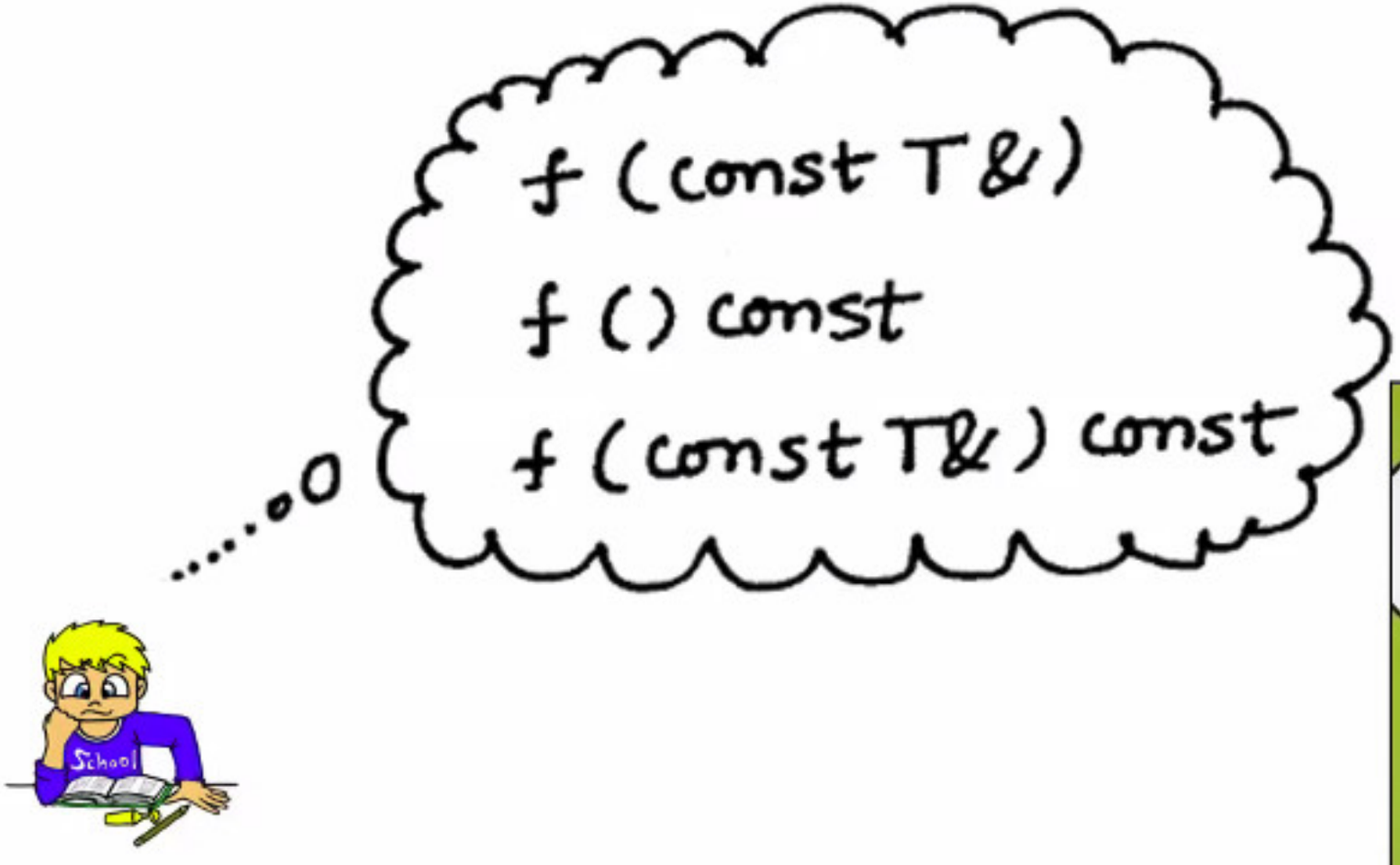
Metodi e oggetti di invocazione costanti



```
class orario {
public:
   void StampaSecondi() const;
   ...
};
```

```
void orario::StampaSecondi() const {
   std::cout << sec << std::endl;
};
```

Il compilatore controlla che nella definizione del metodo dichiarato **`const`** non compaia alcuna istruzione che possa provocare side-effect sull'oggetto di invocazione: assegnazioni ai campi dati dell'oggetto di invocazione ed invocazioni di metodi non costanti.

