Esame informatica III 10 Luglio 2009 Con soluzioni parziali

1 Record di attivazione

Scrivi una funzione in C non ricorsiva che implementa l'algoritmo di Euclide per il calcolo del MCD tra due numeri. L'algoritmo dice così (wikipedia): "*Dati due numeri naturali a e b, si controlla se b è zero. Se lo è, a è il MCD. Se non lo è, si divide a / b e si assegna ad r il resto della divisione (operazione indicata con "a modulo b"). Se r = 0 allora si può terminare affermando che b è il MCD cercato altrimenti occorre assegnare a = b e b = r e si ripete nuovamente la divisione.*"

Scrivi il programma con un main che calcoli l'MCD tra due numeri (piccoli) che usi anche variabili globali e scrivi il record di attivazione fino alla sua massima estensione.

Scrivi la versione ricorsiva, possibilmente con tail recursion dell'MCDC e ridisegna il record di attivazione per la stessa chiamata.

Ecco una possibile soluzione in C, non ricorsiva, fedele alla descrizione nel testo.

```
unsigned int mcdc(unsigned int a, unsigned int b){
        int r = 0;
        if (b == 0) return a;
        while(1){
                r = a%b;
                if (r == 0) break;
                a = b;
                b = r;
        }
        return b;
}
```

Si poteva anche implementare senza r (un po' più compatta, vedi versione ufficiale su wikipedia ad esempio) ma questa riportata è più fedele al testo. Un main semplice era:

```
#include <stdio.h>

int x = 7;
int y = 3;

unsigned int mcdc(unsigned int a, unsigned int b){ ...}


void main() {
        printf("%d",mcdc(x,y));
}
```

Si noti l'uso delle variabili globali. Ecco il RA per la chiamata di mcdc(7,3)

[RA globale]	*[x 7]
	[y 3]
[RA main]	#[CL punta a *]
	[AL punta a *]

[RA non ci interessa]
[RRA non c'è perchè main void]
§[cella per mcdc(7,3)]
[RA mcdc(7,3)] °[CL punta a #]
[AL punta a *]
[RA non ci interessa]
[RRA punta a §]
[a = 7]
[b = 3]
[r]
[record del blocco in line del while] [CL punta a °] AL, RA,RRA non ci sono,

viene eseguito il ciclo fino a quando il valore di r non è zero, cioè la prima volta r = 1, a = 3, b = 1, secondo giro r = 0, return 1; viene copiato 1 in § e viene liberato tutto lo stack

–
La versione ricorsiva, con tail recursion e senza memorizzazione del resto era:

```
#include <stdio.h>

int x = 7;
int y = 3;

unsigned int mcdc(unsigned int a, unsigned int b){
        if (b == 0) return a;
        return mcdc(b, a%b);
}

void main() {
        printf("%d",mcdc(x,y));
}
```

La versione con il resto andava ugualmente bene. Questa è ricorsiva ed è ricorsiva in coda (restituisce inalterato il risultato della chiamata a se stessa). Si poteva di segnare il RA sia per la versione ottimizzata (meglio) che quella distesa con le dovute spiegazioni.

2 Passaggio parametri

Considera il codice precedente (MCDC) e ipotizza l'uso di C++. Quali passaggi di parametri possono andare bene per gli input? Potresti modificare l'MCDC in modo che sia una procedura void e passargli la variabile in cui mettere il risultato? Potresti avere qualche effetto indesiderato? Come evitarlo?

Prendiamo il codice non ricorsivo:

```
unsigned mcdc(unsigned int a, unsigned int b)
```

Potrei modificarlo in modo da prendere le seguenti combinazioni

a	b	commenti
Val	Val	Tutto Ok, come prima. Le modiche però rimangono locali, il risultato va ritornato con return

Val	Ref/puntatore	O viceversa, Funziona però ci sono delle modifiche alle variabili passate come riferimento o come puntatore. Nota che se si passa come riferimento basta cambiare solo l'intestazione:
		`unsigned int mcdc(unsigned int a, unsigned int& b){`
		mentre se si passa come puntatore:
		`unsigned int mcdc(unsigned int a, int* b){`
		bisognerà modificare anche il programma in modo da dereferenziare b (*b) e la chiamata con &y.

Ref/punt	Ref/punt	Potrebbe non funzionare perchè le modifiche di una variabile andrebbero a modificare anche l'altra variabile. Il codice diventerebbe, se passato come ref: `unsigned int mcdc(int &a, int& b){` E si immagini ora di utilizzare la stessa variabile sia per a che per b, cioè che sia int x, e il passaggio mdcd(x,x). in questo caso r == 0 e restituriebbe subito il valore di x;

Nella versione ricorsiva:

a	b	commenti
Val	Val	Ok come prima
ref/puntare	val	Ok, come prima
	Ref/puntatore	Non va bene, perchè passo un'espressione a b.

Per avere indietro il risultato pur essendo void la procedura potrei:

- nella versione non ricorsiva, passare b come riferimento e modificare leggermente il codice nel caso in cui b è zero, copiare a in b. Infatti alla fine b contiene il risultato.
- Oppure aggiungere una variabile passata come riferimento a cui assegnare il risultato.

3 Cyclone

Scrivi una procedura in Cyclone che riempie di '1' una stringa. Elenca tutti i tipi di dichirazione di parametri della procedura che conosci in Cyclone, che potresti utilizzare e quali sono i pro e i contro.

