A COLLOQUIO CON IL PAPÀ DEL C++

INTERVISTA ESCLUSIVA AD UNO DEI PERSONAGGI PIÙ IMPORTANTI DELLA STORIA NELL'IT. IL PADRE DEL C++ PARLA DEL PASSATO, PRESENTE E FUTURO DI UNO DEI LINGUAGGI STORICI DELL'INFORMATICA

ioProgrammo: Grazie per la tua disponibilità, ci puoi parlare dei tuoi impegni attuali, sia dal punto di vista lavorativo, che prettamente informatico?

Bjarne Stroustrup: Sto lavorando per completare il C++0x (il prossimo standard ISO C++), il cui nome è ancora in fase di definizione, porto avanti alcune ricerche universitarie, e insegno agli studenti universitari il C++ utilizzando il mio nuovo libro, *Programming – Principles and Practice using C++*. Le ricerche riguardano principalmente l'analisi statica e la trasformazione per i programmi in C++, mentre per quanto riguarda il C++0x sto già producendo tutorial (come la FAQ che ha già reso disponibile sul suo sito, ndr) e ulteriori articoli riguardanti le nuove funzionalità che sto studiando.

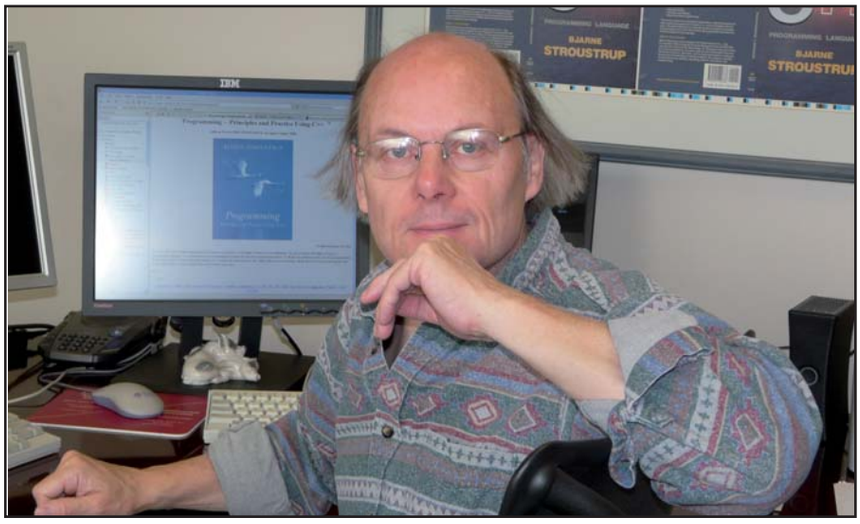

Fig. 1: Bjarne Stroustrup, padre nobile della programmazione moderna

ioP: Nel 2009 si celebra il trentennale della nascita del C++ cosa è cambiato nel mondo dell'IT in questi tre decenni?

BS: Ne è passato di tempo! Il mondo dell'IT è molto cambiato, e alcuni eventi sono avvenuti anche grazie a questo linguaggio. Per fortuna con il passare degli anni il mio linguaggio si è evoluto in modo da rimanere sempre competitivo e utile in svariati settori. L'architettura del C++ ha in un certo modo anticipato quella rivoluzione dei PC, sia dal punto di vista di memoria che di velocità dei processori, che abbiamo visto in questi ultimi anni. Certo questo incremento esponenziale non era così prevedibile. Non ho avuto la lungimiranza di prevedere l'esplosione delle web applications. Ho sempre posto la mia attenzione sull'aspetto infrastrutturale dell'IT, parliamo quindi di *system programming* e *embedded applications*. Quando iniziai, tutti quei concetti che oggi sono comuni, compilatori e sistemi operativi in primis, erano solo ipotesi di possibili futuri. In questi anni browser, virtual machine, e engine di giochi destano molto interesse, sebbene, guardando in profondità, si basano sempre sulle "vecchie" tecnologie. Sono felice che il C++ si sia mantenuto coerente nel tempo e sia stato in grado al contempo di evolversi.

ioP: Quando si tratta delle origini del C++, bisogna guardare indietro verso *Simula*, e considerare *Smalltalk* come un parente, c'è un "figlio legittimo" del tuo linguaggio?

