Introduzione e concetti base

sabato 26 settembre 2020 00:28

Cos'è un computer?

Computer: one that computes; specifically: a programmable usually electronic device that can store, retrieve, and process data [Merriam-Webster Dictionary]

Chi è il programmatore?

- Il programmatore analizza un problema e identifica la serie di passi che il computer dovrà eseguire
- E' suo compito descrivere i passi in modo che il computer possa comprenderli (e poi eseguirli)

Cosa significa programmare?

- *programmare*: pianificare lo svolgimento di un'azione o un evento;
- *programmare un computer*: pianificare una sequenza di passi che un computer possa eseguire;
- *programma*: una sequenza di passi che un computer è in grado di eseguire

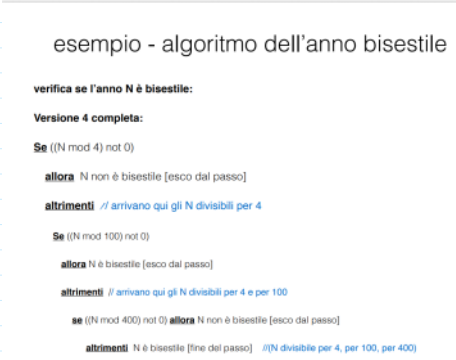
Caratteristiche di un 'buon programma':

• rispondere alle specifiche (fare quello che deve)
• correttezza e completezza
• efficienza ed economia di uso delle risorse
• scalabilità
• robustezza, affidabilità, tolleranza ai guasti

Come porsi davanti a un problema: l'algoritmo

Con uno o più metodi si cerca un procedimento per la risoluzione del problema:

- **Algoritmo**: una procedura in più passi per la risoluzione di un problema in una quantità di tempo finita
- *Algoritmo e programma* hanno definizioni molto simili. In termini generali potremmo dire che il programma è un algoritmo "riscritto per il computer"



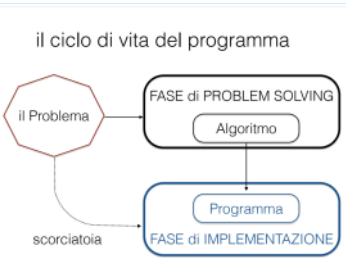
Problem Solving

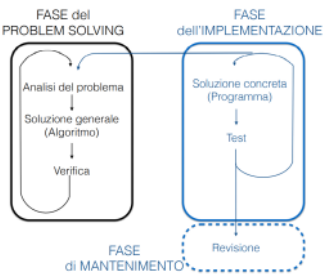
"Problem solving consists of using generic or ad hoc methods, in an orderly manner, for finding solutions to problems" - (Wiki en).

Pensiero computazionale

Coinvolge una serie di abilità di problem solving che sono tipiche dell'attività del programmatore:

• *Decomposizione*: spezzare un problema in passi atomici più semplici da trasferire ad un altro "agente" (persona o computer)
• *Riconoscimento di pattern*: notare similitudini o differenze con altri problemi/attività noti
• *Generalizzazione e astrazione*: filtrare informazioni non necessarie e ottenere rappresentazioni del problema che siano il più generali possibile (e quindi maggiormente riusabili)
• *Progettazione di algoritmi*: riassumere quanto compreso dai passi precedenti in una procedura passo-passo che aiuti l'agente a risolvere il problema

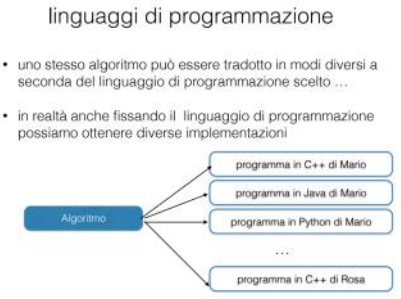




Implementazione e linguaggi di programmazione

L'implementazione consiste nella trasformazione (traduzione) dell'algoritmo in programma.

