Informatica III
modulo programmazione
Angelo Gargantini

20 gennaio 2015

Premessa

In questo documento trovate una breve introduzione a tutti gli argomenti di Informatica III e dove potete trovare il materiale di approfondimento.

Il libro di testo è:

Concepts in Programming Languages John C. Mitchell

Parte I
Programming languages

M0 Introduzione

- Breve storia dei linguaggi di programmazione. Libro 1.3 e 1.4
- Quanti sono i linguaggi di programmazione? Migliaia, vedi `http://en.wikipedia.org/wiki/List_of_programming_languages`
- Quale linguaggio di programmazione è il migliore? vedi TIOBE Programming Community Index
- La rivoluzione silenziosa: programmazione multi linguaggio
  - Aumenta sempre di più il numero di persone che programmano in diversi linguaggi.
  - `http://www.drdobbs.com/architecture-and-design/the-quiet-revolution-in-programming/240152206`

Famiglie di linguaggi di programmazione

- Procedural/Imperative-style programming
  - FORTRAN, Algol, Pascal, C, ...
- Functional/Applicative-style programming
  - LISP, Scheme, ML, Haskell, ...
- Declarative/Logic programming
  - Prolog, ...
- Object-oriented programming
  - C++, C#, Java, ...
- Hybrids
  - concurrent, parallel, dataflow, intensional, domainspecific, ... scripting & extension languages

- alcuni concetti fondamentali (Libro 4.1, 4.4.2)
  - sintassi e semantica
  - compilatori ed interpreti
  - correttezza sintattica, correttezza semantica.

Sintassi e correttezza

Tutti i programmi che sono sintatticamente corretti sono corretti anche semanticamente? Cioè quando li eseguo non ho errori?
Con l'analisi statica come la compilazione, posso trovare tutti gli errori nel mio programma?

In alcuni linguaggi no.
Esempio in C:

```
void main(){
  ((int*) 5)[0]=6;
}
```

Se lo compilo: gcc -o crash.exe crash.c
E lo eseguo ho un errore del SO (segmentation fault)

In altri linguaggi, tipo Java, la compilazione è più severa e un programma Java non fa errori, al più lancia un'eccezione che potrà essere gestita. Il programma precedente, trascritto in Java non viene compilato.

```
class Crash{
  static public void main(String[] args){
  ((int[]) 5)[0]=6;
  }
}
```

Al contrario anche programmi che sarebbero corretti eseguendoli vengono rigettati dal compilatore.
Esempio di programma semanticamente corretto

```
class SemOk{

  void print(){}

  static public void main(String[] args){
    Object o = new SemOK();
    o.print();
  }
}
```

M1 visibilità e funzioni

Intero Capitolo 7 del libro.

- linguaggi a blocchi (sottoprogrammi, ricorsione), introduzione dello stack, env pointer, e dei record di attivazione 7.1
- Bocchi in-line: record di attivazione, control link, 7.2 - esercizio 7.1
- Funzioni e procedure 7.3
- Passaggio di parametri: L-value e R_value,
  - passaggio per valore e per riferimento,
  - passaggio con puntatori,
  - puntatori di puntatori,
  - passaggio di array (il caso del C).
- acccess link (7.3.3) (saltare il caso di funzioni dentro funzioni)
- Tail recursion (7.3.4) e ottimizzazioni varie

- NO 7.4

Passaggio di array in C

Ricordarsi: When an array is passed as a parameter, only the memory address of the array is passed (not all the values). An array as a parameter is declared similarly to an array as a variable, but no bounds are specified. The function doesn't know how much space is allocated for an array. Di fatto le seguenti dichiarazioni sono equivalenti

```
void rcvA(int arr[5])
void rcvB(int *arr)
// Since it sends a pointer and ignores the array size anyway, the system
// lets you leave it out here, too.
void rcvC(int arr[]) {...}
```

Passaggio di puntatori a puntatori

Per modificare un puntatore devo usare il passaggio di puntatori a puntatori: vedi slides.

Passaggio in Java/C++

In Java i tipi primitivi sono sempre passati per valore e i tipi riferimento mediante il valore del riferimento. In C++ posso scegliere sia per i tipi primitivi che per le classi (e struct).