Spiegazione: in un metodo costante di una classe **`C`**, il puntatore **`this`** ha tipo **`const C*`**

Anche un oggetto può essere dichiarato costante:

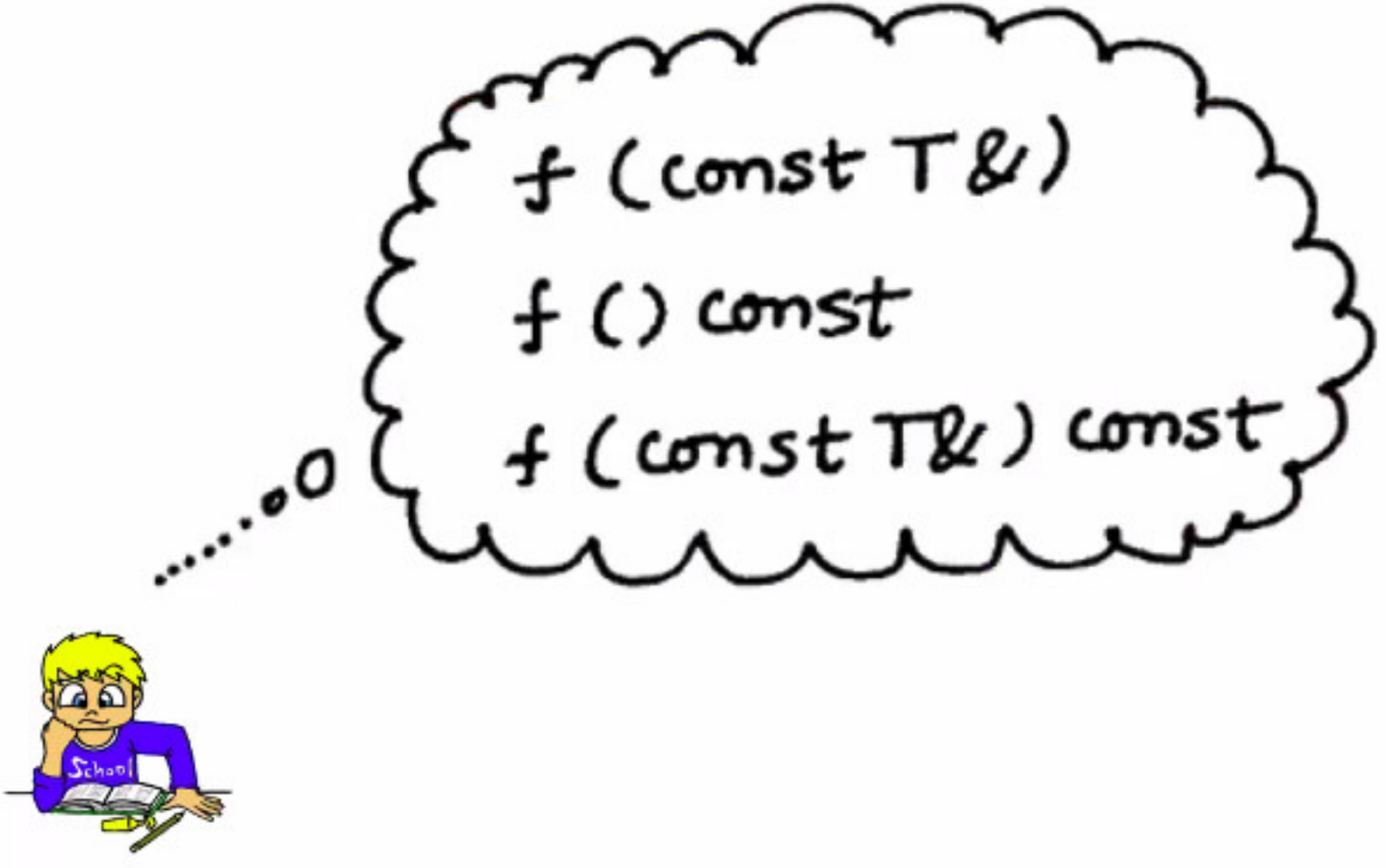
```
const orario LE_DUE(14); // è l'analogo di
const int tre = 3;
```

Attenzione: un oggetto costante si può usare come oggetto di invocazione **soltanto** per metodi dichiarati costanti.

```
const orario LE_TRE(15);
LE_TRE.StampaSecondi(); // OK: stampa 54000
orario t;
t = LE_TRE.UnOraPiuTardi(); // Errore! Non compila!
   // UnOraPiuTardi() non è stato
   // dichiarato come metodo costante
```

Eccezione (ovvia): i costruttori sono dei metodi non dichiarati costanti che possono venire invocati su oggetti dichiarati costanti!

