3. Strumenti per la programmazione software

Questo approfondimento è dedicato in particolare all'analisi degli ambienti software orientati più specificamente al lavoro di programmazione.

Nel corso di tale analisi verrà descritto il percorso operativo che permette di arrivare all'esecuzione di un nuovo programma, partendo dalla sua scrittura in linguaggio evoluto.

Un programma in linguaggio evoluto è un testo scritto usando un certo vocabolario e certe regole grammaticali ben precise: per mezzo delle operazioni di *compilazione* e *linking* il programma viene trasformato in modo tale da poter essere *rilocato* ed *eseguito* da parte del processore.

Compilazione

Per consentire una maggiore facilità d'uso e quindi una maggiore produttività, i linguaggi evoluti fanno impiego di oggetti identificati da nomi. Si possono assegnare nomi a *variabili, procedure* e *funzioni*, e usarli per costruire programmi.

La conseguenza è che la descrizione di un algoritmo, costituita da un programma scritto in linguaggio evoluto, usa espressioni abbastanza vicine a un linguaggio umano.

Spesso il problema che viene risolto dal programma prescinde dal calcolatore che materialmente eseguirà il programma stesso: per esempio, se si scrive un programma che calcola l'ipotenusa di un triangolo rettangolo dati i cateti, l'algoritmo è ben noto e non dipende dal fatto di eseguire il programma su un personal computer, su un computer collegato a una rete o su un mainframe. È certamente desiderabile poter scrivere un unico programma che si possa far capire a tutte queste macchine, nonostante esse parlino linguaggi differenti, piuttosto che dover scrivere un programma diverso per ogni macchina.

La compilazione trasforma le istruzioni del linguaggio evoluto in codice macchina.

Se le macchine *target* (destinazione), dispongono di un compilatore per il linguaggio usato, è sufficiente eseguire la compilazione dello stesso *codice sorgente* su ciascuna per poter in seguito eseguire il *codice oggetto* che ne deriva.

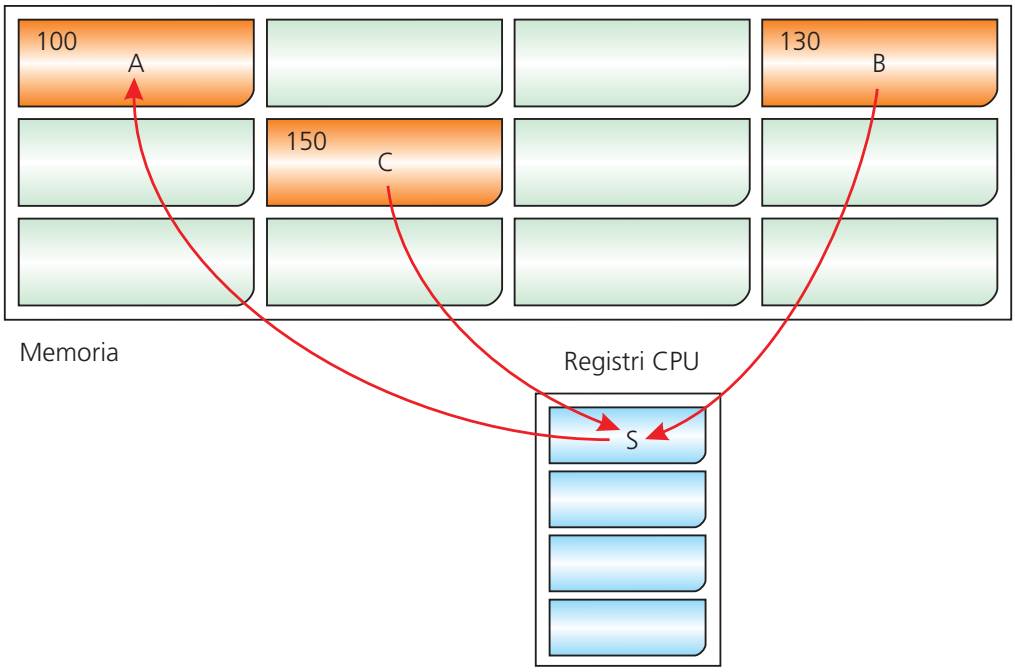
In pratica la compilazione svolge la funzione di interfaccia tra l'algoritmo del programmatore, espresso dal codice sorgente, e la macchina che comprende solo istruzioni in codice oggetto. Un solo programmatore può così sviluppare programmi per molte piattaforme target, conoscendo al limite un solo linguaggio di programmazione e scrivendo una sola volta il codice sorgente.

