Il linguaggio C

Perché il linguaggio C?

- Larga diffusione nel software applicativo
- Standard di fatto per lo sviluppo di software di sistema
  - Visione a basso livello della memoria
  - Capacità di manipolare singoli bit

Perché il linguaggio C? (2)

- Standard di fatto per lo sviluppo di software di sistema (cont.)
  - possibilità di incapsulare parti di codice assembler che effettuano compiti impossibili da esprimere in linguaggi ad alto livello
    - es. esecuzione di istruzioni 'speciali' (TRAP), manipolazione diretta di registri
  - Compilazione estremamente efficiente
  - Occupazione di memoria ridotta

C : caratteristiche fondamentali

- Paradigma imperativo
  - costrutti di controllo simili a Java (`while-do, for, if (..) else, switch`)
  - tipi base molto simili a Java (`int, double` etc…)
- Ma … NON è un linguaggio ad oggetti
  - NON supporta la nozione di oggetti, classi o meccanismi di ereditarietà
  - la principale forma di strutturazione sono le funzioni
    - un programma C è una collezione di funzioni di cui una di nome `main`
    - le funzioni possono risiedere in uno o più file

C : caratteristiche fondamentali (2)

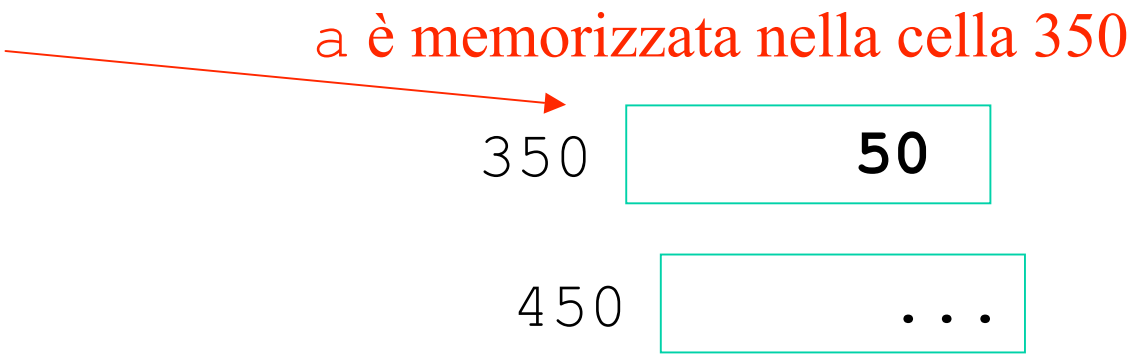
- Memoria e puntatori
  - in C è possibile *conoscere* e *denotare* l'indirizzo della cella di memoria in cui è memorizzata una variabile (il *puntatore*)
    - es :

```
int a = 50; /* una var intera */
int *   b; /* una var puntatore a interi */
   ...
b = &a; /* assegna a b l'indirizzo della
  cella in cui è memorizzata a */
```

C : caratteristiche fondamentali (3)

- Memoria e puntatori
  - in C è possibile *conoscere* e *denotare* l'indirizzo della cella di memoria in cui è memorizzata una variabile (il *puntatore*)
    - es :

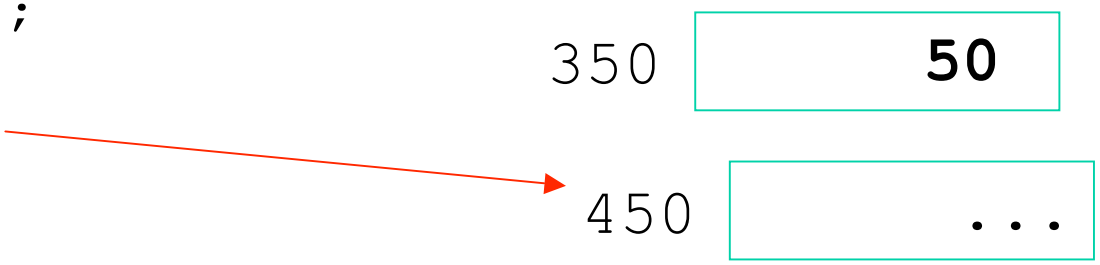
```
int a = 50;
int *b;
  …
b = &a;
```



C : caratteristiche fondamentali (4)

- Memoria e puntatori
  - in C è possibile *conoscere* e *denotare* l'indirizzo della cella di memoria in cui è memorizzata una variabile (il *puntatore*)
    - es :

```
int a = 50;
int *b;
  …
b = &a;
```

