le basi del C++

introduzione alla programmazione

(1) I primi ingredienti

tipi di dato, operatori aritmetici, espressioni

programmazione imperativa in C++

- Il programma più corto in C++

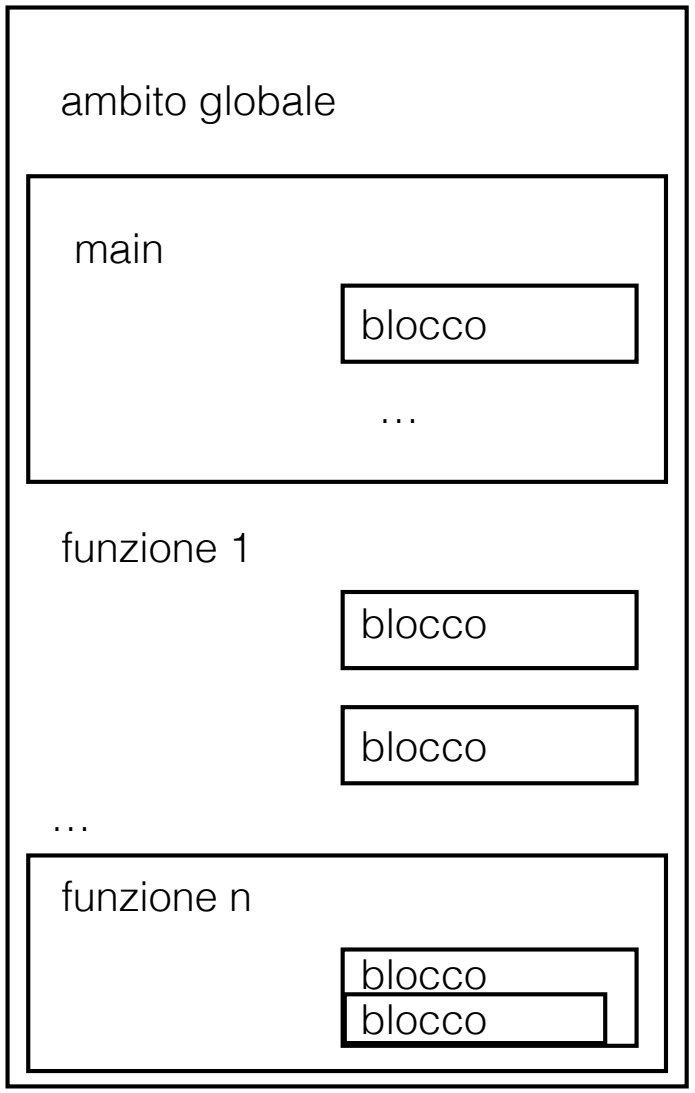
```
int main()
  {

  }
```

- La **struttura** di un semplice programma

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

prime osservazioni sulla struttura di un programma C++

- 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 di un programma

- 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)
- Le regole di **semantica** attribuiscono un significato alle istruzioni (significato)

Elementi fondamentali di un programma

- 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)

prime osservazioni sullo stile di programmazione

- per il compilatore non fa differenza, ma i seguenti programmi non sono accettabili!

```
#include <iostream>   using namespace std; int main(){ cout << "Il mio
primo programma C++" << endl;     cout << "La somma di 2 e 3 e' " << 5
<< endl; cout << "7 + 8 = " << 7 + 8 << endl; return 0; }
```

```
#include <iostream>
using namespace std;
int main(){
cout << "Il mio primo programma C++" << endl;
cout << "La somma di 2 e 3 e' " << 5 << endl;
cout << "7 + 8 = " << 7 + 8 << endl;
return 0;
}
```