Il lavoro svolto dal compilatore consiste nel trasformare le istruzioni del **codice sorgente**, che come si è detto fanno uso di nomi, in istruzioni di **codice oggetto**, che possono solo far riferimento alle primitive di sistema e alle posizioni di memoria.

Per esempio, un'istruzione di assegnamento

$$A = B + C$$

deve essere trasformata in una sequenza di istruzioni in codice macchina che contengono riferimenti agli indirizzi dove sono memorizzati i valori delle variabili B e C e all'indirizzo dove memorizzare il valore calcolato di A.



Se gli indirizzi di A, B e C sono 100, 130 e 150, l'istruzione viene trasformata come segue:

- *carica* nel registro S il contenuto della parola di indirizzo 130;
- *somma* a S il contenuto della parola di indirizzo 150;
- *trasferisci* il contenuto di S nella parola di indirizzo 100.

A seconda della macchina usata, la codifica finale può variare, ma in generale descrive queste stesse tre operazioni.
Le chiamate ai sottoprogrammi vengono trasformate in istruzioni di salto a subroutine che contengono l'indirizzo di caricamento della sequenza che costituisce la subroutine.
Se il sottoprogramma prevede l'uso di parametri, il compilatore scrive le istruzioni in codice oggetto necessarie perché i loro valori vengano trasmessi.

La traduzione del sorgente, memorizzato su un file, richiede due attività principali:

- verifica della correttezza formale del programma, con segnalazione di errori di **lessico** (se si usano termini non permessi) e di **sintassi** (se si scrivono in modo improprio le frasi). Un tipico errore di lessico può essere la scrittura errata di una parola chiave del linguaggio, mentre si commette un errore di sintassi se si dimentica il punto e virgola alla fine di un'istruzione in linguaggio C++. Un altro errore segnalato dal compilatore è l'impiego di un termine che identifica un elemento non dichiarato, per esempio una variabile.
- generazione della versione tradotta del programma, scritta nel formato **oggetto collegabile**, su un file del disco.

L'output della compilazione viene comunemente detto **oggetto compilato**.

Nel caso più semplice, l'intero programma sorgente sta su un solo file e viene compilato in una sola volta.
Per programmi più grandi, allo sviluppo dei quali magari partecipano più programmatori, diverse parti del programma vengono scritte separatamente, risiedono su diversi file e vengono compilate in momenti diversi: si parla allora di *compilazione modulare* e i diversi programmi sorgente vengono detti **moduli**.

Passiamo ora a dettagliare meglio il processo di traduzione dal linguaggio evoluto al codice oggetto.
Occorre guardare con attenzione ad alcuni aspetti importanti che riguardano il codice eseguibile, per comprendere bene i meccanismi che permettono il corretto svolgimento del compito del compilatore. In particolare verrà approfondito il concetto di *indirizzo* entro il codice oggetto.
Una generica istruzione in linguaggio macchina è costituita da:

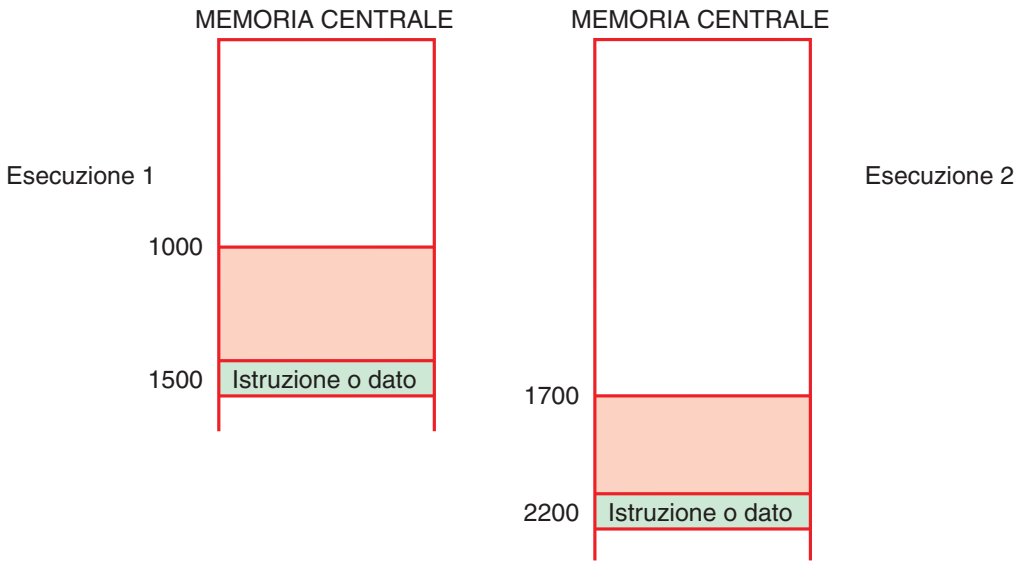
- un **codice operativo** che identifica il tipo di operazione;
- uno o due **operandi** che indicano gli oggetti coinvolti nell'operazione: questi possono essere costanti oppure indirizzi di memoria o anche registri della CPU.

