Linguaggi di Programmazione
Corso C

Parte n.10
Gestione della Memoria

Nicola Fanizzi (*fanizzi@di.uniba.it*)

Dipartimento di Informatica
Università degli Studi di Bari

Gestione della Memoria

I moderni linguaggi di programmazione forniscono strumenti per la decomposizione funzionale del programma in sottoprogrammi

- procedure
- funzioni

La definizione di una procedura (o funzione) associa un identificatore ad una sequenza di istruzioni e dichiarazioni:

- parametri
- dati locali
- procedure locali
- (tipo risultato)

Procedure

Effetti collaterali

accesso di una procedura a dati non locali ossia definiti all'esterno

Chiamata

accesso ad una procedura da parte del programma principale o da altra procedura (*attivazione*)

- Una procedura P *chiama direttamente* una procedura Q sse in P compare una chiamata di Q
- Una procedura P *chiama indirettamente* una procedura Q sse in P compare una chiamata della procedura R che chiama (direttamente o indirettamente) Q
- una procedura si dice *direttamente* (risp. *indirettamente*) *ricorsiva* sse chiama direttamente (risp. indirettamente) se stessa
- una procedura P si dice *locale* a Q sse è definita nell'ambito delle dichiarazioni di Q

Esempio

```
Program POSTFIX(input,output);
var A,B: array [1..80] of char; I: integer;
   procedure S;
   var CH: char; J,L: integer;
      procedure E;
      var CH: char;
         procedure T;
         var CH: char;
            procedure F;
            var CH: char;
            begin {F}
               I:=I+1; CH:=A[I];
               if CH='(' then E
               else begin J:=J+1; B[J]:=CH end;
               I:=I+1
            end {F}
         begin {T}
            F; CH:=A[I];
            while (CH='*') or (CH='/') do
               begin F; J:=J+1; B[J]:=CH; CH:=A[I] end
         end {T}
      begin {E}
         T; CH:=A[I];
         while (CH='+') or (CH='-') do
            begin T; J:=J+1; B[J]:=CH; CH:=A[I] end
      end {E}
   begin {S}
      I:=0; J:=0; E;
      writeln('Forma Polacca Inversa:');
      for L:=1 to J do write(B[L]); writeln;
   end {S}
```

```
begin {main}
   writeln('Espressione ?  :'); I:=0;
   repeat
      I:=I+1; read(A[I])
   until A[I] = '.';
   readln;
end {main}
```

Alberi di Attivazione

Le procedure chiamate durante l'esecuzione di un programma possono essere disposte in una struttura detta **albero di attivazione**

- nodo $\mathbf{X}$: attivazione di una procedura $\mathbf{P}(\mathbf{X})$
- radice: attivazione del main
- $\mathbf{X}$ ha padre $\mathbf{Y}$ sse
  $\mathbf{P}(\mathbf{Y})$ determinata da $\mathbf{P}(\mathbf{X})$
- $\mathbf{X}$ a sinistra di $\mathbf{Y}$, avendo lo stesso padre, sse
  $\mathbf{P}(\mathbf{Y})$ successiva a $\mathbf{P}(\mathbf{X})$
- un percorso $\mathbf{N_0}, \mathbf{N_1}, \ldots, \mathbf{N_j} = \mathbf{X}$ dalla radice ad $\mathbf{X}$ indica la sequenza che porta all'attivazione di $\mathbf{P}(\mathbf{X})$ significa che il main chiama $\mathbf{P}(\mathbf{N_1})$ che chiama $\mathbf{N_1}$ e così via fino alla chiamata di $\mathbf{P}(\mathbf{N_{j-1}})$ che causa l'attivazione di $\mathbf{P}(\mathbf{X})$
- $\mathbf{P}(\mathbf{X})$ e $\mathbf{P}(\mathbf{Y})$ sono concomitanti se $\mathbf{X}$ e $\mathbf{Y}$ appartengono allo stesso percorso dalla radice ad un certo nodo $\mathbf{Z}$

Esempio (continua)

espressione: 2 * (3 + 4)

```
POSTFIX
    |
    S
    |
    E
    |
    T
    ⟼ F
    |
    ⟼ F
        |
        E
        ⟼ T
        |   ⟼ F
        |
        ⟼ T
            ⟼ F
```