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 programmatore

rappresentazione dei dati: tipi

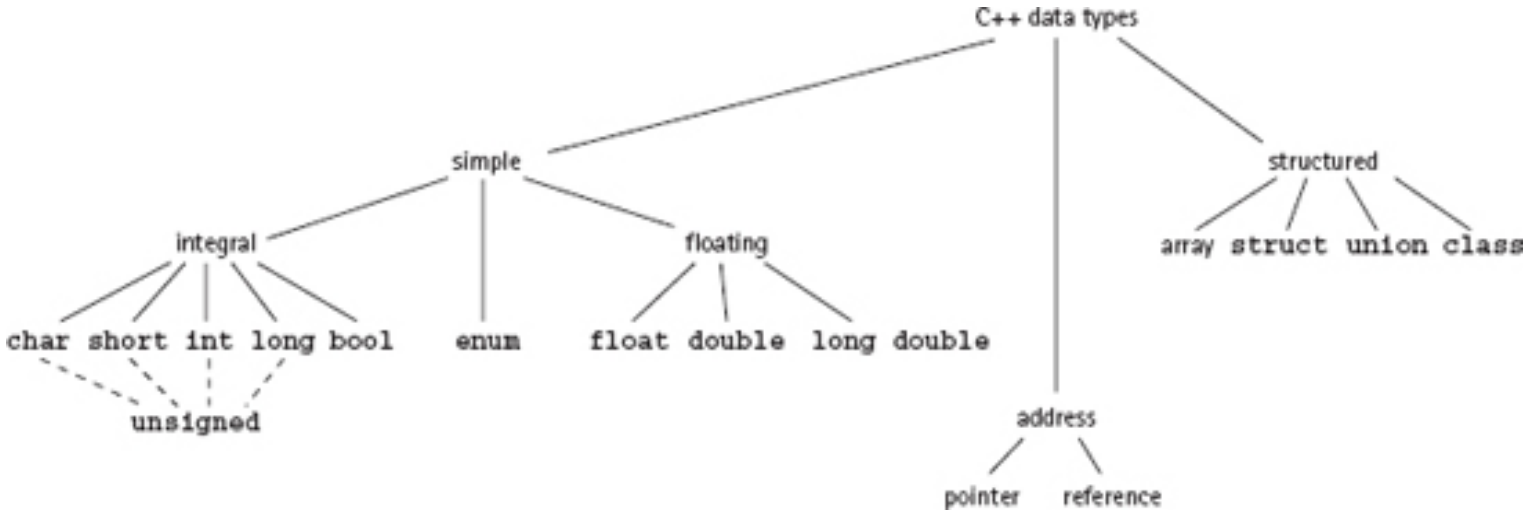
- un tipo caratterizza un *dominio di valori* e lo *spazio* necessario per codificare tali valori (in bit)
- Esempi:
  - interi con segno (per es, `int` in C e C++) e interi senza segno
  - numeri razionali, che approssimano i reali
  - valori booleani (vero e falso)
  - caratteri
  - puntatori: indirizzi di altri dati (non ad una singola cella di memoria ma all'intero spazio occupato da un dato)
  - stringhe (anche se solitamente non sono tipi base)
  - tipi strutturati (array, stream, record…)

Tipi

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

Tipi di dato semplici

- Tipo intero (integral): gestisce numeri interi
- Tipo in virgola mobile (floating point): gestisce numeri con parte frazionaria



tipi di dati interi

- **char**, **short**, **int**, **long** e i loro corrispettivi **unsigned** (senza segno)

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

intervalli minimi!

gli effettivi range possono cambiare (dipende dall'architettura)

- bool - valori vero/falso (1 byte)

tipi di dati interi

- **Esempi di int** -6240 0 78 +43444
  Nota bene: 26.0 *non è un intero!*

- **bool:** true false
  Il loro scopo principale è consentire l'elaborazione delle espressioni booleane

- **char:** oltre a gestire i numeri interi piccoli [-128,127] viene anche usato per rappresentare caratteri: lettere cifre simboli speciali.
  Ciascun carattere va racchiuso tra apici singoli
  Esempi di char: 'A' 'a' 'O' '*' '&' '|' ' '

  Nota bene: 'abc' non è un char (singoli caratteri!!)

tipi di dati interi

- Ancora sui **char…**
  - esistono diversi insiemi di caratteri utilizzati, ASCII (128 valori), EBCDIC (256 valori),..
    - Vedi per esempioAPPENDICE C del Malik
    - oppure http://en.cppreference.com/w/cpp/language/ascii
  - Noi considereremo solo ASCII (ogni carattere è codificato dagli interi [0,127])
  - Ciascun valore rappresenta un carattere diverso (non tutti stampabili)
    Ad esempio il valore 65 rappresenta ‘A’, il valore 43 rappresenta ‘+’…
  - di conseguenza possiamo definire un ordinamento tra caratteri (‘+’ è minore di ‘A’)
  - Caratteri interessanti:
    ‘\n’ - newline (a capo)
    ‘\t’ - tab (tabulazione orizzontale)
    ‘\0’ - null (carattere nullo)

Tipi di dato in virgola mobile

- 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…)

- 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	

ordine di priorità degli operatori

- 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

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*

Esercizi (a penna!)