Per il passaggio in Java guarda qui: `http://javadude.com/articles/passbyvalue.htm`

Passaggio delle struct/class in C/C++

Le struct vengono copiate interamente sullo stack - per questo si preferisce usare il passaggio per riferimento o con puntatore.

Ricorsione

Una funzione matematica è definita ricorsivamente quando nella sua definizione compare un riferimento (chiamata) a se stessa. Esempio: Funzione fattoriale su interi non negativi: f(n) = n! definita ricorsivamente come segue: 1 se n=0 f(n)= n*f(n-1) se n>0 Esempi di problemi ricorsivi: 1) Somma dei primi n numeri naturali: somma(n)= 0 se n=0 n+somma(n-1) altrimenti 2) Ricerca di un elemento el in una sequenza di interi: falso se sequenza terminata, altrimenti ricerca(el,sequenza)=vero se el=primo(sequenza), altrimenti ricerca(el,resto(sequenza))=

Programmi ricorsivi Molti linguaggi di programmazione offrono la possibilità di definire funzioni/procedure ricorsive.
Esempio Calcolo del fattoriale di un numero:

```
int fattoriale(unsigned int n){
        if (n<=1) return 1;
        else return n*fattoriale(n-1);
}
```

Esempi Alcune volte è necessario "complicare" la segnatura del metodo per renderelo ricorsivo:
Ricerca di un elemento in un array (Java)

```
// cerca x in array a a partire dalla posizione pos
boolean search(int x, int[] a, int pos){
        if(pos >= a.length) return false;
        if(a[pos] == x) return true;
        // non trovato nella posizione pos vai alla prossima
        return search(x,a,pos+1);
}
```

In C spesso si passa anche la dimensione dell'array
Ricerca di un elemento in un array (C):
array passato come puntatore

```
// cerca x in array a con lunghezza n
int search(int x, int* a, int n){
        if(n == 0) return 0;
        if(a[0] == x) return 1;
        // non trovato nella posizione a[0] vai alla prossima
        return search(x,a+1,n-1);
}
```

Altri tipi di passaggio

Esistono anche altri tipi di passaggio di parametri alle procedure.Considera il seguente esempio:

Pass-by-Name. `http://www.cs.sfu.ca/~cameron/Teaching/383/PassByName.html`

Perhaps the strangest feature of Algol 60, in retrospect, is the use of pass-by-name. In pass-by-name, the result of a procedure call is the same as if the formal parameter were substituted into the body of the procedure. This rule for defining the result of a procedure call by copying the procedure and substituting for the formal parameters is called the Algol 60 copy rule. Although the copy rule works well for pure functional programs, as illustrated by b reduction in lambda calculus, the interaction with side effects to the formal parameter are a bit strange. Here is an example program showing a technique referred to as Jensen's device: passing an expression and a variable it contains to a procedure so that the procedure can use one parameter to change the location referred to by the other:

```
begin
integer i;
integer procedure sum(i, j);
   integer i, j;
```

```
    comment parameters passed by name;
    begin
     integer sm;
     sm := 0;
     for i := 1 step 1 until 100 do sm := sm + j;
     sum := sm
     end;
   print(sum(i, i*10 ))
end
```

In this program, the procedure sum(i,j) adds up the values of j as i goes from 1 to 100. If you look at the code, you will realize that the procedure makes no sense unless changes to i cause some change in the value of j; otherwise, the procedure just computes 100*j. In the call sum(i, i*10) shown here, the for loop in the body of procedure sum adds up the value of i*10 as i goes from 1 to 100.

M2 Tipi e sicurezza dei Tipi

- Types in Programming (6.1)
- Type Safety and Type checking (6.2)

Runtime e compile time type checking - Statically Typed vs. Dynamically Typed Languages

`http://ofps.oreilly.com/titles/9780596155957/IntroducingScala.html`
One of the fundamental language design choices is static vs. dynamic typing. The word "typing" is used in many contexts in software. Pierce offers the following "plausible" definition that is useful for our purposes.

