# Tipi Predefiniti

- 2 parole sul C++
- i tipi predefiniti del C++
- Capitolo 2.1 del Testo

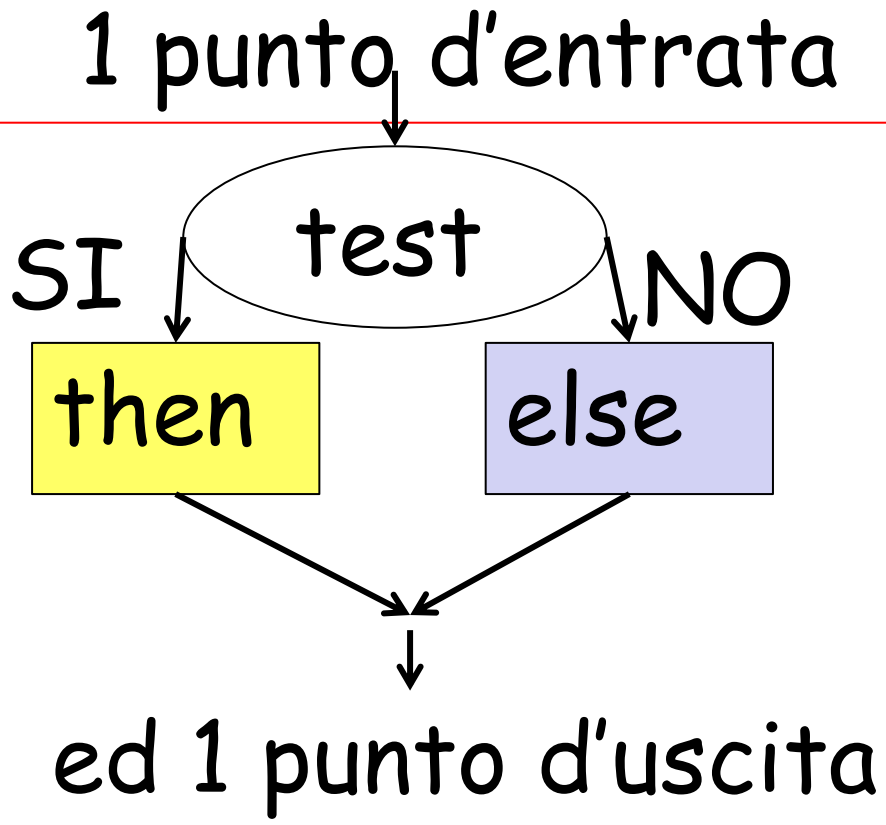
il C++ non è "solo" un linguaggio, esso riflette 40 anni di sperimentazione e innovazione nei Linguaggi di Programmazione (LP):

→ racchiude molti concetti che sono il frutto di questa storia

| | |
|---|---|
| il C++ è | strutturato a blocchi |
| possiede | funzioni |
| possiede | tipi definiti dall'utente |

ereditarietà, classi

# un linguaggio è strutturato a blocchi



# TIPI

un TIPO è una coppia :

- un insieme di valori
- e da operazioni definite su questi valori

- rappresentazione sorgente (costanti)
- rappresentazione interna nella memoria (quanti byte occupano e come sono organizzati)

il C++ non prescrive quanti byte usare per realizzare i valori di ciascun tipo quindi ogni compilatore è "libero" di deciderlo

tenendo conto della CPU che deve eseguire il codice oggetto

per conoscere quanti byte vengono usati per un qualsiasi tipo T dal compilatore che si sta usando:

sizeof(T); // restituisce i byte usati per T

oppure se X è una variabile:

sizeof(X); //restituisce i byte per il tipo di X

# Tipi predefiniti

interi → int
reali → float e double
caratteri → char
booleani → bool
vuoto → void

stringhe alla C → char*

# tipo intero = int

- se abbiamo a disposizione n bit allora l'insieme di valori interi è:

$$-2^{(n-1)} \ldots\ldots\ldots\ldots 2^{(n-1)}-1$$

- le operazioni sugli interi sono le operazioni aritmetiche e quelle di confronto

operazioni frequenti:

**+ - / * %** (modulo)

**==** (uguale)
**!=** (diverso)
**>**
**>=**
eccetera.

# rappresentazione sorgente degli interi

- valori interi:

127

4203

-55

- devono essere tra

INT_MIN e INT_MAX

# interi in base 8 e base 16

0127 rappresenta un numero in base 8

1*64+2*8+7 = 87 in base 10

0x127 rappresenta un numero in base 16

1*256+2*16+7 =295 in base 10

# rappresentazione interna di int

in aula informatica è presente il compilatore C++ GNU 4.4.3 esso usa per i valori interi 4 byte, cioè 32 bit

gli interi sono rappresentati in complemento a 2 come little-endians che significa che i bytes sono ordinati nella memoria dal meno significativo al più significativo

# i reali

- il C++ offre 2 tipi di valori reali:
  float e double
- i double sono più precisi dei float
- anche sui reali le operazioni sono quelle aritmetiche e quelle di confronto

# rappresentazione sorgente dei reali

12.5  12.5e2 (= 12.5 *$10^2$)

sono di tipo double

12.5f

è invece float

le operazioni sono + - / * == > >=
… insomma le stesse che per int

O NO ??