- 2+3*5
- 12.8*17.1 - 30.0
- 6/(4+3)
- 4*3 + 7/5 - 12.4
- 7 + 8
- ‘7’ + ‘8’

Il tipo string (primi accenni)

- con i tipi di dato visti finora non siamo in grado di memorizzare, per esempio, il nome di una persona

- 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 di un programma C++

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

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

Ogni istruzione termina con ;

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

Creazione di un programma C++

```
// I programmi piu' semplici

int main()
{

    5+4;
    2*4;

}
```

Compilazione

Compilare ed eseguire:

```
$ g++ esempio.cpp  -o esempio
```

```
$ ./esempio
```

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)

(2) associare identificatori

variabili, costanti,
memorizzare dati (input),
visualizzare dati (output)

rappresentazione dei dati: variabili e costanti

- nei linguaggi di alto livello è possibile (e utile) associare identificatori o nomi ai dati

- *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.

operazioni principali

- scrivere nel contenitore
- leggere il contenuto

Costanti

- 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

- 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!

Costanti

- perche introdurre una costante con nome invece che utilizzare direttamente il valore?

Variabili

- **Variabile** - una locazione di memoria il cui contenuto può cambiare durante l'esecuzione di un programma

- Sintassi della dichiarazione di variabile:
```
nomeTipo identificatore1, identificatore2, …;
```

- Esempi
```
int alpha;
float rate;
char ch;
```

- Non esistono regole formali su come scegliere i nomi delle variabili. La chiarezza e uno stile personale (uniforme) sono una buona linea guida. Esempio: variabile che memorizza il numero di studenti di un corso
```
int Students_Number; // forse un po' lungo
int StudentsNum;
int students_num;
int sn; // non si capisce cosa sia
```

Ricapitolando, elementi di un codice C++

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” (_)

Ricapitolando, elementi di un codice C++

- Il C++ è **case sensitive**, cioè le lettere maiuscole e minuscole sono considerate diverse
  L'identificatore `Number`, è diverso da `number` e da `NUMBER`

- In C++ si possono usare identificatori di qualunque lunghezza, ma attenzione a non compromettere la leggibilità!
  esempi: `first, conversion, payRate, counter1,first_amount`

- Esempi di identificatori non validi

  - `employee salary` lo spazio non può far parte di un id
  - `Hello!` `!` non può far parte di un id
  - `one+two` `+` non può far parte di un id
  - `2nd` un id non può iniziare con una cifra

Memorizzazione dati in una variabile

IMPORTANTE!! 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)

Una volta dichiarata, come facciamo ad inserire dati in una variabile?

1. Enunciato di assegnazione (assegnamento)

2. Enunciato di lettura (input)

Enunciato di assegnazione

- Sintassi: `variabile=espressione;`

- Esempi
  Dichiarazione:

```
int num1, num2;
double sale;
string str;
```

  Assegnazione

```
num1=4;
num2= 4-5*3;
str="It's a sunny day!";
```

Nota: l'espressione `num=num+1;` ha senso solo se num era stato precedentemente *inizializzato*

Enunciato di lettura

- vediamo ora come trasferire dati in variabili trasferendoli dal dispositivo di ingresso standard (***standard input***) - solitamente la tastiera

```
cin >> variabile >> variabile … ;
```

operatore
di estrazione

- cin, come cout, richiede l'inclusione della libreria iostream e la presenza dell'enunciato using namespace std;

- Abbiamo adesso due modi per inizializzare la variabile

```
feet = 35;
```

oppure

```
cin >> feet;
```

Enunciato di lettura

- quando si inserisce un dato bisogna essere sicuri che sia appropriato per il tipo di dato al quale verrà associato
- Dati validi
  - char ch; cin >> ch; // un carattere stampabile escluso lo spazio
  - int i; cin >> i; // un intero (eventualmente con segno)
  - double d; cin >> d; // un numero frazionario eventualmente in notazione scientifica e /o con segno