Un'istruzione a un solo operando è per esempio l'istruzione di salto a un certo indirizzo, mentre la somma è un'istruzione a due operandi.
Alcune istruzioni non richiedono alcun operando, per esempio l'istruzione di fine programma.

Gli operandi sono posti in indirizzi di memoria, ai quali ci si riferisce in diversi modi:

- l'**indirizzo assoluto** di una parola di memoria è la posizione che questa occupa in memoria centrale all'istante di esecuzione del programma. In diverse esecuzioni del programma, questo può cambiare. Il programma, infatti, può essere caricato (o *impiantato*) una volta a partire dall'indirizzo assoluto 1000 e un'altra a partire dal 1700. Perciò le istruzioni del programma hanno indirizzi assoluti, nella seconda esecuzione, maggiori di 500 rispetto alla prima; per esempio quella che alla prima esecuzione ha occupato l'indirizzo 1500, nella seconda occuperà l'indirizzo 2200.
- l'**indirizzo rilocabile** è l'indirizzo che una parola avrebbe se il programma fosse impiantato in posizione zero. Nell'esempio precedente, l'istruzione considerata ha indirizzo rilocabile 500, cioè la differenza tra il suo indirizzo assoluto e l'indirizzo d'impianto in entrambe le esecuzioni. In pratica l'indirizzo rilocabile è la distanza di una parola dalla prima parola del programma. Per passare dall'indirizzo rilocabile a quello assoluto basta quindi sommare al primo l'indirizzo di impianto del programma:

Indirizzo assoluto = Indirizzo rilocabile + Indirizzo di impianto.



- l'**indirizzo autorilocante** di un operando è espresso come la sua posizione relativamente all'istruzione che lo contiene. L'istruzione esprimerà un concetto simile a "Esegui questo comando prendendo un dato 100 posizioni in avanti e l'altro dato 50 posizioni indietro rispetto alla posizione di questa istruzione".
  +100 e –50 costituiscono esempi di indirizzi autorilocanti.

Per capire meglio il concetto di indirizzo autorilocante facciamo un esempio pratico.
Se un'istruzione di salto ha indirizzo rilocabile 1000, cioè dista 1000 dalla prima istruzione del programma, e richiede il trasferimento all'istruzione di indirizzo rilocabile 600, l'indirizzo autorilocante di quest'ultima è –400. Ossia, si deve saltare 400 posizioni all'indietro per raggiungere l'indirizzo di destinazione.
Per passare dall'indirizzo autorilocante a quello assoluto basta sommare all'indirizzo autorilocante l'indirizzo dell'operazione in esecuzione, contenuto nel contatore di programma (*program counter*):

Indirizzo assoluto =
Indirizzo autorilocante + Indirizzo dell'operazione in esecuzione.

È importante osservare che il processore riconosce solo indirizzi assoluti e che quindi, prima o poi, questi dovranno essere calcolati.

Se un programma in codice oggetto usa solo indirizzi autorilocanti, si dice costituito da **position independent code** (*codice indipendente dalla posizione*), perché gli indirizzi assoluti, quando il programma è in esecuzione, si possono calcolare indipendentemente dall'indirizzo di impianto.