Quando il programmatore è *soddisfatto* dell'algoritmo scritto, **deve tradurlo in un linguaggio appropriato in modo che il computer possa comprenderlo**:
• un linguaggio di programmazione è un insieme di regole, simboli e parole speciali usati per costruire un programma per il calcolatore
• in prima battuta lo possiamo pensare come una forma molto semplificata e rigida di inglese (con formule matematiche) che segue regole grammaticali molto rigide



I calcolatori e i livelli di astrazione

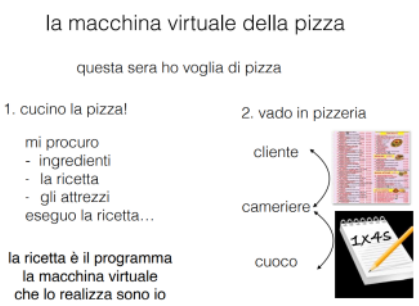
• I sistemi di calcolo (o computer) sono dispositivi elettronici: segnali elettrici si propagano lungo canali al suo interno
• un concetto chiave, che ha permesso di ridurre (o gestire) la grande complessità interna dei computer è la strutturazione in vari livelli di astrazione --> semplificazione tramite stratificazione
• la stratificazione "base" comprende il livello "uomo" (il programmatore che parla il linguaggio L1) e il livello "macchina" (il sistema di calcolo che parla il linguaggio L0)

La macchina virtuale

- nella pratica all'interno di un sistema di calcolo si possono identificare vari livelli
- ogni livello può essere rappresentato in modo astratto tramite il concetto di macchina virtuale (le attività di ogni livello sono delegate a differenti macchine virtuali)
- una macchina virtuale non fa necessariamente riferimento ad una macchina reale — potrebbe essere realizzata da un insieme di dispositivi diversi oppure essere una persona…

Caratteristiche di una macchina virtuale:

- l'alfabeto della macchina virtuale è un insieme di simboli diversi tra loro e riconoscibili, utilizzati dalla macchina virtuale stessa
- il linguaggio della macchina virtuale è definito come l'insieme di tutte le sequenze di simboli dell'alfabeto che identificano comandi eseguibili oppure dati che vengono usati o prodotti dalla macchina virtuale
- la macchina virtuale è in grado di interpretare il linguaggio, ossia manipolare i dati eseguendo i comandi scritti nel linguaggio stesso.
- la macchina virtuale è indipendente da come l'effettiva macchina reale sia realizzata fisicamente



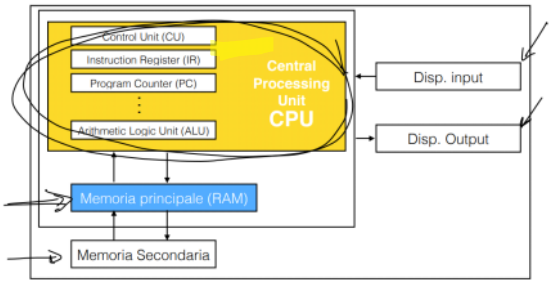
Come è fatto un elaboratore - Von Neumann

sabato 10 ottobre 2020 12:15

Cenni alla Macchina di Von Neumann

- La MVN codifica istruzioni in forma numerica
- Istruzioni e dati vengono inseriti insieme nella RAM
- Questo schema (basata su lavori teorici di Turing) viene usato ancora oggi nella realizzazione di sistemi di calcolo.

Elementi di un sistema di calcolo



- Elementi
  - Unità di elaborazione (CPU) → Unità di controllo, Unità aritmetico-logica, Registri
  - Memoria (RAM)
  - Input / output

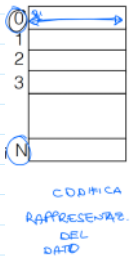
Control Unit

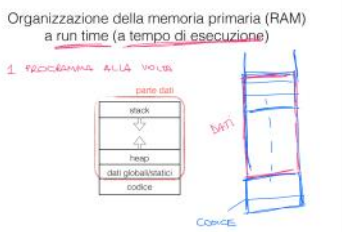
Control Unit (unità di controllo) realizza il funzionamento della macchina. Contiene:

- Due registri che memorizzano valori interi:
  - Accumulatore (ACC)
  - Instruction Register (IR)
- Un registro che memorizza l'indirizzo di una cella della RAM:
  - Program Counter (PC): memorizza l'indirizzo della prossima istruzione

La RAM

- RAM (Random Access Memory, memoria ad accesso arbitrario mediante indirizzo)
- Realizza la memorizzazione di un vettore di numeri interi.
- Sia la dimensione del vettore (numero di elementi componenti) che il massimo valore memorizzabile in ogni elemento del vettore sono predeterminati al momento della costruzione e/o assemblaggio del dispositivo.