> A type system is a tractable syntactic method for preserving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute. – Benjamin C. Pierce Types and Programming Languages (2002)

Note the emphasis on how a type system allows reasoning about what a system excludes from happening. That's generally easier than trying to determine the set of all allowed possibilities. A type system is used to catch various errors, like unsupported operations on particular data structures, attempting to combine data in an undefined way (e.g., trying to add an integer to a string), breaking abstractions, etc.
Informally, in static typing, a variable is bound to a particular type for its lifetime. Its type can't be changed and it can only reference type-compatible instances. That is, if a variable refers to a value of type A, you can't assign a value of a different type B to it, unless B is a subtype of A, for some reasonable definition of "subtype".
In dynamic typing, the type is bound to the value, not the variable. So, a variable might refer to a value of type A, then be reassigned later to a value of an unrelated type X.
The term dynamically-typed is used because the type of a variable is evaluated when it is used during runtime, while in a statically-typed language, the type is evaluated at parse time.
This may seem like a small distinction, but it has a pervasive impact on the philosophy, design, and implementation of a language. We'll explore some of these implications as we go through the book.
Scala and Java are statically-typed languages, while Ruby, Python, Groovy, JavaScript, and Smalltalk are dynamically-typed languages.
For simplicity, we will often use the terms static language and dynamic language as shorthands for statically-typed language and dynamically-typed language, respectively.
An orthogonal design consideration is strong vs. weak typing. In strong typing, every variable (for static typing) or value (for dynamic typing) must have an unambiguous type. In weak typing, a specific type is not required. While most languages allow some mixture of strong vs. weak typing, Scala, Java, and Ruby are predominantly strongly-typed languages. Some languages, like C and Perl, are more weakly typed.

- Errori del C [vedi slides]
- Modi di rendere il C safe [slides]

  1. tools analisi statica e dinamica. esempi

    - rational purify
    - valgrind
    - lint
2. libraries
3. languages

Per questo capitolo oltre al libro di testo, ho preso diverso materiale dal libro:
Secure Coding: Principles and Practices
by Mark G. Graff, Kenneth R. Van Wyk
200 pages Publisher: O'Reilly Media, Inc.;
1 edition (July 2003)
ISBN-10: 0596002424
ISBN-13: 978-0596002428
Un'altro libro molto interessante è:
Writing Secure Code
Michael Howard , David LeBlanc
Publisher: Microsoft Press; 2nd ed. edition (January 4, 2003)
Language: English
ISBN-10: 0735617228 ISBN-13: 978-0735617223

Perchè il C ha NUL terminated strings

The Most Expensive One-byte Mistake
Did Ken, Dennis, and Brian choose wrong with NUL-terminated text strings?
by Poul-Henning Kamp | July 25, 2011
ACM
`http://queue.acm.org/detail.cfm?id=2010365`

Cyclone

Materiale:

- lucidi
- documentazione del sito: cyclone.thelanguage.org

M3 Object oriented programming

Capitolo 10 del libro (tutto)

- Concetti generali dell'Object oriented design
- **Encapsulation**: Come realizzare incapsulamento in C (typedef e moduli), con ADT
    - Esempio un contatore
        * uso di typedef:
        * uso del modulo
        * uso dei record/puntatori opachi
- **Sottotipazione** ed **ereditarietà**: differenze e casi particolari

Subtyping is not inheritance

http://www.claremontmckenna.edu/math/ALee/cs62_f2010/refs/whatisOOPL.pdf
Perhaps the most common confusion surrounding object-oriented programming is the difference between subtyping and inheritance. One reason subtyping and inheritance are often confused is that some class mechanisms combine the two A typical example is C++ where A will be recognized by the compiler as a subtype of B only if B is a public parent class of A. Combining these mechanisms is an elective design decision however there seems to be no inherent reason for linking subtyping and inheritance in this way. We may see the dierences between inheritance and subtyping most clearly by considering an example. Suppose we are interested in writing a program that requires dequeues stacks and queues.
One way to implement these three classes is first to implement dequeue and then to implement stack and queue by appropriately restricting and perhaps renaming the operations of dequeue. For example stack may be obtained from dequeue by limiting access to those operations that add and remove elements from one end of the dequeue. Similarly we may obtain queue from dequeue by restricting access to those operations that add elements at one end and remove them from the other. This method of dening stack and queue by inheriting from dequeue is possible in C++ through the use of private inheritance. We are not recommending this style of implementation we use this example simply to illustrate the dierences between subtyping and inheritance. Note that although stack and queue inherit from dequeue they are not subtypes of dequeue. To see this point consider a function f that takes a dequeue d as an argument and then adds an element to both ends of d. If stack or queue were a subtype of dequeue then function f should work equally well when given a stack s or a queue q. However adding elements to both ends of either a stack or a queue is not legal hence neither stack nor queue is a subtype of dequeue. In fact the reverse is true. Dequeue is a subtype of both stack and queue since any operation valid for either a stack or a queue would be a legal operation on a dequeue. Thus inheritance and subtyping are different relations we defined stack and queue by inheriting from dequeue but dequeue is a subtype of stack and queue not the other way around.