Variabili

Il concetto di variabile astrae quello di cella di memoria e l'assegnazione ne astrae la modifica

Una variabile sono caratterizzate da:

visibilità (scope): insieme di istruzioni dalle quali la variabile è visibile e quindi manipolabile
dipende dalle regole del linguaggio

durata (lifetime): periodo di tempo in cui esiste un'area di memoria associata alla variabile per contenerne il valore

tipo (type): classe dei valori (e loro rappresentazione) e delle funzioni di manipolazione specifiche

valore (value): codificato nella rappresentazione associata e che dipende dal tipo

Visibilità e durata spesso coincidono (per le variabili statiche):
le variabili locali ad una procedura vengono allocate all'attivazione della procedura e deallocate in uscita;
Per le variabili dinamiche allocazione e deallocazione sono esplicite ed operate tramite puntatori

Organizzazione della Memoria

Aree associate ad un programma:

codice oggetto generato dal compilatore + eventuali librerie

variabili statiche del programma principale
di durata pari a tutta l'esecuzione del programma

attivazioni delle procedure organizzata a stack

variabili dinamiche organizzata ad heap

Dimensioni

- Per il codice oggetto le dimensioni sono fisse e note in fase di compilazione (*compile-time*)
- Analogamente per le variabili statiche
- Stack ed Heap: dimensioni dipendenti dalla particolare esecuzione
  Allocati ai due estremi di uno stesso blocco di memoria fisica
  Due puntatori segnalano il livello di riempimento delle due aree.
  Se durante l'esecuzione c'e' sovrapposizione,
  il programma s'interrompe per memoria insufficiente
- Le informazioni necessarie all'esecuzione di una procedura sono contenute in un blocco di attivazione

Blocco di Attivazione del Sottoprogramma P

risultato: nel caso P sia una funzione

parametri attuali: area che riceve i parametri che vengono passati a P

legame di attivazione (*control link*): puntatore al blocco di attivazione della procedura chiamante

legame di accesso (*access link*): puntatore che permette l'accesso alle variabili non locali contenute in altri blocchi di attivazione

dati locali di P

dati temporanei risultati intermedi che non trovano posto nei registri del processore

Dimensioni note a compile time tranne il caso di array locali dimensionabili a seconda dei parametri passati

