

in C++ ogni variabile ha un tipo,
perché? testo 2.5

- tipi diversi di valori servono per molti problemi che vogliamo risolvere
- il tipo di un valore serve ad accedere alle sequenze di byte nella RAM nel modo appropriato
- **test di sensatezza** dei programmi: i valori sono usati in modo coerente col loro tipo

```
int x=0;
while(x<10)
{
    int a=40;
    int y=0;
    while(y<10)
    {
        int z=0;
        while(z<10)
        {
            int a=20;
            z++;

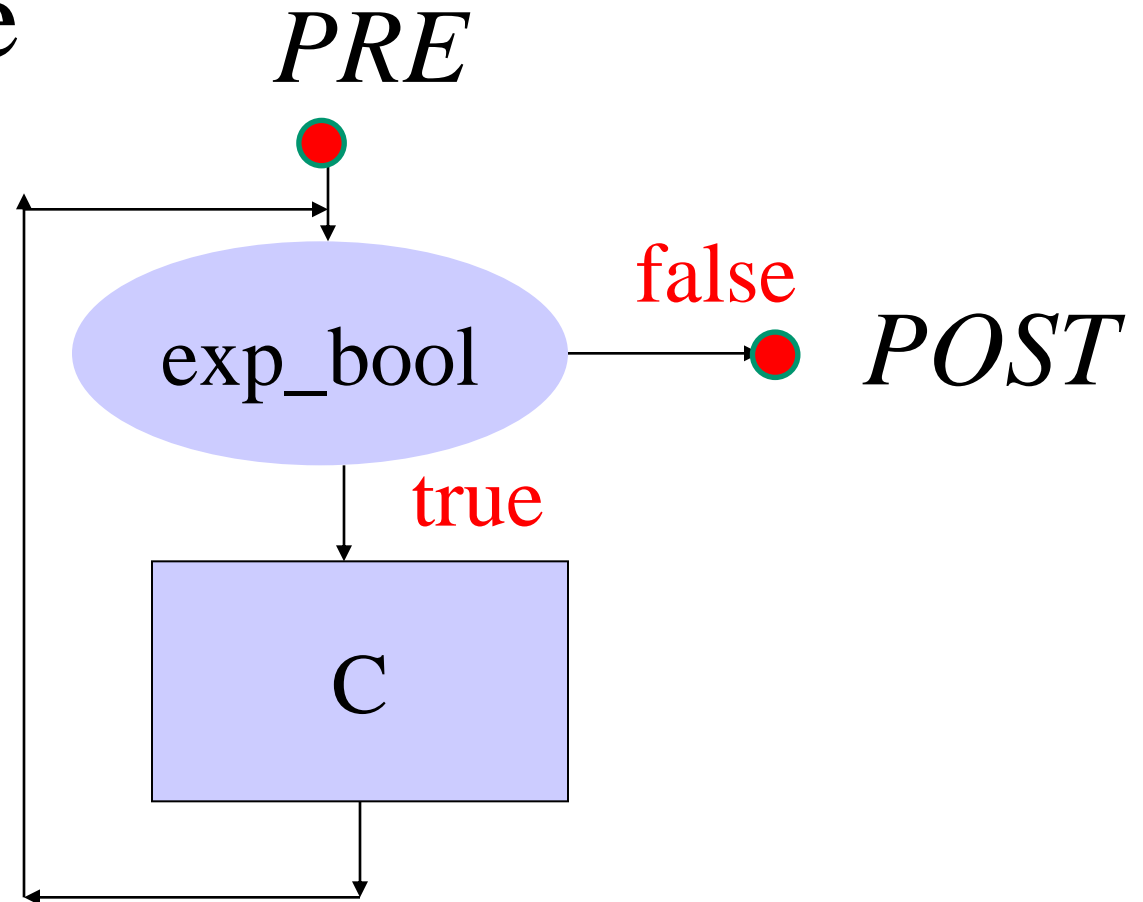
        }
        int b=30;
        y++;
    }
    x++;
}
```

innestamento dei cicli e
gestione delle variabili

testo sezione 2.3

cicli while e correttezza

ciclo while



1 punto d'entrata ed 1 d'uscita

PRE=()

int x=0;

while(x < 10)

{

 x=x+1;

}

cout << x;

POST=(x =10)

$\{0 \leq x \leq 10\}$ sempre vero

$\{0 \leq x < 10\}$

$\{0 \leq x \leq 10\} \ \&\&\{ x \geq 10\}$


Esercizio base: leggere interi da cin fino a che si legga il valore -1 e fare la somma senza sommare il -1

```
pre=(cin=v1..vn -1 ..., n>=0, v1..vn != -1)
```

PROGRAMMA

```
post=(somma=v1+..+vn, se n=0  
somma=0)
```

dal mercoledì

```
#include<iostream>
using namespace std;
main()
{
    int somma=0, x, m=1;
    cin >> x;
    while (x!=-1) 
    {
        somma=somma+x;
        m=m+1;
        cin >> x;
    }
}
```

