DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA

POLITECNICO DI MILANO

Fondamenti di Informatica





Argomenti avanzati

Slide del Prof. Miele

DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA

POLITECNICO DI MILANO

Fondamenti di Informatica





Passaggio di argomenti da riga di comando

Argomenti da riga di comando

- Quando si invoca un programma da riga di comando molte volte specifichiamo degli argomenti
  - Es: `gcc ciao.c -o ciao`
    - Il nome dell'eseguibile invocato è `gcc`
    - Il resto del comando è composto da una lista di stringhe separate da spazi che rappresentano gli argomenti specificati durante l'invocazione
- Gli argomenti rappresentano i parametri in ingresso per l'esecuzione del programma invocato

Programmi con argomenti in C

- In C, per creare un eseguibile a cui sia possibile passare argomenti è necessario definire il prototipo del `main` così:

```
int main(int argc, char *argv[])
```

- `argc` contiene il numero di argomenti con cui il programma viene lanciato (incluso il nome stesso del programma)
- `argv` è un array di stringhe che contiene tutti gli argomenti con cui il programma viene lanciato
  - Più precisamente `argv` rappresenta un array di puntatori a `char` (cioè un doppio puntatore) dove ciascuno dei puntatori punta ad una stringa
  - `argv[0]` contiene il nome completo dell'eseguibile lanciato, con eventuale percorso

Esempio di uso degli argomenti

- Scrivere un programma che riceve come argomenti un numero intero N ed una stringa S. Il programma visualizza prima il nome dell'eseguibile e poi N volte la stringa S

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char *argv[]){
  int n, i;
  if(argc==3){
    n=atoi(argv[1]);
    printf("%s\n", argv[0]);
    for(i=0; i<n; i++)
      printf("%s\n", argv[2]);
  }else{
    printf("Errore nella specifica dei parametri\n");
  }
  return 0;
}
```

Prototipo del `main` con i parametri

Controllo che il numero degli argomenti ricevuti sia corretto

Conversione della stringa `argv[1]` in un intero usando l'apposito sottoprogramma di libreria

Utilizzo degli argomenti

DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA

POLITECNICO DI MILANO

Fondamenti di Informatica





Suddivisione di un programma in moduli

Programmazione modulare

- Un sistema software è costituito da un insieme di moduli e da relazioni tra questi
- Ogni modulo è costituito da una interfaccia e da una implementazione
  - L'interfaccia è l'insieme di tutti e soli i suoi elementi che devono essere conosciuti da chi usa il modulo per farne un uso appropriato
  - L'implementazione è l'insieme dei meccanismi che permettono di realizzare le funzionalità del modulo

Modularizzazione in C

- In C si utilizzano gli header file (file `.h`) per definire le interfacce, principalmente per:
  - Dichiarazioni di costanti (tramite `#define`), tipi e variabili globali
  - Dichiarazione dei prototipi dei sottoprogrammi
- Le implementazioni dei sottoprogrammi vengono invece incluse in corrispondenti file sorgenti (file `.c`)
- Per utilizzare un modulo occorre
  - Includere il suo header file

    ```
    #include "nome_modulo.h"
    ```

  - Compilare l'implementazione del modulo insieme al sorgente che lo utilizza

File header di un modulo

- File `liste.h` per un modulo per la gestione delle liste dinamiche

```
#ifndef LISTA_H
#define LISTA_H

typedef struct nodo_{
  int num;
  struct nodo_ *next;
} nodo_t;

nodo_t* inserisciInCoda(nodo_t*, int);
nodo_t* inserisciInTesta(nodo_t*, int);
nodo_t* rimuovi(nodo_t*, int);
void visualizza(nodo_t*);
nodo_t* distruggiLista(nodo_t*);
int esisteElemento(nodo_t*, int);
...
#endif
```

Dichiarazione del nuovo tipo per il singolo nodo della lista

Dichiarazione dei prototipi dei sottoprogrammi definiti sulle liste

In questo specifico esempio non sono state definite costanti o variabili globali e non sono stati inclusi header di altri moduli

File header di un modulo

- File `liste.h` per un modulo per la gestione delle liste dinamiche