Chiamata di sottoprogramma

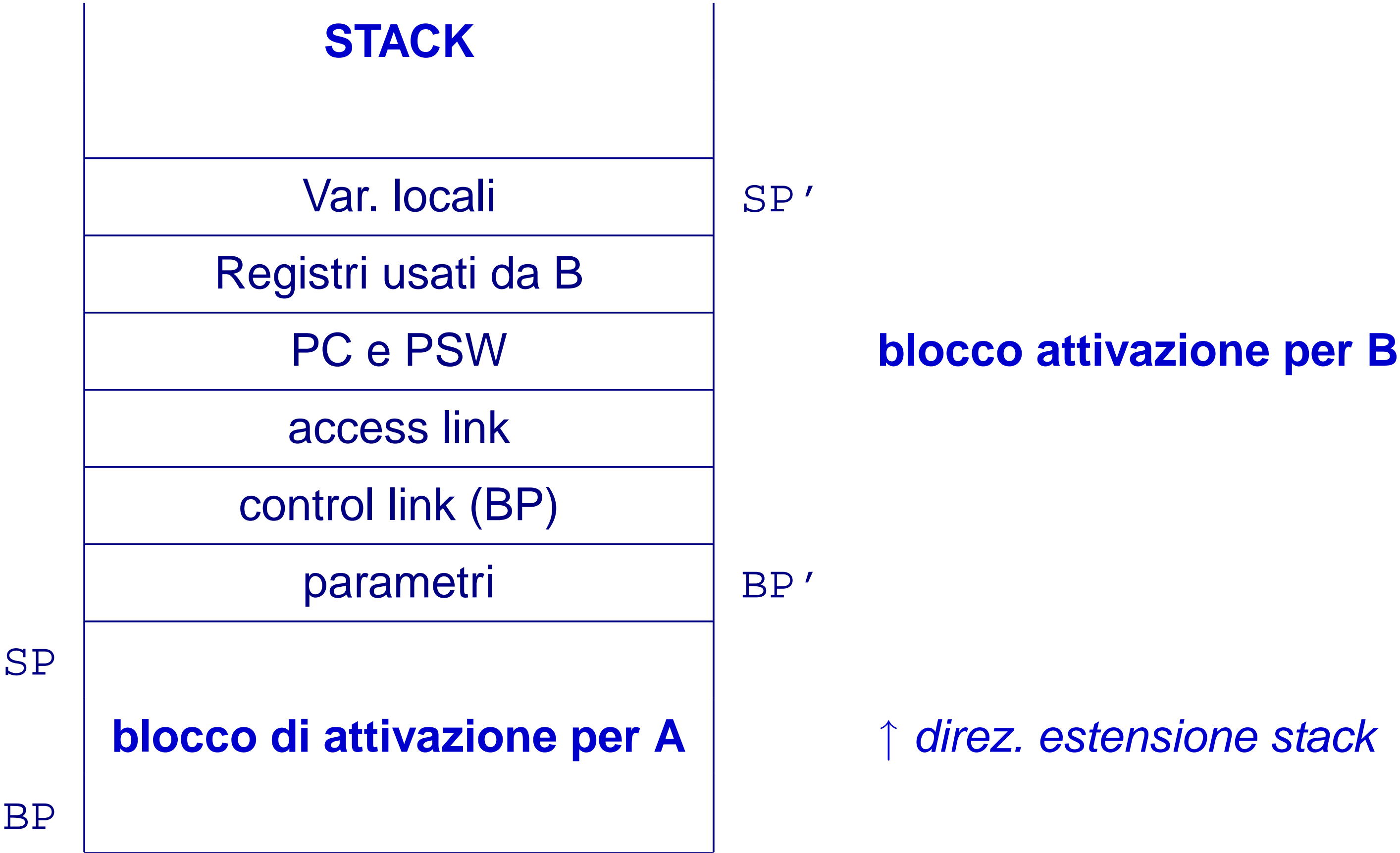
sequenza di chiamata realizza la chiamata del sottoprogramma allocando un blocco di attivazione e riempiendone i campi

sequenza di ritorno ripristina la situazione precedente alla chiamata, deallocando le risorse impegnate

Dipendono da

- linguaggio di programmazione
- sistema operativo
- processore

Esempio



Allocazione Variabili

Assegnazione di un’area di memoria

statica avviene ad inizio esecuzione del programma
tipica per le variabili del main

dinamica avviene nel corso dell’esecuzione del programma
tipica per le variabili locali e variabili puntate

La quantità di memoria dipende dal tipo della variabile:

Il compilatore in base al tipo determina l’offset della variabile rispetto ad un indirizzo base

L’indirizzo fisico sarà costituito da una combinazione base+offset

La rappresentazione scelta per la variabile dipende dal processore

Accesso a Variabili

Il problema si presenta con l'accesso a variabili non locali

- blocchi
- procedure non definibili come locali ad altre procedure (come in C)
- procedure locali ad altre procedure (come in Pascal)

Blocco

Implementabili mediante stack

Visibilità di variabili:

1. V dichiarata in un blocco B è visibile a tutte le istruzioni di B ed anche a tutte le istruzioni di blocchi B1 in B che non ridichiarino V
2. se V è visibile in B1 ma non dichiarata in B1 allora è dichiarata in un blocco B che contiene B1
3. se B1 contenuto in B dichiara V ma anche B dichiara V allora questa copia esterna viene oscurata dalla copia dichiarata in B1 per le istruzioni di B1 ed anche in tutti i blocchi contenuti in B1

No Annidamento di Procedure

variabili globali: accessibili dappertutto;
allocazione statica in aerea apposita

variabili locali: accessibili solo nella procedura dichiarante;
allocazione nel blocco di attivazione della procedura

Annidamento di Procedure

Una procedura P1 può contenere (direttamente) una procedura P2 se questa si trova nelle sue dichiarazioni
Questa relazione è transitiva