Input / Output (I/O)

- Input (unità di **ingresso**) Permette all'utente di interagire con la macchine (per esempio attraverso l'uso di una tastiera numerica) per l'introduzione di valori di tipo intero, uno alla volta.
- Output (unità di **uscita**) Permette alla macchina di stampare in un formato numerico leggibile dall'utente un valore intero per volta.

Realizzazione ed esecuzione programmi

venerdì 2 ottobre 2020 12:02

Struttura dei sistemi di calcolo (stratificazione astratta)

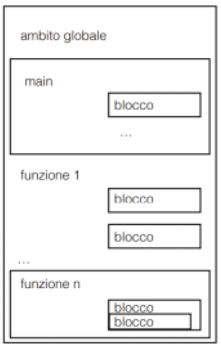
- **L5 Linguaggi di alto livello** (C, C++, Java, ...) - è il livello di macchina virtuale normalmente usato dai programmatori ← IP, ...
- **L4 Assembler e Librerie** - è il livello di macchina virtuale più basso utilizzabile dal programmatore
- **L3 Nucleo del sistema operativo** - permette l'attivazione di processi e l'uso di risorse fisiche del sistema
- **L2 Macchina convenzionale** (microprocessori) - livello di definizione delle istruzioni base del computer

AdC

- **L1 Microarchitettura** - Livello di definizione del funzionamento dei singoli componenti fisici in termini di interconnessione e spostamento di informazioni tra circuiti logici
- **L0 Logica Circuitale** - Livello di realizzazione dei circuiti logici elementari
- **L-1 Elettronica/Fotonica** - Livello di progettazione dei dispositivi fisici — laurea ing elettronica
- **L-2 Fisica dello stato solido** (semiconduttori - quantistica) - progettazione e realizzazione dei circuiti integrati — FISICA

Struttura dei programmi - Programmi sequenziali

- Considereremo programmi sequenziali, nei quali viene eseguita un'istruzione alla volta in sequenza
- tali programmi sono costituiti da un programma principale (il main in C e C++) più eventuali estensioni procedurali (funzioni)



struttura dei programmi C/C++

Stratificazione (semplificata)

Sistema operativo

il sistema operativo (Operating System, OS) è il programma che viene caricato in fase di avvio del sistema (fase di bootstrap)
• l'OS continua a girare e a coordinare l'attività degli altri programmi finche' il sistema è in funzione
• Esso può disporre l'esecuzione "contemporanea" di più programmi in esecuzione (processi) secondo uno schema di condivisione della memoria RAM (per il momento noi ci concentreremo nell'esecuzione di un programma per volta).

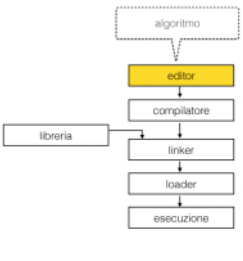
La compilazione

Il compilatore è un programma che prende in input il codice sorgente scritto in un linguaggio di alto livello e lo traduce (compila) in codice eseguibile.

Il processo di compilazione coinvolge un altro programma (il linker) che unisce vari pezzi di codice precompilato, incluse librerie e altri moduli sviluppati dal programmatore

ELABORAZIONE DI UN PROGRAMMA (da algoritmo a file eseguibile)

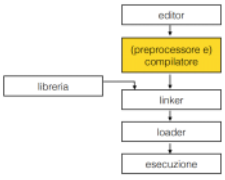
1) elaborazione di un programma



usiamo un editor di testo per scrivere un programma, seguendo la sintassi del linguaggio di programmazione scelto. Il programma viene memorizzato (*salvato*) in un file di testo il cui nome deve avere un'estensione opportuna

[in C++ l'estensione è *.cpp*]

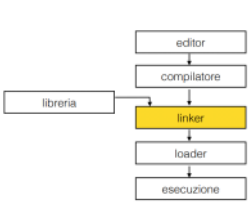
2) elaborazione di un programma



Il compilatore controlla che il programma sorgente *non contenga errori di sintassi* e successivamente effettua la traduzione in **codice oggetto**

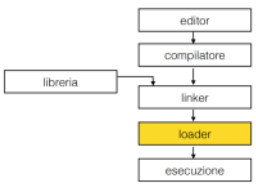
NB: verifica correttezza sintattica!

