La struttura dei programmi

Docente: Ing. Edoardo Fusella
Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione
Via Claudio 21, 4° piano – laboratorio SECLAB
Università degli Studi di Napoli Federico II
e-mail: edoardo.fusella@unina.it

Le frasi di un linguaggio di programmazione

- Tutti i linguaggi di alto livello prevedono quattro tipologie di frasi diverse:
  - le **istruzioni** di calcolo ed assegnazione, che tradotte in linguaggio macchina indicano al processore le operazioni da svolgere;
  - le **strutture di controllo**, che definiscono l'ordine di esecuzione delle istruzioni;
  - le **frasi di commento**, che permettono l'introduzione di frasi in linguaggio naturale utili a rendere più comprensibili i programmi ad un lettore umano; le frasi di commento non vengono tradotte in linguaggio macchina
  - le **dichiarazioni**, con le quali il programmatore da ordini al traduttore del linguaggio di programmazione; anche le dichiarazioni non vengono tradotte in linguaggio macchina poiché servono solo a guidare il processo di traduzione.

Le dichiarazioni

- Tra le tante dichiarazioni le più importanti sono quelle con cui si definiscono le **variabili** sulle quali si svolgono le elaborazioni dell'algoritmo.
  - **Le variabili vanno sempre dichiarate prima di poter essere usate!!**
- Ad una variabile corrisponde una porzione di memoria la cui dimensione e le cui regole di uso dipendono dal suo **tipo**.
  - per *tipo di dato* si intende *un insieme di valori associati a delle operazioni definite su di essi*
- *Es: int var;*

In questo modo si riserva uno spazio in memoria che può contenere un intero e gli si assegna l'etichetta **var**;
Lo spazio di memoria potrebbe contenere già qualche valore spurio, per cui è necessario inizializzare la variabile (vedi slide 7)

Esempio

$$c = a + b$$

- *Somma tra numeri interi:*

  int a;

  int b;

  int c;

- *Somma tra numeri interi positivi:*

  unsigned int a;

  unsigned int b;

  unsigned int c;

- *Somma tra numeri reali (32 bit):*

  float a;

  float b;

  float c;

- *Somma tra numeri reali (64 bit):*

  double a;

  double b;

  double c;

Le frasi di commento

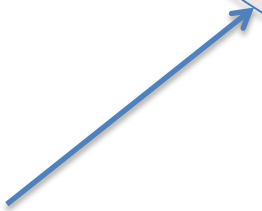
- Le frasi di commento sono frasi in linguaggio naturale prive di ogni valore esecutivo o dichiarativo che consentono di migliorare la leggibilità e la chiarezza del programma.
- Si distinguono in asserzioni e motivazioni.
  - *Le asserzioni sono commenti destinati a fissare in un punto del programma lo stato di una o più variabili. Per tale motivo permettono di chiarire le condizioni nelle quali vengono ad operare le istruzioni successive.*
  - *Le motivazioni sono invece commenti destinati a chiarire le ragioni e/o gli obiettivi per i quali il blocco di programma, successivo al commento, viene scritto.*

- *Es: int var; //var è il nome di una variabile*

Esempio

```
1  #include <cstdlib>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(int argc, char *argv[])
7  {
8      char buff[255];
9      cout<<"Digita qualcosa: ";
10     cin.getline(buff,255);
11     cout<<"Hai scritto: "<<buff<<endl;
12     system("PAUSE");
13     return EXIT_SUCCESS;
14 }
```

```
// crea un array di caratteri
// scrive in output il testo fra apici
// memorizza il testo digitato in buff
// scrive in output
// aspetta la pressione di un tasto
// esce dal programma
```



commenti

Le istruzioni di calcolo e di assegnazione

- Con le istruzioni elementari di calcolo ed assegnazione si prescrive il calcolo di una qualche espressione con memorizzazione del risultato.
- Tali istruzioni prevedono che l'esecutore deve prima risolvere un'espressione e, solo quando ne ha prodotto il risultato, assegnare quest'ultimo ad una variabile
- *Es:* *var=5;*

  var=a+5;