BS: Essenzialmente tutti i linguaggi moderni, almeno tutti quelli che nella loro descrizione dichiarano di essere di tipo "Object Oriented Programming", devono qualcosa a *Simula*. Troppi programmatori dimenticano questa verità e dichiarano di aver inventato qualcosa di nuovo, ma in realtà hanno semplicemente "reinventato la ruota". Ovviamente la base su cui è stato costruito il C++ è il C; purtroppo l'informatica ha la memoria corta, e questo è segno di forte immaturità, che inoltre rende molto difficile instaurare una corretta comunicazione tra i vari programmatori: non ci si può definire un informatico senza sapere l'apporto all'informatica di *Ole-Johan Dahl*, *Kristen Nygaard*, e di *Dennis Ritchie*. Molti linguaggi di oggi dichiarano la loro anima object-oriented solo perché utilizzano l'ereditarietà e i passaggi di parametri per valore o riferimento, il C++ invece utilizza anche costruttori/distruttori, overloading, variabili locali e altre tecniche: è più completo in questo senso. Inoltre il C++ consente di accedere direttamente all'hardware garantendo massima libertà di programmazione. Ci sono veramente pochi linguaggi che possono essere considerati figli del C++, e molti traggono

spunto da questo senza neppure dichiararlo: spero un giorno di potermi accorgere di non aver notato il "vero figlio" del C++ quando mi è passato davanti agli occhi. Qualcosa di simile accadde con Kristen Nygaard , il padre della programmazione e della progettazione ad oggetti: ad una conferenza disse "ho atteso un ragazzo che potesse portare la fiamma di Simula, e questo ragazzo e Bjarne", e mi abbracciò: era un gigante sia fisicamente che come persona.

ioP: C++ è un linguaggio multi-paradigma che supporta la programmazione a oggetti, quali aspetti ritieni siano stati sottovalutati?

BS: L'utilizzo dei costruttori e dei distruttori meritava maggiore attenzione: è alla base di molte tecniche moderne di programmazione in C++. I puntatori "smart" proteggono dall'uso indisciplinato dei puntatori; il *RAII* (*Resource Acquisition Is Initialization*) semplifica la gestione di risorse di alto livello, come lock, socket , file handler e transazioni.

ioP: C'è qualcosa che avresti voluto inserire nel tuo linguaggio?

BS: Molte tecniche sono già disponibili come librerie esterne (ad esempio il tipo di dati a precisione infinita) e altri sono generalmente relegati ad alcune caratteristiche di alcuni sistemi su cui il linguaggio viene eseguito (ad esempio la *perfect type safety*). Se dovessi scegliere, punterei il dito sui multi-metodi (*multi-dispatch*), la possibilità quindi di identificare un metodo in base al tipo di tutti i parametri passati (il C++ ne utilizza uno solo, è single-dispatch, ndr), ad esempio, chiamando il metodo *Intersezione(c,t)* dove *c* è ti tipo *Cerchio* e *T triangolo*, verrà scelto quel preciso metodo se ne esistono tanti omonimi disponibili. Ho pensato a questo aspetto per oltre un decennio e ho trovato forse un modo per supportarlo. I multimetodi sono inoltre dal punto di vista delle prestazioni più performanti di due chiamate virtuali e il codice è di dimensioni minori. Ovviamente, molte delle carenze del C++ sono state ora risolte nel C++0x e compaiono in GCC C++0x (la 4.5 è al momento la più completa), IBM C++ e nel compilatore Microsoft C++, parliamo ad esempio di *concorrenza, uniform* e *general initialization*, *general constant expressions*, *efficient smart pointers*, *regular expressions*, etc.

ioP: Cosa troveremo in più nel nuovo linguaggio C++0x?

BS: Sul mio sito è disponibile una FAQ *What is C++0x?* Per brevità mostrerò le novità più interessanti. Consideriamo un tipico loop:

```
for(std::list<Elem>::iterator p = lst.begin();
        p!=lst.end(); ++p) cout << *p << "\n";
```

in C++0x si potrà realizzarlo in questo modo:

```
for(auto p = lst.begin(); p!=lst.end(); ++p)
                                cout << *p << "\n";
```

auto permette di impostare automaticamente il tipo di una variabile in base a come viene inizializzatato. Realizzai questa funzionalità per il C++ nel lontano inverno del 1983/84 ma per problemi di incompatibilità con il C decisi di rimuoverla.
Un altro esempio è il seguente:

```
for(auto x : lst) cout << x << "\n";
```

