgets (str2, …) legge tutte la riga
questo e' un esempio di lettura

Input/output formattato

- Serve pratica (laboratorio, esercizi)
  - Attenzione a non perdere sincronizzazione tra input e formato
    - spazi, a-capo, errori
    - Mischiare input che leggono i singoli caratteri (%c, getc, fgetc) e/o intere stringhe (compreso a-capo) con input che saltano spazi/a-capo (non è un errore ma occorre cautela)

Test di fine file

- Costante EOF (di solito EOF = -1)
  - fscanf(…)==EOF
  - se file non finito, fscanf ritorna quanti campi % ha letto correttamente
  - getc/fgetc(…)==EOF
  - se file non finito, viene ritornato (come int) il codice (ASCII) di un carattere
- fgets(…)==NULL
  - se file non finito, fgets ritorna puntatore a stringa (destinazione dell'input)
- Funzione feof(): es. if (feof(fp)), while (!feof(fp))
  - ATTENZIONE: feof() vero solo DOPO aver tentato di leggere oltre end-of-file !!! (spesso si fa una lettura in più)

Test di fine file

- Costante EOF (di solito EOF = -1)
  - fscanf(…)==EOF
  - se file non fi… fscanf ritorna quanti campi % ha letto correttamente
  - getc/fgetc(…
  - … carattere
- fg…
  - s… …ell'input)
- Fu…
  - A… …end-of-file !!! (spesso si fa una lettura in più)

```
while (fscanf("%d%f",…)!=EOF) {
   ...
}
// oppure
while (fscanf("%d%f",…)==2) {
   ...
}
```

Test di fine file

- Costante EOF (di solito EOF = -1)
  - fscanf(…)==EOF
  - se file non finito, fscanf ritorna quanti campi % ha letto correttamente
  - getc/fgetc(…)==EOF
  - se file non finito, viene ritornato (come int) il codice (ASCII) di un carattere
- fgets(…)==NULL
  - se file non finito, fge puntatore a stringa (destinazione dell'input)
- Funzio
  - ATTEN
    lettur

```
while (fgets(fp,MAX,riga)!=NULL) {
   ...
}
```

Test di fine file

- Costante EOF (di solito EOF = -1)
  - f
  - s
  - g
  - s

```
while (!feof(fp)) {
   fscanf("%s",riga);
   printf("Ho letto: %s\n", riga);
}
```

- fgets(…)==NULL
  - se file non finito, fgets rito… …tore a stringa (destinazione dell'input)

- Funzione feof(): es. if (feof(fp)), while (!feof(fp))
  - ATTENZIONE: feof() vero solo DOPO aver tentato di leggere oltre end-of-file !!! (spesso si fa una lettura in più)

Test di fine file

- Costante EOF (di solito EOF = -1)
  - f
  - s
  - g
  - s

```
while (!feof(fp)) {
   fscanf("%s",riga);
   printf("Ho letto: %s\n", riga);
}
```

TENTA DI LEGGERE DOPO FINE FILE

- fgets(…)==NULL
  - se file non finito, fgets rito… …core a stringa (destinazione dell'input)

- Funzione feof(): es. if (eof(fp)), while (!feof(fp))
  - ATTENZIONE: feof() vero solo DOPO aver tentato di leggere oltre end-of-file !!! (spesso si fa una lettura in più)

Test di fine file

```
while (!feof(fp)) {
   fscanf(”%s”,riga); // prova
   // se ha letto fine file non usa riga
   if (!feof(fp))
     printf(”Ho letto: %s\n”, riga);
}
```

- fgets(…)=
  - se file non finito, fg … ntatore a stringa (destinazione dell’input)
- Funzione feof(): es. if (eof(fp)), while (!feof(fp))
  - ATTENZIONE: feof() vero solo DOPO aver tentato di leggere oltre end-of-file !!! (spesso si fa una lettura in più)

Funzioni

- Funzione come sotto-programma
  - scritto una sola volta, riutilizzato più volte
    - contenuto: elaborazioni su parametri e variabili locali, risultato con return
  - interfaccia
    - prototipo, chiamate
    - Passaggio di parametri
      - by value
      - by reference (in C NO: si realizza, in pratica, con puntatore by value)
- Regole simili al Python:
  - Interfaccia: parametri formali – parametri attuali (argomenti)
  - Corpo della funzione
    - parametri formali sono "copia" dei parametri attuali (variabili locali inizializzate con argomenti)
    - Eccezione: i vettori sono "messi in mezzo" (condivisi)