3) elaborazione di un programma



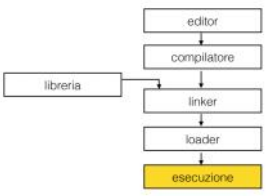
il linker combina il codice oggetto con librerie o altri codici pre-compilati, crea l'eseguibile e lo memorizza su disco

4) elaborazione di un programma



Il loader carica nella memoria principale il programma eseguibile

5) elaborazione di un programma



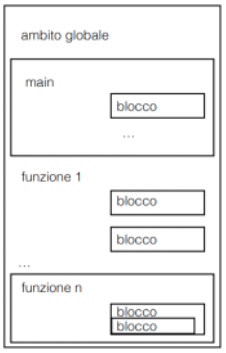
la CPU esegue un'istruzione per volta immagazzinando in memoria i nuovi dati ottenuti

Le basi del C++

venerdì 2 ottobre 2020 13:21

La struttura di un programma

- Ogni programma C++ deve contenere un blocco principale (o meglio una funzione) chiamato main
- L'esecuzione del programma inizia sempre dal main
- I primi programmi che realizzeremo saranno formati dal solo main (o quasi)



Elementi fondamentali

Linguaggio di programmazione: un insieme di regole, simboli e parole usati per comporre programmi

• Le regole di **sintassi** decretano quali siano gli enunciati o istruzioni leciti e quali non lo siano (grammatica), questi sono controllati dal compilatore.

• Le regole di **semantica** attribuiscono un significato alle istruzioni (significato), questi possono essere compresi solo dall'esecutore del programma (ovvero l'utente durante la fase di test)

I commenti: farsi capire

I programmi devono risultare chiari anche a chi li legge

• l'inserimento di commenti è importante
  - identificare gli autori del programma e la data di scrittura o modifica
  - fornire una spiegazione degli obiettivi del programma e delle sue parti (sottoprogrammi)
  - illustrare il significato degli enunciati chiave (se non sono ovvi)

Rappresentazione dei dati

- Ad un livello astratto i programmi hanno il compito di manipolare dati
- a basso livello ogni informazione è codificata come sequenza di bit (0/1)
- nei linguaggi di alto livello possiamo astrarre l'informazione in tipi diversi, alcuni predefiniti altri definiti dal program matore

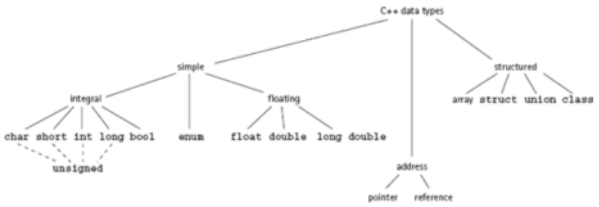
I tipi di dato

un tipo caratterizza un dominio di valori e lo spazio necessario per codificare tali valori (in bit)

- In C++ appartengono a tre categorie
  - tipo di dato semplice
  - tipo di dato strutturato
  - puntatore

Tipi semplici

- Tipo intero (integral): gestisce numeri interi
  - char, short, int, long e i loro corrispettivi unsigned (senza segno)
  - bool - valori vero/falso (1 byte)



Type	Size in Bytes*	Minimum Value*	Maximum Value*
char	1	-128	127
unsigned char	1	0	255
short	2	-32,768	32,767
unsigned short	2	0	65,535
int	4	-2,147,483,648	+2,147,483,647
unsigned int	4	0	+4,294,967,295
long	8	-9,223,372,036,854,775,808	+9,223,372,036,854,775,807
unsigned long	8	0	+18,446,744,073,709,551,615

Gli effettivi range possono cambiare (dipende dall'architettura)

- Tipo in virgola mobile (floating point): gestisce numeri con parte frazionaria
  - Sono usati per rappresentare numeri reali. Hanno una parte intera e una parte frazionaria
  - Esempi 18.0 127.45 0.57 124902.2241111 .8 (notare il punto al posto della nostra virgola…)
  - Esistono tre tipi: float, double, (long double)

- Notazione in virgola mobile (notazione scientifica)

75.924	7.592400E1
0.18	1.800000E-1
0.0000453	4.530000E-5
-1.482	-1.482000E0
7800.0	7.800000E3

Type	Size in Bytes*	Minimum Positive Value*	Maximum Positive Value*
float	4	3.4E-38	3.4E+38
double	8	1.7E-308	1.7E+308
long double	10	3.4E-4932	1.1E+4932

intervalli minimi!

