Funtori e Lambda - Approfondimento

Funtori

- In C++, un funtore (o oggetto funzione) è un oggetto che può essere chiamato come una funzione.
- I funtori sono definiti implementando l'operatore `operator()` in una classe o struttura.
- I funtori possono mantenere uno stato interno, a differenza dei puntatori a funzione.
- Esempio di funtore:

```
class Multiply {
public:
    Multiply(int m) : mult(m) {}

    int operator()(int x) {
        return mult * x;
    }
private:
    int mult;
};
```

- I funtori sono spesso usati con algoritmi della STL come `std::transform`, `std::accumulate`, etc.

Lambda

- In C++11 sono state introdotte le espressioni lambda, che permettono di definire funzioni anonime inline.
- La sintassi generale è `[cattura](parametri) -> tipo_di_ritorno { corpo }`.
- La clausola di cattura specifica quali variabili esterne sono accessibili nel corpo della lambda e come (per valore o per riferimento).
- Esempio di lambda:

```
int main() {
    std::vector<int> v = {1, 2, 3, 4, 5};
    int sum = 0;
```

```
    std::for_each(v.begin(), v.end(), [&sum](int x) {
        sum += x;
    });

    std::cout << "Somma: " << sum << std::endl;
}
```

- Le lambda possono essere catturate in variabili auto e passate come argomenti a funzioni che accettano funtori.

I funtori e le lambda in C++ forniscono un potente strumento per la programmazione generica e per scrivere codice più conciso ed espressivo, specialmente quando combinati con algoritmi della libreria standard.

Esempi di codice

1. Utilizzo di un funtore con `std::transform`:

```
#include <iostream>
#include <vector>
#include <algorithm>

class Square {
public:
    int operator()(int x) {
        return x * x;
    }
};

int main() {
    std::vector<int> v = {1, 2, 3, 4, 5};
    std::vector<int> squares;

    std::transform(v.begin(), v.end(), std::back_inserter(squares),
Square());

    for (int x : squares) {
        std::cout << x << " ";
    }
    std::cout << std::endl;
}
```

2. Utilizzo di una lambda con `std::sort`:

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> v = {5, 2, 4, 1, 3};

    std::sort(v.begin(), v.end(), [](int a, int b) {
        return a > b;
    });

    for (int x : v) {
        std::cout << x << " ";
    }
    std::cout << std::endl;
}
```

3. Cattura di variabili in una lambda:

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> v = {1, 2, 3, 4, 5};
    int threshold = 3;

    auto count = std::count_if(v.begin(), v.end(), [threshold](int x) {
        return x > threshold;
    });

    std::cout << "Ci sono " << count << " elementi maggiori di " <<
threshold << std::endl;
}
```

4. Passaggio di una lambda come argomento di funzione:

```
#include <iostream>
#include <vector>
```

```cpp
#include <algorithm>

void applyFunction(std::vector<int>& v, const std::function<int(int)>& f)
{
    for (int& x : v) {
        x = f(x);
    }
}

int main() {
    std::vector<int> v = {1, 2, 3, 4, 5};

    applyFunction(v, [](int x) {
        return x * x;
    });

    for (int x : v) {
        std::cout << x << " ";
    }
    std::cout << std::endl;
}
```