	ESECUZIONE 1	ESECUZIONE 2
indirizzo di impianto	700: Start	900: Start
destinazione del salto	1000: Istruzione	1200: Istruzione
istruzione di salto	1500: salta a Istruzione	1700: salta a Istruzione

Nel caso descritto nella tabella, sono riportati gli indirizzi assoluti di due istruzioni del programma e l'indirizzo di impianto del programma in due diverse esecuzioni.
Le istruzioni di salto si trovano nei due casi a indirizzi diversi, 1500 e 1700, dovuti al diverso indirizzo di impianto: in entrambi i casi si trovano a +800 rispetto all'indirizzo di impianto *Start:* 700 nel primo caso, 900 nel secondo. L'indirizzo rilocabile dell'istruzione di salto è perciò 800. Le istruzioni a cui saltare hanno poi indirizzo rilocabile +300 (1000–700 nel primo caso, 1200–900 nel secondo).
Per calcolare l'indirizzo autorilocante da usare nell'istruzione di salto basta guardare quanto dista l'indirizzo al quale saltare rispetto all'istruzione di salto: siccome si chiede di saltare 500 posizioni all'indietro, risulta uguale a –500.

A questo punto possiamo tornare al problema della traduzione delle istruzioni dal linguaggio evoluto al codice oggetto.
Le caratteristiche dei moduli in formato **oggetto collegabile** sono le seguenti:

- le istruzioni sono scritte in linguaggio macchina;
- gli indirizzi relativi a operandi sono in formato rilocabile;
- gli indirizzi degli operandi contenuti in altri moduli non sono definiti: essi infatti non possono essere determinati finché tutti i moduli non vengono uniti per formare un solo oggetto;
- contengono due tabelle, *tabella degli ingressi* e *tabella degli oggetti esterni*, che consentono di unire tra loro i moduli.

Queste caratteristiche permettono infatti di ottenere moduli oggetto che potranno essere successivamente legati insieme per costituire il programma vero e proprio. Ogni modulo rende noto agli altri cosa contiene e cosa richiede in modo che la fase di collegamento successiva possa risolvere tutte le questioni in sospeso.

Le tabelle degli ingressi e delle uscite costituiscono l'elemento di interfaccia del modulo verso l'esterno ai fini di impiegare il modulo come elemento di un insieme complesso: chi desidera usare il modulo deve unirlo agli altri per mezzo di tali tabelle.

Osserviamo che non si sta ancora eseguendo il programma: lo si sta semplicemente costruendo. Si sta scrivendo del codice macchina che possa essere eseguito in modo corretto, prelevando variabili dai posti giusti ed eseguendo la corretta serie di operazioni. È la stessa situazione di chi assembla un circuito elettronico: predisporre e unire elementi in modo che, in seguito, il loro insieme funzioni come previsto.

Se un modulo chiama una procedura esterna in modo corretto, ma la costruzione del programma è sbagliata, l'indirizzo a cui si salta è scorretto e i risultati imprevedibili. Un tale errore non è nel programma, i cui moduli sono tutti corretti, ma nella sua costruzione: come se si fosse montato un chip al contrario o con saldature mal fatte.

La **tabella degli ingressi** contiene l'elenco degli oggetti definiti entro il modulo e che possono essere usati dagli altri moduli.
Per ciascun oggetto occorre specificare:

- il nome (della procedura o della variabile);
- il numero e tipo dei parametri formali delle procedure o funzioni;
- il loro indirizzo rilocabile (nel caso di una procedura, l'indirizzo rilocabile della prima istruzione).

La **tabella degli oggetti esterni** contiene l'elenco degli oggetti usati nel modulo ma non definiti. Essa ha lo scopo di assegnare il valore corretto agli indirizzi rimasti indefiniti, una volta che i moduli vengano uniti.
Ogni procedura esterna viene descritta mediante:

- il nome;
- la lista degli indirizzi rilocabili dove compaiono istruzioni che richiamano la procedura;
- il numero e il tipo dei parametri formali, come appaiono nell'intestazione, per poter effettuare il controllo sul **matching** (accordo) con i parametri effettivamente richiesti dalla procedura.

Le variabili globali devono essere dichiarate tutte nella sezione dichiarativa del modulo che contiene il *main body*, mentre negli altri moduli è indispensabile dichiarare solo quelle che vengono effettivamente usate.

In questo modo il compilatore assegna locazioni di memoria a tali variabili solo nel programma generato dalla compilazione del *main body*, mentre le considera come oggetti esterni compilando gli altri moduli.