- **Binding dinamico**: concetto del single dispatching

Design patterns

Libro capitolo 10, sezione 4.

singleton vedi libro (esempio 10.3) ed esercizio

facade vedi libro (esempio 10.4) ed esercizio

visitor vedi libro (esercizio 10.3) ed esercizio

  - visitor versione base
  - visitor generico per il tipo ritornato

M4 java

Capitolo 13 Mitchell (e Manuale ufficiale del linguaggio Java suglia rgomenti segnati)

- Overview del linguaggio Java (scopi, storia, etc): 13.1
- Le classi e l'ereditarietà in Java (classi e oggetti, packages e visibilità, eredità, classi astratte ed interfacce): 13.2
- I tipi in Java e i sottotipi (sottotipazione per classi ed interfacce, array e il problema della covarianza deglia array): 13.3.1,2,3
- Dynamic binding in Java: vedi lucidi, o libro di Informatica II o manuale linguaggio Java
- Novità di Java 1.5: varargs, enum, static import:`http://java.sun.com/j2se/1.5.0/docs/relnotes/features.html#lang` , solo le Java Language Features
- covarianza del valore ritornato: `http://java.sun.com/docs/books/jls/third_edition/html/classes.html#8.4.5`

- Programmazione generica:
  vedi `http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf`
  Sezione da 1 a 5 compreso.
  Oppure: `http://download.oracle.com/javase/tutorial/java/generics/index.html`
  - Perchè si introducono i generici e i vantaggi rispetto alla programmazione generica con gli Object
  - Come si definiscono le classi generiche
  - Generici e sotto tipi: uso del wildcard ?
  - Bounded wildcards (extends and super) - in particolare l'uso nelle collezioni ordinate
  - Metodi generici

Covariance and contravariance

`https://en.wikipedia.org/wiki/Covariance_and_contravariance_%28computer_science%29#Contravariant_method_argument_type`

Constraints over the generics - Bounded Type Parameters

When you declare a type parameter T and use it in the implementation of a generic type or method, the type parameter T still denotes an unknown type. The compiler knows that T is a place holder for a type, but it does not know anything about the type. This is okay in some implementations, but insufficient in others.
For example we wnat to implement a sorted sequence.
Example (of a generic type, so far without bounds):

```
public class SortedSequence<T>{
  private List<T> items;
  ...
  /** add the element e in order in the sequence */
  public boolean add(T e){
    // bubble sort
    ... if (e.compareTo(y)) ...

 }
}
```

The implementation of class SortedSequence invokes the method compareTo on the unknown T type. Since compareTo is not defined for arbitrary types the compiler refuses to invoke the compareTo method on the unknown type T because it does not know whether the key type has a compareTo method.
In order to allow the invocation of the compareTo method we must tell the compiler that the unknown T type has a compareTo method. We can do so by saying that the T type implements the Comparable<T> interface. We can say so by declaring the type parameter T as a bounded parameter.
Example (of the same generic type, this time with bounds):

```
public interface Comparable<T> { public int compareTo(T arg); }
public class SortedSequence<T extends Comparable<T>>{
  private List<T> items;
  ...
  /** add the element e in order in the sequence */
  public boolean add(T e){
    // bubble sort
    ... if (e.compareTo(y)) ...

 }
}
```

M5 C++

Capitolo del C++ del Mitchell + 9.4. Un libro specifico sul C++ è consigliato.

cpp0 Introduzione al C++: storia e design goals

cpp1 *Classes and Data Abstraction*:

- class instead of struct, constructors (use and definition), dynamic objects
- array of objects, dynamic arrays, constructor inizializer list,
- destructors, new() and delete(),
- Function declaration, const modifier, call by value, call by reference
- Object Variable Classification (like in C): extern, static, automatic, static internal
- static members and member functions
- inline functions
- default arguments
- function overloading

cpp2 *Encapsulation*

- struct and class
- visibility and friend

cpp3 *Inheritance*:

- Constructors/destructors and inheritance
- Public, private, protected inheritance, Public and private base classes
- Redefining (non virtual) members
- Single and multiple inheritance,

cpp4

- "This" pointer
- dynamic lookup of virtual functions
- Virtual vs Overloaded Functions

cpp5

- Subtyping and inheritance
- covariance of return type in C++
- Abstract Classes

ccppextra Additions not related to objects

- Type bool
- Reference variables
- The Copy Constructor, Assignment Operator, and Inheritance

cpp-stl Generic programming in C++, templates, STL

The C++ String Class

C++ provides a simple, safe alternative to using char*s to handle strings. The C++ string class, part of the std namespace, allows you to manipulate strings safely. Declaring a string is easy:

```
#include <string>
using namespace std;
string my_string;
// or without a
std::string my_string;
```

You can also specify an initial value for the string in a constructor:

```
using namespace std;
string my_string("starting value");
```

String I/O is easy, as strings are supported by cin.

```
cin>>my_string;
```

If you need to read an entire line at a time, you can use the getline function and pass in an input stream object (such as cin, to read from standard input, or a stream associated with a file, to read from a file), the string, and a character on which to terminate input. The following code reads a line from standard input (e.g., the keyboard) until a newline is entered.

```
using namespace std; getline(cin, my_string, '\n');
```

Strings can also be assigned to each other or appended together using the + operator:

```
string my_string1 = "a string"; string my_string2 = " is this";
string my_string3 = my_string1 + my_string2;
// Will ouput "a string is this"
cout<<my_string3<<endl;
```

Naturally, the += operator is also defined! String concatenation will work as long as either of the two strings is a C++ string–the other can be a static string or a char*.

String Comparisons

One of the most confusing parts of using char*s as strings is that comparisons are tricky, requiring a special comparison function, and using tests such as == or < don't mean what you'd expect. Fortunately, for C++ strings, all of the typical relational operators work as expected to compare either C++ strings or a C++ string and either a C string or a static string (i.e., "one in quotes"). For instance, the following code does exactly what you would expect, namely, it determines whether an input string is equal to a fixed string:

```
string passwd;
getline(cin, passwd, '\n'); if(passwd == "xyzzy") { cout<<"Access allowed"; }
```

String Length and Accessing Individual Elements

To take the length of a string, you can use either the length or size function, which are members of the string class, and which return the number of characters in a string:

```
string my_string1 = "ten chars."; int len = my_string1.length(); // or .size();
```

Strings, like C strings (char*s), can be indexed numerically. For instance, you could iterate over all of the characters in a string indexing them by number, as though the the string were an array. Note that the use of the length() or size() function is important here because C++ strings are not guaranteed to be null-terminated (by a '\0'). (In fact, you should be able to store bytes with a value of 0 inside of a C++ string with no adverse effects. In a C string, this would terminate the string!)

```
int i; for(i = 0; i < my_string.length(); i++) { cout<<my_string[i]; }
```

On the other hand, strings are actually sequences, just like any other STL container, so you can use iterators to iterate over the contents of a string.

```
string::iterator my_iter;
for(my_iter = my_string.begin(); my_iter != my_string.end(); my_iter++) {
        cout<<*my_iter;
}
```

Note that my_string.end() is beyond the end of the string, so we don't want to print it, whereas my_string.begin() is at the first character of the string.

Incidentally, C++ string iterators are easily invalidated by operations that change the string, so be wary of using them after calling any string function that may modify the string.

M6 Scala

`http://docs.scala-lang.org/tutorials/`

Introduction

Abstract Types

Classes

Named Parameters

Functional programming in Scala

http://www.codecommit.com/blog/scala/scala-collections-for-the-easily-bored-part-2

Vedi lucidi

Using collections

Vedi lucidi

M7 Abstract State Machines

Vedi i lucidi

Per l'uso dei tool vedi il sito di Asmeta `http://asmeta.sf.net`