```
#ifndef LISTA_H
#define LISTA_H

typedef struct nodo_{
  int num;
  struct nodo_ *next;
} nodo_t;
```

La direttiva di preprocessore `#ifndef` (terminata dalla direttiva `#endif`) specifica che il codice compreso va mantenuto se il simbolo `LISTA_H` non esiste; altrimenti va cancellato

Definizione del simbolo `LISTA_H`. Si noti che non c'è bisogno di specificare alcun valore per il simbolo come generalmente fatto

```
nodo_t* inserisciInCoda(nodo_t*, int);
nodo_t* inserisciInTesta(nodo_t*, int);
nodo_t* rimuovi(nodo_t*, int);
void visualizza(nodo_t*);
nodo_t* distruggiLista(nodo_t*);
int esisteElemento(nodo_t*, int);
...
#endif
```

La direttiva `#ifndef` è necessaria per evitare dichiarazioni duplicate dello stesso simbolo generate quando lo stesso header viene incluso (indirettamente) più di una volta in un file sorgente

File header di un modulo

- Esempio di elaborazione delle direttive:
  - Organizzazione del codice in più file che si vuole compilare:
    - Si considerino due file header, `lista.h` e `lista_estesa.h`, ed un file sorgente `main.c`
    - Si assuma che `lista.h` sia incluso sia in `lista_estesa.h` che in `mioprog.c` e `lista_estesa.h` sia incluso in `mioprog.c`
  - Quando si compila `mioprog.c`:
    1. Le direttive `#include` vengono sostituite con il corrispondente contenuto del file header
       - Conseguentemente il contenuto di `lista.h` viene copiato due volte nel file `mioprog.c` (che potenzialmente porta ad un errore di compilazione!)
    2. La prima volta che viene incontrata la direttiva `#ifndef LISTA_H`, il simbolo `LISTA_H` non è precedentemente definito e quindi il codice racchiuso viene mantenuto e quindi il simbolo `LISTA_H` viene definito dalla `#define`
    3. La seconda volta che viene incontrata la direttiva `#ifndef LISTA_H`, il simbolo `LISTA_H` è definito e quindi il codice racchiuso viene eliminato (quindi evitando ridefinizioni duplicate degli stessi prototipi)

File sorgente di un modulo

- File `liste.c` per un modulo per la gestione delle liste dinamiche

```
#include <stdio.h>
#include <stdlib.h>
#include "lista.h"
```

Inclusione di tutti gli header di cui si vuole usare i sottoprogrammi in essi dichiarati

- Se il file header è salvato in un percorso di sistema il nome del file viene specificato tra i simboli `< >` (per esempio `stdio.h`)
- Se il file header è salvato in un percorso locale il nome del file viene specificato tra `""` (per esempio `lista.h`)