Operatori aritmetici

- + - * / possono essere usati con tipi di dati interi o a virgola mobile
- % (modulo o resto) operatore che si applica solo agli interi

- **Esempi**

2+5	7	
45-90	-45	
2*7	14	
5/2	2	(agisce su valori interi)
34%5	4	(34/5 quoziente è 6 il resto è 4)
-5	-5	(in questo caso - è un operatore unario)
2-3*5	-13	

Ordini di priorità

- Le parentesi hanno priorità massima
- * / % hanno priorità su + e -
- nel caso di espressioni complesse le parentesi semplificano la lettura e la comprensione. Inoltre permettono di cambiare l'ordine di priorità

- Esempio: 3*7 - 6+2*5/4 ha lo stesso significato di ((3*7) - 6)+((2*5)/4) (5 +7)*12 è diverso da 5+7*12

Tipi di espressioni

• Espressione intera - Un'espressione aritmetica in cui tutti gli operandi sono interi. Fornisce un risultato intero
• Espressone in virgola mobile - Un'espressione aritmetica in cui tutti gli operandi sono numeri in virgola mobile. Fornisce un risultato in virgola mobile
• Espressione mista - Un'espressione che ha come operandi dati di tipo diversi. quando un operatore ha operandi misti, l'operando intero viene convertito in virgola mobile con parte frazionaria nulla. Il risultato è in virgola mobile

Tipi stringa

Una stringa è una sequenza di zero o più caratteri racchiusi tra doppio apice "ecco"
• Esempi "Francesca Odone", "Emma", "" - stringa vuota
• All'interno di una stringa ogni carattere ha una posizione definita "Francesca Odone" - 'F' è in posizione 0, ' ' in posizione 9 La lunghezza della stringa è 15 (contiamo anche gli spazi)
• Come rivedremo in seguito in C++ il tipo di dato string *non* è un tipo semplice

Creazione e compilazione di un programma

Realizzazione:

- Il main deve essere presente in ogni programma e ha la seguente forma:

```
int main()
 { prima istruzione;
    …
   ultima istruzione;
   return 0;
}
```

• il programma viene scritto e memorizzato in un file (o più file) con estensione .cpp - codice sorgente

Compilazione:

Compilare ed eseguire:

$ g++ esempio.cpp -o esempio (compilazione e creazione dell'eseguibile)
$ ./esempio (esecuzione dell'eseguibile situato nell'attuale cartella)

- Scrivendo programmi è inevitabile inserire errori (bachi o bugs) E' buona norma compilare spesso per verificare che non vi siano *errori sintattici*. E' anche buona norma provare ad eseguire il programma in modo da identificare eventuali *errori semantici* (comportamenti non previsti)

Variabili e costanti

Nei linguaggi di alto livello è possibile (e utile) associare identificatori o nomi ai dati. Segue la terminologia:

• dichiarazione: realizza un'associazione logica tra un identificatore e un'area di memoria in grado di immagazzinare un dato di un certo tipo

```
int num;   riserviamo un'area di memoria abbastanza grande per
           contenere un int e le associamo l'identificativo num
num = 10;  assegnamo il valore 10
```

l'identificatore può far riferimento a
- *contenitore:* area di memoria, il cosiddetto valore sinistro
- *contenuto*: il valore, il cosiddetto valore destro.

Se un dato non deve cambiare nel corso della vita del programma allora possiamo memorizzarlo come costante con nome (named constant): una named constant è una

locazione di memoria il cui contenuto **non** può essere modificato durante l'esecuzione del programma (altrimenti da errore).

Sintassi della dichiarazione di costante:

```
const nomeTipo identificatore = valore;
```

Esempi

```
const float PI=3.14159;
const double CONVERSION=2.54;
const char BLANK=' ';
const string NAME="Elizabeth";
```

Nota l'uso dei nomi di costante con MAIUSCOLE!

I nomi di variabili e costanti sono di libera composizione (meno che le key word e altre regole specificate di seguito) ma devono comunque far capire a un eventuale 'secondo' programmatore che legge, che valori vanno concretamente a contenere quelle variabili:

- Dire Numero_Studenti va bene (lungo).
- Dire Num_Stud va bene.
- Dire Ns non va bene! Si compromette la leggibilità!