Esempio di funzione (python e C)

```
##
#  main function

def main() :
    result = cubeVolume(2)
    print("A cube with side length 2 has volume",
      result)

#  cubeVolume function

def cubeVolume(sideLength) :
    volume = sideLength ** 3
    return volume

main
```

```
//  cubeVolume function prototype
int cubeVolume(int sideLength);

// main function
int main(void) {
  int result;
  result = cubeVolume(2);
  printf(
    "A cube with side length 2 has volume %d\n",
  result);
}

// cubevolume function
int cubeVolume(int sideLength) {
  int volume = sideLength*sideLength*sideLength;
  return volume;
}
```

Esempio di funzione (python e C)

```
##
#  main function

def main() :
    result = cubeVolume(2)
    print("A cube with side length 2 has volume",
      result)

#  cubeVolume function

def cubeVolume(sideLength) :
    volume = sideLength ** 3
    return volume

main
```

```
//  cubeVolume function prototype
int cubeVolume(int sideLength);

// main function
int main(void) {
  int result;
  result = cubeVolume(2);
  printf(
    "A cube with side length 2 has volume %d\n",
  result);
}

// cubevolume function
int cubeVolume(int sideLength) {
  int volume = sideLength*sideLength*sideLength;
  return volume;
}
```

Interfaccia: parametri formali

Esempio di funzione (python e C)

```
##
#  main function

def main() :
    result = cubeVolume(2)
    print("A cube with side length 2 has volume",
      result)

#  cubeVolume function

def cubeVolume(sideLength) :
    volume = sideLength ** 3
    return volume

main
```

Corpo della funzione

```
//  cubeVolume function prototype
int cubeVolume(int sideLength);

// main function
int main(void) {
  int result;
  result = cubeVolume(2);
  printf(
    "A cube with side length 2 has volume %d\n",
  result);
}

// cubevolume function
int cubeVolume(int sideLength) {
  int volume = sideLength*sideLength*sideLength;
  return volume;
}
```

Esempio di funzione (python e C)

```
##
#  main function

def main() :
    result = cubeVolume(2)
    print("A cube with side length 2 has volume",
      result)

#  cubeVolume function

def cubeVolume(sideLength) :
    volume = sideLength ** 3
    return volume

main
```

Chiamata con argomenti (parametri attuali)

```
//  cubeVolume function prototype
int cubeVolume(int sideLength);

// main function
int main(void) {
  int result;
  result = cubeVolume(2);
  printf(
    "A cube with side length 2 has volume %d\n",
  result);
}

// cubevolume function
int cubeVolume(int sideLength) {
  int volume = sideLength*sideLength*sideLength;
  return volume;
}
```

Tipi aggregati

- vettori e matrici
  - Aggregati omogenei con indici
    - int v[100]; float M[10][10];
    - X = V[i]*M[j][k];
    - Dimensioni NOTE (costanti) -> spesso sovradimensionati e sotto-utilizzati.
- stringhe
  - vettori di caratteri “speciali”
  - manipolate mediante funzioni di libreria (strlen, strcmp, strcpy, …)
  - Purchè terminati con '\0‘ (terminatore di stringa)
- struct
  - Aggregati eterogenei (campi possono essere di tipo diverso)
  - Campi (di solito pochi) identificati da nomi (come fossero variabili locali alla struct)

Vettori (monodimensionali)

- Dati AGGREGATI, accesso mediante INDICI

```
int eta[20], altezza[20], i;
float etaMedia = 0.0;
i = 0;
for(i=0; i<20; i++) {
  scanf("%d %d", &eta[i], &altezza[i]);
  etaMedia += eta[i];
}
etaMedia = etaMedia/20;
```

Matrici (multidimensionali)

```
int matrice_diagonale[3][3] = { { 1, 0, 0 },
                                { 0, 1, 0 },
                                { 0, 0, 1 }  } ;
float M2 [N][M], V[N], Y[M];
for (r=0; r<N; r++) {
  Y[r] = 0.0;
  for (c=0; r<M; c++)
    Y[r] = Y[r] + M2[r][c]*V[c];
}
```

Stringhe

- NON sono un tipo nuovo, ma:
  - vettori di caratteri
  - terminati da ‘\0’
- Possono essere gestiti:
  - carattere per carattere (come vettori)
  - come dati unitari, mediante:
    - funzioni di I/O per stringhe: es. fgets, sscanf, fscanf/fprintf (con formato %s)
    - con funzioni di libreria (includendo <string.h>): es. strcmp, strlen, strcpy, strcat, ...

