Fondamenti di Programmazione (A)

Lecture 6 - Identificatori, variabili, tipi

Vincenzo Arceri - Università di Parma - vincenzo.arceri@unipr.it

Puntate precedenti

- Problema: dati in input tre numeri interi, calcolare e stampare a video la loro media

Input: tre numeri interi

Output: un numero reale

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

int main() {
    int x, y, z;
    cout << "Inserisci 3 numeri interi" << endl;
    cin >> x >> y >> z;
    float m;
    m = (x + y + z) / 3.0;
    cout << "La media e " << m << endl;
    return 0;
}
```

Identificatori
Variabili
Assegnamenti
Tipi
int
float
...

Identificatori

- Sequenza di caratteri utilizzate all'interno del programma per riferirsi ai componenti che usa (e.g., variabili)

```
return , x , avg
```

  - Identificatori riservati: `return`, `int`, `float` (https://en.cppreference.com/w/cpp/keyword)
  - Identificatori di libreria
  - Identificatori forniti dallo sviluppatore

Identificatori in C++

- Sequenza di caratteri alfanumerici e numerici senza spazi e caratteri speciali (ad eccezione del carattere *underscore* _) che inizia con una lettera o con il carattere *underscore* _

```
return, x, avg, _name, id_a, AVG, Avg
```

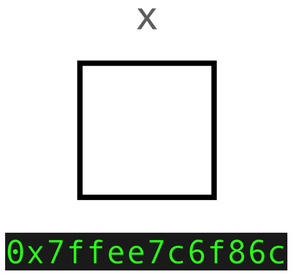
```
1x, 12, !hello, B-1
```

- NB: C++ è *case-sensitive*

```
var ≠ Var ≠ VAR
```

Variabili

- *Astrazione* di una cella di memoria



- **Tipo**: quali dati posso essere memorizzati all'interno della variabile
- **Valore**: il valore contento dalla variabile in un certo momento
- **Indirizzo di memoria/della cella:** a quale indirizzo di memoria si trova la variabile
- Almeno un'operazione di lettura e una di scrittura del valore della variabile

Ad una variabile è spesso associato un **nome,** un identificatore utilizzato per riferirsi in maniera simbolica alla variabile

Variabili

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

int main() {
    int x, y, z;
    cout << "Inserisci 3 numeri interi" << endl;
    cin >> x >> y >> z;
    float m;
    m = (x + y + z) / 3.0;
    cout << "La media e " << m << endl;
    return 0;
}
```

- x, y, z di **tipo** intero
- m di **tipo** float
- Indirizzo di memoria?

Creazione di una variabile

- Allocazione di uno spazio di memoria per contenere valori del tipo specificato
- Opzionale: valore di inizializzazione (valore di partenze della variabile appena creata)

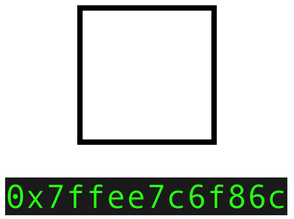
```
#include <iostream>
using namespace std;

int main() {
    int x, y, z;
    cout << "Inserisci 3 numeri interi" << endl;
    cin >> x >> y >> z;
    float m;
    m = (x + y + z) / 3.0;
    cout << "La media e " << m << endl;
    return 0;
}
```

Creazione di tre variabili di tipo intero senza inizializzazione

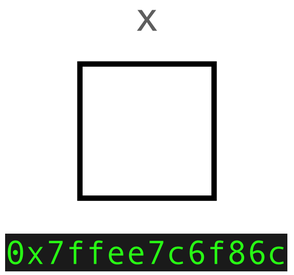
Dichiarazione di variabile in C++

```
int x;
```

- Creazione della variabile (allocazione di memoria)



- Associazione/*binding* fra la variabile e il suo nome



Dichiarazione di variabile in C++

Esempi

```
int x, y, z;
```
dichiarazione di tre variabili di tipo intero non inizializzate

```
float w;
```
dichiarazione di una variabile di tipo float non inizializzata

```
int i = 1;
```
dichiarazione di una variabile di tipo int con inizializzazione

```
int i, j = 1;
```
dichiarazione di due variabili di tipo int, i non inizializzata, j inizializzata

```
int x, y, z;
```
equivalente a
```
int x;
int y;
int z;
```

```
int i, j = 1;
```
equivalente a
```
int i;
int j = 1;
```

