11 - Programmazione modulare

La programmazione modulare è una tecnica basata sul metodo di scomposizione di un problema in sotto problemi logicamente indipendenti tra loro. Ad ogni sotto problema corrisponde un modulo, che viene codificato e compilato separatamente, venendo integrato solo alla fine per formare il programma complessivo

La programmazione modulare dice che operiamo su un problema caratterizzato da

- Un algoritmo A
- Che opera sull'insieme dei dati di partenza D
- Per produrre l'insieme dei risultati R
  Viene suddiviso in un insieme finito di n sotto problemi a differenti livelli dalla tripla (D_i, A_i, R_i)

Si tratta quindi dell'interazione e esecuzione degli algoritmi secondari per ottenere la soluzione del problema originario gestita da un algoritmo coordinatore

Differenziamo i due algoritmi principali:

- **Coordinatore**, il main del programma
- **Secondari**, i sotto programmi, che si contraddistinguono in diversi livelli in una gerarchia di macchine astratte ciascuna delle quali
  - Realizza un particolare compito in modo completamente autonomo (ha delle proprie definizioni di tipi, dichiarazioni di variabili e istruzioni)
  - Fornisce la base per il livello superiore
  - Si appoggia su un livello macchina inferiore (se quest'ultimo esiste)
  La **macchina inferiore** rispetto a quella astratta è la base concreta su cui quest'ultima si baserà. È reindirizzabile al linguaggio macchina o assembly che viene eseguito dal processore.

Macchina concettuale (Notional Machine)

La macchina concettuale è un computer idealizzato e concettuale le cui proprietà sono implicate dai costrutti nel linguaggio di programmazione utilizzato, quindi diverse da ogni tipo di linguaggio (dal C al Python cambiano)
Ogni volta che si crea un programma si genera una macchina virtuale/astratta/concettuale, la quale dipende dal linguaggio utilizzato e definisce come a macchina utilizzerà i dati presenti nel codice, poiché ogni linguaggio di programmazione ha una propria macchina concettuale

Astrazione funzionale

L'astrazione funzionale è una tecnica che consente di ampliare il repertorio di operatori disponibili per risolvere problemi specifici.
Ma cosa è un operatore?
Una funzione o un sottoprogramma che esegue una determinata operazione sui dati e produce un risultato

Le astrazioni funzionali sono fornite dal linguaggio di programmazione ad alto livello e prendono nomi differenti a seconda del linguaggio usato, il loro scopo è creare diverse macchine astratte, dando un nome ad un gruppo di istruzioni e stabilendo il metodo per comunicare tra macchina astratta e il resto del programma

Tecniche per individuare i sotto problemi

Le diverse tecniche per individuare i sotto problemi consistono nello specifico allo scomporre un problema in sotto problemi più semplici, le tecniche sono:

- Sviluppo top-down (con approccio step-wise refinement o per raffinamento per passi successivi e il trial and error, provare costantemente fino alla ricerca della scomposizione ottimale)
- Sviluppo bottom-up (usato sopratutto nell'adattamento di algoritmi codificati già esistenti a nuove situazioni)
- Sviluppo "a sandwich" (basato sulla cooperazione tra top-down e bottom-up)
  Questi argomenti sono già stati spiegati nella lezione 5- Tipi di problemi e scomposizione

Sottoprogramma

Il sottoprogramma corrisponde ad un algoritmo secondario che risolve un sotto problema, è identificato da un nome e concorre alla risoluzione al problema minore in supporto di problemi più complessi, ben definito e non necessariamente fine a se stesso.
Il sottoprogramma prevede l'uso di risorse come variabili, è costituito da istruzioni semplici e composte e specifica come altri pezzi del programma possono utilizzarlo

Il **sottoprogramma** quindi è un'astrazione funzionale che consente di individuare gruppi di istruzioni che possono essere invocate esplicitamente e la cui chiamata garantisce che il flusso di controllo ritorni al punto successivo all'invocazione.

Utilità

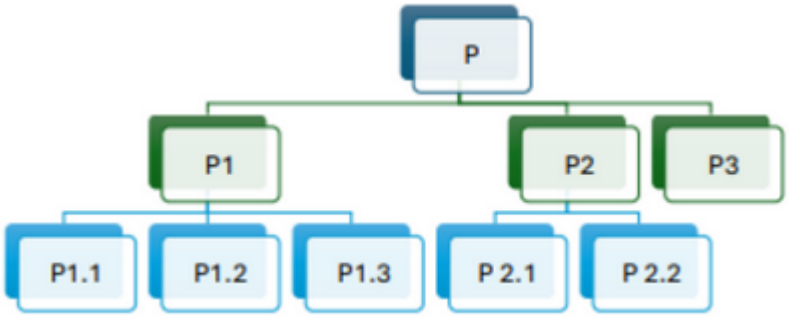
Un programma viene diviso in sotto programmi per rispettare la decomposizione creata nella fasi di progettazione dell'algoritmo ma specialmente perché lo stesso gruppo di istruzioni che rappresentano il sottoprogramma potrebbero essere ripetute più volte durante il corso del programma principale, per questo si definiscono blocchi ripetibili.