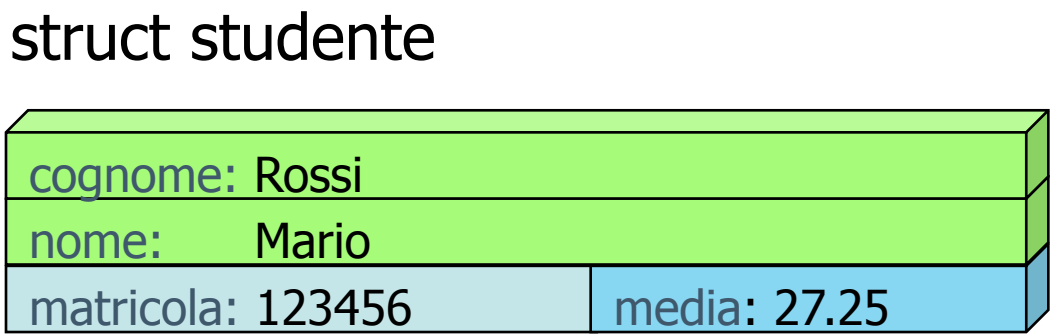
Struct

- Strutture (tipo struct)
  - tipo di dato aggregato
  - campo riferito mediante nome
  - differenze rispetto a vettori
- Es.

```
struct studente
{
  char cognome[MAX], nome[MAX];
  int matricola;
  float media;
};
```

Aggregato di dati eterogenei (struct)

- Più informazioni eterogenee possono essere unite come parti (campi) di uno stesso dato dato (aggregato)



I tipi struct

- Il dato aggregato in C è detto struct. In altri linguaggi si parla di record
- Una struct (struttura) è un dato costituito da campi:
  - i campi sono di tipi (base) noti (eventualmente altre struct)
  - ogni campo all'interno di una struct è accessibile mediante un identificatore (anziché un indice, come nei vettori)

```
struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
```

Nuovo tipo di dato

- Il nuovo tipo definito è `struct studente`
- La parola chiave `struct` è obligatoria

```
struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
```

Nuovo tipo di dato

Nome del tipo aggregato

- Stesse regole che valgono per i nomi delle variabili
- I nomi di `struct` devono essere diversi da nomi di altre struct (possono essere uguali a nomi di variabili)

```
struct studente
{
   char cognome[MAX], nome[MAX];
   int matricola;
   float media;
};
```

Campi (eterogenei)

Nuovo tipo di dato

Nome del tipo aggregato

- I campi corrispondono a variabili locali di una `struct`
- Ogni campo è quindi caratterizzato da un tipo (base) e da un identificatore (unico per la struttura)

Dal C ai programmi

- Costrutti e regole del linguaggio
  - Dati per scontati !!! (quasi)
- Programmare = "dal problema alla soluzione" (utilizzando il linguaggio C)
  - Strategia -> problem solving
  - Esperienza e abilità personale
  - Imparare da soluzioni proposte
  - NOVITA': Classificazione di problemi

Classi di problemi

	Senza vettori	**Con vettori/matrici**
Numerici	Equaz. 2° grado Serie e successioni numeriche …	Statistiche per gruppi Operazioni su insiemi di numeri Generazione numeri primi Somme/prodotti matriciali
Codifica	Conversioni di base (es. binario/decimale) Crittografia di testo …	Conversioni tra basi numeriche Ricodifica testi utilizzando tabelle di conversione
Elab. Testi	Manipolazione stringhe Menu con scelta Grafico di funzione (asse X verticale)	Conteggio caratteri in testo Grafico funzione (asse X orizzontale) Formattazione testo (centrare, eliminare spazi)
Verifica/ selezione	Verifica di ordinamento/congruenza di dati Verifica mosse di un gioco Filtro su elenco di dati Ricerca massimo o minimo Ordinamento parziale	Verifica di unicità (o ripetizione) di dati Selezione di dati in base a criterio di accettazione Ricerca di dato in tabella (in base a nome/stringa) Ordinamento per selezione