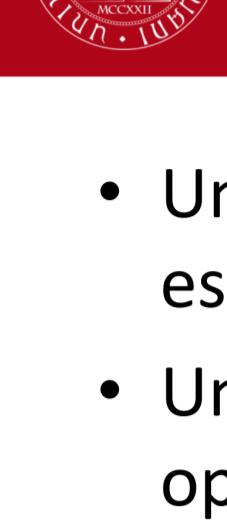


UNIVERSITÀ DEGLI STUDI DI PADOVA

Espressioni e operatori

Stefano Ghidoni



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

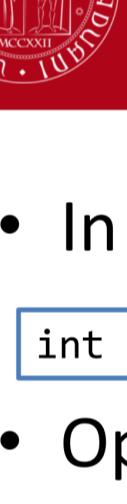
Agenda

- Espressioni
- Operatori con side effect
- Ordine di valutazione di un'espressione

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Espressioni

- Il più piccolo elemento usato per esprimere la computazione è l'**espressione**
- Un'espressione calcola un risultato a partire da uno o più operandi
- Espressioni semplici:
 - Literal: 10, 'a', 3.14, "Norah"
 - Nomi di variabili
- Espressioni semplici possono essere combinate con operatori per formare espressioni più complesse



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Lvalue

- Un elemento molto importante nelle espressioni è l'**lvalue** (left value)
- Un lvalue è ciò che sta alla sinistra di un operatore di assegnamento
 - Variabili
- Un nome di variabile è **sempre** un lvalue?

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Lvalue vs rvalue

- Length si riferisce:
 - Come lvalue: a un oggetto di tipo int che contiene il valore 99 – **length è il box (l'oggetto)**
 - Come rvalue: il riferimento al **valore contenuto** nell'oggetto
- In un'espressione, length può essere usato sia come lvalue che come rvalue



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

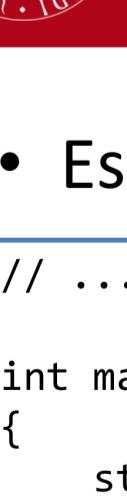
int: length = 99;

length: 99

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Espressioni costanti

- Il software ha bisogno di molte espressioni costanti
- Il C++ permette di esprimere una costante simbolica



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

```
constexpr double pi = 3.14159; // errore!
double c = 2 * pi * r; // OK: sola lettura
```

- Meglio dei magic numbers / magic constants
- Meglio dei #define (macro stile C) perché le variabili costanti hanno un tipo

non ne definisce il tipo

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Constexpr vs const

- Due modi per esprimere una costante
 - Constexpr: il valore è noto a tempo di compilazione
 - Const: il valore può essere noto solo a tempo di esecuzione ed è assegnato in inizializzazione



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

```
constexpr int max = 100;
void use (int n)
{
    constexpr int c1 = max + 7; // OK: da costanti
    const int c2 = n + 7; // OK
    c2 = 7; // errore
}
```

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Valutazione delle espressioni

- In che ordine sono valutate le espressioni?

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

```
int perimeter = (length + width) * 2;
```

- Operazioni:
 - Lettura di length
 - Lettura di width
 - Somma
 - Prodotto
 - Assegnamento

Quale delle due è svolta per prima?

- Sono sensibili al tipo

lo standard non lo decide, se ne occupa il compilatore

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Valutazione delle espressioni

- L'ordine di valutazione delle espressioni non è definito
 - È sbagliato usare la stessa variabile più di una volta se su essa sono applicati operatori con side effect

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

```
v[i] = ++i; // Ordine di valutazione non definito:
// i a sinistra è letto prima o dopo la
// valutazione di ++i?

v[++i] = i; // Ordine di valutazione non definito:
// Come sopra
```

- Note:
 - Anche per i suoi operandi non è definito quale dei due sia valutato per primo

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Valutazione delle variabili

- Maggiori possibilità di ottimizzazione se il compilatore ha qualche grado di libertà
 - Libertà nell'ordine di valutazione delle espressioni
- Libertà del compilatore: comportamento indefinito quando non sono fissate le regole

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

```
int x = ++i + ++i; // Ordine di valutazione non definito:
// lettura e incremento potrebbero
// non essere consecutivi

cout << ++i << ' ' << i << '\n'; // Ordine di valutazione
// non definito
```

- Note:
 - È sbagliato usare la stessa variabile più di una volta se su essa sono applicati operatori con side effect

non ne definisce il tipo

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Valutazione delle espressioni

- Esempio: operatore >> sensibile al tipo

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

```
// ...

int main()
{
    std::cout << "Please enter your first name and age\n";
    string first_name;
    int age;
    std::cin >> first_name;
    std::cin >> age;
    std::cout << "Hello, " << first_name << " (age " << age <<
")\n";
    return 0;
}
```

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Operatore << e >>

- Alcuni operatori hanno un funzionamento che dipende dai tipi degli operandi
 - Un risultato → è sempre presente, anche se per molti operatori viene ignorato
- Alcuni operatori hanno un **side effect**
 - Modificano gli operandi su cui operano
 - Forniscono un risultato
- Es: hanno side effect ++, --, +=, -=, ...

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

```
v[i] = ++i; // Ordine di valutazione non definito:
// i a sinistra è letto prima o dopo la
// valutazione di ++i?

v[+i] = i; // Ordine di valutazione non definito:
// Come sopra
```

- Note:
 - Anche per i suoi operandi non è definito quale dei due sia valutato per primo

non ne definisce il tipo

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Valutazione delle espressioni

- L'ordine di valutazione delle espressioni non è definito
 - È sbagliato usare la stessa variabile più di una volta se su essa sono applicati operatori con side effect

non ne definisce il tipo

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Altri operatori sensibili al tipo

- Note:
 - Anche per i suoi operandi non è definito quale dei due sia valutato per primo

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

```
int count;
std::cin >> count;
std::string name;

int c2 = count + 2; // add integer
std::string s2 = name + " Jr."; // append string

int c3 = count - 2; // subtracts integer
std::string c3 = name - " Jr."; // error: not defined for
// strings
```

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Valutazione delle espressioni

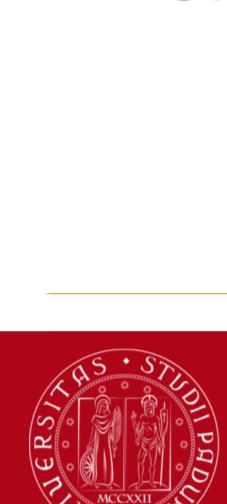
- Gli operatori sono caratterizzati da:
 - 1, 2, 3 operandi
 - Un risultato → è sempre presente, anche se per molti operatori viene ignorato

non ne definisce il tipo

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Espressioni e operatori

Stefano Ghidoni



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Espressioni e operatori