Chiamata

La chiamata di un sottoprogramma da parte del programma provoca l'esecuzione delle sue istruzioni;
All'atto dell'attivazione dell'unità (alla chiamata nel main o in altro sotto programma) viene sospesa l'esecuzione del programma chiamante e il controllo passa all'attività del sotto problema, al completamento poi dell'esecuzione l'attività termina portando il flusso di controllo del programma chiamante.

Nidificazione

Le risorse che usa un sottoprogramma possono includere altri sottoprogrammi , questo porta alla creazione di una gerarchia tra i programmi che realizza una struttura ad albero con relazione **padre-figlio**



Comunicazione

I sottoprogrammi tra di loro comunicano attraverso lo scambio di dati e può avvenire con:

- L'ambiente **esterno** (con l'utente in output/input)
- Con l'ambiente **chiamante** in modo:
  - Implicito(sconsigliato) attraverso variabili globali
  - Esplicitamente(consigliato) attraverso l'uso dei parametri locali che inserirà in input dal programma chiamante e che verranno tramutate in output dal sottoprogramma.
  La comunicazione esplicita avviene sempre per un buon programma.

Variabili

Ciascun sottoprogramma può utilizzare le proprie variabili locali (tra cui quelle interne al sottoprogramma e quelle temporanee all'avvio del sottoprogramma, queste ultime saranno distrutte alla fine del sottoprogramma per liberare spazio in memoria), le variabili globali, poiché definiti nel programma principale e le variabili dichiarate dai sottoprogrammi attualmente in esecuzione, questo fenomeno prende il nome di visibilità dipendente.
La visibilità dipendente può essere di due tipi:

- **Statica**:
  La variabile statica dipende dalla struttura gerarchica delle dichiarazioni del sottoprogramma, ovvero da dove e come sono dichiarati i sottoprogrammi nel codice sorgente. Il compilatore determina quali variabili sono visibili e accessibili in base alla loro posizione nel codice. Avviene in fase di compilazione, per ciò definita statica.

- **Dinamica**
  La visibilità dinamica dipende dall'ordine di chiamata dei sottoprogrammi durante l'esecuzione del programma.
  Le variabili accessibili dipendono da quali sottoprogrammi sono stati chiamati e in che ordine, viene determinato durante la fase di esecuzione, per questo è definita dinamica.

Come si evince anche i sottoprogrammi restituiscono un risultato e se ne deve specificare il tipo richiesto, il più comune e presente negli esempi precedenti è la procedura `void`.
La procedura `void` è un valore di un sottoprogramma in cui non esiste area di memoria in cui sarà restituito un risultato, non è presente un allocazione di memoria per essa.

Shadowing (Oscuramento)

Diversi sottoprogrammi possono dichiarare risorse aventi lo stesso nome ma totalmente non correlate tra loro, potendo avere anche tipo differente.
È necessario verificare determinare quale variabile con lo stesso nome si sta facendo riferimento in un dato momento, la variabile considerata quindi sarà quella che è più vicina nel contesto di dichiarazione, ovvero la variabile dichiarata nel contesto più interno nella gerarchia dei sottoprogrammi sarà quella accessibile per il sottoprogramma;
Un altro fattore che determina la scelta della variabile considerata sarà l'ordine in cui i sottoprogrammi vengono chiamati.

Visibilità dei sottoprogrammi

Un sottoprogramma deve rispettare delle regole per i suoi identificatori e per la loro visibilità:

- Un identificatore è visibile nel sottoprogramma in cui è dichiarato e in tutti i sottoprogrammi locali ad esso nei quali è stato ridichiarato
- Le risorse di un sottoprogramma devono essere dichiarate prima di essere usate (come un programma normale)
- Le variabili globali possono essere viste da main e sottoprogrammi
- Una risorsa (variabile) locale è visibile ad un sottoprogramma P solo dalle istruzioni di P e dagli eventuali sottoprogrammi che sono stati definiti in P

Delimitatori

Esistono due tipo di delimitatori:

- **Delimitazione spaziale di una risorsa:** La zona o campo di visibilità di codice in cui nel programma è possibile fare riferimento ad un identificatore è data dalle regole di visibilità
- **Delimitazione temporale di una risorsa:** Le regole di **visibilità** definiscono anche quanto tempo rimane in vita una variabile, ovvero il tempo in cui un area della RAM è allocata per essa. Per questo la fase in cui è definita un'area di memoria per una

variabile viene detta **fase di allocazione**. Normalmente il tempo di durata di una variabile è pari alla durata d'esecuzione del programma in cui è definita.