Nella tabella degli oggetti esterni di tali moduli compare perciò, per ogni variabile globale, la lista delle istruzioni che la adoperano, mentre nella tabella degli ingressi del *main body* compaiono gli indirizzi rilocabili delle variabili stesse.

Per ogni variabile, inoltre, viene precisato il suo tipo per controlli di accordo con i parametri richiesti (*matching*).

Per mezzo della compilazione si è dunque passati da una collezione di moduli in codice sorgente a una di moduli in codice collegabile.

La dipendenza dal tipo di macchina sulla quale eseguire il programma è rimossa e si può passare alla costruzione del programma.

Linking

Il **linking** (o *linkage*), cioè **collegamento**, viene eseguito sui file che contengono il programma nel formato oggetto collegabile e genera uno o più file in **formato caricabile**.
Il programma che esegue questa operazione si chiama **linker** (*collegatore*). Le sue funzioni sono già state in parte introdotte parlando del compilatore, e vengono qui riassunte e completate.

Il linker:

- verifica che non vi sia nessun **mismatch** (*disaccordo*) tra le caratteristiche degli oggetti nelle tabelle degli ingressi e nelle tabelle degli oggetti esterni: questo può verificarsi se una variabile globale viene dichiarata di tipo reale in un modulo e intero in un altro;
- unisce i moduli, uno di seguito all'altro, calcolando i nuovi indirizzi rilocabili che risultano dalla somma dell'indirizzo, così come appare nel modulo e dell'indirizzo della prima parola di ciascun modulo dopo il collegamento;
- assegna i valori corretti agli indirizzi lasciati indefiniti dal compilatore, perché relativi a oggetti esterni al modulo compilato.

Se l'unione di tutti i moduli non è possibile, perché per esempio il file che ne risulterebbe è troppo grande, il collegamento avviene generando più file che conservano alcune indeterminazioni negli indirizzi. Queste devono essere risolte all'atto dell'esecuzione del programma (*run time*).

In questo contesto si può esaminare una caratteristica del sistema operativo *Windows*, denominata con la sigla **DLL** (*Dynamic Link Library,* libreria per il collegamento dinamico).

Queste librerie sono uno strumento fondamentale per la programmazione in tutti i sistemi operativi che utilizzano l'ambiente operativo con interfaccia grafica.

Un programma per Windows non deve preoccuparsi di gestire tutti gli strumenti standard usati per realizzare le interfacce grafiche, in quanto queste sono già disponibili all'interno delle librerie. Usando strumenti di *programmazione visuale* il programmatore deve limitarsi a scegliere gli strumenti che desidera e collocarli in finestre generate automaticamente.

Questo può avvenire grazie al fatto che il codice necessario a definire il comportamento dei tipici oggetti Windows è già disponibile in apposite librerie.

I bottoni, per esempio, che simulano visivamente il comportamento di pulsanti veri, con l'effetto premuto/sollevato, vengono interamente gestiti da codice preesistente e il programmatore deve solo assegnare le peculiarità dello specifico bottone (cosa scrivere o disegnare all'interno, che routine eseguire quando il bottone viene premuto), senza doversi preoccupare della gestione di base.

Il codice è contenuto in librerie e a esso si accede nel tempo dell'esecuzione. Quando l'utente preme il bottone, la DLL preposta manda in esecuzione il codice necessario alla risposta standard all'evento provocato dall'utente e apporta le necessarie modifiche grafiche.

Per i programmatori le DLL sono estremamente comode per raccogliere le routine di uso comune a più programmi, in quanto permettono di avere una gestione globalmente omogenea dei problemi. Rispetto alle librerie tradizionali, che occorre collegare al programma dopo la compilazione, le DLL presentano il vantaggio di poter essere aggiornate senza bisogno di modificare in alcun modo i programmi che le adoperano. Infatti, siccome il link avviene nel tempo dell'esecuzione, se le procedure contenute nella DLL non variano i parametri richiesti, i programmi possono continuare a usarle anche se il loro contenuto è del tutto diverso.

Per esempio supponiamo di aver creato una DLL contenente una funzione *MCD (A,B)* che calcola il massimo comun divisore di due numeri A e B seguendo un certo algoritmo e di avere poi scritto 100 programmi che usano tale funzione. Se cambiamo l'algoritmo nella funzione sostituendolo con uno più efficiente o più completo o comunque ritenuto migliore e ricreiamo la DLL, i 100 programmi non dovranno in alcun modo essere cambiati, in quanto la loro chiamata a *MCD* resta comunque valida.
Operativamente si tratta di un vantaggio importante, perché non occorre rintracciare singolarmente tutti i programmi e sostituirli, ma basta un unico file con la DLL modificata per effettuare l'aggiornamento completo.