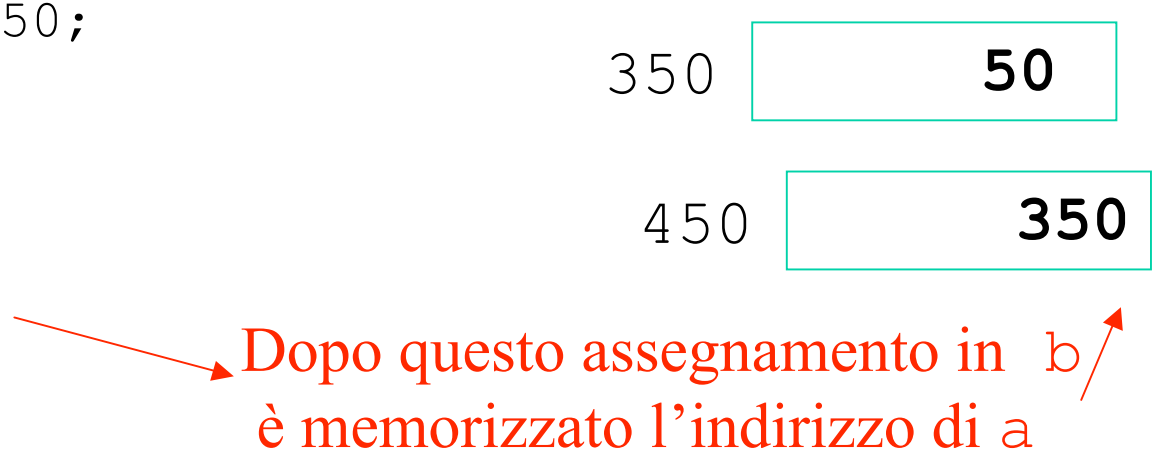


b è memorizzata nella cella 450

C : caratteristiche fondamentali (5)

- Memoria e puntatori
  - in C è possibile *conoscere* e *denotare* l'indirizzo della cella di memoria in cui è memorizzata una variabile (il *puntatore*)
    - es :

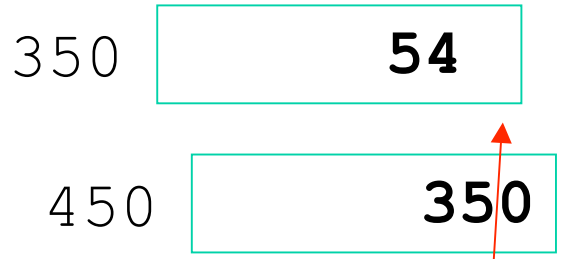
```
int a = 50;
int *b;
  …
b = &a;
```



C : caratteristiche fondamentali (6)

- Memoria e puntatori
  - è possibile *conoscere e/o modificare* il contenuto di una variabile manipolando direttamente il suo puntatore (operatore di *dereferenziazione* ‘ * ’)
    - es :

```
int a = 50;
int *b = &a;
  …
*b = *b + 4;
```



Dopo questo assegnamento in a è memorizzato il valore 50 + 4

C : un primo programma

- Il più semplice programma C è costituito da
  - una singola funzione (`main`)
  - memorizzata in un singolo file

C : un primo programma (2)

```
/* Programma che calcola il massimo fra tre numeri
inseriti da tastiera */

#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
     scanf(" %d", &tmp);
     max = (max < tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```

C : un primo programma (3)

```
/* Programma che calcola il massimo fra tre numeri
inseriti da tastiera */
```

Commenti

```
#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
     scanf(" %d", &tmp);
     max = (max < tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```

Commenti

- I commenti possono essere inseriti con
  - `/* qua un commento anche su più linee */`
  - `// commento su singola linea`

C : un primo programma (4)

Inclusione delle informazioni necessarie per utilizzare le funzioni nelle librerie standard di I/O del C

```
#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
     scanf(" %d", &tmp);
     max = (max > tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```

C : un primo programma (5)

```
#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
    scanf(" %d", &tmp);
    max = (max > tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```

Inclusione delle informazioni necessarie per utilizzare le funzioni nelle librerie standard di I/O del C

Funzioni per leggere da tastiera, visualizzare dati

C : un primo programma (6)

```
#include <stdio.h>
#define  N 3
int main (void) {
   int i, tmp, max = 0;
   printf("Inserisci %d interi positivi\n", N);
   for (i = 0; i < N; i++) {
      scanf("%d", &tmp);
      max = (max > tmp)? max : tmp ;
   }
   printf("Il massimo è %d \n", max);
   return 0;
}
```

Definizione di MACRO (costante simbolica)

- N è il nome simbolico per la sequenza di caratteri "3"
- Ogni occorrenza di N viene sostituita da "3" prima di iniziare la compilazione

C : un primo programma (7)

Intestazione della funzione, fornisce :
- il tipo del valore prodotto dalla funzione (`int`)
- descrizione degli argomenti (in questo caso nessuno (`void`))

```
#include <stdio.h>
#define  N 3
int main (void) {
   int i, tmp, max = 0;
   printf("Inserisci %d interi positivi\n", N);
   for (i = 0; i < N; i++) {
      scanf("%d", &tmp);
      max = (max > tmp)? max : tmp ;
   }
   printf("Il massimo è %d \n", max);
   return 0;
}
```