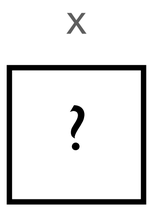
Dichiarazione di variabile in C++

Valore iniziale

NB: In C++, variabili non inizializzate contengono comunque un valore, ma è **indefinito** (dipende dal compilatore, dai bit presenti nella cella allocata per la variabile al momento della dichiarazione…)

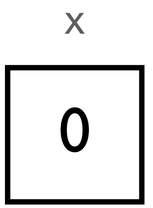
C++

```
int x;
```



Go

```
var x int;
```



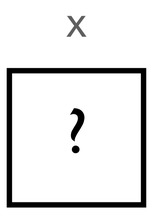
Dichiarazione di variabile in C++

Valore iniziale

NB: In C++, variabili non inizializzate contengono comunque un valore, ma è **indefinito** (dipende dal compilatore, dai bit presenti nella cella allocata per la variabile al momento della dichiarazione…)

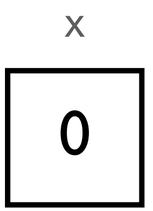
C++

```
int x;
```



Go

```
var x int;
```



In generale, è un errore utilizzare una variabile prima di avergli assegnato un valore esplicitamente

Dichiarazione di variabile in C++

Dove dichiarare le variabili?

- In C++, in qualsiasi punto del programma, in altri linguaggi di programmazione all'inizio del programma

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

int main() {
    int x, y, z;
    cout << "Inserisci 3 numeri interi" << endl;
    cin >> x >> y >> z;
    float m;
    m = (x + y + z) / 3.0;
    cout << "La media e " << m << endl;
    return 0;
}
```

Dichiarazione di variabile in C++

Dove dichiarare le variabili?

- In C++, in qualsiasi punto del programma, in altri linguaggi di programmazione all'inizio del programma

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

int main() {
    cout << "Il valore di x e' "  << x << endl;
    int x = 5;
    return 0;
}
```

Uso

Dichiarazione

ERRORE! Ogni uso di variabile **deve** essere preceduto da una dichiarazione

Tipi

```
int x, y, z;
```

```
float w;
```

- Il **tipo** di una variabile denota l'insieme dei possibili valori che la variabile può contenere e le possibili operazioni applicabili ad essa

- **Tipo primitivo**: prefissato dal linguaggio (e.g., int, float, bool, …)

Il tipo int

```
int x;
```

- Insieme dei numeri **interi** con **segno**
- In C++, int è l'insieme dei numeri interi con segno rappresentabili in complemento a 2 con 32 bit

$$[-2^{31}, 2^{31} - 1]$$

- INT_MIN: $-2^{31} = -2147483648$
- INT_MAX: $2^{31} - 1 = 2147483647$
- Costanti intere: 5, 7, -12,…

Il tipo int

Operazioni

- Operazioni aritmetiche di base. Il risultato è di tipo `int`

```
x + y    x - y    x * y    x / y
```

  - Divisione **intera** con troncamento del risultato

```
5 / 2 = 2
```

- Operazione modulo (resto della divisione intera). Il risultato è di tipo `int`

```
5 % 2 = 1
```

- Operazioni di confronto. Il risultato è di tipo `bool`

```
x == y    x != y    x < y    x > y    x >= y    x <= y
```

Il tipo float

```
float x;
```

- Insieme dei numeri **reali** con **segno**
- In C++, float è l'insieme dei numeri reali con segno rappresentabili in virgola mobile (seguendo il formato IEEE 754) con 32 bit
- Costanti float: -5.2, 0.33333, 3.8, 5.0,…

Il tipo float

Operazioni

- Operazioni aritmetiche di base. Il risultato è di tipo `float`

```
x + y    x - y    x * y    x/y
```

- Operazioni di confronto. Il risultato è di tipo `bool`

```
x == y    x != y    x < y    x > y    x >= y    x <= y
```

Il tipo double

```
double x;
```

- Insieme dei numeri **reali** con **segno** con precisione **doppia** rispetto ai `float`
  - `float`: 32 bit
  - `double`: 64 bit
- Le operazioni applicabili ai `double` sono le stesse dei `float`

Il tipo char

```
char c;
```

- Insieme dei caratteri **alfabetici, numerici** e **speciali** rappresentabili tramite la codifica ASCII