Regole di visibilità delle variabili come per i blocchi

Regole di visibilità delle procedure

1. P può chiamare se stessa e tutte le procedure Q che la contengono
2. P può chiamare tutte le T che contiene direttamente
3. P può chiamare R che non la contiene, purchè R sia contenuta in Q che contiene P (e la dichiarazione di R preceda la definizione di P)

Livello = Grado di località di una procedura:

- livello(main)=1
- livello(Q) = livello (P)+1 se Q contenuta in P

Variabili (non locali) → (livello,offset)

Metodi d'accesso

alla variabile V definita in P:

catena statica: si utilizzano gli access link dei blocchi d'attivazione che corrispondono agli indirizzi primari del blocco

- se V è locale il valore è reperito sullo stack sommando l'offset all'indirizzo primario del blocco
- se V non è locale il valore è reperito sullo stack sommando l'offset all'indirizzo primario della procedura W che dichiara V;
  questo si determina seguendo la catena di blocchi per un numero di passi pari a: livello(W)-livello(P)

vettore d'accesso: mantiene un numero di puntatori di puntatori a blocchi di attivazione pari al livello massimo
l'indirizzo di una variabile non locale si calcola mediante l'offset e calcolando l'indirizzo primario del blocco che la contiene;
questo si trova nel vettore d'accesso alla posizione corrispondente al livello di P

Passaggio di Parametri

per valore (by value):

vengono calcolati i parametri effettivi e i loro valori sono copiati nell'area dei parametri del blocco di attivazione della procedura chiamata

per indirizzo (by reference):

la procedura chiamante copia l'indirizzo referenziato nell'area parametri del blocco di attivazione della procedura chiamata.

Lo spazio occupato per mantenere questa informazione è costante

procedure (possibile in taluni linguaggi)

Per alcuni linguaggi basta un puntatore alla prima istruzione del codice della procedura;

in linguaggi come il Pascal si fornisce anche il legame d'accesso

Allocazione Dinamica

Le variabili allocate dinamicamente sono sistemate nell'**heap**
e vi rimangono fino a quando non vengono esplicitamente deallocate
(ad es. con `new` e `dispose` in Pascal e `malloc()` e `free()` in C)

Problemi:

- *recupero memoria inutilizzata*:
  parte della memoria allocata dinamicamente può diventare irraggiungibile quando non esiste alcun percorso attraverso le variabili dinamiche per raggiungere un'area di memoria precedentemente allocata
  - In alcuni linguaggi come Lisp e Java c'è il recupero automatico (garbage collection)
  - Gli altri (C, Pascal) lasciano il compito al programmatore
- *controllo consistenza riferimenti*:
  la deallocazione esplicita può produrre riferimenti inconsistenti (variabili che puntano ad aree deallocate delle memoria)

Gestione dell'Allocazione Dinamica

- Blocchi a lunghezza Fissa:
  Heap gestito come contenitore di blocchi di lunghezza fissa
  Viene gestita una lista dei blocchi liberi
- Blocchi a lunghezza Variabile:
  - All'inizio l'intero heap è disponibile come unico blocco libero
  - la procedura `GETAREA` alloca l'area di memoria richiesta tenendone traccia in una lista dei blocchi liberi di cui si mantengono i puntatori ad inizio e fine lista
    la politica di allocazione si dice *first-fit*:
    si impiega il primo blocco libero di dimensioni maggiori della memoria richiesta spezzandolo in due parti
  - la procedura `FREEAREA` rilascia il blocco allocato in base al suo indirizzo: fondendolo con il blocco contiguo precedente (e/o quello successivo) quando questo risulti libero
  - l'heap risulta durante l'esecuzione frammentato in blocchi liberi e blocchi allocati
- Deallocazione Implicita:
  può essere effettuata automaticamente (fine memoria) o ad intervalli regolari
  Tecniche:
  - contatore d'uso: a ciascun blocco è associato un contatore dei puntatori che vi fanno riferimento;
    un blocco con contatore pari a 0 potrà essere deallocato
  - marcatura: periodicamente si sospende l'esecuzione del programma per marcare i blocchi raggiungibili mediante i riferimenti attivi in quell'istante; i blocchi non marcati vanno deallocati