che può essere interpretato come "per ogni elemento *x* nella sequenza *lst* esegui …". Il C++ mancava di strumenti standard per gestire la concorrenza in ambienti multicore, ora il C++0x consente di avere un preciso controllo su questo aspetto: tipi di dato atomici per programmazione concorrente a basso livello, finalmente una classe per il *multithread, mutex, condition variables, locks, async(), future* e *promise* per lo sviluppo a più alto livello. Ecco un esempio di come adoperare *async()* per eseguire tasks concorrenti e il tipo di dato *future* per gestire il risultato:

```
double accum(double* b, double* e, double
      init); // compute the sum of [b,e) and init
// se v è sufficientemente grande analizzalo
                                  con dei thread.
double comp(vector<double>& v)          {
if (v.size()<10000) return accum(&v[0],
                                  &v[0]+v.size(), 0.0);

          // esegue un task su ogni quarto div:
          auto f0 = async(accum, &v[0],
                                  &v[v.size()/4], 0.0);
          auto f1 = async(accum,
                    &v[v.size()/4], &v[v.size()/2], 0.0);
          auto f2 = async(accum,
                    &v[v.size()/2], &v[v.size()*3/4], 0.0);
          auto f3 = async(accum,
                    &v[v.size()*3/4], &v[0]+v.size(), 0.0);

          return
                    f0.get()+f1.get()+f2.get()+f3.get();
}
```

async() avvia task concorrenti, monitorizzando inoltre numero di thread in uso, e il numero di core utilizzati per ottimizzare la sua esecuzione, tutto in maniera automatica; *get()* attende il risultato.

ioP: Quando venne rilasciato, il C++ era uno dei pochi linguaggi disponibili e venne largamente utilizzato, ora ogni anno vediamo decine di nuovi linguaggi affacciarsi dichiarando di essere migliore, più performante e robusto: dobbiamo apprezzare ciò?

BS: Il numero di linguaggi rilasciati negli ultimi tre decenni è molto alto: mediamente 200 all'anno! Apprezzo il fatto che molti di questi cercano di esplorare aree non ancora coperte da altri, quello che non approvo è realizzare linguaggi con la minima innovazione che però obbligano a sviluppare su un numero molto limitato di piattaforme: fanno molti danni in termini professionali, rendendo poco riutilizzabili le nozioni acquisite, impedendo anche uno scambio valido di informazioni tra diverse persone, per non parlare dei problemi legati al loro apprendimento che può condurre ad una certa confusione.

ioP: C'è un linguaggio che avresti voluto creare, oltre ovviamente al tuo C++?

BS: Vorrei realizzare un linguaggio type-safe e più leggero, con meccanismi di astrazione performante, vere variabili locali per tutti i tipi e tanti altri aspetti. Tutto è fattibile, ma non semplice: ne ho parlato approfonditamente nella parte finale nel della conferenza *ACM History of Programming Languages conference* (HOPL-3). È importante sottolineare che il C++ attuale è molto più performante della prima versione rilasciata, e il C++0x sarà ancora più adatto per realizzare applicazioni di qualunque tipo.

ioP: Java è divenuto alla fine del millennio sinonimo di linguaggio Object Oriented

Fig. 2: Stroustrup ha accolto con entusiasmo il nostro invito

Machine e Garbage Collectors hanno forse completamente allontanato le nuove generazioni da una consapevolezza di cosa c'è sotto tutti questi automatismi? C e C++ sono ormai visti come estremamente difficili.

ed è utilizzato in molte università italiane come unico mezzo per insegnare questa tecnica di programmazione. È un linguaggio "lontano" dall'hardware, molte risorse e molti non badano a come si realizza il proprio codice in termini di prestazioni, tale aspetto sembra quasi superato dalle grandi risorse ora disponibili con i PC. Che ne pensi a riguardo?