Enunciato di lettura

- supponiamo che venga digitato il carattere 2.
  - come fa l'operatore >> a capire se si tratta di un int o di un char?
  - e' l'operando che segue (la variabile) che fa la differenza
- Se viene digitato il carattere 25 `cin >> var;`
  - se var è un char viene memorizzato solo il 2
  - se var è un int viene memorizzato 25
  - se var è un double avviene una conversione e viene memorizzato 25.0

Enunciato di lettura - esempi

```
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)

mancata corrispondenza tra tipi e dati: input failure!

Visualizzazione (output)

- In questi primi esempi abbiamo utilizzato alcuni enunciati di visualizzazione o output, per mostrare i risultati delle nostre elaborazioni

- In C++ si possono visualizzare informazioni sul *dispositivo di uscita predefinito* (lo ***standard output***) usando `cout` e l'operatore << (operatore di inserimento in un flusso)
  `cout << espressione << espressione << … ;`
  dove le espressioni vengono valutate e il loro risultato visualizzato sul dispositivo nel punto di inserimento attuale (sullo schermo nella posizione del **cursore**)

- Per determinare il formato dei dati possiamo usare un manipolatore

Visualizzazione (output)

- ```
  char ch='A';
  cout <<ch;
  ```

- ```
  cout << 'A';
  ```

  visualizzano entrambi

  A

Visualizzazione (output) - esempi

il manipolatore endl sposta il punto di inserimento all'inizio della riga successiva

```
cout << 29/4 << endl;
 cout << "Hello there" << endl;
cout << "Hello \t there" << endl;
cout << "Hello \n there" << endl;
cout << "4 + 7" << endl;
cout << 4+7 << endl;
cout << '4'+'7' << endl;
```

```
skin:codice odone$ ./a.out
7
Hello there
Hello	there
Hello
 there
4 + 7
11
107
skin:codice odone$
```

Realizziamo il nostro primo programma in C++ (versione 2)

```
…
// I programmi piu' semplici

int main()
{

    cout << 2+3*5 << endl;
    cout << 12.8*17.1 - 30.0 << endl;
    cout << 6/(4+3) << endl;
    cout << 4*3 + 7/5 - 12.4 <<endl;