Ecco il codice direttamente con 5 alternative (mancano i pro e i contro)

```
#include <stdio.h>
#include <string.h>



/** prima versione semplice con un puntatore a char
di default è @zeroterm*/

void writeones(char * s){
          unsigned int len = strlen(s); // posso lo stesso sapere la dimensione
      for (unsigned int i = 0; i < len; i++, s++)
        *s = '1';
}

// altri puntatori che posso utilizzare:
// con @not null
void writeones_notnull(char * @notnull s){
          unsigned int len = strlen(s); // posso lo stesso sapere la dimensione
```

```
      for (unsigned int i = 0; i < len; i++, s++)
        *s = '1';
}
// versione fat, rinucio al zero term,
void writeones_fat(char ? @nozeroterm s){
         unsigned int len = strlen(s); // posso lo stesso sapere la dimensione
      for (unsigned int i = 0; i < len; i++, s++)
        *s = '1';
}
// versione errata con fat che usa numelts invece di strlen
// versione fat, rinucio al zetro term,
void writeones_fat2(char ? @nozeroterm s){
         unsigned int len = numelts(s); // posso lo stesso sapere la dimensione
      for (unsigned int i = 0; i < len; i++, s++)
        *s = '1';
}
// versione con bounded pointers
// contiene ancora qualche errorino, ma in principio va bene
void writeones_bound(tag_t num, char s[num] @nozeroterm){
      for (unsigned int i = 0; i < num && s[i] != 0; i++)
        s[i] = '1';
}


int main() {
         // chiamo la prima versione
         char* p1 = new {'b','a','r',0};
         writeones(p1);
         printf("%s\n",p1);
         // not null:
         char* p2 = new {'b','a','r',0};
         writeones_notnull(p2);
         printf("%s\n",p2);
         // fat e not zeroterm
         char? p3 = new {'b','a','r',0};
         writeones_fat(p3);
         printf("%s\n",p3);
         // fat not zero term con uso di numelts
         char? @nozeroterm  p4 = new {'b','a','r',0,'r','g'};
         writeones_fat2(p4);
         printf("%s\n",p4); // stampa più uni
         // bonded pointers
         char p5[4] = "bar";
         writeones_bound(4,p5);
         printf("%s\n",p5);
         // end
         return 0;

}
```

---

4 Dinamic Binding in Java

Considera le seguenti classi

```
class Person {
   void setEta(int x){}
   boolean equals(Person p){return true;}
}
class Docente extends Person {
   void setEta(int x){}
```

```
   boolean equals(Docente p){return true;}
}
```
la istruzioni (in sequenza):
```
Person p = new Docente();
p.setEta(40);
p.equals(p);
```
sono corrette? Quale codice viene eseguito quanda chiamo i metodi e perchè?

Questo è il problema classico.

5 Java generics

Come faccio a dichiarare una classe generica in Java? Come la uso poi? Quali vincoli posso mettere sui tipi generici?

6 C++ - virtual

Date le seguenti dichiarazioni (corrette):

```
#include <iostream>
using namespace std;

class Strumento {
public:
  void play() { cout << "PS" << endl;}
  virtual void tune() { cout << "TS" << endl;}
};
```

```
class Flauto : public Strumento{
public:
  void play() {cout << "F" << endl;}
  virtual void tune() { cout << "TF" << endl;}
};

class Chitarra : public Strumento{
public:
  void tune() { cout << "TC" << endl;}
};
```

Spiega quale output produce il seguente main. Se ci sono alcuni errori, spiega perchè e ignorali.

```
int main() {
  Flauto f;
  Strumento s = f;
  s.play();
  s.tune();
  Chitarra c;
  s = c;
  s.tune();
  Flauto * pf = new Strumento();
  Strumento * ps = &f;
  ps->play();
  ps->tune();
  ps = &c;
  ps->play();
  ps->tune();
}
```

Flauto f; //OK
Strumento s = f; //OK, Flauto è una sottoclasse public di Strumento
s.play(); → PS, play non è virtual
s.tune(); → TS, tune è virtual ma s non è un puntatore
Chitarra c; // OK
s = c; //OK
s.tune(); → TS, tune è virtual ma s non è un puntatore
Flauto * pf = new Strumento(); // ERRORE: Strumento non è assegnabile a Flauto
Strumento * ps = &f; // OK, sootoclasse public ps punterà ad un Flauto
ps->play(); PS, play non è virtual
ps->tune(); TF, tune è virtual e ps punta ad un Flauto
ps = &c; // OK, ps punterà ad una chitarra
ps->play(); PS
ps->tune(); TC tune è virtual anche se non è dichirata tale in Chitarra

---

7 C++ - costruttori

Data una classe `Studente` che rappresenta studenti con un nome, cognome e matricola (che è opzionale). Come definiresti la classe con il costruttore (o i costruttori)? Come potresti usare i costruttori per creare oggetti Studente o puntatori ad essi (scrivi il maggior numero di forme sintattiche dell'uso dei costruttori).

```
#include <iostream>
#include <string>
using namespace std;

class Studente {
private:
  string nome,cognome;
  int matricola;
public:
  Studente(string nome, string cognome, int matricola = 0){....}
};

int main() {
  // tre modi di usare il costruttore
  Studente s("nome","cognome",5);
  Studente s2("nome","cognome");
  Studente s3 = Studente("nome","cognome");
  // per i puntatori:
  Studente* ps = new Studente("nome","cognome");
  Studente* ps2 = new Studente("nome","cognome",7);
}
```

---

8 Semantica assiomatica

Dimostra la correttezza del seguente programma che dovrebbe calcolare la potenza (^):

```
{ M>0 and N>=0 }
a := M; b := N; k := 1;
while b>0 do
        if b=2*(b/2) // b%2 =0
        then a := a*a; b := b/2
        else b := b–1; k := k*a
        end if
end while
{ k = M^N }
```

---

soluzione di questo esercizio è a pagina 410 e 411 di:
www.cs.uiowa.edu/~slonnegr/plf/Book/Chapter11.pdf