C++: The const Mantra



The **const mantra** to keep in mind while writing C++ code is to **make everything const,** that is, as much as possible.

const saga

Quora Search for questions, people, and topics

C++ (programming language) Programming Languages

Is C++ really as hard as people say?

56 Answers

C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg. – Bjarne Stroustrup

If C++ has taught me one thing, it's this: Just because the system is consistent doesn't mean it's not the work of Satan. – Andrew Plotkin

I invented the term Object-Oriented, and I can tell you I did not have C++ in mind. – Alan Kay

C++ damages the brain ... – EWD

C++ is an insult to the human brain – Niklaus Wirth

Life is too long to know C++ well. – Erik Naggum

C is C++ without the BS. – SocialPhatology

const saga

```
int x=2;
int& a = x;   // ALIAS
int& a1 = 2; // ILLEGALE
a=5;
int y=3;
a=y;          // LEGALE, r-valore di y assegnato a
              // a l-valore di x
```

```
int x=2;
int* p = &x;
*p=5;
int y=3;
p=&y;         // LEGALE
```

const saga

```
int x=2;
int * const p = &x;
*p=5;     // LEGALE
int y=3;
p=&y;     // ILLEGALE
```

```
int x=2;
int & const r = x; // ILLEGALE (tipo illegale)
```

const saga

```
int x=2;
const int* p = &x;
*p=5;     // ILLEGALE
int y=3;
p=&y;     // LEGALE
```

```
int x=2;
const int *const p = &x;
*p=5;     // ILLEGALE
int y=3;
p=&y;     // ILLEGALE
```

const saga

```
int x=2;
const int& r = x; // RIFERIMENTO A TIPO COSTANTE
r=5;              // ILLEGALE
int y=3;
r=y;              // ILLEGALE
```

```
const int& r = 4; // LEGALE
r=5;              // ILLEGALE
int y=3;
r=y;              // ILLEGALE
```

```
const int & const r=2; // ILLEGALE
```

const saga

```
void fun(const int& r);
// PASSAGGIO PER RIFERIMENTO (A TIPO) COSTANTE
int x=2;
fun(x); // LEGALE
fun(4); // LEGALE
```

```
const int& fun() { return 4; /* LEGALE */ }
// RITORNA RIFERIMENTO (A TIPO) COSTANTE

fun()=5;      // ILLEGALE
```

const saga

```
void fun_ref(const int& r);
// VERSUS
void fun_ptr(const int* p);
int x=2;
fun_ref(x);
// VERSUS
fun_ptr(&x);

fun_ref(4);   // LEGALE
// VERSUS
fun_ptr(&4); // ILLEGALE
```

const saga

```
void fun1(const Big& r); // PER RIFERIMENTO COSTANTE
// VERSUS
void fun2(Big v);        // PER VALORE

Big b(...);
fun1(b);     // copia di un riferimento a Big
// VERSUS
fun2(b);     // costruttore di copia di Big
```

const saga

FAQ What is "const correctness"?

A good thing. It means using the keyword `const` to prevent `const` objects from getting mutated.

FAQ How is "const correctness" related to ordinary type safety?

Declaring the `const`-ness of a parameter is just another form of type safety.