Rappresentazione interna: in virgola mobile (floating point)

- un bit s serve per il segno, m bit per la mantissa e n bit per l'esponente
- se la mantissa rappresenta il valore y, l'esponente il valore x allora il valore reale rappresentato è

$$s\ 1.y * 2^x$$

nel nostro compilatore i float vengono realizzati con 4 byte mentre i double usano 8 bytes.

Per i float la mantissa col segno occupa 24 bit e l'esponente 8 bit

Torniamo al +, *, ecc. per interi e reali

sono operazioni diverse anche se rappresentate con gli stessi simboli

5+2 → + per interi

2.2+3.1 → + per reali

questo fenomeno si chiama

**overloading** (sovraccaricamento)

il compilatore usa l'operatore giusto sulla base del tipo degli operandi

il tipo bool ha 2 valori vero e falso
la rappresentazione sorgente è true
e false (ma anche 1 e 0)

gli operatori naturali del tipo bool sono:

| | |
|---|---|
| ! | NOT |
| && | AND |
| \|\| | OR |

in C++ i 2 valori bool sono implementati internamente con 1 byte dove

false è rappresentato dal byte che contiene 0 (tutti i bit a 0)

true un byte con valore 1 (ma anche qualsiasi valore diverso da 0)

è un'eredità del C che non ha il tipo bool

a causa di questo fatto:

le seguenti espressioni sono corrette in C++:

0 && 5 == 0

true + false == true

quale + viene usato ?? Lo vedremo più avanti

# i caratteri

il tipo char ha come dominio 256 caratteri che contengono:

- i caratteri alfabetici maiuscoli e minuscoli
- caratteri accentati
- le cifre 0,1,2, ….
- la punteggiatura , le parentesi
- alcuni caratteri di controllo , tra cui  invio, tab

rappresentazione sorgente:
'a'  'z'  '9'  ')'  '.'  'è'  'ò'
'\n' = invio  '\t' = tab

# rappresentazione interna dei char

occupano 1 byte

i caratteri corrispondono a interi da -128 a 127 ASCII ESTESA

da 0 a 127 codifica ASCII tradizionale

da -128 a -1  caratteri accentati e altri

# una parte di ASCII esteso

| Decimale | Carattere | Binario | Decimale | Carattere | Binario |
|---|---|---|---|---|---|
| 000 | Null | 00000000 | 128 | Ç | 10000000 |
| 001 | Start of eading | 00000001 | 129 | ü | 10000001 |
| 002 | Start of text | 00000010 | 130 | é | 10000010 |
| 003 | End of text | 00000011 | 131 | â | 10000011 |
| 004 | End of transmit | 00000100 | 132 | ä | 10000100 |
| 005 | Enquiry | 00000101 | 133 | à | 10000101 |
| 006 | Acknowledge | 00000110 | 134 | å | 10000110 |
| 007 | Audible bell | 00000111 | 135 | ç | 10000111 |
| 008 | Backspace | 00001000 | 136 | ê | 10001000 |
| 009 | Horizontal tab | 00001001 | 137 | ë | 10001001 |
| 010 | Line feed | 00001010 | 138 | è | 10001010 |
| 011 | Vertical tab | 00001011 | 139 | ï | 10001011 |
| 012 | Form feed | 00001100 | 140 | î | 10001100 |
| 013 | Carriage return | 00001101 | 141 | ì | 10001101 |
| 014 | Shift out | 00001110 | 142 | Â | 10001110 |
| 015 | Shift in | 00001111 | 143 | Å | 10001111 |
| 016 | Data link escape | 00010000 | 144 | É | 10010000 |
| 017 | Device control 1 | 00010001 | 145 | æ | 10010001 |
| 018 | Device control 2 | 00010010 | 146 | Æ | 10010010 |
| 019 | Device control 3 | 00010011 | 147 | ô | 10010011 |

# ci sono insiemi di caratteri più estesi

esistono altri tipi carattere che comprendono un insieme di caratteri molto più esteso (cinesi e giapponesi)

wchar_t  2 o 4 byte

UTF-8 codifica  a lunghezza variabile

# in ASCII esteso

i valori char sono rappresentati internamente con

'a' = 97, 'b'=98....

quindi

si può usare operazioni + e – con i valori char

ma attenzione a cosa si ottiene!

'a' + 1 = ?? 'b', ma 'a'+'a' ??

# tipo vuoto = void

- non ha valori
- non ha operazioni
- serve a rappresentare l'assenza di valori: al momento giusto lo useremo

# stringhe di caratteri alla C

le stringhe si usano spesso per scrivere a video dei messaggi:

```
cout << "ecco un messaggio\n";
```

oppure

```
cout<<"ecco un messaggio"<<endl;
```