}
```

Conversione esplicita di tipo (cast)

- Sintassi static_cast<nometipo> (espressione)
- Esempio
  `static_cast<int>(7.8+static_cast<double>(15)/2)` è 15
  `static_cast<int>(7.8+static_cast<double>(15/2))` è 14
- Altro esempio
  `static_cast<int>('8')` è 56
  `static_cast<char>(56)` è '8'

NOTA:
Le conversioni di tipo non dovrebbero servire quasi mai…

(3) qualche nozione in più

Operatori aritmetici in forma abbreviata,
Regole di visibilità di var e const
Type safety

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)`

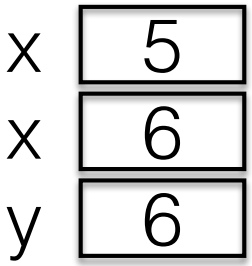
Operatori di incremento e decremento

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

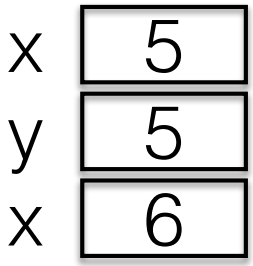
- decremento prefisso: `--variabile`
- decremento postfisso: `variabile--`

Operatori di incremento e decremento

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




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



Esercizi

- a=5;
  b=2+(++a);

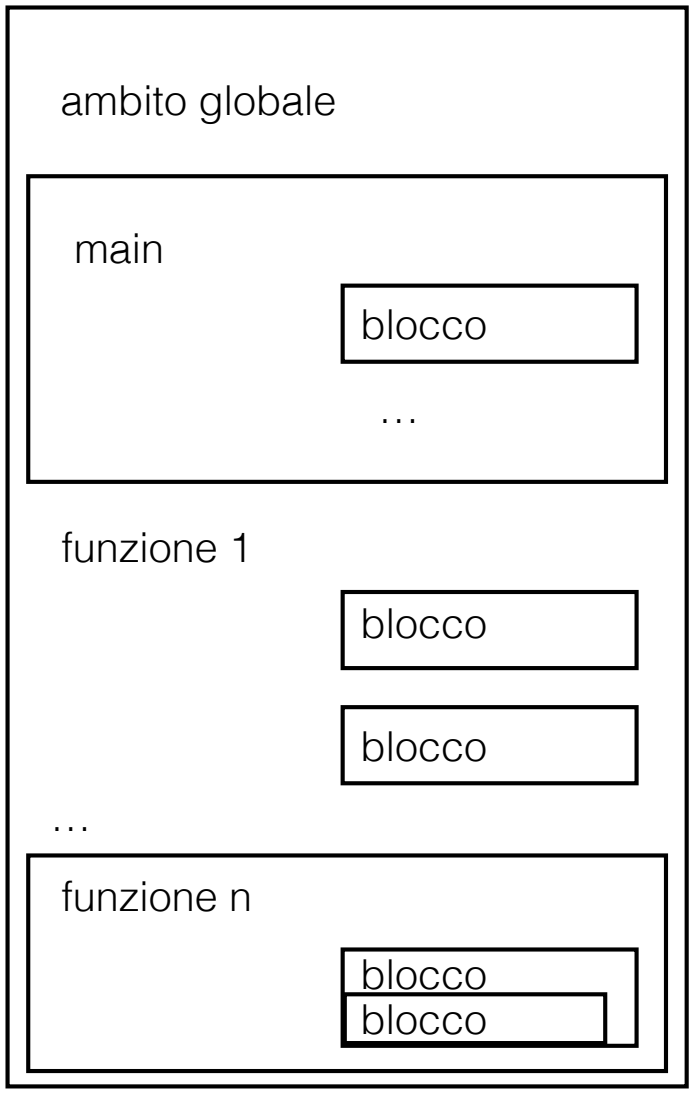
- a=5;
  b=2+(a++);

regole di visibilità e vita delle variabili

- **visibilità** o **scope** della variabile: possibilità di usare una data variabile in un dato ambito (regione del programma)
- **vita** o **lifetime** della variabile: lasso di tempo (a run time) nel quale la variabile è disponibile

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*

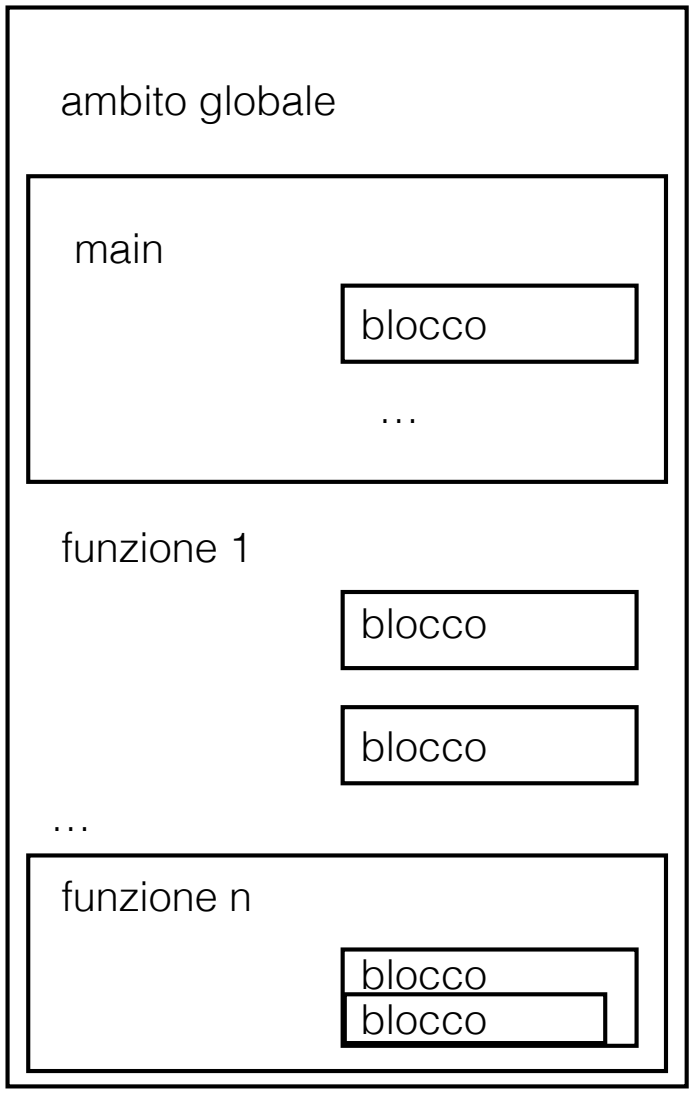


scope delle variabili

- 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!*



lifetime delle variabili

- “lifetime” è il tempo durante il quale la variabile è presente in memoria ed è quindi accessibile dal programma in esecuzione

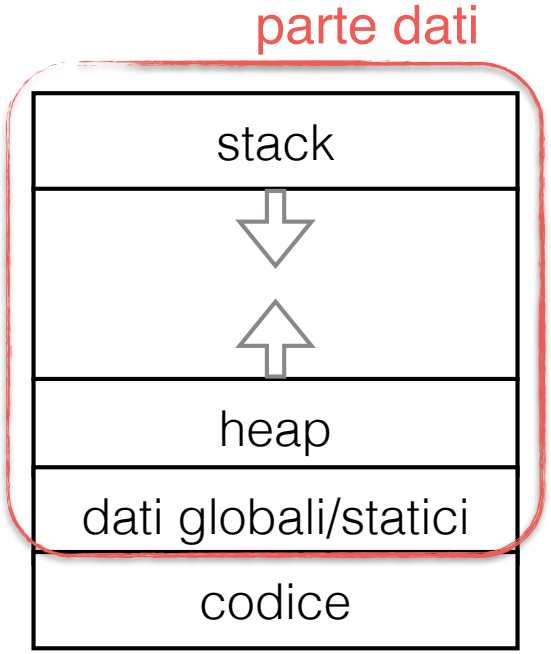
- Si tratta di un concetto rilevante rispetto al *tempo di esecuzione del programma*

- *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

Type safety

- Un programma, o parte di esso, è *type safe*, se tutti gli elementi che lo compongono vengono manipolati secondo le regole del loro tipo.

- basta poco per ottenere una situazione *unsafe*