I costrutti di controllo - costrutti di sequenza

- I costrutti di controllo indicano all'esecutore l'ordine in cui le operazioni devono essere eseguite. Si dividono in *costrutti di sequenza, di selezione ed iterazione.*
- Il costrutto di controllo più semplice è quello che specifica che due o più operazioni (elementari o no) devono essere eseguite una dopo l'altra. Tali costrutti sono detti **costrutti di sequenza** e vengono indicati inserendo un «;» dopo ogni istruzione

I costrutti di controllo – costrutti condizionali

- Alcune volte i costrutti di sequenza possono prescrivere la selezione di un'operazione
- I **costrutti condizionali** consentono di subordinare l'esecuzione di una certa operazione al verificarsi o meno di una specificata condizione. La notazione tipicamente utilizzata è IF-THEN-ELSE, ma si può anche avere solo IF-THEN

SE (la carta scoperta è quadri) **ALLORA** vinci **ALTRIMENTI** perdi

IF (la carta scoperta è quadri) **then** vinci **else** perdi

SE (hai fame) **ALLORA** mangia	**IF** (hai fame) **then** mangia

I costrutti di controllo – costrutti di selezione

- I **costrutti di selezione** consentono di selezionare una operazione fra più alternative e si esprimono tipicamente con la notazione

NEL CASO (in cui il colore del semaforo) **è** ROSSO: fermati **è** VERDE: passa l'incrocio **è** GIALLO: passa con prudenza o fermati	**CASE** (colore del semaforo) **OF** ROSSO: fermati VERDE: passa l'incrocio GIALLO: passa con prudenza o fermati **END**

I costrutti di controllo – costrutti iterativi

- I **costrutti iterativi e ciclici** prescrivono di ripetere l'esecuzione di una o più operazioni; tale ripetizione viene sospesa al verificarsi di un evento.

RIPETI i compiti **FINCHE'** (suona la sveglia)	**REPEAT** i compiti **UNTIL** (suona sveglia)
MENTRE (piove) **DEVI** usare l'ombrello	**WHILE** (piove) **DO** usa l'ombrello
PER (10 giorni) **DEVI** non venire all'università	**FOR** giorni:=1 **TO** 10 **DO** non venire all'università

Costrutti iterativi

Costrutto iterativo ciclico

- In generale i costrutti ciclici consentono anche di specificare il passo nel caso sia diverso da 1 con una notazione del tipo:

for i:=vp **to** vg **STEP** v **do** azione
for i:=vg **downto** vp **STEP** v **do** azione

Costrutti equivalenti (1/3)

- Per quanto riguarda la selezione è sempre possibile ricondurre un costrutto **if-then-else** ad una sequenza di soli **if-then.**

```
if      (condizione)
then    azione 1
else    azione 2
```

```
if      (condizione)
then    azione 1
if      (not condizione)
then    azione 2
```

Attenzione: sono equivalenti solo se azione1 non cambia il valore di condizione!!

- La struttura **case** può essere ricondotta ad un insieme di **if** disposti l'uno dentro l'altro

```
case (espressione) of
        a: azione 1
        b: azione 2
        c: azione 3
end;
```

```
if (espressione=a)
then azione 1
else
     if (espressione=b)
     then azione 2
     else
          if (espressione=c)
          then azione 3
```

Costrutti equivalenti (2/3)

- Per quanto riguarda le due strutture iterative **while** e **repeat** si noti che è sempre possibile ricondurre l'una all'altra.
- Il while prescrive prima la valutazione della condizione e dopo l'esecuzione delle azioni qualora il valore ottenuto sia vero, mentre con il repeat la condizione viene valutata dopo e quindi le azioni vengono fatte almeno una volta

repeat azione **until** (condizione)	Azione **while** (**not** condizione) **do** azione

while (condizione) **do** azione	**repeat** **if** (condizione) **then** azione **until** (**not** condizione)

Costrutti equivalenti (3/3)