C : un primo programma (8)

```
#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
     scanf("%d", &tmp);
     max = (max > tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```

Dichiarazione di variabili *locali* alla funzione
(il loro spazio di memoria viene allocato all'inizio della esecuzione della funzione e deallocato alla fine)

C : un primo programma (9)

Invocazione della funzione di libreria che scrive su video

```
#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
     scanf("%d", &tmp);
     max = (max > tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```

C : un primo programma (10)

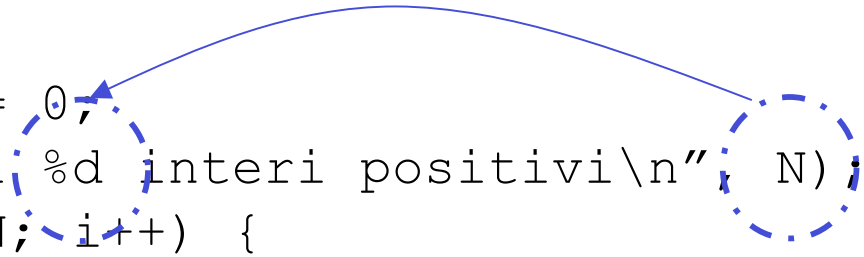
Placeholder (segnaposto)
per un valore intero (%d)

```
#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
     scanf("%d", &tmp);
     max = (max > tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```

C : un primo programma (11)

Il segnaposto viene rimpiazzato con il valore dei parametri (in ordine)

```
#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
     scanf("%d", &tmp);
     max = (max > tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```



C : un primo programma (12)

```
#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
     scanf("%d", &tmp);
     max = (max > tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```

Invocazione della funzione di libreria che legge da tastiera

C : un primo programma (13)

```
#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
     scanf("%d", &tmp);
     max = (max > tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```

Indirizzo (`&`) della variabile (`tmp`) in cui deve essere inserito il valore intero (`%d`) letto da tastiera.

C : un primo programma (14)

Espressione condizionale

```
#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
     scanf("%d", &tmp);
     max = (max > tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```

C : un primo programma (15)

Comando `return` : termina l'esecuzione della funzione e restituisce il valore delle espressione specificata

```
#include <stdio.h>
#define  N 3
int main (void) {
  int i, tmp, max = 0;
  printf("Inserisci %d interi positivi\n", N);
  for (i = 0; i < N; i++) {
     scanf("%d", &tmp);
     max = (max > tmp)? max : tmp ;
  }
  printf("Il massimo è %d \n", max);
  return 0;
}
```

Per convenzione se tutto è andato bene il `main` restituisce 0.

Come si esegue un programma C?

- Prima di essere eseguito dal processore il programma deve essere
  - 1. pre-processato
  - 2. compilato
  - 3. collegato (*linking*)
  - 4. caricato in memoria (*loading*)

- Vediamo come funzionano le fasi 1,2,4
  - il *linking* verrà discusso più avanti

Pre-processing e compilazione

- Il programma deve essere editato e salvato in un file con estensione `'.c'`
  - es. posso salvare il programma esempio nel file `prova.c`
- Non c'è alcun rapporto fra il nome del file ed il nome delle funzioni in esso contenute (!= Java)

Pre-processing e compilazione (2)

- Il compilatore può essere invocato con il comando

```
gcc -Wall -pedantic -g prova.c -o nomees
```

  - `-Wall -pedantic` opzioni che aumentano il numero di controlli e di messaggi di 'avvertimento' visualizzati
  - `-g` opzione che include anche le informazioni necessarie al debugger
  - `prova.c` nome del file da compilare/preprocessare
  - `-o nomees` opzione che permette di decidere il nome del file eseguibile (`nomees`). Altrimenti il file eseguibile si chiama di default `a.out`

Pre-processing e compilazione (3)

- Preprocessing
  - `#include <file>`
  - `#include "file"`

    queste linee vengono rimpiazzate dal contenuto dei file specificati (`<file>` viene cercato tra gli header standard, su Linux `/usr/include`; `"file"` viene cercato a partire dalla directory corrente)
  - `#define nome testo`
    - rimpiazza ogni occorrenza di nome con il testo specificato
    - è possibile specificare del testo parametrico

Pre-processing e compilazione (4)

- Compilazione
  - trasforma il file preprocessato (senza più `#include` o `#define`) in un *file eseguibile* che contiene
    - alcune informazioni di carattere generale
    - rappresentazione binaria di (parte dei) dati del programma (variabili globali)
    - codice assembler eseguibile dal processore target (*testo*)
  - per capire come avviene questa traduzione, dobbiamo specificare come un programma vede la memoria durante la sua esecuzione (*spazio di indirizzamento*)
  - un programma in esecuzione viene comunemente detto *processo*