```
nodo_t* insInCoda(nodo_t* l, int n){
   ...
}

nodo_t* insInTesta(nodo_t* l, int n){
   ...
}
```

Implementazione dei sottoprogrammi dichiarati nell'header

File sorgente di un programma

- File `main.c`

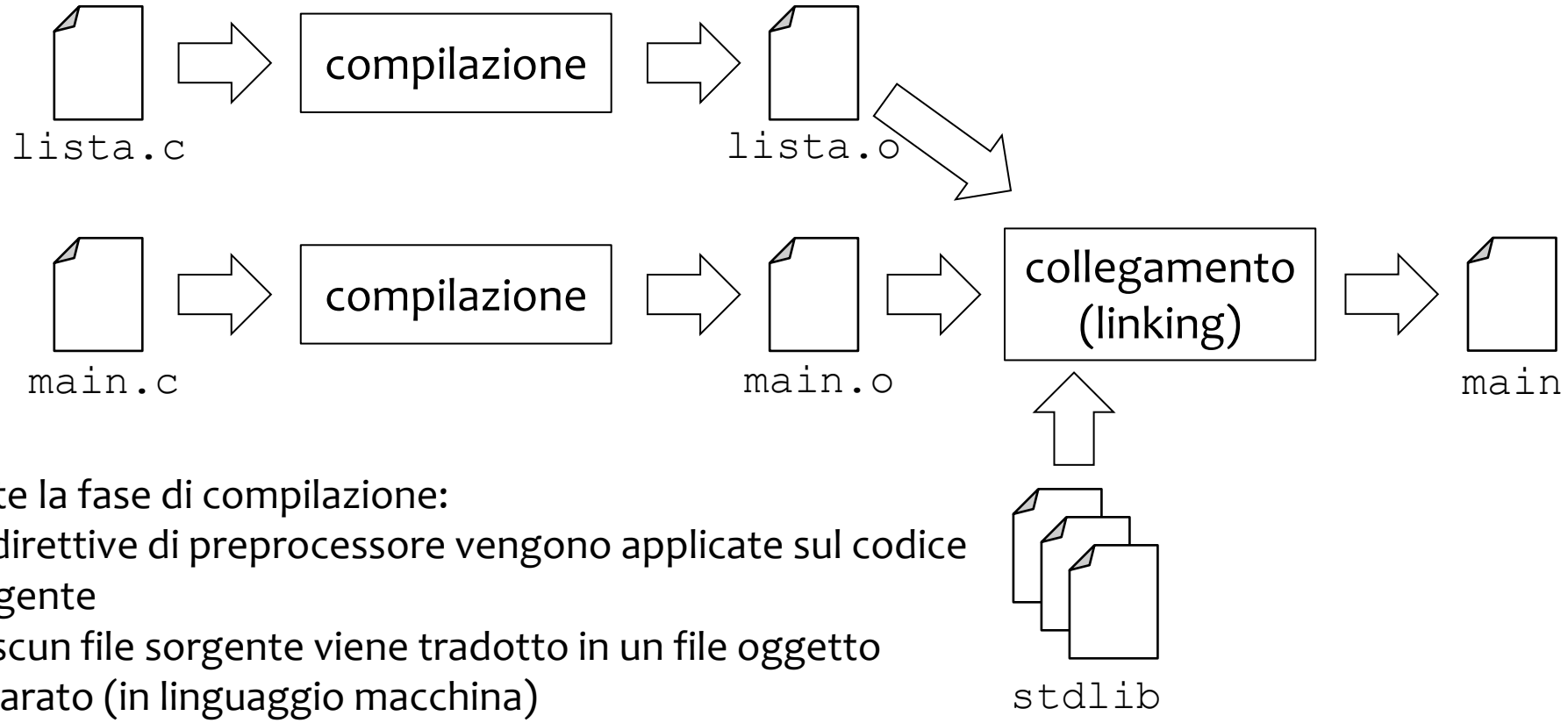
```
#include<stdio.h>
#include"lista.h"