Codifica ASCII

- Codifica dei caratteri utilizzando 7 bit
  - 97 codifica il carattere *a*
  - 65 codifica il carattere *A*
  - 9 codifica il carattere *\t* (carattere di tabulazione)
  - In C++ un `char` occupa 1 byte (8 bit, da -128 a 127)

Il tipo char

Costanti carattere

- Costanti carattere sono denotate fra **singoli apici**
    - Singolo carattere (*nella tabella ASCII da 32 e 126*): 'a', '4', 'A', …
    - Caratteri di controllo (*nella tabella ASCII da 0 a 31 e il carattere 127*)

      '\n': carattere di *new line* (carattere "a capo")

      '\t': carattere di tabulazione

ASCII Art

Immagini create utilizzando esclusivamente i caratteri ASCII

Lanciate `fortune | cowsay` da terminale…

Il tipo char

Operazioni

- Operazioni di confronto. Il risultato è di tipo `bool`

`x == y` `x != y` `x < y` `x > y` `x >= y` `x <= y`

  - L'ordinamento segue la codifica ASCII

    'a' codificato come 97, '4' codificato come 52, quindi '4' < 'a'

- Operazioni aritmetiche di base. Il risultato è di tipo `char`

`x + y` `x - y` `x * y` `x/y` `x % y`

- I caratteri in C++ non sono altro che **interi** (tipicamente -128 a 127)

  posso vedere char come int e (alcuni) int come char

Esempio

- Problema: dati un carattere minuscolo in input, stampare a video il corrispondente carattere maiuscolo

Input	Output
a	A
c	C
z	Z

Il tipo bool

Operazioni

```
bool b;
```

- Insieme dei valori di verità `true` e `false`
- Operazioni logiche di base. Il risultato è di tipo `bool`

```
x && y     x || y     !x
```

- && Operatore AND: entrambi gli operandi devono essere `true`
- || Operatore OR: uno dei due operandi deve essere `true`
- ! Operatore NOT: ritorna come risultato la negazione dell'operando

Il tipo bool

Operazioni

```
bool b;
```

- Insieme dei valori di verità `true` e `false`
- Operazioni logiche di base. Il risultato è di tipo `bool`

`x && y` `x || y` `!x`

&& (and)	`true`	`false`
`true`	`true`	`false`
`false`	`false`	`false`

|| (or)	`true`	`false`
`true`	`true`	`true`
`false`	`true`	`false`

`!true = false`

`!false = true`

Il tipo bool

Operazioni

- Operazioni aritmetiche di base. Il risultato è di tipo `bool`

```
x + y    x - y    x * y    x/y    x % y
```

- Operazioni di confronto. Il risultato è di tipo `bool`

```
x == y    x != y    x < y    x > y    x >= y    x <= y
```

- Simile al caso di `char`: `true` viene trattato come 1, `false` viene trattato come 0

Tipi

Riassunto

`int`	`float`	`char`	`bool`
4 byte	4 byte	1 byte	1 byte
(32 bit)	(32 bit)	(8 bit)	(8 bit)

Wikipedia: *"Historically, a byte was the number of bits used to encode a single character of text in a computer and it is for this reason the* ***basic addressable element*** *in many computer architectures"*

Modificatori di tipo

- Modificano un tipo di base
  - Segno
    - ✦ `signed`: il tipo da modificare avrà una rappresentazione con segno
    - ✦ `unsigned`: il tipo da modificare avrà una rappresentazione senza segno
  - Dimensione
    - `short`: modifica il minimo valore che il tipo da modificare può contenere
    - `long`: modifica il massimo valore che il tipo da modificare può contenere

Modificatori di tipo

Riassunto

Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
`short`	`short int`	at least **16**	**16**	**16**	**16**	**16**
`short int`						
`signed short`						
`signed short int`						
`unsigned short`	`unsigned short int`					
`unsigned short int`						
`int`	`int`	at least **16**	**16**	**32**	**32**	**32**
`signed`						
`signed int`						
`unsigned`	`unsigned int`					
`unsigned int`						
`long`	`long int`	at least **32**	**32**	**32**	**32**	**64**
`long int`						
`signed long`						
`signed long int`						
`unsigned long`	`unsigned long int`					
`unsigned long int`						

https://en.cppreference.com/w/cpp/language/types