- Anche la struttura ciclica **for** è riconducibile ad una iterativa **while** o **repeat**
- Un ciclo iterativo prescrive la ripetizione di azioni un numero di volte fissato a priori e determinato dal fatto che una variabile detta *contatore* di ciclo, a partire da un valore iniziale raggiunge un valore finale o per valori crescenti (indicato dal to) o decrescenti (indicato dal downto).

for i:=vp **to** vg **do** azione	i:= vp; **while** i≤vg **do begin** azione; i:= i +1 **end**

for i:=vg **downto** vp **do** azione	i:= vg **while** i≥vp **do begin** azione; i:=i-1 **End**

La modularità: sottoprogrammi

- Per facilitare la scrittura e la comprensione di un programma è utile individuare al suo interno dei **moduli funzionali**, ciascuno dei quali realizza un singolo e ben preciso compito e ha un solo punto di ingresso e di uscita.
- Ogni sotto-problema viene risolto da un pezzo di programma detto **sottoprogramma**
  - Un sottoprogramma ha un **nome**, con cui può essere richiamato all'interno di un programma
  - Se il sottoprogramma fornisce un risultato, allora si chiama **funzione**, altrimenti si chiama **procedura**

Indicazione di un sottoprogramma

- L'*indicazione* o *definizione* di un sottoprogramma si compone di due parti: il **titolo** e il **corpo**.
  - Il titolo comprende:
    - il nome del sottoprogramma
    - i parametri di ingresso con il loro tipo (tipicamente racchiusi fra parentesi tonde)
    - il risultato con il suo tipo (una funzione ritorna un solo risultato)
  - Il corpo comprende una sequenza di dichiarazioni e di istruzioni tipicamente racchiusa fra parentesi graffe { }
- Una volta definito, è possibile richiamare il sottoprogramma utilizzando il suo nome e fornendo i parametri di ingresso, se presenti.
  - L'invocazione del sottoprogramma è un'istruzione come le altre
  - Evita di riscrivere più volte lo stesso insieme di istruzioni laddove queste vadano ripetute
  - Consente di concentrarsi su un problema di piccole dimensioni rispetto al programma principale

Parametrizzazione del codice (1/2)

- L'utilizzo dei sottoprogrammi è particolarmente vantaggioso quando le operazioni da essi effettuate posso essere «adattate» su dati diversi

```
{
    stampa('Dammi riempimento:');
    leggi(riemp)
}
```

```
{
    stampa('Numero fogli :');
    leggi(num_fogli)
}
```

NOTA: stampa() e leggi() sono a loro volta dei sottoprogrammi; riemp e num_fogli sono variabili che devono essere dichiarate

- I due sottoprogrammi differiscono solo per le costanti da stampare e per le variabili di cui si devono leggere i valori

```
{
    stampa(messaggio);
    leggi(x)
}
```

- *messaggio* è una variabile che contiene una stringa che deve essere passata al sottoprogramma (input);
- *x* è una variabile dove verrà salvato un numero fornito tramite tastiera (input)

Parametrizzazione del codice (2/2)

- Si definiscono **parametri formali** i parametri utilizzati nel corpo del sottoprogramma (*messaggio* e *x* dell'esempio); essi vanno indicati nel titolo del sottoprogramma al momento della definizione del sottoprogramma.
- Quando il sottoprogramma viene richiamato all'interno di un altro programma, ai parametri formali verranno sostituiti i **parametri effettivi**
- **I parametri effettivi e quelli formali devono essere dello stesso tipo e devono essere specificati nello stesso ordine!**

DEFINIZIONE:

```
stampa_valore (messaggio, x){
  stampa(messaggio);
  leggi (x);
}
```

INVOCAZIONE:

```
stampa_valore('Dammi il riempimento:',riemp); oppure
stampa_valore('Numero di fogli:',num_fogli);
```

Sostituzione dei parametri formali (1/3)