int main(){
  nodo_t* testa=NULL;
  ...
  visualizzaLista(testa);
  ...
  return 0;
}
```

Inclusione di tutti gli header di cui si vuole usare i sottoprogrammi in essi dichiarati

Invocazione delle funzioni dei vari header inclusi

- In generale è possibile suddividere un progetto software in quanti file header e sorgente si vuole
- Non è obbligatorio avere un rapporto uno a uno tra file header e file sorgente

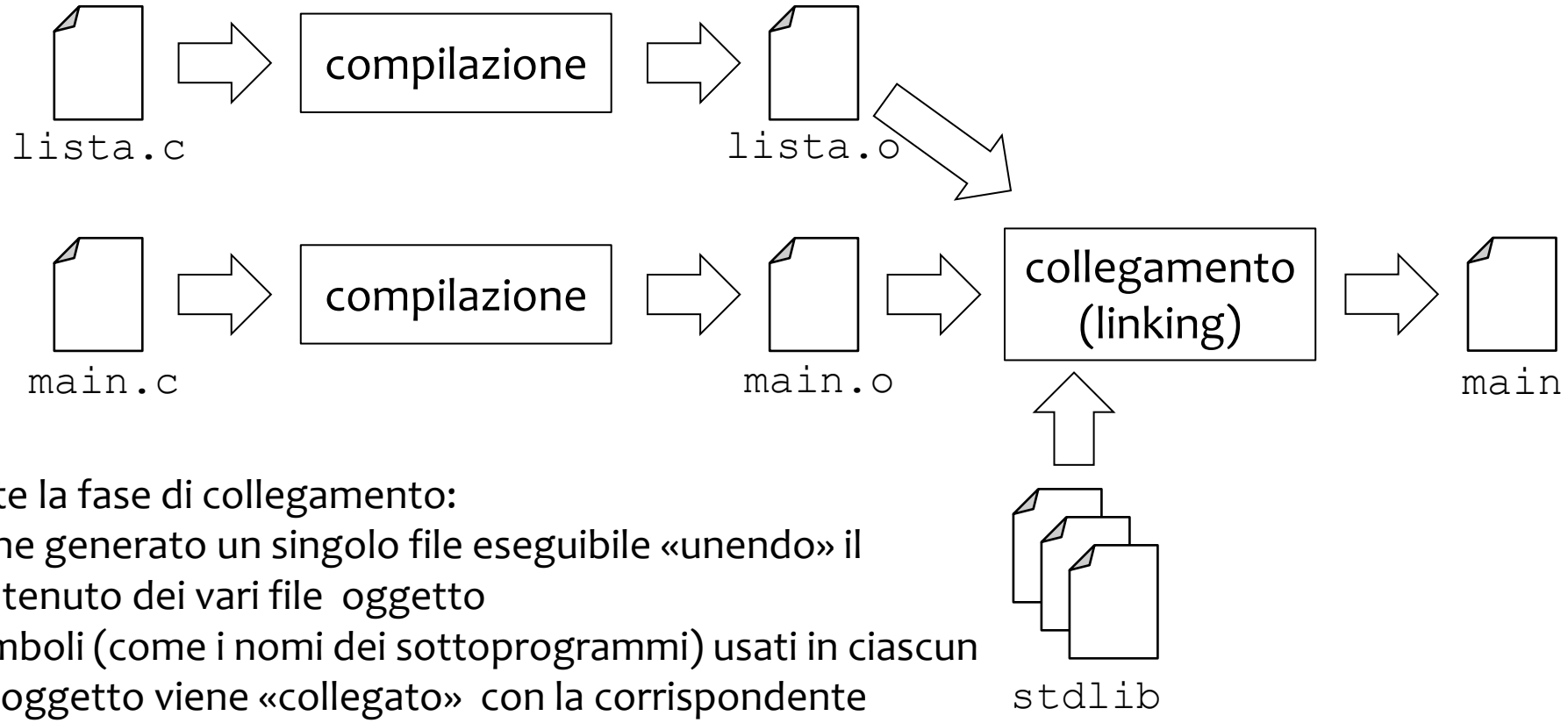
Compilazione di un progetto composto da più file



Durante la fase di compilazione:

- Le direttive di preprocessore vengono applicate sul codice sorgente
- Ciascun file sorgente viene tradotto in un file oggetto separato (in linguaggio macchina)
- Vengono segnalati eventuali errori di compilazione (sintassi sbagliata)

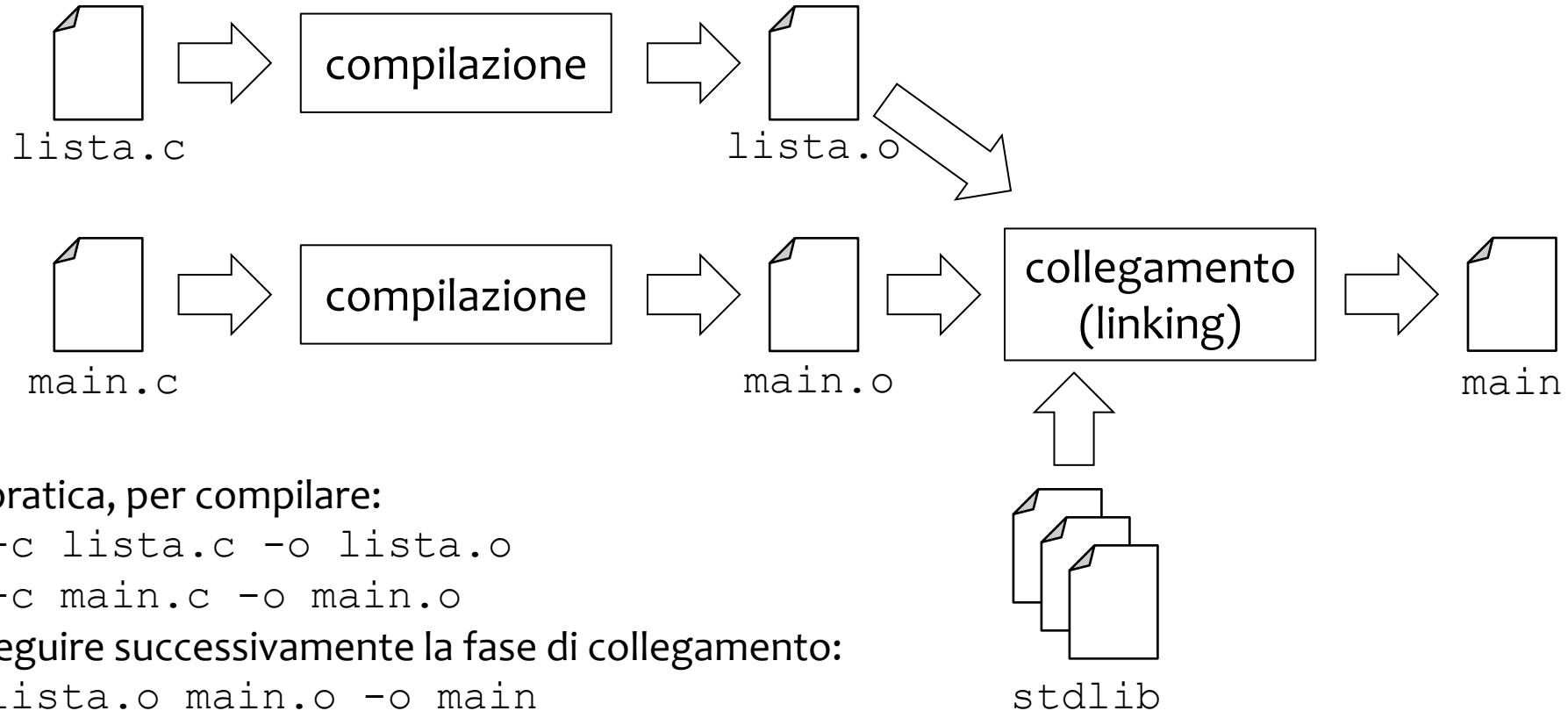
Compilazione di un progetto composto da più file



Durante la fase di collegamento:

- Viene generato un singolo file eseguibile «unendo» il contenuto dei vari file oggetto
- I simboli (come i nomi dei sottoprogrammi) usati in ciascun file oggetto viene «collegato» con la corrispondente implementazione fornita in un altro file oggetto (o in una libreria tipo quelle standard del C)
- Vengono segnalati eventuali errori di collegamento dovuti a simboli che non possono essere risolti (mancanza dell'implementazione di un sottoprogramma) o dichiarati più volte

Compilazione di un progetto composto da più file



Nella pratica, per compilare:

```
gcc -c lista.c -o lista.o
gcc -c main.c -o main.o
```

Per eseguire successivamente la fase di collegamento:

```
gcc lista.o main.o -o main
```

Per eseguire entrambe le fasi in una sola volta:

```
gcc lista.c main.c -o main
```