```
double x; // mi sono dimenticato di inizializzare
double y=x; // il valore di y è indefinito
```

- Un esempio classico di operazione che può fare sorgere *unsafety* è la conversione di tipo.

Type safety

- Conversioni implicite safe (senza perdita di informazione)
  - Bool to char
  - Bool to int
  - Bool to double
  - Char to int
  - Char to double
  - Int to double

```
// char-to-int in due modi diversi
    char c='x';
    int i=c;
    int j='x';
```

Type safety

- **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
```

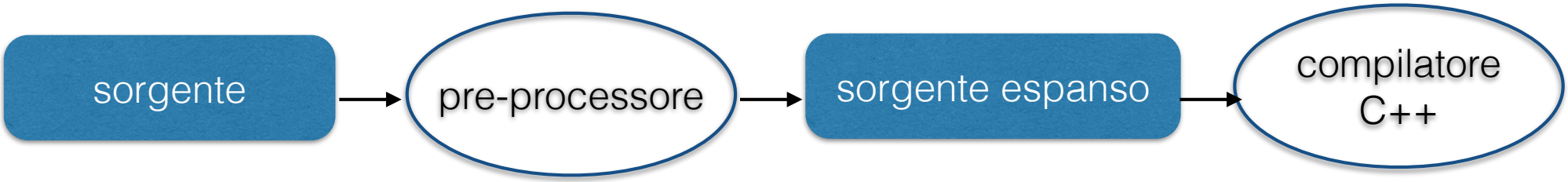
- Il cast esplicito visto in precedenza (operatore static_cast) può aiutare a ridurre i rischi di conversioni implicite fuori controllo…

(4) compilazione etc

Dettagli in più su compilazione, preprocessore, namespace

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)

- Proviamo a compilare le seguenti righe

```
#include <iostream>

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

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;
}
```