INVARIANTE

letti i primi m valori da
cin, x contiene l'ultimo,
i primi $m-1$ sono $\neq -1$ e
somma è la somma dei
primi $m-1$ valori
(se $m-1=0$ somma=0)

R

R && x= -1 => POST

Esercizio:

dato $X \geq 1$ vogliamo calcolare il minimo e
tale che $2^e \geq X$

PRE=(cin contiene intero >0)

PROGRAMMA

POST= (esponente è il minimo esponente
tale che $2^{\text{esponente}} \geq X$)

potenza=2 ^{esponente}

$X > 2^{(\text{esponente}-1)}$

R

```
while( X > potenza)  
{
```

```
    potenza=potenza*2;
```

```
    esponente = esponente + 1;
```

```
}
```

```
cout<< "l'esponente e'=" << esponente<<endl;
```

potenza=2 ^{esponente}

$X > 2^{(\text{esponente}-1)}$

R

&& X < = potenza => POST

INIZIALIZZAZIONE

```
int X, potenza=1,  
esponente=0;  
cin >> X;    // X > 0
```

// diventa vera

potenza=2 esponente

$X > 2^{(\text{esponente}-1)}$

R

esercizio 4.3:

si legge sequenza di caratteri numerici
fino a leggere 'a'

esempio: '3' '2' '6' 'a'

vogliamo calcolare l'intero 326

PRE=(cin contiene $c_1 \dots c_k$ **a**, $k \geq 0$, c_j carattere numerico $j \in [1, k]$)

programma ?

POST=(calcola NUM($c_1 \dots c_k$))

NUM('2' '3' '1')=231 NUM()=0

inizio :

leggi un carattere

se è numerico fai i conti e torna all'inizio
altrimenti fine


fai i conti ?

se abbiamo letto

c1c2c3 abbiamo NUM(c1c2c3) e se c4 ok
dobbiamo ricavare NUM(c1c2c3c4)

come ??

$$\text{NUM}(c1c2c3c4) = 10 * \text{NUM}(c1c2c3) + \text{NUM}(c4)$$



all'inizio del ciclo abbiamo un
carattere letto da gestire così

```
char q; int num=0, n=1;  
IN>>q;
```

```
R=(num=NUM(c1..c(n-1)), q=cn)
```

```
while(q != 'a')  
{  
    num=num*10+(q-'0');  
    IN >> q;  
    n++;  
}
```

Regola Generale di prova del while

PRE $\langle ini; \text{while}(B) C \rangle$ POST

si trova invariante R e si dimostra:

1) PRE $\langle ini \rangle R$ *condizione iniziale*

2) $R \ \&\& \ B \ \langle C \rangle R$ *invarianza*

3) $R \ \&\& \ ! B \Rightarrow \text{POST}$ *condizione d'uscita*

Esercizio 4.7: leggere valori interi da IN e scriverli su OUT

al più 10 valori, ma

fermarsi con due 0 consecutivi **senza
scrivere**

$PRE = (IN = b_1 \dots b_k, k > 9 \text{ o contiene due } 0 \text{ consecutivi})$



programma

$POST = (OUT = \text{Prefix}(b_1 \dots b_k))$

$\text{Prefix}(b_1 \dots b_k) = b_1 \dots b_{10}$ se $b_1 \dots b_{10}$ non
contiene due 0 consecutivi, altrimenti
 $\text{Prefix}(b_1 \dots b_j 00 \dots) = b_1 \dots b_j$

se leggo 0 non lo posso stampare

situazione a inizio ciclo:

- se ho letto 2 zeri consecutivi devo terminare,

- altrimenti **devo distinguere:** se ho letto uno zero al giro precedente o no

idea: due booleani per distinguere

- ho letto uno zero o no

- ho letto due zeri o no

invariante R

1. $0 \leq n \leq 10$
2. letti $b1 \dots bn$
3. $(!uno0 \ \&\& \ !due0) \Rightarrow (OUT = b1 \dots bn)$
4. $due0 \Rightarrow (bn-1 = bn = 0 \ \&\& \ OUT = b1 \dots bn-2)$
5. $(uno0 \ \&\& \ !due0) \Rightarrow (bn = 0 \ \&\& \ OUT = b1 \dots bn-1)$

```
int X, n=0;
bool uno0=false, due0=false;
while(! due0 && n < 10)
{ IN >> X; n++;
  if(X==0)
    if(uno0)
      due0=true;
    else
      uno0=true;
  else
    {if(uno0)
      {OUT<<0 <<' '; uno0=false;}
    OUT << X << ' ';
    }
  if(uno0 && !due0) OUT << 0;
}
```

manca:

```
if(uno0 && !due0)  
    OUT<< 0;
```