- I parametri effettivi possono essere sostituiti a quelli formali secondo 3 diverse modalità: *per valore, per riferimento* o *per nome*.
- **Passaggio per valore**: al parametro formale viene assegnato il valore del parametro effettivo
  - Il parametro effettivo può essere un'espressione e, come casi particolari di espressione, una costante o una variabile. Se è un'espressione, se ne calcola il valore e lo si assegna al corrispondente parametro formale.
  - Nel caso di variabili, la sostituzione per valore garantisce che la variabile passata come parametro effettivo non venga alterata (il sottoprogramma lavora sulla sua copia)
  - All'atto della chiamata viene fatta la copia del parametro effettivo con conseguente consumo di tempo e spazio

Sostituzione dei parametri formali (2/3)

- **Passaggio per riferimento**: al parametro formale viene assegnato l'indirizzo del parametro effettivo
  - Il parametro effettivo può essere solo una variabile
  - Il sottoprogramma lavora direttamente sulla variabile in memoria
  - Il parametro effettivo occupa solo lo spazio necessario per contenere un indirizzo
- **Passaggio per nome**: il parametro formale viene testualmente sostituito dai parametri effettivi senza alcuna valutazione
  - Durante l'esecuzione del programma ogni volta che compare il parametro formale esso viene sostituito al quello effettivo (è come se si riscrivesse il programma)
  - Questo metodo è ormai presente solo in pochissimi linguaggi

Sostituzione dei parametri formali (3/3)

- Come scegliere la politica di sostituzione?
  - quando un parametro rappresenta un argomento di un sottoprogramma, si sceglie la sostituzione per valore;
  - quando un parametro rappresenta invece un risultato, oppure un dato che viene passato al sottoprogramma e viene modificato da esso per poi essere restituito in uscita, occorre usare la sostituzione per riferimento.

Il programma principale

- Si chiamerà **programma principale** quella unità di programma che si interfaccia direttamente con il sistema operativo.
- In molti linguaggi il programma principale si chiama **main**
  - La funzione main è il punto da cui comincia l'esecuzione, indipendentemente dalla posizione nel listato
  - La funzione main può prendere dei parametri in ingresso (che servono per interagire col sistema operativo) e restituisce solitamente un valore intero, che indica l'esito dell'esecuzione (codice di errore)
  - **Tutti i programmi devono contenere la funzione main!!**

Visibilità

- L'uso dei sottoprogrammi permette di costruire una gerarchia di *unità* di programma, ciascuna delle quali realizza una particolare operazione in modo autonomo, con proprie definizioni di tipi, dichiarazioni di variabili e istruzioni
- Spesso, certe variabili vengono impiegate soltanto all'interno di una sequenza di istruzioni, mentre non hanno influenza nel resto del programma
- Le **regole di visibilità** (o di **scope**) consentono di determinare i campi di influenza di una variabile (e più in generale di un qualunque oggetto) in tutte le varie parti costituenti un programma.

Visibilità (1/2)

- Se un oggetto (una costante, un tipo, una variabile, o un sottoprogramma) è definito ed usato solo all'interno di un determinato sottoprogramma, allora viene detto **locale** a quel sottoprogramma
- Se, viceversa, è definito in un'unità di programma chiamante, ma risulta visibile al sottoprogramma chiamato attraverso qualche meccanismo, allora viene detto **non locale** al sottoprogramma

Visibilità (2/2)

- Un oggetto viene detto **globale** se risulta definito nel programma principale, perché tale unità è l'unica che può rendere visibili i propri oggetti a tutte le altre in quanto rappresenta la radice della gerarchia di unità di programma
  - Oggetti con lo stesso scope non possono chiamarsi allo stesso modo, mentre non esiste alcun problema se si sceglie per un oggetto locale ad una unità di programma lo stesso identificatore utilizzato per un altro oggetto di una diversa unità di programma.
  - Gli oggetti locali a un sottoprogramma vengono allocati in memoria solo quando il sottoprogramma viene richiamato, e all'uscita da esso vengono deallocati (**allocazione dinamica**)
  - Le variabili definite dal programma principale sono invece globali e **statiche**, nel senso che esistono in memoria dall'inizio alla fine del programma principale