Ricorda che il C++ è **case sensitive** (suscettibile alle maiuscole), quindi la variabile 'Studenti' è diversa da 'studenti'.

Queste sono le regole da seguire nella creazione degli identificatori:

I token in C++ (elementi non divisibili)

- **simboli speciali**
  + - * / . ; ? , <= >= == !=
  lo spazio (blank)
- **keyword** o parole riservate:
  int, float, double, char, const, void, return
- **identificatori:** nomi di entità (variabili, costanti, funzioni) che compaiono nei programmi. Possono essere predefiniti o definiti dal programmatore
  Possono includere lettere (A-Z, a-z), numeri (0-9), il carattere "underscore" (_)

Dichiarazione delle variabili

In C/C++ gli identificatori (costanti o variabili) devono essere dichiarati prima di poter essere usati!
Le dichiarazioni possono essere inserite in varie parti del programma (e questo determina il loro scope, ovvero una regione (locale) del programma)
Una volta dichiarata, come facciamo ad inserire dati in una variabile?
1. Enunciato di assegnazione (assegnamento)
2. Enunciato di lettura (input)

Enunciato di assegnazione (assegnamento, input)

L'espressione per definire la lettura di un valore destro (contenuto) alla variabile (valore sinistro) è la seguente:
cin >> nome_variabile;

Ricorda >> e il ; alla fine!

Nota bene. L'espressione num = num + 1 (aggiornamento della stessa variabile) necessita di un'inizializzazione (es. int num = <valore>). Se durante le operazioni faccio riferimento a variabili non inizializzate incorro in un errore semantico: il valore 1 si somma a una sequenza di numeri (che corrispondono a una sequenza di una cella della RAM, che costituisce il valore base di ogni variabile appena creata.

```
int i,j;
char ch;
double x;
```

Statement	Data	Contents After Input
1. cin >> i;	32	i= 32
2. cin >> i >> j;	4 60	i = 4, j = 60
3. cin >> i >> ch >> x;	25 A 16.9	i = 25, ch = 'A', x = 16.9
4. cin >> i >> ch >> x;	25	
	A	
	16.9	i = 25, ch = 'A', x = 16.9
5. cin >> i >> ch >> x;	25A16.9	i = 25, ch = 'A', x = 16.9
6. cin >> i >> j >> x;	12 8	i = 12, j = 8 (Computer waits for a third number)
7. cin >> i >> x;	46 32.4 15	i = 46, x = 32.4 (15 is held for later input)

Per leggere più variabili:
cin >> a;
cin >> b;
E' uguale a:
cin >> a >> b;

Enunciato di assegnazione (assegnamento)

L'espressione per definire l'assegnazione di un valore destro (contenuto) alla variabile (valore sinistro) è la seguente:

<variabile> = <espressione (o valore)>
Esempio:
Numero = 82;

Occhio ai domini (tipi) delle variabili:

Definire char ch, int num **può** causare una perdita di valori se converto: ch = num!

Enunciato di visualizzazione (output)

L'espressione per definire l'output dei dati è la seguente:

```
cout << variabile << "testo" << endl;
cout << variabile << "\n";
```

Scrivere endl o "\n" termina la riga, andando a capo, è indifferente usare uno o l'altro.

Cast (conversione di tipo)

Cast esplicito (più safe): static_cast (espressione);
Cast implicito (spesso unsafe):

```
int i, char ch;
ch = i;
```

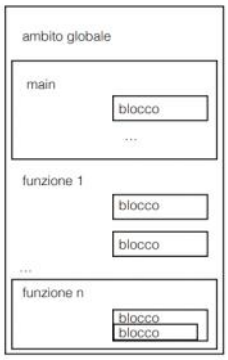
Il cast implicito spesso **può** portare sviste e, come nell'ultimo caso, la trasformazione può far perdere dei valori (che il char non può contenere)

- **Conversioni unsafe (ATTENZIONE!)**
  - Double to int, char o bool
  - Int to char o bool
  - Char to bool

```
double x=2.7;
int i=x; //i diventa 2
```