If you find ordinary type safety helps you get systems correct (it does; especially in large systems), you'll find `const` correctness helps also.

const saga

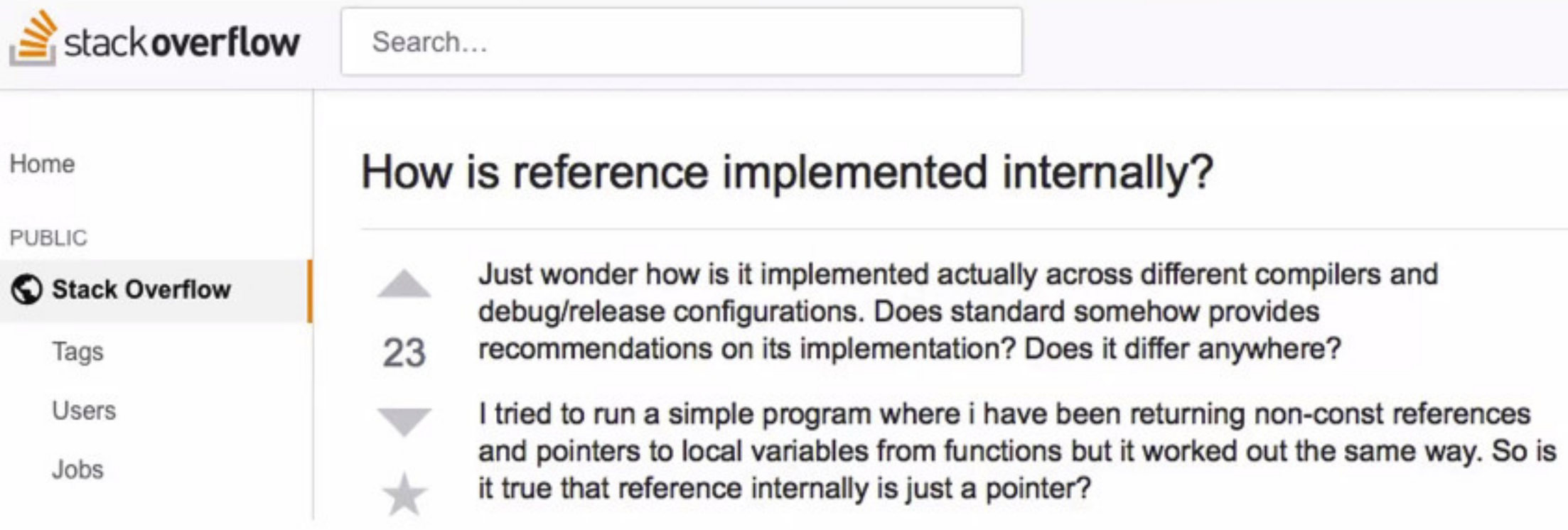
Come sono implementati i reference?

Lo standard C++ non lo prevede, dipende quindi dal compilatore.

In pratica: (quasi sempre) mediante puntatori

const saga

Come sono implementati i reference?



const saga

Come sono implementati i reference?

In Bjarne's words:

Like a pointer, a **reference** is an alias for an object, is usually implemented to *hold a machine address* of an object, and does not impose performance overhead compared to pointers, but it differs from a pointer in that:

- You access a reference with exactly the same syntax as the name of an object.
- A reference always refers to the object to which it was initialized.
- There is no “null reference,” and we may assume that a reference refers to an object

---

Though a ***reference*** is in reality a *pointer*, but it shouldn't be used like a *pointer* but as an *alias*.

New Share | Pause Share | Annotate | More

You are screen sharing 01:15:51 Stop Share

const saga

Esercizio (prevalentemente di Programmazione)

```
class C {
private:
  int x;
public:
  C(int n = 0) {x=n;}
  C F(C obj) {C r; r.x = obj.x + x; return r;}
  C G(C obj) const {C r; r.x = obj.x + x; return r;}
  C H(C& obj) {obj.x += x; return obj;}
  C I(const C& obj) {C r; r.x = obj.x + x; return r;}
  C J(const C& obj) const {C r; r.x = obj.x + x; return r;}
};

int main() {
  C x, y(1), z(2); const C v(2);
  z=x.F(y); // OK
//! v.F(y); // ILLEGALE: "passing const C as this discards qualifiers"
  v.G(y); // OK
  (v.G(y)).F(x); // OK
  (v.G(y)).G(x); // OK
//! x.H(v); // ILLEGALE: "no matching function for call to C::H(const C&)"
//! x.H(z.G(y)); // ILLEGALE (!!): no matching function for call to C::H(C)
  x.I(z.G(y)); // OK (nota bene!)
  x.J(z.G(y)); // OK
  v.J(z.G(y)); // OK
}
```