Rilocazione

Il programma in formato caricabile, viene appunto caricato in memoria centrale per essere eseguito. L'indirizzo a partire dal quale esso viene caricato è, come già detto, il suo **indirizzo di impianto**.
La rilocazione trasforma gli indirizzi rilocabili del programma in indirizzi assoluti sommando a ciascuno l'indirizzo di impianto.

Questo può avvenire in tre modi:

- se l'indirizzo di impianto è sempre lo stesso, ed è noto al linker, esso può provvedere direttamente alla rilocazione che è detta **rilocazione statica**: nei sistemi dedicati uniprogrammati i programmi utente vengono sempre caricati nella stessa zona di memoria, perciò è possibile seguire questo metodo. Questo ha però lo svantaggio di non consentire l'esecuzione del programma se per qualche motivo occorre cambiare l'indirizzo di impianto. In tal caso occorre ricollegare il programma partendo dai moduli collegabili.
- se l'indirizzo d'impianto può variare o non è noto al linker, la rilocazione può essere effettuata staticamente ogni volta che il programma viene caricato, prima di eseguirlo. In altre parole, mentre il programma viene copiato in memoria, quando si incontra un indirizzo rilocabile questo viene sostituito con l'indirizzo assoluto, che è ormai noto e pari alla somma dell'indirizzo di impianto e dell'indirizzo rilocabile.
- nello stesso caso, si può invece ricorrere alla **rilocazione dinamica**, che si effettua calcolando gli indirizzi assoluti degli operandi nell'istante in cui l'istruzione che fa riferimento a essi viene eseguita. In tempo di esecuzione, incontrando un indirizzo rilocabile, viene automaticamente sommato l'indirizzo di impianto per ottenere l'indirizzo assoluto, che, ricordiamo, deve necessariamente essere fornito al processore.

Qualora il compilatore generi codice autorilocante, l'operazione di rilocazione non è strettamente necessaria, in quanto tutti gli indirizzi possono essere calcolati dinamicamente e in modo indipendente dall'indirizzo di impianto.

Esecuzione

A questo punto il programma può essere eseguito.
Restano da risolvere le indeterminazioni del programma quando esso è suddiviso in più file caricabili.
Quando nel corso dell'esecuzione si trova un riferimento a un oggetto esterno, deve essere attivato un meccanismo che consenta di caricare in memoria la libreria corretta ed eventualmente rilocata, in modo che l'elaborazione possa proseguire.
Un meccanismo soddisfacente al requisito può essere una tabella, analoga alla tabella delle uscite vista in compilazione.
Questa dovrà contenere i dati necessari alla corretta esecuzione della chiamata, ossia almeno il nome della libreria, l'identificativo della procedura richiesta e gli indirizzi dove si trovano i parametri che debbono essere trasmessi e dove si devono scrivere quelli ricevuti.

MATERIALI ON LINE

ESERCIZI

1 Quali di queste affermazioni, riferite al compilatore, sono vere (V) e quali false (F)?

a) Acquisisce in ingresso file eseguibili V F
b) Fornisce come output informazioni sugli errori di sintassi V F
c) Esegue i programmi un'istruzione alla volta V F
d) Fornisce in uscita oggetti collegabili V F
e) Fornisce in uscita file eseguibili V F

2 Un programma in esecuzione ha indirizzo di impianto 1100. L'istruzione all'indirizzo 1350 effettua la somma del contenuto delle locazioni 1700 e 1720 e scrive il risultato nella locazione 1710. Scrivere tale istruzione usando indirizzi assoluti, rilocabili ed autorilocanti. Usare la forma ADD A1 A2 S, dove A1 e A2 sono gli addendi e S la somma.

3 Scrivere le tabelle dei collegamenti di due moduli in formato collegabile, contenenti ciascuno chiamate a procedure contenute nell'altro.

4 Schematizzare con due disegni l'attività di linking dei due moduli dell'esercizio precedente. Nel primo disegno si mostri la situazione prima del linking, nel secondo la situazione dopo, evidenziando la risoluzione degli indirizzi indefiniti.