Una variabile quindi presenta esattamente 6 caratteristiche:

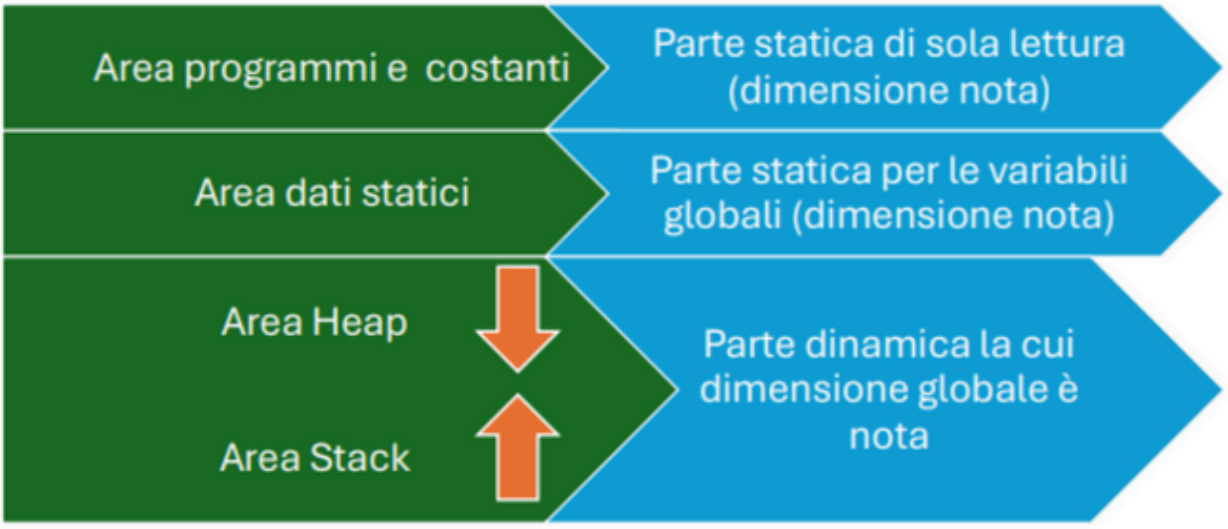
- Nome
- Valore
- Indirizzo
- Tipo
- Ambito
- Durata

Durata e ambito di una variabile

Ogni volta che si definisce una variabile si stabilisce:

- La durata della sua vita con la corrispettiva creazione e distruzione attraverso particolari istruzioni
- La sua visibilità (se globale, locale o estesa ad altri blocchi)

Allocazione delle variabili nella RAM



L'**area di stack** è utilizzata per memorizzare il record di attivazione dei programmi, costituito da:

- Indirizzo del codice dell'istruzione successiva
- Parametri del sottoprogramma
- Variabili locali al sottoprogramma

L'area di memoria Heap è utilizzata per l'allocazione dinamica, per questo più grande e lenta di quella stack, essa rimane valida finché non viene deallocata.

Parametri

Come abbiamo visto negli esempi precedenti, ci sono diversi modi di passare i dati tra un sottoprogramma e l'altro, questo fenomeno viene definito **passaggio di parametri**, esso si divide in diverse tipologie:

- Parametri **formali**: Sono il **segnaposto** per indicare simbolicamente gli oggetti (con il loro tipo e struttura) su cui il sottoprogramma lavora, i loro nomi compaiono nell'intestazione del sottoprogramma e non hanno legami con nomi usati altrove. Sono specificati all'atto della definizione del sottoprogramma, sono quindi simbolici e consentono di definire i dati e i loro tipi che devono essere passati al sottoprogramma quando sarà invocato. All'atto della chiamata vengono sostituiti dai parametri effettivi, ovvero i dati su cui effettivamente il sottoprogramma adopererà.
- Parametri **attuali o effettivi**: Sono i parametri su cui il sottoprogramma lavorerà e coincidono con i parametri formali. L'esecuzione dell'istruzione di chiamata comporta la sostituzione dei parametri formali con quelli reali.

Tipi di passaggio

Esistono vari tipi di passaggio di parametri:

- **Per valore**: Si calcola il valore del parametro reale e lo si sostituisce al corrispondente formale, è usato generalmente per parametri che rappresentano un argomento e non il risultato di un sottoprogramma.
- **Per referenza**: Il parametro effettivo è una variabile ed ha a disposizione una locazione di memoria il cui indirizzo viene passato al parametro formale. Usato più spesso quando il parametro rappresenta il risultato e ha dimensioni notevoli.
- **Per nome (o valore)**: Il nome del parametro formale viene sostituito col nome del parametro reale

Supponiamo di avere un sottoprogramma P a cui passiamo un parametro formale pf e viene chiamato con parametro attuale pe.

- Per valore: pf si comporta come una variabile locale a P
- Per riferimento: Al momento dell'attivazione di P viene calcolato l'indirizzo di pe, e pf viene creato con riferimento alla stessa locazione di memoria.
- Per nome: Al momento dell'attivazione di P il valore di pe viene calcolato e memorizzato in una nuova locazione assegnata a pf.

NB: Il passaggio per nome non esiste in C.

Pro e contro del passaggio di parametri

Metodo	Pro	Contro
Valore	- Permette la trasmissione del valore di un parametro dal chiamante	- Aumenta l'occupazione di memoria e il tempo
	- Consente la separazione tra programma chiamante e programma chiamato	- Rende difficile la gestione di parametri di dimensione variabile
Riferimento	- Evita problemi di passaggio grazie al trattamento degli indirizzi gestito dal compilatore	- Causa effetti collaterali spesso imprevedibili
	- Non occupa memoria aggiuntiva	