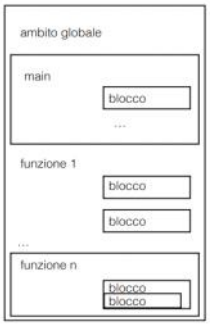
scope delle variabili

- variabili globali: sono variabili dichiarate nell'ambito globale, vivono per tutta la durata del programma e sono visibili ovunque
- variabili locali: sono le variabili dichiarate in un ambito diverso da quello globale. Sono visibili nell'ambito (blocco) in cui sono state dichiarate e in tutti gli ambiti in esso annidati.
- NB: le variabili dichiarate nel main *non sono globali!*



scope delle variabili

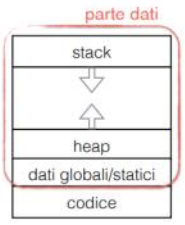
- Lo *scope* o ambito di visibilità è la "regione" del programma dove una data variabile è visibile, ossia dove il suo identificatore può essere utilizzato
- La nozione di *scope è rilevante a compile time*



lifetime delle variabili

- "lifetime" è il tempo durante il quale la variabile è presente in memoria ed è quindi accessibile dal programma in esecuzione
- Si tratta di un concetto rilevante rispetto al *tempo di esecuzione del programma*
- *Casi particolari :*
  - variabili statiche: sono dichiarate in un ambito non globale e sono visibili solo in tale ambito, ma vivono per l'intera durata del programma
  - variabili dinamiche: non sono dichiarate in modo convenzionale ma vengono create dal programma durante l'esecuzione. Di norma sono raggiungibili tramite puntatori.
    La loro vita è l'intervallo che intercorre tra la loro creazione e la loro distruzione

Dati in memoria a run time



- la parte del codice è statica, le istruzioni vengono caricate in memoria al lancio del programma e lette in sequenza. Non possono venir modificate dal programma
- Dati globali/statici: variabili globali e statiche. Allocazione statica, ma possono venir modificate da qualunque istruzione del programma
- Stack: variabili locali all'atto dell'attivazione di un blocco (regole di allocazione dello spazio definite durante la compilazione - *compile time*) blocco di celle contigue
- Heap: variabili dinamiche (per la richiesta di spazio a *run time*) possibile frammentazione

Notazione abbreviata degli operatori aritmetici

Applicabile a tutti gli operatori aritmetici:

- `number+=1;` equivale a `number=number+1;`
- `total -= discount;` equivale a `total = total - discount;`
- `amount *= count1+count2;` equivale a `amount=amount*(count1+count2)`

Operatori di incremento e decremento

- incremento prefisso: `++variabile`
- incremento postfisso: `variabile++`
- decremento prefisso: `--variabile`
- decremento postfisso: `variabile--`

Operatori di incremento e decremento

```
int x, y;
x=5;
y=++x;
```

x	5
x	6
y	6

```
int x,y;
x=5;
y=x++;
```

x	5
y	5
x	6

Altri dettagli sulla compilazione

direttive per il preprocessore

- nel linguaggio C++ sono definite in modo esplicito poche operazioni
- molte delle funzioni e dei simboli necessari sono forniti in una raccolta di *librerie* ognuna con un *file di intestazione* (***header***) associato
  - un esempio già visto: il file `iostream`
- le direttive per il pre-processore sono comandi che consentono al pre-processore di modificare un programma sorgente C++ prima che venga compilato

direttive per il preprocessore

- tutte le direttive iniziano con il carattere # e non terminano con ;
- La sintassi prevista per includere un file di intestazione è `#include <fileIntestazione>`

```
#include <iostream>
```

- Vanno posizionate come prime righe del programma

namespace (cenni)

- L'errore che otteniamo è dovuto al fatto che il file iostream nasconde i suoi identificatori in un blocco chiamato std

```
namespace std
{   // Start of namespace block
  :  // Declarations of variables, data types, and so forth
}   // End of namespace block
```

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Il classico hello world" << endl;
        return 0;
}
```

namespace (cenni)

- L'errore che otteniamo è dovuto al fatto che il file iostream nasconde i suoi identificatori in un blocco chiamato std
- In alternativa possiamo utilizzare l'identificatore *specificando ogni volta* il namespace nel quale è stato definito

nota che l'include non può mancare!

```
#include <iostream>

int main()
{
    std::cout << "Il classico hello world" << std::endl;
    return 0;
}
```