BS: Non è purtroppo colpa di un linguaggio se i curriculum-vitae si sono ridotti a poche righe. Java si è rivelato molto utile in campo lavorativo: per un datore di lavoro un programmatore che conosce solo un linguaggio è sicuramente più economico; inoltre molti pensano che porre la corretta attenzione su algoritmi, performance, e problematiche varie sia ormai inutile. Sun ha ascoltato e ha risposto con il suo linguaggio-panacea. Non tutti dovrebbero essere formati sugli aspetti più "fini", considerando il fatto che qualcuno ha già passato anni per risolvere alcuni problemi e presentare soluzioni performanti (di nuovo, non reinventiamo la ruota), ma almeno un sottoinsieme merita di essere adeguatamente istruito. Anche se in alcuni contesti molti programmatori Java e .NET hanno creato software di grande qualità, si stanno perdendo quelle conoscenze a livello più basso, come nel campo embedded, che per molti sembra essere il passato, ma che in realtà sarà il futuro: si pensi inoltre che molte infrastrutture software esistenti, anche nuove, sono in fondo realizzate in C o C++.

ioP: Se si chiedesse ad un campione di giovani programmatori come si muoverebbero in campo embedded, ci sarebbe credo un certo imbarazzo. Virtual

BS: Hai ragione, la paura è dovuta alla mancanza di conoscenza, e il livello di istruzione è diminuito. Inoltre, utilizzare il C++ per questi utenti non deve solo significare "convertire un programma da Java a C++/C", bensì dovrebbe corrispondere ad uno studiare, metabolizzare e adoperare tutte quelle tecniche fornite da questi due "vecchi" linguaggi. Molti pensano che il C++ sia ancora la versione del 1985!

ioP: L'ondata di baby-sviluppatori motivati dal fenomeno Apple Store ha incrementato di molto la popolarità dell'Objective-C, cosa ne pensi di questo linguaggio?

BS: Objective C è purtroppo solo un modo che Apple ha trovato per incatenare gli utenti alla sua piattaforma, impedendo la portabilità su altri sistemi, è un linguaggio comunque molto intuitivo.

ioP: Pensi esista un iter ideale per divenire un programmatore "completo"?
BS: Purtroppo non esiste un tragitto corretto, penso che sia fondamentale la passione per la programmazione, che dovrebbe poi condurre verso un amore per il pensiero logico che consente di analizzare e risolvere problemi nelle aree di interesse.
I miei consigli? Studiare almeno due linguaggi di programmazione, se non più, con un livello di conoscenza elevato, leggete molto codice e documentazione, non siate ottusi o estremisti, imparate da qualcuno e, soprattutto, scrivete migliaia di righe di codice. Il vostro primo codice sarà terribile, ma perseverate sempre, otterrete i risultati che desiderate con il tempo. Ricordate che un linguaggio deve permettere di trovare soluzioni a vari problemi, non esistono "buoni" o "cattivi" linguaggi, esistono linguaggi validi in alcuni contesti, e meno in altri.

ioP: Quando un programmatore accorto deve iniziare a conoscere un linguaggio è bene che consulti diversi libri e legga molta documentazione. Purtroppo accade che molti si basano su risorse web di dubbia origine, frutto a volte di numerosi passi di copia/incolla: un libro è sempre il migliore modo per iniziare e il web il secondo passo...

BS: Certo, utilizzare i cosiddetti "how To" non è altro che filtrare informazioni già rimaneggiate e interpretate da non si sa quante persone: chi si professa "esperto" deve maturare una conoscenza profonda, deve sapere perché una tecnica è migliore dell'altra e effettuare una scelta consapevole. Le risorse web diventano utili quando si vuole confrontarsi, completare il proprio iter formativo e valutare la propria cultura.

ioP: A parte il tuo "The C++ Programming Language", quale libro consiglieresti?

BS: Il mio *Principles and Practice using C++* per iniziare; poi *Sutter and Alexandrescu's C++ Coding Standards*: *101 Rules, Guidelines, and Best Practices*, per gli algoritmi, *Stepanov and McJones: Elements of programming*, per la grafica qualunque manuale su QT e/o OpenGL, *Anthony Williams' C++ Concurrency in Action: Practical Multithreading* per il multi-threading, multi-code programming. *Thinking in C++* di Bruce Eckel è stato uno dei primi libri liberamente disponibili sul web sul C++, conosco Bruce dagli anni '80, prima ancora che lo pubblicasse, ed è stato uno dei migliori libri scritti in quel periodo.

ioP: Quali sono i progetti realizzati in C++ che più hai apprezzato:

BS: Quello che gestisce il Mars Rover (la sonda che ha esplorato la superficie del pianeta Marte), o i motori delle navi, Adobe Photoshop, i browser, e tanti altri.

Ringraziamo Bjarne per la sua cortesia e disponibilità per averci concesso questa intervista esclusiva.

Andrea Leganza